TECHNICAL UNIVERSITY OF CRETE, GREECE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Auction-based Coordination for Task Allocation in Drone Teams

Georgios Vastardis

Thesis Committee
Associate Professor Michail G. Lagoudakis (School of ECE)
Associate Professor Vasilis Samoladas (School of ECE)
Associate Professor Panagiotis Partsinevelos (School of MRE)

Chania, July 2019

Πολυτεχνείο Κρήτης

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Συντονισμός με Εικονικές Δημοπρασίες για Ανάθεση Εργασιών σε Ομάδες από *Drones*

Βασταρδής Γεώργιος

Εξεταστική Επιτροπή

Αναπληρωτής Καθηγητής Μιχαήλ Γ. Λαγουδάκης (Σχολή ΗΜΜΥ)

Αναπληρωτής Καθηγητής Βασίλης Σαμολαδάς (Σχολή ΗΜΜΥ)

Αναπληρωτής Καθηγητής Παναγιώτης Παρτσινέβελος (Σχολή ΜΗΧΟΠ)

Χανιά, Ιούλιος 2019

Abstract

In our days, natural disasters occur on a daily basis and typically require immediate human rescue team intervention, in order to minimize casualties. In most cases, the scene conditions may be prohibitive for human rescuers to intervene and the rescue mission may be delayed, possibly with serious consequences. Such scenarios are ideal for deploying autonomous mobile robot systems, which may provide assistance to the human rescuers by autonomously searching for threatened individuals and, in general, by providing information about the current situation. In this thesis, we present the use of a multi-drone team as a support tool to search-and-rescue missions. Using drones can potentially help to reduce the search time, because they can provide useful information from various locations through their sensors. Particularly, we focus on the optimization of the task allocation problem among the drone team members by using a distributed sequential auction method. The (simulated) auction has low computation and communication requirements, is purely decentralized, and eliminates the need for a centralized auctioneer. A number of tasks (target locations) that the drones must visit for the purpose of acquiring information is communicated to all drones at the start of the process. Then, the drones localize themselves in the environment and they initiate the auction mechanism for negotiating the targets, one at a time. Within the auction, we implemented and tested different objectives for the calculation of bids, which serve a variety of team goals in different missions this project may be used on. At the end of the auction, the drones have allocated all tasks among themselves and they proceed to visiting each one of the won tasks in the already determined order. A simple, collision-free navigation method ensures that the drones will not collide during path following. The entire project has been implemented within the Robot Operating System (ROS) and is available as an open source package. All experiments were conducted on the Gazebo robot simulator in obstacle-free environments to demonstrate the results of the proposed approach on smaller and larger problems with as many as 20 drones and 50 tasks. The main advantage of the proposed scheme is the ability to run in a fully-distributed manner over a team of inter-connected drones, or other mobile robots in general, and to provide an efficient approach for task allocation, a key problem in search-and-rescue missions.

Περίληψη

Στις μέρες μας, συμβαίνουν φυσικές καταστροφές σε καθημερινή βάση και συνήθως απαιτείται άμεση ανθρώπινη επέμβαση ομάδων διάσωσης, ώστε να μειωθούν οι ανθρώπινες απώλειες. Στις περισσότερες περιπτώσεις, οι συνθήκες στο χώρο του συμβάντος ενδέχεται να είναι απαγορευτικές για την παρέμβαση των διασωστών και η αποστολή διάσωσης μπορεί να καθυστερήσει, ενδεχομένως με σοβαρές συνέπειες. Τέτοια σενάρια είναι ιδανικά για χρήση αυτόνομων κινητών ρομποτικών συστημάτων, τα οποία μπορούν να παρέχουν βοήθεια στους διασώστες με αυτόνομη αναζήτηση απειλούμενων ατόμων και γενικότερα με την παροχή πληροφοριών σχετικά με την τρέχουσα κατάσταση. Στην παρούσα διπλωματική εργασία, παρουσιάζουμε τη χρήση μιας ομάδας από *drones* ως εργαλείο υποστήριξης σε αποστολές αναζήτησης και διάσωσης. Με τη χρήση *drones* έχουμε μείωση του χρόνου έρευνας σε αυτές τις αποστολές, καθώς μπορούν να παρέχουν χρήσιμες πληροφορίες από διάφορες τοποθεσίες μέσω των αισθητήρων τους. Ιδιαίτερα, επικεντρωνόμαστε στη βελτιστοποίηση του προβλήματος ανάθεσης καθηκόντων μεταξύ των μελών της ομάδας, χρησιμοποιώντας μια κατανεμημένη μέθοδο ακολουθιακής δημοπρασίας. Η (εικονική) δημοπρασία έχει χαμηλές απαιτήσεις υπολογιστικής ισχύος και επικοινωνίας, είναι πλήρως κατανεμημένη και εξαλείφει την ανάγκη ενός κεντρικού δημοπράτη. Κατά την έναρξη της διαδικασίας, ένας αριθμός καθηκόντων (τοποθεσίες στόχων), που πρέπει να επισκεφθούν τα *drones* για την απόκτηση πληροφοριών, διαβιβάζεται σε όλα τα μέλη της ομάδας. Στη συνέχεια, τα *drones* εντοπίζουν τη θέση τους στο περιβάλλον και ξεκινούν τον μηχανισμό δημοπράτησης για το διαμοιρασμό των στόχων, έναν σε κάθε γύρο δημοπρασίας. Στο πλαίσιο της δημοπρασίας, υλοποιήσαμε και δοκιμάσαμε διαφορετικά αντικειμενικά κριτήρια για τον υπολογισμό των προσφορών, τα οποία εξυπηρετούν ποικίλους στόχους σε διαφορετικές αποστολές, όπου η παρούσα εργασία μπορεί να χρησιμοποιηθεί. Στο τέλος της δημοπρασίας, τα *drones* έχουν κατανείμει όλα τα καθήκοντα μεταξύ τους και προχωρούν στην επίσκεψη κάθε ενός από τα ανατεθέντα καθήκοντα με την σειρά που έχει ήδη οριστεί. Μια απλή μέθοδος πλοήγησης με αποφυγή συγκρούσεων διασφαλίζει ότι τα *drones* δεν θα συγκρουστούν κατά τη διάρκεια της διαδρομής που θα ακολουθήσουν. Η εργασία στο σύνολό της έχει υλοποιηθεί στο πλαίσιο του *RobotOperatingSystem*(*ROS*) και διατίθεται ως πακέτο ανοιχτού κώδικα. Όλα τα πειράματα διεξήχθησαν στον προσομοιωτή ρομπότ *Gazebo*, μέσα σε ένα περιβάλλον χωρίς εμπόδια, για να επιδειχθούν τα αποτελέσματα της προτεινόμενης προσέγγισης τόσο σε μικρότερα όσο και σε μεγαλύτερα προβλήματα (20 *drones*, 50 *tasks*). Το κύριο πλεονέκτημα της προτεινόμενης μεθόδου είναι η ικανότητα να εκτελείται με πλήρως κατανεμημένο τρόπο σε μια ομάδα διασυνδεδεμένων *drones* ή γενικότερα άλλων κινητών ρομπότ και να παρέχει μια αποδοτική προσέγγιση για ανάθεση καθηκόντων, ένα βασικό πρόβλημα στις αποστολές έρευνας και διάσωσης.

Acknowledgements

Firstly, I would like to thank my supervisor Michail G. Lagoudakis for his guidance and support during the course of this research and professor Panagiotis Partsinevelos for his confidence, trust and for letting me do this thesis at his laboratory.

Next, I would like to thank my friends and members of the SenseLab team, especially Dimitris Chatziparaschis and Dimitris Trigkakis who stood by me and shared with me amazing experiences and ideas.

My friends Giannis B., Sotiris L., Giannis Gk., Vasilis D., Vasilis G., Andreas S. We shared together some amazing and unique experiences throughout these years, that will always be remembered.

Finally, I would like to thank my family for their endless support, encouragement and unconditionally love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

These days, natural hazards happen nearly on a daily basis and generally require immediate rescue intervention, so as to avert human casualties. Much of the time, the scene conditions can be entirely unusual or can be changed in amazingly quick rates because of factors that can't be constrained by people. Additionally, the conditions might be restrictive for human rescuers to provide instant aid, as they have to act in extreme environments, like air-poisoned situations, radioactive areas or collapsed buildings. Furthermore, sometimes there is a need to immediately examine a wide area, so as to decide an appropriate approaching scenario.

Numerous associations and research groups are creating rescuing robots in order to assist "human" emergency intervention groups. These mobile robots can be geared up with a variety of sensors, cameras and embedded processing units, depending on the scenario they are expected to assist. As a result, due to their specialized structure, they can achieve sufficient maneuverability in the terrain they were designed for. Hence, they can perform searching and reconnaissance procedures by processing the captured data from their sensors. These robots can either be self-sufficient or remotely-controlled, depending on their current setup, and consequently enhance the rescuers' actions.

Nevertheless, in instances of natural disasters, like earthquakes, floods, and fires, the remote control of the robot may be restricted or inapplicable. The control of mobile robots can be accomplished manually by trained pilots, but this approach may not lead to good efficiency, as the coordination between pilots during such events is not straightforward.

The robots needs to be capable to determine their own pose in the environment and communicate with the users. In cooperative multi-robot scenarios, it is also critical for the robots to be able to identify and communicate with other robots during the search-and-rescue operation. By doing so, this ability helps to reduce the search time and increase the number of successful rescues. Furthermore, assuming that the mobile robots can communicate with each other and know their relatives positions in the environment, auction mechanisms can provide fast allocations of their tasks and more efficient results.

Therefore, mobile robots in search-and-rescue missions need to be specially modified and programmed to reach a position where they face problems better than humans could do and cooperate with other robots, in order to accomplish their mission more efficiently.

1.1 Thesis Contribution

This thesis focuses on the search-and-rescue problem of deploying a team of multiple drones and addresses their collaboration with each other. Particularly, we focus on the problem of multi-robot routing, where a team of mobile robots (*drones* in our case) must coordinate to visit and serve a set of target locations.

The tasks for the robots are known locations, which are given to them in the begging of the operation. Firstly, the robot has to locate itself in relation to the environment and the other robots. Then, collaboration is accomplished through an auction-based approach, which is decentralised and is based on a bidding-auctioning process. This process tries to solve the multi-robot task allocation problem sequentially in multiple rounds, whereby the robot with the best bid wins one task in each round. A task gets assigned to the most suitable robot, according to the bidding value calculated using an appropriate cost function based on distances between robots and tasks. Finally, when all the tasks are allocated, each robot follows the path of the tasks it has won.

This collaborative approach is designed to make the best out of the auction process, with respect to robots' computational capabilities, as the requirements for communication and computation are rather low. The entire project has been implemented as a package within the Robot Operating System (ROS) and is supported for any multi-robot system, including any type of mobile robots within a natural terrain.

1.2 Thesis Outline

In Chapter 2 we present all the background information needed for this thesis. We give a brief overview of the software and the basic components we use in this thesis, such as the Robot Operating System, Gazebo and RViz. Furthermore, we provide basic information and knowledge about mobile robot localization. In Chapter 3 we state the problem of cooperative multi-robot response to natural disasters, referencing related approaches. In Chapter 4, we describe in detail the implementation steps of the proposed cooperative multi-robot approach based on distributed auctions. In Chapter 5, we present the results of our implementation through different simulated scenarios. This work is concluded in Chapter 6, in which are presented some future plans that could extend our approach.

Chapter 2

Background

In this chapter, we provide a brief overview of the software, the algorithms and the technologies used to implement the approach of this thesis, in order for the readers to further comprehend and understand each part of the development process. The topics that are going to be examined, which are critical for the comprehension of the development process, are the Robot Operating System, Gazebo, RViz and the basics of mobile robot localization.

2.1 Robot Operating System (ROS)

The Robot Operating System(namely ROS) [26] is a framework designed by Willow Garage and Stanford University as a part of STAIR project, as a free and open-source robotic middleware for maintenance and development of complex robotic applications. The primary advantage of ROS is that it permits different devices to coexist in the created environment and interact with each other through a peer-to-peer network.

Each device in this space represents a node, that is a process of performing computation. A node is able to communicate with another one by passing a message. A message is carefully composed information structure, which supports standard primitive types (like integer, floating point and so forth), arrays of primitive types and arrays of different messages. A node sends a message by publishing it to a given topic. If a node is interested in a particular sort of information, it will subscribe to the suitable topic. There may be multiple simultaneous publishers and subscribers for a single topic and

```
Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

Figure 2.1: A PoseStamped Message.

a single node may publish and/or subscribe to multiple topics. Generally, publishers and subscribers are not aware of each others' presence.

The message shown in Figure[1] 2.1 is a PoseStamped message, that any drone can publish. Every PoseStamped message comes with a *header* parameter that includes the *time stamp* and the *frame* information, which identifies it uniquely in time and gives information of the frame that it was taken in respect to. Also, *position* and *orientation* define the position and the orientation of the drone in that frame and time.

In contrast with topics, in case of a Remote Procedure Call (RPC) request/reply interaction, there is a need of two-way communication. Especially, given a node with a providing service, a client firstly calls the service by sending a request and awaits service node's reply, after the service initialization. However in services' case, only one node can advertise a service with a certain name. Lastly, the communication can also be done via a parameter server, which is a shared multi-variate lexicon that is available by means of network APIs. Nodes utilize this server to store and retrieve parameters at runtime. This way of exchanging data is used for sharing static, non-binary data such as configuration files. Basically it is a shared, multi-variate lexicon that is accessible from all nodes to store and retrieve data from.

[1] https://docs.ros.org/diamondback/api/geometry_msgs/html/msg/PoseStamped.html

For example, a ROS environment could be a single drone with a camera that publishes on topics the information about all its sensors and his pose. It could also, publish a topic of the localization process that executes on its embedded processor. Furthermore, it could provide two services, like setting the camera parameters or resetting the belief of it's position, in which an another computer or drone on the same ROS environment could call. Also, there could be a scenario including of two connected drones in the same ROS environment, in which through communication and cooperation will explore an unknown area. In this scenario, those drones can share information through their published topics about their state and belief and by the data fusion they can cooperate and navigate in the unknown environment and accomplish certain given tasks.

In conclusion, ROS framework usability, wide-range of capabilities and it's open source philosophy are some of the reasons for choosing it as the basis framework of our scenario. In particular, our vision is to create a distributed network of drones that through the process of action will allot the tasks in between them. By using this ROS framework, this project can be applied in every multi robot scenario, not specifically drones, that have the appropriate modifications.

2.2 Gazebo

The Gazebo [16] software is making a 3D dynamic multi-robot environment capable of reproducing the complex scenes and worlds that will be experienced by robots in the real world. As a result, Gazebo is intended to precisely recreate the dynamic environments a robot may experience. Every simulated object has mass, velocity and various other different attributes that enable them to behave realistically in simulations. These actions can be utilized as vital parts of an experiment, like in our scenario.

Gazebo has a noteworthy element to effortlessly making new robots, actuators, sensors, and arbitrary items. Thus, Gazebo keeps up an easy API for addition of those objects, that we will in general term models, and also the important hooks for interaction with client programs. Gazebo has many dependencies, as seen in Figure 2.2 [14], and utilizes a various number of third party libraries to represent the physics, the visualization and the rendering. Gazebo utilizes ODE (Open Dynamics Engine) [23], Bullet [2], Simbody [28] and Dynamic Animation and Robotics Toolkit

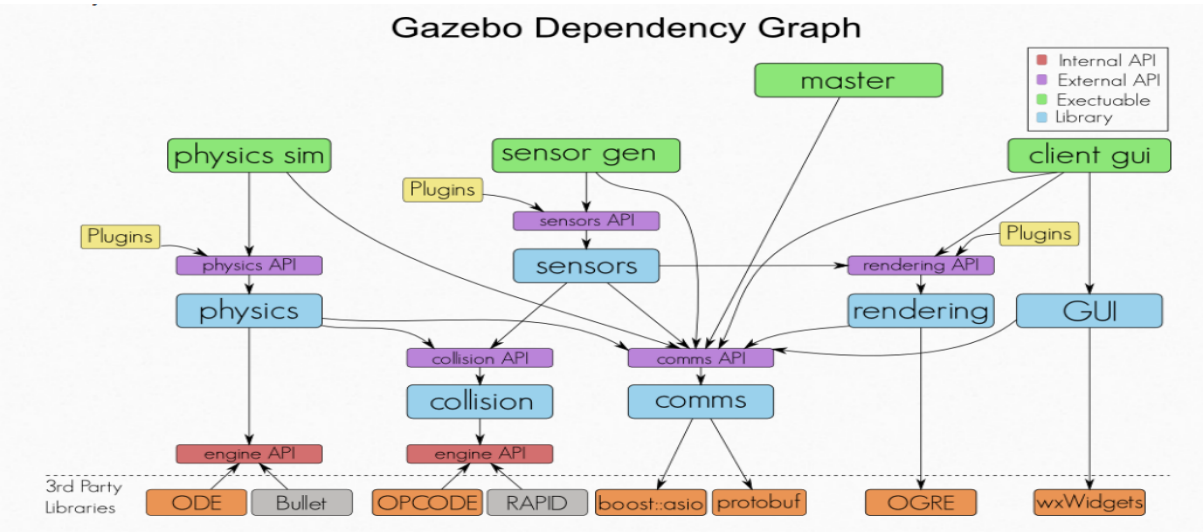

Figure 2.2: Gazebo Dependency Graph.

(DART) [4] to mimic the dynamics and kinematics related with articulated rigid bodies. OpenGL [24] and GLUT [7] are utilized as the default visualization tools. So as to have correspondence between various segments in gazebo, it uses open source Google Protobuf [12] for the message serialization and boost::ASIO [1] for the transport mechanism. Gazebo uses OGRE [22] as a rendering library for rendering 3D scenes to both GUI and sensor libraries. Lastly, Gazebo provides Qt for users to interact with the simulation.

Gazebo provides an exceptionally basic and simple approach to access the resources by using plugins, in order to control objects in a simulation. A plugin is a lump of code that is compiled as a shared library and is inserted into the simulation. There are six sorts of plugins, which are the World,the Model,the Sensor,the System,the Visual and the GUI. Each one of them gives access to an alternate part of the model, for instance: a Sensor and a Model plugin controls every aspect of a sensor and a model respectively.

In spite the fact that plugins are the least demanding and an ideal approach to access the assets, yet there is an alternate way also. Access to the simulator can be done through the utilization of Gazebo's APIs. We can do the exact things that Gazebo does at the time of startup and make pointers to every element. In any case that technique

requires a very great degree of comprehension of how gazebo functions and how it utilizes its APIs.

A Gazebo simulation to function properly has seven fundamental components [3]. The first component is the *World Files*, which contains description about all the elements in a simulation. That includes the robots and their sensors, the static items and the lights. This component is formatted by utilizing the SDF library [27] and commonly has a .world expansion. The *Model File* contains all the data about a specific entry. Those data are connected and behave as a single body and this type of file is formatted by utilizing the SDF library.

The next component is the *Environmen variables*. Gazebo utilizes a few of them to know about the files which are going to load at startup. Those files include word files, model files, plugins and so forth. An other component of Gazebo is the *Server*, which is and its workhorse. The Server given on the command line, it parses a world description file and then it simulates the world by utilizing a physics and sensor engine.

The last two components are the *Graphical Client* and the *Server and Graphical Client*. The first one, of those two, connects with a running *gzserver* and visualizes the components. This is additionally a tool which enables you to adjust the running simulation. For the other one, the Gazebo command combines *server* and *client* in one executable.

To conclude, Gazebo software is an essential tool in our scenario. By utilizing realistic scenarios in a well-structured and designed simulator, we can quickly test algorithms, train AI systems and design robots. Gazebo offers the ability to precisely and efficiently simulate robots in complex conditions in indoor and outdoor environments. For that reasons Gazebo is the perfect tool for the creation of a distributed network of drones in an outdoor environment. By using Gazebo, this project can be simulated in different scenarios with exceptional resemblance to real life situations.

2.3 RViz

In computational science and computer graphics, it is very useful to visualize and represent information in the real domain. RViz [15] is a 3D visualization toolkit for ROS, that can visualize newly invented data structures and algorithms and is also, independent of the input information data structures.

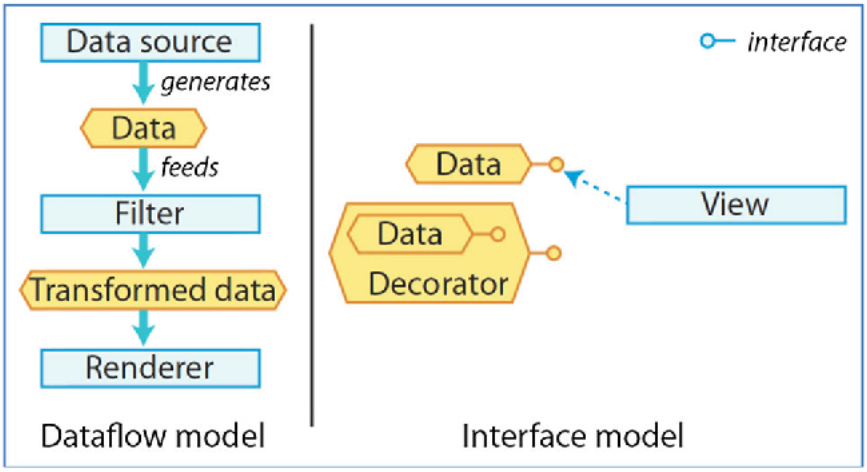

Figure 2.3: RViz dataflow and interface model.

RViz makes easy for the user to view the simulated robot model. Also to look at the log sensor data from the robot's sensors and it can replay the logged sensor data. Furthermore, from the visualization of what the robot is "seeing" and "doing", meaning the sensor inputs and the planned actions, the debugging process of a robot application is made much simpler. In the form of point clouds or depth images we can see, in RViz, the display of 3D sensor data (from sensors like lasers, etc.). On the other hand, as image data we can view the 2D sensor data (from sensors like cameras) and 2D laser range finders.

If a real (or a virtual) robot is communicating with a computer system that is running RViz, then the program will display the robot's current configuration on its representation in the RVIz environment. ROS topics will be shown as depictions of the sensor data, that are published by sensors that exist in the robot's system. Robot systems and controllers can be developed and debugged more easily. Rviz provides a configurable Graphical User Interface (GUI) to enable the user to show just the data that he needs for his task.

For example, if we had a dataflow model like the one in Figure[1] 2.3 (left side), informations are caught in specific data structures and after that they are changed

[1] www.semanticscholar.org/paper/RViz3A-a-toolkit-for-real-domain-data-visualization-Kam-Lee/e4ab668238eebb31801a2c2c19d4517aae4baabf/figure/0

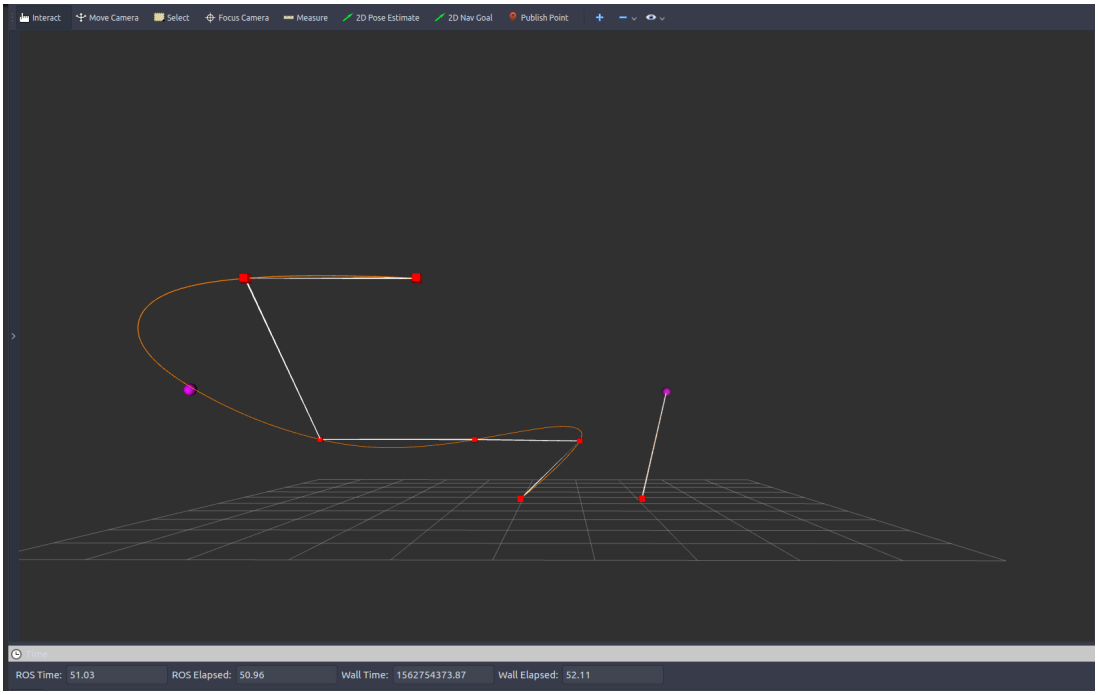

Figure 2.4: RViz interface example.

by some filters. On the contrary, in Figure 2.3 (right side),information is viewed as an execution of the basic interface and this enables adaptability in choosing what to display in the GUI, different data structures and algorithms that are pertinent to his task.

To summarize, the visualization of real domain data, that are created by arbitrary data structures and algorithms, is the main aim of the RViz toolkit. An example of the RViz interface is shown in Figure 2.4. It is the package that used in this thesis for the visualization of algorithms and data structures and it helped a lot in the different scenarios that we implemented.

2.4 Mobile Robot Localization

Mobile robot localization [21] is a standout amongst the most crucial issues in robot navigation and is the problem of deciding the pose of a robot in respect to a given environment/map. In a scenario that the robot requires knowledge of its location, namely to establish correspondence between its local coordinate system and the global

coordinate system, so it can express the location of any surrounding objects or even obstacles in a relation to it. The localization issue can be solved by exploiting robot sensors data that regularly originates from two unique sensors, the motion sensors and the perception sensors. Motion sensors give information about the robot movements (e.g odometer readings) and the perception sensors give spatial data relative to the robot position.

The localization problem can be divided in three main categories, depending on the type of knowledge that is available at the first moment of the navigation process and at run-time, which are with an increasing degree of difficulty as follows [31]:

- **Position Tracking** : In position tracking the initial pose of the robot is known and this problem is characterized as the issue of localization of a moving robot. We assume that the robot's motion model error is small and can be approximated. The solution of the localization problem is given through the odometry information, by compensating the robot's motion model uncertainty, we update the local robot pose belief in relation with the initial pose.

- **Global Localization** : is the issue when the robot has the need of locating itself in a known environment and it has no information about its initial pose. The robot must exploit various observations from its sensors, to infer its location in the environment and then proceed with the position tracking approach.

- **Kidnapped robot problem** : During this scenario, that is a variation of the global localization problem, the robot can be moved whenever and set to a different area in comparison to the one it was, without getting notified. The main objective is to make the robot realize that it was kidnapped and then adapt accordingly (e.g. initialize its belief about its pose in the environment).

2.4.1 Robot Pose

All mobile robots that operate in planar environments have their state described by a three dimensional vector, as shown in Equation 2.1, this is known as a robot's pose. Robot's pose is described, at time step k, by three variables, the two translational in

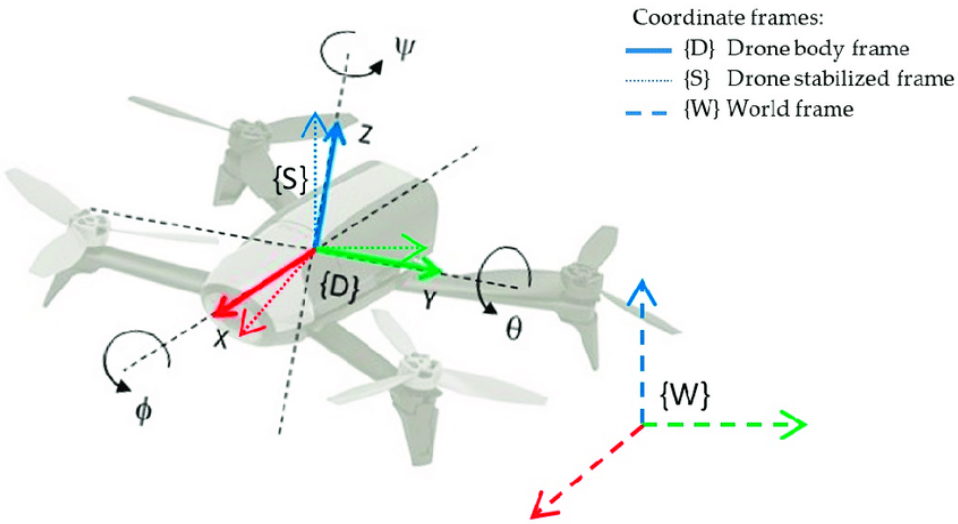

Figure 2.5: Drone poses in a 3D global coordinate system.

each dimension (x and y) and one that defines the angular orientation (θ). In three-dimensional space, the pose is described by six variables, the robot's three dimensional Cartesian coordinates (x, y, z) and its three Euler angles (ϕ, θ, ψ), describing the translation and rotation of its body relative to an external coordinate frame as shown in Equation 2.2. For example, the robot pose of a drone in a 3D coordinate system can be shown in Figure 2.5 [13].

$$\mathbb{R}^2 : \mathbf{x}_k = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{2.1}$$

$$\mathbb{R}^3 : \mathbf{x}_k = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \textit{ and } \boldsymbol{\theta}_k = \begin{pmatrix} \phi \\ \theta \\ \psi \end{pmatrix} \tag{2.2}$$

2.4.2 Motion Model

A mobile robot has most of the times some type of locomotion mechanisms to be able to move and navigate in an environment. They are equipped with actuators like servos, motors etc. for that purpose. Nonetheless, the use of these actuators can be noisy and may result to a difference between the purposed motion and the one that actually happens. With this error, that gradually is added to the odometry information, the localization vadility is affected. To solve this problem, robots are fitted with additional sensors that measure the occured deviation. After the action has taken place, the data about the real action that happened are available. Hence, given the current state of the robot x_k and the current control input u_k (*"the action"*), the robot can make a transition to a new state x'_k. These transitions description is the probabilistic model that is named ***motion model*** and is formed as:

$$P(x_k|x_{k-1}, u_k) \tag{2.3}$$

This model describes the posterior distribution of the robot pose x_k in time step k, given that at time step $k-1$ had the pose x_{k-1} by performing the action u_k . Last to note, that the *motion model* adopts the Markov property of the memoryless stochastic process, as it depends only on the previous pose and the current action.

2.4.3 Sensor Model

A reliable localization process cannot be done without any external sensors and based only on the *motion model*. There must be external sensors that can provide additional information of the robot's environment, like a range finder, a camera or even and a global positioning system. So, the robot can be able to locate features in its surrounding environment and estimate its position in it.

Unfortunately, sensors also have noise to their measurements. For this reason, we can describe in probabilistic terms the sensing of the robot, as a conditional probability distribution. This probabilistic model is called ***sensor model*** and is written as follows:

$$P(z_k|x_k) \tag{2.4}$$

This Equation 2.4 describes the likelihood of making the observation z_k at time step k, given the pose x_k at the same time step. Likewise, this model adopts the Markov property, as *motion model* in Chapter 2.4.2, specifically given the latest state, observations are independent of history.

Chapter 3

Problem Statement

3.1 Drone Teams in Search-and-Rescue Scenarios

Search-and-rescue operations are performed every day in emergency situations, which are caused by natural disaster outbreaks or are due to human error and sometimes (unfortunately) due to human intention. Natural hazards have, sadly, become a very common phenomenon lately, with numerous recent instances, like forest fires that cleared out many sections of forest area, destroyed properties and sometimes resulted in human losses. Subsequently to the way that the majority of these perils demonstrated are amazingly dangerous, it is critical for search-and-rescue teams to be well prepared to face and resolve them in order to maximize the number of survivors. By employing a team of multiple drones we can address those issues.

Fast allocation of targets in a search-and-rescue mission is critical for its success. Particularly, in most cases the operation of a drone is performed by a user and the allocation of tasks is done using his judgment, which in many cases is not the best. By using a sequential auction algorithm with the appropriate cost function, we can improve the efficiency of the allocation and eliminate human user errors. Clearly mobile robots need to have the ability to allocate the tasks efficiently for better results in such scenarios and by using the proposed auction algorithm we can achieve that.

In addition, multi-robot approaches are often suggested, as they have several advantages compared to single robot systems, especially in search-and-rescue scenarios. Through their cooperation, robots can manage and assign each requested task accord-

ing to the current system status, they can merge overlapping information and thus accomplish single tasks faster than a single robot system would do. However, these approaches require careful fine tunning and sufficient problem awareness, in order to create an effective cooperative solution.

In conclusion, this thesis will focus on a decentralised, multi-round, sequential auction mechanism, providing near-optimal allocations of the tasks given, by using different cost objectives. In addition, we simulate a number of task allocations scenarios with n mobile robots (drones in our case) and m tasks (locations in our case) within an obstacle-free environment. We aim to provide a system that could be easily implemented on multi-robot teams and used in real-time scenarios.

3.2 Related Work

In recent years, auction based systems have been an area of increasing research interest. The problem of Multi-Robot Task Allocation (MRTA) is fundamental in multi-robot systems and it has been modelled in a number of different ways, the most representative method is CNP(Contact Network Protocol) which was proposed by Smith [29]. Other typical examples include The Power of Sequential Single-Item Auctions for Agent Coordination [17] and MURDOCH [10]. This is a non-deterministic polynomial-time (NP) hard problem [20] and can require a large amount of time to solve efficiently. These solutions can be segmented into two categories centralised and distributed.

For multi-robots systems, like ours, distributed solutions are more effective. A highly efficient distributed solution uses auction-based task allocation [18], in which tasks are locally sold to nearby robots and the more capable robots are able to bid higher than the less capable ones. Also, there can be used bidding rules to minimize the total cost, the maximum cost and the average service cost, like in [19] where the authors presented the first theoretical complexity analysis of the performance of auction-based methods for multi-robot routing for different objective functions.

Finally, the implementation of these algorithms is facilitated by using the Robot Operation System (ROS), described in Section 2.1. One of them is a behavioural task allocation architecture known as ALLIANCE [25], which recently was implemented

in ROS [5]. The one that is close related to our work is a ROS package which is called *task allocation* [1], which implements a centralised auction mechanism.

Our approach is differentiated from these works by essentially being a package for ROS and not a research conjecture of how these algorithms work and are optimized. ROS middleware is used as the network infrastructure, due to its open-source philosophy and its high modularity on package construction and deployment. Furthermore, our implementation is not a centralised auction, but a ***distributed sequential auction***, that allocates the tasks to robots, by using certain cost objectives, given a series of tasks and a number of robots.

[1] https://github.com/Nick-Sullivan/task_allocation

Chapter 4

Our Approach

4.1 Total Coordinate System Setup and tf Library

Robots are complex systems as they have a lot of moving parts, sensors and other complex mechanisms. When doing assignments with a robot it is significant that the robot knows about where it is itself as well as where the rest of the world is in relation to itself. Like in our scenario, where many drones coexist in the same environment and need in every moment to know their position in relation to the world. This creates a need to represent the position and orientation of the robots, including their sub-systems parts, in the same coordination system.

After we establish a coordinate system, we cal locate any point P in the coordinate frame A with a $3x1$ vector, like a point in space, as we see in Equation 4.1. Thus, it is fundamental to represent a point in space and also to describe the orientation of a body. To accomplish that we attach a coordinate system to the body and then we give a description of this coordinate system relative to the reference system. So, on a coordinate system A, the description of body-attached coordinate system B can be written by expressing the unit vector if its three principal axis in reference to A.

$$^{A}\mathrm{P} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \tag{4.1}$$

As shown in Figure 4.1, we denote the unit vector giving the principal directions of coordinate system B as the $\hat{X_B}$, $\hat{Y_B}$ and $\hat{Z_B}$. Also, the principal direction vectors

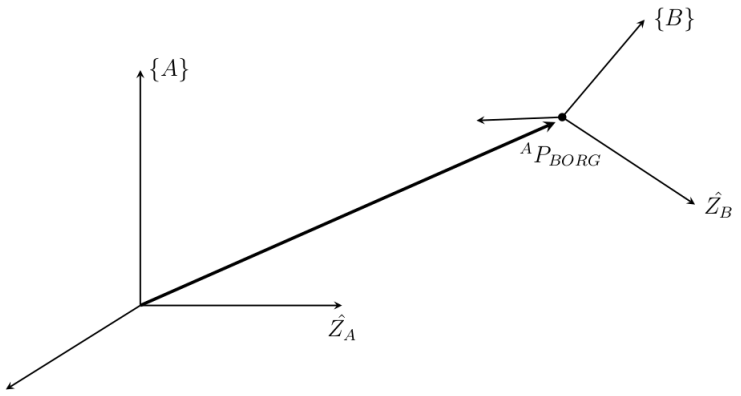

Figure 4.1: The relative position of A and B coordinate systems.

are denoted in coordinate system A, as ${}^{A}\hat{X}_B$, ${}^{A}\hat{Y}_B$ and ${}^{A}\hat{Z}_B$. Therefore, we define the matrix of the expressed B principal axis in A coordinate system as in Equation 4.2, which is known as the rotation matrix, as it describes the frame B relative to A.

$$ {}^{A}_{B}R = \begin{bmatrix} {}^{A}\hat{X}_B & {}^{A}\hat{Y}_B & {}^{A}\hat{Z}_B \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{4.2} $$

Also, Equation 4.2 can be expanded as follows,

$$ {}^{A}_{B}R = \begin{bmatrix} {}^{A}\hat{X}_B & {}^{A}\hat{Y}_B & {}^{A}\hat{Z}_B \end{bmatrix} = \begin{bmatrix} {}^{B}\hat{X}^{T}_{A} \\ {}^{B}\hat{Y}^{T}_{A} \\ {}^{B}\hat{Z}^{T}_{A} \end{bmatrix} = {}^{B}_{A}R^{T} \tag{4.3} $$

Thus, ${}^{A}_{B}R$ is the description of the frame B relative to A and is given by the transpose ${}^{B}_{A}R$, which is also the inverse of the rotation matrix. Hence we have,

$$ {}^{A}_{B}R = {}^{B}_{A}R^{-1} = {}^{B}_{A}R^{T} \tag{4.4} $$

Thence, the frame of B is described relative to A as,

$$ B = \left\{ {}^{A}_{B}R, {}^{A}P_{BORG} \right\} \tag{4.5} $$

Lastly, the mapping/transformation of a point P in coordinate system B to A, is denoted by ${}^{A}_{B}T$ and forms as follows,

$$ {}^{A}P = {}^{A}_{B}T, \text{ where } \quad {}^{A}_{B}T = \left[\begin{array}{c|c} {}^{A}_{B}R & {}^{A}P_{BORG} \\ \hline 000 & 1 \end{array}\right] \tag{4.6} $$

However, many complex robot systems are consisted with parts that are moved with respect to the base frame. This means that the transformation matrix between two specific moving parts (frames) is differentiated from time to time. Thus, it's necessary to store the time-stamp information when any of those transformation is calculated, despite if it's static or not. The ROS communication infrastructure, which we discussed in Section 2.1, is based on timestamps and frames to express measurements, robot poses and more, as they were posted at a given time in the corresponding coordinate system. The ROS ecosystem has an integrated transform library, the *tf* [6], which follows the aforementioned "frame philosophy", supports fully the previous transformation methods and is for the most part utilized as essential approach to monitor positional data. Therefore, it is extensively used in our approach as we need constantly information about the robots' relative poses and their positions in the generated map frame.

Thus, we created a dedicated ROS node to provide the hexacopter static transforms. We precisely measured the sensor position in reference to a common hexacopter body point (assumed the *base frame* origin) and included the resulted real world positions into the *tf* tree. By this way, the *tf* tree contains frame transformations described in real world units, and hence, any processed measurement and extracted results will be spatially described in the same way. The hexacopter *tf* tree node is illustrated in Figure 4.2 and its structure visualization in the RViz plug-in, shown in Figure 4.3.

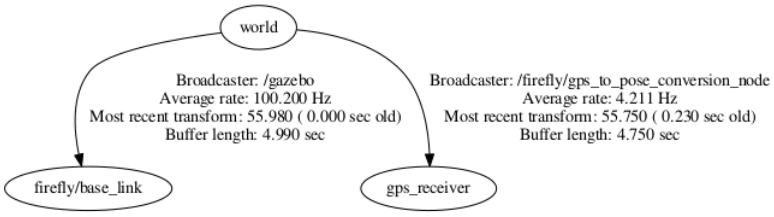

Figure 4.2: Hexacopter firefly *tf* tree in frames.

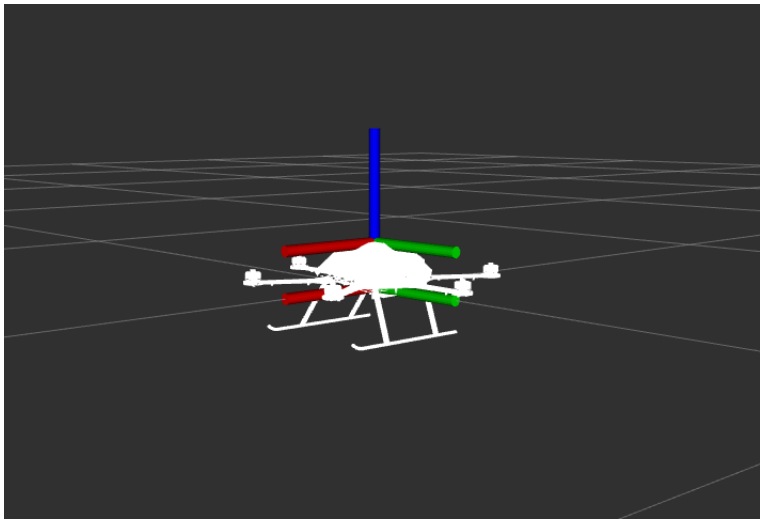

Figure 4.3: Hexacopter firefly *tf* tree in RViz.

In conclusion, by using the *tf* library every hexacopter have perception of its exact position in relation to the environment-wolrd that its inside, in every frame. That is crucial for the follow-up process.

4.2 Distributed Sequential Auction

4.2.1 Classification

Initially, we need to attempt to respond to the principal query of which robot ought to execute which task. This question is actually a multi-robot task allocation problem (MRTA). The task allocation downside tends to the subject of finding the task-to-robot assignments that improve utility objectives or global cost. Consequently, most of basic methodologies are approximate or heuristic in nature. MRTA might be an essential issue of the multi-robot systems, which epitomizes the high-level system organization and operation mechanism. The performance of a multi-robot system is legitimately impacted by the quality of the task allocation algorithm.

Thus, we followed a classification that proposed by Gerkey and Mataric [11] for the task allocation problem. It is portrayed in the following way,

- *Single-task robots (ST) vs. multi-task robots (MT)*: single-task robot indicates that every robot is able to execute at most one task at a time, in opposition to multi-task robot, which indicates that some robots are capable of executing many tasks synchronously.

- *Single-robot tasks (SR) and multi-robot tasks (MR)*: single-robot tasks are tasks that demands at most one robot to achieve it, in opposition to multi-robot tasks, where some tasks demand many robots.

- *Instantaneous (IA) and time-extended (TA) assignment*: In Instantaneous assignments there is no arrangement for future allocations and the robots are only concerned with the task that they are executing or plan to execute. On the contrary, in time-extended assignments more information is available and they can turn up with longer term plans involving task schedules.

Of the above taxonomy, we find eight different combinations of task allocation. The combination ST-SR-IA is the simplest, and is actually a case of the *Optimal Assignment Problem* (OPA) [9]. This combination of robots, tasks and assignments is the one that we used in our scenario. Every drone can execute one task at any given moment and every task demands exactly one robot to achieve it. The data concerning the robots,

the tasks and the map/environment allows only an instantaneous assignment of the initially tasks to robots, with no arrangement for future task distributions.

4.2.2 Auction Mechanism

In multi agents systems, auctions have become common place when allocating tasks. There is a variety of auctions formats that can be used when allocating a group of tasks, like in our scenario where drones allocate a group of tasks that they must visit. The most well known mechanisms are the sequential and the combinatorial auctions.

The sequential auction is an auction in which several tasks/items are allocated by the seller. Throughout the auction, the format stays the same for early and late tasks. One after the other the tasks are allocated to the same group of bidders. As the auction proceeds sequentially, the bid value of an additional task depends on the number of tasks procured by the bidders up until this point. Complements arise if the value of an additional task increases in the number of acquired tasks, as long as substitutes arise if the value decreases in the number of acquired tasks.

On the other hand, combinatorial auctions allow bidders to submit bids to a number of tasks/items, that are auctioned simultaneously. Then for each task separately, the price and the allocation is resolved, depending entirely on the bids that are submitted on it.

We selected to implement a sequential auction in this thesis. Combinatorial auctions can be computationally complex to implement, although they provide economically efficient allocations. In our multi-drone scenario, a sequential auction fits better than a combinatorial. Drones have minimal processing power and computationally complexity of the implementation of combinatorial auctions is sometimes too heavy to handle.

Additionally, there was a dilemma between a distributional and a centralized approach. A distributional approach, as we saw in Section 3.2, is more common and is also highly efficient for the MRTA problem. On the other hand, a centralized approach will prove restrictive in a multi-agent scenario, as too many bidders (drones in our case) may overload the server-auctioneer. Hence, making the whole auction process less responsive than the auctioneers and buyers would prefer.

For those reasons, we conclude that the best combination of auction mechanisms is a ***distributed sequential auction***. Every drone can participate in this auction model with minimal computational power. The need of a central server that allocates the tasks is eliminated, making the whole system more portable, which is a crucial need in search-and-rescue scenarios.

4.2.3 Auction Framework

Our auction-based coordination system for multi-robot task allocation follows the steps as described bellow,

1. Our system considers the robots as bidders and therefore the tasks as merchandise. All tasks are initially unallocated and known to every robot. Then each robot creates its topic and subscribes to the newly created topics of the other robots, as we saw in Section 2.1.

2. During every round of bidding, all robots bid on unallocated tasks. Firstly, every robot calculates its bid for every unallocated task. Then it selects the lowest bid and bid it for the round.

3. Hence, each robot individually publishes the selected bid to the created topic and wait until all the robots in the environment made a bit or until a given default time.

4. After the collection of the round's bids is complete, through the subscriptions, each robot finds the lowest bid of the round locally and automatically learns the winner of the task.

    - If the robot is the winner, it allocates that specific task to its path and removes it from the unallocated task list.
    - If the robot is not the winner, it changes the status of the unallocated task to allocated.

5. A new round of bidding will reinitiate , as explained in steps 2, 3 and 4 until all the tasks have been allocated to robots.

A pseudo-code of this described algorithm can be seen at Algorithm 1. It is needed to emphasize the fact that each robot needs to bid only on a single task at each round. In particular on the task for which its bid is the lowest and then all that bid, since all the other bids from the same robot have no chance of winning. In addition, we point out that each robot publish its bid to a certain topic and subscribes to the all the respectively topics of the other robots. Through these subscriptions, every robot receives the bids of the other robots in each round.

Upon allocation of all tasks, every robot computes a path for visiting the tasks allocated to it and then moves along that path. If no tasks are allocated to a robot, then it stays at its initial position. Satisfactory bid choice and path calculation are key factors in group performance.

The primary advantage of this multi-round auction mechanism is its simplicity and the fact that it allows for a decentralized implementation on real robots. Without any doubt, there is no requirement for a central auctioneer and as a result there is no single purpose of failure.

4.2.4 Cost Objectives

As we indicated before, the objective of multi-robot routing is to find an allocation of tasks to robots, along with finding a path for each robot. Then the robot visits all its tasks that was allocated to it. All that is done, so a team objective can be optimized. Through the bidding process of the auction we implemented, we try to optimize this team objective.

We focus our research in three team objectives, which are the MiniSum, the MiniMax and the MiniAve [19]. As we have seen in Section 4.2.3, each robot must calculate a set of bids independently for each round. The ***bidding algorithm*** decides the value of the bids and sequentially which tasks they bid on.

For each one of the three objectives, we created the corresponding bidding algorithm. The user can, through input, determine which algorithm will run, depending on the scenario he implements. The bidding algorithms are as seen below,

1. **MiniSum**
   The MiniSum objective is to minimize the sum of robot path costs over all robots. In respond to this objective, we created the *MiniSum Algorithm*. Every task is bid

Algorithm 1 Auction Framework

```
1: Input:
2: n ← number of robots
3: m ← number of tasks
4: Initialize: publisher → p, subscribers → {s_1, ..., s_n}, queue.Add(tasks)
5: if (n <= 0) or (m <= 0) then
6:     done
7: else
8:     while queue not empty do
9:         bids ← Calculate Bids
10:        bid ← Select min (bids)
11:        publish ← bid
12:        finalBids ← other robots bids, bid
13:        FindWinner (finalBids)
14:        if winner then                                  ▷ Round Winner
15:            add task to path
16:        end if
17:        queue.Remove(allocated task)
18:    end while
19: end if
```

on and the bidding amount is determined by the following. To determine the bidding value on every unallocated task we perform the following,

- Determine the cost of completing both the currently allocated tasks and the task being bid on.
- Calculate the cost of completing currently allocated tasks.

The final bid is the *difference* between those costs. So, by using this MiniSum algorithm, we allocate the tasks to the robots that can complete them with the smallest extra cost.

2. **MiniMax**
   The MiniMax objective is to minimize the maximum robot path cost over all robots. In respond to this objective, we created the *MiniMax Algorithm*. Every

task is bid on and the bidding amount is dictated by computing the cost of completing both the currently allocated tasks and the task being bid on. The final bid is the cost of the path. With this bidding algorithm, the tasks will be alloted fairly evenly between robots.

3. **MiniAve**
   The MinAve objective is to minimize the average target path cost over all targets. The *MiniAve Algorithm* was created in response to that objective. Every task is bid on and the bidding amount is determined by calculating the cumulative average cost of completing both the currently allocated tasks and the task being bid on and then subtract the cumulative average cost of completing the currently allocated tasks. This algorithm tries to balance the solutions given by the MiniSum and MiniMax algorithms and for that reason the solutions are somewhat in between.

From the three biding algorithms that was implemented, the MinAve Algorithm is the most optimal for search-and-rescue missions. In this missions we want the drones to coordinate between each other to accomplish tasks in the minimum amount of time while saving the maximum number of survivors.

4.2.5 Auction Resolution Algorithms

The last step in every round is to properly allocate the task to the winner. After the creation and the collection of the bids in Section 4.2.4, the selection of the winning bid is done by the *auction resolution algorithm*. There are two different algorithms that we implemented. Those are as follows,

i. **Lowest Bid**
   This algorithm finds the lowest bid and assign the task to its bidder. It is the most common approach to these MRTA problems.

ii. **Least Contested Bid**
   In this algorithm [30], we find the the task with the largest difference between the two lowest bids. Then, we allocate to the bidder the task. So, we have faster allocation of tasks that can be completed by one robot.

In our implementation, we set as default the *Lowest Bid* algorithm, which is and the most common. If the user wants to allocate the tasks with the *Least Contested Bid* algorithm, he must give and the appropriate input in our package.

4.3 Waypoint Navigator

Following the finale of our auction, every robot has a set of tasks. Each robot must follow those tasks with the order it won them, namely to follow the waypoint path it created through the auction. We wanted a waypoint follower that can simulate realistically real-life scenarios and make easy the transition from simulation to real robots. The Waypoint Navigator[1] package that was developed by the Autonomous Systems Lab in ETH Zurich, is a perfect match to what we want to achieve.

This package contains a high level waypoint following mechanism for drones. An example of this, is provided in the RotorS simulator [8]. An instance of this package is shown in Figure 4.4. With this package we have secured a high-end trajectory tracking controller with collision avoidance and path planing. We opted for using this package and not creating our own, because it made easier the transition from simulation to real drones.

Figure 4.4: A screenshot of the RotorS simulator.

[1] https://github.com/ethz-asl/waypoint_navigator

The connection of the waypoint navigator with our work was done with the use of services. As we saw in Section 2.1, the publish-subscribe model is a very adaptable communication example, however its many-to-many one-way transport is not proper for RPC request-reply interactions, which are frequently required in a distributed system like our own. Each drone, at the end of the auction, sends its waypoint path to the waypoint navigator. The waypoint navigator receives the waypoints as a list of geometry messages [1] and command the drone to follow the path.

[1] http://docs.ros.org/jade/api/geometry_msgs/html/msg/Pose.html

Chapter 5

Results

In this chapter, we will see how the auction mechanism, that was explained in the previous chapter, works through three scenarios. All the scenarios belong in the ST-SR-IA case, as seen in Chapter 4.2.1 and in every scenario is used the MiniSum cost objective **1.** and Lowest Bid Algorithm **i.**. That is done so we can compare more easily the results. Simulations are performed many times, so we can confirm that they produced the same results, as expected.

5.1 Scenario 1

This scenario involved two drones and six tasks. The tasks are randomly placed in a (5x5x3) metre area. The task are as follows in the table below,

	x	y	z
task 1	-4	2	1
task 2	-3	-4	3
task 3	-1	-4	3
task 4	-1	2	1
task 5	3	4	2
task 6	1	1	1

Table 5.1: Unallocated tasks coordinates.

And the drones *firefly0* and *firefly1* are at positions $(0,0,0)$ and $(2,0,0)$ respectively.

After they locate themselves in the environment, they start the auction process, as described in Chapter 4.2.3. Firstly, they create a set of bids of all the tasks and select the lowest bid (in red) and bid it for the round.

	task1	task2	task3	task4	task5	task6
bids	4.6	5.8	5.1	2.4	5.4	1.7

Table 5.2: *firefly0* round 1 bids.

	task1	task2	task3	task4	task5	task6
bids	6.4	7.1	5.8	3.7	4.6	1.7

Table 5.3: *firefly1* round 1 bids.

Then, they publish their bid and collect the bid of the other drone. Each now has the following table,

	task1	task2	task3	task4	task5	task6
firefly0	-1	-1	-1	-1	-1	1.7
firefly1	-1	-1	-1	-1	-1	1.7

Table 5.4: Round 1 winner selection.

where the winner is *firefly0*, because when two or more robots bid the same value the winner is the first robot alphabetically. Then, *firefly0* adds the task to its path, *firefly1* marks task6 as allocated and we continue to the next round.

	task1	task2	task3	task4	task5	task6
firefly0 bids	5.1	6.7	5.7	2.2	3.7	-1
firefly1 bids	6.4	7.1	5.8	3.7	4.6	-1

Table 5.5: Round 2 bids.

In round two the same process repeats itself. As we see in Table 5.5, the drones select their lowest bid for the round and bid it. Then the lowest bid is found and consequently the winner (Table 5.6), which is *firefly0* for task 4.

	task1	task2	task3	task4	task5	task6
firefly0	-1	-1	-1	2.2	-1	-1
firefly1	-1	-1	-1	3.7	-1	-1

Table 5.6: round 2 winner selection

The same process repeats itself until all the tasks are allocated. As we see in the tables below,

	task1	task2	task3	task4	task5	task6
firefly0	3	6.6	6.3	-1	4.6	-1
firefly1	6.4	7.1	5.8	-1	4.6	-1

Table 5.7: round 3 bids

	task1	task5
firefly0	3	-1
firefly1	-1	4.6

Table 5.8: round 3 winner

	task1	task2	task3	task4	task5	task6
firefly0	-1	6.4	7	-1	7.3	-1
firefly1	-1	7.1	5.8	-1	4.6	-1

Table 5.9: round 4 bids

	task2	task5
firefly0	6.4	-1
firefly1	-1	4.6

Table 5.10: round 4 winner

	task1	task2	task3	task4	task5	task6
firefly0	-1	6.4	7	-1	-1	-1
firefly1	-1	10	9	-1	-1	-1

Table 5.11: round 5 bids

	task2	task3
firefly0	6.4	-1
firefly1	-1	9

Table 5.12: round 5 winner

	task1	task2	task3	task4	task5	task6
firefly0	-1	-1	2	-1	-1	-1
firefly1	-1	-1	9	-1	-1	-1

Table 5.13: round 6 bids

	task3
firefly0	2
firefly1	9

Table 5.14: round 6 winner

After the last task allocation, the auction process is complete and each drone has its own path of waypoints, as seen in Table 5.15. Then, each drone visits their won waypoints with the correct order. By using the Waypoint Navigator package, described in

Chapter 4.3, we create the path that the drones will follow as depicted in Figure 5.1, Figure 5.2 and Figure 5.3. In the RViz environment, the pink sphere represents the drone and the red squares represent the tasks that must be visited.

firefly0	task6	task4	task1	task2	task3
firefly1	task5				

Table 5.15: Navigation path for each drone.

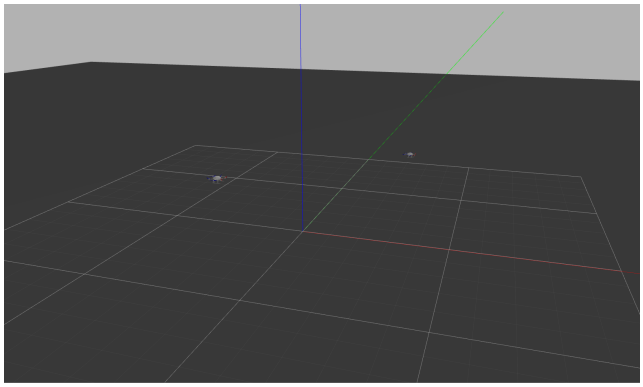

Figure 5.1: View of the drones in the Gazebo environment.

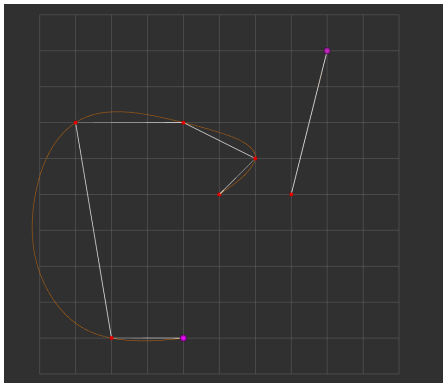

Figure 5.2: View of the drones paths in 2D in the RViz environment.

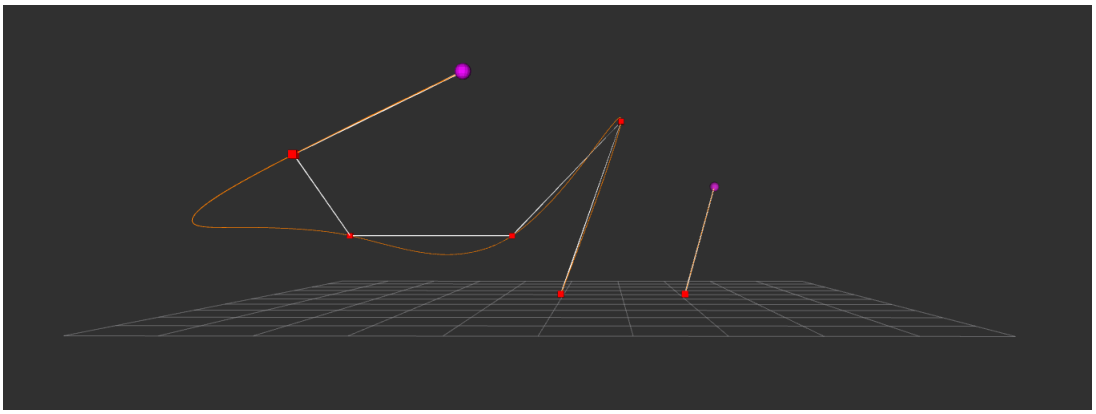

Figure 5.3: View of the drones path in 3D in the RViz environment.

5.2 Scenario 2

This scenario involved five drones (*firefly0...firefly4*) and twenty tasks (*task0..task19*). The tasks are randomly placed in a (40x40x20) metre area. The task are randomly generated in this area and the drones are also randomly placed within the same area, at ground position. The initial positions of the drones are shown in Table 5.16 and the coordinates of the tasks in Table 5.17.

	x	y
firefly0	9	-7
firefly1	-13	10
firefly2	7	-8
firefly3	-15	10
firefly4	20	-4

Table 5.16: Initial positions of the drones.

	x	y	z
task0	-9	-13	10
task1	8	3	6
task2	-3	-16	18
task3	7	-17	3
task4	-17	2	16
task5	20	4	13
task6	-17	14	15
task7	-7	18	5
task8	7	-19	13
task9	-11	16	14
task10	-6	11	20
task11	9	18	12
task12	-2	-7	16
task13	2	18	2
task14	-5	14	8
task15	-11	9	7
task16	-12	6	2
task17	-8	19	1
task18	-3	-14	18
task19	11	16	7

Table 5.17: Initial coordinates of the tasks.

After the auction process is complete and the last task is allocated, each drone has its own path of waypoints, as seen in Table 5.18. We observe that according to their initials positions some drones have more tasks to visit than others, which is expected. Lastly, in Figure 5.4 and in Figure 5.5 we observe the path following of the drones in the Gazebo environment. In Figure 5.6 we visualize each drone's path in the RViz environment, firstly in 2D in Figure 5.7 and then in 3D Figure 5.8.

firefly0	task1	task5											
firefly1	task16	task15	task0	task4	task6	task9	task10	task14	task7	task17	task13	task19	task11
firefly2	task3	task8	task2	task18	task12								
firefly3	-												
firefly4	-												

Table 5.18: Navigation path for each drone.

This scenario was also tested with the MiniMax and MiniAve objectives, so we can have a comparison between them. The navigation paths of each objective are shown in

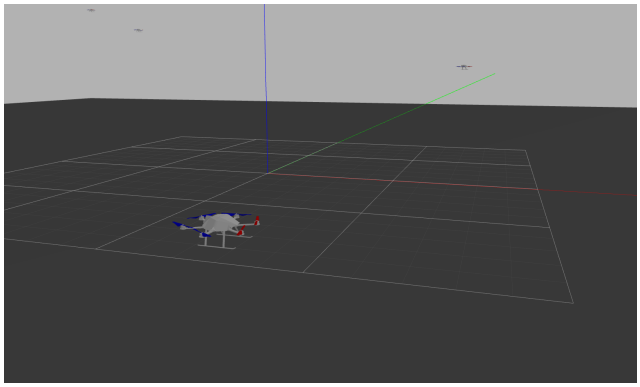

Figure 5.4: View of the drones in the Gazebo environment.

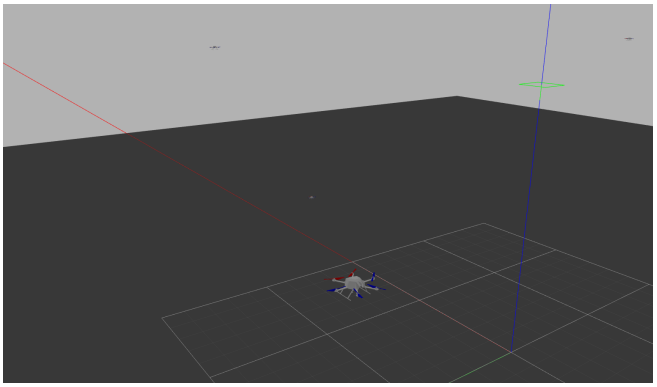

Figure 5.5: Another view of the drones in the Gazebo environment.

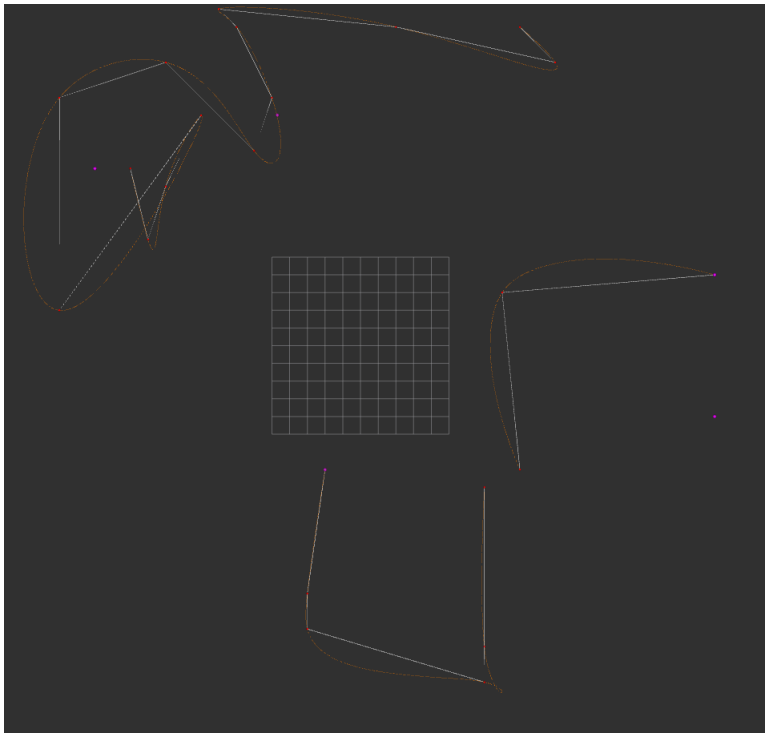

Figure 5.6: View of the drones path in 2D in the RViz environment.

Table 5.19 for MiniMax and in Table 5.20 for MiniAve. As we observe, the results are as expected. With the MiniMax, the tasks are allocated fairly evenly between the drones and with the MiniAve the results are somewhat in between the other two objectives. In Figure 5.9 and in Figure 5.10, we see the MiniMax paths visualization in 2D and in

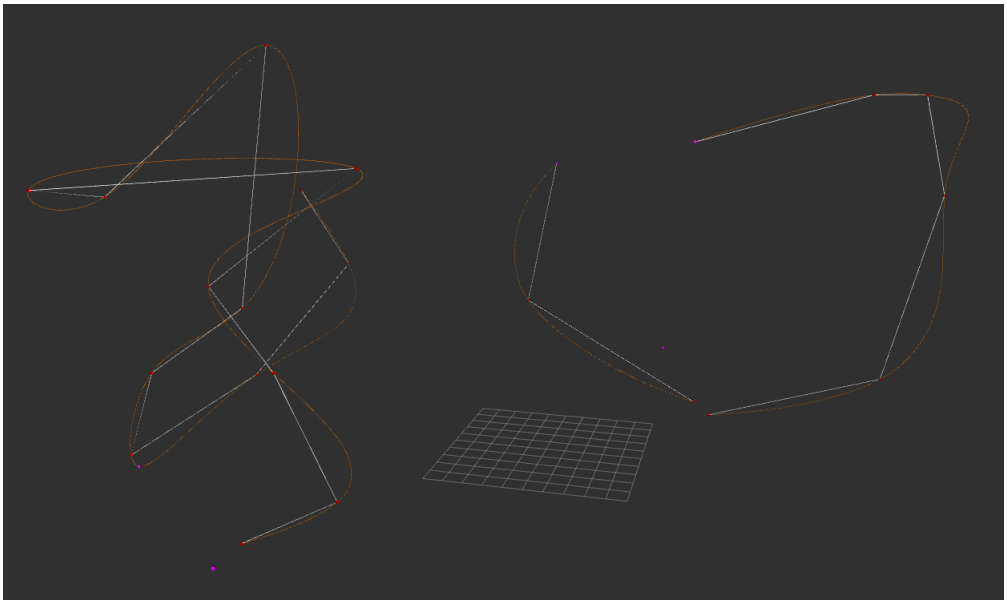

Figure 5.7: View of the drones path in 3D in the RViz environment.

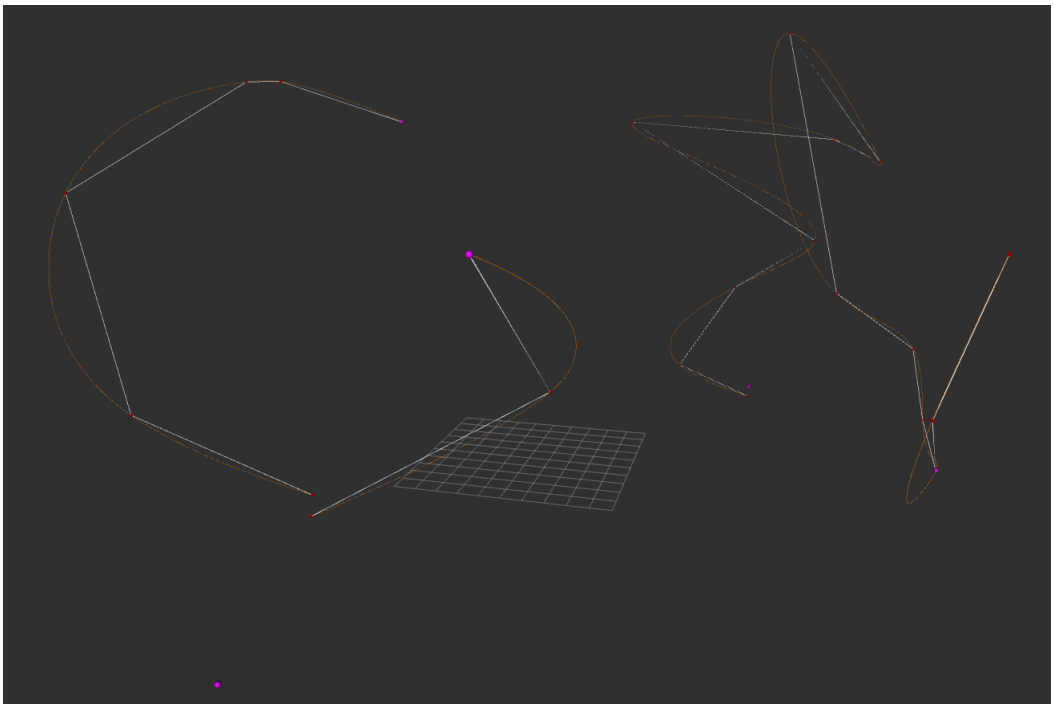

Figure 5.8: View of the drones path in 3D in the RViz environment.

3D respectively in the RViz environment. The MiniAve paths visualization is shown in Figure 5.11 and in Figure 5.12.

firefly0	task1	task19	task11	task10	
firefly1	task16	task14	task7	task17	task13
firefly2	task3	task8	task2	task18	
firefly3	task15	task0	task9	task6	task4
firefly4	task5	task12			

Table 5.19: Navigation path for each drone with the MiniMax objective.

firefly0	task1	task19	task13		
firefly1	task16	task14	task7	task17	
firefly2	task3	task8	task2	task18	task12
firefly3	task15	task0	task9	task6	task4
firefly4	task5	task11	task10		

Table 5.20: Navigation path for each drone with the MiniAve objective.

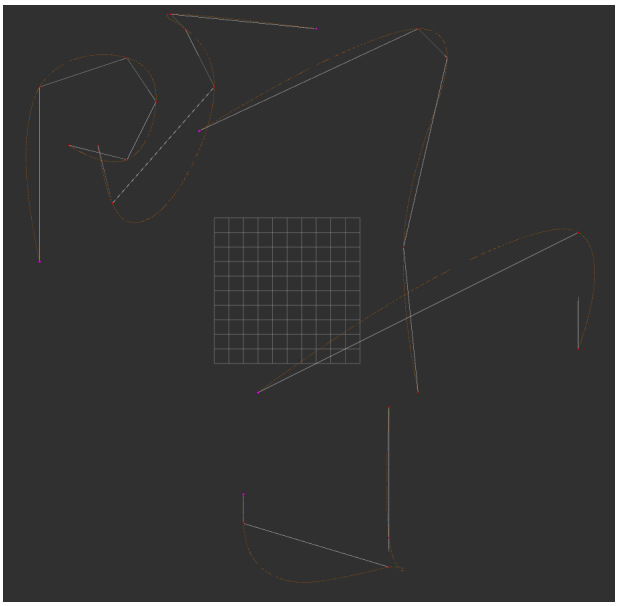

Figure 5.9: View of the drones path in 2D in the RViz environment (MiniMax).

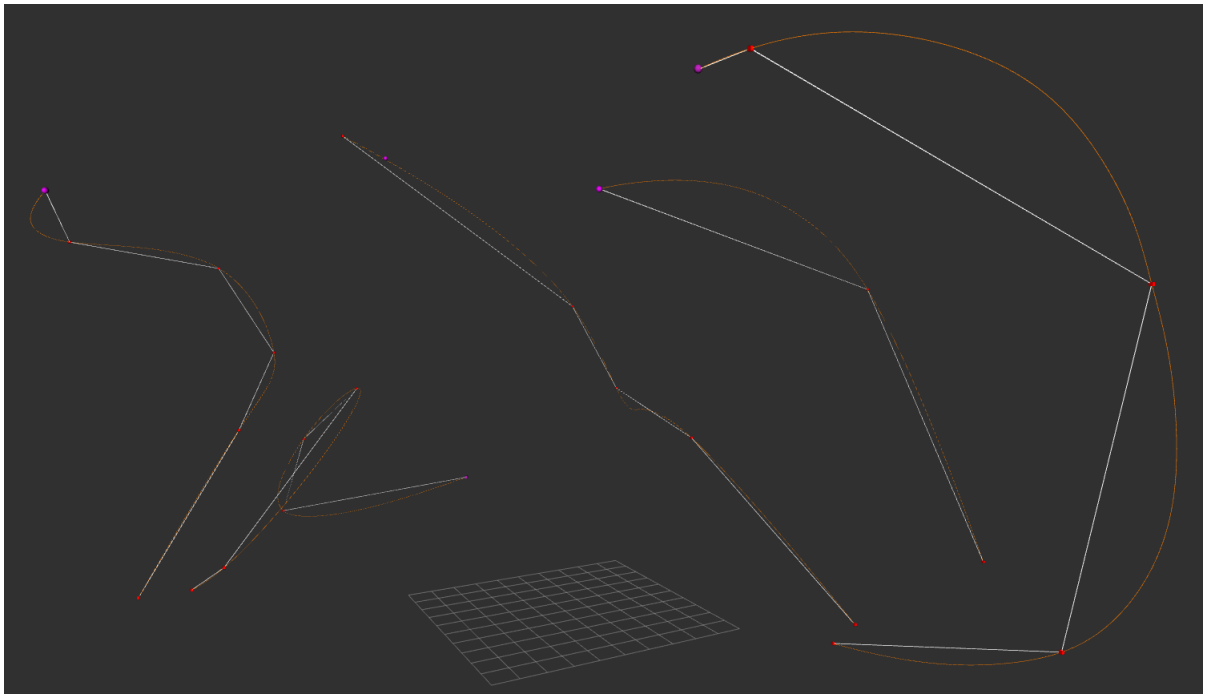

Figure 5.10: View of the drones path in 3D in the RViz environment (MiniMax).

Figure 5.11: View of the drones path in 2D in the RViz environment (MiniAve).

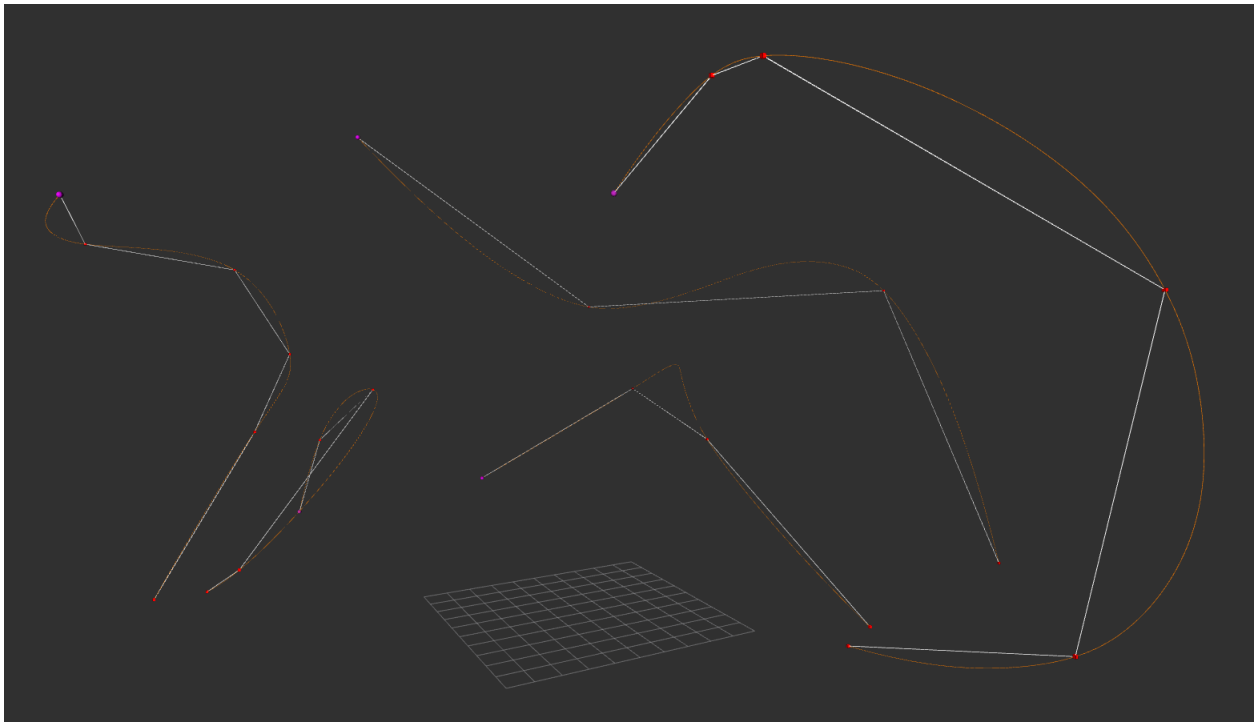

Figure 5.12: View of the drones path in 3D in the RViz environment (MiniAve).

5.3 Scenario 3

This scenario involved twenty drones(*firefly0*...*firefly19*) and fifty tasks (*task0*..*task49*). The tasks are randomly placed in a (75x75x20) metre area. The task are randomly generated in this area and the drones are also randomly placed within a smaller area of (20x20) meters, at ground position. After the auction process is complete we will have the navigation path for each drone.

firefly0	task21														
firefly1	-														
firefly2	-														
firefly3	-														
firefly4	-														
firefly5	task40	task39	task19	task4	task3	task0	task1	task2							
firefly6	-														
firefly7	-														
firefly8	-														
firefly9	task26	task7	task5												
firefly10	-														
firefly11	-														
firefly12	-														
firefly13	task47	task11	task10	task28	task15										
firefly14	-														
firefly15	-														
firefly16	task35	task24	task23	task46	task29	task22	task43	task48	task31	task18	task44	task14	task32	task25	task38
firefly17	task33														
firefly18	task20	task16	task13	task41	task36	task37	task49	task27	task45	task42					
firefly19	task9	task34	task17	task30	task6	task8	task12								

Table 5.21: Navigation Path for each drone.

As we see in Table 5.21, some drones have many tasks (like drones *firefly16* and *firefly18*) and some none (like drones *firefly10* and *firefly15*). This is a normal process, because some drones are closer to some tasks than others and seemingly they are gonna win them. The drones initial positions are shown in Figure 5.13 and the tasks positions in Figure 5.14. We can see the drones and the tasks positions in the same environment in Figure 5.15 and the drones' paths in Figure 5.16. The paths in 2D may seem oddly allocated. Some drones visit tasks that are further away firstly and then tasks that are closer. That happens, because in the 2D diagrams we loose the height (z) information. So, a task may seem further away than others, but in reality it is closer to the drone. One clear example of this case is *firefly19*, where it visits first *task9* in position $(44, 18, 2)$ and then *task17* in $(50, 26, 13)$, although in 2D it seems that the first choice

must be *task17*. In Figure 5.17, in Figure 5.18 and in Figure 5.19 we see the the paths in 3D, so we can have the missing z info displayed clearer.

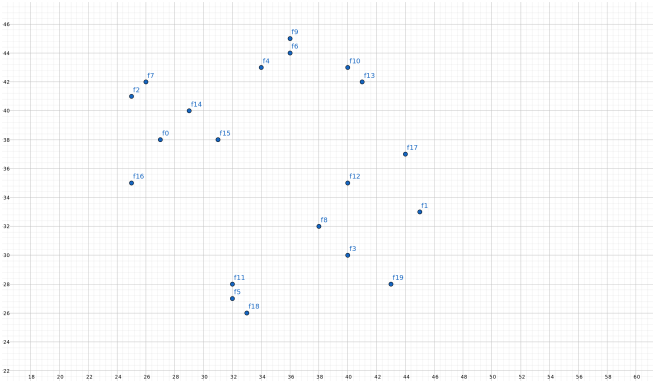

Figure 5.13: Drones positions.

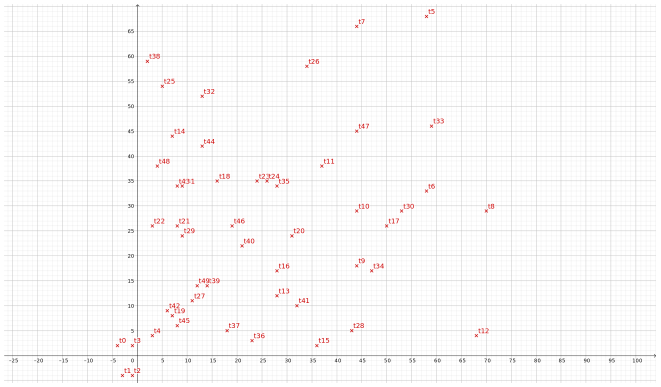

Figure 5.14: Tasks positions.

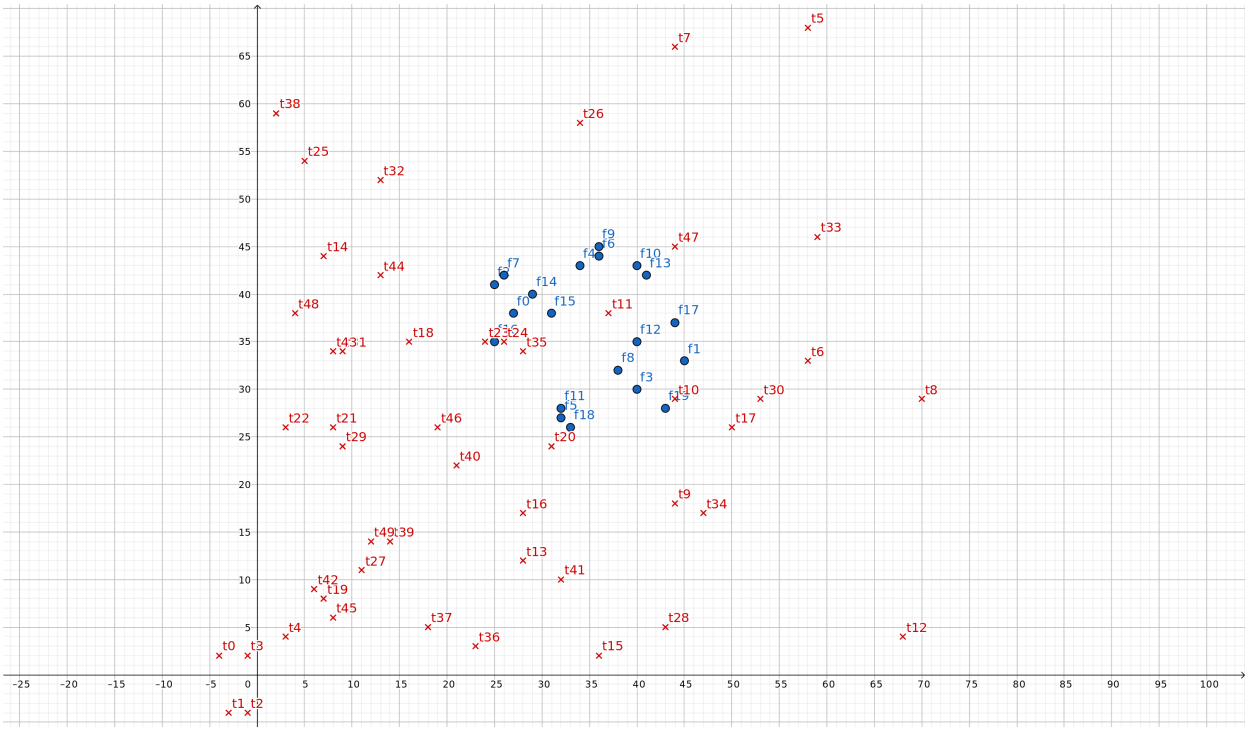

Figure 5.15: Drones and tasks positions.

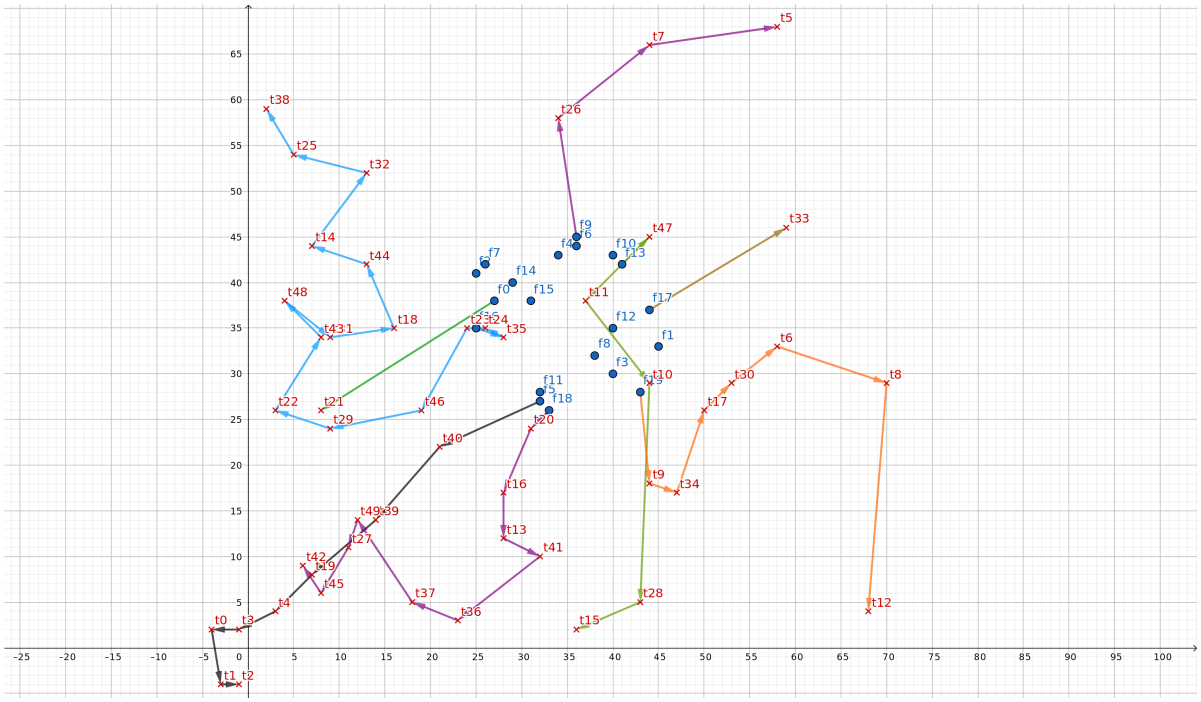

Figure 5.16: View of the drones paths in 2D.

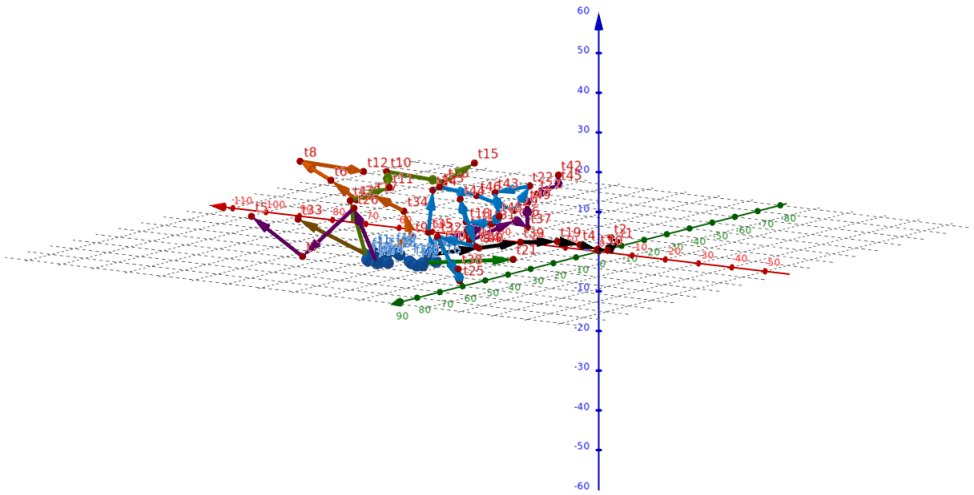

Figure 5.17: View of the drones paths in 3D.

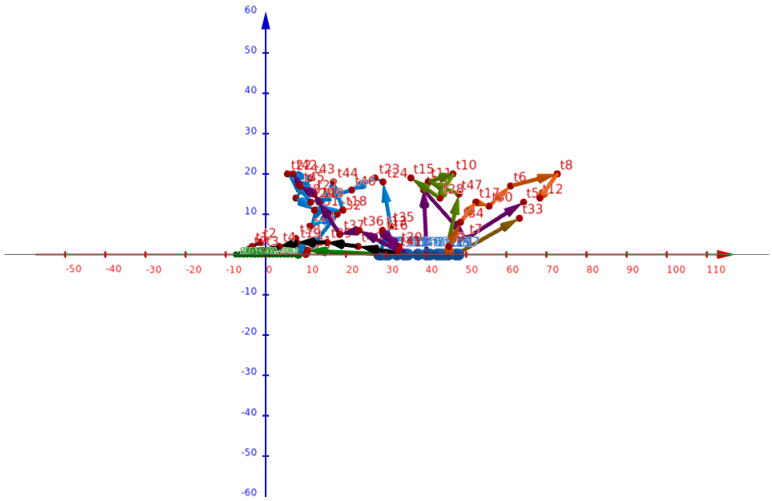

Figure 5.18: View of the drones paths in 3D.

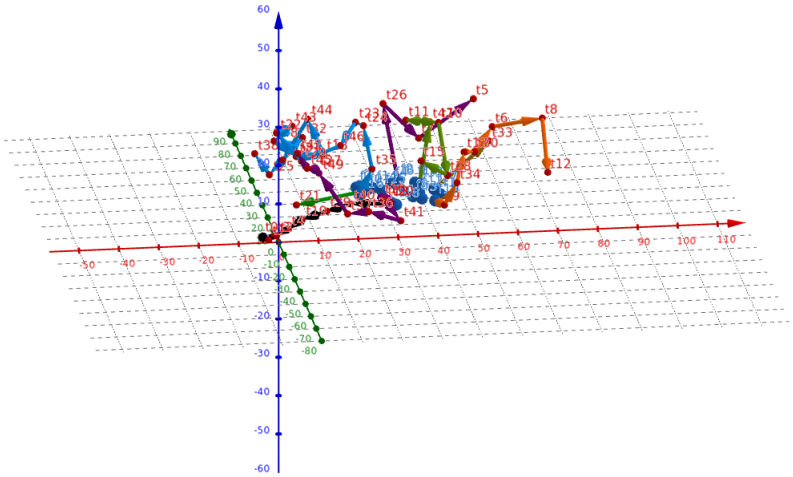

Figure 5.19: View of the drones paths in 3D.

This scenario was also tested with the MiniMax and MiniAve objectives, so we can have a comparison between them. The navigation paths of each objective are shown in Table 5.22 for MiniMax and in Table 5.23 for MiniAve. As we observe, the results are as expected. With the MiniMax, the tasks are allocated fairly evenly between the drones and with the MiniAve the results are somewhat in between the other two objectives. In Figure 5.20 and in Figure 5.21, we see the MiniMax paths visualization in 3D in the RViz environment. The MiniAve paths visualization is shown in Figure 5.22 in 2D and in Figure 5.23 in 3D.

firefly0	task24	task23	task21					
firefly1	task30	task6	task8					
firefly2	task18	task31	task48					
firefly3	task34	task12						
firefly4	task44							
firefly5	task40	task39	task19	task4	task3	task0	task1	task2
firefly6	task7	task5						
firefly7	task32	task25						
firefly8	task10							
firefly9	task26							
firefly10	task33							
firefly11	task41	task49	task27	task42				
firefly12	task11							
firefly13	task47							
firefly14	-							
firefly15	task14	task38						
firefly16	task35	task46	task29	task22	task43			
firefly17	task17							
firefly18	task20	task16	task13	task36	task37	task45		
firefly19	task9	task28	task15					

Table 5.22: Navigation Path for each drone with MiniMax.

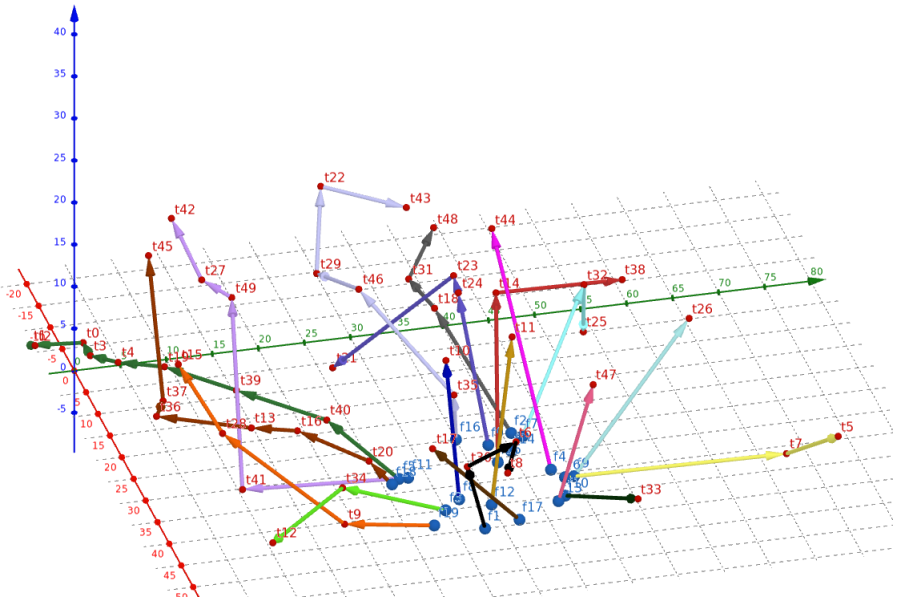

Figure 5.20: View of the drones paths in 3D (MiniMax).

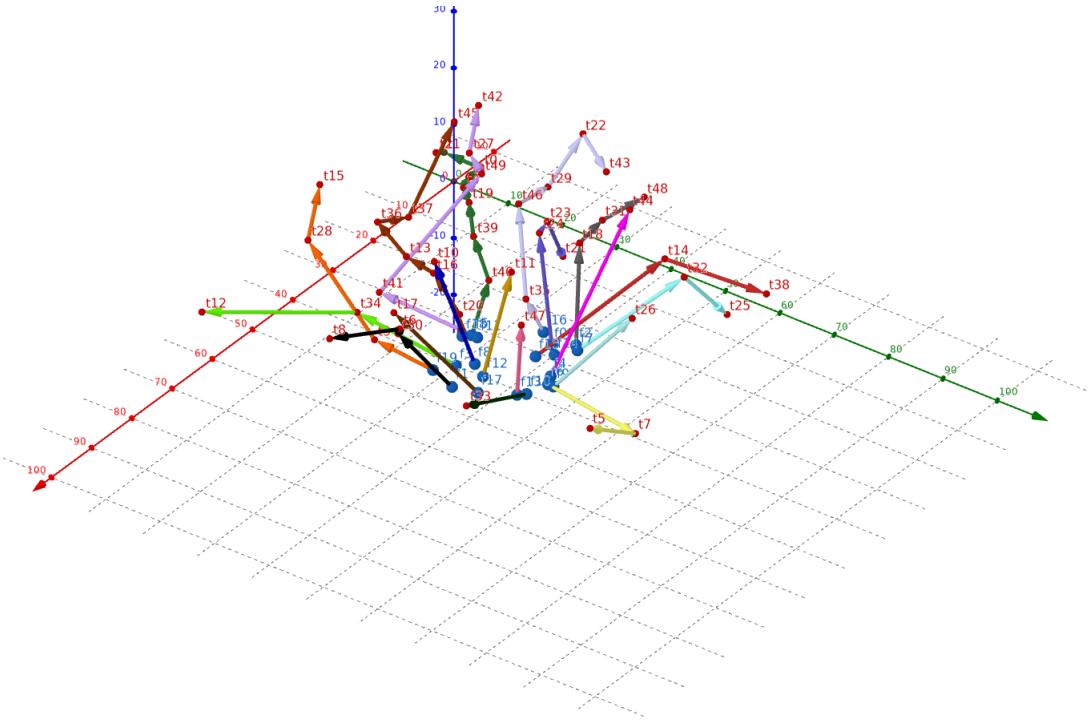

Figure 5.21: View of the drones paths in 3D (MiniMax).

firefly0	task24	task22	task1	
firefly1	task30	task6		
firefly2	task18	task31	task19	
firefly3	task34	task28		
firefly4	task44	task43	task0	
firefly5	task16	task13	task36	task2
firefly6	task7	task5		
firefly7	task32	task14	task38	
firefly8	task10	task12		
firefly9	task26	task25		
firefly10	task33	task3		
firefly11	task40	task39	task49	
firefly12	task11	task42		
firefly13	task47	task45		
firefly14	task23	task48		
firefly15	task46	task29	task27	
firefly16	task35	task21	task4	
firefly17	task17	task8		
firefly18	task20	task41	task37	
firefly19	task9	task15		

Table 5.23: Navigation Path for each drone with MiniAve.

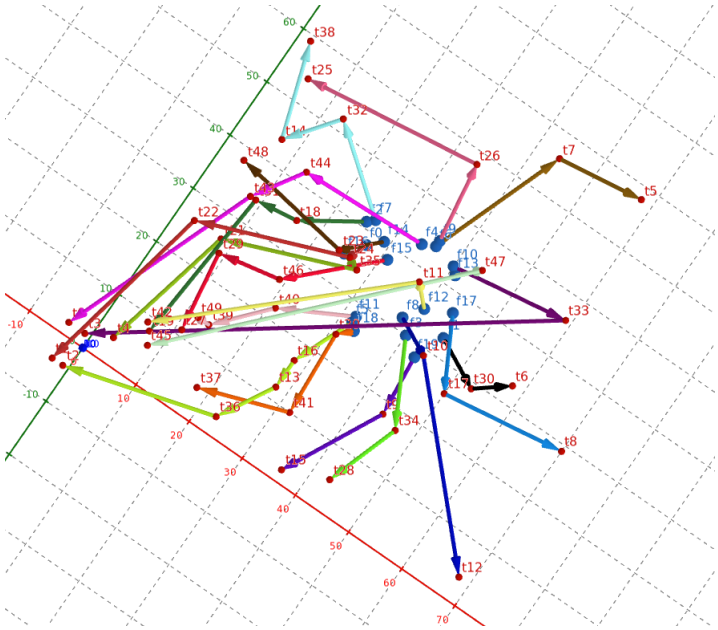

Figure 5.22: View of the drones paths in 2D (MiniAve).

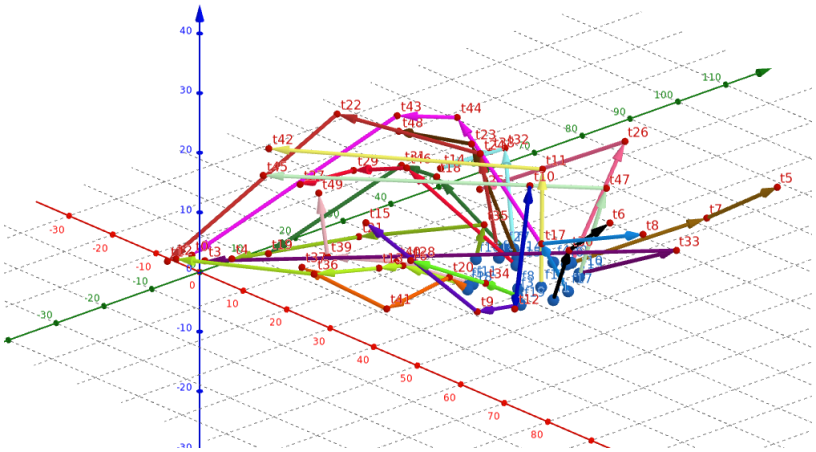

Figure 5.23: View of the drones paths in 3D (MiniAve).

Chapter 6

Conclusion

6.1 Discussion

This thesis describes a multi-robot collaboration approach, in witch a team of drones cooperate to solve a basic Search-and-Rescue scenario. Assuming that we have a team of drones in an environment and a set of known tasks-locations that must be visited, we implemented a *distributed sequential auction* to distribute the tasks. Foremost, each drone finds its position in relation to the environment, by using the tf library. Afterwards, in every round, each drone communicate with each other to determine the winner of the task with the best bid. The bidding value is calculated as the distance between the robot and the task by using a set of algorithms, as we saw in Chapter 4.2.4. The winner bid is by default the lowest, which is found by using the Lowest Bid Algorithm **i.**. The user has the option to choose the winning bid with the Least Contested Bid Algorithm **ii.** by giving the correct input. Finally, after all the tasks has been allocated, a waypoint navigator package is used to command the drone to follow the created path. Each system part is presented in detail and the entire system is ready for real-world experiments. The entire project has been implemented within the Robot Operating System (ROS), as an open source package.

6.2 Future Work

6.2.1 Advanced Bidding Model

In our approach, we used a bidding model, where the robot's bid is based on its distance from the task-location. Also, the drones have not any sensors with them, like high quality cameras, 3D LiDAR, etc. First of all, in real life situations each robot will have sensors and the sensors will not be necessary the same in all the robots. One robot may have a sensor for radiation detection and another may have a thermal camera. Also, the percentage of the battery of each drone could count as an extra "sensor", because they can fly as long as they have enough battery. Through, those sensors additions, the bidding model must be reevaluated and improved in many areas. Weights to the bidding process must be introduced, each one depending to the different sensors and the battery level that the drones will have. This will allow for a better simulation based on real-life search-and-rescue scenarios and also for better results in real-life situations.

6.2.2 Real-World Experiments

In this thesis, out of eight combinations of the *Optimal Assignment Problem* (OPA) we focused only in the simplest case, which is the *ST-SR-IA* (*Single-task robot - Single-robot task - Instantaneous assignment*). In real-life search-and-rescue missions all the combinations exist. Tasks that demand more than one robot and robots that can execute multiple tasks synchronously are some of the cases that firstly must be studied and simulated in different scenarios, so they can implemented in real situations. Our scenario must also, be tested in real circumstances, so along with the others cases we can minimize the amount of time needed, while saving the maximum number of survivors.

6.2.3 Additional Path-Finding Algorithms

Ultimately, this proposal concentrated uniquely in three out of several algorithms that exist for finding a minimum path cost. A path can also be assessed without a minimum cost by many more algorithms, in various ways and speeds. Additionally, in our

case the three algorithms that were utilized can be additionally refined, with the goal that they will become progressively more effective and quick. In this way, through different experiments a superior version of the utilized algorithms or even a brand new algorithm could be implemented to calculate better paths that the robots will follow, so as to to get to the objectives in the minimum amount of time.

References

[1] Boost asio. https://www.boost.org/doc/libs/1_66_0/doc/html/boost_asio.html. 8

[2] Bullet. https://www.pybullet.org/wordpress/. 7

[3] Gazebo components. http://gazebosim.org/tutorials?tut=components&cat=get_started. 9

[4] Dart (dynamic animation and robotics toolkit). http://www.dartsim.github.io/. 8

[5] Wallace Pereira Neves dos Reis and Guilherme Sousa Bastos. Implementing and simulating an alliance-based multi-robot task allocation architecture using ros. In Fernando Santos Osório and Rogério Sales Gonçalves, editors, *Robotics*, pages 210–227, Cham, 2016. Springer International Publishing. 19

[6] T. Foote. tf: The transform library. In *2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA)*, pages 1–6, April 2013. 23

[7] freeglut. http://www.freeglut.sourceforge.net/. 8

[8] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. Robot operating system (ros). *Studies Comp.Intelligence*, 1(625), 2016. 31

[9] David Gale. *The Theory of Linear Economic Models*. Number 9780226278841 in University of Chicago Press Economics Books. University of Chicago Press, April 1989. 25

[10] B. P. Gerkey and M. J. Mataric. Sold!: auction methods for multirobot coordination. *IEEE Transactions on Robotics and Automation*, 18(5):758–768, Oct 2002. 18

[11] Brian P. Gerkey and Maja J. Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *I. J. Robotics Res.*, 23:939–954, 2004. 25

[12] Google protobuf. https://www.github.com/protocolbuffers/protobuf. 8

[13] María Guillén, Sergio García, Rafael Barea, Luis Bergasa, Eduardo J. Molinos, Roberto Arroyo, Eduardo Romera, and Samuel Pardo. A multi-sensorial simultaneous localization and mapping (slam) system for low-cost micro aerial vehicles in gps-denied environments. *Sensors (Switzerland)*, 17, 04 2017. 13

[14] Serena Ivaldi, Vincent Padois, and Francesco Nori. Tools for dynamics simulation of robots: a survey based on user feedback. *CoRR*, abs/1402.7050, 2014. 7

[15] HyeongRyeol Kam, Sung-Ho Lee, Taejung Park, and Chang-Hun Kim. Rviz: a toolkit for real domain data visualization. *Telecommunication Systems*, 60:1–9, 10 2015. 9

[16] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, Sep. 2004. 7

[17] Sven Koenig, Craig Tovey, Michail Lagoudakis, Evangelos Markakis, David Kempe, Pinar Keskinocak, Anton J. Kleywegt, Adam Meyerson, and Sonal Jain. The power of sequential single-item auctions for agent coordination. In *Proceedings of*, 01 2006. 18

[18] Anis Koubâa and J.Ramiro Martínez-de Dios, editors. *Multi-robot Task Allocation: A Review of the State-of-the-Art*, pages 31–51. Springer International Publishing, Cham, 2015. 18

[19] Michail Lagoudakis, Evangelos Markakis, David Kempe, Pinar Keskinocak, Anton J. Kleywegt, Sven Koenig, Craig Tovey, Adam Meyerson, and Sonal Jain. Auction-based multi-robot routing. In *Proceedings of Robotics: Science and Systems*, pages 343–350, 06 2005. 18, 28

[20] Jairo R. Montoya-Torres, Julián López Franco, Santiago Nieto Isaza, Heriberto Felizzola Jiménez, and Nilson Herazo-Padilla. A literature review on the vehicle routing problem with multiple depots. *Computers and Industrial Engineering*, 79:115 – 129, 2015. 18

[21] T. Moore and D. Stouch. A generalized extended kalman filter implementation for the robot operating system. In *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*. Springer, July 2014. 11

[22] Ogre. https://www.ogre3d.org/. 8

[23] Open dynamics engine (ode). http://www.ode.org/. 7

[24] Opengl. https://www.opengl.org/. 8

[25] L. E. Parker. Alliance: an architecture for fault tolerant, cooperative control of heterogeneous mobile robots. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'94)*, volume 2, pages 776–783 vol.2, Sep. 1994. 18

[26] Robot operating system (ros). http://www.ros.org. 5

[27] Sdf (simulation description format). http://sdformat.org/spec. 9

[28] Simbody. https://www.simtk.org/projects/simbody. 7

[29] Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12):1104–1113, Dec 1980. 18

[30] Nick Sullivan, Steven Grainger, and Ben Cazzolato. Sequential single-item auction improvements for heterogeneous multi-robot routing. *Robotics and Autonomous Systems*, 115:130 – 142, 2019. 30

[31] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust monte carlo localization for mobile robots. *Artificial Intelligence*, 128(1):99 – 141, 2001. 12