[Introduction 2](#_Toc1)

[Background and motivation 2](#_Toc2)

[Research questions and objectives 2](#_Toc3)

[Scope and limitations 2](#_Toc4)

[Literature Review 2](#_Toc5)

[Overview of ROS1 and ROS2 2](#_Toc6)

[Introduction to Google Cartographer and Navigation Stack 2](#_Toc7)

[Review of related work on porting navigation stack from ROS1 to ROS2 2](#_Toc8)

[Overview of open source fleet management systems, particularly Open-RMF 2](#_Toc9)

[Methodology 2](#_Toc10)

[Description of research platform and hardware used 2](#_Toc11)

[Steps taken to port the navigation stack from ROS1 to ROS2 2](#_Toc12)

[Hardware adaptation process from Rover Zero to Rover Mini 2](#_Toc13)

[Integration of the navigation stack with Open-RMF 2](#_Toc14)

[Testing and evaluation of the system 2](#_Toc15)

[Results and Analysis 2](#_Toc16)

[Presentation of results from testing and evaluation 2](#_Toc17)

[Comparison of system performance before and after the porting process 2](#_Toc18)

[Discussion on the challenges encountered and how they were addressed 3](#_Toc19)

[Conclusion and Future Work 3](#_Toc20)

[Summary of the research findings 3](#_Toc21)

[Contributions to the field of electrical engineering and robotics 3](#_Toc22)

[Recommendations for future work and improvements 3](#_Toc23)

[References [[references]] 3](#_Toc24)

[List of sources cited in the thesis 3](#_Toc25)

[Appendices 3](#_Toc26)

[Additional technical details and documentation 3](#_Toc27)

# Introduction

## Background and motivation

Elettra è una ente di ricerca internazionale che fornisce due forme di sorgenti di radiazioni (light source): il sincrotrone Elettra ed il laser a elettroni liberi fermi.

https://www.elettra.eu/

Essendo un grande ente di ricerca si sviluppano applicazioni che posso rendere migliore lo svolgimento delle attività all’interno dell’ente. Uno di questi è con lo sviluppo esponenziale della robotica negli ultimi anni e lo sviluppo e l’utilizzo della stessa. In questo ambito ci sono diversi robot con diversi produttori per diversi ambiti, al trasporto di pezzi all’interazione con le persone.

In tutto questo hanno sviluppato, attraverso il Robotics and Remotization initiative, un planner di navigazione per ROS1 ottimizzato per la navigazione in ambienti industriali grandi e complessi come quello di elettra.

<https://doi.org/10.1007/978-3-031-45770-8_29>

E dato che hanno un ulteriore “mobile robot” della rover robotics che desiderano controllare in ros2[[1]](#footnote-2)

https://roverrobotics.com/

Inoltre data la grandezza dell’azienda e dalla moltitudine di robot differenti presenti si è recente sviluppato un sistema di gestione di robot multipiattaforma “mutli fleet” chiamata OPEN-RMF che sarebbe utile implementare nella rete.

<https://www.open-rmf.org/>

## Research questions and objectives

## Scope and limitations

Lo scopo di questa tesi è aggiornare il sistema di planning sviluppato portandolo da ROS1 a ROS2 utilizzando la struttura rinnovata di NAV2 rispetto a NAV1, applicarlo al nostro robot MINI della rover robotics ed inserirlo dentro l’ambiente OPEN-RMF che accoglierà in futuro tutte le altre famiglie di robot presenti in azienda.

Le uniche limitazioni che al momento non abbiamo un controllo diretto agli ascensori e non permettono di testare tutte le funzionalità offerte da OPEN\_RMF.

# Literature Review

Adesso descriveremo l’insieme delle tecnologie usate

## Overview of ROS1 and ROS2

Il Robot Operating System[[2]](#footnote-3) (ROS) è un insieme di librerie e strumenti open-source che aiutano a sviluppare applicazione in ambito robotico. Diversamente dal nome non è un sistema operativo ma un software development kit (SDK) che ti fornisce tutti gli strumenti per creare il tuo robot fornendo alcune funzionalità tipiche di un OS: hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.

I principali strumenti che forniscono è:

un sistema di comunicazione message-passing system chiamato “middleware” or “plumbing”. Che permette la comunicazione tra i “nodi” del sistema con un anonymous publish/subscribe pattern permettendo fault isolation, separation of concerns, and clear interfacesfault isolation, separation of concerns, and clear interfaces ed altro.

Sistemi di controllo del software come launch, introspection, debugging, visualization, plotting, logging, and playback. Che migliorano gli strumenti per lo sviluppo. provides tools and libraries for obtaining, building, writing, and running code across multiple computers

rqt is a graphical user interface (GUI) tool for ROS 2

rqt\_console (log visualizer)

**Launch** files allow you to start up and configure a number of executables containing ROS 2 nodes simultaneously

Several advanced robot **simulators** can be used with ROS 2, such as Gazebo, Webots, etc.

The sros2 package provides the tools and instructions to use ROS 2 on top of DDS-Security. The security features have been tested across platforms (Linux, macOS, and Windows) as well as across different languages (C++ and Python). The SROS2 has been designed to work with any secure middleware, although not all middleware is open source and support varies depending on the ROS distribution in use.

The sros2 package can be used to create keys, certificates and policies necessary to enable ROS 2 security. However, the security configuration is extremely flexible

ros2 bag is a command line tool for recording data published on topics and services in your ROS 2 system. It accumulates the data passed on any number of topics and services, then saves it in a database. You can then replay the data to reproduce the results of your tests and experiments.

rosdep is a dependency management utility that can work with packages and external libraries. It is a command-line utility for identifying and installing dependencies to build or install a packageis a meta-package manager that uses its own knowledge of the system and the dependencies to find the appropriate package to install on a particular platform

RViz is a 3D visualizer for the Robot Operating System (ROS) framework.

ros2 node list

ros2 topic list

ros2 service list

ros2 action list

Una communità attiva e diffusa con al centro la Open Robotics[[3]](#footnote-4) che mantiene il codice e hosta le release

La struttura tipica di ROS è la seguente:

ROS runtime "graph" is a peer-to-peer network of processes (potentially distributed across machines) that are loosely coupled using the ROS communication infrastructure.

The ROS graph is a network of ROS 2 elements processing data together at the same time. It encompasses all executables and the connections between them if you were to map them all out and visualize them.

Middleware : synchronous RPC-style communication over services, asynchronous streaming of data over topics, and storage of data on a Parameter Server

a distributed framework of processes (aka Nodes) that enables executables to be individually designed and loosely coupled at runtime

I nodi che rappresentano parti del sistema e che permettono di essere isolate in modo da essere intercambiabili. permettono di isolare la complessità di un certo dispositivo o procedura partendo già da cose già fatte e rese pubbliche e unificati i messaggi che permettono di applicare diversi

Però il progetto, nato per uso accademico e data l’enorme crescita che ha avuto, ha trovato nuovi ambienti di utilizzo.

Descrizione delle parti di ROS2

La documentazione[[4]](#footnote-5) fornisce una descrizione esaustiva di tutti gli strumenti base offerti da ROS2, alcune funzionalità qui descritte sono solo disponibili in ROS2 in quanto ROS1 ha raggiunto il fine vita

The **ROS graph** is a network of ROS 2 elements processing data together at the same time. It encompasses all executables and the connections between them if you were to map them all out and visualize them.

**ROS node** should be responsible for a single, modular purpose, Each node can send and receive data from other nodes via topics, services, actions, or parameters.

**Topics** are a vital element of the ROS graph that act as a bus for nodes to exchange messages.

**Services** are based on a call-and-response model versus the publisher-subscriber model of topics. While topics allow nodes to subscribe to data streams and get continual updates, services only provide data when they are specifically called by a client.

A **parameter** is a configuration value of a node

**Actions** are one of the communication types in ROS 2 and are intended for long running tasks. They consist of three parts: a goal, feedback, and a result.

Actions are built on topics and services. Their functionality is similar to services, except actions can be canceled. They also provide steady feedback, as opposed to services which return a single response.

Actions use a client-server model, similar to the publisher-subscriber model . An “action client” node sends a goal to an “action server” node that acknowledges the goal and returns a stream of feedback and a result.

ROS adoption in domains beyond the mostly academic research community that was our initial focus. ROS-based products are coming to market, including manufacturing robots, agricultural robots, commercial cleaning robots, and others.

## **New use cases**

Of specific interest to us for the ongoing and future growth of the ROS community are the following use cases, which we did not have in mind at the beginning of the project:

* Teams of multiple robots: while it is possible to build multi-robot systems using ROS today, there is no standard approach, and they are all somewhat of a hack on top of the single-master structure of ROS.
* Small embedded platforms: we want small computers, including “bare-metal” micro controllers, to be first-class participants in the ROS environment, instead of being segregated from ROS by a device driver.
* Real-time systems: we want to support real-time control directly in ROS, including inter-process and inter-machine communication (assuming appropriate operating system and/or hardware support).
* Non-ideal networks: we want ROS to behave as well as is possible when network connectivity degrades due to loss and/or delay, from poor-quality WiFi to ground-to-space communication links.
* Production environments: while it is vital that ROS continue to be the platform of choice in the research lab, we want to ensure that ROS-based lab prototypes can evolve into ROS-based products suitable for use in real-world applications.
* Prescribed patterns for building and structuring systems: while we will maintain the underlying flexibility that is the hallmark of ROS, we want to provide clear patterns and supporting tools for features such as life cycle management and static configurations for deployment.

Ulteriori informazioni possono essere ottenute da le note di sviluppo[[5]](#footnote-6).

anonymous publish-subscribe middleware system that is built almost entirely from scratch -> It is now possible to build a ROS-like middleware system using off-the-shelf open source libraries. We can benefit tremendously from this approach in many ways

with ROS 2, we will design new APIs, incorporating to the best of our ability the collective experience of the community with the first-generation APIs

I principali miglioramenti implementati sono:

A livello di API il sistema dispone di molti più layer. Alla base c’è la libreria RCL (Library to support implementation of language specific ROS Client Libraries.) da cui si sviluppano le librerie client RCLCPP per cpp, RCLPY per python e molti alti. The ROS client library common implementation. This package contains an API which builds on the ROS middleware API and is optionally built upon by the other ROS client libraries.

<https://index.ros.org/p/rcl/>

A livello di node sviluppato ROS1 non aveva una struttura predefinita su come realizzare il node, adesso è una classe che eredita dall’oggetto Node che dispone di tutte le funzionalità dei ROS2

<https://docs.ros2.org/foxy/api/rclcpp/classrclcpp_1_1Node.html>

<https://docs.ros2.org/foxy/api/rclpy/api/node.html>

Intercommunicazione tra nodi nello stesso executables per migliorare le prestazioni su HW limitati che bypassa the ROS2 communication overhead che era implementata come Nodelets adesso sono components in ROS2

ROS 1 nodes are compiled into executables. ROS 1 nodelets on the other hand are compiled into a shared library which is then loaded at runtime by a container process

This makes it easy to add common concepts to existing code, like a life cycle. use the same API.

By making the process layout a deploy-time decision the user can choose between:

* running multiple nodes in separate processes with the benefits of process/fault isolation as well as easier debugging of individual nodes and
* running multiple nodes in a single process with the lower overhead and optionally more efficient communication (see Intra Process Communication).

Since a component is only built into a shared library, it doesn’t have a main function (see Talker source code). A component is commonly a subclass of rclcpp::Node. Since it is not in control of the thread, it shouldn’t perform any long running or blocking tasks in its constructor. Instead, it can use timers to get periodic notifications. Additionally, it can create publishers, subscriptions, servers, and clients.

https://docs.ros.org/en/jazzy/Concepts/Intermediate/About-Composition.html#using-components

Adding Lifecycle

a managed node presents a known interface, executes according to a known life cycle state machine, and otherwise can be considered a black box. This allows freedom to the node developer on how they provide the managed life cycle functionality, while also ensuring that any tools created for managing nodes can work with any compliant node.

<https://design.ros2.org/articles/node_lifecycle.html>

## Introduction to Navigation Stack and Google Cartographer

## Review of related work on porting navigation stack from ROS1 to ROS2

## Overview of open source fleet management systems, particularly Open-RMF

# Methodology

## Description of research platform and hardware used

## Steps taken to port the navigation stack from ROS1 to ROS2

## Hardware adaptation process from Rover Zero to Rover Mini

## Integration of the navigation stack with Open-RMF

## Testing and evaluation of the system

# Results and Analysis

## Presentation of results from testing and evaluation

## Comparison of system performance before and after the porting process

## Discussion on the challenges encountered and how they were addressed

# Conclusion and Future Work

## Summary of the research findings

## Contributions to the field of electrical engineering and robotics

## Recommendations for future work and improvements

# References [[references]]

## List of sources cited in the thesis

# Appendices

## Additional technical details and documentation

1. S. Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, “Robot Operating System 2: Design, architecture, and uses in the wild,” Science Robotics vol. 7, May 2022.

   @article{

   doi:10.1126/scirobotics.abm6074,

   author = {Steven Macenski and Tully Foote and Brian Gerkey and Chris Lalancette and William Woodall },

   title = {Robot Operating System 2: Design, architecture, and uses in the wild},

   journal = {Science Robotics},

   volume = {7},

   number = {66},

   pages = {eabm6074},

   year = {2022},

   doi = {10.1126/scirobotics.abm6074},

   URL = {https://www.science.org/doi/abs/10.1126/scirobotics.abm6074}

   } [↑](#footnote-ref-2)
2. <https://www.ros.org/> [↑](#footnote-ref-3)
3. <https://www.openrobotics.org/> [↑](#footnote-ref-4)
4. <https://docs.ros.org/en/rolling/index.html> [↑](#footnote-ref-5)
5. <https://design.ros2.org/> [↑](#footnote-ref-6)