

# Blackboard fleet manager implementation for ROS2

Marc Meulensteen



---

Fontys University of applied sciences – Mechatronics & Robotics lab

Internship Assignment Report, Eindhoven, August 2021 – January 2022

Marc Meulensteen

Blackboard Distributed Fleet Manager Implementation for ROS2  
Fontys Mechatronics and Robotics Lab – Marc Meulensteen

Internship report – Fontys university of applied sciences

HBO-ICT & Technology: Dutch stream

<b>Data student:</b>	
Family name, initials:	Meulensteen, M.P.C
Student number:	3534537
Project period (from – till):	30/08/2021 - 20/01/2022
<b>Data company:</b>	
Name company/institution:	Kenniscentrum Mechatronica & Robotica
Department:	-
Address:	De Random 1, 5612 PA
<b>Company tutor:</b>	
Family name, initials:	Alers, S.H.M
Position:	Teacher – researcher
<b>University tutor:</b>	
Family name, initials:	Van Gastel, F.J.P.M
<b>Final Report:</b>	
Title:	Reimplementation and testing of a blackboard based multi-robot task execution for ROS 2
Date:	31/08/2021

## Foreword

This document contains the final report of my internship project as ICT & Technology student. I carried this internship out at the Fontys Mechatronica & Robotica lectoraat in Eindhoven. The assignment of this internship project was to convert the blackboard-based fleet manager application from ROS1 to ROS2. Phases of this project include researching the blackboard design pattern and ROS, learning how to develop with ROS and Python, designing improvements on the blackboard system or the general system, and testing the improved prototype in ROS2. This report describes the process and results of the assignment.

I have learned a lot during this internship about technical, but also professional and personal fields. Before starting this internship, I did not know what to expect and was a little stressed about diving into software I have not worked with. Looking back I am very glad that I tried this new experience because it gave me some insight into what I like within my study path and where my interests lie. It also showed me I should not be scared to try new learning experiences.

Thanks to my study coaches Mr. Alers, Sjriek and Mr. Negrete Rubio, Pablo and the help of my internship assessor Van Gastel, Franco I have had a great internship where I have learned a lot about Python and ROS but also in the professional fields and about myself.

## Table on contents

Foreword.....	3
Summary.....	5
Glossary .....	6
1. Introduction.....	7
1.1 Readers guide .....	8
1.2 About the company .....	9
1.3 Assignment overview .....	10
1.4 Planning.....	12
2. Research.....	13
2.1 Blackboard research .....	13
2.2 ROS Research .....	17
3. Conversion process .....	20
3.1 Orientation.....	20
3.2 Conversion from ROS1 to ROS2.....	22
Launching and error resolving .....	22
Re-writing the nodes and testing.....	23
Launching after re-writing nodes.....	24
3.3 Design.....	28
ROS2 Blackboard Class diagram.....	28
ROS2 simulation flow chart.....	29
4. Testing.....	30
5. Conclusions.....	31
6. Recommendations.....	33
Evaluation .....	34
References/Literature list .....	35
Attachments .....	36

## Summary

This document is a report of an internship project taking place at Fontys mechatronics and robotics lab in Eindhoven. According to the website of the lectoraat, they aim to help the research community to overcome several robotics challenges, for that they participate in different founded projects at National or European level.

The Holland Robotics Logistiek (OPZuid) is one of such projects where they would like to accomplish challenges related to having different robots interacting with each other in highly dynamic environments such as warehouses.

The goal of this project is implementing the blackboard based fleet manager application in a more stable and decentralised environment. This means converting the current implementation from ROS1 to ROS2. This project also aims to improve the current fleet manager design and by doing so increase the efficiency of the robots executing their tasks.

In this project to focus will be on answering the following question:

*“How can the current blackboard solution be implemented and improved in a more reliable environment with a more realistic robot behaviour?”*

The current blackboard solution can be implemented and improved in a more reliable environment with a more realistic robot behaviour by converting to ROS2 where there is no centralized system and all nodes are independent. This will give nodes more freedom communicating with each other and every node can be found easily by other nodes. The 3d simulation environment can be improved to make the simulation more realistic and thus easier to implement on physical robots in the future.

## Glossary

These terms are defined by Open Robotics (2018) in the official ROS documentation online.

**ROS:** ROS is an abbreviation of Robot Operating System. It is an open-source, meta-operating system for your robot. It provides 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 is 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.

**ROS service:** Services implement a request-response type of communication. This consists out of 2 message types: One for requesting data and one for the response.

**Gazebo:** A simulation application that offers the ability to simulate robots accurately and efficiently in a complex indoor and outdoor environment.

**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.

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

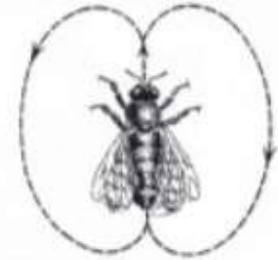
**Rqt:** Rqt is a software framework of ROS that implements the various GUI tools in the form of plugins.

**Colcon:** ROS build tool. Iteration on the build tools: catkin\_make, catkin\_make\_isolated, catkin\_tools and ament\_tools.

**Terminal:** The Linux command line is a text interface to your computer.

## 1. Introduction

I have always been fascinated by how small animals like ants or bees can accomplish huge things when all working together. Bees for instance work with each other to make food and keep the hive alive. There are 3 types of bees in a hive. The queen is responsible for laying the eggs, the male drones for fertilizing them and the female workers for gathering food and caring for the hive. They all have a specific task and communicate with each other to accomplish this. The female workers communicate via dancing for instance. According to a study done by zoologist Karl von Frisch 1973, the female workers inform the other bees with a “round dance” if there is food within 100 meters from the hive (illustrated in [figure 1](#)). “The successful bee waggles dances in the shape of a figure of eight” Social ties hold the hive together.



*Figure 1: Round dance of the female worker bee.*

I’m also fascinated by how we can bring this type of collaboration and communication to the world of technology. Imagine bees as robots and the hive mind as the computer system. One Working together in order to carry out assigned tasks, communicate with each other and make decisions without the need of external intervention. Basically, working as one organism but consisting of a number of autonomic parts. Because it consists out of autonomic parts and not a centralized system, we call it decentralized. This is what inspired The Fontys mechatronics and robotics lab to create the blackboard base decentralized distributed fleet manager. This fleet manager is able to execute tasks of different robotics platforms in a highly dynamic environment, inspired by the concept of collective intelligence. This means the working together of individuals and the communication of knowledge, data or skills for the purpose of problem solving.

Fontys has always been an innovative university contently pushing students to go beyond and create new things. This can be helpful for research or development in certain projects. The location of the technical university is known as the most innovative region of our country and perhaps the whole of Europe. It is possibly the most exciting place to be for anyone with an interest in technology, entrepreneurship, and creativity.

## 1.1 Readers guide

In chapter 1 an introduction will be given into the subject of this research internship, the company and the actual assignment. Chapter 2 consists out of two researches conducted. These are detailed summaries of research documents. In chapter 3 the process of the practical assignment is explained in detail with problems encountered and how they were solved. Chapter 4 consists of test cases of conducted tests with steps on re-creating the test and the result. Chapter 5 draws conclusions and summarizes the process. Finally, chapter 6 contains recommendations for further research and development in this project.



## 1.2 About the company

Fontys is all about inspiring and challenging education and conducting practical research that is truly meaningful to society. The Fontys Mechatronics and Robotics Lab “Lectoraat” plays a big role in this. The Lectoraat works closely with the industry and 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 the lectoraat projects take place in many fields such as robotics, mechatronics, embedded technology and much more.

The Fontys mechatronics and robotics lectoraat are always working on applied research and education renewal. The lectoraat has executed different subsidized projects within binpicking applications, autonomous assemblage and security, mobile robots and is partnered with the BIC Fieldlab Flexible manufacturing. In addition to that the lectoraat is also a part of the ISO workgroup for Safety & Modularity and is part of the euRobotics, the organization that connects the European robotica research.

According to the Fontys site about the lectoraat, different areas of research are:

- Flexible Manufacturing
- Manufacturing Logistics
- Human-Robot Interaction
- Precision Engineering
- Health
- Education renewal

This project will focus on Flexible Manufacturing as the AGV’s will be used in a warehouse to help with assembly. It also touches on Human-Robot interaction as humans and robots will essentially be working together in the warehouse and it is important for this project to take that in consideration.

### 1.3 Assignment overview

This assignment is part of the Holland Robotics Logistiek “OpZuid”. A project taking place at the Fontys Mechatronics and Robotics Lab at the Brainport Industries Campus (BIC). The 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.

#### Project description

The goal of the project is implementing a blackboard based execution of tasks for multiple robotics platforms in a virtual environment. This execution must consider the capabilities of each robot (e.g. type of robot, kinematics, payload, speed, battery capacity, ect.) for scheduling robots to certain tasks. Using an algorithm each robot calculates the cost it would take the robot to execute a certain task. The result of this calculation is sent to the blackboard and the robot with the smallest cost will execute the task. This robot has the coordinates of the task, maps out the path it is going to take and navigates towards the objective.

A prototype of the blackboard fleet manager was made by a previous intern and working in a ROS1 environment. In this prototype it is possible to publish a task using the GUI. The blackboard then finds the robot physically closest (so not cost calculated) to the task coordinates and gives this robot the task. Currently this prototype is running in a simulated environment using the 3D model of the BIC location in Gazebo.

#### Assignment description

The assignment is converting this ROS1 prototype to ROS2. The project owner wants this because ROS Noetic is the last ROS1 distribution. ROS Noetic’s EOL (end of life) is scheduled for 2025. After that there will be no more ROS1.

Converting to ROS2 brings more opportunities for this project such as a decentralized blackboard execution. Converting is not the only assignment for this project. Improvements are also going to be researched for the current fleet manager design and try to implement those into the project.

We try to answer the following question: *“How can the current blackboard solution be implemented and improved in a more reliable environment with a more realistic robot behaviour?”*

A more reliable environment would be ROS2 for the distributed fleet manager, because ROS2 has no centralized system. Each node has the capacity to discover other nodes. This allows for a fully decentralized distributed system which is exactly what this project needs.

The following sub-questions will help answer our main question :

*“How can the current blackboard architecture be improved and/or extended?”*

*“How can the current blackboard functionality be extended towards realistic robot behaviour?”*

### **Assignment constraints**

The prototype will be implemented and tested in a virtual environment. The project is highly dependent on the early phase results, such as research, learning the coding languages and the operating system, analysing and brainstorming. There is a lot of research and learning about ROS and Python before developing can start.

## 1.4 Planning

This project is managed using the Scrum method. Scrum is known to be ideal for self-organizing teams, adapting to changes, continued improvement, and sustainability. This project consists of multiple phases. Below is a short description of each phase.

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

*Research/learning:*

This phase focuses on 4 things:

- Research
  - ROS
  - Blackboard
- Learning basics
  - ROS1 and ROS2
  - Python

I have chosen to see this as one phase as they go hand in hand. During this phase I will make 2 research documents about ROS and the Blackboard design pattern. I will also learn the ROS1, ROS2 and python basics by understanding the current prototype and doing tutorials. This material will be supplied to me by my internship supervisors.

*Design:* Designing the UML diagram using the research results from the previous phase.

*Implementing the navigation:* When a task is created the available robots will receive the task details. The robot best fit to the task will be assigned. This robot will navigate the shortest route to the objective, however the shortest route might not be the fastest or most efficient route. People might walk in front of the robot or other robots might hinder the task. If a robot knows routes with high traffic and tries to avoid these it would be more efficient for every party involved. In short, this phase will focus on implementing navigation in the application and trying to improve it by for instance, implementing these routes with high traffic to avoid.

*Implementing UML:* A UML design has been made for ROS1. In this project the UML design will be reimplemented in ROS2 and improved on if possible.

*(Optional) Modelling the virtual environment:* The project will be tested in a virtual environment that looks like the physical environment. This 3d model of the environment already exists but it is not very detailed. This phase is optional because it is not the client's priority. If there is extra time, a new model identical to the physical environment will be made.

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

## 2. Research

In this chapter the researches conducted to get a clear view of the project are summarized. The Blackboard research is about the design pattern used in the project. This research explains how the design pattern works in general and in this specific project and what the pros and cons are.

The ROS research is about how ROS works, what the differences are between ROS1 and ROS2 and where this comes forward in this project. The purpose of this research is an introduction to ROS and clearing up what needs to be changed during conversion from ROS1 to ROS2.

### 2.1 Blackboard research

The blackboard architecture is a design pattern which can be implemented for numerous projects including the blackboard fleet manager. To understand this design pattern a research question was created to see how the blackboard implementation works. This part of the research is intended to answer the following question:

#### **How is the blackboard architecture implemented in the current prototype?**

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

*‘What kind of information is communicated using blackboard and why?’*

*‘How does the blackboard architecture work?’*

*‘What are the pros and cons of using this design pattern?’*

The results of this research are used for improvements in the newly designed blackboard.

#### **Hypothesis**

It is expected that the robots are the knowledge centre of this project. These communicate with the blackboard to figure out who has the best qualities for the job.

#### **Research strategies**

To answer the research question the following strategies were applied:

- Library: Literature study, Design pattern research, Available product analysis
- Field: Document analysis

This research started by informing myself of the blackboard using the Stanford university study and the summary of previous interns’ research about the design pattern. Then playing around with the available product helped to see how this design pattern could be implemented in a project.

### How does the blackboard work:

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 as seen in [figure 2](#) below:

*Knowledge source:* This component is needed to solve the problem. It is possible to have multiple knowledge sources which operate separately and independently. The knowledge sources can be seen as specialists in sub-fields of global application and are only able to solve sub- problems. They read and write relevant data in a blackboard which is a structured global memory where a solution to the problem under consideration is incrementally constructed.

*Blackboard data structure:* This is where the problem-solving data is kept in a global database, the blackboard. Knowledge sources change to the blackboard which lead incrementally to a solution to the problem. Communication among the knowledge sources takes place solely through the blackboard.

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

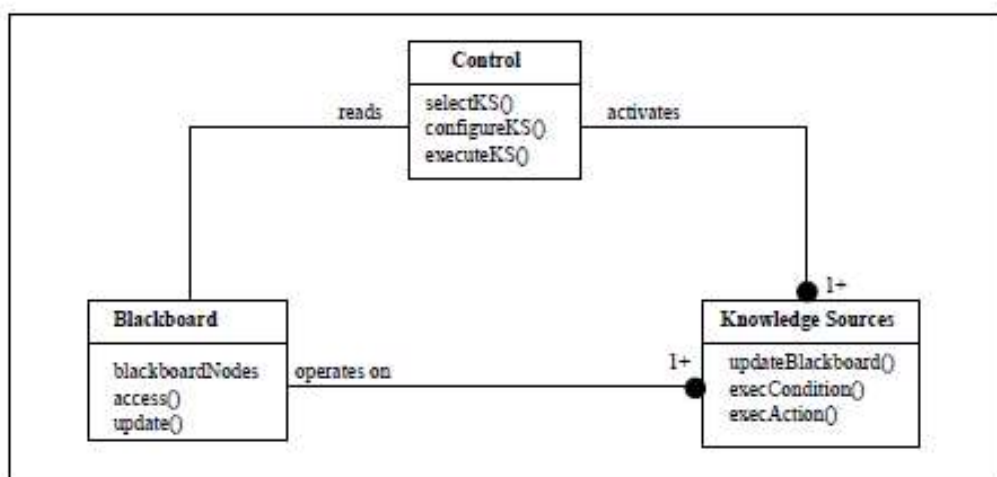


Figure 2 General blackboard structure

Generally, each knowledge source has a set of triggering conditions that can be satisfied by particular kinds of events. For this project the triggering conditions would be adding a task. When an event satisfies a knowledge source's triggering conditions, the knowledge source is enabled and its parameters bound to variable values from the triggering situation. A given knowledge source will be enabled, and therefore executable, whenever events satisfying its triggering conditions occur, regardless of its relative utility in achieving the current goals.

It is the purpose of the control component to select the best knowledge source for immediate execution. The problem solving process is opportunistic in that sense that the activations of knowledge sources are not scheduled in advance but determined at every control cycle depending on the current situation. In the basic blackboard model, control strategies are fixed.

## Fleet manager communication using blackboard

To understand what is communicated over the blackboard we first have to understand what the blackboard is used for in this project.

For this fleet manager project the knowledge sources are the robots. The problem that they have to solve is the cost of the task that is assigned to the blackboard. This task gets broadcasted over a topic and received by all the robots. The robots calculate the cost of the task and send back the results. A controller then selects the robot (knowledge source) with the lowest cost. This robot executes the task.

This task could be anything from navigating to picking up an object and moving it somewhere. This task is specified in a TaskMsg.msg:

```
int16 taskid
int16 priority
int16 tasktype
int16 payload
int16 taskstate
float32 cost
float32 energycost
int16 robotidgeometry_msgs/Pose[] pose
```

The robots (knowledge sources) calculate the cost of the task and send back a TaskCost.msg:

```
int16 taskid
float32 taskcost
int16 robotid
float32 energycost
```

With this information the controller knows which robot has the lowest energy cost for the specific task and can assign a robot to the task by assigning the taskId to the robotId.

The communication between the Blackboard and the knowledge sources is done via ROS Topics. The following diagram ([figure 3](#)) elaborates on the communication:

## Blackboard design pattern pros and cons

Pros	Cons
Changeability and maintainability	Difficulty of testing
Fault tolerance	No good solution is guaranteed
Reusable knowledge sources	Good control is hard to build
Can be used in physical environments	High development effort
	Expensive

Table 1 Blackboard design pattern pros and cons

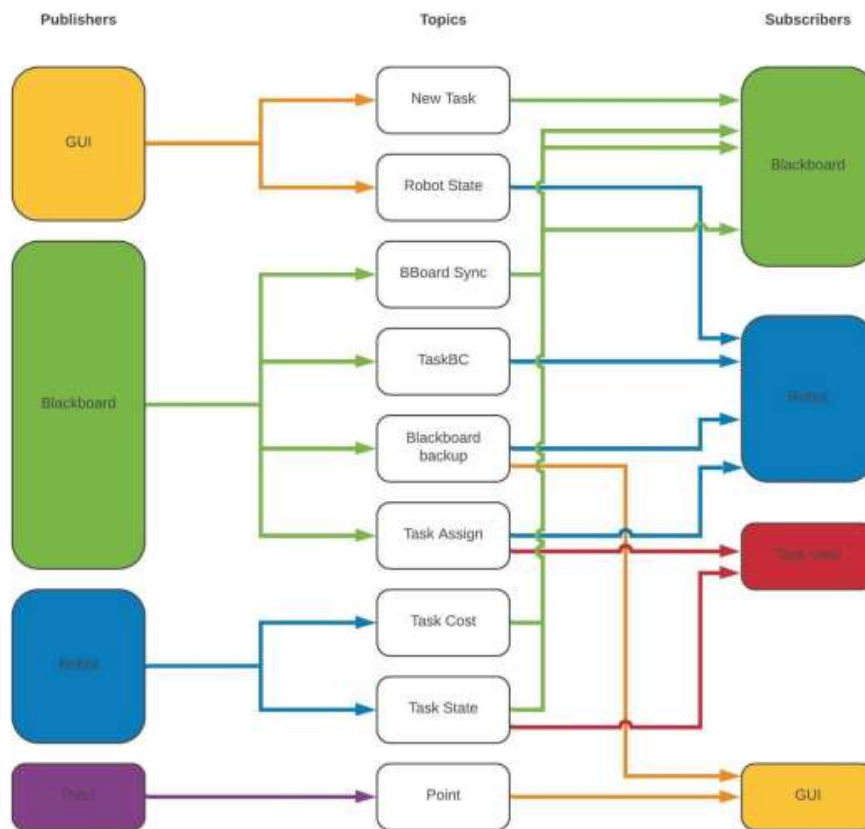


Figure 3 Topic communication diagram

## Conclusion

As mentioned in the research, a blackboard consists out of 3 major components. In the existing prototype's case these 3 components are the blackboard, the controller, and the robots (knowledge sources). A user adds a task to the fleet manager via the GUI. This task is then broadcasted to the robots. The robots calculate the cost of executing this task. This includes things like:

- How far is it?
- How much energy is this going to cost me?
- Can I handle the payload?
- What is the priority of this task and does it outweigh my current task?

The robot sends back the result of this calculation to the blackboard. The robot with the lowest cost result will receive the task. The controller makes sure this task gets executed in a separate thread and thus not blocking the entire system.

As previously described ROS communicates via ROS topics. A ROS node publishes a topic to communicate over. This means this node can publish data (send data) over a certain topic. A subscriber subscribes to a topic to receive information published by the publisher. So [figure 3](#) shows the data flow from one node to the other via the topics.



## 2.2 ROS Research

The main focus of this project is to convert the current existing ros1 blackboard-based fleet manager program to ROS2. To achieve this a research question was created to see how ROS works and how it relates to this project. This part of the research is intended to answer the following question:

### **What parts of the current prototype should be reimplemented in the conversion to ROS2 and how will they be reimplemented?**

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

*‘What is ROS?’*

*‘What are the differences of implementation between ROS1 and ROS2?’*

*‘Which of these differences are currently in the prototype and thus have to be reimplemented in ROS2?’*

*‘How will these parts be reimplemented?’*

### **Hypothesis:**

It is expected that a few little changes have to be made in the python code and the most drastic changes are going to be in the Launch files, communication and package building. The launch files have to be converted to python code instead of XML and packages are build different due to the lack of a ROS core.

### **Research strategies:**

To answer the research question the following strategies were applied:

- Library: Literature study, Available product analysis
- Workshop: Prototyping

This research started by informing myself of ROS via the ROS wiki page and tutorials provided to me. Then trying to start the current prototype gave me a lot of insight into how ROS works. Then I repeated this for ROS2 which resulted a list of differences.

### **Definition of ROS:**

ROS is a Robot Operating System. It is a flexible open-source framework for writing robotic software. It basically is a collection of tools, libraries and conventions that aim to simplify the task of creating a complex and robust robot behaviour across a wide variety of robotic platforms.

### **ROS1 and ROS2 differences:**

To know what needs changing when converting from ROS1 to ROS2, it is important to look at the differences between the two. Wiki.ros.org, created by open robotics, has made a list of all the differences between ROS1 and ROS2. These changes are listed in the attached ROS research document.

### Changes in the current program:

There are several major changes to the ROS implementation. In the following section I will compare the differences in ROS1 and ROS2 with the packages of the current prototype to see where they need to be changed.

Pkg names	Nodes	Launch files	Services	Actions	Parameters	Messages
SimenV						
Robot1						
Bbinstance						
Blackboard						

	Have changing points in package and thus need to be changed
	Not in package

Table 2: Relation between packages and the differences in ROS1 and ROS2

As you can see in the [table 2](#) above, almost all difference points have been touched except for Services and Actions. These are not used in the current prototype and do not have to be changed. This does not mean they are useless. Maybe the Services and Actions can be used to improve on the blackboard.

### Reimplementation:

Difference points	Ros1	Ros2	Change in implementation
<b>Nodes</b>	<ul style="list-style-type: none"> <li>Uses the Rospys library</li> <li>No coding structure required</li> </ul>	<ul style="list-style-type: none"> <li>Uses the Rcl(py) library</li> <li>Convention on writing nodes</li> </ul>	<ul style="list-style-type: none"> <li>Use rcl instead of rospys/roscpp</li> <li>Inherit from node class for all ROS2 functionality</li> </ul>
<b>Launch files</b>	<ul style="list-style-type: none"> <li>Written in XML</li> </ul>	<ul style="list-style-type: none"> <li>Mainly written in Python but XML still works</li> </ul>	<ul style="list-style-type: none"> <li>Convert launch files from XML to python. This brings more modularity and is more documented.</li> </ul>
<b>Parameters</b>	<ul style="list-style-type: none"> <li>Handled by parameter server which is handles by ROS master</li> </ul>	<ul style="list-style-type: none"> <li>No ROS master so no parameter server</li> <li>Each parameter is specific to a node. Like each node has its own parameter server.</li> </ul>	<ul style="list-style-type: none"> <li>Make .YAML files node specific</li> </ul>
<b>Messages</b>	<ul style="list-style-type: none"> <li>Messages, actions and services all use "PKG/MSGFILE." as their message type. This is confusing.</li> </ul>	<ul style="list-style-type: none"> <li>Messages, actions and services use their folder name in the message type. "PKG/COMTYPE/MSGFILE."</li> </ul>	<ul style="list-style-type: none"> <li>When importing msg files, add the folder name to the path.</li> </ul>

Table 3: Reimplementation changes

Blackboard Distributed Fleet Manager Implementation for ROS2  
Fontys Mechatronics and Robotics Lab – Marc Meulenstein

To know how we can reimplement these parts in ros2 we need to look at the differences between ros1 and ros2 again. [Table 3](#) will give a good overview of the difference points and what needs to be changed.

### Conclusion:

As the [table 3](#) above suggests, changes need to be made in these four fields. The following [table 4](#) will show the package names with the changes in implementation that need to be made in that package:

Pkg names	Change point 1	Change point 2
<b>Simenv</b>	Convert simenv.launch from XML to python.	N/A
<b>Robot1</b>	Convert robot1 launch files from XML to python.	Make .YAML files node specific.
<b>Bbinstance</b>	Use rcl instead of rospy And inherit from node class.	Convert blackboard.launch from XML to python.
<b>Blackboard</b>	Use rcl instead of rospy and inherit from node class.	When importing msg files, add the folder name to the path

*Table 4: Packages with changes that need to be made*

### 3. Conversion process

In this chapter the process and results of the conversion will be discussed. The first part of the process is the orientation phase which is an important phase for this project due to the lack of knowledge in ROS and Python. In this phase the old system in ROS1 is also launched to give more insight on the project.

The conversion of the program from ROS1 to ROS2 will also be discussed. The process of this phase will be explained thoroughly, as will problems that were encountered and how these were fixed.

#### 3.1 Orientation

Orientation was a big part of this project due to the lack of knowledge in Python and ROS. Several tutorials provided by internship tutors and ones found online like [emmanuel.robotis.com](http://emmanuel.robotis.com) helped solve this problem. The decision was made to only learn the Python basics as more complicated functions are not needed yet and will be learned on the way. The ROS tutorials are more elaborate than the Python tutorials. Here the focus is first on ROS basics and how the operating system worked all together and then navigating a robot and getting a mapping of the environment.

The next orientation step is understanding and running the current code implementation (the ROS1 implementation). This is done by explaining the current understanding of the class diagram ([figure 4](#)) to experts on this project (Pablo and Sjriek). This gives a good insight into the project and the code classes as they corrected the mistakes in the explanation and filled in the missing parts. It also gives an opportunity to already discuss possible improvements the project might have.

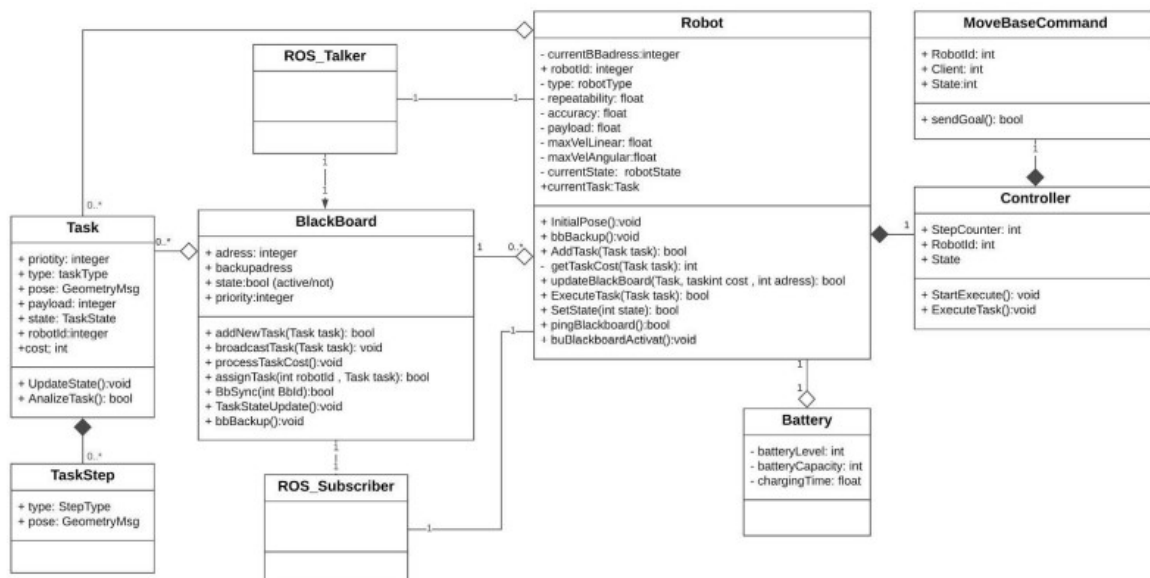


Figure 4: Discussed class diagram ros1

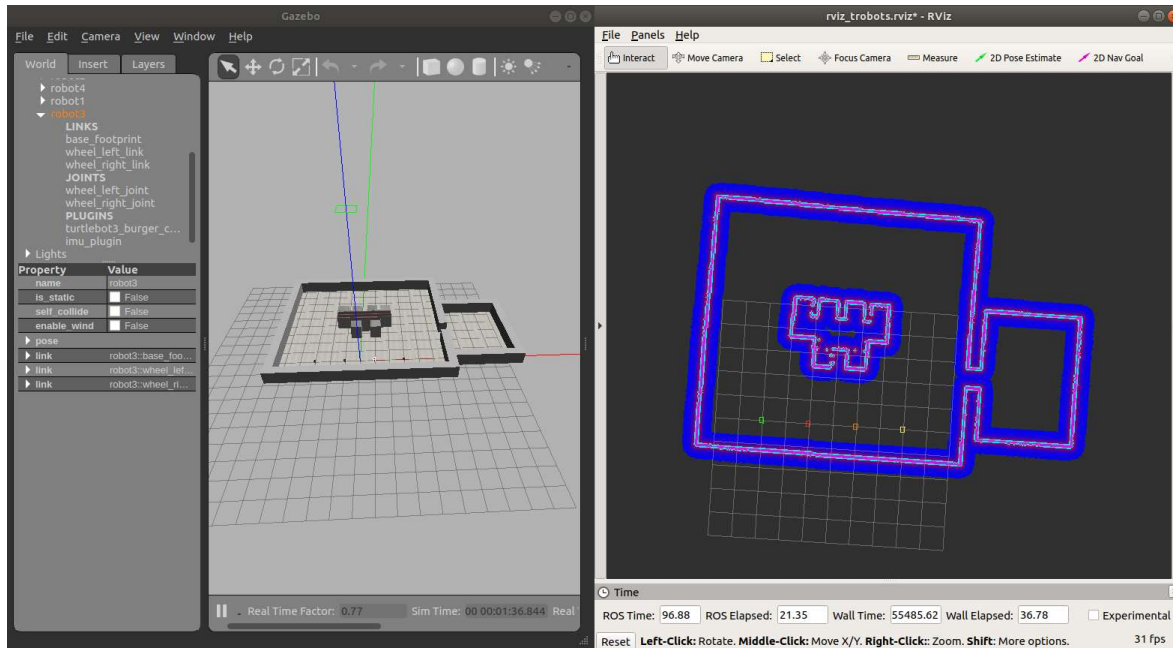


Figure 5: Gazebo and RVIZ when first launched correctly

Running the current code is more difficult than initially expected. The code is downloaded from a git repository. No explanation is given about how to start the program. The tutorials about launch files done in the beginning of the orientation phase luckily helped solve this problem. Launch files are used to start nodes. When launching the `simenv.launch` file, it starts the RVIZ and Gazebo program. This however results in an error. This error is solved by changing one of the file paths inside of the `environment.world` file, as it was hardcoded to the drive of the previous intern. When the `simenv.launch` file is launched now, it results in [Figure 5](#) above. To get a good idea on how the program works, some testing was done using the ROS commands. By using the commands, it is possible to publish a message to a topic in run-time or listing the nodes that are currently active. `Rqt_graph` was also used to see the communication between nodes. The communication of the robots with the blackboard was more apparent.

### 3.2 Conversion from ROS1 to ROS2

The conversion from ROS1 to ROS2 is the main goal of this project. After the existing prototype was fully running, it had to be converted to ROS2. Converting begins by copying all the files from the ROS1 prototype to a ROS2 environment. According to the ROS research conducted in the previous phase this will not be enough to run the program in ROS2. The first actual step of converting is rewriting the launch files from XML to Python. To do this, the external ROS package “ros2-launch-file-migrator” is used. This package takes a ROS1 XML launch file and converts it to a ROS2 Python launch file. The conversion in this package is not perfect so some manual changes in the launch file parameters are necessary. For instance, the remapping of the topics is not implemented in the ros2-launch-file-migrator and have to be manually added. As the ROS research suggested, some of the Python nodes have to be changed too. Classes now have to inherit from “Node” and the library on top of ROS for python (Rospy) does not exist anymore and has been replaced by Rclpy. Certain functions are therefore different and have to be changed.

#### Launching and error resolving

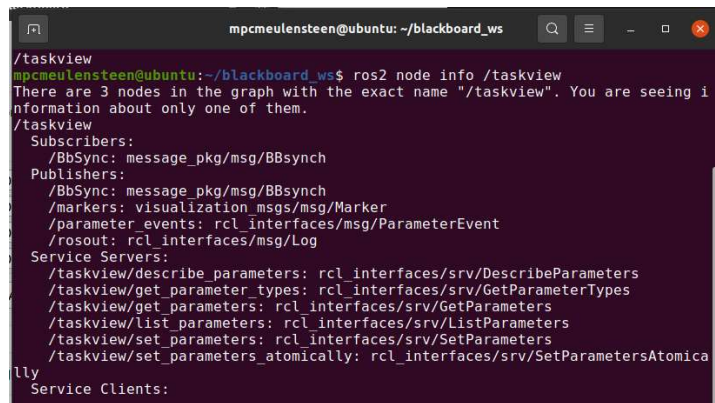
The first plan of launching is the same as the plan of launching for the old implementation. When all the changes in the launch files and nodes are applied the launch file is started. This results in a lot of errors and bugs. Trying to solve these bugs results into more bugs that were harder to solve. The conclusion was reached that the current method of converting is backwards and a different approach is necessary. This method was backwards because launching the launch file is what you should do at the final stage of converting when you are sure every node does what it is supposed to. The amount and type of errors do give clarity on certain things missed in the ROS research. For instance, there was no consideration of the changes in python, even though ROS1 uses python2 and ROS2 uses python3. This means all functionality in the current nodes that were specifically python2 have to be changed as well. After discussing this a new approach is taken in consideration.

## Re-writing the nodes and testing

The new approach of converting is rewriting all the classes and testing them as they were being implemented. This approach also brings another problem to light. Some ROS1 packages are no longer in ROS2 or have changed to other packages. This problem is solved by finding another method that could be used for the same purpose in ROS2 or even rewriting those packages to ROS2 as well. For instance, the Rviz\_tools package is not working for the ROS2 implementation. The file used in the ROS1 implementation has now been rewritten to a ROS2 format and used again. This process worked very well for the blackboard, and bbinstance packages as they contain actual self-programmed nodes.

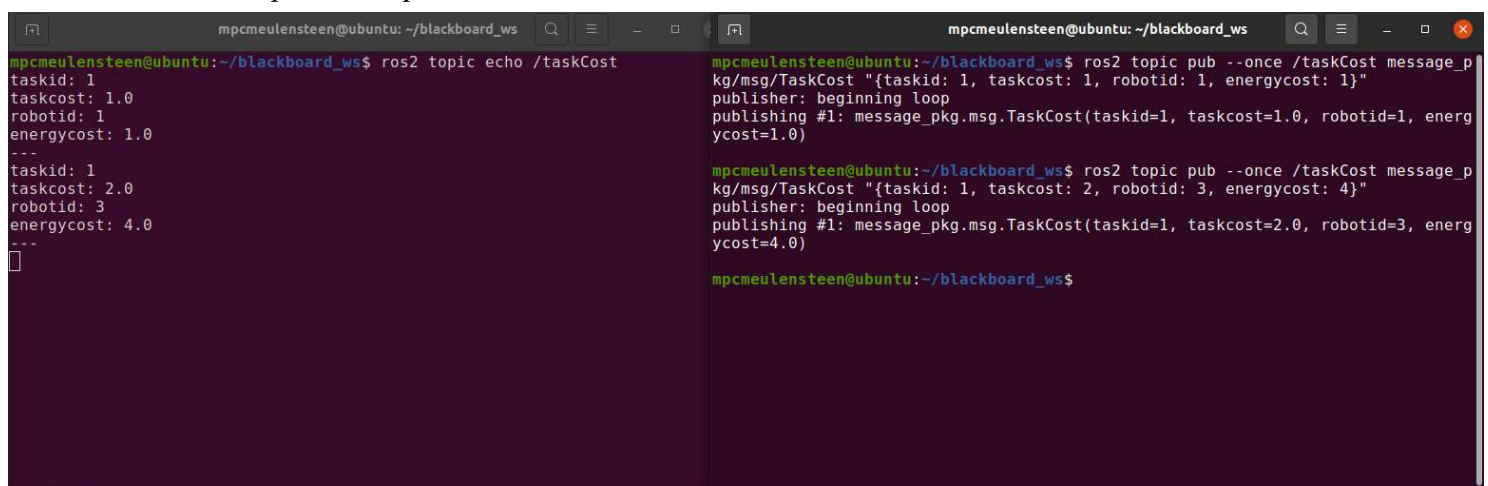
The testing of the classes was done by using commands in the terminal. For instance, to test communication between 2 nodes over a topic, you need to know which nodes are running. This was done by: **Ros2 node list**. To see if this node publishes or is subscribed to a node, the command: **Ros2 node info** was used.

The ROS terminal commands are also very useful for testing real time. To test if a callback function (a function triggered when a certain event occurs, e.g. a message is published) was triggered and handled, **Ros2 topic pub** was used. This command allows the user to publish a message over a certain topic during runtime. To see if this message is actually published over the topic the command **Ros2 topic echo** can be used. This command prints the data of any message published on the specified topic.



```
mpcmeulensteen@ubuntu: ~/blackboard_ws
/taskview
mpcmeulensteen@ubuntu:~/blackboard_ws$ ros2 node info /taskview
There are 3 nodes in the graph with the exact name "/taskview". You are seeing i
nformation about only one of them.
/taskview
Subscribers:
  /BbSync: message_pkg/msg/BBsynch
Publishers:
  /BbSync: message_pkg/msg/BBsynch
  /markers: visualization_msgs/msg/Marker
  /parameter_events: rcl_interfaces/msg/ParameterEvent
  /rosout: rcl_interfaces/msg/Log
Service Servers:
  /taskview/describe_parameters: rcl_interfaces/srv/DescribeParameters
  /taskview/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
  /taskview/get_parameters: rcl_interfaces/srv/GetParameters
  /taskview/list_parameters: rcl_interfaces/srv/ListParameters
  /taskview/set_parameters: rcl_interfaces/srv/SetParameters
  /taskview/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomica
lly
Service Clients:
```

Figure 6: taskview node information



```
mpcmeulensteen@ubuntu:~/blackboard_ws$ ros2 topic echo /taskCost
taskid: 1
taskcost: 1.0
robotid: 1
energycost: 1.0
---
taskid: 1
taskcost: 2.0
robotid: 3
energycost: 4.0
---
[]

mpcmeulensteen@ubuntu:~/blackboard_ws$ ros2 topic pub --once /taskCost message_p
kg/msg/TaskCost "{taskid: 1, taskcost: 1, robotid: 1, energycost: 1}"
publisher: beginning loop
publishing #1: message_pkg.msg.TaskCost(taskid=1, taskcost=1.0, robotid=1, energ
ycost=1.0)

mpcmeulensteen@ubuntu:~/blackboard_ws$ ros2 topic pub --once /taskCost message_p
kg/msg/TaskCost "{taskid: 1, taskcost: 2, robotid: 3, energycost: 4}"
publisher: beginning loop
publishing #1: message_pkg.msg.TaskCost(taskid=1, taskcost=2.0, robotid=3, energ
ycost=4.0)

mpcmeulensteen@ubuntu:~/blackboard_ws$
```

Figure 7: Communication over /taskCost topic



In [Figure 7](#) above a message of type TaskCost is published over the topic '/taskCost'. This message contains the following: taskid = '1', taskcost = '1', robotid = '1', energycost = '1'. When this message is published on right side of the image, the left side echo's the message. Even when the parameters are changed and the message is published again the left side echo's the message with the changed parameters which suggests the communication over this topic is working.

#### Launching after re-writing nodes

In order to launch the program correctly all nodes have to be running and communicating perfectly. To test this, all packages were launched separately first.

### Ros2\_blackboard

The ros2\_blackboard package is the package containing the blackboard logic. To run and test this package the testMain.py file was created. This allows the user to make instances of the classes and actually run the code. Right now the testMain.py creates a publisher (class inside RosCommunication) for the Blackboard and creates a Blackboard instance. It also creates a publisher for the Robot and a robot instance. This allows the user to test the communication over topics using ros2 commands in the terminal.

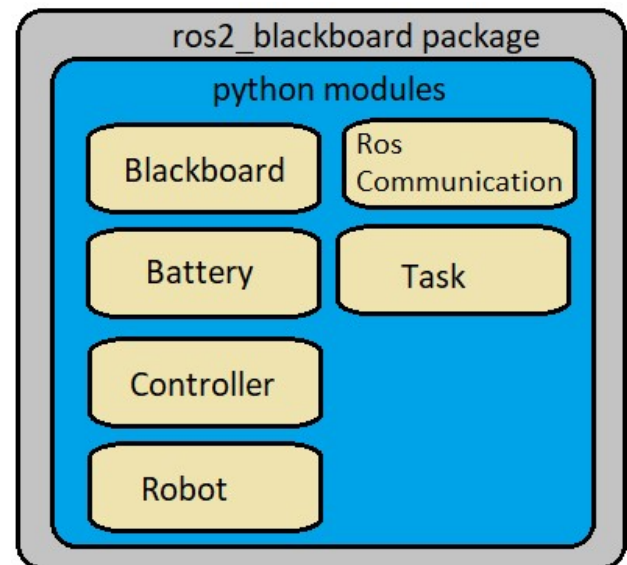
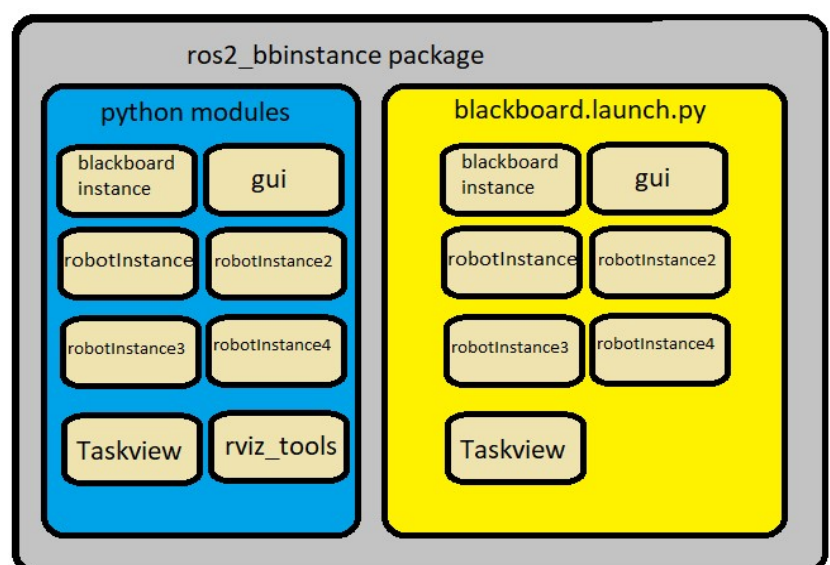


Figure 8: ros2\_blackboard package

This package does not have a launch file or any way to start the nodes except for the testMain.py. The python modules from this package are launched in the Ros2\_bbinstance package. The ROS1 implementation of the blackboard package also contained ROS messages which are used for the communication via topics. These are now located in a separate package as messages (msg) in ROS can only be in a C++ package. Ros2\_blackboard package is a Python package (see [figure 8](#)).

### Ros2\_bbinstance

The ros2\_bbinstance package is the link between logic and user input. When launched this package creates a GUI (graphical user interface) for creating tasks and publishing them to the blackboard over the '/newTask' topic. This package also creates a blackboard





instance and the robot instances. The GUI can also change states of the robots by publishing a message over the '/robotState' topic.

The `ros2_bbinstance` package creates a taskview. This too is a graphical interface and shows what tasks are in the tasks list with a status of the task as showed in [figure 10](#). Taskview gets the information from the blackboard over the '/bbSync' topic. For information about the python modules and the launch file of this package, see [figure 9](#).

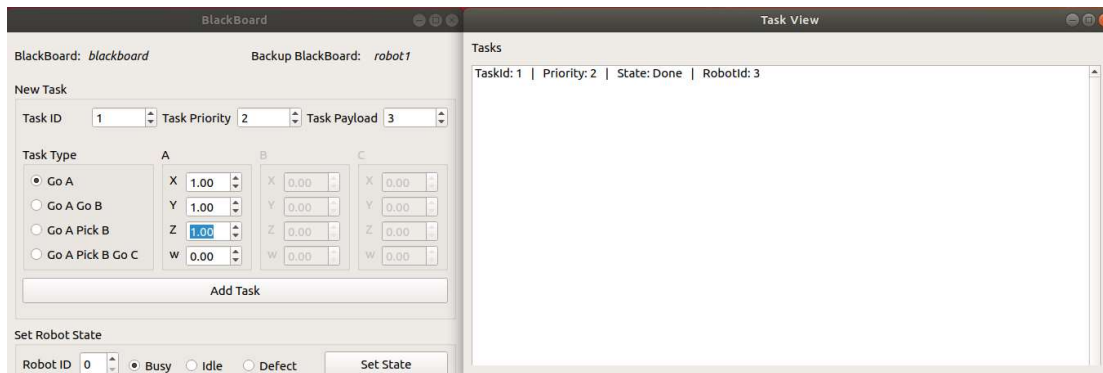


Figure 10: GUI and taskview with added task

In the current state of the project the taskview experiences some problems. Either the taskview GUI (task list) is shown or the taskview GUI does not load and the node keeps looking for new tasks and updating the list. At the moment it's believed that the cause of this problem lies within a difference between `rospy.spin()` (ros1 spin function) and `rclpy.spin()` (ros2 spin function). The spin function makes sure the node is not exited after going through the code ones. The following line is for subscribing to a topic and updating the taskview via the `self.taskview()` call-back function: `self.create_subscription(BBsync, 'bbSync', self.taskview, 1)`

The main of the taskview opens a QtWidget application (the actual UI). When `ui.setup()` is called the UI is visually created, in that same function the subscriber is called. By calling `rclpy.spin_once(ui)` the UI actually gets loaded instead of a black screen that is just processing information. This problem is trying to be solved by running the visualization of the UI and the communication in separate threads. This is currently still not working.

```
def main(args=None):
    rclpy.init(args=args)
    app = QtWidgets.QApplication(sys.argv)
    mainwindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(mainwindow)
    mainwindow.show()
    rclpy.spin_once(ui)

    sys.exit(app.exec_())
```

#make ui instance  
 #setup ui + subscription  
 #show ui  
 #Run node ui once more,  
 without spin\_once ui won't  
 start. Using spin() the ui  
 won't start either.

## Navigation

The navigation package is responsible for launching Gazebo and RVIZ as well as the static\_transform\_publisher (STP) nodes and the map\_server. The map\_server provides maps to the rest of the Nav2 system using both topic and service interfaces.

The static\_transform\_publisher nodes are part of the tf2\_ros package. These nodes are useful to define the relationship between a robot base and its sensor or non-moving parts.

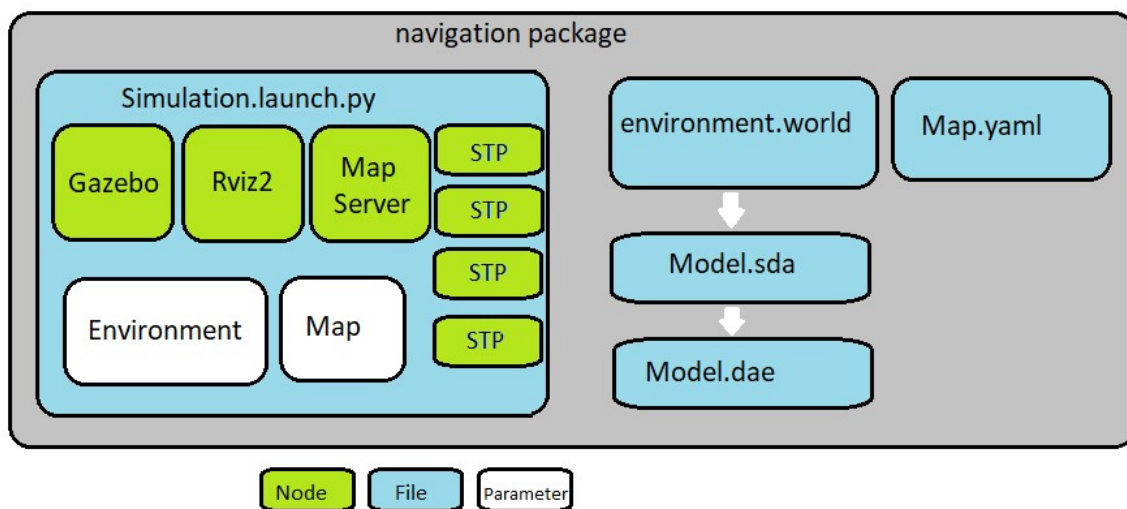


Figure 11: Navigation package

This package is comparable to the SimenV package from the ROS1 implementation, but some major changes are implemented in this package. There are a lot of packages used in the SimenV package that no longer exist or have changed. For instance the map\_server package has become part of the larger Nav2 package, and the TF package has changed to the tf2\_ros package with different arguments then in ROS1. The RVIZ and Gazebo packages are also launched in a different way then in ROS1.

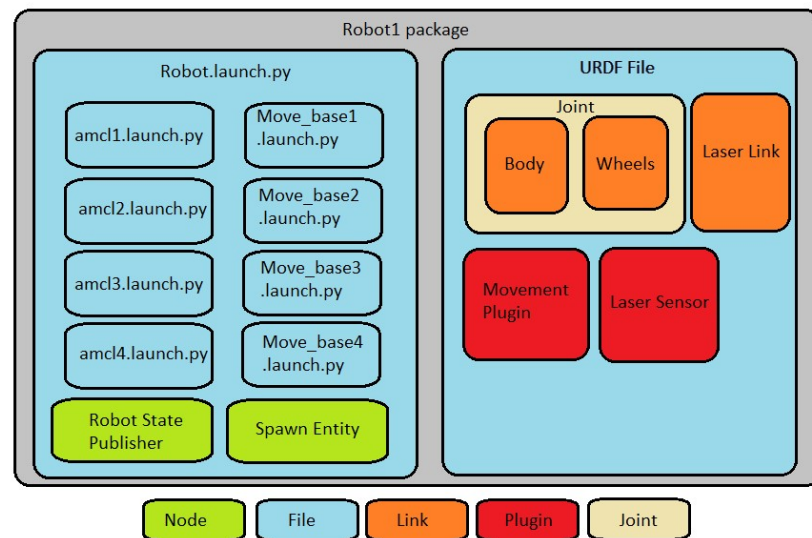
Gazebo is now launched by launching a gazebo server (gzserver) and a gazebo client (gzclient). The environment in which the robots are simulating is also in this package in the 'worlds' folder and is passed as a argument with the gazebo server.

RVIZ is launched almost the same way as in ROS1, but the configuration file needed to display what needs to be displayed has changed. The config file currently used by Rviz is called 'rviz\_troboots.rviz'. This file contains rviz classes that help with visualizing the robot in the environment. These classes have also changed names with the conversion to ROS2.

In its current state the simulation.launch.py starts all its nodes. The Gazebo node shows the environment.world and spawns in 4 robots. The map server node starts correctly and the static\_transform\_publisher nodes had some problems. The syntax of the arguments for the static\_transform\_publisher have changed in ROS2. This is fixed by finding the ros2 implementation of the STP and changing the arguments to fit the ones in the STP file. The RVIZ node starts up the rviz program with the correct Rviz config file, but does not receive the map and therefor does not show the environment. Right now its believed some communication between the the rviz config file or the map server does not work correctly. Research into this problem suggested communication over the /global\_costmap topic does not function but the reason why is currently still unknown.

## Robot1

The robot1 package contains the actual description of the simulated robot. This package consists out of 2 major files, the urdf and the launch files. [Figure 12](#) shows the files in the robot1 package. The Robot.launch.py file contains the nodes to parse the urdf file and spawn the robot in gazebo. The urdf is used as a parameter in the Spawn Entity node which spawns the robot in the simulated environment. The Robot state publisher node is used by gazebo to process the joints positions and visualize them in the 3D environment.



The move\_base launch files includes robot specific arguments such as odometry frame, laser topic, and base frame which allows bt\_navigator (ROS1 version of bt\_navigator is move\_base) to communicate with a robot. It also has parameters that refer to the configuration files which describe the behaviour of global and local planners.

Amcl stands for Adaptive Monte-Carlo Localizer. It is a probabilistic localization system for a robot moving in a 2D environment. The amcl uses a particle filter to track the position (pose) of a robot against a known map. With current implementation, Amcl only works with laser scans and laser maps. These launch files contain robot specific information and the robot's initial pose when starting up. Topics in these launch files are remapped from the default namespace to a robot specific namespace (/robot1/\*default\_ns\*).

The robot1 package also includes the robot\_description.urdf. URDF stands for Unified Robotic Description Format and is an XML file format used in ROS to describe a robot with all its elements, such as physical robot parts and joints, simulation plugins, and topic names. This robot\_description is not necessarily used as the old ROS1 implementation also did not use this URDF file in its final version. URDF files have not changed from ROS1 and thus should not need any changes.

### 3.3 Design

In this chapter the class diagram of the blackboard system in ROS2 will be discussed. Changes from the old class diagram will be explained.

ROS2 Blackboard Class diagram

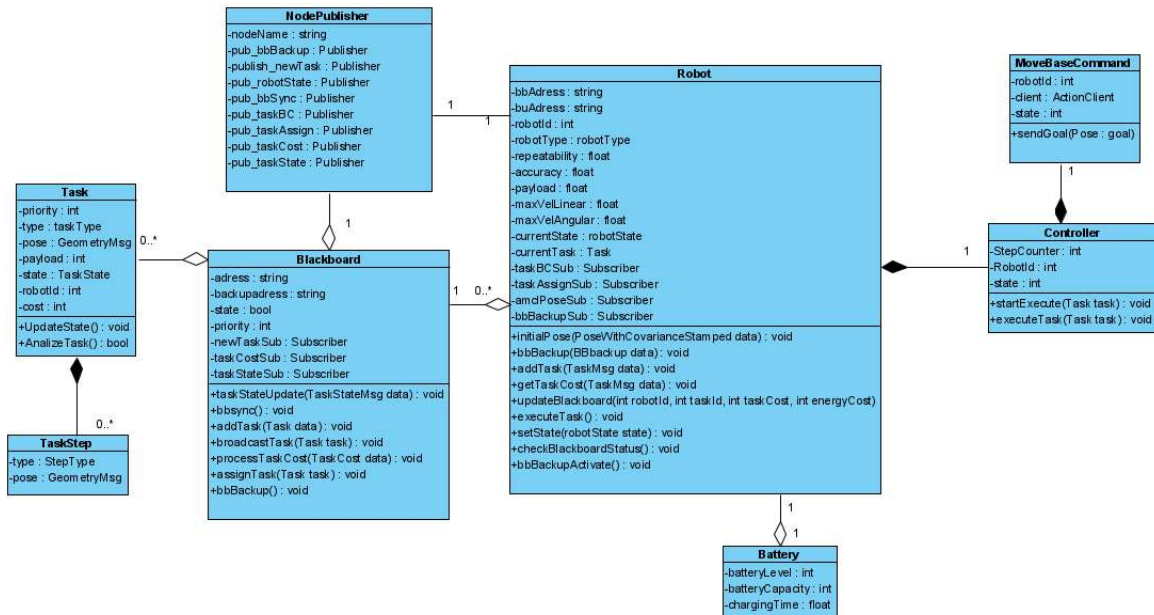


Figure 13: ROS2 Blackboard class diagram

The ROS2 blackboard execution looks like [figure 13](#) above. The **Blackboard** class is the main component of this system. This is where newly created tasks are added to the tasklist, tasks are broadcasted to the robots and tasks get assigned to robots. **Tasks** are pretty self-explanatory. They have some parameters, for instance, a position, a taskType, a payload, etc. The blackboard communicates this task to the robot using a Topic. This is why the blackboard has a **NodePublisher**. The task is published over a topic and the robot who uses a subscriber like “taskAssignSub” receives this task. As previously mentioned in the Blackboard research, the **Robot** class is the knowledge source. This means they collect data and are there to solve sub-problems. A sub-problem in this case is the cost it takes to execute a task. When the blackboard node suddenly crashes, one of the robots can run a backup of the blackboard instance. The **Battery** class just keeps track of the battery of the robot. The **Controller** class creates a separate thread for the execution of a **TaskStep** which is just a part of a task. This makes sure the whole program doesn’t wait for one robot to finish it’s task before assigning the next one. The **MoveBaseCommand** class sends the goal (pose of task) to the MoveBaseActionClient. This ActionClient actually makes the robot move and returns a result.

Changes from the old diagram are the RosPublisher and RosSubscriber classes are gone. Now there is one NodePublisher with all topics which the Blackboard and robot uses. The subscriber class has changed into attributes inside of the classes. For instance, Blackboard has a attribute newTaskSub which is a Subscriber. Further changes are small like new/different attributes or operations.

## ROS2 simulation flow chart

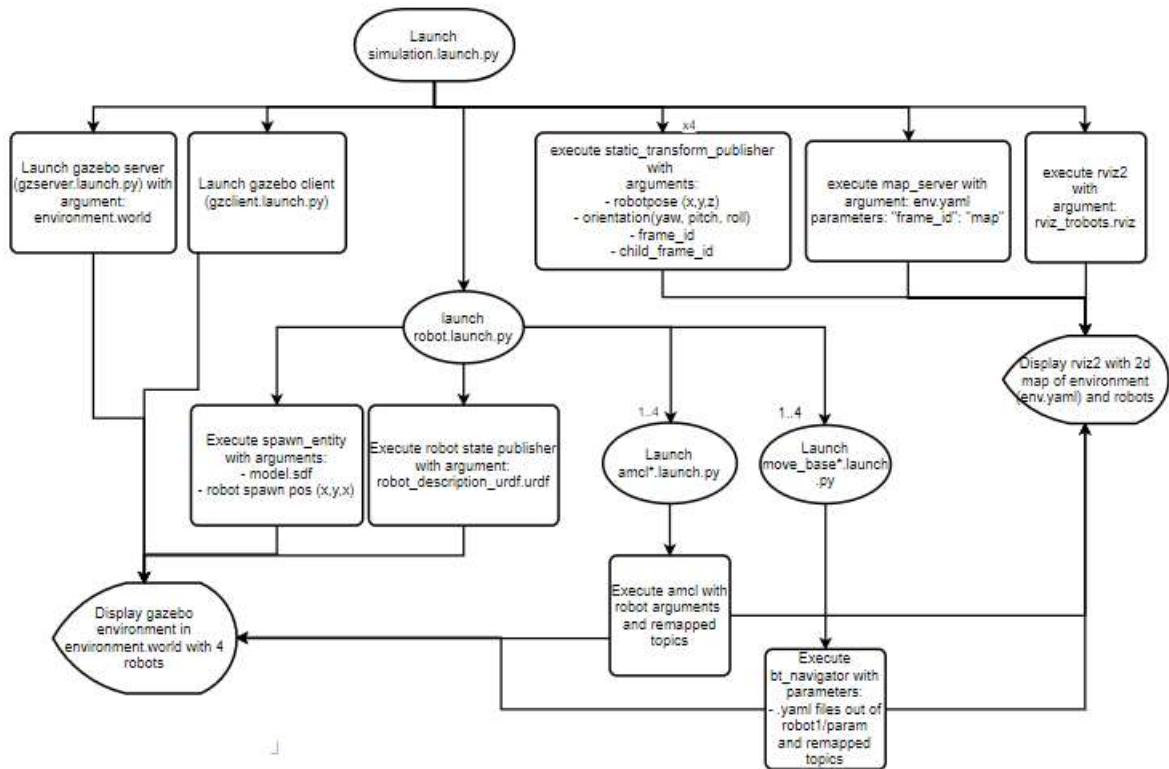


Figure 14: Flow chart of launching simulation.launch.py

The ROS2 simulation part looks like the flow chart in [figure 14](#) above. This image clarifies what nodes are started where, and how launching the launch file results into a gazebo and rviz2 display. As seen in the image above, gazebo and Rviz2 are both launched in simulation.launch.py. By launching robot.launch.py robots are spawned in gazebo and communication between the virtual robot and the displays is launched via amcl.launch.py and move\_base.launch.py.

## 4. Testing

In this chapter the test cases and reports are created in order to test the new system. Also an example of a test case will be showed. No excessive testing was required by the client at this stage of the project. A list of test cases is listed below. More information about the test cases and results can be found in the attached test plan.

Test cases:

- Starting all required nodes
- Adding a 'Go A' task to the taskview
- Adding a task with multiple goals.
- Adding multiple tasks to the taskview
- Adding a task with highest priority while other tasks exist
- Killing a blackboard node and starting the process
- Executing a 'Go A' task
- Functionality of previous test cases with backup blackboard

Not all test cases have been tested as parts of the project currently are not working yet. These will be done in the future of the project when all functionality is fully supported. For a test case example see [table 5](#) below.

Testcase 1	
Name	Test starting all required nodes
Prerequisite	No nodes are started
Steps	1. Open terminal 2. Go to workspace ('cd blackboard_ws') 3. 'ros2 launch navigation simulation.launch.py' 4. 'ros2 launch ros2_bbinstance blackboard.launch.py' 5. 'ros2 node list'
Data	Nodes: /amcl1, /amcl2, /blackboard, /gazebo (etc.)
Result	<b>Passed</b> Nodes: /amcl1, /amcl2, /blackboard, /gazebo (etc.)

Table 5: testcase example



## 5. Conclusions

As stated in the initial assignment description in this document, the goal of this project was converting the old ROS1 implementation of the distributed fleet manager to a new ROS2 implementation. The different aspects of the project were:

- Research into new programming languages, ROS and the blackboard design pattern.
- Converting the working ROS1 prototype into a ROS2 prototype
- System or design improvements that can be made in this system.

This project I tried to answer the following question: *“How can the current blackboard solution be implemented and improved in a more reliable environment with a more realistic robot behaviour?”*

After concluding the research about ROS and what differences ROS1 and ROS2 have, it became clear how similar a simulation of a robot and working with an actual robot is, and how working with an operating system like ROS can make the controlling of a robot much easier. As the research said, ROS1 has a ROScore and ROS2 does not. This means ROS2 is a decentralized system which is exactly what this decentralized blackboard fleet manager needs. The current blackboard solution can be improved in a more reliable environment with more realistic robot behaviour by converting from the centralized ROS1 to the decentralized ROS2.

The research conducted to investigate the blackboard design pattern provided more insight on the ROS1 implementation code and class diagram. As shown in the conclusion on the blackboard research, the communication between nodes became clear in [figure 3](#) which provided many answer to the questions at the time.

The conversion from ROS1 to ROS2 took more time then anticipated. Currently the ROS2 prototype is not functional. Every node in the `ros2_blackboard` package has been re-written in python3 instead of python2 and tested by using ROS Commands. All the messages are received over the topics. Publishers publish their messages and subscribers receive them and call there call back functions. All launch files have also been converted from XML to Python and fully functional.

The nodes in the `ros2_bbinstance` package have also been re-written and tested by using ROS commands. In this package there are some problems around the UI applications. The Taskview and GUI seem to only receive messages over the subscribed topics if the UI is not loaded in yet, and when loaded in, they don't receive any messages. It is currently believed that the problem lies within the `rcipy.spin_once()`. Without this code the GUI and Taskview do not display, but with it the messages will not be received. Multiple possible solutions like threading the communication and UI in different threads have been experimented with but currently it is still not working. All other aspects of the `ros2_bbinstance` package are functional.

The navigation and robot1 package launch files have been converted from XML to Python and are functional. The gazebo node is starts up and launches in the right .world map with 4 robots. The Rviz node experiences one problem of not receiving the map over the global\_costmap/costmap topic. It is currently believed to be an issue with the move\_base. The possibility of it being a problem in the Rviz config file is considered, but multiple tests indicate that the config file is working.

The current blackboard solution can be implemented and improved in a more reliable environment with a more realistic robot behaviour by converting to ROS2 where there is no centralized system and all nodes are independent. This will give nodes more freedom communicating with each other and every node can be found easily by other nodes. The 3d simulation environment can be improved to make the simulation more realistic and thus easier to implement on physical robots in the future. More improvements on the system can be found in the recommendations (chapter 6)

During the remaining period of this project, all the listed problems will be worked on, as well as testing and implementing possible improvements.



## 6. Recommendations

It is highly recommended to improve the cost calculation algorithm as it is one of the most important features for this project. Investigating into this further may lead to more efficient and realistic robot behaviour. At the moment the algorithm mostly consists of the robot closest to the goal will receive the task.

Investigating into creating high traffic paths taken by humans or robots to avoid and improve efficiency can be useful. Because the fleet manager is now decentralized in ROS2, robots nodes can find and communicate with each other easier then before. This way the most efficient paths can be communicated to each other as well as fixed points like charging stations, inventory area, pickup point, etc.

To improve the chance of easily using this project in a physical environment it is recommended the 3d simulated environment is improved by better measuring the BIC location and mapping this into a .world file.

It is also recommended investigating into the possibility of expanding the fleet manager with non-ROS robots. This could be very useful for this system in the future.

## Evaluation

I have learned a lot the past semester, in both technical and professional fields. I have developed many skills within programming but also conducting research and writing a report. I have learned a new programming language and of course ROS. Taking on this challenge of first learning the software I am going to work with and then jumping into a project, made me more confident in taking on new challenges like this.

This semester I wanted to improve on asking feedback as well as confidence and my reflecting. With asking feedback my goal was asking more frequently and in time for me to finish my documents. What I have noticed about myself is that I don't think asking feedback about my documents is a problem for me or needs to be improved. I actually had feedback from multiple coaches on multiple occasions which helped me a lot. However I have noticed that asking help when I need it is a bigger problem for me. This was extremely apparent when sitting alone in a working environment or working from home due to the covid-19 outbreak. If there is no one for me to physically ask a question, I find it hard to take the extra step and go out of my way to ask for help. I would really like to improve on this as asking help is extremely useful and would have probably saved me a lot of time.

I wanted to improve my confidence as I have always been afraid of making mistakes which is where my problem with asking help originates from, but as I previously said, taking on this challenge of learning something new and jumping into a project has given me confidence to just try new things.

Finally I wanted to improve my reflecting to find out where my interests lie within ICT technology. My goal was reflecting about different parts of this internship to find out what I like best. I have come to the conclusion that I actually like working with ROS and python a lot. Its very interesting to me how these robots are controlled, and even though I found the internship assignment hard, I did enjoy myself finding out how it all works. This has pushed me to choose the adaptive robotics minor here at Fontys mechatronics which will hopefully be just as exciting as this internship was.

I have expanded my knowledge of robotics software development by working with ROS. Writing and debugging a ROS package and python script for robotic tasks seemed very hard and confusing in the beginning and has become natural and fun.

Overall, I am a bit disappointed the whole fleet manager is not functional at the time of this report, but I would consider this internship a success as I have been able to run a large part of the system in ROS2 and learned a lot about the software involved in this project, as well as how to report properly. I am mostly happy I found something I enjoyed within ICT & Technology.

## References/Literature list

- Karl von Frisch (1973) Decoding the language of the bee*  
nobelprize.org <https://www.nobelprize.org/uploads/2018/06/frisch-lecture.pdf>
- M. van Osch (2020) Mechatronica and Robotica*  
Fontys.nl. <https://fontys.nl/Onderzoek/Mechatronica-en-Robotica.htm>
- Open Robotics (2021) ROS documentation*  
Wiki.ros.org. <http://wiki.ros.org/>
- Tina heidborn(2010) Dancing with bees*  
Mpg.de. [https://www.mpg.de/789351/W006\\_Culture-Society\\_074-080.pdf](https://www.mpg.de/789351/W006_Culture-Society_074-080.pdf)
- H. Ycnnny Nii (1986) Blackboard systems*  
i.stanford.edu. <http://i.stanford.edu/pub/cstr/reports/cs/tr/86/1123/CS-TR-86-1123.pdf>
- The Robotics Back-End (2021) ROS1 vs ROS2*  
Roboticsbackend.com. <https://roboticsbackend.com/ros1-vs-ros2-practical-overview/>
- Venessa Mazzari (2019) ROS vs ROS2*  
Generationrobots.com. <https://www.generationrobots.com/blog/en/ros-vs-ros2/>
- Open robotics (2021) ROS2 Documentation*  
Docs.ros.org. <https://docs.ros.org/>

## Attachments

Test Cases:

<https://docs.google.com/document/d/1supI17RufYG9TSmZ0xm9gPPdg7O-21OO-uXpkbuwJ8/edit?usp=sharing>