UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE AUTOMAÇÃO E SISTEMAS

Alexander Silva Barbosa

Hierarchization and contextual information management for performing collaborative tasks in land mobile robots

Florianópolis - SC
2022

Alexander Silva Barbosa

Hierarchization and contextual information management for performing collaborative tasks in land mobile robots

Dissertação submetida ao Programa de Pós-Graduação em Engenharia de Automação e Sistemas da Universidade Federal de Santa Catarina para a obtenção do título de mestre em Engenharia de Automação e Sistemas.
Supervisor:: Prof. Edson Roberto de Pieri, Dr.
Coorientadora: Patricia Della Méa Plentz, Dra.

Florianópolis - SC
2022

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Barbosa, Alexander Silva
Hierarchization and contextual information management for performing collaborative tasks in land mobile robots / Alexander Silva Barbosa ; orientador, Edson Roberto de Pieri, coorientadora, Patricia Della Méa Plentz, 2022.
56 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia de Automação e Sistemas, Florianópolis, 2022.

Inclui referências.

1. Engenharia de Automação e Sistemas. 2. Sistemas multi robôs. 3. Árvore de comportamento. 4. Aprendizado de máquina. 5. Alocação de tarefas multi-robôs. I. Pieri, Edson Roberto de. II. Plentz, Patricia Della Méa. III. Universidade Federal de Santa Catarina. Programa de Pós Graduação em Engenharia de Automação e Sistemas. IV. Título.

Alexander Silva Barbosa

Hierarchization and contextual information management for performing collaborative tasks in land mobile robots

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Jônata Tiska Carvalho, Dr.
INE/UFSC

Prof. Marcelo Ricardo Stemmer, Dr.
DAS/UFSC

Prof. Ricardo Alexandre Reinaldo de Moraes, Dr.
CIN/UFSC

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de mestre em Engenharia de Automação e Sistemas.

ICP-Edu
Documento assinado digitalmente
Julio Elias Normey Rico
Data: 27/07/2022 20:00:26-0300
CPF: 762.840.859-15
Verifique as assinaturas em https://v.ufsc.br

Coordenação do Programa de
Pós-Graduação

ICP-Edu
Documento assinado digitalmente
Edson Roberto de Pieri
Data: 27/07/2022 15:59:52-0300
CPF: 002.042.498-13
Verifique as assinaturas em https://v.ufsc.br

Prof. Edson Roberto de Pieri, Dr.
Supervisor:

GOV.BR
Documento assinado digitalmente
PATRICIA DELLA MEA PLENTZ
Data: 27/07/2022 17:50:32-0300
CPF: 959.243.700-97
Verifique as assinaturas em https://v.ufsc.br

Patricia Della Méa Plentz, Dra.
Coorientadora

Florianópolis - SC, 2022.

Dedico este trabalho inicialmente aos meus pais, Edneia e Adalmo, por sempre me apoiarem a seguir meus objetivos e sempre serem minhas maiores inspirações. Também aos meus irmãos, Igor e Jéssica, por estarem sempre ao meu lado, mesmo que distantes fisicamente. Dedico também ao Romulo por me apoiar e estar ao meu lado durante todo o processo deste trabalho, nos bons e nos maus momentos. Agradeço aos meus queridos orientadores, Edson e Patrícia, sem eles este trabalho não teria sido possível, ambos foram excelentes do início ao fim. Agradeço também aos meus professores de graduação, especialmente Leonardo Olivi, Ana Sophia, Exuperry e Elias, eles foram minhas motivações. Este trabalho é também dedicado aos meus colegas, em especial ao Afonso, Geovana e Willian, foram essenciais.

ACKNOWLEDGEMENTS

The authors are grateful for the financial support granted by CNPq and CAPES to this research at the Graduate Program in Automation and Systems Engineering of the Federal University of Santa Catarina as well as the CAPES/PRINT - Call no. 41/2017 Senior Visiting Professor at Biomimetics and Intelligent Systems Group - BISG, University of Oulu, Finland.

RESUMO

Um Sistema Multi-robô (Multi-Robot System - MRS) é um grupo de vários robôs que se comunicam e cooperam para realizar algum comportamento coletivo. Nesse contexto, a Alocação de Tarefas Multi-Robô (Multi-robot Task Allocation - MRTA) é um dos problemas de um sistema multi-robô que consiste em atribuir tarefas aos robôs de forma eficiente. Uma Árvore de comportamento (Behavior Tree - BT) é uma estrutura hierárquica composta por nós, e é responsável pela tomada de decisão em um agente artificial inteligente. Este trabalho apresenta um novo framework para MRTA, consistindo em uma árvore de comportamento, um sistema de comunicação descentralizado e um sistema de aprendizado de máquina. A árvore de comportamento é responsável por coordenar a alocação de tarefas em cada robô ao mesmo tempo que o mantém seguro, permitindo a negociação e transferência da tarefa considerando as restrições do sistema em tempo real. O sistema foi validado em ambiente de simulação, considerando diversos cenários variando o número de robôs e de tarefas.

Palavras-chave: Robótica. Robôs móveis. Árvore de comportamento. Tomada de decisão hierárquica. Inteligência artificial. Sistemas multi-robôs. Alocação de tarefas multi-robô. Tomada de decisões. Árvores de decisão. Multiplos robôs. Redes neurais artificiais. Redes neurais profundas. Aprendizado de máquina.

ABSTRACT

A Multi-robot System (MRS) is a group of multiple robots communicating and cooperating to perform some collective behavior. In this context, Multi-robot Task Allocation (MRTA) is one of the problems of a multi-robot system consisting of assigning tasks to robots in an efficient way. A Behavior Tree (BT) is a hierarchical structure composed of nodes and responsible for the decision-making in an intelligent artificial agent. This work proposes a novel framework for MRTA, consisting of a Behavior Tree, a decentralized communication system and a machine learning system. The BT is responsible for coordinating the task allocation on each robot while keeping it safe, allowing the task negotiation and transference considering the system's constraints in real-time. The system was validated in a simulated environment, considering many scenarios varying the number of robots and tasks.

Keywords: Robotics. Mobile robots. Behavior tree. Hierarchical decision making. Artificial intelligence. Multi-robot systems. Multi-robot task allocation. Decision making. Decision trees. Multiple robots. Artificial neural networks. Deep neural networks. Machine learning.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AND	Logical operator 'AND'
ANN	Artificial Neural Network
BT	Behavior Tree
CID	Context Information Database
FSM	Finite State Machine
GP	Genetic Programming
MAE	Mean Absolute Error
ML	Machine Learning
MRS	Multi-robot System
MRTA	Multi-robot Task Allocation
NN	Neural Network
NPC	Non-Player Character
OR	Logical operator 'OR'

LIST OF SYMBOLS

xI	Initial position in meters (x)
yI	Initial position in meters (y)
xF	Target position in meters (x)
yF	Target position in meters (y)
np	Number of path segments
d	Length of the path in meters (real estimated distance)
B	Robot battery (%)
V	Robot speed (m/s)
TR	Task requirements
ΔB	Required battery (%)
ΔT	Required time (minutes)
FB	Final battery level (%)
ΔB_i	Required battery for the initial virtual task (%)
ΔT_i	Required time for the initial virtual task (minutes)
FB_i	Final Robot battery after completing the first virtual task(%)
ΔB_t	Required battery for the real task (%)
ΔT_t	Required time for the real task (minutes)
B_i	Final Robot battery after completing the first virtual task(%)
FB_t	Final Robot battery after completing the real task(%)
ΔB_f	Required battery for the final virtual task (%)
ΔT_f	Required time for the final virtual task (minutes)
B_t	Final Robot battery after completing the real task(%)
B_f	Final Robot battery after reaching the station (%)
BID	Bid for the robots auction
D	Distance from initial position to target position in meters
TNT	Total number of tasks
NR	Number of robots
NT	Number of tasks per robot

CONTENTS

1 INTRODUCTION

Robotics has succeed in the industry where predominantly ground manipulator robots are used, which perform precision tasks in repetitive applications and are limited by an operational space defined by the robot's physical extensions and restrictions imposed on its movement. Among the tasks most commonly performed by these systems are painting, welding (actively applied in the automobile and electronics industry), manipulation of objects in biologically controlled environments, and maintenance services in unhealthy environments (GARCIA et al., 2007). Mobile robotics has evolved and should become a solution for industry and several other sectors of society, using its ability to move, quickly adapt to little-known environments, and constantly change the physical configuration. The main current tasks using mobile robots are the transport of parts, surveillance, maintenance, the inspection of pipelines and in hostile environments (GARCIA et al., 2007).

For a mobile robot to perform intelligent navigation, it must, as a human, perceive the environment and use the information to adapt its behavior to the current scenario, making it as independent of intervention as possible (DE LUCCA SIQUEIRA; DELLA MEA PLENTZ; DE PIERI, 2016). The autonomy of a mobile robot can be improved using information about the environment, which can be provided by sensors installed in the robot or by information shared by several robots collaborating in the execution of tasks and the information provided by sensors in the environment. This research field involving several robots is known as a swarm of robots or cooperative robots and have several research topics that need formalization and scientific contributions. Several works deal with the problem, and the subject has deserved the attention of several researchers, with different results presented in congresses, symposia, and specialized magazines. Interest in multi-robot systems research is increasing day by day as multiple robots can perform tasks that are impossible or too costly for a single robot (CAO; FUKUNAGA; KAHNG, 1997).

The cooperation of multiple robots can involve many moving in environments with static and dynamic obstacles, which can be problematic. One way to analyze this problem is to hierarchize the tasks and consider real-time restrictions. The information that is made available for a robot to perform a task free of an obstacle can also be hierarchical, establishing priorities and formalizing activities that should prioritize others. Such tasks can be battery recharging, task redefinition between different robots, etc., which are cases in which must make real-time decisions.

According to (KHAMIS, Alaa; HUSSEIN, Ahmed; ELMOGY, 2015a), a Multi-robot System (MRS) is a group of robots designed to perform some collective behavior. An MRS can solve problems of significant complexity allowing the increase of performance and reliability. One difficulty of MRS is known as Multi-robot Task Allocation

(MRTA), which is the act of assigning a group of robots to a group of tasks considering a set of constraints to optimize the overall system performance (KORSAH; STENTZ; DIAS, 2013).

Behavior trees (BTs) are a type of Hybrid Dynamical Systems that has emerged as a tool to create intelligent agents in computer games and later spread to other areas, especially in robotics. According to iovino2020survey, btmod6942752 and Ogren588170, BTs can be seen as a recent addition to a long research because of increasing modularity, robustness, and the safety of robot control software. Due to its hierarchical nature, a behavior tree offers a simple and efficient way of creating reactive agents, but the BT is a static structure. It cannot learn or adapt its behavior decisions directly. Different from a BT, an Artificial Neural Network (ANN) is a structure designed to model the way the brain performs a task or a function of interest, which allows it to learn over time using data from experience (HAYKIN, S. S., 2009).

This work seeks to contribute to the Multi-robot task allocation area using the modularity of behavior trees with the system constraints as a key tool, as the behavior tree allows the robot to be naturally reactive. It tries to solve the problem of the BT being a static structure by using an artificial neural network to allow that some specific nodes of the BT improve its decision based on the robot experience and specificities. The document is organized as follows: Chapter 2 presents an overview of the main areas required for this work. In Chapter 3 it is presented tool developed to analyse the behavior trees, the proposed BT architecture and the CID module. The results of simulations are discussed in Chapter 4 and the conclusions are provided in Chapter 5.

1.1 OBJECTIVES

This section formalizes the general and the specific objectives of this dissertation project.

1.1.1 General objective

This dissertation aims to formalize the problem of cooperation of land mobile robots using behavior trees with local and contextual information and incorporating real-time constraints, so that it uses this static tool to work as a dynamic structure. It also aims to conduct experimental robot testing in simulation by performing target achievement tasks with obstacle avoidance in environments with static and dynamic obstacles.

1.1.2 Specific objectives

1. Develop a tool to create, simulate and analyze behavior trees;

2. Formalization of the land mobile robot cooperation problem using behavior trees and hierarchical methods;
3. Definition of an information exchange architecture between robots;
4. Definition of a robot and real-time constraints for task planning, incorporating tasks that require sequencing and/or prioritization;
5. Adaptation of the behavior tree of item 2 to integrate the constraints of item 4;
6. Implementation of a path planning algorithm that considers the real-time constraints and avoidance of static and dynamic obstacles;
7. Implementation of a robot selection algorithm based on items 2 and 4.
8. Implementation of a machine learning model that can learn and predict the time for a given task and the robot battery level.

1.1.3 Research delimitation

Some conditions are taken into account to delimit the research, being them:

1. The robots will use well-known computer vision-based algorithms for navigating in a way that computer vision tasks are not in this research;
2. Robots can communicate globally and locally;
3. There are fixed stations in the environment where robots can recharge and from where they receive primary tasks;
4. A station delegating a task can communicate with at least one robot.
5. There is a global coordinate system used by all robots so that the positions of targets detected by different robots can be translated into a single coordinate system;
6. The size of an environment is bounded by the communication range among robots and the fixed stations;
7. The robots have access to the static environment map, so it is not in the scope of this research the construction of the maps.
8. The system will use well-known frameworks for machine learning so that implementing of the underlying system for ML is not in the scope of this research.

1.1.4 Scientific contribution

1. Obtaining a tool for behavior trees design and evaluation;
2. Obtaining of a behavior tree model of the mobile robot cooperation problem;

3. Adaptation of the primary model to accomplish the system constraints ;
4. Adaptation of a path planning algorithm to consider the model and constraints of item 3;
5. A robot selection algorithm based on system constraints.

2 BACKGROUND

This chapter presents the background areas used to compose this work. Each area area is described so that the reader can understand what is needed to fully understand the whole research.

In (NEHMZOW, 2006), mobile robots are defined as embedded, situated agents. It is embedded because it interacts with its environment through its actions and situated because its actions affect future states. The set of actions that a robot can perform in the environment depends on which tools it has, like wheels, legs, arms, grasps, and many others. The mobile robot itself and any tool it can use is powered by a battery, which creates a restriction on how long the robot can work before recharging. A mobile robot also has other restrictions like maximum speed and navigable terrain and battery restriction. When multiple robots work together, other aspects need to be considered while modeling the system, like the robot selection for a task and collision-free navigation.

A Behavior Tree is a mathematical model of plan execution in an autonomous agent such as a virtual entity in a computer game or a robot (COLLEDANCHISE, Michele; ÖGREN, 2017). As BTs are modular by design, they are a very efficient way of creating reactive and modular systems, and this has led to its spread to artificial intelligence and robotics (IOVINO et al., 2020).

This research aims to apply behavior trees to solve the task allocation problem in a multi-robot context considering the robot and the task constraints. Together with the behavior tree, a new module called CID (Context Information Database) is introduced (Context information is considered as all information related to the sensing of the robot, its state or any other meaningful information for a task). This module stores the robot context information and using this at runtime to predict the robot capacity to accept or continue executing a given task.

2.1 MULTI-ROBOT SYSTEMS

As mentioned in (ARAI; PAGELLO; PARKER, L. E., 2002), several robotics application areas can benefit from the use of multi-robot systems as it can deal with tasks that are difficult or even impossible for a single robot. (ARAI; PAGELLO; PARKER, L. E., 2002) identified seven main topic areas of MRS:

- Biological Inspirations;
- Communication;
- Architectures, task allocation, and control;
- Localization, mapping, and exploration;
- Object transport and manipulation;

- Motion coordination;
- Reconfigurable robots.

In an MRS, robots must be able to work on different parts of a higher-level goal at the same time and eventually in the same workspace. In this context, the MRS allocates tasks to each robot, providing some communication between the robots, coordinating a group of robots, and also ensuring that one robot does not interfere with each other (PARKER, Lynne E., 2007).

Another essential characteristic of MRS is that it makes possible to achieve two desired features in robotics: adaptivity and fault-tolerance (IOCCHI; NARDI; SALERNO, 2001). These two features are possible in an MRS because it can change itself dynamically with the environment. A fault in one or some robots does not have the same impact on the overall system as in a single robot system.

2.2 BEHAVIOR TREES

Modularity is crucial for code reuse and incremental design of features. The control structures of Non-Player Characters (NPCs) in games were often formulated as a Finite State Machine (FSM). Just like Petri Nets is an alternative to FSMs allowing the design of concurrent systems, the Behavior Trees is an alternative view of FSMs that supports the design of modular systems (COLLEDANCHISE, Michele; ÖGREN, 2017).

A BT is formally a directed rooted tree composed of control flow nodes, decorators and execution nodes, where all leaf nodes are execution nodes, a decorator applies a function to a node, and control flow nodes are all the other nodes. A common terminology of parent and child is used for each connected node, where the root is a node without parents. The number of children of all the control flow nodes varies from one to many children. (COLLEDANCHISE, Michele; ÖGREN, 2017).

When executing, the root node generates a signal called a tick. The tick has a frequency defined by the application and is propagated to the root child. According to the rules of the nodes, the tick is forwarded to its children. A node executes if and only if it receives a tick signal. Each node can return three responses to its parent: Success, Failure or Running.

Following the success in the game industry, BTs have spread to other areas, as robotics being one of them. A simple BT performing a Pick and Place task can be seen in Figures 1 and 2, where Figure 1 shows the high-level task and Figure 2 shows the task at a lower level.

There are seven basic types of nodes in a BT: Root, Selector, Sequence, Parallel, Inverter, Condition, and Action.

Figure 1 – A high level BT carrying out a task consisting of first finding, then picking and finally placing a ball.

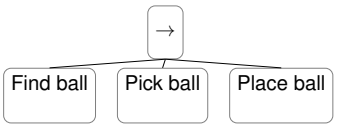
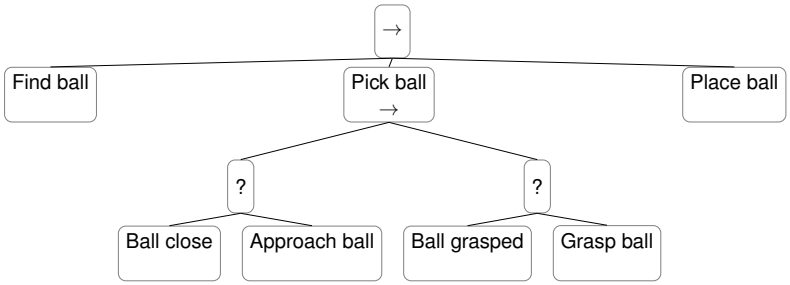


Figure 2 – The action 'Pick Ball' from the BT in Figure 1 is expanded into a sub-BT. The Ball is approached until it is considered close, and then the action 'grasp' is executed until the ball is securely grasped.



The node "Root", represented by the symbol $\emptyset$, is responsible to periodically send a tick to its only child in a way that the only rule to tick is forwarded is that the child exists. The "Selector" (?) and "Sequence" ($\rightarrow$) nodes act like the logical ports *AND* and *OR*. The node "Selector" sends the tick to its first child, from left to right, returning success if the child succeeds or forwarding to the next if it fails. If all children fail, then the selector also fails, acting like an *OR* port. The node "Sequence" acts as an *AND* port, returning success only if all children succeed. The execution order is also from left to right, and when a child fails, the sequence instantly fails too.

The "Parallel" node ($\Rightarrow$), as the name suggests, executes all of its children in parallel, returning a value based on a policy. The most common approaches are to return immediately when a child fails or succeeds and to return the opposite when all children meet the policy condition. "Inverter" (!) is a node that can have only one child, and its status is the opposite of its child. This node only returns the same of its child if the status is "running".

The node "Condition", represented by an ellipse, returns only success or failure but never returns the running status. It checks a condition, something like True or False, from common programming languages. Last but not least, the "Action" node (an empty rectangle) executes a task and return the execution status.

One of the key features of BTs is modularity, allowing specific functionalities of a system to be implemented as an individual BT that later can be coupled to other higher-level BT representing a higher level of the system. That coupling can be done until the whole is described as a BT (GONZALEZ-PEREZ; HENDERSON-SELLERS; DROMEY, 2005). This modular modeling aspect makes evident another feature of BTs:

the easy visualization of the whole system, even when the complete behavior tree is complex.

BTs are traditionally designed manually for a specific application. Still, they can also be evolved and optimized using genetic programming (GP) as it is formally a way to represent a computer program. When using this method, each individual in the population is a complete BT, and the results may be better than a handmade BT (ZHANG et al., 2018). Using GP also allows finding BTs that are too difficult to design manually.

2.3 BEHAVIOR TREE DESIGNING

There is a wide range of tools capable of designing behavior trees presented in the literature, where each one has its special ability to differentiate it from the others. Below are some of them:

2.3.1 AIPaint

This tool was developed to solve the problem of designing a behavior-tree for games independently of the game itself. The tool allows the game's agent behavior to be modified at run-time with the game paused. As stated in (BECROFT et al., 2011), the authors consider the game designer as a director who needs to evaluate and change the behavior of the agents, seeing how they act and telling them to stop when happens something it doesn't like.

2.3.2 Groot

From the authors of colledanchise2017behavior, the tools BehaviorTree.CPP behaviortreecpp and Groot groot are respectively: A BT framework focused on robotics projects; and a BT graphical editor. The structure consists of the main library easily integrated with ROS (STANFORD ARTIFICIAL INTELLIGENCE LABORATORY ET AL., 2018) that allows the execution of a previously designed BT that can be fully embedded in the code or loaded dynamically. Working integrated with the framework, Groot enables the visualization of the BTs and its execution at runtime, so that executing nodes are highlighted.

2.3.3 Intera Studio

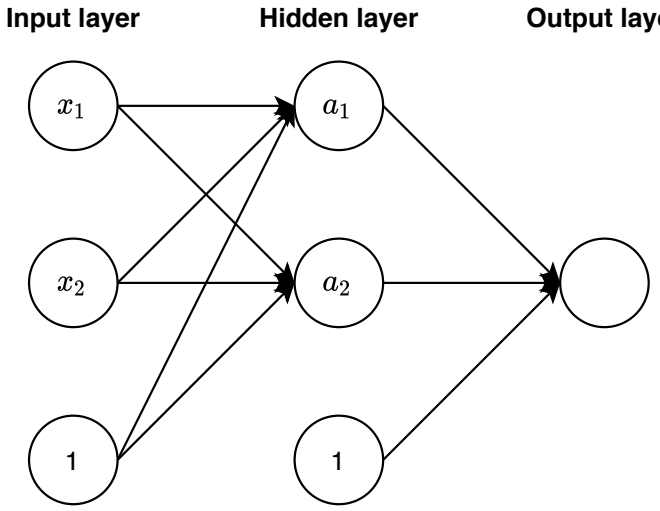
Intera Studio is a commercial tool from Rethink Robotics developed to work with its robotic manipulator platform, Sawyer (ROBOTICS, 2020b). Intera Studio (ROBOTICS, 2020a) allows the designing of BTs using a pick-and-place interface and the online evaluation in the robot through a network connection or in a simulation screen.

2.4 ARTIFICIAL NEURAL NETWORK

In literature, there is no consensus and definitive definitions of Artificial Neural Networks (ANNs). A good explanation is found in (HAYKIN, S., 1994), which describes ANNs as a massively parallel distributed processor made up of simple processing units with the ability to acquire knowledge from a learning process and store it in its connections.

In the conventional programming approach, a programmer tells the computer what to do by breaking big problems into smaller ones, precisely defined tasks that the computer can efficiently perform. In a neural network (NN), it is not needed to tell the computer how to solve the problem. And instead, it learns from observational data. Figure 3 presents a simple neural network structure to solve the XOR problem (LI; DA, 2000). The input and the hidden layers have a bias value, allowing to shift of the activation function to the left or right.

Figure 3 – Simple neural network for the XOR problem



A neural network can be used for classification when an input predicts the class for these values (classify images, voice data, groups of data, etc...). Another use is for regression when the NN indicates an output based on the input, so that the NN will be modeling a function that maps the inputs to the outputs.

2.5 MULTI-ROBOT TASK ALLOCATION

The act of distributing a set of *T* tasks among a group of *R* robots is known as Multi-Robot Task Allocation (MRTA). This problem consists of finding an optimal assignment from tasks to robots to optimize the overall system performance subject to constraints. The whole problem is a dynamic decision and thus should be solved iteratively over time ((KHAMIS, Alaa; HUSSEIN, Ahmed; ELMOGY, 2015a)).

The problem can be formulated as:

1. R: a team of mobile robots r_i; $\{i = 1, 2, \ldots, n\}$;
2. T: a set of tasks t_{ij}; $\{j = 1, 2, \ldots, t\}$;
3. U: a set of robots' utilities, u_{ij} is the utility of robot i to execute task j.

With the assignment of tasks to robots being defined as:

$$A : T \rightarrow R \tag{1}$$

Two MRTA approaches can be divided into two main categories, centralized and decentralized. In the centralized case, each robot or system agent is connected to a central agent responsible for distributing the tasks to all other agents in the system. The second category consists of an architecture where the agents exchange information to negotiate the tasks by themselves or achieve the mission efficiently.

One of the decentralized trending approaches is Market-based techniques such as auctions, which have gained considerable attention because of several desirable features, such as efficiency, robustness, and scalability. These techniques are inspired by economic systems, as in economic theory, an auction is defined by any mechanism of trading rules for exchange (KHAMIS, Alaa; HUSSEIN, Ahmed; ELMOGY, 2015b). In this kind of system, the robots must pay the price to acquire an auctioned task, and then payback is done to the robot once the task is completed (TALEBPOUR; MARTINOLI, 2018).

Auctions are computationally low-cost and do not have high communication requirements. In practical applications it can be performed centrally by an auctioneer or distributively by the robots (HUSSEIN, A.; KHAMIS, A., 2013).

3 DEVELOPMENT

This chapter presents the developed system and all of its modules. The system is composed of a software to design the behavior trees, a framework to manage the communication between robots and stations, a behavior tree to coordinate the hierarchization of the robot tasks and a database of contextual information, used to store data and extract information.

3.1 BEHAVIOR TREE DESIGNER

The developed software for behavior trees in this work can design any BT through a simple interface, allowing the use and simulation of many types of nodes (BARBOSA; PLENTZ; DE PIERI, 2020). This software was developed using C# language. One of the main reasons to use this language is the great set of tools available in the .Net Framework, allowing the fast and robust development of graphical interfaces. Another key feature while choosing the C# language was its modularity, allowing new features and nodes to be easily implemented. This section describes the nodes supported by the tool and the procedure to extend them. Besides, there is a description of the software and its features.

3.1.1 Nodes

Mostly, BT can have three categories of nodes (Composite, Decorator, and Leaf) with six basic types, being four of them for control flow (Selector, Sequence, Parallel, and Decorator) and two for execution (Action and Condition). By default, there are three decorator nodes and a new execution node in the developed tool.

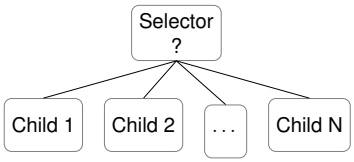
Along with the control flow nodes (Selector, Sequence, Parallel, Invert, Success and Failure) and execution nodes (Action, Condition, and Flipper) implemented, the tool also adds new nodes by extending the base class Node. The default nodes are described below:

Root: This node is represented by the symbol $\emptyset$ and is responsible for executing the BT. At every interval time, ΔT, the node sends a signal *Tick* to its child, which will be forwarded to other nodes according to that node's behavior.

Selector: this is a composite node, which means it implements a behavior to execute its children. The selector node allows selecting a successful child node by executing the children nodes in sequence, from left to right, until any of them succeeds, as shown in Figure 4 and Algorithm 1. If a child returns *success*, the Selector returns *success* too, if all the children fails then the Selector fails and if any child returns *running*, then the Selector also returns *running*.

Sequence: allows selecting a failed child node by executing the children nodes,

Figure 4 – Selector node

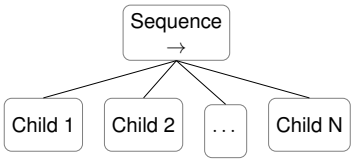


Algorithm 1: Selector node with N children

```
1  for i ← 1 to N do
2      status ← Tick(child(i))
3      if status = success then
4          return success
5      else
6          if status = running then
7              return running
8          end
9      end
10 end
11 return failure
```

from left to right, until any of them fails, as can be seen in Figure 5 and Algorithm 2. When a child returns *failure* then the Sequence instantly returns *failure*, and the same occurs with the *running* status. If all children return *success*, then the Sequence succeeds.

Figure 5 – Sequence node

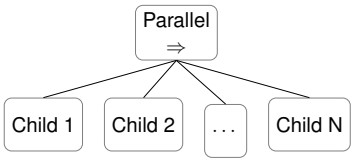


Algorithm 2: Sequence node with N children

```
1  for i ← 1 to N do
2      status ← Tick(child(i))
3      if status = failure then
4          return failure
5      else
6          if status = running then
7              return running
8          end
9      end
10 end
11 return success
```

Parallel: it is a particular node that runs all children in parallel. It is important to note that "parallel" does not mean that all children will run, even if some libraries implement this node that way, it is just a way to execute several nodes at once conceptually. The parallel node, shown in Figure 6 and Algorithm 3, uses a Sequence policy, so when any of the children fail, the parallel node immediately fails, stopping all children with the *running* status. When all the children succeed, the node also returns *success*. The more significant difference of this node with the Sequence node is that the Parallel node allows more then child to be in the *running* status.

Figure 6 – Parallel node

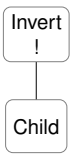


Algorithm 3: Parallel node with N children

```
1 for i ← 1 to N do
2 |   status ← Tick(child(i))
3 |   if status = failure then
4 |   |   return failure
5 |   end
6 end
7 return success
```

Invert: this node is a decorator, so it has only one child and changes the child's status. The invert node inverts the status of its child. If a child returns *success*, this node returns *failure*, if the child status is *failure*, the return will be *success*. If the child status is *running*, the Invert node also returns *running*. That behavior can be seen in Figure 7 and Algorithm 4.

Figure 7 – Invert node



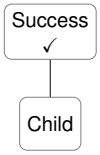
Success: this node always return success if the child status is not *running*, as seen in Figure 8 and Algorithm 5.

Failure: like the Success node, this node always returns failure when the child status is not *running*, as shown in Figure 9 and Algorithm 6.

Algorithm 4: Invert node

```
status ← Tick(child)
if status = failure then
    return success
else
    if status = success then
        return failure
    end
end
return running
```

Figure 8 – Success node



Algorithm 5: Success node

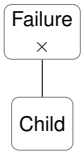
```
status ← Tick(child)
if status = running then
    return running
end
return success
```

Action: this node is a leaf node, so it has no children. When *ticked*, this node executes some function defined by designer and returns a status (*success*, *failure* or *running*).

Condition: Just like the Action node, this node also returns a status, but instead of running a function, a boolean condition is checked, returning *success* for True or *failure* for False. This node never returns *running*.

Flipper: This node is a special node that changes its status at every *Tick* signal. It starts by returning *success* and when *ticked*, the status is changed to *failure*. When *ticked* again, the status is again changed to *failure*, and that behavior repeats as long

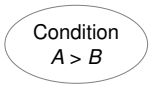
Figure 9 – Failure node



Algorithm 6: Failure node

```
1 status ← Tick(child)
2 if status = running then
3 |   return running
4 end
5 return failure
```

Figure 10 – Condition node



Algorithm 7: Condition node

```
1 if condition is true then
2 |   return success
3 end
4 return failure
```

as the BT is running.

Figure 11 – Flipper node

To create a new node, a programmer must create a new class extending the Node class and override the *Run* method. This is the method that controls the behavior of the node. The programmer must also define the new node type in its constructor (Composite, Decorator, or Leaf).

3.1.2 Features

The developed tool implements a set of features to make it easier to develop BTs, being some of those features present in other tools from Section 2.3, and some specially developed to fill the gaps detected in that tools. The features implemented are divided into four major categories: Basic, Interface, Integration and Debug.

The Basic features are expected in any BT designing software like add nodes, link nodes, undo, redo, copy, paste and cut. Nodes can be added and linked with mouse gestures like click, drag and drop in the developed tool.

The Interface features control how the BT are drawn on the screen. The Grid feature draws a grid on the screen and aligns nodes based on that grid. Another feature in that category is the "Auto arrange": When designing a BT the designer cannot

Algorithm 8: Flipper node

```
1 if stored status = failure then
2 |   stored status ← success
3 |   return stored status
4 end
5 stored status ← failure
6 return stored status
```

organize the nodes, especially if BT has many nodes. This feature runs an algorithm in any node and calculates the best position for that node to keep the entire tree as visible as possible and without links crossing nodes and other links. The result of this feature is shown in Figure 13.

Figure 12 – Auto arrange disabled

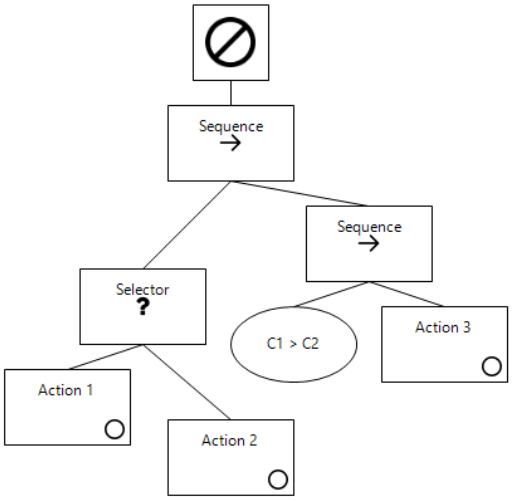
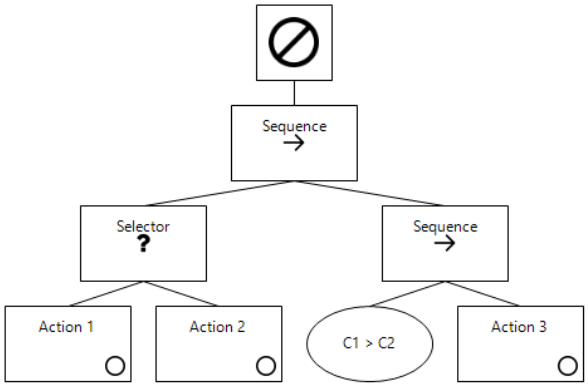
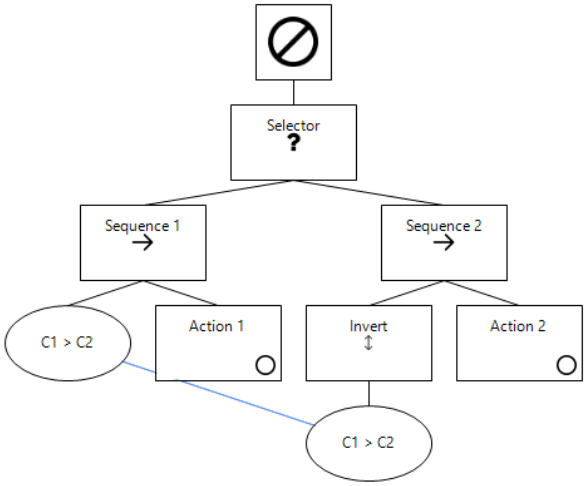


Figure 13 – Auto arrange enabled



Some BTs have leaf nodes that must always return the same status, even if they are in totally different positions on the tree, and for that, the developed software implements a concept of attached nodes. The designer can link two or more leaf nodes if they are of the same type, and these nodes become twin nodes. The twin nodes' status is always the same, for example, if a node is a condition and the designer changes the node status, the twin node status is also changed for the equal value. Figure 14 presents an example of that feature being used, the blue link means that the nodes are attached, and the status of the nodes is always the same, making it easy to create blocks mutually exclusives.

Figure 14 – Attached nodes



The Integration features allows the tool to interact with external tools. The BTs can be saved and loaded as expected. While saving a BT there are two file formats to choose from. A binary format keeps the tree accurately as it is presented in the tool, but only the software can open it. The other format is an XML format compatible with the BehaviorTree.CPP tool from the Section 2.3 and allows that one can design and debug a BT using the software presented in this paper and deploy it using the well-established BT library.

As modularity is one of the key features of BTs, the software also allows importing a BT inside another, making it easier to work with modules to compose a more complex BT. The software also allows exporting the BT as an image or as LaTeX. Later, the package *tikz* will ensure the tree is drawn as vectors, guaranteeing good quality independently of zoom and resolution. Figures 15 and 16 presents the same BT exported in both formats.

The last set of features is used for debugging BT and is the greatest differential from other tools. When pressing the *F*5 key or selecting "Always running" from the menu, the software starts to ticking the tree from the root node. The tick interval can be selected, and the *ticking* occurs even if the designer is modifying the BT. With this

Figure 15 – BT exported as image

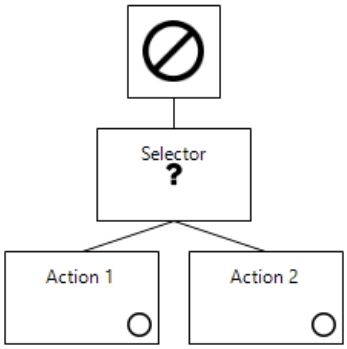
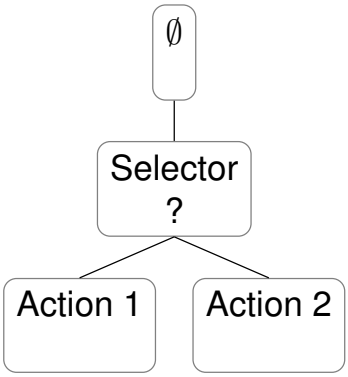


Figure 16 – BT exported as LaTeX



mode enabled, every node reached by the *Tick* signal shows a little box indicating its current status, being blue for *success*, green for *running* and red for *failure*. The same color scheme is used in the links between nodes. If the signal does not reach a node, the colored box is not shown, and the link to that node remains black. With this color feature, it is possible to visualize in real-time the nodes being executed and the behavior of the tree as a whole.

To verify how a status change in the leaf nodes affects the tree, the designer can double click a leaf node to change its status. That feature is available for the Action and Condition nodes, as they are the only nodes that depend on external resources to obtain its status. As expected, when the status of an attached node is changed, its twin nodes automatically have their status changed. If the designer double-click a leaf node that is not reached by the tick signal, the new status will be discarded in the next tick so this.

Figures 17 and 18 presents the BT from 14 being executed. In Figure 17 the condition "C1 > C2" is true, making that the status of the first condition node is *success*, allowing the "Action 1" to be executed. In Figure 18, the condition is false and then its status is *failure*, allowing the "Action 2" to be executed. It can be seen that the two attached nodes always have the same status.

Figure 17 – Condition is true

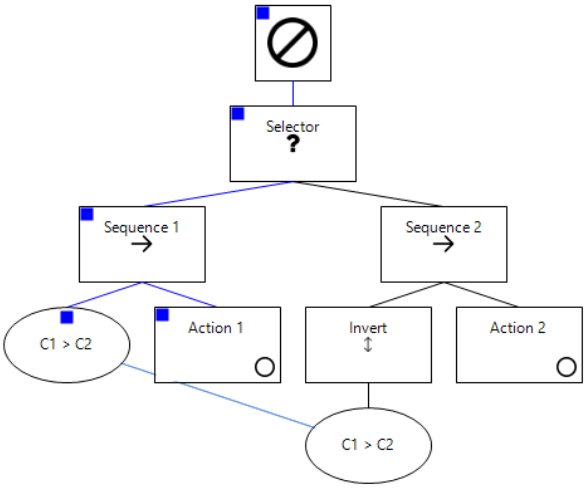
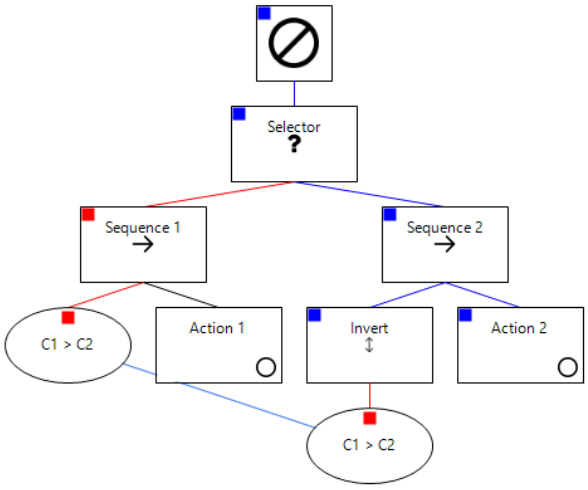


Figure 18 – Condition is false



3.1.3 Interface

The implemented software interface was developed with the "easy-of-use" in mind, putting all the most useful features right in front of the user. The software's main screen is focused on designing and debugging the BT, and in this way, a large empty area is presented, where the user can drag and drop nodes to design the tree. Initially, the only node present is the root node, from where the user must start creating the tree. A bar with the nodes is located in the left panel. The user can select the nodes from this panel and add them to the screen.

The other software features are all available through buttons in the software menu at the top and using shortcuts, such as "F5" to start debugging or "Del" to delete a node. The whole interface works with a touch display in the same way as with a Mouse-Keyboard, and that is why all functions are available from buttons and shortcuts. Figure 19 presents the main screen of the software.

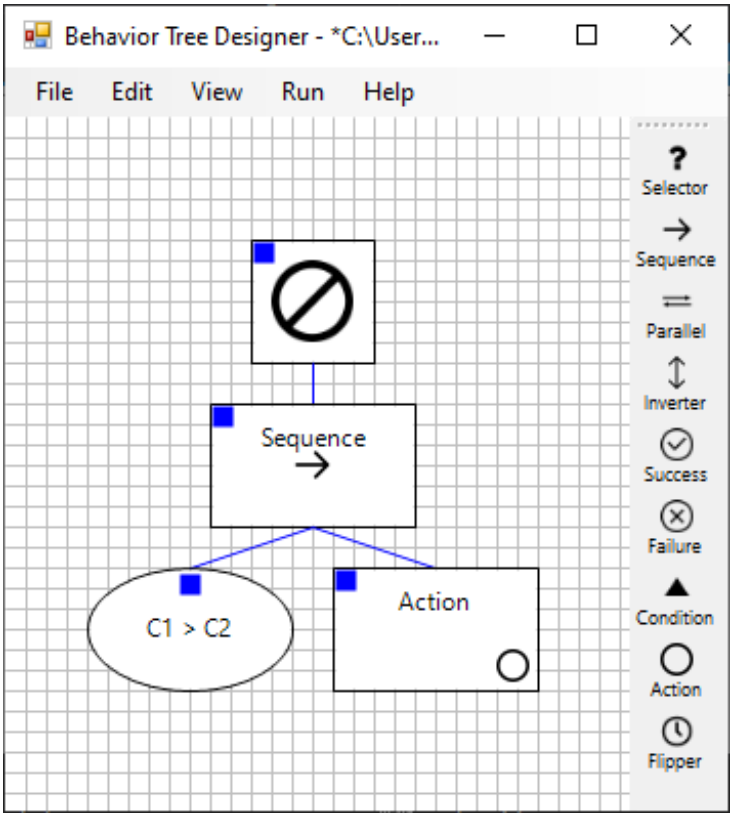


Figure 19 – Main screen of the software

3.2 FRAMEWORK

As can be seen in (DE LUCCA SIQUEIRA; DELLA MEA PLENTZ; DE PIERI, 2016), BT can be used as a solution for the problem of a single robot executing a patrol task subjected to interruptions due to the battery recharging as BT can provide the reactiveness required for such tasks.

The work from (COLLEDANCHISE, M. et al., 2016) explores the fault tolerance nature of BTs associated with the inherited fault tolerance in an MRS. After a brief introduction about BTs for a single robot, this work presents a BT architecture for when many robots can execute a group of tasks, and replace any robots.

In (YANG et al., 2019) the authors present a BT-based framework focusing on the multi-robot task allocation problem. The proposed solution is an architecture that allows the robots to negotiate which one will get which task at runtime based on a priority system based on the characteristics of the individual robots.

Based on this literature review, a novel framework is proposed. The framework is composed of a library that can be used with ROS or any other system that provides the minimum required features, being those integrations done by a plug-in system. The current implementation already provides support out of the box for ROS (STANFORD ARTIFICIAL INTELLIGENCE LABORATORY ET AL., 2018).

The only required features for a system to integrate this framework is:

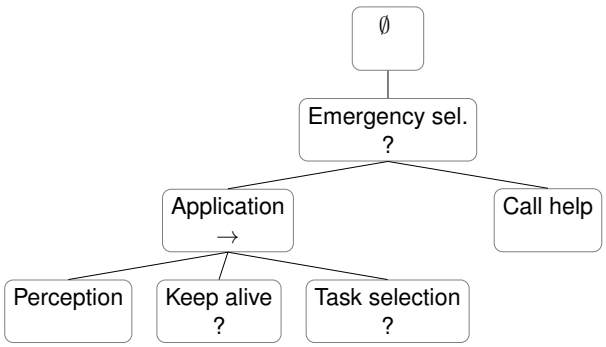
- It supports controlling the robot based on the linear and angular velocity
- Provides a map of the environment
- Provides the current position of the robot
- Provides the current velocity of the robot
- Provides the current battery state of the robot

3.2.1 Behavior Tree Architecture

This new framework allows the robots to negotiate tasks in real-time, considering the tasks and robot constraints. The framework also allows the tasks to be transferred to a most capable robot, making the now-free robot get a new simple task or return to a recharge station.

Figure 20 presents the base structure of the BT. It can be seen that the whole application is a sequence, so for it to return success, all the direct children must also return success. The "Emergency Selector" guarantees that when something goes wrong with the robot application, the robot calls help as it is not working properly. The "Call help" action is executed only when the robot cannot even move, it must be a simple recurrent task of sending a help signal using the robot's network. This signal must contain the robot position, the robot identification, and, when possible, why that robot is calling for help.

Figure 20 – Proposed BT



The "Perception" action obtains the robot sensing of the world in the current tick, like reading sensors and so on. This leaf is not expected to fail, but if it does, the robot immediately must call help as it cannot do any work if its perception of the world is not precise enough.

The node "Keep-alive" is a selector responsible for keeping the robot working as long as possible and informing the system if it cannot continue. As can be seen in Figure 21, there are two possible situations for this selector, when the robot is committed to a task, Figure 22, and when it still does not have a task, Figure 23.

Figure 21 – Keep alive selector

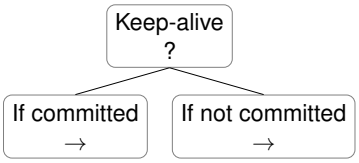


Figure 22 – Transfer Task sub-tree

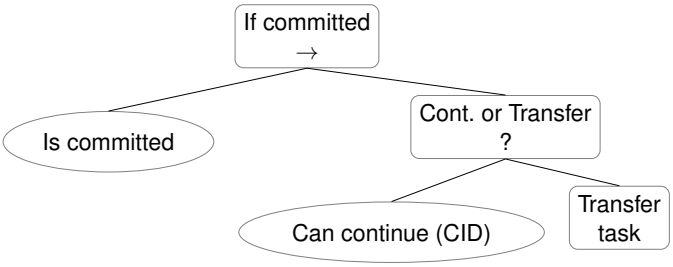
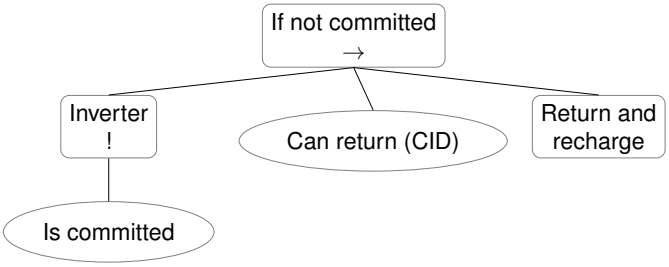


Figure 23 – Return and Recharge sub-tree



When the robot is committed, the "Can continue" checks if the robot can continue the task by verifying the current state of the robot and its perception, passing all this information to the underlying CID module, which will give the final response. If this node determines a robot cannot continue, it tries to transfer the task to another robot on the node "Transfer Task", calling help if the transference fails. The information about the task, the robot's current state, and its perception are saved in CID when a task is transferred, allowing to use of this information to improve the subsequent task-related decisions made by the robot.

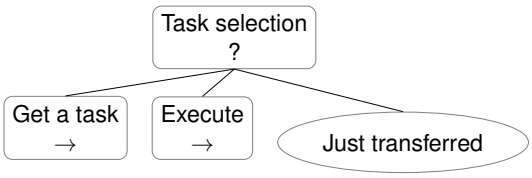
While transferring a task, the robot adds the exchanging requirements to the current task if needed, i.e: The ability to transfer a package to another robot.

If the robot is not committed, the block "Can return" checks if the robot can return to the nearest fixed station for a recharge, which is also done by using the CID. If it cannot return or if the action "Return and recharge" fails, the robot also calls help because it is probably without charge or physically cannot continue. When calling help,

the robot always informs the reason.

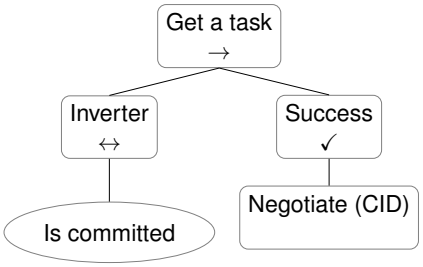
When the "Keep-alive" block succeeds, the robot can eventually execute a task, and to do so, the selector "Task selection" is executed, as seen in Figure 24. This node can perform three different behaviors: Try to get a task if the robot is not committed, execute the current task if the robot is committed, or succeed if the robot commitment has changed, the robot was committed and then transferred its task in the current tick.

Figure 24 – Task selector



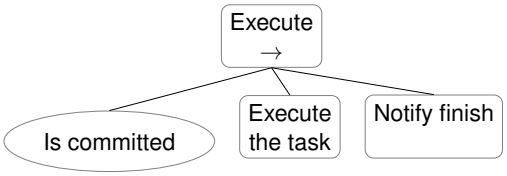
The "Get a task" sequence, Figure 25, first checks if the robot is not committed, as the robot cannot get a new task if it has not finished the current one. If the robot is available, being in the station or currently returning to it, it tries to negotiate a task with other available robots. The negotiation can be done through an auction with the bids given by the CID. If the robot tries to negotiate a task, the "Get a task" selector always returns success as the fact that the robot didn't get a task does not mean a problem, and it is still available to get a new task.

Figure 25 – Task negotiation sequence



The "Execute" sequence, Figure 26, continues the task execution and notify the system when it finishes. The "Execute the task" action can be any required task, and a sub-tree can even replace it. The "Notify finish" action sends a notification to the system that the robot finished the task and stores the data collected during the task execution considering the task properties, robot state, and perception in CID. This stored data is used to improve the system negotiation phase.

Figure 26 – Execution sequence



3.3 CONTEXT INFORMATION DATABASE

To make the robot decision in the negotiation phase to be always improved based on the robot experience, it was developed a module called Context Information Database, which is able to store the context information for a given task and provides methods to use this information for future decisions, in this case, it is used to predict the time and required a battery for each task. As described in Section 3.2, three nodes require the CID evaluation, one for negotiation and two for checking if the robot can continue or return. This section will first present the definition of a task for this work, then the CID structure and how it stores data, and then use it for negotiation and continue/return.

3.3.1 Task definition

In this work it was chosen a formal definition for tasks that allow other robots to continue a task if the task is transferred. All tasks are composed of a header and a body. The header describes the auctioneer, the task's initial and final position, and its restrictions. The body describes the task itself with all the sub-tasks. The task definition is presented in Table 1.

Table 1 – Task definition

Header	
Name	The auctioneer name
Address	The auctioneer address
x^R,y^R,θ^R	Initial position for the task
x, y, θ	Current position for the task
x, y, θ	Final position for the task
Requirements list	The list of requirements for the task. i.e: Can walk; can fly; size restrictions; weight restrictions; speed restrictions; can pick; can take picture; ...
Body	
Sub-task 1	Done or not done
Sub-task 2	Done or not done
⋮	⋮
Sub-task n	Done or not done

The name and address of the auctioneer in the task header allows a robot receiving this task to identify for which robot it needs to send a response back.

When a robot is executing a task, it marks the sub-tasks as done, so when it tries to transfer a task, other robot can continue without doing the whole task again. While

transferring the task, its initial position is also adjusted so that the robot will not return to the initial point of the task.

3.3.2 Context Information Database Structure

The CID was designed to store information about the robot state perception and the task data. It also implements a Neural Network that uses this information. For this purpose, a set of parameters was chosen that allow the use of the same structure for all three kinds of checking required by the BT and analyze the system overall. Not all the stored fields are used in the neural network.

- xI, yI: Initial position (m)
- xF, yF: Target position (m)
- np: Number of path segments
- d: Length of the path in meters (real estimated distance)
- B: Robot battery (%)
- V: Robot speed (m/s)
- TR: Task requirements

The initial position is a point in the current map expressed in meters from the origin of the map. This position can be either the robot position or the initial position of the task. In the same way, the target position is also expressed in meters and can be the final position of the task or a station position. In these parameters, the orientation of the positions is not considered as it does not considerably affect the time and battery required, as detected in experiments.

The path segments and the path length are calculated using an estimated path found by a pathfinder algorithm from the initial point to the target. In this case, it was chosen empirically the time-efficient version of the A* algorithm (GURUJI; AGARWAL; PARSEDIYA, 2016). The path obtained is stored in the CID as the total length (real estimated distance) and the number of segments of that path.

The robot battery is the current battery level of the robot. The robot speed can have two meanings depending on the BT's node. It can be the robot's current linear velocity when the CID checks if the robot can continue a task, or it can be the robot's average linear velocity when the CID decides if the robot must accept a task or not.

The task requirements are the requirements for a task to be chosen by a robot, which are analyzed by the CID when the robot first receives the task.

The CID predicts two pieces of information about the task (ΔB and ΔT) using the information gathered from another task. When the CID still does not have a sufficient amount of data from the robot, it uses a neural network trained with synthetic data.

- ΔB: Required battery (%)

- ΔT: Required time (minutes)

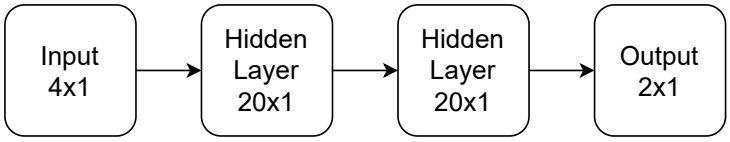
The required battery level is expressed as a percentage, and the required time is given in minutes. For each time that an evaluation is done, the final battery level (*FB*) is calculated according to the Eq. (2). If *FB* is smaller than zero, then it means that the robot has not enough energy for the task, and, in this case, ΔB is considered infinity.

$$\begin{gathered} FB = B - \Delta B \\ \text{If } FB < 0 \text{ then } \Delta B = +\infty \end{gathered} \tag{2}$$

3.3.3 Neural Network

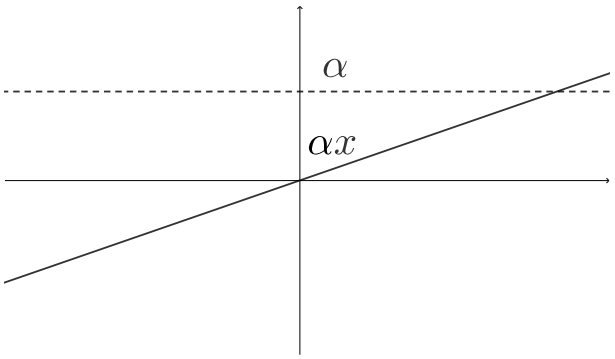
The neural network structure was chosen experimentally using a cross-validation, in a way that it was added many hidden layers and neurons, then they were removed until the mean absolute error (MAE) of the output increases. Based on this, the final structure is composed of two hidden layers, as showed in Figure 27.

Figure 27 – Neural Network Structure



With this structure, there is a 4 inputs layer (distance, number of path segments, current level of battery, and speed), followed by two hidden layers with 20 neurons each and finally, the output layer with 2 neurons (predicted time and battery required). All the layers uses the linear activation function, Figure 28, so the input is the output multiplied by a constant, or $f(x) = \alpha x$. The summary of the layers is presented in Table 2.

Figure 28 – Linear activation function and its derivative



For the optimization algorithm, Adam (KINGMA; BA, 2017) is used, it is an adaptive learning rate method, meaning that it computes individual learning rates for different parameters. Adam can be seen as a combination of RMSprop, and Stochastic Gradient Descent with momentum (RUDER, 2017). It uses the squared gradients to

Table 2 – Summary of layers in the DNN

Type	Shape	Activation function	Parameters
Input	4,1		
Dense	20,1	Linear	100
Dense	20,1	Linear	420
Output	2,1	Linear	42

scale the learning rate just like RMSprop does and takes advantage of momentum by using the moving average of the gradient and not the gradient itself like SGD with momentum.

3.3.4 Negotiation

For the negotiation, the system uses an auction with the bids governed by the CID. According to (SCHNEIDER et al., 2015), auctions for multi-robot task allocation have a performance close to an optimal allocation, with the advantage of scaling much better.

The communication was developed based on the UDP protocol (KUMAR; RAI, 2012), it means that a node on the network (fixed stations or a robot) can broadcast messages to the network and any other node can receive the messages, at the same time that one node can send messages directly to another. The UDP protocol consumes less energy but it also does not guarantee that a message is received, but the auction algorithms developed for this research can deal with this without any problem.

In this system, at the first moment, the robot receives the task basic info and requirements (header) from the fixed stations or other robots trying to transfer their tasks. With this information, the robot evaluates three sets of inputs to the CID:

1. One virtual task that makes the robot go from its current position to the start position of the task, generating the outputs ΔB_i and ΔT_i. The battery input considered here is the current battery level of the robot, and the final battery level is given by $FB_i = B - \Delta B_i$.
2. The real task generating the outputs ΔB_t and ΔT_t. For this evaluation, the battery level considered is the final battery level after the first virtual task, given by $B_i = FB_i$, and the final battery level is given by $FB_t = B_i - \Delta B_t$.
3. Another virtual task that makes the robot go from the final task position to the nearest fixed recharge station, which generates the outputs ΔB_f and ΔT_f. For this virtual task, the battery input used is based on the output of the real task ($B_t = FB_t$), so the final battery level is given by $B_f = B_t - \Delta B_f$.

As the robot's speed for the task trajectory is not known at the evaluation time,

the velocity input used is the average speed of the robot in the last task. All these three evaluations can estimate the time and battery required for the whole estimated task trajectory. The total estimated battery and time required are given simply by the sum of the outputs of each task evaluation, as showed in the Equation (3). For each task, if the required battery is more than the battery available ($FB < 0$), the final battery required ΔB is considered infinity.

$$\begin{aligned} \Delta T &= \Delta T_i + \Delta T_t + \Delta T_f \\ \Delta B &= \Delta B_i + \Delta B_t + \Delta B_f \end{aligned} \tag{3}$$

With the results of the Equation (3), the system calculates the final *BID* for the action, which is given by the Equation (4). The less time and battery the robot spends on the task, the lower will be the value of the bid, which will be in the interval $[0 - +\infty]$. If the robot does not meet the task requirements, the *BID* is also considered ∞ as the robot cannot accept this task.

$$BID = \Delta B \times \Delta T \tag{4}$$

After calculating the *BID*, the robot sends this value to the auctioneer, following the "less is better" policy and choosing the winner for the task. The auction pseudo-codes for auctioneer and bidder are presented in Algorithm 9 and Algorithm 10.

Algorithm 9: Auctioneer algorithm

```
1  broadcasts the header t for task t;
2  proposals ← [];
3  while proposals max time not passed do
4      if receive proposal p then
5          add p to proposals list;
6      end
7  end
8  if length of proposals greater than 0 then
9      bp ← best proposal;
10     remove bp from proposals;
11     send task t directly to the proposer of bp;
12     if task t is accepted by the proposer then
13         mark task t as accepted;
14     else
15         go to line 8;
16     end
17 else
18     schedule task t for later;
19 end
```

Algorithm 10: Bidder algorithm

```
if receive a header h for a task then
    if not committed then
        calculates a bid for h;
        reply the bid directly to the auctioneer;
    end
end
if receive a complete task t then
    if not committed then
        current task ← t;
        reply to the auctioneer that task was accepted;
    else
        reply to the auctioneer that task t was not accepted;
    end
end
```

3.3.5 Continue/Return

For checking if the robot can continue a task or return to the nearest station, just one virtual task needs to be evaluated. If the robot is executing a task, the virtual task has a final position in the final position of the task. If the robot is returning to a station, the final task position is the position of that station. In both cases, the initial position is the robot's current position, and the speed is the current robot speed.

After evaluating, as the checking nodes are of the type "Condition", they must return a boolean value indicating "yes" or "no". The return value is based only in the battery output, according to Equation (5).

$$ContinueOrReturn = FB > 0 \tag{5}$$

3.3.6 Training

The initial training of the ANN is done with synthetic data, which is the same for all robots. These data is generated according to some simple rules:

1. Generates the initial position (from –50m to 50m): xI, yI
2. Generates the final position (from –50m to 50m): xF, yF
3. Generates a battery level (from 10% to 100%): B
4. Generates a speed (from 0.1m/s to 0.5m/s): V
5. Calculates the distance required:

   $$D = \sqrt{(xF - xI)^2 + (yF - yI)^2}$$

6. Calculates the required time in minutes:

   $$\Delta T = (D/V)/60$$

7. Calculates the required battery considering 5% per minute:

$$\Delta B = \Delta T \times 0.05$$

These generated data is far from ideal when individualized, and thus when used to train the ANN it gives sub-optimal results. The results given by the network trained with this data consider that there are no obstacles in the environment. It also considers that all robots have a battery with a linear discharge rate and with the same max/min speed.

To fix this problem over time, every time the robot returns to a fixed station for recharge, it is checked if it can retrain its neural network, this is done by calculating the ratio of new vs old data, if more than a percentage threshold of the data have changed, defined as 20% for this simulation, the the network is retrained, being one network by robot. The real data of the robot is generated when the robot finishes a task (real or virtual) and when it aborts it. A new row of data is stored at all these moments considering a rule of smart replace: If there is already a row with the same initial and final position, then replace that row with the current one. Otherwise, append the data as a new row. When the robot is recharging, it does not need to negotiate tasks or do any other task, so if there are sufficient data, it trains a copy of its ANN with the new data, and when finished, it replaces its neural network with the new one. When new rows of data are added to the training dataset, the old ones are always removed to keep a fixed number of rows and not train the network with data from ancient tasks.

With this training scheme, the robot is incrementally improving its negotiation capabilities, and indirectly, it is modeling the battery discharge curve, the robot movement details, and the map specificities.

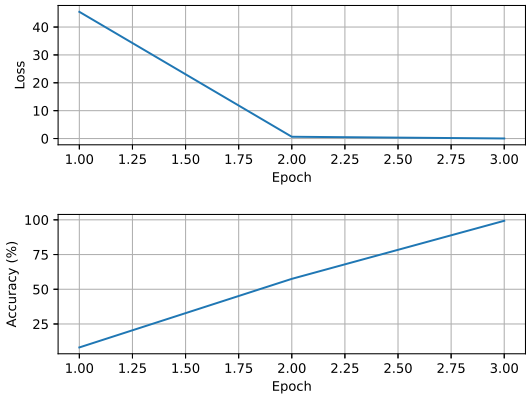
4 RESULTS

Before evaluating the system, the ANN was trained and evaluated individually to ensure that the BT nodes that depend on this would work properly. The simulation considers an area of $50m \times 50m$, varying the number of robots and number of tasks.

The training process was set to execute for as many epochs are necessary to get an accuracy greater than a target, defined as 96% for the simulation. The initial data is composed of a set of 10000 entries (synthetic data) and was split for cross-validation in sets of 80% for train and 20% for tests. In Figure 29, it is presented the training loss and accuracy over the epochs until the desired accuracy is reached for the synthetic data, and Figure 30 presents the training loss and accuracy for the data gathered from the first simulation.

The *Loss* is the measurement of how far the predicted value is from the real value, while the accuracy measures the ratio of the number of correct predictions to the total number of input samples. As the training is a stochastic process, the same procedure of was ran 100 times and the resulting graph is the mean of all of them. As the this is a regression network, for calculating the accuracy, the outputs were considered equal to the target if the error were smaller than 5%.

Figure 29 – Loss and accuracy of the training process with synthetic data



For evaluating the system, it was configured a simulation in Gazebo 3D Simulator according to the Table 3. As the map has some obstacles, we can expect that the results of the system will increase over time as the ANN learns how to give better predictions considering the trajectory of robots through the map.

The simulation was executed twice, considering two different cases: The first case uses the initial training of the ANN, considering the synthetic data, and the second one uses the data from the first simulation to retrain the ANN.

Figures 31 and 32 present the difference between the predictions and the real required time and battery, respectively for tasks when considering the case of synthetic

Figure 30 – Loss and accuracy of the training process with simulation data

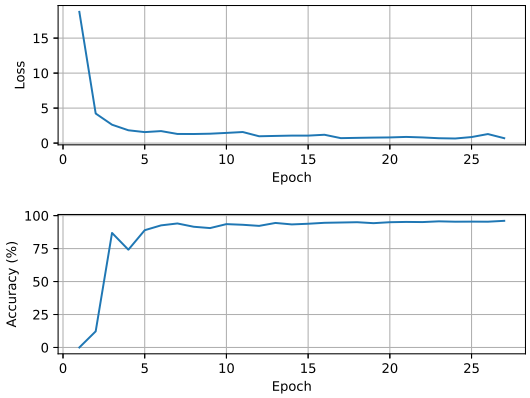


Table 3 – Simulation parameters

Parameter	Value
Number of robots	1, 2, 4, 8
Number of tasks	12, 24, 36, 48
Max speed	0.5 m/s
Number of runs	10
Area size	50 m x 50 m

data. It can be seen in those figures that both time and battery differs very much from the predicted values, and this is quite comprehensive for a system trained with generic data created randomly not taking all system constraints into account. When looking at the time is possible to visualize that it is almost the double of the predict values, while the required battery can get to 10 times easily, clearly showing that the approximation used for battery discharge is extremely inaccurate. With the training of the system with simulation data, is expected that the predicted and real values tend to be equal or very close.

Figures 33 and 34 presents the same for the second case, when the data gathered from the first simulation is used to train the ANN. As can be seem, with the neural network trained with the data from simulation, the absolute error tends to be smaller as the network now considers the main characteristics of the robot and not only some generic data. The system still present differences between the predicted and the real values, but those are very small compared to the training without the simulation data.

Considering the isolated analysis from the Figures 31, 32, 33, 34, we can combine the results to analyze the absolute error for the predictions. In Figure 35 it is presented the absolute error between the predicted and the real required time before and after training the system with the simulation data. The same is applied for the battery in Figure 36. Considering the isolated results, the lower prediction errors after using the simulation data is an expected result.

Figure 31 – Difference between the predicted and spent time to finish the tasks for the case of synthetic data

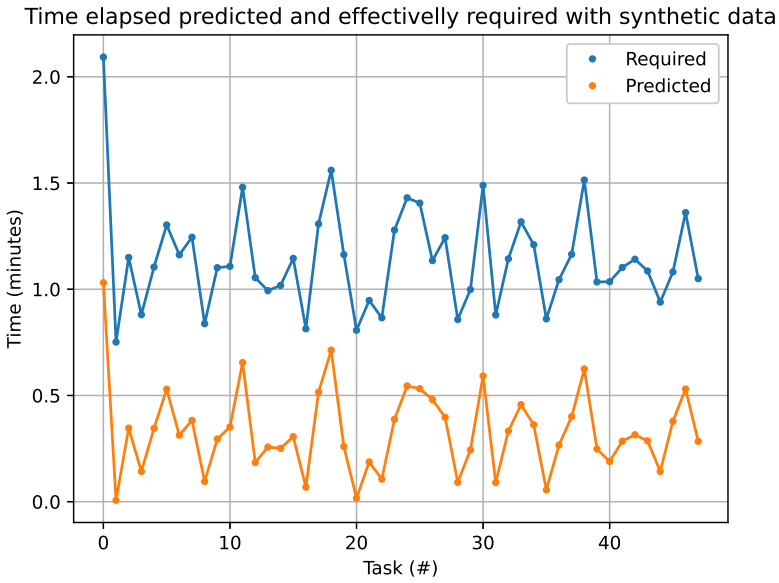
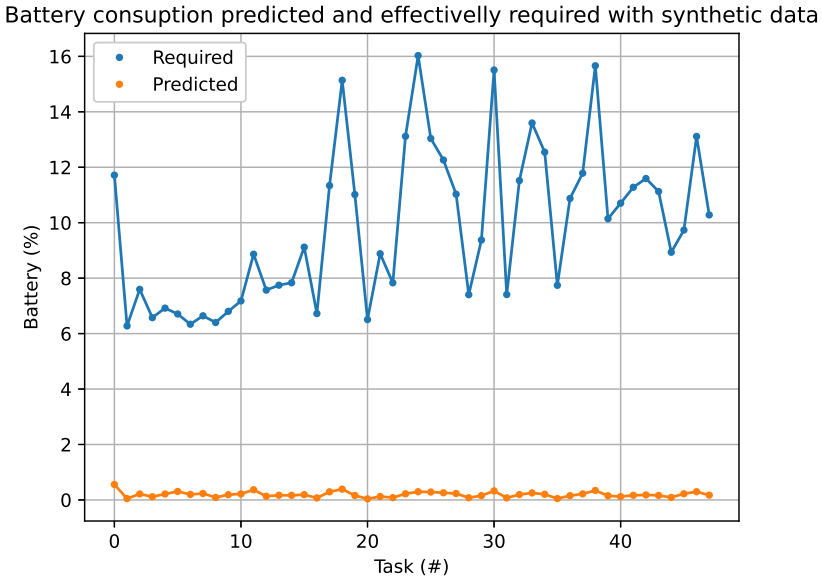


Figure 32 – Difference between the predicted and spent battery to finish the tasks for the case of synthetic data



The predicted time and battery are used to compute the bid for the tasks auction. Improving the prediction allows the robots to make a better decision of which robot is the best option for some task, and this best decision can be analyzed using the distance and time required by the robots to finish a set of tasks. The Figures 37 and 38 presents the results for the overall time and the overall traveled distance for the first case when only the synthetic data is taken into account. The time grow quicker than distance because the more robots on the map, more obstacles to deviate. In these and

Figure 33 – Difference between the predicted and spent time to finish the tasks for the case of simulation data

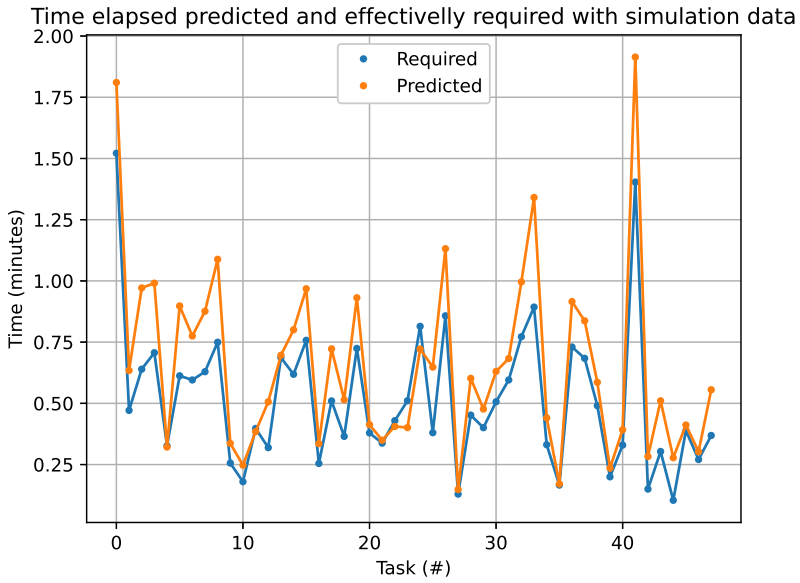
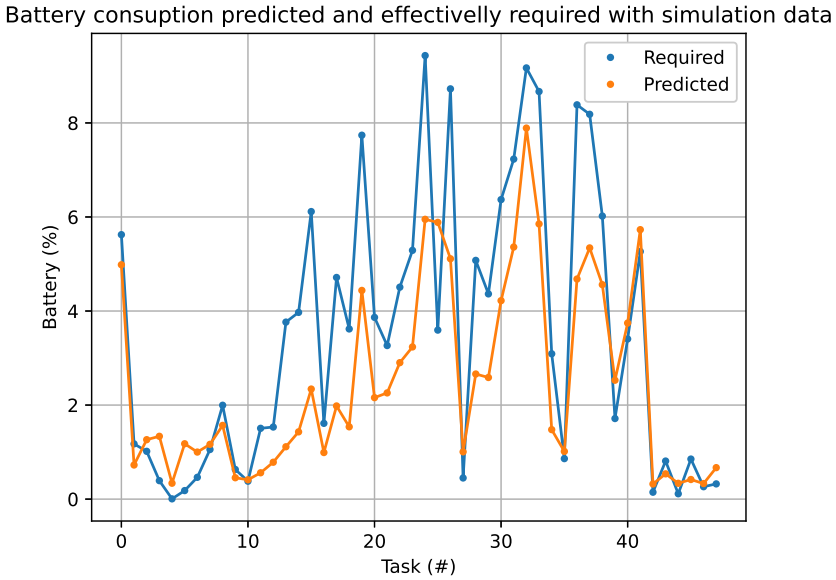


Figure 34 – Difference between the predicted and spent battery to finish the tasks for the case of simulation data



all following figures, the number of tasks is per robot, and because of this, the time and distance increases as the number of tasks also increases. The total number of running tasks is:

$$TNT = NR \times NT \tag{6}$$

Figures 39 and 40 presents the same for the simulation data. As seen in the

Figure 35 – Absolute error between the predicted and spent time to finish the tasks for the case of synthetic data

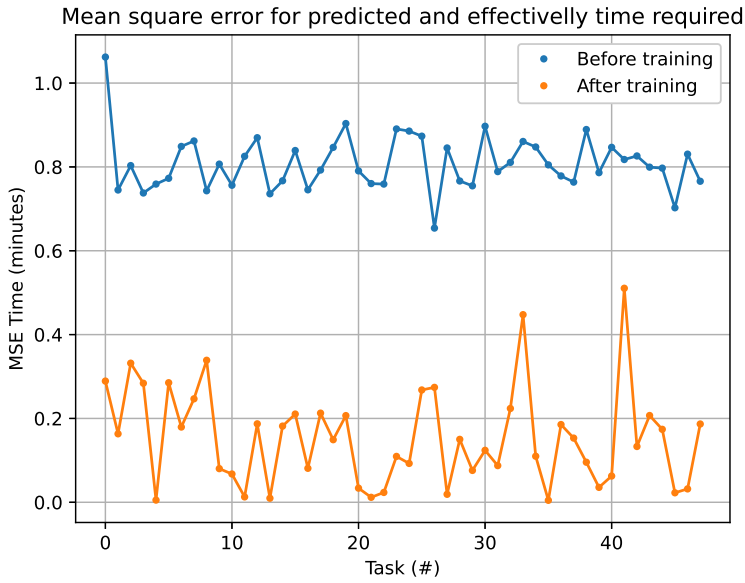
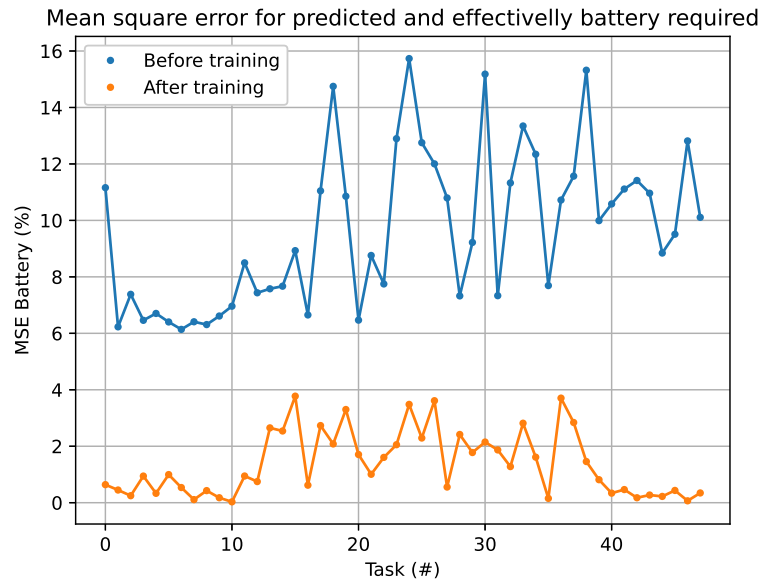


Figure 36 – Absolute error between the predicted and spent battery to finish the tasks for the case of synthetic data



comparison, after retraining the system with the data from the first simulation, the results compare quite well and give much better results, mainly when comparing the traveled distance, which is expected as the ANN is now considering the disturbance in the trajectory caused by the obstacles.

Figure 37 – Total time for all robots comparison for the case of synthetic data

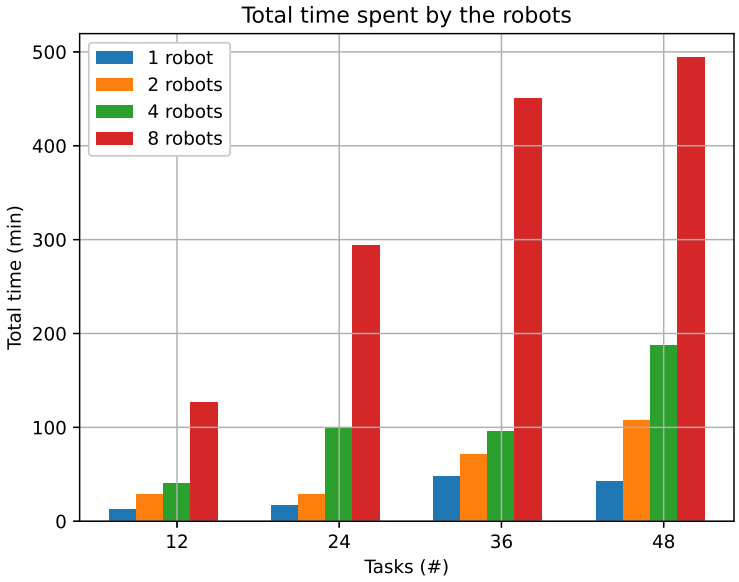


Figure 38 – Total traveled distance for all robots comparison for the case of synthetic data

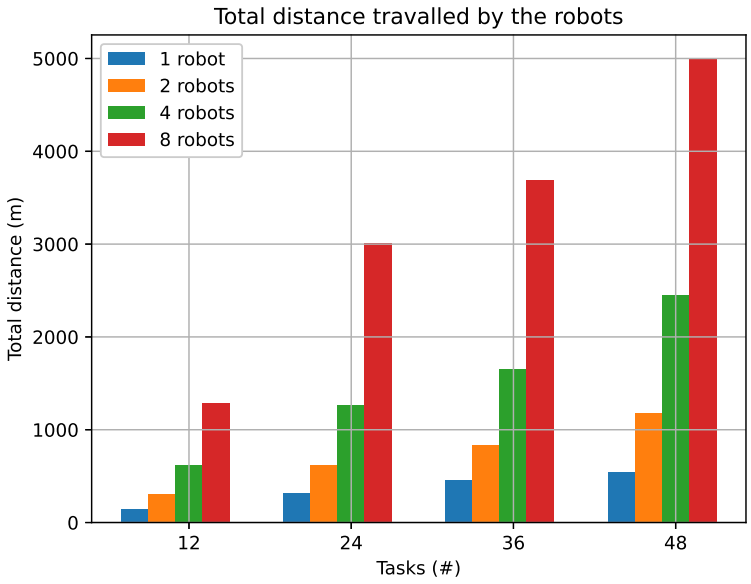


Figure 39 – Total time for all robots comparison for the case of simulation data

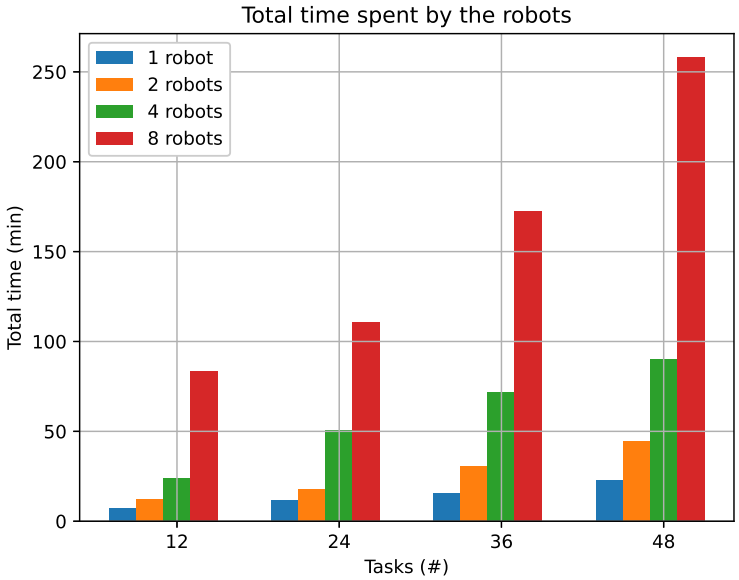
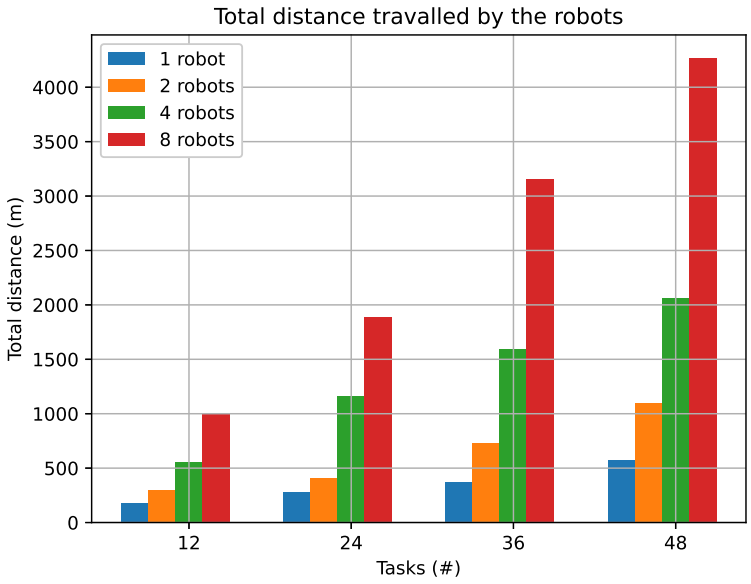


Figure 40 – Total traveled distance for all robots comparison for the case of simulation data



5 CONCLUSION

One of the main problems in the literature for the robotic field is allocating tasks for multi-robot systems. Even with the last advances in this area, this is still an opened-problem with many points to be explored.

This work develops a framework focused on multi-robot systems considering real-time constraints and using behavior trees as a vital tool and a deep neural network as the underlying intelligence to improve task allocation. As the system is a framework, it doesn't require many programming skills to be used and provides tools to manage a multi-robot system as a whole (task allocation, communication, individual data).

The framework allows a group of robots to organize themselves to finish tasks optimized with no centralized control. The framework also provides that the experience of the robots continuously improves the system.

This new framework can be easily integrated into any system that allows one to control a robot, a simulator or a real robot, with the integration with ROS already developed and tested on the Gazebo Simulator.

Based on the results presented in Chapter 4, a BT can be used with success to coordinate the task allocation among multiple robots only with a single instance of the BT being executed in each robot. The coupled ANN considers the system and robot constraints during the negotiation phase with the developed framework. It also considers the robot state at every tick, allowing the dynamic and fast task-transference in case the robot cannot conclude the task.

The results also showed that with experience, the robot learned quite well how to allocate the tasks better than the initial allocation based on the ANN trained with synthetic data, allowing that the overall time and traveled distance of each robot to be improved by its own experience.

The system presented good results but is still not mature yet to be used in a real scenario. The whole framework considers that all robots will be connected to a network all the time, in the range of at least of the fixed station, and this is not the case of most applications where the robots often lose their connection but still need to do their job.

Another problem with the framework is that it still cannot deal with one of the research delimitations, it was assumed that the robots would be able to communicate locally, but the framework itself lacks support for that.

One way that this work can be improved is by optimizing the way the system deals with sub-tasks. In the current implementation, all tasks must be in the form of "go from A to B", a better implementation should allow tasks defined by a sequence of atomic sub-tasks so that complex real-world tasks could be used.

Another improvement for this framework is to allow the robots to work with a mesh network (K.C, 2016), improving coverage and network connectivity.

Considering the navigation of the robots, a great improvement would be to work without a static map, but working with Simultaneous Localization and Mapping - SLAM (DURRANT-WHYTE; BAILEY, 2006), so that the robots would construct the maps as they are navigating to the environment.

REFERÊNCIAS

ARAI, T.; PAGELLO, E.; PARKER, L. E. Guest editorial advances in multirobot systems. **IEEE Transactions on Robotics and Automation**, v. 18, n. 5, p. 655–661, Oct. 2002. ISSN 2374-958X.

BARBOSA, Alexander S.; PLENTZ, Patricia D. M.; DE PIERI, Edson R. A Behavior Tree Designing Tool for Online Evaluation. In: IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society. [S.l.: s.n.], Oct. 2020. P. 537–542.

BECROFT, David; BASSETT, Jesse; MEJIA, Adrián; RICH, Charles; SIDNER, Candace. AIPaint: A Sketch-Based Behavior Tree Authoring Tool. In: PROCEEDINGS of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. Stanford, California, USA: AAAI Press, 2011. (AIIDE'11), p. 2–7.

CAO, Y Uny; FUKUNAGA, Alex S; KAHNG, Andrew. Cooperative Mobile Robotics: Antecedents and Directions. **Autonomous Robots**, v. 4, n. 1, p. 7–27, Mar. 1997. ISSN 1573-7527.

COLLEDANCHISE, M.; MARZINOTTO, A.; DIMAROGONAS, D. V.; OEGREN, P. The Advantages of Using Behavior Trees in Mult-Robot Systems. In: PROCEEDINGS of ISR 2016: 47st International Symposium on Robotics. [S.l.: s.n.], June 2016. P. 1–8.

COLLEDANCHISE, Michele; ÖGREN, Petter. Behavior Trees in Robotics and AI: An Introduction. **Computing Research Repository (CoRR)**, abs/1709.00084, 2017. arXiv: 1709.00084.

DE LUCCA SIQUEIRA, F.; DELLA MEA PLENTZ, P.; DE PIERI, E. R. Semantic trajectory applied to the navigation of autonomous mobile robots. In: 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA). [S.l.: s.n.], Nov. 2016. P. 1–8.

DURRANT-WHYTE, H.; BAILEY, T. Simultaneous localization and mapping: part I. **IEEE Robotics Automation Magazine**, v. 13, n. 2, p. 99–110, June 2006. ISSN 1558-223X.

GARCIA, E.; JIMENEZ, M. A.; DE SANTOS, P. G.; ARMADA, M. The evolution of robotics research. **IEEE Robotics Automation Magazine**, v. 14, n. 1, p. 90–103, Mar. 2007. ISSN 1558-223X.

GONZALEZ-PEREZ, C.; HENDERSON-SELLERS, B.; DROMEY, G. A metamodel for the behavior trees modelling technique. In: THIRD International Conference on Information Technology and Applications (ICITA'05). [S.l.: s.n.], July 2005. v. 1, 35–39 vol.1.

GURUJI, Akshay Kumar; AGARWAL, Himansh; PARSEDIYA, D.K. Time-efficient A* Algorithm for Robot Path Planning. **Procedia Technology**, v. 23, p. 144–149, 2016. 3rd International Conference on Innovations in Automation and Mechatronics Engineering 2016, ICIAME 2016 05-06 February, 2016. ISSN 2212-0173.

HAYKIN, Simon. **Neural Networks: A Comprehensive Foundation**. 1st. USA: Prentice Hall PTR, 1994. ISBN 0023527617.

HAYKIN, Simon S. **Neural networks and learning machines**. Third. Upper Saddle River, NJ: Pearson Education, 2009.

HUSSEIN, A.; KHAMIS, A. Market-based approach to Multi-robot Task Allocation. In: 2013 International Conference on Individual and Collective Behaviors in Robotics (ICBR). [S.l.: s.n.], Dec. 2013. P. 69–74.

IOCCHI, Luca; NARDI, Daniele; SALERNO, Massimiliano. Reactivity and Deliberation: A Survey on Multi-Robot Systems. In: BALANCING Reactivity and Social Deliberation in Multi-Agent Systems. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. P. 9–32.

IOVINO, Matteo; SCUKINS, Edvards; STYRUD, Jonathan; ÖGREN, Petter; SMITH, Christian. **A Survey of Behavior Trees in Robotics and AI**. [S.l.: s.n.], 2020. arXiv: 2005.05842 [cs.RO].

K.C, Karthika. Wireless mesh network: A survey. In: 2016 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET). [S.l.: s.n.], Mar. 2016. P. 1966–1970.

KHAMIS, Alaa; HUSSEIN, Ahmed; ELMOGY, Ahmed. Multi-robot Task Allocation: A Review of the State-of-the-Art. In: **Cooperative Robots and Sensor Networks 2015**.

Ed. by Anis Koubâa and J.Ramiro Martínez-de Dios. Cham: Springer International Publishing, 2015. P. 31–51. ISBN 978-3-319-18299-5.

KHAMIS, Alaa; HUSSEIN, Ahmed; ELMOGY, Ahmed. Multi-robot Task Allocation: A Review of the State-of-the-Art. In: [s.l.: s.n.], May 2015. v. 604, p. 31–51. ISBN 978-3-319-18299-5.

KINGMA, Diederik P.; BA, Jimmy. **Adam: A Method for Stochastic Optimization**. [S.l.: s.n.], 2017. arXiv: `1412.6980 [cs.LG]`.

KORSAH, G.; STENTZ, Anthony; DIAS, M. A comprehensive taxonomy for multi-robot task allocation. **International Journal of Robotics Research**, v. 32, p. 1495–1512, Oct. 2013.

KUMAR, Santosh; RAI, Sonam. Article: Survey on Transport Layer Protocols: TCP UDP. **International Journal of Computer Applications**, v. 46, n. 7, p. 20–25, May 2012. Full text available.

LI, Hong-Xing; DA, Xu Li. A neural network representation of linear programming. **European Journal of Operational Research**, v. 124, n. 2, p. 224–234, 2000. ISSN 0377-2217.

NEHMZOW, U. **Scientific Methods in Mobile Robotics: Quantitative Analysis of Agent Behaviour**. [S.l.]: Springer London, 2006. (Springer series in advanced manufacturing). ISBN 9781846282607.

PARKER, Lynne E. Distributed Intelligence: Overview of the Field and Its Application in Multi-Robot Systems. In: FININ, Tim; KAGAL, Lalana; KENDALL, Elisa F.; LI, Jason H.; LYELL, Margaret; TRUSZKOWSKI, Walt (Eds.). **Regarding the Intelligence in Distributed Intelligent Systems, Papers from the 2007 AAAI Fall Symposium, Arlington, Virginia, USA, November 9-11, 2007**. [S.l.]: AAAI Press, 2007. FS-07-06. (AAAI Technical Report), p. 1–6.

ROBOTICS, Rethink. **Intera Studio**. [S.l.: s.n.], 2020. Available from: `https://www.rethinkrobotics.com/intera`. Visited on: 19 May 2020.

ROBOTICS, Rethink. **Sawyer**. [S.l.: s.n.], 2020. Available from: `https://www.rethinkrobotics.com/sawyer`. Visited on: 19 May 2020.

RUDER, Sebastian. **An overview of gradient descent optimization algorithms**. [S.l.: s.n.], 2017. arXiv: 1609.04747 [cs.LG].

SCHNEIDER, Eric; SKLAR, Elizabeth I.; PARSONS, Simon; ÖZGELEN, A. Tuna. Auction-Based Task Allocation for Multi-robot Teams in Dynamic Environments. In: DIXON, Clare; TUYLS, Karl (Eds.). **Towards Autonomous Robotic Systems**. Cham: Springer International Publishing, 2015. P. 246–257.

STANFORD ARTIFICIAL INTELLIGENCE LABORATORY ET AL. **Robotic Operating System**. [S.l.: s.n.], 23 May 2018. Available from: https://www.ros.org.

TALEBPOUR, Z.; MARTINOLI, A. Multi-Robot Coordination in Dynamic Environments Shared with Humans. In: 2018 IEEE International Conference on Robotics and Automation (ICRA). [S.l.: s.n.], May 2018. P. 4593–4600.

YANG, Q.; LUO, Z.; SONG, W.; PARASURAMAN, R. Self-Reactive Planning of Multi-Robots with Dynamic Task Assignments. In: 2019 International Symposium on Multi-Robot and Multi-Agent Systems (MRS). [S.l.: s.n.], Aug. 2019. P. 89–91.

ZHANG, Q.; XU, K.; JIAO, P.; YIN, Q. Behavior Modeling for Autonomous Agents Based on Modified Evolving Behavior Trees. In: 2018 IEEE 7th Data Driven Control and Learning Systems Conference (DDCLS). [S.l.: s.n.], May 2018. P. 1140–1145.