THE CONTROL AND OPTIMIZATION OF SWARM ROBOTS ON ROS2 PLATFORM

2023
MASTER THESIS
MECHATRONICS ENGINEERING

Mohammed SEDEG

Thesis Advisor
Assoc. Prof. Dr. Can FİDAN

THE CONTROL AND OPTIMIZATION OF SWARM ROBOTS ON ROS2 PLATFORM

Mohammed SEDEG

Thesis Advisor
Assoc. Prof. Dr. Can FİDAN

T.C.
Karabuk University
Institute of Graduate Programs
Department of Mechatronics
Prepared as
Master Thesis

KARABUK
August 2023

I certify that in my opinion the thesis submitted by Mohammed SEDEG titled “THE CONTROL AND OPTIMIZATION OF SWARM ROBOTS ON ROS2 PLATFORM” is fully adequate in scope and in quality as a thesis for the degree of Master of science.

Assoc. Prof. Dr. Can FİDAN ....

Thesis Advisor, Department of Mechatronics Engineering

This thesis is accepted by the examining committee with a unanimous vote in the Department of Mechatronics Engineering as a Master of Science thesis. 28 / 08 /2023

Examining Committee Members (Institutions)	Signature
Chairman : Assoc. Prof. Dr. Can FIDAN (KBU)	....
Member : Assoc. Prof. Dr. Murat TUNA (KLU)	....
Member : Assist. Prof. Dr. A. Talha SÖZER (KBU)	....

The degree of Master of Science by the thesis submitted is approved by the Administrative Board of the Institute of Graduate Programs, Karabuk University.

Assoc. Prof. Dr. Zeynep OZCAN ....

Director of the Institute of Graduate Programs

“I declare that all the information within this thesis has been gathered and presented in accordance with academic regulations and ethical principles and I have according to the requirements of these regulations and principles cited all those which do not originate in this work as well.”

Mohammed SEDEG

ABSTRACT

M. Sc. Thesis

THE CONTROL AND OPTIMIATION OF SWARM ROBOTS ON ROS2 PLATFORM

Mohammed SEDEG

Karabük University
Institute of Graduate Programs
The Department of Mechatronic Engineering

Thesis Advisors:
Assoc. Prof. Dr. Can FIDAN
August 2023, 80 pages

With the increase in application demands in the field of robotics, 5th level automation is aimed in many areas of our daily life, while the development of fault-tolerant multi-agent networks that can perform complex tasks without the need for a central unit is seen as a great need.

In this study, a ROS2 framework for multi-agent systems focusing on cooperation and coordination for executing distributed optimization and control algorithms is presented. The framework works within a peer-to-peer, Decentralized structure, with a special emphasis on heterogeneous networks that lack a central unit. The functionality of the system allows the implementation of highly complex optimization-oriented control schemes, including distributed dynamic task assignment and distributed Model Predictive Control (MPC).

Moreover, the proposed framework supports a simpler yet equally functional implementation of control feedback laws. This feature allows agents to exchange information without optimization requirements, fostering seamless communication among agents. This architectural design empowers developers and programmers to seamlessly incorporate and execute intricate optimization and control algorithms on a heterogeneous fleet of robots, all without the necessity of a central unit.

Importantly, this streamlined implementation process encourages a focus on innovative optimization and problem-solving techniques.

Keywords **:** ROS2 framework, multi-agent system, cooperative coordination, distributed optimization, peer-to-peer structure, heterogeneous networks, dynamic task assignment, Model Predictive Control (MPC).

Science Code **:** 92902

ÖZET

Yüksek Lisans Tezi

ROS2 PLATFORM ÜZERINDE SÜRÜ ROBOTLARININ KONTROLÜ VE OPTIMIZASYONU

Mohammed SEDEG

Karabük Üniversitesi
Lisansüstü Eğitim Enstitüsü
Mekatronik Anabilim Dalı

Tez Danışmanı:
Doç. Dr. Can FİDAN
Ağustos 2023, 80 sayfa

Robotik alanındaki uygulama taleplerinin artmasıyla birlikte günlük hayatımızın birçok alanında 5. seviye otomasyon hedeflenirken, merkezi bir birime ihtiyaç duymadan karmaşık görevleri yerine getirebilen hataya dayanıklı çok aracılı ağların geliştirilmesi büyük bir ihtiyaç olarak görülmektedir.

Bu çalışmada, dağıtılmış optimizasyon ve kontrol algoritmalarını yürütmek için iş birliği ve koordinasyona odaklanan çok aracılı sistemler için bir ROS2 çerçevesi sunulmaktadır. Çerçeve, merkezi bir birimden yoksun heterojen ağlara özel bir vurgu yaparak, eşler arası, merkezi olmayan bir yapı içinde çalışmaktadır. Sistemin işlevselliği; dağıtılmış dinamik görev ataması ve dağıtılmış Model Tahmini Kontrolü (MPC) dahil olmak üzere, oldukça karmaşık optimizasyon odaklı kontrol şemalarının uygulanmasına olanak sağlar.

Üstelik önerilen çerçeve, kontrol geri bildirim yasalarının daha basit ama aynı derecede işlevsel bir uygulamasını desteklemektedir. Bu özellik, aracıların optimizasyon gereksinimleri olmadan bilgi alışverişinde bulunmasına olanak tanıyarak aracılar arasında kesintisiz iletişimi teşvik eder. Bu mimari tasarım, geliştiricilere ve programcılara, karmaşık optimizasyon ve kontrol algoritmalarını, merkezi bir üniteye ihtiyaç duymadan, heterojen bir robot filosu üzerinde sorunsuz bir şekilde birleştirme ve yürütme yetkisi verir.

Daha da önemlisi, bu kolaylaştırılmış uygulama süreci, yenilikçi optimizasyon ve problem çözme tekniklerine odaklanmayı teşvik eder.

Anahtar Kelimeler : ROS2 çerçevesi, çoklu ajan sistemi, iş birliği koordinasyonu, dağıtılmış optimizasyon, eşten eşe yapısı, heterojen ağlar, dinamik görev atama, Model Öngörülü Kontrol (MPC).

Bilim Kodu : 92902

ACKNOWLEDGMENT

I would like to extend my sincere appreciation to my advisor Assoc. Prof. Dr. Can Bülent FIDAN, for his invaluable guidance and support throughout the preparation of this thesis. His unwavering interest and assistance have been instrumental in making this thesis possible.

Furthermore, I would like to express my heartfelt gratitude to my parents and especially my mother, whose unwavering belief in me and continuous support have been a constant source of motivation. I am also grateful to my brothers, who have always been my pillars of strength in life. I am immensely thankful to my friends, who keep me grounded, remind me of the important things in life, and provide unwavering support in all my endeavors. Lastly, I would like to extend my thanks to everyone who has stood by me during this journey.

CONTENTS

Page

Page

Page

LIST OF FIGURES

Page

LIST OF TABLES

SYMBOLS AND ABBREVITIONS INDEX

ABBREVITIONS

AI	: Artificial Intelligent
MRS	: Multi-robot System
PSO	: Particle Swarm Optimization
ACO	: Ant Colony Optimization
BCO	: Bee Colony Optimization
GA	: Genetic Algorithm
ML	: Machine Learning
DSDV	: Destination sequenced distance vector.
TGL	: Team Guidance Layer
MPC	: Model Predictive Control
GSO	: Glowworm Swarm Optimization
FA	: Firefly Algorithm
SDF	: Simulation Description Format
ROS	: Robotic Operating System

PART 1

INTRODUCTION

1.1. BACKGROUND INTRODUCTION

There is a rapid race we facing as a human race which is the race of meeting our growing needs and demands with the aid of technology. Therefore, a similar rapid race in technology is taking place in the industrial fields mostly to reduce the factors of time and cost. As a result, in our days the humans have been replaced by semi-autonomous system but in the near future the same semi-autonomous systems will leave for fully autonomous systems as the industrial fields have been moving aggressively toward the level 5 automation (there are five level of automation level zero is fully Manuel to level 5 which mean the human factor in the operation is totally zero)[1].This leads us to robotics, generally we can define the robot as any mechanical system which perform certain tasks either by direct guide from human interference or indirect by pre-defined program or even auto-generated guides by AI [2].Robotics researches could be classified in to three main categories and under them even more categories as shown in figure 1.1:

- Robot manipulators: or robot arm is a serial of combined actuators that perform a certain task like carrying an object with it end effector. This type is used heavily in production line and I other industrial applications.
- Mobile robots: this term describe a certain type of robot that specialized in moving from place to place while carrying tasks and consisting of a platform locomotive element. Within this category there are other branches differ on the medium of the travel (aerial, aquatic or terrestrial)
- Biologically inspired robots: a more complex in the designed because the similarity with the complex biological beings [3].

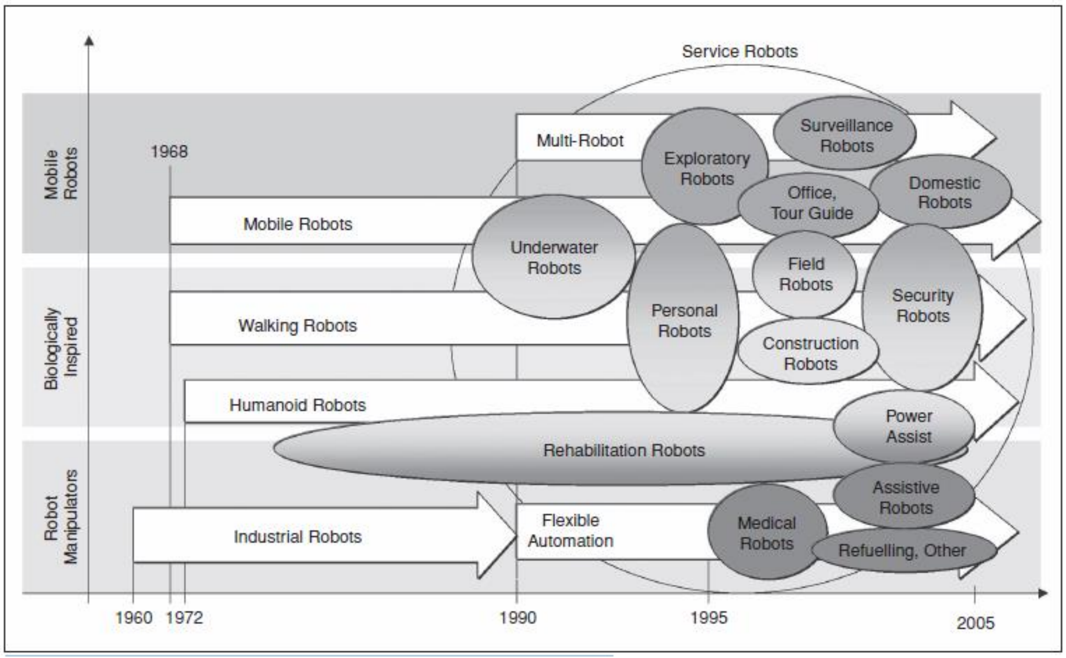


Figure 1.1. Evolution of the robotics research.

1.2. MULTI- ROBOTICS SYSTEM

We can define multi-robotic system (MRS) as collection of two or more robots working in coordination to achieve certain goals. The fundamental theory behind (MRS) state that dividing a main complex problem in to multiple sub-problem and give it to an individual robot among the group may be a very optimal solution, rather than trust the whole process to one main robot even if it’s very capable [4].

Multi-robotic system holds many advantages over single robotic systems, the most obvious advantage is the parallelism of the system in the sense that the system can work parallelly to enhance the performance and work in tasks which is not possible for single robot. Robustness mean that the system has no single fault point like the single robot system, so if a single agent faces a fault the rest of the can carry on the task with no stop. Scalability over the centralized systems that mean covering a bigger area. The (MRS) could be both homogeneous or heterogeneous this mean that the robot team can both the similarity and dissimilarity when it come to the physical hardware. Other advantages can be summarized as: flexibility, economic benefits and low cost for assembly and maintenance, stability and efficiency in both energy and task handling [5].

The classification of (MRS) can be judged by many criteria like efficiency, effectiveness, the task’s nature, robustness, and flexibility. In addition, other classification was found [6] on other researches based on:

- The size of the swarm.
- The communication probabilities such as:(bandwidth, structure and range).
- Reconfigurability.
- The processing capability of every single agent.

There is no clear difference between the two terms multi-robotics and swarm robotics, in fact the two term is used interchangeably through the previous researches, but others distinguished them not on the hardware aspect but on the problems and their solutions [7].

The designing of swarm robotic should be accomplished under a number of abstraction layers according to [8]:

- The first layer: the micro-layer of the system as every single induvial of the swarm has its own automation identity.
- The second layer: the macro layer as the whole control identity of the system represented by a set of differential equation.
- The third layer: the communication layer and it’s all probabilities, rules and approaches.
- The fourth layer: the sensor and actuator layer as how the agents will communicate and interact with the surrounding environment.
- The fifth layer: the swarm intelligent algorithm layer and which method will be used for example particle swarm optimization (PSO).

1.2.1. Coordination Movement

Like our human society the (MRS) has a collective behavior that could described as cooperative and comparative. Cooperative behavior refers to a scenario where multiple robots nearby need to interact with each other to achieve a certain a task, this task divided by the robots to serval sub-tasks handled collectively by the team while increasing the utility of the system. On the other hand, comparative behavior is the opposite behavior which includes a conflicting function like a chess match between two robots [9].

Coordination movement is the core of (MRS) technology. Its directly lead to the success or the failure of the system. Coordination movement can be separated in to two categories: static and dynamic. The static coordination (also known as offline coordination) is based on previously inherent commands that fed to the robot for example traffic rules [10]. The dynamic coordination (also known as online coordination) on the hand is based on real time information fed to the robot via communication, therefore we can classify dynamic coordination into implicit and explicit coordination based on the communication type (implicit and explicit communication) [11].The static coordination is very suited for complex tasks but unreliable in real time controlling, in the same time dynamic coordination is perfect for real time control but unreliable in complex tasks.so its best to combine the two types and take the best characteristics of the both .

Flocking is one of the most known tasks among swarm robotics, other tasks include aggregation, object clustering and sorting, chain formation, self-deployment and collaborative, manipulation. The explanation for every task mentioned is shown in table 1.1 :

Table 1.1. The description of common swarm robotics tasks [12].

Task	**Description**
Aggregation	The collection of the swarm is to gather in one around one main goal.
Flocking	Flocking is the collective movement of the swarm toward one desired goal.
Clustering	The swarm collect different objects to one unified position. Like ant collecting food.
Chain Formation	The main object of chain formation is to form the shortest path chain between two locations.
Collaborative manipulation	Collective effort from all the swarm to move unified object from one point to the other.
Shape formation	The swarm form a certain formation geometrical or non-geometrical by exchanging position information between the swarm.

1.2.2. Communication

Communication is one of the most important characteristic and challenges in multi-robot system (MRS). The robots in (MRS) are generally simple in structure and design, so in order for the system to achieve a goal a communication protocol must be set in order to multiply the capability and maximize the efficiency[13]. communication is mode of interaction between the agents for mainly two purposes: the first is to communicate the state of the environment through the sensors, the second is to communicate and exchange information with the other agents. Communication methods can be classified in many ways for example we can classify based on the interaction mode into three types: interaction via sensing, interaction via environment and via explicit communication [14]. Other way of classification is based on the information exchange between the agents which include: direct and indirect communication [15]. But mostly there are mainly two types of communication implicit and explicit, the explanation as follow:

Explicit communication in the robotic term refer to the operation of sending the information from one robot to the other through the different communication methods like Bluetooth and wi-fi and Bluetooth. This type of communication is very beneficial in applications that require fast reaction in smaller systems. On the other hand, implicit communication is the operation of one robot observing other robot behavior to copy it without the observed robot contribution in the communication operation [16]. Even so there is no clear indication on what type of communication must be applied, some researchers aiming toward implicit communication for its robustness[17] , and other introduced the implementation of both implicit and explicit communication . The most important question remains how to choose the communication characteristic like structure, algorithm and medium to insure the desired performance from the system. The answer to this question is lying under four main factors:

- Application: the application at hand is a very important factor in choosing the right communication specially the communication range, for example a swarm of drone for agriculture monitoring requires a wide range in contrast other application like chain formation requires a short-range communication for control [18].

- Robot: in other words, the physical hardware which include the receive and transmit capability of the hardware, for example the raspberry pi Pico W controller has a build in Wi-Fi, another example is the GPS which is very important factor in Ariel swarm control [19].
- Algorithm: the used algorithm is the connection link between the application and the robot. Various algorithm can be implemented in (MRS) like: Particle swarm optimization (PSO), nature inspired algorithm like Ant colony and bee colony approaches and machine learning (ML) algorithm [18].
- Environment: when it come to the environment surrounding the robot many questions must be answered to determine the network design, some of these questions are is the environment empty of full of obstacles, is there a wireless interference, is the environment indoor or outdoor.

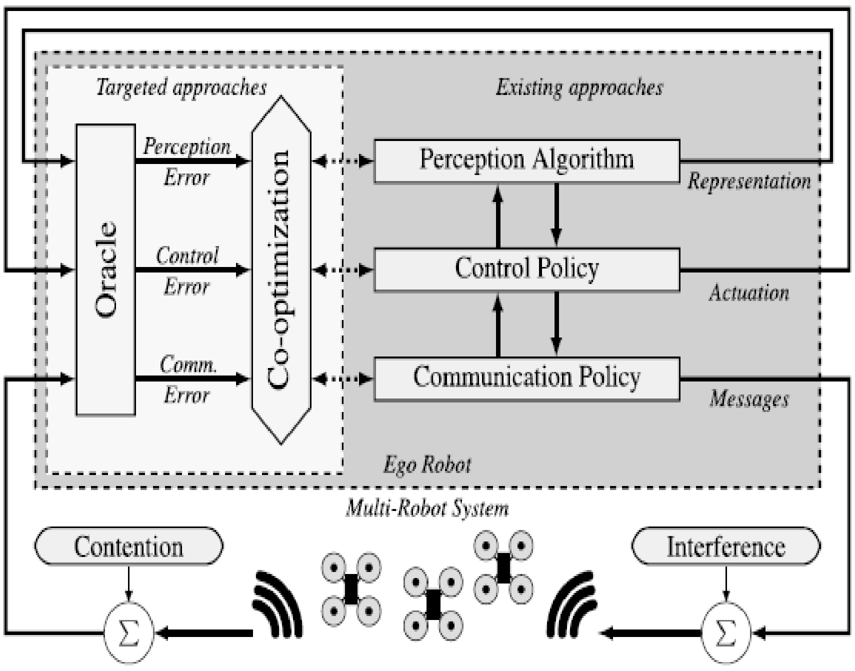


Figure 1.2. Flow diagram of communication in MRS.

1.2.3. Swarm Intelligent (SI)

Swarm intelligence, as a problem-solving ability, arises from interactions among simple information-processing units. The term "swarm" implies diversity, randomness, and complexity, while "intelligence" indicates the method's effectiveness in solving problems. The information-processing units within a swarm can take various forms, such as animate beings like insects, birds, or humans, or they can be mechanical, computational, or mathematical entities like robots, standalone workstations, or array elements. The interactions between these units can possess diverse characteristics, but interaction among them is essential [20].

Swarm behavior is commonly applied in mobile robots and involves the cooperative movement of creatures. Swarm robotic collective decision-making is the ability to make joint decisions through local interactions without a centralized leader [21]. Task distribution exemplifies this collaborative behavior.

Swarm robots efficiently perform tasks by moving towards specific targets and creating coordinated patterns. This behavior is akin to bacterial colonies' distribution of molecules [22], inspired by pattern formations and foraging behavior observed in ants and social insects [23].

The aim of this thesis is to enable multiple ground mobile vehicles to cluster and form predetermined geometric formations. To achieve this, the study applies what we called Team Guidance Layer (TGL) which is a peer-to-peer non-centralized communication layer that act as a platform for different control algorithms to be built and tested on GAZEBO-ROS simulation. TGL allow us to execute complex distributed multi-robot tasks, such as model predictive control (MPC) and tasks assignment and formation control either in simulation or experimentally.

Chapter 2 presents a literature review on the subject, while Chapter 3 delves into swarm intelligent (SI), including mathematical models and clustering algorithms. The models created in Chapter 4 are evaluated through GAZEBO-ROS simulation. Finally, Chapter 5 provide the conclusion.

PART 2

LITERATURE REVIEW

The characteristics of nature beings living in form of swarm have capture the attention of the scientific world for many years. The swarm behavior phenomenon can be observed very clearly in nature in the form of bird and fish flocking and or migrating cells etc. their collective actions display advanced behaviors and impressive feats that can be observed showing the total benefit of swarm behavior to the whole system[24]. therefore, researcher aim to observe the swarm behavior to mimic it through modeling. the herd intelligence of living species in nature. Behaviors of living creatures modeling can yield productive results in many areas [25].

Figure 2.1. Swarm behavior in nature.

Multi-agent, also known as swarm research, contributes significantly to search and rescue missions, military operations, economics, finance, and numerous applications in engineering. It is also vital in solving optimization problems and other autonomous tasks. The desired approach in solving engineering problems is to reach a global target using simple local rules. The concept of the best solution in multi-agent optimization was first introduced by F.Y. Edgeworth (1881), who worked on problems of buying and selling decisions in the field of economics. V. Pareto's (1896) concept of Pareto Optimality has gained significant acceptance in the field of economics. The first studies of biologists examining swarm behaviors were conducted by Breder [26], Warburton and Lazarus [27], Okubo and Grunbaum [28], and Parrish [29]. Inspired by these studies, recent research [30] has made important contributions to swarm formation.

2.1. EARLY LITERATURE

We can trace through the literature reviews that the researchers in the late 1980's have been more motivated to design and build a team of robots capable to execute different tasks through cooperation, coordination, and communication. This determination stems from the fact that (MRS) hold several advantages against the typical single robot system, and all the early studies aims to harvest these advantages. To understand how multiple entities can work together in harmony researchers has to turn to natural and study different example like bees and ant colonies.

Among the earlier research is [31] which proposed a multi-robot system scheme goes by the name M+, this scheme is a decentralized system for the purpose of task distribution and task cooperation. An early and simpler version of the system is implemented and test on a simulation. ALLIANCE is "a novel, fault tolerant cooperative control architecture for small- to medium-sized heterogeneous mobile robot teams applied to missions involving loosely coupled, largely independent tasks " [32], this architecture operate in two Leve: level one is the induvial robot level which allow the agent to execute any task it choose without relying on centralized command and level two is on the robot team level gives equal control command to every agent which allow any robot to choose a sub-task without relying in centralized command. This system is applied into both the physical and simulation medium, specifically on box pushing task. The study [33] offers a comparison between three different approaches in (MRS) coordination mentioning the pros and cons of every induvial approach, in addiction the study also discussed the communication scheme between the system in every approach and also the problem of fault tolerance and robustness. The paper [34] provide an overview of a project named GOFER, which is the implementation of serval dozen robot in indoor environment, the project dealt with many research issues such as: the communication of man to robot and robot to robot, the multi-task planning, fault tolerance and the implementation of non-conflicting sensor system. In addition, the study [35] proposed the idea of (CEBOT) or Cellular Robotic System, this system consist of a huge number of robots every single one called cell, and the robotic manipulator described by geometric calculation. Also, all the experimental results are showed in GEBOT Mark 2 which is a prototype of CEBOT. Since this early research the field of (MRS) has been grown dramatically with a wider range of topic to be addressed, despite this we cannot yet describe the field as mature, because there are many specific topics to be addressed and researches compared to the single robot system. In survey [36] stated the main topic that need to addressed are in (MRS) through the literature:

- Communication.
- Movement coordination.
- Task allocation and control.
- Localization and mapping.
- Object transportation and manipulation.

2.2. MULTI-ROBOT COMMUNICATION

Since the beginning of (MRS) studies communication has been an important issue, where is several studies focus specifically on the effect of communication on the system, in addition studying the different types and every task suited for it. Generally, all the research agreed on that even a small scale of communication can lead to huge benefit to the system.

Several communication protocols have been explored through the literature, for example (PRNET) [37] which stand for Packet Radio Networking is highly reliable communication protocol aim to exchange information between two separated computers geographically via radio channels. The (DSDV) or ‘Destination sequenced distance vector routing’ is a table-driven algorithm developed in [38] utilized the mobile nodes as routers and every single router assigned to sequence number to prevent loops by making distinct different between new and old routers. In addition, the communication protocol known as (WRP) or Wireless Routing Protocol [39] is designed for the purpose of reducing the number of loops by having a massage transmitting list, every node uses this updating massage for exchanging information between them for fast coverage. Taking the next step further the study [40] propose communication protocol (GRS) or Global State Routing designed for multi-hop mobile wireless network to avoid the overflow of the updated massages, this achieved by making every node have a list of neighbor nodes including next step hop and distance hop. So, to overcome the drawback of (GRS) which include the size of the massage of the close and distance neighbor, the study [41] purpose (FSR) standing for Fisheye State Routing which provide an accurate list of information only for the close node neighbor to avoid the massage size problem associated with [40].

The interaction between the human operator and the swarm robotic via communication protocols can be divided in to two main parts: first the remote interaction which can be define as a communication method when the human operator operates outside the swarm, on the other hand the second type of communication is proximal interaction, this type assume that the human operator and the swarm share the same environment [42]. Majority of the research focus on remote interaction despite the many technical difficulty facing it, but at the same time it’s the ideal method for operating in dangerous area for humans, which is the main objective of swarm robotics in the first place. One example of centralize control can be found in study [43], in this study the swarm consist of 112 robots with the gate-way robot its main objective is to receive programming from the operator and podcast it to the swarm, on the contrary the centralized operator interface with the gate-way robot to receive information about the state of the swarm. One major problem in the communication domain is the bandwidth limitation which explored in [44], the study examined three bandwidth conditions (low, medium and high) in coordination tasks and the result as follow: the low bandwidth condition only a single robot communicate with the operator in single time step. The medium condition centralizes the location information in a single robot then send it to the operator. In the high condition all the robots in the swarm communicate with the operator in a single time step. The proximal interaction allows the operation to observe the swarm as they share the same environment. The operator communicates in most studies by a gesture as face recognition [45] or speech [46] recognized by the robot and then act upon according to the commands attached to the exact gesture. One example of this type of communication interaction in the GAURDIAN project [47] which was implemented in firefighting Sicario's. Generally, the proximal interaction is not focus on by the literature as it shows shortcoming when it comes to control a large swarm.

2.3. MULTI-ROBOT COORDINATION:

Another unavoidable topic in the domain of (MRS) is the concept of motion coordination. We can define coordination in multi-Robot system as any collective and cooperative behavior from two or more robots to achieve certain task according to specific algorithm. The research in this topic have been very extensive and well documented, although the real challenge seems to be the physical demonstration of these studies. Some of these well documented studies in coordination includes: path planning [48] , formation changing [49], traffic control [50], target tracking [51]. All these topics is well studied and understood in the simulation area rather than the physical real-life applications.

the coordination movement of (MRS) is based on algorithms inspired by nature. These algorithms are: Bee Colony Optimization (BCO) [52], Ant Colony Optimization [53], Firefly Optimization [54], Bat Optimization [55], Cuckoo Optimization [56] and Particle Swarm Optimization and the later been the most used and research optimization methods since its proposal by James Kennedy and Russel Eberhart in 1995 [57], in this study the researchers presented an algorithm that mimic the social behavior of swarms in nature (like ant and fish) by continually updating the velocity and victor movement of the swarm to better the state of the swarm. Taking it a step further the study [58] offer an improvement of the original (PSO). The study [59] implemented the (PSO) on a number Elisa-3 robot on Webots simulator, the result show that the robots trying to gather around the pre-assigned supervisor robot. A comparison between Particle Swarm Optimization (PSO) and genetic algorithm (GA) was made in study [60] , the study was performed on experiment robot behavior based on neural learning animat and the result show that (PSO) outperform the simple (GA) on simple task with neural network learning, also the result compares to a previous study [58] which did the same thing by training with (PSO)and (GA) while adding noise resistance modification and similar to [59] (PSO) shows a superior performance compare to (GA).

On the practical side the study [61] propose a mini-robot well suited for multi-Robotic application, the robot equipped with a stereo camera for environment monitoring and image processing applications, for data exchange between the robot, in addition the study uses three serial buses (SPI, I2C and UART). The study [62], there are two level of control a lower and higher level, as the lower level the control is executed by PID and CNN for more complex situation. All this ran on a simulation successfully. In addition, the study [60] propose a more flexible formation than a traditional one by [63]. For industrial application the study [61] propose [64]. Results show that these methods can improve the operating efficiency of (MRS) in industrial applications. For switching formation, the study [65] uses GOACM. A leader robot is responsible for path planning and guiding the follower, while the follower switch in obstacle avoidance mode. The proposed system executes successfully in both simulation and physical. The common idea is to control one team of robot but in 2005 the study [66] proposed (VOMAS) or Virtual Operating Multi-Agent System to control several robot teams. The structure of the proposed system in divided into two main parts, the first one is the user agent handle the high-level control, the second is the robot agent handling the low-level control. The main benefit of this system is that possibility of adding robots to the system.

Flocking is a known trait of (MRS) which enable the robot team to cohesively put maintain a safe space to avoid collisions. In [67] the flocking method handle all the objects as obstacles whether it’s really an obstacle or another robot in the robot team, the researcher argue that this method cut the amount of information exchange to the bare minimum as the information of position and velocity is not required. In the opposite side the study [68] required a connectivity between the robots since the study propose a leader-follower scheme and the leader act as the center of the flocking. In addition, the study combines the flocking control with reinforcement learning to make the robot team to flock while avoiding obstacle and maintain connectivity. In summary the flocking seems to rely on decentralized control, but the challenge seems in the rising complexity of the foraging task since the team need to coordinate and avoid obstacles while maintain the connectivity between all the entities in the (MRS).

In the cooperative manipulation field, we can't fail to mention the box-pushing problem which became very synonymous with (MRS) early researchers. One of these researches is [69], which proposed the moving of objects from place to place by a group of mobile robots. The robot team consist of two group: steerer robots programmed with path tracking and push robots to exert force into the object. In addition, the study [70] propose a formation control method to transport an object where a group of robots surround the object to move it cooperatively. Contrary to the studies and [69] [70] the AVERT project proposed in study [71] require a strong and stable connectivity to exchange path planning with the command base. AVERT is used to move and extract vehicles from one location to the other by using a lifting robot unit equipped with trajectory planning and object detection.

A very promising and relatively new area is cloud robotics, which can be defined as a mixture of robotics and cloud computing. The advantages of cloud computing can be incorporated in robotics to help agents to complete tasks in their environment by designing a cloud-base system to enhance the effectiveness of the system. One of the very first research in this area is RoboEarth [71] and DaVinci [72], which both aim to offload the huge load of information on the board of the robots to a cloud-computing structure for maximum the effectiveness of (MRS).

2.4. FORMATION

Examining the mathematical model of natural phenomena has drawn the interest of the scientific community. Clustering behavior or the gathering of life forms in communities is abundant in nature. This clustering behavior can be observed in animal flocks, such as goose flocks, bird flocks, fish schools, mammal herds, and so on. Goose flocks usually fly in a reversed "V" formation. Flying in a formation like a reversed "V" provides several advantages. Some of these advantages include 24% more flying power for each goose compared to flying alone and 71% greater flight range [73]. The control of the formation and the alignment of the generated formation points with mobile robots are necessary. The assignment process aims to reach the formation in the minimum time and without collisions. The Hungarian algorithm [74] is used to solve discrete (combinatorial) problems. The formation control process can be classified according to interaction topology as position, displacement, and distance control.

Turpin et al. (2004) proposed a decentralized method, called D-CAPT, as a solution to the Concurrent Assignment and Trajectory Planning (CAPT) problem for swarm robots. The C-CAPT algorithm is the most suitable solution to the CAPT problem in a collision-free environment. In their approach, they developed the C-CAPT algorithm, which is a centralized solution for the assignment and route planning problem, reducing the cost function based on the square of velocity during the trajectory. They successfully applied this algorithm to eight quadrotor micro aerial vehicles and reported that the trajectories were globally optimal and safe [75].

Gazi et al. (2007) developed strategies using the sliding mode control method and artificial potential functions for clustering and formation control problems in swarms. They detailed their work by applying the same approach to tasks such as food searching, formation control, and tracking moving targets through simulation studies, which were successfully accomplished [76].

Yao et al. (2006) presented a decentralized stable control method for swarm agents to achieve formation control and follow a moving target. They utilized artificial potentials for target tracking and formation control processes. Through simulation tests, they demonstrated that their proposed method was more stable than the leader-follower model [77].

Tanner et al. (2004) examined the stability properties of mobile agent formations based on leader following. They created nonlinear gain estimates of how inter-agent distance errors affected leader behavior. They proposed a method to improve the Leader Formation Stability (LFS) gains [78].

Miswanto et al. (2015) achieved formation control for swarms by having agents follow the path of a leader. They used the Pontryagin Maximum Principle method for leader control and a geometric approach for formation control, showing that each agent's orientation and position could be controlled relative to the leader [79].

Desai et al. (2001) developed control strategies using graph formation control for non-holonomic mobile robots in environments with obstacles. Their approach aimed to maintain the desired formation and enable formation transitions when needed [80].

Xie et al. (2000) studied the natural algebraic structure of the chained form system along with ideas from the sliding mode theory while designing control laws. They reported that the sliding mode approach needed to be considered for the stabilization and tracking problem of the system called "chained system" from non-holonomic systems [81].

Mancini et al. (2007) proposed an approach for solving the problem of following a leader in a swarm of equal-numbered robots. They used a discrete-time sliding mode approach for controlling non-holonomic robots performing the task of following a leader [82].

Pranoto et al. (2012) performed simulation studies of the leader-following algorithm for formation control. They tackled the best tracking control problem for a swarm with a specific geometric formation using the Dubin's car model. They created a model of three agents and a swarm leader, showing that the agents moved to follow the leader's path, and the resulting error during tracking was very low. They demonstrated that the position and orientation values of each swarm member (agent) were controlled relative to the swarm leader [73].

Mısır et al. (2020) proposed a clustering method for swarm robots, which involved homogeneous robots clustering with limited distance sensor and angle data without central control. They noted that the collision avoidance controller was separate from the clustering controller. Each robot made individual decisions during the clustering process, following the nearest perceived robot's movements, and directed itself based on the motion of the other perceived robots. They reported the successful application of the clustering behavior and that performance decreased as the number of robots increased [83].

Sial et al. (2021) developed a new search and mission execution model for the control of distributed formations of UAVs (Unmanned Aerial Vehicles). In the first stage of their study, they aimed to find solutions to swarming, collision avoidance, and tracking problems for distributed UAV formations using multiple artificial potential fields and agent graph theory. The proposed algorithm's first stage involves target search and mission execution operations, while in the final stage, swarming is applied during the mission process. They reported the successful testing of swarm swarming operations and mission execution algorithms in a simulation environment[84] .

Mechali et al. (2021) proposed a new control method for UAVs that creates a leader-follower interaction and handles non-linear behaviors with continuous non-vanishing disturbances. They designed a control law for a distributed formation. The formation's formation points were determined, and a reference formation trajectory was created and followed using a synthesized fixed-time position control method. They conducted simulation studies in the ROS/GAZEBO environment to analyze the control performance, and they reported that their approach showed higher performance compared to other control methods [85].

In their study, Gauci et al. (2014) proposed a controller as a solution to a self-organizing aggregation problem that does not require complex computations. They used a "if-then-else" structure as the controller, enabling the swarm to self-organize during aggregation. The research focused on collective gathering behavior. They employed 2-bit sensors for their swarm robots to detect their neighbors. Initially, they set the detection range of these sensors to be unlimited. However, through experiments, they varied the detection range to examine its impact on the aggregation behavior and controller performance [86].

Mısır et al. (2020) proposed a fuzzy logic-based self-organizing aggregation method in their study. Unlike traditional clustering methods, they evaluated limited sensor data using fuzzy logic as the controller. They conducted regular experiments in a simulation environment with swarm robots having different detection ranges, different numbers of robots, and various field sizes. They reported that the swarm robots exhibited clustering behavior despite changes in the detection range and the number of robots during the regular experiment phase [87].

Parhizkar et al. (2020) took inspiration from the clustering behavior of the social amoeba Discotyostelium Discodeum and applied this biological example to swarm robots. They examined the signal propagation model of the biological aggregation behavior in a simulation environment. In their study, they demonstrated the biological aggregation behavior in physical swarm robots called "kilobots" [88].

PART 3

THEORETICAL BACKGROUND

3.1. SWARM INTELLIGENT ALGORITHMS

Since the dawn of man, the idea of absorbing nature and even borrowing from it is a well-established practice. Humans always looking to nature for solving complex problems they face, and nature always directly or indirectly provides the perfect solutions, since nature faced through millions of years variety of challenges, it’s only natural for human to come with nature-inspired algorithms for real-life applications and solutions. One of these algorithms is the swarm intelligent (SI) which created by absorbing the different colonies and collections of living beings in nature like: bees, ant and bats [89].

The concept of swarm intelligent (SI) has gathered a lot of attention in the researching filed. We can define SI as a collective behavior based on collective intelligent in self-organized and decentralized system. Another definition was provided by Bon beau [80]: “The emergent collective intelligence of groups of simple agents”. Two main concepts are the properties of SI according to [90]: self-organization and the divining of tasks. Self-organization is defined as the ability of the system to organize his its agents without an external force. This idea relies on four main fundamentals: positive feedback, negative feedback, fluctuations, and multiple interactions. In addition, the second concept is the task division which is defined as the parallel execution of simpler sub-tasks by agents which allow the system to execute a complex task as the bigger objective.

The Nature-Inspired Algorithms (NIA) showed a great result; therefore, a wide variety of algorithms has been discussed in the literature (up to 140 algorithms), however such a wide range of algorithms could lead to confusion around the researcher regarding which algorithm work and which doesn't based on mathematically approach. Every (NIA) mimic a unique natural process, some of these algorithms are well-known and established like Genetic Algorithms, Particle Swarm Optimization (PSO) and Ant Colony Optimization

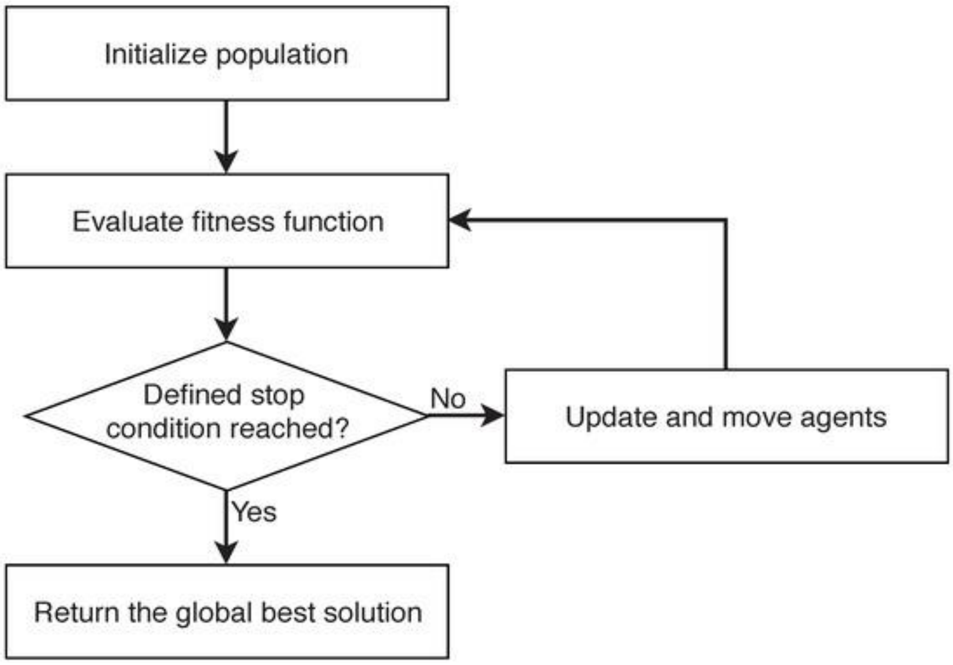

Initialize population
Evaluate fitness function
Defined stop condition reached?
No
Update and move agents
Yes
Return the global best solution

Figure 3.1. Swarm intelligence framework [91].

In the addition other algorithms are not at the same level as the previous mentioned algorithms for reasons such as: under-development or efficiency etc. In this chapter we will explore mainly the most known algorithms and will discuss the lesser-known algorithms briefly [92].

In this section we will explore different types of (NIA)s or SI-based algorithms highlighting their main properties and applications. These algorithms are:

- Genetic Algorithms (GA).
- Particle Swarm Optimization (PSO).
- Ant Colony Optimization (ACO).
- Artificial Bee Colony (ABC).

- Glowworm Swarm Optimization (GSO).
- Other Evolutionary Algorithms.

3.1.1. Genetic Algorithms (GA)

Introduced in 1975 by John Holland [93], the basic concept of (GA) is to mimic the natural process called survival of the fittest which a mechanism of selecting the strongest options among the population. In GA a set of population is alter for the best by crossover and mutation by following these steps:

- The Generation of a random set of population, this population can be represented by a set of strings called chromosomes.
- Ranking the population by performing a calculation on the chromosome based on the fitness function, the value of the fitness function is what determine the process of the selection between the population.
- The operation of reproduction is to select the best candidate of the population according to the fitness function,
- Performing the crossover which basically to crossover the fittest candidate to the next generation of the population.
- Performing the mutation which to alter some gene of the chromosome to fit the next generation of the better population.
- Replacing the old population with the new and better population [94].
- Check if algorithm achieve the goal if not repeat the operation until achieving the desired goal.

3.1.2. Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is one of the most used and research optimization methods since its proposal by James Kennedy and Russel Eberhart in 1995 [57]. PSO is an SI-algorithm that mimic the social behavior of swarms in nature (like bird and fish) by continually updating the velocity and victor movement of the swarm to better the state of the swarm. PSO can be describes by three main behaviors according to as shown in figure (9):

- Separation: is a non-collision behavior between the swarm.
- Alignment: is behavior of maintaining space and velocity between the swarm
- Cohesion: is the collective behavior of moving toward and united direction.

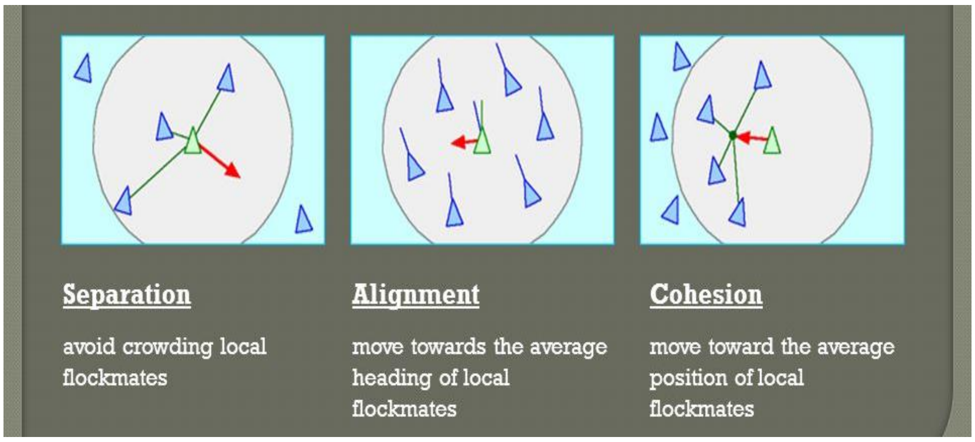

Separation
avoid crowding local flockmates
Alignment
move towards the average heading of local flockmates
Cohesion
move toward the average position of local flockmates

Figure 3.2. the three main behavior of PSO.

On the mathematical side PSO can be represented by the following formula [95]:

$$v_{\mathrm{id}}^{t+1} = v_{\mathrm{id}}^{t} + c_1 \cdot \mathrm{rand}(0,1) \cdot (P_{\mathrm{id}}^{t} - x_{\mathrm{id}}^{t}) + c_2 \cdot \mathrm{rand}(0,1) \cdot (P_{gd}^{t} - x_{\mathrm{id}}^{t}) \quad (3.1)$$

$$x_{\mathrm{id}}^{t+1} = x_{\mathrm{id}}^{t} + v_{\mathrm{id}}^{t+1} \quad (3.2)$$

Where:

v_{id}^{t+1} : the velocity of the particle

x_{id}^{t+1} : the position of the particle

d: dimension

i: the particle index

t: the iteration number

C1 and C2: the speed regulating the space.

rand (0,1): random value between 1 and 0

Pi: the best position by particle i

Pg: the best position by neighbor particle

PSO flow chart of processing as follow: first the PSO utilize the population, the higher number of the population the better the result, calculating for each particle the fitness value, thirdly updating the velocity and positions for each particle, lastly continuing the process until the desired goal achieved [95].

To improve the performance of PSO researchers use many approaches, for example using a higher number of populations which lead to faster convergence. Other approaches also introduced like balancing between exploration and exploitation and using a sub-swarm to increase the efficiency of the PSO which is a common approach these days [96]. Continuing in the PSO performance improvement, Shi and Eberhart [97] introduce the inertia weigh[98]t (w) as a new variant in the PSO equation. According to the (w) value the process of exploration and exploitation will occur, if (w) is high this led to exploration behavior and if its low it will lead to exploitation behavior. The new proposed PSO equation as follow:

$$v_{\mathrm{id}}^{t+1} = w.v_{\mathrm{id}}^{t} + \ c_1 \cdot \mathrm{rand}(0,1) \cdot (P_{\mathrm{id}}^{t} - x_{\mathrm{id}}^{t}) \quad + c_2 \cdot \mathrm{rand}(0,1) \cdot \left(P_{gd}^{t} - x_{\mathrm{id}}^{t}\right) \quad (3.3)$$

the introduction of (w) in the PSO equation was an improvement in the speed of convergence, later the study [98] suggest to increase the value of (w) higher than 1 to encourage the exploration in the early stages and reduce the value less than 1 to find the best exploitation toward the end. Later the study [99] propose the K factor to lower the possibility of convergence and the particle leaving the searching area. The new equation with the addition of k factor as follow:

$$v_{\mathrm{id}}^{t+1} = K[v_{\mathrm{id}}^{t} + \ c_1 \cdot \mathrm{rand}(0,1) \cdot (P_{\mathrm{id}}^{t} - x_{\mathrm{id}}^{t}) \ + c_2 \cdot \mathrm{rand}(0,1) \cdot \left(P_{gd}^{t} - x_{\mathrm{id}}^{t}\right)] \quad (3.4)$$

in summary both factors (w) and (K) show an improvement in the performance of basic PSO, and that according to study [91] which conduct a comparison between the two factors and their contribution on the basic performance of PSO.

3.1.3. Ant Colony Optimization (ACO)

ACO is an algorithm aim to mimic the natural behavior of ants in activities such as foraging. ACO propose as a PhD thesis by Marco Dorigo in 1992 [53]. ACO consist mainly of four components: first ant as the algorithm imaginary agents responsible for the exploration and exploitation. Second is the pheromones is a chemical component used by the ants as trail mark in their search for the target, the intensity of the chemical is a sign and indicator for the collective group and operate as global memory to the colony. Third is the daemon actions is to gather global information for the trail and decides whether a more pheromones is needed or not, this must be done as a formation group and not as a single agent. The fourth and last element In ACO is decentralized control which provide a false tolerance mechanism against the failure of ne agent. Figure (10) explain the process of ACO in three steps: first, the ants move randomly back and forth between the colony and the source of food. Second, the ants discovered multiple paths to the target. Lastly, the shortest path is chosen which lead to the ants replacing the pheromones to indicate the path for other ants and that leads to the ant colony using the selected path.

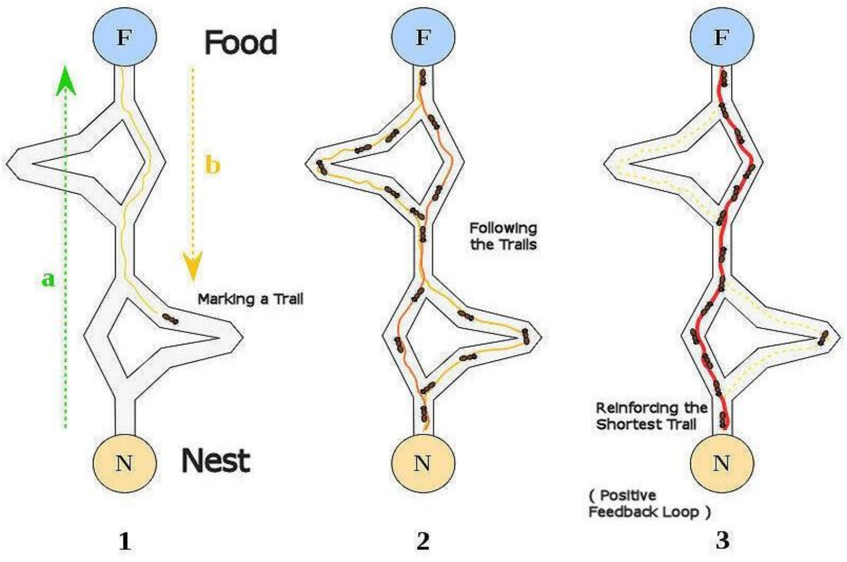


Figure 3.3. Ant colony optimization.

Mathematically ACO for finding the best paths is represented by the following equation:

$$P_{(ij)}^{k}(t) = \frac{\tau_{ij}(t)^{2}[n_{ij}]^{\beta}}{\Sigma_{k\epsilon Jk}\left[([\tau_{ij}\ (t)\right]^{\alpha}.[n_{ij}]^{\beta}} \tag{3.5}$$

where:

$P_{(ij)}^{k}(t)$: the movement from node i to j node.
$\tau_{ij}(t)$: the array of nodes which agents allowed to travel to from node I visibility between I and j.
n_{ij} : amount of pheromone between i and j.
α and β: factors control the behavior of the colony, depending on their values.

The deposing of the pheromone is depended on the following formula:

$$\varDelta\tau_{ij}^{k}(t) = \sum_{0}^{\frac{Q}{L}} k^{(t)} \tag{3.6}$$

where:

Q: constant
L: The length of the path
t: iteration number
K: ants

the amount of pheromone determines the behavior of the ants whether they will have an exploration behavior or exploitation, too high pheromone amount will result in ants getting lost and too low amount will lead to ants not finding the optimal path.

3.1.4. Artificial Bee Colony (ABC)

ABC was one of the most recent algorithms introduced in 2005 by Dervis Karaboga [52]. ABC is very simple and easy to apply like PSO and DE, the algorithms inspired by the natural behavior of the bee's colony when it comes to find food sources and share information of the sources between the bees. The algorithms depend on three types of bees, every type has its own function in the algorithm. The three types are: first, the employee which their job is to find the source of food and save the information in their memories. The second type is the onlooker which to receive the information from the employee and pass it to the scout bees to gather the food from the nectar.

The whole process of ABC can be summarized in the following steps [100]:

Step1. All the control parameters are set, the scout bees are initializing the food source, which is represented by vector of the population xi, the vector xi has n variables and been optimized by the higher value of the bound xi represented ui by and the lower value of xi bound represented by li. the previous mentioned phase is represented by the following formula.

$$x_i = l_i + \text{rand } (0,1) * (u_i - l_i) \tag{3.7}$$

Step2. The searching phase by the employee bees which the value of the new food source vi is increased for the purpose of having food around the near neighborhood of the previous food source xi. the new source of food is defined by the following formula:

$$v_i = x_i + \phi_j(x_i - x_j) \tag{3.8}$$

Where:

xj: is a random selected food source
Øi: random number in [a, -a]

after the value of equation (7) and (8) is produced a greedy evaluation happened between the two values. The behavior of exploration and exploitation happened according to the value of xi- xJ. if the value if high then is exploration happening where if the value is low the is exploitation happening. the fitness operation is conducted by the following equation:

$$f\mathrm{i}^{x}(\overrightarrow{x_i}) = \begin{cases} \frac{1}{1+f_i(\overrightarrow{x_i})} \quad if\ f_i(\overrightarrow{x_i})\ more\ or\ equal\ to\ 0 \\ 1 + abs\left(f_i(\overrightarrow{x_i})\right)\ if\ f_i(\overrightarrow{x_i}) more\ than\ 0 \end{cases} \tag{3.9}$$

where:

fi(xi) is the objective function of the solution xi

Step3. The onlooker bee phase. Here the onlooker is waiting for the information from the employee bees and fitness value calculation in equation (9) to calculate the probability value based on the following equation:

$$p_i = \frac{f\mathrm{it}_i(\vec{x}_i)}{\Sigma_{\mathrm{i}=1}^{SN} f\mathrm{it}_i(\vec{x}_i)} \tag{3.10}$$

Step4. The scout bee phase which a certain food sources is terminated due to its unimproved fitness value through the iteration, this called abandoned criteria.

Step5. All the information, positions and fitness value are saved and memorized.

Step6. The termination of the program in case the conditions are met, if not the process continue from step2 to step6 until the condition are met.

3.1.5. Glowworm Swarm Optimization (GSO)

A relatively new proposed SI-algorithm by Krishnanad and Ghose in 2005 where physical the agents in GSO are called glowworm[101] . this algorithm has three main parameters changing over time [99]: position in the search space (xm(t)), luciferin level (lm(t)) and a neighborhood range (rm(t)). Three phases are repeated according to the three previously mentioned parameters until the condition and the program is terminated [100].

The first parameter position in the search place calculated by the following formula:

$$x_m(t) = x_m(t-1) + \ \ s(\frac{x_n(x_n lt-1) - x_m(t-1)}{x_n(x_n lt-1) - x_m(t-1)}) \tag{3.11}$$

where s represents the step size. According to the difference value between xn and xm the behavior is determined, if the value is high the exploration behavior occurs, in the contrary the exploitation occurs.

After obtaining the xm(t) parameter the update luciferin[89]level lm(t) is calculated by the following formula:

$$l_m = (1-p) \cdot \mathrm{Im}(t-1) + \gamma J\big(x_m(t)\big) \tag{3.12}$$

Where:

p: is the luciferin factor
γ: the luciferin constant
J: the objective function.

the glowworm m is considered a neighbor to glowworm n under the condition that the distance between m and n is shorter than the neighborhood rm(t). but in case there are multiple choice close to the glowworm m the probability is calculated via the following equation:

$$P_m(t) = \frac{\mathrm{Im}(t) - \ln(t)}{\Sigma_{k \in N_i(t)}\, l_k(t) - I_n(t)} \tag{3.13}$$

the glowworm with the highest probability is chosen.

The final step is to update the final parameter which is the range of the neighborhood rm(t). this updates the range of the communication between the glowworm community.

The following formula is used to calculate rm(t):

$$r_m(t+1) = \min\{r_s, \max[0, r_m(t) + \beta(n_d - |n_m(t)|)]\} \quad (3.14)$$

where:

rs: the sensing range nd: number of neighborhoods. nm: number of neighborhoods.

3.1.6. Other Evolutionary Algorithms

While the previous sections focused on introducing and discussing well-known and commonly used SI-based approaches, it is important to acknowledge the existence of several other interesting evolutionary algorithms. These algorithms have unique characteristics and applications that set them apart from the ones previously discussed. In this section, we will provide a brief overview of some of these algorithms:

Like Genetic Algorithms (GA), GP follows a series of steps involving initial population creation, fitness evaluation, selection, and reproduction. However, GP uses the term "program" to represent the solution instead of "chromosome" used in GA. The key distinction lies in the selection procedure. While GA selects predefined percentages of the fittest population for reproduction, GP allows each program to select one or a few programs from the population based on their fitness probabilities [102].

EP shares some similarities with GA in terms of initialization, mutation, and evaluation operations. However, the main difference lies in the absence of crossover operations in EP. Instead, EP relies on stochastic selection, where solutions compete against a predetermined number of other solutions, and the least-fit solutions are eliminated [103].

ES is an optimization approach that shares methodology similarities with GA and Differential Evolution (DE). However, ES introduces self-adaptive mutation rates, which enhance its efficiency. There are three types of ES: (1+1)-ES, (1+λ)-ES, and (μ/ρ +, λ)-ES. Each of these types involves specific rules for generating mutants and selecting new parents for the next generation[104].

Inspired by the behavior of fireflies, the FA algorithm involves using flashing light to attract each other. Fireflies' fitness determines the brightness of their flashes, which decreases over distance. Fireflies move towards brighter ones, and if there are none, they move randomly[105].

Introduced by Yang and Gandomi in 2012, the Bat Algorithm draws inspiration from bats' foraging behavior. Similar to Particle Swarm Optimization (PSO), it consists of velocity and position equations. The algorithm also incorporates echolocation capabilities and a frequency equation to influence the velocity equation, determining the search direction [55].

Inspired by the predatory behavior of grey wolves, the GWO algorithm organizes agents (wolves) into a hierarchy of alpha, beta, delta, and omega categories. Each hierarchy has distinct roles in finding solutions, resembling preys in this context [106].

It is worth noting that the above algorithms represent just a few examples of the vast array of evolutionary algorithms available. There are many more techniques that have not been discussed in this section.

3.2. SOFTWARE PLATFORM

3.2.1. Robotic Operation System (ROS)

Writing robotic software could be a very complex process especially in the modern robotic application which demand a complex structure from the simple driving software to the computer perception, logic, and reasoning. Therefore, a swarm of experimentations and software has been introduced to meet the ever-growing complexity of robotic applications.

Robotic Operation System (ROS) is not an operation system in the typical sense but rather a framework for robotic programming which include a variety of tools and liberties to simplify the complex robotic behavior for both programmers and users. Dealing with real life robotic problems can be a very taunting task for induvial and even institutes, therefore (ROS) provide the ground up solutions instead of from scratch method of programming or problem solving [107].

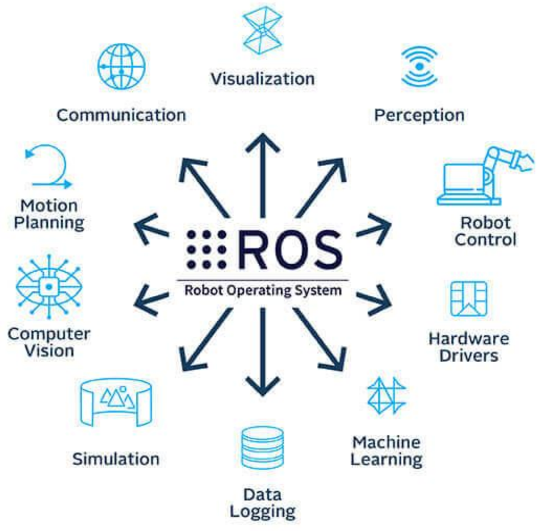


Figure 3.4. ROS as a framework for all applications.

In summary the ROS framework provide the following:

- A structure commination field.
- Passing massages between the nodes and processes.
- The management of the packages.
- Low-level device control.
- Hardware abstraction.
- A variety of tools and libraries.

The communication protocol in ROS is a peer-to-peer communication combined with the ROS communication infrastructure, therefore the process of data transportation between the machines utilizes different mechanisms. The main concept of ROS is presented in different mini concepts which are: nodes, massages, topics, and services [112].

Nodes: the ROS node is basically a software with a computing power which increase the ROS fault tolerance by dividing the bigger system into small, organized nodes which are communicate with each other via peer-to-peer communications. different nodes can utilize different technologies and software as long as they communicate via ROS mechanisms. rosnode is the ROS command assigned to showing the information about any nodes. Every node has a node type aligned with a class assigned to specific function or task.

Massages: communicate the nodes with each other. The standard massage type (integer, Boolean and float…etc.) are supported in ROS. Every massage is identified in a simple massage type which show the type of the massage.

Topics: is just the name that identified the massage flow between the nodes, every node in ROS either a publisher or subscribers. When a node needs a massage, it called subscriber node and request information from the topic and subscribe to it and then receive the massage from a publisher node. The publisher and subscriber nodes are not aware of each other exitance but rather communicate via the topic. A node is either a publisher or subscriber, although a single topic can be assigned to by different subscriber and publisher. rostopic is the command in ROS to show all the information about a certain topic like the massage periods, the number of the subscriber and publishers and the size of the information traffic in Bytes. The previously stated flow of information between nodes is shown in Figure 3.5.

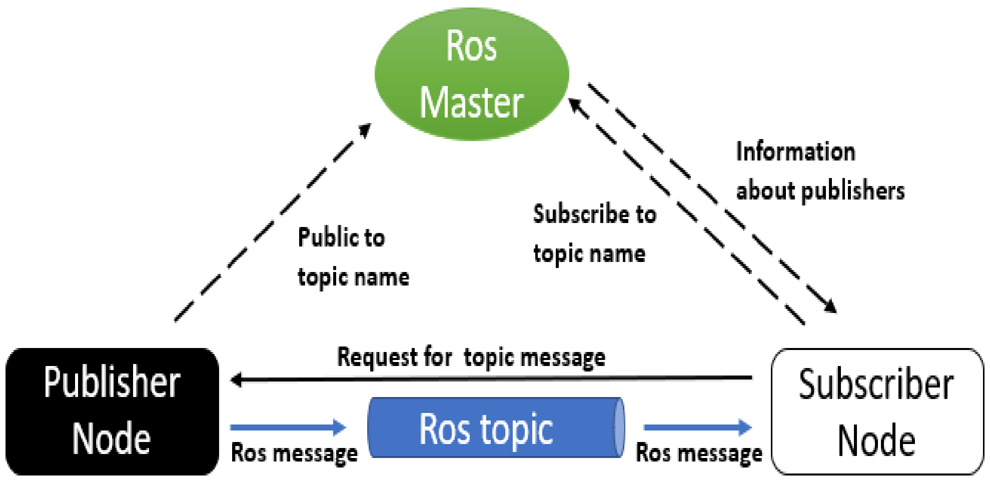


Figure 3.5. the flow of the massages between nodes.

Services: in ROS services is a request/reply mechanism for the unidirectional movement of the ROS massages between the nodes. ROS service is defined by two set of massages one foe request and other for reply. Every node can offer a service under the name of client. rossrv and rosservice are the ROS command line tool to display information about services [108].

3.2.2. Gazebo

Gazebo is a 3D sensor-based simulator suitable for indoor or outdoor simulation and compatible with the ROS nodes by utilizing plugin presentable as the same massages interface in the ROS structure. The Gazebo simulator provides realistic sensor feedback and physical interaction between the robots and their environment. The main components of the Gazebos simulator explained simply as follow:

- World File: an SDF file (Simulation Description Format) Contains all the elements in a simulation.
- Model File: also, SDF file used to describe a single model.
- Environment Variables: For storing environment, communication settings.
- Gazebo Server Client: The two main components of a simulation.
- Plugins: A simple mechanism to interface with the simulation world [109].

3.2.3. RVIZ

RVIZ which stand for ROS Visualization is a powerful 3D visualization tool in ROS that allows users to view and interact with various types of sensor data and robot-related information in a 3D environment. Here are some key points about Rviz and its functionalities:

- Visualization of Sensor Data: Rviz allows to visualize sensor data, such as camera images, laser scan data, and point clouds. For example, if you have a robot model equipped with a Kinect sensor in Gazebo (a robotic simulator), you can visualize the laser scan data generated by the Kinect sensor in Rviz.

- Mapping and Navigation: Laser scan data, when processed appropriately, can be used to build a map of the environment. This map can be utilized for autonomous navigation or other purposes.
- Frames: In Rviz, the concept of "frames" refers to coordinate frames used in the robot's kinematic chain. Different components of the robot may have their own coordinate frames, and RVIZ helps in visualizing data with respect to these frames.
- Displays in RVIZ: RVIZ provides various display types, allowing users to view data from different sensors. Some common display types include Grid Display, Laser Scan Display, Point Cloud Display, Camera Display and Axes Display
- Adding Displays: Users can add displays to Rviz by clicking on the "Add" button and selecting the type of data they want to visualize. For example, if you want to visualize laser scan data, you can add a Laser Scan Display, and if you want to see the camera feed, you can add a Camera Display.

Overall, RVIZ is a valuable tool for roboticists and developers working with ROS, as it allows them to visually inspect and debug the robot's sensor data and other relevant information in a 3D space [110].

PART 4

METHDOLOGY

4.1. SYSTEM OVERVIEW

In response to the escalating demands within the realm of robotics applications at large, and the intricate domain of multi-agent systems in particular, this thesis presents an innovative proposition: a dynamic ROS2 multi-agent framework tailored explicitly for fostering coordination and facilitating cooperative applications. Since its inaugural introduction in 2007 by Willow Garage, Stanford Artificial Intelligence Laboratory, and Open Robotics, the Robotic Operating System (ROS) has soared in popularity, captivating the imagination of robotic researchers and developers alike. Its allure can be attributed to a constellation of attributes: a meticulously crafted modular architecture, seamless abstraction of hardware intricacies, a fabric of standardized communication threads, a spirited and thriving community, immersive simulation capabilities, pedagogical significance, programming dexterity, adeptness in distributed computing environments, a suite of diagnostic aids, and an ethos of open-source collaboration. This confluence of factors renders ROS an unrivaled choice, an indispensable toolkit for the inception, trial, and coalescence of novel robotic systems.

Furthermore, as the sun sets on ROS and the dawn of ROS 2 emerges, the community reaps even greater rewards. With amplified performance benchmarks, fortified communication protocols, and a more streamlined development pipeline, ROS 2 epitomizes progress and ingenuity. Addressing the previous iteration's limitations, ROS 2 unfurls as an even more tantalizing prospect for those intrepid researchers and developers poised on the precipice of innovation, seeking a vehicle to architect the next generation of advanced robotic systems.

While it is widely acknowledged fact that multi-robotic systems confer distinct advantages over their single-robot counterparts, as elucidated in the preceding sections of this thesis, it is imperative to acknowledge an intrinsic complexity hurdle that multi-robot systems grapple with. This complexity stems from the dearth of comprehensive data and computational resources requisite for tackling intricate control and optimization conundrums. Notably, challenges such as formation control, task allocation, and model predictive control (MPC) burgeon in complexity. The intricate nature of these control paradigms is exacerbated by the absence of holistic knowledge, localized computation, and effective communication channels. It is within this contextual tapestry that the proposed peer-to-peer, non-centralized framework shines forth—a beacon designed to emulate and meticulously experiment within a fleet of robots, thus infusing much-needed insights into the intricate choreography of multi-robot systems.

The proposed framework is fully written in Python and based on ROS2 platform. Facilitates the creation of various multi-robot applications like the encoding of distributed control algorithms, the establishing of agent-to-agent communication, the devising and planning of control strategies, and visually conducting experiments on robotic fleets by simulations using Gazebo while visualizing data through RVIZ.

4.2. ARCHITECTURE DESCRIPTION

To simplify the process, the framework is divided into three primary layers: The first layer, known as TGL (High-Level Layer Multi-Agent Communication Framework), manages peer-to-peer communication among agents and interfaces with the second control layer. The second and third layer forms a low-level layer focused on individual robot control. This layer comprises two key elements: Roboplanning and Robocontrol. Seamless communication between these framework segments, depicted in Figure 4.1, is essential for the overall functioning:

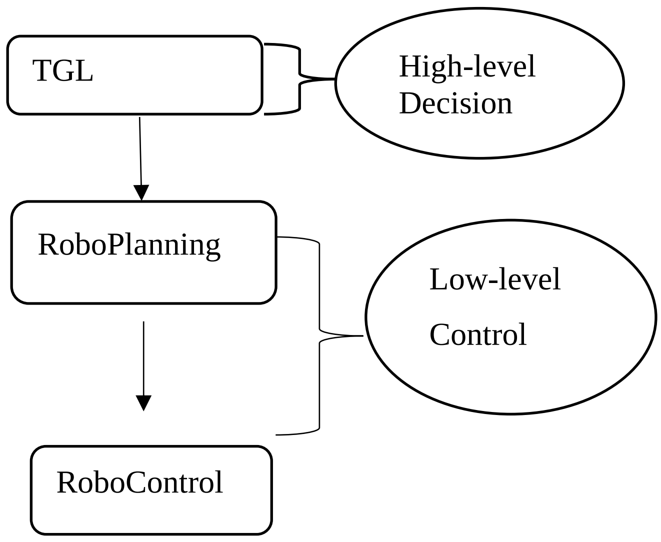


Figure 4.1. System overall flow-chart.

4.2.1. TGL

The Team Guidance Layer (TGL) stands as the central pillar within this software platform. A comprehensive examination of this layer is slated for the upcoming section; however, a brief preview is offered here. Primarily, the TGL revolves around the "Guidance" class, designed to capture fundamental data from each agent. It's important to note that this "Guidance" layer remains abstract, devoid of any reasoning or control logic, necessitating further extensions for functionality. In our instance, we've crafted two extension classes, each bearing distinct features and functions. The two extension classes intertwined with the "Guidance" layer are as follows: the "OptimizationGuidance" class, which encompasses optimization-oriented attributes, laying the groundwork for optimization-driven distributed control strategies like task allocation, optimal control, and model predictive control. On the other hand, the "DistributedControlGuidance" class comes into play for more straightforward applications like containment and formation control.

4.2.2. Robot Planning and Robot Control

Unlike the TGL layer, the trajectory and control layers operate in isolation. In other words, agents within this layer do not communicate. Each robot maintains independence in terms of planning and control, with communication responsibilities resting upon the layer. The planning layer is overseen by the "Planner" class, primarily functioning as a point-to-point planner within both 2D and 3D contexts. This "Planner" class synergizes harmoniously with controllers, steering and guiding robots towards designated points. Notably, this class suits both ground and aerial robots, ensuring versatility. On the flip side, the "Controller" class caters specifically to unicycle ground robots, laying the foundation for control strategies. Within the domain of mobile ground robots, a pair of unicycle controllers offer alternatives for reaching designated points or achieving specific velocities.To culminate this ecosystem, the platform achieves seamless integration with the Gazebo and Rviz simulation environments. Facilitated by the "RoboIntegration" class, this integration not only streamlines visualizations of distributed control and optimization algorithms but also enhances the overall experiential coherence. The overall ecosystem is graphically illustrated in figure 4.2:

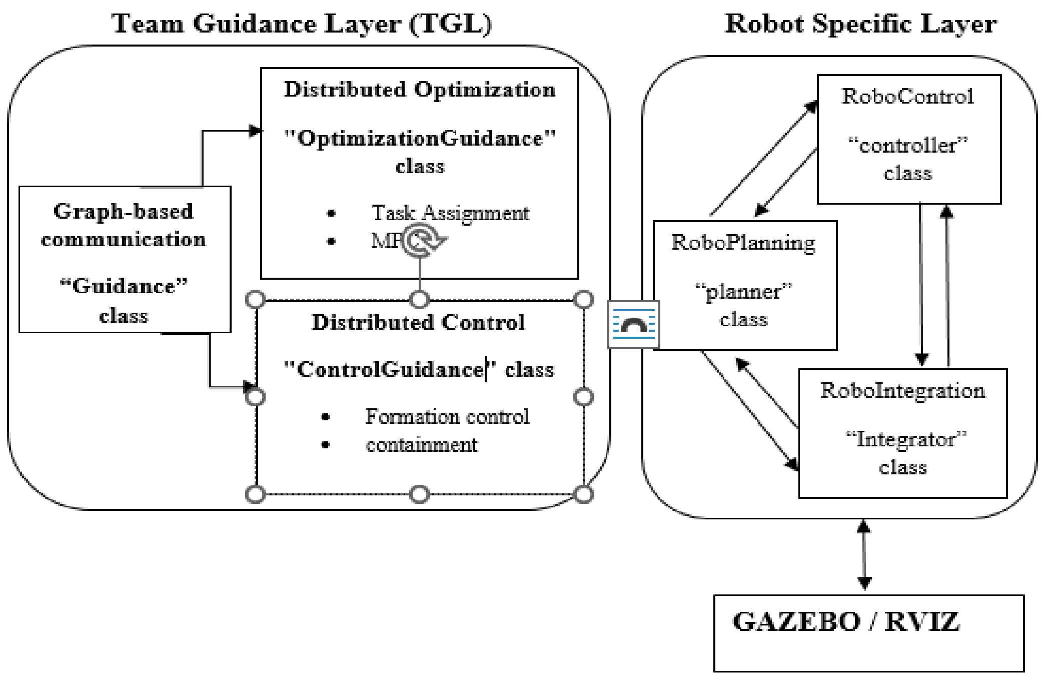


Figure 4.2. Framework architecture.

4.3. EXPLORING TGL

As highlighted earlier, the framework's foundation significantly hinges on the TGL, which stands as a pivotal and comprehensive cornerstone housing the entirety of its functionality. Therefore, this section endeavors to delve even deeper into this layer to glean a more profound comprehension. The TGL, in essence, comprises three central components: the first is graph-based communication, followed by the primary applications embodied within the OptimizationGuidance and DistributedControlGuidance classes. This exploration aims to offer a comprehensive grasp of TGL's intricate workings.

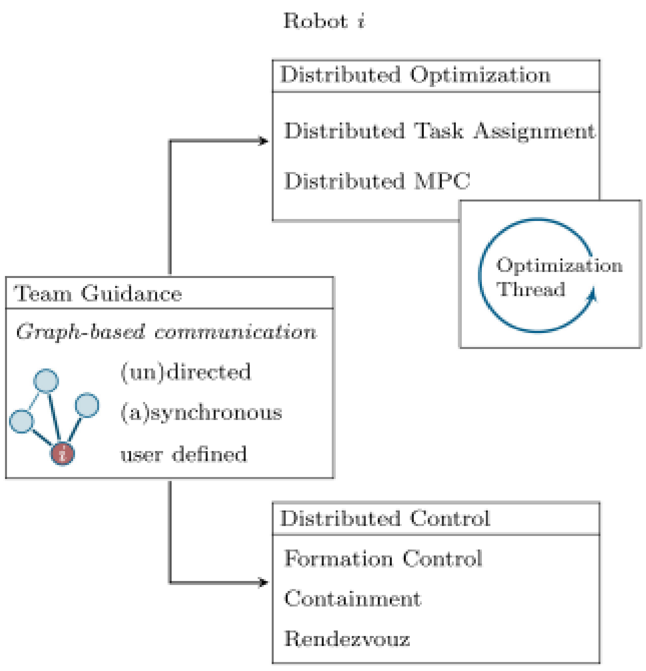


Figure 4.3. TGL Architecture.

The foundation of this layer predominantly rests upon the "Guidance" class, previously introduced as an abstract entity devoid of control logic and reasoning. The "Guidance" class forms the bedrock for the guidance level in multi-robot scenarios. It establishes the subsequent attributes:

- `agent_id`: Identification of the agent.
- `n_agents`: Total count of agents.
- `in_neighbors`: List of in-neighbors.
- `out_neighbors`: List of out-neighbors.
- `current_pose`: Present robot pose.
- `communicator`: Facilities for neighboring communication.

Extending from Guidance, the OptimizationGuidance surfaces as a subclass, augmenting the framework with optimization-centric attributes. It introduces the subsequent features: `optimizer` and `optimization_thread` classes.

4.3.1. Graph-based Communication

At the heart of distributed control and optimization lies Graph-Based Communication, where robots exchange information within a network structured as a graph. Nodes depict robots, while edges symbolize communication links, allowing decentralized coordination and data sharing for enhanced autonomy and teamwork in intricate settings.

The BestEffortCommunicator and TimeVaryingCommunicator classes are subclasses of the CommInterface class. They implement distinct versions of the neighbors_send() and neighbors_receive() methods.BestEffortCommunicator employs an asynchronous communication approach. This means that neighbors_send() and neighbors_receive() don't halt the calling thread; instead, they return immediately while messages are sent and received asynchronously in the background.

Conversely, TimeVaryingCommunicator adopts a synchronous communication mechanism. neighbors_send() and neighbors_receive() methods here pause the calling thread until all messages have been sent and received. TimeVaryingCommunicator also introduces neighbors_exchange(), synchronously exchanging information with neighbors. It forwards the send_obj to in-neighbors, then awaits messages from all out-neighbors before returning a dictionary of received messages keyed by the sender.

BestEffortCommunicator creates a publisher and subscription for each neighbor. The publisher transmits messages, while the subscription receives them. neighbors_send() serializes the send_obj and publishes it for each neighbor, while neighbors_receive() repeatedly invokes spin_once() of the rclpy.node.Node object until a message is received from each neighbor.

TimeVaryingCommunicator, akin to BestEffortCommunicator, differs in its QoS profile, ensuring reliable message transmission. neighbors_exchange() follows the pattern of sending send_obj to in-neighbors, then waiting for messages from out-neighbors before returning the received messages in a dictionary. These classes offer flexibility based on the context. They are beneficial for handling scenarios where communication links dynamically change due to factors such as limited range or energy consumption. Unlike standard ROS applications, explicit message type declaration is not required for robot communication. The std_msgs/ByteMultiArray message type, managed by the Communicator class and Python's dill package, allows versatile message exchange including vectors, matrices, text, and images. Importantly, changes can be made at runtime without predefined message types.

4.3.2. Distributed Optimization

This segment within the framework is devoted exclusively to the execution of optimization tasks. The main entry point for this division is the OptimizationGuidance class. It's widely acknowledged that achieving desired outcomes through optimization requires multiple iterations. In this context, the OptimizationThread is introduced as an iterative loop, separate from the main class. This design empowers users to customize the optimization process according to specific tasks. Serving as a foundational element for threads engaged in optimization tasks, the OptimizationThread class encompasses attributes such as:

- `_halt_event`: An event that triggers the halt of optimization.
- `_begin_event`: An event that initiates the start of optimization.
- `_quit_event`: An event that signifies the termination of the thread.
- `_lock`: A lock to ensure concurrent access protection to `_is_optimizing`.
- `_is_optimizing`: A flag indicating the ongoing optimization status.
- `_gc_end`: A Guard Condition activated at the end of each optimization.
- `guidance`: A Guidance object.
- `optimizer`: An Optimizer object.

The run method of OptimizationThread constitutes a loop that awaits either the `_begin_event` or the `_quit_event` to be set.

- If the `_begin_event` is triggered, the thread commences an optimization task.
- If the `_quit_event` is set, the thread discontinues.

The `do_optimize` method is abstract and must be implemented by subclasses. This is where optimization algorithms need to be added, as demonstrated later with task assignment and MPC implementation.

The framework facilitates modeling and problem-solving at two levels:

- Locally on the robot level.
- Globally on the network level.

On that note an integration between TGL and the DISTROP package [111] must be implemented. The DISTROP is a Python package for distributed optimization over peer-to-peer networks. The package allows users to define and solve optimization problems through distributed optimization algorithms. It offers a range of pre-implemented distributed optimization schemes and supports the semantic modeling of optimization issues. The compatibility of the Communicator classes with DISROPT streamlines the utilization of its distributed algorithms, requiring no additional adjustments.

4.3.3. Distributed Control

In contrast to the previously mentioned OptimizationGuidance class, the DistributedControlGuidance class is designed for simplicity and user-friendliness. This class serves as a foundational framework for communication and control. It commences by sharing the current position with neighboring robots, proceeds to calculate a velocity profile based on exchanged data and chosen feedback strategies, and ultimately publishes the control input to a designated topic. Despite its relative simplicity compared to the optimization class, this class's structure is highly capable of executing distributed control algorithms like formation control and containment. These functionalities will be implemented within the Gazebo/Rviz simulation environment.

The DistributedControlGuidance class inherits attributes from the Guidance class, which underpins basic functionalities for guidance systems.Functionally, the DistributedControlGuidance class implements a distributed control system. This system involves a group of robots communicating to exchange their position data. The robots then utilize this data to compute control inputs that maintain them in a desired formation.

The "__init__" method of the DistributedControlGuidance class takes specific arguments:

- "update_frequency": The rate at which the control law is evaluated.
- "pose_handler": The name of the ROS topic for publishing the robot's pose.
- "pose_topic": The ROS topic for subscribing to the robot's pose.
- "input_topic": The ROS topic for publishing the robot's control input.

The "control" method of this class is periodically invoked at the defined update frequency. It checks if the robot's current position is available; if not, it returns. Otherwise, it shares position information with neighboring robots, computes the robot's control input, and conveys it to the planner/controller.The "send_input" method broadcasts the control input to the specified ROS topic. The "evaluate_input" method is a placeholder, requiring implementation by subclasses. This method's purpose is to compute the control input for the robot.

In simple terms, the DistributedControlGuidance class establishes a distributed control system that sustains a group of robots in a desired formation. This involves periodic sharing of position information among the robots, which then use this information to calculate a control input for maintaining the desired formation. The update frequency dictates how often these processes occur.

4.4. THE IMPLEMENTATION OF DISTRIBUTED CONTROL AND OPTIMIZATIONS

Following the comprehensive exploration of TGL along with the involved Python classes detailed in the preceding sections, this segment will delve into the practical implementation of a distributed robotics scenario. This scenario involves a mobile ground robot platform named Turtlebot, simulated within the Gazebo environment and visualized using Rviz. The forthcoming implementation will encompass four distinct distributed scenarios, each having been predefined in academic papers and research:

- Containment in Leader-Follower Networks [112].
- Distributed formation control [113].
- Distributed Model Predictive Control (MPC) [114].
- Distributed dynamic task [115].

These scenarios are ranked by their level of complexity., with the initial implementation being the simplest and the final one being the most intricate.

4.4.1. Containment in Leader-Follower Networks for Single-Integrator Systems

Math problem: we examine a group of N robots navigating within the (x, y) plane. Each individual robot is represented as a single-integrator dynamical system, characterized by the equation:

$$\dot{x}i(t)=ui(t) \tag{4.1}$$

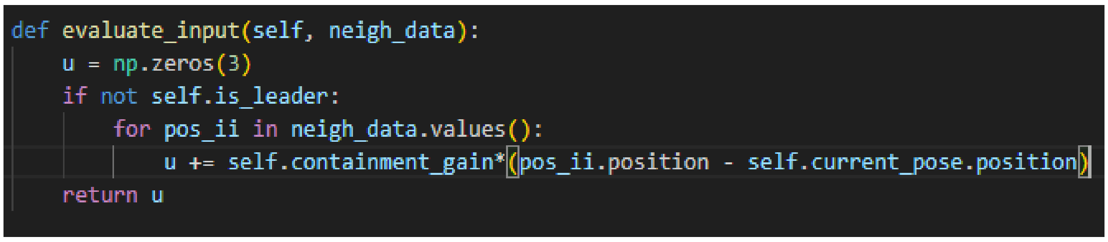
where for all $i \in \{1,\ldots,N\}$, $xi \in R2$ denotes the i-th state (position), and $ui \in R2$ is the i-th input (velocity). Communication among robots handled by TimeVaryingCommunicator as undirected graph $G = (V, E)$, with $V = \{1\ldots, N\}$ representing the robot set and $E \subset V \times V$ denoting the set of edges. If $(i, j) \in E$, then $(j, i) \in E$ as well, allowing robots i and j to exchange information. The neighbors of agent i are denoted as $Ni = \{j \in V \mid (i, j) \in E\}$.

The robots are categorized into leaders and followers. Followers aim to converge towards the convex hull formed by the positions of leaders. Achieving this goal involves implementing dynamics that drive the followers towards the desired formation.

$$\dot{\boldsymbol{x}}_{\boldsymbol{i}}(\boldsymbol{t}) = \mathbf{0} \qquad \text{(leaders)} \tag{4.2a}$$

$$\dot{\boldsymbol{x}}_{\boldsymbol{i}}(\boldsymbol{t}) = \sum_{j \in N_i} \boldsymbol{x}_{\boldsymbol{i}}(\boldsymbol{t}) - \boldsymbol{x}_{\boldsymbol{j}}(\boldsymbol{t}) \qquad \text{(followers)} \tag{4.2b}$$

Implementation: as far as the software implementation can be achieve by implementing two main classes which an extension from DistributedControlGuidance. the classes are: ContainmentGuidance and TimeVaryingContainmentGuidance. Both implement a distributed control law for robots to maintain a formation and use parameters like update frequency, gain, and pose handling. The evaluate_input method calculates control inputs using neighbor data. The implementation of the control law in the software as follow:



```
def evaluate_input(self, neigh_data):
    u = np.zeros(3)
    if not self.is_leader:
        for pos_ii in neigh_data.values():
            u += self.containment_gain*(pos_ii.position - self.current_pose.position)
    return u
```

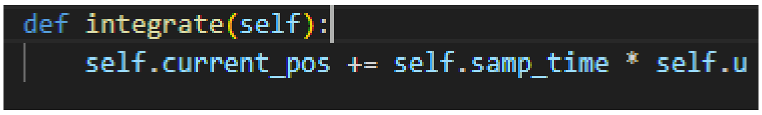
ContainmentGuidance computes the error between robot positions and desired positions, applying control inputs to minimize this error. TimeVaryingContainmentGuidance allows dynamic neighbor changes based on probabilities, useful in uncertain environments. ContainmentGuidance distinguishes leaders and computes control inputs based on errors. TimeVaryingContainmentGuidance selects active neighbors randomly and adapts neighbor connections.

Next step is to implement the SignalIntegrator class which is an inherited class from the main Integrator class. The class implement the Euler method for single-integrator dynamics:

$$\dot{x} = u \tag{4.3}$$

where $\dot{x}$ is the state of the system and u is the control input.

The implantation on the code as follow:



```
def integrate(self):
    self.current_pos += self.samp_time * self.u
```

4.4.2. Distributed Formation Control

Math problem: similar to the previous implantation of containment in leader-follower scenario, we assume that:

N robots navigate the 2D plain.
Every robot presented by single-integrator dynamic system.
Communication handles by graph-based communication.
The distributed control law must be applied for every robot as follow:

$$\boldsymbol{u_i} = \sum\nolimits_{\boldsymbol{j \in N_i}} || (\boldsymbol{x_i}(\boldsymbol{t}) - \boldsymbol{x_j}(\boldsymbol{t}))\text{\^{}}\boldsymbol{2}|| - \boldsymbol{d_{ji}}\text{\^{}}\boldsymbol{2})(\boldsymbol{x_i}(\boldsymbol{t}) - \boldsymbol{x_j}(\boldsymbol{t})) \tag{4.4}$$

Next, we assume that several N robots must form a desired geometric shape, so in that light we must adjust the adjacency matrix for robot communication based on graph theory, in addition of distance matrix to adjust the distance between the robot to form the desired shape. Usually, the adjacency matrix used to represent direct and undirect graph communication. Each entry in the adjacency matrix represents the connection between two nodes. If there is an edge between two nodes, the corresponding entry in the matrix is 1. If there is no edge between two nodes, the corresponding entry in the matrix is 0.For example, consider the following graph:

```
A---B
|      |
|      |
C---D
```

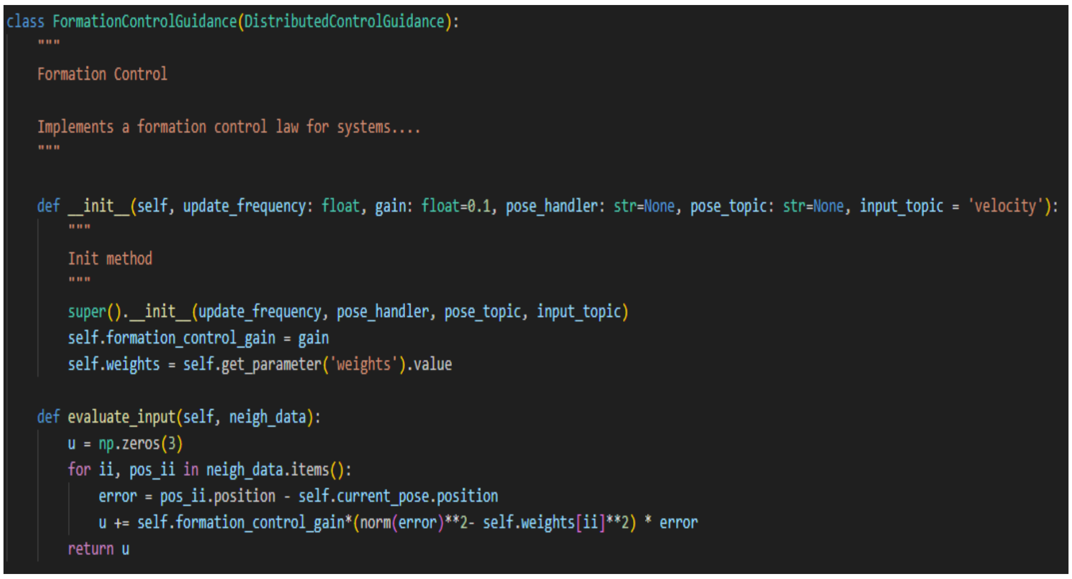
The adjacency matrix for this graph would be:

```
[ 1 0 0 1]
[ 0 1 0 0]
[ 0 0 1 0]
[ 1 0 0 1]
```

The first row of the matrix represents the connections between node A and the other nodes in the graph. The second row represents the connections between node B and the other nodes in the graph, and so on.

Implementation: to implement the formation control into the software framework we must consider the following two building blocks: firstly, a "Team Guidance" node facilitates the exchange of current positions among neighboring robots, enabling computation of the input ui(t) at a defined frequency. Secondly, a "Control" node undertakes the task of converting the single-integrator input (vector velocity) into appropriate unicycle inputs (angular and linear velocity).and finally visualize it through a launch file in Gazebo simulation.

In more detail regarding the first block, a DistributedControlGuidance class must be extended to a child class by the name of FormationControlGuidance to override the velocity methods as shown in the main part of the code:



```python
class FormationControlGuidance(DistributedControlGuidance):
    """
    Formation Control

    Implements a formation control law for systems....
    """

    def __init__(self, update_frequency: float, gain: float=0.1, pose_handler: str=None, pose_topic: str=None, input_topic = 'velocity'):
        """
        Init method
        """
        super().__init__(update_frequency, pose_handler, pose_topic, input_topic)
        self.formation_control_gain = gain
        self.weights = self.get_parameter('weights').value

    def evaluate_input(self, neigh_data):
        u = np.zeros(3)
        for ii, pos_ii in neigh_data.items():
            error = pos_ii.position - self.current_pose.position
            u += self.formation_control_gain*(norm(error)**2- self.weights[ii]**2) * error
        return u
```

4.4.3. Distributed Model Predictive Control (MPC)

Model Predictive Control (MPC) in multi-robotic systems is a sophisticated control strategy that involves predicting the future behavior of the robots and optimizing their actions over a finite time horizon to achieve desired objectives. MPC considers a mathematical model of the system dynamics and constraints to make informed decisions about the robots' actions.

In the context of multi-robot systems, MPC operates by formulating an optimization problem that considers various factors such as robot dynamics, environmental conditions, task requirements, and constraints. It then computes a sequence of control actions that each robot should follow to optimize a specified objective function. This objective function could be related to tasks like formation control, collision avoidance, path tracking, or any other cooperative behavior. Key characteristics of MPC in multi-robot systems include:

- Prediction.
- Constraints.
- Optimization.
- Real-time Adaptation.
- Cooperative Behavior.

Math Problem: The linear dynamics of the N robots, along with the associated constraints, outputs, and local cost functions.

Linear Dynamics: The dynamics of each robot i can be represented as:

$$x_i(t+1) = A_i x_i + B_i u_i \tag{4.5}$$

where is the state vector and is the control input vector for robot i.

Constraints: Each robot must adhere to local state and input constraints:

$$x_i(t) = X_i \tag{4.6a}$$

$$u_i(t) = U_i \tag{4.6b}$$

where and are the state and input constraint set for robot i, respectively.

Outputs: For each robot i, the output is defined as:

$$z_i(t+1) = C_i x_i(t) + D_i u_i(t) \tag{4.7}$$

Where $z_i(t+1)$ is the output for robot i.

Coupled Output Constraint: The outputs of all robots are constrained to satisfy:

$$\sum_{i=1}^{N} z_i(t) \epsilon s \tag{4.8}$$

where S constraint set for the coupled outputs.

Local Cost Function: Each robot i has an associated local cost function that it aims to minimize:

$$i = (x_i, u_i) \tag{4.9}$$

Overall Objective: Considering the above components, the overall objective can be formulated as a multi-objective optimization problem is to minimize the sum of local cost functions while adhering to constraints: subject to:

- linear dynamics
- Local state and input constraints:
- Coupled output constraint.

Solving this multi-objective optimization problem involves finding control inputs for each robot i that minimize their respective local cost functions while satisfying the dynamics, state/input constraints, and the coupled output constraint.This formulation captures the interplay between the robots' dynamics, constraints, outputs, and local objectives, aiming to achieve coordinated behavior that optimizes local cost functions while adhering to overall constraints.

Implementation: to implement the MPC distributed algorithms we must apply the following steps:

- Implement the TGL node to compute and communicate the MPC algorithms through MPCGuidance class.
- The implantation of integrator node to override the MPCGuidance class frequency.
- The visualization step to communicate the necessary data like robots' position to ROS topic.

The MPCGuidance class, which is a subclass of OptimizationGuidance. This class employs a model predictive control (MPC) algorithm for optimizing robot motion within a multi-agent system. Key methods include:

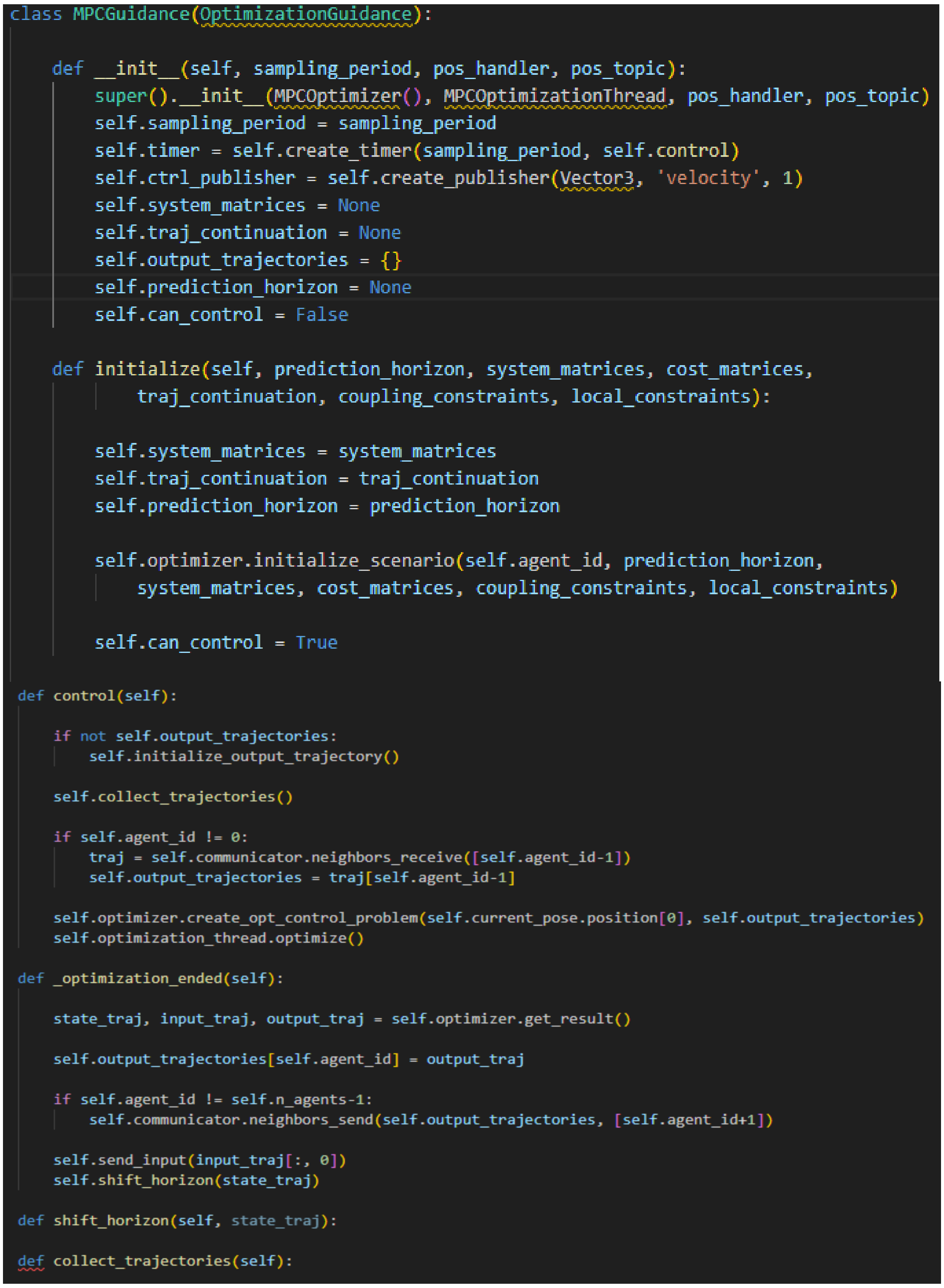
init__(): Initializes the class, taking parameters like sampling period, pose handler, and topic name.

initialize(): Prepares the MPC problem with parameters like prediction horizon, matrices, and constraints.

control (): Main control loop: initializes trajectories, gathers and receives data from agents, solves local optimal control, and updates trajectories.

optimization_ended(): Handles post-optimization actions, updating trajectories and sending data.

shift_horizon (): Extends output trajectory via trajectory_continuation() function.
collect_trajectories (): Aggregates trajectories from neighboring agents at agent 0.

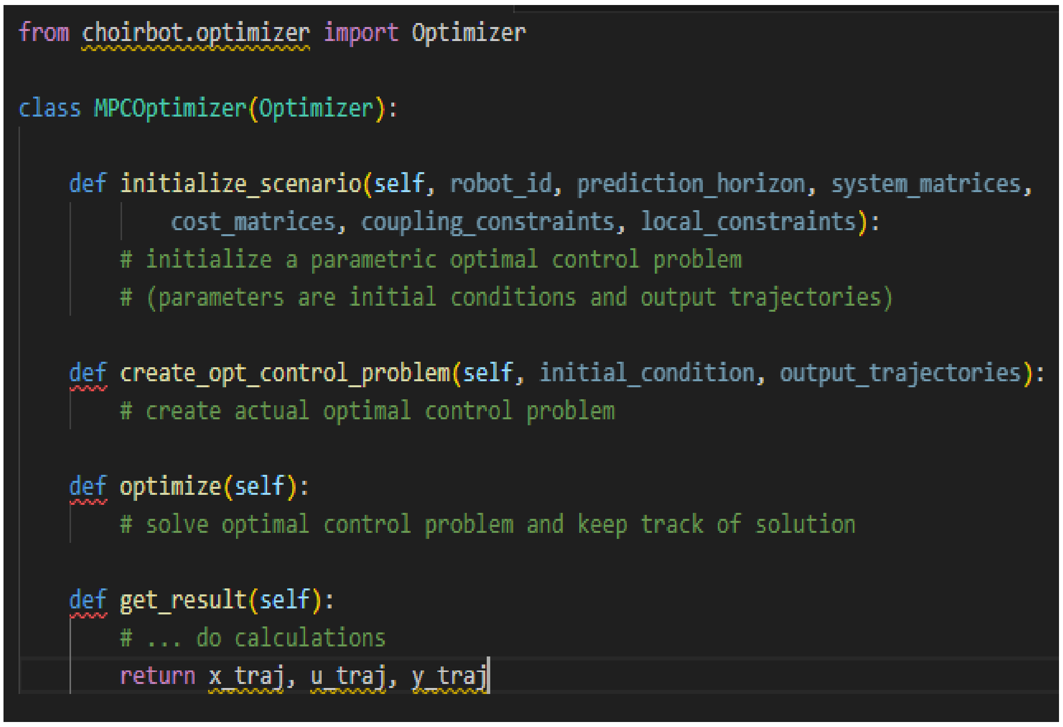


```python
class MPCGuidance(OptimizationGuidance):

    def __init__(self, sampling_period, pos_handler, pos_topic):
        super().__init__(MPCOptimizer(), MPCOptimizationThread, pos_handler, pos_topic)
        self.sampling_period = sampling_period
        self.timer = self.create_timer(sampling_period, self.control)
        self.ctrl_publisher = self.create_publisher(Vector3, 'velocity', 1)
        self.system_matrices = None
        self.traj_continuation = None
        self.output_trajectories = {}
        self.prediction_horizon = None
        self.can_control = False

    def initialize(self, prediction_horizon, system_matrices, cost_matrices,
            traj_continuation, coupling_constraints, local_constraints):

        self.system_matrices = system_matrices
        self.traj_continuation = traj_continuation
        self.prediction_horizon = prediction_horizon

        self.optimizer.initialize_scenario(self.agent_id, prediction_horizon,
            system_matrices, cost_matrices, coupling_constraints, local_constraints)

        self.can_control = True

def control(self):

    if not self.output_trajectories:
        self.initialize_output_trajectory()

    self.collect_trajectories()

    if self.agent_id != 0:
        traj = self.communicator.neighbors_receive([self.agent_id-1])
        self.output_trajectories = traj[self.agent_id-1]

    self.optimizer.create_opt_control_problem(self.current_pose.position[0], self.output_trajectories)
    self.optimization_thread.optimize()

def _optimization_ended(self):

    state_traj, input_traj, output_traj = self.optimizer.get_result()

    self.output_trajectories[self.agent_id] = output_traj

    if self.agent_id != self.n_agents-1:
        self.communicator.neighbors_send(self.output_trajectories, [self.agent_id+1])

    self.send_input(input_traj[:, 0])
    self.shift_horizon(state_traj)

def shift_horizon(self, state_traj):

def collect_trajectories(self):
```

After implementing the first main guidance class, the OptimizationThread loop come in place which is a straightforward to start and stop the optimization when the required number of the iterations is reached. Therefore, the entire optimization process is relying on MPCOptimizer class. The MPCOptimizer class implements a model predictive control algorithm that can be used to optimize the motion of a robot. The MPC algorithm considers the system dynamics, cost functions, coupling constraints, and local constraints of the robot to find the optimal control inputs for the next prediction horizon.

The main body of the MPCOptimizer contain the following methods:

- The initialize_scenario() method first initializes the system matrices, cost matrices, coupling constraints, and local constraints of the MPC problem. It then creates a parametric optimal control problem that has the two parameters: the initial condition of the robot and the output trajectories of the robot.
- The create_opt_control_problem() method creates the actual optimal control problem. It takes in the initial condition and output trajectories as parameters, and it uses the system matrices, cost matrices, coupling constraints, and local constraints that were initialized in the initialize_scenario() method.
- The optimize() method solves the MPC problem using a numerical solver. It keeps track of the solution in a dictionary, where the keys are the time steps, and the values are the states, controls, and outputs of the robot.
- The get_result() method returns the solution of the MPC problem. It takes the dictionary that was created in the optimize() method and returns the states, controls, and outputs of the robot.



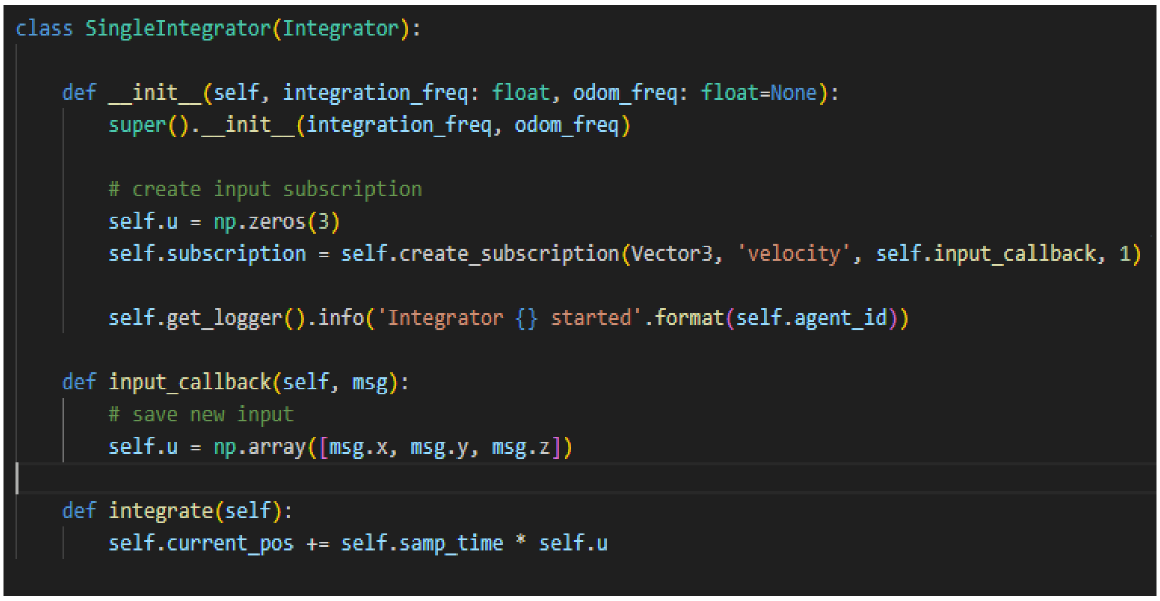
```
from choirbot.optimizer import Optimizer

class MPCOptimizer(Optimizer):

    def initialize_scenario(self, robot_id, prediction_horizon, system_matrices,
            cost_matrices, coupling_constraints, local_constraints):
        # initialize a parametric optimal control problem
        # (parameters are initial conditions and output trajectories)

    def create_opt_control_problem(self, initial_condition, output_trajectories):
        # create actual optimal control problem

    def optimize(self):
        # solve optimal control problem and keep track of solution

    def get_result(self):
        # ... do calculations
        return x_traj, u_traj, y_traj
```

After implementing the TGL node the next step is to consider the integrator node to override the TGL frequency similar to the previous implantation. The integrate() method integrates the robot's motion for one-time step. It takes the current position and velocity of the robot as input, and it returns the new position of the robot. The integrate() method uses the following equation to integrate the robot's motion:

new_pos = old_pos + samp_time * u (4.10)

where:

- new_pos is the new position of the robot.
- old_pos is the old position of the robot.
- samp_time is the sampling time of the integrator.
- u is the robot's velocity.



```
class SingleIntegrator(Integrator):

    def __init__(self, integration_freq: float, odom_freq: float=None):
        super().__init__(integration_freq, odom_freq)

        # create input subscription
        self.u = np.zeros(3)
        self.subscription = self.create_subscription(Vector3, 'velocity', self.input_callback, 1)

        self.get_logger().info('Integrator {} started'.format(self.agent_id))

    def input_callback(self, msg):
        # save new input
        self.u = np.array([msg.x, msg.y, msg.z])

    def integrate(self):
        self.current_pos += self.samp_time * self.u
```

Finally, the implantation is completed after the next two steps related mainly to ROS2 and further exploration will be done in the next chapter, but we briefly mentioned the steps as follow:

We employ the utils.Visualizer class for the visualization node. This node's objective is straightforward: it subscribes to the odom topic and transfers messages to the visualization_marker topic. RVIZ then interprets data from this source, illustrating circles denoting robots, accurately positioned. Finally, we write a launch file to launch the distributed MPC algorithms through RVIZ.

4.4.4. Distributed Dynamic Task Assignment

Distributed dynamic task allocation involves a decentralized strategy for distributing tasks among multiple agents or robots in a dynamic setting. The term "dynamic" implies that tasks, agents, or the environment can change over time, necessitating ongoing adjustment and reallocation of tasks. In scenarios of distributed dynamic task assignment, multiple agents or robots cooperate to effectively distribute tasks among themselves, factoring in evolving circumstances, preferences, and limitations. This stands in contrast to centralized methods where a sole entity handles all task assignments. In distributed systems, decision-making responsibility is shared across agents, allowing them to make localized decisions while taking broader objectives into account.

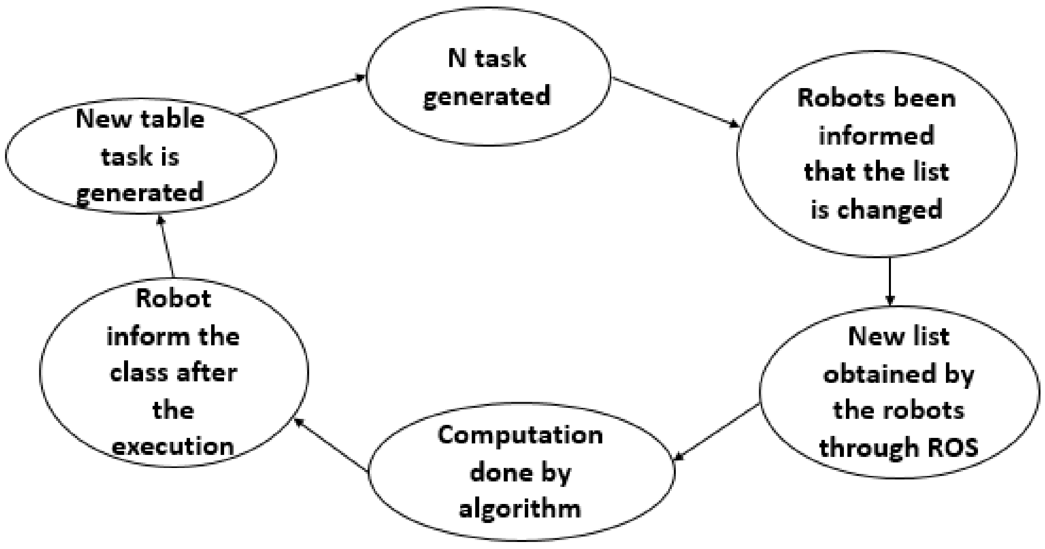
Math problem: Consider a set of N agents or robots, indexed by i = 1, 2, ..., N, and a set of M tasks, indexed by j = 1, 2, ..., M. Each agent i has a set of possible tasks it can perform, denoted as Ti, and each task j has a set of potential agents that can perform it, denoted as Aj. Let xij be a binary decision variable representing whether agent i is assigned to task j. The value of xij is 1 if agent i is assigned to task j, and 0 otherwise. The goal is to optimize the assignment of agents to tasks while considering various objectives or constraints. This can be formulated as an optimization problem, often with the objective of maximizing or minimizing a certain metric. For example, maximizing the overall efficiency, minimizing the total cost, or achieving a balanced assignment.

Mathematically, a common formulation for the distributed dynamic task assignment problem can be expressed as follows.

Maximize/Minimize $\sum i,k \; c_{ik} \, x_{ik}$

Solving this mathematical optimization problem involves finding the values of the decision variables xij that optimize the specified objective while adhering to the constraints. In a distributed setting, agents collaborate to make local decisions about their task assignments, considering both their own preferences and the constraints imposed by the problem. Communication and coordination between agents are crucial to achieve a balanced and efficient task assignment.

In our scenario there is a M number of tasks send the mobile robot’s overtime through a cloud, as soon as one task is completed, other tasks will be sent to be executed through the optimization loop. To maintain the tasks, overflow a task table is implanted through PositionTaskTable and the flow chart of the process is shown in 4.4:



N task generated
Robots been informed that the list is changed
New list obtained by the robots through ROS
Computation done by algorithm
Robot inform the class after the execution
New table task is generated

Figure 4.4. Task assignment chart flow.

The implementation: the implantation of the task assignment is similar to the previous implementation of MPC. The main guidance class TaskGuidance is responsible for the distribution of the task optimization and communicate the information to the task table, in addition solving the optimization problems is handled by TaskOptimizer and TaskOptimizationThread similar to the previously mentioned MPCOptimizer and MPCOptimizationThread. But in contrast to MPC implementation. The task assignment needs more classes to communicate the information and algorithms to gazebo through the Turtlebot unicycle robot platform, therefore:

PositionTaskExecutor class: this class notify TGL layer when the task completed by controlling the navigation of the robot to the desired position.

TwoDimPointToPointPlanner class: a part of planner class, simply act as the bridge between TGL and the controller class by forwarding the position target from the first to the later.

Controller class: a class that implement the unicycle control explain in [116]

PART 5

SIMULATION AND RESULTS

The ROS2 platform served as the foundation for our comprehensive multi-layer software framework, which was entirely realized using Python programming. Leveraging various Python libraries, we successfully brought this framework to fruition. The outcomes were effectively visualized through the Gazebo 9 simulator and the data visualization tool RVIZ2.

Our approach involved implementing diverse control and optimization scenarios by integrating the respective mathematical models into dedicated software classes. This integration facilitated the execution of various control and optimization strategies seamlessly.

Both hardware and software components played a pivotal role in the project:

- HP ProBook 650 (Intel Core i7)
- ROS2 Dashing Diademata
- Gazebo 9
- RVIZ2
- TurtleBot 3 ROS package
- Xterm – terminal emulator

The simulation was grounded in the visualization tools Gazebo and RVIZ, which effectively illustrated the outcomes of four distinct distributed optimization and control scenarios. These scenarios were built upon the team guidance layer (TGL) and encompassed:

- Containment within a leader-follower network for a single-integrator system.

- Distributed geometrical formation control designed for unicycle ground robots.
- Implementation of a distributed model predictive control (MPC) strategy.
- Execution of distributed dynamic task assignment for unicycle robots.
- Through these meticulously developed scenarios, our framework demonstrated its capabilities in addressing a spectrum of control and optimization challenges.

5.1. CONTAINMENT

The simplest approach among the various scenarios involved the combination of two core components: the "ContainmentGuidance" and "SingleIntegrator" classes. The outcomes of this particular implementation were visualized without the requirement of external tools. Instead, the ROS2 toolbox RVIZ was employed. This tool effectively showcased the robots as markers and acquired pose data from the "SingleIntegrator" component.

To initiate the simulation, a ROS2 launch file was employed. This file encompassed all the essential nodes responsible for both distributed control and visualization within RVIZ. The culmination of these efforts is illustrated in Figure 13, providing a clear visual representation of the achieved results.

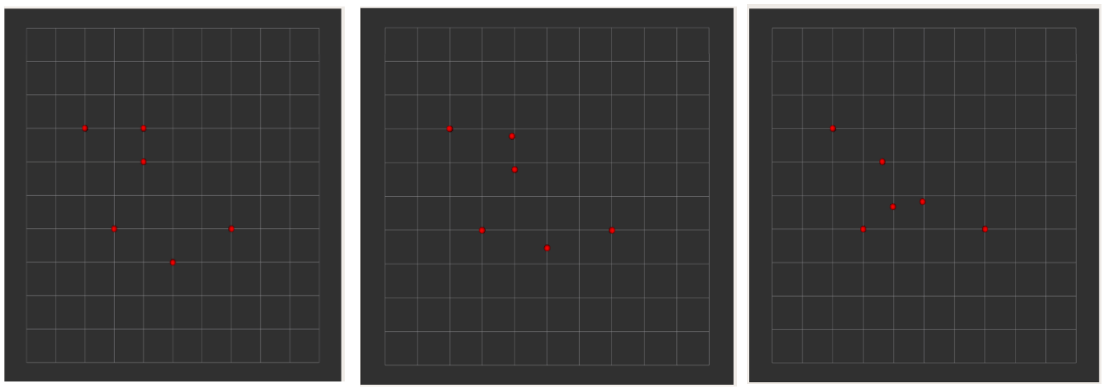


Figure 5.1. RVIZ containment simulation.

The RVIZ simulation outcome vividly depicts a group of six robots, intelligently categorized into leaders and followers. This division is balanced, with an equal number of robots in each role. The simulation showcases a dynamic movement pattern among the follower robots, as they efficiently transition from their initial positions to occupy specific positions that collectively form a triangular configuration. This geometric arrangement is strategically designed to align with the positions of the leader robots, illustrating a successful containment and guidance strategy in action.

5.2. FORMATION CONTROL

The core objective of this implementation is to guide a cluster of robots from initially scattered positions to adopt specific and discernable geometric formations within the (x, y) plane. This objective is achieved by harnessing the potential of a distributed control law. The results of this endeavor are expertly showcased through the Gazebo simulator, which provides a visual representation of each distinct formation pattern.

For initiating the simulation, a ROS2 launch file takes center stage. This comprehensive file harmoniously brings together all the essential nodes, orchestrating their roles in enabling both the facilitation of distributed control and the generation of visual representations within the Gazebo environment. This integrated approach guarantees a seamless and coherent portrayal of the achieved outcomes. Moreover, the simulation capitalizes on the utilization of the TurtleBot 3 Burger model and the corresponding ROS2 package. These elements contribute to the physical simulation of the formations, enhancing the authenticity of the outcomes.

Each unique formation pattern is meticulously demonstrated in isolation, offering a clear insight into its specific geometric configuration. These demonstrations serve as compelling evidence of the efficacy of the employed distributed control strategies. the patterns are :

- Triangle formation
- Square formation
- Side-by-side tringles
- Pentagon formation
- Hexagon formation

5.2.1. Triangle Formation

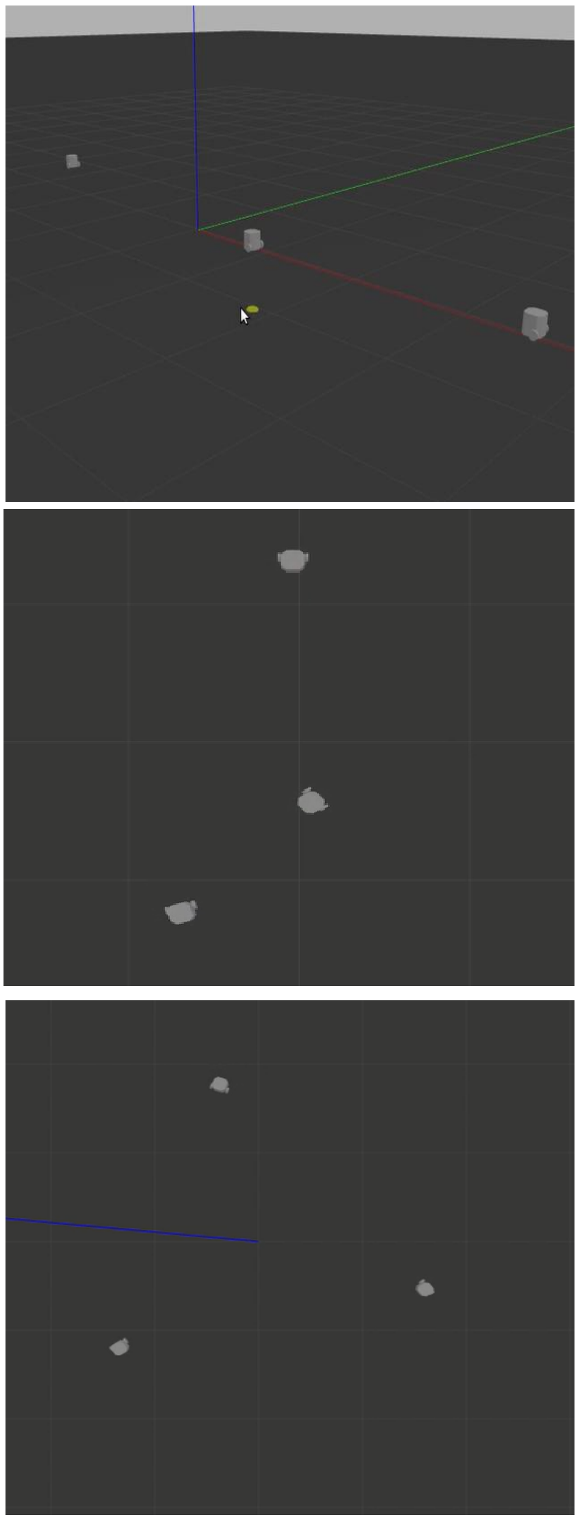


Figure 5.2. Tringle formation with 3 robots.

5.2.2. Square Formation

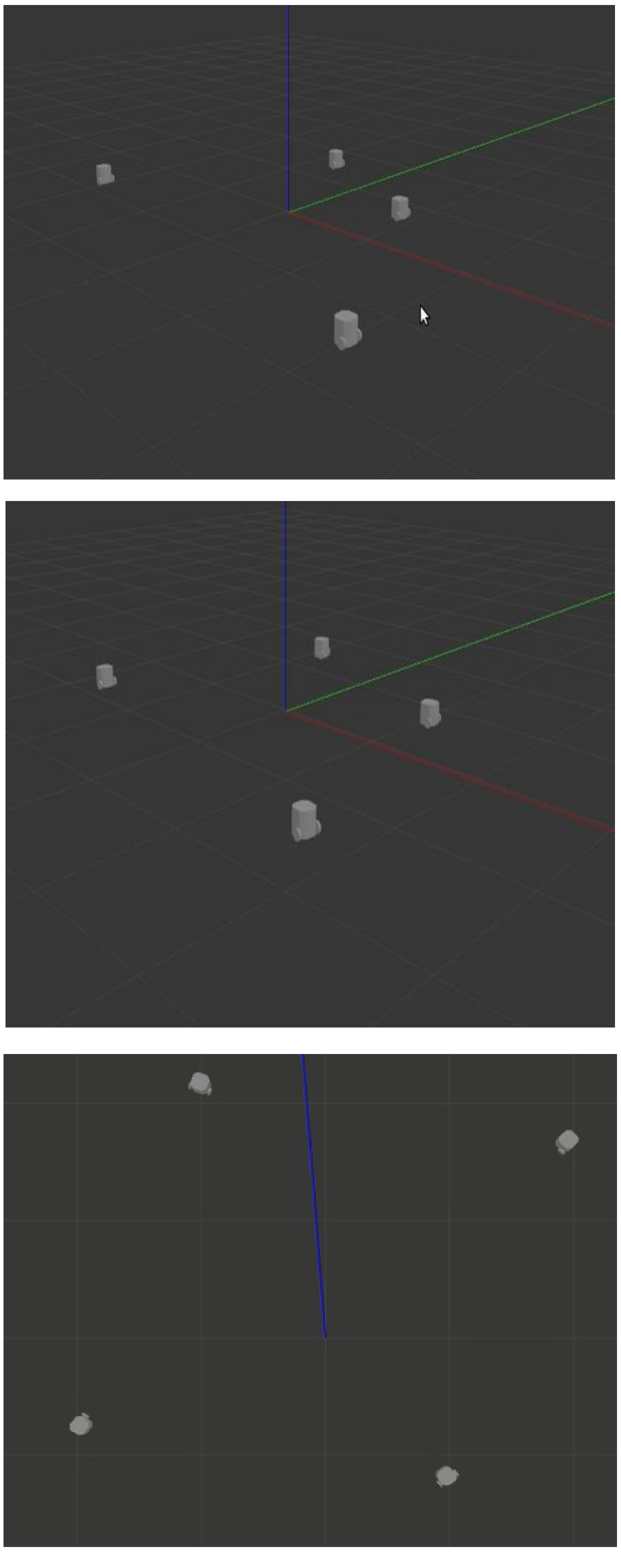


Figure 5.3. Square formation with 4 robots.

5.2.3. Side-by-side Triangle

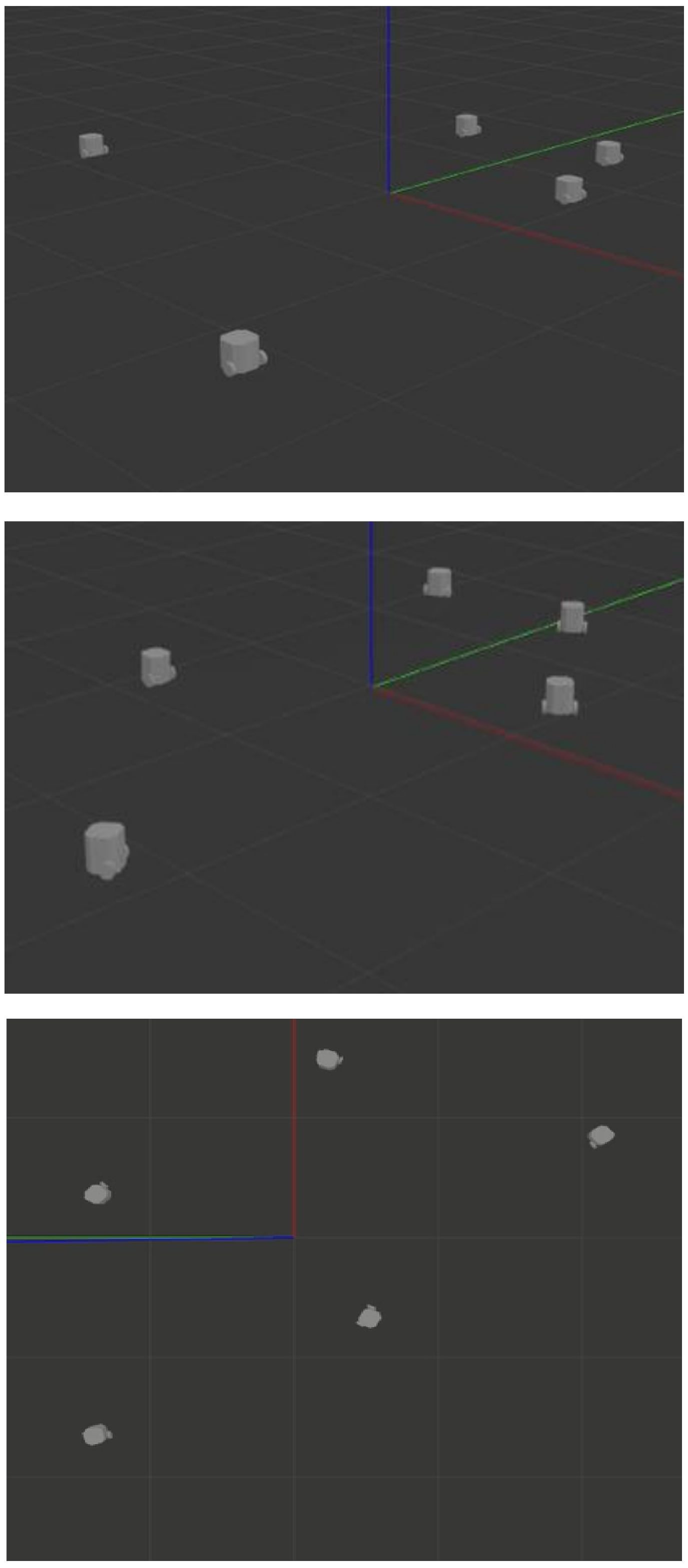


Figure 5.4. Side-by-side tringles with 5 robots.

5.2.4. Pentagon Formation

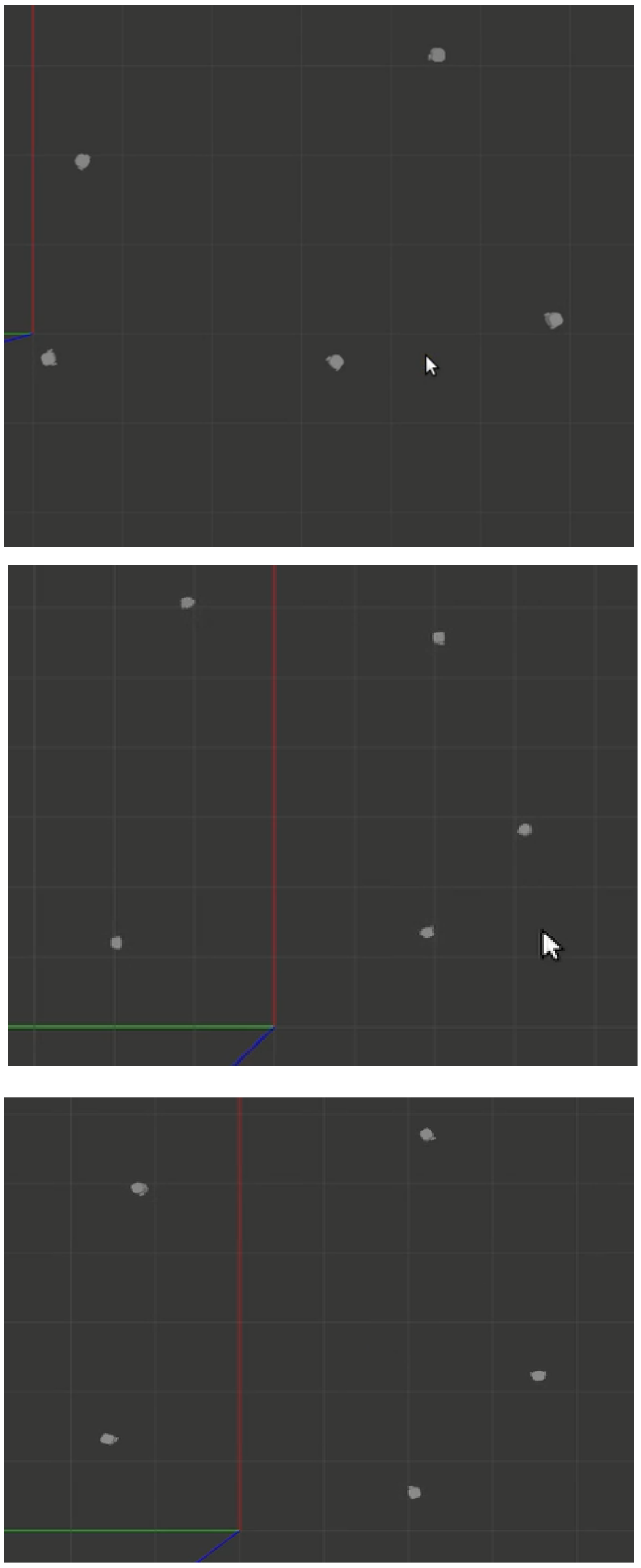


Figure 5.5. Pentagon formation with 5 robots.

5.2.5. Hexagon Formation

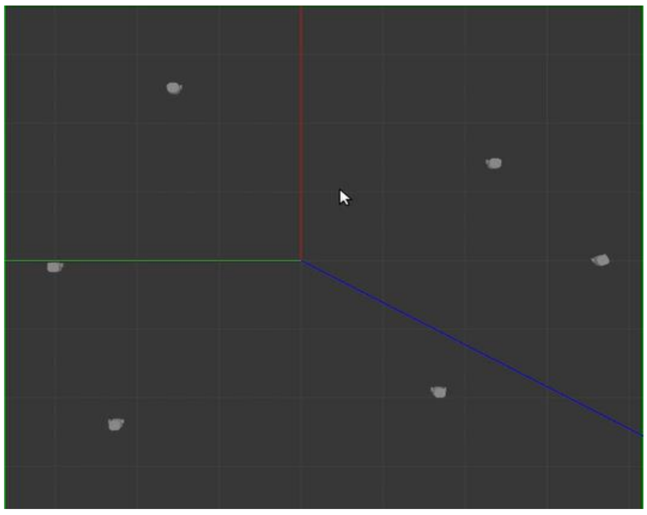
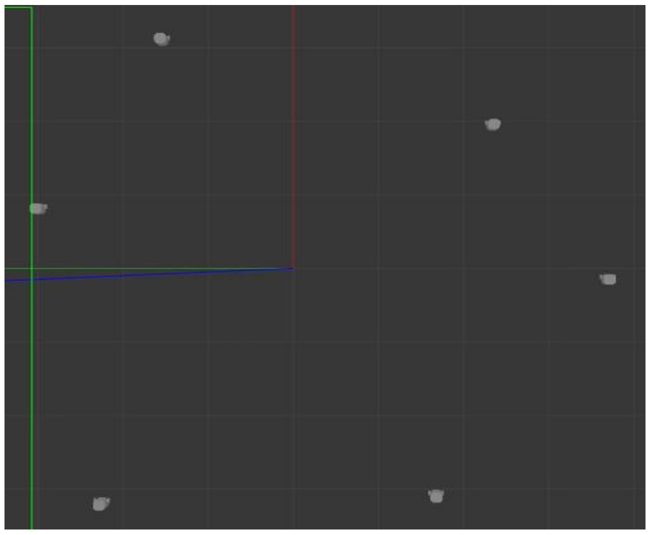
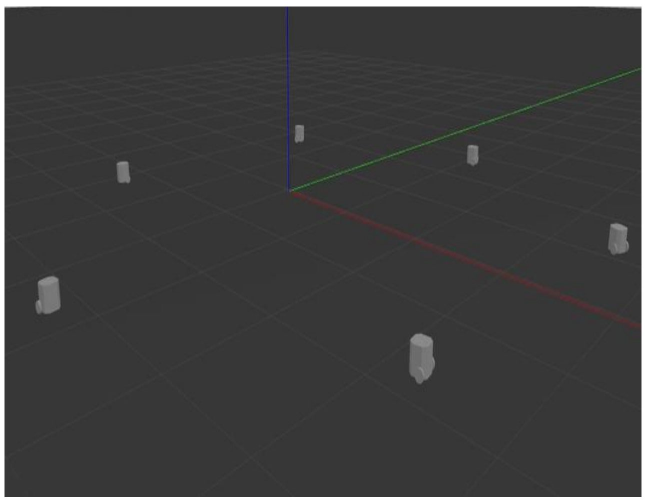




Figure 5.6. Hexagon formation with 6 robots.

5.3. MPC

The implementation of Model Predictive Control (MPC) introduces a sophisticated optimization scenario integrated into the Distributed Optimization segment of the Team Guidance Layer (TGL). This complex scenario employs a classical algorithm designed for linear single-integrator systems, as exemplified through a numerical instance. The overarching objective of this optimization framework is to guide a group of N robots, each characterized by linear dynamics, towards the origin. Throughout this process, it remains crucial to respect the coupling constraints inherent in the robots' output interactions.

The resultant algorithmic approach was validated and assessed using simulation, effectively leveraging the RVIZ data visualization tool as part of the software integration. Upon initiating the simulation, an RVIZ window comes to life. Within a brief span of time, a collection of circles, representing the individual robots, initiates movement towards the origin point. Notably, these robots coordinate their motion to ensure their mutual inter-distances remain within predefined limits, effectively showcasing the control strategy's success in action.

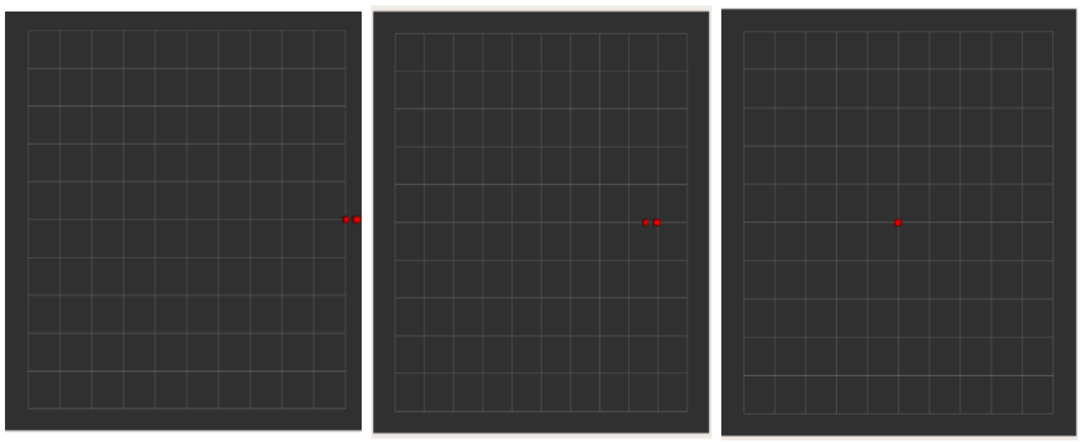


Figure 5.7. Model Predictive Control (MPC) algorithm.

5.4. TASK ASSIGNMENT

The algorithm orchestrates the actions of a quartet of Turtlebot 3 Burger robots, assigning them a collection of M tasks that manifest as positions within the Gazebo simulator's (x, y) plane. This algorithm operates in a continuous optimization loop, recalculating and determining fresh tasks promptly after each task completion. Simultaneously, it updates the task information table, subsequently influencing the trajectories of the robots. These recalibrations are relayed to the dedicated controller class, which takes on the role of guiding the robots.

The framework for distributing task positions operates via a cloud service, employing a task table that is visually represented through the xterm application. As the simulation unfolds, the group of four Burger robots dynamically aligns itself with the tasks stipulated in the task table. New tasks are computed and introduced, a sequence that unfolds within the confines of the xterm terminal. These tasks are then disseminated among the robots, with each robot's specific task distribution being displayed within its corresponding xterm window.

Consequently, the robots adapt their trajectories to address the new set of tasks. This adaptation involves the resolution of optimization challenges both at an individual robot's scale and across the network level. As this synchronization occurs, the algorithm manages tasks locally within each robot while simultaneously striving to achieve global optimization across the network of robots. This dynamic interplay serves to efficiently allocate tasks and guide the robots in a coordinated manner, ensuring effective task execution and trajectory adjustment.

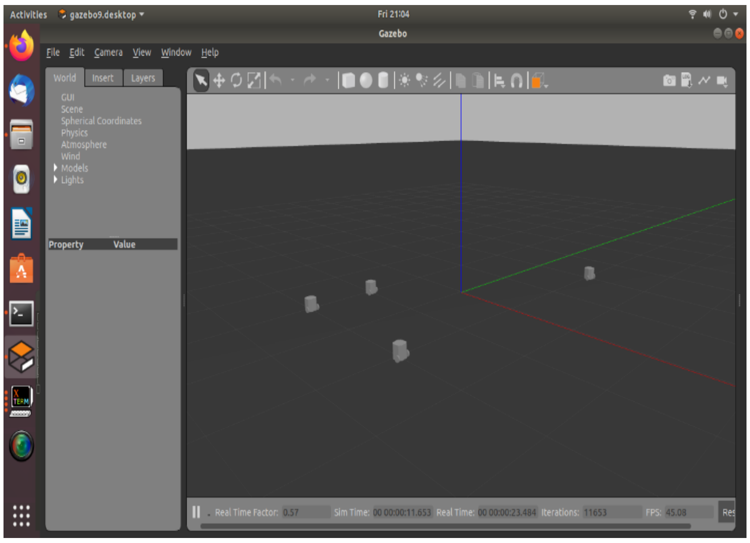


Figure 5.8. Turtlebot burger launch in the environment.

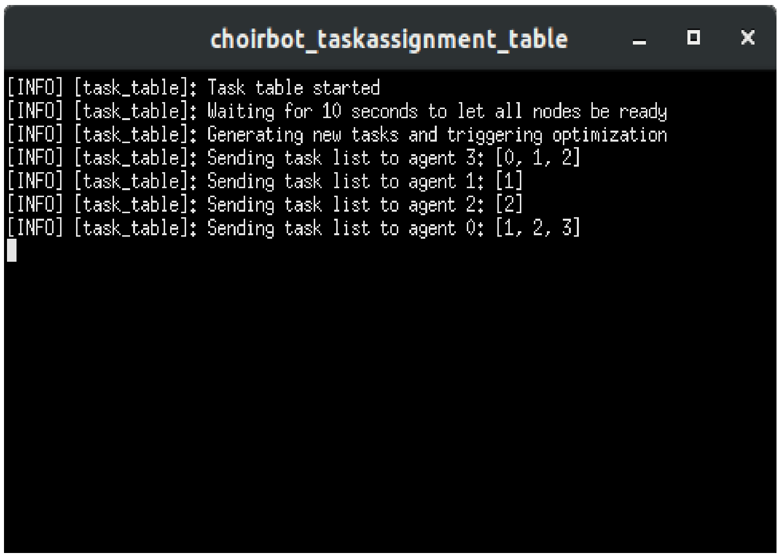
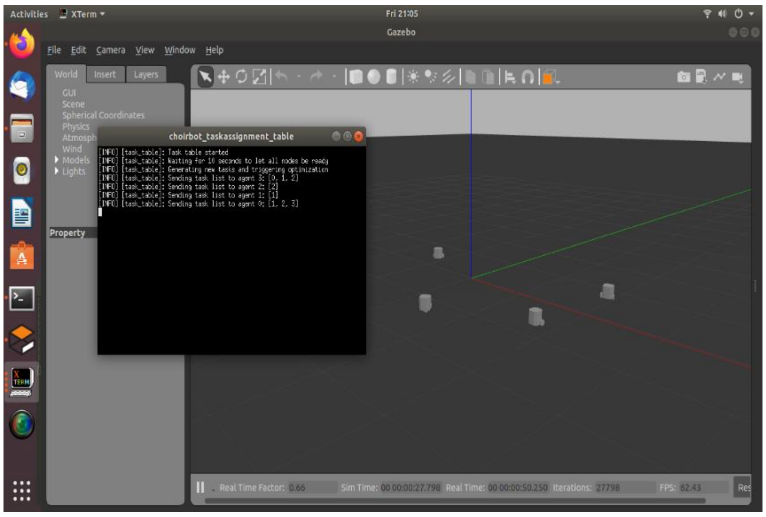
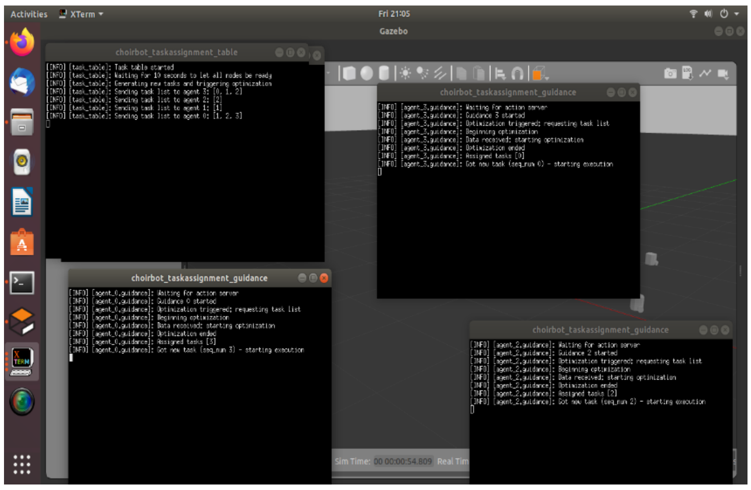

```
[INFO] [task_table]: Task table started
[INFO] [task_table]: Waiting for 10 seconds to let all nodes be ready
[INFO] [task_table]: Generating new tasks and triggering optimization
[INFO] [task_table]: Sending task list to agent 3: [0, 1, 2]
[INFO] [task_table]: Sending task list to agent 1: [1]
[INFO] [task_table]: Sending task list to agent 2: [2]
[INFO] [task_table]: Sending task list to agent 0: [1, 2, 3]
```

Figure 5.9. Xterm terminal showing the task table.



```
[INFO] [task_table]: Task table started
[INFO] [task_table]: Waiting for 10 seconds to let all nodes be ready
[INFO] [task_table]: Generating new tasks and triggering optimization
[INFO] [task_table]: Sending task list to agent 3: [0, 1, 2]
[INFO] [task_table]: Sending task list to agent 2: [2]
[INFO] [task_table]: Sending task list to agent 1: [1]
[INFO] [task_table]: Sending task list to agent 0: [1, 2, 3]
```

Figure 5.10. Xterm terminal launch in the simulation environment.


Activities
XTerm
Fri 21:05
Gazebo
choirbot_taskassignment_table
choirbot_taskassignment_guidance
choirbot_taskassignment_guidance
choirbot_taskassignment_guidance
Sim Time:

```
[INFO] [task_table]: Task table started
[INFO] [task_table]: Waiting for 10 seconds to let all nodes be ready
[INFO] [task_table]: Generating new tasks and triggering optimization
[INFO] [task_table]: Sending task list to agent 3: [0, 1, 2]
[INFO] [task_table]: Sending task list to agent 2: [2]
[INFO] [task_table]: Sending task list to agent 1: [1]
[INFO] [task_table]: Sending task list to agent 0: [1, 2, 3]
```

```
[INFO] [agent_3.guidance]: Waiting for action server
[INFO] [agent_3.guidance]: Guidance 3 started
[INFO] [agent_3.guidance]: Optimization triggered: requesting task list
[INFO] [agent_3.guidance]: Beginning optimization
[INFO] [agent_3.guidance]: Data received: starting optimization
[INFO] [agent_3.guidance]: Optimization ended
[INFO] [agent_3.guidance]: Assigned tasks [0]
[INFO] [agent_3.guidance]: Got new task (seq_num 0) - starting execution
```

```
[INFO] [agent_0.guidance]: Waiting for action server
[INFO] [agent_0.guidance]: Guidance 0 started
[INFO] [agent_0.guidance]: Optimization triggered: requesting task list
[INFO] [agent_0.guidance]: Beginning optimization
[INFO] [agent_0.guidance]: Data received: starting optimization
[INFO] [agent_0.guidance]: Optimization ended
[INFO] [agent_0.guidance]: Assigned tasks [3]
[INFO] [agent_0.guidance]: Got new task (seq_num 3) - starting execution
```

```
[INFO] [agent_2.guidance]: Waiting for action server
[INFO] [agent_2.guidance]: Guidance 2 started
[INFO] [agent_2.guidance]: Optimization triggered: requesting task list
[INFO] [agent_2.guidance]: Beginning optimization
[INFO] [agent_2.guidance]: Data received: starting optimization
[INFO] [agent_2.guidance]: Optimization ended
[INFO] [agent_2.guidance]: Assigned tasks [2]
[INFO] [agent_2.guidance]: Got new task (seq_num 2) - starting execution
```

Figure 5.11. Several xterm launched for every agent.

PART 6

CONCLUSION

This study introduces a robust ROS2 framework for facilitating distributed multi-agent cooperation behaviors. The framework operates as a peer-to-peer, non-centralized layer that excels in executing distributed optimization and control algorithms without requiring a central unit. The implementation was accomplished using Python and leveraged the capabilities of ROS2 Dashing, Gazebo 9, and Rviz2 platforms. The framework's design is based on a three-layer structure, with the Team Guidance Layer (TGL) being the most pivotal. TGL incorporates three crucial components:

- Graph-based Communication: Facilitates direct, indirect, synchronous, and asynchronous data exchange among agents.
- Distributed Optimization: Empowers complex optimization scenarios such as task assignment and model predictive control (MPC).
- Distributed Feedback Control: A simplified yet fully functional version of distributed optimization, supporting control algorithms like formation control and leader-follower containment control.

Numerous scenarios were implemented to showcase the framework's capabilities. The scenarios were categorized based on their complexity and the distributed algorithms they employed. Containment and formation control scenarios utilized distributed feedback control, while more intricate optimization scenarios like task assignment and MPC were built on distributed optimization principles. The outcomes of the implementations are as follows:

- A leader-follower containment scenario for a single integrator system was executed using distributed feedback control algorithms. Visualized in RVIZ2 simulator, the results showcased a group of 6 robots, divided equally into leaders and followers, forming a triangular arrangement as followers clustered around leader robots.
- Formation control for unicycle TurtleBot robots was realized through distributed control law implementation. Gazebo 9 simulator displayed various geometrical formations achieved by a minimal number of robots for each formation. Formations included triangle, square, pentagon, side-by-side triangles, and hexagon.
- The implementation of a classical MPC algorithm for a linear system leveraged distributed optimization algorithms on a single integrator system. The outcomes were simulated and visualized via RVIZ2 integration.
- A dynamic task assignment scenario was demonstrated. Multiple TurtleBot robots aimed to execute tasks defined by positions in the (x, y) plane. The task execution was showcased through a cloud node and task table visualized in xterm alongside the Gazebo simulation.

The combined results of experiments and simulations portrayed a heterogeneous fleet of robots functioning seamlessly in a decentralized manner. These robots were guided by distributed optimization and control algorithms, effectively tackling complex scenarios through the proposed ROS2 framework. This architecture empowers developers and programmers to seamlessly incorporate and execute intricate optimization and control algorithms on a heterogeneous robot fleet without necessitating a central unit. Importantly, this streamlined implementation process allows for a focus on innovative optimization and problem-solving techniques.

REFERENCES

1. Doncieux, S., Bredeche, N., Mouret, J.-B., and Eiben, A. E. (Gusz), "Evolutionary Robotics: What, Why, and Where to", ***Frontiers In Robotics And AI***, 2: (2015).

2. Royakkers, L. and Van Est, R., "A Literature Review on New Robotics: Automation from Love to War", ***International Journal Of Social Robotics***, 7 (5): 549–570 (2015).

3. Garcia, E., Jimenez, M. A., De Santos, P. G., and Armada, M., "The evolution of robotics research", ***IEEE Robotics & Automation Magazine***, 14 (1): 90–103 (2007).

4. Darmanin, R. N. and Bugeja, M. K., "A review on multi-robot systems categorised by application domain", ***2017 25th Mediterranean Conference on Control and Automation (MED)***, Valletta, Malta, (2017).

5. Gautam, A. and Mohan, S., "A review of research in multi-robot systems", ***2012 IEEE 7th International Conference on Industrial and Information Systems (ICIIS)***, Chennai, India, (2012).

6. Bayındır, L., "A review of swarm robotics tasks", ***Neurocomputing***, 172: 292–321 (2016).

7. Khamis, A., Hussein, A., and Elmogy, A., "Multi-robot Task Allocation: A Review of the State-of-the-Art", Cooperative Robots and Sensor Networks 2015, ***Springer International Publishing***, Cham, 31–51 (2015).

8. Oh, H., Ramezan Shirazi, A., Sun, C., and Jin, Y., "Bio-inspired self-organising multi-robot pattern formation: A review", ***Robotics And Autonomous Systems***, 91: 83–100 (2017).

9. Verma, J. K. and Ranga, V., "Multi-Robot Coordination Analysis, Taxonomy, Challenges and Future Scope", ***Journal Of Intelligent & Robotic Systems***, 102 (1): 10 (2021).

10. Hur, Y., Alur, R., Lee, I., Grudic, G., and Southall, B., "Aveek Das John Spletzer Joel Esposito Vijay Kumar James P. Ostrowski George Pappas Camillo J. Taylor", .

11. Gielis, J., Shankar, A., and Prorok, A., "A Critical Review of Communications in Multi-robot Systems", ***Current Robotics Reports***, 3 (4): 213–225 (2022).

12. Doriya, R., Mishra, S., and Gupta, S., "A brief survey and analysis of multi-robot communication and coordination", ***2015 International Conference on Computing, Communication & Automation (ICCCA)***, Greater Noida, India, (2015).

13. Rekleitis, I., Lee-Shue, V., Ai Peng New, and Choset, H., "Limited communication, multi-robot team based coverage", ***IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004***, New Orleans, LA, USA, (2004).

14. Balch, T. and Arkin, R. C., "Communication in reactive multiagent robotic systems", ***Autonomous Robots***, 1 (1): 27–52 (1994).

15. Unycao, Y., "Cooperative Mobile Robotics: Antecedents and Directions", .

16. Zhigang Wang, Zhou, M., and Ansari, N., "Ad-hoc robot wireless communication", ***SMC '03 Conference Proceedings. 2003 IEEE International Conference on Systems, Man and Cybernetics***, Washington, DC, USA, (2003).

17. Nett, E. and Schemmer, S., "Reliable real-time communication in cooperative mobile applications", ***IEEE Transactions On Computers***, 52 (2): 166–180 (2003).

18. Bayram, H. and Bozma, H. I., "Multi-robot navigation with limited communication - deterministic vs game-theoretic networks", ***2010 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2010)***, Taipei, (2010).

19. Jones, C. and Mataric, M. J., "Automatic synthesis of communication-based coordinated multi-robot systems", ***2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)***, Sendai, Japan, (2004).

20. Vicsek, T. and Zafeiris, A., "Collective motion", ***Physics Reports***, 517 (3–4): 71–140 (2012).

21. Schutte, J. F. and Groenwold, A. A., "A Study of Global Optimization Using Particle Swarms", ***Journal Of Global Optimization***, 31 (1): 93–108 (2005).

22. Mayya, S., Wilson, S., and Egerstedt, M., "Closed-loop task allocation in robot swarms using inter-robot encounters", ***Swarm Intelligence***, 13 (2): 115–143 (2019).

23. Nedjah, N. and Junior, L. S., "Review of methodologies and tasks in swarm robotics towards standardization", ***Swarm And Evolutionary Computation***, 50: 100565 (2019).

24. Warburton, K. and Lazarus, J., "Tendency-distance models of social cohesion in animal groups", ***Journal Of Theoretical Biology***, 150 (4): 473–488 (1991).

25. Bayindir, L. and Sahin, E., "Modeling self-organized aggregation in swarm robotic systems", ***2009 IEEE Swarm Intelligence Symposium (SIS)***, Nashville, (2009).

26. Okubo, A., "Dynamical aspects of animal grouping: Swarms, schools, flocks, and herds", ***Advances In Biophysics***, 22: 1–94 (1986).

27. Nbaum, D. G., "Schooling as a strategy for taxis in a noisy environment", .

28. Gazi, V. and Passino, K. M., "Stability analysis of swarms", ***IEEE Transactions On Automatic Control***, 48 (4): 692–697 (2003).

29. Delboeuf, J., "Review", (1881).

30. Breder, C. M., "Equations Descriptive of Fish Schools and Other Animal Aggregations", ***Ecology***, 35 (3): 361 (1954).

31. Botelho, S. C. and Alami, R., "M+: a scheme for multi-robot cooperation through negotiated task allocation and achievement", ***International Conference on Robotics and Automation***, Detroit, MI, USA, (1999).

32. Parker, L. E., "ALLIANCE: An Architecture for Fault Tolerant Multirobot Cooperation", ***IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION***, 14 (2): (1998).

33. Asama, H., Matsumoto, A., and Ishida, Y., "Design Of An Autonomous And Distributed Robot System: Actress", IEEE/RSJ International Workshop on Intelligent Robots and Systems ’. (IROS ’89) ’The Autonomous Mobile Robots and Its Applications, Tsukuba, Japan, (1989).

34. Caloud, P., Choi, W., Latombe, J.-C., Pape, C. L., and Yim, M., "INDOOR AUTOMATION WITH MANY MOBILE ROBOTS", .

35. Fukuda, T. and Kawauchi, Y., "Cellular robotic system (CEBOT) as one of the realization of self-organizing intelligent universal manipulator", ***, IEEE International Conference on Robotics and Automation Proceedings***, (1990).

36. Gerkey, B. P. and Matarić, M. J., "Murdoch: publish/subscribe task allocation for heterogeneous agents", ***Agents00: 4th International Conference on Autonomous Agents***, Barcelona Spain, (2000).

37. Jubin, J. and Tornow, J. D., "The DARPA packet radio network protocols", ***Proceedings Of The IEEE***, 75 (1): 21–32 (1987).

39. Murthy, S. and Garcia-Luna-Aceves, J. J., "An efficient routing protocol for wireless networks", ***Mobile Networks And Applications***, 1 (2): 183–197 (1996).

40. Tsu-Wei Chen and Gerla, M., "Global state routing: a new routing scheme for ad-hoc wireless networks", ***ICC ’98 1998 IEEE International Conference on Communications. Conference Record***, Atlanta, GA, USA, (1998).

41. Chiang, C.-C., Wu, H.-K., Liu, W., and Gerla, M., "ROUTING IN CLUSTERED MULTIHOP, MOBILE WIRELESS NETWORKS WITH FADING CHANNEL", .

42. Johnson, D. B. and Maltz, D. A., "Dynamic Source Routing in Ad Hoc Wireless Networks", Mobile Computing, ***Springer US***, Boston, MA, 153–181 (1996).

43. Park, V. D. and Corson, M. S., "A highly adaptive distributed routing algorithm for mobile wireless networks", ***INFOCOM ’97***, Kobe, Japan, (1997).

44. Toh, C.-K., "Associativity-Based Routing for Ad Hoc Mobile Networks", .

46. Zapata, M. G., "Secure ad hoc on-demand distance vector routing", ***ACM SIGMOBILE Mobile Computing And Communications Review***, 6 (3): 106–107 (2002).

47. Ko, Y.-B. and Vaidya, N. H., "Location-aided routing (LAR) in mobile ad hoc networks", ***MobiCom98: 4th Annual ACM International Conference on Mobile Computing and Networking***, Dallas Texas USA, (1998).

48. Yamashita, A., Fukuchi, M., Ota, J., Arai, T., and Asama, H., "Motion planning for cooperative transportation of a large object by multiple mobile robots in a 3D environment", ***2000 ICRA. IEEE International Conference on Robotics and Automation***, San Francisco, CA, USA, (2000).

50. Arai, T., Ogata, H., and Suzuki, T., "Collision Avoidance Among Multiple Robots Using Virtual Impedance", IEEE/RSJ International Workshop on Intelligent Robots and Systems ’. (IROS ’89) ’The Autonomous Mobile Robots and Its Applications, Tsukuba, Japan, (1989).

51. Matarić, M. J., "Reinforcement Learning in the Multi-Robot Domain", Robot Colonies, ***Springer US***, Boston, MA, 73–83 (1997).

52. Teodorović, D., "Bee Colony Optimization (BCO)", Innovations in Swarm Intelligence, ***Springer Berlin Heidelberg***, Berlin, Heidelberg, 39–60 (2009).

54. Johari, N. F., Zain, A. M., Noorfa, M. H., and Udin, A., "Firefly Algorithm for Optimization Problem", ***Applied Mechanics And Materials***, 421: 512–517 (2013).

55. Yang, X. and Hossein Gandomi, A., "Bat algorithm: a novel approach for global engineering optimization", ***Engineering Computations***, 29 (5): 464–483 (2012).

56. Rajabioun, R., "Cuckoo Optimization Algorithm", ***Applied Soft Computing***, 11 (8): 5508–5518 (2011).

57. Kennedy’, J. and Eberhart, R., "Particle Swarm Optimization", .

58. Elloumi, W., El Abed, H., Abraham, A., and Alimi, A. M., "A comparative study of the improvement of performance using a PSO modified by ACO applied to TSP", ***Applied Soft Computing***, 25: 234–241 (2014).

59. Nedjah, N., Mendonça, R. M. D., and Mourelle, L. D. M., "PSO-based Distributed Algorithm for Dynamic Task Allocation in a Robotic Swarm", ***Procedia Computer Science***, 51: 326–335 (2015).

60. Alaliyat, S., Yndestad, H., and Sanfilippo, F., "Optimisation Of Boids Swarm Model Based On Genetic Algorithm And Particle Swarm Optimisation Algorithm (Comparative Study)", ***28th Conference on Modelling and Simulation***, (2014).

61. Honkote, V., Ghosh, D., Narayanan, K., Gupta, A., and Srinivasan, A., "Design and Integration of a Distributed, Autonomous and Collaborative Multi-Robot System for Exploration in Unknown Environments", ***2020 IEEE/SICE International Symposium on System Integration (SII)***, Honolulu, HI, USA, (2020).

62. Wu, X., Wang, S., and Xing, M., "Observer-Based Leader-Following Formation Control for Multi-Robot With Obstacle Avoidance", ***IEEE Access***, 7: 14791–14798 (2019).

63. Xia, Y., Chen, C., Shi, J., Liu, Y., and Li, G., "Two-Layer Path Planning for Multi-Area Coverage by a Cooperated Ground Vehicle and Drone System", Preprint, (2020).

64. Yi Guo and Parker, L. E., "A distributed and optimal motion planning approach for multiple mobile robots", ***2002 IEEE International Conference on Robotics and Automation***, Washington, DC, USA, (2002).

65. Yanyan Dai a, YoonGu Kim b, SungGil Wee b, DongHa Lee b, SukGyu Lee, "A switching formation strategy for obstacle avoidance of a multi-robot system based on robot priority model", 56: 123–134 (2016).

66. Niazi, M. A., Hussain, A., and Kolberg, M., "Verification &Validation of Agent Based Simulations using the VOMAS (Virtual Overlay Multi-agent System) approach", .

67. Gervasi, V. and Prencipe, G., "Coordination without communication: the case of the flocking problem", ***Discrete Applied Mathematics***, 144 (3): 324–344 (2004).

68. Saber, R. O. and Murray, R. M., "Flocking with obstacle avoidance: cooperation with limited communication in mobile networks", ***42nd IEEE International Conference on Decision and Control (IEEE Cat. No.03CH37475)***, Maui, Hawaii, USA, (2003).

69. Mataric, M. J., Nilsson, M., and Simsarin, K. T., "Cooperative multi-robot box-pushing", ***1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots***, Pittsburgh, PA, USA, (1995).

70. Brown, R. G. and Jennings, J. S., "A pusher/steerer model for strongly cooperative mobile robot manipulation", ***1995 IEEE/RSJ International Conference on Intelligent Robots and Systems. Human Robot Interaction and Cooperative Robots***, Pittsburgh, PA, USA, (1995).

71. Waibel, M., Beetz, M., Civera, J., D'Andrea, R., Elfring, J., Gálvez-López, D., Häussermann, K., Janssen, R., Montiel, J. M. M., Perzylo, A., Schießle, B., Tenorth, M., Zweigle, O., and De Molengraft, R., "RoboEarth", ***IEEE Robotics & Automation Magazine***, 18 (2): 69–82 (2011).

72. Moran, M. E., "The da Vinci Robot", ***Journal Of Endourology***, 20 (12): 986–990 (2006).

73. Miswanto, M., "Formation Control of Multiple Dubin's Car System with Geometric Approach", ***IOSR Journal Of Mathematics***, 1 (6): 16–20 (2012).

74. Kuhn, H. W., "The Hungarian method for the assignment problem", ***Naval Research Logistics Quarterly***, 2 (1–2): 83–97 (1955).

75. Turpin, M., Michael, N., and Kumar, V., "C APT : Concurrent assignment and planning of trajectories for multiple robots", ***The International Journal Of Robotics Research***, 33 (1): 98–112 (2014).

76. Gazä, V. and Fä, Bariå., "Aggregation, Foraging, and Formation Control of Swarms with Non-Holonomic Agents Using Potential Functions and Sliding Mode Techniques", (2007).

77. Yao, J., Ordonez, R., and Gazi, V., "Swarm Tracking Using Artificial Potentials and Sliding Mode Control", ***San Diego***, (2006).

78. Tanner, H. G., Pappas, G. J., and Kumar, V., "Leader-to-Formation Stability", ***IEEE Transactions On Robotics And Automation***, 20 (3): 443–455 (2004).

79. Miswanto, M., Pranoto, I., Mhammad, H. M., and Mahayana, D., "The Control Design of Ship Formation with the Presence of a Leader", ***IAES International Journal Of Robotics And Automation (IJRA)***, 4 (1): 53 (2015).

80. Desai, J. P., Ostrowski, J. P., and Kumar, V., "Modeling and control of formations of nonholonomic mobile robots", ***IEEE Transactions On Robotics And Automation***, 17 (6): 905–908 (2001).

81. Jiangzhou, L., Sakhavat, S., Ming, X., and Laugier, C., "Sliding Mode Control for Nonholonomic Mobile Robot", .

82. Orlando, G., Frontoni, E., Mancini, A., and Zingaretti, P., "Sliding mode control for vision-based leader following", .

83. Misir, O. and Gökrem, L., "Sürü Robotları için Esnek ve Ölçeklenebilir Toplanma Davranışı Metodu", ***European Journal Of Science And Technology***, 100–109 (2020).

84. M. B. Sial, S. Wang, X. Wang, J. Wyrwa, Z. Liao and W. Ding, "Mission Oriented Flocking and Distributed Formation Control of UAVs", ***IEEE***, 1507–1512 (2021).

85. Mechali, O., Iqbal, J., Wang, J., Xie, X., and Xu, L., "Distributed Leader-Follower Formation Control of Quadrotors Swarm Subjected to Disturbances", ***2021 IEEE International Conference on Mechatronics and Automation (ICMA)***, Takamatsu, Japan, (2021).

86. Gauci, M., Chen, J., Li, W., Dodd, T. J., and Groß, R., "Self-organized aggregation without computation", ***The International Journal Of Robotics Research***, 33 (8): 1145–1161 (2014).

87. Mısır, O., Gökrem, L., and Serhat Can, M., "Fuzzy-based self organizing aggregation method for swarm robots", ***Biosystems***, 196: 104187 (2020).

88. Parhizkar, M., Di Marzo Serugendo, G., Nitschke, J., Hellequin, L., Wade, A., and Soldati, T., "First-order agent-based models of emergent behaviour of Dictyostelium discoideum and their inspiration for swarm robotics: A selection of aggregation phase behaviour with biological illustrations", ***Artificial Life And Robotics***, 25 (4): 643–655 (2020).

89. Slowik, A. and Kwasnicka, H., "Nature Inspired Methods and Their Industry Applications—Swarm Intelligence Algorithms", ***IEEE Transactions On Industrial Informatics***, 14 (3): 1004–1015 (2018).

90. Yang, X.-S., "Swarm intelligence-based algorithms: a critical analysis", ***Evolutionary Intelligence***, 7 (1): 17–28 (2014).

91. Chakraborty, A. and Kar, A. K., "Swarm Intelligence: A Review of Algorithms", Nature-Inspired Computing and Optimization, ***Springer International Publishing***, Cham, 475–494 (2017).
92. "Evolutionary and Swarm Intelligence Algorithms", ***Springer International Publishing***, Cham, (2019).

93. Sumida, B. H., Houston, A. I., McNamara, J. M., and Hamilton, W. D., "Genetic algorithms and evolution", ***Journal Of Theoretical Biology***, 147 (1): 59–84 (1990).

94. Lambora, A., Gupta, K., and Chopra, K., "Genetic Algorithm- A Literature Review", ***2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)***, Faridabad, India, (2019).

95. Wang, D., Tan, D., and Liu, L., "Particle swarm optimization algorithm: an overview", ***Soft Computing***, 22 (2): 387–408 (2018).

96. Venter, G. and Sobieszczanski-Sobieski, J., "Particle Swarm Optimization", ***AIAA Journal***, 41 (8): 1583–1589 (2003).

97. Yuhui Shi and Eberhart, R. C., "Fuzzy adaptive particle swarm optimization", ***2001 Congress on Evolutionary Computation***, Seoul, South Korea, (2001).

98. Hsieh, S.-T., Sun, T.-Y., Liu, C.-C., and Tsai, S.-J., "Efficient Population Utilization Strategy for Particle Swarm Optimizer", ***IEEE Transactions On Systems, Man, And Cybernetics, Part B (Cybernetics)***, 39 (2): 444–456 (2009).

99. Wenhua Han, Ping Yang, Haixia Ren, and Jianpeng Sun, "Comparison study of several kinds of inertia weights for PSO", ***2010 International Conference on Progress in Informatics and Computing (PIC)***, Shanghai, China, (2010).

100. Nikolić, M. and Teodorović, D., "Empirical study of the Bee Colony Optimization (BCO) algorithm", ***Expert Systems With Applications***, 40 (11): 4609–4620 (2013).

101. Krishnanand, K. N. and Ghose, D., "Glowworm swarm optimisation: a new method for optimising multi-modal functions", .

102. Smith, M. G. and Bull, L., "Genetic Programming with a Genetic Algorithm for Feature Construction and Selection", ***Genetic Programming And Evolvable Machines***, 6 (3): 265–281 (2005).

103. Cao, Y. J. and Wu, Q. H., "Mechanical Design Optimization by Mixed-Variable Evolutionary Programming", .

104. Dianati, M., Song, I., and Treiber, M., "An Introduction to Genetic Algorithms and Evolution Strate…", .

105. Yang, X.-S., "Firefly Algorithms for Multimodal Optimization", Stochastic Algorithms: Foundations and Applications, ***Springer Berlin Heidelberg***, Berlin, Heidelberg, 169–178 (2009).

106. Mirjalili, S., Mirjalili, S. M., and Lewis, A., "Grey Wolf Optimizer", ***Advances In Engineering Software***, 69: 46–61 (2014).

107. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A., "ROS: an open-source Robot Operating System", .

108. "Robot Operating System (ROS)", ***Springer International Publishing***, Cham, (2016).

109. Okoli, F., Lang, Y., Kermorgant, O., and Caro, S., "Cable-Driven Parallel Robot Simulation Using Gazebo and ROS", ROMANSY 22 – Robot Design, Dynamics and Control, ***Springer International Publishing***, Cham, 288–295 (2019).

110. Megalingam, R. K., Teja, C. R., Sreekanth, S., and Raj, A., "ROS based Autonomous Indoor Navigation Simulation Using SLAM Algorithm", .

111. Farina, F., Camisa, A., Testa, A., Notarnicola, I., and Notarstefano, G., "DISROPT: a Python Framework for Distributed Optimization", ***IFAC-PapersOnLine***, 53 (2): 2666–2671 (2020).

112. Tauchnitz, S., "APPLICATION OF DISTRIBUTED CONTAINMENT CONTROL TO MULTI-ROBOT SYSTEMS", ***University of Rhode Island***, Kingston, RI, (2021).

113. Mehran Mesbahi and Magnus Egerstedt, "Graph Theoretic Methods in Multiagent Networks", ***Princeton University Press***, 424 (2010).

114. Richards, A. and How, J. P., "Robust distributed model predictive control", ***International Journal Of Control***, 80 (9): 1517–1531 (2007).

115. Bürger, M., Notarstefano, G., Bullo, F., and Allgöwer, F., "A distributed simplex algorithm for degenerate linear programs and multi-agent assignments", ***Automatica***, 48 (9): 2298–2304 (2012).

116. Park, J. J. and Kuipers, B., "A smooth control law for graceful motion of differential wheeled mobile robots in 2D environment", ***2011 IEEE International Conference on Robotics and Automation (ICRA)***, Shanghai, China, (2011).

RESUME

Mohammed SEDEG completed his primary, preparatory and secondary education in Khartoum schools, Sudan, and obtained a bachelor's degree in Electrical Engineering from Sudan University of Science and Technology - Sudan in 2017. After graduation, He worked as an electrical engineering in the renewable energy department at Africa City of Technology (ACT) for one year while taking several automation and industrial courses at Industrial Automation Center (IOC). Then, in 2019, he moved to Turkey to study at Karabuk University to obtain a master's degree in mechatronic engineering.