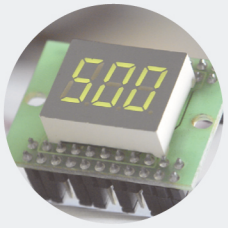
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Mapas Topológicos para Exploração Robótica de Minas Subterrâneas Submersas

PEDRO MIGUEL SERRÃO DA VEIGA MARTINS
novembro de 2022

POLITÉCNICO
DO PORTO

Politécnico do Porto
Instituto Superior de Engenharia do Porto

Topologic Maps for Robotic Exploration of Underground Flooded Mines

Pedro Martins

Master in Electrical and Computer Engineering
Specialization Area of Autonomous Systems

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

November, 2022

This dissertation partially satisfies the requirements of the Thesis/Dissertation course of the program Master in Electrical and Computer Engineering, Specialization Area of Autonomous Systems.

Candidate: Pedro Martins, No. 1191201, 1191201@isep.ipp.pt

Scientific Guidance: Alfredo Martins, aom@isep.ipp.pt

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

November, 2022

For my parents, Hélio and Anabela, and for my brother Carlos.

Abstract

The mapping of confined environments in mobile robotics is traditionally tackled in dense occupancy maps, requiring large amounts of storage. For some use cases, such as the exploration of flooded mines, the use of dense maps in processing slow down processes like path generation. I introduce a method of generating topological maps in constrained spaces such as mines. By taking a structure with fewer points, traversal and storage of explored space can be made more efficient, avoiding complex graphs generated by methods like RRT and it's variants. It's simpler structure also allows for more intuitive human-machine interactions with it's fewer points. I also introduce an autonomous frontier-based exploration approach to generate the topological map during exploration, taking advantage of it's traversal to navigate through known space. With this work, simulation tests show it is possible to successfully extract a simpler graph structure describing the topology during autonomous exploration and that this structure is robust through explored regions.

Keywords: Topological maps, exploration, mobile robotics, autonomous systems, underground.

Resumo

O mapeamento de ambientes confinados em robótica móvel, é tradicionalmente abordado em mapas densos de ocupação, necessitando de grandes quantidades de armazenamento. Para certos casos, tal como a exploração de minas submersas, o uso de mapas densos no processamento, atrasa processos como geração de caminhos. Utilizando uma estrutura com menos pontos, a travessia e o armazenamento de espaço explorado tornam-se mais eficientes, evitando grafos complexos gerados por métodos como RRT e variantes. A sua estrutura mais simples permite também interações homem-máquina com o seu número reduzido de pontos. Introduzo também uma abordagem autónoma de exploração baseada em fronteiras, para gerar o mapa topológico durante a exploração, tirando vantagem da travessia do mesmo para navegar por espaço conhecido. Com este trabalho, testes em simulação mostram ser possível extrair uma estrutura sob forma de grafo, descrevendo a topologia ao longo de explorações autónomas e que esta estrutura é robusta para a travessia em regiões exploradas.

Palavras-Chave: Mapas topológicos, exploração, robótica móvel, sistemas autónomos, subterrâneo.

Contents

List of Figures

List of Algorithms

Listings

List of Acronyms

2.5D	Two-and-a-half-Dimensional
2D	Two-Dimensional
3D	Three-Dimensional
AUV	Autonomous Underwater Vehicle
CERBERUS	CollaborativE walking & flying RoBots for autonomous ExploRation in Underground Settings
COG	Centre of Gravity
CPC	Cylindrical Prior Constraint
CPU	Central Processing Unit
CSV	Comma Separated Values
DARPA	Defence Advanced Research Projects Agency
DOF	Degress of Freedom
DVL	Doppler Velocity Logger
ESDF	Euclidean Signed Distance Field
EU	European Union
FOV	Field of View
GBPlanner	Graph-Based PathPlanner
INESC-TEC	*Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência*
INS	Inertial Navigation System
ISEP	*Instituto Superior de Engenharia do Porto*
LOS	Line of Sight
MEEC	*Mestrado em Engenharia Electrotécnica e Computadores*

MORSE	Modular Open Robots Simulation Engine
PS	Propulsion System
RAM	Random Access Memory
ROS	Robot Operating System
RRT	Rapidly-exploring Random Tree
SAR	Search and Rescue
SLAM	Simultaneous Location and Mapping
SLS	Structured-Light Sensors
SubT	Subterranean Challenge
TSDF	Truncated Signed Distance Field
UAV	Unmanned Aerial Vehicle
UK	United Kingdom
UNEXMIN	UNderwater EXplorer for flooded MINes
UNEXUP	UNEXMIN UPscaling
USA	United States of America
USBL	Ultra-Short BaseLine
UWSim	UnderWater SIMulator
¡VAMOS!	Viable Alternative Mine Operating System

Chapter 1

Introduction

Autonomous exploration and mapping has always had a major development focus, in part due to the risks many environments may pose to humans, but also for the convenience of offloading a task to a robotic platform. In complex scenarios, autonomous navigation requires knowledge of the environment and tracking of already visited areas. As such, many different efforts come from projects aimed at Search and Rescue (SAR), and some from a safety and inspection standpoint.

An often overlooked problem of such systems, is the processing complexity and difficulty that stems from the information density. One such case stems from the density of a point cloud or an occupancy map, although the information is there, the knowledge of what constitutes a corridor or an intersection for example, is left to human interpretation. This often poses a problem in Three-Dimensional (3D) maps, wherein projecting to a display, the closest walls obstruct view of the map, interfering with the interpretation. Such problems and dependency on human input, reveals a clear need of automation as such high level information could in turn be used to aid in the exploration effort.

Topology is often used as a support for human location and navigation, using the structure of the path (intersections or junctions) as local landmarks, rather than global landmarks (known building or a monument) and often simplifying uninterrupted winding paths as straights (roads). The same is even more necessary in structured scenarios without many differentiating features like mines, where galleries and shafts are usually made with the same width, and structures like supports are often spaced equally along its length.

This work proposes the use of topological maps to aid in the process of exploring , traversing and interpreting the scenario. Much like how road maps are approximated by lines with fixed thickness to aid in readability, a topological map could then be used to aid in the interpretation of scenarios like complex mine structures by approximating them to graphs.

1.1 Context

This work is result of a dissertation for *Instituto Superior de Engenharia do Porto* (ISEP)'s *Mestrado em Engenharia Electrotécnica e Computadores* (MEEC), following projects developed by *Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência* (INESC-TEC)'s research group on robotics and autonomous systems. In particular, major focus is placed on the UNderwater EXplorer for flooded MINes (UNEXMIN) and it's continuation project UNEXMIN UPscaling (UNEXUP).

These projects funded by the European Union (EU), aim to map and explore European flooded mines, a notoriously inaccessible environment requiring the the use of Autonomous Underwater Vehicle (AUV)s.

UNEXMIN aimed to develop technologies to guarantee a sustainable supply of non-energetic and non-agricultural raw materials, with the intent of reducing dependency on importations. [1, 2]

UNEXUP aims to take the predecessor project and commercially deploy exploration and mine mapping services. It intends to address the requirements of mining companies, geological services and other stakeholders. With that in mind, the upscaling process will address the limitations detected in field missions as well as meet the requirements from potential customers. [3]

While current methods allow for exploration and mapping, the information density often poses a major problem for constrained systems with limited resources. A full 3D map from the resulting point cloud or a full pose graph quickly become impractical for autonomous navigation, and given the scenario, planning a path through known space could take a noticeable processing time to perform on resource limited systems.

Furthermore, should an operator ever need to interpret the map to intervene, it would often result in interpreting a segment of raw sensor data rather than the complete map. Such an approach could be changed using a high level criteria like topology for the segmentation.

1.1.1 Flooded Mines

The scenarios this project focuses on are based around mine exploration and mapping, with projects like UNEXMIN [2] and UNEXUP [3] focusing on submerged and

MineHeritage, which also creates interactive experiences like 3D maps of mines.

Structure

Underground mines are usually structured with efficiency of access in mind, planned for fast entry and exit, both for safety and for extraction of materials. They can sometimes extend kilometres and have multiple levels as described by the Subterranean Challenge (SubT) Challenge [4].

As described in [5], mines usually have one or more shafts following the lode (ore deposit embedded in a rock formation). As water can start to fill the shafts some of them are sunk leveled to the nearest low ground for drainage, from where it can be drained. From the shafts, access tunnels are dug out horizontally for ventilation purposes (levels), for the same reason, it is not uncommon to find ventilation shafts aside from the access shaft.

It is usually in the mining of the lode or stope where bigger changes in structure occur, which depend on the quality or density of the lode and on the rock structure and it's minerals.

Ecton

One such example of the typical scenario is the Ecton mines in the United Kingdom (UK), a historic copper mine believed to date back to the bronze age under 4000 years ago. [6] Being used extensively from the 17th to the 19th century, the mine is now preserved for its historical significance. [7]

Since being abandoned and the pumps being stopped the mine has flooded to river level. The layout of the mine above and below water can be seen in Figure 1.1.

In May of 2019, the UNEXMIN conducted tests in the Deep Ecton mine using its UX-1 robots, exploring the shafts and levels to the blockages. [8] The results of the exploration showed archaeological evidence unseen since the mine's flood, both from the cameras as well as the resulting point cloud. [8]

Urgeiriça

Another testing place for the UNEXUP and now UNEXMIN projects is the *Urgeriça* uranium mine, in Portugal. [9] Once considered one of the most important mineral deposits in Europe its mining operations began in early 20th century and ceased in 1991. The mine has the approximate depth of 500 metres below ground level, and stretches nearly 1 kilometre in length. [10]

It's structure and general layout can be seen in Figure 1.2. The mine is described as having murky waters, confined spaces and obstacles [10], making it a challenging environment for autonomous exploration systems.

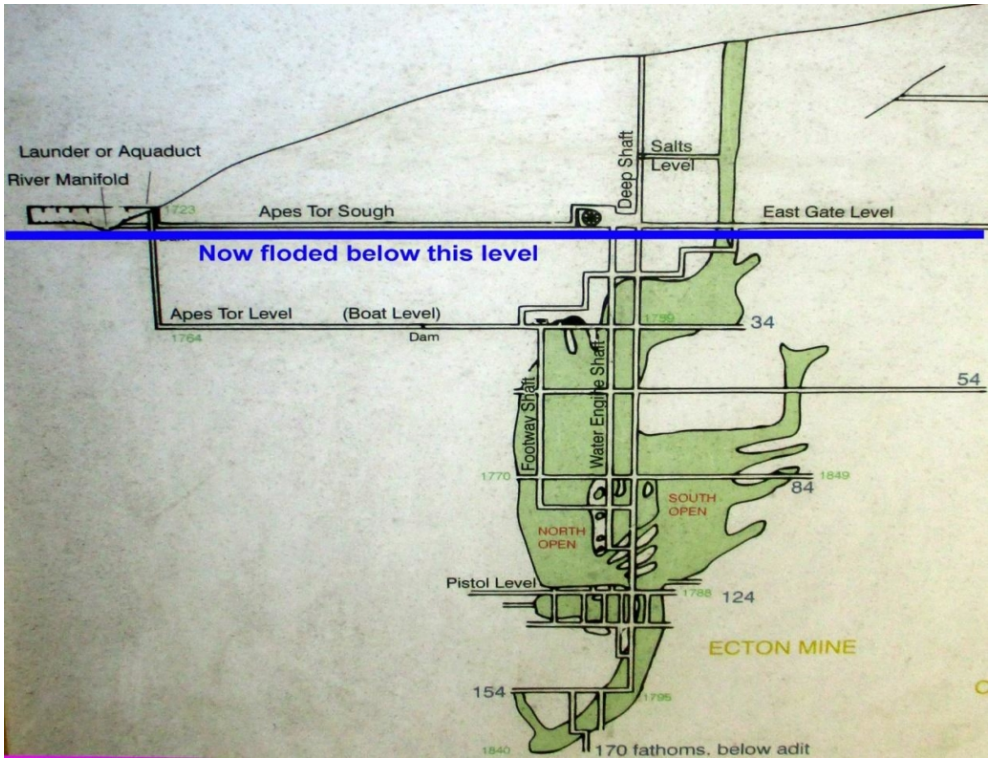


Figure 1.1: Map of the Ecton mine, where the various shafts and levels can be seen.

1.1.2 Robotic Platforms

Given the objective of exploring accessible space in structured scenarios, it's 3D scan coverage and mapping, the platforms are expected to be able to move in constrained environments and scan its surroundings. Although not required, in cases where there is a limited Field of View (FOV) of the range sensors, control over all Degress of Freedom (DOF) is expected. One such case of a system that will not be taken into account is a quadrotor with a range sensor facing downwards, since it won't be able to scan the ceiling of a structure. However a system with a quadrotor with a forward facing range sensor with a wide FOV would be valid.

Below some existing platforms are described, which will be the main application examples and use cases for the proposed project, together with the presented scenarios they will be used as a basis for simulations and development.

UX-1

Developed as a solution to the UNEXMIN project, three robotic platforms UX-1a b and c were created for the use in exploration and mapping of submerged mines. [2]

These platforms have a spherical shape as can be seen in Figure 1.3, with a diameter of around 0,6 m and are made to stand a working depth of 500 m. [2] It's

Figure 1.2: Map of the Urgeiriça Mine in Viseu.

equiped with a Propulsion System (PS) consisting of eight thrusters in a cross manifold configuration, giving the robot 5 DOF, as well as a variable pitch system that shifts its Centre of Gravity (COG), to aid in long vertical movement and compensate for buoyancy at different depths it also possesses a variable ballast system. [10]

For sensors and perception, each robot is capable of location, navigation and mapping, using relative motion sensors, pressure, vision and Structured-Light Sensors (SLS). [2]

Figure 1.3: A photograph of the UX-1 robotic platform. [11]

UX-1 Neo

Built as an upscaling of the UX-1 the newer UX-1 Neo as can be seen in Figure 1.4, was developed by the continuation project UNEXUP to improve many aspects of its predecessor. Given the focus on commercialization it, many of the changes are focused autonomy, ease of use and tetherless control. [3]

Figure 1.4: A photograph of the UX-1 Neo robotic platform. [12]

It was made modular, while also increasing maximum operational depth, it now has 6 cameras and 6 SLS with increased baseline and has now full 6 DOF control with the thrusters. [3]

EVA

Developed for the Viable Alternative Mine Operating System (¡VAMOS!), the EVA [13] robotic platform was used and shown to be able to perform 3D scans of flooded inland mines.

Its available sensors consist of a multibeam, a 3D sonar, a scanning sonar, cameras, SLS, a Doppler Velocity Logger (DVL), an Inertial Navigation System (INS), and pressure [13], and enable it to navigate and explore the scenario both autonomously and remotely.

It could be further adapted for use in more structured scenarios such as flooded underground mines but being limited by its dimensions and range of movement.

IRIS

Aiming to reduce and prevent maritime litter, the NetTag project was created, and with it the IRIS platform, aiming to use the projects' acoustic tags with unique

identification of lost fishing equipment to retrieve them. [14]

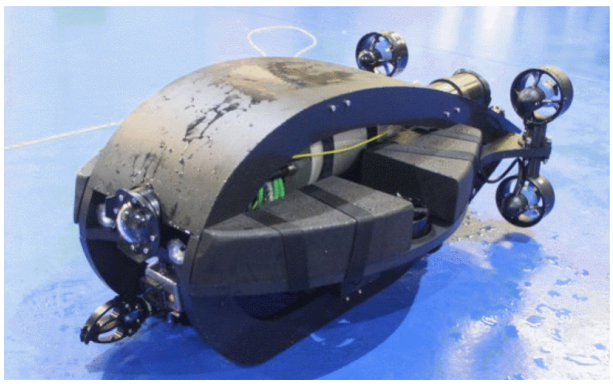


Figure 1.5: IRIS robotic platform [14].

The platform possesses 2 cameras with associated lights, an arm, an acoustic tag receptor, an INS and a DVL. An Ultra-Short BaseLine (USBL) and scanning sonars can be added. [14]

IRIS is more limited in its sensors quantity but its modularity allows its sensors to be swapped and new sensors to be added depending on its mission. One such example could be the addition of structured light projectors to add 3D scanning capabilities to its cameras.

Due to it's smaller size compared to EVA, it is able to explore scenarios with smaller space constraints.

1.2 Objectives

With this in mind, a major part of this project will be integrating a topological map into state of the art exploration systems. Keeping the existing results and advantages of existing systems, and adding to it the properties of topological maps, and it's simplicity, could result in a shorter exploration time through explored space, with shorter processing time.

It is imperative that the resulting graph is not dense, in the sense that nodes should not be closely together, facilitating display and interpretation of the results. Should it be too dense, it would approximate the results from already existing methods, lowering it's impact.

Given the potential of a simpler graph structure, by nature, it can be used to store or group collected data. By storing inside the nodes, information aside from position like: 1) points of interest; 2) neighbouring point clouds; 3) or images of interest; processed data can be accessed using graph adjacency, aside from common geometric criteria such as proximity. This results in a powerful way to store information on the map, that can be exploited for faster querying operations. To achieve this, the

graph structure should allow for information to be associated, either by providing an identification to the nodes, or by a flexible data structure.

Keeping these ideas in mind, the success of this work relies on the following key aspects:

- The extraction of the environment's topology;
- The structure containing topological information is smaller than the corresponding point cloud;
- It allows the use graph traversal algorithms in the scenario;
- The project can be integrated to existing robotic systems.

1.3 Outline

This work is structured in 6 chapters, accompanying the development and implementation, while giving a background on the decisions and choices.

To begin with, in this first chapter an introduction, as well as a contextualization behind the work is described, including its scenarios and objectives, presenting necessities and motivation behind the project.

For the second chapter, the state of the art will be presented with existing work on autonomous exploration, topological maps for autonomous systems as well as relevant works in different areas of research, keeping into account it's pros and cons.

In the third chapter the system architecture is presented, describing the expected system architecture, as well as the architecture of the developed system and it's interfaces.

Chapter seven presents the implementation, describing choices taken, giving a critic view on what can be improved and what might not be ideal.

The next chapter presents and analyzes results of the implementation, taking into account the requirements and objectives of the system.

To conclude, we have chapter six giving an overview of the project taking into account its results, presenting shortcomings and future work.

Chapter 2

State of the Art

2.1 Challenges

2.1.1 DARPA Subterranean Challenge

Funded by the United States of America (USA)'s Defence Advanced Research Projects Agency (DARPA), the SubT was held until 2021. Its aim was to "develop innovative technologies that would augment operations underground", by exploring "new approaches to rapidly map, navigate, search and exploit complex underground environments" [15].

The challenge consisted of two competitions, one where teams used "virtual models of systems, environments and terrain", named Virtual Competition, and the other where the teams competed using "physical systems" on "physical, representative courses". [4, 15]

Contestants were tested in three different scenarios such as: 1) tunnel systems, "extending many kilometres" and containing "constrained passages, vertical shafts and multiple levels", akin to subterranean mines; 2) urban underground systems, that can "have complex layouts with multiple stories" and can "span several city blocks"; and 3) cave networks, which are often composed of "irregular geological structures, with both constrained passages and large caverns". [4, 15]

The challenge resulted in many methods for autonomous exploration for different robotic platforms with varying characteristics, including quadruped and Unmanned

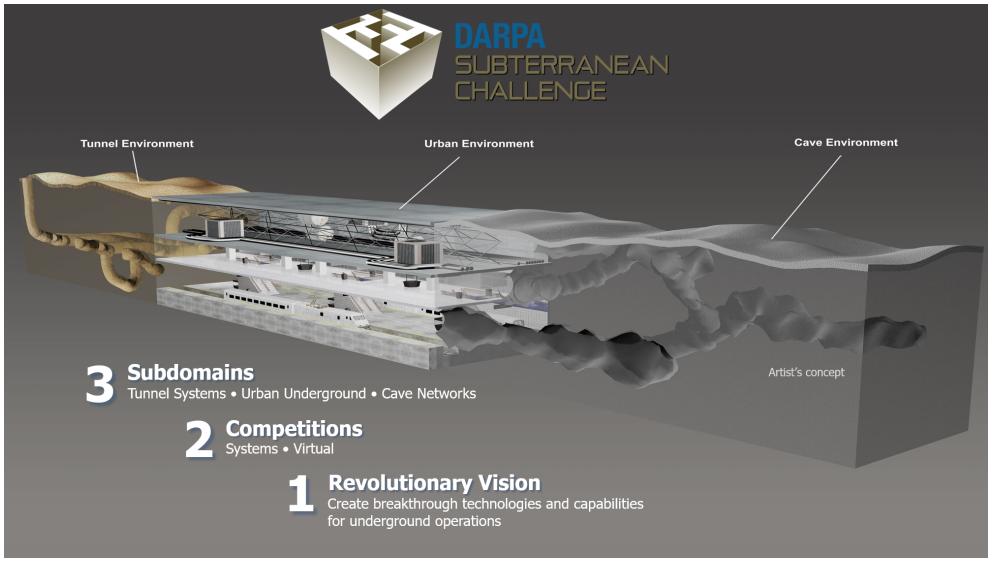


Figure 2.1: DARPA SubT Infographic, showing concepts of the scenarios posed on the different challenges. [15]

Aerial Vehicle (UAV)s. Some of the methods used by the teams will be described further ahead, as some of the requirements very much align with those of this project.

2.1.2 UNEXMIN and UNEXUP

As previously described, UNEXMIN's developments were aimed towards guaranteeing sustainable supplies of raw materials by developing towards providing surveying and exploration technologies, evaluating abandoned mines for their mineral potential. [2]

In the end, the project resulted in three similar robotic platforms capable of autonomous operation navigation and mapping of flooded mines. [11]

The continuation project UNEXUP takes all the previous successes of its predecessor and improves upon them, and focuses on launching its technologies to the market. [12]

Some key upgrades include better range and depth, its hardware and improved data acquisition management and processing. [12]

2.2 Mapping

2.2.1 Occupancy Map

One of the earliest implementations of occupancy mapping in autonomous systems is described in [16], where the author utilized range sensors to iterativelly update voxels in a grid as void or full.

This simple algorithm uses a depth image and checks voxels in the scope of the sensor which are not yet marked void [16]. For each checked voxel, all of its eight vertices are compared to the pixel in the image and if their distance is smaller than the range measured, they are marked void [16]. However if the range of the vertices intersects the maximum distance, it is marked as full [16].

2.2.2 Octree

Octrees are a hierarchical tree-based data structure which subdivides the space into volumes, with each recursively subdividing in eight sub-volumes until a set depth or voxel size, which determines the resolution of the map.

It has been shown to be very efficient in storage and ray tracing operations [17]. And have the capability to be compressed, removing redundant information. One example of compression can be thought of as pruning the leaf nodes of a fully occupied or free parent node, effectively removing an exponential number of nodes depending on the pruning.

The nodes of theses trees can be used to store a value, for example colors, or occupancy with a boolean.

2.2.3 Octomap

Octomaps are a probabilistic 3D mapping framework implemented in C++, which maps free, occupied and unmapped (unkown) areas probabilistically, with an efficient use of memory and in runtime [18].

The framework uses octrees, with the nodes storing a probabilistic value, integrated from sensor readings' noise and uncertainty. More complex values can still be stored, such as color, ambient temperature, terrain information, etc.

The framework also possesses an extension which maintains a collection of sub-maps in a tree-structure, this is of high interest for future projects, as sub-maps can be associated with the topological map to efficiently store and even perform culling of far-off regions.

Every range measurement is integrated using an implementation of ray-casting, this iterates over every voxel in a line, setting them as free, and the observed obstacles as occupied. A maximum range can be set, integrating readings even when there are no readings as free space.

Furthermore, the framework possibilitates compact storage of the maps for later use, for scenarios as described by the author, like a setup phase where a map is generated to be later used by mobile robots.

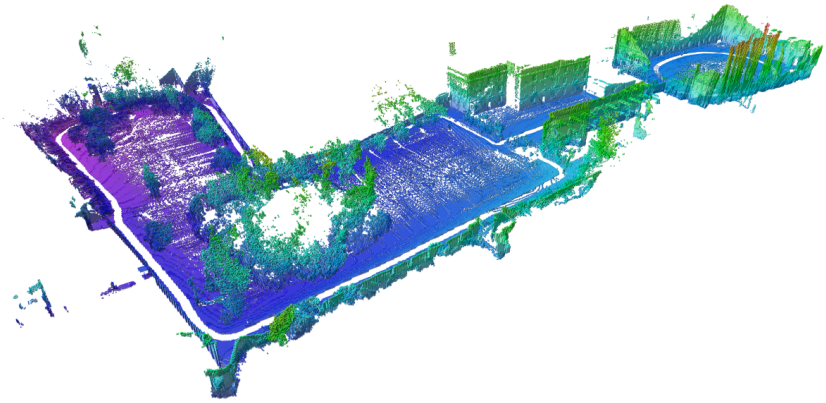


Figure 2.2: Octomap Visualization of the New College dataset. [18]

2.2.4 Voxblox

Another framework which attempts to solve the occupancy mapping in autonomous mobile systems is VoxBlox [19]. It focuses on generating maps in real-time, for local planning, which finds a path through newly-explored or dynamic terrain, while also providing a human-readable representation of the environment.

It differs from other mapping methodologies by incrementally building Euclidean Signed Distance Field (ESDF). ESDFs are a voxel grid where each point stores its distance to the nearest obstacle.

On the other hand Truncated Signed Distance Field (TSDF) are faster to construct, filter out sensor noise, and can be used to create meshes with sub-voxel resolution. They use the projective distance along a ray from the sensor to the measured surface within a truncation radius around the surface boundary.

What Voxblox does is incrementaly build ESDFs from TSDFs, which results in a faster map building than Octomaps. Successive readings are merged into the TSDF by performing raycasts once per end voxel, speeding up cases where multiple sensor readings end in the same voxel. This provides a speedup relative to the naive raycasting approach.

2.2.5 Topological Maps

Topological Maps constitute representations of the robot's environment using graphs, as [21] describes, their nodes represent distinct situations, places or landmarks, and are connected if a direct path exists between them. These maps can be built on top of grid maps but as can be seen in [21–23], are usually done offline using a complete map, and in a 2D environment.

Figure 2.3: Voxblox example mesh generated onboard the aerial vehicle on the left. [20]

Both these methods use Voronoi diagrams as a basis [21,22], which are generated using Delaunay triangulation [23]. In short, it involves a partitioning of the environment into regions, from which nodes in the graph are chosen, and subsequently its edges.

Another method addressing 3D topological map generation is described in [24], using a visual Simultaneous Location and Mapping (SLAM) map as it's input and downsampling it into a voxel occupancy map. Similarly to previous methods, it relies on triangulation of landmarks to segment the environment, and utilizes the convex hull of clusters and operations on said clusters, to generate the resulting graph.

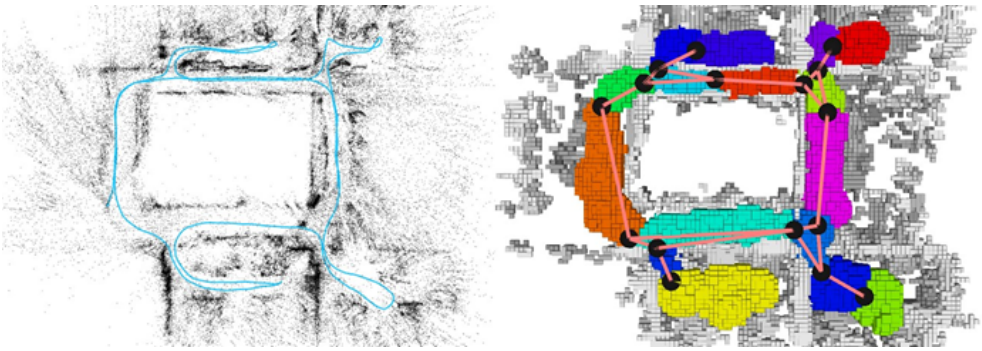


Figure 2.4: Topomap's test results with SLAM maps (left) and it's corresponding topological maps (right). [24]

One suggested method of generating such maps is described in [25] using visual landmarks and geometric constraints a conceptual grouping is generated. With these features a graph is generated representing relations between each obtained image and using metrics evaluating visual features as well as said geometric constraints. The result as can be seen in Figure 2.5, gives a very solid grouping keeping each room as an individual group.

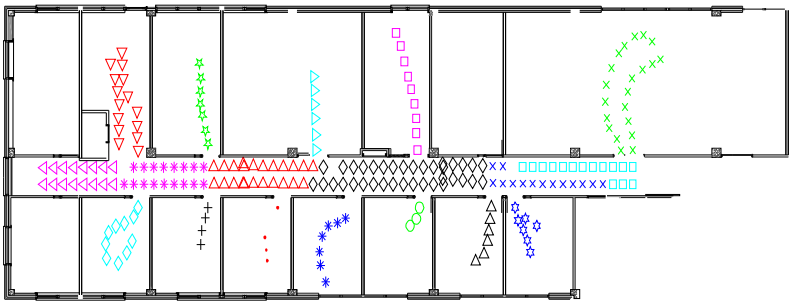


Figure 2.5: Result of conceptual grouping with a bird's eye view of the environment with each location where an image was taken where each group is represented by a different symbol. [25]

2.2.6 Skeletonization

Another problem with similar characteristics used in different scenarios and industries are the skeletonization of structures. Skeletons are abstractions which as described in [26] facilitate shape understanding and manipulation. These structures closely represent the topologogy of objects and the strategies involved can be adapted to meet the project's requirements.

A common use for these structures is in modeling trees as presented in [27], with applications in extracting these skeletons from 3D scans to reconstruct its model described in [28].

Tree Skeletonization

The method in [28] which will be summarized here assumes and exploits the cylindrical structure of the tree and its branches to extract a skeleton and later refining it.

The skeletonization starts with a point cloud and and voxelizes it using an octree, downsampling data and making it more robust to noise and varying density of the point cloud. Points are then extracted using L1-medians of the octree, and the skeleton's transitions are estimated using a k-nearest neighbour graph.

To recenter points on the structure the authors describe an optimization method called Cylindrical Prior Constraint (CPC) which assumes points in a section all lie in a cylinder, as such the middle point should lie in its centre of symmetry. With this, it iterativelly minimizes the sum of euclidian distances from the median to the local points and their weighted variance. The result is robust to differences in density and incomplete datasets.

The resulting skeleton although capturing the topology of the structure, possesses unexpected vertices and edges, focusing especially on the branching regions. Using geodesic distances to the root, the structure is separated into clusters in a top-down

order (from leaves to the root), it then detects clusters containing branches and refines their positions.

The results show smooth topological representations of trees with well centred skeleton points and joints, even in noisy and partly occluded data. The skeleton as described, can be used to reconstruct the structural model of the tree.

2.3 Planning

2.3.1 Rapidly-Exploring Random Trees

Rapidly-exploring Random Tree (RRT) is a sampling algorithm which can be used for path generation in both holonomic and nonholonomic systems [29]. By randomly sampling the state space, a tree is generated from the current configuration to the target one. [29]

As the algorithm is probabilistically complete [29], and has been used in many applications [30], it's a solid choice for mobile robotics in unknown environments.

RRTs are an algorithm which samples the configuration space $X = \{X_{free}, X_{occupied}, X_{unknown}\}$, X being the set of configurations. The algorithm returns a tree structure $\mathcal{T}$ where all the vertices and edges lie in X_{free}.

The algorithm assumes a initial state $x_i \in X_{free}$ which will be assumed as the starting configuration and a target state x_t from which a target region X_t is defined and used as a stopping condition.

A state transition function $f(x, u)$ takes into account non-holonomic constraints of the system, for holonomic planning $f(x, u) = u$, and is used to define a function $NEW\_STATE(x, u, \Delta t)$ which returns a new state given an initial state, a time step and input.

With that, the resulting algorithm for the generic RRT $\mathcal{T}$ with K samples is given by the Algorithm 1.

Algorithm 1 Generic RRT algorithm.

```
1: function GENERATE_RRT(x_i, K, Δt)
2:     T.add_vertex(x_i)
3:     for k = 1 to K do
4:         x_rand ← RANDOM_STATE()
5:         x_near ← NEAREST_NEIGHBOR(x_rand, T)
6:         u ← SELECT_INPUT(x_rand, x_near)
7:         x_new ← NEW_STATE(x_near, u, Δt)
8:         T.add_vertex(x_new)
9:         T.add_edge(x_near, x_new, u)
10:    end for
11:    return T
12: end function
```

In many cases however, the generated RRT doesn't have constraints, and doesn't need to take into account the underlying system, for example to quickly sample reachable space in holonomic systems. For such cases, the generic algorithm can be simplified.

Most path finding algorithms might quickly arrive at a state with LOS to the target, in such cases, a purely random state will not be the fastest converging method. One trick to bias the sampling towards the target, is to set some of the states as the target, this can be done and parameterized in many ways, one such example is by using a probabilistic approach as shown in Algorithm 2.

Algorithm 2 Biased Random State Generator

```
1: function BIASED_RANDOM_STATE(prob, x_target)
2:     if (random() mod prob) = 0 then
3:         x_new ← x_target
4:     else
5:         x_new ← RANDOM_STATE()
6:     end if
7:     return x_new
8: end function
```

On its own, RRTs aren't asymptotically optimal as they don't converge towards the optimal solution, for that reason, many variants have been developed, most notably the RRT*. [30]

2.4 RRT*

By introducing graph optimization to the tree, RRT* results in shorter and straighter lines.

Based on the A* graph search algorithm, RRT* starts by randomly sampling the space, much like the RRT, but also checks all adjacent states for reachability and removes edges that are not part of the shortest path to the root [30]. Although not explicitly RRT or a variant, [31] uses a similar graph sampling method and optimization to generate maps for path planning, resulting in the GBPlanner which will be discussed ahead.

This algorithm may not fit all needs as in some cases it may be used as an indiscriminate search rather than optimization problem. One such example is the search of frontiers which doesn't require the storage of relations between the vertices of the tree, but only the frontiers crossed.

Its generic algorithm can be seen in the Algorithm 3.

Algorithm 3 Generic RRT*.

function GENERATE_RRT_STAR($x_i, K, \Delta t$)
  $\mathcal{T}$.add_vertex(x_i)
  for $k = 1$ to K **do**
    $x_{rand} \leftarrow$ RANDOM_STATE()
    $x_{nearest} \leftarrow$ NEAREST_NEIGHBOR($x_{rand}, \mathcal{T}$)
    $u \leftarrow$ SELECT_INPUT(x_{rand}, x_{near})
    $x_{min} \leftarrow x_{nearest}$
    $X_{near} \leftarrow$ NEAR($\mathcal{T}, x_{new}$)
    $\mathcal{T}$.add_vertex(x_{new})
    $c_{min} \leftarrow$ Cost($x_{nearest}$) + Cost(Line($x_{nearest}, x_{new}$))
    for each $x_{near} \in X_{near}$ **do**
      if Cost(x_{near}) + Cost(Line(x_{near}, x_{new})) $< c_{min}$ **then**
        $x_{min} \leftarrow x_{near}$
        $c_{min} \leftarrow$ Cost(x_{near}) + Cost(Line(x_{near}, x_{new}))
      end if
    end for
    $\mathcal{T}$.add_edge(x_{min}, x_{new})
    for each $x_{near} \in X_{near}$ **do**
      if Cost(x_{new}) + Cost(Line(x_{new}, x_{near})) $<$ Cost(x_{near}) **then**
        $x_{parent} \leftarrow \mathcal{T}$.Parent($x_{near}$)
        $\mathcal{T}$.remove_edge(x_{parent}, x_{near})
        $\mathcal{T}$.add_edge(x_{new}, x_{near})
      end if
    end for
  end for
  return $\mathcal{T}$
end function

2.4.1 Frontier-Based Exploration

The use of frontiers in autonomous exploration was introduced in 1997 by [32] who defined frontiers as the region in the border between known free space and unknown space. The core principle behind frontier-based exploration is simple, and as said in [32], "To gain the most new information about the world, move to the boundary between open space and uncharted territory".

Using an evidence grid which consists of an occupancy grid with associated occupancy probability as a map generated from a range based sensor like the laser-limited sonar used by the author. The resulting map goes through a process "analogous to edge detection and region extraction", to label open cells adjacent to unknown cells as frontier edge cells. Only frontier regions larger than a given size will be considered a frontier, the size being relative to the robot's dimensions checking if it's reachable. This process can be visualized in the Figure 2.6. [32]

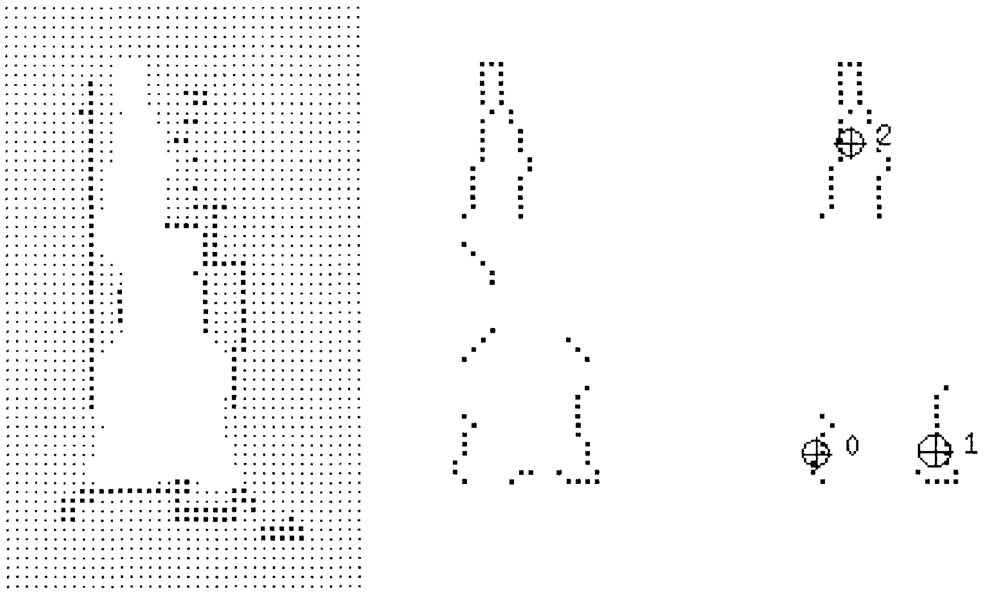


Figure 2.6: Frontier Generation: evidence grid (left), frontier edge segments (middle) and frontier regions (right). [32]

With the frontiers detected, a path planner would explore the nearest accessible frontier and dynamically update the evidence grid. Dynamic obstacle avoidance is still possible using this method, since while the frontier point is fixed, the path can be adapted as necessary making sure now unreachable frontiers are removed.

2.4.2 GBPlanner

A proposed method for exploring the subterranean environments described by the SubT challenge comes in the form of a Graph-Based PathPlanner (GBPlanner), developed by the winning team CollaborativE walking & flying RoBots for autonomous

ExploRation in Underground Settings (CERBERUS), a collaboration between multiple institutions and resulting in many open-source packages.

This approach separates the planning between two searches, a local exploration planner that focuses around the robot's pose, and a global one which explores globally inside explored space. [31]

The local planner builds upon the rapidly exploring random graph, expanding the graph around a given bounding box centred on the robot's pose, similarly to RRT*, it searches for the nearest neighbors of the new sample, and creates a transition if it passes a collision test, unlike RRTs however, given its graph structure, vertices can have more than two transitions. To traverse the graph, the Dijkstra's algorithm is used on the local graph from the root node, returning only the minimum length paths. A best path is selected from the root of the local map to the vertex with the highest exploration gain. This gain was calculated simulating a sensor measurement at the vertices and getting the volume that a range sensor would read (volumetric gain), and other parameters such as the euclidean distance along the path. [31]

The global planner starts from the minimum length paths of the local planner, and chooses the "principal" paths, consisting of the longest to add to it's graph. It's leaf nodes will be assumed as possible frontiers, and periodically the volumetric gain of these nodes is recalculated, updating the list of preferred frontiers. The planner also periodically calculates Dijkstra's algorithm to find the shortest paths from the current location to potential frontiers, as well as from all the potential paths towards home, whenever the local planner cannot build any informative path. A path home is also calculated continuously in the same way, and uses this path when it needs to return home. [31]

Each path generated is passed through an improvement step, pruning short edges caused by the random sampling, and moving the vertices further away from obstacles, returning a safer and smoother path. [31]

This method is of especially great interest to this project, given similarity of the global graph and its use to define a more general topology of the environment, it however remains very dense in structure.

2.5 Simulation Environments

Some work has already been done developing a simulation environment [33, 34] for the UNEXMIN project, it presented many advantages and disadvantages over many different simulators, namely GAZEBO [35], UnderWater SIMulator (UWSim) [36] and Modular Open Robots Simulation Engine (MORSE) [37].

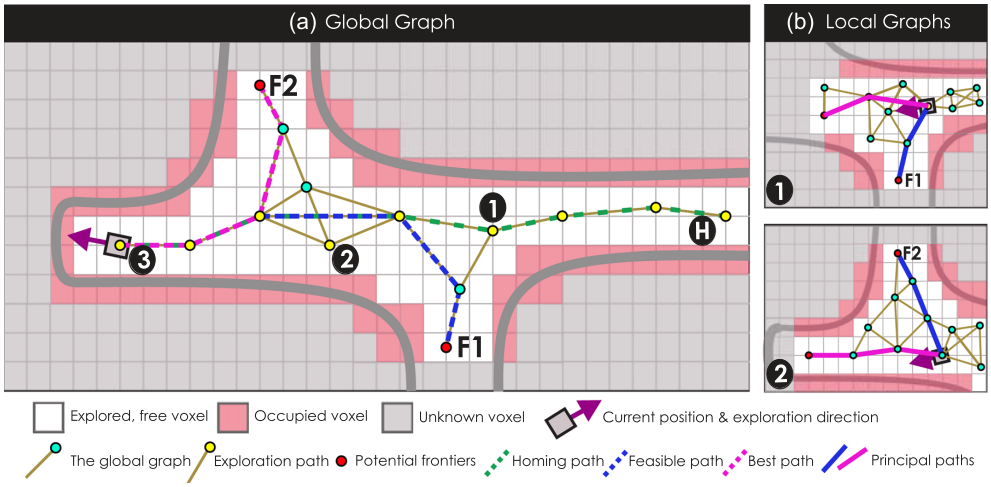


Figure 2.7: GBPlanner planning visualization. In (b) are two visualizations of the local planner graphs in positions 1 and 2. In (a) is a visualization of the global planner graph, showing also position 3 in a dead-end and home in H. [31]

2.5.1 GAZEBO

One existing solution for robotic simulators comes in the form of Gazebo, a project designed to reproduce dynamic scenarios, simulating both the robots' and objects' dynamics properties in the scene. [35]

Gazebo comes already packaged with the current Robot Operating System (ROS) distributions, containing already documentation and examples created by the community. Many resources are shared and competitions are held using Gazebo as their environment, with examples such as the previously mentioned SubT [4, 15].

By default it uses an ODE fork for its physics simulation, however it supports others, such as Bullet. The simulator includes some robotic models and sensors, but can be further expanded using it's plugin system, making it possible to develop a model for non existent sensors, as has already been described in [34] and [38].

2.5.2 MORSE

Another simulation option is the MORSE which was developed as a simulator at the system level rather than a highly specialized simulator [37], which gives more attention to the sensors' accuracy .

The MORSE simulator uses Blender and its game engine to simulate environments [37], right away, the Blender game engine uses Bullet physics simulation, giving it a solid dynamic system simulation, fused with Blender's graphics and visual characteristics.

It comes with many sensors, actuators and robotic bases, and further can be added. The simulation being set up using python, scenes, robots and its sensors

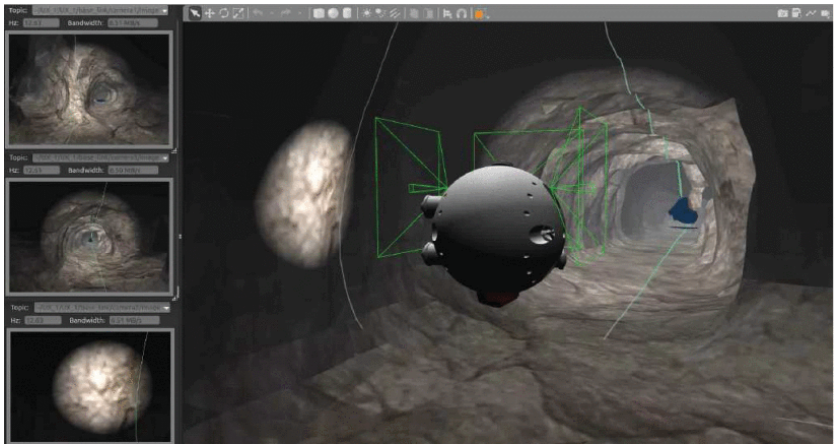


Figure 2.8: Simulation Environment in Gazebo used for the UNEXMIN project. [34]

and actuators are developed using scripts, while the physics and 3D rendering are handled separately [39].

2.6 Centering

A centering algorithm is helpful for obtaining a point far away from constrained passages. It can be used as an obstacle avoidance by getting a safest position, or be used to estimate a skeleton of a structure.

There exist already some methods and algorithms to estimate skeletons in trees by exploiting symmetries in the underlying structure [40]. By switching to an optimization problem, minimizing the average distance to neighbouring points, and geometric constraints, in this case equal-distance. This method can be viewed as $c = \arg\min_c \sum_{i=1}^{m} ||c - x_i|| + \lambda\sigma^2$ with c being the estimated centre point, x_i a point in the region, σ^2 the variance of the distance to the centre and λ its gain or weight.

To solve this problem one can use many methods, such as gradient descent, particle swarm, even brute-force using methods similar to computer vision and calculate many samples in the region of interest, picking the one with best score.

2.7 Graphs

There are two main ways to store graphs, either through adjacency matrices, which for undirected graphs can be simplified with an upper diagonal matrix, or as adjacency lists, which don't store non existent edges [41].

Adjacency matrices have an advantage where some problems can be solved using matrix operations which are known to be parallelized [42]. Each line and column

represents a vertex of the graph and their intersection the corresponding edge, having a 0 when there are no edges or $\mathbb{N}$ representing the number of edges [41, 42].

They however are not very memory efficient, given that they store every possible transition, including those that don't exist, resulting in a theoretical efficiency of $O(n^2)$ for storage, $O(1)$ in traversal [42], $O(n^2)$ when changing the number of vertices and $O(1)$ when editing edges, where $n = number of vertices$.

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

Figure 2.9: Example of an adjacency matrix

Adjacency lists, contrary to adjacency matrices, do not represent non-existent transitions, being more efficient in storage for high number of vertices and edges.

They are represented by a vertex list, where each vertex has a list of transitions. This gives it a theoretical efficiency of $O(n + t)$ for storage [42], with $t = number of adjacent vertices$, increasing with the number of vertices and edges. Its traversal is slower, given that it's necessary to iterate over every transition of the vertex, resulting in $O(t)$ [42].

The addition of vertices is more efficient, since only an item needs to be added to the list resulting in $O(1)$ efficiency. The removal needs a check in all vertices and transitions, to remove transitions to the deleted vertex, this results in an efficiency of $O(n \times t)$.

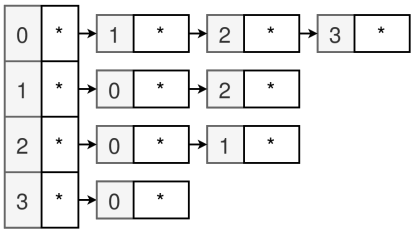


Figure 2.10: Example of an adjacency list. Left and vertically a list of vertices, pointing to vertices that it creates edges to on the right.

2.8 Dijkstra Algorithm

A common incentive over the use of graphs are the existing methods to perform high level operations. One such case is the computation of the shortest path to a target.

Many methods and algorithms exist to solve this problem, trading off between the quality of the solution and its computation time. Dijkstra is an algorithm that returns the shortest path on a weighed directed graph, when weights are non-negative [41].

It's use ranges all branches of robotics and computation, although it is widely used, it's not always the right choice for all problems, since it uses a greedy approach [41,42] it's usually one of the slowest methods. Nevertheless for graphs with a small amount of edges such is the case in some topological maps, or tree-like structures it's downsides are mostly offset by it's quality.

The algorithm works by keeping a set of vertices and their shortest path weights to the source, and repeatedly selecting vertices adjacent to those already weighted with the shortest estimated path, and adds them to the set [41]. At the end the graph has all vertices labeled with the shortest path, and all that's needed is to traverse from the target vertex to it's adjacent vertex with smallest label.

One way to implement it using min-priority queues, as described in [41], and visible in the Algorithm 4 assumes a weighted graph $G(V, E)$ with a set V of vertices and a set E of non-negative edges (u, v) with weights $w(u, v)$. It initializes the vertices' shortest path estimate d and predecessor vertex π, the set S to the empty set and the set of vertices not in $Q = V - S$ that starts with all the vertices of the graph V. Iteratively $Q = V - S$ will be kept a constant, taking a vertex u, which is the shortest path estimate of any vertex, from Q and adding it to the set S. All vertices from edges leaving u are then relaxed, therefore updating the estimate $v.d$ and the predecessor $v.\pi$ if their shortest paths pass through u. The answer is then got following the shortest path from the ending node to the start.

Algorithm 4 Dijkstra

```
1: function DIJKSTRA(G, w, s)
2:     for each v ∈ G.V do                          ▷ Initialization of the single-source
3:         v.d ← ∞
4:         v.π ← NIL
5:     end for
6:     s.d ← ∅
7:     S ← 0
8:     Q ← G.V
9:     while Q ≠ ∅ do
10:        u ← Q.extract_min()                      ▷ Min-priority queue
11:        S ← S ∪ {u}
12:        for each v ∈ G.Adj(u) do                 ▷ Relaxation
13:            if v.d > u.d + w(u, v) then
14:                v.d ← u.d + w(u, v)
15:                v.π ← u
16:            end if
17:        end for
18:    end while
19:    return G
20: end function
```

Chapter 3

System Architecture

A system was projected as a set of processing blocks, each solving a different problem. The planned system in Figure 3.1 will take as input the point cloud generated from range sensors, which could come from SLS, sonar, lidar, stereo vision or a combination of sensors. Added to that it will be assumed that the localization problem is solved, and as such will be used as ground truth.

All input clouds will be pre-processed, mainly focusing on downsampling and efficient storage in a global point cloud of the known environment. This stage is made using a system like a voxel grid, an octomap [18] or voxblox [19], all of them already take into account the input sensor model, and integrate sensor readings, forming a volumetric map.

A topological map server manages its graph and its interfaces, here some functions will be provided, like the traversal between two nodes. From this server, points of interest will be detected, generating the topological map, and Line of Sight (LOS) checks will be performed for cycle closure.

From there, the exploration is done by extracting frontiers from the grid map, assuming them as potential targets, followed by a path planner that would take into account both the existing topological map for long ranges, and sample a path from known map points to the targeted position.

One other processing block dedicated to control and obstacle avoidance will be used as the interface between the actuators and the planner, this block will be the most platform dependent. Collision prevention measures will however be expected as a fail-safe should any other system fail.

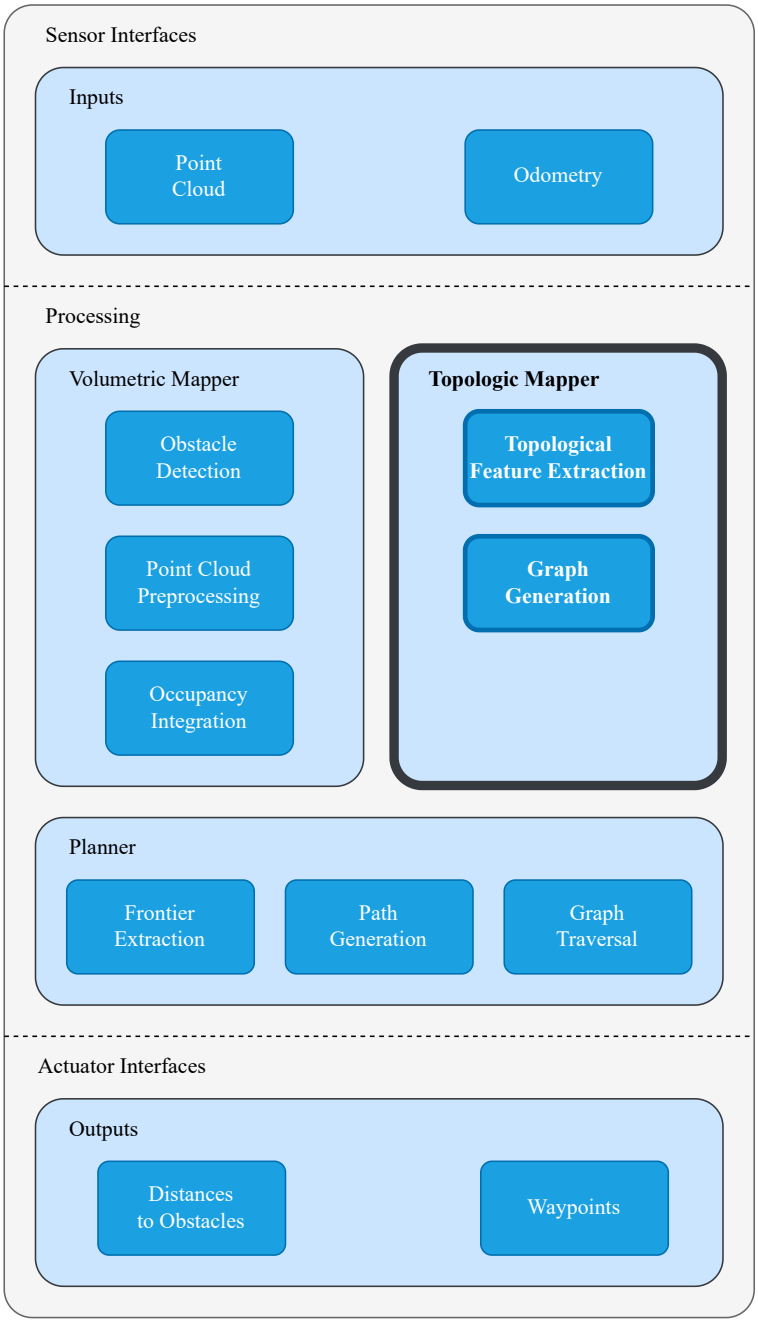


Figure 3.1: High Level System Architecture, where the topological mapper and its interfaces are where the development focus was taken (in bold).

As shown in Figure 3.2, this project will use range measurements from the robotic system via point clouds, which are processed into an occupancy map. The robotic system is assumed to be the pairing of robotic platforms and their operational environments.

Using the occupancy, topology will be extracted by detecting its features, storing regions of interest as components in a graph. The graph has the added benefit of being faster and repeatable for path generation through explored scenarios using graph traversal algorithms, and as such it is used in conjunction with the planner for more robust traversal.

A planner then uses the graph and the frontiers or points of interest, to generate a path through the scenario, while using other sampling based methods, taking advantage when possible of the known topology to create quick and safe waypoints.

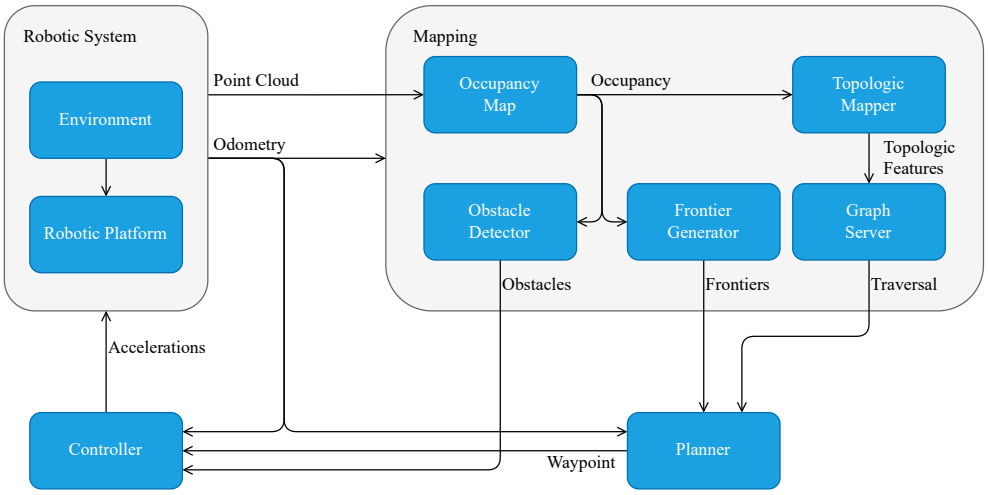


Figure 3.2: Developed System Architecture

Chapter 4

Implementation

4.1 Development Environment

4.1.1 Software Choices

ROS

For middleware the ROS [43] was chosen, along with its various available development tools and environments, it's already used in many relevant projects [31, 33], and has many examples of usage and documented modules by the community.

ROS operates in a graph like structure [43], containing nodes where the processing is held, and peer-to-peer messages are passed between them.

These messages are data structures which include basic data types (integers, floating points, booleans, etc.), and can have arrays of said types. Nodes send or receive using topics, with nodes subscribed to a topic receiving messages that other nodes published to it asynchronously.

This publisher/receiver relationship provides advantages wherein many subscribers can read data published in a topic, and many publishers can publish to the same topic.

For synchronous requirements another method called a service waits for a request to send a response, this way even use cases requiring synchronous communication are accounted for.

The modularity provided by this structure means that topics using standardized messages, support the development of different modules with similar functionality,

e.g. a node which subscribes to raw sensor data and pre-processes it and publishes to another topic, can be changed with another pre-processing block without a need to change any other part of the system.

4.1.2 Simulation

Although some of the existing simulators have already been developed [34], some differences in requirements for the exist, while the previously mentioned aims for visual and dynamical accuracy, this project assumes some problems such as control and sensor preprocessing as solved and focuses instead on taking that data and testing its traversal and mapping capabilities. This means that visual fidelity, sensor accuracy and noise simulations, are not the main focus and can be sacrificed over other requirements such as low footprint and ease of use and setup.

Testing and use of the Gazebo simulation of the UNEXMIN platforms quickly revealed that it was computationally intensive resulting in a big chunk of resources which could be used by the developed systems. This posed a need to simplify the model or alternatively use a different environment trading simulation models' accuracy for faster processing and lower resource usage. Many features such as visual fidelity, noise models, and complex buoyancy and hydrodynamic simulations, can be swapped for approximations.

Although no simulator has been developed using MORSE for the UX-1 platforms, one can be quickly created, collectives of range sensors like laser scanners and sonars can be simulated and approximated by depth cameras, allowing for configuration using resolution, max range, and FOV. Actuators can also be grouped into a single actuator which changes all degrees of freedom, removing the need to use a complete controller to orchestrate the system. Furthermore, localization and pose estimation can be parameterized by noise and drift, simulating the collective sensor fusion drift rather than each of their parts.

For those reasons, MORSE was chosen as the simulator to be used while developing the project, being easily changed and customized, and being less resource intensive than the existing GAZEBO simulation.

4.1.3 Visualization

Given the need to visualize the topological map and the system's operations, a visual interface is required not only for the end solution, but also for development. This will result as the main feedback to the user or operator, since the use of the raw data such as graphs, or point clouds takes more effort to validate than visual inspection.

RVIZ

With this in mind, the visualization system ROS RVIZ was chosen for this project. This tool comes packaged by default in most ROS distributions, and it is well documented, and full of examples. ROS and RVIZ already have many visualization primitives which can be used to display lines and point clouds, for example some of which don't need any additional processing. This makes integration into a system which already uses RVIZ easy to understand and adapt for projects which use ROS.

4.2 Occupancy Map

An important part of this project is the generation and storage of the occupancy map, given it's a 3D scenario, they become bigger problems to tackle, the growth of the added dimension exponentially increases its requirements. Another challenge it poses is that the scene cannot easily be mapped to Two-Dimensional (2D) like in Two-and-a-half-Dimensional (2.5D) scenarios, most commonly height maps which only store the occupied depth and assume everything above as free and anything below is ignored or assumed occupied.

Octomap has already a wide use by the community, with many examples and feedback, as well as an in-depth documentation in Doxygen, and example code in the source. Furthermore it already has a ROS implementation which provides the necessary functionality to get started and being open-source, can be adapted if needed.

Tests performed using this system however posed problems with its ROS implementation, even affecting the robot's control [9], which points to Central Processing Unit (CPU) usage. This doesn't however necessarily invalidate it's use as the implementation can be adapted to mitigate this issue and can even be used as a placeholder during prototyping until a more viable method is found.

With its benefits, Voxblox is a strong option, being faster than Octomaps, providing more accurate occupancy maps and returning a mesh of the environment. However, being more recent, less implementation examples exist in the community when compared to the Octomap, and more time would be needed searching documentation and testing during development, making for slower prototyping.

Because of the resources available, Octomap was chosen over Voxlbox, and although they have their differences, the work done with Octomap can be adapted to Voxblox since the functions used like ray tracing are implemented in both frameworks.

4.3 Topological Map

4.3.1 Graph

To implement the graph, prioritizing lower memory usage, an adjacency list was chosen and assuming the linear growth of traversal time as an acceptable trade-off. For this reason a structure was created to store the nodes and transitions.

The developed object uses an unordered map to associate each node to an unique identifier. Unordered maps already implemented in standard libraries, optimize querying using hash functions.

The nodes are made simple and hold the necessary data, be it position of the node, color in colored graphs, weights or even a pointer to a local point cloud if necessary. Transitions are defined as a vector of identifiers to adjacent nodes, to access them, that identifier can be used in conjunction with the unordered map to quickly query.

All the necessary functions are defined along with the created structure, and can be adapted in accordance to the needs of the project. Aside from the usual functions to create, remove or change nodes and transitions, some additional ones were created, namely a function to estimate the distance to a transition and a function to traverse the graph.

Distance to Transitions

The computation of the euclidean distance to a transition takes into account the positions of the adjacent nodes and calculates the closest point to the target position.

To note that for optimization reasons the distance squared can be used in tasks such as getting the minimum distance this avoids the computation of the square root.

To calculate the the distance to the closest point in the transition, first this point must be calculated. This point is located in the normal of the line created by the transition which intersects the current position. Assuming T_1, T_2, P and M as the two transition points, the current position and nearest point in the transition respectively, we can use the dot product between $\overline{T_1T_2}$ and $\overline{PM}$, which should be 0 since they are perpendicular, resulting in:

$$(P - M) \cdot (T_2 - T_1) = 0 \tag{4.1}$$

By defining the line between T_1 and T_2, we can also define the point M as a transformation from one point and a scaled vector:

$$M = T_1 + u(T_2 - T_1) \tag{4.2}$$

Substituting M in equation 4.1 by 4.2, we obtain:

$$[P - T_1 - u(T_2 - T_1)] \cdot (T_2 - T_1) = 0 \tag{4.3}$$

And solving u, we get:

$$u = \frac{(P_x - T_{1x})(T_{2x} - T_{1x}) + (P_y - T_{1y})(T_{2y} - T_{1y}) + (P_z - T_{1z})(T_{2z} - T_{1z})}{||T_2 - T_1||^2} \tag{4.4}$$

Again, it can be noted that if we already work with squared vector norms to avoid square roots, the denominator requires one less operation since the square root cancels out the square.

To check if the projected M lies inside the line segment $\overline{T_1T_2}$, the condition $0 \geq u \geq 1$ can be used to validate if it lies within its boundaries.

The resulting point M can then be calculated using the equation 4.2 as:

$$M = \begin{pmatrix} T_{1x} + u(T_{2x} - T_{1x}) \\ T_{1y} + u(T_{2y} - T_{1y}) \\ T_{1z} + u(T_{2z} - T_{1z}) \end{pmatrix} \tag{4.5}$$

Lastly, the squared euclidean distance is calculated, resulting in:

$$||P_2 - P_1||^2 = (P_{2x} - P_{1x})^2 + (P_{2y} - P_{1y})^2 + (P_{2z} - P_{1z})^2 \tag{4.6}$$

Traversal

The traversal of the graph was done using Dijkstra algorithm over the whole topological graph. To possibilitate adaptation to different scenarios, the weights weren't generated alongside the graph, but as cost function which can be changed depending on needs. This was done for cases where not only purely geometric factors impact the weight but also sequential, one such example is valuing going straight over harsh corners.

With this in mind to calculate weights the square norm between two nodes was used, although it might not always be the ideal method, it works well on most.

Other methods considered for weighting transitions, included estimated traversal time, momentum changes or even energy usage. As most of these methods require a known robot model, they were assumed to be less of a priority over the scope of the project.

4.3.2 Feature Detection

One very important part of this work relies on the detection of topological features. As described before many different kinds of features exist, and with them many methods to estimate them.

The features and the corresponding detection methods used and considered for this project will be described below.

Openings

Using the generated 3D maps, open spaces can be detected checking if the cross-section is wider than a threshold using a plane perpendicular to the displacement s from the nearest node.

The cross-section can be approximated using ray casting, by casting many rays in the current position and perpendicular to the displacement, the resulting polygon approaches its cross-section, with the accuracy increasing with the number of equally spaced rays.

This method also allows the estimation of the COG of the resulting polygon, by averaging pairs of non coincident co-linear ray hits.

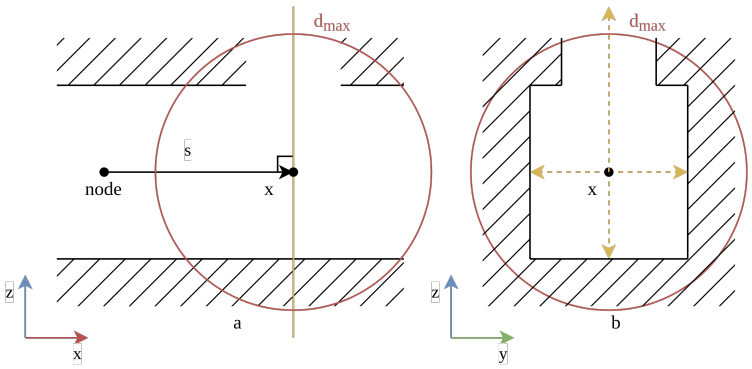


Figure 4.1: Opening Detection—the plane perpendicular to the displacement s between the nearest node and the current position x in yellow shown in a), and co-planar rays in yellow shown, with the rays that cross a threshold d_{max} displayed as a red circle in b).

The position where a graph node can be placed will lie in the line between the previous traversed node and the current position, ahead of the current point. If ahead of the position, an obstacle exists, another method will be used and isn't taken care in this system otherwise, two other different methods exist to choose a position.

Out of these methods the simpler one is achieved by applying a fixed offset to the current position, making a node always a fixed distance from the current position. The other more complex and computationally intensive method uses consecutive positions ahead to find the end of the closing, and place the node in the point in the middle of the first opening detection and the last one. The offset method however is more efficient and produces sufficiently good results without the added computational cost of the subsequent ray casts so it will be preferred.

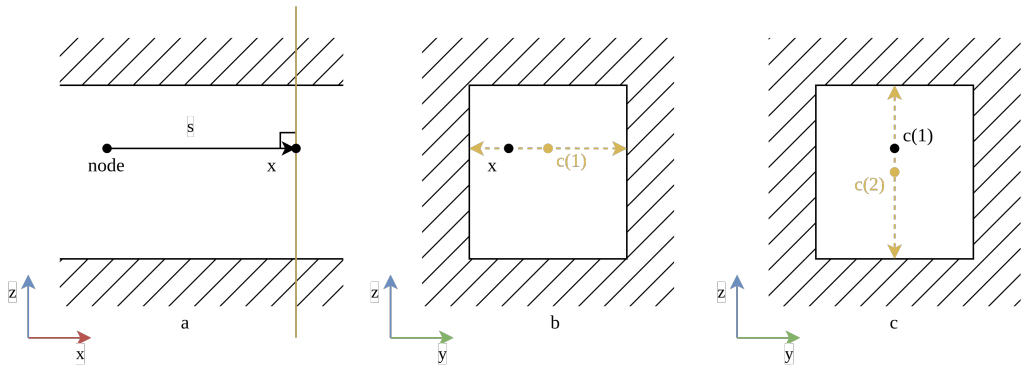


Figure 4.2: Estimation of COG—the plane perpendicular to the displacement s is calculated as seen in a), the COG is iteratively estimated with pairs of colinear points with each iteration slowly converging towards an approximation shown in b) and c).

Obstacles

Obstacle detection is the simplest method of detecting features being used, and can be achieved with a ray cast from the current position in the direction of the movement from the previous node.

Preferably this method has a maximum distance threshold from the previous node to the current position, to better centre the node in the structure, confirming the surrounding region has already been observed before attempting to centre.

The resulting node can then be placed by applying an offset to the distance to the obstacle. Said offset should take into account two factors, the dimensions of the platform and/or the dimensions of the scenario, be it average radius of mine galleries or any other metric, making sure intersections do not intersect the scenario.

It should however be emphasized that the resolution of the map strongly influences both the accuracy of the cross-sections' COG, and distances to obstacles, being maps with higher resolutions being preferred when raw sensor readings cannot be used.

Cycle Closure

For cycle closure we can take advantage of the occupation map. The scenarios where we verify that a transition between two nodes exists, is when they have LOS. This can be simply checked by casting a ray from one node to the other and if there is no collision to an obstacle or unknown space, a transition can be assumed to exist.

As most occupation maps already include a ray cast function and most are aware of unknown space, it is possible to check for a collision if the last position of the ray with max length is equal to the distance between the nodes.

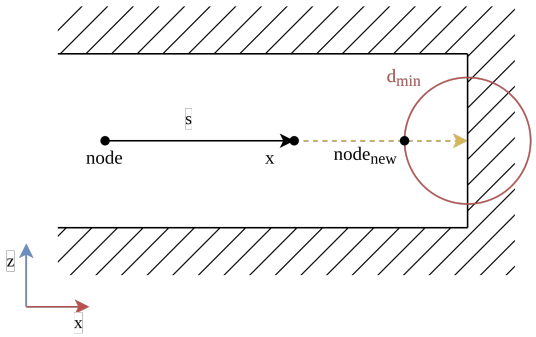


Figure 4.3: Obstacle detection—When farther than a threshold to the nearest node, a ray is cast from the current position x in the direction of the displacement s, a new node is placed at the minimum safe distance d_{min} allowed to obstacles.

System Flow

Using the features described above, we can create a topological feature detection system by managing and joining the various processes, avoiding ambiguity and unpredictability.

The system flow represented in Figure 4.4 begins with the creation of the first node, that node should ideally be initialized with the initial position of the mission.

The system then waits for an update in either the odometry or the occupation map before starting a new iteration, since there's no need to keep iterating if there are no significant changes.

With the received information it updates the current node to the nearest graph node using euclidean distance and LOS.

It then checks for topological features, attempting to centre the current position in the map and in the process checking for openings. After centering, obstacles are checked, casting a ray in the direction of movement.

A node is created if it's not near a transition or node. In case it hit an obstacle, it will create a node between the wall and itself. If no obstacle exists, and an opening was found, a node is created slightly ahead of its position.

4.4 Exploration

The exploration part will take care of covering the scenario and acquire the necessary sensor readings to generate an occupancy map and obtain the necessary features for the topological graph.

With this in mind, exploration can be divided in two sub problems, the generation of a frontier, which will return regions that have yet to be explored, it will give points which are yet to be covered, and the other problem, the mission plan,

focusing on picking frontier of interest and navigation through already covered space to reach said regions.

4.4.1 Frontier Generation

As previously described, frontiers are situated in the transition between known free space and unknown space, this however is computational dense for 3D environments and requires a sampling method avoiding a check over the whole grid.

While it is possible to sample only the recently updated nodes using some occupation maps, it is still very dense, prone to noise in the map with some frontiers generated not being reachable and the vast majority of samples will be regions of known free space rather than frontiers. For this reason the chosen method involves a random sampling of the map using an implementation of RRT. This way a frontier can be detected whenever a new sample crosses into unknown space without colliding with an obstacle.

Concretely, being $F = f_0, f_1, ..., f_n$ the set containing all frontier points and $S = \{s_0, s_1, ..., s_n\}$ the set containing all the n samples, the initial sample s_0 is set to the current position of the robotic platform according to odometry. Each iteration a new sample s_{new} is generated and its nearest neighbour $s_{neighbour} = nearest\_neighbour(S, s_{new})$ is calculated, and its position changed to a fixed distance d away from $s_{neighbour}$, achieved by transforming the vector between s_{new} and $s_{neighbour}$ such that:

$$s_{new} = s_{neighbour} + \frac{s_{new} - s_{neighbour}}{|s_{new} - s_{neighbour}|} \times d \tag{4.7}$$

It is then followed by a ray casting between $s_{neighbour}$ and s_{new}, should the ray reach its destination without collision, the sample is added to the set $S = \{S, s_{new}\}$ otherwise, the point where the ray stopped $s_{collision}$ is checked, if it lies in an occupied region the iteration ends, but if it doesn't then the point is considered a frontier and added to the set $F = \{F, s_{collision}\}$.

The resulting frontier set is usually very dense so a down-sampling is applied, removing duplicate points, and reducing the number of points significantly.

Another process then further reduces the number of frontiers by removing points too close to obstacles, described further ahead, this removes frontiers from situations like narrow openings where the platform wouldn't fit and small noise which results in single points in a wall being falsely labeled as adjacent to unknown space.

This is achieved using two methods, first a voxelization process which removes duplicates and effectively down-samples the number of points. And another process which samples a region around each frontier for obstacles, checking the occupation map.

Every iteration, frontiers that have been explored need to be removed, this is achieved by checking if every frontier point is still adjacent to unknown space by sampling over neighbouring nodes of the occupation map.

4.4.2 Planning

The planner, which will be described in this section, is composed of multiple sub processes. These processes consist of picking a frontier, obstacle avoidance and path generation to the frontier.

Starting from the frontier choice, picking a frontier point highly impacts the exploration time, since ideally you go to each place the least amount of times possible.

To solve this, the frontier point picked is the closest reachable, which makes sure the current space is explored before going to the next. The point is picked using the nearest-neighbour method, although it's a fast method it's not necessarily optimal, however it quickly picks a good enough point.

After the target point is selected, a path is generated towards the closest topological node, given that the topological graph quickly calculates the shortest path on the explored space.

The topological graph node to reach, are usually be the nearest, or the last explored, however some edge cases occur if these assumptions are taken, depending on the scenario the nearest node may not have LOS and require a longer path. For this reason, two different approaches were thought up, nearest-neighbour into RRT*, which should be quick to converge and optimize, and an RRT* towards the closest converging graph node, which in some circumstances could converge towards a farther node in some complex scenarios. This problem is mitigated in scenarios where the passages are narrow and the topological graph generates nodes frequently, such as the typical underground mine.

The RRT* as described previously in the Algorithm 3, takes into account the same parameters, but the implementations from here on forth also assume another stopping condition since we don't want to stop before reaching the target topological node.

Having reached a graph node, the next step is to traverse the graph towards the target, this can be done using Dijkstra as described in Algorithm 4, returning the shortest path between two nodes.

The target is picked in much the same way as the first one, either by nearest-neighbour or using a RRT* to the target frontier point.

4.4.3 Obstacle Detection

Another critical point is avoiding collisions in unknown scenarios, that is, keeping a distance from obstacles like walls and unknown space. Depending on the dimensions

of the occupancy map and the robotic platform, sampling the entire region around can be slow and inefficient, as such various rays are cast from the current position in various directions.

For this project the directions the rays are cast form a tessellated octahedron with its vertices aligned to the axis of the world, exploiting geometry in subterranean mines where the shafts are mostly perpendicular to the gravitational pull. The number of subdivisions for the tessellation heavily impacts processing time and number of samples, growing exponentially.

In some cases however, the sampling region is small enough or it's critical that it's sampled, one such example are collision detections for the RRT and frontiers, where for each sample, if an unknown or an occupied voxel lies in that region, the sample is discarded. For such a case voxels are selected around a radius.

These operations can very efficiently be optimized, firstly by keeping a lookup table stored in cache, and secondly because ray-casts often sample the same point multiple times when sampling near the source.

With the returned information the control system can be adapted to avoid obstacles detected in the map. The way this was achieved in this project was using a force field repulsing from the occupied and unknown space, using the sampling above mentioned, the resulting force can be achieved from the sum of all the inverted rays squared. This resulting force can in turn be used to avoid the nearest obstacles and tuned with a gain to effectively navigate through safe space.

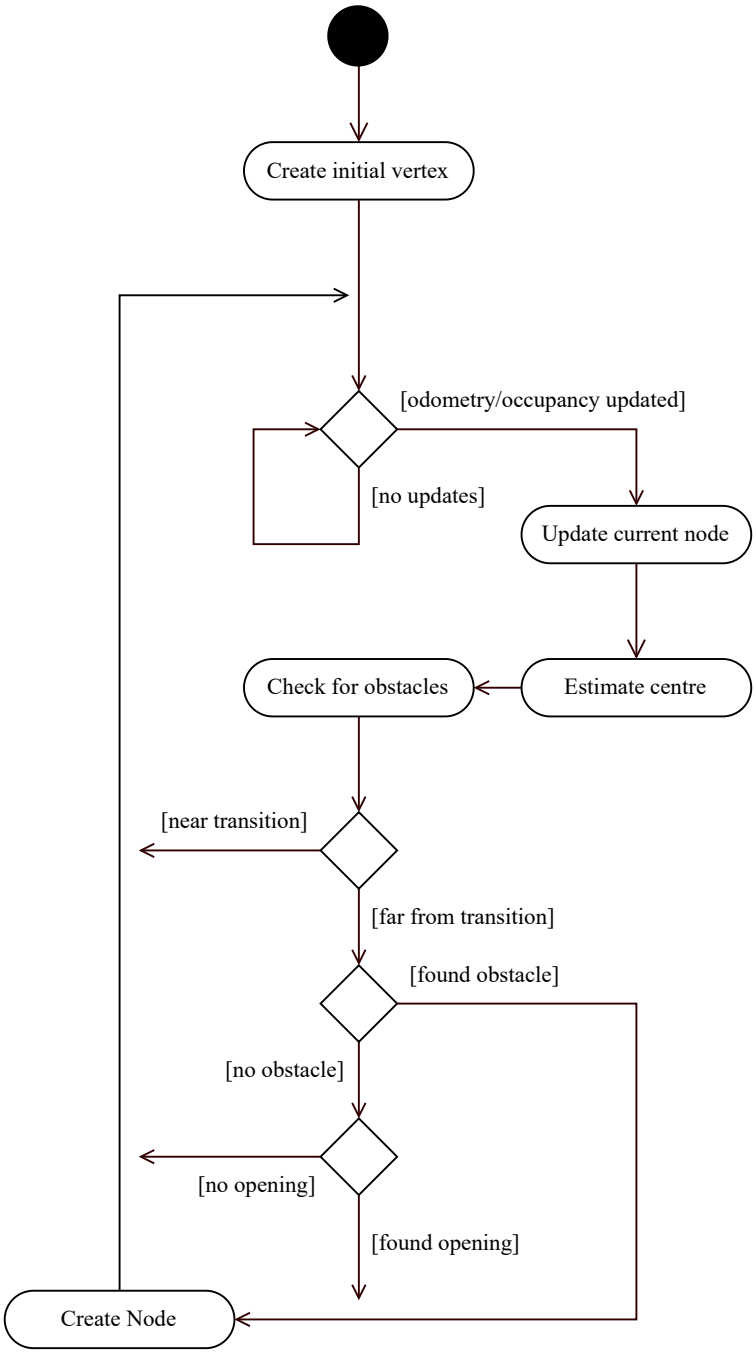


Figure 4.4: Topological node creation flow chart.

Chapter 5

Results

To test the system, the resulting scan from a previous mission in a real scenario with some added adaptations for non-existent scenarios such as caves was used. The existing mesh was then adapted closing off openings, avoiding erroneous scans such as scans that go through walls resulting from rendering issues with the normals on the opposite side of a surface.

The AUV performs scans of the scenario in the movement direction, never travelling blindly towards unknown space. The course starts in a vertical mine shaft, that connects to a crossing with four exits, two of them form a loop, one leads to a dead-end and the other connects to a cave generated to simulate great openings. The whole simulation, the roscore and the nodes were run in an Intel i7-8550U with 8GB of Random Access Memory (RAM).

5.1 Exploration

The generated frontiers during exploration through the test environment creates a small amount of points and correctly removes them after explored. After all the tests, all explorations ended with full coverage and no frontiers.

Obstacle avoidance during exploration yielded satisfying results, causing no collisions and never interrupting the traversal.

The generation time remained fairly consistent throughout the navigation being slightly slower when traversal is required, compared to simple LOS path generation.

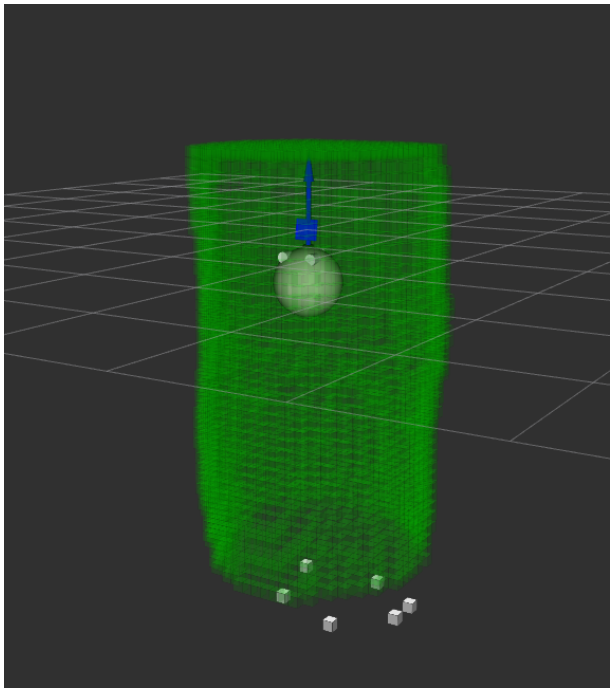


Figure 5.1: Frontiers generated represented as white cubes in a local map with obstacles in green.

5.2 Path Planning

As can be seen in Figure 5.2, Generated RRT* resulted in a quick and smooth path through already already explored scenario, and approaches the ideal path towards a waypoint around a 90º bend.

It can be observed that RRT* with few iterations is sufficient for such distances around obstacles, making them very capable in these scenarios where a partial occupancy map exists.

The graph traversal through extracted topological maps as can be seen in Figure 5.3, shows the successful pairing of the RRT* towards the nearest node, followed by the shortest path towards the node nearest to the target frontier and another RRT* closing the path.

It however also poses an edge case wherein two generated nodes do not have LOS caused by the inconsistent placement of a topological node after its detection, making the graph traversal not the shortest known path through the scenario.

5.3 Topological Map

The resulting tests, no topological transitions crossed the scenario, making the result a good map for navigation through the scenario, and it's structure closely resembles the topology.

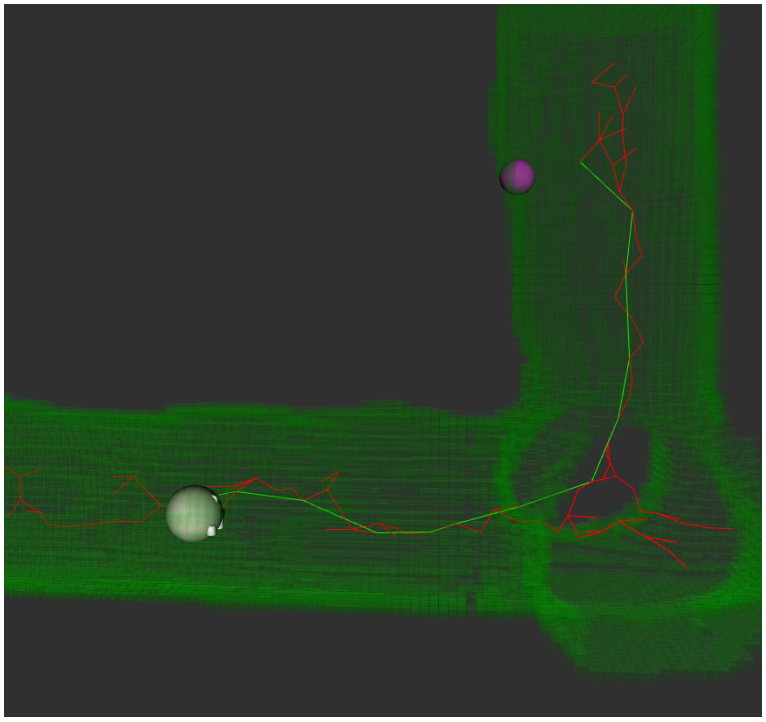


Figure 5.2: RRT* path generation from the robotic platform towards the purple waypoint, with initial samples connected by the edges in red, and the optimized path in green.

Observations during tests showed that the graph server which interfaces with the octomap directly used a full CPU core, while obstacle detection service used throughout most other nodes stayed at half a core. Other functions like frontier generation and planning combined, stayed well below half core usage and can still be parameterized to fit resources available.

Octomap's resource usage has previously been identified as a possible weak link in the processing pipeline, even at different resolutions. However, as the obstacle detection, avoidance as well as mapping can be merged using ESDF maps for less processor usage, it shows that the system can be further optimized.

Multiple complete explorations of the set scenario took on average 30 minutes. Variations of up to 5 minutes have been verified between tests, result of the sampling-based methods used.

As can be observed from the results in Figure 5.4, and as expected, larger openings increase point density, and generate a large unnecessary graph. This is caused by detecting an opening in multiple places inside the same opening, while ideally these points would be merged in their centre.

However, checking Figure 5.5, it can be verified that the number of topological nodes remains much lower compared to the occupancy map.

However running the multiple tests showed that the final graph node positions

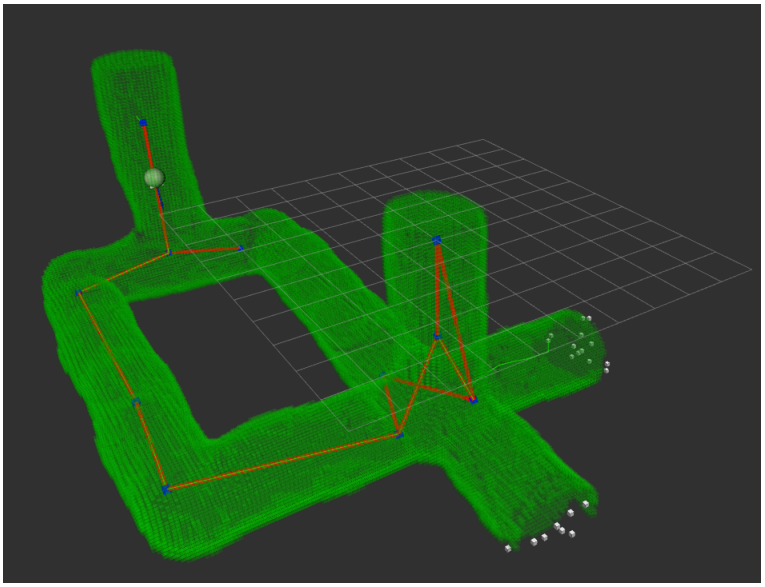


Figure 5.3: Example traversal in green through the graph to reach a frontier represented as white cubes

are not deterministic and change for each run, as well as its density, this is derived from the sampling methods to centre points, as well as the way points are selected in crossings or openings.

Ideally the preferred method to pick final node positions would be probabilistic and incrementally optimized by finding local minima away from nearby obstacles, this however should not be used every iteration as it computationally expensive.

For better visibility a simple teleoperated test was ran, generating a smaller amount of topological nodes and a smaller manageable graph shown below in Figure 5.6 and it's accompanying Comma Separated Values (CSV) file, File 5.1.

```
1 %id,pos_x,pos_y,pos_z,transitions
2 4,0.100000,6.575000,-4.825000,1
3 3,-3.638954,0.005809,-4.804358,1
4 2,3.581909,0.195561,-4.933206,1
5 0,0.000000,0.000000,0.000000,1
6 1,-0.168364,0.098206,-4.049659,0,2,3,4
```

File 5.1: Resulting graph CSV file.

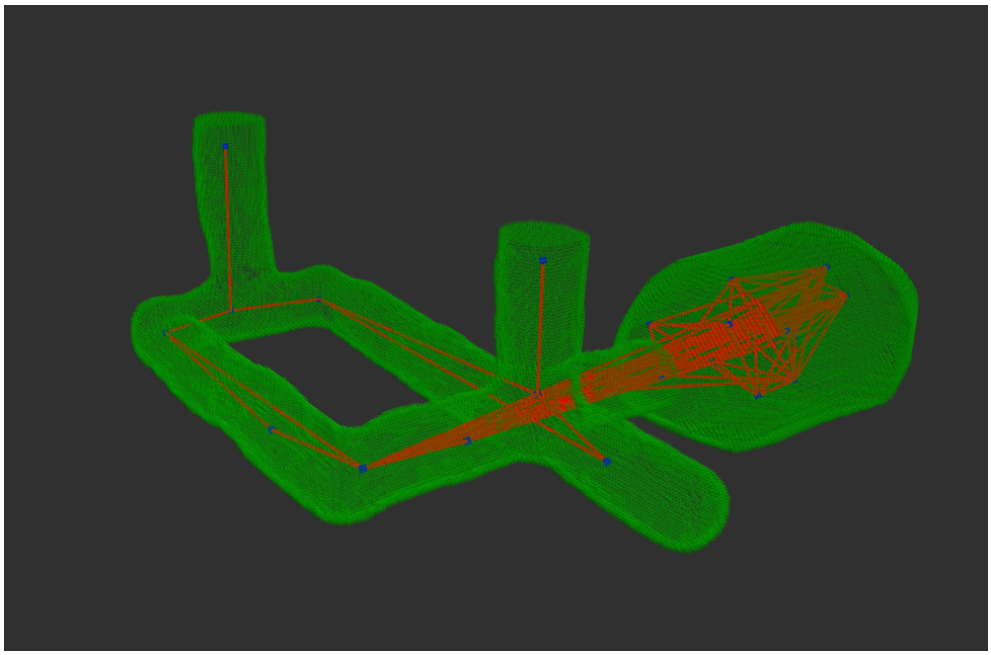


Figure 5.4: The resulting topological graph with nodes represented as blue squares and its transitions as red lines.

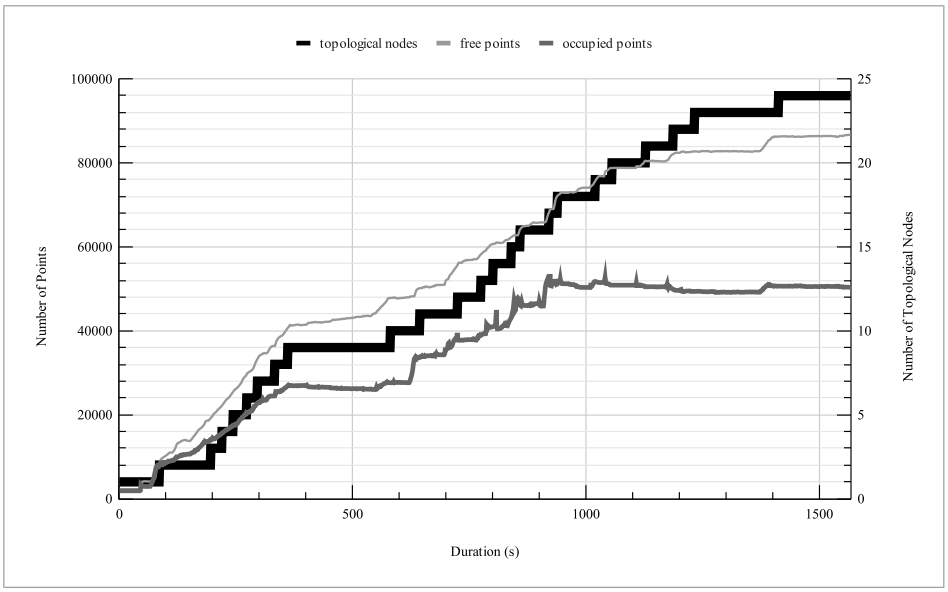


Figure 5.5: Point (representing voxel centres) and topological map node count over the duration of a test.

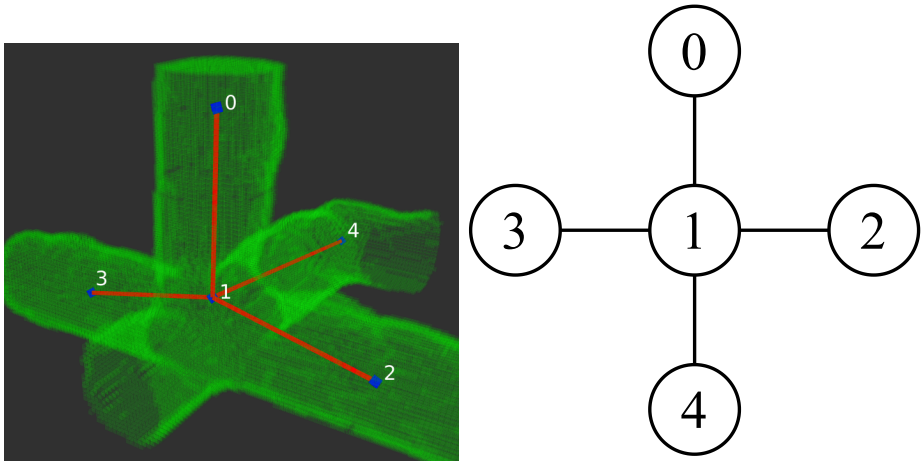


Figure 5.6: Small graph generated from a teleoperated mission (left), and extracted topology representation as a graph (right), IDs may not align with output received from autonomous control.

Chapter 6

Conclusions and Future Work

In this work, a new approach towards feature-based 3D topology extraction is proposed for use in autonomous robotic subterranean mine exploration. It was shown that using ray casting approaches, it is possible to identify topological features in underground mines, which can in turn be used to generate a graph containing the topology.

It was also verified that the proposed method can be integrated in AUVs and used alongside frontier-based exploration to autonomously explore a scenario, simplifying navigation through explored space with graph traversal algorithms.

As seen from the results, the main objectives of the project were met, having successfully generated a topological representation of the environment in a graph structure. The resulting graph is modular and can be expanded in the future to segment the scenario, storing association to sensor readings and point clouds to perform culling of non visible scenes for further optimization.

Accompanying this, a RVIZ interface is possible using the published markers for the graph nodes and transitions. This interface further possibilitates high level user input, via placement of waypoints in a transition or a vertex, making it simpler and intuitive to revisit explored regions.

Furthermore, exploration of virtual scenarios was held completely autonomously, requiring no human input to cover and scan the whole scene. With this an easier usage of a robotic platform is possible removing the requirement for complex and resource consuming AUV operator training.

The modular nature of ROS, makes this project easily adaptable and implementable in existing platforms as long as they have localization, range sensors and run on the same middleware.

Future work as described before would pick up on this project and improve it in either of its key areas ideally making its node placement deterministic.

Work can be done on optimizing the occupancy map, one suggestion being to test alternatives, with one of high interest being voxblox. Voxblox provides some great benefits when it comes to resource usage, requiring less time for each sensor reading integration, and providing ESDFs which further simplify and remove processing.

Furthermore with this work being tested in a simulation environment, it lacks validation in real scenarios and on real hardware where processing power may be limiting and is often already taken by other on-board processing.

References

[1] L. Lopes, N. Zajzon, S. Henley, C. Vörös, A. Martins, and J. M. Almeida, "Unexmin: a new concept to sustainably obtain geological information from flooded mines," *European Geologist Journal*, vol. 44, pp. 54–57, 11 2017. [Cited on page 2]

[2] L. Lopes, N. Zajzon, B. Bodo, S. Henley, G. Žibret, and T. Dizdarevic, "Unexmin: developing an autonomous underwater explorer for flooded mines," *Energy Procedia*, vol. 125, pp. 41–49, 09 2017. [Cited on pages 2, 4, 5, and 10]

[3] M. T. Pinto, G. Žibret, L. Lopes, B. Bodo, and N. Zajzon, "Unexup: robot-based exploration technology for underground flooded mines," *Advances in Geosciences*, vol. 54, pp. 109–117, 2020. [Cited on pages 2 and 6]

[4] "Darpa subterranean challenge." `https://subtchallenge.com`. Accessed: 2022-02-03. [Cited on pages 3, 9, and 20]

[5] J. H. Collins, *Principles of metal mining*. Collins' elem. sci. series, W. Collins, 1874. [Cited on page 3]

[6] S. Timberlake, "Prehistoric copper extraction in britain: Ecton hill, staffordshire," *Proceedings of the Prehistoric Society*, vol. 80, pp. 159–206, 12 2013. [Cited on page 3]

[7] "Chatsworth and buxton: built on copper!," *Reflections*, p. 28–31, 8 2013. [Cited on page 3]

[8] S. Henley, J. Barnatt, R. Shaw, G. Žibret, E. Pučko, H. van Moerkerk, M. McLoughlin, J. Tweedie, L. Lopes, A. Sanchez, and N. Zajzon, "Unexmin d7.6 – geoscientific evaluation of pilots," tech. rep., 9 2019. [Cited on page 3]

[9] G. Žibret, A. Martins, C. Almeida, D. Lombarinhas, E. Soares, E. Pučko, H. van Moerkerk, J. Almeida, J. Aaltonen, L. Lopes, M. L. Kiss, M. Koba, N. Zajzon, R. Suarez, R. Pereira, R. Papp, S. Henley, and Z. Milošević, "Unexmin d7.3 – pilot report from urgeiriça mine, portugal," tech. rep., 5 2019. [Cited on pages 3 and 31]

[10] R. A. S. Fernandez, Z. Milošević, S. Dominguez, and C. Rossi, "Motion control of underwater mine explorer robot ux-1: Field trials," *IEEE Access*, vol. 7, pp. 99782–99803, 2019. [Cited on pages 3 and 5]

[11] "Unexmin - underwater explorer for flooded mines." `https://www.unexmin.eu/`. Accessed: 2022-02-03. [Cited on pages 5 and 10]

[12] "Unexup." `https://unexup.eu/`. Accessed: 2022-02-03. [Cited on pages 6 and 10]

[13] J. Almeida, A. Martins, C. Almeida, A. Dias, B. Matias, A. Ferreira, P. Jorge, R. Martins, M. Bleier, A. Nuchter, J. Pidgeon, S. Kapusniak, and E. Silva, "Positioning. navigation and awareness of the !vamos! underwater robotic mining system," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1527–1533, 2018. [Cited on page 6]

[14] A. Martins, C. Almeida, P. Lima, D. Viegas, J. Silva, J. M. Almeida, C. Almeida, S. Ramos, and E. Silva, "A robotic solution for nettag lost fishing net problem," in *Global Oceans 2020: Singapore – U.S. Gulf Coast*, pp. 1–6, 2020. [Cited on page 7]

[15] T. Chung, "Darpa subterranean (subt) challenge." `https://www.darpa.mil/program/darpa-subterranean-challenge`. Accessed: 2022-02-07. [Cited on pages 9, 10, and 20]

[16] Y. Roth-Tabak and R. Jain, "Building an environment model using depth information," *Computer*, vol. 22, no. 6, pp. 85–90, 1989. [Cited on pages 10 and 11]

[17] A. S. Glassner, "Space subdivision for fast ray tracing," *IEEE Computer Graphics and Applications*, vol. 4, no. 10, pp. 15–24, 1984. [Cited on page 11]

[18] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Autonomous Robots*, 2013. Software available at `https://octomap.github.io`. [Cited on pages 11, 12, and 25]

[19] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017. [Cited on pages 12 and 25]

[20] "Voxblox - voxblox documentation." `https://voxblox.readthedocs.io/`. Accessed: 2022-02-03. [Cited on page 13]

[21] S. Thrun, "Learning metric-topological maps for indoor mobile robot navigation," *Artificial Intelligence*, vol. 99, no. 1, pp. 21–71, 1998. [Cited on pages 12 and 13]

[22] S. Thrun and A. Bücken, "Integrating grid-based and topological maps for mobile robot navigation," in *AAAI/IAAI, Vol. 2*, 1996. [Cited on pages 12 and 13]

[23] Y. Chen, J. Zhang, and Y. Lou, *Topological and Semantic Map Generation for Mobile Robot Indoor Navigation*, pp. 337–347. 10 2021. [Cited on pages 12 and 13]

[24] F. Blochliger, M. Fehr, M. Dymczyk, T. Schneider, and R. Siegwart, "Topomap: Topological mapping and navigation based on visual slam maps," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3818–3825, 2018. [Cited on page 13]

[25] Z. Zivkovic, B. Bakker, and B. Krose, "Hierarchical map building using visual landmarks and geometric constraints," pp. 2480 – 2485, 09 2005. [Cited on pages 13 and 14]

[26] A. Tagliasacchi, H. Zhang, and D. Cohen-Or, "Curve skeleton extraction from incomplete point cloud," in *ACM SIGGRAPH 2009 Papers*, SIGGRAPH '09, (New York, NY, USA), Association for Computing Machinery, 2009. [Cited on page 14]

[27] J. Bloomenthal, "Modeling the mighty maple," in *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '85, (New York, NY, USA), p. 305–311, Association for Computing Machinery, 1985. [Cited on page 14]

[28] L. Fu, J. Liu, J. Zhou, M. Zhang, and Y. Lin, "Tree skeletonization for raw point cloud exploiting cylindrical shape prior," *IEEE Access*, vol. 8, pp. 27327–27341, 2020. [Cited on page 14]

[29] S. M. LaValle, "Rapidly-exploring random trees : a new tool for path planning," *The annual research report*, 1998. [Cited on page 15]

[30] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *CoRR*, vol. abs/1105.1186, 2011. [Cited on pages 15 and 16]

[31] T. Dang, M. Tranzatto, S. Khattak, F. Mascarich, K. Alexis, and M. Hutter, "Graph-based subterranean exploration path planning using aerial and legged robots," *Journal of Field Robotics*, vol. 37, no. 8, pp. 1363–1388, 2020. Wiley Online Library. [Cited on pages 16, 19, 20, and 29]

[32] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. 'Towards New Computational Principles for Robotics and Automation'*, pp. 146–151, 1997. [Cited on page 18]

[33] A. Martins, J. Almeida, C. Almeida, and E. Silva, "Uxnexmin auv perception system design and characterization," in *2018 IEEE/OES Autonomous Underwater Vehicle Workshop (AUV)*, pp. 1–7, 2018. [Cited on pages 19 and 29]

[34] D. Sytnyk, R. Pereira, D. Pedrosa, J. Rodrigues, A. Martins, A. Dias, J. Almeida, and E. Silva, "Simulation environment for underground flooded mines robotic exploration," in *OCEANS 2017 - Aberdeen*, pp. 1–6, 2017. [Cited on pages 19, 20, 21, and 30]

[35] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, pp. 2149–2154 vol.3, 2004. [Cited on pages 19 and 20]

[36] M. Prats, J. Pérez, J. J. Fernández, and P. J. Sanz, "An open source tool for simulation and supervision of underwater intervention missions," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2577–2582, 2012. [Cited on page 19]

[37] G. Echeverria, N. Lassabe, A. Degroote, and S. Lemaignan, "Modular open robots simulation engine: Morse," in *Proceedings of the IEEE ICRA*, 2011. [Cited on pages 19 and 20]

[38] W. Yao, W. Dai, J. Xiao, H. Lu, and Z. Zheng, "A simulation system based on ros and gazebo for robocup middle size league," in *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 54–59, 2015. [Cited on page 20]

[39] "What is morse?." `https://www.openrobots.org/morse/doc/latest/what_is_morse.html`. Accessed: 2022-03-03. [Cited on page 21]

[40] L. Fu, J. Liu, J. Zhou, M. Zhang, and Y. Lin, "Tree skeletonization for raw point cloud exploiting cylindrical shape prior," *IEEE Access*, vol. PP, pp. 1–1, 02 2020. [Cited on page 21]

[41] T. H. Cormen, C. E. Leiserston, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. The MIT Press, third ed., 2009. [Cited on pages 21, 22, and 23]

[42] M. T. Goodrich and R. Tamassia, *Algorithm Design and Applications*. Wiley, 2014. [Cited on pages 21, 22, and 23]

[43] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Ng, "Ros: an open-source robot operating system," vol. 3, 01 2009. [Cited on page 29]