Leveraging BIM Models for Skill-Specific Robot Navigation and Localization

Z.J.J. Ruijters (1239033), M. Macura (1810472), P.J van den Akker (1758160), A.H De Pauw (1877267)

***Abstract*—This paper describes the development of a system for autonomous mapping and navigation of indoor environments using a mobile robot. The system consists of three components: database, map generator, and robot control. The database component stores a digital twin of the building, which is used to generate maps, this digital twin is a graph database then gets updated by using objects detected by a robot making the database dynamic. The map generation component uses the digital twin to create different types of maps, such as line maps, point maps, and grid maps, which can be used for localization and navigation. Different from other research, each unique robot has unique maps created based on the robot's specific skills.**

The robot control component uses the generated maps to plan paths and avoid obstacles while navigating through the environment. Object detection is also implemented to update the database when new objects are detected. The system is tested using different IFC files and has shown promising results in generating accurate maps and navigating through the environment.

The significance of this research lies in its potential real-world applications. The approach in this study can improve the performance of robots, performing navigation and localization tasks within complex structures with different types of robots. This could be particularly useful in scenarios where robots are required to navigate unfamiliar or dynamically changing environments, such as barns. By these processes, this research contributes to the goal of enhancing the autonomy and effectiveness of robotic systems.

I. Introduction

The use of mobile robots has increased across a multitude of environments such as restaurants, warehouses or barns. This can be seen in companies such as Lely that provide different types of autonomous robotics solutions for smart farming. Because of the highly different types of environments and the more common use of multiple distinct robots in them, the current state-of-the-art localization and navigation methods such as SLAM (Simultaneous Localization And Mapping) are insufficient. SLAM, in some contexts, has disadvantages such as dynamic elements which can be added or removed from the environment, making the map inaccurate or the need for operators to manually operate the robot. To overcome these problems there is an alternative way of localization and navigation which is presented in this paper [1]. Here, data is queried from a building digital twin in the form of an RDF graph database which would improve robot navigation. However, this method still has some shortcomings such as that each robot that will navigate the environment will be provided the same map. The idea is to improve this method by generating robot-specific maps based on the specifications of the robot and the environment at hand. This will ultimately result in a pipeline that can be used for different robots and environments. These robot-specific maps will be useful when multiple robots have to navigate the same environment, the pipeline will create a specific map of the environment for each robot depending on their size and skills which is likely to affect the paths they will take from point a to b. Thanks to this method robots will be able to take more efficient paths and access areas that are uniquely suited to their skills. Furthermore, the digital twin can be updated by the robots when they sense object that are not on the map. This means that all the robots navigating the same building can update the digital twin and have access to the same data of the building which will get more accurate over time.

This paper is structured as follows: Is start with an introduction to the topic with related works, after which the project is divided into parts in the project decomposition. In the Methods section contains an comprehensive explanation in the methodology used in the development of the server, map generation, robot control, path planning and object detection. The results section outlines the outcomes of the research demonstrating the effectiveness and shortcomings of the solutions for the server, map generation, robot control, path planning and object detection. The discussion section provides an analysis of the results highlighting areas for improvement and potential challenge. It also discusses research findings and the suggests directions for further research. The conclusion summarizes the key findings of the research. After which the article ends with individual contributions of each team member.

II. Related work

This project is based on work done by Hendrikx et al. [2] [3] which explores a method to use semantic data retrieved from a BIM model for use by a robot-specific world model representation which a robot can then use to query semantic objects in its immediate surrounding. A graph-based approach is then used to localize the robot where it incorporates explicit-map features for localization. This results in a method where a robot can use to find routes to previously unexplored locations by using data from the BIM model.

Previous research has shown multiple ways of localizing with the use of BIM models. Acharya , Khoshelham and Winter. [4] use a Convolutional Neural Network(CNN) to extract features from synthetic images acquired from a BIM model, which are then used by the robot to locate its position. Laska [5] proposes a method also using a CNN but instead of predicting a specific location it is more coarse by predicting only zone/area where the robot is. This results in a faster training that is computationally cheaper to run.

Schaub et al. [6] use a method where a synthetic pointcloud is retrieved from the BIM model and then compared with the local pointcloud from the robot to find where the robot is located. This method does not require new training on each new building introduced.

Kim and Peavy [7] convert an entire BIM model into a dynamic URDF model. The result is a model where dynamic objects like doors and windows are movable and the robot can interact with them. This URDF model can then also be used for localization and navigation purposes.

For the navigation of the robot, the baseline used is taken from the de Koning et al. paper [8] which showcases robots navigating a map generated from BIM exported data.

III. Project Decomposition

In this section the project is divided into the individual parts to give an overview. Please refer to Figure 1 for an overview of the pipeline.

The pipeline begins with the creation of a digital twin, this digital twin is created from an Industry Foundation Classes (IFC) file. This is a standardized file format used in Building Information Modeling (BIM) to enable interoperability and exchange of data between different software applications and disciplines involved in the construction and building management processes.

From this IFC format a TTL (Terse RDF Triple Language) file is created, this file represents a graph database as in Figure 3. This file then updated to a database on a server. This server then contains the digital twin with semantic and geometry data of the building. Note that this server is not used for calculations, only for storage of the digital twin.

When the robot starts the load_map, it will use the linemap_server function to request the LiDAR sensor and robot height from the URDF. Using these heights the linemap_server will generate two maps, one at the height of the sensor, creating the localization map, and one volume rendering everything between the robot height and the floor, creating the navigation map. The localization map will then be used by the map_delta function to find the difference between the data given by the extrinsic sensors and the map created from the database. The localization map is send to the robots localization function and the navigation map is send to the path planner of the robot.

If any differences are detected then the update_map function is used to send a JSON query to the database with the added geometry and the robot will request a new map using the linemap_server function.

In order to properly show how each part is implemented and tested, the system is divided into three parts in the following sections. First the database and server are discussed, where it is shown how the pre-processing works and how the data is stored on the server. Then the map generation is discussed, showing how the maps are generated. Finally, the robot control is discussed where is shown how the robot localizes and navigates using the maps. This also discusses how the robot detects objects not on the map and updates the digital twin in the database.

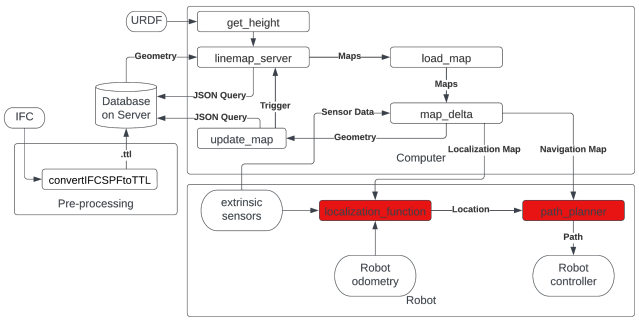

Fig. 1: Overview of the pipeline

IV. Methods

This project aims to develop a system for autonomous mapping and navigation of indoor environments using a mobile robot. The system consists of three components: database/server, map generation, and robot control. The database/server component stores a digital twin of the building, which is used to generate maps, this digital twin is a graph database then gets updated by using objects detected by a robot making the database dynamic. Importantly individual maps are created based on the physical characteristics of each individual robot.

To make this possible the following section discusses the creation of a database, the pre-processing of the model, the map generation, robot control, path planning and object detection.

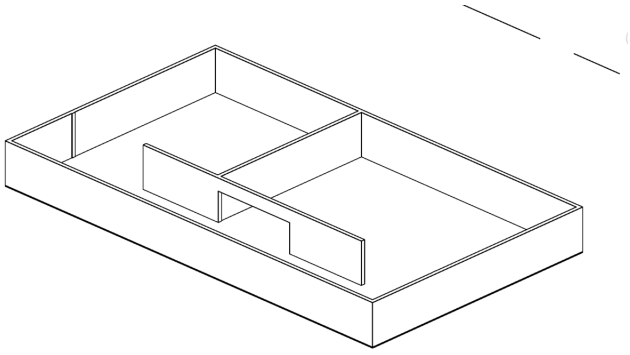

Fig. 2: The test environment for the demonstration (Revit model)

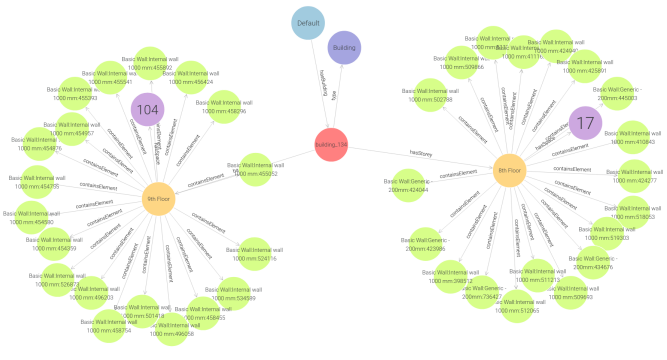

Fig. 3: Example of a database structure

A. Testing Environment

To test the performance of the robot and show the completion of the project, a test environment was set up to demonstrate the created pipeline working as intended. In Figure 2 the test environment is shown that will be used as a final test for this project. This environment consists of two rooms connected via a hallway. One of the rooms is connected to a hallway via either a tunnel or an opening. The tunnel is closer to the second room than the opening. This means that when a robot wants to navigate from one room to the other and can fit underneath the tunnel, it should take that path since it is the shortest. If the robot is too high to fit underneath the tunnel however, it should take the opening. Still, if both robots have their LiDAR sensor on the same height, they will need the same localization map, but different navigation maps. A localization map being a map used for the robot to find its position based on its sensor, while a navigation map is used for path planning. For this reason, this demonstration is created as a simple way of showing that the pipeline can generate maps tailored to a robot's specific skills while using the same building model. The results of using the pipeline on the test environment can be found at the end of the results section.

B. Database

Pre-processing: In order to have a database that can be queried from and updated, first the IFC file should be pre-processed into a TTL file. This TTL file can be imported as a graph database made up of nodes connected to each other with properties describing their relation to each other. In this way it describes the hierarchy of a building. See Figure 3 for an example of what such a database looks like. As a basis, the IFCtoLBD [9] script has been used. This script converts the IFC to a TTL file, containing the hierarchy and semantics of the building. However, for generating maps from the database, it needs to contain geometry as well. For this reason the script has been adapted to add geometry to each element node. This has been done by extracting the transformation, faces, vertices and edges of the elements in the IFC using the IfcOpenShell python library [10]. This data is then attached to the corresponding element node. The original data consist of an array of floats for each property (faces, edges, vertices and transformation), which is converted to a string for storage in the database. Overall, these properties together describe the polygons that the mesh is made out of The transformation describes the position of the object relative to the storeys origin point. The vertices describe the corners of the polygons which make up the element's mesh. These are given in groups of three describing the coordinates of the position relative to the point the transformation points to. The faces describe the surfaces of the polygons. These are given in groups of three vertices. These vertices connected create the given face. The edges describe the edges of the polygons and are also given in term of the vertices that are connected.

Server: In order to get data from the database, a method needs to be devised to query or add data. the database. To get the information from the database, a SELECT query has to be used, which selects specific portions of the database. This SELECT query can for instance be used to select all the elements of type wall. In addition there has to be an INSERT INTO query for adding a new element. This can be used to add new elements into the server when updating new elements.

An example of query for getting faces vertices and edges from all columns is:

```
PREFIX props: <https://w3id.org/props#>
SELECT ?faces ?verts ?T ?edges
WHERE {
?inst props:Category 'Column' .
?inst props:Verts ?verts .
?inst props:Faces ?faces .
?inst props:T ?T .
?inst props:Edges ?edges}
```

The server, written in programming language Go [11], is an agent that acts as the communication between the robot and database. The database is stored as an TTL format, to interact with the TTL format the GraphDB [12] tool is used. The purpose of the server is therefore exclusively for querying and updating the database. Therefore there are two APIs from which the robot can get data. Both are POST requests, where the JSON file is sent with the field "query". The first API is SELECT for getting the data from the database, the other API is UPDATE for updating the database. Server is there just for security and updating the database, so the server is only able to update the database and to get the data from it, but not to perform complex calculations.

C. Map Generation

Now that a database containing a digital twin of the building exists, it is needed to use this data to generate the maps needed for the robot. For this purpose scripts are developed to obtain the needed elements either directly from an IFC or from the database on the server and use the mesh of these elements to generate the required maps.

The maps are generated based on the geometry of the digital twin and the robot's specific skills. For localization maps, the geometry is sliced at the height of the robot's LiDAR, so that the map contains only the objects that it is able to sense. When making navigation maps, the geometry is sliced based on the height of the robot. This slice is not a 2D plane, but a volume, so it contains all the elements that would be in the robot's way when moving around the building. In order to make these slices, functions are needed that can fulfil this objective. As a basis for these functions, the python library Pymesh [13] is used. This library contains the function slice_mesh, which returns a given number of slices evenly distributed over the height of the mesh. This function has been adapted to create two new functions, slice_height and slice_volume. Both these functions work by creating a new mesh in the form of a box that goes from the bottom of the given mesh up to the height required for slicing. After this, it takes the Boolean intersection of the new box mesh and the given mesh. This gives a new mesh containing the parts that are present in both the box mesh and the given mesh, which is basically the given mesh up to the slice height given. Now, for slice_height, it takes the top

face of this mesh, which represents a 2D plane of the given mesh at the slice height. For slice_volume the complete mesh is returned resulting from the intersection, which contains all geometry of the given mesh up to the slice height.

Now that the necessary slice functions are in place it is possible to get the geometry information needed to generate the maps. This is done using three scripts; create_linemap.py, create_pointmap.py and create_gridmap.py. All these scripts contain one function to generate the maps directly from the IFC and one that generates the map from the server. These three scripts are all made to generate a different kind of map. Examples of the maps that these scripts generate are given in Figure 5. The line map is a general plot of the polygons of the elements, the point map is a scatter plot of coordinates that are generated between the vertices of the polygons of the elements and the grid map is a grid base map showing if an element is present in the given grid. Aside from the plot, the grid map also outputs a matrix with 1's and 0's describing the grids pass-ability. Parameters like the distance between points and grid size can all be changed accordingly. Note that these three different maps can all be generated using both slice methods, and can thus all be used for localization or navigation, depending on the robot's needs. A detailed description of the map generation process with algorithms in pseudo code is given below.

1) Query elements: First the required elements are queries from either the IFC directly or from the graph database on the server. For the IFC, this is done using IfcOpenShell to retrieve the wanted elements and generate the shape of each of the individual elements. These shapes can then be used to retrieve the transformation, faces, edges and vertices of the elements. For the algorithm, first all elements are obtained from the IFC per storey and checked if they are present in the element types queried by the user. This also allows for storing the relative elevation of the storey per element for determining the proper slicing height. Next, the geometry data is obtained from each of the loaded elements. Some elements, like stairs, consist of multiple "sub" geometries. This means that these elements should first be subdivided. The algorithm for this is shown in pseudo code in Algorithm **??**. For the server, these are obtained using SparQL queries which retrieves the transformation, faces, edges and vertices of the elements directly. The geometry data is already stored on the server for each element. This means that for each element type that needs to be queried the geometry can just be retrieved and used. Note that edges are not used for constructing the meshes later on. Even though faces and vertices are enough to reconstruct a mesh, the edges are stored on the server to have the complete geometry available. The algorithm for this is shown in Algorithm 2.

2) Generate meshes: Having obtained the needed geometry of the individual elements, the faces and vertices are used to generate meshes. These meshes are an object type native to the PyMesh library. The faces and vertices are stored per element in one long array, even though they belong in groups of three. For this reason, they first need to be grouped before they can be used to generate meshes. The algorithm for this is shown in Algorithm 3.

3) Slice meshes: Now that meshes of all queried elements are obtained, they are sliced using one of the two previously mentioned slice functions. If a localization map is needed, they are sliced using the slice_height function and if a navigation map is needed they are sliced using the slice_volume function. This will return the sliced mesh. The relative storey elevation is extracted from the given slice height here to make up for an element being on a different storey. The algorithm for this is shown in Algorithm 4.

4) Generate polygons: The geometry stored in the sliced meshes are now converted into polygons using the previously obtained transformations. These polygons are a data type native to the Shapely python library [14]. These are the polygons that together formed the meshes previously. Because the transformation is used in this process, these polygons also contain the necessary location information for plotting them into a map. First the new vertices and faces are obtained from the slices, secondly, the previously obtained transformations are applied on these slices vertices and faces to obtain the polygons. The algorithm for this is shown in Algorithm 5.

5) Obtain coordinates: Using the polygons that are obtained, x and y coordinates can be extracted from them. These coordinates describe the corner points of the polygons. the z coordinate is ignored, since the map needs to be flat. This automatically flattens the 3D volume mesh obtained from using the slice_volume function. The algorithm for this is shown in Algorithm 6.

6) Plotting map: The x and y coordinates of the polygons are now used to plot the map. For the line map a simple plot is used that automatically draws lines between these points. The algorithm for this is shown in Algorithm 7. For the point map, a script is used that generates coordinates for points between the polygon corners. The rate of points can be adjusted. These are then plotted as a scatter plot to show the map. The algorithm for this is shown in Algorithm 8. For the grid map, the coordinates obtained by the point generating script are used to see if a grid in the grid map is passable. First, a general grid is generated. After this, each grid is checked with the coordinate points established, and if there exist a coordinate inside the given grid it becomes obstructed. The algorithm for this is shown in Algorithm 9.

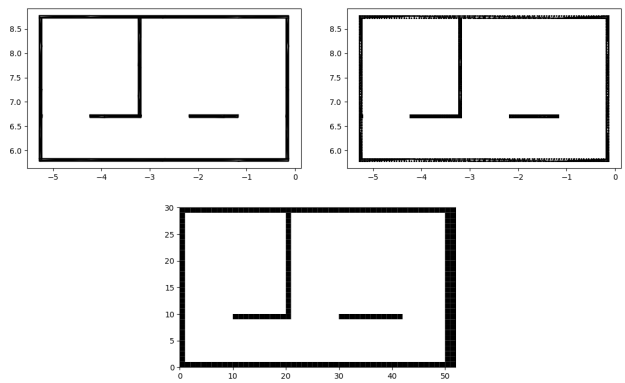

Fig. 4: Examples of generated images; left = linemap, right = pointmap, bottom = gridmap

Algorithm 1 Querying from IFC file

Input: location of IFC file and list of desired element types
Load IFC file
Obtain storeys from loaded IFC file
for All storeys **do**
    Obtain all elements of storey from loaded IFC file
    for All storey elements **do**
        if Element type is in desired element_types **then**
            Store element in loaded_elements
            Store storey elevation in element_elevations
        end if
    end for
end for
for All elements in loaded_elements **do**
    if Element has geometry **then**
        Obtain geometry from element
        Store geometry in faces, vertices, transformations
    end if
    if Element has no geometry **then**
        Obtain "sub" elements from decomposition of element
        Store sub elements in sub_elements
        for All elements in sub_elements **do**
            Obtain geometry from element
            Store geometry in arrays of faces, vertices, transformations
        end for
    end if
end for

Algorithm 2 Querying from server

Input: list of desired element types
for All element types in element_types **do**
    Query all elements of element type from server
    Store queried elements in loaded_elements
end for
for All elements in loaded_elements **do**
    Obtain geometry from element
    Store geometry in arrays of faces, vertices, transformations
end for

Algorithm 3 Generate meshes

for All element in loaded_elements **do**
    Get the element's vertices from vertices
    Get the element's faces from faces
    Divide the vertices in groups of 3
    Divide the faces in groups of 3
    Form mesh from grouped vertices and grouped faces
    Store the mesh in element_meshes
end for

Algorithm 4 Slice meshes

Input: height to slice and slice_mode (navigation/localization)
for All meshes in element_meshes **do**
    Slice the mesh using the slice_height or slice_volume functions at the correct height. The elevation stored in element_elevation is used here to compensate for relative height of storey.
    Store slices in element_slices
end for

Algorithm 5 Generate polygons

for All slices in element_slices **do**
    Obtain new vertices and faces from slices
    Get coordinates from obtained vertices and faces by using matrix multiplication with transformations
    Construct polygons using the obtained coordinates
    Store polygons in element_polygons
end for

Algorithm 6 Generate polygons

for All polygons in element_polygons **do**
    Obtain coordinates of corners from the polygon's exterior
    Store corner coordinates in element_coordinates
end for

Algorithm 7 Plot linemap

for All coordinates in element_coordinates **do**
    Plot the coordinate using a standard plot
end for

Algorithm 8 Plot pointmap

for All coordinates in element_coordinates **do**
    Generate new point coordinates between corner coordinates of polygons
    Plot the generated coordinates and point coordinates using a scatter plot
end for

Algorithm 9 Plot gridmap

for All coordinates in element_coordinates **do**
    Generate new point coordinates between corner coordinates of polygons
    Initialize empty gridmap based on maximum and minimum coordinates
    Use generated coordinates and corner coordinates to check and fill the gridmap with pass-ability
    Plot gridmap using a pseudocolor plot
end for

D. Robot Control

Localization is the process of finding the robots location and orientation based on a map and other sensor data. This project adopts the ROS built-in AMCL [15] localization method. This is a method based on the adaptive (or KLD-sampling) Monte Carlo localization approach. The implementation from ROS is based on the book Probabilistic Robotics by Sebastian Thrun [16]. To test this the ROSbot [17] from Husarion was used.

AMCL functions by making recursive Bayesian estimates based on the sensor data and the robot motion model. The model makes many guesses called particles on where the robot might be and updates them based on how well they match the data from the sensors. The more particles that agree with each other the more confident the location score.

AMCL has multiple advantages for this project according to [16]:

- It can handle noisy sensor data and dynamic environments.
- It can adapt the number of particles based on the uncertainty of the robot's pose.
- It can provide a measure of localization confidence by calculating entropy of the particle distribution.
- It is possible to integrate for different sensor types like: LiDAR, sonar and vision. Note that the AMCL implementation for ROS only supports LiDAR data.

There are also disadvantages to the AMCL method.

- It is possible that initial pose estimate is required. Especially on larger maps.
- It can be affected by map quality and sensor model.
- It can be computationally expensive.

Especially the initial pose estimate which would require knowing the general location on startup could be problematic. This would require either human supervision, a fixed startup location, or some additional localization method. This is not necessarily a problem during this phase of the project but may require additional consideration in the future.

To verify these claims and test the correct functioning of the AMCL method, the following tests will be performed: in the simple room described in Figure 2 in simulation, the robot will be given an initial position in a room 20 times, 10 of these times the initial position will be located somewhere in the room it is physically located, and 10 times the initial position will be given in the wrong room. The localization is considered successful if, within 10 seconds the robot can localize itself correctly.

However, the advantages mentioned in addition to the convenience of the build in ROS implementation makes this very suitable for this project. For more information on how this implementation works please refer to the ROS wiki.

In this project the AMCL node takes information from both the localization map generated from the map server and information from the odom node generated by the robot. This node gives estimated movement based on the wheel rotation and accelerometer. Combined with an general indication of where the robot starts this information can create a localization.

E. Path planning

Path planning is the process of calculating how a robot navigates through an known or unknown spaces while avoiding collisions. This project uses the built-in ROS move_base node. The ROS move_base node works by using a global planner to generate a plan that avoids static obstacles on a known map, and a local planner to create commands that follow the global plan while avoiding dynamic obstacles sensed by the onboard sensors. The move_base node is also capable of performing recovery, For instance, if the robot gets stuck, it can rotate in place to try and free itself. The path planning nodes takes in the position goal and the navigation map. The position goal is the destination that the robot aims to reach, and the navigation map represents the environment in which the robot can move. Using these inputs, the path planning nodes output a path for the robot to follow, along with velocity commands for the wheel motors to execute this path.

F. Object detection

To update the map the robot needs to detect objects in the real world that are not indicated on the map. This is done by using the following steps.

1) Map Retrieval: The method begins by retrieving the map from the server. This map represents the known environment in which the robot is operating. The map generation process is discussed in the Map Generation subsection. After retrieving the map the pixels are converted into 2D coordinates, these coordinates are created using a pixel to meter conversion defined by the user. The map is retrieved at the height of the sensor.
2) Data Acquisition: The robot receives data from the Lidar from the ROS laserscan topic. Which is transformed by the ROS package into a 2D point cloud consisting of X and Y coordinates. These coordinates and then rotated and translated to match the orientation of the map so that the map and the laserscan can be easily compared. The rotation and translation is retrieved from the transform topic provided by ROS.
3) Confidence Check: The method checks the confidence of the Adaptive Monte Carlo Localization (AMCL) model. If the confidence is below a certain threshold the process will restart until the confidence reaches the threshold.
4) Distance Calculation: The method calculates the distance from each point on the laserscanner to the nearest point on the map. If a laserscan point does not have a neighbour from the map, it is considered not on the map, indicating the presence of a new object. This distance is user defined
5) Clustering: If there are multiple new objects, the method uses a clustering algorithm to group the points, creating multiple individual objects. The clustering algorithm used is the fcluster function from the SciPy library, which is an implementation of the Fast optimal leaf ordering for hierarchical clustering [18].
6) Bounding Boxes: The method then creates bounding boxes around the objects. This involves drawing a square around each detected object, which helps in visualizing

and tracking the objects. Bounding boxes with a pixel count under user a defined threshold are considered noise and discarded. This gives the user a trade off where the variable can be set high, which will result in less noise, or the variable can be set low, which will result in more noise but allows for detection of smaller objects.

7) Server Update: Finally, the corner points of these squares (representing the detected objects) are sent to the server for updating the map.

This method was chosen for implementation due to its relative simplicity and efficiency, which aligns well with the project's requirements.

V. Results

A. Map Generation

The map generation scripts should be consistent across different IFC files in order to be considered functional. For that reason the scripts have been tested using multiple IFC files to show that it can properly generate the maps. The IFC files that have been used for testing the scripts can be seen in Figure 5 including the IFCs of the 8th and 9th floor of Atlas, the IFC of the test setup and a random room consisting of elements with different heights.

First, in Figure 8, two line maps of the Atlas building can be seen. These two maps are sliced using the slice_height function, so they only show the geometry that is present at the given height. One of the maps is sliced at a height of 1 meter and shows the 8th floor of Atlas and the other one is sliced at a height of 5 meters and shows the 9th floor. Since the slice height is relative to the bottom of the 8th floor, it shows that it properly slices the geometry and plots the correct elements.

In Figure 6 two grid maps of the test setup can be found. These are sliced using the slice_volume functions, and thus contain all geometry from the bottom up to the given height.

Algorithm 10 Object detection

```
while Laserscan received do
    if Localization confidence is high enough then
        Given the robot sensor height retrieve the map
        Calculate distances from the map points to the laserscan points
        Transform the map coordinates to correspond with the laserscan coordinate system
        Determine which laserscan points do not have a map point within a user defined threshold distance
        Cluster these points
        Discard clusters that a number of points below a threshold
        if There exist any clusters then
            Determine the maximum X and Y coordinates of each cluster to determine a bounding box
            Send the bounding box coordinates to the server
        end if
    end if
end while
```

Now, to clearly show the slice functions correctly obtain the correct geometry from the meshes, maps of the random room using both the slice_height and slice_volume functions can be seen in Figure 9 and Figure 10 respectively. Both figures contain four plots showing the map sliced at the same four heights. This also clearly displays the difference between the two slice functions and how they can be used to generate both localization and navigation maps using the three different map forms. As can be seen on these plots, they perfectly correspond to the IFC of the random room shown before, confirming that the map generation works.

Lastly, in Figure 7, two of the same maps generated using the random map IFC are displayed. The left one is generated from the server, while the right one is generated directly from the IFC. This shows that the geometry remains correct after storage on the server, and that the server generated maps are consistent with generating them directly from the IFC.

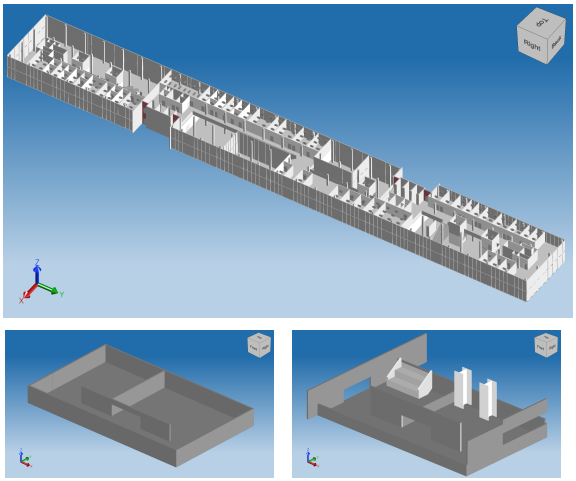

Fig. 5: Example IFCs; top = Atlas floor 8 and 9, left = test setup, right = random map

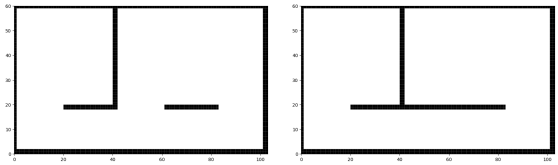

Fig. 6: Gridmaps of the test setup; left = height 0.34m, right = height 0.36m

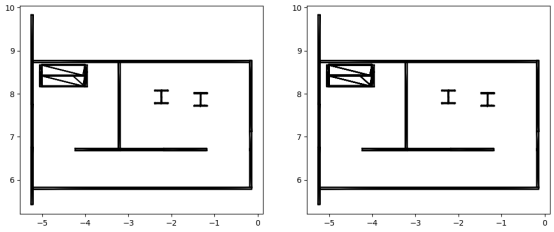

Fig. 7: Linemaps of the random map; left = generated from server, right = generated from IFC

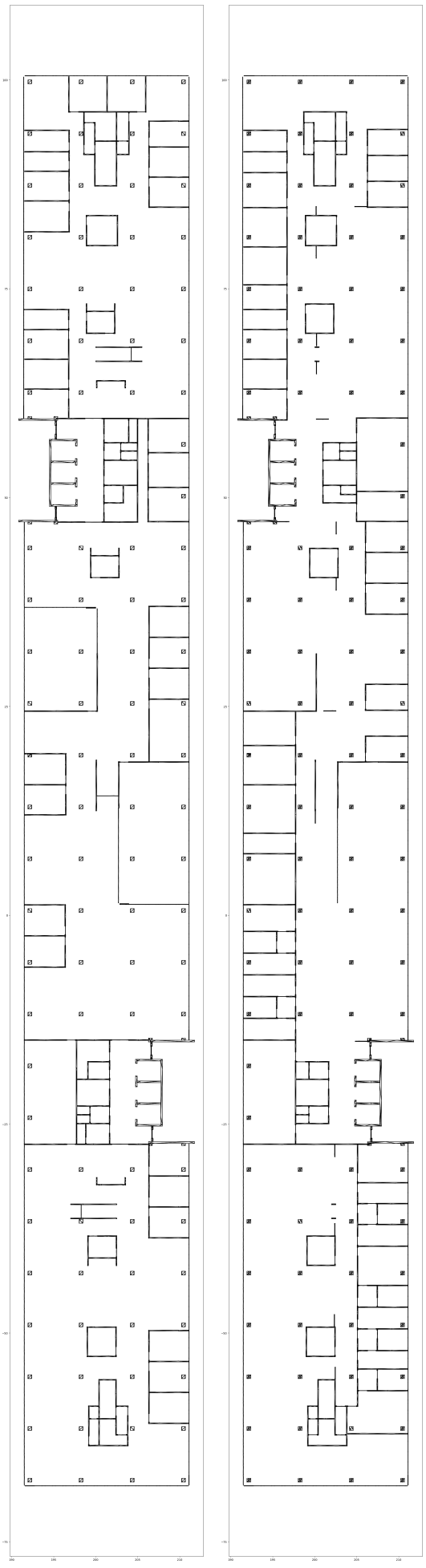

Fig. 8: Linemaps of Atlas; left = height 1m (floor 8), right = height 5m (floor 9)

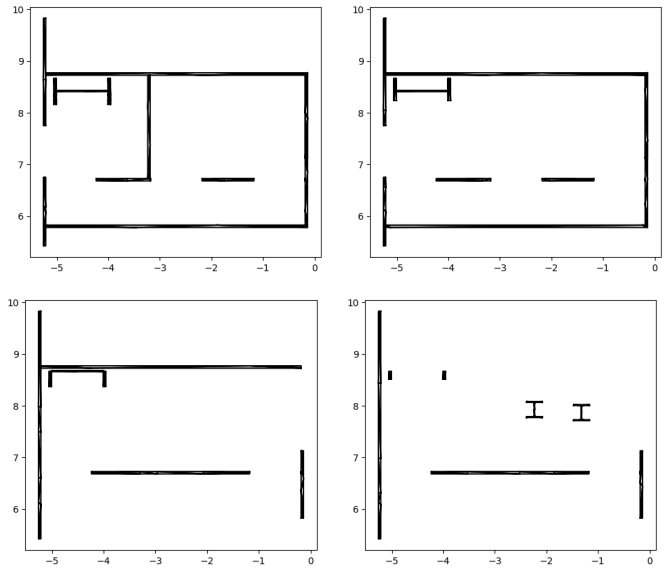

Fig. 9: Line maps of the random room using slice_height; Top left = 0.2m, top right = 0.3m, bottom left = 0.4m and bottom right = 0.5m

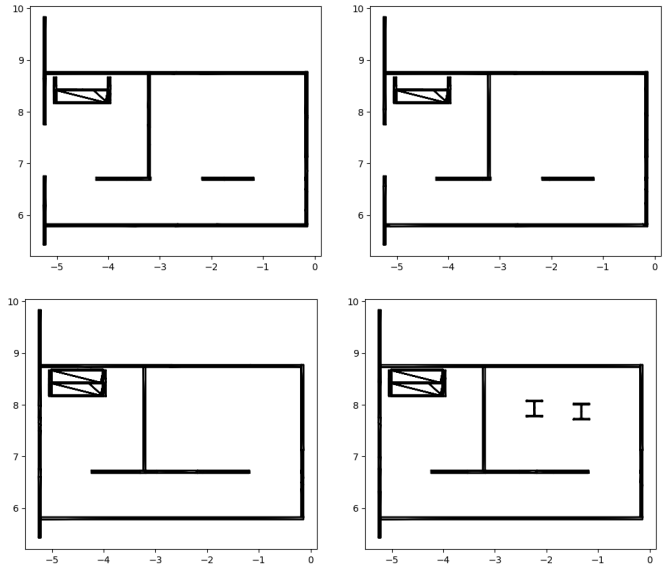

Fig. 10: Line maps of the random room using slice_volume; Top left = 0.2m, top right = 0.3m, bottom left = 0.4m and bottom right = 0.5m

B. Robot Control

The outcomes of this part of the project were derived from a set of operational testing and observations. Through these tests, it was established that a robot could localize itself effectively within a distinct room. However, when this scenario was introduced to the simple room used for demonstration purposes, a few challenges surfaced. The uniformity of the room, its lack of distinctive characteristics, and similarity to different rooms, made it difficult for the robot to localize.

To quantify these results a simple test was done: The robot, in simulation, was given an initial position in a room 20 times,

10 of these times the initial position was in the room it was physically located and 10 times the initial position was given in the wrong room. The localization was considered successful if within 10 seconds the robot could localize itself correctly. These tests result in the following: A robot given the correct room as initial positions would localize 10 out of 10 times. If the robot was given the wrong room the robot would correctly localize 6 out 10 times. It is noteworthy that two of these times the robot did find the correct room, but did not find the correct orientation.

Despite these challenges, it was observed that the robot could still localize in the presence of objects not accounted for in the map. This was determined by using the same tests as done previously. The robot was times given the initial position somewhere randomly in the correct room 10 times, and 10 times somewhere randomly in a different room, this time anywhere 1 and 3 columns were added that were not present on the provided map. This resulted in the following: when the robot was located in the same room as the given initial position the robot localized correctly 9 out of 10 times. When not given the correct room the robot localized correctly 5 out of 10 times. This observation shows the flexibility and robustness of the adaptive Monte Carlo localization (AMCL) method. Such a result further demonstrates the method's adaptability in considering unexpected elements or changes within the environment. However it also highlights the need for the robot to be given a correct initial position.

When considering path planning, the robot successfully employed map data, enabling it to determine its path in parts of the environment it hadn't directly observed, as can be seen in Figure 11 and Figure 12. This capability, resulting from the integration of the map data within the robot control, affords the robot the ability to plan using data beyond its immediate sensory range.

Furthermore, the robot showcased adaptive navigation skills by manoeuvring around objects not on the map, as in Figure 12. This skill shows the robot's ability to successfully navigate by merging real-time sensory input with the pre-existing map data, even when presented with unforeseen obstacles. Taken together, these findings indicate that the AMCL method of localization together with the move_base function from ROS are suitable for this application.

C. Object detection

The object detection system demonstrated satisfactory performance, with the ability to from an object not on the map (Figure 13), to detect that object (Figure 14), to update the map (Figure 15).

This was tested by first having the robot localize, then placing zero to three objects in the simulation space that are not on the map. And then running the object detection. Out of 10 tests the algorithm managed to detect all columns not on the map.

The algorithm has been shown to not detect objects when too close to an object depicted on the map. Also when two or more objects are close together, depending on the set threshold, can be detected as one object.

It turned out challenging to automatically ascertain whether the AMCL had localized correctly. An attempt was made to retrieve the variance of the AMCL model as a measure of localization accuracy. However, this approach proved to be unpredictable and varied significantly from map to map, and attempt to attempt, making it unreliable as a standalone measure of localization accuracy. As a temporary solution, a simpler method was implemented to that the system had localized. This method involved checking whether a certain percentage of the laserscan overlapped with the map. This approach proved to be effective enough for the current stage of the project, but it is not a long-term solution due to its simplicity and lack of robustness. Another noteworthy aspect of this method is the number of parameters that require tuning. These parameters include the minimum number of coordinates an object should have to not be considered noise, the minimum distance from a laserscan point to the closest map point, and the minimum distance between clusters to be considered distinct clusters. While the method works well and is relatively simple, it does require a significant amount of tuning. Therefore, future work may need to explore alternative methods that are less sensitive to parameter tuning. The results indicate that the object detection system's performance is highly dependent on the accuracy of the AMCL localization. Despite these challenges, the object detection method has shown promise in its current form. It has demonstrated the ability to detect objects effectively given that the AMCL localization has converged to the correct location. This suggests that with further refinement and development, this system could be suitable for simple object detection tasks.

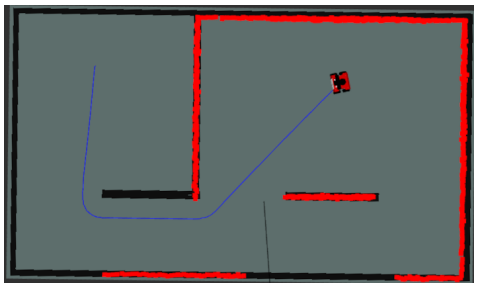

Fig. 11: The path taken by the robot given no object, where blue is the path, and red is what the robot Lidar detects

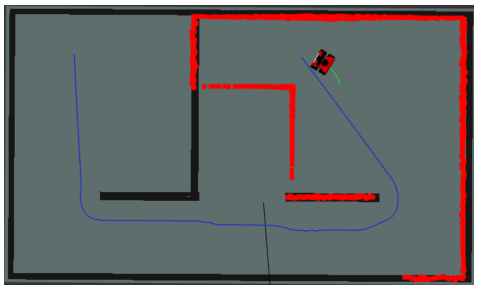

Fig. 12: The path taken by the robot when there is an object that is not on the map blocks the path, where blue is the path, and red is what the robot Lidar detects

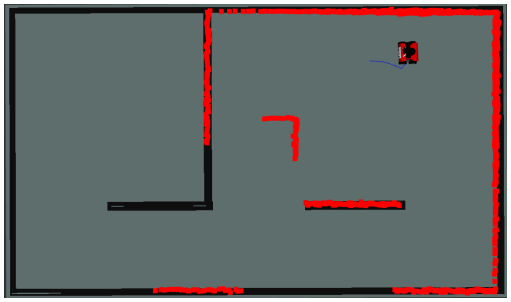

Fig. 13: Robot after having converged on its location, note that there is a pillar that is detected by the robot, but is not on the map

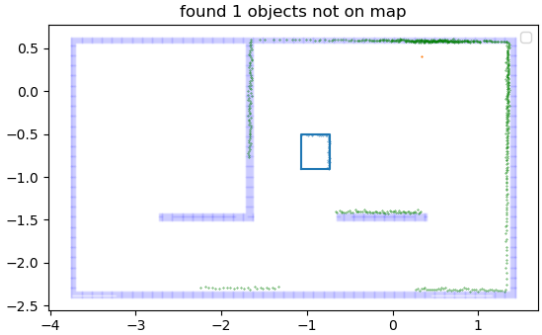

Fig. 14: The robot has detected the pillar shown in figure 13

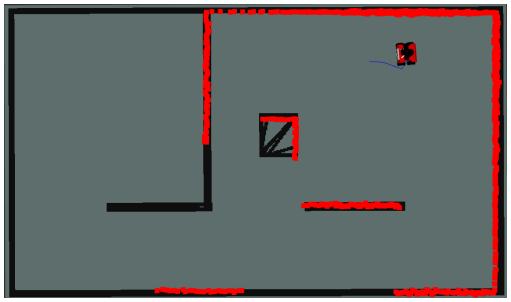

Fig. 15: The map has been updated now including the column

D. Demonstration

Now that the correct functionality of all the sub parts has been confirmed, it is time to bring everything together and see if the individual components can work together. To demonstrate this, as explained earlier, a test setup has been made that require robots with different physical dimensions to take a different path, even though their Lidar will see the same. for this the 3d geometry of the test setup space Figure 2 will be used. In Figure 16 the robot is 0.2m while the door in the middle is 0.35 m high, this means that the robot can fit under the door, allowing for a short path. However if the robot were bigger, but the LiDar at the same height, as in Figure 17. The robot would not be able to detect the obstruction with the sensor however, it would not be able to pass through the door. This means that the navigation map (where the robot can go) has to be different than the localization map (what the robot can see). This results in the longer path seen in

Figure 17. Illustrating that a robots with different dimensions have different paths. Note that both the sensor-height and the robot-height is retrieved from the URDF file automatically. This demonstration shows that the pipeline functions correctly and can generate different localization and navigation maps depending on the robot's specific skills using the same building model.

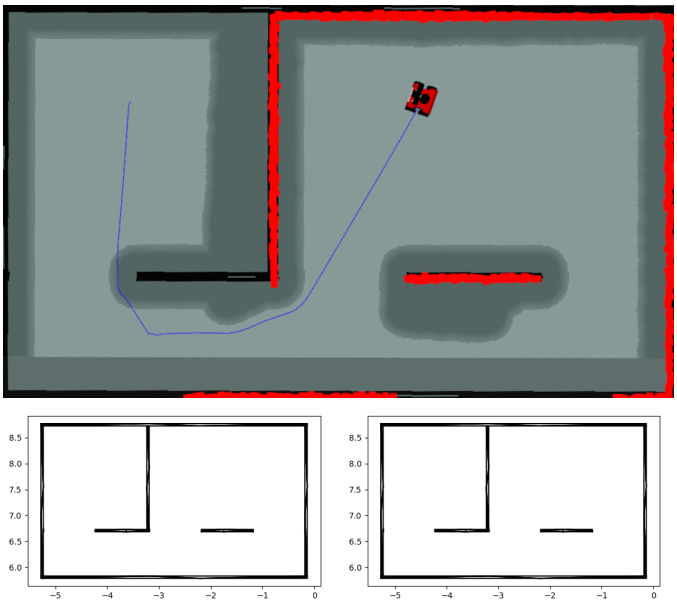

Fig. 16: A robot that is 0.2m high will have the same navigation map (bottom left) as localization map (bottom right), this allows the robot to take a shorter route (top image)

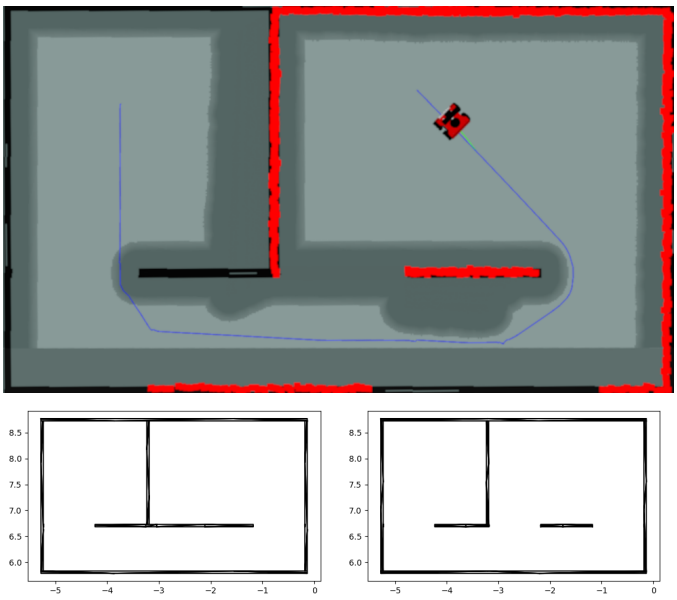

Fig. 17: A robot that is 0.6m high will have a different navigation map (bottom left) than localization map (bottom right), this forces the robot to take a longer route (top image)

VI. Discussion

In this section, several areas for improvement are highlighted in the object detection system, technology choices, feedback implementation, localization, and graph database design. The feedback suggests exploring alternative detection methods, upgrading to a newer version of ROS, considering computation distribution between the robot and server, improving localization convergence certainty, redesigning the graph database structure, and addressing issues encountered when running the system on a physical robot. These suggestions aim to enhance the overall performance and reliability of the system.

Object detection improvements

While the object detection works well for the current use case, a more holistic approach may be warranted. Different detection methods such as using vision algorithms such as YOLO [19] or a RNN network such as [20] may gain better performance for object detection tasks. This could solve the problems of objects that are close together getting classified as one object, of objects close to the wall not getting recognized. Another note is that the object recognition is imperfect, because a certain tolerance in necessary for localization noise or inconsistencies. This means that if the localization is offset in some way, or the laserscan data is noisy, this will also be reflected in the updated map.

Outdated technology

In this project, ROS1 is used which is not the most recent version of ROS, it was originally thought that this version was more stable. However it turns out that it uses software like python 2 which is outdated, it doesn't allow access to some of the newer packages, forces use of outdated packages which may include security flaws. Therefore it might be advisable to switch to a newer version in the future.

Computation and Communication Strategy

Currently the robot sends it's odometry and laserscan information to a local computer, does the calculations and communication with the server, send the updated data to the server and sends the path back to the robot. While this works well while there is a strong Wi-Fi connection, it may be more advantageous to do the computations directly on the robot and occasionally send and receive information from a server. This would decrease the dependence on a strong Wi-Fi connection but would increase the computation and energy cost of the robot. This choice would have to be considered based on the application.

Localization improvements

A large problem for this project is the question of when the localization is accurate. This is important because if the localization is not converged, or is inaccurate, it is possible that the database would be updated incorrectly. This could result in not making the map navigable for robots, or localization not working anymore.

Graph database design improvement

As it is shown on Figure 3, there is one main node which is connected to the nodes that represent floors of the building. Element nodes are connected to the floor node and the nodes that represent the geometry of the elements are connected to their element node. This design can be updated so the elements that are physically connected are connected in the database as well. In that way, database would have more information.

Issues when trying to run on the physical robot

As a final demonstration, the created pipeline was tested on a physical robot. This however did not go as expected as some previously unseen problems appeared when trying to run the physical system. The main problem faced was that the robot's odometry data was telling ROS it was going the opposite way than the one it was actually going in resulting in localization problems. This could have been caused by the lidar being mounted backward or other underlying issues with the physical system. Therefore, only simulation results are reported and conclusive.

Possible improvements for automation

While the map generation and updating is automated, to achieve this many parameters that have to be manually adjusted, these include:

- Pixel per meter that is necessary for map generation, this could be automatically calculated
- Offset for the server map for updating the database, where the coordinates from the robot are translated to the coordinates of the server
- Any of the variables in the object detection part where, minimum distance from objects to be considered distinct, distance from map to be considered a new object and more are manually defined.

A. Updating the database over time

Currently, when an object is detected this will be updated to the database and stay there forever. However, for an actual implementation there should be a mechanism for decay. Meaning that when an object is detected once, but is not there for a subsequent detection it should be removed from the map. This is would result in dynamic objects such as chairs or people automatically being removed when no longer present.

VII. Conclusion

In this paper the idea of localizing and navigating a robot in a building environment with only the digital twin of the building has been presented. The pipeline that has been designed and discussed innovates over common methods such as SLAM by using data of the environment that was already available. It has been shown that the developed map generation scripts can create proper localization and navigation maps based on a robot's specific skills using a digital twin. Simulations show that using ROS, these maps can be used to localize and navigate. One interesting aspect of this approach is that the

digital twin in the database can be updated when a robot senses something that is not coherent with the map they received. In this way, even though the initial building model may not be up to date or different than the actual building that has been build, the digital twin can become more and more accurate of the real scenario. If multiple robots are present in the same building, this also means that when they generate new maps from the database, the information gathered by other robots are known to them as well. It has been shown that through simulation it is possible to sense new objects and add them to the already existing database. In addition it was shown that functions correctly and can generate different localization and navigation maps depending on the robot's specific skills using the same building model. Meaning that different sized on skilled robots can navigate in different ways.

Future pROSpects for this project include running more on the server and even less on the robots, improving the object detection and the localization of the robot. Furthermore, the updating process could be enhanced so that existing geometry can be changed and work could be done to have robots detect and specify what kind of object they sense. Also, the project could be moved from simulation to real robot testing.

Overall, in this paper, a new approach is presented that could make robot localization and navigation better by using an already widely available resource for building environments and let multiple robots with specific skills and specifications function from one database that can be updated and refined. This work can find a lot of various implementations in very different real-world environments.

REFERENCES

[1] R. de Koning, E. Torta, P. Pauwels, R. Hendrikx, and M. J. G. van de Molengraft, "Queries on semantic building digital twins for robot navigation," in *9th Linked Data in Architecture and Construction Workshop*, vol. 3081, 2021, pp. 32–42.

[2] R. W. M. Hendrikx, P. Pauwels, E. Torta, H. J. Bruyninckx, and M. J. G. van de Molengraft, "Connecting semantic building information models and robotics: An application to 2d lidar-based localization," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 11 654–11 660.

[3] ——, "Connecting semantic building information models and robotics: An application to 2d lidar-based localization," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 11 654–11 660.

[4] D. Acharya, K. Khoshelham, and S. Winter, "Bim-posenet: Indoor camera localisation using a 3d indoor model and deep learning from synthetic images," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 150, pp. 245–258, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924271619300589

[5] M. Laska and J. Blankenbach, "Deeplocbim: Learning indoor area localization guided by digital building models," *IEEE Internet of Things Journal*, vol. 9, no. 16, pp. 15 323–15 335, 2022.

[6] L. Schaub, I. Podkosova, C. Schönauer, and H. Kaufmann, "Point cloud to bim registration for robot localization and augmented reality," in *2022 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, 2022, pp. 77–84.

[7] K. Kim and M. Peavy, "Bim-based semantic building world modeling for robot task planning and execution in built environments," *Automation in Construction*, vol. 138, p. 104247, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580522001200

[8] R. de Koning, E. Torta, P. Pauwels, R. Hendrikx, and M. van de Molengraft, "Queries on semantic building digital twins for robot navigation," in *9th Linked Data in Architecture and Construction Workshop*, ser. CEUR Workshop Proceedings. CEUR-WS.org, 2021, pp. 32–42, 9th Linked Data in Architecture and Construction Workshop, LDAC 2021, LDAC ; Conference date: 11-10-2021 Through 15-10-2021. [Online]. Available: https://www.cibw78-ldac-2021.lu/,http://linkedbuildingdata.net/ldac2021/

[9] P. Pauwel, "Gitlab." [Online]. Available: https://github.com/ISBE-TUe/IFCtoLBD

[10] "Ifcopenshell." [Online]. Available: https://ifcopenshell.org/

[11] A. A. Donovan and B. W. Kernighan, "The go programming language," 2015, accessed on June 12th, 2023. [Online]. Available: https://golang.org

[12] Ontotext, "Graphdb," 2023.

[13] "Pymesh." [Online]. Available: https://pymesh.readthedocs.io/en/latest/

[14] "Shapely." [Online]. Available: https://pypi.org/project/shapely/

[15] B. P. Gerkey, "Wiki." [Online]. Available: http://wiki.ros.org/amcl

[16] S. Parsons, "Probabilistic robotics by sebastian thrun, wolfram burgard and dieter fox, mit press, 647 pp., $55.00, isbn 0-262-20162-3," *The Knowledge Engineering Review*, vol. 21, no. 3, pp. 287–289, 2006.

[17] Husarion, "Rosbot," 2023, accessed on June 12th, 2023. [Online]. Available: https://husarion.com/manuals/rosbot

[18] Z. Bar-Joseph, D. K. Gifford, and T. S. Jaakkola, "Fast optimal leaf ordering for hierarchical clustering ," *Bioinformatics*, vol. 17, no. suppl_1, pp. S22–S29, 06 2001. [Online]. Available: https://doi.org/10.1093/bioinformatics/17.suppl_1.S22

[19] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. [Online]. Available: http://arxiv.org/abs/1506.02640

[20] A. Mujika, F. Meier, and A. Steger, "Fast-slow recurrent neural networks," *CoRR*, vol. abs/1705.08639, 2017. [Online]. Available: http://arxiv.org/abs/1705.08639

APPENDIX

Appendix I Individual contributions

Zeph Ruijters: For the duration of the project, I have worked on a couple of different aspects. I was assigned to work on the map generation at the beginning of the project, so that is where my focus was. A general overview of the things I did can be found below.

Adapt the PyMesh slice function to our needs. The first thing I did was changing the slice function that is present in the PyMesh python package so that it can slice meshes in a way that is useful for the map generation. This resulted in the two functions that I made; slice_height and slice_volume.

Make the map generation scripts. Since the map generation was the main objective assigned to me at the beginning of the project, this was were most of my time was spend on. I created and wrote the create_linemap.py, create_pointmap.py and create_gridmap.py scripts, which can generate the maps directly from an IFC file or server containing a graph database digital twin.

Adapt the IFC to TTL pre-processing script to have the database contain the building's geometry. In order to fill the graph database with geometry, the provided IFCtoLBD script needed to be adapted so that each element contains the necessary information to create a mesh when the data is queried. I changed the script in such a way that it fills the TTL file with the geometry and made the new pre-processing script used to initalize the server.

Figure out how to query the necessary geometry from and to the server for map generation and updating and adapt the map generation scripts to use the query result of the server. With the server and API in place, I needed to be able to obtain the geometry from the server and be able to add new geometry to the server for later use. For this reason I learned the basics of SparQL and made queries that can fulfil this objective. Furthermore, I adapted the map generation

scripts to use the geometry in this new form to generate the maps.

Integrate the map generation and server updating with the robot control work in ROS. With both the map generation and robot localization/navigation working, the two parts should be able to work together so that the robot can generate maps when needed and update the database when it encounters something. I worked together with Pim to combine our works into one cooperating structure in ROS.

Work on project proposal, midterm report and final report. For the project proposal, I worked on several sections of the report. Firstly, I wrote the introduction at the beginning of the proposal. Furthermore, I read and made several summaries of different pieces of literature connected to our project. These were for use in the related works section. I also made the basis for the project decomposition by writing the system section and making the diagram of the architectural overview. I also wrote the first version of the milestones, actions and scenarios and established the first version of the tests. These were later changed and improves by Alex and Marko. Lastly, I wrote a part of the team planning section.

I also worked on several sections of the midterm report and the midterm presentation, besides frequently updating the architectural overview during the project, I wrote a more detailed system description for the project decomposition. I also updated the project objective to match our changed perspective on the project after the weeks of work. I also worked on the methods section where I wrote the pre-processing and map creation parts describing in more detail how the system pipeline will function. Lastly, for the midterm report, I revised the planning in the revised planning section and I also described our view for the final demonstration here. I also worked on the midterm presentation by making and fixing a part of the slides.

For the final report, I wrote the methods and results sections of the map generation. Furthermore, the pre-processing part of the database methods and parts of the conclusion.

Document my work on the Gitlab. Finally, I documented my work on the Gitlab, so that people using it in the future will be able to understand and further adapt it.

Alexander De Pauw: I have worked on multiple aspects of this project over its entire duration. I was assigned to the robot control part of the project for which I did a lot of robot testing, I dealt with the design and creation of the different test setups in Revit, and I also managed the GitLab repository to bring it up to date and finally made presentations and contributed to the written midterm and final reports.

Running the pipeline on the physical system. For this task, I went to the robotics lab to test our simulation results on the real system. This involved recreating the setup seen in Figure 2 with real blocks in the lab and letting the ROSbot navigate it. As mentioned in the discussion section, despite our many attempts, the software and the physical systems ran into some issues. The software was having problems updating the localization map making the ROSbot crash into walls, which has been fixed. The hardware problems involved the odometry data thinking it was going the opposite way from where it was actually going, essentially making the ROSbot think it was going backward when it was moving forward. I tried to solve this problem by configuring the ROSbot as the makers (Husarion) intended it but because of lack of time and this unforeseen issue, this problem wasn't fixed.

Creation of the test setup in Revit. Creating the test setup is the first step in the pipeline. The IFC of the created setup is used with the URDF information to create the different maps the robot needs to localize and navigate. I started with creating the base setup which can be seen in Figure 2 which was meant to prove the working of our pipeline at a very easy and fundamental level. For this, I had to learn how Revit works and spend some time understanding the mechanism behind it and modifying the families of objects to create customized elements such as small door openings. Then at a later point, I worked on more intricate environments for the robot to navigate such as the one in **??**. This more complex environment was created to test our pipeline and generate a lot of results with different robots of different heights.

Keeping the Gitlab up to date and making the general README file. I was put in charge of putting the Gitlab up to date and keeping it that way. I also made the general README file that gives an overview of the entire project structure and code structure.

Contribution to the written documents My main contributions to the written papers included the introduction (midterm), the project objectives (proposal), the related work section (midterm), the project decomposition in which I redefined the milestones, actions and tests to be taken (proposal) and lastly I worked on the results for the robot control of the midterm paper. For the final paper, my contributions were the environment testing in the methods, the discussion, and reviewing the conclusion.

Pim van den Akker: During the project I worked mainly on the robotics part of the project, specifically on the ROS and object detection parts.

learning and using ROS During the development phase, I focused on learning and using ROS (Robot Operating System), a framework for developing robot software. I installed and learned to work with ROS and Gazebo, a simulation tool that integrates with ROS. I also learned how to convert a png map into a map_server that ROS can use. In addition, I learned how to write and run .launch files that launch multiple nodes at once, how to use gmapping, an implementation of SLAM (Simultaneous Localization And Mapping) algorithm that builds a 2D map from laser data and odometry information, and how to run python scripts using rospy. I have also learned how to use AMCL (Adaptive Monte Carlo Localization), a probabilistic localization system that uses a map and laser scans to estimate the pose of a robot.

Object detection and ROS The latter half of the year my focus mainly lay on the integrating ROS, object detection, and fixing the many, many issues cropping up from using so many different pieces of software. A large part of my work was on creating the object detection system.

Various techniques I also attempted to use Docker ROS, a containerized version of ROS that runs on any platform without installation. However, I faced some technical difficulties

and compatibility issues that prevented me from successfully using it. Therefore, I decided to dual boot my PC with Ubuntu in order to run ROS natively. During this I learned a lot not only about robot control but also more general skills such as working with command prompt, other operating systems, graph databases and more.

Report In the process of writing this report and the research proposal I also developed my skills in literature review and report writing. In this report I contributed to the methods and results of robotics and object detection, introduction, summary, conclusion and the related works section.

Integration and interdisciplinary I was also responsible together with the help other team members integrate everything, which includes running the server, understanding the graph database, and running ROS at the same time. This really added to my interdisciplinary, giving me at least a understanding of all the different aspects of this project including server and robot control. Enough to communicate with domain experts about it.

documentation I also wrote the documentation of the ROS part and the main readme in the gitlab and helped test other documentation.

Marko Macura: As a member of the software team, I mostly contributed to that part. Mostly I contributed in making the server, but I also contributed in the other aspects of the project. I have helped the robotics part of the project with the setup, I extracted the information about the robot, and updated the GitLab documentation and documentation of the functions I wrote. There are also some ideas that did not come to realization. There are some ideas with the Docker image, for easier setup of everything, which came to issues that could not be to resolve, as well as trying to make the map generation to work in C++ for the better performance.

Help in setup the environment. I helped into the setup of the Gazebo environment. It is difficult to keep everything working, so I helped to the robotics team in that.

Contribution in making a database. The algorithm is used for making a TTL file from IFC [9]. The algorithm had some issues, which I resolved and made it possible to work with.

Extracting information about the robot. I made a scripts that get the height of the robot, as well as getting the height of the LiDAR sensor from the URDF of the robot. Because of the structure of the URDF file and connection between joints and links and other elements of the URDF, this was not easy work. Every element has a parent element and the position of the element is relative to the parent element, so this has to be recursively done until the program reaches the base element.

Making the server. My biggest contribution to the project is making the server that has a communication with the robot and database. The team decided to use Go as a language for server. For this project I had to understand what is API and how to make it. Also I had to understand the basics of the networks. Overall this part took me a lot of time. I learned a lot about Go and how to implement backend of the application in that language. I made POST request for selecting the data from the database as well as a POST request for updating the database. Both are very well documented on the GitLab.

I have documented my work on GitLab. I was keeping everything up to date, so I believe it is very clear and that people can see how my work is done and use it properly.

I have contributed to writing the reports.