R o b o – H i v e

A blackboard distributed fleet manager

Fontys University Of applied sciences – ***Mechatronics & Robotics Lab***

Internship Assignment Report, Eindhoven, September 2020 – February 2021

Hussam Ayoub

Version 1.0

Internship report - Fontys university of applied sciences

HBO-ICT: English Stream

Data student:	
Family name, initials:	Ayoub,H
Student number:	2953536
project period: (from – till)	28-08-2020 01-02-2021
Data company:	
Name company/institution:	Kenniscentrum Mechatronica & Robotica
Department:	-
Address:	De Rondom 1, 5612 PA
Company tutor:	
Family name, initials:	Negrete Rubio,P
Position:	Teacher - researcher
University tutor:	
Family name, initials:	Shaghelanilor,M
Final Report:	
Title:	design, development and testing of a blackboard- based distributed fleet manager
Date:	01-09-2020

Approved and signed by the company tutor on 15 – 01 – 2021

Foreword

This document is the final report of my internship project, carried out at Fontys Kenniscentrum Mechatronica & Robotica, Eindhoven of my ICT & Technology studies at the Fontys university of applied sciences in Eindhoven. The mission of this internship project was to research, design, develop, and test a prototype of a blackboard-base distributed fleet manager in order to execute tasks for multiple robotics platforms from different vendors in highly dynamic working environments. This report describes the process and results of the described assignment.

Since Robotics as an advanced and challenging field of study has always been my passion, I consider this internship to be one of the most important steps of my journey to becoming a robotics engineer.

During this internship I learned a lot. Both in the technical, and in the professional fields. Starting by planning a project, managing and prioritizing my tasks, working on highly complex technologies and topics, ending by functioning within a company and a team.

I would like to thank my company tutor Mr. Negrete Rubio,Pablo for believing in me by offering me this great opportunity, and for his constant help during the past five month.

I would also like to thank my university mentor Mr. Shaghelani Lor,Mikaeil for his guidance and constructive advice during this internship. Without those people I would have not been able to accomplish what I have done in this project.

Finally, I would like to thank all employees of Fontys Mechatronics and robotics lab for the pleasant working atmosphere, and the friendly times.

Hussam Ayoub,

Eindhoven, January 2021.

Table of Contents

Summary

This document is a detailed report of an internship project taking place at Fontys mechatronics and robotics lab in Eindhoven, the lab aims to help the research community by overcoming many challenges in different projects.

The goal of this project is to accomplish challenges related to having different robots interacting with each other's in highly dynamic working environments. The assignment is to implement a blackboard-based execution of tasks for multiple robotics platforms known as "orchestration" in a virtual environment.

The project is considered to be a highly complex system, due to that the project was split into many phases. In the beginning the project was simplified in order to specify the requirements and list the following phases. Later a research was conducted to list the characteristics which could define a robot, and to further understand the architecture of the blackboard design.

Based on the results from the previous phases the design phase took place, in order to create the system architecture including the diagrams and the communication system. After the design approval the implementation phase was started to create the simulation environment and the programs that form the entire system.

The final phase was testing the created system functionality and reliability with customized test cases and scenarios.

The project resulted into a working prototype, that enables adding tasks to the system and assigning those tasks to the robots based on the unique tasks and robot types, and on the cost of executing a task by a certain robot. Further the tasks were executed in the simulation environment.

Creating a unified fleet manager for different robotics platforms is found to be an efficient method of having different robots interacting with each other's in highly dynamic working environments, if the robot offers a communication interface which can be utilized by the fleet manager to send commands and receive updates.

It was recommended to further research the possibility of using machine learning algorithms to determine the cost of executing tasks by robots. Since it is believed that such algorithms will provide more precise results if provided with sufficient data.

Glossary

ROS : ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

ROS package: A package might contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module. The goal of these packages it to provide this useful functionality in an easy-to-consume manner so that software can be easily reused.

ROS node: an executable file within a ROS package. ROS nodes use a ROS client library to communicate with other nodes. Nodes can publish or subscribe to a Topic. Nodes can also provide or use a Service.

ROS topic: Topics are named buses over which nodes exchange messages.

ROS Message: a simple data structure, comprising typed fields.

Gazebo: an application that offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments.

Rviz!: a 3d visualization tool for ROS applications. It provides a view of your robot model, capture sensor information from robot sensors, and replay captured data. It can display data from camera, lasers, from 3D and 2D devices including pictures and point clouds.

Urdf: the Unified Robotic Description Format is an XML file format used in ROS to describe all elements of a robot.

Dae: A DAE file is a 3D interchange file used for exchanging digital assets between a variety of graphics programs.

Sdf: An SDF file contains a compact relational database saved in the SQL Server Compact (SQL CE) format, which is developed by Microsoft.

Qt: is a framework to create innovative devices, modern UIs & applications for multiple screens. Cross-platform software development at its best.

Rqt: is a software framework of **ROS** that implements the various GUI tools in the form of plugins.

1. Introduction

It is amazing how nature extends a helping hand in development of different technologies. It can be adapted from anatomy like the aerodynamic shape of planes inspired by birds or like in the case of this project – the hive behavior. Hive mind or shared intelligence in the meaning used in this project is the collective mental activity expressed in the complex, coordinated behavior of a colony of insects as comparable to a single mind controlling the behavior of an individual organism.

Now imagine bees as robots and hive mind as the computer system. One big organism working flawlessly in order to carry out assigned tasks, communicate with each other and make decisions without the need of external intervention. Working as one, consisting of a great number of autonomic parts. At the Fontys mechatronics and robotics lab inspired by the concept of collective intelligence, the idea of a blackboard base distributed fleet manager that enables the execution tasks on different robotics platforms in highly dynamic working environment was born.

To bring such idea to life a project by Mr.Negrete, Pablo in the form on an internship assignment was created in order to investigate the possibilities of such concept. The goal of this project is to research, design, develop and test a prototype of such system.

By working on bleeding edge technologies, and solving complex problems, the project offers a great opportunity for ICT students who are willing to specialize in the field of robotics and develop advanced skills, starting by research, system architecture design, developing robotics and simulation programs and ending by conducting tests.

More information about the approach and the process of creating this system can be found in this report.

Chapter 2 provides information about the company,

Chapter 3 gives all the details about the assignment,

Chapter 4 describes the development process and results of the simulation,

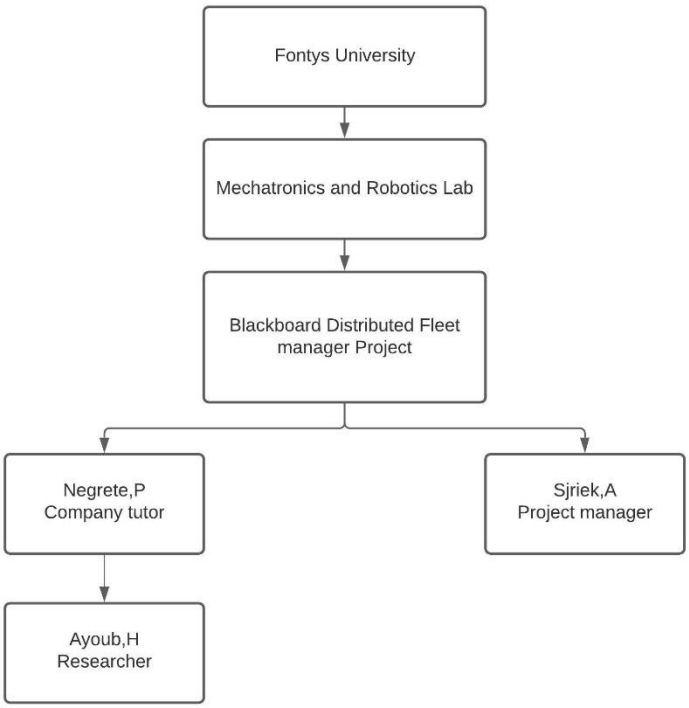
Chapter 5 summarises the final findings and explains the results.

2. About the company

Fontys is one of the largest universities of applied sciences in the Netherlands and located in the most innovative region of our country and perhaps the whole of Europe. It is the most exciting possible place to be for anyone with an interest in technology, entrepreneurship and creativity. Students of more than 100 nationalities study at one of our campuses.

Fontys mission is to provide inspiring, challenging and outstanding higher vocational education and to conduct practical research that is truly meaningful to society. With 477 bachelors, masters, and other programs, 44.320 students, 7.697 graduates per year, 5.200 international students, 100 countries, and 4.938 employees.

The Fontys Mechatronics and Robotics Lab “Lectoraat” works closely with the industry, the lab helps the research community to overcome different challenges by participating in many projects at a national or European level. Since students and teachers are involved in the research, the knowledge and experience from these projects is an important factor in the education process. At Lectoraat many projects take place in various fields such as robotics, vision, mechatronics, embedded technology and more.



3. Assignment overview

This assignment is a part of the Holland Robotics Logistiek “OpZuid” a project taking place at the Fontys Mechatronics and Robotics Lab. Lectoraat is aiming to accomplish challenges and answer questions related to having different robots interact and execute tasks with each other in highly dynamic environments.

3.1 Initial situation:

Currently the logistics sector is increasingly looking for autonomous solutions, often in the form of autonomous Guided Vehicles technology. But only an AGV usually does not offer the complete solution that is desired to further automate and robotize. Cooperation between AGVs cannot be taken for granted, while they can be complementary.

Orchestration in robotics is a field that is still being developed, it is currently vendor specific and limited to a fleet manager, which allows to schedule vendor specific robots to perform tasks. The limitations fleet managers are evident when there are different vendor robots sharing the same working environment, also, it does not consider the possibility of robots sharing tasks or collaborating, each robot receives a specific task, and the fleet manager ensures that tasks will be executed for that robot or another in case something happens. Being able to use orchestration in robotics for different vendors and types of robots, while allowing robots to be scheduled based on their capabilities and tasks to be created or posted while sorted in a priority based, creates a solution that allows for having a better non-vendor specific autonomous working environment.

3.2 Assignment description:

There are different architectures that can allow orchestration in robotics, those architectures can be separated in two main groups, distributed and centralized. Fleet managers are centralized, which means that an external computing unit schedules and tracks the robot's tasks execution. Examples of distributed systems could be a blockchain based architecture and a swarm-based architecture.

Lectoraat is focused on creating a virtual proof of concept of a blackboard-based distributed fleet manager which allows execution of tasks for multiple robotics platforms.

It is desired to research, design, develop, and test a prototype of the blackboard-base distributed fleet manager in order to execute tasks for multiple robotics platforms in a virtual environment while considering the capabilities of each robot (type of robot, kinematics, payload, speed, battery capacity, etc.) for scheduling a robot to a certain task.

3.3 Project constrains:

With many constraints this project must be done within 20 weeks starting September 2020 ending February 2021. The prototype will be implemented and tested in a virtual environment with many limitations when it comes to simulating certain task execution such as picking objects, energy consumption. The product is highly dependent on early phase results, such as research, data gathering, analysing, brainstorming and requirements specification. Working on a bleeding edge technology has its disadvantages when it comes to finding resources or solutions for similar challenges.

This project is simulated using ‘gazebo’, a simulation environment provided by ROS. The system architecture is designed using UML standards and implemented using Python and C++ programming languages. Source code will be written in visual studio code and compiled into ROS nodes and packages using catkin. All the communication in this project will take place over ROS topics with no additional communication protocols created specifically for the purpose of this project.

3.4 Planning:

When it comes to managing this project Scrum method is chosen since it is known to be ideal for self organizing teams, adapting to changes, continued improvement and sustainability. This project consists of seven phases each phase in a category of multiple actions. Below is a short description of each phase.

Orientation: The orientation phase focuses mainly on simplifying and understanding the current situation listing the project requirements and planning the project phases.

Research:

This phase focuses mainly on two research topics:

- Investigating the characteristics of a set of industrial robots.
- Researching the blackboard architecture model.

UML diagram design: Designing the UML diagrams, Using the research results from the previous phase will take place in this phase.

Cost calculation algorithm: When a new task is created the available robots will receive the task details. Each robot will run an algorithm to calculate the cost of executing the given task, then send the results back to the blackboard for a decision to be made. This phase focuses on creating the mentioned algorithm.

Modelling the virtual environment: The project will be tested in a virtual environment that is identical to the physical environment described by the client. Modeling the virtual environment will take place in this phase.

Implementing UML architecture: Using the products of the previous phases the project will be implemented in this phase.

Test phase: The test phase focuses on creating the test cases to be executed during the phase.

Research strategies.

More information about planning this project can be found in the attached Project plan.

As mentioned before in the project constraints this prototype is highly depending on the results of early phases and research. The DOT framework research methods and strategies used to reach the results in each phase are listed in the table below.

phase	method	Strategy
Orientation	Field	Task analysis
		Stakeholder analysis
		Problem analysis
		Explore user requirements
		Interview
	Workshop	Requirement prioritization

Research	Library	Available product analysis
		The community research
		Design pattern research
	workshop	IT architecture sketching
Cost calculation	Library	Expert interview
Testing	Lab	System test
		Computer simulation

4. Process and Results

4.1 Orientation

The orientation phase is the one of the most important phases in this project where the planning, phasing, task analysis, and requirements specification is happening.

Planning and phasing this project is a straightforward process which depends on the load of each phase.

The real challenge is task analysis and requirements specification where most of the time was spent in the form of client meetings and brainstorming sessions. A mock-up of this system was sketched on paper in order to have a full understanding of how the system should work, first step was to find all objects that are considered crucial for the system to work, first version included a black board a robot a user interface and a task, later some objects were added to the system such as a controller which is responsible for executing the tasks a simulator to visualise the robots and the environment.

After specifying the main objects of the system, the next step was to work on the communication between those objects, for example the user interface should be able to communicate with the black board in order to send tasks and with the robot object to display information about the robot state or simulator a change in the robot state. The robot and the blackboard can communicate in order to send, assign and receive tasks. the robot can communicate with the simulator to visualise the robot executing tasks.

Since the main objects and the communication channels of this system were specified the next step was to create an ideal scenario where a task is created and executed successfully. it was clear that a task will be created within the user interface then sent to the blackboard. When the blackboard receives a task, it will be broadcasted to all the online robots. Each robot will receive the task and run a cost calculation function which will result in a number between 0 and 10 to represent how much it would cost the robot to execute this task then it will send the result back to the blackboard. The black ball will decide to which robot should this task be assigned. The robot will be updated with an assignment then the controller within the robot will analyse the task, split it into smaller steps and start executing it, at the end when the task is successfully executed the robot will update the blackboard.

One of the most important factors of this system is reliability. That's why many questions were created to specify what challenges and problems this system could face. Multiple scenarios were created to simulate those situations that needed to be considered.

First scenario was what if the blackboard crashed. There are many options to overcome this problem such as having a backup blackboard or by enabling one of the robots to run an instance of the blackboard. After some discussions for the client, it was decided that each robot can run an instance of the blackboard in case the blackboard went offline.

Second scenario was what if the robot goes offline during the execution of a task. The solution chosen for this problem was to assign an estimated time to finish the task by the robot. And for some tasks with high priority the robot should update the blackboard with an estimated time to finish each step of the current task. if after the estimated time the blackboard still didn't receive an update it will assume that the robot is offline, and it will try to handle that situation with many ways one of them is story assigned the task to a different robot.

Third scenario was to create a task that could not be executed by one robot. An example of dad is a higher payload than any of the robot’s payload. In this case the blackboard should analyse the task and split it into sub-tasks which will be assigned and executed by multiple robots.

Another scenario what's to create a highest priority task while all the robots are busy executing other tasks. This has added an option for the blackboard to send a message to the robots to cancel the current tasks and assign a new task with higher priority.

Many more scenarios were created in order to list the challenges that the system could face, but for the scope of this assignment and because of the time-constrained it was agreed with the client to address only the above-mentioned problems. The Orient result of how the system should work ideally. And what are the main components and parts of the system and how do those parts communicate.finally, what kind of problems could this system face and which are the best solutions for each problem.

4.2 Research

4.2.1 Robot characteristics research:

One focus of this project is to create a proof of concept of a blackboard-based orchestration which can be used with centralized or distributed systems, while working with several robots from different vendors. To achieve this a research question was created to list the crucial characteristics of the robots. This part of the research is intended to answer the following question:

Which characteristics define an industrial robot?

Sub questions are created in order to help answering the main question:

What are the types of industrial robots?

What are the use cases of warehouse robots?

What are the characteristics of each type of the robots?

What are the common characteristics?

What are the unique characteristics?

The results of this research are used to create a database of the characteristics which will be a part of the task cost calculation, and the task filtration. The question will be answered using multiple methods of the library and workshop research strategies.

Hypothesis:

It is expected the following characteristics should be listed in order to define a robot:

Communication interfaces,

Movement speed,

Payload,

Unique features.

Delineation:

No sketches or designs will be used in this section, tables will be used to present the results.

Types of industrial robots:

There are many available types and categories of industrial robots available in the market. After studying a wide group of the currently available products in the market using the available product analysis research method the table below was created with three main categories of the most used industrial robots' types and subtypes

Stationary	mobile	Heterogeneous
Articulated robots	Land-based wheeled robots	A combination of multi robot systems
Cartesian robots	Land-based tracked robots	
SCARA robots	Land-based legged robots	
Delta robots	Air-based robots	
Polar robots		
Cylindrical robots		

Each of the categories lists different robots with common and unique characteristics. For the purpose of this project after agreement with the client a decision was made to disregard the stationary category. The characteristics of the included robots in the stationary category will be listed in the research still, since they are included in the heterogeneous robot's category.

Use Cases of warehouse robots:

One focus of this assignment is for the robots in this system to be able to work in highly dynamic environments such as warehouses, in order to find the most important robot characteristics to be listed in the database it is important to analyse the use cases of the robots. After studying the available robotics products for warehouses the table below was created to list three main categories of robotics in warehouses and their most common use cases, some categories are disregarded because of the non-relativity to this project such as stationary robots, and collaborative robots which are mostly operated by humans.

Automated guided vehicles "AGVs":	Automated storage and retrieval systems "AS/RS"	heterogeneous robots
Transporting materials	Storage, and retrieval of goods	Picking and palletizing

Robotics characteristics:

After categorizing the robots intended to be used in the system and listing some of their use cases a set of available products of each industrial robots type was studied separately. For each product the studied product and the characteristics were listed in the table below.

articulated	cartesian	SCARA	Delta	Cylindrical	Mobile
UR5 Denso robots	Newmark-sys series xy	Shibaura THE600	Atom Robot D3W series	St-robotics R19E	Kuka kmp 1500 sharp agv xf series 6river -chuck
Payload Repeatability Arm reach Weight Accuracy	Travel range Encoder Unidirectional repeatability Payload	Payload Working envelope Arm reach Weight	Axis Payload Energy consumption	Working range Payload Repeatability Accuracy	Footprint Weight Payload Max velocity straight Max velocity diagonally

Energy consumption Mounting options.		Accuracy Energy consumption Mounting options.	Rotation range Repeatability Accuracy		Battery capacity Charging time Interfaces Communication system Sensors

Common characteristics:

Below is a table of the common characteristics for both the stationary and mobile robots.

stationary	mobile
Payload Repeatability Accuracy Energy consumption Mounting options	Footprint Weight Payload Max velocity straight Max velocity diagonally Battery capacity Charging time Interfaces Communication system Sensors

Unique characteristics:

Mobile robots mostly share the same characteristics. While stationary robots differ when it comes to defining their working envelope.

Conclusion:

There are many categories and types of industrial robots in today's industrial environment, for the scope of this project after agreeing with the client the types of industrial robots which will be used are automated guided vehicles "AGVs", and heterogeneous robots which include land-based, and articulated robots.

One robot of each type will be duplicated with different characteristics for the simulation purposes.

4.2.2 Blackboard architecture

This research topic was created to have a better understanding of the blackboard software design pattern the research question desired to be answered is:

How to implement the blackboard architecture for different robots and tasks?

Sub questions were created to help answering the main question:

What is the blackboard architecture?

How to use the blackboard architecture for distributed decision making?

How to use the blackboard architecture for centralized decision making?

Hypothesis:

It is expected that the blackboard architecture model consists of a main class where decisions are made after collecting data from multiple knowledge sources.

Delineation:

UML diagrams will be used in this section to visualize the blackboard architecture model.

The blackboard architecture:

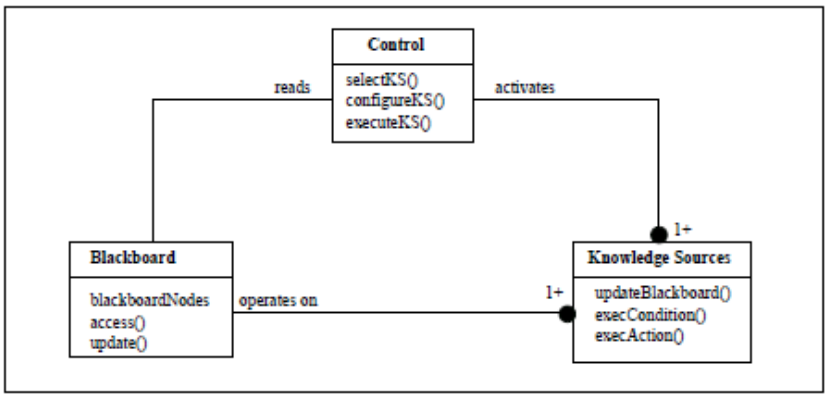
The community research method was adapted to answer this question. A study at the knowledge systems laboratory at the department of computer science in Stanford university describes the blackboard model for problem solving.

According to the study mentioned above the blackboard model is usually described as consisting of three major components:

The knowledge source: it is needed to solve the problem; it is partitioned into multiple knowledge sources which operate separately and independently.

The blackboard data structure: where the problem-solving data are kept in a global database, the blackboard. Knowledge sources produce changes to the blackboard which lead incrementally to a solution to the problem. Communication among the knowledge sources take place solely through the blackboard.

Control: the knowledge sources respond opportunistically to changes in the blackboard system.



Centralized and distributed decision making:

In the centralized case when a new task is created the blackboard will receive robot specific data "e.g., current state, energy level,", the received data will be considered in the decision-making process which will be done exclusively by the blackboard. In the distributed case when a new task is created each robot will receive task description data, the robot/knowledge source will use the received data and the available robot data to calculate the cost of executing the task. Then the blackboard where the final decision is made will be updated with task execution cost which is considered a part of the decision making.

Conclusion:

After studying the blackboard architecture model, it is clear the robots in this project will form the knowledge source which will update the blackboard in order to solve a given

problem. The main difference between the centralized and distributed decision making is the type of data which will be sent to the blackboard. When it comes to choosing which decision-making method will be implemented in this project, two main factors were considered, processing power and network traffic.

The centralized decision-making costs more processing resources than the distributed decision making since all the calculations are made in the blackboard. While the distributed decision-making costs fewer processing resources and more traffic will be transmitted over the network. The distributed method is chosen to be implemented after agreeing with the client.

4.3 UML Design

In this phase the process of designing the system architecture took place, starting by creating the class diagram, see figure 4.3.1 of the blackboard package, the sequence diagram of adding a task to the system ending by execution. More information about the classes and functions of this design can be found in the attached design document.

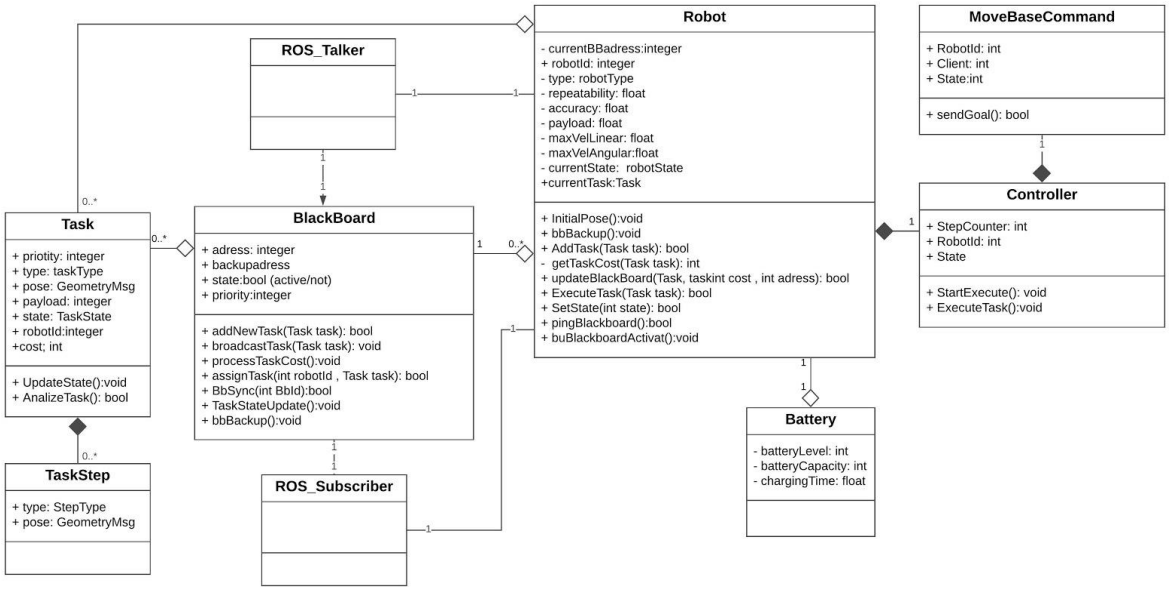


Figure 4.3.1Class diagram

Further a special communication diagram was created to describe the messaging in this system. In this diagram ROS nodes responsible for publishing data, ROS topics used to transfer data, and the ROS nodes subscribed to the topics are explained. See figure 4.3.2

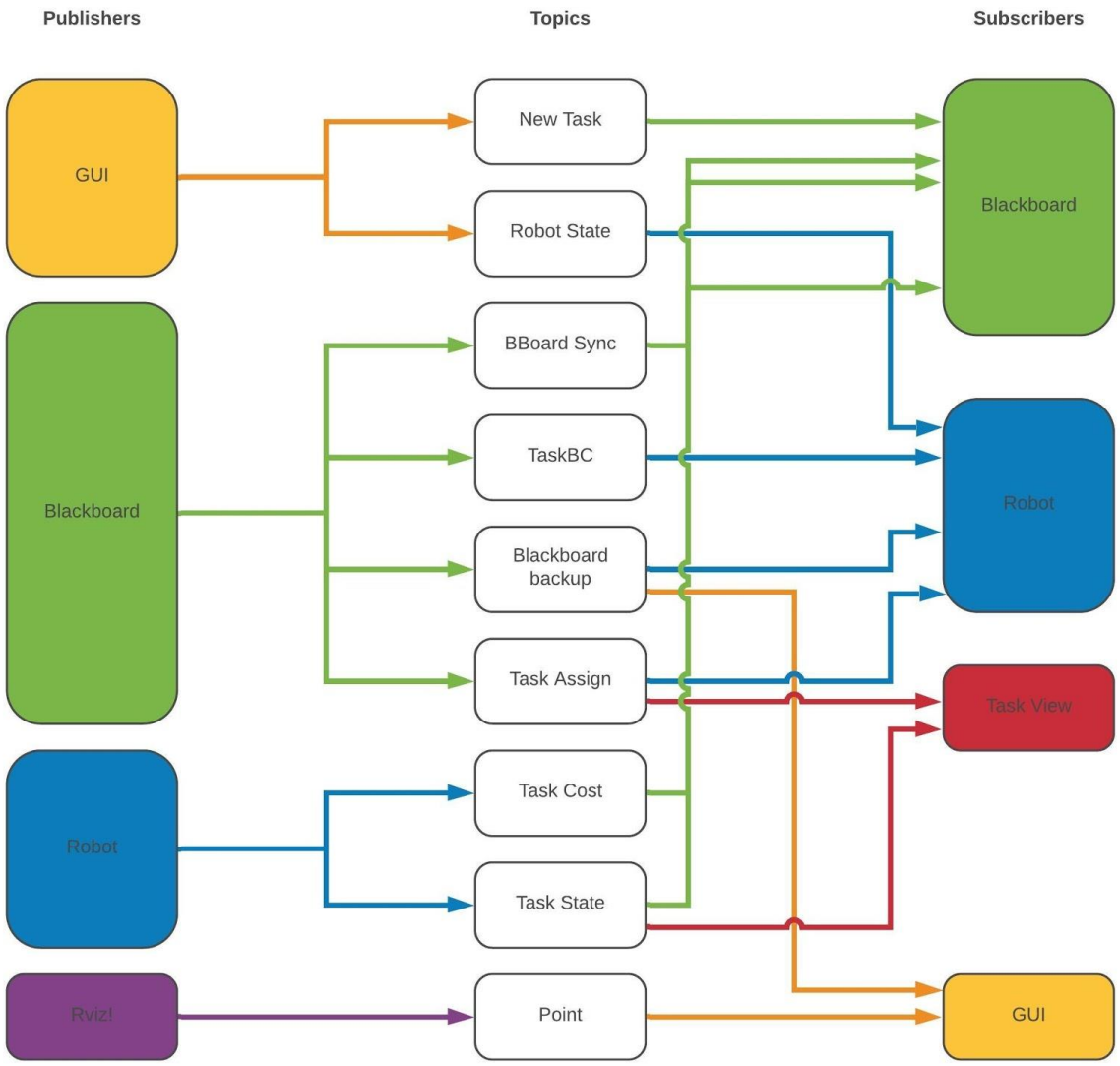


Figure 4.3.2 Communication diagram

4.4 Cost calculation

The cost calculation is considered one of the most important feature of this project since it's the most important factor in decision making when it comes to assigning tasks to robots. The cost of executing a task consists of numerous components, and its split into two main parts:

- The energy cost where a robot calculates how many amps its battery will use to execute a given task.
- The robot state cost: where the robot will produce the final cost value considering the energy cost and many other factors which will be explained further.

Each task has a max cost of 1000 declared when a task is created, before running the energy cost calculations some exceptions were created in the cost calculation function:

- If the task's payload is greater than the robot's payload a value of 1000 is returned.
- If the type of a task cannot be executed by the robot a value of 1000 is returned.

Energy cost:

To simulate the consumption of energy in this system, a software model of a battery was created. The battery model is used to define a physical battery by storing the current battery level, the battery capacity, and charging time. In addition to the mentioned variables three functions where created:

getVolts: simulates a voltage drop function and returns the current voltage of the battery.

getAmps: returns a value which represents how many amps are available in the battery.

updateLevel: used to simulate the drop in battery level by passing the number of amps to be subtracted.

After agreeing with the client, it was decided to consider only the friction coefficient, and to disregard the temperature, the payload aerodynamics coefficient, and the acceleration / deceleration of the robot in this simulation.

The above-mentioned variables and functions are used further to calculate the energy cost of executing a task according to the following formulas:

E = Fr * d

Fr = Cf * m

Fr: rolling friction, Cf: friction coefficient a constant that is defined for each robot, E: energy in watts, m: Robot + payload mass, d: Distance currently being calculated as the shortest path between two points using the formula:

Sqrt [(x2-x1)^2 + (y2-y1)^2]

The energy in watts is divided by the value of the getVolts function to convert to amps.

EnergyCost = e / getVolts

Final Cost:

After calculating the energy cost of executing the given task the robot calculates the estimated battery level at the time of execution, since the task will be ordered in the robot's tasks list based on the task priority.

A constant eTolerance was defined to be 10% of the robot's battery level, if the energy at execution time is less than the eTolerance the robot will return a cost of 1000. Otherwise, the final cost is calculated according to the formula:

Cost = Ce * Tp / Xe

Where Ce is energy cost, Tp is the number of tasks to be executed before the given task, Xe is the estimated battery level at execution time.

4.5 Virtual environment

This section explains the process of replicating the existing physical environment at the client's location. For the purposes of this simulation the client requires a simulation environment which matches the physical environment dimensions regardless of the modelling quality. Since the working environment is considered to be a highly complex model "see fig 4.5.1" it is not recommended to include a high-quality version of the environment in the simulation.

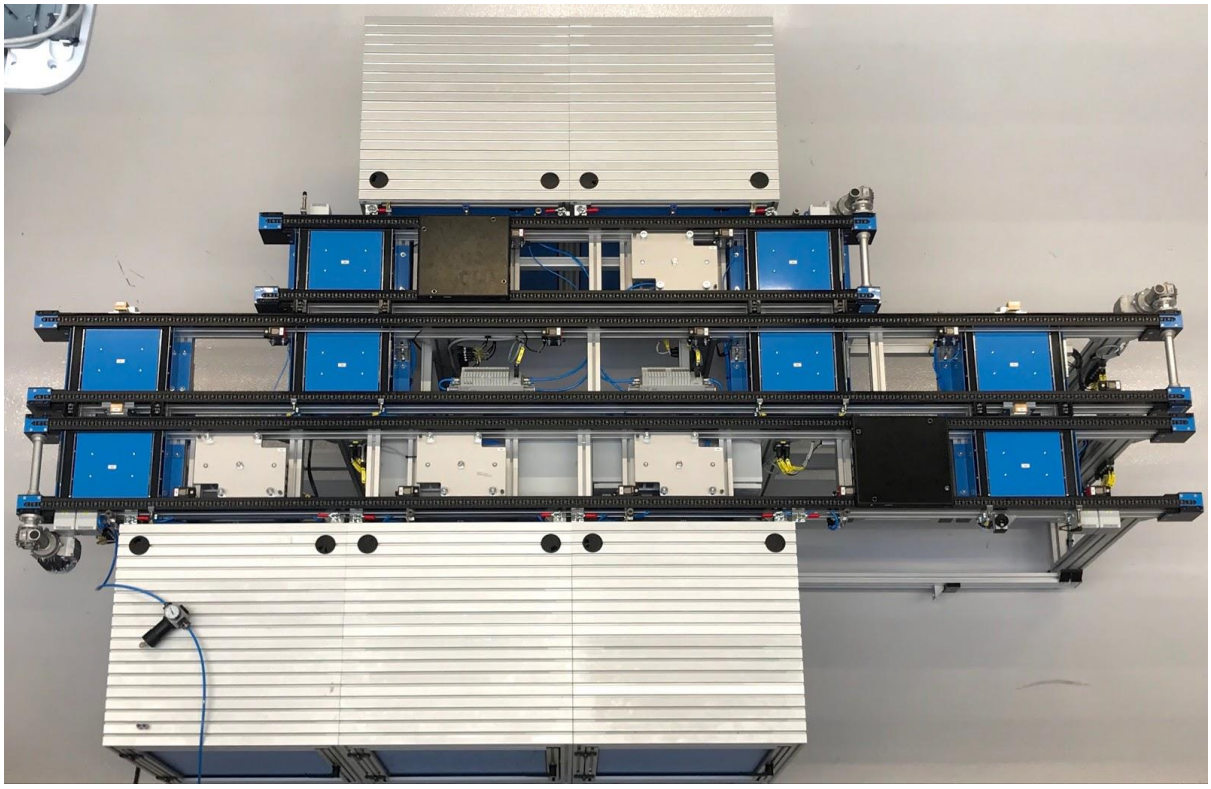


Figure 4.5.1 Physical working environment

The working environment was measured at the client's location and a simple model was created using Blender "3D modelling software" fig 4.5.2 below shows the model in blender.

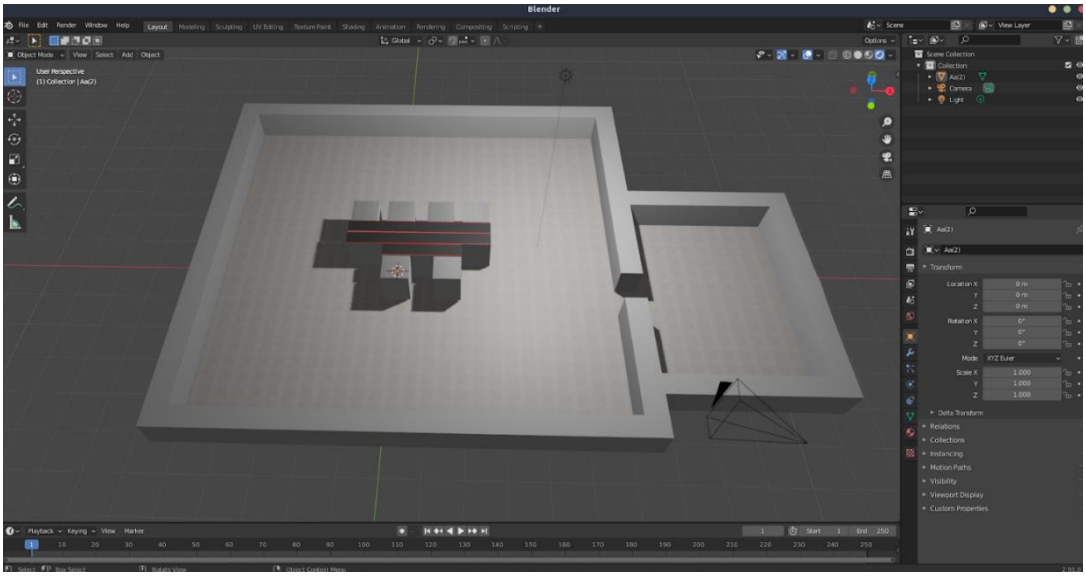


Figure 4.5.2 virtual environment design in blender

The model is then exported as a dae file with the textures used to be imported and processed as a model into Gazebo, the simulation package offered by ROS. The model is set to a static environment in gazebo which means it will not be affected by physical simulation forces such as gravity and wind. Fig 4.5.3 below shows the editor of gazebo.

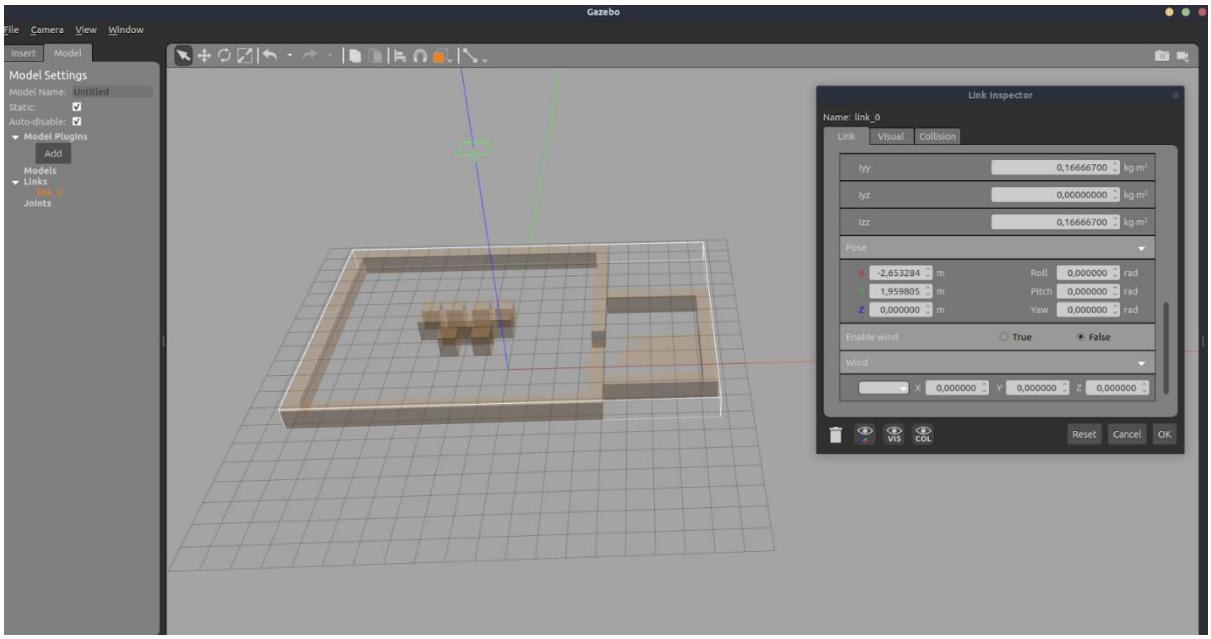


Figure 4.5.3 Gazebo model editor

After editing the model and exporting it to the simulation package the generated model.sda file is edited in order to remove the default materials and shading of gazebo, and use the textures generated by blender. Then it is imported into a new world file with a lighting

setup which is the final simulation environment where the robots will be spawned later. Fig 4.5.4 below shows the simulation world in gazebo.

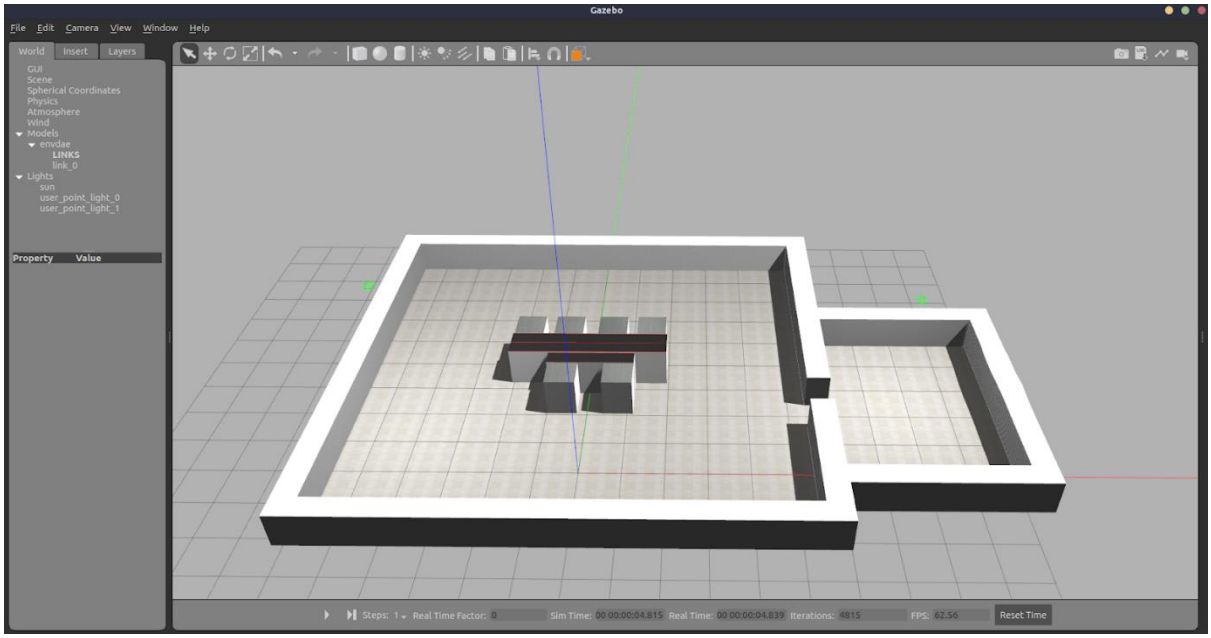


Figure 4.5.4 gazebo simulation envrionment

4.6 Simulation packages

Implementation is split into two main parts: simulation and blackboard logic. simulation is responsible for visualizing the environment and the robots while executing tasks, the blackboard package is the one executing the logic behind the scenes and contacting the simulation to visualize results. This section explains in depth the process of creating the ROS packages that form the simulation part of this project. See fig 4.6.1

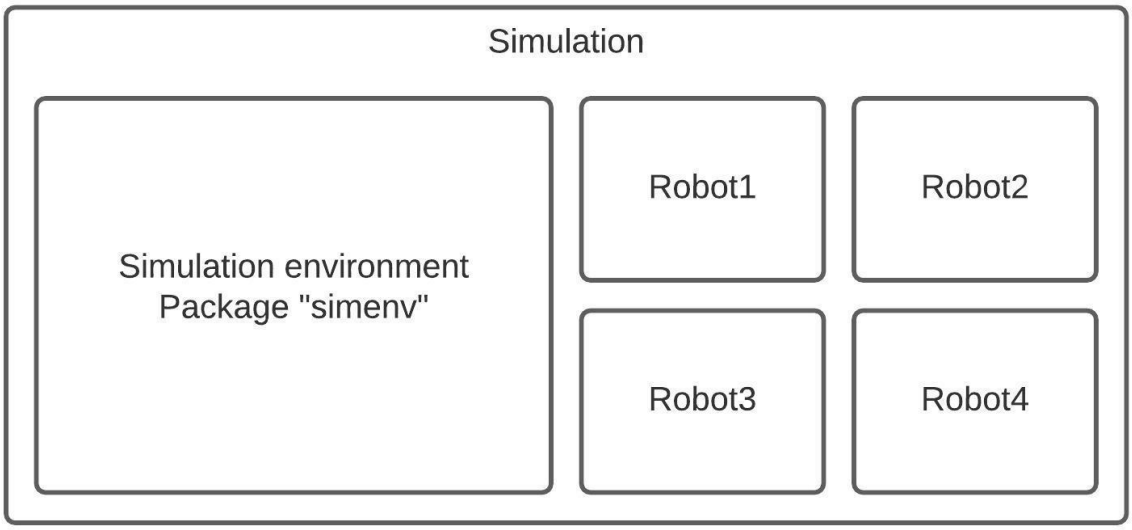


Figure 4.6.1 Simulation package

4.6.1 Simenv package

This package is a collection of 3D, configuration, and ROS launch files used to start the required ROS nodes for the simulation environment, the map server, and the data visualization application Rviz. Fig 4.6.1.1 shows the files included in the simenv package at the first stage.

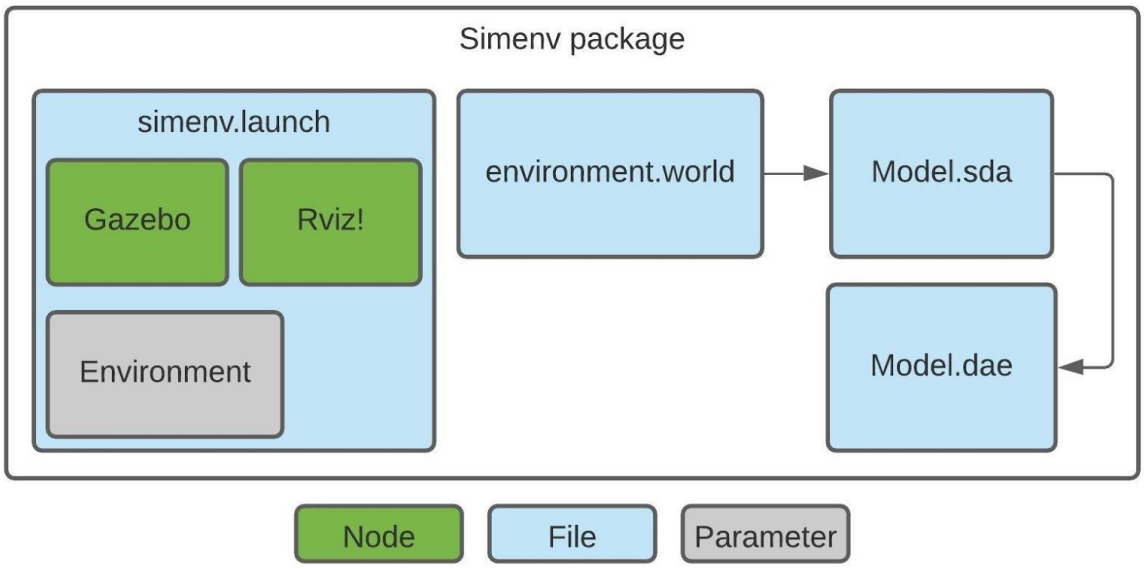


Figure 4.6.1.1simenv package

Simenv.launch:

This file is responsible for starting two nodes: gazebo “the simulation application”, and Rviz “the data visualization application”. Gazebo is started with the environment.world file with references to model.sda, and model.dae created previously in section 4.5 virtual environment.

This package will be extended later with the required files to map the environment and run a map server.

4.6.2 Robot description package

In order to visualize robots executing tasks in the simulation environment is necessary to create a robot description package. The robot is spawned in gazebo the simulation environment and contacted via topics by the blackboard nodes to execute a task. Robot description packages consist of two main files, urdf and launch files. Fig 4.6.2.1 below shows the files of the robot description package.

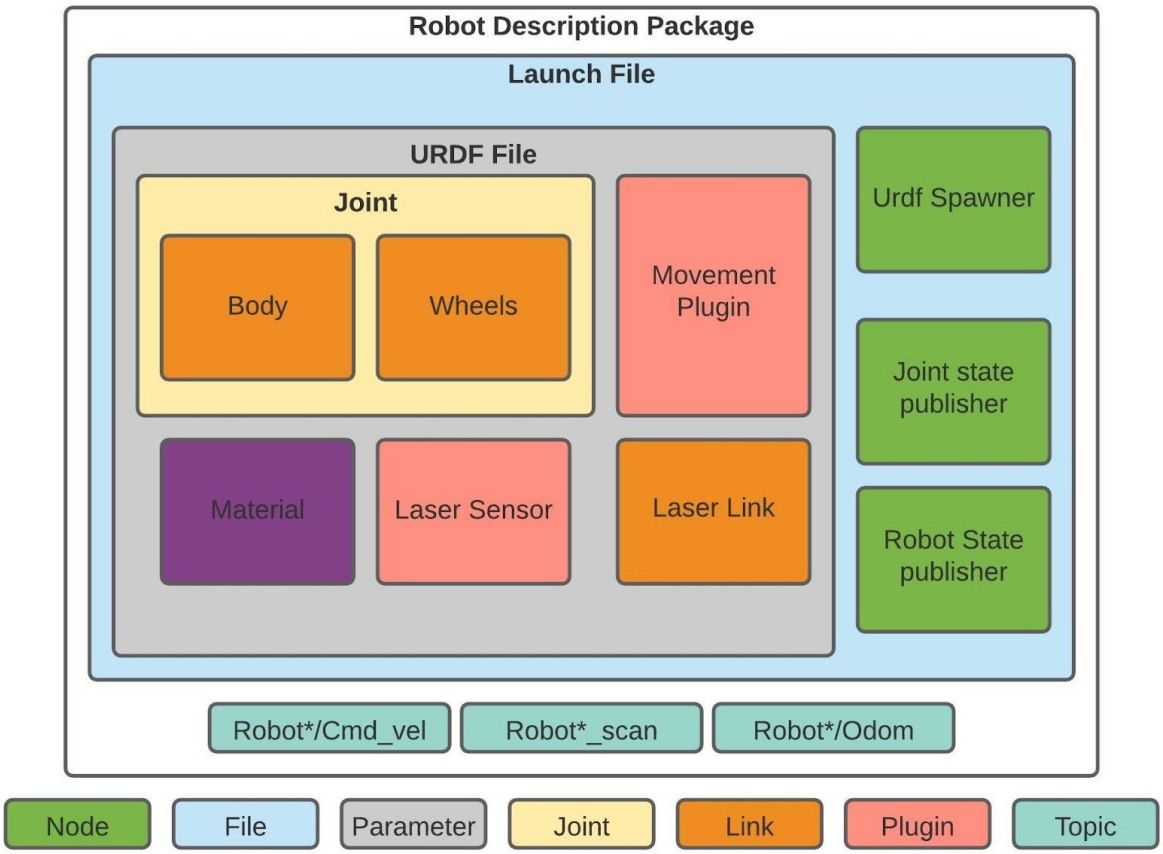


Figure 4.6.2.1 Robot description package

URDF:

The Unified Robotic Description Format (URDF) is an XML file format used in ROS to describe all elements of a robot, such as physical robot parts and joints, simulation plugins, and topic names. For the purposes of this project, it was agreed to use a simplistic design for the agv's, while having the option to plug any ROS robot package into this system, which will be explained later.

Physical components:

The simple robot consists of two physical components or so called in URDF "links", a simple box which forms the base of the robots, and a laser sensor. In addition to a virtual "link" which forms the parent of the robot's body. Below is an example of an URDF link and a joint.

```
<link name="laser">
  <visual>
    <origin xyz="0 0 0" rpy="0 0 0" />
    <geometry>
      <cylinder length="0.06" radius="0.04" />
    </geometry>
  </visual>
</link>

<joint name="laser_joint" type="fixed">
  <origin xyz="0 0 0.06" rpy="0 0 0" />
  <parent link="base_link" />
  <child link="laser" />
</joint>
```

GPU Laser plugin:

A simulated laser range sensor plugin works by broadcasting LaserScan messages, this plugin runs a package called hokuyo_node A ROS node to provide access to SCIP 2.0-compliant Hokuyo laser range finders. See fig 4.6.2.2

Figure 4.6.2.2 hokuyo laser sensor

This plugin is used by adding the following xml tags in the URDF file:

```
<gazebo reference="laser">
  <sensor type="gpu_ray" name="laser_sensor">
    <update_rate>40</update_rate>
      <samples>720</samples>
      <resolution>1</resolution>
      <min_angle>0</min_angle>
      <max_angle>6.28</max_angle>
      <range>
        <min>0.10</min>
        <max>3.0</max>
        <resolution>0.01</resolution>
      </range>

    <plugin name="gazebo_ros_head_hokuyo_controller"
filename="libgazebo_ros_gpu_laser.so">
      <topicName>robot1_scan</topicName>
    </plugin>
  </sensor>
</gazebo>
```

It's important to set a unique topic name tag since the laser scan data for this specific robot will be accessed via the topic name of this plugin.

Planar move plugin:

a model plugin that allows arbitrary objects (for instance cubes, spheres and cylinders) to be moved along a horizontal plane using a geometry_msgs/Twist message. The plugin works by imparting a linear velocity (XY) and an angular velocity (Z) to the object every cycle.

```
1.    <plugin name="object_controller"  filename="libgazebo_ros_planar_move.so">
2.      <commandTopic>/robot1/cmd_vel</commandTopic>
3.      <odometryTopic>/robot1/odom</odometryTopic>
4.      <odometryFrame>robot1_odom</odometryFrame>
5.      <odometryRate>20.0</odometryRate>
6.      <robotBaseFrame>robot1_base_footprint</robotBaseFrame>
7.    </plugin>
8.
9.
10.
11.
```

This plugin allows the simulation of a holonomic movement of the robot's base, by publishing messages on the command topic which has a unique name per robot. The Odom topic is used to read the position and the current velocity of the robot.

Finally, for the robot to look unique in the simulation a material tag is added to the urdf file.

```
1.  <gazebo reference="base_link">
2.    <material>Gazebo/Green</material>
3.  </gazebo>
```

Robot.Launch:

This file contains the parameters and the required nodes to parse the urdf file and spawn the robot in the simulation environment:

Robot description parameter points to the urdf file.

Urdf spawner node used to spawn the urdf file as a robot in the simulation environment.

Robot state publisher node: used by gazebo to process the joints positions and visualize them in 3D environment.

Launching the simenv.launch first and then robot.launch at the current state will start the simulation environment and spawn the robot. Fig 4.6.2.3 below shows the robot with laser data in the simulation environment.

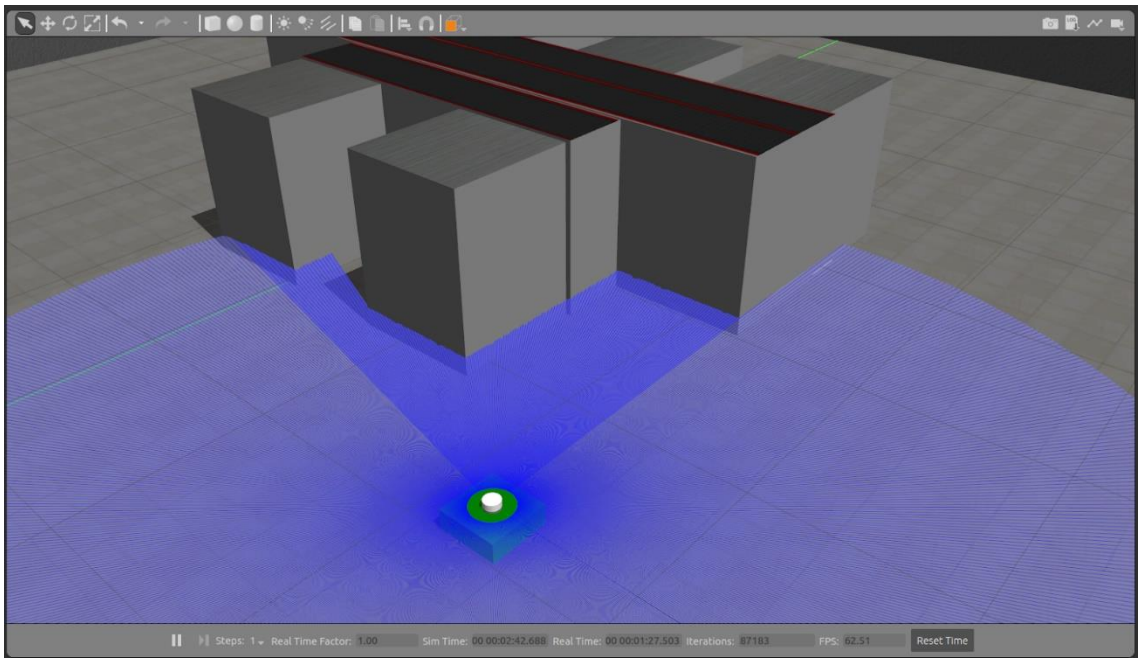


Figure 4.6.2.3 Robot laser data in gazebo

4.6.3 Mapping

Mapping is the process of taking raw sensor data from the LIDARs, and using it to determine where obstacles are located around the robot. All these obstacles can be collated into one map, which can then be saved and later used for navigation. This allows the robot to have knowledge of parts of the world that are beyond it's sensor range, and plan accordingly.

In order to start the mapping process the simenv package is extended with a mapping.launch file which contains the arguments which are the key of using the robot laser data for the mapping process.

```
1.  <!-- Arguments -->
2.  <arg name="set_base_frame" default="robot1_base_footprint"/>
3.  <arg name="set_odom_frame" default="robot1_odom"/>
4.  <arg name="set_map_frame"  default="map"/>
5.  <arg name="set_scan_topic" default="robot1_scan"
```

Fig 4.6.3.1 shows the ROS graph after launching the simulation environment and robot description package. The graph shows the publishers, subscribers, and the topics running in the environment.

Using the sensor data published on robot1_scan topic, and moving the robot around the environment by publishing data on robot1/cmd_vel topic allows the creation of the map which is stored later in map.yaml file.

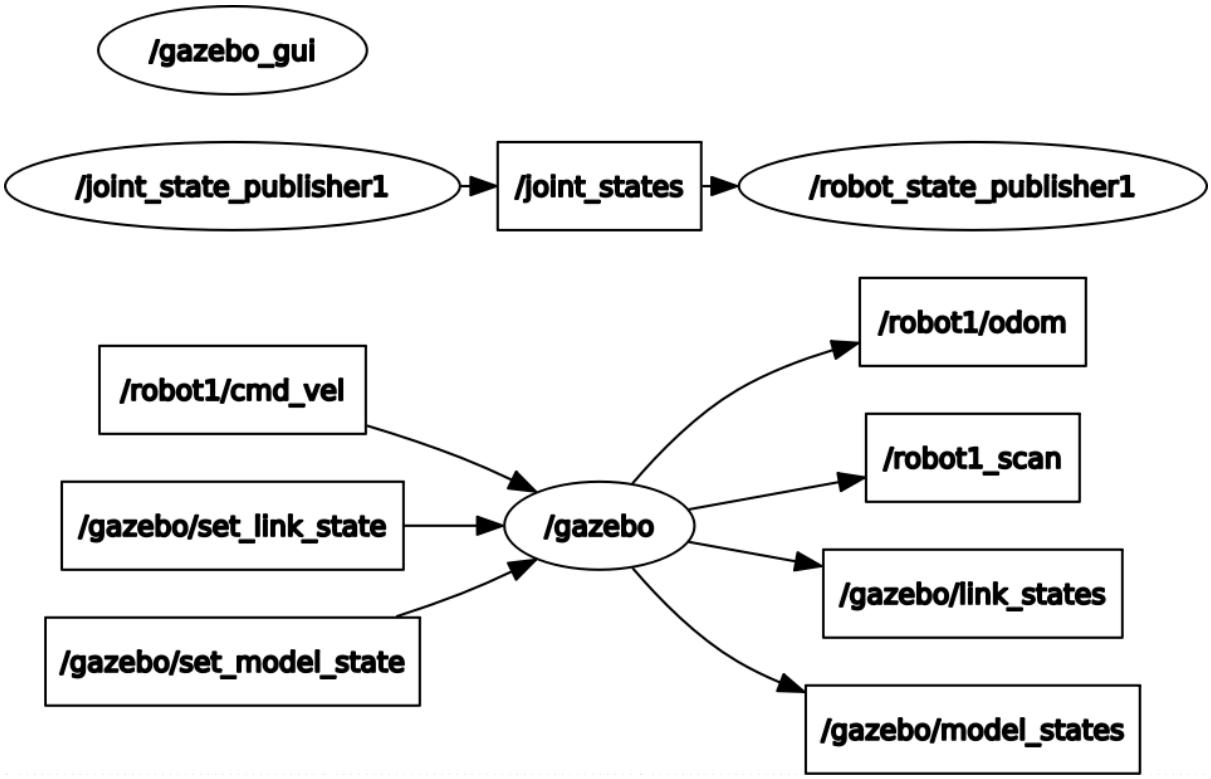


Figure 4.6.3.1Rqt diagram

4.6.4 Finalizing simenv package

Since the map is saved in a file, it is possible to start a map server node which will publish map data of the environment on a map topic. That will allow robots to use the generated map as a static obstacle map. The map server node is included in the simenv.launch file with a reference to the created map.yaml file. See the final simenv package in fig 4.6.4.1

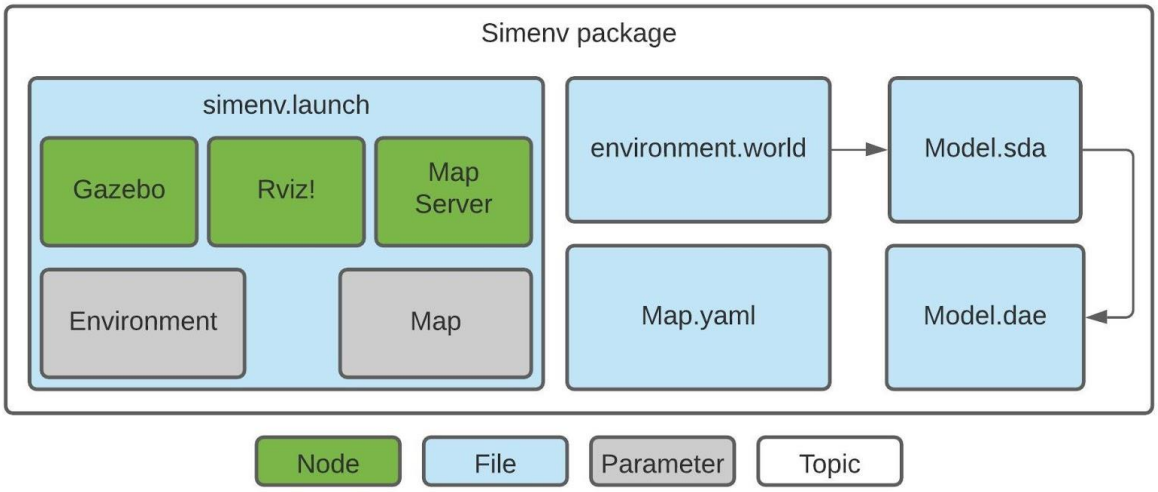


Figure 4.6.4.1 simenv package

4.6.5 Navigation

The final step in the simulation part of this implementation is to set up the navigation using ROS navigation stack see fig 4.6.5.1 The Navigation Stack is simple on a conceptual level. It takes in information from odometry and sensor streams to determine the current robot position and outputs velocity commands to send to the robot's base. The navigation stack consists of two planners, a global and a local one. Both have their own cost map. A cost map is a method of planning paths around obstacles. Areas close to obstacles, and paths through these areas, are assigned a higher 'cost' than those free from obstacles. This section explains the process of extending the robot description package by creating the files required by the navigation stack.

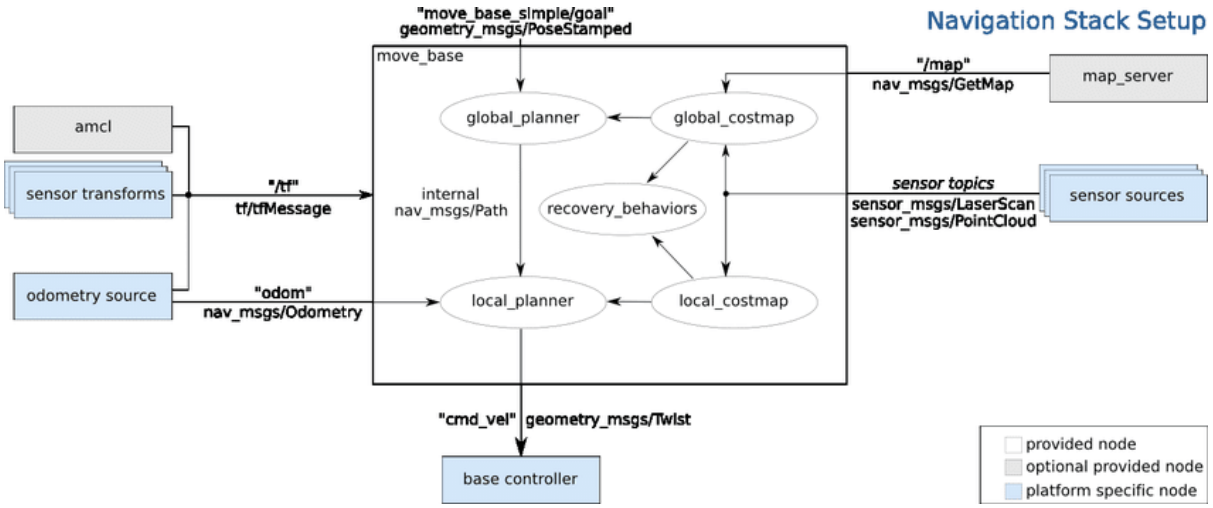


Figure 4.6.5.1ROS navigation stack

Configuration files:

When using the navigation stack, move base has to be started with some parameters that will describe the behaviour of both global and local planners. These parameters are usually stored in. yaml configuration files and called later through launch files.

Costmap_common_params: stores information about the robot's size, obstacles range, inflation and static layers.

Local_costmap_params and **global_costmap_params**: stores information about the robot's base frame map resolution and update frequency.

Teb_local_planar_params: stores the local planner configurations and behaviours.

Move base launch file:

This file includes robot specific arguments such as odometry frame, laser topic, and base frame which allows movebase to communicate with a specific robot. It also includes parameters that refer to the configuration files explained earlier.

The file starts a movebase node which subscribes to many topics that will be used later to send goals and commands to a robot. Inorder to make move base robot specific it is required to remap the topics from theri default namespaces to a robot namespace.

```
1.  <remap from="/move_base_simple/goal" to="/robot1/move_base_simple/goal" />
```

Amcl launch file:

Amcl is a probabilistic localization system for a robot moving in 2D. It implements the adaptive Mont Carlo localization approach, which uses a particle filter to track the pose of a robot against a known map. As currently implemented, this node works only with laser scans and laser maps. This launch file contains robot specific parameters and robot's initial pose. It offers robot initial pose and an Amcl pose topics which are used by the navigation stack. The topics are remapped from the default namespace to a robot specific namespace.

4.6.6 Finalizing the robot description package

After the robot description package is extended with required files for the navigation stack to be used. The package is duplicated into four instances with unique names. All topics, frames, and node names have a prefix "/robot(roboid)/" to make them unique. It was agreed with the client to run 4 robots in the simulation. See Fig 4.6.6.1 for the final robot description package, fig 4.6.6.2 for the ROS graph of running simenv and robots 1 packages. Fig 4.6.6.3 Rviz, and 4.6.6.4 gazebo.

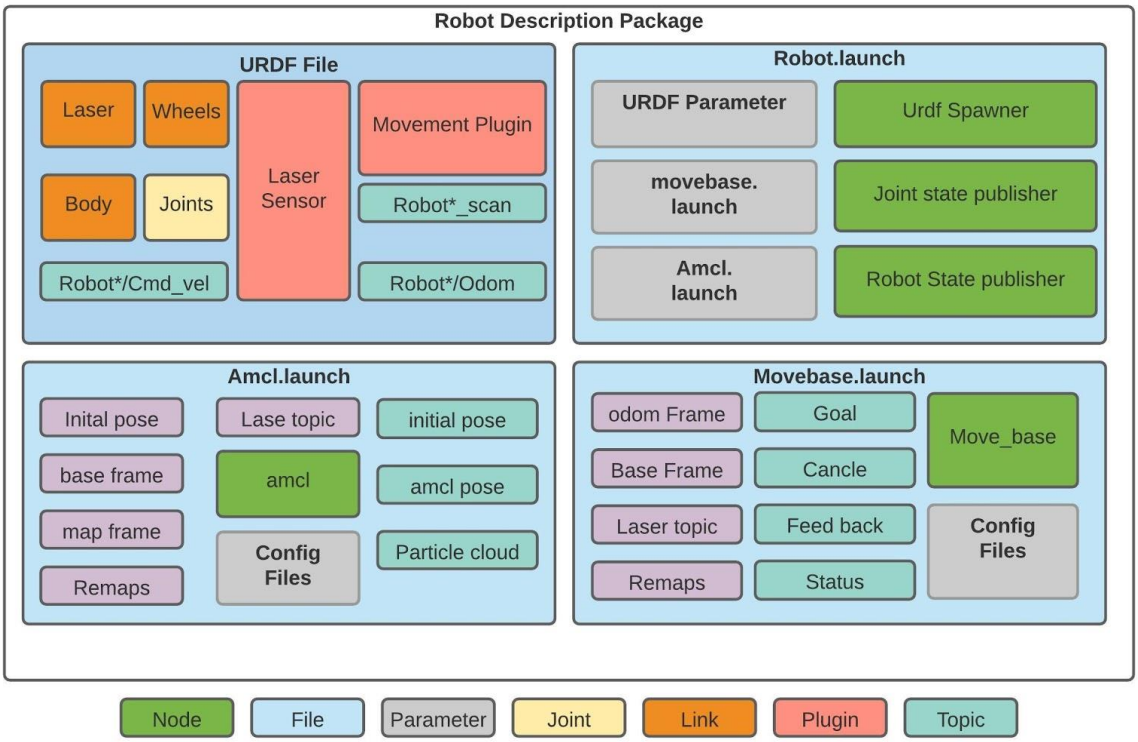


Figure 4.6.6.1 Final Robot description package

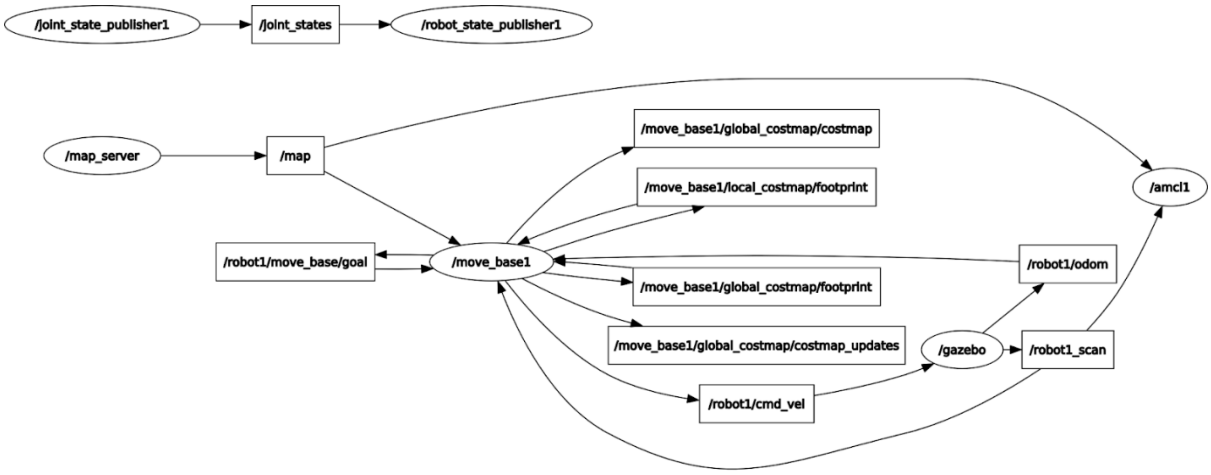


Figure 4.6.6.2 rqt diagram

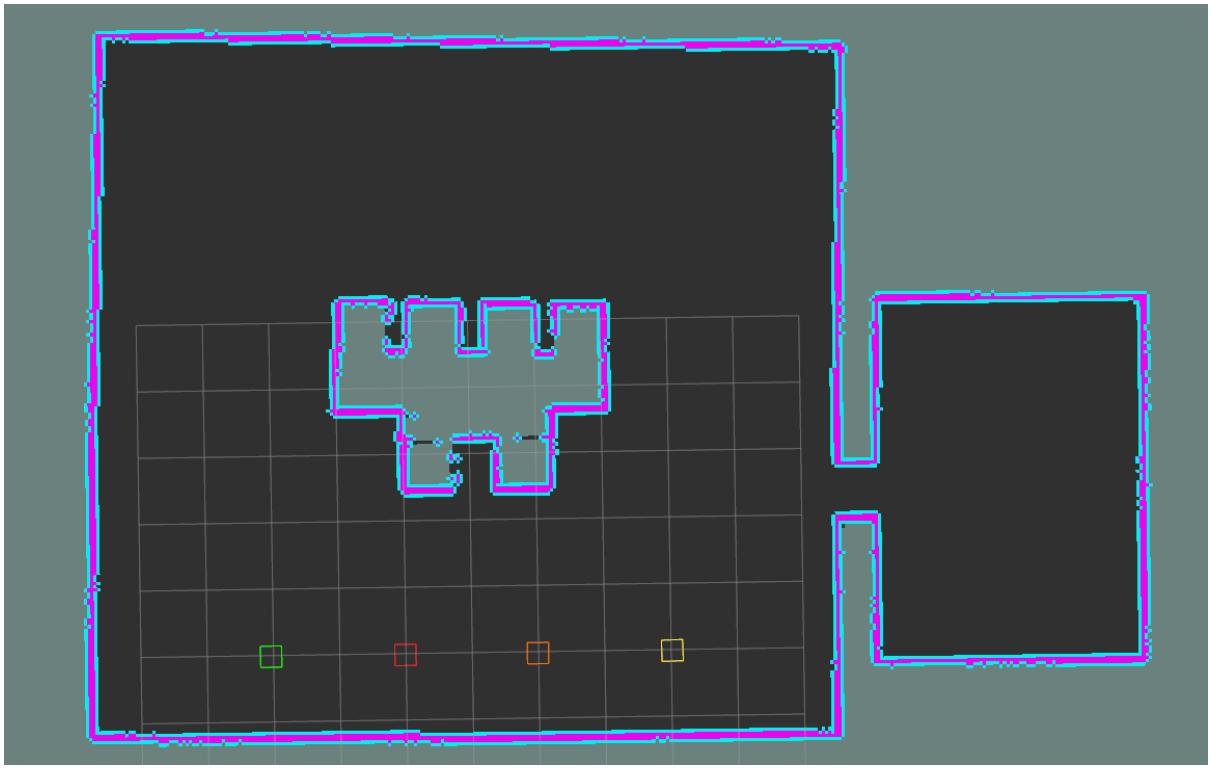


Figure 4.6.6.3 Rviz

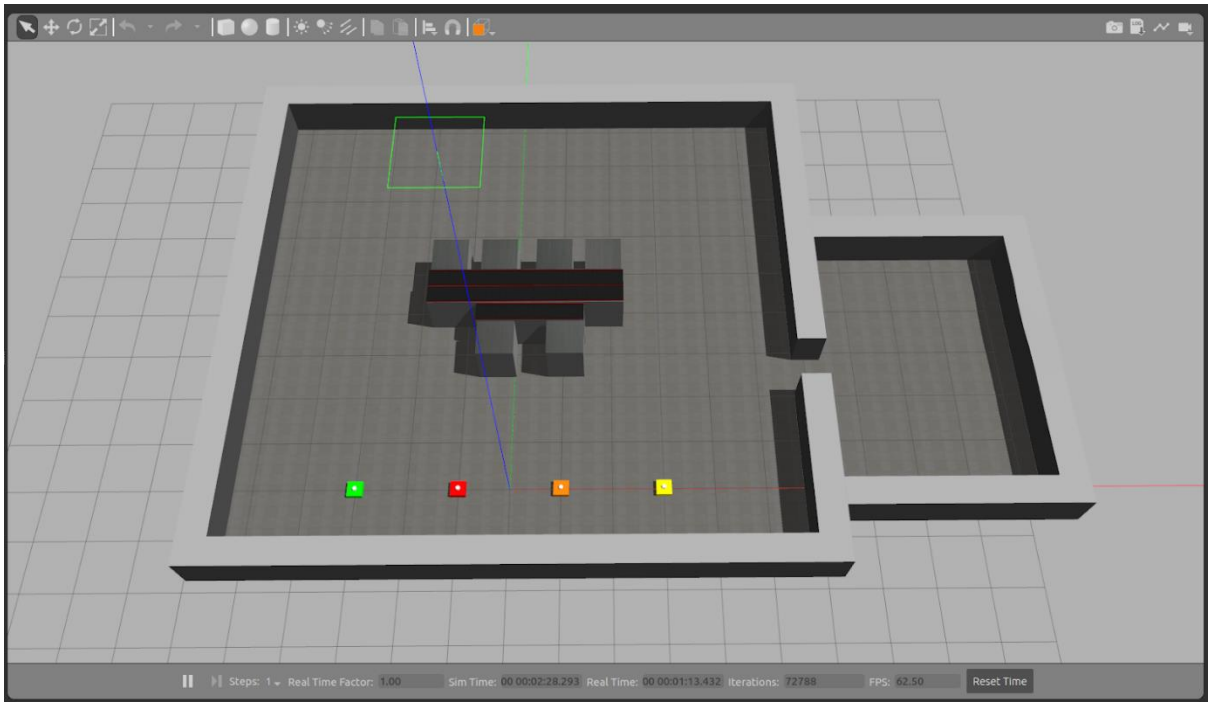


Figure 4.6.6.4 gazebo simulation

4.7 Blackboard package

This section explains in depth the process of creating the Blackboard package. This package is the implementation of the UML design in section 4.3. The package contains Python modules and ROS messages definition files. Fig 4.7.1 describes the files included in the Blackboard package.

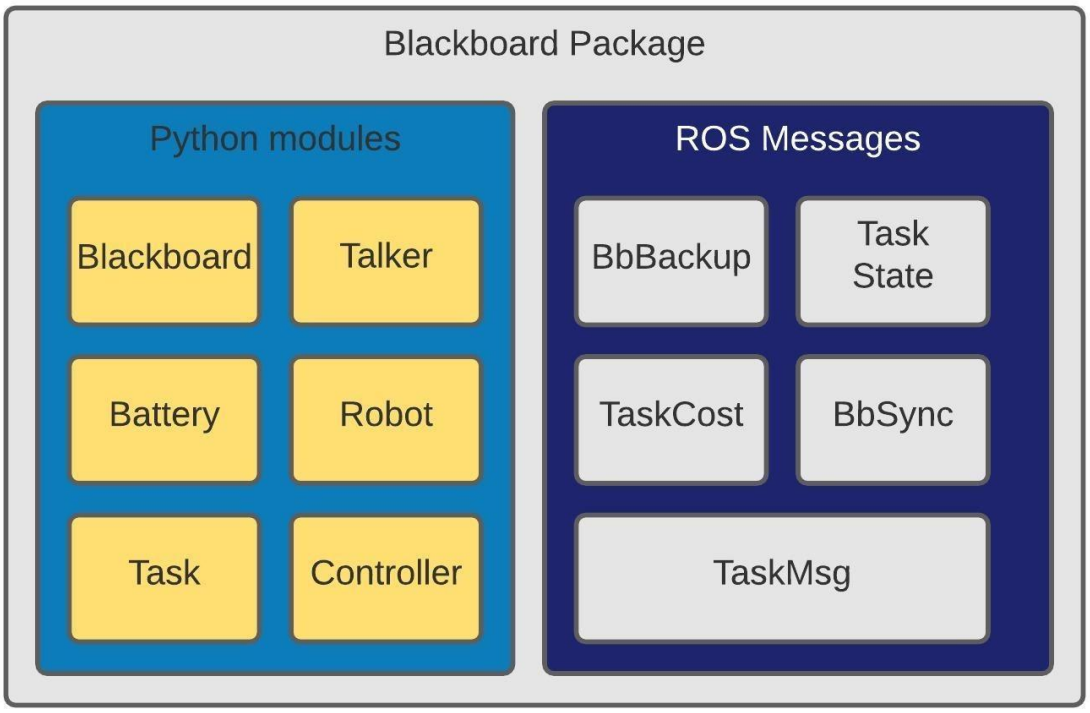


Figure 4.7.1 blackboard package

More detailed information about the Python modules, classes, and functions can be found in the attached “design document”.

4.7.1 Programming languages

ROS programs known as “Nodes” can be created mainly with two programming languages. C++ and Python. There are other languages that can be used but those are not fully supported. Since the nodes communicate using standard ROS messages, the actual language implementation of the nodes is not affecting their communication which is a big advantage of ROS.

C++ vs. python in ROS:

In order to make a decision on which programming language to use for the Blackboard package implementation both languages were researched. The pros and cons of each language were listed and compared for making some advice to the client.

Pros of programming ROS in Python

- Is faster to build prototypes. A working demo can be created very fast with Python, because the language takes care of a lot of things by itself, so the programmer doesn't have to bother.
- No compilation or hidden in Python, the nature of bugs in python makes them easier and faster to catch.
- Python can be learned relatively fast.
- The final code of a python node is quite easy to read and understand.
- Python is a very powerful language with available libraries for almost anything.

Cons of programming ROS in Python

- Python is an interpreted language, which means that it is compiled in run time, while the program is being executed. That makes the code slower.
- Higher chances of crashing in run time, that is, while the program is running on the robot.
- Possibility of ending with "messy" code in a short time if procedures are not defined clearly.

Pros of programming ROS in C++:

- The code runs fast.
- By having to compile, errors are found during compilation time, instead of having them in run time.
- C++ has a huge number of libraries.
- It is the language used in the robotics industry.

Cons of programming ROS in C++:

- C++ is more complex to learn and master.
- A working demo requires a lot of code.
- Reading and understanding C++ code requires a lot of time.
- Debugging errors in C++ is complex and takes time.
- Since the goal of this project is to research, design, develop, and test a "*prototype"* of the blackboard-base distributed fleet manager, it was clear that using Python serves the purposes of this project. The final decision to use Python was made after agreeing with the client.

4.7.2 Blackboard package setup

Compilation of ROS programs is done using Catkin, which requires a special type of setup for the created Python modules to be used by other packages. This is done by creating a standard catkin package then adding a "setup.py" and "__init__.py" files to the package. When the package is compiled using Catkin, it's added to the ROS environment and can be referenced and instantiated in other ROS packages.

4.7.3 ROS messages

ROS uses a simplified messages description language for describing the data values that ROS nodes publish. This description makes it easy for ROS tools to automatically generate source code for the message type in several target languages. Message descriptions are stored in .msg files in the msg/ subdirectory of a ROS package. There are two parts to a .msg file: fields and constants. Fields are the data that is sent inside of the message. Constants define useful values that can be used to interpret those fields (e.g., Enum-like constants for an integer value).

Message types are referred to using package resource names. For example, the file Blackboard/msg/TaskCost.msg is commonly referred to as Blackboard/Task Cost.

Below is a description of all custom created ROS messages for the purposes of this system:

- BbBackup.msg: used by the active blackboard to broadcast the address of the currently active blackboard and the backup blackboard.

- Bbsynch.msg: used by the blackboard to synchronize the tasks list between the blackboard and the backup blackboard.
- TaskCost.msg: used by a robot to reply with the cost calculation results to the blackboard, includes the robot id, task id, task cost, and energy cost.
- TaskMsg.msg: used by different objects to serialize a task object and publish it as a message over topics.
- TaskState.msg: used by a robot or blackboard object to update a task state.

4.7.3 ROS communication module

The communication module includes the Talker class, which is responsible for initializing the ROS node and ROS publishers. The Talker class is considered the base of all communications of this system, and it is further instantiated in other classes.

4.7.4 Blackboard module

This module contains the blackboard class which is responsible for handling new tasks and assigning a task to a certain robot. New tasks are monitored over “newTask” topic, added to the tasks list and broadcasted over “taskBc” topic for receiving costs in order to assign tasks robots. Comparing the blackboard class to other classes of this system, it is considered relatively complicated. This is since this class does not implement a main loop which includes the logic, whereas it depends on events and timers.

Most of the functions in this class are Call-back functions triggered when data is received through ROS subscribers or triggered by ROS timers (see fig 4.7.4.1). This implementation was chosen since it requires less processing resources. Fortunately, in rospy, every subscriber or timer gets its own thread. So, for a single topic that means the call-back will be executed sequentially (e.g., receiving task cost from the robots) which eliminate the need of using thread locks in this class. A reasonable queue size for publishers was chosen to solve some problems created when messages are received before the previous call-back has finished.

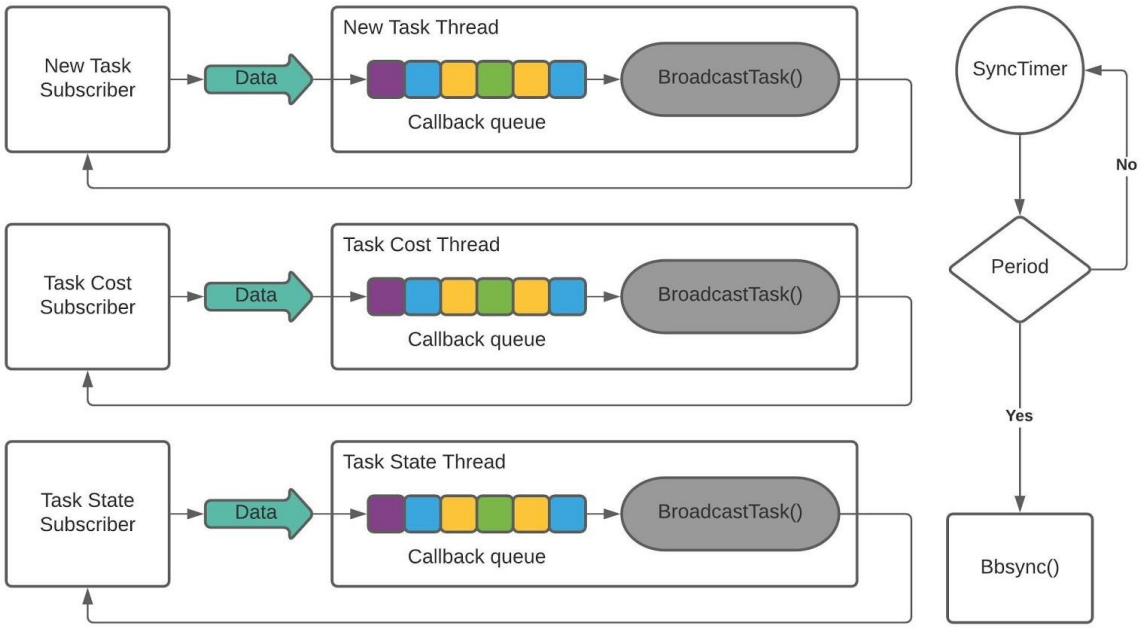


Figure 4.7.4.1 blackboard subscribers and timers

4.7.5 Task module

This module defines the types of tasks that could be executed by robots, and task states. It also includes a Task class and TaskStep Class.

Using TaskType Enum four types of tasks can be added:

GA	GAB	GPA	GPAB
navigation to a single goal.	Navigation with two goals.	picking task at a goal.	Picking and navigation task.

The Enum TaskState defines four states a task could be in:

waiting	assigned	started	done
A new task is created with default state waiting	A task is assigned to a robot	Execution of a task started	Task is executed successfully

Task class defines an object with many fields that describe a task (e.g., id, priority, type, etc). It also includes the analyzeTask function which breaks the task into multiple TaskStep objects to be handled by the controller.

TaskStep is used by the controller to send commands to the ROS navigation stack, or Moveit! depending on the step type.

4.7.6 Controller module

This module is considered the connection between the "logic" part of and the simulation part of this system. It contains two classes MoveBaseCommand and Controller classes.

MoveBaseCommand class:

A simple action client used to contact the ROS navigation stack's move_base action server. Its constructor is called with a unique topic name to be used with different move_base nodes running in the simulation package.

Controller class:

Instantiates the MoveBaseCommand class and is responsible for executing task steps. One of the problems when using the action client is the blocking nature of the client.wait_for_result() function. This problem is solved by creating a new thread for each step to call the function send goal. See fig 4.7.6.1

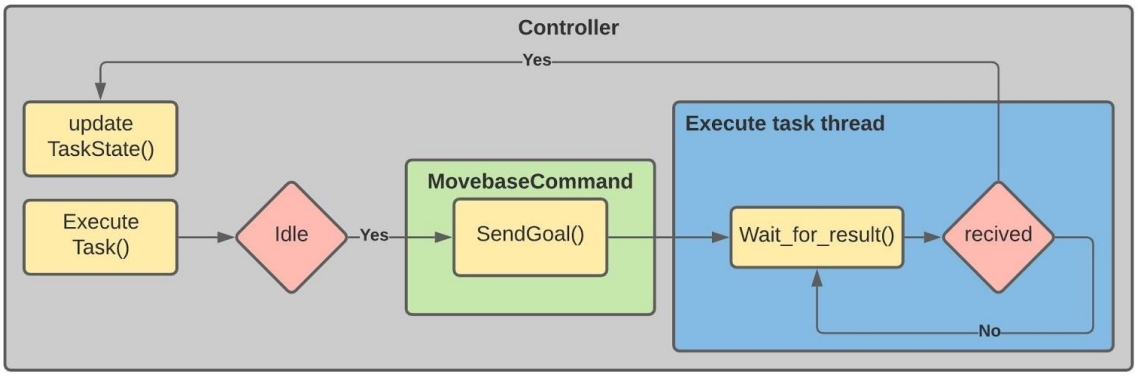


Figure 4.7.6.1 controller class task execution flow

4.7.7 Battery module

This module contains a battery class which is relatively a simple implementation of the UML design.

The class contains three functions:

- getVolts: returns the voltage value according to the voltage drop function.
- Getamps: returns current amps in the battery.
- updateLevel: edit the battery level to simulate energy usage.

For more information about this module see design document in the attachments.

4.7.8 Robot module

This module contains a RobotState Enum with Idle, busy, and defect states. It also contains the definition of the Robot class which is intended to run as an independent ROS node on different robots. The robot class implements the functions to process new tasks, calculate the cost of executing a certain task based on the robot's characteristics, and it is responsible for executing the tasks through a Controller object instance as explained in section 4.7.6.

The Robot class was designed and implemented considering the following requirements:

- Using the researched industrial robots' characteristics, the Robot class should be sufficient to define any robot.
- Expanding the fleet, by running an instance of the Robot class on a physical robot does not require any changes to the Robot class.

The mentioned requirements are satisfied by passing the robot's characteristics and the topic's names used to contact the robot in the class constructor, then the topics are passed to the controller class instance in the Robot class.

Similar to the Blackboard class the Robot class depends on timers and events for executing it's logic. Fig 4.7.8.1 describes the ROS subscribers included in this class.

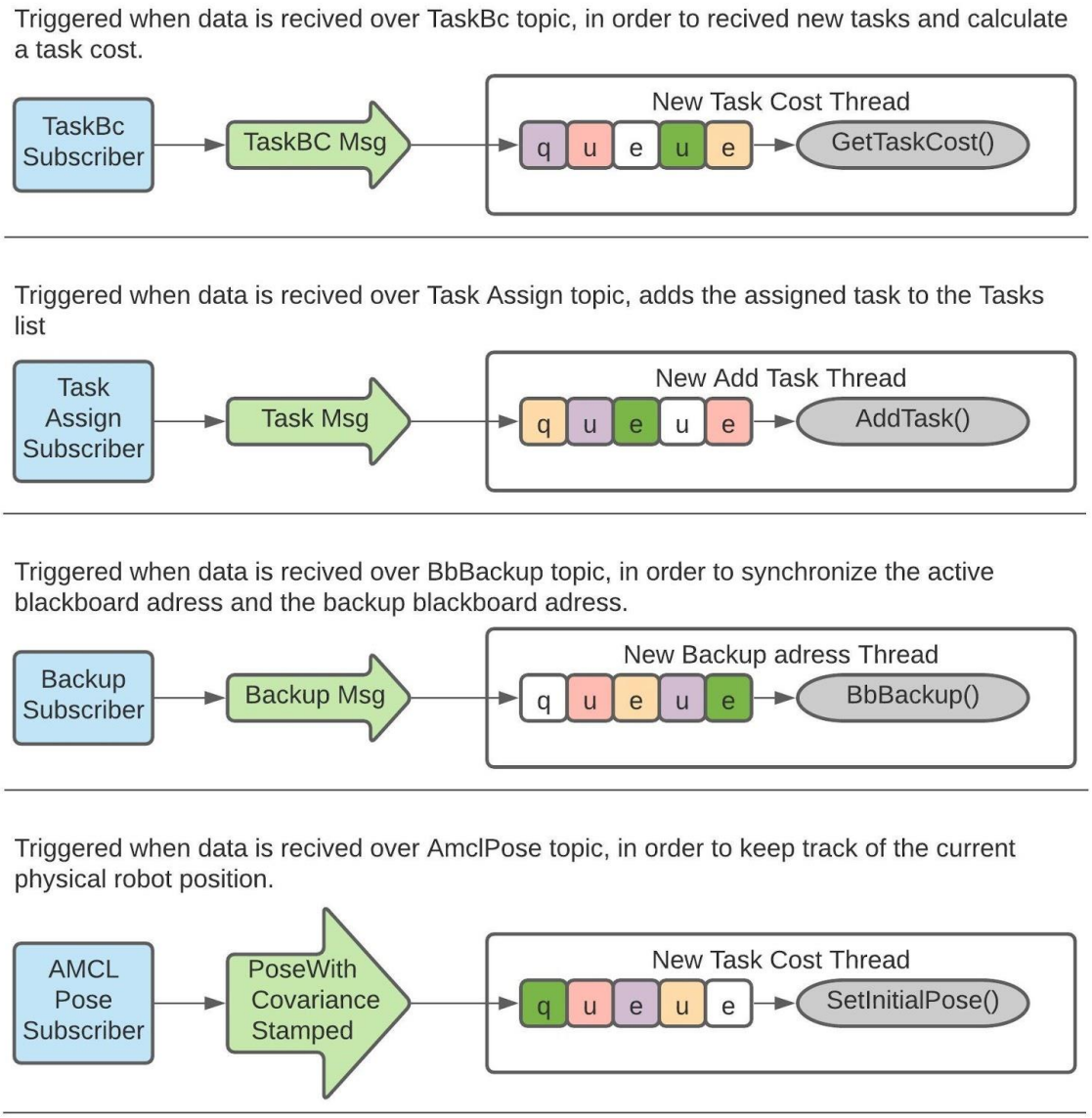


Figure 4.7.8.1robot subscribers and callbacks

The Robot class includes three ROS timers described further in this section:

Ping Timer: triggers a function that pings the blackboard node every predefined period to ensure connection to the blackboard. The connection state is then stored in a local variable “blackboardstate”.

Backup Timer: triggers the function responsible for executing the backup process in case the connection to the blackboard is broken.

An earlier implementation of this class included the backup function within the Ping Timer call-back function. The earlier implementation resulted into a problem believed to be caused by the nature of the "rosnode_ping()" function which returns true value "if node is pinged" as described in the package documentation. In cases of packet loss, the Robot was starting the backup process even if the blackboard is online. The problem was solved by creating the backup timer with a longer period which will start the backup process based on the blackboard_state variable in the robot class.

The backup process is a simple series of conditions that checks the value of the blackboard_state variable and acts accordingly. See fig 4.7.8.2

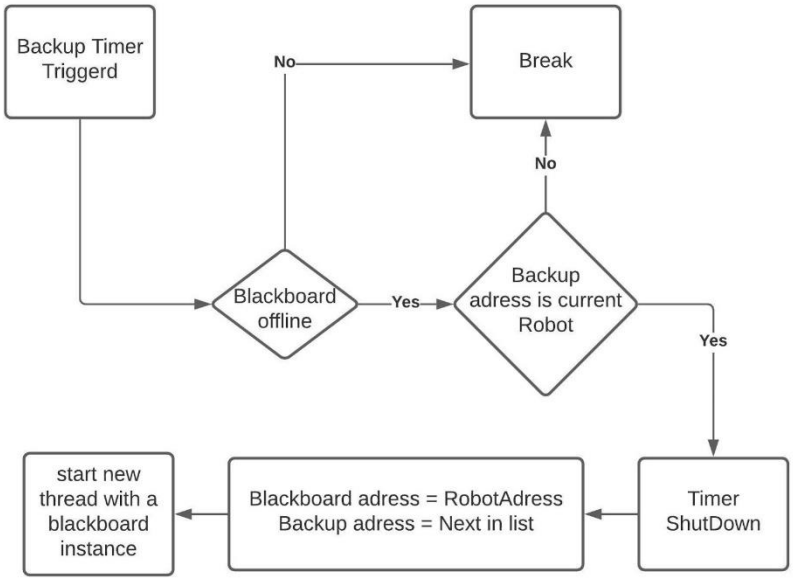


Figure 4.7.8.2 backup process flow

Task Execution Timer:

triggers the function responsible for controlling and executing the tasks in the robot's tasks list. In this timer a threading lock was used in the task execution function since the task execution duration is longer than the timer period. The thread lock makes sure that the function is not triggered again if the lock is previously acquired.

Task execution functionality is described in fig 4.7.8.3

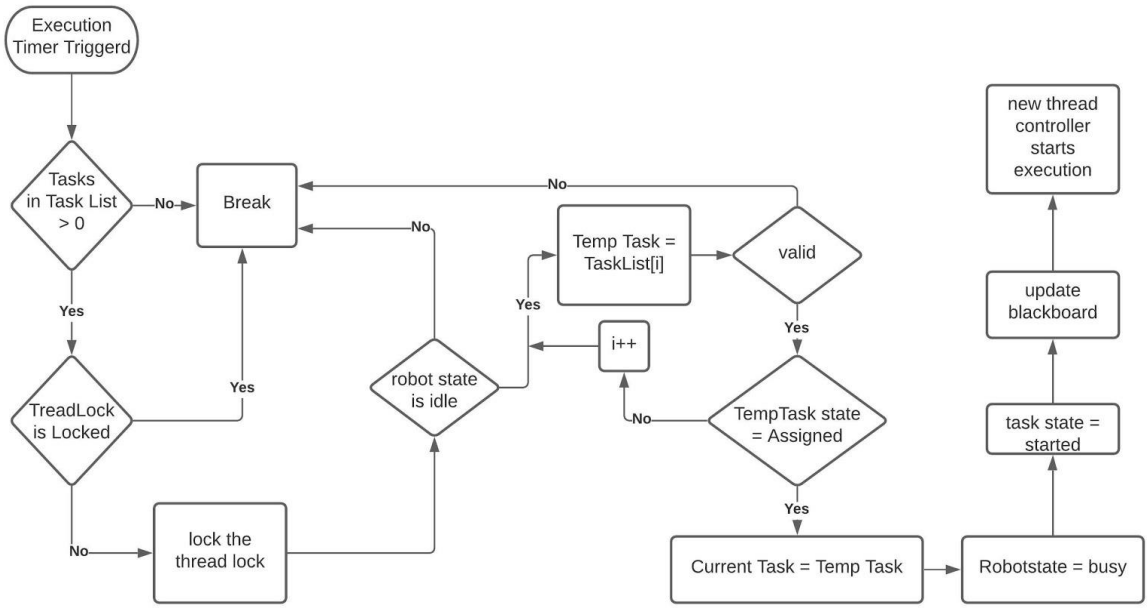


Figure 4.7.8.3 task execution flow

4.8 Fleet manager package

This package is considered the end user application. It includes the user interface for adding new tasks and simulates a change in robots states. The package includes an instance of the blackboard class running as Blackboard ROS node. For the purposes of this simulation since all packages are running in the same environment, instances of Robot class are also started in this package's main launch file. Fig4.8.1 describes the files included in the Fleet manager package.

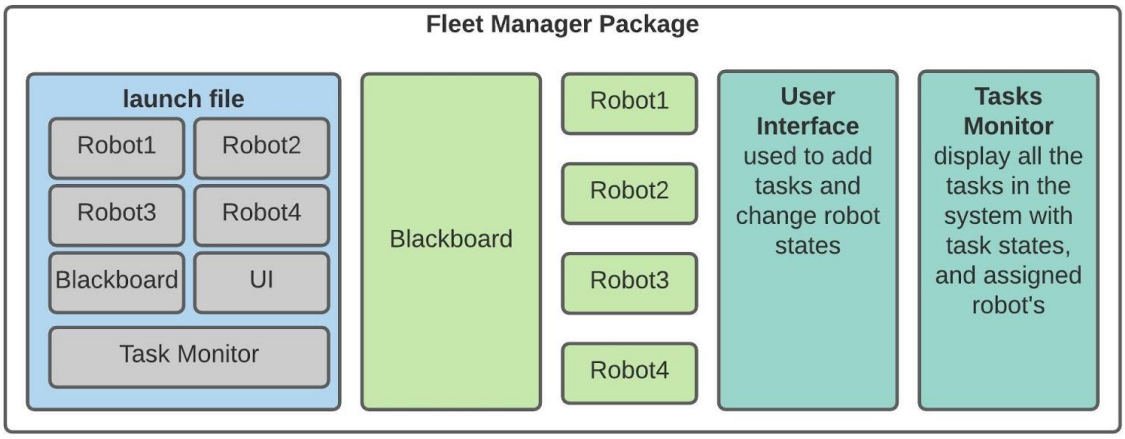


Figure 4.8.1Fleet manager package

Further in this section the process of creating the files of the fleet manager package is explained in depth.

4.8.1 User interface

ROS comes with built-in support for the famous Qt in the form of python_qt_binding library which is used to create the graphical interfaces for both the user interface and the task view. The user interface is designed in Qt 5 designer see fig 4.8.1.1. It is a simplistic design intended to allow the users to add tasks and simulate a change of a robot state graphically for demonstration purposes of how this system works.

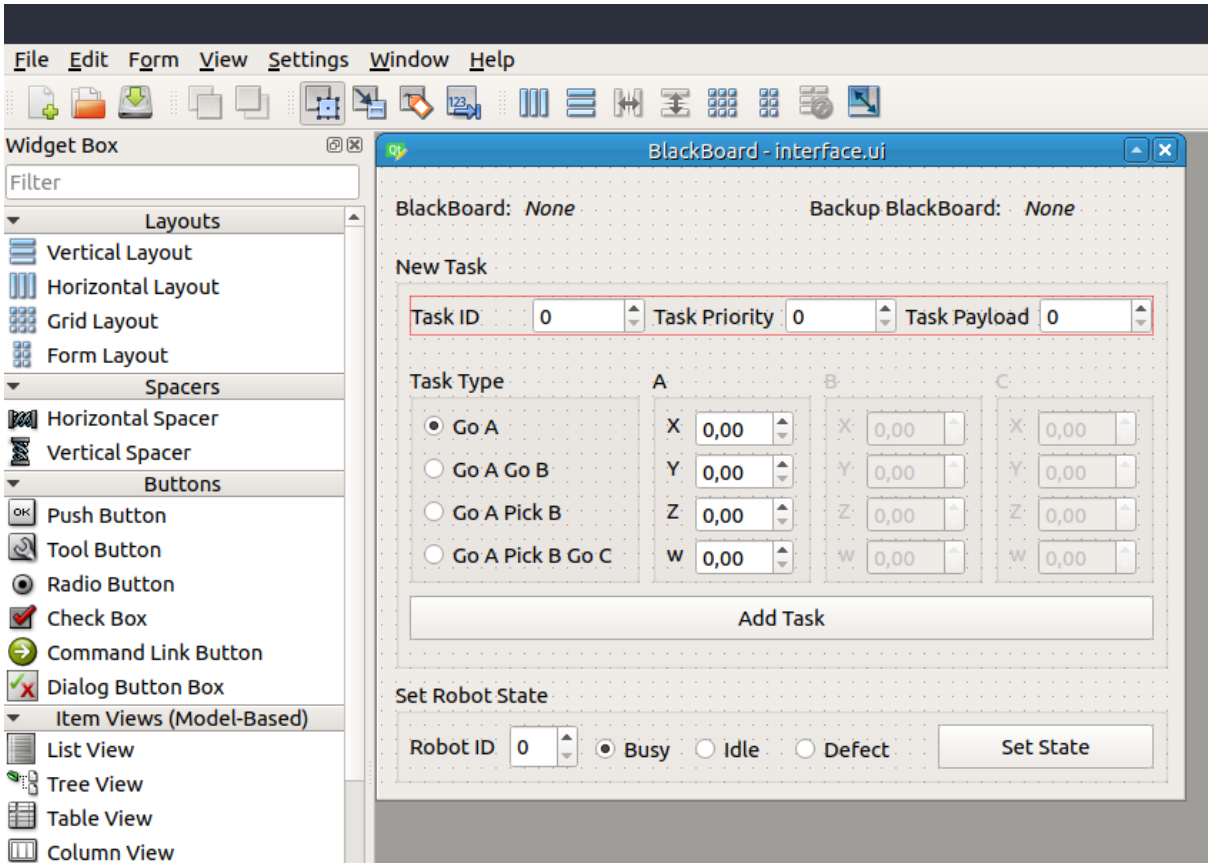


Figure 4.8.1.1user interface design

With this interface users can give task specific values such as Id, priority, and payload. Users are also able to choose the task type and navigation goals.

Using the Qt user interface compiler, the interface is compiled into a python script then a ROS node is created in this interface with subscribers to Rviz! Published point topic. This implementation allows the user to click navigation goals on the map in Rviz resulting in an update of the A, B, and C values on the user interface. See fig 4.8.1.2

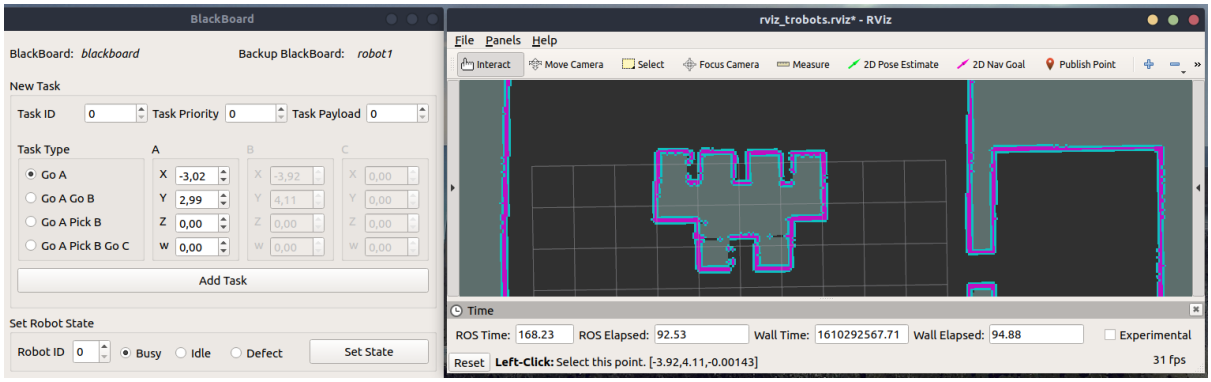


Figure 4.8.1.2 rviz & user interface

4.8.2 Task view

Like the user interface the task view is created using Qt designer then compiled into python script. The task view ROS node is subscribed to the “BbSync” topic which is published by the blackboard to synchronize the tasks list with the backup blackboard. The task view keeps track of tasks in the system and lists the tasks with current states and assigned robots.

Additionally, to tracking tasks the task view includes an implementation of rviz_tools library which is used for Opengl drawing in Rviz. using this library allows the creation of markers in Rviz, those markers are used to represent navigation tasks goals as an arrow in the Rviz environment. See fig 4.8.2.

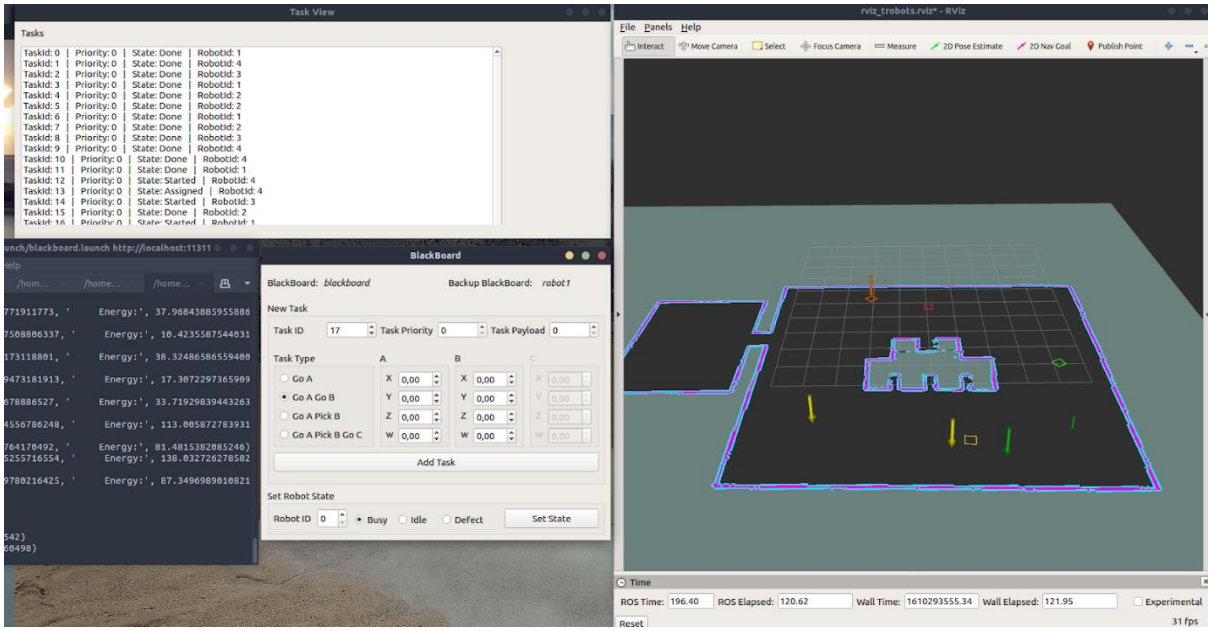

Task View
Tasks
BlackBoard
BlackBoard: blackboard
Backup BlackBoard: robot1
New Task
Task ID
Task Priority
Task Payload
Task Type
Go A
Go A Go B
Go A Pick B
Go A Pick B Go C
Add Task
Set Robot State
Robot ID
Busy
Idle
Defect
Set State
rviz_trobots.rviz* - RViz
File
Panels
Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Time
ROS Time:
ROS Elapsed:
Wall Time:
Wall Elapsed:
Experimental
Reset
31 fps

Figure 4.8.2

4.9 Test phase

In this phase the test cases and the test report were created in order to test the reliability of the system. No benchmarking statistics or excessive testing was required by the client at this stage of the project. below the created test cases are listed without including test results. More information about the test cases, expected results and execution results can be found in the attached test plan.

Test cases:

- starting the system and required packages.
- Aadding a Go A task.
- Executing a Go A task.
- Aadding multiple tasks to the queue.
- Aadding a task with highest priority while other tasks exist in the queue.
- Aadding a task with multiple navigation goals
- Killing the blackboard node, starting the backup process.
- Ffunctionality of previous test cases with backup blackboard

5. Conclusions and Recommendations

The goal as stated in the initial assignment description created by the client was to implement the blackboard architecture for robot tasks scheduling (orchestration) in a virtual environment. The different aspects of this project were:

- The UML diagrams that describe the architecture of the system and the elements in it.
- The virtual environment where the robots will be deployed and scheduled.
- The developed system that can orchestrate different robots in a virtual environment.

After concluding the research originally conducted to investigate the characteristics of a set of industrial robots it became clear that it is possible for a software model to describe and define a physical robot based on certain characteristics which were listed as a result of the research.

The research conducted to answer whether the blackboard architecture can be implemented for different robots and tasks provided sufficient answers on how to implement the architecture with minor adjustments in both centralized and distributed cases.

The physical working environment was measured at location and a 3D version of the environment is created and used as the simulation environment.

The system architecture was designed, developed and tested. It has successfully orchestrated the robots in the simulation environment when adding multiple tasks with different task types and robot specific tasks to the system, even when the blackboard or robots were shutdown to simulate defect, the backup plan showed high reliability and tasks were still successfully executed as expected. Expanding the fleet with any robot is possible if the communication channels with the added robot are provided and described in the software robot model. Expanding the fleet with a non-ROS robot is possible by adjusting the controller class.

After demonstrations and tests the prototype has proved that it is possible to accomplish the goal of robot orchestration, during the remaining period of this project it was requested to investigate further in the possibilities of an alternative architecture and the possibilities of adding a benchmarking GUI to the prototype. The project is considered to be a huge success by the client, and it is believed that this project will be adapted by the lab for further development and expansion.

6. Recommendations:

One of the most important features of this project was the cost calculation algorithm it is highly recommended to further investigate the factors which could determine the cost of executing a task. Using machine learning algorithms is believed to provide more accurate results on both the energy and time costs of tasks execution.

It is recommended to create a custom global planner which takes in consideration the position of all the robots in the working environment and provides more efficient plan.

Investigating the possibilities of expanding the fleet with non-ROS robots could be an important feature of this system.

Evaluation

In the past semester I have learned a lot, in both technical and professional fields. I have developed many skills in project planning and management, for the first time I have worked with SCRUM and improved my organizing, planning and prioritizing tasks. I have conducted a detailed research and had the chance to use the Dot framework in a real-life project.

I have gained more experience in system architecture design and came to realize the importance of early project phases. I feel satisfied with the project results in general and I am more confident after I have received feed back from the company tutor and project manager, since they were “really happy and satisfied” with the prototype.

In this project I had the chance to code in python and create graphical user interfaces, which I have learned by myself, using experience from different programming languages I was able to create a highly reliable complex product which seemed really challenging at the time when I have read the assignment description.

I have expanded my knowledge regarding robotics software development, working with ROS at the latest phases was flawless. Writing or debugging a ROS package or a python script for robotics tasks became a natural thing, while it was complicated at the beginning of the project.

As I have expected when I decided to take this challenge, this project was the right step on my road to becoming a robotics engineer. I have enjoyed working on every detail of this project including the problems and frustrations that I have faced.

Overall, I consider this project a huge success and I am proud to be able to meet my client’s and my expectations.

References/Literature list

John Hsu,a. (2011). Why gazebo?.
Gazebosim.org. http://gazebosim.org/

Open Source Robotics Foundation. (2014). using a URDF in gazebo.
Gazebosim.org.http://gazebosim.org/tutorials/?tut=ros_urdf

File Info. (August 2020). Dae file extension.
Fileinfo.com. https://fileinfo.com/extension/dae.

Tellez,R. (September 2019). ROS developer class.
Multiple robots navigation in gazebo.
https://www.youtube.com/watch?v=es_rQmlgndQ.

Brian P. Gerkey, Jeremy Leibs, Blaise Gassend (November, 2018). Hokuyo_node
Package summary. http://gazebosim.org/tutorials?tut=ros_gzplugins.

Tellez.R (January 2019). Learn ROS with C++ or Python
Programming ROS with python and c++. https://www.theconstructsim.com/learn-ros-python-or-cpp/

Dirk Thomas, Dorian Scholz, Aaron Blasdel. (August 2018). RQT.
Package summary. http://wiki.ros.org/rqt.

Lentin Joseph. (2015) Learning robotics using python 2nd edition.

Packt>. www.packt.com

Attachments

Design document:

https://drive.google.com/file/d/1I_Hc6ygU1MXufvX99uQNJReunNZnh-1O/view?usp=sharing

Test Plan:

https://drive.google.com/file/d/11GuFTdBOuHY7TOoEzSA7T-xnq3tn6Kxy/view?usp=sharing