Engineering Assistant Internship

Royal Military Academy of Belgium

Supervisor: Emile Le Flécher

Design and Integration of Multi-Robot Cooperation Capabilities for Military Application

Hugo Yverneau
Student Engineer in Autonomous Robotics
`hugo.yverneau@ensta-bretagne.org`

Contents

Introduction **v**
Engineering Assistant Internship in ENSTA Bretagne v
RMA & RAS-lab v
iMUGS v

1 Context **1**
1.1 iMUGS Project 1
1.1.1 Overview of the different Sub-projects 1
1.1.2 Swarming 3
1.2 Swarming Architecture 3
1.2.1 Swarming in iMUGS 3
1.2.2 Cloud, Fog and Edges 4
1.2.3 Global Swarming Workpackages 4
1.2.4 RAS-lab job 7
1.3 Swarm Platforms 7
1.3.1 Robots 7
1.3.2 HMI 7
1.4 Objectives of the Internship 8
1.4.1 Course of the Internship 8
1.4.2 HMI 8
1.4.3 Demonstration iMUGS 8
1.4.4 Organization and Management 9

2 Web Interface **10**
2.1 Requirements 10
2.1.1 Use for Swarm Developpement 10
2.1.2 Demonstrations Support 10
2.1.3 Modularity for Other Projects 10
2.2 Tools 11
2.2.1 Language Used 11
2.2.2 Roslib js 11
2.3 Project Architecture 12
2.3.1 Outside link 13
2.4 Docker 13
2.4.1 ROS2 on Robotniks 13
2.4.2 Contenarization on dotOcean Server 13
2.5 Work Done 14
2.5.1 Model 14
2.5.2 Web Pages 14

3 Demonstration **20**
3.1 Purpose and contexts 20
3.1.1 Swarming Demonstration 20
3.1.2 Next Demonstrations 20

3.2 RMA Preparation 21
3.2.1 High Level Control 21
3.2.2 Autonomy 21
3.3 Course of the demonstration 21
3.3.1 Other Partners 21
3.3.2 RMA 22

Conclusion **24**

Annexes **I**
List of Acronyms I
List of Figures II
References III
Glossary IV
Assessment Report VII
Models IX
README.md XVI

Résumé

Un an avant la fin de mes études en robotique à l'école d'ingénieur École Nationale Supérieur des Techniques Avancées de Bretagne (ENSTA Bretagne), j'ai réalisé un stage en tant qu'assistant ingénieur au Robotic & Autonomous System laboratory (RAS-lab), ce document en est le rapport. Durant ce stage, j'ai accompagné les chercheurs du laboratoire dans la réalisation de la démonstration 4 du projet de l'Union Européenne integrated Modular Unmanned Ground System (iMUGS), mais aussi mis en place le développement d'une site web ayant pour objectif d'assister les chercheurs lors du développement de robots d'une manière adaptée à leurs outils de travail.

Abstract

One year before the end of my studies in robotics at ENSTA Bretagne engineering school, I realized an internship as an assistant engineer at RAS-lab, this document is the report of my work. During this internship, I accompanied the researchers of the laboratory in the realization of the demonstration 4 of the project of the European Union (EU) iMUGS, but also set up the development of a website having for objective to assist the researchers during the development of robots in a way adapted to their working tools.

Thanks

I would like to thank Royal Military Academy (RMA) for its support during this internship as well as the ENSTA Bretagne for allowing me to have the necessary skills for this internship.

I thank in particular Emile Le Flécher, Enzo Ghisoni and Alexandre La Grappe for the organization of my internship, their technical explanations as well useful as numerous and for having integrated me so well in the team. Congratulation for your work, especially for the iMUGS demonstration.

Thanks also to Geert De Cubber for recruiting me to the RAS-lab and for making sure that everything went well.

Introduction

Engineering Assistant Internship in ENSTA Bretagne

ENSTA Bretagne training covers the last year of the bachelor's degree and the two years of master's degree. The first year of Master's degree, so one year after the specialization in autonomous robotics, ends with a four-month internship.

The objective of the technical internship as an assistant engineer in the first year of the Master's program is to allow the student to confront his or her knowledge with the activities of the internationalized industrial and technological sectors corresponding to the openings in the program and to put this knowledge into perspective for the final phase of professionalization that constitutes the third and final year of the program.

RMA & RAS-lab

RMA is a military academy; a university in charge of training officer candidates for all components of the Belgian Defense. Courses are given in three languages (Dutch, French and English). This military university is composed of two faculties: the polytechnic faculty which delivers the diploma of civil polytechnic engineer with specializations (telecommunications, ballistics, construction, mechanics) and the faculty of social and military sciences (management and weapon systems, defense and political sciences).

The aim of the labratories[1] is to provide cutting-edge researches on specific topics. Research at RMA tends to cover a broad spectrum of expertise. Given the particular status of the research topics tackled at RMA, the laboratories have the opportunity to develop a valuable and unique expertise, as well for the research itself but also for the Defense and the industry. RMA promotes research in collabora-tion with industrialists and the filing of patents, this may involve responding to EU calls for tenders.

RAS-lab[2] is a unit of Department of Mechanics (MECA) of RMA specialized in unmanned vehicles of all components: air, land, marine but also counter UXV (UXV) and some more transersal projects. As a university laboratory, the RAS-lab conducts research in all areas related to unmanned systems: from another side, the laboratory aims to develop new applications for these systems as well as solutions to prevent their potentially harmful use. As a military research institute and member of the Belgian defence, the RAS-lab focuses on niche fundamental research but also on practical applications that bring direct benefits to the security and defence fields.

iMUGS

The iMUGS project is the largest and most ambitious project currently ongoing at RAS-lab. The objective is to develop a modular and scalable architecture for hybrid manned/unmanned systems to meet a wide range of missions and to allow easy upgrading

[1]Further information on www.rma.ac.be/en/information-for/research-at-the-rma/laboratories

[2]Further information on www.mecatron.rma.ac.be/

or modification of system assets and functionalities: air and ground platforms, command, control and communication equipment, sensors, payloads and algorithms. To demonstrate the features of the project, the prototype is based on an existing unmanned ground vehicle: the THeMIS and a specific list of payloads such as a communication antenna or a machine gun. The idea is to create a standardized robot architecture for a better cooperation within EU's armies.

Whereas many of robotics projects are small-scale and have strong budget constraints, iMUGS is a two and a half year project with about 30 million euros budget and a collaboration of 14 partners from 7 countries. As with all EU projects iMUGS is organized in several sub-projects, more or less linked each other, which have been opened by a call for tenders to laboratories and companies of any sizes. This project takes place within the framework of European Defence Industrial Development Programme (EDIDP) (European Defence Industrial Development Programme) which weighs around 150 to 200 million euros a year. This program is part of a political will to promote greater cooperation and coordination between the armies of EU members at right angles to North Atlantic Treaty Organization (NATO).

The iMUGS project regroup 14 partners from Belgium, Estonia, Finland (not in NATO), France, Germany, Latvia and Spain. The partners are

- Milrem Robotics (iMUGS coordinator) - Estonia - www.milremrobotics.com
  Milrem Robotics is the global coordination of iMUGS.
- Safran - France - www.safran-group.com/fr/societes/safran-electronics-defense
- GTCT - Estonia
- Bittium - Finland
- INSTA - Finland
- GMV - Spain - www.gmv.com/en/communication/press-room/press-releases/gmv-participates-european-defense-program-development
- KMW & CO KG - Germany
- Diehl Defence& CO KG - Germany
- Nexter - France
- LMT - Latvia
- (Un)manned - Belgium
- RMA - Belgium - www.mecatron.rma.ac.be/
- dotOcean - Belgium - www.dotocean.eu
- Estonian Military Academy - Estonia

1 Context

1.1 iMUGS Project

The iMUGS project consists in studying, designing, prototyping and testing the fundamental functionalities of manned-unmanned systems and their integration on the basis of a modular, standardized and open source system. The project will develop a modular and scalable architecture for hybrid manned/unmanned systems to perform a wide range of missions and to allow easy upgrading or modification of assets and functionalities within the system: air and ground platforms, command, control and communication equipment, sensors, payloads and algorithms. The project is structured into seven sub-projects, all of which cover studies, design, prototyping, testing and technical coordination:

1. System integration
2. Autonomous functions
3. Cyber security
4. Communication
5. Swarming
6. Command, control and interoperability
7. Teamwork with men and women

Six demonstrations have been and will be carried out across Europe in each of the participating member states, and a final engagement event will mark the end of the project. The project's governing bodies also include end users, i.e., the military. The military is involved in the design and execution of the demonstration scenarios. The aim is to make this project a success in order to convince the added value of EU-funded defence research.

iMUGS fully answers the EDIDP call in 2019. The work is intended to pave the way for building a strong and innovative competence in the EU. The overall budget of iMUGS is 32.6 million euros, of which 30.6 million euros is funded by the EDIDP programme and 2 million euros by the participating Member States.

1.1.1 Overview of the different Sub-projects

Sub-project 1: System engineering and platform This work package is leaded by Milrem Robotics.

The objective is to provide system engineering management and overall architecture for the entire iMUGS project, establish the system requirements, design and interface the specifications, conduct the system level test campaign, and provide data to verify that the requirements are met. In addition, a set of system-level security measures will be provided, as well as a legal, ethical, and perception analysis on the use of unmanned ground systems.

Sub-project 2: Autonomous functions This work package is leaded by Safran and the other particpants are DIEHL, Nexterand Milrem Robotics.

This sub-project consists of specifying the high-level architecture of the autonomy framework, the design of such an autonomy framework is based on specific tools such as the data warehouse (fed by data collected during specific campaigns to meet the use cases defined by the project, to train the artificial intelligence algorithms of the autonomy functions), the autonomy simulation (with the hardware in the loop, for development and integration purposes). But also to provide the necessary sensors for autonomy and a complete autonomy framework to perform the testing and integration in THeMIS.

Sub-project 3: Cybersecurity This work package is leaded by GTCT.

The objective of the cybersecurity studies will be to understand the types of cyber threats that iMUGS will face in an operational military environment. In addition, the risk assessments will consider these threats and evaluate risk-introducing vulnerabilities in the consortium partners' designs for individual iMUGS subsystems. In order to create the design of the CDS. This will start with a generic framework that can be applied to any autonomous vehicle and then refined to incorporate the scope of the iMUGS vehicle. An operational demonstration of the cyber defense system is required to validate the sub-project.

Sub-project 4: Communication equipment and network This work package is leaded by Bittium.

The objective is to analyze the use cases and select the appropriate technologies of communications to implement and integrate into iMUGS. The design work includes development and implementation activities and, in addition, testing as well as integration on THeMIS. Prototyping includes the physical construction of the demonstration system as well as the integration of subsystems to form a functional entity that meets the defined requirements.

Sub-project 5: Swarming This work package is leaded by RMA and the other particpants are dotOcean and INSTA.

The first objective is to explore and find ways to integrate existing technologies for real-time swarming capabilities, addressing the use cases. The second objective is to design the efforts related to the architecture, interfaces, data structuring, search algorithms, solvers and optimizers needed to address the use cases. A demonstration in a local swarming simulation will be finalized to enable the demonstration of the developed algorithms. The simulation application will be capable of simulating military vehicles and personnel. INSTA will structure the data in a simulation application to enable demonstration of the developed algorithms. The global swarming must be validated during a real demonstration.

Sub-project 6: Command and Control and Interoperability This work package is leaded by GMV and the other particpants are (Un)manned and KMV.

The objectives of this sub-project are to define the Control and Command (C2) subsystem operational concept and develop the C2 subsystem technical requirements

specification, design, prototype and develop C2 subsystem components, plans and procedures, software and hardware.

Sub-project 7: Manned-Unmanned Teaming This work package is leaded by KMW.

The objective is to define military operational scenarios for the development and engagement of THeMIS in different combinations, and those with soldiers and/or other systems. The connection between an unmanned air or ground system and a Boxer, as well as the integration of the property will also be defined. Military operational scenarios will be tested and demonstrated for evaluation.

1.1.2 Swarming

Swarm robotics is a branch of robotics applying distributed intelligence methods to multi-robot systems. It usually involves the use of simple, even simplistic, and inexpensive robots of limited individual interest, but which together, meaning with self-assembly or self-organization capabilities, form a complex and robust system. Swarm robotics seeks to study the design and behavior of robots. Relatively simple rules can give rise to a complex set of swarm behaviors or even emergent behaviors. A key component of the swarm is the communication between its members, establishing a feedback loop that aims at the cooperation of the group. Swarm robotics is inspired by entomological studies on social insects such as ants, termites or bees, or the group or cooperative behaviors of other organisms (bacteria, worms (e.g. Lumbriculus variegatus), schools of fish, birds, etc.). The interest is the ability of these simple agents to collectively produce intelligent systems; in this way, they perform together tasks that would be unattainable for an insect alone. Swarm robotics seeks to do the same with simple robots. As we will see below, some interesting properties arise from this situation.

One of the most promising uses of swarming involves search and rescue missions. Swarms of robots of various sizes could be sent to places where rescuers cannot safely go in order to explore unfamiliar environments and use onboard sensors to solve complex puzzles. On the other hand, swarm robotics may be useful for tasks that require low-cost designs, such as mining or agricultural grazing tasks. Navy has tested a swarm of autonomous ships that can drive themselves and take offensive actions. These ships are unmanned and can be equipped with any equipment to detect and destroy enemy ships.

1.2 Swarming Architecture

1.2.1 Swarming in iMUGS

There are two methods of swarming in iMUGS requiremnts: local swarming and global swarming.

The global swarming must answer the following problem: how to control multiple robots (several dozen) with one or two operators? This method consists in using a data-centric architecture, which requires a connection to the control station, the C2. All the information of the swarm is centralized, and the computational station at C2

calculates the optimal task solution for each agent in the swarm and sends the calculated data directly to each agent. This allows having all the data and massive computational power since the computational station does not need to be embedded. This method is a major expectation of the iMUG project, a full implementation on standalone is expected. However, this method requires a real-time connection between the central station and each agent, a decentralized method is needed to take over if an agent loses its connection to the global network. The global swarming project is carried out by RMA and dotOcean.

Local swarming uses the embedded computing power of each agent to control the swarm. Only the simulation part will be developed on the iMUGS project. This method allows for swarm behavior without a long-range network, however, the available computing power is limited. The goal of local swarming is to allow swarm management without any control from outside the swarm. The iMUGS project requires a high degree of flexibility between local swarming and global swarming. Depending on the connection conditions in the field, the local swarm must be ready to take over at any time. The local swarming work package is realized by the Finnish company INSTA.

Both approaches are used in the iMUGS project as complementary modules. Both are integrated into the swarm architecture in a mutually exclusive way. The global swarm approach is used when communications are available and allow a direct link to the C2 and, by extension, to the swarm operator. However, as soon as the network fails or on direct order from the operator, communication can be interrupted. As a result, the global swarm enters a standby state, while the local swarm wakes up to continue the mission until it is accomplished or communications are restored.

1.2.2 Cloud, Fog and Edges

The cloud, fog and edges organization is common to embedded and connected systems. The edges are the embedded systems with low computational capabilities and not necessarily a direct internet connection. The fog is made up of servers connected to edge systems and the internet, it has better computing performance than edge systems. The cloud consists of single remote servers, which are connected to the fog via the Internet. This configuration is of great interest for systems that require a lot of computing power, such as real-time Artificial Intelligence (AI), and it also makes it possible to connect edges without going through the Internet, even if the edges are not directly connected to each other.

In our case, the edges are the robots. The fog is with the command station on the field: the C2 which can be connected by a secure network to the base outside the field. The base then plays the role of the cloud.

1.2.3 Global Swarming Workpackages

The swarming sub-project was organized into several modules:

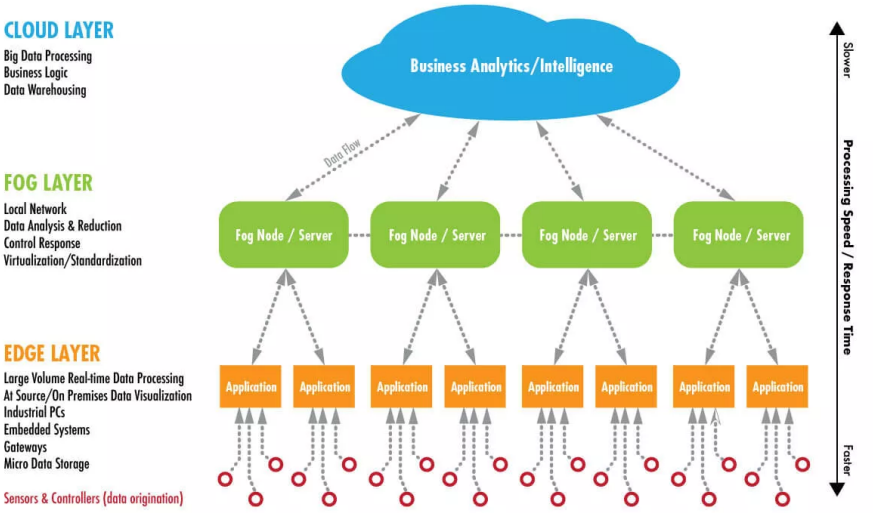

Figure 1: Cloud, Fog and Edges Layer Stack from winsystems.com

Planner The goal of the planning module is to find an optimal path for each robot according to the environment and the mission requirements.

Swarm Interface Planner The purpose of this module is to refactor the outputs of the planner module to respond to the inputs of swarm manager and vice versa.

Database The database should record all features of the swarm state, such as mission parameters, robot characteristics, etc.

C2 The C2 must obtain all user input.

Swarm C2 Interface The purpose of this module is to refactor the outputs of the C2 module to meet the inputs of swarm manager and vice versa.

Swarm Manager The purpose of this module is to allow all other modules to communicate with each other, to remove the right inputs at the right time, and to handle problems such as order errors or module disconnection.

Mission Manager Mission managers delegate the necessary information to the right agent manager and escalate the information.

Agent Manager The module manages the communication between a robot and the fog on the fog side.

Swarm Edge Client The module handles the communication between a robot and the fog on the robot side.

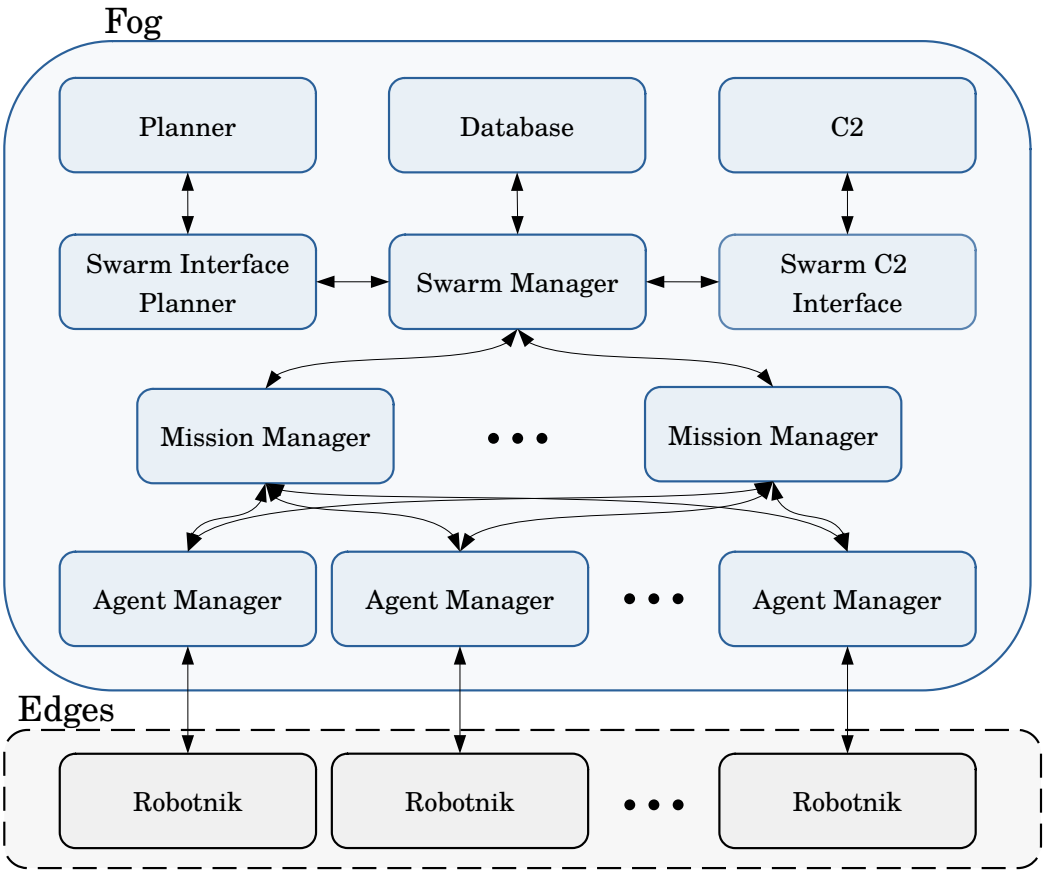

Figure 2: Diagram of the Swarming Modules in the Fog

The planner can manage the trajectory of a single robot to meet the requirements of several missions. It takes into account a map for global planning but obstacle avoidance, which must take into consideration the sensors of each robot in real time, is managed by the autonomy sub-project.

The planner can manage the trajectory of a single robot to meet the requirements of several missions. The C2 sends the number of missions requested by the user. There is one mission manager module created per mission that is connected to each agent manager of the robot used by its mission. There is one agent manager per connected robot. The swarm edge client module is not on the fog but in the robot, its role is to communicate with the agent manager and to save the planned path, to update it if necessary and to manage the case of loss of connection by launching the local swarming.

In the case of iMUGS demonstration 4, C2 was only simulated by a ROS node and the robots used were all of the same type, they were the Robotniks in the laboratory. However, the whole fog is designed to handle different types of robots with different types of sensors or antennas.

1.2.4 RAS-lab job

The swarm manager is an important module of the swarm architecture. Indeed, due to the complex reality of the military domain (human life at stake, stealth, malicious attack, degraded operating conditions, need to have humans in the loop, changing environment, etc.), there are many possible use configurations of the swarm. This can be from the user controlling a single robot directly to local swarming. Furthermore, the management of the state of each robot and the user's command must change as quickly as the situation evolves, but also all the different operating modes must work together. The job of the swarm management module is to ask all the other modules to do the right job at the right time and distribute the right results for an efficient swarming.

The swarm edge client is an embedded module. Its role is to keep in memory all the useful inputs from the agent manager to give the inputs (basically a waypoint) to the autonomy module. This module is the interface between the swarm and the autonomy but also manages the passage between the local and global swarm modes.

1.3 Swarm Platforms

1.3.1 Robots

RAS-lab has chosen to purchase five SUMMIT-XL[3] from the Spanish brand Robotnik, with different payloads (Lidar, sonar, stereo camera). One of the assets of the Robotniks is to be "plug and play". The autonomy is supposed to be done, in fact, a lot of work has been done to finalize the autonomy of the Robotniks.

1.3.2 HMI

Robotnik offers a web interface to control the robot. Moreover, this interface is not specific enough to be useful for the RAS-lab. See FIGUE 3.

Robotniks are powered by ROS: ROS1 melodic on Ubuntu 18.04[4]. This is a great asset, a simple `cmd_vel` allows to control the robots with a `Twist`, and all relevant information is published on accessible topics. But interoperability and the goal of being easily upgradable for iMUGS make the use of ROS2 necessary. Moreover, with the implementation of Data Distribution Service (DDS) directly in the ROS2 messages, the use of ROS2 increases the cyber security of the system. ROS2 is only available on Ubuntu 20.04, so RAS-lab chose to implement one docker for its work (the Swarm Edge Client module) and another docker to implement the ROS1 noetic/ROS2 bridge. The ROS1 melodic/ROS2 bridge doesn't exist, so a ROS1 noetic roscore is needed, but the connection between ROS1 melodic and noetic is done by classical ROS1 topics. See Figure 4.

[3]Further information on www.robotnik.eu/products/mobile-robots/summit-xl-en-2.

[4]Ubuntu 18.04 only supports ROS1 melodic. ROS2 will be available at the end of 2022 on Ubuntu 22.04.

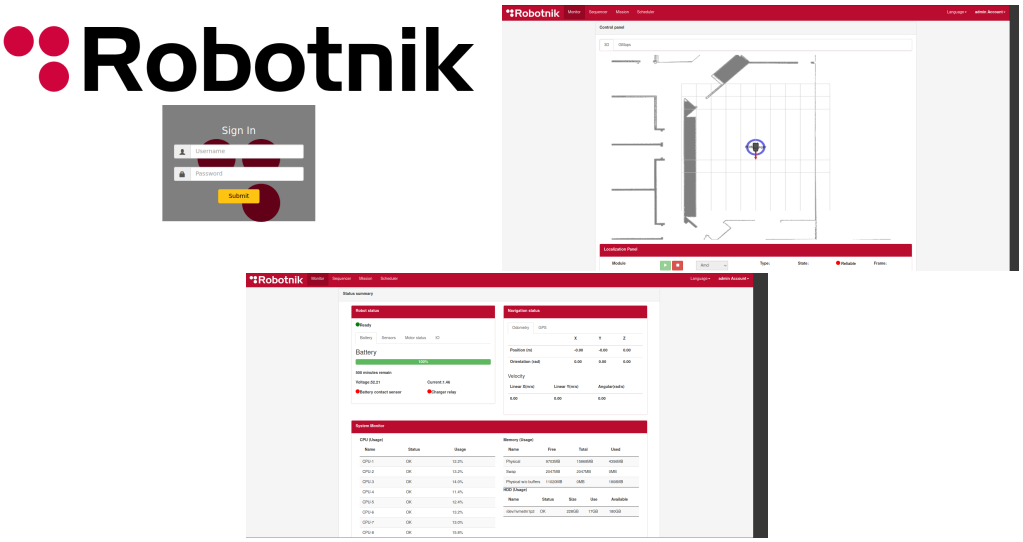

Figure 3: Screenshot of the Robotniks Web Graphic User Interface (GUI)

1.4 Objectives of the Internship

1.4.1 Course of the Internship

The internship started with two weeks of independent training under the supervision of Mr. Ghisoni. Then, I had to understand the critical expectations for the Human Machine Interface (HMI) in order to study the feasibility of the GUI and propose a draft. Once the critical points were identified and the draft was approved, I created the overall code architecture and began development. After a three-week break for demonstration support, I continue to develop the application's features with the other intern. At the end of the internship, I set up the deployment on a docker that is available online at http://193.190.204.92:3030.

1.4.2 HMI

Developing swarms with ROS often requires having several terminals launched at the same time. The use of launch files, rviz or terminator can improve the development, but the ergonomics remains to be completed to allow an efficient development. The development of a more adapted GUI could improve the efficiency of the laboratory. The development of an adaptive GUI is a good way for the iMUGS project to value the work of an intern without assigning critical tasks to the project.

1.4.3 Demonstration iMUGS

In June the iMUGS demonstration of the sub-project swarming took place. This demonstration was organized by the RAS-lab and was a great challenge. The idea was to make the intern available for support if needed. My contribution for this part of my internship was not specified in advance on purpose.

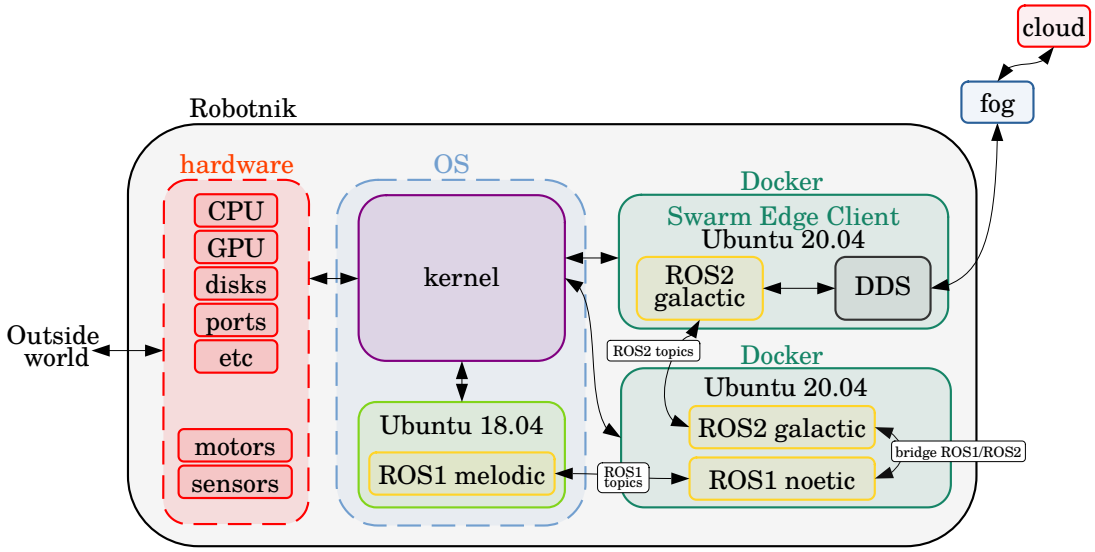

Figure 4: Software Configuration of the Robotniks

1.4.4 Organization and Management

Mr. Ghisoni was my technical supervisor. He mentored me and gave me very helpful support. Mr. Le Flécher, my internship director, supervised my internship with a quick weekly meeting. Mr. Le Flécher and La Grappe were the end-users of the product, so it was planned to consult them at all key moments of the development.

During the last month of my internship, I received help from another intern on the HMI. The new intern was finishing her bachelor's degree, had never developed any web code and was not familiar with ROS. I took part in her management: I gave her a training goal, tasks and technical support. This allowed us to advance on the website project in parallel and in complementarity, thanks to the use of gitlab, it was done relatively naturally.

2 Web Interface

2.1 Requirements

2.1.1 Use for Swarm Developpement

The first requirement of the HMI is to improve the efficiency of the swarm development. Thus, the main idea of the interface is to be efficient, all design considerations remain at a second level. The GUI must be intuitive enough to allow a new user to use it quickly and easily. With this in mind, regular feedback was provided by future users.

The HMI must allow the connection of the robots through the fog, which implies at least a connection with ROS2. The connection with ROS is specific because ROS logic and web logic are different, this makes sense for my internship topic as a robotics student.

The HMI must be easily accessible in simulation or in the field. The deployment must also be simple enough to be done quickly in a busy work environment.

2.1.2 Demonstrations Support

The great advantage of a web GUI is to be easily available. Indeed, everyone has a web browser on his computer and on his smartphone. Therefore, why not use the web page as a demonstration medium. This would allow to show specific aspects of what is going on in the code or to explain a strategy that is difficult to see in a field of several thousand square meters. Even without making the graphical interface public, it could be used on a big screen.

This part of the web application requires the development of two modes, one for the developers and a second one for the demonstrators. This requires a lot of work on the robustness of the graphical interface but also some front-end work to make it more attractive. It also requires the web page to be available online and to be effective enough to be relevant to users.

2.1.3 Modularity for Other Projects

One of the major points of the development is the modularity. The idea is to have a web page with elementary blocks (map, topic list, topic echo, robot state, fog state, etc.) and an overall structure that can be reused and adapted to several robotics projects. The modification of the web page for each project must be accessible to all researchers, even those who are not comfortable with JavaScript. This specification is very important because it forces attention to it at all levels of the code.

2.2 Tools

2.2.1 Language Used

Since it was decided to make a web interface for the HMI, HTML and CSS were essential. These two languages are the two languages at the base of the web display which allow respectively to structure and display a web page, and to make the display pleasant. In addition, JavaScript is one of the main languages used for the front-end to make a page dynamic. JavaScript is designed for the web, but it can be difficult to use on its own, however there are various frameworks available that allow easier development. These three languages were completely new to me, with the help of Mr. Ghisoni and the training of some online tutorials. I was able to quickly begin development.

In our case, I use the React framework. Technically React is a library, but since it is very complete it can compete with frameworks and is often considered a framework in itself. React was created by Facebook in 2013 under the Apache 2.0 license, which is open source, is now one of the leading JavaScript frameworks with a large community. The main point of using React for our GUI is the component-based approach. Each component gathers all the CSS[5] and the HTML code. The architecture is easy to set up and allows intuitive object-based habits[6] and to keep each component simple and general as much as possible by calling more elementary components and using arguments. The logic of React also allows for simple management of web page refreshment[7], especially when the link to ROS needs to be made.

React provides many other libraries. Yarn is a package manager that manages all the necessary dependencies and versions. Yarn allows to serve the web application, for development and for deployment. Yarn is a good tool for the portability of the project from the development phase to the deployment phase. It is a derivative of NPM which is much more used but slower, NPM and Yarn allow the same possibilities.

2.2.2 Roslib js

To make the connection between ROS and the GUI, I used Roblib js which is a JavaScript library, available with React. This library allows the communication with the ROS package rosbridge server. rosbridge server converts all ROS topics in a JSON format and sends them with a web socket and vice versa.

The rosbridge-server package allows to launch a web socket for each ROS master (for ROS1) or ROS domain (for ROS2), then an IP address and a port allow the communication with JavaScript.

[5] With the use of styled componants.

[6] To be precise, I use the function-based React style, which means that each component is a function and not an object. However, this consideration is not important here.

[7] With the use of the use effect, the refresh is done entirely underground by React.

2.3 Project Architecture

```
|-- imugs_hmi
| | Dockerfile
| | docker-compose.yaml
| | docker_readme.md
| | README.md
| | .gitlab-ci.yml
| | ... some other files
| |-- gui_app
| |-- public
| | package.json
| | README.md
| | .dockerignore
| | ... some other files
| |--src
| | | index.jsx
| | | README.md
| | |-- assets
| | |-- pages
| | |-- structure
| | |-- utils
| | |-- components
    | |-- AComponent
    | | | AComponent.jsx
    | | | few other files
    | |--AnOtherComponent
    | | | AnOtherComponent.jsx
    | | | few other files
    | | ...
    | | README.md
```

Architecture of the Web Application Scripts

React projects have a default architecture, which is made to be at the root of a git. This architecture gathers all the files and folders useful to manage the flow, sharing and sharing and a `src` folder that contains all the code. In this `src` folder the architecture is based on React. There are folders to manage the navigation and structure of the web page and a folder to manage the elementary components meant to be called by the pages. I will not go further in the explanation of the code architecture, even if attention was paid to it in order to allow the easiest possible evolution after my training course.

This architecture is very important for modularity. It allows the addition of a component or the removal of a component without changing anything in the rest of the code. All the properties concerning the space occupied by a component are grouped in the same

file so that they are strictly common to all components, which makes the components perfectly interchangeable.

2.3.1 Outside link

Besides the structure of the web application, there is communication with two other systems. The first is a database that contains useful data and the second is the ROS system. The database requires the development of a rest API that will allow the web application to make requests. The role of the rest API is to make the information, in our case in JSON format, available to an endpoint, i.e. a url. When the web application makes a request to the url, the API will ask the database and provide the necessary information.

This strategy of using a rest API can also be used for ROS, which can be useful to solve some problems. The major difficulty of using a rest API for ROS is to make an efficient data pipe when the logic of the ROS topic / API request / web page refresh are different for some aspects. For this reason, I chose to use the Roslib js library rather than a rest API for ROS communication.

On the architecture of the web app, there is a folder next to the components that gathers, in particular, all the IP, endpoint, port and url needed for each connection. This makes it easy to change them if necessary in the future without having to go into the code.

2.4 Docker

Docker is a containerization technique. It is a very useful tool for deployment. It allows a great portability of any code without the heaviness of a virtual machine.

To deploy a docker image, we use gitlab CI/CD. CI/CD is a powerful tool that allows to configure an automatic script for each push. In our case, we can configure some tests on the code and an automatic build of the docker image. gitlab makes the last built image available on a url, which allows to download it and reuse it easily.

2.4.1 ROS2 on Robotniks

The Robotniks use, for the moment, ROS1 melodic. And on the other hand, all fog modules use ROS2 galactic. ROS1 melodic has the disadvantage of not having a ROS1/ROS2 bridge, which is only available on noectic for ROS1. For this reason, several installations of ROS1 (melodic and noetic) are needed. Creating a docker especially for the ROS1/ROS2 bridge is a clean solution that does not require any modification on the operation of ROS1 melodic neither on ROS2.

2.4.2 Contenarization on dotOcean Server

Docker is really useful for a collaboration like the iMUGS swarming sub-project. In our case, the whole fog is supposed to be hosted on the dotOcean partner server. This

can be done easily by having one docker per module and making the image available online with the gitlab CI/CD.

The web application needs to be containerized in the fog to be available to users. For dockerization, I use an alpine[8] image that already includes Yarn to be as light as possible. Another big difference in docker is how the web page is served. In fact, in development, the service is not optimized at all. The docker image should build an optimized version of the web application to serve it well.

Before having the dotOcean server, all the tests were done on an RMA server available online. The web application is, as a reminder, available on http://193.190.204.92:3030. The web page is still under development and depends on the current connection of other modules and robots.

2.5 Work Done

2.5.1 Model

During numerous exchanges with users, I established a model of the site that was validated by the end users but also by my technical supervisor. This model, available in annex, is meant to be a starting point rather than a rigid document. It is a working document that focuses only on the functionality and not on the design of the final site.

2.5.2 Web Pages

It appeared that to meet the requirements, it is necessary to develop different pages. This was done, without being finalized, with the help of another intern: Alexandra Haim.

State Machine The purpose of this page is to provide a quick overview of the status of each mission. See Figure 5.

Module Graph The purpose of this page is to report the status (connected or not and active or not) of each module of the swarming sub-project. See Figure 6.

Robots The purpose of the robots page is to report the status of each robot. See Figure 7.

Log The purpose of the log page is to report errors and warnings about the swarming software. See Figure 8.

Topic Explorer The purpose of the topic exploration page is to make a topic list and a topic echo directly on the website. See Figure 9.

[8] node:18.4.0-alpine

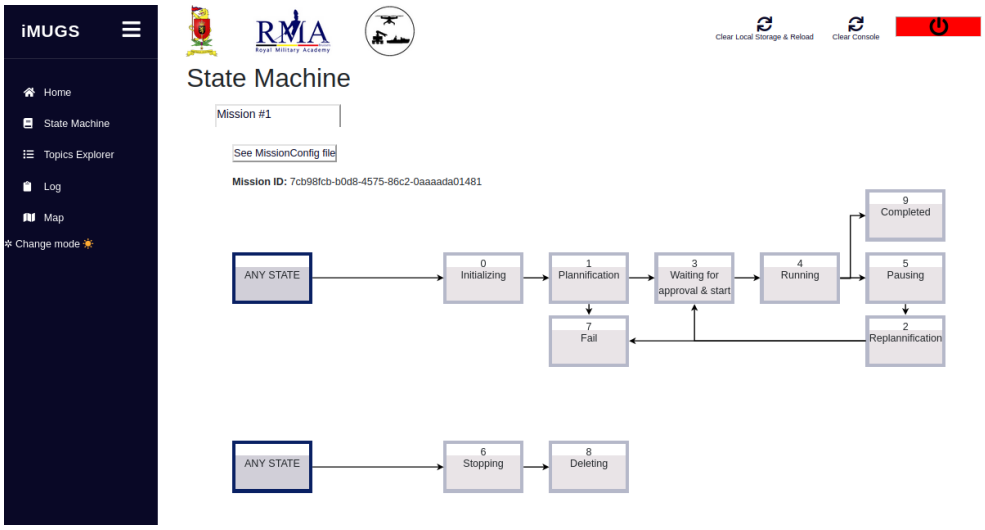

Figure 5: Screenshot of the State Machine Page

Map The purpose of the map page is to display the positions of the robots with their logo Allied Procedural Publication 6 (APP-6) and their planned trajectory. See Figure 10.

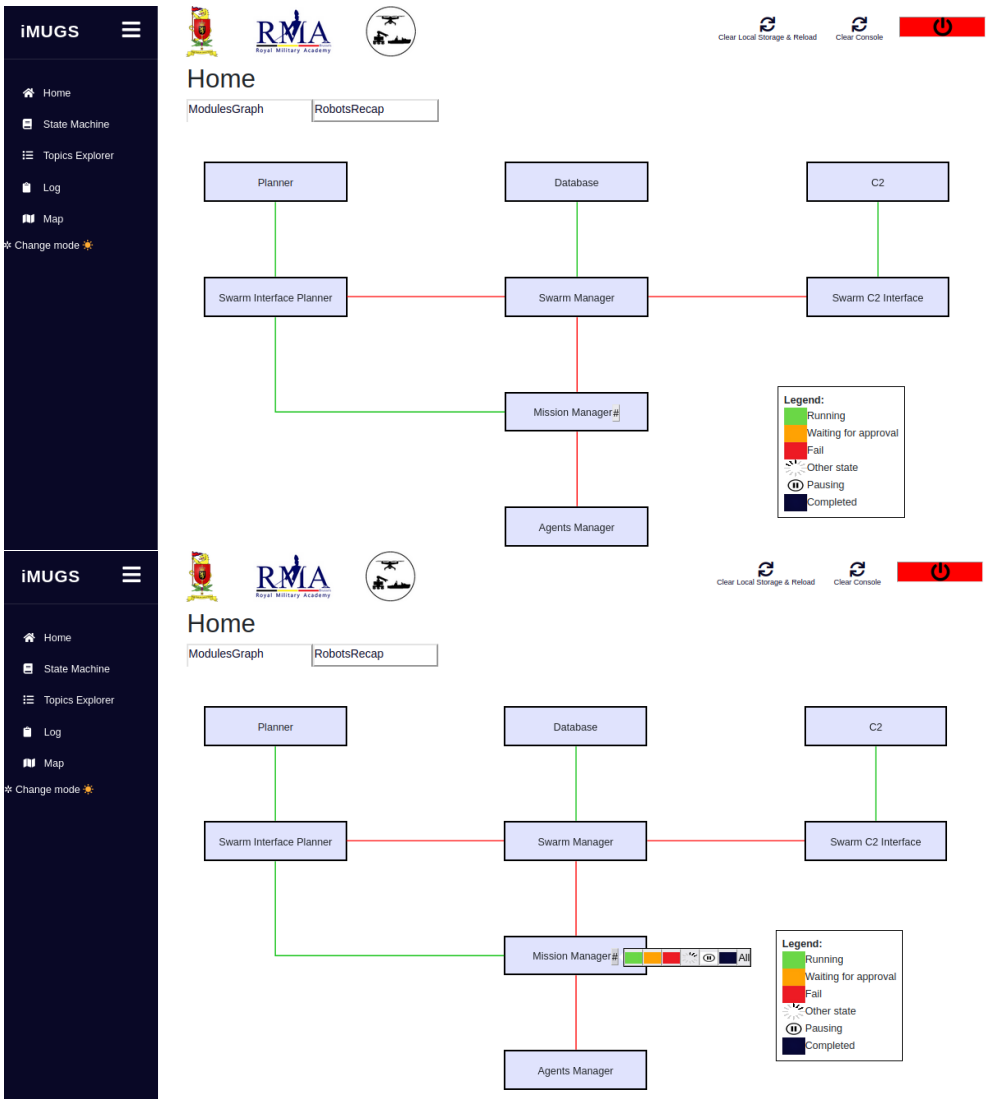

Figure 6: Screenshot of the Module Graph Page

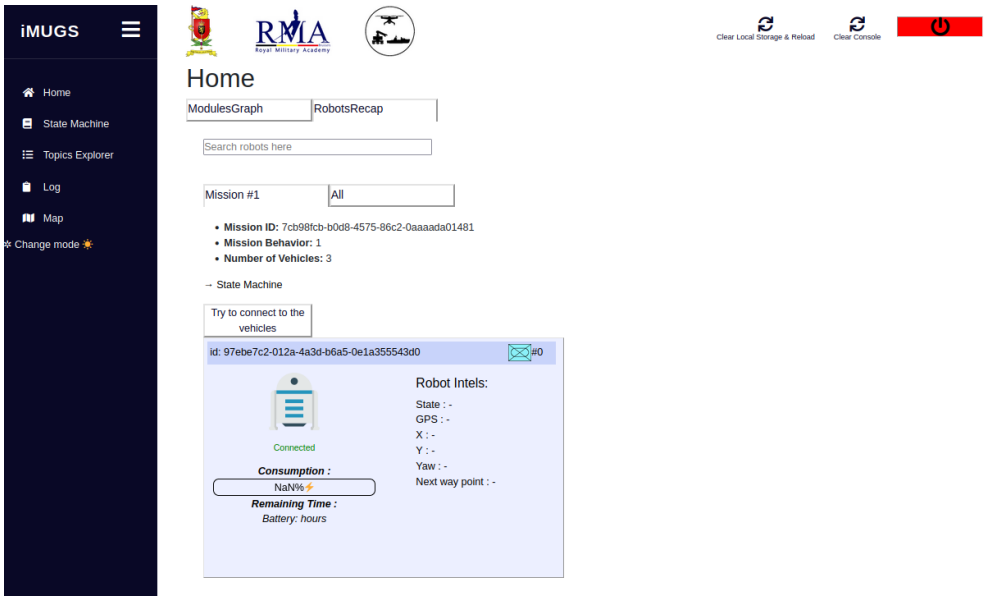

Figure 7: Screenshot of the Robots Page

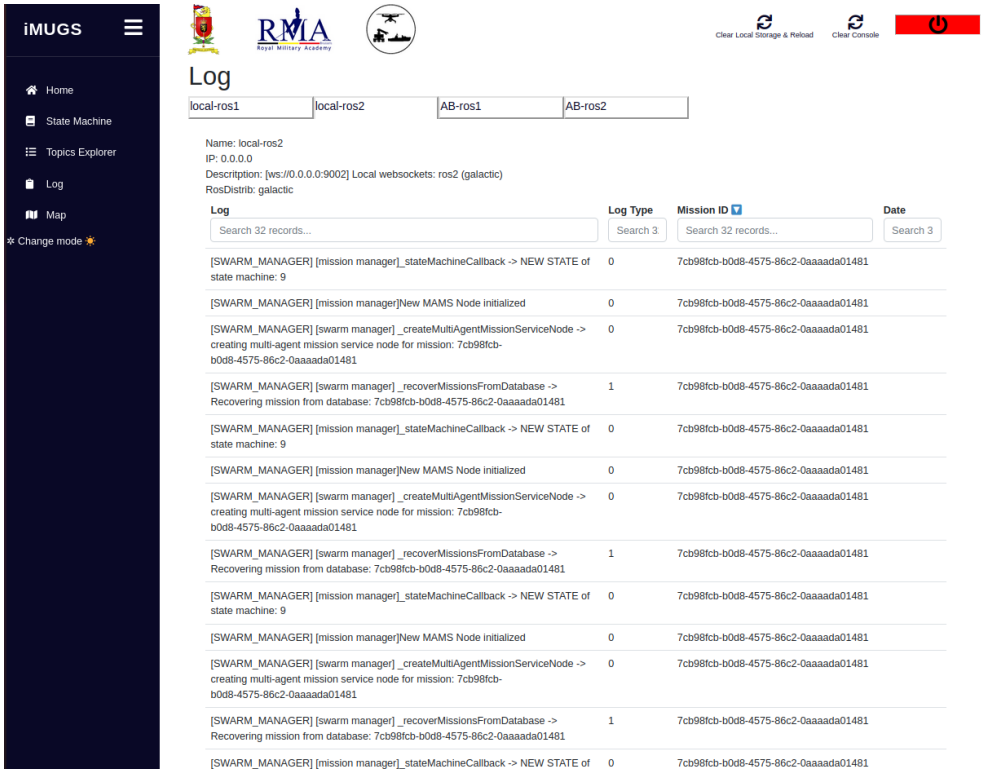

Figure 8: Screenshot of the Log Page

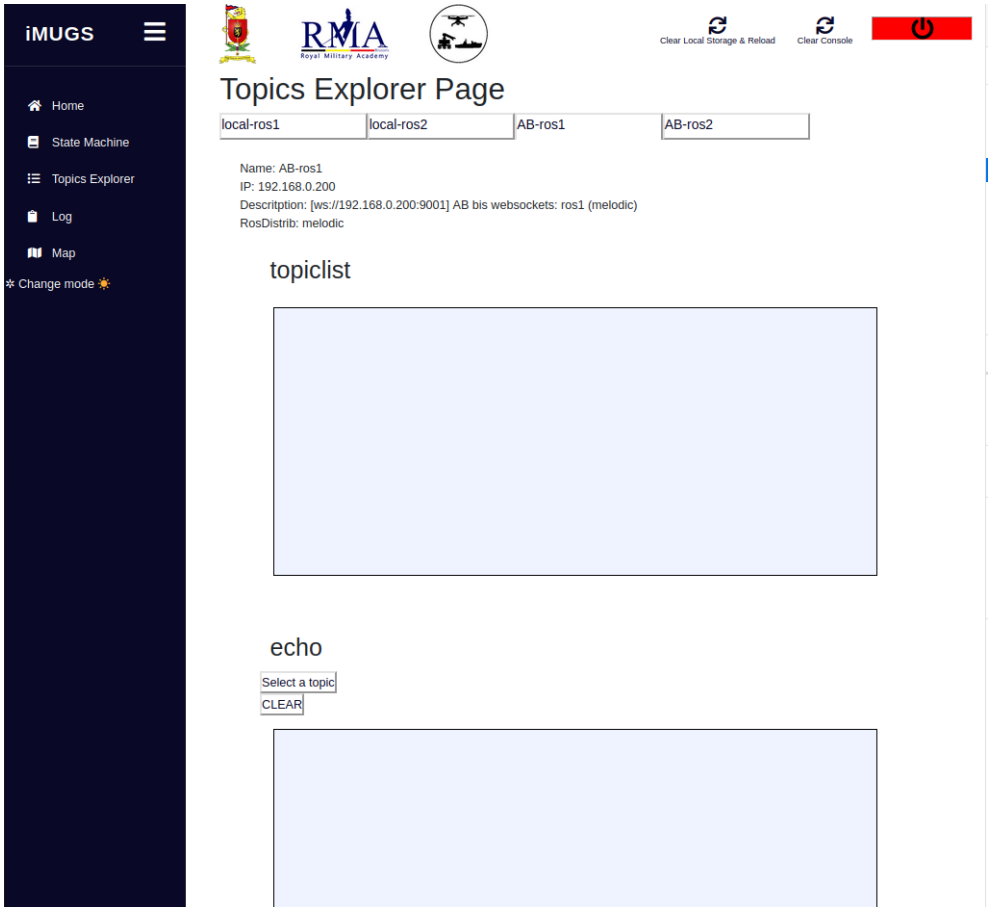

Figure 9: Screenshot of the Topic Explorer Page

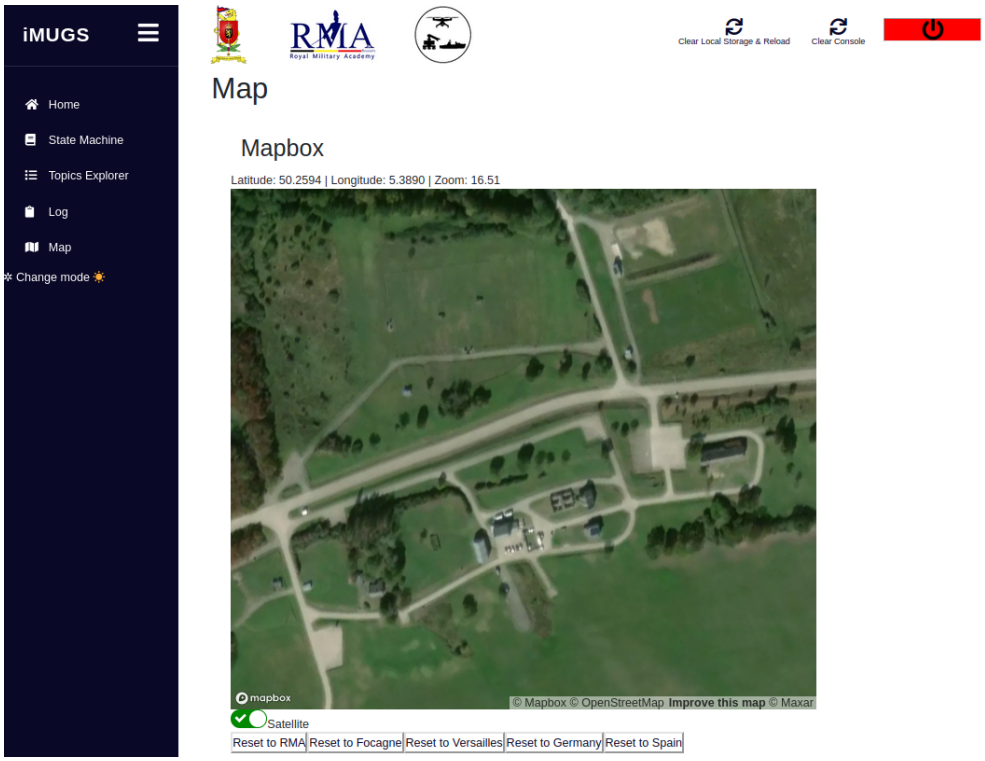

Figure 10: Screenshot of the Map Page

3 Demonstration

3.1 Purpose and contexts

3.1.1 Swarming Demonstration

All iMUGS sub-projects must be validated in a demonstration, over the entire project seven demonstrations are planned. Each demonstration is an opportunity for all partners to see the evolution of the overall project and to collaborate for the deployment in the field. In addition, it allows the European Commission, as a client, to verify the proper use of the funds.

The swarming demonstration must validate the ability to simultaneously control multiple robots to accomplish a task in an optimized manner. These tasks can be the supply of material to the soldier, mapping, area observation, etc.

The swarming demonstration is the fourth iMUGS demonstration, it took place in Marche-en-Famenne (Belgium), in the military base called King Albert Camp. The demonstration took place on Thursday, June 2, in front of about two hundred guests. The guests were members of the European Commission and officers from the different client armies of the project but also from other partners of the consortium.

Figure 11: Aerial Image of Focagne (King Albber's Camp), Site of the Demonstration 4 [3]

3.1.2 Next Demonstrations

The next demonstration will take place at the end of October 2022, its goal is to prove the capacity of THeMIS to navigate automatically without any risk for the people or the objects which could be near themselves.

This demonstration will fill the gap between high-level swarming development and operational robots for EU militaries.

3.2 RMA Preparation

3.2.1 High Level Control

The work of RAS-lab on iMUGS is to coordinate the spin-off sub-project to give waypoints to follow for autonomy. However, due to delay from Autonomy sub-project, the iMUGS autonomy sub-project was not yet ready for demonstration 4, it has to be ready for demonstration 5. All the partners of the swarming sub-project decided to make a demonstration in their own way although the work was done in collaboration. RMA chose to use the Robotniks to show the work done in the path planning, swarm manager, swarm client, database and C2 interface.

3.2.2 Autonomy

While the high end of the demonstration was ready in time, the autonomy of the Robotniks was not when I arrived in the laboratory. Even though the Robotniks are supposed to be "plug and play," it took a lot of setup, calibration, and testing to get them working.

During the preparation, we had a lot of problems with the localization, especially because of the GPS noises that were clearly not Gaussian: a drift, up to several meters, was observed from time to time in a seemingly random way. While the scenario requires sub-meter accuracy in order to follow small roads. Finally, after a lot of testing, debugging, ideas and testing, we managed to make the Robotniks' autonomy work in time.

3.3 Course of the demonstration

All demonstration went well, all partners involved managed to make it a success, despite the fact that the project was still in development and there was little time in the field to coordinate all the different technologies and technical requirements.

3.3.1 Other Partners

The THeMISes are the main robot of the iMUGS project, they are really impressive and have a huge commercial importance for Milrem Robotics, even if they were not autonomous and even less swarming, the THeMISes took a big part of the show. They have demonstrated the ability to navigate and purchase missions in the field in cooperation with Belgian soldiers with different sensor or propulsion technologies depending on the mission requirements.

dotOcean developed a simple robot especially for the occasion. Their robot was used to demonstrate the work of the swarming sub-project but with a different scenario than the RMA Robotniks. Indeed, the dotOcean robots and the Robotniks did not have the

Figure 12: Pictures of THeMIS During the Demonstration 4 [3]

same limitations or the same problems to solve for the demonstration. The dotOcean robots had to explore the demonstration area.

Figure 13: Pictures of dotOcean Robots During the Demonstration 4 [2] [3]

3.3.2 RMA

RMA demonstrated the work of the swarming sub-project with three Robotniks, the mission was to resupply the Belgian soldiers who participated in the demonstration. The Belgian soldiers were divided into two groups, which had to be resupplied in order to continue their mission against the terrorists.

The RMA part of the demonstration went as planned, all the Robotniks reached their replenishment targets on time.

Figure 14: Pictures of the RAS-lab Robotniks During the Demonstration 4 [2] [3]

Conclusion

This engineering assistant internship was a great success. From the point of view of discovering the world of research, of course, since it allowed me to have a complete experience in a laboratory. This experience is all the more valuable since my arrival took place at a key moment for RAS-lab, which made this period even more stimulating and interesting. The success is also related to the broadening of my field of expertise, whether it is directly related to robotics or connected to robotics. This one has considera-bly enlarged and is a perfect complement to my training at ENSTA Bretagne. Finally, the management of the development of an ambidextrous HMI allowed me to set up the first bricks of a project which will be developed at the end of my internship. The setting up of the benchmarks of this project, which will go beyond the scope of my internship, was very formative. Moreover, the RAS-lab team appreciated the added value of my work, which confirms the relevance of the robotics training at ENSTA Bretagne.

List of Acronyms

AI Artificial Intelligence

APP-6 Allied Procedural Publication 6

C2 Control and Command

DDS Data Distribution Service

EDIDP European Defence Industrial Development Programme

ENSTA Bretagne École Nationale Supérieur des Techniques Avancées de Bretagne

EOL End Of Life

EU European Union

GUI Graphic User Interface

HMI Human Machine Interface

iMUGS integrated Modular Unmanned Ground System

LTS Long Time Support

MECA Department of Mechanics

NATO North Atlantic Treaty Organization

RAS-lab Robotic & Autonomous System laboratory

RMA Royal Military Academy

UXV UXV

List of Figures

1 Cloud, Fog and Edges Layer Stack 5
2 Diagram of the Swarming Modules in the Fog 6
3 Screenshot of the Robotniks Web GUI 8
4 Software Configuration of the Robotniks 9
5 Screenshot of the State Machine Page 15
6 Screenshot of the Module Graph Page 16
7 Screenshot of the Robots Page 17
8 Screenshot of the Log Page 17
9 Screenshot of the Topic Explorer Page 18
10 Screenshot of the Map Page 19
11 Aerial Image of Focagne (King Albber's Camp), Site of the Demonstration 4 [3] 20
12 Pictures of THeMIS During the Demonstration 4 [3] 22
13 Pictures of dotOcean Robots During the Demonstration 4 [2] [3] 22
14 Pictures of the RAS-lab Robotniks During the Demonstration 4 [2] [3] . . 23
15 Model of the Home Page X
16 Model of the Home Page (with the required pop-up) XI
17 Model of the State Machine Page XII
18 Model of the Log Page XIII
19 Model of the Map Page XIV
20 Model of the Map Page (with the required pop-up) XV

References

[1] ROS2 Documentation.

[2] Belgian Army. iMUGS.

[3] Belgian Defence. iMUGS KMS ERM.

[4] EDIDP-MUGS-2019-002-iMUG. Grant agreement.

[5] Daniel Stouch Thomas Moore. A generalized extended kalman filter implementation for the robot operating system.

[6] Alexia Toulmet. Créez une application React complète.

Glossary

(Un)manned (Un)manned is a Belgian company participating in the iMUGS command and control and interoperability sub-project. vi, 2

Bittium Bittium is a Finnish company that leads the communication and network equipment sub-project of iMUGS. vi, 2

Boxer The Boxer is a multi-role armored combat vehicle designed by an international consortium to perform a number of operations using installable mission modules. 3

CI/CD gitlab CI/CD, for Continuous Development / Continuous Integration, is a tool that allows to set up automation at each push. It is very useful to automate tests or deployment tools. 13, 14

Diehl Defence Diehl Defence is a German company involved in iMUGS autonomy sub-project. vi

dotOcean dotOcean is a Belgian company working with the RMA on the swarming sub-project. vi, II, 2, 4, 13, 14, 21, 22

Estonian Military Academy The Estonian Military Academy is part of the iMUGS consortium. vi

galactic Galactic was the last distribution of ROS2 when iMUGS started. Galactic was deployed in 2021 and its End Of Life (EOL) is in novembre 2022. 13

GMV GMV is a Spanish company which is laeading command and control and interoperability sub-project of iMUGS. vi, 2

GTCT GTCT for GT Cyber Technologies OÜ is an Estonian defense company working on cyber technology. GTCT leads the cyber security sub-project of iMUGS. vi, 2

INSTA INSTA is a Finnish company that is carrying out the local swarming on the sub-project 2 of iMUGS. More information on www.insta.fi/en/en/. vi, 2, 4

KMW KMW for Krauss-Maffei Wegmann GmbH & Co. KG is a German company that is implementing the Manned-Unmanned Teaming sub-project of the iMUGS project. vi, 3

launch file In ROS, a launch file is a XML file (it may be a Python script in ROS2.) that allow to launch easly different scripts with different parameters. 8

LMT LMT for Latvijas Mobilais Telefons is a Latvian company which participates in various sub-projects of iMUGS. vi

melodic Melodic is the second to last distribution of ROS1. Melodic was deployed in 2018 and its EOL is in 2023. 7, 13

Milrem Robotics Milrem Robotics is an Estonian company that provides the overall coordination of iMUGS. More information on https://milremrobotics.com/. vi, V, 1, 2, 21

Nexter Nexter is a French company working on the autonomy sub-project of the aciMUGS. vi, 2

noetic Noetic is the last distribution of ROS1. Noetic was deployed in 2020 and its EOL is in 2025. V, 7, 13

Robotnik Robotnik is a Spanish company that sells "plug to play" ground vehicles. More information on their website www.robotnik.eu. By extension, the Robotniks refer to the SUMMIT-XL robots of the Robotnik brand bought by RAS-lab. II, 6–9, 13, 21–23

ROS ROS for Robot Operating System, is the most used middleware use in robotics. The function of a middleware is to handle all the threads, the communication and the launching of the scripts. IV, V, 6–11, 13

ROS1 ROS1 is the first version of ROS, released in 2007 for an Long Time Support (LTS) of the last distribution (noetic) ending in 2025. V, 7, 11, 13

ROS2 ROS2 is the latest version of ROS that takes time to replace ROS1, distributions are available since 2018. IV, 7, 10, 11, 13

rviz rviz is a 3D visualizer for the ROS framework, for more information, please see the wiki: www.wiki.ros.org/rviz. 8

Safran Safran Electronics & Defense is a French company which leads the autonomy sub-project of iMUGS. vi, 2

terminator Terminator is a shell interface that allows you to split the window to display several terminals. 8

THeMIS The THeMISes are the operational robots created by Milrem Robotics. vi, II, 2, 3, 20–22

thread A thread is a sequence of programmed instructions that can be managed independently and in parallel by a computer. V

topic In ROS, topics are named buses over which nodes exchange messages. Topics have anonymous publish/subscribe semantics, which decouples the production of information from its consumption. 7, 10, 11, 13

Assessment Report

RAPPORT D'EVALUATION
ASSESSMENT REPORT

Merci de retourner ce rapport par courrier ou par voie électronique en fin du stage à :
At the end of the internship, please return this report via mail or email to:

ENSTA Bretagne – Bureau des stages - 2 rue François Verny - 29806 BREST cedex 9 – FRANCE
☎ 00.33 (0) 2.98.34.87.70 / stages@ensta-bretagne.fr

I - ORGANISME / *HOST ORGANISATION*

NOM / *Name* Royal Military Academy

Adresse / *Address* 8 Rue Hobbema, 1000 Bruxelles

Tél / *Phone (including country and area code)* +32 2 441 36 73

Nom du superviseur / *Name of internship supervisor*
Emile Le Flecher

Fonction / *Function* Project Manager

Adresse e-mail / *E-mail address* Emile.LeFlecher@mil.be

Nom du stagiaire accueilli / *Name of intern* Hugo Yverneau

II - EVALUATION / *ASSESSMENT*

Veuillez attribuer une note, en encerclant la lettre appropriée, pour chacune des caractéristiques suivantes. Cette note devra se situer entre **A (très bien)** et **F (très faible)**
*Please attribute a mark from **A (excellent)** to **F (very weak).***

MISSION / *TASK*

- La mission de départ a-t-elle été remplie ? A B C D E F
  Was the initial contract carried out to your satisfaction?
- Manquait-il au stagiaire des connaissances ? ☑ oui/*yes* ☐ non/*no*
  Was the intern lacking skills?

  Si oui, lesquelles ? / *If so, which skills?* _____

ESPRIT D'EQUIPE / *TEAM SPIRIT*

- Le stagiaire s'est-il bien intégré dans l'organisme d'accueil (disponible, sérieux, s'est adapté au travail en groupe) / *Did the intern easily integrate the host organisation? (flexible, conscientious, adapted to team work)*

  A B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* Le stagiaire a su s'intégrer rapidement à l'équipe et adopter les pratiques du service.
Il a accomplit son travail avec beaucoup de sérieux et d'implication.

COMPORTEMENT AU TRAVAIL / *BEHAVIOUR TOWARDS WORK*

Le comportement du stagiaire était-il conforme à vos attentes (Ponctuel, ordonné, respectueux, soucieux de participer et d'acquérir de nouvelles connaissances) ?
Did the intern live up to expectations? (*Punctual, methodical, responsive to management instructions, attentive to quality, concerned with acquiring new skills)?*

A B C D E F

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* Le stagiaire a eu comportement satisfaisant et a montré un grand intérêt pour les differents projets du laboratoire, tant d'un point de vue pratique que théorique.

INITIATIVE – AUTONOMIE / *INITIATIVE – AUTONOMY*

Le stagiaire s'est –il rapidement adapté à de nouvelles situations ? A B C D E F
(Proposition de solutions aux problèmes rencontrés, autonomie dans le travail, etc.)

Did the intern adapt well to new situations? A B C D E F
(eg. suggested solutions to problems encountered, demonstrated autonomy in his/her job, etc.)

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* Le stagiaire était autonome et a su se former et acquérir les compétence nécessaires aux sujets liés à sa mission.

CULTUREL – COMMUNICATION / *CULTURAL – COMMUNICATION*

Le stagiaire était-il ouvert, d'une manière générale, à la communication ? A B C D E F
Was the intern open to listening and expressing himself /herself?

Souhaitez-vous nous faire part d'observations ou suggestions ? / *If you wish to comment or make a suggestion, please do so here* Le stagiaire a su demander des informations lorsqu'il en avait besoin et communiquer avec l'équipe sur ses avancés.

OPINION GLOBALE / *OVERALL ASSESSMENT*

❖ La valeur technique du stagiaire était : A B C D E F
Please evaluate the technical skills of the intern:

III - PARTENARIAT FUTUR / *FUTURE PARTNERSHIP*

❖ Etes-vous prêt à accueillir un autre stagiaire l'an prochain ?

Would you be willing to host another intern next year? ☑ oui/*yes* ☐ non/*no*

Fait à Brussels, le 23/08/2022
In _____, on _____

Signature Entreprise _____ Signature stagiaire
Company stamp _____ *Intern's signature*

Merci pour votre coopération
We thank you very much for your cooperation

Models

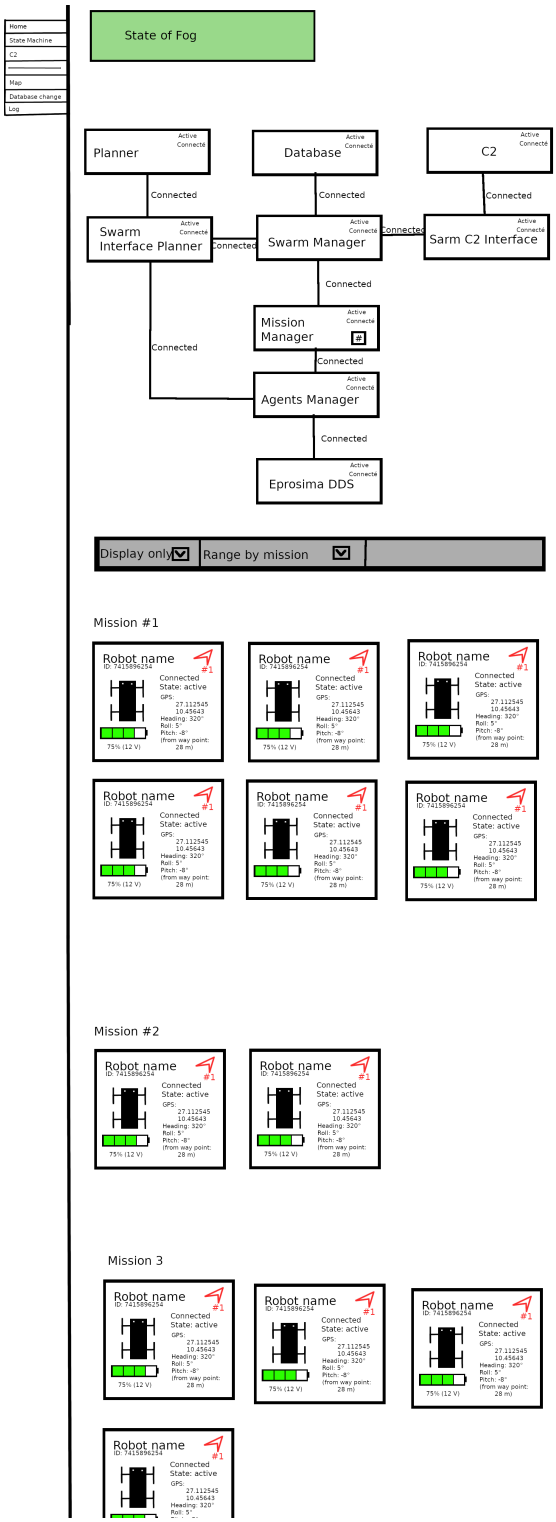

Figure 15: Model of the Home Page

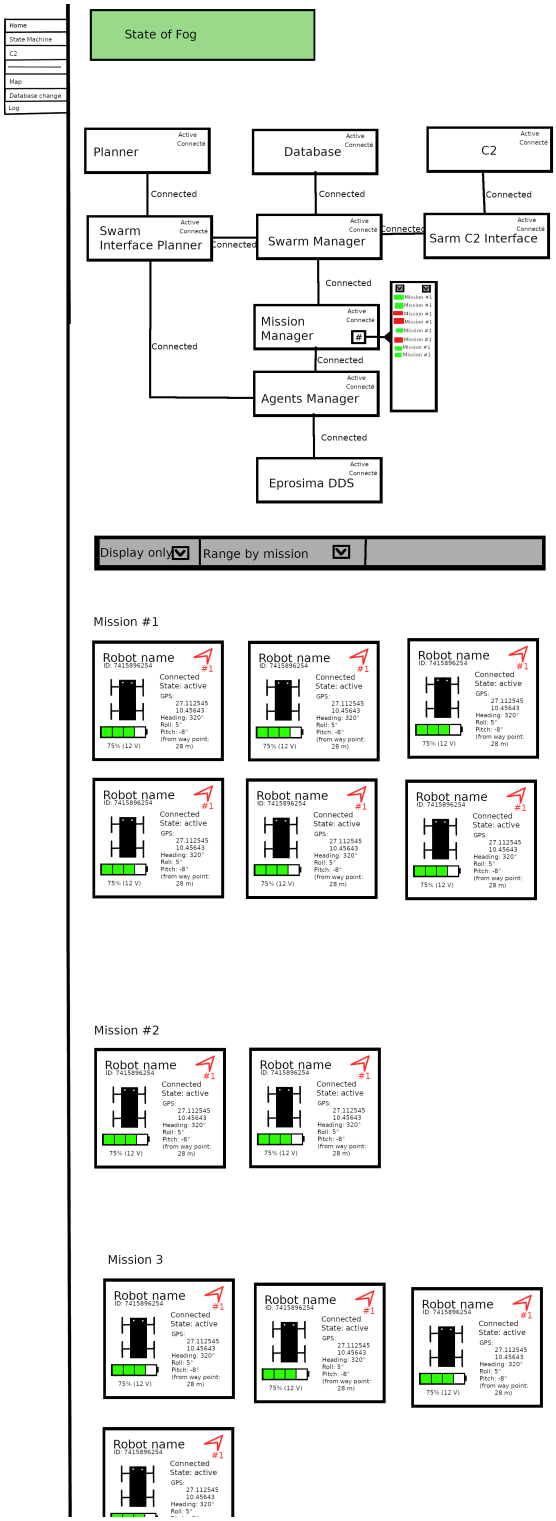

Figure 16: Model of the Home Page (with the required pop-up)

Figure 17: Model of the State Machine Page

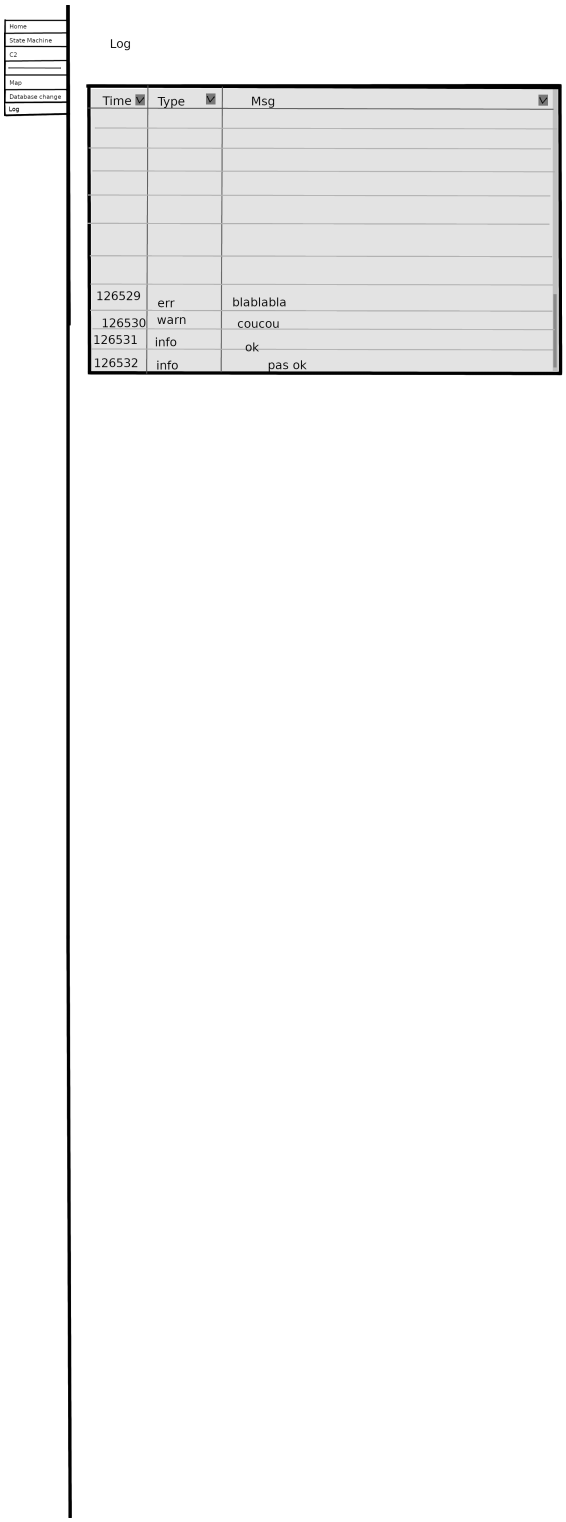

Figure 18: Model of the Log Page

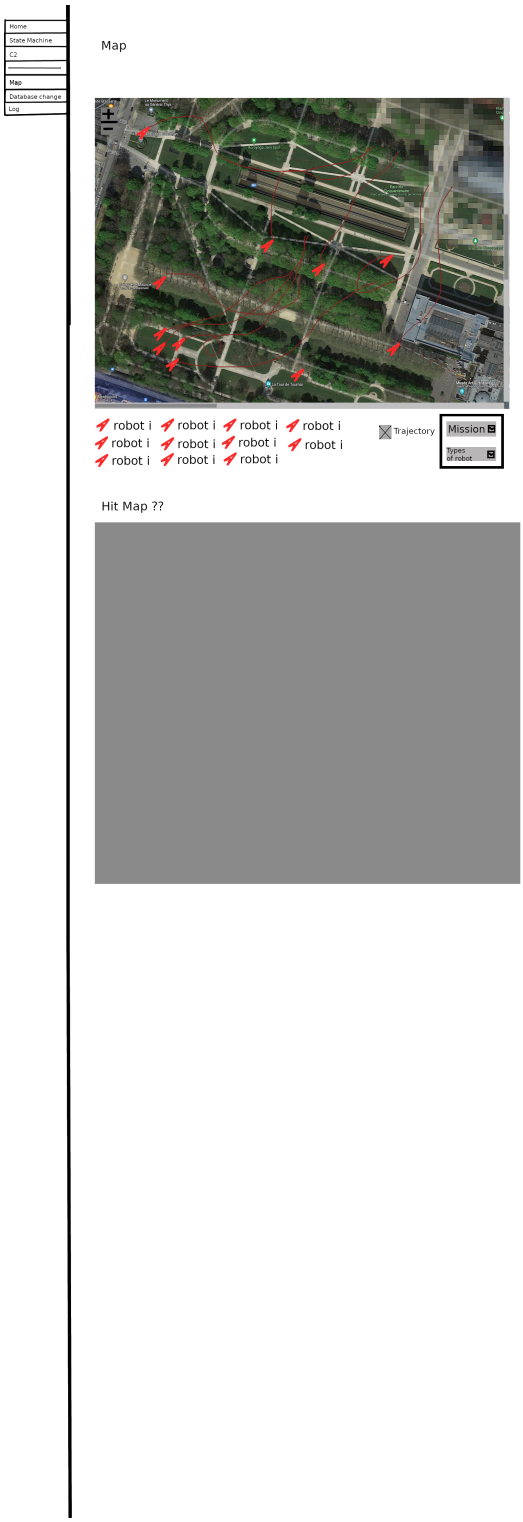

Figure 19: Model of the Map Page

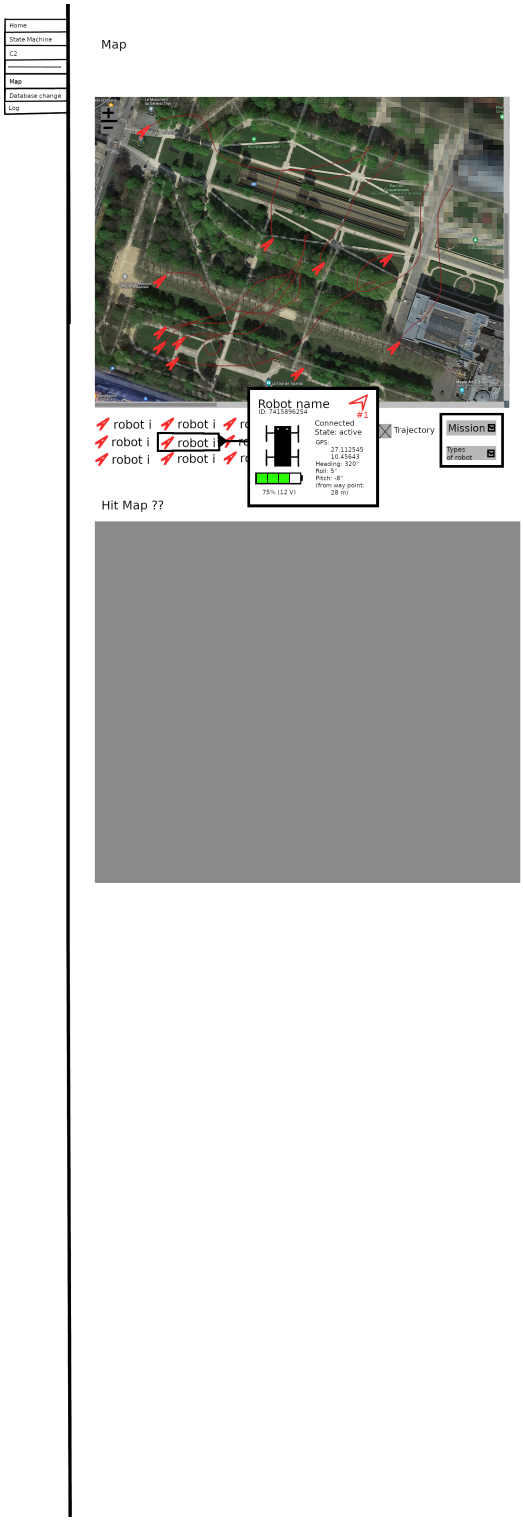

Figure 20: Model of the Map Page (with the required pop-up)

README.md

Throughout the internship, documentation has been an important point. The code is quite little commented in itself, since a lot of attention has been paid to the names of the variables and the application relies on an important architecture. Here is a compilation of the most important `README.md`.

imugs_hmi

Last edit: 20 July 2022 by Hugo Yverneau

A webserver HMI applies to the iMUGS project for the RMA.

The objective of the HMI is to create an user friendly interface for swarming development in ras-lab for the iMUGS project. This interface may be also used for public demonstration.

This project uses the package manager yarn. The web development use the library React. For the robotics aspect we use the package rosbridge_server available on ros1 and ros2. It uses the roslibjs package in particular roslib React package.

For deployment on Docker please directly refer to `./docker_readme.md`.

Installation

Deployment

Please refer to `./docker_readme.md`

Development

The App

1. install yarn

   ```
   sudo apt install curl #If curl in not installed yet
   ```

   ```
   curl https://raw.githubusercontent.com/creationix/nvm/master/install.sh | bash
   source ~/.profile
   nvm install v16.15.0
   nvm use v16.15.0
   npm install --global yarn
   ```

   versions details The project have been develop with

   ```
   $ node -v
   v16.15.0
   $ npm -v
   8.5.5
   $ yarn -v
   1.22.19
   ```

2. Clone

   Clone this repository in a `imugs_widgets` folder.

   ```
   mkdir imugs_widgets # if imugs_widgets does not exist yet
   cd imugs_widgets
   ```

```
git clone https://gitlab.cylab.be/rma-ras/modular_hmi_webserver/imugs_widgets/imugs_hmi
```

You may need to clone other repository, look on https://gitlab.cylab.be/rma-ras/modular_hmi_webserver/imugs_widgets to see which can be needed.

Working on a git branch:

```
git checkout <your_branch_name>
```

The API

Please refer to https://gitlab.cylab.be/rma-ras/modular_hmi_webserver/imugs_widgets/api_app/-/blob/master/README.md

The ROS connection with websockets

Websockets are use by ros to send and recive all the intels needed by the web app. You only need to install `rosbridge-suite` for the ros distributions that you whant to connect.

```
sudo apt-get install ros-$ROS_DISTRO-rosbridge-suite #Be sur to have source /opt/ros/<your_
```

VSCode Useful Extensions

- For React:
  - ES7+ React/Redux/React-Native snippets
  - vscode-styled-components
  - Prettier - Code formatter
  - Sublime Text Keymap and Settings Importer
- For git:
  - GitLens — Git supercharged
- For Docker:
  - Docker
- For ssh connection:
  - Remote - SSH
  - Remote - SSH: Editing Configuration Files
- For ROS:
  - ROS
  - For CMake and XML:
    - CMake
    - CMake Language Support
    - CMake Tools
    - XML Tools
  - For C++ and Python:

    * C/C++
    * Pylance
    * Python

Launch

Deployment

Please refer to `./docker_readme.md`

Ports

The ports needed for deployment on a sever are:

- `3030` for the app (refer to `./docker_readme.md` for further informations).
- `5050` for the API (refer to https://gitlab.cylab.be/rma-ras/modular_hmi_webserver/imugs_widgets/api_a
  /blob/master/README.md for further informations).
- `9000` for the websocket with the swarm manager (refer to `gui_app/src/utils/README.md` for further informations).

Development

The App

In `./gui_app` intall the dependecies declare in `package.json` with

```
yarn install
```

Then you can launch the applicaton with

```
yarn start
```

Those two commands are raising a lot warnings and errors, this is normal.

> If you want to undo the installation of dependency made by `yarn install` you can run (always in `./gui_app`)
>
> ```
> rm -r node_modules
> # rm -r yarn.lock # TODO is it useful? What yarn.lock do?
> ```

The API

Please refer to https://gitlab.cylab.be/rma-ras/modular_hmi_webserver/imugs_widgets/api_app/-/blob/master/README.md

The ROS connection with websockets

Websockets are use by ros to send and receive all the intels needed by the web app. You only launch the node for the ros distributions that you want to connect.

- Connection with the swarm manager:

```
ros2 launch rosbridge_server rosbridge_websocket_launch.xml port:=9000
```

- Connections with ros installation needed

  If you are using ROS 1:

  ```
  roslaunch rosbridge_server rosbridge_websocket.launch port:=9001
  ```

  If you are using ROS 2:

  ```
  ros2 launch rosbridge_server rosbridge_websocket_launch.xml port:=9002
  ```

Fill free to change the port, you must be coherent with `gui_app/src/utils/ros/websokets_ports.js`, refer to `gui_app/src/utils/README.md` for further informations

Continuous Integration & Continuous Development

TODO

Contribute

Roadmap

Burlk imporvement way:

- Create a complete modualr robots HMI
  - Fork this project to https://gitlab.cylab.be/rma-ras/modular_hmi_webserver/modular_hmi and refactor it to create a template
- Make a puclic application
  - Serve the application online
  - Create a developer and public mode
- Make the app more fancy
  - Refactor all the buttons to be in the same graphic style.
  - Look to all the text font
  - Make the app responsive
  - Create a dark mode
  - Make a page for unkow routes
- ROS connections
  - Refactor the ROS connection to be more modular
  - Make the connection for the services
  - Create a publisher component thet allow to publish any message on selected a topic.
- Connect to the database
  - Create a modular API in https://gitlab.cylab.be/rma-ras/modular_hmi_webserver/imugs_widgets/a
- Improve current components
  - Improve the module graph

        * Set up the connection with the ROS and/or the database to set Active and Connected sates by module.
        * Developpe the mission manager pop-up
            · Handle the mission dynamicly
            · Add a filter
            · Add a sort
    - Improve the robots overview
        * Add a filter
        * Add a sort
        * Handle the robot number
        * Handle the case with no ros connection
            · Console warning
            · History of last data
            · Handle a topic vacancy
    - Improve state machine
        * Handle the arrow
        * Make the state change dynamicly
    - Improve the map
        * Add trajectories (disablable with a check box)
        * Make the robot icons move with the robots
        * Add a nombre to the icon
        * Add the robot recap pop-up in the legend and on the map icone
        * Send a goal to a robot
    - Improve the log - Make it work with the topic `/swarming/log` - Improve robot page
- New components:
    - Create a page for each missions

Other

Getting started

To make it easy for you to get started with GitLab, here's a list of recommended next steps.

Already a pro? Just edit this README.md and make it your own. Want to make it easy? Use the template at the bottom!

Add your files

- [ ] Create or upload files
- [ ] Add files using the command line or push an existing Git repository with the following command:

```
cd existing_repo
git remote add origin https://gitlab.com/ras_lab/ras_robot_hmi_template.git
```

How to Contenarized an React App with Docker?

Requierement

`docker`

can be installed with

```
sudo apt install docker
```

`docker-compose`

can be instaled with

```
sudo apt install docker-compose
```

The files of the react app.

```
--gui_app
|--build
|--node_modules
|--public
|--src
|--useful
|package.json
|README.md
|...
```

Set Up (pass this section if you have cloned the gitlab repository)

Create a `.dockerignore` file to ignore the folders `node_modules` and `build` of `./gui_app`:

```
cd  gui_app
echo "node_modules
package-lock.json
build
" > .dockerignore
```

Create the `Dockerfile` in `./gui_app`:

```
cd  gui_app #if you are not alredy there
```

```
echo '
 # node image to have npm and yarn in the minimal image
FROM node:18.4.0-alpine
# Make app the work directory of the container
WORKDIR /app
```

```
#copy all the app code
COPY gui_app .

#install react dependencies
RUN yarn install --production --pure-lockfile --non-interactive --cache-folder ./ycache; rm

#build the app in order to serve it with better performance than with the developpement way
RUN yarn run build
RUN npm install -g serve
RUN rm -r node_modules src package.json yarn.lock README.md public
CMD ["serve", "-s", "build", "-p", "3030", "--no-port-switching"]
' > .Dockerfile
```

Create the `docker-compose.yaml` in `gui_app/..`:

```
cd .. #if you are in gui_app
```

```
echo "
version: '3.3'
services:
    gui:
        build: .
        # image: registry.gitlab.com/ras_lab/ras_robot_hmi_template:latest
        container_name: gui
        ports:
            - '3030:3030'
        # tty: interactive mode
        tty: true
" > .docker-compose.yaml
```

Build and launch the Docker image

Build the image with `docker-compose`

```
sudo docker-compose build
```

Create and launch the container

```
sudo docker-compose up
```

Enjoy your web app

Your react app could now be found on `<IP>:3030/` Where `<IP>` have to be replace by your sever (or computer) IP. (If you put the container IP, the app will only be available on local).

gui_app

Last edit: 19 July 2022 by Hugo Yverneau

This folder contains all the code and file util to run the web app. his organisation is automatically created by yarn.

- `node_module`

  Contains the compiled file for running the app in development.

- `public`

  ? TODO @Alexandra

- `src`

  Contains all the code and the datas. See `src/README.md` for further information.

- `package.json`

  Describe the package to install in `yarn install`.

- `yarn.lock`

  Contains the compiled file for running the app in development.

- Some document for the dockerization see `../docker_readme.md` for further information.

Getting Started with Create React App

This project was bootstrapped with Create React App.

Available Scripts

In the project directory, you can run:

`npm start`

Runs the app in the development mode.
Open http://localhost:3000 to view it in your browser.

The page will reload when you make changes.
You may also see any lint errors in the console.

`npm test`

Launches the test runner in the interactive watch mode.
See the section about running tests for more information.

`npm run build`

Builds the app for production to the `build` folder.
It correctly bundles React in production mode and optimizes the build for the best performance.

The build is minified and the filenames include the hashes.
Your app is ready to be deployed!

See the section about deployment for more information.

`npm run eject`

Note: this is a one-way operation. Once you `eject`, you can't go back!

If you aren't satisfied with the build tool and configuration choices, you can `eject` at any time. This command will remove the single build dependency from your project.

Instead, it will copy all the configuration files and the transitive dependencies (webpack, Babel, ESLint, etc) right into your project so you have full control over them. All of the commands except `eject` will still work, but they will point to the copied scripts so you can tweak them. At this point you're on your own.

You don't have to ever use `eject`. The curated feature set is suitable for small and middle deployments, and you shouldn't feel obligated to use this feature. However we understand that this tool wouldn't be useful if you couldn't customize it when you are ready for it.

Learn More

You can learn more in the Create React App documentation.

To learn React, check out the React documentation.

Code Splitting

This section has moved here: https://facebook.github.io/create-react-app/docs/code-splitting

Analyzing the Bundle Size

This section has moved here: https://facebook.github.io/create-react-app/docs/analyzing-the-bundle-size

Making a Progressive Web App

This section has moved here: https://facebook.github.io/create-react-app/docs/making-a-progressive-web-app

Advanced Configuration

This section has moved here: https://facebook.github.io/create-react-app/docs/advanced-configuration

Deployment

This section has moved here: https://facebook.github.io/create-react-app/docs/deployment

`npm run build` fails to minify

This section has moved here: https://facebook.github.io/create-react-app/docs/troubleshooting#npm-run-build-fails-to-minify

Code Architecture

Last edit: 20 July 2022 by Hugo Yverneau

- `index.jsx`

  Is the main file of the app.

  This is where the

  - react dom client;
  - context providers;
  - the page global structure (side bar, header, footer, current page);
  - the routes

  are handled.

  This file is made to be the more modular as possible. To add or modify a page look `./utils/routes/routesList`, further information in `utils/README.md` or `pages/README.md`.

- `structure`

  This folder regroups the `SideBar`, the `Header` and the `Footer` components. Those components are normal components as those in the `components` folder. They are separate because they are only call in `index.jsx` alongside a page component. Refer to `structure/README.md` for further information.

- `pages`

  This folder regroups all the page component. The point of a page component is to call the right components in the right way. Refer to `pages/README.md` for further information.

- `components`

  This folder regroups the modular components. Each component can be called from a page or another component simply (watch out the props). Refer to `components/README.md` for further information.

- `assets`

  This folder contains the static data, as the images. Refer to `assets/README.md` for further information.

- `utils`

  This folder regroups all the code than can be called in every component as useful functions, connections intels or custom hooks. Refer to `utils/README.md` for further information.

Architecture of components

All the React components are regrouped on three folders: `components`, `pages` and `structure`. A component is a `.jsx` file in an eponymous folder in one of the three previous folders, please respect the PascalCase naming convention.

```
--src
|--components
| |--AComponent
| | | AComponent.jsx
| | | few other files
| | ...
|--pages
| |--APage
| | | APage.jsx
| | | few other files
| | ...
|--structure
| |--AComponent
| | | AComponent.jsx
| | | few other files
| | ...
| ...
| README.md
```

All the modular feature components are in `components`, further details in `../components/README.md`.

All the components that are creating a page accessible by a route are in `pages/README.md`, further details in `pages/README.md`.

All the components always on the web site, beside the current, page are in `structure/README.md`, further details in `structure/README.md`.

Please pay attention to keep `few other files` few. Too many other files may mean that you need to rethink your architecture and maybe add one or several components in `components` (see `components/README.md` if needed). Moreover, all CSS style is handle by styled components in `utils/style`, see `utils/README.md` for further information.

components

Last edit: 20 July 2022 by Hugo Yverneau

Architecture

A component is a `.jsx` file in an eponymous folder in the folder `components`, please respect the PascalCase naming convention.

```
--src
|--components
| |--AComponent
| | | AComponent.jsx
| | | few other files
| |--AnOtherComponent
| | | AnOtherComponent.jsx
| | | few other files
| | ...
| | README.md
| ...
```

Please pay attention to keep `few other files` few. If you have too many files, it may mean that you have to rethink your architecture and perhaps add one or more components.

`InteractiveMap`

TODO: finish the component The component is decomposed in 2 subcomponents: - `MapBackground.jsx` - The map is based on mapbox API. - `MapLegend.jsx` - To add a location in the reset go to `../utils/intels/mapIntels.js`.

`ListOfTopics`

- `ListOfTopics.jsx` This component provides a modular display for several ros topic. There are mutiple options in the probs in order to allow different use.
- `ListOfTopicsBody.jsx` The main code of ListOfTopics component.
- `TopicAdder.jsx` Create a modal popup in order to allow a to make a selection during a refresh of ListOfTopics component.
- `addTopic2Echo.jsx` This function takes a useState and a useState setter and return a function which needs the new entry to update the useState the right way.

`LogTable`

- `LogTable/Table.jsx` Is a compilation possibilities of `react-table` React

component of example on https://github.com/ipsjolly/react–table-tutorial-app (https://www.freakyjolly.com/react-table-tutorial/).

TODO @Alexandra

`ModulesGraph`

This component shows a dynamic (static for now) representation of the connections between modules. To add a module, go to `../utils/intels/modulesIntels.jsx` and precise its name and position. A `popup` is also available to enumerate the missions for the Mission Manager

This graph will have to become dynamic when the connection conditions are determined.

`PublisherMap`

First try for a component which publishes a goal on a topic ros when the user clicks on the map. Here only for development, can be remove. TODO @Alexandra : remove it.

`RobotsRecap`

TODO @Hugo This component is a block containing the essential information of a robot. The information is displayed dynamically via a websocket connection. The file `RobotsRecap.jsx` calls each component created in the `RobotRecap.jsx` file. The link with ros is made in the RobotsRecap file and the information obtained on the robots are passed in props to the concerned component:

- `Energy.jsx` displays data about the consumption and the battery. The `ProgressBar.jsx` component is called here to display the consumption.
- `RobotIntels.jsx` displays data about the connection of the robot, its position and next way point.

Furthermore, the robots are all clickable to have access to further information. The component then called is `RobotInfo.jsx` and displays those data on another route.<

To add a robot, please refer to `../utils/README.md`.

`StateMachine`

This component is a dynamic (static fo now) state machine graph. The tabs allow you to navigate between the missions to see the corresponding state machine. To add a mission or a state block, please refer to the file `../utils/intels/missionsIntels.jsx` or the file `../utils/intels/statesIntels.jsx` respectively.

The arrows are static and are defined in the file `../components/StateMachine/MissionsStateMachine.jsx`. (TODO @Alexandra refactor the way of doing arrow)

The current and previous states will be obtained from the database. (TODO @Alexandra)

Structure

<<<<<<< HEAD *Last edit: 19 July 2022 by Hugo Yverneau* ======= >>>>>>> d2a1eba0d7a1137c6b39f81701aa2e31f728981a

Last edit: 14 July 2022 by Hugo Yverneau

<<<<<<< HEAD This folder regroups the `SideBar`, the `Header` and the `Footer` components. Those components are normal components as components in `../components`, they are separate because they are only call in `index.jsx` alongside a page component. ======= This folder includes the `SideBar`, the `Header` and the `Footer` components. Those components are components as components in `../components`. However, they are separated because they are only called in `index.jsx` as for a page component. >>>>>>> d2a1eba0d7a1137c6b39f81701aa2e31f728981a

`SideBar`

The sidebar is made with cdbreact. For more details, please refer to `github.com/azouaoui-med/react-pro-sidebar`. To add a page in the sidebar, add a new route and use the sideBarDisplay boolean in the file `../utils/routes/routesList.jsx`. the version numbering follows the next convention: "v" + #version_number_on_the_main_branch + "." + #number_of_the_version_of_the_website

`Header`

The header is composed of three buttons and an image. Two of the buttons are created directly in the file `../structure/Header/Header.jsx` and the third one (the `stop button` which publishes 0 on the cmd_vel topic when it is activated) is called from this same file.

`Footer`

An arrow allows you to scroll directly to the top of the page.

Pages

Last edit: 19 July 2022 by Hugo Yverneau

Pages are normal React components. We chose to put them in a different folder than the othe components (which are in `../components`) because they are not used the same way. A component in the folder `pages` is call allow alongside the structure (see `../structure/README.md`) and it is only call by the route in `../index.jsx` (may exeption exist? see `Robots/Robots.jsx`?).

Pages are normal React components. We chose put them in a different folder than the other components (which are in `../components`) because they are not used the same way. A component in the folder `pages` is call allown alongside the structure (see `../structure/README.md`) and it is only call by the route in `../index.jsx` (may exist exeption? see `Robots/Robots.jsx`?).

Architecture

```
--src
|--pages
| |--MyFirstPage
| | | MyFirstPage.jsx
| | | few other files
| |--MySecondPage
| | | MySecondPage.jsx
| | | few other files
| |--MyThirdPage
| | | MyThirdPage.jsx
| | | few other files
| | ...
| | README.md
| ...
```

Please pay attention: very `few other files` are rarely useful in a page folder. This can be util during the development but if you are doing a lot of things here that surly means that you need to rethink your architecture and may be add one are sevral components in `../components` (see `../components/README.md` if needed).

Add a new page

To add a new page, you must add in `../utils/routes/routesList.jsx` in order to make it call by `../index.jsx` the right way. Please refer to `../utils/README.md` for further explanation.

Utils

Last edit: 19 July 2022 by Hugo Yverneau

This folder regroups all the code than can be cold in every component.

`contexts`

Contains all the providers for contexts. - A context is a way in React to make available a variable everywhere inside a provider. > More information for French speakers can be found on this openclassroom tutorial: https://openclassrooms.com/fr/courses/7150606-creez-une-application-react-complete/7256029-partagez-vos-donnees-avec-le-contexte-et-usecontext - More of them are using local storage (this means cookies in the cache) to keep their value even when the page is reloaded. - For now, to allow an acces to each context everywhere in the application, all the provider are call at the top level in `index.jsx`. - To upgrade the variable scop handling you can use a state manager as Redux
> A openclassroom tutorial for French speakers: https://openclassrooms.com/fr/courses/7150626-utilisez-le-state-manager-redux-pour-gerer-l-etat-de-vos-applications.

`hooks`

Contains all the custom hooks. - To have more details on context custom hooks refer to the previous paragraph. - There is none other custom hooks for now.

`intels`

Contains all the intels used by the concerned components.

- `mapIntels.js` to add a location in the map.
- `modulesIntels.js` to add a module in the modules graph.
- `robotsIntels.js` to add a robot.
- `statesIntels.js` to add a states in the states machine.

`maths`

Contains all mathimatical function than can be useful. - `eulerQuaternion.js` convert Euler angles to quaternions and reverse from https://en.wikipedia.org/wiki/Conversion_between_quatern source equations.

`ros`

Contains all the data or function needed to make the connection with ROS. - `topicsTypesHandled.js` export an array of object that allow to convert stringify json in more lisible string. - `websokets_ports.js` Handle the url and intel of the websockets.

- `WSfunction.j` Usefuls functions for websocket handling.

`routes`

Contains all the intels about routes.

- `routesList.jsx` Contains the list of all the routes, `index.jsx` makes a map of this list to create the routes. You can add dynamic routes using useParams.

`style`

For the CSS style, this project use styled components. All the styled components which are used in more than one script are grouped in this folder.

- `CommunStyle.jsx` Regroup all the stylde components.
- `GlobalStyle.jsx` Setup the global style.

assets

Last edit: 15 July 2022 by Alexandra Haim

This folder contains all the images used in the components. It's a way to gather the global static assets.

The images are classified in folders:

- `app6` for the symboles to put on the map and in the RobotRecap component,
- `headerImages` for the logo of the RMA,
- `missionsStates` so that missions have their color linked to their state (red: error, orange: waiting for intervention, green: all is fine),
- `robotImages` for the RobotRecap component.

If you need additional pictures, please put them in the corresponding folder, or create a new one.