



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

Afonso da Fonseca Braga

HeMuRo: A Generic Framework for Heterogeneous Multi-Robot Systems

Florianópolis
2021

Afonso da Fonseca Braga

HeMuRo: A Generic Framework for Heterogeneous Multi-Robot Systems

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 Automação e Sistemas.
Supervisor:: Prof. Edson Roberto de Pieri, Dr.
Coorientadora: Profa. Patricia Della M^ea Plentz, Dra.

Florianópolis
2021

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

Braga, Afonso da Fonseca

HeMuRo : a generic framework for heterogeneous multi
robot systems / Afonso da Fonseca Braga ; orientador,
Edson Roberto De Pierri, coorientador, Patricia Della Mía
Plentz, 2022.

80 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 Heterogêneos. 3. Robôs Heterogêneos. 4. Alocação de
tarefas. 5. Decomposição de Tarefas. I. Pierri, Edson
Roberto De. II. Plentz, Patricia Della Mía. III.
Universidade Federal de Santa Catarina. Programa de Pós
Graduação em Engenharia de Automação e Sistemas. IV. Título.

Afonso da Fonseca Braga

HeMuRo: A Generic Framework for Heterogeneous Multi-Robot Systems

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

Profa. Eliane Pozzebon, Dra.
Universidade Federal de Santa Catarina

Prof. Leandro Buss Becker, Dr.
Universidade Federal de Santa Catarina

Prof. Mário Antônio Ribeiro Dantas, Dr.
Universidade Federal de Juiz de Fora

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

Coordenação do Programa de
Pós-Graduação

Prof. Edson Roberto de Pieri, Dr.
Supervisor:

Florianópolis, 2021.

This master's work is dedicated to my parents, brothers
and my colleagues.

ACKNOWLEDGEMENTS

A master's student faces many challenges during the process of producing a master's work. I would like to thank all my professors, colleagues and family that somehow helped me to achieve success.

When I started my studies in 2019, I couldn't imagine how my life would turn around the way it did. I would like to thank all my lab colleagues for the overnights at the Uni, going together to the gym and also spending some quality time together.

I would like to specially thank Ana Caroline Tondo Bonafin and Alexander Silva Barbosa for being by my side in those years.

I would also like to thank my orientator Dr. Edson Roberto de Pieri and my co-orientator Dra. Patricia Della M^éa Plentz for all the support, advises and trust not only for this work but also personal matters.

I could not be where I am today without my parents Eugenio Pacelli Braga and Renata Simone da Fonseca Braga and brothers Bernardo da Fonseca Braga, Guilherme da Fonseca Braga and Humberto da Fonseca Braga. My family gave me support and trusted me to do my best. Being away from my hometown and living in another city is not really easy.

Finally, I would like to express my appreciation for the financial support contributed by CNPq during my studies.

RESUMO

A robótica móvel é apresentada como solução frente a falta de mobilidade dos manipuladores robóticos. Com diferentes formas de locomoção inspiradas em comportamentos biológicos, robôs móveis ganharam espaço em diferentes áreas de atuação, podendo ser encontrados desde ambientes acadêmicos até em ambientes nocivos para seres humanos. Com o passar do tempo, foram surgindo novas ferramentas para simplificar a utilização de sistemas robóticos. Robot Operating System (ROS) possui uma coletânea de ferramentas e algoritmos open-source para controle de robôs, incluindo mapeamento, localização e navegação para robôs móveis. Entretanto, quanto mais funcionalidades são anexadas a um robô móvel, maior será seu custo financeiro e também seu gasto energético, sendo necessário um equilíbrio entre funcionalidade e viabilidade. Sistemas Multi-Robôs são sistemas onde é possível utilizar mais de um robô para executar missões. Sistemas Multi-Robôs heterogêneos combinam diferentes tipos de robôs com diferentes funcionalidades para executar missões, aumentando a eficiência e robustez do sistema. Desta forma, não se faz necessário ter apenas um robô com todas as funcionalidades, mas vários onde cada tipo será responsável por uma determinada função. Novas problemáticas são introduzidas como, por exemplo, alocação e decomposição de tarefas específicas para cada tipo de robô, coordenação para execução de missões e a capacidade de realocar em caso de falhas. Neste trabalho é desenvolvido um framework chamado Heterogeneous Multi-robot (HeMuRo) Framework, responsável pela alocação e decomposição de tarefas para robôs heterogêneos. Com o objetivo de ser open-source e flexível, sua arquitetura modular e distribuída permite modificação e aperfeiçoamento de seus módulos. Aceitando missões simples como entrada, o framework realiza a decomposição da missão e alocação de tarefas utilizando o algoritmo de leilão baseando-se em nível de bateria, tempo necessário para executar a missão e se determinado tipo de robô é capaz de executar a missão. Além disso, em caso de falha ou algum problema durante a execução, as missões podem ser realocadas entre os robôs disponíveis. Este framework funciona de maneira independente havendo também a possibilidade de interação com ROS para comunicação com robôs físicos ou ambientes de simulação. Por último, foram realizadas simulações, envolvendo diferentes cenários como hospital e armazém de logística, onde HeMuRo Framework demonstrou versatilidade para decompor diferentes tipos de missões, fornecendo informações gráficas para análise. Durante as simulações o framework foi capaz de redirecionar missões devido a baixos níveis de bateria dos robôs e também nas situações onde o tempo máximo de execução foi excedido.

Palavras-chave: Sistemas Multi-Robôs Heterogêneos. Robôs Heterogêneos. Alocação de tarefas. Decomposição de Tarefas.

RESUMO EXPANDIDO

INTRODUÇÃO

Por muitos anos, estudos na área de robótica tinham enfoque em sistemas envolvendo apenas um único robô para executar determinadas tarefas. Com o surgimento da Internet das coisas (IoT) e o aumento tanto da complexidade quanto do número de missões, surgiram sistemas multi-robôs (MRS).

Nesta linha de pesquisa, robôs são capazes de inteagir uns com os outros para executar determinadas ações. É possível enumerar algumas vantagens de um MRS perante sistemas com apenas um robô: redundância, robustez, paralelismo e uma maior tolerância à falhas.

Por outro lado, o desenvolvimento de MRS tornou-se complexo devido à vasta abrangência de linhas de pesquisa relacionadas ao tema como: arquitetura, comunicação, enxame de robôs, sistemas heterogêneos, alocação de tarefas e aprendizado. Também se faz necessária uma integração do sistema em si com os robôs para executar as missões, tanto por simulação quanto em ambientes reais.

OBJETIVOS

Essa dissertação tem como objetivo conceber e desenvolver um MRS para robôs heterogêneos com foco em decomposição e alocação de tarefas, utilizando informações centralizadas e descentralizadas e levando em consideração algumas restrições de tempo real como: nível de bateria, tempo máximo para execução da missão e robôs disponíveis.

METODOLOGIA

Para alcançar os objetivos propostos para este trabalho a primeira tarefa se dá na expansão dos conceitos relacionados à MRS, focando em arquitetura, comunicação, robôs heterogêneos e alocação de tarefas. Como um dos objetivos é desenvolver um ambiente para testar o framework proposto, a próxima tarefa se dá na busca de tecnologias e ferramentas para possibilitar simulações. Em seguida o framework é concebido e implementado, sendo apresentado suas características e implementações. Para verificação do framework proposto, serão realizados experimentos em três diferentes ambientes de simulação: o primeiro utilizando apenas o HeMuRo Framework e os dois outros ambientes utilizando ROS e Gazebo.

RESULTADOS E DISCUSSÃO

Após o desenvolvimento e a implementação do HeMuRo Framework foram criados três diferentes cenários para demonstrar sua utilização. A decomposição de

missões e alocação de tarefas foram realizadas de acordo com cada categoria e modelo de robôs. Ao longo das simulações, diferentes situações aconteceram como: redirecionamento de missão devido ao baixo nível de bateria ou por exceder o tempo máximo para execução da missão. Em todas essas situações HeMuRo foi capaz de identificar e alocar outro robô para a realização das missões e nenhuma missão ficou sem ser concluída. Todos os resultados foram apresentados em forma de gráficos plotados pelo próprio framework, facilitando a análise de resultados.

CONSIDERAÇÕES FINAIS

Este trabalho propõe o desenvolvimento de um framework genérico para Sistemas Multi-Robôs Heterogêneos. HeMuRo foi capaz de realizar a decomposição e alocação de tarefas de acordo com cada modelo de robô e gerenciando a execução em tempo real.

Palavras-chave: Sistemas Multi-Robôs Heterogêneos. Robôs Heterogêneos. Alocação de tarefas. Decomposição de Tarefas.

ABSTRACT

Mobile robotics presents as an alternative due to the lack of mobility of robotic manipulators. Biological behaviors inspired many ways of locomotion increasing the usability of robots in multiple areas, adding robots in academic environments and also in environments that are harmful to humans. Over time, new tools emerged to simplify the use of robotic systems. ROS has a collection of open-source tools and algorithms to help engineers build robotic systems, featuring robot control, mapping, localization, and navigation for mobile robots. However, adding more functionalities to a mobile robot impacts higher costs and higher energy consumption. Therefore, a balance between functionality and feasibility is needed. Multi-Robot System (MRS) are systems where it is possible to combine different types of robots with multiple abilities to perform missions, increasing the efficiency and robustness of the system. This way it is possible to use multiple robots with specific abilities, instead of using a single robot with all the sensors and functionalities. New issues are introduced such as, for example, task allocation and task decomposition taking into consideration each type of robot, robot coordination to execute missions, and also the ability to reallocate missions in case of failure. This work presents a framework called Heterogeneous Multi-Robot (HeMuRo) Framework responsible for task allocation and decomposition of missions for heterogeneous robots. With the main goal to be open-source and flexible, HeMuRo Framework was built with a modular and distributed architecture allowing modification and improvements. With simple missions as input, the framework performs mission decomposition and task allocation using an auction algorithm taking into consideration battery level, time to execute the mission, and if the robot has the capability of executing the mission. In case of failure or not being able to finish the mission, there is also the possibility to reallocate to another robot. This framework works independently but there is also the possibility of interaction with ROS to communicate with real robots or simulated environments. Simulations were also conducted, involving different scenarios such as hospital and logistics warehouse. HeMuRo Framework applied versatility to decompose different types of results, obtaining graphical information for analysis. During simulation HeMuRo Framework handled task reallocation due to low-battery levels and also timeout.

Keywords: Heterogeneous Multi-robot Systems. Heterogeneous Robots. Task Allocation. Task Decomposition.

LIST OF FIGURES

Figure 1 – Classification of MAS adapted from (ZAKIEV et al., 2018)	17
Figure 2 – Centralized Topology represented by a task manager assigning tasks for each robot inside the fleet. Adapted from (TIWARI; YOUNG CHONG, 2020)	22
Figure 3 – Decentralized Topology represented by the task manager at the top, fleet managers in the middle and robots in the bottom. Adapted from (TIWARI; YOUNG CHONG, 2020)	23
Figure 4 – Distributed Topology illustrated by an example of the assemble of a car. Each team will have a local goal, independent of the common goal. Adapted from (TIWARI; YOUNG CHONG, 2020)	24
Figure 5 – Solitary Confinement Topology, Regions are well delimited and the robots will only actuate in their specific region. Adapted from (TIWARI; YOUNG CHONG, 2020).	25
Figure 6 – Synchronous Communication. Adapted from (TIWARI; YOUNG CHONG, 2020)	27
Figure 7 – Asynchronous Communication. Adapted from (TIWARI; YOUNG CHONG, 2020)	28
Figure 8 – Example of Heterogeneous MRS	29
Figure 9 – Example of Heterogeneous MRS	30
Figure 10 – Workflow for the MRS proposed in (KIENER; VON STRYK, 2010) adapted from (RIZK et al., 2019).	31
Figure 11 – AWS Robotics - Hospital World	34
Figure 12 – AWS Robotics - Small Warehouse World	35
Figure 13 – Agent’s Architecture for HeMuRo Framework	38
Figure 14 – Data serialization of a message	40
Figure 15 – Example of terminal message printed by the Logger agent	45
Figure 16 – Example of the text file printed by the Logger agent	45
Figure 17 – Graphic report created with from mission messages	46
Figure 18 – Example of the web app by the Logger agent	47
Figure 19 – HeMuRo_Demo Repository Tree	51
Figure 20 – HeMuRo Framework Repository Tree	52
Figure 21 – Simulation of the hospital world with defined positions tagged	59
Figure 22 – Simulation of the small warehouse world with defined positions tagged	61
Figure 23 – In the web interface it is possible to debug the current status of each agent.	63
Figure 24 – In the web interface it is possible to debug the current status of each mission.	64

Figure 25 – Simulation with 2 robots and 8 missions	65
Figure 26 – Simulation with 2 robots e 16 missions	65
Figure 27 – Simulation with 2 robots e 32 missions	66
Figure 28 – Simulation with 4 robots e 8 missions	66
Figure 29 – Simulation with 4 robots e 16 missions	67
Figure 30 – Simulation with 4 robots e 32 missions	67
Figure 31 – Simulation with one Unmanned Aerial Vehicle (UAV) and one Un- manned Ground Vehicle (UGV)	69
Figure 32 – Simulation with two UGVs	70

LIST OF TABLES

Table 1 – Missions available for the first simulation	53
Table 2 – Parameters of simulated atomicTasks	54
Table 3 – ROSModuleRosbot	55
Table 4 – Available Tasks to be executed: Robots eligible to execute	55
Table 5 – ROSModuleMavros	57
Table 6 – AtomicTasks available for the Clover robot	57
Table 7 – Available Tasks to be executed	58
Table 8 – Available Tasks to be executed and Deadline to complete the mission	58
Table 9 – Available Tasks to be executed: Robots eligible to execute and Dead- line to complete the mission	60
Table 10 – Available Tasks to be executed: location in the map	60
Table 11 – Available Tasks to be executed: Robots eligible to execute and Dead- line to complete the mission	61
Table 12 – Available Tasks to be executed: location in the map	62
Table 13 – Available Tasks to be executed: Robots eligible to execute and Dead- line to complete the mission	62
Table 14 – Duration of each simulation	68
Table 15 – Duration of each simulation	70

LIST OF ABBREVIATIONS AND ACRONYMS

2D	Two Dimensional Space
AMCL	Adaptive Monte Carlo Localization
AWS Robotics	Amazon Web Services - Robotics
CPU	Central Processing Unit
CSS	Cascading Style Sheets
FIPA	Foundation for Intelligent Physical Agents
GUI	Graphical User Interface
HeMuRo	Heterogeneous Multi-Robot
HTML	HyperText Markup Language
IA	Instantaneous assignment
IP	Internet Protocol
LIDAR	Light Detection And Ranging
MAS	Multi-Agent System
MR	multi-robot tasks
MRS	Multi-Robot System
MRTA	Multi-robot Task Allocation
MT	multi-task robots
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
SR	Single-robot tasks
ST	Single-task robots
TA	time-extended assignment
tf	Transform tree
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
USAR	Urban Search and Rescue
USV	Unmanned Superficial Vehicle
Wt	Web Toolkit

CONTENTS

1	INTRODUCTION	16
1.1	MOTIVATION	16
1.2	LITERATURE REVIEW	18
1.3	RESEARCH GOALS	20
1.3.1	Main Goal	20
1.3.2	Specific Goals	20
1.4	WORK DELIMITATION AND CONTRIBUTION	20
1.5	CHAPTER ORGANIZATION	21
2	MULTI-ROBOT SYSTEMS	22
2.1	ARCHITECTURE	22
2.2	COMMUNICATION	25
2.3	HETEROGENEOUS SYSTEMS	29
2.4	TASK ALLOCATION	30
2.5	MRS WORKFLOW	31
3	BACKGROUND	32
3.1	ROS	32
3.1.1	Gmapping	32
3.1.2	Navigation Stack	33
3.2	GAZEBO	33
3.2.1	Amazon Web Services - Robotics	34
3.2.2	Important Parameters	35
3.3	WEBTOOLKIT	36
3.4	DEFINITIONS	36
4	FRAMEWORK'S DEVELOPMENT	38
4.1	MODULES	38
4.1.1	Core Modules	39
4.1.1.1	Blackboard	39
4.1.1.2	Default Module and Periodic Module	39
4.1.1.3	DataTypes	39
4.1.2	Communication Modules	39
4.1.2.1	UDPBroadcast	39
4.1.2.2	UDPSender	40
4.1.2.3	UDPReceiver	40
4.1.2.4	UDPReceiverSIM	40
4.1.3	Task Module	40
4.1.3.1	AtomicTasks	40
4.1.3.2	DecomposableTasks	41

4.1.4	Mission	42
4.1.4.1	Auction Module	42
4.1.5	Energy Module	44
4.1.5.1	BatteryManager	44
4.1.6	ROSBridge Module	44
4.1.7	Debug Modules	45
4.1.7.1	Logger Module	45
4.1.7.2	Web Module	46
4.2	SPECIAL AGENTS	46
4.2.1	Logger Agent	46
4.2.2	Charging Station	47
4.3	ROBOTS	48
4.3.1	UGVs	48
4.3.1.1	ROSbot 2.0	48
4.3.1.2	Pioneer 3DX	48
4.3.2	UAV: COEX Clover	48
5	EXPERIMENTAL ENVIRONMENT AND RESULTS	50
5.1	ATOMICTASKS IMPLEMENTATION	53
5.1.1	General Robot	53
5.1.2	The UGVs	54
5.1.3	The UAVs	56
5.2	SIMULATED ENVIRONMENTS	58
5.2.1	Empty Environment	58
5.2.2	Hospital World	59
5.2.3	Small Warehouse World	60
5.3	SIMULATION AND ANALYSIS	62
5.3.1	First simulation: Empty Scenario	63
5.3.2	Second Simulation: Hospital World	64
5.3.3	Third Simulation: Warehouse World	69
6	CONCLUSION	71
6.1	FUTURE WORK	72
	REFERÊNCIAS	74

1 INTRODUCTION

1.1 MOTIVATION

The word robot was first introduced during a play in the Czech Republic called *Rossum's Universal Robots*, by Karel Čapek. The word in the Czech language means serf labor but can also mean colloquially drudgery or hard work (CORKE, 2017). Outside the artistic world, the first robot patent was filed by George C. Devol in 1954. It was a robot with a gripper mounted on a track (CORKE, 2017). Since then, robotics terms started to become popular and robotic companies began to show up.

Robotics Systems can be divided into two main categories: fixed robots (manipulators) and mobile robots. Manipulators are mainly used in industries, and they are used for repetitive but often precise mechanical and physical tasks (SICILIANO et al., 2010). Their structure is similar to a human arm composed of rigid bodies interconnected by joints. However, the lack of mobility of the fixed robots limits their workspace. Mobile robots are a younger field that came to solve this issue. There is a large variety of possible ways for a robot to move across an environment. Walk, jump, run, slide, swim, fly, and roll are some of the locomotion approaches found in the literature (SIEG-WART et al., 2011). Biological behaviors inspired these approaches, and each one has advantages and disadvantages related to the environment the robot will move.

With mobility as an advantage also come some disadvantages as energy storage and consumption, for example. Mobile Robots must be efficient in terms of energy so they can work for more extended periods. Docking and recharging are crucial abilities of an autonomous mobile robot to ensure its performance (RAO; SHIVAKUMAR, n.d.).

Research involving UAV has been on focus for the past few years due to its advantages among various applications. The use of UAVs reduces operational costs, avoids human risks, or even makes possible situations that were not possible before as remote sensing, search and rescue missions, and low-cost mapping and identification, among others, (SHAKHATREH et al., 2018). The use of a camera attached to the UAV to capture images helps to locate defects on the structure of wind turbines, monitoring of high precision agriculture (MUCHIRI; KIMATHI, 2016), Search and Rescue missions (SILVAGNI et al., 2017), surveillance tasks (SEYEDI et al., 2019), and others.

UAVs are a great deal for different applications. However, in extended missions, UAVs cannot perform well due to the lack of energy to complete the mission. A UGV performs better in this situation because, depending of the environment, it does not consume too much energy to move from one place to another when compared to an aerial robot. Another advantage of UGVs compared to UAVs is the battery recharging process, which tends to be less restrictive and easier to implement.

For many years the robotics' study focused on single-robot applications. The emergence of the Internet of things (IoT) applications and robots becoming more ad-

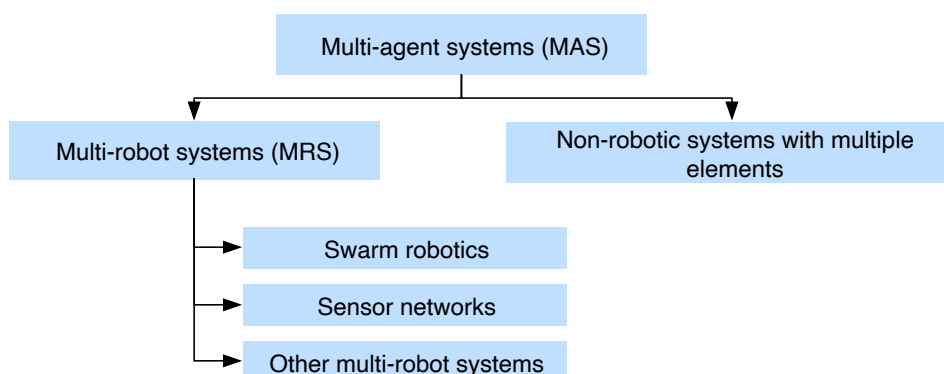


Figure 1 – Classification of MAS adapted from (ZAKIEV et al., 2018)

vanced led the applications to increase their complexity. As the applications became more complex, it was necessary to use more complex solutions to execute them.

Multi-Agent System (MAS) are systems composed of multiple interacting computing elements, known as agents (WOOLDRIDGE, 2009). Agents must be able to interact with other agents and be capable of autonomous actions. There are some advantages when comparing MAS to single-robot Systems. (WANG et al., 2016) enumerate some of these advantages:

- Distributed sensors and actuators, as well as inherent parallelism;
- Larger redundancy, higher robustness, and greater fault tolerance. If one agent fails or is destroyed, its task can be re-allocated;
- Performing tasks that single-agent systems cannot do, such as multiple-vehicles cargo transportation;
- Completing missions usually with higher performance and lower cost than single-agent systems.

MRS is a particular case of MAS and is presented to combine the use of multiple robots to execute one or several missions. An MRS can be composed of swarm robotics, sensor networks, and other multi-robot systems (Figure 1) (ZAKIEV et al., 2018).

According to (PARKER et al., 2016) most recent studies on mobile robot cooperation can be categorized into six topics of studies: Architectures, Communication, Swarm Robots, Heterogeneity, Task Allocation, and Learning. Architecture and Communication topics can be found in all robot systems, and they are relevant to their functioning. Those topics dictate how the robots will interact and be organized in the system. Swarm robots are a large group of homogeneous robots that must interact with each other, and they are a particular field of MRS. Heterogeneity, which is the focus of this work, represents the use of multiple configurations of robots in the same MRS, exploring the best qualities of each robot. Task Allocation designates which robot will

execute each task. When robots vary in capabilities, the task allocation becomes more challenging, selecting the best robot available to accomplish the tasks. The last topic learning focuses on how robots can learn new behaviors and how teams can adapt over time.

This section discussed how robotics, in general, are related to the nowadays society. The following section exemplifies some MRS developed by researchers. A more detailed review about MRS will be presented in chapter 2.

1.2 LITERATURE REVIEW

Solutions aiming at the interaction between mobile robots face many challenges in both theoretical and practical fields. One of the leading research areas is the conception of cooperative systems with two or more robots exchanging information and executing tasks together.

A large spectrum of research involves military applications, and it is not easy to access experiments and results in this area. (LI et al., 2011) describes a MAS for military application. There are multiple agents in this system working to destroy a missile threat. The agents must decide which is the best available option to intercept and destroy the missile. It was implemented with JADE Framework (BELLIFEMINE et al., 2000) and it communicates using the Foundation for Intelligent Physical Agents (FIPA) Contract-Net-Protocol (FIPA, 2002). There are also many non-military applications in the literature. (SHAKHATREH et al., 2018) presents a survey focused on civil applications involving UAVs. (SANTOS et al., 2015) developed and evaluated a model for a single UAV using an agent architecture, exploring its ability to react quickly to changes in the environment. The model was embedded in a real UAV system.

Another important field of MRS is the Multi-robot Task Allocation (MRTA). Research aiming to improve a mission's performance by optimizing the allocation of tasks plays a massive role in robot coordination. (SAMPEDRO et al., 2016) presents an architecture for UAV robot coordination, using two planners to allocate missions: a global planner, responsible for assign and monitor high-level tasks, and an agent planner who will monitor and control each task of the mission allocated to the agent. This framework is implemented in C++ and does not provide any mechanism for agent failures.

The use of heterogeneous fleets increases the complexity of the Task Allocation problem. A widespread MRS application field is the Urban Search and Rescue (USAR). In this scenario, robots must cooperate in an optimized way to help with disasters. In this scenario, there might occur limitations for each kind of robot, e.g., an UGV can only move in places without floods. In this case, the algorithm must identify a suitable robot to execute the mission. The DOMAP Framework (CARDOSO; BORDINI, 2019) presents decentralized online planning for Multi-Agent Programming Platforms. This framework was implemented in JaCaMo. Despite providing an online allocation and

using decentralized communication, it is impossible to re-allocate individual tasks after the execution starts, and the allocation algorithm must allocate all tasks again.

(GUNN; ANDERSON, 2015) describes a framework for heterogeneous robot coordination. This work allows a team to reshape to compensate lost or failed robots caused by network communication failures, mechanical failures, and missing robots. Besides that, the robots' teams are formed dynamically. They can be reshaped during the execution of tasks to improve performance.

(EIJYNE et al., 2020) presents a task-oriented, auction-based task allocation framework. In that work, the author proposes two modules: task allocation and the other responsible for communication. With a centralized communication, a computer will run the MRTA algorithm based on the auction model and assign tasks to the heterogeneous fleet. As input to the auction model, it was considered available resources, battery, and goals distance. The framework was tested on simulated and real environments, and the pseudo-code is available. Failure and task re-allocation were not considered.

(PORTUGAL et al., 2019) presents a framework for simulation and benchmarks of MRS algorithms. This work is used to study patrolling missions. Developed in C++, the framework is open-source and integrated with ROS, which MRTA algorithms are tested. The Stage Simulator was used to execute simulations, and, unfortunately, Stage is no longer maintained.

Energy consumption is also a big topic in MRS. It might be necessary for a UAV to recharge while executing a mission. (YU et al., 2019) presents a study where a UGV needs to visit several spots in an environment, and during this mission, the robot must recharge. The simulation was performed multiple times using fixed and mobile charging stations. The mobile charging stations were installed on UGVs. (ARBANAS et al., 2017) also presented a study focusing on sharing energy between heterogeneous robots. Using the TÆMS Framework (DECKER, 1996), it was developed a decentralized protocol to operate an underwater system based on the battery level of each robot and their position on the environment.

The work presented by (OBDRZALEK, 2017) uses agents to implement a cooperative MRS. This work describes a simulation where agents implemented in failure robots are obligated to migrate from the host to another robot. This presented work was testing basic functionalities of the JADE Framework.

As the reader can conclude, there are multiple subjects, topics, and research involving MRS. According to the environment and specifications, each author has chosen different characteristics and features to implement on their systems. However, for a beginner or learner, these choices are hard to make, and there is not much code available online to run tests or to be used as a start point.

1.3 RESEARCH GOALS

1.3.1 Main Goal

This work has as a main goal to concept and develop a MRS focused on task decomposition and cooperation of robots, obtaining collision-free trajectories, incorporating centralized and decentralized information, and real-time restrictions.

1.3.2 Specific Goals

In order to achieve the main goal, the following specific goals are defined:

1. Design a mobile robot coordination in a system with heterogeneous agents;
2. Propose a software architecture to exchange messages and information among the agents;
3. Define real-time constraints considering energy management;
4. Create a simulation environment to implement the proposed architecture.

This set of specific goals will drive the work to reach a multi-robot system focused on robots' task decomposition and cooperation.

1.4 WORK DELIMITATION AND CONTRIBUTION

This work will focus on creating a MRS Framework with some delimitation.

- The robots move in a controlled environment, without severe network failures;
- The multiple access to the same environment resource was not considered in this approach. E.g. Multiple robots trying to access the same spot in the environment. In this case they will wait until the spot is clear but if it take a while the robot might get stuck;
- The tests of the proposed architecture will be carried out in simulations;
- The execution of some specific task was described by printing messages and waiting a while. Measuring temperature, picking up an object and taking a picture are examples of tasks that were implemented using printed messages.

All experiments were executed in a computer with a Intel Core I5-4570S, 8 GB of RAM and using Ubuntu 18.04 operating system and using ROS1 Melodic. Using this computer as the main computer to execute all simulations resulted in a limited performance. Adding multiple robots to the Gazebo's simulated environment resulted in

a increase of computational power required to perform the simulation. Therefore, four is the maximum amount of simulated robots at the same time using gazebo.

One of the goals of this work is to help beginners to develop and deploy a MRS. As for scientific contribution, the Framework described in this work will be available online for testing and debugging.

The first Framework's proposal was introduced in (DA FONSECA BRAGA et al., 2020). Since this published work, new features were developed, and the framework increased its complexity.

The repository available in (DA FONSECA BRAGA, 2021a) will also accept further contributions and improvements.

1.5 CHAPTER ORGANIZATION

This dissertation is presented in the following chapters. Chapter 1 presents a brief overview and introduces the reader to the problem addressed in this work. Chapter 2 introduces the most common research topics involving Multi-robot Systems. Chapter 3 introduces important concepts and definitions needed for a better understanding of this work. The HeMuRo Framework is presented in Chapter 4. Architecture, modules, task decomposition, and special agents are also covered in Chapter 4. Chapter 5 describes a simulated environment with robots communicating and completing missions with HeMuRo Framework. The last Chapter concludes the work by summarizing the proposed Multi-robot System Framework and how it is performed, and also presents the difficulties encountered and open questions to be addressed in further works.

2 MULTI-ROBOT SYSTEMS

MRS comprehends multiple subjects of studies. (PARKER et al., 2016) enumerated six main areas: Architecture, Communication, Swarm Robots, Heterogeneous Systems, Task Allocation, and Learning. This chapter presents four of those subjects: Architecture, Communication, Heterogeneous Systems, and Task Allocation. Swarm Robots and Learning are not the main scopes of this Master's Work, therefore they will not be discussed. The last section presents a Workflow to develop a MRS proposed by (RIZK et al., 2019).

2.1 ARCHITECTURE

Designing an efficient architecture for MRS impacts directly on the robustness and scalability of the system (PARKER et al., 2016). Scaling up the size of a robot team to execute a mission increases the monetary cost of setting up a team and the complexity of executing the task. Efficient strategies must be chosen and optimized for each scenario. (TIWARI; YOUNG CHONG, 2020) presents optimal architectures for team control strategies.

- Centralized strategies

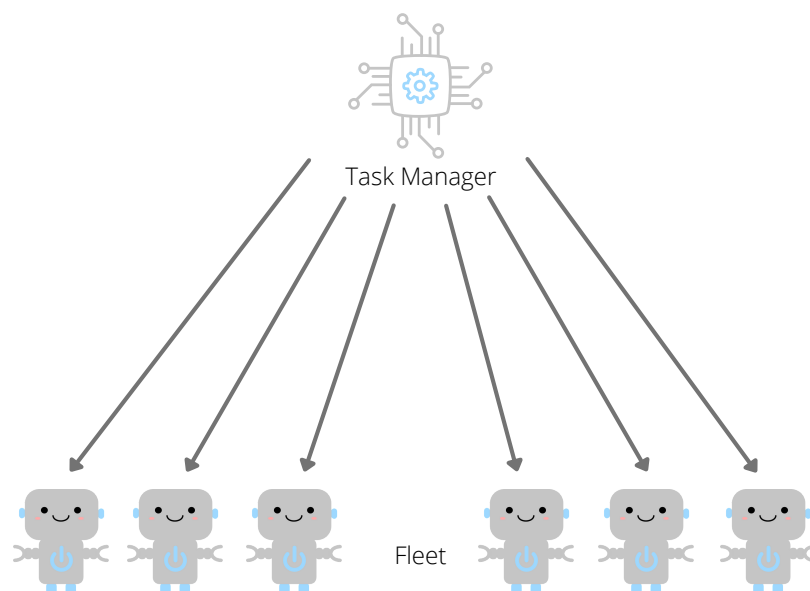


Figure 2 – Centralized Topology represented by a task manager assigning tasks for each robot inside the fleet. Adapted from (TIWARI; YOUNG CHONG, 2020)

In a typical centralized strategy, the system concedes to a single agent the power to acquire information and delegate tasks to other agents. The main characteristic of this centralized strategy is having a simple architecture and it is easy to be

maintained, and also it is easy to set up and replicate. In case of a failure on the master agent, the whole system associated to it will collapse. Another disadvantage is the increase of complexity when the number of agents increases because the master agent will have to delegate tasks to all agents. Figure 2 represents this kind of architecture where a task manager at the center decides what every robot will execute.

- Decentralized strategies

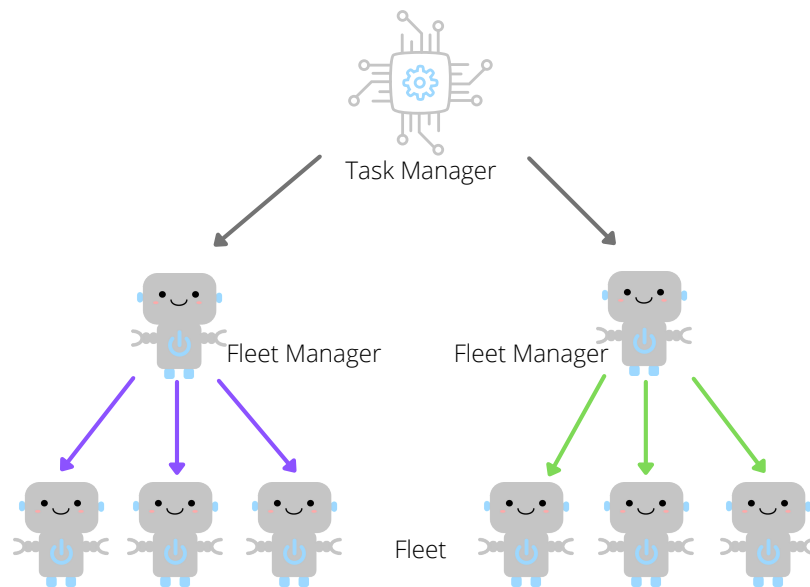


Figure 3 – Decentralized Topology represented by the task manager at the top, fleet managers in the middle and robots in the bottom. Adapted from (TIWARI; YOUNG CHONG, 2020)

Decentralized strategies are implemented to solve problems found in centralized strategies. There will be a global master to delegate global decisions, but there will be a second level of local masters to take decisions too. As advantages, this system will have multiple decision-making points, a failure of any master in an intermediary level does not compromise the whole system. In this aspect, the system is fault-tolerant. As for disadvantages, there is the risk of duplication when two agents decide to do the same task and the risk of conflict of resolution when two tasks are assigned for the same agent. Figure 3 illustrates this architecture.

- Distributed strategies

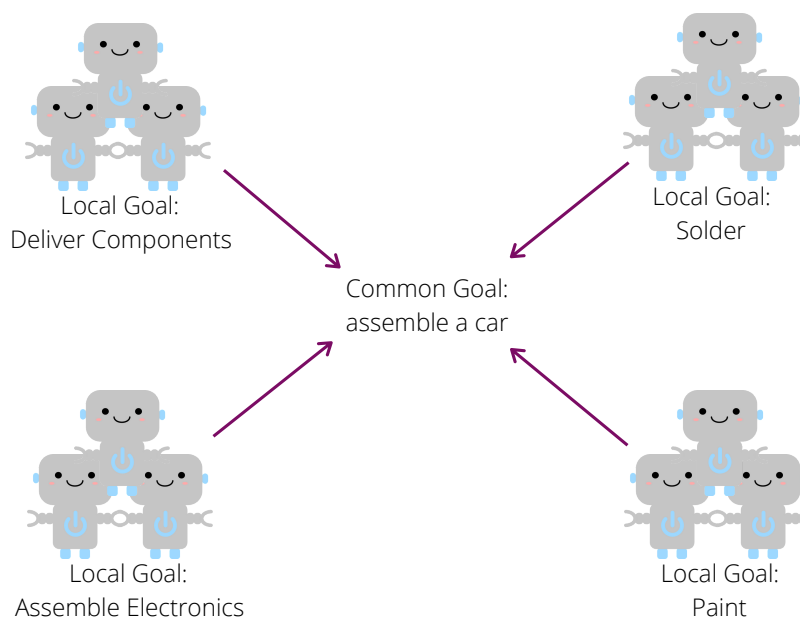


Figure 4 – Distributed Topology illustrated by an example of the assemble of a car. Each team will have a local goal, independent of the common goal. Adapted from (TIWARI; YOUNG CHONG, 2020)

Distributed strategies are based on the "divide-and-conquer" approach. Multiple agents can be working to accomplish a common goal, but at the same time, they can be doing drastically different tasks. As advantages, it's possible to enumerate the scalability, fault tolerance, and computationally efficient. As for disadvantages, there is some difficulty in troubleshooting and deployment and preliminary costs are high. Figure 4 illustrates this architecture. In this case each group of robots are responsible for a task and they will work independently from each other group.

- Solitary Confinement strategies

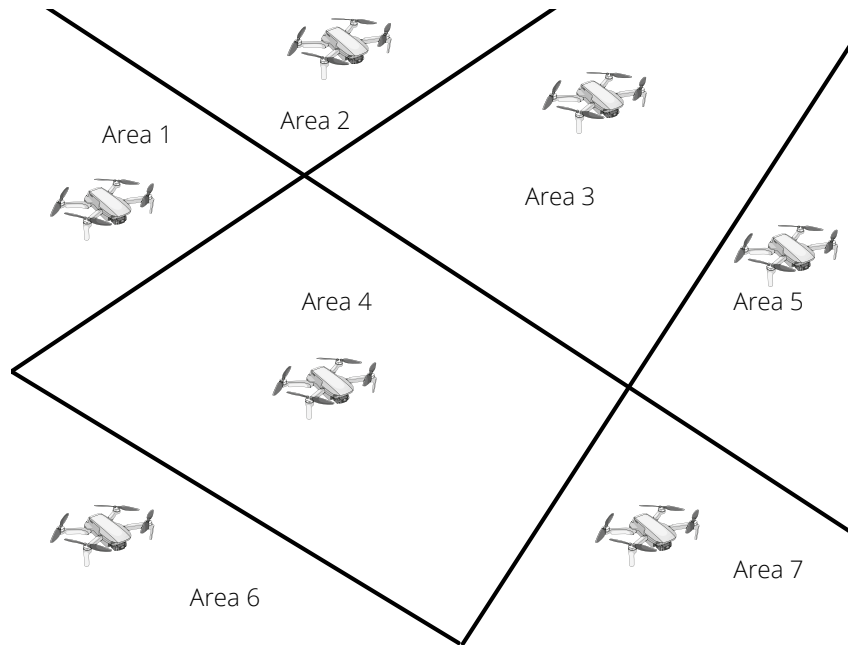


Figure 5 – Solitary Confinement Topology, Regions are well delimited and the robots will only actuate in their specific region. Adapted from (TIWARI; YOUNG CHONG, 2020).

In this strategy, each robot is confined to a region within the target environment. One approach to delimit these regions is by applying the Voronoi tessellation (BHATTACHARYA; GAVRILOVA, 2007). Each agent will only perform actions in its region. This strategy is highly scalable, every single agent will have its autonomy and the confinement regions can be adapted dynamically. As for disadvantages, the initial confinement regions need to be well planned and the smaller the region, the higher the redundancy in observations. Figure 5 exemplifies this architecture where each UAV is confined to an region of the map.

2.2 COMMUNICATION

Communication is a fundamental area of MRS. There will be scenarios where the robots might not have all the information they need to complete a mission. However, it is also possible to achieve a globally coherent and efficient solution through the interaction of robots lacking complete global information. When information is incomplete or missing, the robots must communicate to obtain the missing parts. (PARKER et al., 2016) enumerates three of the most common techniques to obtain the missing information:

1. The use of implicit communication through the world (called *stigmergy*)

This technique relies on sensing the effects of the other agent's actions through their impacts on the world. This is a simple approach and there is no need of

a explicit communications protocol and channels. Nonetheless, this technique is limited by the extent to which a robot's perception of the world reflects the salient states of the mission the robot team must accomplish.

2. Passive action recognition

The agent uses its sensors to directly observe the actions of their teammates. This technique is useful because it doesn't depend upon a limited bandwidth, fallible communication mechanism. However, it is limited by how far the sensors can observe the teammate's progress, and the difficulty of analyzing the actions of robot team members.

3. Explicit (intentional) communication

In this technique, the robots communicate directly and intentionally with each other. In most MRS cases, it is commonly used to synchronize actions, exchange information, and negotiate between robots. In this situation the robots are aware of the actions and goals of teammates. This technique is limited in terms of fault tolerance and reliability. It also provides mechanisms to handle communication failures and lost messages.

According to (TIWARI; YOUNG CHONG, 2020), there are two aspects to consider while designing the communication of a MRS: (i) What information will the agent pass around? and (ii) How will the information be passed around? Those two aspects are really important and they interfere directly with the latency and overhead of the communication.

- Synchronous Communication

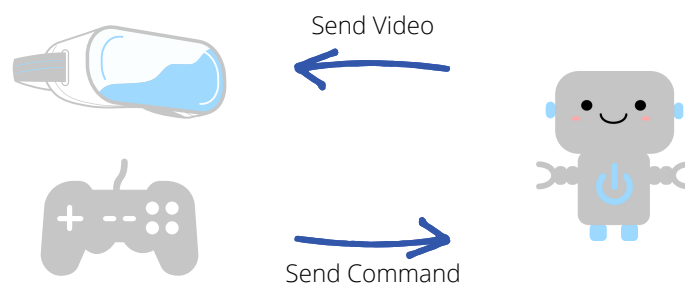


Figure 6 – Synchronous Communication. Adapted from (TIWARI; YOUNG CHONG, 2020)

This type of communication happens when the exchange of information is time coordinated. An example of this communication is a robot's teleoperation. The robot receives commands from a joystick and streams the video in real-time to the user. As pros, it can be quoted real time exchange of information, with almost negligible delays; the response is immediate; can achieve high throughput, and minimal overhead as the crucial data/information can be directly transmitted. As for cons, a complex system is required to synchronize the communication, system complexity increases with the number of peers involved and does not scale well with the size of the team.

- Asynchronous Communication

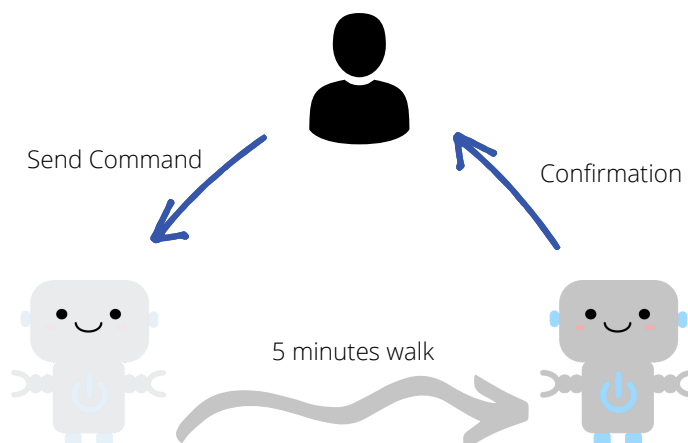


Figure 7 – Asynchronous Communication. Adapted from (TIWARI; YOUNG CHONG, 2020)

This type of communication works without co-ordination between transmitters and receivers. An example: the user sends a command to the robot to go to a specific position and waits for the feedback when the task is completed. This strategy is useful because the information is transmitted when suitable to the sender and is easily scalable. However, it is important to mention the prone to infinite waiting if there is no time-out configured and Larger communication overhead. There must be some flags used to control and allocate the communication servers.

- Disconnected Strategies

The author refers to this type of strategy as a technique created for harsh communicated devoid scenarios. The team coordination and intra-team communication are a challenge. This is applied in sub-terrain explorations, for example. The space-race is another example related to this strategie. In this case, the companies involved in a Rocket project don't want information about their projects to be leaked out during communication, so they have created strict protocols for communication to enforce them not let information leak with this strategy. There is no communication overhead, no restriction to maintain a communication link with peer or base. However, there might be a high risk of redundancy and computational resources are local and limited.

2.3 HETEROGENEOUS SYSTEMS

Robot heterogeneity is possible to be defined in terms of variety in robot behavior, morphology, performance quality, size, and cognition.

According to (PARKER et al., 2016), in most large-scale MRS the benefits of parallelism, redundancy, and solutions distributed in space and time are obtained through the use of homogeneous robots. However, in complex applications where the robot teams may require a large variety of sensors and robots, the use of heterogeneous robots make, the execution of tasks more efficient. It is also important to emphasize that it is expensive or infeasible to add all sensors to a single robot. The use of heterogeneous robots in MRS can amplify its overall performance using the best suitable robot for each task.

There are many scenarios and configurations for a heterogeneous MRS. The use of grounded, aerial, superficial and underwater robots can be combined to amplify the workspace of the missions. Using different robot models at the same category can also be interesting. For example using multiple UGV's models varying its size, bigger robots might carry heavier objects but they tend to be slower and/or consume a great amount of energy, while smaller robots carries small objects but tends to be faster, consuming small amounts of energy.

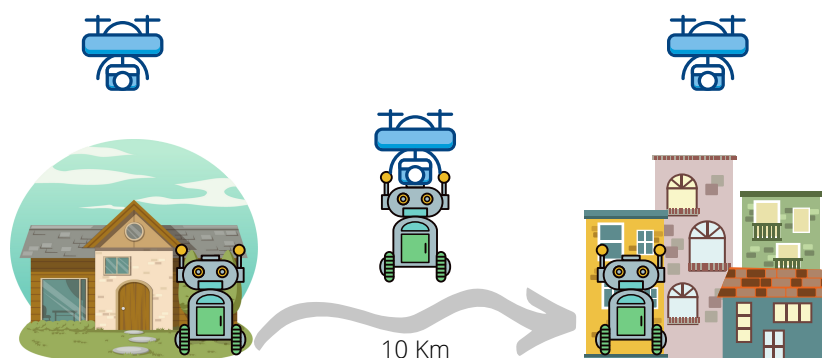


Figure 8 – Example of Heterogeneous MRS

Figure 8 represents a combination of UAV and UGV. The UAV is required to take some aerial pictures from different places. In this case, the UGV has better battery lifetime than the UAV. The UGV can carry the UAV until a specific position, saving the UAV's battery to be used when needed.

There is also another relevant example is for USAR. UAVs tend to be faster than UGVs to search an area. On the other side, UGVs can carry objects and/or people. During a disaster UAVs can be used to search the entire area for people and inform the UGVs where to collect them (Figure 9).



Figure 9 – Example of Heterogeneous MRS

2.4 TASK ALLOCATION

MRTA solutions have been the subject of increasing research over the years. MRTA uses mathematical algorithms, biological behaviors and others to inspire and implement solutions to robot's tasks. (KORSAH et al., 2013) Choosing the best suitable robot to execute each task can considerably improve the efficiency of MRS.

To classify MRTA problems, (GERKEY; MATARIĆ, 2004) proposes three axes new taxonomy for Multi-Robot Task Allocation. They are defined as follows:

- Single-task robots (ST) versus multi-task robots (MT)
ST means that each robot can execute at most one task at a time, while MT means that some robots can execute multiple tasks simultaneously.
- Single-robot tasks (SR) versus multi-robot tasks (MR)
SR means that each task requires exactly one robot to achieve it, while MR means that some tasks need multiple robots.
- Instantaneous assignment (IA) versus time-extended assignment (TA)
IA uses the available information concerning the robots, the tasks, and the environment to an instantaneous allocation of tasks to the robots, with no planning for

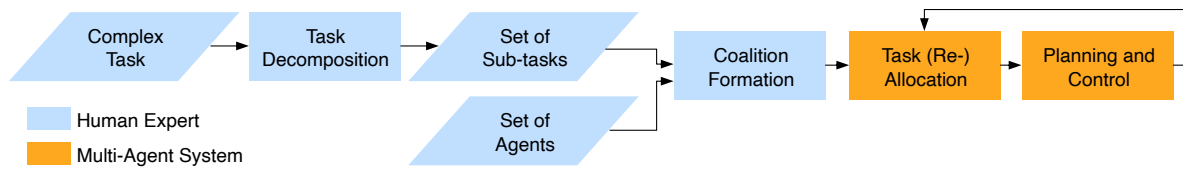


Figure 10 – Workflow for the MRS proposed in (KIENER; VON STRYK, 2010) adapted from (RIZK et al., 2019).

future allocations; TA means that more information is available, such as the set of all tasks that will need to be assigned, or a model of how tasks are expected to arrive over time.

To implement these features, a system is proposed to construct this framework. With these premises the first allocation model implemented in this framework is the ST-SR-IA. This framework will support the decomposition of simple tasks to be executed.

2.5 MRS WORKFLOW

(RIZK et al., 2019) proposes four main design block workflow to design an MRS capable of accomplishing complex tasks systematically: (1) task decomposition, responsible for dividing complex tasks into simpler ones; (2) coalition formation, where teams of agents are created; (3) Task allocation, responsible for assigning sub-tasks to the teams for execution; and (4) task execution/planning and control, responsible for executing a sequence of actions on the environment.

Figure 10 describes a workflow presented by (RIZK et al., 2019) based on the paper from (KIENER; VON STRYK, 2010). In this example, a human designer was required to manually decompose complex tasks to simpler sub-tasks based on the available robots' capabilities and form coalitions from a set of agents. They formed teams that automatically execute task allocation, planning and control. According to (RIZK et al., 2019) many researchers have decided to assign coalitions in MRS to simplify the design statically.

This chapter covered the main characteristics and theory of MRS. The next chapter covers some background on tools used to develop the proposed framework and also important definitions used in this work.

3 BACKGROUND

This chapter covers essential topics and the software used to develop this work. There is some popular software used in the industry to develop and deploy robotic applications. The first section introduces the middleware ROS through some main concepts. The Gazebo simulator is presented in the next section with scenarios from Amazon Web Services - Robotics (AWS Robotics). The third section presents a framework called WebToolkit used to design a GUI web application, providing a more user-friendly interface debug. One of the main chores of the framework developed is task decomposition. The last topic covered in this chapter will be the definitions used to decompose missions into simple tasks.

3.1 ROS

ROS is an open-source, meta-operating system for robots. Created to integrate different devices and programming languages, ROS provides libraries and tools to help software developers create robot applications (ROS, 2020d). Any framework based on ROS could be easily implemented in different programming languages, for example, Python, C++, and Lisp. There are also experimental libraries for Java and Lua.

Some main concepts help understanding how it works and why it is useful for academics and the industry. ROS is a distributed framework composed of applications, and in this case, they are called Nodes. The Nodes are processes that perform computation. The implementation of a robotics control system using ROS requires multiple nodes. For example, a node responsible for acquiring sensor information, one node to perform localization, another node to control the motors, etc.

Nodes communicate with each other through messages containing data structures. There are some predefined messages, but the end-user can also create its message types. These messages are routed via a transport system with publish/subscribe semantics. This transport system is made of topics.

Packages are the main unit for organizing software in ROS. A package may contain nodes, configuration files, datasets, and other valuable files organized together. A stack is composed of multiple packages. The following subsections will be presented packages and a stack related to the robot's localization and mapping.

3.1.1 Gmapping

Gmapping is a ROS package, and it contains a ROS wrapper for OpenSlam's Gmapping (ROS, 2020b). This package provides a laser-based Simultaneous Localization and Mapping (SLAM) and is used to create a 2-D occupancy grid map from laser and pose data collected by a mobile robot.

This algorithm is vital for developing an autonomous robot. However, the SLAM process can be costly to the robot, especially when there is not enough computational power available in the robot. The objective of this node is to create a map of the environment and save it before starting the execution of the missions. A saved map permits the implementation of algorithms related to localization, which are less costly than running it simultaneously with mapping.

The occupancy grid map created by Gmapping provides information about obstacles. Then the Navigation Stack can provide the robot's position on a map and obtain trajectories for the robot to move across the environment. During the execution of the missions, the localization provided by the Navigation Stack will be available instead of the Gmapping SLAM.

3.1.2 Navigation Stack

The Navigation Stack provided by ROS is on a conceptual level, and it is designed to be as general-purpose as possible. It takes in information from odometry and sensor streams and outputs velocity commands to send to a mobile base (ROS, 2020c). The main goal of this stack is to provide algorithms helping a robot locate itself in an environment and move it from one place to another.

There are some basic requirements to use the Navigation Stack on a robot:

- The ROS running on the robot;
- A relation between links and frames of the robot also called Transform tree (tf);
- A sensor publishing data using the correct ROS Message types.

There are multiple algorithms implementing robot mapping, localization, path planning, and trajectory planning included. The user can decide which ones will be used, and there is also the possibility of developing their algorithm and integrate with the algorithms provided by the stack. Some parameters of the Navigation Stack also need to be configured to achieve better performance.

3.2 GAZEBO

Testing real robots in real scenarios can be dangerous and costly. Therefore, a simulation environment to test robotic algorithms is essential. It can reduce costs drastically, and it can predict the particularities of the system. Testing in a virtual environment, or a simulated world, increases test coverage, reduces safety risk, and decreases development time.

Gazebo is a robot simulator capable of efficiently simulate populations of robots in complex indoor and outdoor environments. It makes it possible to test algorithms

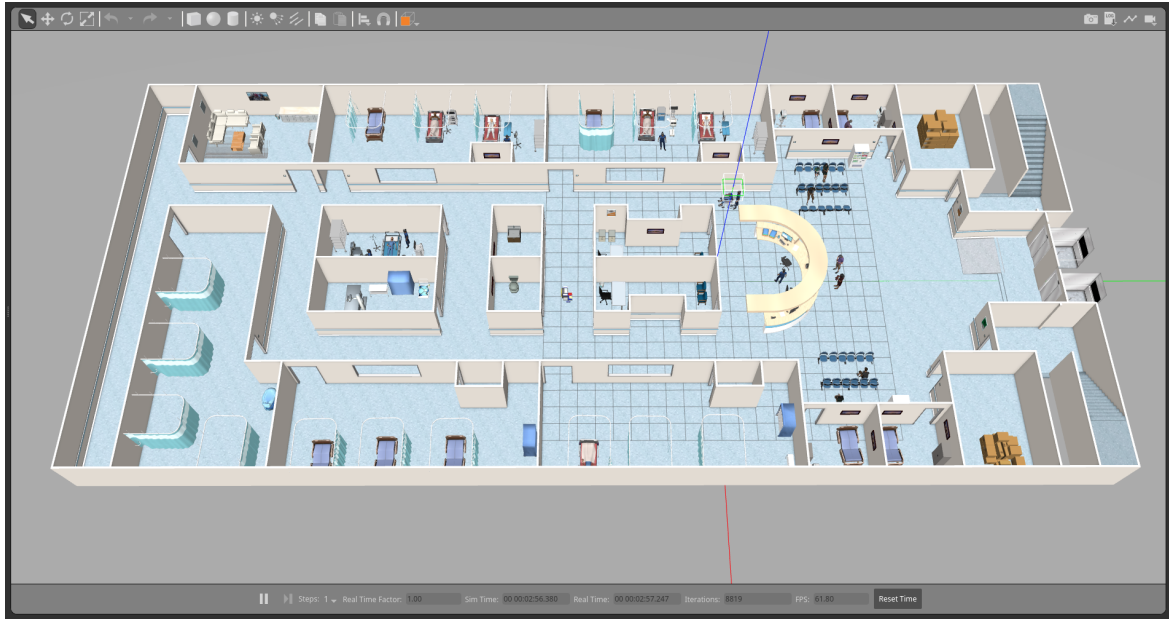


Figure 11 – AWS Robotics - Hospital World

rapidly, design robots, perform regression testing, and provides the necessary data to train an AI system using realistic scenarios (FOUNDATION, 2014). Gazebo is available for Linux, Mac OS, and Windows. Gazebo is integrated with ROS, and there are plenty of robots and scenarios, also called worlds, available through the community. (TAKAYA et al., 2016) presents a integration between ROS and Gazebo.

3.2.1 Amazon Web Services - Robotics

The use of simulated environments is an excellent deal for testing new codes and strategies. However, it takes a significant amount of time to develop simulation worlds. For example, indoor environments require artificial lightning configuration, inserting objects, floor, walls, and configuring their physics parameters, such as density, weight, and collision. Focusing on simulating robotics environments, AWS Robotics have developed several simulated world models for Gazebo to help developers quickly debug robotics codes, e.g., hospital, warehouses, among others (ROBOTICS, 2021).

One of the environments implemented by AWS Robotics is the Hospital World Model. This feature was developed as a solution for testing robot applications in hospital facilities in this particular case. There is a front waiting area, with the reception including patients and medical staff nearby, four small rooms for consults, two storage rooms, and infirmaries. Figure 11 describes all the rooms available on the first floor of the hospital. There are also more extensive medical environments containing two or three floors.

Another modeled world is the small warehouse. The scenario is used for projects involving logistics and warehouse applications. In this particular case, some industrial objects were inserted into the scenario. Pallets, shelves, trash cans, buckets, and big boxes are examples of objects added to make the environment as authentic as possible.

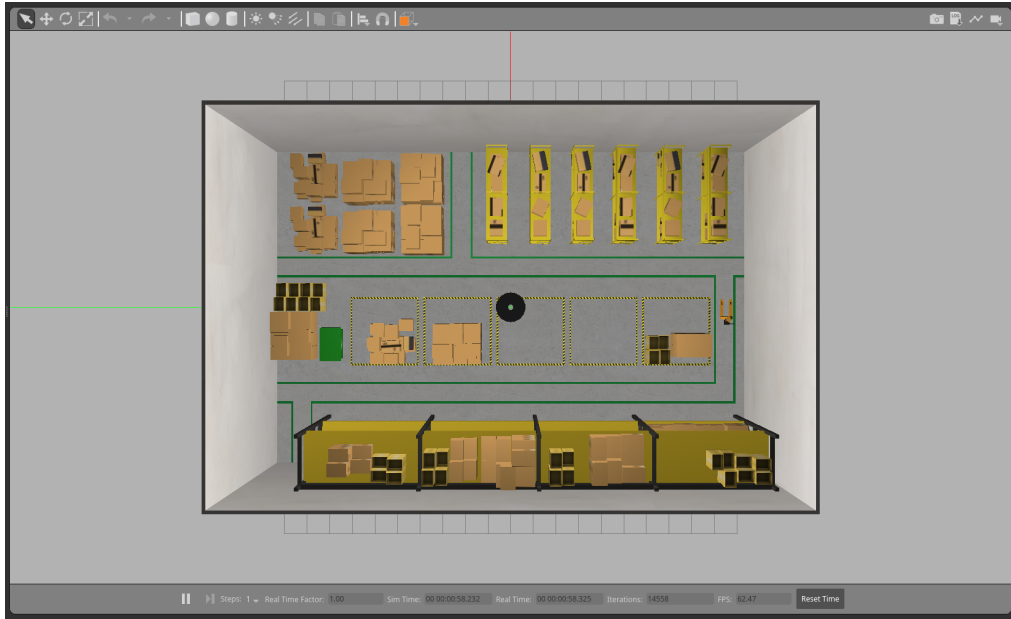


Figure 12 – AWS Robotics - Small Warehouse World

For better visualization of the world, inserting a roof in the scenario is an optional feature. Figure 12 describes the simulated warehouse implemented by AWS Robotics.

All world models are editable. The user can add other models to the environment or remove some of the default objects. Working with virtual machines, this flexibility might be helpful for better performance.

3.2.2 Important Parameters

Depending on which computer will simulate the environment, some adjustments are necessary to improve its performance. The main goal is to execute the simulation keeping the simulation time and real-time as equal as possible.

Some important physical parameters for the simulation are the `max_step_size`, `real_time_update_rate`, and `real_time_factor`. The first parameter refers to the maximum time for each iteration in the simulation solver and the second one refers to the frequency at which the simulation time steps are advanced. Those two parameters are used to obtain the `real_time_factor` in Equation 1.

$$\text{real_time_factor} = \text{max_step_size} * \text{real_time_update_rate} \quad (1)$$

A `real_time_factor` equals 1.0 means that the simulation time and the real-time are equal. If a value is smaller than 1.0 it means that the simulation is slower than the real-time. Configuring the `real_time_update_rate` parameter as 0 the system will run as fast as it can. These parameters just set an upper bound for the simulation system. If the power available is not enough to solve at the right time, the simulated time will run slower, and the `real_time_factor` will increase.

3.3 WEBTOOLKIT

Creating an application can be challenging. Deciding how the information will be displayed to the end-user is crucial to create intuitive software. Developing Graphical User Interface (GUI) interfaces for C++ applications can be complex. The programmer needs to implement solutions using resources available for each operating system the application will run. There is also concern about how the application will be displayed.

Web Toolkit (Wt) is a web GUI library developed for modern C++. With Wt it is possible to quickly develop interactive web UIs with widgets (EMWEB, 2021b). To organize the layout of the web page, Cascading Style Sheets (CSS), layout managers and HyperText Markup Language (HTML) templates are available.

There are essential basic widgets and building blocks to build web applications, including tables, texts, graphics, and PDF rendering. As an example, (EMWEB, 2021a) brings a form programmed in C++ where the end-user must fill in with personal data. This is an easy way of acquiring data.

3.4 DEFINITIONS

To present this work's MRTA solution, some basic definitions were introduced. It was adopted the terminology proposed by (ZLOT, 2006) and a subset of the definitions of the original work was adapted for better understanding. The reader is invited to check all definitions at the original paper.

Definition 3.4.1 (Allocation). Given a set of robots R , let $\mathcal{R} = 2^R$ be the set of all robot subteams. An *allocation* of a set T of tasks to R is a function, $A : T \rightarrow \mathcal{R}$ mapping each task to a single robot or subteam of robots capable of completing it. Equivalently, \mathcal{R}^T is the set of all allocations of the tasks T to the team of robots R . Let $T_r(A)$, $r \in \mathcal{R}$ be the set of tasks allocated to subteam r in an allocation A .

Definition 3.4.2 (Allocatability). Given a team of robots, a task is *allocatable* if it is possible to assign the task to a some robot (or subteam) with the appropriate capabilities and resources to achieve it.

Definition 3.4.3 (Multirobot Allocatability). Given a team of robots R , a set of tasks T is *multirobotallocatable* if there exists some feasible allocation A of tasks to robots (or subteams) in which more than one robot (or subteam) is assigned at least one task, i.e., $\exists r, s \in R, r \neq s | T_r(A) \neq \emptyset \wedge T_s(A) \neq \emptyset$. A set of tasks is not multirobot-allocatable if there are no feasible allocations, or if every feasible allocation requires that all tasks are assigned to a single robot or subteam.

Definition 3.4.4 (Decomposition and Decomposability). A task t is *decomposable* if it can be represented as a set of subtasks σ_t for which satisfying some specified

combination (ρ_t) of subtasks in σ_t satisfies t . The combination of subtasks that satisfy t can be represented by a set of relationships ρ , that may include constraints between subtasks or rules about which or how many subtasks are required. The pairs (σ_t, ρ_t) is also called a *decomposition* of t . The term *decomposition* can also be used to refer to the process of decomposing a task.

Definition 3.4.5 (Multiple Decomposability). A task t is *multiply decomposable* if there is more than one possible decomposition of t .

Definition 3.4.6 (The Multirobot Task Allocation Problem). Given a set of tasks T , a set of robots R and a cost function for each subset of robots $r \in \mathcal{R}$ specifying the cost of performing each subset of tasks, $c_r : 2^T \rightarrow \mathbb{R}^+ \cup \{\infty\}$ find the allocation $A^* \in \mathcal{R}^T$ that minimizes a global objective function $C : \mathcal{R}^T \rightarrow \mathbb{R}^+ \cup \{\infty\}$.

Definition 3.4.7 (Elemental Task). An *elemental* (or atomic) *task* is a task that is not decomposable.

Definition 3.4.8 (Decomposable Simple Task). A *decomposable simple task* is a task that can be decomposed into elemental or decomposable simple subtasks, provided that there exists no decomposition of the task that is multirobot-allocatable.

Definition 3.4.9 (Simple Task). A *simple task* is either an elemental task or a decomposable simple task.

Definition 3.4.10 (Full Decomposability). A task t is *fully decomposable* if a set of simple subtasks can be derived within a finite number of decomposition steps. Such a decomposition (containing only simple subtasks) is called a *full decomposition* of t .

Definition 3.4.11 (Compound Task). A *compound task* t is a task that can be decomposed into a set of simple or compound subtasks with the requirement that there is exactly one fixed full decomposition for t (i.e., a compound task may not have any multiply decomposable tasks at any decomposition step).

Definition 3.4.12 (Complex Task). A *complex task* is a multiply decomposable task for which there exists at least one decomposition that is a set of multirobot-allocatable subtasks. Each subtask in a complex task's decomposition may be simple, compound, or complex.

This chapter discussed some relevant concepts and software related to the proposed framework. The following chapter will present the development of Heterogeneous Multi-Robot (HeMuRo) Framework.

4 FRAMEWORK'S DEVELOPMENT

This Chapter is composed of three sections. The first section presents all the modules in the HeMuRo Framework. The second one describes the special agents developed for the framework. The last section provides information about all the robot models implemented.

The conception of this Framework was based on independent modules. This architecture was chosen over a more conservative approach, allowing the end user to choose which features will be used without huge code modification. It also provides easier integration for future contributions. Figure 13 presents all the modules developed. The orange module in the center represents the only obligatory module every agent must have. The blue ones represent modules that can be activated or deactivated according to each agent's capabilities.

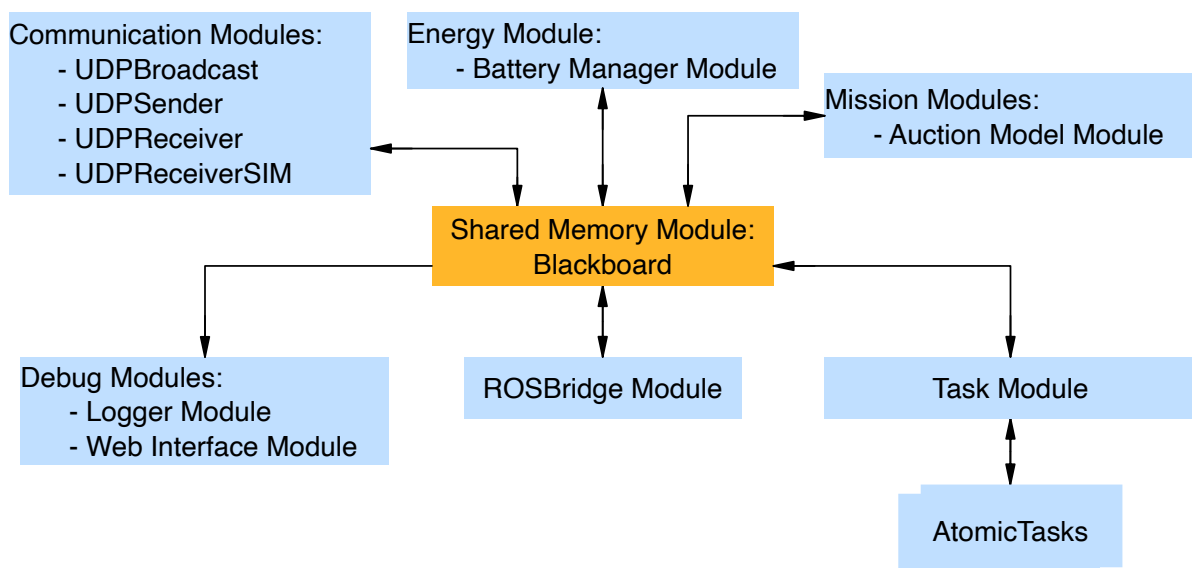


Figure 13 – Agent's Architecture for HeMuRo Framework

4.1 MODULES

There are seven group modules in the framework: Core, Communication, Debug, Energy, Mission, ROSBridge, and Task Modules. They must be instantiated in every agent to use their functions. When looking at the MRS, every agent will have an architecture similar to Figure 13. An agent will have its modules and it will communicate with other agents using communication modules. The end-user can add, remove or edit functions inside the modules.

4.1.1 Core Modules

Core Modules are the basis of each agent. Basic module declaration, shared memory, and log management are some examples of the basic modules.

4.1.1.1 Blackboard

Every Agent has a module responsible for managing their global variables. The Blackboard module implements this feature and it is in fact a shared memory. All modules access the shared memory to get parameters, invoke global methods, and communicate. To connect managing of global variables it is necessary to implement methods to prevent simultaneous access to these variables. The communication between modules is also implemented inside the Shared Memory by using message buffers.

4.1.1.2 Default Module and Periodic Module

Most of the modules derive from these two modules. They have a looping thread inside and they differ from each other at the loop: the Default Module uses a non-periodic thread. This means that the end-user must pay attention to when the loop executes, otherwise it will consume much power of the Central Processing Unit (CPU). As the name shows, the Periodic Module contains a periodic loop and the end-user informs the periodicity.

4.1.1.3 DataTypes

It is not a module but it has vital importance for the Framework. All data types used in the Framework are declared here. As will be explained later, task decomposition, atomicTasks declaration and message types are declared inside this file.

4.1.2 Communication Modules

All agents must be able to communicate with each other at the same MRS. There are four communication modules available at HeMuRo Framework.

4.1.2.1 UDPBroadcast

UDPBroadcast sends periodic messages to all agents containing information, such as the agent's name, position, current status, battery level, and category. This module feeds the Blackboard module and the Logger agent uses this information to create the webpage with the logs.

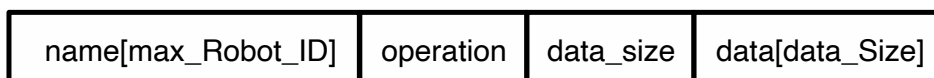


Figure 14 – Data serialization of a message

4.1.2.2 UDPSender

UDPSender sends messages on demand to a single agent or broadcasts a specific message. This module gets the messages from a buffer in Blackboard. An example of the use of this module is sending requests to execute missions to other agents. Figure 14 shows the composition of a message. The first block is the receiver's name. It can be the name of an agent or `broadcast`, to send it to all agents. The maximum size of this field is defined on `DataTypes`. The following field is the operation code. It is the size of an integer and defines who will treat the messages at the receiver. The third field is the size of the data sent, followed by the last field: the data itself.

4.1.2.3 UDPReceiver

UDPReceiver receives all types of messages. This module is used if there is only a single agent running on the computer. The module will acquire all messages sent to the machine's Internet Protocol (IP) address and it process them if they are for this agent.

4.1.2.4 UDPReceiverSIM

If over one agent is running at the same machine, this module should be used instead of the UDPReceiver. This module will acquire all messages sent to the machine's IP address, and will forward to the respective agent's blackboard.

4.1.3 Task Module

The task module decomposes a task in a sequence of actions to be performed in the environment. It is also responsible for controlling the execution of those actions, and treating interruptions in case of an emergency.

4.1.3.1 AtomicTasks

Using all definitions presented in the Background Section, atomicTasks are, according to Definition 3.4.7, tasks that cannot be decomposed. HeMuRo Framework is a generic framework, and it will support different robots. Each robot has its particularities and methods to execute actions. Therefore, each robot will have its atomicTasks implementations. For example, a wheeled robot moves by rotating its wheels, and humanoid moves by walking. The atomicTask `goTo(x,y)` has different implementations for these

two robots, but both represent the same action: to move from where they stand to the desired position. When the action `goTo(x,y)` is invoked, each robot will execute it according to its list of implementation list.

4.1.3.2 DecomposableTasks

A decomposableTask (Definition 3.4.8) is a simple task (Definition 3.4.9) and it is composed of one or more atomicTasks. The decomposableTaskList is like a user guide for a robot. It contains all actions a robot can perform and the execution plan. If a robot can execute a decomposableTask, it must be declared on the decomposableTaskList with their sequence of atomicTasks. It is important to notice that all decomposableTasks with the same name must have the same arguments to be parsed on every robot. For example, the decomposableTask `checkSpot(Position)` can be decomposed for a UAV as `takeOff(Position.Z)`, `goTo(Position)`, `takePicture()`, `sendPicture()` and, on the other side, for a UGV as `moveBaseGoal(Position)`, `takePicture()` and `SendPicture()`.

In order to decompose a decomposableTask into a sequence of atomicTasks, every robot should have a list of all possible atomicTasks and decomposableTasks available in the whole environment, including the ones it cannot perform. Listing 4.1 shows how to declare all possible atomicTasks and decomposableTasks as "enumerators". This data is implemented inside the DataTypes presented at the Core Modules.

```
1 enum class enum_AtomicTask{null, chargeBattery, turnOn, goTo,
   moveBaseGoal, takePicture};
2 enum class enum_DecomposableTask{null, checkSpot, lowBattery, takePicture,
   flightTest, deliverPicture};
```

Listing 4.1 – atomicTask implementation

The atomicTasks are implemented for each type of robot. It is also possible to use the same implementation for two types of robot. After that, a human specialist must declare the sequence of atomicTasks for each decomposableTask and store this information into the robot's memory.

```
1 std::vector<enum_AtomicTask> atomicTaskEnumerator;
2 enum_DecomposableTask dTask = enum_DecomposableTask::checkSpot;
3
4 atomicTaskEnumerator.push_back(enum_AtomicTask::moveBaseGoal);
5 atomicTaskEnumerator.push_back(enum_AtomicTask::takePicture);
6 atomicTaskEnumerator.push_back(enum_AtomicTask::sendPicture);
7 Blackboard->addDecomposableTaskList(dTask,atomicTaskEnumerator);
```

Listing 4.2 – DecomposableTask declaration

Listing 4.2 presents an example of Task declaration for a robot. A vector of `atomicTask` stores the sequence of `atomicTasks` to be performed. After the declaration, this sequence is added to the robot's shared memory into the `decomposableTaskList`.

At this point, the agent knows if it is possible to execute a task. The next step is to provide an algorithm that receives a sequence of `atomicTasks` enumerators with their attributes and converts it into a sequence of `atomicTasks`. Algorithm 1 presents the steps to do this conversion.

Algorithm 1 Algorithm for TaskDecompose Module

```

1: enumList ← find a valid atomicTaskEnumSequence
2: for i = 0 to enumList.size do
3:   atomicTaskSequence[i] ← createAtomicTask(enumList[i])
4: end for
5: checkConsistencyOf(AtomicTaskSequence)
6: if AtomicTaskSequence is consistent then
7:   return true
8: else
9:   return false
10: end if

```

After all these steps to decompose a `decomposableTask`, the `TaskModule` is ready to execute all actions. However, task allocation still needs to be solved for each system.

4.1.4 Mission

Mission modules organize and decide which robot will execute a particular mission. This problem solves MRTA. A mission is composed by one or more `decomposableTasks`. If there is over one `decomposableTask`, they must be independent from each other, creating the possibility of multiple robots executing a `decomposableTask`.

In HeMuRo Framework, it is possible to use Task Allocation Methods that are already implemented or, if it is convenient, the user can implement its own MRTA solution, sending the chosen task to be executed to the `TaskModule`.

4.1.4.1 Auction Module

The Auction Model was implemented as the default MRTA solution. To keep the system decentralized, all agents can offer a mission to all agents. This way, the system will be partially decentralized and partially centralized. It is decentralized because missions can be offered by any agent, but it is also centralized because the agent who offers the mission will be the one choosing executioner of the mission.

The mission owner will offer the mission and wait a few seconds for the bids. All agents capable of executing this mission will send back a proposal to the mission owner. The agent who offered the mission as the winning bid will choose the cheapest

bid. Avoiding conflicts in case of equal bids, the first arrival will be chosen. The user can choose the way the total cost of the mission is calculated. Each atomicTask cost is based on energy consumption and time to complete the execution. A cost function was created as a standard for all agents.

$$Totalcost = \alpha \times EnergeticCost + (1 - \alpha) \times TimeCost \quad (2)$$

Where *EnergeticCost* will be expressed as a percentage of the total energy of the agent, *TimeCost* is the estimated time to execute the mission divided by the relative deadline and $0 \leq \alpha \leq 1$. Changing α allows the cost function to give priority to time or energy. By default, the Agent will sum all the costs of each atomicTask that needs to be executed and $\alpha = 0.5$. This means that the *EnergeticCost* and *TimeCost* will have the same weight.

The auction's winner will receive the command to perform the task after confirming that it is still available. If the winning agent is not available, the second-best bid will be chosen, and so on. When there are no more bids left on the list, the agent owner will offer the mission to all robots again.

The agent must report to the owner of the mission after completing it or in case of not completing the mission, e.g., low battery, hardware failure, or not being able to finish the task before the deadline. In case of mission failure, the mission owner will restart the auctioning process to allocate the mission to another agent. Timeout functions prevent deadlock in the auctioning process.

The agent offering a mission can select different filters, selecting a specific group of agents to execute the mission. For example, specifying a category of agents (e.g. UGV, UAV or Unmanned Superficial Vehicle (USV)), a deadline to execute the whole mission (slower agents cannot complete the mission in time), and of course, depending on the requested mission the agents who cannot perform it will not bid.

At this point is important to inform that when an emergency occurs the agent will report to the mission owner that something occurred and it cannot proceed executing the original mission. The mission owner allocates another robot to finish or restart the mission. If the mission owner is no longer available to do so, the mission will be canceled.

It is also possible to assign a specific mission directly to the robot by sending a TaskModuleMessage using the command `enum_TaskMessage::addTask`. If the agent is available, it will execute the mission. If it is an emergency, it can also send a TaskModuleMessage using the command `enum_TaskMessage::addEmergency` and the agent will stop what it is doing and perform the mission.

4.1.5 Energy Module

In this group, there is only one module developed yet. This single module is responsible for managing the battery level and the charging stations.

4.1.5.1 BatteryManager

The Battery Manager module can operate in two modes: the first is for the agents with an internal battery and the second for charging stations.

1. This module checks periodically the battery status of the agent. If the battery level is low or if distance between the agent and the closest charging station available is growing and there might be no sufficient energy to return and charge if the agent keeps going, the module will contact all charging stations and require a spot to charge. The auctioning process mentioned in the Mission module section was used to select the best charging spot available by choosing the closest available charging spot.
2. Selecting the charging station operational mode will enable the charging spots previously declared and it will offer the available charging spots at the auctioning process. A charging spot has their own limitations, for example power offered and models of robots assisted in charging.

4.1.6 ROSBridge Module

There is a special module for the agent communicating with ROS. The ROSModule will acquire information from ROS and store it on the Blackboard and it will also publish information on ROS based on incoming messages from a buffer in the shared memory.

This module must be implemented individually for each type of agent. Each agent has its topic names and services and probably a different way to operate them. The user must specify all topics that will publish and subscribe. There is also the possibility of creating ActionClients, e.g., using the `Move_base` application. For this module, each implementation must have a different name allowing multiple ROSModules to work simultaneously, for example, when using the same machine to simulate multiple robots. The author recommends inserting the agent's type at the end of the module's name, e.g., `ROSModulePioneer`, `ROSModuleRosbot`, etc.

When using ROS, it is important to inform that all topics regarding an agent must have its name included as a namespace. For example, the robot named `Robot0` has a topic `cmd_vel`, so the topic on ROS must be called `Robot0/cmd_vel`. By doing so, running multiple agents at the same machine with the same topic's name is a possibility. This is an essential feature for simulating multiple agents on the same computer.

4.1.7 Debug Modules

There are two main modules responsible for debugging the system: Logger module and Web module. Those two modules are related and normally it will be instantiated only inside the agent who will keep track of everything that is happening inside the MRS.

4.1.7.1 Logger Module

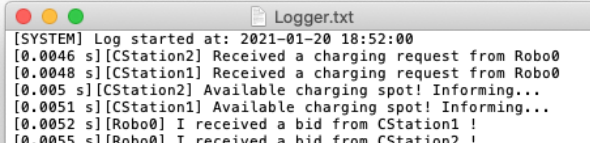
Responsible for printing messages at a terminal station and storing all messages in text files. Modules must not print a message directly at the terminal. Because of concurrent tasks, all terminal messages might be mixed up, and it will be ineligible.

The correct way to print a message at the terminal is by using the `print()` method available on the Blackboard. This method will add the current message to the `loggerMessageModule`. When the message arrives at the logger buffer, the logger will add the sender's name and arrival time to the message before printing it. This way will be possible to identify who sent the message. Figure 15 shows an example of a printed message.

```
[CStation2] Received a charging request from Robo0
[CStation1] Received a charging request from Robo0
[CStation2] Available charging spot! Informing...
[CStation1] Available charging spot! Informing...
[Robo0] I received a bid from CStation1 !
[Robo0] I received a bid from CStation2 !
[CStation1] Waiting for Bids
[CStation1] Waiting for Bids
```

Figure 15 – Example of terminal message printed by the Logger agent

It is also possible to store all the terminal's messages in one text file to analyze and debug later. In this case, before storing every message, the Logger agent will add a time informing when the message was received. The time is measured in seconds counted after the beginning of the simulation. Figure 16 illustrates an example of a text file.



```
Logger.txt
[SYSTEM] Log started at: 2021-01-20 18:52:00
[0.0046 s][CStation2] Received a charging request from Robo0
[0.0048 s][CStation1] Received a charging request from Robo0
[0.005 s][CStation2] Available charging spot! Informing...
[0.0051 s][CStation1] Available charging spot! Informing...
[0.0052 s][Robo0] I received a bid from CStation1 !
[0.0055 s][Robo0] I received a bid from CStation2 !
```

Figure 16 – Example of the text file printed by the Logger agent

Another feature presented by this module is saving information about all missions in another text file, allowing the user to debug and create mission reports only. Figure 17 presents some charts made using the Logger information.

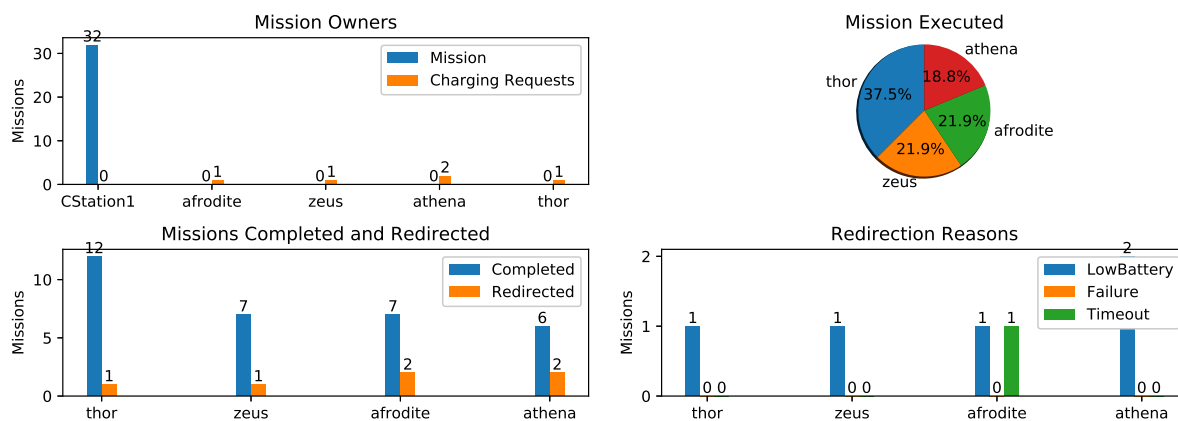


Figure 17 – Graphic report created with from mission messages

4.1.7.2 Web Module

When this module is activated, a local web page is created using the Wt (EMWEB, 2021b). To enable this module, the user must install the library on the computer. At the moment, this interface shows all the available agents in the environment (except for the logger agent), informing its category, global position, battery level and, status (if it is available, executing a mission, or in failure). Mission status is also available, informing the mission owner and the agent who will execute it, including estimated time to execute the mission, deadline, and the execution time. There is also a terminal window to verify everything printed at the global terminal window. Figure 18 exemplifies a web page provided by this module.

4.2 SPECIAL AGENTS

HeMuRo Framework has a set of agents responsible for maintaining some of the essential functions of the framework. These agents can also be used as an example to create other agents.

4.2.1 Logger Agent

The Logger agent handles the communication between the MRS and the end-user. By default, both Logger modules are initiated. It has three fundamental blocks: the first one prints all agents' messages at the terminal; the second one is responsible for storing messages in a text file. The last one acquires data and displays it on a web page for better visualization.

Enabling this agent is optional for every MRS. If this agent is not enabled, there will be no way of gathering information from the system. It is also important to remind that multiple graphic stations can be multiple logger agents under the same name

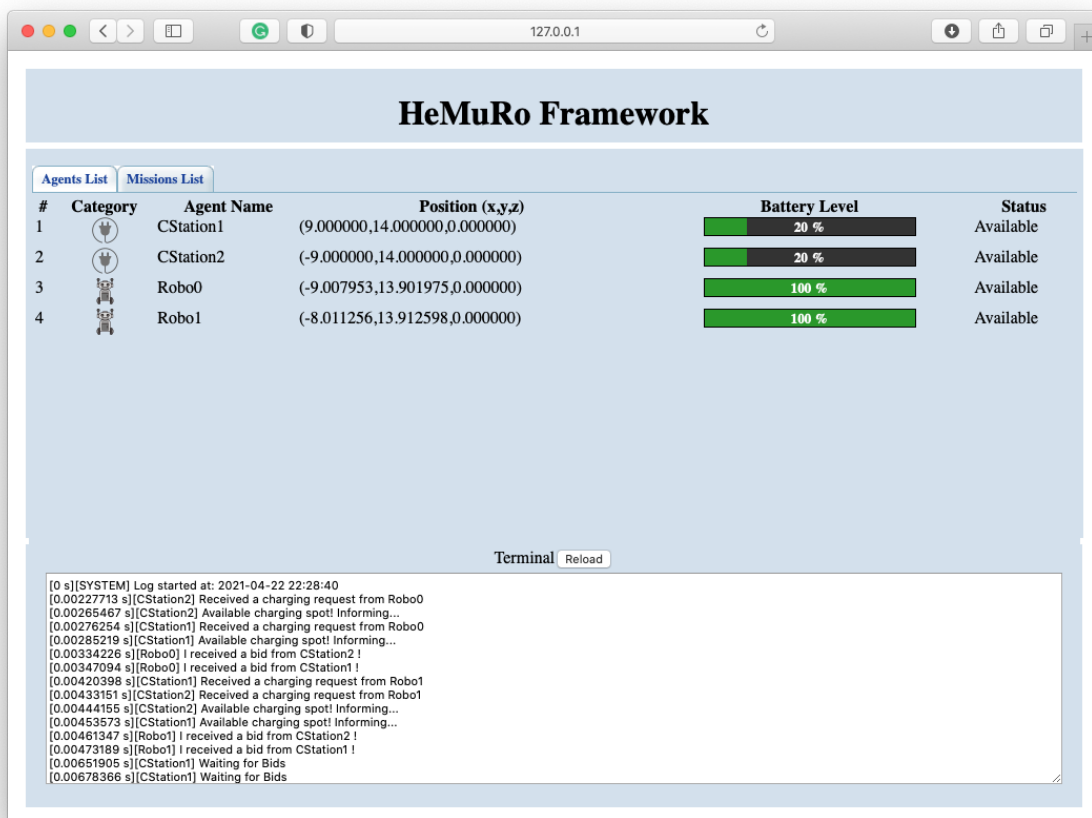


Figure 18 – Example of the web app by the Logger agent

"logger". This is the only case possible for having multiple agents under the same name, and all the agents will behave equally.

4.2.2 Charging Station

The Charging Station agent offers a place for physical agents to recharge its battery. Every Charging station has one or multiple chargers. They are called Charging Spots, and they have their own physical limitations working for some robot's category.

The recharging process starts with a charging request from an agent. This process is similar to the auctioning presented by the Mission module and the request includes the agent's position and its category. After the first request, an auction decides which of the Charging Stations available suits the best for this occasion.

The winning bid is selected by the distance between the agent and the Charging Station, and a Charging Spot for the agent at the selected Charging Station is reserved. The agent moves to the selected spot and starts charging. It is important to remember that after the charging is complete, the agent must free the charging spot so other agents can use it.

4.3 ROBOTS

Some robots were included as models to execute the examples provided by the author. Three robots are already compatible with some implemented atomicTasks.

4.3.1 UGVs

Two UGVs are implemented and they are ready to use. Both robots have the Light Detection And Ranging (LIDAR) sensor to map the environment and to identify where they are in the environment. ROS was used for localization, path planning and controlling the robots. Gmapping package and Navigation stack were used to move the robot across the environment.

Both robots don't provide a battery simulator. Therefore, a battery simulator module (SILVA BARBOSA, 2021) was used to simulate the behavior of a physical battery for each robot.

4.3.1.1 ROSbot 2.0

ROSbot 2.0 model (HUSARION, 2021) is an educational, autonomous, open-source platform based on ROS. This 4x4 drive robot is equipped with LIDAR, RGB-D camera, IMU, encoders, and distance sensors.

4.3.1.2 Pioneer 3DX

Pioneer 3DX is also an educational, autonomous, open-source platform based on ROS. It is a two-wheel two-motor differential drive robot, it comes with sonar, wheel encoders, and it is possible to attach other devices (ADEPT TECHNOLOGY, 2011).

4.3.2 UAV: COEX Clover

The UAV model implemented is the COEX Clover. Clover is an educational kit of a programmable quadcopter. It has a Raspberry Pi 4 as a controlling onboard computer, a camera module, distance sensor, GPS and other sensors (COEX, n.d.). It has open-source software and documentation. The flight controller has the PX4 flight stack, and it is integrated with ROS.

This UAV doesn't provide any LIDAR for obstacle avoidance. The navigation of Clover drone is provided by an algorithm developed by COEX. The algorithm moves the robot in straight line to the goal.

Before using this model, some configuration must be done. Some parameters must be altered. As default, the PX4 estimator parameters do not include GPS fusion, changing the EKF2_AID_MASK parameter to 3 enables the GPS fusion for a more precise position.

The battery simulator is not activated by default as well. By default, the battery only drains a bit and it won't be completely without charge. Altering the parameter `SIM_BAT_DRAIN` to 300s means that the battery will last for 300s when the drone is on air. After landing the battery will recharge automatically. With this behavior, it was set a basis for the drone, it will execute its mission and then it will go back to the basis for recharging. Another parameter altered was `SIM_BAT_MIN_PCT` to 0. This sets the lower boundary of the battery.

A complete overview of the modules is presented in this Chapter, along with some robot models implemented. However, another critical step of this work is to test the framework with different scenarios. This will be cover in the next Chapter.

5 EXPERIMENTAL ENVIRONMENT AND RESULTS

This chapter presents how to implement different scenarios for MRS using HeMuRo Framework. A brief description of the simulations is introduced, followed by implementing atomicTasks used by each robot.

After implementing all atomicTasks, the declaration of each model of the robot is required, including all their decomposableTasks. The following step is to configure the environment by creating the main program, instantiating all agents and offered missions.

Three scenarios with different characteristics are presented below. Each scenario has its particularities, using other robots and missions.

- Empty world

In this simulation, it will be considered the use of the HeMuRo Framework with no simulator or framework communicating with it. Here, an empty map will be used. There will be no collision avoidance in this simulation because there are no physical constraints to the robots.

The only model used in this simulation scenario will be a general robot model. The robots must pick up and deliver samples, inspect areas by taking pictures and measure the temperature of selected spots.

- Hospital World

The scenario chosen for this simulation is the Hospital World provided by AWS Robotics. In this simulation, there will be two models of UGVs, and the robots must pick up and deliver samples of different sizes, inspect rooms by taking pictures and measure the temperature of selected rooms. As the robot models have different physical attributes, they will perform different missions.

- Small Warehouse World

The scenario chosen for this simulation is the Small Warehouse World provided by AWS Robotics. In this scenario, there will be two simulation scenarios. The first one there will be one UAV and one UGV working together. The UAV will inspect areas by taking pictures from above, and the UGV will pick up and deliver objects. The second simulation scenario will have two UGVs performing the same missions as the previous simulation. The UGVs will inspect areas by taking pictures and will also pick up and deliver objects.

The auctioning process of all three environments will have the same configuration. The communication max response will be 1 second, and the bidding time will be 5 seconds. The Charging Station 01 will offer all tasks for each scenario. With these

scenarios established, a description of atomicTasks and implementation are proposed in the following section.

The ROS workspace is available on (DA FONSECA BRAGA, 2021b). For a better comprehension Figure 19 presents a tree with the files in the ROS workspace repository.

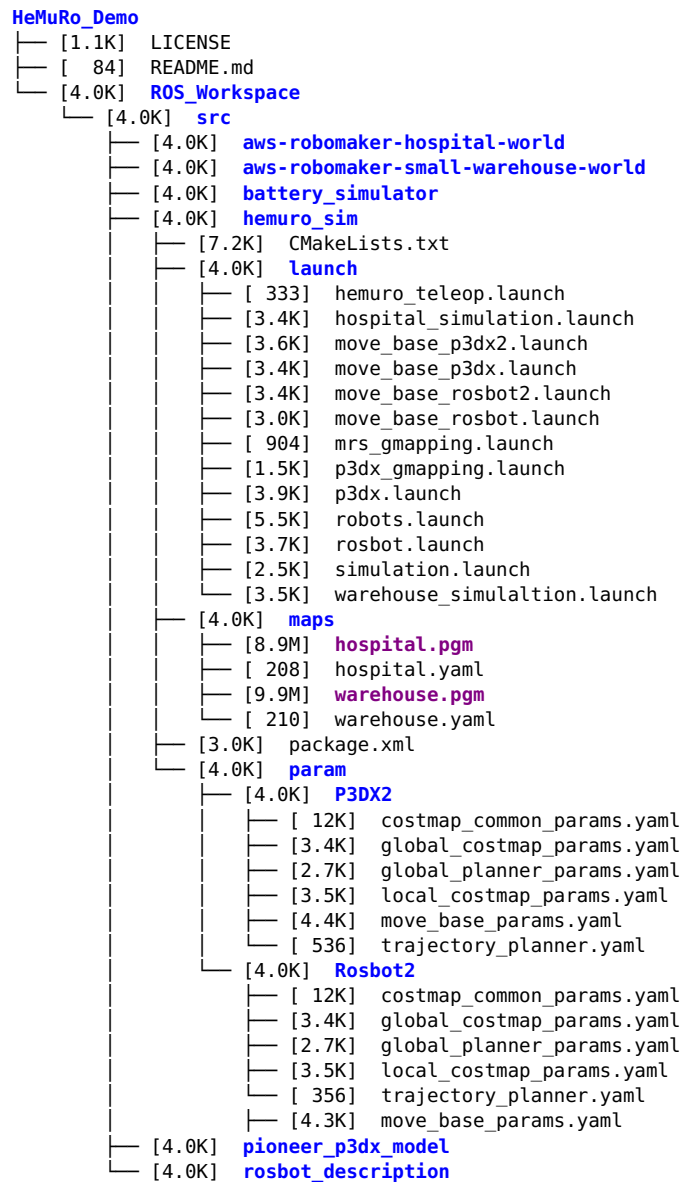


Figure 19 – HeMuRo_Demo Repository Tree

The HeMuRo Framework files including the atomicTasks, decomposableTasks, and robots' implementations are available on (DA FONSECA BRAGA, 2021a). Figure 20 shows a tree with all HeMuRo files.

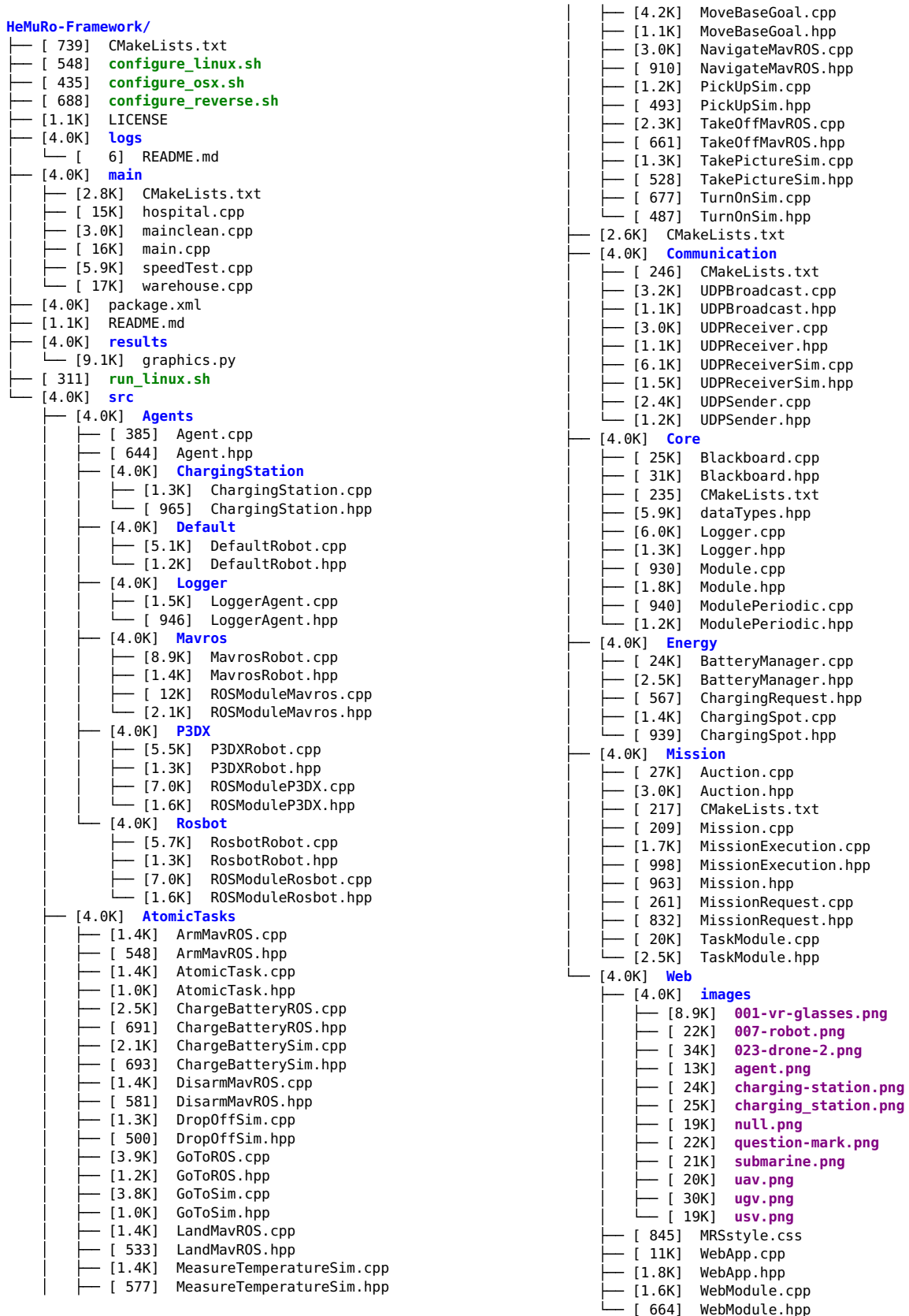


Figure 20 – HeMuRo Framework Repository Tree

5.1 ATOMICTASKS IMPLEMENTATION

The simulation contains four models of robots: a general UGV robot, the Pioneer 3DX, the Husarion Rosbot 2.0, and the COEX Clover. Implementing the atomicTasks will be split into three groups: (i) the general robot; (ii) the UGVs, comprehended as Pioneer 3DX and Husarion Rosbot 2.0; (iii) the UAVs, represented by the COEX Clover.

5.1.1 General Robot

Implementing the atomicTasks for this robot does not involve any integration with another framework, for example, ROS. All atomicTasks are simulated by printing messages at the terminal and changing values at the shared memory of each agent. This simulation can execute missions and debug them using the messages written on a text file or using the web application and generating graphics with the results.

The first step defines the missions and decides which atomicTasks will be implemented for this robot category. This first simulation is simple, and it intends to show some of the basic features of a robot. Given an environment, the robot receives missions to inspect places by taking pictures, measuring temperature, picking up samples from a spot, and delivering it to another spot, and charging the battery when needed.

Based on the explanation above it is easy to identify and create some atomicTasks: `GoToSim`, `MeasureTemperatureSim`, `PickUpSim`, `DropOffSim`, `TakePictureSim`, and `ChargeBatterySim`. The term `Sim` was added at the end of each atomicTask to identify that these atomicTasks are simulated. Each mission is composed by one decomposableTask: `inspectPlace`, `deliverSample`, `measureTemperature` and `lowBattery`. Table 1 contains all decomposableTasks with their atomicTasks sequence.

Table 1 – Missions available for the first simulation

Mission Name	Task Decomposition Sequence
Deliver Sample	<code>GoToSim</code> , <code>PickUpSim</code> , <code>GoToSim</code> , <code>DropOffSim</code>
Inspect Place	<code>GoToSim</code> , <code>TakePictureSim</code>
Measure Temperature	<code>GoToSim</code> , <code>MeasureTemperatureSim</code>
LowBattery	<code>GoToSim</code> , <code>ChargeBatterySim</code>

The first atomicTask implemented is the `measureTemperatureSim`. The atomicTask prints a message that the robot is measuring the temperature and waiting for the task to be completed. The configured parameters are the cost and time to execute the task. They are configured by setting the parameters `costFactor` and `timeFactor`. Each related function will calculate the complete cost and time. By default, the cost and time will be equal to their factors. If there are any particular equations to provide cost and time, they must be implemented.

PickUpSim, DropOffSim and TakePictureSim have similar implementation as the MeasureTemperatureSim. The difference between these atomicTasks are the message printed, cost and time. Table 2 presents the cost for each AtomicTask, the time to execute it and also the message printed by each one. The cost and time were selected arbitrarily.

Table 2 – Parameters of simulated atomicTasks

AtomicTask	Cost [%]	Duration [s]	Message
MeasureTemperatureSim	1	1	Measuring the Temperature.
PickUpSim	1	2	Picking up Sample.
DropOffSim	1	2	Dropping off Sample.
TakePictureSim	1	1	Taking a Picture.

The AtomicTask `chargeBatterySim` has a particular behavior. It will charge the battery of the robot after each execution cycle. It is necessary to get how many execution cycles will be necessary until the battery is full to estimate how long this process will take.

The last atomicTask implemented for this MRS is the `GoToSim`. For this implementation, a simple control was implemented to make the robot go from a position to another. The cost is calculated by obtaining the distance between the start point and endpoint and multiplying this value with the cost factor. The cost factor is calculated by how much power will be drained pro meter. This value was assumed to be linear to simplify the calculus.

After implementing the atomicTasks, the next step is to define a robot model that will decompose missions into atomicTasks and execute them. By default, modules do not start their main task at the moment of declaration, so they must be started. The task decomposition must also be programmed in this file.

The function `addAtomicTask()` contains the task decomposition algorithm. There is also a consistency check-up in this function, and it returns true if everything is all right or false otherwise. The `decomposableTaskList()` method contains the sequence of atomicTasks for every decomposableTask. This method will add to the robot's shared memory, which tasks the robot can execute.

By doing this configuration, the robot will be ready to decompose missions and execute them.

5.1.2 The UGVs

Both UGVs used in the simulation are based on ROS. Therefore, a ROSModule configured for each type of robot is required among the atomicTasks.

The UGVs are quite similar in actions, they work with ROS, and they have similar topics to acquire information and to be controlled. The ROSModule will acquire

information on position and battery level and send information to recharge or move the robot through the map.

If an atomicTask needs to perform an action or acquire any information that depends on ROS, it needs to communicate through this module. Table 3 contains all the topics, services and actionServers used by ROSBot 2.0.

Table 3 – ROSModuleRosbot

Name	Type	Description
/odom	subscriber	Robot's Odometry
/battery/percent	subscriber	Battery level in percentage
/cmd_vel	publisher	Control the speed of the robot
/battery/recharge	publisher	Boolean to start charging the robot
/move_base	ActionServer	Set a goal to the robot
/move_base/make_plan	service	Returns a plan

Concerning the missions, these two robots will be like the simulated robots. However, they will be executed in a Gazebo scenario. Table 4 contains all Tasks that will be executed for each type of UGV robot.

Table 4 – Available Tasks to be executed: Robots eligible to execute

Task Name	Task Decomposition Sequence	Robot
DeliverSmallSample	MoveBaseGoal, PickUpSim, MoveBaseGoal, DropOffSim	P3DX and ROSbot
DeliverBigSample	MoveBaseGoal, PickUpSim, MoveBaseGoal, DropOffSim	P3DX
InspectPlace	MoveBaseGoal, TakePictureSim	P3DX and ROSbot
MeasureTemperature	MoveBaseGoal, MeasureTemperatureSim	ROSbot
LowBattery	MoveBaseGoal, ChargeBatteryROS	P3DX and ROSbot
InspectArea	arm, takeOff, [MoveBaseGoal, takePictureSim](5x), MoveBaseGoal	ROSbot

To differ between the two robot models, P3DX will carry bigger objects than the ROSbot model, and the ROSbot will measure the temperatures of environments. Some atomicTasks previously defined will be used as well. *MeasureTemperatureSim*, *PickUpSim*, *DropOffSim*, and *TakePictureSim* were already defined in Table 2. The task *InspectArea* was defined to replace the same mission that the UAV will perform in the Warehouse Simulation. The robot will move through five areas taking pictures returning to a basis.

Gmapping will be used to generate a map of the environment. The Navigation Stack will be used for localization, planning where to go, and execute the path across the

map. Considering that multiple robots will be simulated on a single computer, and if each one uses its SLAM or Adaptive Monte Carlo Localization (AMCL) module, the computer could not provide sufficient power to simulate everything. Instead of using SLAM or AMCL robot's localization, Fake_Localization (ROS, 2020a) from Navigation Stack is used instead. The Fake_Localization simulates the behavior of a localization algorithm by acquiring the robot's position direct from Gazebo, reducing the computational power to obtain the robot's position.

A new atomicTask is implemented `MoveBaseGoal`. This atomicTask will send a message containing the destination to the ROSModule. The ROSModule will send the goal through an actionServer which will request a trajectory to the Navigation Stack and execute it. Implementing both robots is equal. However, the configuration of cost and time to execute will be different. Those parameters will be adjusted later.

The battery recharging process is also different from the previous category of robots. The atomicTask `ChargeBatteryROS` was designed to send a boolean value through the ROSModule to recharge the robot. The Battery capacity of each robot was set with a low capacity to force the robots to recharge several times during the simulation. Before starting the simulations, the battery level was also set around 60% to force the robots to recharge.

The next step is to create two types of robots, as done with the simulated robot model. `P3DXRobot` and `RosbotRobot` contains the declaration of modules and decomposition of decomposableTasks into atomicTasks. These files are useful for creating over one robot of the same type. Exclusive parameters for each type of robot are also declared (E.g. `costFactor` and `timeFactor` personalized for each model).

5.1.3 The UAVs

The UAV implemented for the simulation is the COEX Clover model. Differing from the UGVs, the UAVs need some extra atomicTasks to move from a place to another. To complete a flight successfully, the UAV must arm the motors, take off, go to the destination, and then land at the position, disarming the motors.

The Clover UAV is also based on ROS and a ROSModule was created. Table 5 contains all the topics and services integrated with this ROSModule, allowing the UAV to be controlled by the framework. There are some basic services as arm, land and disarm the robot and also the command to navigate, where the user needs to send the destination to the UAV. The module also subscribes to get some feedback of position, battery level and motors' state.

COEX provides a service for ROS called `navigate()`. This service is responsible for flying the UAV from a start point to a destination. It is important to remark that this service does not have an obstacle avoidance or trajectory planner. This service moves the robot as if nothing is preventing the UAV from achieving its position. The robot will

Table 5 – ROSModuleMavros

Name	Type	Description
/mavros/state	subscriber	Return if the UAV is armed or not
/mavros/global_position/local	subscriber	Global position
/mavros/battery	subscriber	Battery Level in percentage
/navigate	service	Navigate the UAV from a place to another
/mavros/cmd/arming	service	arm the UAV
/mavros/cmd/land	service	land the UAV
/mavros/cmd/disarming	service	disarm all motors

fly above the obstacles to avoid colliding with them. Table 6 contains all atomicTasks for the UAV.

Table 6 – AtomicTasks available for the Clover robot

atomicTask Name	Description
armMavROS	Arm all motors
NavigateMavROS	Navigate from current position to a goal
LandMavROS	Land the UAV and disarm motors
TakeOffMavROS	Take off to a specific high
DisarmMavROS	Disarm all motors
TakePictureSim	Take a picture (simulation)
ChargeBatteryMavROS	Do nothing and wait until battery is charged

The same process to create the robot type is done for the Clover UAV. All modules were declared: communication, core, debug, energy, mission, ROSBridge and task modules.

The next step is to provide information about the missions that the UAV will perform. The information focuses on the auctioning process and all the decomposition of the decomposableTasks presented in Table 7 were described so the UAV could place a bid during the auctions.

The robot can inspect places in two different ways: the first will be a single spot inspection and the second will inspect five areas of the map in the same mission. This longer task was created to take advantage of the speed of the UAV and to optimize the inspection of multiple areas.

UAVs need to have a behavior to land in case of an emergency otherwise, they will fall from the sky. This was not the case for UGVs because when there is an emergency the grounded robots, at worst case, will stop where they are. To cope with these problems an emergency behavior was also implemented.

With all atomicTasks created and robot models adequately described, the next step is configuring the environment and executing the simulation.

Table 7 – Available Tasks to be executed

Task Name	Task Decomposition Sequence	Description
CheckPosition	arm, takeOff, goTo, goToBasis, land	Visit a position on map
InspectArea	arm, takeOff, [goTo, takePictureSim](5x), goToBasis, land	Take a picture of 5 areas
TakePicture	arm, takeOff, goTo, takePictureSim goToBasis, land	Take a picture of an area
LowBattery	arm, takeOff, goTo, land, chargeBattery, arm, takeOff, goToBasis, land	Go to recharge and return to basis
EmergencyLanding	land	Land in case of an emergency

5.2 SIMULATED ENVIRONMENTS

To exemplify the use of HeMuRo Framework, three simulations using different scenarios were created: the first simulation runs only with a HeMuRo Framework; the second simulation uses the Hospital World; the last simulation uses the Small Warehouse World.

5.2.1 Empty Environment

For this first simulation, there is no physical environment configured. The robots are instantiated and missions are created. Spots on a Cartesian map and a charging station agent are created as well.

The Charging Station is the agent responsible for creating missions and offering them to the robots. This brief example, will instantiate only one mission of each kind, and two robots will execute them. Table 8 contains the offered missions with their relative deadlines.

Table 8 – Available Tasks to be executed and Deadline to complete the mission

Task Name	Task Decomposition	Deadline [s]
DeliverSmallSample	MoveBaseGoal, PickUpSim, MoveBaseGoal, DropOffSim	100
DeliverBigSample	MoveBaseGoal, PickUpSim, MoveBaseGoal, DropOffSim	100
InspectPlace	MoveBaseGoal, TakePictureSim	50
MeasureTemperature	MoveBaseGoal measureTemperatureSim	50

The configuration process to execute the first simulation is done. The results and analyses will be presented in the next section after describing all simulated environ-

ments.

5.2.2 Hospital World

The scenario chosen for this simulation is the Hospital World provided by AWS Robotics. This environment, if full of objects unnecessary to the execution of this example, e.g., curtains, beds, and objects inside the rooms in the back of the building. The number of extra objects in the scenario was reduced to increase the simulation speed and real-time factor. Two robots are used for this simulation: Rosbot 2.0 and Pioneer 3DX.

Figure 21 presents a Two Dimensional Space (2D) visualization of the map. It was defined 14 positions in the Hospital World. Two of these positions are charging stations located inside the storage rooms.

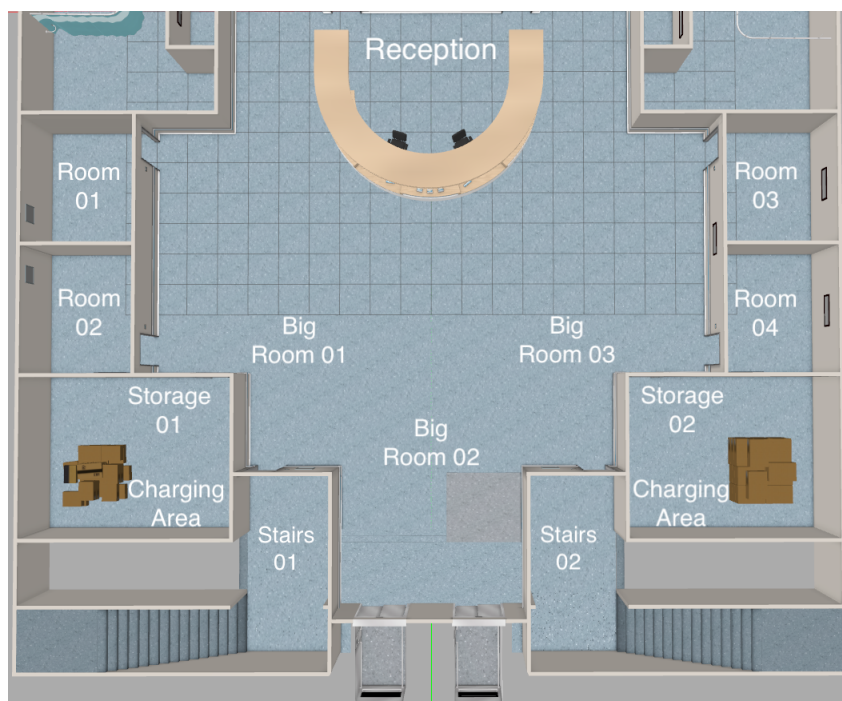


Figure 21 – Simulation of the hospital world with defined positions tagged

The two robots have different sizes and physical limitations. In this simulation, the ROSbot will move faster than the Pioneer 3DX but, Pioneer 3DX will carry bigger objects than the ROSbot. The average speed for each robot was calculated to estimate how long it will take to finish the mission. Since we didn't have this value, it was estimated by taking the duration between moving across several places on the map.

In this simulation, four missions will be offered to the robots: `inspectPlace`, `DeliverSmallSample`, `DeliverBigSample`, and `MeasureTemperature`. Table 9 contains all tasks available in this MRS, their decomposition, which robot can execute it, and the maximum time to execute it. Note that the missions are practically the same as the first simulation. They differ on the restrictions by robot type.

Table 9 – Available Tasks to be executed: Robots eligible to execute and Deadline to complete the mission

Task Name	Task Decomposition	Robot	Deadline [s]
DeliverSmallSample	MoveBaseGoal, PickUpSim, MoveBaseGoal, dropOffSim	P3DX and ROSbot	100
DeliverBigSample	MoveBaseGoal, PickUpSim, MoveBaseGoal, dropOffSim	P3DX	100
InspectPlace	MoveBaseGoal takePictureSim	P3DX and ROSbot	50
MeasureTemperature	MoveBaseGoal measureTemperatureSim	ROSbot	50

As an example of how the framework can easily increase the size of robots and missions, this simulation will run several times. The number of robots will vary from 2 to 4 robots divided equally between the two models. The number of tasks will be divided equally: 8, 16, and 32 tasks. It will be asked to `inspectPlace` at the `BigRoom01` and `BigRoom03`; `DeliverSmallSample` from `Room01` to `Reception` and from `Room03` to `Reception`; `DeliverBigSample` from `Room01` to `Reception` and from `Room03` to `Reception`; and to `MeasureTemperature` at `Room02` and `Room04`. The relation of tasks and goals is described in Table 10. All data will be gathered using the `Logger` agent.

Table 10 – Available Tasks to be executed: location in the map

#	Task Name	From	To
1	<code>InspectPlace</code>	-	<code>BigRoom01</code>
2	<code>InspectPlace</code>	-	<code>BigRoom03</code>
3	<code>DeliverSmallSample</code>	<code>Room01</code>	<code>Reception</code>
4	<code>DeliverSmallSample</code>	<code>Room03</code>	<code>Reception</code>
5	<code>DeliverBigSample</code>	<code>Room02</code>	<code>Storage01</code>
6	<code>DeliverBigSample</code>	<code>Room04</code>	<code>Storage02</code>
7	<code>MeasureTemperature</code>	-	<code>Room02</code>
8	<code>MeasureTemperature</code>	-	<code>Room04</code>

In the environment configuration file, the user should insert the number of missions and robots it will execute and run the simulation environment and the robot simulators.

5.2.3 Small Warehouse World

The last simulation also includes a scenario provided by AWS Robotics. The warehouse scenario contains common objects available in a warehouse. Figure 22 presents all the available positions inside the warehouse facility.

In this scenario, two robots were used: the `ROSbot` and the `COEX Clover`. Some minor modifications to the scenario were also implemented. For example, the shelves

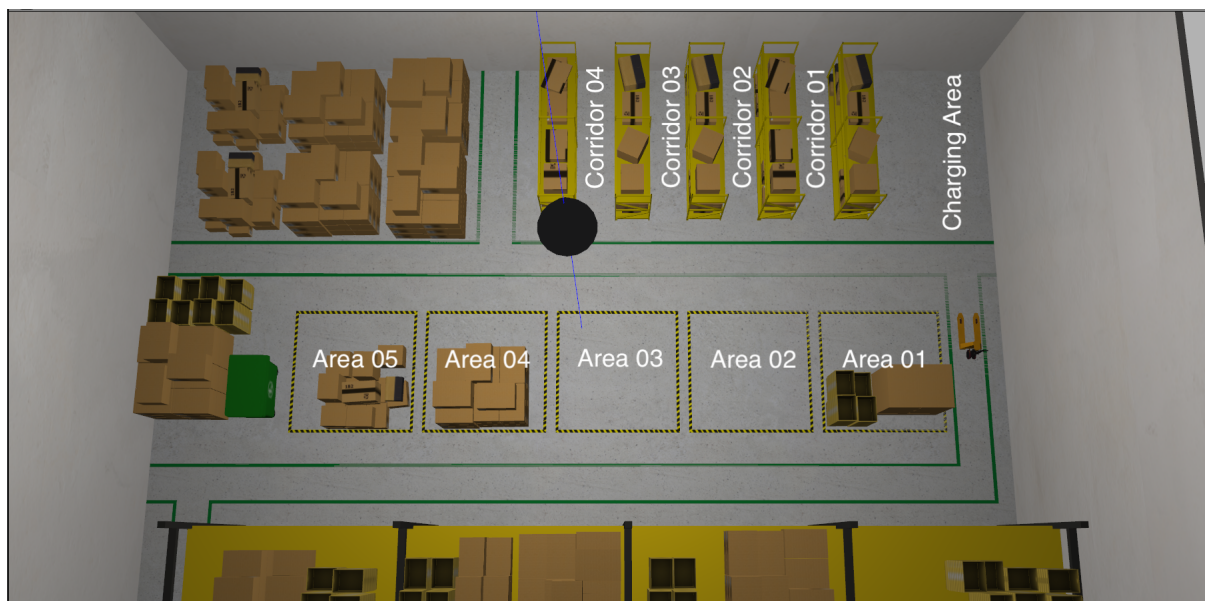


Figure 22 – Simulation of the small warehouse world with defined positions tagged

were too high compared to the robot size. Here, the laser could not identify the shelves, a workaround was to reduce their height. Another modification was to remove one of the shelves to add the charging area that will also work as basis for the UAV. In this scenario both robots will start with a full battery.

For the first simulation, an instance of each model was created. The UAV will inspect some areas of the warehouse and the UGV will pick and deliver some packages. Table 11 contains all the offered missions with the deadline and also the restrictions by the robot's model.

Table 11 – Available Tasks to be executed: Robots eligible to execute and Deadline to complete the mission

Task Name	Task Decomposition	Robot	Deadline [s]
InspectArea	arm, takeOff, [goTo, takePictureSim] (5x), goToBasis, land	Clover	380
DeliverSmallSample	MoveBaseGoal, PickUpSim, MoveBaseGoal, dropOffSim	ROSbot	100

During the simulation, there will be four missions offered, two of each type. They will be offered simultaneously at the beginning of the simulation.

As for the locations on the map, the UAV will patrol moving from area01 to area05 and taking pictures of them. The UGV will pick products from the shelves in corridor01 to corridor03 and will deliver them on the area02. Table 12 contains all the positions where the tasks will be executed.

In the second simulation, there will be two ROSbots available. The offered missions will be the same as the previous simulation. The number of offered missions is

Table 12 – Available Tasks to be executed: location in the map

#	Task Name	From	To
1	Inspect Area	-	Area01, Area02, Area03, Area04, Area05
2	DeliverSmallSample	Corridor01	Area02
3	DeliverSmallSample	Corridor02	Area02
4	DeliverSmallSample	Corridor03	Area02

the same as before, with four missions comprising two of each type. These missions are displayed at the beginning of the simulation.

In this case, there will be only UGVs available to perform the missions. Table 13 contains all the offered missions with the deadline and also the restrictions by robot's model. The `InspectArea` mission was created to simulate the UAVs mission. A last `MoveBaseGoal` atomicTask was added to simulate the `goToBasis` atomicTask performed by the UAV.

Table 13 – Available Tasks to be executed: Robots eligible to execute and Deadline to complete the mission

Task Name	Task Decomposition	Robot	Deadline [s]
InspectArea	[MoveBaseGoal, takePictureSim] (5x), MoveBaseGoal	ROSbot	380
DeliverSmallSample	MoveBaseGoal, PickUpSim, MoveBaseGoal, DropOffSim	ROSbot	100

All the locations configured for this second simulation will be the same as the previous simulation presented on Table 12.

This chapter presented three possible scenarios to execute MRS using the HeMuRo Framework: one simulation running the framework alone and two other simulations integrated with ROS. The next chapter contains the analysis of the execution of the simulations, including some comments.

5.3 SIMULATION AND ANALYSIS

This section analyses the HeMuRo Framework and simulations concerning the different scenarios and the analysis of the results with some comments and remarks.

The first simulation will evaluate the GUI of the framework. The hospital world environment is the second simulation and contains multiple combinations of robots and missions. Here, the analysis is done by the execution of the missions and the feature of generating graphical reports. The last simulation focuses on the warehouse world environment and analyses running different categories of robots together.

5.3.1 First simulation: Empty Scenario

The first simulation did not have an integration with a graphical simulator. The web interface generated by the logger agent provided information about the execution of the missions.

The web page displays information in real-time about the simulation. Figure 23 shows the main page that provides information about the current state of each agent and their positions. The robot's category is also available on the main page.

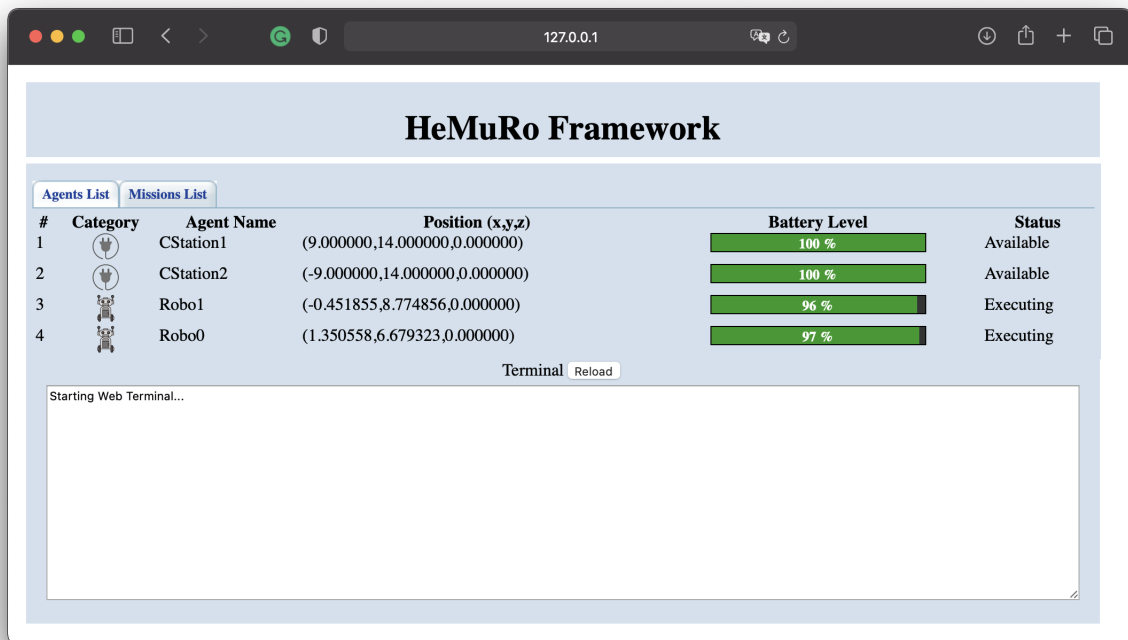


Figure 23 – In the web interface it is possible to debug the current status of each agent.

In this screenshot taken in the first simulation, both robots were executing missions, and the battery level of both robots was almost fully charged. As for the agent's categories, for this simulation, there were two charging stations and two UGV robots. It is also possible to follow the terminal messages with the log of messages sent by the agents.

Figure 24 refers to the second tab of the main page. It displays information regarding the missions. It's shown, two missions were on execution, and two were waiting for allocation. For the ones with executing status, there is also the estimated execution time. After completing one there will also be the time taken to execute it. It is also possible to follow the terminal messages.

The following subsection will discuss other framework features, for example, graphics and reports, to obtain better data visualization.

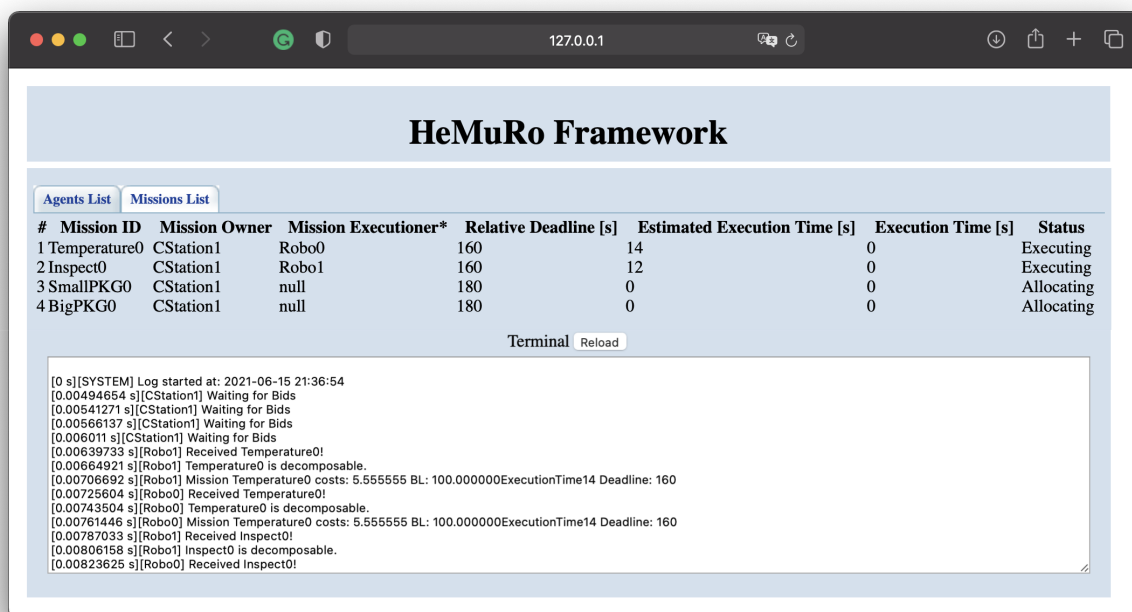


Figure 24 – In the web interface it is possible to debug the current status of each mission.

5.3.2 Second Simulation: Hospital World

The analysis of the second simulation focuses on performing the task allocation and execution of missions. With the robot models configured, it was easier to scale the system. Having the simulated model launched in Gazebo, we only need to create more instances of the robots on HeMuRo Framework.

Although two charging stations were created only the first one will offer missions to the robots. The second charging station will have its essential function: recharge the robots. All missions contains one decomposableTask.

Six simulations were conducted varying the number of robots and missions. Three simulations run with two robots offering eight, sixteen and thirty-two missions. The other three simulations run with four robots offering eight, sixteen and thirty-two missions. Simulations using two robots have one Pioneer 3DX robot model called Afrodite, and one ROSBot robot model called Thor. Simulations using four robots contain two models of Pioneer 3DX, called Afrodite and Athena, and two models of ROSBots, called Thor and Zeus.

Figure 25 shows the results of the first simulation. Two robots executed eight offered missions. There were two charging requests, one for each robot. This behavior can be explained by the fact that the robots did not started with a full battery. All robots were configured to start with around 60% of battery level to force at least one charging per simulation.

As the reader can expect for this first simulation, the tasks were equally divided

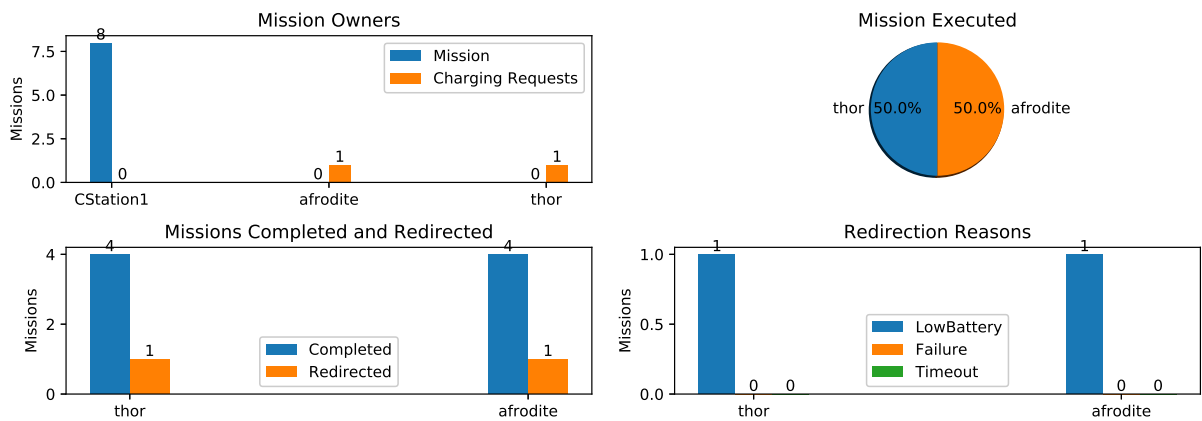


Figure 25 – Simulation with 2 robots and 8 missions

between the two robots. From the eight offered missions, Thor could execute six of the offered missions and so does Afrodite. They both executed their two exclusive missions plus two more tasks.

Figure 26 describes the second simulation. The same two robots executed sixteen missions offered by the Charging Station. There were three charging requests, two made by Afrodite, and one by Thor. In this scenario, Afrodite had to redirect its missions twice due to a low battery. The energetic cost estimation to execute a mission is not always accurate. In these simulations, it was decided not to have a precise algorithm to predict the cost because the main goal was to observe the cost estimation failing and tasks being redirected. Thor didn't redirect its mission to recharge, this can be justified by when the battery hit a low level, Thor wasn't executing a mission.

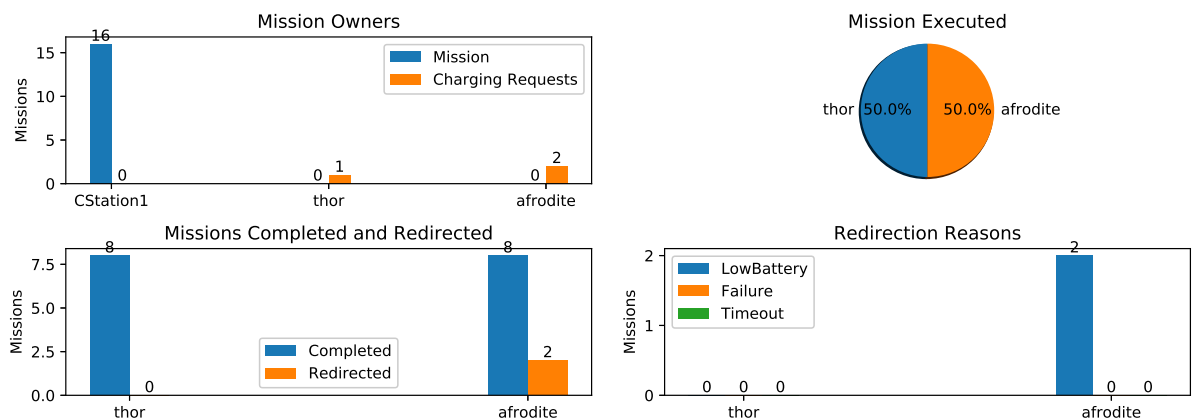


Figure 26 – Simulation with 2 robots and 16 missions

All two robots executed the same amount of tasks. Randomly, the common tasks were executed first and at the end, there were only the specific tasks left and Thor was on standby while Afrodite recharged and finished its tasks.

Figure 27 presents a scenario with more tasks than the first two. 32 missions were offered for two robots. In this scenario, there were 5 charging requests, two made

by Thor and three by Afrodite. In all five cases, the battery hit low level during the execution of a mission, so all five were redirected.

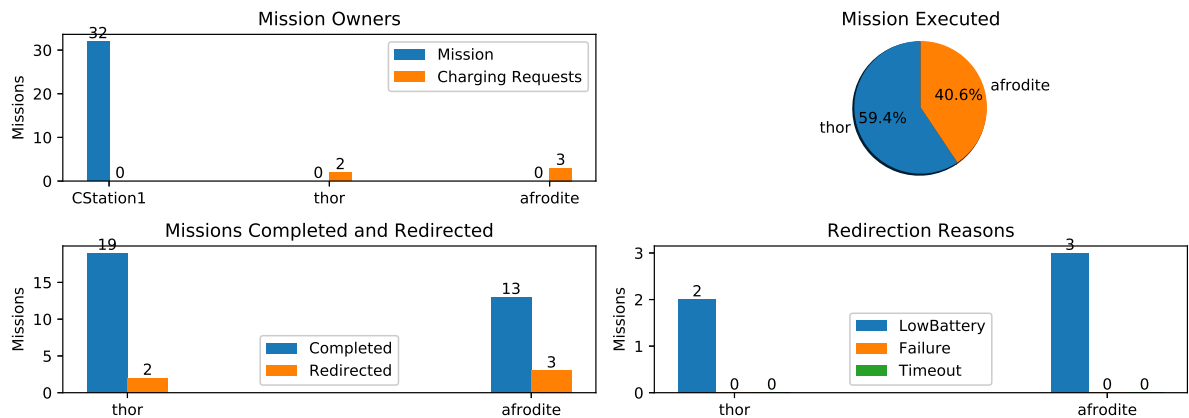


Figure 27 – Simulation with 2 robots e 32 missions

In this simulation, twenty four common missions were being offered. Restricted by its speed, Afrodite moves slower than Thor. With more common missions available, Thor could complete more common missions than Afrodite. Removing the specific missions of the list Thor completed eleven common missions against five completed by Afrodite. With only two robots in the scenario, the probability of getting stuck because of other robots is low. Therefore, low battery was the reason for all mission redirection.

Figure 28 represents the first simulation using four robots. Eight missions were offered and four charging requests were made, one per robot, as usual for fewer missions. In this scenario, Zeus completed more missions than the others, four in total, followed by Thor with three completed missions. Athena and Afrodite completed one mission each.

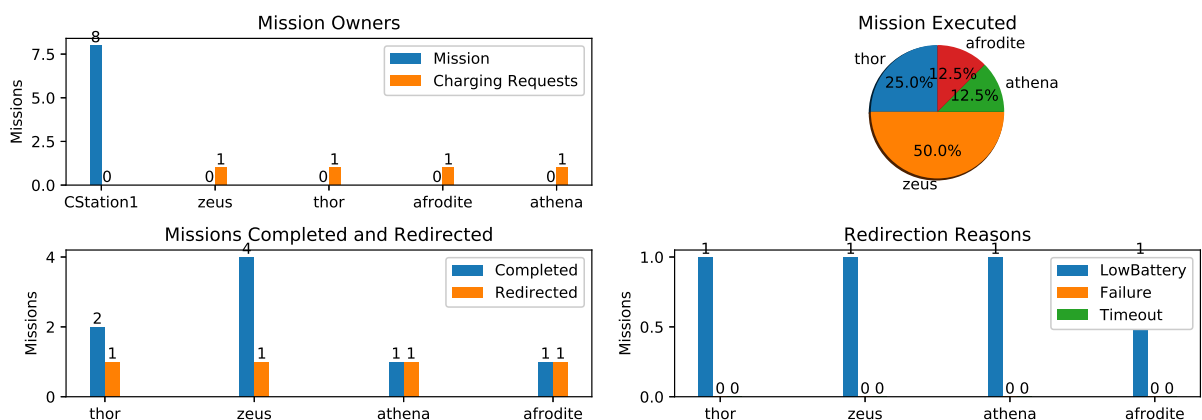


Figure 28 – Simulation with 4 robots e 8 missions

One of the plausible justifications for Thor and Zeus completing more missions than Athena and Afrodite is the ROSBot model's top speed. This can't be affirmed

because there were only four common tasks available. Another justification would be that Thor and Zeus were closer to the goal, offering a cheaper bid. There were four redirected missions due to low battery.

Figure 29 shows the simulation running four robots and offering sixteen missions. In this scenario, Thor and Athena completed five missions each, Zeus with four completed missions, and Afrodite with two completed missions. There were four charging requests, one for each robot.

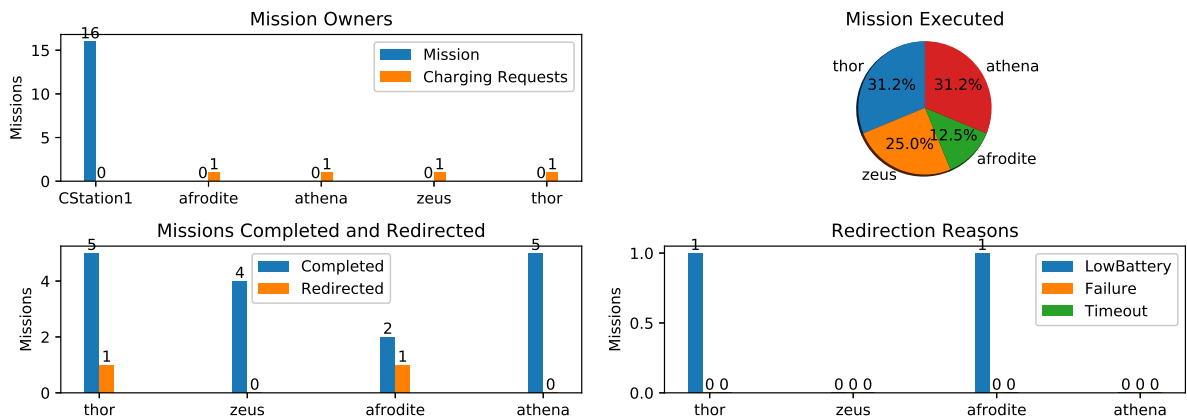


Figure 29 – Simulation with 4 robots e 16 missions

In this simulation, Athena and Zeus were not executing a mission when the battery hit a low level. That's why they didn't redirect their missions. In this simulation Afrodite's path planner got lost for a while, slowing it down a bit, completing only two missions. This situation also happens in real life, the execution time wasn't longer than the deadline, so the mission was completed after a while. Thor and Afrodite redirected two missions due to a low battery.

The last scenario is displayed in Figure 30. In this scenario, there were four robots executing thirty-two missions. Thor completed twelve missions, followed by Zeus and Afrodite with seven completed missions. Athena completed only six missions.

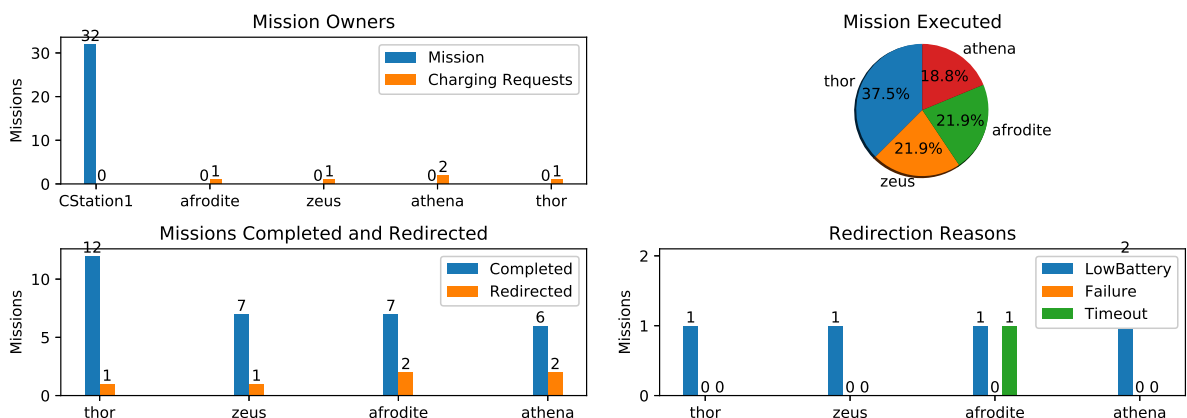


Figure 30 – Simulation with 4 robots e 32 missions

There were five charging requests during the simulation, Athena requested twice, and the others requested once. There were six redirected missions, Afrodite and Athena redirected two missions each. Thor and Zeus redirected one mission each. Five of the redirected missions were related to low battery issues. However, Afrodite redirected one mission due to timeout.

Table 14 contains the duration of each simulation. Simulations with two robots took between 408 seconds and 1519 seconds to be completed. Simulations using four robots took between 311 seconds and 854 seconds.

Table 14 – Duration of each simulation

#	Robots	Missions	Duration [s]	Duration [m:s]
1	2	8	408.856	06:49
2	2	16	841.346	14:01
3	2	32	1519.02	25:19
4	4	8	311.471	05:11
5	4	16	468.589	07:49
6	4	32	854.497	14:14

Comparing simulations regarding the number of missions, the difference between simulations with eight missions using two or four robots is insignificant. One reason is that all robots had to recharge once in both scenarios, increasing the whole simulation's execution time. The difference between the two groups of robots becomes more stressed when more missions are added to the system. With thirty-two missions, the time almost doubled using only two robots compared to the four-robot system. Figure 27 shows that the robots had to recharge two and three times respectively, and Figure 30 two robots recharged only once and the other two recharged twice. Since there are multiple spots for charging, this process can be done simultaneously, improving the efficiency of the MRS. The more available robots can execute the more missions in parallel.

Increasing the number of missions is possible to observe a better distribution of the missions among the robots. It is also noticed that the faster the robot, the more completed tasks. However, this parameter cannot be taken as the only truth because the task allocation is random. A fast robot might get a more extended mission to execute, while a slower robot can get faster missions because it might be near the task goal. In this allocation process, the task owner chooses the best executioner, but it does not mean that the task chosen was the best choice for the executioner. For example, a robot gave two bids in different tasks. If he won the bidding process of both tasks and the answer of the first task arrives before the second one, the robot will execute the first one. This issue can be solved by adding a better strategy from the bidder when giving bids, such as choosing only the near ones, etc. However, this strategy might cause the robot to lose the opportunity of executing a task when the task is too far away. Another

curious fact about the random pick of tasks is that multiple tasks were to be executed in the same place. The same robot did not pick the same task to be executed in a row. For example, to measure the temperature of the same room. If that happened, it could significantly reduce the duration of a task.

Another observation happened in Figure 30 because of a timeout. A reason for this timeout might be justified by the number of robots in the system. With more robots moving around, the robot might have gotten stuck and did not have time to avoid all obstacles until reaching its goal. Another reason for that is that the robot might have got lost. In this case, the task was redirected to another robot to execute it, completing it.

5.3.3 Third Simulation: Warehouse World

The third simulation focuses on two different categories of robots executing tasks in the same environment. For better comparison, two simulations were executed. The first one is a heterogeneous MRS with one UGV and one UAV. The second simulation is a homogeneous MRS, with UGVs.

Figure 31 describes the results obtained in the heterogeneous MRS. Four missions were offered in total: two inspect area missions were described to be performed by the UAV and two deliver small sample missions were described to be executed by the UGV. The battery level of all robots in this scenario started at 100% to avoid a forced recharge and mission redirection as it was done in the Hospital environment. Therefore, there was no charging request.

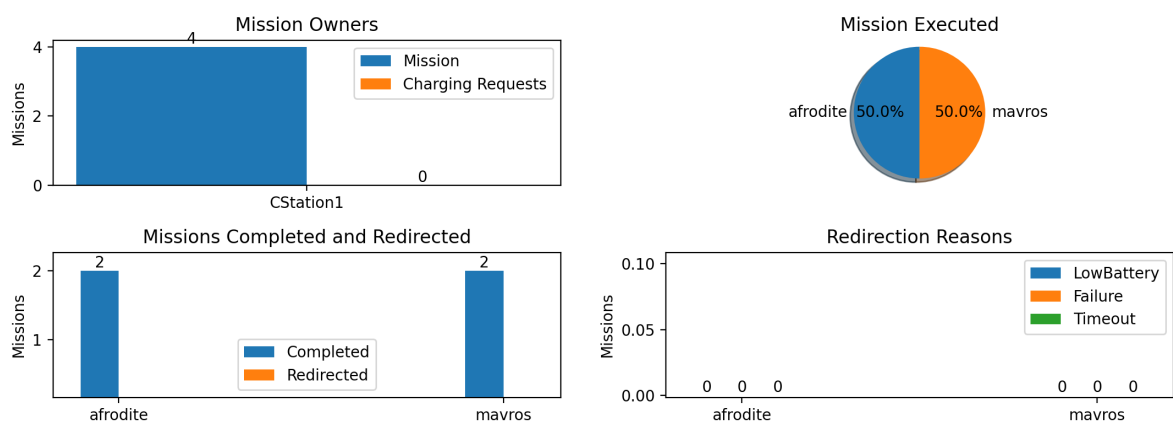


Figure 31 – Simulation with one UAV and one UGV

In the second simulation, represented by Figure 32 there was only a type of robot available. In this simulation, both robots could execute all tasks. Afrodite started by executing the inspect area mission. Since this mission takes longer than the deliver small sample mission, Zeus executed two missions while Afrodite executed one. Zeus finished the two missions and won the auction to execute the last inspect area mission because Afrodite was still busy finishing its mission.

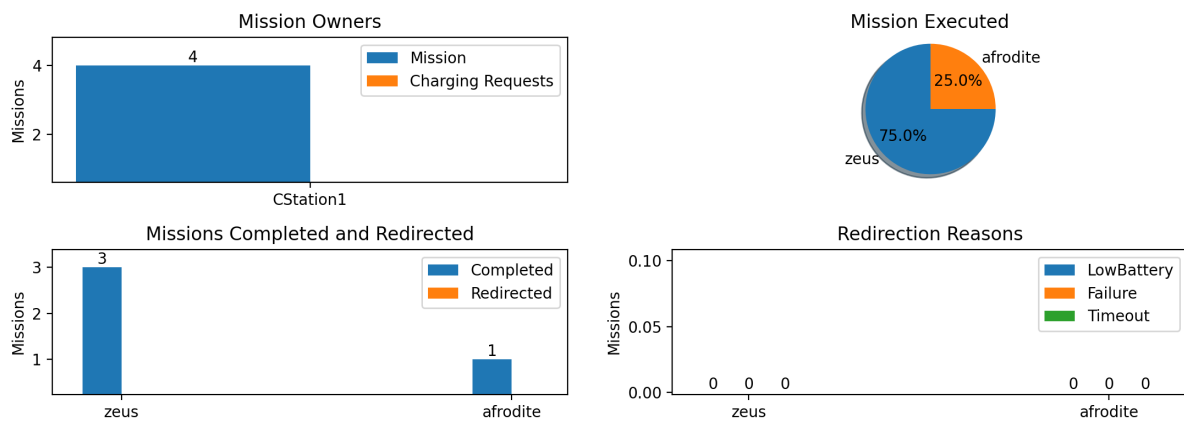


Figure 32 – Simulation with two UGVs

Both simulations finished with no errors and all missions were completed. However, when comparing the time to execute it there were some differences. Table 15 contains the duration of each simulation.

Table 15 – Duration of each simulation

#	UAVs	UGVs	Duration [s]	Duration [m:s]
1	1	1	176.507	02:56
2	0	2	317.459	05:17

The heterogeneous MRS used the best ability of each robot: the UAV can travel faster than the UGV and the UGV can carry objects while the UAV can't, resulting in a faster simulation when compared with the second simulation.

During the inspect area mission, the UGV needed to create a path avoiding the obstacles on the ground such as other robots, pallets and boxes, resulting in a slower performance compared to the same mission performed by the UAV.

Another scenario for the UAV in this simulation would navigate after observing a significant amount of packages in an area, it would create a mission to require a robot to pick all the packages and deliver them somewhere.

This chapter analyzed all three proposed simulations described in Chapter 5. Each simulation was considered a feature to be described to illustrate how flexible this framework is. There is also room for improvements and future work. This will be discussed with the conclusion of this work in the next chapter.

6 CONCLUSION

The development of a MRS involves multiple areas and subjects. This work's proposal aims to provide an open-source framework to be used as a start point for the development of a MRS, fulfilling the lack of open-source tools to simulate and control multiple robots.

This framework's focus consists of task decomposition and allocation according to each robot model, task reallocation in case of failure or an unexpected event, robot coordination incorporating centralized and decentralized information, and real-time restrictions. The framework should work with multiple robot models in different environments. Flexibility and modularity are prerequisites allowing future contributions.

HeMuRo Framework was designed to improve task decomposition and cooperation of robots to execute missions. The architecture of the presented framework was programmed in C++.

Each agent carries an instance of the framework and can communicate with other agents through network messages. It is also possible to broadcast a message containing basic data to inform that the agent is online and working.

The system was designed to work with multiple categories of robots, being capable of assigning missions to different robot models, taking into account their abilities to execute each task. Missions previously described were decomposed into independent tasks to be assigned, called `decomposableTasks`. According to each robot model, the assigned task is decomposed into a sequence of `atomicTasks` to be executed. The process occurs in real-time and therefore is classified as SR-ST-IA.

Besides the instantaneous allocation, the system reacts in real-time as well. Missions can be re-allocated if needed. Each agent can monitor their battery level and in case of low battery level, they can request a spot in a charging station. This behavior assures a certain autonomous level to the robot, including redirecting a mission in case of executing it during a low battery alert. Another example of re-allocation would be if the robot gets stuck or timeout.

A significant amount of information and messages are sent during a simulation or real-time application. Therefore, all systems can be debugged and deployed online with the help of a GUI. A web page displays all the information regarding agents and missions that are available.

Taking into consideration that many robotic applications involve the use of ROS, there is also the possibility of integrating with the middleware. In this work multiple ROS features were used to develop the simulations. Gmapping was used to generate a map for localization and navigation. AMCL and fake localization were used for localization. The Navigation Stack was used to provide a collision free trajectory for UGVs. There is also the possibility of adding or replacing some of the packages used in this work.

Three different simulations were described in this work: (i) a simulation with only HeMuRo Framework, without an environment; (ii) a simulation inside a warehouse facility; and (iii) a simulation inside a hospital. The simulated environments have proven the versatility of HeMuRo Framework and how can use this framework in different scenarios.

In general, the framework performed well as expected. As the results shown in Chapter 5, HeMuRo Framework presented flexibility of working with multiple robot models and different tasks. However, there are some important observations to present: the auction model implemented has some flaws. With a high number of missions being offered simultaneously, there will be many messages being exchanged. When the auctioning process is finished, the top winning bids might already be assigned to another task. This might slow down the allocation process, but it will be completed with some delay.

Another relevant observation is that the mission owner will select the best candidate to execute a mission. This selection is a local decision and it does not take into account a global view. For example, there are two offered missions in a scenario: mission1 and mission2, and two available robots: robot1 and robot2. Robot1 can execute both missions and robot2 can execute only mission2. If the mission owner selects robot1 to perform mission2 because it is the best local option, robot2 will stay on standby and mission1 will have to wait until mission2 is completed. A better case scenario would be mission1 executed by robot1 and mission2 by robot2.

These issues and other observations are presented in the next section as suggestions for future work.

6.1 FUTURE WORK

There is plenty room for improvements on HeMuRo Framework. As it will be an open-source framework the community will also be allowed to contribute. As future work, some tasks can be enumerated:

- There are two ways of assigning tasks implemented: the auctioning process and assigning tasks directly for each robot. Adding different task-allocation algorithms will make the framework more robust and it will make it possible to compare the efficiency and and best method for each scenario and number of robots;
- In MRS there is also the possibility of multiple robots working together to complete a mission. At the moment, in HeMuRo the robots can execute simple missions, where a single robot execute the tasks. Adding the feature to synchronize the execution of tasks, robots would cooperate to execute a mission. An example would be multiple robots carrying a bigger object together;

- The battery prediction model implemented in HeMuRo is based on the distance between the robot and the goal and the energy consumption of the robot. Another improvement would be the implementation of intelligent ways to predict the battery consumption during a mission, analyzing the weight carried by the robot, environmental conditions and also the health of the battery;
- There is no protocol when multiple robots want to access the same spot. During the simulation, a few robots got stuck because another robot was resting at the goal position. Therefore, another improvement is to create a protocol for multiple access to objects or places;
- There are three robot models implemented. To increase flexibility robustness of the framework, there should be more categories and robot models available;
- In HeMuRo Framework the collision avoidance was taken in consideration for UGVs. The collision avoidance was not take into consideration for UAVs. This was also set as a future work.
- Every year, a new version of ROS is released. The current version used was the ROS1 noetic. There is also a new version of ROS, called ROS2. The versions of ROS2 are launched in parallel with ROS1. New features for localization and navigation are available in ROS2. For future improvements a migration for ROS2 might be considered.

Last but not least, this master's thesis has brought many challenges to the author, starting from how to program and develop a framework in c++ to trending topics in robotics. The use of ROS middleware is also a highlight of this work and since is commonly used in the industry, mastering this tool is essential for a robotics engineer.

REFERÊNCIAS

ADEPT TECHNOLOGY, Inc. **Pioneer 3DX - Rev A**. [S.l.]: Adept Technology, Inc., 2011. <https://www.generationrobots.com/media/Pioneer3DX-P3DX-RevA.pdf>.

ARBANAS, Barbara; BOLJUNCIC, Sara; PETROVIC, Tamara; BOGDAN, Stjepan. Decentralized Energy Sharing Protocol using TÆMS Framework and Coalition-based Metaheuristic for Heterogeneous Robotic Systems. **IFAC-PapersOnLine**, v. 50, n. 1, p. 5914–5919, July 2017. ISSN 24058963. DOI: 10.1016/j.ifacol.2017.08.1377.

BELLIFEMINE, Fabio; POGGI, Agostino; RIMASSA, Giovanni. Developing multi-agent systems with JADE. In: SPRINGER. INTERNATIONAL Workshop on Agent Theories, Architectures, and Languages. [S.l.: s.n.], 2000. P. 89–103.

BHATTACHARYA, Priyadarshi; GAVRILOVA, Marina L. Voronoi diagram in optimal path planning. In: IEEE. 4TH International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007). [S.l.: s.n.], 2007. P. 38–47.

CARDOSO, Rafael C.; BORDINI, Rafael H. Decentralised planning for multi-agent programming platforms. In: PROCEEDINGS of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS. [S.l.: s.n.], 2019.

COEX. **Introduction · Clover**. [S.l.: s.n.]. Accessed: 2021-04-15. Available from: <https://clover.coex.tech/en/>.

CORKE, Peter. **Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised**. [S.l.]: Springer, 2017. v. 118.

DA FONSECA BRAGA, A.; DELLA MÉA PLENTZ, P.; DE PIERI, E. R. Development of a Generic Framework for Multi-robot Systems. In: IECON 2020 The 46th Annual Conference of the IEEE Industrial Electronics Society. [S.l.: s.n.], 2020. P. 783–788. DOI: 10.1109/IECON43393.2020.9254516.

DA FONSECA BRAGA, Afonso. **HeMuRo Framework**. [S.l.]: GitHub, 2021a. <https://github.com/afonsofonbraga/HeMuRo-Framework>.

DA FONSECA BRAGA, Afonso. **HeMuRo Framework Demo**. [S.l.]: GitHub, 2021b. https://github.com/afonsofonbraga/HeMuRo_Demo.

DECKER, Keith. TAEMS: A framework for environment centered analysis & design of coordination mechanisms. **Foundations of distributed artificial intelligence**, p. 429–448, 1996.

EIJYNE, TAN; G, Rishwaraj; S G, Ponnambalam. Development of a task-oriented, auction-based task allocation framework for a heterogeneous multirobot system. **Sādhanā**, v. 45, n. 1, p. 115, Dec. 2020. ISSN 0256-2499. DOI: 10.1007/s12046-020-01330-4. Available from: <http://link.springer.com/10.1007/s12046-020-01330-4>.

EMWEB. **Wt Widget Gallery**. [S.l.: s.n.], 2021a. <https://www.webtoolkit.eu/widgets/forms/integration-example>. Accessed: 2021-04-28.

EMWEB. **Wt, C++ Web Toolkit**. [S.l.: s.n.], 2021b. <https://www.webtoolkit.eu/wt>. Accessed: 2021-02-06.

FIPA, ORG. FIPA Contract Net Interaction Protocol Specification. **Document no. SC00029H**, 2002.

FOUNDATION, Open Source Robotics. **Gazebo**. [S.l.: s.n.], 2014. Accessed: 2021-03-01. Available from: <http://gazebosim.org>.

GERKEY, Brian P.; MATARIĆ, Maja J. A formal analysis and taxonomy of task allocation in multi-robot systems. **International Journal of Robotics Research**, v. 23, n. 9, p. 939–954, 2004. ISSN 02783649. DOI: 10.1177/0278364904045564.

GUNN, Tyler; ANDERSON, John. Dynamic heterogeneous team formation for robotic urban search and rescue. **Journal of Computer and System Sciences**, v. 81, n. 3, p. 553–567, May 2015. ISSN 00220000. DOI: 10.1016/j.jcss.2014.11.009. Available from: <https://linkinghub.elsevier.com/retrieve/pii/S0022000014001500>.

HUSARION. **ROSbot 2.0 & ROSbot 2.0 PRO - Husarion Docs**. [S.l.]: Husarion, 2021. <https://husarion.com/manuals/rosbot/>. Accessed: 2021-04-28.

KIENER, Jutta; VON STRYK, Oskar. Towards cooperation of heterogeneous, autonomous robots: A case study of humanoid and wheeled robots. **Robotics and Autonomous Systems**, Elsevier B.V., v. 58, n. 7, p. 921–929, 2010. ISSN 09218890.

DOI: 10.1016/j.robot.2010.03.013. Available from:
<http://dx.doi.org/10.1016/j.robot.2010.03.013>.

KORSAH, G. Ayorkor; STENTZ, Anthony; DIAS, M. Bernardine. A comprehensive taxonomy for multi-robot task allocation. **International Journal of Robotics Research**, v. 32, n. 12, p. 1495–1512, 2013. ISSN 02783649. DOI: 10.1177/0278364913496484.

LI, Rui; WANG, Hangyu; SHI, Zhangsong. MAS based mission planning for distributed guidance system. In: THE Fourth International Workshop on Advanced Computational Intelligence. [S.l.]: IEEE, Oct. 2011. P. 387–390. DOI: 10.1109/IWACI.2011.6160037. Available from: <http://ieeexplore.ieee.org/document/6160037/>.

MUCHIRI, N.; KIMATHI, S. A Review of Applications and Potential Applications of UAV. **Proceedings of Sustainable Research and Innovation Conference**, v. 0, n. 0, p. 280–283, 2016. Available from:
<http://sri.jkuat.ac.ke/ojs/index.php/proceedings/article/view/451>.

OBDRZALEK, Zbynek. Mobile agents and their use in a group of cooperating autonomous robots. In: 2017 22nd International Conference on Methods and Models in Automation and Robotics (MMAR). [S.l.]: IEEE, Aug. 2017. P. 125–130. DOI: 10.1109/MMAR.2017.8046810. Available from:
<http://ieeexplore.ieee.org/document/8046810/>.

PARKER, Lynne E; RUS, Daniela; SUKHATME, Gaurav S. Multiple mobile robot systems. In: SPRINGER Handbook of Robotics. [S.l.]: Springer, 2016. P. 1335–1384.

PORTUGAL, David; IOCCHI, Luca; FARINELLI, Alessandro. A ROS-Based Framework for Simulation and Benchmarking of Multi-robot Patrolling Algorithms. In: STUDIES in Computational Intelligence. [S.l.: s.n.], 2019. P. 3–28. DOI: 10.1007/978-3-319-91590-6_1. Available from:
http://link.springer.com/10.1007/978-3-319-91590-6_1.

RAO, MV Sreenivas; SHIVAKUMAR, M. Overview of Battery Monitoring and Recharging of Autonomous Mobile Robot. **International Journal on Recent and Innovation Trends in Computing and Communication**, v. 6, n. 5, p. 174–179.

RIZK, Yara; AWAD, Mariette; TUNSTEL, Edward W. Cooperative heterogeneous multi-robot systems: A survey. **ACM Computing Surveys**, v. 52, n. 2, 2019. ISSN 15577341. DOI: 10.1145/3303848.

ROBOTICS, AWS. **AWS Robotics**. [S.l.]: GitHub, 2021.
<https://github.com/aws-robotics>.

ROS. **Fake Localization - ROS Wiki**. [S.l.]: Open Robotics, 2020a. Accessed: 2021-03-01. Available from: http://wiki.ros.org/fake_localization.

ROS. **Gmapping - ROS Wiki**. [S.l.]: Open Robotics, 2020b. Accessed: 2021-03-01. Available from: <http://wiki.ros.org/gmapping>.

ROS. **Navigation - ROS Wiki**. [S.l.]: Open Robotics, 2020c. Accessed: 2021-03-01. Available from: <http://wiki.ros.org/navigation>.

ROS. **ROS Wiki**. [S.l.]: Open Robotics, 2020d. Accessed: 2021-03-01. Available from: <http://wiki.ros.org>.

SAMPEDRO, Carlos; BAVLE, Hriday; SANCHEZ-LOPEZ, Jose Luis; FERNANDEZ, Ramon A. Suarez; RODRIGUEZ-RAMOS, Alejandro; MOLINA, Martin; CAMPOY, Pascual. A flexible and dynamic mission planning architecture for UAV swarm coordination. In: 2016 International Conference on Unmanned Aircraft Systems (ICUAS). [S.l.]: IEEE, June 2016. P. 355–363. DOI: 10.1109/ICUAS.2016.7502669. Available from: <http://ieeexplore.ieee.org/document/7502669/>.

SANTOS, Fernando Rodrigues et al. Avaliação do uso de agentes no desenvolvimento de aplicações com veículos aéreos não-tripulados, 2015.

SEYEDI, Sepehr; YAZICIOGLU, Yasin; AKSARAY, Derya. Persistent Surveillance With Energy-Constrained UAVs and Mobile Charging Stations. **IFAC-PapersOnLine**, v. 52, n. 20, p. 193–198, Aug. 2019. ISSN 24058963. DOI: 10.1016/j.ifacol.2019.12.157. arXiv: 1908.05727. Available from: <https://linkinghub.elsevier.com/retrieve/pii/S2405896319320087%20http://arxiv.org/abs/1908.05727>.

SHAKHATREH, Hazim; SAWALMEH, Ahmad; AL-FUQAHA, Ala; DOU, Zuochoao; ALMAITA, Eyad; KHALIL, Issa; OTHMAN, Noor Shamsiah; KHREISHAH, Abdallah; GUIZANI, Mohsen. Unmanned Aerial Vehicles: A Survey on Civil Applications and Key

Research Challenges. **IEEE Access**, Apr. 2018. ISSN 21693536. DOI: 10.1109/ACCESS.2019.2909530. arXiv: 1805.00881. Available from: <http://dx.doi.org/10.1109/ACCESS.2019.2909530>.

SICILIANO, B.; SCIAVICCO, L.; VILLANI, L.; ORIOLO, G. **Robotics: Modelling, Planning and Control**. [S.l.]: Springer London, 2010. (Advanced Textbooks in Control and Signal Processing). ISBN 9781846286414. Available from: <https://books.google.com.br/books?id=jPCAFmE-logC>.

SIEGWART, Roland; NOURBAKHSH, Illah Reza; SCARAMUZZA, Davide. **Introduction to autonomous mobile robots**. [S.l.]: MIT press, 2011.

SILVA BARBOSA, Alexander. **Battery Simulator**. [S.l.]: GitHub, 2021. https://github.com/robotizandoufsc/battery_simulator.

SILVAGNI, Mario; TONOLI, Andrea; ZENERINO, Enrico; CHIABERGE, Marcello. Multipurpose UAV for search and rescue operations in mountain avalanche events. **Geomatics, Natural Hazards and Risk**, Taylor & Francis, v. 8, n. 1, p. 18–33, 2017.

TAKAYA, K.; ASAI, T.; KROUMOV, V.; SMARANDACHE, F. Simulation environment for mobile robots testing using ROS and Gazebo. In: 2016 20th International Conference on System Theory, Control and Computing (ICSTCC). [S.l.: s.n.], 2016. P. 96–101. DOI: 10.1109/ICSTCC.2016.7790647.

TIWARI, Kshitij; YOUNG CHONG, Nak. Multi-robot systems. In: MULTI-ROBOT Exploration for Environmental Monitoring. [S.l.]: Elsevier, 2020. P. 159–169. DOI: 10.1016/B978-0-12-817607-8.00027-7. Available from: <https://linkinghub.elsevier.com/retrieve/pii/B9780128176078000277>.

WANG, Xiangke; ZENG, Zhiwen; CONG, Yirui. Multi-agent distributed coordination control: Developments and directions via graph viewpoint. **Neurocomputing**, 2016. ISSN 18728286. DOI: 10.1016/j.neucom.2016.03.021. arXiv: 1505.02595.

WOOLDRIDGE, Michael. **An introduction to multiagent systems**. [S.l.]: John Wiley & Sons, 2009.

YU, Kevin; BUDHIRAJA, Ashish Kumar; BUEBEL, Spencer; TOKEKAR, Pratap. Algorithms and experiments on routing of unmanned aerial vehicles with mobile

recharging stations. **Journal of Field Robotics**, 2019. ISSN 15564967. DOI: 10.1002/rob.21856.

ZAKIEV, AUFAR; TSOY, TATYANA; MAGID, EVGENI. Swarm Robotics: Remarks on Terminology and Classification. In: SPRINGER. INTERNATIONAL Conference on Interactive Collaborative Robotics. [S.l.: s.n.], 2018. P. 291–300.

ZLOT, ROBERT MICHAEL. An auction-based approach to complex task allocation for multirobot teams. **ProQuest Dissertations and Theses**, v. 3250901, October, 179–179 p. 2006.