XploreR

Autonomous Exploration with Virtual Reality Visualization

Artemis Georgopoulo

Fabiano Junior Maia Manschein

Shani Israelov

Yasmina Feriel Djelil

Foreword

The XploreR project is the result of eight weeks of work in cooperation with the Karelics company for the *Robotics and XR* class in University of Eastern Finland, lectured by Professor Ilkka Jormanainen. The aim of this project was to integrate an autonomous exploration algorithm in a SLAM-equipped robot and link it to a Virtual Reality (VR) environment, allowing visualization of the robot's published data (e.g., the map, pose, path, etc.).

The choice of this project subject was linked to our previous interest in the SLAM (Simultaneous Localization and Mapping) algorithm. Also, as IMLEX students we are deeply interested in XR-related subjects. As such, we decided to enhance autonomous navigation with VR visualization and interaction.

As all the group members came from diverse backgrounds, dividing the tasks and making sure that all the members have a good understanding of the project content was quite challenging. Thankfully, this could be solved with the help of our colleagues at Karelics. As most of the technologies used in this project were new to us, they helped us in understanding both the concepts and how the distinct parts in the robot and ROS (Robotics Operating System) side worked.

We would like to sincerely express our appreciation to **Karelics** for the support they've sent our way working on this project.

Summary

Introduction

The goal of this project is to combine autonomous exploration algorithms implemented in a mobile robot with a virtual reality environment for visualizing collected data. On the physical side, the SAMPO2 robot is outfitted with the Karelics Brain software and all the necessary hardware. The SLAM algorithm is already implemented and ready-for-use. While mapping the environment, the robot will move around autonomously according to goal poses set by the exploration algorithm. The user will be able to view the robot sensor data and the 2D map generated by the SLAM algorithm in the digital side, I.e., the Unity scene.

The project's potential use cases include exploring unknown environments with the robot while a human user remotely monitors and possibly controls the robot and its navigation through the unknown. Natural disaster areas, for example, where potential victims are found and humans cannot traverse, are an example of an unknown environment. Another example is space exploration, in which the robot explores new planets or astral bodies.

SAMPO2 and the Karelics Brain software were provided by Karelics Ltd., a Finnish robotics software development company created in 2019, focused on robotics solutions for the construction industry. They claim to be the key link between robot manufacturers and construction companies, with their software making robots smart and easy-to-use. This is achieved by programming and updating robots from different manufactures, and enabling them to work together without humans, using the same data and user interface.

Target and minimum requirement

The project is considered successful if it achieves at least the following requirements:

- Autonomous exploration with automatically generated goal poses.
- Map data communication to the VR environment, and visualization of the map in 2D.
- Integration with the SAMPO2 robot.

The following is a list of extra features that were brainstormed for the project:

- *Dockerization* of all implementations (project runs inside Docker containers, except for Unity).
- Multi-algorithm autonomous exploration.
- Visualization of the map and costmap in 3D (i.e., upwards projection of the 2D map, or use of point-clouds).
- Robot ghost path possibility (previously traversed paths).
- Multi-robot autonomous exploration.
- Camera feed view in the VR scene.
- User interaction in the VR scene: switch between autonomous mode and manual control.

Technologies

Hardware:

- SAMPO2 URobot equipped with NVIDIA Jetson.
- Samsung Odyssey VR headset (WMR) and controllers.
- Computers with enough power to run the Docker containers and the VR scene.

Software:

- ROS2 galactic.
- Docker and Docker Compose.
- Unity version 2021.3.14f1 (LTS).

Technical description

Overview

The project can be separated into two main environments: simulation and *real-robot.* Figure 1 illustrates the different environments, with the *real-robot* being specifically defined as SEMPO2 URobot for our project.

The simulation environment has all the Docker containers up:

- rostcp: communication with Unity.
- gazebo: Turtlebot3 Gazebo simulation and Rviz instance.
- explore: autonomous exploration algorithm.

For use with a real robot, the *gazebo* container is not necessary and should be disabled to avoid data overlap. On both use-cases, the Unity scene is fitted with the ROS-TCP Connector plugin to communicate with ROS2, and a VR Scene for visualization of the data with a VR Headset.

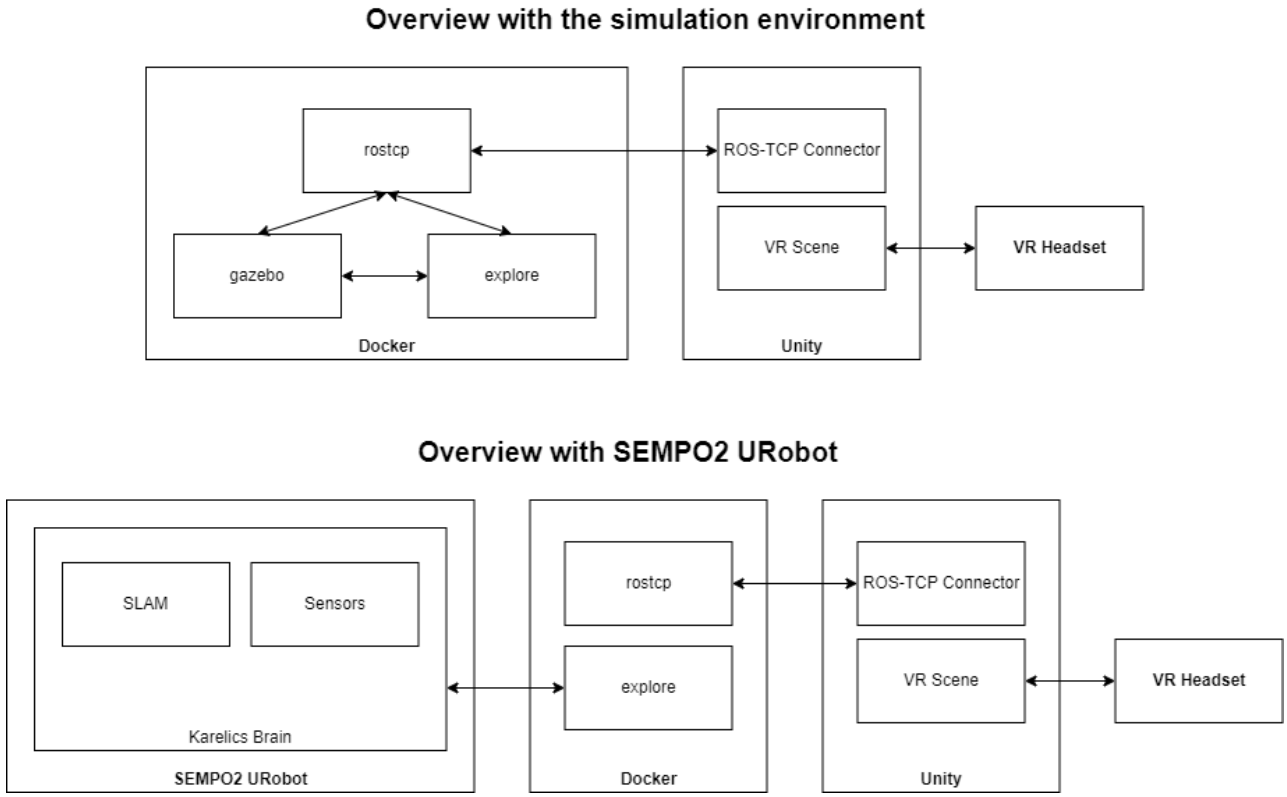

Figure 1: Project overview.

Docker

Docker is an open platform that enables users to create and run applications called images within isolated containers. It is possible to run multiple containers concurrently. It is also possible for users to share containers with the same content.

A Dockerfile was created to install the ROS2 Galactic distro and all necessary packages for the project. This allowed all team members to work in the same environment and easily share progress and new packages. Furthermore, a docker compose file was created to launch multiple containers simultaneously, each running its own ROS2 nodes:

- explore: autonomous exploration node responsible for receiving the robot costmap, running the frontier identification algorithm, and sending new goal poses to the robot.
- rostcp: node containing an endpoint to send and receive ROS messages from a Unity scene.
- gazebo: bringup container that launches nodes required to run a gazebo simulation and a Rviz instance.

Additionally, some quality-of-life features were implemented:

- Automatic sourcing of ROS2 on new terminals inside the containers;
- automatic fix for the line endings issue that often happens on Windows;
- easy cloning and *colcon* building of new ROS2 packages by adding a line in the Dockerfile.

It's also worth noting that getting GUI interfaces to work from inside docker containers was a challenging task. Although the current implementation makes it easy for UNIX system users, Windows users must go through an annoying process on every reboot (and when the IP changes) for it to work.

The usage of Docker and setting up the containers in the compose file shows the modularity of the project. New containers can be easily added to the compose file, existing ones can be modified or accessed, and the Dockerfile can be changed to include new packages. This also enables the user to launch only the containers necessary for the use-case. For example, to test the autonomous exploration on a real robot, only the *explore* container is needed. As such, the *rostcp* and *gazebo* services can be commented out in the compose file.

SEMPO 2 URobot

The URobot company created SEMPO 2 to help both in the construction industry and domestic tasks. The robot already has all the necessary sensors (GPS, gyroscope, etc.) and components to allow navigation.

Karelics outfitted the robot with the Karelics brain software, which turns the robot into an autonomous, self-sustaining, and easy-to-manage team member that can perform difficult or dangerous tasks without human risks. By integrating this technology into the robot, it will quickly and effortlessly integrate functionality, communication H2R and R2R, riding elevators and data flow, and enable quicker on-site setup.

Communication with the robot is done through the network by connecting to its Wi-Fi router. This gives the docker containers access to all its ROS2 topics, services, and actions.

Figure 2: Kurelics' SEMPO 2 URobot wearing a VR headset.

Gazebo and TurtleBot3

Gazebo is a simulator that allows testing algorithms, designing robots, and simulating tasks in complex environments. It can accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. As Gazebo is a stand-alone application which can be used independently of ROS2, the integration of Gazebo with ROS2 is done through a set of packages called *gazebo_ros_pkgs*. These packages provide a bridge between Gazebo's C++ API and transport system, and ROS 2 messages and services.

In this project, Gazebo was implemented in a docker container and used as the simulation environment to test and verify the autonomous exploration algorithm. It comes with two pre-installed useful maps: *turtlebot3_world* and *turtlebot3_house*. It also comes with different turtlebot3 models. For testing and verification, both maps were used with the Burger and Waffle models.

Future improvements or new autonomous navigation algorithms can be tested and evaluated in this environment.

Robot Navigation

Robots navigate in a similar fashion to humans. To move from one point to another:

- The robot needs to know the place (**Mapping**)

- Then it needs to know where it is in the environment (**Localization**)
- Next, the robot needs to plan how to move between two points (**Path Planning**)
- Finally, it needs to send messages to the wheels to make the robot follow the path while avoiding obstacles (**Robot Control** and **Obstacle Avoidance**)

In robotics, maps are mainly acquired to enable autonomous navigation [1]. However, mapping using robots can also be useful when it comes to exploring and mapping dangerous environments or places that are inaccessible to human beings. Hence, autonomous exploration strategies were developed, which enabled the robot to find the next location to explore to improve its map and localization estimates.

The integration of a SLAM algorithm, an exploration strategy and a path planning/navigation algorithm allows the robot to autonomously explore and map an unknown environment using its sensory data.

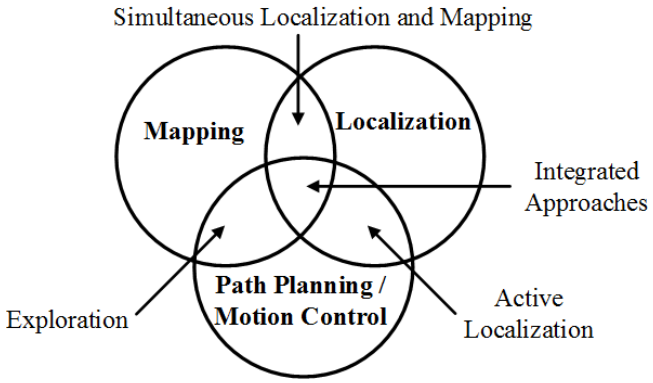

Figure 3: Illustration of the integrated approaches, adapted from (Makarenko et al., 2002).

Autonomous exploration

Exploration strategies place an emphasis on autonomously mapping as much of an unknown environment as possible, and in the shortest time possible.

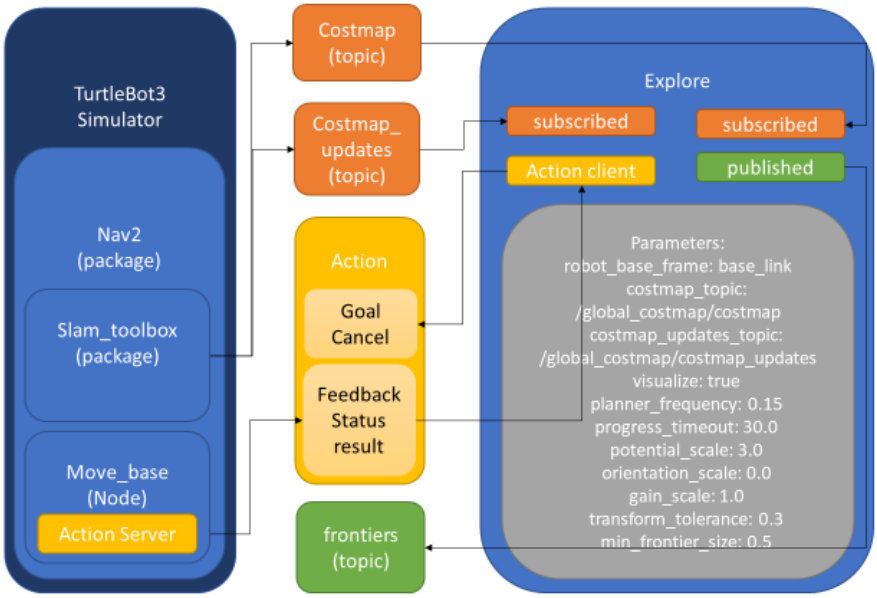

Figure 4: exploration and turtulebot3 communication in ROS2.

Occupancy Grid Map - this map is segmented into cells, where each cell is assigned with a probability of occupancy:

- probability of zero = free from obstacles = OPEN
- probability of one = OCCUPIED
- probability of 0.5 = no information about the occupancy of the cell/not yet explored = UNKNOWN

Nearest Frontier approach - popular and simple exploration strategy. This algorithm simply analyses an Occupancy Grid Map and detects all potential borders (called candidate frontiers) between the cells marked as OPEN and UNKNOWN. The distance between the current robot poses and each frontier is analyzed, and the frontier closest to the robot is chosen as the next location to explore.

Next Best View (NBV) approach - the evaluation criteria of the candidate destinations, try to strike a balance between the amount of unknown area that the robot can explore from that location, and the distance that the robot must travel to arrive there.

For both exploration strategies, the potential candidates for exploration appear using frontier detection. The algorithm traverses the map data and creates a vector of frontier cells. These frontier cells are then grouped into frontiers according to their adjacency. Each frontier is then stored in a two-dimensional vector used by the exploration algorithm for further analysis. The exploration process ends when the frontier detection algorithm can no longer detect frontiers that are at least ten cells long.

Both strategies are implemented in ROS packages, but none support ROS2. In this project, the Nearest-Frontier approach was implemented using code from the m-explore-ros2 repository, a port of the popular explore-lite ROS package.

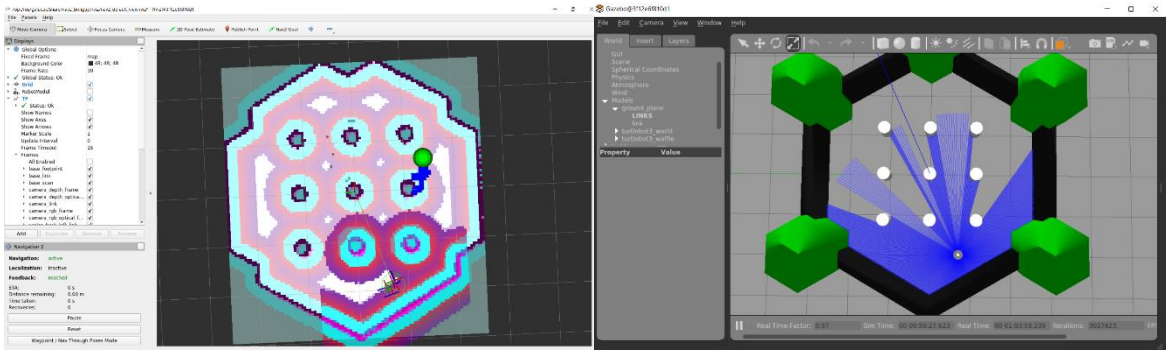

Figure 5: on the left TurtleBot3 in Rviz, and on the right TurtleBot3 in Gazebo.

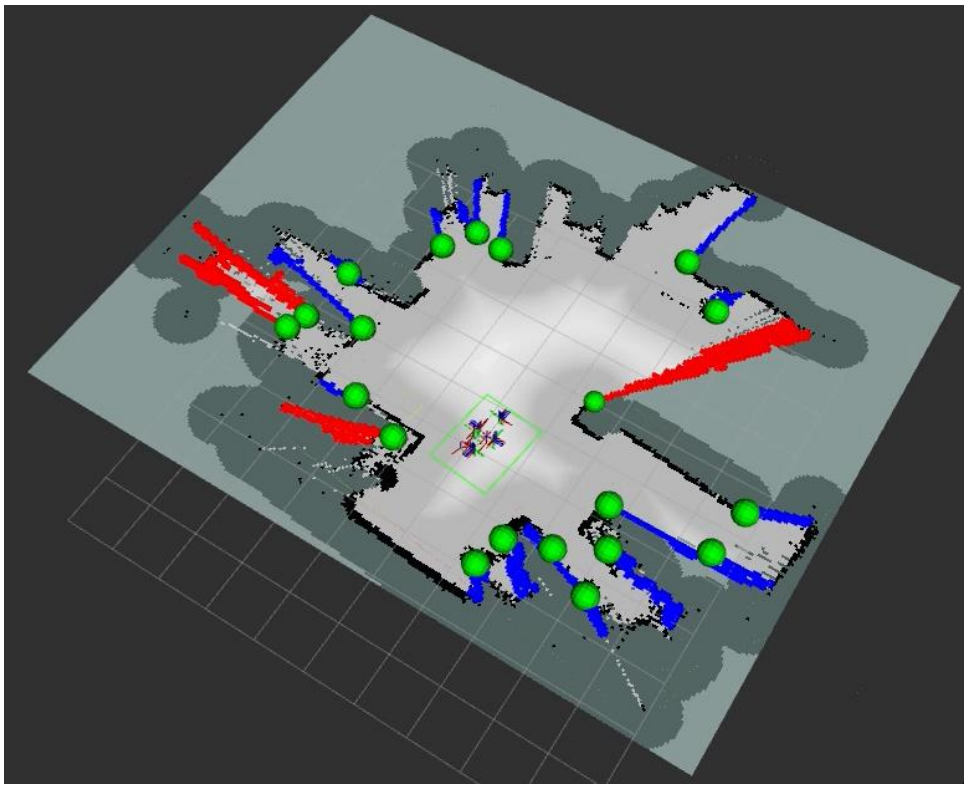

Figure 6: Rviz view of the SAMPO2 robot exploring the Karelics office. Green spheres are possible navigation goals generated by the *explore* node. Blue and red lines are accessible and inaccessible frontiers, respectively.

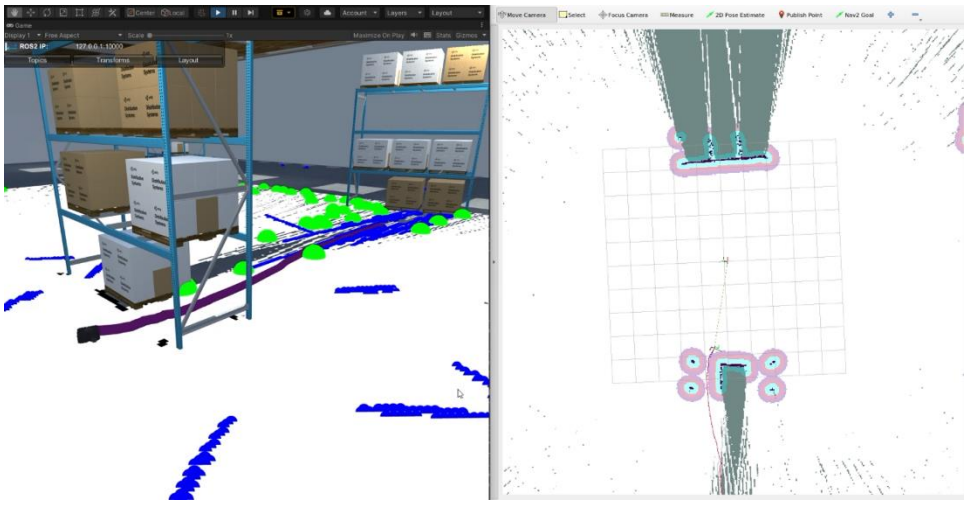

Figure 7: On the left, Nav2-SLAM-Example in Unity simulating a Turtlebot3. Autonomous exploration goals generated by the *explore* node. On the right, Rviz visualization of the map.

Tuning Parameters

The parameters on table [1] can be tuned as part of the exploration task. For example, the minimum frontier size can be changed according to the simulator world dimensions or real-life room dimensions.

parameter	type	description
costmap_topic	(string, default: costmap)	Specifies topic of source nav_msgs/OccupancyGrid. Mandatory.
costmap_updates_topic	(string, default: costmap_updates)	Specifies topic of source map_msgs/OccupancyGridUpdate. Not necessary if source of map is always publishing full updates, i.e. does not provide this topic.
gain_scale	(double, default: 1.0)	Used for weighting frontiers. This multiplicative parameter affects frontier gain component of the frontier weight (frontier size).
min_frontier_size	(double, default: 0.5)	Minimum size of the frontier to consider the frontier as the exploration goal. In meters.
orientation_scale	(double, default: 0)	Used for weighting frontiers. This multiplicative parameter affects frontier orientation component of the frontier weight. This parameter currently does nothing and is provided solely for forward compatibility.
planner_frequency	(double, default: 1.0)	Rate in Hz at which new frontiers will be computed and goal reconsidered.
potential_scale	(double, default: 1e-3)	Used for weighting frontiers. This multiplicative parameter affects frontier potential component of the frontier weight (distance to frontier).
progress_timeout	(double, default: 30.0)	Time in seconds. When the robot does not make any progress for progress_timeout, the current goal will be abandoned.
return_to_init		
robot_base_frame	(string, default: base_link)	The name of the base frame of the robot. This is used for determining robot position on map. Mandatory.
transform_tolerance	(double, default: 0.3)	Transform tolerance to use when transforming robot pose.
use_sim_time		
visualize	(bool, default: false)	Specifies whether to publish visualized frontiers.

Table 1: parameters for tuning exploration task.

Unity and VR visualization

Unity is a cross-platform game engine development platform. It supports desktop, mobile, and virtual reality games. The benefit of using Unity is that it is beginner friendly and already sustains a variety of features.

In this project, Unity was used to visualize the mapping process as the robots moved. Using the ROS-TCP Connector plugin to connect Unity to our ROS2 environment allows Unity to interact with ROS2 nodes and thus have the necessary data to make the visualization possible.

The Microsoft Mixed Reality Headset was connected to Unity through SteamVR using the OpenXR and the OpenVR plugins. The scene consists of a floor where the map is displayed (Figure 8). The player can use controllers to move around the scene.

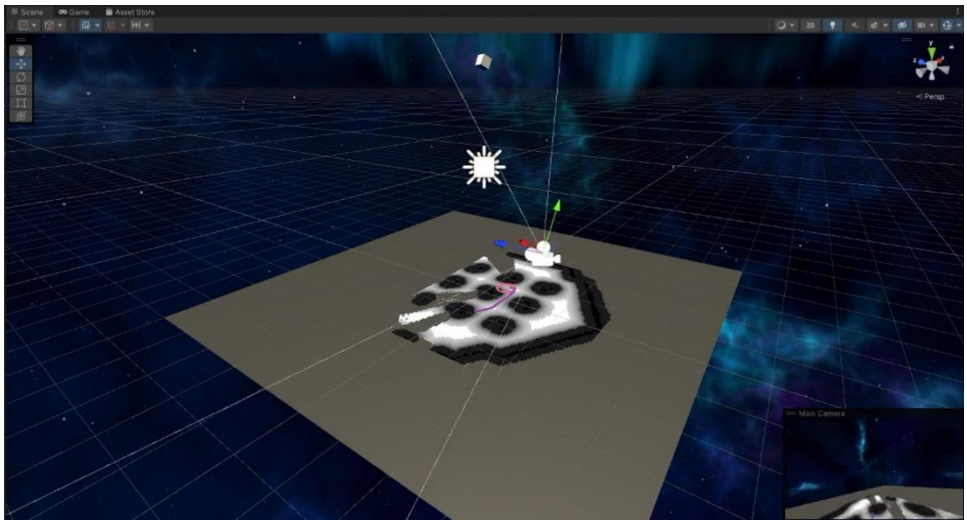

Figure 8: VR scene in Unity. Main camera shows what is being seen in the VR headset. On the floor, the costmap from *turtlebot3_world* of the Turtlebot3 Gazebo simulation can be seen. The pink circle if the robot footprint (its location), and the purple line is its planned path.

Timeline

The following is a simplified timeline of tasks executed in the project:

- Creating a working environment in Docker.
- Self-learning navigation concepts – SLAM and path-planning. For SLAM we used [2].
- Research autonomous exploration algorithms – we used [1].
- Self-learning Ros2 – we used ros2 galactic tutorials that are a collection of step-by-step instructions meant to steadily build skills in ROS 2.
- Self-learning VR in Unity – we used YouTube video playlist "How to make a VR game in Unity".
- Running TurtleBot3 simulation with Gazebo – we used nav2 project instructions in order to navigate a simulated Turtlebot3 in the Gazebo simulator.
- Integration of autonomous exploration nearest frontier strategy – we used m-explore-ros2 git repository. Since the documentation is short, we learnt from the node it based on, explore_lite.
- Integration with real robot SAMPO2.
- Full test runs: in the simulation environments; in the Karelics office with SAMPO2; in the Carelia hall with SAMPO2.

Repository and Videos

More details about our project can be found in our GitHub repository: git repository.

You can watch a short video describing the XploreR Project here: XploreR video.

A teaser video is also available in the following link: XploreR Teaser.

Issues and solutions

While working on the XploreR, our group met various issues with different aspects of the project. As described below:

On the Docker/ROS2 side:

- Docker implementation for different OS was quite challenging, as OS-specific configuration was required for certain parts (e.g., allowing GUI interfaces from inside Docker containers).
- When using Docker, the actual files are in the container and if we want to change the config parameters for the simulator launch, we don't have access to the config file. To solve it, we forked the git repository and changed the parameter there. This was essential because parameters like *minimum frontier size* are tuned from the config file and are essential to change when working with the robot in size-changing rooms.
- When working on Windows, Gazebo and Rviz take a long time to launch, especially for the first time.
- In the Turtlebot3 Gazebo simulation, the autonomous exploration works well, but not in the house world (Figure 8). It seems like the robot is unable to map its surroundings properly and often gets stuck inside rooms. Without proper mapping, the occupancy grid and the costmap are not built in a way that the autonomous exploration algorithm can detect unknown areas as frontier and possible goals. We considered this a problem with the world itself.

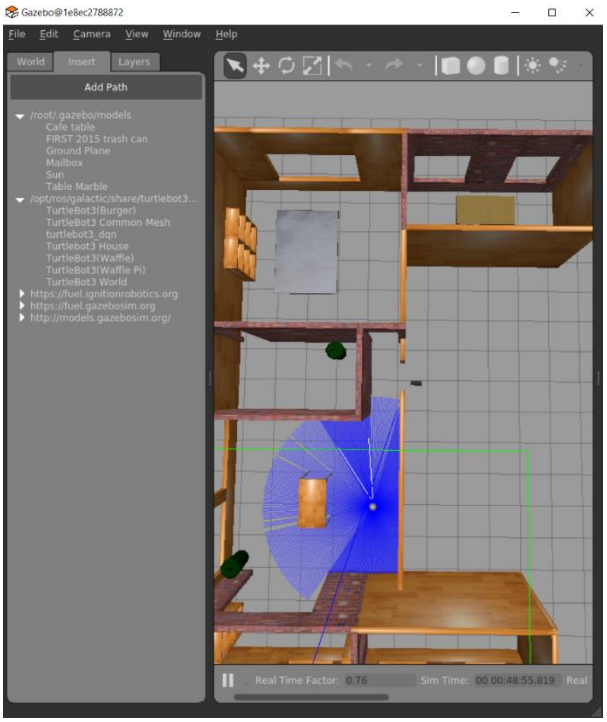

Figure 9: Turtlebot3 Gazebo simulation with the *house* world.

On the autonomous implementation side:

- We planned to integrate two algorithms for autonomous exploration, the nearest frontier and the best-next-view, and tune these to find the best performance and compare between both algorithms to develop our own, but we ran into issues:
    - Best-next-view is not implemented on ROS2-galactic distribution. The migration from ROS1 to ROS2 is not trivial and demands a lot of effort.

    - Nearest-frontier is also not implemented on ROS2-galactic distribution, but we found a GitHub repository port and we used it and made efforts to make it work on our project.
- Due to time limitations, we didn't do the planned comparison and didn't implement our own version of the algorithms.

On the VR Environnement side:

- We spent a long time trying to create the VR environment in Unreal Engine. After a lengthy implementation, we discovered that ROS2 cannot be linked to Unreal Engine via the ROS-Bridge package because the plugins are incompatible (TCP connection not supported in ROS2). As a result, we had to start over with Unity.
- Making the scene in Unity was challenging due to the following issues:
    - We had to wait a long time before getting the headset, and our testing phase began far too late. We only had about a week to work on the VR scene. Also, most of the headsets available in the lab were incompatible with our computers (they needed a Display Port, and we only had computers with HDMI). The only possibility left was the Samsung Odyssey, a Microsoft Mixed Reality headset.
    - Lack of materials (e.g., batteries for the headset controllers).
    - It took some time and was frustrating to figure out the communication between Unity and Microsoft Mixed Reality. The headset appeared to be refusing to interact with Unity (implementing movement and animations). To make it work, we had to use an outdated SteamVR plugin to link both. In addition, the VR headset appeared unstable because we could see the camera moving in an unsteady manner in the scene.
    - Unity and Visual Studio synchronization issues. Standard variables were not recognized in Visual Studio. To make the script work, we had to go through a lot of advanced settings and additional package downloads.
- The Unity plugin used to visualize ROS2 topic data lacked an essential configuration: quality of service of a topic. This issue forced us to change parameters in the robot and made it impossible to see the laser scan from the Gazebo simulation. Furthermore, the laser scan from SAMPO2 didn't align with the map (likely an issue with the transforms from */tf*). More details in the following GitHub issues: in the ROS-TCP-Connector repo and in the XploreR repo.

Self-evaluation

We've set the project goals above and beyond the project description, and we are proud of the process and results.

As a group, we ensured each member worked optimally and helped each other whenever possible, contributing to our collective goals. We enjoyed working closely and we communicated effectively.

We want to continue developing our skills in XR, Unity and ROS2. We can see that we made progress in these skills and would like to double down on the growth.

Karelics team was pleasant and transparent. We appreciate their openness and direct communication. We knew what was expected from us and how well we met these expectations.

As such, we evaluate our project with a grade of 5/5.

Conclusion

In this project, we combined an autonomous exploration algorithm implemented in a mobile robot with a virtual reality environment for visualizing collected data. For the simulation environment, we used Turtlebot3, and in real-life

Karelics's SAMPO2 robot. While mapping the environment, the robot moves around autonomously according to the Nearest-Frontier algorithm. In addition, we implemented communication to the VR environment, and visualization of the ROS2 topics (map, robot position, path, etc.). This project included setting up docker containers, working with ROS2 packages, development in Unity, a VR environment, and integrating all the different pieces to achieve our goals.

Future work

Some suggestions for future works include:

- Change the VR scene to a 3D map view, allowing the user to see the map in more detail. Both the 2D map and the 3D map should be interactable, where user interaction (tap, touch) is used to switch from autonomous exploration to manual control, allowing the user to set the navigation target point.
- Extended autonomous exploration strategy implementation, in precise, Next-Best-View exploration, and comparison between the strategies.
- New Gazebo worlds for testing autonomous exploration (e.g., a labyrinth).
- Remaking the VR scene with up-to-date XR plugins.

We'd like to point-out that this project can be developed in 3 big points:

- New autonomous exploration algorithms in the *explore* node.
- New Gazebo worlds for the *gazebo* node.
- New or improved scenes in Unity, both PC view and VR view.

Of course, it is also possible to improve the ROS2-Unity communication by contributing (or forking) the repository with the plugin we used (ROS-TCP-Connector).

Bibliography

[1] Darmanin, R. and Bugeja, M., "Autonomous Exploration and Mapping using a Mobile Robot Running ROS", In Proceedings of the 13th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2016), Volume 2, pages 208-215

[2] Durrant-Whyte, H. and Bailey, T., "Tutorial: Simultaneous Localization and Mapping: Part I", IEEE Robotics & Automation Magazine, June 2006