

**Digital, autonomous, Intelligent and Synchronous system for
Continuous identification, Optimization and Value Extraction of
Resources from the end-of-use built environment**



DISCOVER

D1.2 Navi-Wall

**WP 1. Autonomous non-invasive scanning and
identification system (M1 – M18)**

WP Leader: UPC

Submission date: 31 July, 2025



**Funded by
the European Union**

The DISCOVER (GA 101129909) project is funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Health and Digital Executive Agency (HADEA). Neither the European Union nor the granting authority can be held responsible for them.

Project name	Digital, autonomous, Intelligent and Synchronous system for Continuous identification, Optimization and Value Extraction of Resources from the end-of-use built environment
Grant	101129909
Funding scheme	Horizon Europe
Project Acronym	DISCOVER
Project starting date	01/06/2024
Project duration	48 months
Deliverable number	1.2
Deliverable title	Navi-Wall
Deliverable version	V1
Work Package number	1
Work Package title	Autonomous non-invasive scanning and identification system
Due date of delivery	31/07/2025
Actual date of delivery	31/07/2025
Dissemination level	PU - Public
Type	OTHER
Editor(s)	Alba Perez Gracia (UPC), Gvantsa Jichoshvili (UPC)
Contributor(s)	Alba Perez Gracia (UPC), Lluís Bonet Ortuño (UPC), Muhammad Zain Bashir (UPC), Yeray Navarro Soler (UPC), David Caballero Flores (UPC), Artem Kushniryk Herasym (UPC), Albert Lopez (UPC), Vega Perez Gracia (UPC), Rahul Tomar (DTT), Manuel Jungmann (DTT), David Garcia Estevez (TECNALIA), Francesco Di Maio (TU Delft), Yongli Wu (TU Delft), Mohanad Abukmeil (TU Delft), Jorge de Brito (IST), João Gomes Ferreira (IST).
Reviewer(s)	Lies Mertens (TRACIMAT), João Pacheco (IST).
Rights	DISCOVER consortium

Confidentiality	
Does this report contain confidential information?	Yes <input type="checkbox"/> No <input checked="" type="checkbox"/>
Is the report restricted to a specific group?	Yes <input type="checkbox"/> No <input checked="" type="checkbox"/>

Document history

Version	Date	Beneficiary	Description
0.1	29.07.2025	UPC	Initial draft
0.2	30.07.2025	UPC, TRACIMAT, IST	Internal review by partners
1.0	31.07.2025	UPC	Final version submitted after partner input

List of abbreviations

Abbreviation	Meaning
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
FSM	Finite State Machine
GPR	Ground-Penetrating Radar
MRL	Mobile Robotics Laboratory
PLC	Programmable Logic Controller
RANSAC	Random Sample Consensus
RTAB-Map	Real-Time Appearance-Based Mapping
SLAM	Simultaneous Localization and Mapping
ToF	Time of Flight

List of figures

Figure 1. Oliwall robot design and workspace.....	10
Figure 2. Sensors' plate at the end effector of the robot.	11
Figure 3. Left, the two LiDARs used for navigation. Right, the dome LiDAR used to capture the geometry of the space.	11
Figure 4. AgroMOBY robot with UR10 navigating in the CDEI-UPC Mobile Robotics Laboratory (MRL) space.....	12
Figure 5. The GPR to be used in the project (From Screening Eagle).....	13
Figure 6. Testing laboratory space (CDEI-UPC MRL).	13
Figure 7. 3D virtual model of the CDEI-UPC MRL.	14
Figure 8. Simulation of the AgroMOBY robot in the CDEI-UPC MRL space.....	14
Figure 9. The laboratory world and the robot within the Gazebo environment.....	15
Figure 10. Reference frames for the robot and the space.....	16
Figure 11. Reference frames within the robot.	16
Figure 12. tf tree showing the relationship between the reference frames.....	17
Figure 13. Example of the point cloud to segmented geometry algorithm.....	18
Figure 14. AgroMOBY with the OSDome LiDAR.....	19
Figure 15. The 3D registered point cloud of the environment	19
Figure 16. Example of Dijkstra's algorithm [Wikipedia].....	21

Figure 17. Reinforcement learning strategy	22
Figure 18. Latent space for the percentage of underground humidity.....	23
Figure 19. Left, improvement on the water seen by the agent after the training. Right, an example of a path selected by the agent after the training.....	23
Figure 20. Example of robot finding the path with less water (blue and yellow background) while avoiding surface obstacles (white).	24
Figure 21. Left, the reachable workspace of the robot manipulator; Centre, a trajectory avoiding obstacles within the workspace; Right, the robot arm with a planned trajectory	25
Figure 22. Docking space for the specified poses on the wall. Green colour indicates docking points that can reach all specified wall poses.....	26
Figure 23. Location of range sensors on the sensor plate.	29
Figure 24. The schematics of the distance and parallelism from sensors to wall.	31
Figure 25. Above, scanning recommendations from Robo Scan. Below, the division in docking panels (left) and subdivision in scanning panels in order to create the arm trajectory	32
Figure 26. The mapping and scanning strategy.....	33
Figure 27. Finite-state machine for Navi-Wall.	35

Table of contents

About DISCOVER project.....	7
Abstract.....	8
1. Introduction.....	9
1.1. Work package 1.....	9
2. Deliverable 1.2: Navi-Wall.....	9
2.1. Contents of the report.....	9
3. Navi-Wall equipment and facilities.....	10
3.1. Oliwall	10
3.2. Current mobile manipulator	11
3.3. Oliwall sensors.....	12
3.4. Testing grounds.....	13
4. Sensing to geometry	15
4.1. The robot in the space (frame transformations)	15
4.2. Model from simulated LiDAR data	17
4.3. Data capture with OSDome LiDAR.....	18
5. Navi-Wall: robot navigation for non-invasive data acquisition	19
5.1. Navigation mapping and localization.....	20
5.2. Global and local path planning.....	20
5.3. Sensor fusion for navigation with combined surface and subsurface data	21
5.4. Reinforcement learning for embedded navigation.....	22
5.4.1. Reinforcement learning for embedded navigation: subsurface-only and surface–subsurface data fusion	23
5.4.2. Summary	24
5.5. The mobile manipulator: combined path planning and docking.....	24
5.5.1. Motion planning for the UR10 robot.....	24
5.5.2. Docking for the mobile manipulator.....	25
5.6. Oliwall’s control.....	26
5.6.1. Omnidirectional base control.....	26
5.6.2. Robotic arm control	27
5.6.3. Software controllers.....	27
5.7. Wall approximation and wall following	28
5.8. Plane discretization.....	31

5.9. Finite state machine.....	32
6. Summary of results and deliverables.....	35
References.....	38
APPENDICES.....	39
Appendix 1. Pseudocodes for mapping, navigation, and path planning	39

About the DISCOVER project

DISCOVER aims to develop an autonomous, synchronous, continuous and intelligent identification and data analysis system for materials and products in existing end-of-life built works. The proposed approach will provide key stakeholders, including academia research performers, along with construction industry representatives, with data-driven insights to make deconstruction more efficient, optimise the use of resources, improve the environmental footprints and enhance the circularity of construction and demolition, unlocking the potential of end-of-life built works, which will become material banks. The expected outcomes include an autonomous robotic platform coupled with continuous identification tools to scan built works and provide synchronous quantitative and qualitative data from different materials, including complex and concealed elements. Artificial intelligence algorithms will allow a rapid analysis of the properties and characteristics of components, and feed the automated scan-to-BIM model creation. The multi-dimensional BIM, including selective demolition processes, labour productivity, and technical planning, will become a Digital Twin of the demolition site optimised by social, economic, and environmental multi-criteria assessments. This approach will highly contribute to significantly increasing the supply of traceable and sustainable construction materials and products for enhancing their wider market acceptance, following the waste hierarchy. The social impacts of digital transformation in the construction sector will be considered, and also new professional development tools for the relevant stakeholders will be proposed. The system will be tested in 4 different real demolition sites (Spain, Portugal, Poland and Belgium), offering a complete range of built work typologies and wide geographical coverage to demonstrate the replicability potential of DISCOVER, increasing the project dissemination capacity and awareness among the construction sector.

Abstract

Deliverable D1.2 of the DISCOVER project (HORIZON-CL4-2023-TWIN-TRANSITION-01-11-101129909) presents the development and implementation of Navi-Wall, an open-source navigation software for autonomous mobile manipulators tasked with non-invasive data acquisition in buildings designated for demolition. Navi-Wall is a core component of Work Package 1 (WP1), which focuses on creating the *Oliwall* robot—a mobile manipulator equipped to scan and identify construction materials and hidden elements in walls, floors, ceilings, and columns using advanced sensors such as LiDAR, GPR, hyperspectral cameras, and RGB-D vision.

The deliverable describes the methodology and results associated with robot navigation and data collection strategies. It includes algorithms for 3D environment reconstruction, semantic mapping, global and local path planning, and wall-docking manoeuvres for high-precision scanning. The software architecture integrates ROS 2 control frameworks, custom planning strategies, and reinforcement learning methodologies to support future integration of surface and subsurface navigation. The approach is validated in both simulated and real test environments using an equivalent mobile robot platform.

The report documents the facilities, sensor setup, software components, and control strategies developed, highlighting the modular design and adaptability of the Navi-Wall system. The work lays the foundation for autonomous, accurate, and safe material identification workflows in circular construction and deconstruction processes.

1. Introduction

This document contains the information on the Navi-Wall deliverable 1.2 (D1.2) of the DISCOVER project (HORIZON-CL4-2023-TWIN-TRANSITION-01-11-101129909). Navi-Wall is a navigation open software for mobile manipulators for external and internal wall, floor and ceiling data acquisition.

1.1. Work package 1

Work package 1 (WP1) encompasses tasks essential to create the functional *Oliwall* robot prototype. Oliwall is a mobile manipulator robot, whose task is to autonomously navigate buildings and infrastructure, continuously collecting data about spaces, elements, and materials without invasive testing. The sensors allow for characterization of both visible elements and materials, as well as those hidden within the structure.

The key objectives of this work package are:

- The design of the *Oliwall* robot according to the needs and requirements of the construction sector, especially adapted to operate in demolition environments.
- The development and implementation of the non-invasive identification techniques for data collection, which will be based on vision sensors, the ground-penetrating radar (GPR) and LiDAR.
The construction of the prototype robot and the user interface, and its validation in a controlled environment.

2. Deliverable 1.2: Navi-Wall

Within the objectives of WP1, the Deliverable D1.2 includes the results on:

- Robot action in order to successfully navigate the geometry of the building. Data collection process, fast geometry reconstruction process, implementation, simulation results, and results on testing environment.
- Robot path planning for identification of hidden elements (*testing run*) following the geometry of the space. Path planning algorithm and implementation, simulation results and results on testing environment included.
- Robot path planning for wall scanning and identification of materials and internal elements. Base path planning and docking algorithm and implementation, robotic arm path planning and implementation, control techniques and implementation, and simulation results included.

2.1. Contents of the report

This report contains information on the techniques, tests and results that lead to the creation of the autonomous behavior for robot scanning of buildings. It also contains links to all the material related to this deliverable: code, simulations and videos of tests and tests results.

The report is structured as follows: Section 3 contains the description of the facilities, equipment and materials used in the development and testing for the deliverable. Section 4 defines the methodology to model the space where the robot will be working. Section 5 describes the Navi-Wall strategy and implementation, and Section 6 summarizes the results and structures the components of this deliverable.

3. Navi-Wall equipment and facilities

The Navi-Wall deliverable requires experiments with the mobile manipulator and tests on a space that can be used as ground truth. In this section we present the equipment and facilities used to develop this deliverable.

3.1. Oliwall

Navi-Wall is going to be implemented in the *Oliwall* robot. Oliwall is an omnidirectional mobile manipulator, developed for the DISCOVER project, whose task in the project is to autonomously perform non-destructive data collection for identification of elements, components and materials in the building to be demolished. A brief summary of Oliwall characteristics is presented below.

The dimensions of Oliwall's base are 810x610mm. The mounted manipulator arm is a commercial UR10 robot from Universal Robots. This arm is mounted on a lifting column, so that the total height of the robot can be up to 3,2m. Its slender build means that stability will be an issue for the robot, which must be carefully considered. The robot is powered by three motors, making it possible for the robot to move instantaneously in any direction. The robot can navigate slopes of up to 30% and overcome 55mm-high obstacles. Its total weight is around 250 kg. [Figure 1](#) shows the design of the robot, that will be built at CDEI-UPC during the execution of WP1.

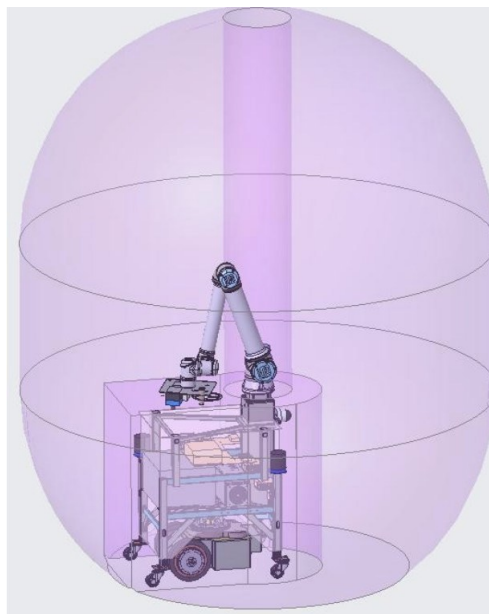


Figure 1. Oliwall robot design and workspace.

The arm manipulator of Oliwall has a sensor plate at its end effector, where all the non-invasive sensors used to detect materials and elements are located. The plate contains a GPR, an RGB-D camera, a hyperspectral camera, a metal detector (optional), and some range sensors to keep the parallel distance between plate and wall (see [Figure 2](#)).

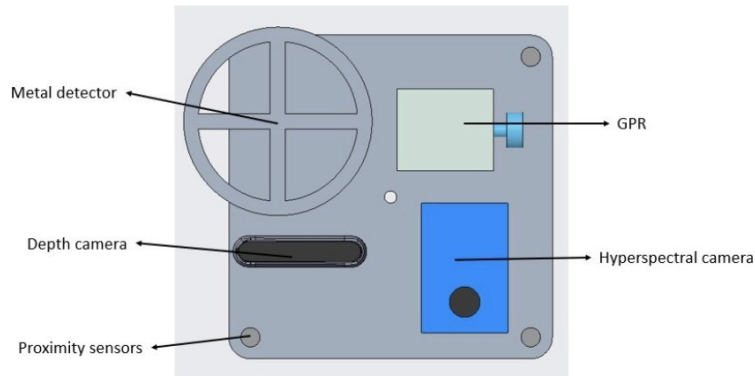


Figure 2. Sensors' plate at the end effector of the robot.

For the navigation and for mapping the space, Oliwall has three LiDARs. The dome Ouster LiDAR will be used to create the point cloud that will define the geometry of the space. Two smaller LiDARs will be mounted on the sides of the robot for the navigation, as shown in [Figure 3](#).

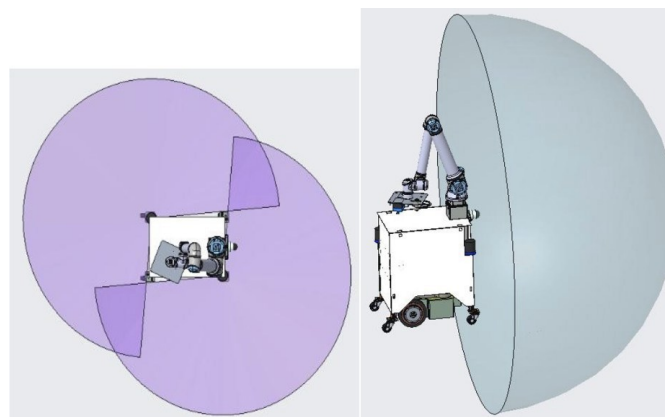


Figure 3: Left, the two LiDARs used for navigation. Right, the dome LiDAR used to capture the geometry of the space.

For more information on the Oliwall design, please check [1].

3.2. Current mobile manipulator

Because of the fact that Oliwall is under construction until the end of Task 1.6 of WP1, a similar existing robot at the CDEI-UPC laboratory is being used for the testing and software developing. The robot has the same kinematics and similar dimensions as Oliwall, however, it is an agricultural robot (AgroMOBY), which means that the different type of wheels will create different dynamic effects and also lower precision in its

positioning. [Figure 4](#) shows AgroMOBY, equipped with the arm manipulator to be installed in Oliwall.



Figure 4. AgroMOBY robot with UR10 navigating in the CDEI-UPC Mobile Robotics Laboratory (MRL) space.

3.3. Oliwall sensors

The sensors that will be attached to the end-effector of the robot and will collect data during the Navi-Wall behaviour include:

- OAK-D pro RGB-D camera: an active-stereo depth camera combined with a high-resolution color camera; the OAK-D Pro camera is used for the automatic identification of elements in the space. This camera can help during the Navi-Wall behaviour, and is also actively used during the general scanning of the space.
- *Ouster* Dome LiDAR: The *Ouster* Hemispherical LiDAR offers a complete 180 degrees field of view in 128 beams, with a range of up to 20 meters and a high resolution, with 10% reflectivity. Other interesting features are the multi-sensor cross talk suppression and the proprietary software for point cloud evaluation. This sensor will be attached to the base of the robot and will be used to create the point cloud that will yield the precise geometry of the space. In addition, it may complement the robot navigation.
- ScreeningEagle GPR: The *Screening Eagle* GP8800 is a ground penetrating radar, whose main advantages are its small size, range of frequencies and proprietary software for quick visualisation of the data. This GPR will be mounted on the wrist of the robot, and will be enabled for data taking during some of the Navi-Wall actions. [Figure 5](#) shows the aspect of the GPR.



Figure 5. The GPR to be used in the project (From Screening Eagle).

- Hyperspectral sensors: A couple of hyperspectral sensors are to be installed also on the wrist of the robot, to be used in scanning operations: a *Ximea HIS* camera and a *Nexos 2K VIS-NRI* spectrometer. These sensors will be integrated later in the system.
- Ultrasonic and laser-range sensor for wall location: A combination of three ultrasonic ranging sensors *HC-SR04* (2 to 400 cm) and three laser *Adafruit VL6180* time-of-flight sensors (5 to 200 mm) are used to adjust the proximity and correct orientation of the sensor plate with respect to the surface to be analysed (wall, floor, ceiling or column).

3.4. Testing grounds

All the laboratory testing will be performed in the CDEI-UPC Mobile Robotics Laboratory (MRL), a 250 m² open space that can be adapted to create different environments. [Figure 6](#) shows part of this space.

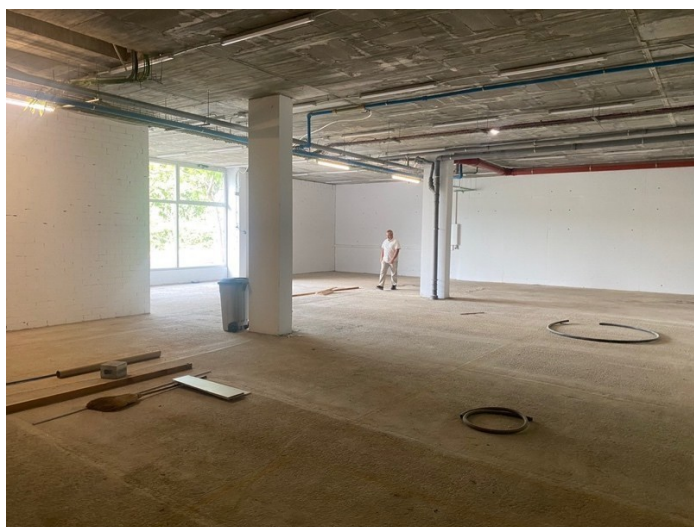


Figure 6. Testing laboratory space (CDEI-UPC MRL).

A virtual model of the MRL has been created for testing the algorithms of Navi-Wall. [Figure 7](#) shows this model. The model is based on SDF code written in Python to create a world called “castelldefels_indoors”, which contains walls, floor, ceiling, columns and entry ramp. There are also several non-fixed objects that can be placed in it, such as tables. The model is dimensionally correct and is available at [2].

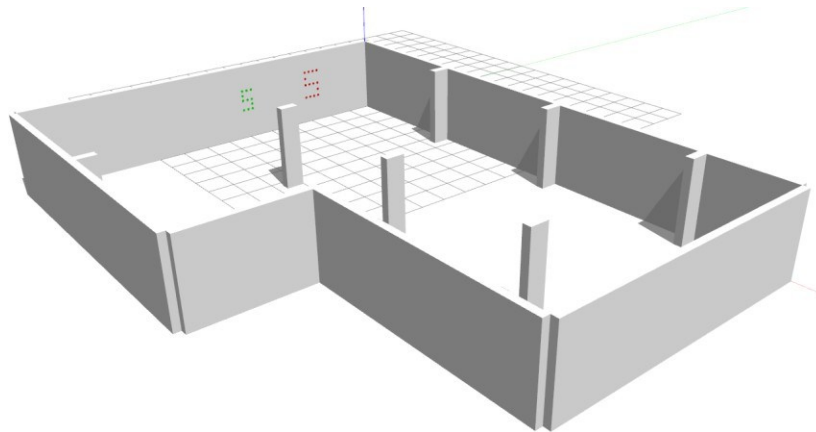


Figure 7. 3D virtual model of the CDEI-UPC MRL.

This model has been imported in the ROS 2 simulators, where the model of the AgroMOBY robot has been made to navigate and to observe the space through a model of the RGB camera. The videos, from which the snapshot in [Figure 8](#) has been taken, can be obtained at [2]. [Figure 9](#) also shows the laboratory model within the Gazebo simulator environment.

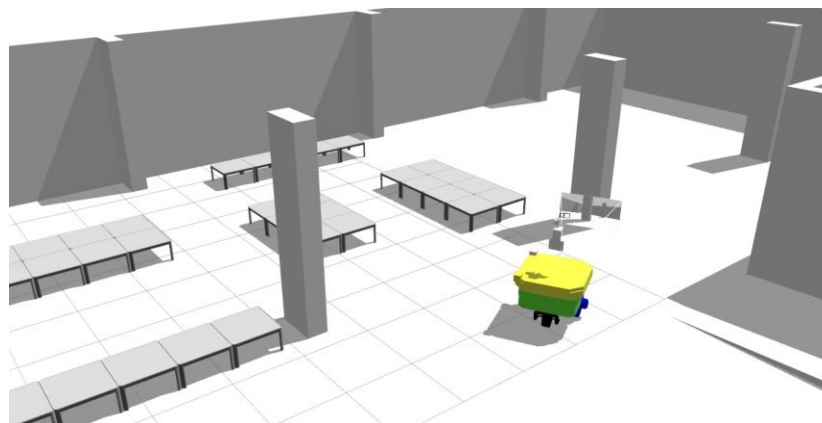


Figure 8. Simulation of the AgroMOBY robot in the CDEI-UPC MRL space.

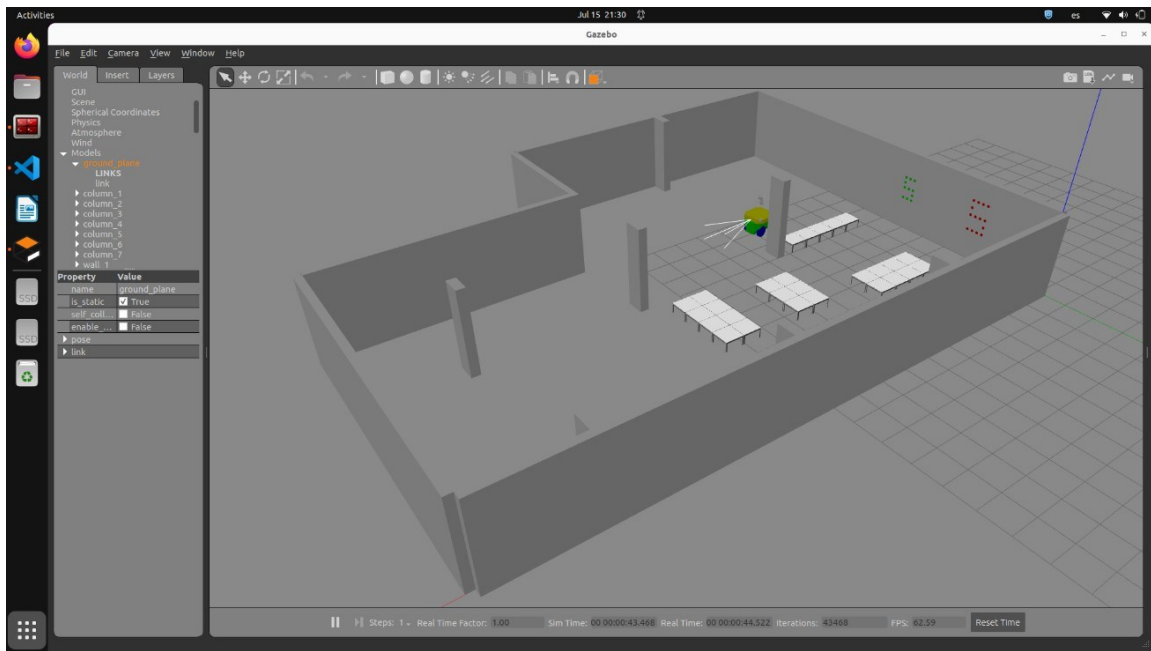


Figure 9. The laboratory world and the robot within the Gazebo environment.

4. Sensing to geometry

The robot needs to have a representation of the space in which it navigates; that is the navigation map. We assume that no map will be available for many deconstruction sites. In addition, the robot needs to understand the space in which it will be moving, identify walls, ceiling, floor, and columns and to compute their geometry; this is the semantic map, which will be the base for the Navi-Wall planning.

Even though the precise creation of the geometry is a task to be completed later in the project, in order to test Navi-Wall, we created a simple procedure for the robot to scan the environment and to identify the geometry with an acceptable precision. This process is described in this section.

4.1. The robot in the space (frame transformations)

For the main operations that need to take place in the autonomous robot (navigation and data acquisition), georeferencing is very important. That means that we need to know where we are measuring from, both for the robot and any of the sensors, and what is the relative positions among all the active components.

We have three main frames: map frame, which could be pre-established or computed, odometry frame, which will be set at the beginning of the robot operation, and the robot frame, corresponding to the actuated chassis, which will be moving in the scenario. [Figure 10](#) shows these frames.

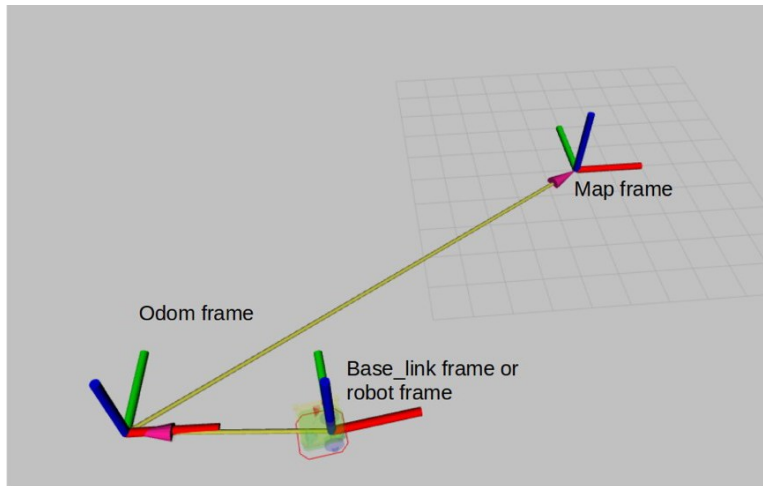


Figure 10. Reference frames for the robot and the space.

Within the robot, we distinguish the chassis (base frame), the turret (which is the platform supporting arm and sensors) and the sensors (camera_link frame), as well as local frames belonging to each wheel, as shown in [Figure 11](#).

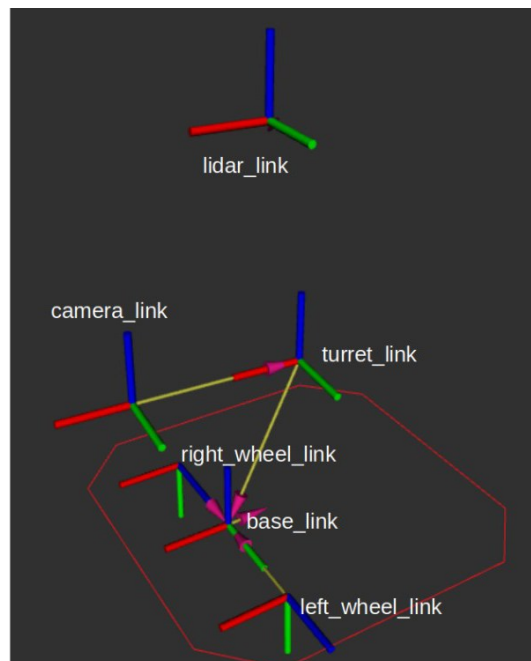


Figure 11. Reference frames within the robot.

The parent-child relationship between these frames is captured in [Figure 12](#). The format follows the ROS tf package, whose function is to define and maintain the relationship among reference frames.

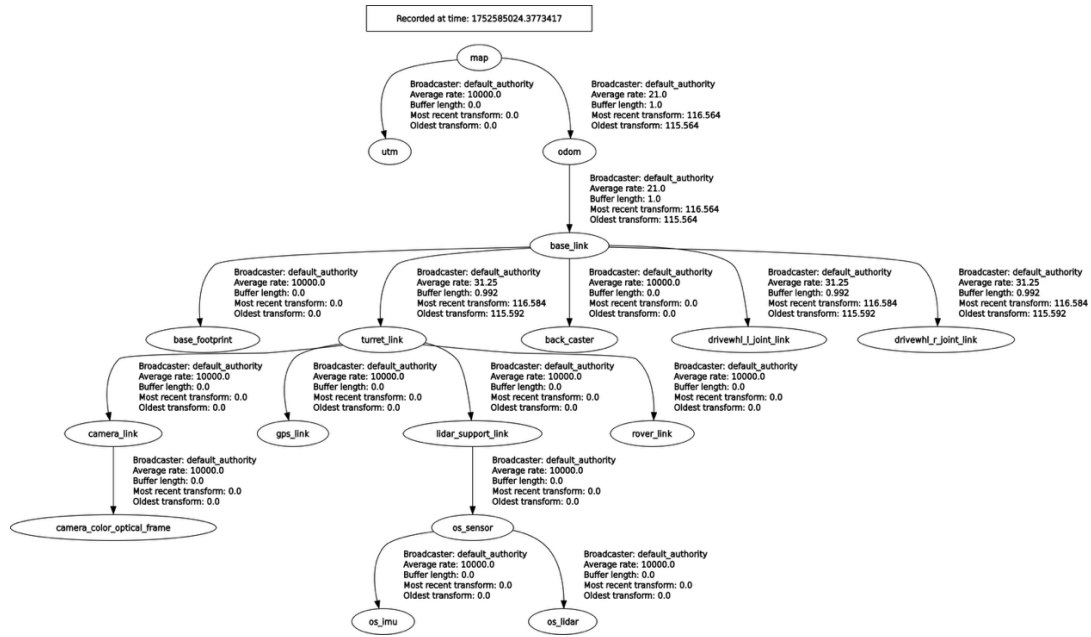


Figure 12. tf tree showing the relationship between the reference frames.

4.2. Model from simulated LiDAR data

A digital twin computed with simulated LiDAR data was developed in ROS 2 in order to create the pipeline and check the expected precision. For the experiments, the simulated LiDAR was configured using a custom URDF model representing a Sick MultiScan sensor. It was simulated in Gazebo with a vertical field of view from -22.5° to $+42.5^\circ$, a horizontal resolution of 100 samples per scan, and 16 vertical laser channels. The simulated sensor had a maximum range of 90 meters, a minimum of 0.5 meters, and included Gaussian noise with a standard deviation of 0.005 to emulate real-world inaccuracies. It operated at 10 Hz and published point cloud data to the `/points` topic using the `libgazebo_ros_velodyne_laser.so` plugin. The physical model was based on the dimensions of a compact cylindrical LiDAR unit, and the optical properties were defined to closely mimic the behaviour of real-world sensors for accurate digital twin generation [3]. The process is detailed below.

The point cloud map is saved in a `.db` file and later processed using the RTAB-MAP tool. The output is a mesh with a `.ply` file format that represents the 3D model of the scanned environment. The model can then be exported in `.stl` and `.sdf` formats, making it ready for integration into CAD software or as a solid model in the Gazebo simulator.

In order to reduce the error, the cloud is cleaned up with a Radius Outlier Removal algorithm. Then the cloud is downsampled to reduce the computational cost.

The next step is the segmentation of walls, floor, and ceiling. It consists of calculating the normal to all points and applying a RANSAC (Random Sample Consensus) algorithm. This method fits a plane to a random number of points and then clusters all the points close to this plane. This process is repeated until obtaining the minimum error.

For the points not belonging to walls, ceiling or floor, a DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm is applied, to cluster those belonging to

different objects, and to further eliminate noise. This algorithm identifies dense point sets by computing the distance among points and minimizing the number of points belonging to the group. [Figure 13](#) shows the results of each step for a simple square room.

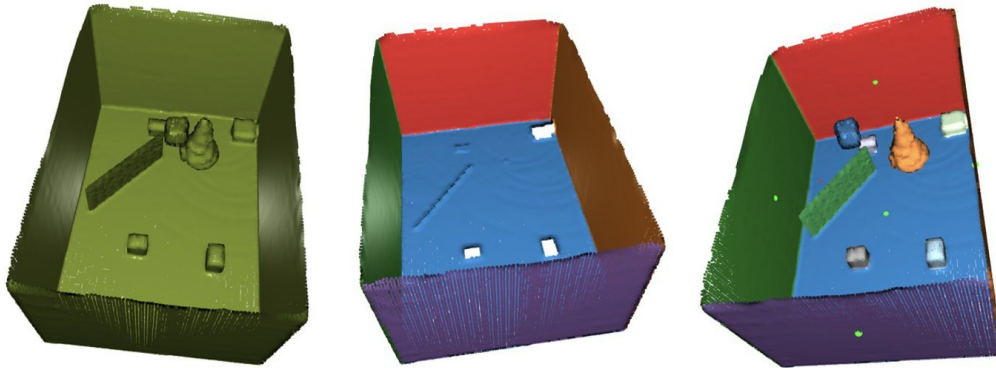


Figure 13. Example of the point cloud to segmented geometry algorithm.

The last step is to calculate distances of the centroid of each plane and object with respect to the global frame of the point cloud. All this information is to be stored in a .csv file.

The error between the precise model and the one obtained with the LiDAR information is calculated as a difference in the distance between the walls for the models used. We obtained an error of approximately 4% in these initial measurements. This value is higher than the expected value to comply with the KPIs. However, it is sufficient for the tests needed to develop Navi-Wall. It is important to point out that the dimensionally accurate representation of the space is part of a different deliverable to be developed later in the project.

4.3. Data capture with OSDome LiDAR

The OSDome LiDAR is mounted onto the AgroMOBY robot at the same position as it will be placed in Oliwall, as can be seen in [Figure 14](#). Ouster provides with a ROS 2 driver to stream point cloud data. The OSDome LiDAR provides the raw point cloud data that will be used for full environment reconstruction. For our experiments we use the RTABMAP algorithm to reconstruct the environment from the point clouds. RTABMAP, a robust algorithm for simultaneous localization and mapping (SLAM), uses the incoming point cloud data to build and update a 3D map of the environment over time. In the context of point cloud data, RTAB-Map processes the LiDAR data, registering it to the existing map, and uses loop closure techniques to identify when the robot has revisited a previously mapped area, which helps correct errors in the map over time; see [Figure 15](#) as an example of the geometry reconstruction. This enables the creation of detailed and accurate 3D reconstructions of the environment, which can be used for navigation or further analysis.



Figure 14. AgroMOBY with the OSDome LiDAR.

The raw data is stored in a rosbag. This raw point cloud data can be published and accessed by listening to the topic called: **dome/points**, while other dome related topics are also present under the dome namespace. In order to assess the results, sample data has been provided (see references in Section 6).

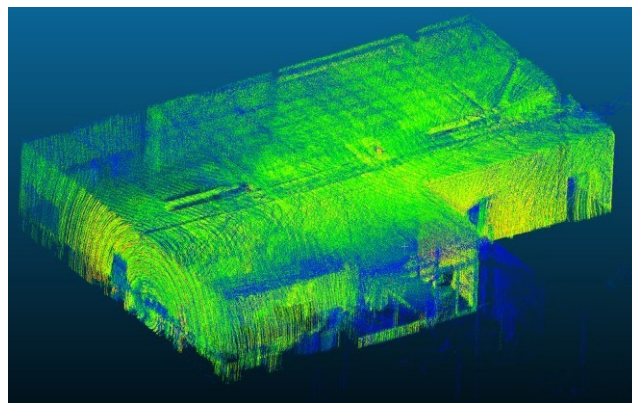


Figure 15. The 3D registered point cloud of the environment.

In Navi-Wall, the reconstructed and segmented geometry of the space is the input to the process. Additional refinements to this process will be done, which will increase the precision of the results.

5. Navi-Wall: robot navigation for non-invasive data acquisition

In its core, **Navi-Wall is a path planning and navigation behaviour for the robot, suited for the specific operation of non-invasive data acquisition in the demolition sites.** As such, it requires the robot to be able to map, locate itself, and navigate the space

in which the data is taken, while keeping a certain precision with respect to the plan. These are standard requirements for any autonomous mobile robot, independent from the precise geometric computation of the space and the semantic identification of its parts.

As part of our project's indoor navigation subsystem, we have implemented a complete navigation pipeline using the standard ROS 2 Nav2 stack. This includes SLAM-based mapping, localization, and both global and local planning components tailored to our use case.

For the mobile manipulator to follow the planned trajectory, a new control has been developed, suited for the omnidirectionality of the base and allowing the combination with the control of the arm manipulator.

The precise approximation to the walls is accomplished with a set of small range sensors and a reactive control.

The geometry is identified with a half-sphere (dome) LiDAR and the sensing for materials and elements is executed by cameras and a GPR mounted at the wrist of the manipulator.

All these elements are combined in the robot's finite state machine to define the behaviour of the system.

5.1. Navigation mapping and localization

To map the environment for the purpose of navigation, we are using RTAB-Map (Real-Time Appearance-Based Mapping) combined with a 3D LiDAR sensor. The raw 3D data is processed and projected into a 2D occupancy grid, which serves as the basis for global navigation. This allows us to leverage the precision of 3D sensing while maintaining compatibility with 2D navigation tools. Mapping is conducted in an initial exploration of the space, to generate a consistent and accurate floorplan of the indoor space.

The action of knowing the position and orientation of the robot with respect to a global reference frame is called localization. For localization, we rely on RTAB-Map's odometry and SLAM modules. The odometry component estimates relative motion using LiDAR data, providing the transform between the robot's base frame and the odometry frame. Simultaneously, the SLAM component maintains global consistency by detecting loop closures and generating the map-to-odometry transform. Both systems use only the 3D LiDAR, which simplifies sensor fusion while maintaining reliable pose estimation.

A link to the launching file for mapping and localization in the space can be found in Section 6. The pseudocodes are presented in the Appendix 1.

5.2. Global and local path planning

The global path planning consists of generating trajectories within the map to arrive to desired positions. It is handled by the Nav2 Grid-based planner plugin, which is configured to operate at a high update frequency. It is designed to tolerate some level of

unknown space in the map and does not rely on A^* but rather on a Dijkstra-based approach. This ensures robust, cost-optimal path generation across the mapped environment. The planner is tuned for responsive performance and smooth goal tracking in static and partially known indoor settings. This Navigation 2 stack is an open-source project for the ROS 2 framework, freely available on GitHub.

In the global planner, the grid encodes the environment as a matrix of cells, whose values represent free space (0), occupied or obstacle space (1), and unknown space (-1). Dijkstra's algorithm works by assigning costs to each cell, starting from the goal (or start, depending on implementation), and then expanding outward in all directions, updating the minimal cost to reach each cell. [Figure 16](#) shows an example of this algorithm.

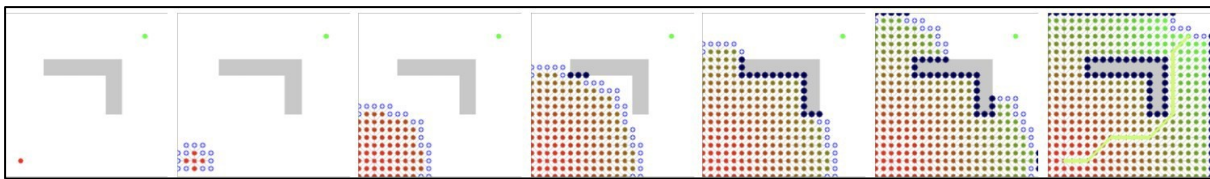


Figure 16. Example of Dijkstra's algorithm [Wikipedia].

The local planner creates the behaviour to follow the path and to avoid any obstacle that could appear in the planned path. For local trajectory execution, we use the Regulated Pure Pursuit Controller. This controller is configured for smooth and safe path tracking, with careful handling of velocity regulation, curvature adaptation, and rotation behaviour. Key aspects of the configuration include:

- High-frequency control updates for smooth actuation.
- Collision prediction mechanisms to ensure safe operation near obstacles.
- Dynamic adjustments to linear and angular velocity based on path curvature and distance to the goal.
- Goal-checking logic that ensures precise final positioning and orientation.

These settings provide a balance between agility and stability, making the controller well-suited for indoor navigation with tight corners and cluttered paths.

5.3. Sensor fusion for navigation with combined surface and subsurface data

One of the novelties proposed in the Navi-Wall behaviour was to combine the navigation in the space (surface navigation) with a navigation guided by the elements embedded in the walls, floor, ceiling or columns. The rationale behind it was that it could be interesting to follow an identified embedded element (for instance, a pipe) while not colliding with external objects.

The solution presented here uses reinforcement learning to fuse the surface and internal navigation while following a goal. We show that it is possible to train a mobile robot for this purpose, and the modularity of the training allows to change the behaviour with additional, less costly training.

The solution was not implemented in Navi-Wall. Technical discussions led to the decision that, for demolition buildings, it is not a needed feature; the hidden elements in buildings follow a very specific and well-known distribution, which can be inferred from a single data point. For instance, reinforcement in concrete will consist of vertical bars that are not going to change direction suddenly within the wall. In any case, the methodology presented here is ready if we want to implement it later on.

All the content in this section is from the master thesis by Riccardo Rettore (2025) [4].

5.4. Reinforcement learning for embedded navigation

The algorithm is based on the action of the robot, with the corresponding measure of results, in the simulation environment. The agent's behaviour is controlled by a policy, which maps state to action. This policy is shaped by the reward signal that the agent gets after each of the actions (see [Figure 17](#)).

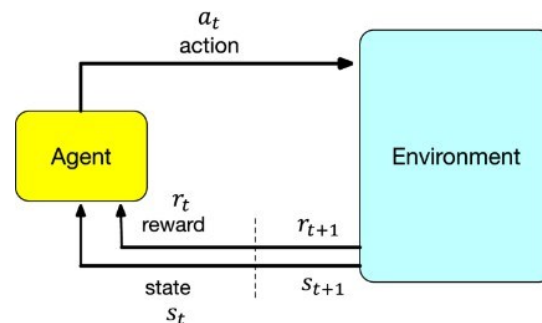


Figure 17. Reinforcement learning strategy.

The system was trained with previously-available GRP results which were already processed by an AI algorithm in order to detect water content in the ground. The algorithm is a convolutional autoencoder in which the GPR signal is the input and the latent space representation yields a percentage of water content at a given georeferenced position.

For the training simulation, the agent is located on a grid representing the underground map. The state representation includes the position of the agent, the position of the goal, and the partial knowledge of the percentage of humidity around the agent. We assume the agent will not have a global humidity map, as this is a knowledge that will be acquired as the robot navigates. These grids are created based on modifications of real data acquired by the sensor, in order to generate more training sets. [Figure 18](#) shows a couple of examples.

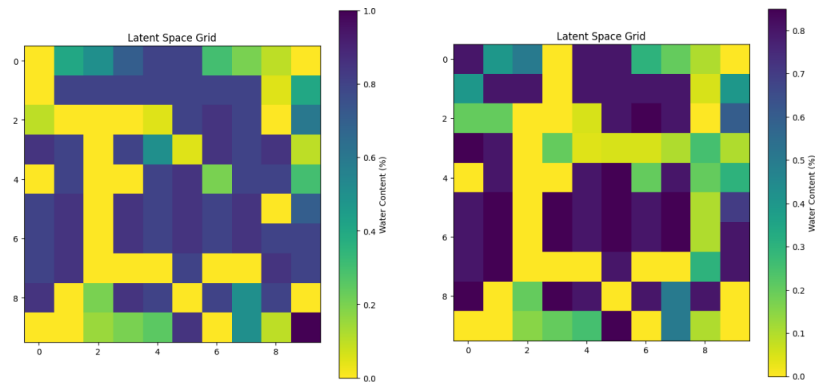


Figure 18. Latent space for the percentage of underground humidity.

The reward function contains several terms that include distance to the goal, exploring behaviour, water percentage and number of steps, among others. The results of the training are presented in [Figure 19](#).

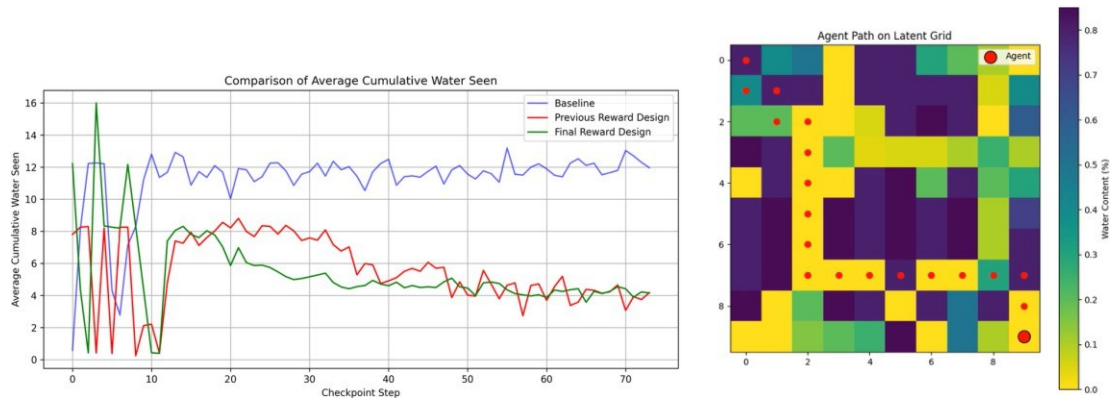


Figure 19. Left, improvement on the water seen by the agent after the training. Right, an example of a path selected by the agent after the training.

5.4.1. Reinforcement learning for embedded navigation: subsurface-only and surface-subsurface data fusion

In this case we assume that the robot (the agent) has a surface map of the space where it is navigating. This matches the real application, in which the surface navigation map will be created before applying the Navi-Wall behaviour.

The reinforcement learning starts with the pre-trained system for the embedded navigation. A series of walls and obstacles are added to the system, the information is added to the state and action spaces, some terms are added to the reward function, and the agent is trained again via simulation. An example of the results is shown in [Figure 20](#).

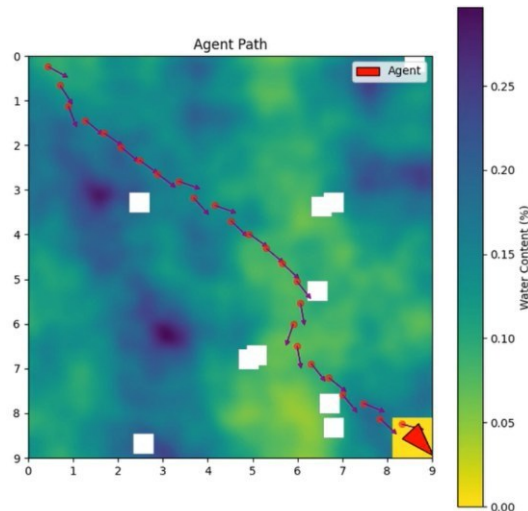


Figure 20. Example of robot finding the path with less water (blue and yellow background) while avoiding surface obstacles (white).

5.4.2. Summary

In summary, the method to create a map fusion within Navi-Wall has been created and tests were successful. As mentioned before, it will not be implemented in this first version of the behaviour of the robot, as in technical talks with the team and the partners, it is considered unnecessary for now. However, the method is ready to be applied later if needed.

5.5. The mobile manipulator: combined path planning and docking

The goal of Navi-Wall is to create an action planning for the overall mobile robot and arm manipulator, what is called a mobile manipulator. In order to do this, the motion planning of the base is complemented with the motion planning of the arm, and a combined docking strategy is also implemented. In addition, both arm and base will be controlled together to follow a given trajectory (mostly base control) while reacting to accomplish the desired proximity to the objects to be analysed, mostly - but not solely - through the motion of the arm.

5.5.1. Motion planning for the UR10 robot

For the UR10 arm manipulator, a motion planning algorithm has been developed using a discretization of the reachable workspace of the robot and the A* star path finding algorithm. The planner computes the trajectory of the robot's end-effector by taking into account the possible obstacles defined inside the reachable workspace of the robot. This planner computes up to 8 different joint configurations at each location of the end-effector origin, for the robot to achieve a given pose. Out of the multiple solutions obtained, the algorithm selects the one closer to the previous pose of the robot, to ensure that no unnecessary joint movements are performed.

The data of the reachable workspace with the joint configurations of the robot has been pre-computed and it is stored locally in the computer. The algorithm has been developed in Python and is implemented into ROS 2 and verified through simulation and testing.

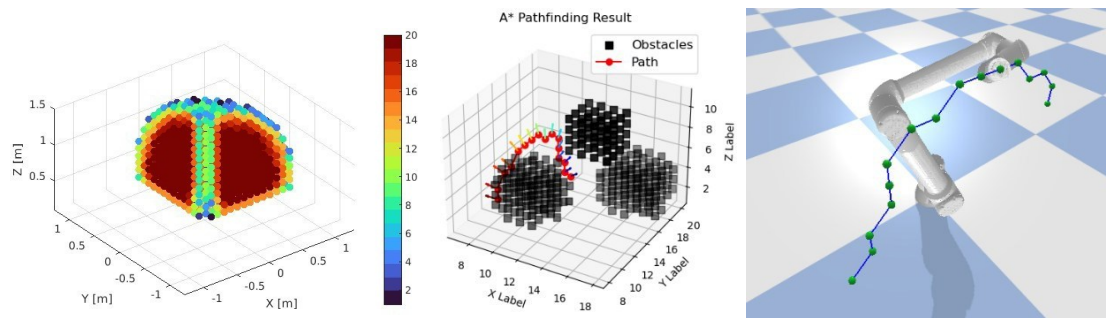


Figure 21. Left, the reachable workspace of the robot manipulator; Centre, a trajectory avoiding obstacles within the workspace; Right, the robot arm with a planned trajectory.

The manipulator simulation and control use the ROS 2 driver of Universal Robots, in which trajectories can be sent to the real (or simulated) robot. In this version, the smoothness of trajectories and robot's movements, as well as the presence of dynamic obstacles, have not been included and will be the object of future refinements (see [Figure 21](#)).

5.5.2. Docking for the mobile manipulator

When a wall has to be scanned, Oliwall should move to the optimal point to perform the scanning. This can be done following two strategies: in the first one, a combined motion of the mobile base and the arm is used to follow the wall trajectories. In the second one, the robot's mobile base moves to the optimum point to perform the scan, docks in this position, and the arm manipulator performs the scanning. We have selected the second option for Navi-Wall, as it allows to greatly increase the positional precision.

An algorithm to find the optimal base placement of the manipulator has been developed. This algorithm takes as input a list of points to be reached by the scan. Then, it computes all the feasible base positions for the list of points and selects the optimal one. At the moment, the optimal base is being selected taking into account the distance to the points (considered to be in the wall) and selecting the most-centred position available.

The results of the algorithm can be seen in [Figure 22](#).

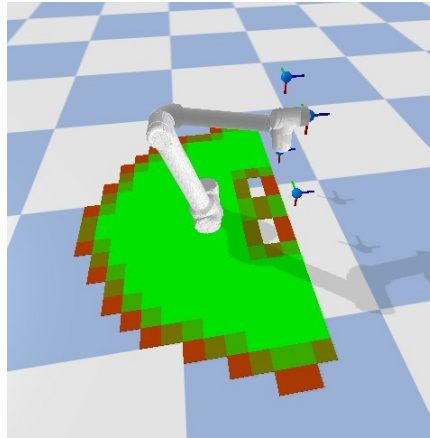


Figure 22. Docking space for the specified poses on the wall. Green colour indicates docking points that can reach all specified wall poses.

5.6. Oliwall's control

The control of the *Oliwall* robot is structured around the *ros2_control* framework, a modular and scalable control interface designed for the ROS 2 ecosystem [5]. This framework enables standardized interaction between software controllers and hardware components, facilitating integration, modularity, and reusability across robotic platforms.

The control architecture of the *Oliwall* robot encompasses two main subsystems: the omnidirectional mobile base and the UR10 robotic arm. Each subsystems employs a dedicated hardware interface and control strategy suited to its functionality and requirements.

5.6.1. Omnidirectional base control

For the omnidirectional base, a custom ROS2 hardware interface has been developed specifically for this application. This platform ensures full compatibility with any standard ROS 2 controller by abstracting the details of the hardware implementation. At the hardware level, the base relies on three *Infranor Easy-AK-60/60* drivers and a programmable logic controller (PLC), the *Xtracontrol EC21*, which serves as the intermediary between the servo drives and the ROS 2 control infrastructure.

Within this configuration, the PLC plays a critical role. It is responsible for gathering real-time feedback from the motors, including encoder tick counts and estimated velocities, and forwarding this information to the ROS2 software via the hardware interface. Conversely, the PLC receives motor velocity commands computed by the ROS2 controller and transmits them to the corresponding motor drivers. The PLC also handles low-level error management, such as exception handling and the triggering of error flags, ensuring a safe and robust control loop. The PLC software is implemented using CODESYS, a widely used IEC 61131-3 compliant development environment, which allows for structured, real-time industrial programming.

5.6.2. Robotic arm control

The control of the UR10 e-Series robotic arm is achieved through the proprietary hardware interface provided by Universal Robots. This interface is integrated within the ROS 2 ecosystem using the Universal Robots ROS 2 Driver, which exposes high-level functionality for both command transmission and state feedback. The driver allows the ROS 2 controllers to issue commands in various modes—position, velocity, and torque—depending on the task requirements. Simultaneously, it enables the reception of comprehensive state data from the robot, including joint positions, velocities, efforts, input/output (I/O) signals, motor currents, and other internal diagnostics. This hardware interface facilitates seamless communication between ROS 2 and the UR10 hardware, enabling complex motion planning, task execution, and integration with external systems. The documentation about this hardware interface can be found in [6].

5.6.3. Software controllers

The Oliwall robot employs two main software controllers to manage motion: the `sim_controller` for the omnidirectional base, and the `joint_trajectory_controller` for the UR10 robotic arm. Both controllers conform to the `ros2_control` architecture and are loaded into the ROS 2 control manager at runtime.

The `sim_controller` is specifically developed to handle the motion of the robot's omnidirectional base. It receives as input a desired velocity vector defined in the turret frame and computes the necessary wheel velocities to achieve that motion. This computation takes into account the kinematic configuration of the base, the geometry of the wheels, and the orientation of the robot. In addition to motor commands, the controller publishes odometry data and internal diagnostic variables to a range of ROS 2 topics. This data publication is essential for enabling interoperability with the robot's navigation stack, which relies on accurate localization and motion feedback.

The controller is designed with a set of configurable parameters that enable flexible and modular behaviour. These parameters include:

- `should_publish_tf`: When set to true, the controller publishes transformation data to the ROS 2/`tf` topic, enabling real-time visualization and spatial reasoning within the system.
- `should_publish_odom`: If enabled, this parameter causes the controller to publish standard ROS 2 odometry messages to a predefined topic, which is necessary for seamless integration with navigation tools.
- `update_mode`: This parameter dictates how the internal state of the controller is updated during each control loop. It supports three modes:
 - In `readings` mode, the controller updates based on real-time encoder feedback from the hardware interface.
 - In `tf` mode, the state is updated using transform information from a designated ROS 2 topic.
 - In `open_loop` mode, the state is propagated forward based solely on issued command velocities, without relying on feedback.

- **cmd_type:** This parameter can be set to either relative or absolute. In relative mode, the velocity commands are interpreted in the robot's local turret frame. In absolute mode, the input is assumed to be in a global reference frame, defined at the time of initialisation. The velocity calculations performed by the controller depend on the values of these parameters. Different control strategies and integration behaviours can be achieved by tuning them appropriately for a given deployment scenario.

5.7. Wall approximation and wall following

A critical aspect of the *Oliwall* robot's autonomous inspection capability is its wall approximation procedure, which governs the way the robot approaches and positions itself relative to a wall for inspection or scanning tasks. This behaviour is executed under the assumption that the robot has already completed an environmental scan and has accurate knowledge of the walls' positions in the workspace.

The wall approximation procedure begins by determining the optimal scanning start and end points for each identified wall segment. Based on a predefined selection criterion—such as proximity, priority, or scan completeness—the robot autonomously selects the next wall to approach.

Once a target wall is chosen, the robot computes a motion trajectory to navigate towards it. As the robot reaches the vicinity of the wall, it docks, positioning its turret such that the UR10 robotic arm faces the wall perpendicularly. This orientation is critical to ensure that the sensor attached to the UR10 is aligned with the surface for accurate data acquisition. The turret orientation is computed based on the geometric relationship between the wall's endpoints and the robot's current position.

Following orientation, the UR10 arm begins its unfolding sequence, extending the sensor toward the wall in a direction perpendicular to the surface. During this phase, the robot enters a feedback-controlled approach mode. To scan the walls, the various sensors used will need to maintain a constant distance with respect to the wall, along with a defined sensing perpendicularity with respect to the wall plane. To ensure both distance and orientation of the sensor's plate at the end effector during the wall-scanning operation, a triangular distribution of distance sensors has been added to the sensor plate.

The information of the sensors is used in a closed-loop control scheme to iteratively adjust the arm's position and orientation in real time.

The goal of this correction process is to achieve two specific conditions:

- The sensor must reach a predefined target distance from the wall.
- The sensor's orientation must be normal (i.e. perpendicular) to the wall surface.

Once these conditions are met, the robot considers the wall approximation phase complete and transitions to the scanning operation. This scanning process is dependent on the specific application—such as visual inspection, 3D reconstruction, or surface

defect detection—but is enabled by the precise alignment achieved during this preceding phase.

Since the arm will be gathered up and far from walls and also really close to them, it is important to be able to accurately ensure the sensor's plate distance to the wall in a large range. To accomplish that, the system uses both ultrasonic and time of flight (ToF) sensors, 3 of each in the edges of the plate. Since the ToF sensors acquired have more precision (up to 1 mm) than the ultrasound sensors, they are used when the end effector is close to the wall during the scanning operation, even though they have less operational range (5-20 cm). In addition, the ultrasounds sensors, despite having less precision (up to 1 cm) they offer a larger operational range (2 cm – 4 m). This allows to use them for approximation purposes, by detecting the wall and controlling the approximation sooner, when the arm is still far from the wall. Only one type of sensors will be used at the same time. The placement of the sensors can be seen in [Figure 23](#).

The control of both distance and orientation has been implemented into the ROS 2 ecosystem to use it along with the controller of the arm to perform the corrections. To input the sensor's values into the system, an Arduino Mega is used to take the measurements and pass them to the Oliwall's computer through serial. These measurements are published in a ROS 2 topic to make them available for all the nodes. The algorithm to correct distance and orientation consists on taking the measurements of the sensors and compute the projected sensor's points to obtain the wall intersection points. With these, the normal vector of the wall is obtained and the distances from the sensors to the wall are obtained. The distance from the sensor's plate center to the wall has been considered as the average of the 3 measurements, which is then compared to the required distance specified to compute the distance correction.

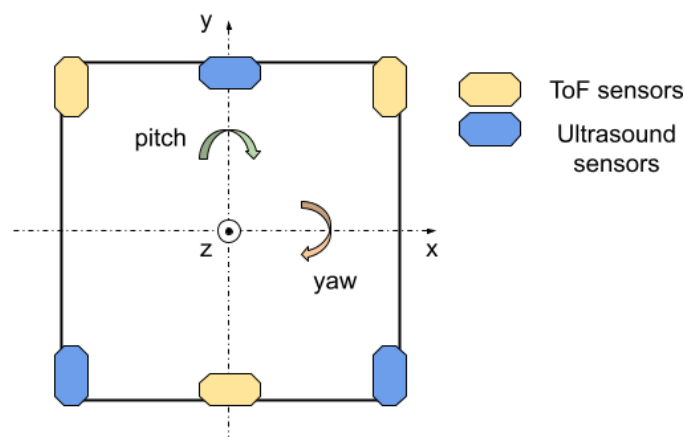


Figure 23. Location of range sensors on the sensor plate.

Also, the differential angle that needs to be corrected is computed as yaw and pitch angles on the end effector frame. Since the transformation between the base and end effector frames is known by the ROS 2 tf-tree, the goal orientation is computed as the composition of both current orientation and differential rotation needed. After that, a roll compensation is applied to ensure that the sensor's plate y-axis is always facing upwards as vertical as possible. Then, the goal position and orientation are used to compute the robot's joint positions by means of the inverse kinematics using a closed form algorithm. Finally, the joint positions are commanded to the robot.

The approach is based on the distance data (d_A, d_B, d_C) from the three non-colinear sensors located at given known positions on a plate mounted on the end effector. These three sensors form a triangle on a plane and ideally all three maintain the same desired distance reading d from the wall, ensuring a proper distance and orientation of the plate with respect to the wall:

$$\begin{aligned}\mathbf{w}_A &= \mathbf{p}_A + \mathbf{d}_A \\ \mathbf{w}_B &= \mathbf{p}_B + \mathbf{d}_B \\ \mathbf{w}_C &= \mathbf{p}_C + \mathbf{d}_C\end{aligned}$$

With the known positions of the sensors and the corresponding measured distances to the wall, we can obtain the position vectors of the projected points on the wall. Once these intersection points are obtained, two vectors (\mathbf{v}_1 and \mathbf{v}_2) that belong to the plane on the wall can be obtained, and the cross products between them will give us the normal vector of the wall with respect to the plate \mathbf{n}_w , which ideally should be parallel to the plate's normal vector:

$$\begin{aligned}\mathbf{v}_1 &= \mathbf{w}_B - \mathbf{w}_A \\ \mathbf{v}_2 &= \mathbf{w}_C - \mathbf{w}_A \\ \mathbf{n}_w &= \mathbf{v}_1 \times \mathbf{v}_2 = \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix}\end{aligned}$$

Once this vector is obtained, we can compute the pitch and yaw angular errors through its components:

$$\begin{aligned}\beta &= \tan^{-1}(w_x/w_z) \\ \gamma &= \tan^{-1}(w_y/w_z)\end{aligned}$$

With these angular errors we can compensate for the errors at the end effector level of the robot. Once the plate is correctly aligned with the wall, the distance can also be compensated directly in the z direction of the end effector according to the measurements (see [Figure 24](#)).

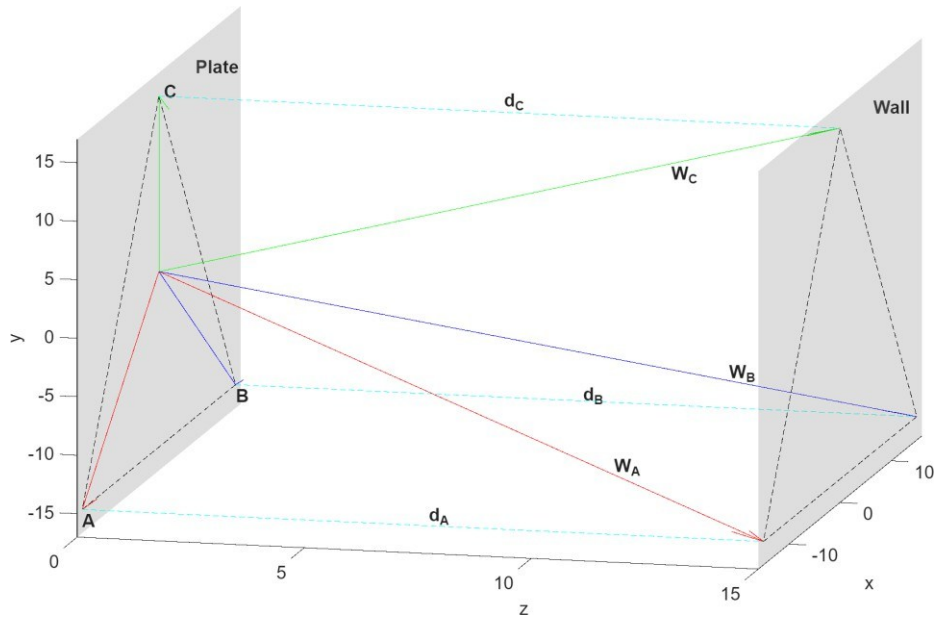


Figure 24. The schematics of the distance and parallelism from sensors to wall.

5.8. Plane discretization

For those planes (wall, ceiling, floor or column) where hidden elements are detected, the results of Deliverable 1.1 Robo Scan [7], are used to discretize the plane and obtain the overall information on the hidden elements.

The dimensions of the plane of interest (let us assume a wall in the following description) are calculated from the geometry and the wall is divided into panels, reachable by the workspace of the robotic arm once the base is docked. The horizontal divisions will be reached by the mobile base, while the horizontal ones correspond to the motion of the column on which the arm is mounted. The centroid of those subdivisions will be projected onto the moving plane of the robot and will be target points for its docking trajectory.

For each of these panels, subdivisions are created corresponding to the range of the sensor to be used. The centroid of those subdivisions is calculated, and the arm will perform a smooth trajectory through those via points and following the recommendations of Robo Scan. The trajectory will be actively compensated to keep the distance and parallelism from the wall as explained before. [Figure 25](#) shows the recommended paths and their implementation.

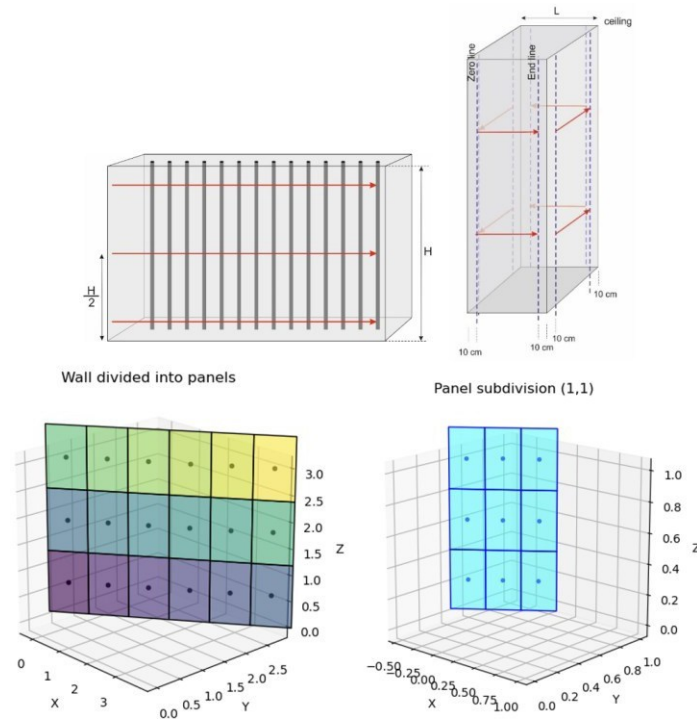


Figure 25. Above, scanning recommendations from Robo Scan. Below, the division in docking panels (left) and subdivision in scanning panels in order to create the arm trajectory.

This strategy can be easily extended and applied to all elements that need to be scanned in the building.

5.9. Finite state machine

The Navi-Wall strategy can be divided into several phases depending on the desired behaviour and the actual state of the robot. A general overview classification of the different steps for this process is:

- **Mapping:** The robot does not know where it is, and since it is assumed that no map will be available for many deconstruction sites, the robot must generate it. In this phase the robot will move autonomously through the room (or rooms) to create the map of the site, along with gathering all the sensors' data (such as LiDARs and cameras) in order to allow the reconstruction of the geometry site.
- **Geometry reconstruction:** With all the 3D data gathered during the mapping phase, the precise creation of the geometry has to be performed. This reconstruction should include the semantic map that later will be used in the scanning phases. This is a task to be completed later in the project.
- **Scanning first line:** The first scanning of the site must be to identify the basic materials and features, and also point out the areas that need to be scanned in detail. This first stage will consist of following a horizontal line on the walls with the robotic manipulator at a defined height of 1.5 meters from the floor. During this scan, the data from the different sensors on the sensor's plate will be

gathered. Also, the wall approximation control is used, to ensure a proper and constant distance for the sensor's measurements. The wall selected is the one with the closest start or end point to the robot.

- **Identification of interesting areas:** After all the walls and columns have been scanned for the first time, the ones which need to be scanned exhaustively are identified and marked in the semantic map. The algorithms that perform this feature identification are a task to be completed later in the project. For the moment, and to test the robot's behaviour, the areas have been manually identified.
- **Exhaustive wall scanning:** From all the walls identified for the detailed scan, the one with the closer start or end point to the robot is selected again. Then, the wall is discretized in several parts (along horizontal and vertical directions), as explained in the previous section. Then, the docking for the mobile manipulator is computed for each one of these vertical divisions, and the height of the lifting column is adjusted properly for the horizontal division to scan. Finally, the robot arm performs the scanning by following horizontal lines along the wall with the sensor's plate. The followed paths will be separated a distance determined by the sensor used. For the moment, the distance needed for the GPR scanning is considered. See Deliverable 1.1. Robo Scan [7].

A summarized scheme of this strategy can be seen in [Figure 26](#).

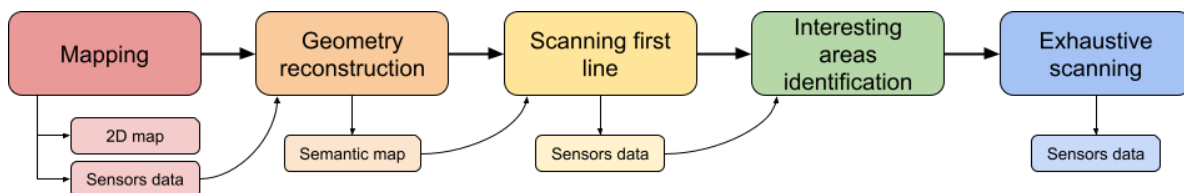


Figure 26. The mapping and scanning strategy.

The previous strategy has been implemented by means of a finite state machine (FSM). This FSM has been created using ROS 2 and python. First, two different python classes are defined:

- **State Machine:** This class defines the state machine, its variables and behaviour. A state machine object is created by calling the constructor of the class while indicating the states names and the context. The context is a dictionary containing the data that must be shared between the nodes of the FSM, such as the semantic map or errors triggering.
- **State:** This class defines the states. A state object is created by just indicating its name when calling the constructor. In this class the basic functions of the states are also defined: initialization, state operations, checking transition and end operations. These functions are fully defined in the definition of each one of the states.

Then, the FSM is implemented into ROS 2 by creating a node. In its initialization, the different states and context data are defined. Also, all the subscriptions and publishers must be created here, along with the definition of the services and servers that are going to be used to connect and interact with the ROS 2 ecosystem. Finally, a timer is defined

to control the update rate of the FSM. Also, it is needed to indicate by a topic message that the FSM process should begin. Each state is defined as a new class that inherits from the State class. In the class definition the operations and functionalities of each state are defined.

The states created to implement the Navi-Wall strategy are as follow:

- **Initialization:** This state allows to initialize the FSM in a resting state for the robot. The FSM will stay in this state until the external signal to begin is sent through the corresponding topic. Once the sequence has started, it is completely autonomous and requires no user intervention. This state saves the initial position of the robot in the place as the home position to return when all the procedure is finished.
- **Create map:** In this state, the robot will generate the map of the deconstruction site. Also, the data from different sensors is gathered during this process. A service to begin mapping is used, which will give a feedback response when the map is done. When this happens, the FSM moves to the next state.
- **Geometry reconstruction:** In this state, all the data gathered in the previous state will be used to create the reconstructed environment of the site and the semantic map.
- **Compute wall points:** With the information present in the semantic map, such as the start and end points of all the walls, the positions for the robot base to perform the line scans are computed for each wall.
- **Wall target selection:** The wall with its base start or end point closest to the robot's current location is selected to scan. This state will be repeated each time a wall needs to be chosen to scan.
- **Navigate to target:** This state communicates with the Nav2 stack of the robot to send the command initial pose for the wall scan. Since the action server for navigation offers feedback, when the robot reaches the goal, the FSM moves to the next state.
- **Arm unfolding:** The command to extend the UR10 robot is sent to approach the wall. As explained in the wall approximation section, the distance sensors are used to correct the orientation and obtain the required distance for the sensors. Once the robot arm places the sensors near the wall, the scan of the first line begins.
- **Scan wall:** Both the base and the UR10 robot move at the same time. The base moves along the wall, while the arm corrects the orientation and distance to the wall with the reactive control explained before. All the data gathered with the sensors is stored.
- **Arm folding:** This state ensures that the arm is in a safe configuration for the mobile manipulator displacement. This safe configuration is always the same with respect to the robot base and has been determined in advance.
- **Areas of interest:** When all the walls have been scanned, the ones to be scanned exhaustively are identified as so in the semantic map.
- **Wall discretization:** Both length and height of the walls to scan in detail are discretized in such a way that the arm reaches all the points of these small areas (panels) for the same base location, taking into account the reachable workspace of the robot. Then, each panel is decomposed into cells by following the same

Link to Task Planner https://github.com/DISCOVER-Horizon-Europe-Project/task_planner_fsm

Link to Navi-Wall Navigation https://github.com/DISCOVER-Horizon-Europe-Project/navi_wall_navigation

Link to Arduino Sensors https://github.com/DISCOVER-Horizon-Europe-Project/arduino_sensors

Link to Arm Control https://github.com/DISCOVER-Horizon-Europe-Project/ur_arm_control

Link to Planner https://github.com/DISCOVER-Horizon-Europe-Project/robotic_arm_planner/blob/main/robotic_arm_planner/planner_node.py

Link to Base Placement https://github.com/DISCOVER-Horizon-Europe-Project/robotic_arm_planner/blob/main/robotic_arm_planner/base_placement_node.py

Link to Control https://github.com/DISCOVER-Horizon-Europe-Project/ur_arm_control/blob/main/ur_arm_control/sensors_distance_orientation.py

Link to Wall Discretization https://github.com/DISCOVER-Horizon-Europe-Project/robotic_arm_planner/blob/main/robotic_arm_planner/wall_discretization_node.py

Link to the FSM https://github.com/DISCOVER-Horizon-Europe-Project/task_planner_fsm

Link to Robotic Arm Planner https://github.com/DISCOVER-Horizon-Europe-Project/robotic_arm_planner

Instructions to use the OSDome LiDAR data

Stream by playing the bag in a terminal:

```
ros2 bag play 2025-07-28_17-18-53_dome_data.bag
```

A sample point cloud file named **dome_cloud.ply** has also been provided, which can be viewed in any point cloud viewer such as CloudCompare.

Moreover, a database file called **dome_database.db** has also been provided, which contains all the raw point cloud data which is compatible with the [RTABMAP](#) project. The project provides a database viewer executable which can be used to load the database and process it offline for tasks such as processing, cleaning, filtering and reconstruction.

To use the viewer, install the rtab package for ros humble: **sudo apt install ros-humble-rtabmap**

Launch the viewer: **rtabmap-databaseViewer**

Load the file and experiment with different functionalities. The function export 3D map allows one to apply the full 3D reconstruction pipeline on the point cloud contained in the database file.

Simulations and videos of tests

[Link to Navi-Wall simulations and videos of tests](#)

Deliverables

Deliverable 1.1 Robo Scan (<https://zenodo.org/records/16575781>)

References

- [1] "Oliwall Design Concept," 2024. [Online]. Available: <https://zenodo.org/records/16577678>.
- [2] "Navi-Wall Navigation," [Online]. Available: https://github.com/DISCOVER-Horizon-Europe-Project/navi_wall_navigation/tree/master.
- [3] "Navi-Wall Description," [Online]. Available: https://github.com/DISCOVER-Horizon-Europe-Project/navi_wall_navigation/blob/master/navi_wall_description/description/sick_multiscan.xacro.
- [4] R. Rettore, "Leveraging Ground Penetrating Radar with Deep Reinforcement Learning for Subsurface-Informed Navigation in Autonomous Agricultural Robotics," M.S. Thesis, Dept. of Industrial Engineering, Politechnic University of Catalonia, Barcelona, Spain, 2025.
- [5] "ROS control package," [Online]. Available: <https://control.ros.org/rolling/index.html>.
- [6] "ROS2 Driver for Universal Robots," [Online]. Available: <https://github.com/UniversalRobots/UniversalRobots> ROS2 Driver..
- [7] V. Pérez-Gracia y A. Perez Gracia, "Deliverable 1.1. Robo Scan", Zenodo, jul. 2025. doi: 10.5281/zenodo.16575781.

APPENDICES

Appendix 1. Pseudocodes for mapping, navigation, and path planning.

Pseudocode for mapping

```
begin mapping

1. Point Cloud Acquisition
  subscribe to LiDAR pointcloud topic
  while receiving point cloud:
    store incoming 3D points in point cloud buffer

2. Voxel Map (OctoMap) Construction
  initialize empty OctoMap
  for each point in the point cloud:
    calculate ray from sensor origin to point
    for each voxel along the ray:
      mark voxel as free in OctoMap
      mark voxel at endpoint as occupied
  update OctoMap log-odds probabilities

3. Height Filtering
  for each voxel in OctoMap:
    if voxel.z < RangeMin OR voxel.z > RangeMax:
      ignore voxel
    else:
      retain voxel for 2D projection

4. 2D Grid Map Creation
  initialize empty 2D OccupancyGrid
  for each XY cell in grid:
    if any voxel in vertical column is occupied:
      set cell as occupied
    else if all voxels are free:
      set cell as free
    else:
      set cell as unknown

5. Grid Post-processing (Optional)
  apply inflation filter to expand obstacles by robot's safety margin
  apply hole-filling filter to close small gaps

end
```

Pseudocode for global navigation

```
1. Load global map
  Load previously built map (pose graph + 3D point cloud or voxel grid) into memory
  Associate each keyframe with its corresponding LiDAR point cloud scan

2. Receive new LiDAR scans
  As the robot moves:
    Receive a new LiDAR scan
    Pass the scan to the odometry node (rtabmap_odom)
```

Forward the scan to RTAB-Map core for localization

3. Perform localization via scan matching

For each new scan:

Attempt to match the current scan against the global map using:

- **ICP matching**: align scan to nearby local scans
- **Scan-to-map matching**: align scan to the global assembled cloud
- **Scan Context** (if enabled): detect loop closures via scan signature comparison

4. Estimate pose

Generate an initial pose guess using:

- ICP alignment to nearby keyframes
- IMU or odometry data as a prior, if available

Estimate the robot's current pose in the map frame based on alignment results

5. Do not expand map

In localization mode:

Update the robot's position estimate

Do not add new keyframes or scans to the map

6. Publish output

Publish the following topics:

- /map → /odom transform (tf)
- /rtabmap/localization_pose (PoseStamped)
- The robot's estimated global pose

Maintain the tf tree:

- map→odom transform published by RTAB-Map localization
- odom→base_link transform published by odometry node(s)

Pseudocode for global path planning

1. Initialization:

- The grid is converted into a **graph** where each cell is a node.
- A **cost map** (or potential field) is initialized. All cells start with infinite cost, except the start cell (or goal if reversed) which is set to 0.

2. Priority Queue:

- A min-priority queue (usually a binary heap) stores nodes to visit, prioritized by their current cost.

3. Expansion:

- The algorithm removes the node with the lowest cost from the queue.
- It examines the 4 or 8 neighbors (depending on connectivity).
- If a neighbor is **free space** and the new cost is lower, update the cost and add the neighbor to the queue.

4. Path Extraction:

- Once the goal is reached, the algorithm reconstructs the shortest path by **backtracking** from goal to start using the cost gradients.