*Journal of Field Robotics*

WILEY

RESEARCH ARTICLE OPEN ACCESS

# Autonomous Navigation in Large-Scale Underground Environments Based on a Purely Topological Understanding of Tunnel Networks

Lorenzo Cano[1,2] | Danilo Tardioli[2,3] | Alejandro R. Mosteo[1,2]

[1]Departamento de Informática e Ingeniería de Sistemas, University of Zaragoza, Zaragoza, Spain | [2]Instituto Universitario de Investigación en Ingeniería de Aragón (I3A), University of Zaragoza, Zaragoza, Spain | [3]Centro Universitario de la Defensa, University of Zaragoza, Zaragoza, Spain

**Correspondence:** Lorenzo Cano (lcano@unizar.es)

**Received:** 10 June 2025 | **Revised:** 3 December 2025 | **Accepted:** 31 December 2025

**Funding:** Spanish Ministry of Science, Innovation and Universities (MICIU), Grant/Award Number: D2022-139615OB-I00

**Keywords:** neural-based-perception | topological-mapping | topological-navigation | underground-environments

## ABSTRACT

This work presents a non-geometrical navigation approach based on a purely topological understanding of underground environments. By conceptualizing subterranean scenarios as a set of tunnels that intersect with each other, and taking a navigation approach based on topological instructions, we simplify the navigation problem to the sequential execution of human-understandable instructions. This approach is built on top of a lightweight Convolutional Neural Network (CNN) that processes the readings of a 3D LiDAR sensor and produces an estimation of the angular positions of the surrounding tunnels with respect to the robot. As a result of this approach, our method can navigate these underground environments by only being provided with the necessary topological instructions, without the need for a map, or for building one during navigation. Additionally, it can also rely on a lightweight graph representation of the environment. This graph can be either defined by the user, generated during navigation or explicitly built in an exploration task. To showcase these capabilities, this article provides an experimental evaluation of the method both in simulation and in a real environment.

## 1 | Introduction

Subterranean environments are interesting targets for automation, as they are an important part of our infrastructure and economic development, but, at the same time, they share certain characteristics that make them hostile environments to human workers. One example of this are underground mining operations. In these environments, the presence of dust and fine particles greatly increases the risk of respiratory diseases in workers (Ross and Murray 2004), the operation of heavy machinery and explosives poses a significant risk of traumatic injury (Mitchell et al. 1998) and the oppressive environment can lead to a multitude of psycho-social illnesses (Donoghue 2004). Other important examples are utility tunnels in cities, which carry crucial distribution lines, like electricity, gas, water etc., and are thus an important infrastructure for the functioning of any large urban area. Given their criticality, they require frequent inspection and maintenance by qualified operators, but, due to their scale (from the 10 s to 100 s of km), these inspections are time consuming, expensive, and require special equipment—like ultrasonic sensors—to detect defects. These requirements imply that operators must be exposed to the hazards of these environments (e.g., gas and electricity lines) for long periods of time. The costs and hazards associated with this activity have motivated the development of robots adapted to execute these tasks. However, given the many challenges of autonomous navigation in these environments, they still require

the presence and supervision of human operators (Montero et al. 2015).

All these factors make the reduction of human presence in these environments a desirable goal and, given that most of the work is already done by machines operated by workers (heavy machinery in mines and inspection equipment in tunnels), the most straightforward solution would be their remote operation. However, telecommunications in these environments are notoriously challenging, making teleoperation rarely viable, as connecting a robot with a base station (frequently outside the tunnel network) involves great effort, and might not even be possible. The use of cables, for example, is often unfeasible, due to the large scale of these environments, and because the operation of more than one robot would likely lead to tangles. The use of wireless technologies is also problematic. The structure of these environments, usually composed of long tunnels that intersect with each other, makes the wave propagation unpredictable, due to the destructive interferences that cause the fading effect. This often results in reduced bandwidth and unexpected losses of connection, even if a line-of-sight is still present, as demonstrated in Rizzo et al. (2013).

The drive for automation and the limitations of teleoperation have spurred efforts to create autonomous systems tailored specifically to these environments. For example, in 2018, DARPA[1] launched the Subterranean Challenge[2] with the objective of encouraging the development of new technologies in the field of autonomous underground exploration. In this challenge, 20 different teams worked on new solutions addressing the unique challenges that subterranean scenarios pose to autonomous robots, with special focus on the autonomy, perception, networking and mobility aspects

The goal of the competition was to autonomously explore an unknown underground environment with a team of robots, and to report back to base the location of certain pre-defined elements (humans, cellphones, backpacks...) with a certain level of accuracy. During the 3 years that the competition lasted, the teams made important advancements in the field of underground robotics and pushed it forward significantly. However, given the constraints of the competition, there was an over-representation of the approaches best suited to the main objective. More specifically, the requirement to accurately report a precise location of each of the artifacts made the use of geometrical mapping methods implicitly necessary, and, given the limited size of the competition scenarios, minimal emphasis was put on the efficient representation of the environment. All these factors have meant that, while geometrical methods have been thoroughly explored and refined, topological approaches have received little attention.

Within this context, this work aims to explore the use of topological approaches to address the specific challenges that underground environments pose to autonomous systems as well as to exploit their underlying structure. With this in mind, what we propose is a complete framework that:

- Makes use of a purely topological, graph-based representation of the environment that, without storing any metric data, is capable of producing high-level topological plans based on simple, human-readable instructions.
- Has a perception approach based on a Convolutional Neural Network that can detect the distribution of the galleries around the robot by processing a 3D LiDAR scan. This work provides detailed information about how this network has been trained using a fully procedurally generated synthetic dataset but can be used in real-world environments with a zero-shot transfer.
- Is capable of navigating by following high-level topological instructions (e.g. take the second exit on the left and then the third on the right), operating in arbitrarily large environments without the need for precise self-localization. This removes one of the main challenges that underground environments pose, and allows a human operator with rough knowledge of the environment layout to define a mission as a set of topological instructions, without providing a map to the robot.
- Can build topological representations of the environment during navigation (topological mapping), or explicitly explore acyclic environments in a fully autonomous fashion.

This proposal is a direct evolution of two of our previous works (Cano et al. 2022), (Cano et al. 2024b), that have been enhanced and extended with the following contributions:

- The topological representation has been simplified. In our previous works, tunnels were considered as nodes, which meant that the environmental representation was more complex, and the topological navigation task involved more instructions than necessary.
- The topological navigation has been enhanced with the introduction of hybrid instructions, that allow the user a finer control over the behavior of the robot.
- The dataset generation has been completely modified, significantly increasing its quality. This, in turn, has allowed for the reduction of network size, and thus the computational requirements.
- In this work, the CNN has been trained in fully procedural environments generated using our own tool (Cano et al. 2024a), allowing for a greater diversity in the training dataset. In consequence, the trained model presents a zero-shot transfer learning capability from simulation to a real-world environment.
- This work introduces the concept of *prediction stability* with regards to the outputs of the CNN, improving its reliability in intersections, and removing tunable parameters that were present in our previous works.
- Finally, this work provides an experimental evaluation of both the navigation and mapping approaches in a challenging real-world environment, as reported in Section 5.2, in which conventional navigation solutions struggle.

## 2 | Motivation

It is our view that, in many real-world use cases, topological approaches offer significant advantages for navigating underground environments over geometric ones.

The main difference between geometrical and topological approaches is how the environment is represented. In geometrical approaches the environmental representations consist on data

structures that store features with an associated pose. This way, when the robot detects a previously mapped feature it can infer its own position. Within this definition, there are two main families of mapping methods. The first one are the *occupancy grids*, which are based on the discretization of the environment into a 2D or 3D grid, where each cell of the grid typically contains occupancy information (whether a cell is free space or contains an obstacle). These methods tend to be used with range-based sensors (LiDARs or Depth Cameras), and their memory footprint tends to increase with the volume of the explored environment (or horizontal surface if a 2D map is used). On the other hand, *feature maps* contain a set of features, each with an assigned location in the environment. These features are typically extracted from images, so they are necessarily associated with a surface, which means that these kinds of representations typically increase their memory footprint linearly with the number of surfaces of the environment that have been seen.

In contrast to geometrical methods, topological approaches represent the environment as a graph, which, from now on, will be referred to as a topological map. In this map, nodes represent a place or position in the environment and typically contain a set of coordinates along with additional information, such as a point cloud or an image, to assist in localizing the robot with respect to the node. Edges represent, instead, the traversability between nodes, typically meaning that a node is directly reachable from another node. As a general rule, this type of maps tend to provide less accurate position estimates, as they are more sparse than geometrical ones but they are more lightweight and computationally efficient in path planning, which becomes more relevant as the size of the environment increases. Additionally, topological maps facilitate the inclusion of semantic information and relationships between semantic elements, so they are proven to be the preferred approach in semantically-aware navigation approaches (Kostavelis and Gasteratos 2015). There are a number of reasons why topological maps are a good fit for underground environments in comparison with geometrical ones, but the essential one is that most human-made underground environments share an underlying structure of tunnels that intersect each other (Figure 1), that can be directly mapped into a graph representation. By interpreting the tunnels as edges and the intersections and dead ends as nodes, a simple yet complete topological map of the environment is obtained, as seen in Figure 2. This simplified representation has some interesting consequences:

- Given its simplicity, it is feasible for a human operator to define the layout of the environment. This could prove valuable in emergency situations, where the time needed to map the environment may not be available.
- Another consequence of this representation is the significant decrease in computational complexity and memory footprint of the map. As mentioned before, geometrical

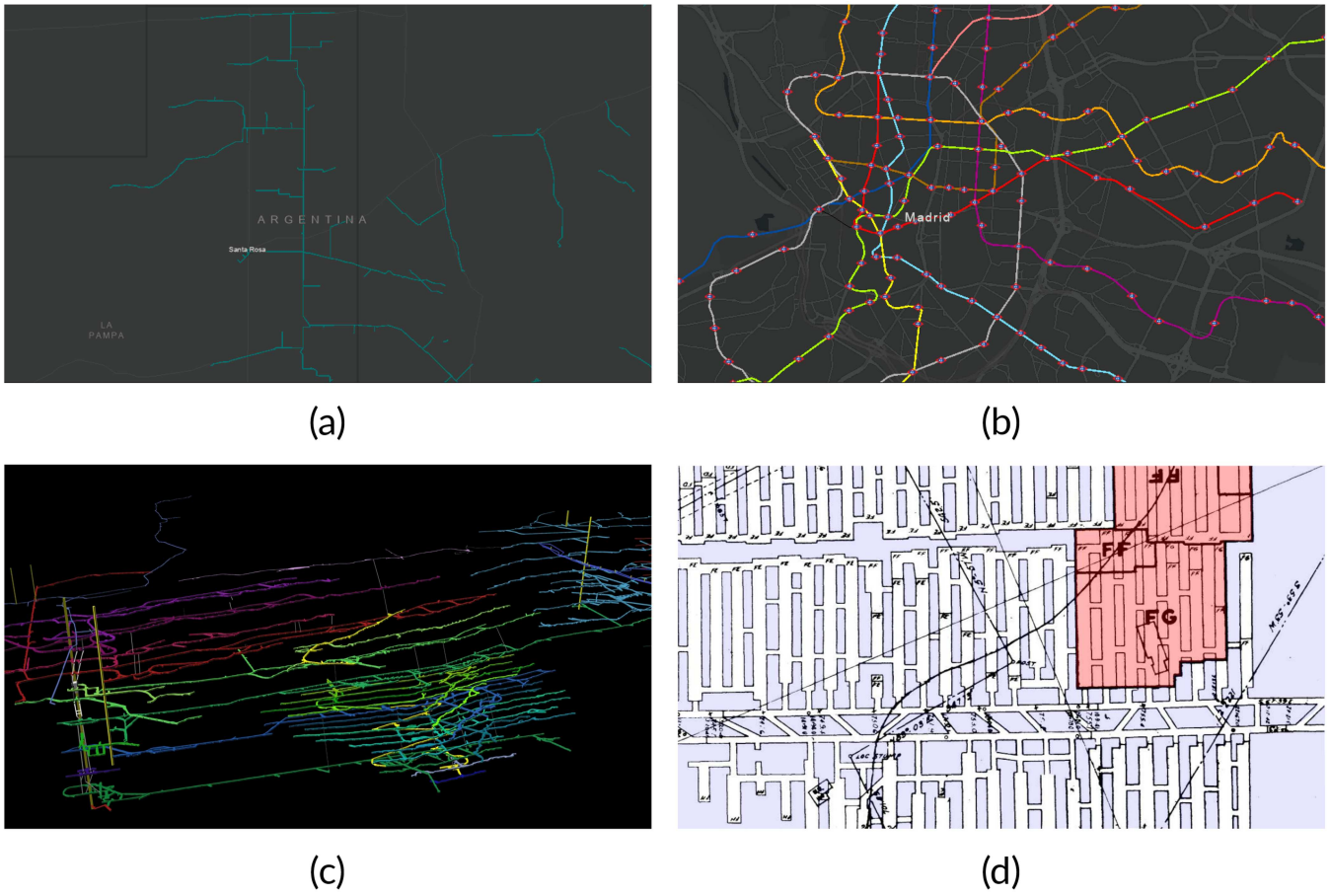


**FIGURE 1** | Man-made underground environments share the same underlying structure of intersecting tunnels. a) Underground waterways in Argentina, b) Metro Network in Madrid c) 3D model of a mine in Cornwall, UK d) Map of a mine in Pennsylvania, US. [Color figure can be viewed at wileyonlinelibrary.com]

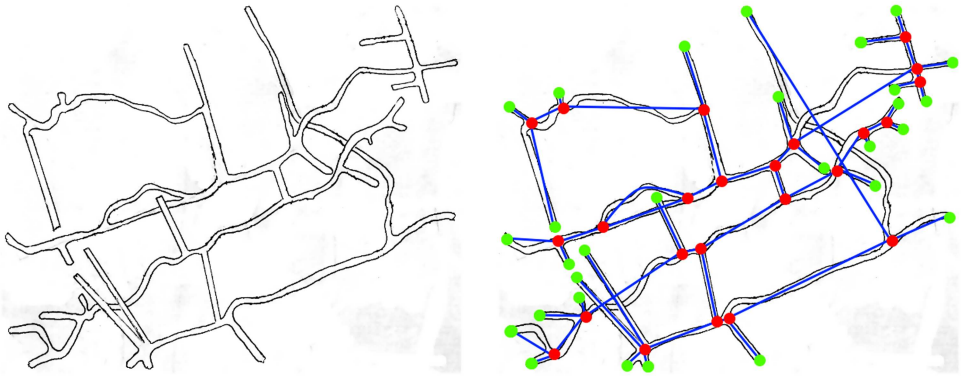


**FIGURE 2** | Simplicity of our topological map. a) Layout of a real mine. b) Hand-made topological representation of said mine where blue lines are the edges, red dots are the nodes associated to intersections and green dots are nodes associated with the dead-end. [Color figure can be viewed at wileyonlinelibrary.com]

maps memory footprint increases with the size of the environment. This is easily manageable in most indoor environments, but can become an issue when reaching the order of tens, hundreds or even thousands of kilometers of tunnels. In contrast, our proposal increases in size linearly with the number of tunnels and intersections regardless of the length of the tunnels, which would be advantageous in large-scale environments like mines. Additionally, this would reduce significantly the bandwidth requirements to exchange the map with the base station or among members of a hypothetical robot team.

- The final characteristic of our approach is that precise self-localization is no longer needed, and it is instead replaced with a system that detects when the robot is in a tunnel or intersection and can follow high-level instructions of the sort "in the fifth intersection, take the tunnel to the right." This can be a significant advantage over metric methods, as maintaining a consistent and precise enough self-localization is one of the main challenges that our target environments pose.

As a tradeoff of operating in such environments, our method is tailored to their particular characteristics and, consequently, cannot operate in more general environments whose structure cannot be recognized as a set of intersecting tunnels, galleries, or corridors. In this sense, the approach is not a general-purpose method, but a specialized solution. Nevertheless, this specialization makes it particularly useful, or even necessary, in parts of environments unsuited for more general approaches, be it because of lack of features, or because there is no map available to the robot, so defining a goal is not possible.

## 2.1 | Localization in Underground Environments

One of the biggest challenges for mobile robotics in underground environments is reliable self-localization (Tardioli et al. 2019). Self-localization in robotics involves determining the robot's position by comparing detected environmental features with those in a reference map. In underground settings, this process is particularly challenging.

The localization process infers the robot's position based on detected features. When exploring new areas, the absence of pre-existing features can lead to discrepancies between the estimated and actual positions. In typical environments, these errors are corrected when the robot revisits a known location. However, in underground environments like tunnels, which can extend for kilometers, these errors accumulate, making accurate self-localization difficult and potentially unmanageable.

To illustrate this, we have tested five different SLAM (Simultaneous Localization And Mapping) systems on real-world data captured on the Somport Tunnel, a 7 km, feature-deprived tunnel with 19 lateral galleries (see Section 5 for details). This makes it especially challenging for traditional, geometry-based localization methods. These methods are *Gmapping* (Grisetti et al. 2007), *Hector-SLAM* (Kohlbrecher et al. 2011), *Fast-LOAM* (Wang et al. 2020), *Faster-LIO* (Bai et al. 2022) and *Direct LiDAR Odometry* (Chen et al. 2022).

In the first dataset, used for the Gmapping, Hector-SLAM and Fast-LOAM methods, the robot starts at the end of one of the lateral galleries, advances to the main tunnel, and navigates to the end of the lateral gallery to the right. From there, it goes back to the main tunnel, advances to the original gallery, skips it, and enters into the next lateral gallery. Finally, it goes back to the main tunnel, advances to the intersection between the original gallery and the main tunnel and stops there. The second dataset contains additional IMU measurements and is used for the Faster-LIO and Direct LiDAR Odometry methods. The robot starts from the main tunnel, advances skipping the first gallery, and enters the next two until reaching the end of both. Then it advances 100 m more through the main tunnel, before turning around and returning to the starting point. The results of these tests are shown in Figure 3.

In the case of Gmapping (Figure 3a). The result is the best of the three methods because it takes advantage of odometry readings, but ends up failing due to the accumulation of error, which causes the duplication of a lateral gallery and the creation of many duplicate niches on the map.

Figure 3b shows the result of using the hector-SLAM system. This method relies exclusively on a LiDAR scan, and as such, is incapable of reliably detecting the advancement of the robot through the tunnel, resulting in an unusable map.

Figure 3c shows the results of using Fast-LOAM. This method relies exclusively on a 3D LiDAR for localization, and, similarly to hector-SLAM, fails to detect when the robot is advancing through the tunnel, and ends up super-imposing all the lateral galleries at the same position.

The results of the Faster-LIO simulation can be appreciated in Figure 3d. This method relies on IMU measurements and 3D LiDAR data. The IMU, as can be seen, helps significantly with regards to aligning the LiDAR measurements so that they are consistently placed. However, it also fails to detect the advancement of the robot through the tunnel, so the distance between lateral galleries, and the length of the lateral galleries themselves is significantly smaller than in reality.

The Direct LiDAR Odometry method is the most successful at accounting for the small features in the tunnel, and manages to detect the advancement of the robot through the tunnel. However, in the experiments conducted in our environment, it showed a tendency to "detach" from the map and oscillate violently back and forth when map-to-LiDAR matching fails.

This produces both a trajectory much larger than real, and an incorrect reconstruction of the environment, in which the main tunnel is much longer than in reality (Figure 3e).

These are problems recognized by the underground SLAM research community (Ebadi et al. 2022), and is still an active area of research, with substantial progress being made every year. In contrast, this work intends to explore an alternative solution to these problems by removing the need for precise self-localization and exploiting the inherently graph-like structure of such underground environments.

The rest of this article is structured as follows: Section 3 provides an overview of the field of underground navigation systems and other topological navigation approaches, Section 4 contains a detailed explanation of our proposed method, Section 5 showcases the experimental evaluation that has been carried out, Section 6 comments on the results of the tests and the lessons learned, and in Section 7 we provide our conclusions and final thoughts.

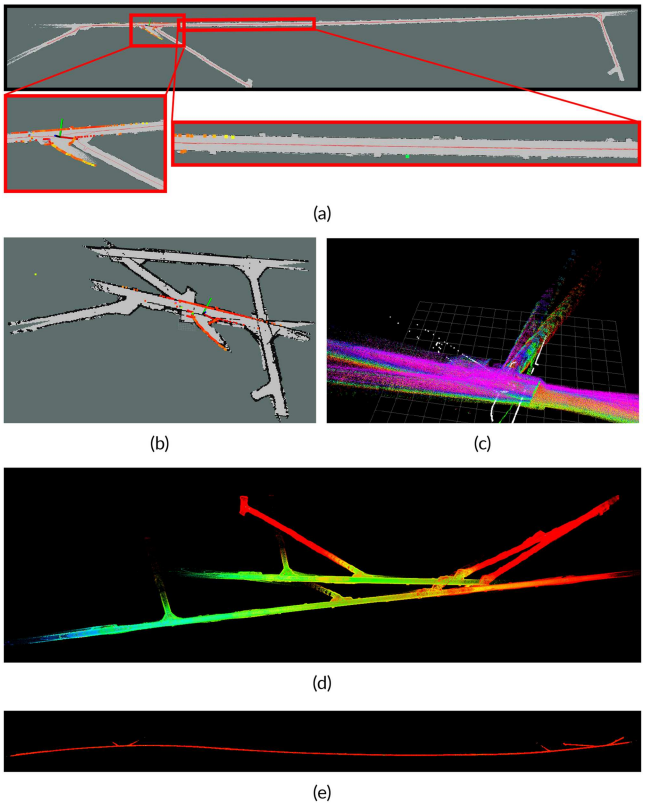


**FIGURE 3** | Results of geometrical mapping approaches in an environment with long, feature-deprived tunnels. The robot starts at the end of the central gallery, goes to the end of the right gallery, then enters 20 m into the left gallery, and comes back to the intersection between the central gallery and the main tunnel. a) Gmapping obtains the best result due to the use of odometry, but fails to maintain a correct localization, causing it to detect the lateral gallery at an incorrect location. b) Hector-SLAM fails to detect when the robot advances through the tunnels, resulting in large errors in map generation. c) In this run, the robot visits two lateral galleries, but Fast-LOAM also fails to detect when the robot advances through the tunnel, and ends up superimposing the two intersections. d) Faster-LIO fails to detect the advancement trough the corridors, making the main tunnel appear much shorter than in reality. It also has trouble with the estimation of the heading in the return, causing dealignment. e) The Direct LiDAR Odometry method detects the advancement through the tunnel, however a combination of vibrations on the IMU and a detachment from the generated map cause the method to overestimate the advancement trough the tunnel. [Color figure can be viewed at wileyonlinelibrary.com]

## 3 | Related Work

This work finds itself at the intersection between topological navigation and mapping, autonomous navigation in underground environments and CNN-based perception for robotics.

As previously mentioned in this work, the tasks of self-localization and navigation in underground scenarios are challenging, especially in comparison to other more structured environments. This has motivated the robotics community to develop methods tailored to these types of environments, which generally rely on 3D LiDAR sensors, as cameras tend to perform poorly, due to the low-light conditions. However, even with LiDAR sensors, the presence of long and featureless tunnels still presents a challenge, which typically requires the tuning and refining of established methods to work correctly (Prados Sesmero et al. 2021; Ren et al. 2019; Yang et al. 2022; Chen et al. 2023)

It is generally recognized by the robotics community that representing an environment as a graph of places and connections can be more efficient than building a detailed, globally consistent geometrical map and be more resilient to error accumulation. Additionally, these representations allow for more efficient path computations, so they are commonly used in conjunction with geometrical SLAM approaches. In these hybrid systems, the robot constructs simultaneously a geometrical map using SLAM and a topological pose-graph to aid in path-planning, exploration or self-localization (Dang et al. 2020; Duberg and Jensfelt 2022; Blochliger et al. 2018; Fredriksson et al. 2024; Montano-Oliván et al. 2024; Xue et al. 2020; Placed et al. 2022).

These efficiencies become more pronounced in corridor-based environments, as certain underground environments tend to be, reason why these approaches are more represented among

underground robotics applications. Another interesting property of the topological representation of environments is the flexibility that the graph structure provides, as nodes can contain any arbitrary type of information (geometric position, images, point clouds, neural embeddings, semantical labels, etc.) and edges can represent different relationships between nodes (distance, reachability, traversal time, distance, semantic relationship, etc.). This flexibility is especially useful in the case of hierarchical semantic graphs, where, starting from a semantically-labeled occupational map, different clusters of semantically-related nodes are grouped (Hughes et al. 2022).

An interesting approach to topological maps is presented in Chaplot et al. (2020), where nodes contain images and edges communicate traversability, but no explicit geometric information is stored, as the authors propose the use of a Neural Network to directly obtain the velocity commands to go from the current position to a certain node. In a similar way, the authors in Li et al. (2023) also store image data in each of the nodes to help with place recognition, although they rely on more classical image descriptors to determine similarity.

A more extended approach is the use of hybrid, topo-metric methods (Blanco et al. 2008; Muravyev et al. 2025; Bosse et al. 2003), where nodes are considered local coordinate frames with a local map associated. This way, the robot can self-localize only w.r.t the local map improving the resilience against noisy estimations. However, they do not explicitly deal with feature poor environments, as they still require a geometrical pose estimation, whereas our proposal, by not depending on any type of geometrical pose estimation is not affected.

One of the first works exploring a mainly topological representation of an underground mine can be found in Morris et al. (2005), where the authors followed a similar representation as the one we propose (an acyclic graph with corridors as edges and dead-ends and intersections and nodes). However this approach uses a geometrical approach for node detection that could become unreliable in presence of obstacles. Other authors defend that, in the case of robotic operations in underground mines, centimeter-level accuracy is unnecessary, and that the focus must be put on safely traversing the tunnels and correctly exiting the intersections (Tampier et al. 2021; Mascaró et al. 2021). For these reasons, the authors propose the use of a two-tiered level topological map that, on one level, contains the overall topological structure of the entire environment, while each of the intersection nodes also contain a sub-graph that aids in executing the complex maneuvers that are needed when operating a large haul-dumping vehicle through narrow tunnels. These works serve as the foundation upon which our earlier research (Cano et al. 2022) was constructed, which in turn forms the basis for this current proposal.

One of the main benefits that comes from a lightweight representation of the environment is that sharing said map can be done in a much faster fashion (Cowley et al. 2011; Chang et al. 2007). This is especially relevant for underground scenarios, where the communication between robot team members is one of the central challenges (Tardioli et al. 2016). For this reason, topological maps are being explored as a way of communicating essential information about the structure of the environment among teams of robots without having to share the complete geometrical map. This becomes more advantageous if the robots operate in low-bandwidth environments, as is often the case in underground environments (Bayer and Faigl 2021).

Similar to our approach, the authors in Worley and Anderson (2025) use the inherently graph-like structure of pipe networks to bypass the challenges they pose to self localization. By enforcing certain assumptions (the robot only turns in intersections, and only advances in pipes) over the possible evolution of the robot state, they manage to use the Viterbi algorithm to discriminate between multiple hypotheses of the robot trajectory.

Given the robustness that topological approaches provide against localization errors, they have also been used in sensing-denied environments, as is the case in underwater robotics. In Rossi et al. (2023), the authors used a pre-built topological map to successfully navigate a flooded mine environment. They exploit the flexibility that graph representations provide to store relevant sonar data in each of the nodes of the map, which allows the robot to localize without the need of a precise geometric map, which is notoriously difficult to build with the use of sonar sensors (Ribas et al. 2006). Or in the case of (Morlana et al. 2024), they make use of topological mapping methods to deal with the severe uncertainty that visual mapping inside organic tissue imposes.

Additionally, the topological understanding of underground environments has been demonstrated to be useful for more than mapping and navigation, as is the case in Saroya et al. (2020), where the authors exploit the inherently topological nature of subterranean scenarios to predict how the graph structure is going to evolve, and therefore optimize the exploration task based on these predictions.

Another side of this proposal that has been previously explored is the use of Convolutional Neural Networks or other deep learning approaches as a perception system. In the case of (Mansouri et al. 2020, 2018), the authors explore the use of a CNN to process images from a front-facing camera mounted onboard a drone. The CNN then predicts from the images the relative yaw of the robot with respect to the axis of the tunnel, allowing the drone to stay aligned with the tunnel, and navigate it safely without the need for a precise pose estimation. A similar approach was followed by a previous work of ours, (Cano et al. 2023), in which the CNN processed LiDAR-based depth images to predict the necessary bearing of the robot so that it could safely and rapidly traverse tunnels while staying centered and avoiding obstacles.

An important part of our perception system is detecting certain features in the environment (corridors, intersections and dead-ends) that are assumed to be there. This is in the vein of, but more general than (Romeo and Montano 2006), where the authors use Principal Component Analysis to classify 2D laser readings as a priori features (T-intersections, dead-ends, corridors, left/right turns etc.) to aid in the self-localization task.

## 4 | Approach

As mentioned earlier, our proposed method interprets the environment as a graph, where tunnels serve as edges connecting intersections and dead ends, which act as nodes. This approach is substantially different from geometrical approaches, so it has been necessary to rethink the typical mobile robotics stack (perception, navigation and mapping layers).

In this section, we describe in detail how the topological map is implemented and how it is used to generate navigation plans based on topological instructions (Section 4.1). We also describe how the perception layer (Section 4.2) has been implemented with the use of a Convolutional Neural Network (CNN), which processes the point cloud data from a 3D LiDAR and provides an estimation of the distribution of the galleries around the robot. Thanks to this perception stack, it is possible to use a navigation approach based on the sequential execution of high-level topological instructions of the type *"take the left exit on the third intersection"* (Section 4.3).

## 4.1 | Topological Map and Path Planning

The field of topological mapping in robotics has yielded many different approaches to the problem. As mentioned earlier, as a general rule, in these maps the nodes contain both a geometric pose, and some information that allows the robot to self-localize (usually images or point clouds), so they are typically referred to as *topo-metric maps*.

### 4.1.1 | *Data Structure*

Instead, we aim to obtain a pure topological representation of the environment, which means that the only information that a node contains is to which other nodes it is connected by an edge of the graph and their relative angular order.

Figure 4 shows a topological map that contains seven nodes and its corresponding data structure. As mentioned, it consists, for each node, on a circular list of its neighbors ordered in a *counterclockwise* manner. By having the neighbors of each node ordered it is possible to look at their respective positions (index on the neighbor list) and know which edge should be chosen to go from one to another. This structure allows us to create topological plans without the need to store any explicit geometrical information, only the angular ordering of neighbor nodes.

### 4.1.2 | *Topological Path Planning*

Traditional path-planning methods usually produce a (dynamic) plan consisting on a sequence of geometrical positions that gradually take the robot from its initial position to a target position. In our case, this approach is not feasible, as we do not store geometric information in the topological map.

In our approach, instead of an origin pose and a target pose, we have an origin node or edge and a target node, and the topological plan consists of a sequence of high-level topological instructions of the type "take the first exit clockwise in the next intersection." One characteristic of these instructions is that

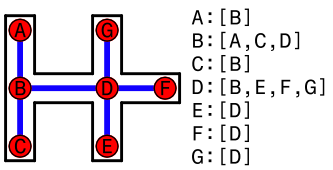


**FIGURE 4** | Data Structure of our Topological map. On the left, the graphical representation where the Black lines represent the walls of the environment, the red dots are the nodes and the blue lines are the edges. On the right there is the actual data structure used in our implementation. [Color figure can be viewed at wileyonlinelibrary.com]

they are only relevant when entering an intersection, which allows us to make an important assumption: As our robot can only advance forward, every time it enters a node, it will have the guarantee that there is at least an exit at its back (from where it entered) and as such, this is taken as the reference point for the topological instructions, and will be henceforth referred to as the *rear exit*.

In a more specific fashion, these instructions take the form of a simple integer number that can be either positive, negative or 0. The value of 0 is the one corresponding to the rear exit, a positive value $N$, refers to the $N^{th}$ counterclockwise exit and a negative value $-N$ refers to the $N^{th}$ clockwise exit.

### 4.1.3 | *Obtaining a Plan*

The process of obtaining this plan is divided into two steps. The first step consists in obtaining the sequence of nodes that need to be traversed to go from the initial node to the objective node. To do this, Dijkstra's algorithm is used (Dijkstra 1959). It is important to note that, as we aim for a purely topological method, no metric information is contained in the map, and as such all edges are set to have the same weight. This means that Dijkstra's algorithm will always provide the sequence with the least number of nodes possible, even if it is not the shortest in terms of real-world metric distance or time of travel. However, this information could easily be added to the map (e.g., for visualization purposes, or to estimate faster paths) without loss of generality for the rest of the algorithm.

The second step is to take the sequence of nodes and transform it into a set of high-level topological instructions, a process that is better illustrated with an example.

Consider again Figure 4. In this example, we want to obtain the *topological path* that corresponds to the set of exits the robot must take to go from node $C$ to node $G$ which will be represented by a set of integer numbers. By applying Dijkstra's algorithm to the map, the sequence of nodes that is returned is $[C, B, D, G]$. Thanks to the enforced ordering of the node connections, this sequence can be mapped to topological instructions using the data on the right of the figure that represent the nodes reachable from a given node ordered in a counterclockwise manner as explained before.

This path must traverse two intersections (nodes $B$ and $D$). In the case of $B$, it is known that the robot is coming from $C$ (the previous node) and heading to $D$ (the next node). By looking at their position in the neighbor array of $B$, we obtain that $C$ has an index of 1 and $D$ has an index of 2. From this information, we can obtain the corresponding instruction by subtracting the index of the previous node ($C$) to the index of the next node ($D$), obtaining $2 - 1 = +1$. If we repeat this same reasoning with the intersection node $D$, we will see that the previous node ($B$) has an index of 1, and the next node ($G$) has an index of 4, so the instruction would be $4 - 1 = +3$. This would result in the path $[+1, +3]$, which would translate to: at the next intersection take the first exit counter-clockwise and at the intersection after that, take the third counterclockwise exit.

As an extra note, it is clear that two different numbers can result in the same behavior, as there is an equivalent counterclockwise value for each clockwise one, and vice versa. For example, the previous plan $[+1, +3]$ is equivalent to $[-2, -1]$.

## 4.2 | Perception

The previous section details how the topological plan is generated from the topological map. However, to be able to follow it, a robot needs to have a topological understanding of its surroundings. To be more specific, it is necessary to know whether the robot is in an intersection or in a tunnel, and how the galleries are distributed around the robot.

To accomplish this task we make use of the concept of *exits*. In our method, an exit is defined as the direction in which a tunnel advances. For example, when the robot is at a dead-end, only one exit is present (Figure 5a). When the robot is in the middle of a tunnel, two exits are present (in opposite directions, Figure 5b). Finally, when the robot is at an intersection, there are as many exits as tunnels that meet at said intersection (Figure 5c).

By detecting these exits we address the two requirements: by counting the number of exits present around the robot, we know if it is in an intersection (more than 2 exits), tunnel (two exits) or the dead-end (one exit), and thus, whether it is in a node or an edge.

### 4.2.1 | *Exit Detection*

To solve this exit-detection problem we have made use of a 2D CNN. The use of a Deep Learning solution, instead of one based in a geometrical analysis (eg. Voronoi Diagrams), was motivated by the difficulty to account for all possible variations in local geometry that do not affect the topological structure, such as, for example, the different shapes (square, circular, horseshoe, arch, etc.), varying radii and cross-section, different roughness (from smooth concrete walls to rough bare-rock walls), or the presence of small obstacles, people or vehicles. The main reason to use a 2D CNN architecture over others (like 3D CNN (Ji et al. 2012) or PointNets (Qi et al. 2017)), was the goal of reducing the computational cost to a minimum.

The detection pipeline is divided into four processes: First, we transform a 3D LiDAR point cloud into a depth image. Then, this image is fed into the CNN, that outputs a vector of 360 numbers with values ranging from 0.0 to 1.0. In this vector, the galleries appear as peaks, so the next step is to process it to extract the angle at which the galleries are being detected, and the strength of the detection. The final step is to track how these exits behave over time as the robot moves and rotates, and the exits appear and disappear when the robot transitions from edges to nodes (or vice versa).

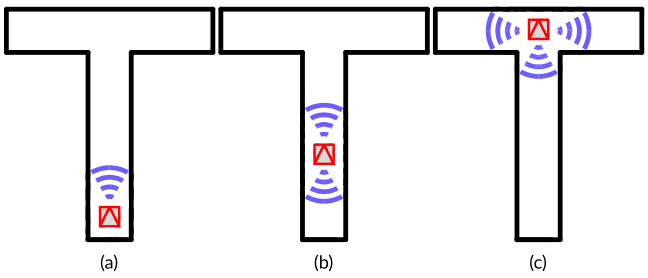


**FIGURE 5** | What is an *exit*. The black lines are the walls of the environment, the red square is the robot and the blue lines indicate the exits present around the robot in each situation. a) Exits detected in a dead-end. b) Exits detected in a gallery. c) Exits detected in an intersection. [Color figure can be viewed at wileyonlinelibrary.com]

As previously mentioned, the first step is the transformation of the point cloud obtained from the 3D LiDAR (Figure 6b) into a depth image (Figure 6c). This begins by discarding all points that are further away than 50 m, maintaining only the points with the most useful information. Secondly, a blank image (filled with zeros) 16 px high (the number of beams of the LiDAR) and 720 px wide is created. Then, for each point in the cloud, we obtain the corresponding position in the image (according to its spherical coordinates), and assign the distance of that point as the value of the pixel. Finally, the image is divided by the max range of the LiDAR (50 m in our case, because of the cutoff) to normalize it in [0, 1].

This image is then fed into the CNN, whose architecture is displayed in Figure 7. This model is inspired on the VGG-16 architecture (Simonyan and Zisserman 2014), having multiple Convolutions + ReLU between the Pooling layers, but has some differences.

Specifically, our last-layer activation is a ReLU layer, instead of the more commonly used Sigmoid, and the kernel depth is kept relatively low, reaching only 32 channels, while VGG-16 reaches 512. The use of ReLU activation instead of Sigmoid for the last layer was decided based on empirical results, as it resulted in a significant reduction of the loss during training. The number of channels was also reduced so that the resulting network was lightweight enough to run at high frequencies in compute-constrained platforms. More details on how this network has been trained can be found in Section 4.2.2.

The output of the network (Figure 6d) is an array of 360 numbers in the range [0, 1], where the position of each element corresponds to an angle around the robot in the $z$ axis. As can be seen in the figure, this vector presents peaks; each of them corresponds to one exit.

The last step of the detection is the extraction of the angular position of the exits from the output of the CNN (Figure 6e). For this, we use two simple heuristics: 1) First, we extract all the local maxima in the vector and 2) we only keep the values that are over a certain threshold. This way, we ensure that only high-confidence detections are used in the following processes.

### 4.2.2 | *Training Dataset*

Given the specific nature of our approach, no previous datasets were available to train the CNN, so it was necessary to create a custom one.

Due to the close sim-to-real gap that LiDAR sensors have, it was decided to create a fully synthetic dataset using procedural environments that where generated using a tool that we presented in Cano et al. (2024a). This tool allows tuning many of the generation parameters, like the number of tunnels and intersections, diameters and curvatures of tunnels and roughness of the tunnels' walls. This way, by using different combinations of these parameters, a varied set of environments where obtained, such that they contain the overall variability that real-world underground scenarios present (Figure 8).

For each individual training sample, the process is the following: 1) Since the axis of the tunnels in the environment are known (they are provided by the procedural generation tool, Figure 9a), the robot is placed in a random position on any of the axis (Figure 9b,c), 2) A random pose transformation is

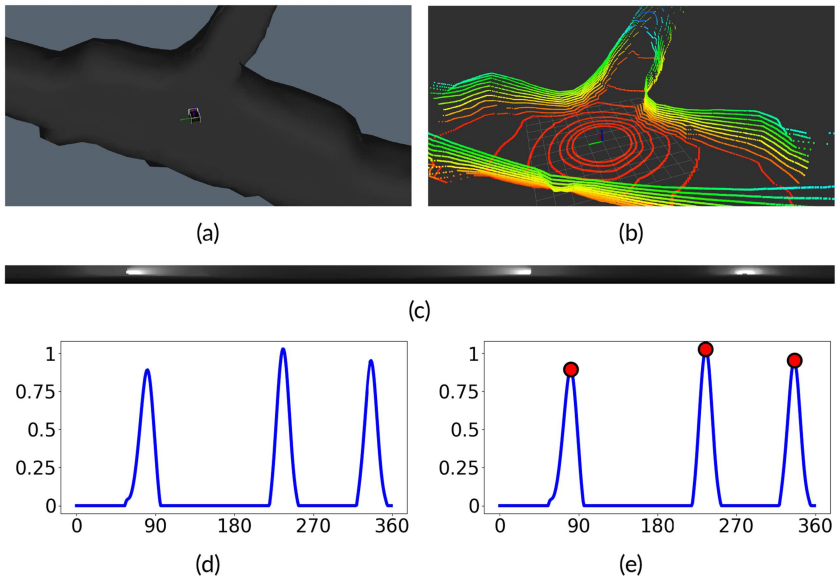


**FIGURE 6** | Perception pipeline. a) Robot and Environment. b) Point cloud from the VLP-16. c) Rasterization of the point cloud into a depth-image. d) Raw output of the CNN. e) Detected galleries after analyzing the CNN's output. [Color figure can be viewed at wileyonlinelibrary.com]

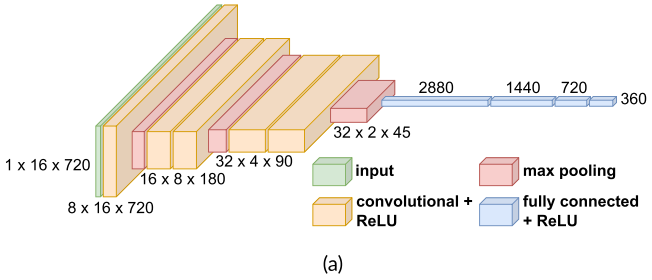


**FIGURE 7** | Architecture of our Convolutional Neural Network. [Color figure can be viewed at wileyonlinelibrary.com]

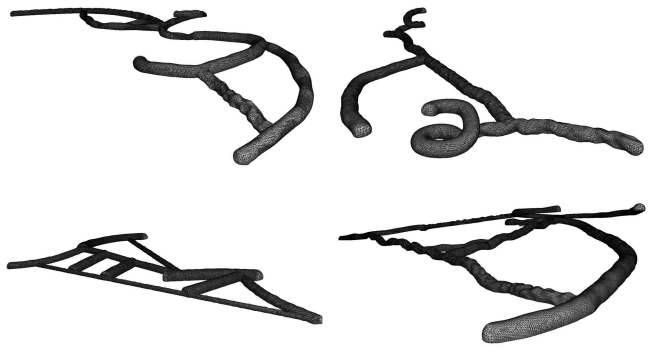

**FIGURE 8** | Some of the environments used to train our CNN.

applied to the robot, adding rotations and displacements that ensure the robot stays within the tunnel (Figure 9d), 3) A LiDAR scan is captured for this position (Figure 9e) and the corresponding depth image is obtained (Figure 9g), and finally, 4) The axis points that are at a specific distance from the robot (5 m has shown to be a value that generalizes well to smaller and larger tunnels) are selected (Figure 9f) These axis points are the ones used to create the label of this training sample. By calculating their relative position to the robot, we can obtain their respective heading w.r.t. the robot. These headings are the ideal output of our complete perception pipeline, but they need to be transformed into a format that can be produced by a CNN, meaning the 360-element array that will be the label for training. To do this, we start from a zeros-only vector and place gaussian-shaped "peaks" centered in each of the angles that correspond to the headings (Figure 9h).

For the training dataset, 50 different environments were generated, and 10,000 data points were captured for each of them, for a total of 500,000 data points.

The network is trained with this dataset, using the Adam optimizer (Kingma 2014), with a starting Learning Rate of $4 \times 10^{-5}$, a Learning Rate decay of 0.999 between epochs and the Mean Square Error cost function. It was trained for 512 epochs, with a batch size of 64.

#### 4.2.3 | *Exit Tracking and Stability*

At this point, the perception pipeline provides only a series of values whose peaks indicate the presence of an exit. However, this information is not refined enough to navigate so it needs further processing.

One of the problems of using the raw angles is that, as the robot moves and rotates, the same exit will be detected at different angles, so it is necessary to track this change over time for all of the detected exits. Additionally, as the robot enters and leaves the nodes, some exits will appear while others will disappear.

To take this into account, we discriminate between *detected* and *tracked* exits. Detected exits are the output of the neural networks, and consist of an angle and a magnitude, while tracked exits are the result of the continuous processing of the detected exits, and consist of a unique ID, an angle and a *confidence score*.

For each prediction of the CNN, a new set of detected exits must be processed with the objective of: 1) updating the angles of the tracked exits, 2) checking if one of the tracked exits is no longer being detected and 3) checking if there is a new exit that needs to start being tracked.

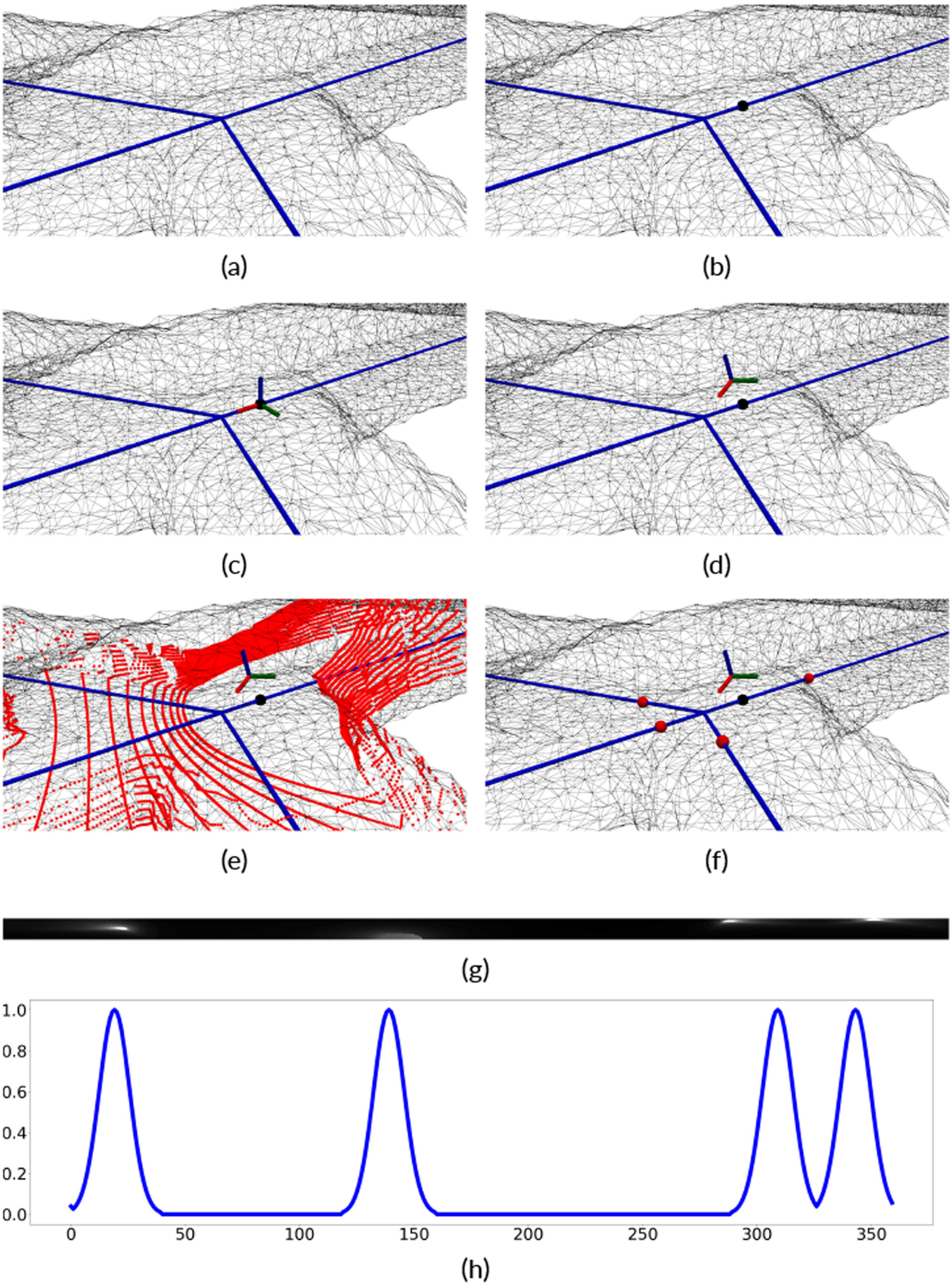


**FIGURE 9** | Collection of a synthetic training sample. a) Procedural environment (black) and tunnel axis (blue). b) Selected axis point (black sphere). c) Robot aligned with tunnel (red, green and blue are the robot's x,y and z axis respectively). d) Transformation applied to the robot. e) Captured LiDAR point cloud (red dots). f) Axis points used to generate the label (red spheres). g) Training Image. h) Training Label. [Color figure can be viewed at wileyonlinelibrary.com]

To update the angles of the tracked exits, we first start by calculating the angular distances between each of the detected exits to each of the tracked exits, obtaining a triangular matrix where each row corresponds to a detected exit, each column to a tracked exit and each cell contains the angular distance between the corresponding tracked and angular distances. To consider that a specific detected exit is the same as a specific tracked exit, the angular distance between them must be the minimum in both its row and column. This ensures that a detected exit can only be associated to one tracked exit, and vice-versa. Additionally, this distance must be smaller than a certain threshold, that is established depending on the robot's maximum angular speed and the frequency of the predictions. Once a detected exit is associated with a tracked exit, the angle of the tracked exit is updated to that of the latest detection.

After this step there are two possible scenarios. The first one is that each of the detected exits has been associated to one of the tracked exits. If this is the case, the tracking is finished. In the second scenario, two possibilities exist: either some detected exits have not been associated with a tracked exit and/or some of the tracked exits have not been associated with a detected exit. These two possibilities translate to either a tracked exit that is no longer being detected, or a new exit that is not being tracked yet. To deal with the possibility of spurious detections, the creation and deletion of tracked exits is handled with a simple hysteresis method in which the important criterion is the number of consecutive observations or, equivalently, the time elapsed since an exit is detected for the first/last time.

Each tracked gallery has a confidence value ($\gamma$) that can be in the interval $\gamma \in [0, \gamma_{max}]$. When a detected exit is not associated with a tracked exit, a new tracked exit is created, with: 1) a unique `ID`, 2) the same angle as the detected exit and 3) a confidence score of $\gamma = \delta_d$, $\delta_d$ being the magnitude of the exit (the height of the peak in the CNN's output). After being created, every time a detected gallery is associated with it, its confidence is increased by the height of the detected gallery $\gamma = \min(\gamma_{max}, \gamma + \delta_d)$. However, if it is not, the confidence is decreased: $\gamma = \max(0, \gamma - \delta_s)$, $\delta_s$ being a parameter that defines the rate at which the confidence decreases. If the the confidence reaches a certain threshold ($\gamma > \gamma_t$, it is considered reliable, however if it reaches 0 ($\gamma = 0$), it is deleted. To ensure a behavior equivalent to hysteresis, the confidence should grow slower than it decreases, meaning $\delta_s > \delta_d$, and because $\mathbb{E}(\delta_d) \sim 0.8$, we have chosen the value $\delta_s = 1$. Regarding $\gamma_{max}$ and $\gamma_t$, their values have been set to 5 and 3 respectively, as they have shown to provide reliable results in our configuration.

#### 4.2.4 | *Dealing with Instability*

Even with the introduction of the tracking system, there are periods of substantial instability (especially in realistic environments), when the output of the neural network presents significant inconsistencies between consecutive predictions. This occurs during the transitions between galleries and intersections, and vice versa. This can cause the quick creation and deletion of tracked galleries, making the decision process less reliable. In our previous work, this issue was addressed by starting an "instability timer" when a new tracked exit was created or removed which was used to wait for the output of the network to stabilize. However, this timer was a parameter that had to be adjusted from one environment to another, and was prone to failures if intersections or galleries of different dimensions where present in the same environment. For this reason in this work we introduce the concept of prediction stability.

To determine whether a CNN output qualifies as stable, we initially examined the attributes of predictions during both stable and unstable periods. One clear characteristic of stable periods is that subsequent predictions look very similar, meaning that they present the same number of peaks in close-by positions. From this observation, we can define two conditions for a prediction to be stable: 1) Every tracked exit has a high-enough confidence level, 2) every detected exit is associated to a tracked exit. Also, when the robot enters an intersection, it tends to detect the new exits closer to already tracked exits, making their respective "bulbs" overlap, creating a valley between them. From this observation, we derive a third condition: 3) there are no local minima in the output. Finally, transitional phases often manifest minor peaks in the output that fall below the threshold for detection, leading to our final criterion: 4) there are as many local maxima in the output of the CNN as tracked exits.

If all these conditions are met, it means that there are well defined peaks in the output of the network, and that all present exits are tracked and reliable. Figure 10 shows how the exit tracking and prediction stability behave in the context of entering and exiting an intersection. In Figure 10a, the robot is beginning to detect the lateral gallery, so a small peak appears in the left side. This is a local maxima, but too small to be considered a detection, so it is considered unstable. As the robot fully enters the intersection (Figure 10b), the small peak grows and becomes a detection, and after being detected enough times (its confidence reaches 3, as explained before), it also becomes a reliably-tracked gallery, so the situation is considered stable. Then, the robot begins to exit the intersection through the lateral gallery. As it does so, the exits 1 and 2 start to decrease in size, and a new exit with $ID = 4$ is tracked. However, the part of the network output corresponding to the back of the robot presents local maxima and local minima that make the prediction unstable (see in Figure 10c the network output in polar coordinates).

From this point onward, every time we refer to an exit, we will be referring exclusively to the tracked exits, which are the actual output of the perception pipeline.

### 4.3 | Navigation

This section will showcase how the information that the robot has access to about its environment (odometry and tracked exits) is used to execute a sequential set of high-level navigation tasks. Each task has some requirements on the initial conditions, and will provide information about the successful completion or failure. We differentiate between two types of tasks: *purely topological* and *hybrid* tasks, due to the fact that limiting the navigation to purely topological commands would preclude some intuitive interactions with the robot.

The purely topological tasks are executed without using any extra information outside the perception system outlined in Section 4.2. They are the following:

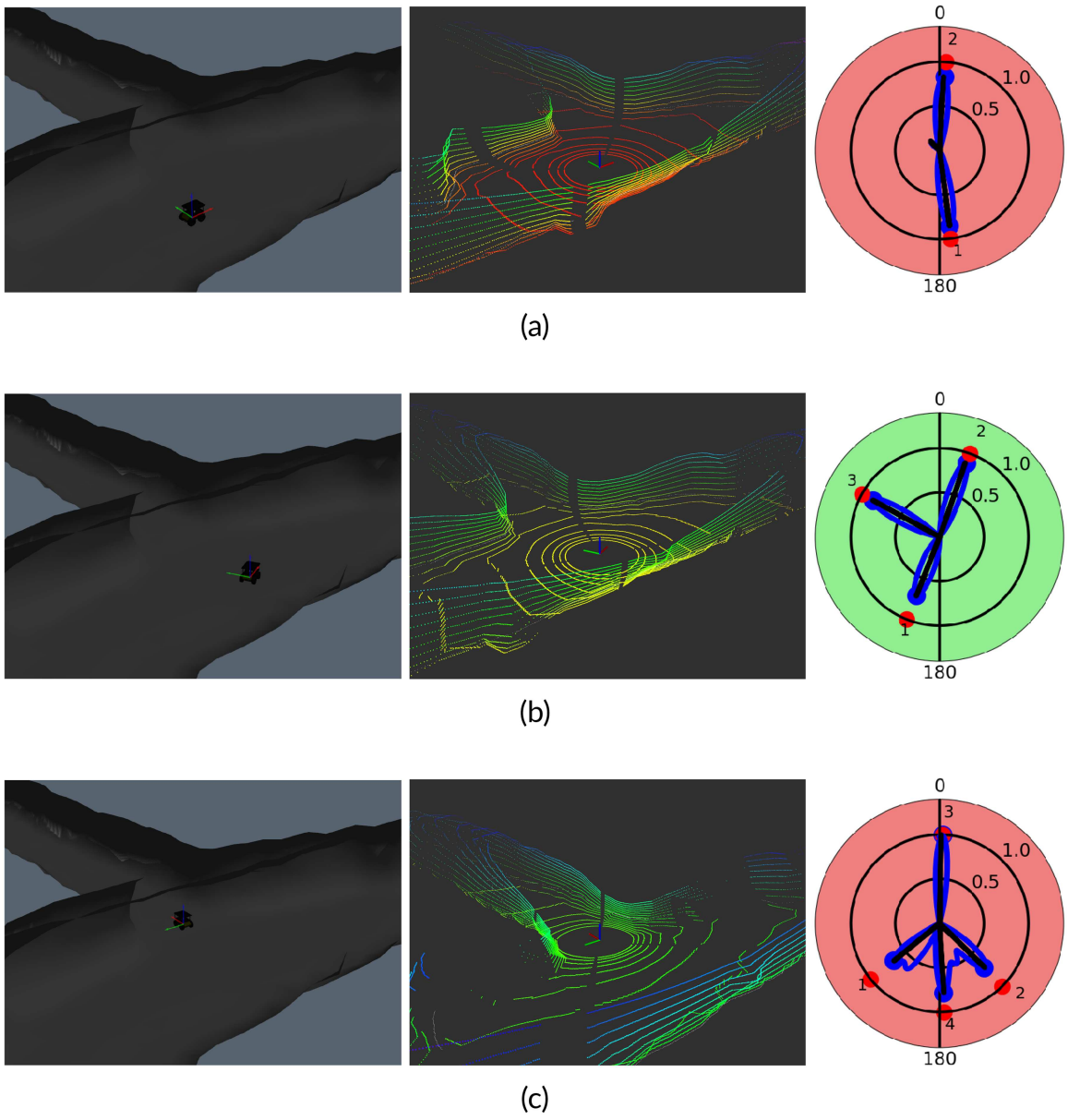


**FIGURE 10** | Instability stages when entering and exiting an intersection. Images on the left are the situation of the robot, images on the middle show the point cloud of the 3D LiDAR. The images on the right show: the network prediction in blue, the raw local maxima as a black line, the detected exits as a blue dot, the tracked exits as a red dot with a number (the ID) and the stability as the background color (red is instable and green is stable). The presented situations are: a) The robot enters the intersection, and the prediction is unstable. b) The robot fully enters the intersection, causing the prediction to become stable. c) The robot starts leaving the intersection, and the prediction becomes unstable again. [Color figure can be viewed at wileyonlinelibrary.com]

- `take <N>`: This instruction works as described in Section 4.1, where `N` is the $N^{th}$ exit to take. If $N > 0$, it means to take the $N^{th}$ counterclockwise exit, if $N < 0$ it means to take the $N^{th}$ exit clockwise, and if $N = 0$, it means to take the rear exit. If the instruction is received while in an intersection, it will take the specified exit and indicate a success. On the other hand, if this instruction is received while the robot is in a gallery or a dead-end, it will advance until an intersection is found and, once there, if the node is an intersection it will take the specified exit.
- `advance_to_node`: If in a gallery or a dead-end, the robot will advance through the tunnel following the exit closest to its front until either an intersection or a dead-end is encountered.
- `turn_around`: The robot must be in a gallery (otherwise it will report a failure). Then, it will turn until it faces the exit originally closest to its back and report a success.

The hybrid tasks cannot be considered purely topological, as they make use of additional information like wheel odometry, time passed or angular values instead of angular ordering. However, they allow extra functionalities and more options for the user to interact with the robot:

- `take <leftrightfrontback>`: This instruction behaves similarly to the `"take [N]"` one, but including the angles of the exits into the decision process. Each of the four directions is associated with an arch of 90°, centered in said direction. For example, if the robot is in an intersection

and receives the instruction `take [right]` it will check if there is an exit within the 90° arch centered in 270°. If that is the case, it will take said exit; however, if no exit is detected to the right of the robot, it will report a failure.

- `advance_met <M>`: The robot must be in a gallery or dead-end (if it is in an intersection it will report a failure), and will advance `M` meters (according to the odometry), unless it reaches a node, in which case it will stop and report a success.
- `advance_sec <S>`: This instruction behaves exactly like the previous one, except in this case the robot will advance `S` seconds.

The reporting of success or failure is used to address the mismatch between user expectations and reality. For example, in a hypothetical rescue operation inside a mine, the operator may have indicated a path that is no longer available because a gallery is now blocked. In this situation, the robot could have received a `take <N>` instruction, but instead of reaching the intersection, it finds the blocked gallery. If this was the case, it is important for the robot not to continue with the following instructions, as they would lose their intended meaning.

#### 4.3.1 | *Obstacle Avoidance*

During the execution of the previously detailed instructions, the robot is always advancing towards an exit and, as explained in Section 4.2.2, this means that the robot stays centered in the tunnel. This is true even if there is an obstacle in the middle of the tunnel, so it is necessary to add an avoidance system that prevents any collision with such obstacles.

Given that the navigation target is an angle w.r.t the robot, there is the need to use an obstacle avoidance system that can take this information, combine it with the information from the LiDAR, and output a new direction that, while avoiding the obstacle, keeps the robot advancing towards the objective.

We developed a specific solution (presented and validated in Cano et al. (2022)), which involves assigning a value to each of the 360 angles around the robot and choosing the angle of highest value. This means that the value of each angle should be the highest if it points in the desired direction and is free of obstacles, and lower if it points in a different direction or towards an obstacle.

Each of these values are obtained by multiplying two weights, one derived from the desired angle of advancement (advancement weight), and the other one derived from the horizontal LiDAR scan (obstacle weight).

To derive the directional weight of any angle ($\delta$) for given a specific angle of advancement ($\beta$) the following formula is used:

$$W_\delta = 1 - |[(\beta - \delta + 180) \bmod 360] - 180|/180 \quad (1)$$

This results in a linear decrease of $W_\delta$ from 1 at the desired angle of advancement to 0 in the opposite direction, as can be seen in Figure 11d.

The process of obtaining the obstacle weights can be divided into two steps. In the first step, we discretize the angles around the robot into 360 sections, assigning the corresponding closest obstacle distance as the initial value of each angular sector. The second step is to cap all the values larger than a certain threshold. In our case this is 5 m, as the maximum speed of our robot is 1 m $s^{-1}$, so that gives a time horizon of 5 s. Finally, the remaining values are inflated so that every obstacle appears wider than it is, depending on the desired safety distance, obtaining thus the obstacle weight of each angle (Figure 11e).

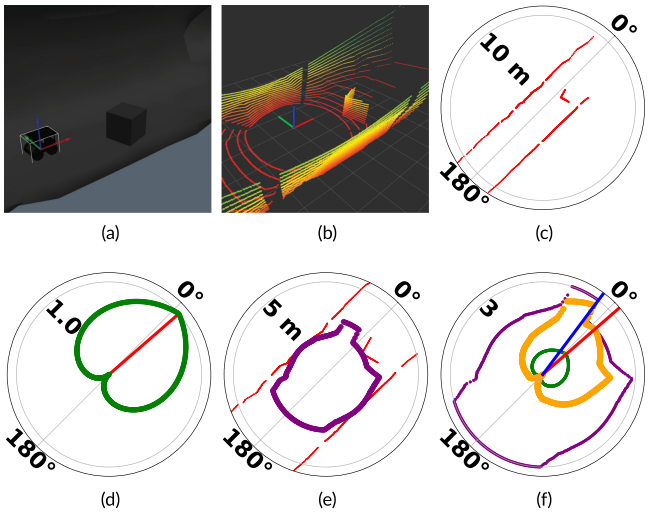


**FIGURE 11** | Proposed method for obstacle avoidance. a) Situation of the robot. b) Point Cloud captured by the LiDAR. c) Laser Scan extracted from the point cloud. d) Desired direction (red line) and directional weights (green dots). e) Scan in red and obstacle weights in purple. f) Combination of directional weights (green) and obstacle weights (purple) into the final weights (orange). The corrected angle (blue line) is different from the initial angle (red line), in a way that the robot avoids the obstacle. [Color figure can be viewed at wileyonlinelibrary.com]

Then the two sets of weights are multiplied, obtaining the final values. These are then used to select the angle that contains the largest value. This angle is referred to as the *corrected angle* ($\delta'$), and it is the direction the robot should advance towards so that it avoids any obstacles (Figure 11f). The linear and angular velocities required to advance in this direction are calculated as follows:

$$\omega = \max(|\delta'| \times A, \omega_{max}) \times \delta'/|\delta'| \quad (2)$$

$$v = \max(v_{max}*(\omega_{max} - |\omega|)/\omega_{max}, 0) \quad (3)$$

This results in the following behavior: A proportional controller for the angular speed with a cutoff maximum angular speed and a linear speed inversely proportional to the angular speed that drops to 0 if the angular speed is high. With this approach, if the corrected angle is large enough, the linear velocity is 0, allowing the robot to rotate in place, while if the corrected angle is 0, meaning the robot is perfectly aligned with it, the linear velocity is the maximum possible.

This behavior ensures that when the robot enters an intersection and needs to take an exit at a considerable angular distance, the robot will rotate in place, maintaining stable predictions from the network, and ensuring that the robot will stay in the intersection until it is aligned with the desired exit.

### 4.4 | Mapping and Exploration

The final component of our proposal is the mapping framework. The goal is to enable the robot to build a topological map of the environment as it navigates through it.

At its core, every mapping approach revolves around a simple principle: Relate the robot's observations to specific elements in the map, use this relationship to determine the robot's current location within the map, and update the map with new relevant information as the robot explores the environment.

In our approach, the perception system provides the angular distribution of detected exits around the robot, while the map is represented as a graph where edges correspond to galleries, and nodes represent either intersections or dead-ends. This means that the relationship between the observations and the topological map is that each detected exit corresponds to a unique edge in the graph. Consequently, the most critical task of our mapping approach is to maintain this correspondence, ensuring that each detected exit is accurately associated with the correct node in the map.

During mapping, we consider two types of nodes. The first one is the *visited* nodes, whose number of connections is known. The second type of node are the *unvisited* nodes, which are nodes that are known to exist, but have not yet been visited, and thus, whose number of exits is unknown.

The mapping process starts with an initialization step that varies depending on whether the mapping starts in a node or an edge. If the robot starts in an edge (meaning that only two exits are being detected, namely back and front exits), two unvisited nodes are added to the map and connected by an edge. Otherwise, if the robot starts in a node (either dead-end or intersection), a visited node and as many unvisited nodes as exits are added to the map, with edges that connect each unvisited node to the visited node. This initialization step is shown in Figure 12, where three possible initial situations are illustrated.

After this initialization step, the initial map contains unvisited and/or visited nodes, and the relationship between the observed exits and nodes has been established. With this information, the mapping system can start. It has been implemented as a simple state machine formalized in a binary Petri net shown in Figure 13. The main motivation behind the implementation of the mapping system as a state machine was the need to account for the instabilities in the network prediction when moving between nodes and edges (as explained in Section 4.2.3). Both the fluctuations that the prediction has at node-gallery boundaries, and the risk of spurious detections made it necessary to have an unstable state where these detections can be ignored until a fully stable state is reached.

The analysis of this Petri net shows that it is safe (1-bounded), and conservative with an invariant total of one token. Structurally, it is a state machine (each transition has exactly one input and one output place). Its reachability graph is strongly connected, so markings can be revisited. The net is live (no transition can become permanently disabled) and, therefore, it is deadlock-free from a structural point of view.

Since transitions have associated guards, the overall system correctness also depends on these guard conditions. The transition inputs are mutually exclusive CNN outputs: stable ($s$) and unstable ($\neg s$) detection, and the number of detected galleries $n = 2$ and $n \neq 2$. Since navigation will eventually produce a change from stable to unstable CNN output, or vice versa, each state will in due time trigger its output transition. (During initialization, the robot can be manually driven to a stable place if not initially at one.)

The inner workings of each state are as follows:

1. *Stable Gallery State (**Gallery** in* Figure 13): The robot is in a gallery, far enough from any node that the output of the CNN is stable. In this state, there are only two detected exits and the main task of the mapping system is to track to which node the robot is heading to. Given that the robot can only move forward, this node (the *target* node) is the one associated with the exit closest to the front. When there is a change in the number of exits, the mapping system transitions to the **Unstable** state.
2. *Stable Node State (**Node**):* When the robot fully enters an intersection or reaches a dead-end, the prediction becomes stable again, with a number of exits $n \neq 2$. If this is a previously unvisited node, we mark it as visited, and add new edges to unvisited nodes for each exit other than the one we are arriving from. The relative ordering of exits is preserved, as this is the only information required for navigation (Section 4.1).
3. *Node-Gallery Boundary Unstable State (**Unstable**):* When entering or leaving a node, there is a period of instability of CNN output in which some exits can briefly appear or

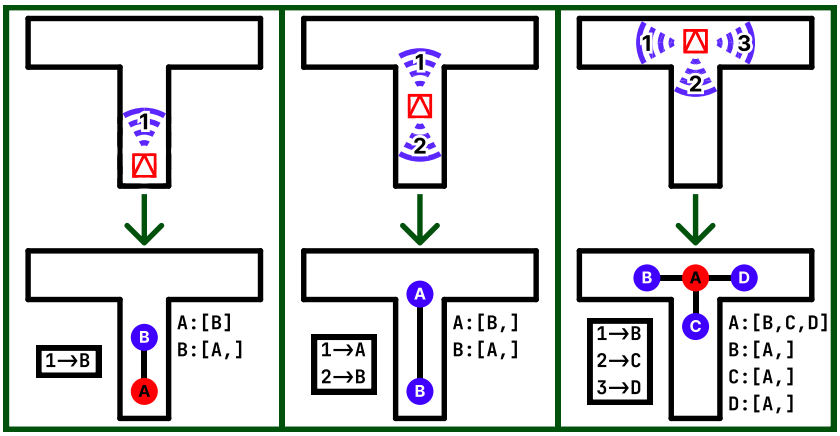


**FIGURE 12** | Three possible initializations depending on the starting situation. From left to right, starting in a dead-end, starting in the middle of a gallery or starting in an intersection. The upper figures illustrate the situation of the robot as well as the exits around it. The bottom figures illustrate the initial maps, with red nodes being visited and blue ones being unvisited. To the left of each bottom figure there is the correspondence between exits and nodes, and to the right, the data that the map contains, as in Figure 4. [Color figure can be viewed at wileyonlinelibrary.com]

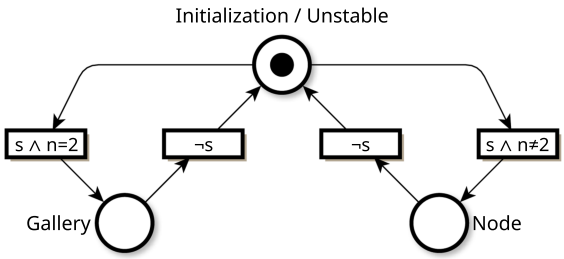


**FIGURE 13** | Petri net describing the state machine used in the mapping approach, with state-changing inputs inside transitions. Places have associated actions described in the text. Input combinations not shown imply that the Petri net stays in the same state. $n$ (a positive number) is the number of detected exits and $s$ (a boolean) indicates whether the detection is stable, as explained in Section 4.2.4.

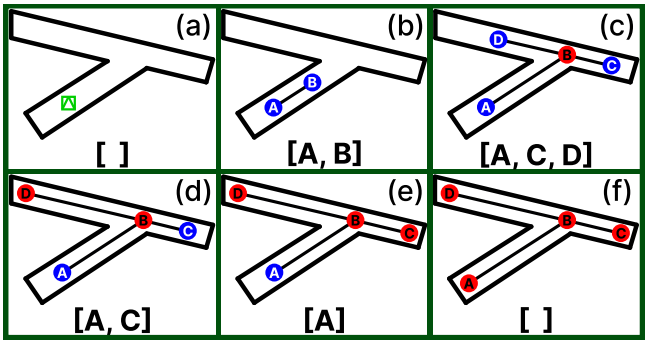


**FIGURE 14** | Topological Exploration. a) Starting pose of the robot (green). b–f) Exploration process. Blue nodes are unvisited while red nodes are visited. The brackets contain the stack of unvisited nodes. [Color figure can be viewed at wileyonlinelibrary.com]

disappear. We filter out these periods to avoid spurious updates on the map. When entering/exiting a node, we always know the gallery we are coming from/leaving through, located at the back/front of the robot, respectively. During this state, the only task of the mapping system is to keep track of this exit. This state lasts as long as the exit prediction is unstable. Once our filtering deems the CNN output as stable, we know whether we are entering a gallery or a node by the number of exits being reported. If the latter, we know the identity of the connecting exit in the former and current node. This information is used to update newly visited nodes as explained in the **Node** state.

4. *Start-up (**Initialization**):* On startup, as explained, the map initialization depends on being located at a gallery or node. The Unstable state already discriminates for this initial condition, so we can simply reuse it.

As can be seen, the only information that our proposed mapping approach requires is the tracked exits and the level of stability of the network's prediction. This makes it possible to generate the topological map even if the robot is navigating with a non-topological system, or is being operated by a user. Note that, given its purely topological nature, this mapping proposal is not yet capable of loop closure. For loop closure to work, it would be necessary to store some extra information about the nodes, like their position or some distinguishable feature. Future work will explore the possibility of performing loop closures without having to rely on accurate pose estimation. As a final comment on this, it is important to note that, even if the mapping system cannot create a map with loops, the path-planning and navigation systems can use them without any adjustments.

### 4.5 | Topological Exploration

The final module in our proposal is an autonomous exploration method built on top of our mapping system. As mentioned in the previous section, our mapping approach differentiates between nodes that have been visited and nodes that have not. This explicit difference makes it trivial to state the goal for the exploration system: Navigate to every unvisited node, until there are only visited nodes in the map.

To accomplish this task, the exploration system acts as a "director," that indicates to the navigation system where to go. More specifically, it keeps track of all the unvisited nodes using a LIFO stack data structure. This way, every time the robot arrives at an unvisited node and sets its state to visited, all the new unvisited nodes that are added to the map are also pushed to the stack. Then, the exploration system pops an unvisited node from the stack, and instructs the navigation system to head there. The complete exploration process is shown in Figure 4, and can be summarized as follows:

1. The exploration begins with an empty map (Figure 14a) that has to be initialized as defined in Section 4.4. As a result, the mapping system generates a partial map that contains visited and/or unvisited nodes. The later ones are pushed to the stack (Figure 14b).
2. The exploration system pops an unvisited node from the stack and uses the already generated map to obtain a path to it.
3. The navigation system executes the generated path, eventually reaching an unvisited node.
4. Once the robot reaches the unvisited node, it becomes visited. If new unvisited nodes are detected, they are added to the map and pushed into the stack (Figure 14c).
5. If the stack is not empty, the system goes back to step 2 (Figure 14d,e), otherwise the exploration is complete and the process stops (Figure 14f).

This approach is equivalent to a depth-first exploration of a graph. This means that the robot will explore as deep as possible into each branch of the tunnel network, before backtracking to explore a new branch, which is, as well known, the most efficient way possible to completely traverse our target environments (in absence of loops). A simpler way of accomplishing this would have been a strategy of "always go left," as done in Pereira et al. (2021); de S Thiago Filho et al. (2025), although this would sacrifice the ability to prioritize unexplored branches according to some criterion.

## 5 | Experimental Evaluation

Our proposal has been evaluated in both simulated and real environments. While simulated environments allow for a greater variety of scenarios, they lack some of the challenges

associated with real underground environments, like truly degraded odometry, dust particles or erroneous readings of the LiDAR. In the simulated experiments a variety of environments have been used, including some generated from the DARPA SubT challenge scenario assets and a 3D reconstruction of a real environment.

Regarding the implementation, all the components in the navigation stack were implemented using the Robot Operating System (ROS) framework, with nodes written in both Python and C++. In the real-world experiments, all computations where performed on the onboard computer of the robotic platform. More specifically, we used the Clearpath's Husky A200 UGV,[3] equipped with a Velodyne VLP-16 3D LiDAR,[4] a picture of which can be found in Figure 22a. A model of this exact robot configuration was also used for the experiments in simulation.

## 5.1 | Simulated Experiments

Simulation allows for the execution of experiments that could not be carried out in a real environment, due to limitations regarding the battery life, or access to testing scenarios, which, in the case of subterranean environments, can be difficult to obtain.

We propose three sets of fully-autonomous experiments to test the reliability of our solution and different use cases. The first set of experiments will showcase the path-planning and navigation stack, working on a user-defined map, in scenarios created using the DARPA SubT challenge tunnel tiles. [5]. The second and third experiments will exercise the system while using its exploration capabilities; one of them will be performed on a SubT challenge scenario while the other on a modified 3D reconstruction of the Somport tunnel.

### 5.1.1 | *Path Planning and Navigation in DARPA Subt Environments*

To carry out this set of experiments, a testing environment was created using the models provided by DARPA for their SubT challenge. These models consist of a set of tiles of different sections of tunnels that can be connected to generate arbitrarily complex environments. More specifically, the available tunnel tiles are the straight section, the 3-way intersection, the 4-way intersection, the curve, a ramp and a "block" figure to terminate the tunnels in a dead-end.

With these tiles, we built an environment rich with intersections and tunnels at different heights, which is presented in Figure 15a and the corresponding user-defined topological map was made available to the robot (Figure 15c).

To test the repeatability of the results, three different missions were devised: the first one going from the edge between nodes 4 and 1 (`E4-1`) to node 22 `N22`, the second one from `E18-17` to `N20` and the third one going from `E16-17` to `N22`. Each of these tasks was repeated 5 times resulting in a total of 15 runs. For each of the tasks, the path planner produced the following instructions respectively:

- **From** `E4-1` **to** `N22`:`take [-1], take [2], take [1], take [-1], advance_until_node`
- **From** `E18-17` **to** `N20`:`take [-1], take [-2], take [3], take [2], take [1], advance_until_node`

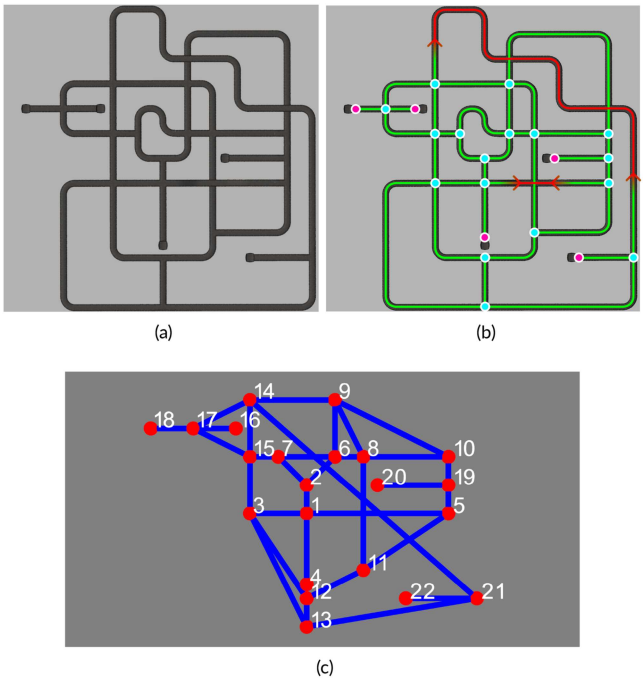


**FIGURE 15** | Environment for the path-planning tests in simulation using a user-defined map. a) Complete environment in the Gazebo simulation. b) Overlay with indications about the features of the environments. The green lines indicate tunnels at level 0, red lines indicate tunnels at level +1, blue circles are intersections, pink circles are dead-ends and arrows indicate ramps (pointing in the ascending direction). c) User-defined topological Map of the environment that the robot will use for path planning. [Color figure can be viewed at wileyonlinelibrary.com]

- **From** `E16-17` **to** `N22`:`take [-3], take [-1], take [-2], advance_until_node`

In all of them, the robot successfully arrived at the target node by following the topological instructions provided by the planner.

Figure 16 shows the trace of the robot for the first run of each of the tasks.

### 5.1.2 | *Topological Exploration in DARPA Subt Environment*

For this experiment, a different, non-cyclical environment was created using the assets provided for the DARPA SubT challenge (Figure 17a). In this scenario, there are tunnels present at three different levels, as it can be seen in Figure 17b.

The robot was then placed at position (0, 0, 0), and tasked with fully exploring the environment and obtaining the corresponding topological map.

The robot took 65 min to fully explore the environment, doing so without any errors. Figure 18a shows the topological map created by the robot during exploration. It can be seen from this image that the topological map correctly reflects the topological structure of the target environment. It is worth noting that, geometrically, the topological map is not fully consistent with the actual environment. For example, several of edges that correspond to parallel tunnels are not parallel in the map. This is due to the use of the raw odometry to assign a position to each of the nodes. This is done exclusively for the purposes of displaying the map in a more understandable form.

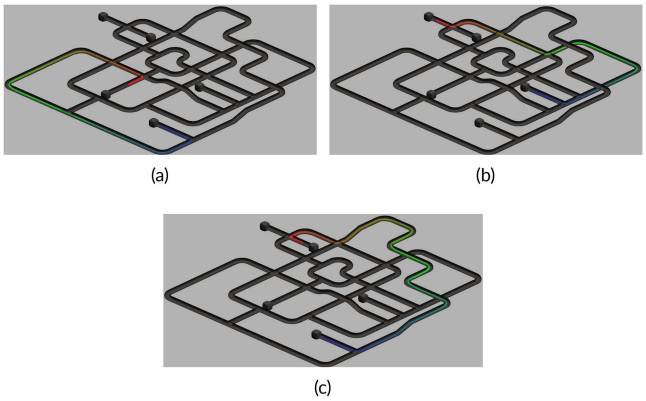


**FIGURE 16** | Traces of the first experiment for each of the path planning tasks. a–c) Each trace has a color gradient, with red indicating the initial positions, transitioning to green for the middle ones, and ending in blue for the last ones. [Color figure can be viewed at wileyonlinelibrary.com]

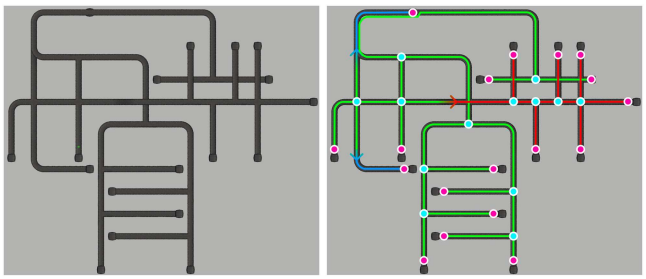


**FIGURE 17** | Environment for the topological exploration experiments. a) Complete environment in the Gazebo simulation. b) Overlay with indications about the features of the environments. The green lines indicate tunnels at level 0, red lines indicate tunnels at level +1, blue lines indicate tunnels at level −1, blue circles are intersections, pink circles are dead-ends and arrows indicate ramps from level 0 to levels +1 or −1. [Color figure can be viewed at wileyonlinelibrary.com]

However, these geometrical inconsistencies do not have any impact on the navigation, as the geometrical position of the nodes is irrelevant to our fully topological approach. The storage requirements of the resulting map can be calculated by multiplying the storage space of one edge by the number of edges. An edge is composed of two integers, each weighing 8 bytes, so the space to store one edge is 16 bytes. Given that the map contains 30 edges, it can be stored in just 480 bytes.

### *5.1.3 | Topological Exploration of a 3D Reconstruction of the Somport Tunnel*

This final set of simulated experiments has been performed in a modified 3D reconstruction of the Somport Tunnel. This environment consists of a 7 km long tunnel with 17 lateral galleries, all on the same side.

The original 3D model is an accurate representation of the real environment; however, due to its large dimensions, and the extreme aspect ratio (see Figure 23), displaying the results would not be viable. Additionally, a complete exploration of the original environment would take greater than 600 min. For these reasons, some modifications have been performed to the

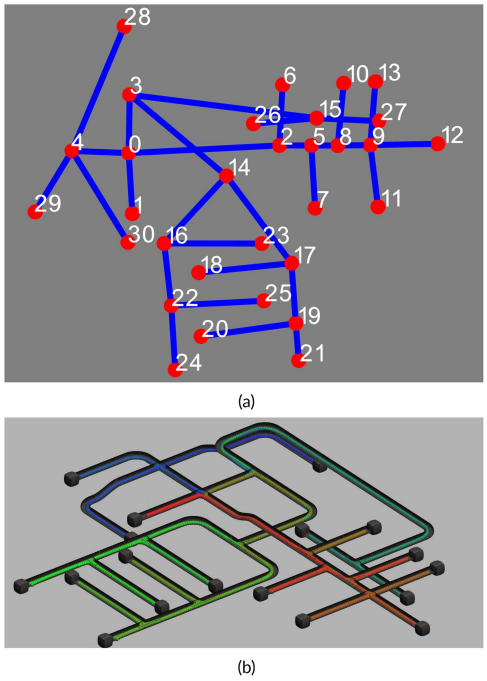


**FIGURE 18** | Results of the exploration tasks in the DARPA SubT environment. a) The obtained topological map. b) The trace of the path followed by the robot during the experiment. The trace has a color gradient, with red indicating the initial positions, transitioning to green for the middle ones, and ending in blue for the last ones. [Color figure can be viewed at wileyonlinelibrary.com]

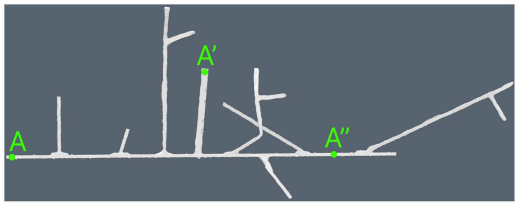


**FIGURE 19** | Modified 3D model of the Somport Tunnel, with eight lateral galleries and a reduced main tunnel of 530 m in length. The crossing galleries have different slopes, hence not really intersecting each other. [Color figure can be viewed at wileyonlinelibrary.com]

model. First, of the original 17 lateral galleries, 8 have been selected. Secondly, the distance between them has been substantially reduced. Finally, to avoid collisions between lateral galleries, one of them (corresponding to the original 6th gallery) has been flipped. The result of these modifications is shown in Figure 19, where the main tunnel has a length of 530 m.

Three starting points have been defined, which will be referred to as A, A' and A" (Figure 19). From each starting point, five exploration tasks were executed, for a total of 15 exploration runs. Of the 15, 14 were completed successfully, with the single failure being caused by an intersection not meeting the stability criteria and thus the lateral gallery not being explored. Figure 20 shows the topological maps obtained after exploration runs started from A and A'. By comparing the two, it is clear that there are some differences between them. There are slight variations between the positions of the nodes in each map, which are caused by the accumulated errors of the odometry.

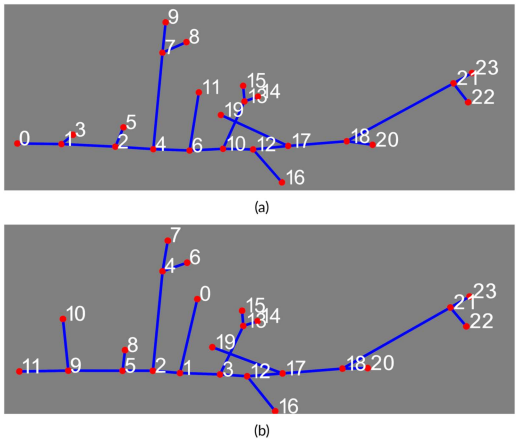


**FIGURE 20** | Topological maps obtained from explorations starting from A (a) and A′ (b). [Color figure can be viewed at wileyonlinelibrary.com]

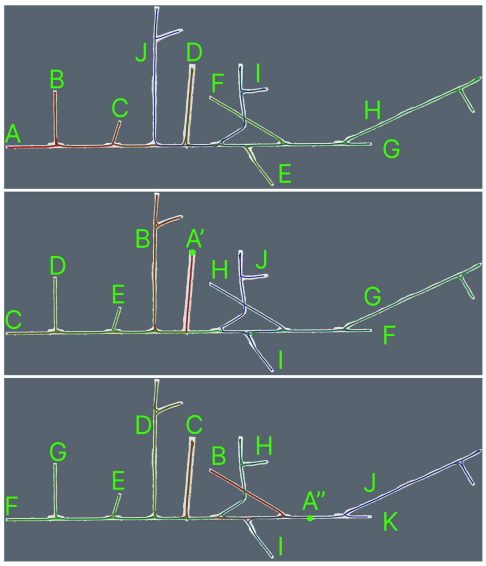

**FIGURE 21** | Traces obtained during the exploration experiments in the modified 3D model of the Somport Tunnel from three different starting points (A, A′ and A″). They present a color gradient starting in red, transitioning to green and finally ending in blue. The green letters enumerate the order at which the robot has entered each of the branches from the main tunnel. [Color figure can be viewed at wileyonlinelibrary.com]

Given that these maps contain 23 edges each, the map can be stored in only 368 bytes of memory.

Additionally, the enumeration of the nodes depends on when they are created, so they also vary between topological maps. Regardless of these differences, the underlying topological structure stays the same.

Finally, Figure 21 shows the traces of three different exploration runs, each of them starting from a different point.

## 5.2 | Real-World Experiment

The main objective of this test was to validate the complete framework in a real-world environment and to showcase the zero-shot transfer from simulation to reality when using deep learning on 3D LiDAR scans. It was performed in the Somport Tunnel, which has been described previously. As can be seen in Figure 22b, the tunnels have substantially degraded floors, which make odometry much less reliable, and the main tunnel has few geometrical features (Figure 22c), complicating the process of scan matching of more traditional geometric methods. The complete environment is of significant size, so our experiment is carried out in 3 of the 17 galleries (Figure 23).

For this experiment, the task that the robot had to perform was the following: Starting from the end of G8, reach the end of G7. Then, go to G9 and advance 30 m. Finally, return to the intersection of G8 and stop. During this process, the system also had to build a topological map of the environment. This experiment aims to showcase three different aspects of our proposal:

1. The capability to carry out missions specified using high-level instructions, without the need to have a pre-built map of the environment or build one.
2. The zero-shot transfer learning between simulation and the real-world environment.
3. The mapping approach working on a real-world environment.

To perform the task defined above, we provided the navigation system with the following set of topological instructions:

1. `advance_to_node`: To go from the end of *G*8 to the main tunnel.
2. `take [right]`: To exit intersection in the direction of *G*7.
3. `take [right]`: Will advance until reaching the intersection with *G*7, and enter into *G*7.
4. `advance_to_node`: To reach the end of *G*7.
5. `take [back]`: To exit the end of *G*7 and face towards the main tunnel.
6. `take [left]`: To reach the main tunnel and face towards *G*8.
7. `take [1]`: Advance towards the intersection of *G*8 and continue straight.
8. `take [left]`: Advance towards the intersection of *G*9 and enter *G*9.
9. `advance_met 20`: Advance 20 m into *G*9.
10. `turn_around`: To face the intersection of *G*9.
11. `take [right]`: To reach the intersection of *G*9 and face the intersection of *G*8.
12. `advance_to_node`: To advance through the main tunnel to the intersection of *G*8.

During the run, our mapping approach generated the topological map shown in Figure 24. As mentioned before, for visualization reasons, the position of each node is set using the raw odometry from the robot, which accumulates error over time. This is especially true in this context, where the floor is substantially degraded. For this reason, the poses of `N0`, `N1`, `N2` and `N4` are much more consistent with the 3D model than the position of `N3`, which is the last visited node.

The robot took 35 min to traverse the 1572 m of the assigned task, and managed to do so without any intervention. To provide an


**FIGURE 22** | Pictures taken inside the Somport tunnel during our experimental run. a) Close-up of our robotic platform, a Clearpath A200 equipped with a VLP-16 LiDAR sensor. b) Picture taken from the intersection of G7 with the main tunnel, where the floor is especially rough. c) Picture taken in the main tunnel, between G8 and G7, it can be seen that the tunnel walls present few geometric features. d) Entrance to G8 seen from the main tunnel. [Color figure can be viewed at wileyonlinelibrary.com]

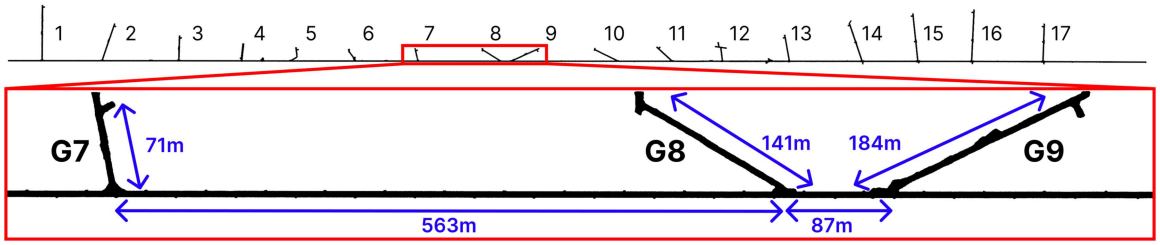


**FIGURE 23** | Ground-truth reconstruction of the Somport Tunnel made with the use of high-precision topographic equipment. The top image is the complete environment while the bottom image is a zoom on the section where the experiment takes place. The distances, shown in blue, have been measured using the ground truth data. [Color figure can be viewed at wileyonlinelibrary.com]

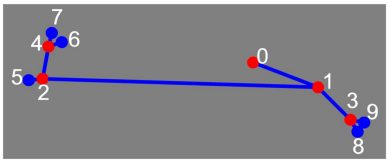


**FIGURE 24** | Topological map generated during the real-world experiment in the Somport Tunnel. Red circles indicate visited nodes, blue circles indicate unvisited nodes while blue lines indicate edges. The numbers are the ID of each of the nodes. The position of each node is established using the raw odometry. [Color figure can be viewed at wileyonlinelibrary.com]

approximation of the route that the robot followed, Figure 25 shows the best attempt at reconstructing a geometric map from the data gathered during the experiment, which has some imperfections due to the reasons presented in Section 2.1. The reconstructed path of the robot is shown as a red line, which also presents some inconsistencies and jumps, but it does show how the robot stays close to the center of the tunnel at all times.

## 6 | Discussion

Our experimental evaluation is aimed at demonstrating the viability and usability of a fully topological approach for the tasks of navigating, mapping and exploring underground tunnel networks.

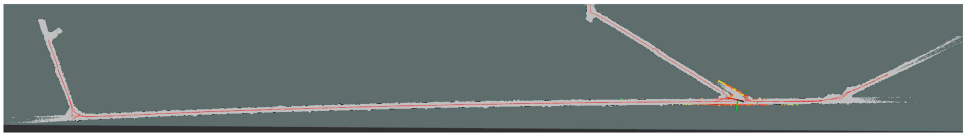

**FIGURE 25** | Best geometrical attempt at reconstructing the experiment in the Somport tunnel. The rough floor degrades the odometry and the lack of features in the walls are challenging for the scan matching. This results in the duplication of elements like galleries and sidings. The red line is the reconstructed path of the robot, while the axes on the middle intersection represent the final position of the robot. [Color figure can be viewed at wileyonlinelibrary.com]

Repeated experiments show a high degree of reliability, which is accomplished in no small part by removing the necessity of having a precise self-localization. It achieves a success rate of 100% in all tasks in the DARPA-based environments, and of 93.3% in the 3D reconstruction of the Somport Tunnel.

In the exploration tasks, it manages to create a very light representation of substantially large environments, managing to store all necessary information for navigation in maps that occupy storage in the order of hundreds of bytes.

Regarding the real-world experiment, we demonstrate that, even in a challenging real environment (Section 2.1), our method is capable of following a set of simple instructions that can be communicated directly by a human operator, and operate autonomously for 40 min while it builds a topological map of the traversed environment.

All these advantages come from exploring a purely topological understanding of these environments. However, due in part to this focus, and in part to the implementation details, our approach presents some limitations. The CNN that our method relies on has been tailored to work in scenarios composed of tunnels and intersections. For this reason, if parts of the environment deviate strongly from these assumptions, the perception pipeline will not perform as intended. Additionally, the focus on the topological aspect means that, for example, any position inside the same tunnel is the same for our method, so if there is a requirement to operate in a specific point along a tunnel, our method would be insufficient. Additionally, as will be discussed in Section 4.4, using a purely topological representation makes the loop-closure problem an extremely difficult one to solve. Even taking this into account, we consider our solution a solid demonstration of how it can help with some of the issues that these environments tend to pose.

### 6.1 | Lessons Learned

Lessons learned from navigating underground environments highlight the many compounding challenges that make reliable navigation extremely difficult.

It is well known that odometry, which often works well in structured or predictable settings, becomes highly unreliable in these environments due to uneven surfaces, wheel slip, and sudden changes in terrain, causing accumulated errors that rapidly degrade positional estimates.

General-purpose SLAM methods, used to correct such drift, work extremely well in more structured environments but are themselves problematic here: the high degree of auto-similarity and long stretches of nearly identical tunnels or corridors cause frequent mismatches, loop-closure errors and often map inconsistencies. Additionally, SLAM algorithms are notably sensitive to parameter tuning; settings that work in one section of the environment can fail severely in another, making robust operation difficult without careful, location-specific calibration.

IMUs, commonly used to provide short-term motion estimates also struggle: potholes, uneven floors, and frequent rattling and vibrations introduce significant noise, limiting their usefulness for long-term localization.

Finally, neural network-based approaches, while attractive for their potential to learn highly diverse patterns and features, show sometimes inconsistent performance which makes it difficult to know when the network is providing a reliable output and when is not. In this specific case, this forced us to "engineer" a method to manage the instability of the network output in critical areas of the environment (mainly at intersections).

Finally, we learned that using synthetic data to train networks that rely on LiDAR readings often produces quite satisfactory results, allowing zero-shot transfer from simulation to the real world.

## 7 | Conclusions

The interest in autonomous robotics in underground environments has increased substantially during the last decade, in part thanks to the DARPA's SubT challenge. However, the progress has concentrated on adapting geometrical approaches to these environments, while topological methods have not received the same interest. Topological methods represent their environments as a graph, which makes them especially well-suited to handle underground environments, which typically consist of tunnels and galleries that intersect.

In this article we have presented a purely topological method for the navigation, mapping and exploration of subterranean tunnel networks, which are environments especially suitable for this approach. We begin by establishing a purely topological representation of these these scenarios, where the tunnels are interpreted as edges and the intersections and dead-ends as nodes. To exploit this, we propose a perception system based on a CNN, that can interpret 3D LiDAR data and obtain the distribution of the tunnels around the robot. This novel method of perception is then used to create a navigation system based on human-readable, high-level instructions. Finally, these perception and navigation approaches allow us to create a purely topological mapping and exploration system.

Our purely topological approach provides some definite advantages over traditional methods, as is the complete disregard for precise self-localization (an especially challenging task in our target environments). Another important advantage is the substantial reduction in the size of the maps, which can be of substantial usefulness in such communication-degraded environments.

This work also provides successful experimental demonstrations of the system, both in simulation and in a real-world environment. Simulation experiments have focused on the path-planning, navigation and exploration capabilities of our method. The real world experiment has shown how a user can specify a mission using high-level instructions, without needing to provide the robot with any information about the environment. Additionally, it has proven that our network presents a zero-shot transfer learning from a completely synthetic dataset, obtained in simulation, to a real-world environment.

In future work, our main priorities will be exploring methods for purely topological loop-closure and implementing fallback strategies that do not require human intervention in case of failure or mismatch between the topological map and the actual environment.


### Acknowledgments

This research was supported by the project D2022-139615OB-I00 "ROBOCOMPLEX" funded by the Spanish Ministry of Science, Innovation and Universities (MICIU).


### Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

### Endnotes

[1] Defense Advanced Research Projects Agency (https://www.darpa.mil).

[2] https://www.darpa.mil/program/darpa-subterranean-challenge.

[3] https://www.clearpathrobotics.com/wp-content/uploads/2013/02/HUSKY_A200_UGV_2013_TEASER_email.pdf.

[4] https://ouster.com/products/hardware/vlp-16.

[5] https://app.gazebosim.org/openrobotics/fuel/collections/SubT%20Tech%20Repo.

### References


Bai, C., T. Xiao, Y. Chen, H. Wang, F. Zhang, and X. Gao. 2022. "Faster-Lio: Lightweight Tightly Coupled Lidar-Inertial Odometry Using Parallel Sparse Incremental Voxels." *IEEE Robotics and Automation Letters* 7, no. 2: 4861–4868.

Bayer, J., and J. Faigl. 2021. "Decentralized Topological Mapping for Multi-Robot Autonomous Exploration Under Low-Bandwidth Communication." In *2021 European Conference on Mobile Robots (ECMR)*, 1–7. IEEE.

Blanco, J.-L., J.-A. Fernández-Madrigal, and J. Gonzalez. 2008. "Toward a Unified Bayesian Approach to Hybrid Metric-Topological Slam." *IEEE Transactions on Robotics* 24, no. 2: 259–270.

Blochliger, F., M. Fehr, M. Dymczyk, T. Schneider, and R. Siegwart. 2018. "Topomap: Topological Mapping and Navigation Based on Visual Slam Maps." In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 3818–3825. IEEE.

Bosse, M., P. Newman, J. Leonard, M. Soika, W. Feiten, and S. Teller. 2003. "An Atlas Framework for Scalable Mapping." In *2003 IEEE International Conference on Robotics and Automation (Cat. No. 03CH37422)*, 1899–1906. IEEE.

Cano, L., A. R. Mosteo, and D. Tardioli. 2022. "Navigating Underground Environments Using Simple Topological Representations." In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1717–1724. IEEE.

Cano, L., A. R. Mosteo, and D. Tardioli. 2024a. "Procedural Generation of Tunnel Networks for Unsupervised Training and Testing in Underground Applications." In *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.

Cano, L., D. Tardioli, and A. R. Mosteo. 2023. "Fast Tunnel Traversal for Ground Vehicles by Bearing Estimation With Neural Networks." In *Iberian Robotics Conference*, 284–296. Springer.

Cano, L., D. Tardioli, and A. R. Mosteo. 2024b. "Purely Topological Exploration of Underground Environments." In *2024 7th Iberian Robotics Conference (ROBOT)*, 1–8. IEEE.

Chang, H. J., C. S. G. Lee, Y. C. Hu, and Y.-H. Lu. 2007. "Multi-Robot Slam With Topological/Metric Maps." In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1467–1472. IEEE.

Chaplot, D. S., R. Salakhutdinov, A. Gupta, and S. Gupta. 2020. "Neural Topological Slam for Visual Navigation." In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE.

Chen, J., H. Wang, and S. Yang. 2023. "Tightly Coupled Lidar-Inertial Odometry and Mapping for Underground Environments." *Sensors* 23, no. 15: 6834.

Chen, K., B. T. Lopez, A.-A. Agha-mohammadi, and A. Mehta. 2022. "Direct Lidar Odometry: Fast Localization With Dense Point Clouds." *IEEE Robotics and Automation Letters* 7, no. 2: 2000–2007.

Cowley, A., C. J. Taylor, and B. Southall. 2011. "Rapid Multi-Robot Exploration With Topometric Maps." In *2011 IEEE International Conference on Robotics and Automation*, 1044–1049. IEEE.

Dang, T., M. Tranzatto, S. Khattak, F. Mascarich, K. Alexis, and M. Hutter. 2020. "Graph-Based Subterranean Exploration Path Planning Using Aerial and Legged Robots." *Journal of Field Robotics* 37, no. 8: 1363–1388.

de S Thiago Filho, A. M., I. F. S. Amaral, N. C. P. de S Thiago Neto, et al. 2025. "Robotic Pipe Inspection: Low-Cost Device and Navigation System." *Journal of Control, Automation and Electrical Systems* 36, no. 1: 101–116.

Dijkstra, E. W. 1959. "A Note on Two Problems in Connexion With Graphs." *Numerische Mathematik* 1, no. 1: 269–271.

Donoghue, A. M. 2004. "Occupational Health Hazards in Mining: An Overview." *Occupational Medicine* 54, no. 5: 283–289.

Duberg, D., and P. Jensfelt. 2022. "Ufoexplorer: Fast and Scalable Sampling-Based Exploration With a Graph-Based Planning Structure." *IEEE Robotics and Automation Letters* 7, no. 2: 2487–2494. https://doi.org/10.1109/LRA.2022.3142923.

Ebadi, K., L. Bernreiter, and H. Biggie, et al. 2022. "Present and Future of Slam in Extreme Underground Environments: The Darpa Subt Challenge." *IEEE Transactions on Robotics* 40, no. 1: 936–959.

Fredriksson, S., A. Saradagi, and G. Nikolakopoulos. 2024. "Grid-Fast: A Grid-Based Intersection Detection for Fast Semantic Topometric Mapping." *Journal of Intelligent & Robotic Systems* 110, no. 4: 154.

Grisetti, G., C. Stachniss, and W. Burgard. 2007. "Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters." *IEEE Transactions on Robotics* 23, no. 1: 34–46.

Hughes, N., Y. Chang, and L. Carlone. 2022. "Hydra: A Real-Time Spatial Perception System for 3D Scene Graph Construction and Optimization." In *Robotics: Science and Systems (RSS)*.

Ji, S., W. Xu, M. Yang, and K. Yu. 2012. "3d Convolutional Neural Networks for Human Action Recognition." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, no. 1: 221–231.

Kingma, D. P. 2014. Adam: A Method for Stochastic Optimization." In *International Conference on Learning Representations (ICLR 2015)*.

Kohlbrecher, S., O. Von Stryk, J. Meyer, and U. Klingauf. 2011. "A Flexible and Scalable Slam System With Full 3d Motion Estimation." In

*2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, 155–160. IEEE.

Kostavelis, I., and A. Gasteratos. 2015. "Semantic Mapping for Mobile Robotics Tasks: A Survey." *Robotics and Autonomous Systems* 66: 86–103.

Li, X. S., T. Nguyen, A. G. Cohn, M. Dogar, and N. Cohen. 2023. "Real-Time Robot Topological Localization and Mapping With Limited Visual Sampling in Simulated Buried Pipe Networks." *Frontiers in Robotics and AI* 10: 1202568.

Mansouri, S. S., C. Kanellakis, G. Georgoulas, and G. Nikolakopoulos. 2018. "Towards Mav Navigation in Underground Mine Using Deep Learning." In *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 880–885. IEEE.

Mansouri, S. S., C. Kanellakis, D. Kominiak, and G. Nikolakopoulos. 2020. "Deploying Mavs for Autonomous Navigation in Dark Underground Mine Environments." *Robotics and Autonomous Systems* 126: 103472.

Mascaró, M., I. Parra-Tsunekawa, C. Tampier, and J. Ruiz-delSolar. 2021. "Topological Navigation and Localization in Tunnels-Application to Autonomous Load-Haul-Dump Vehicles Operating in Underground Mines." *Applied Sciences* 11, no. 14: 6547.

Mitchell, R. J., T. Driscoll, and J. E. Harrison. 1998. "Traumatic Work-Related Fatalities Involving Mining in Australia." *Safety Science* 29, no. 2: 107–123.

Montano-Oliván, L., J. A. Placed, L. Montano, and M. T. Lázaro. 2024. "G-Loc: Tightly-Coupled Graph Localization With Prior Topo-Metric Information." *IEEE Robotics and Automation Letters*9, no. 11: 9167–9174.

Montero, R., J. G. Victores, S. Martinez, A. Jardón, and C. Balaguer. 2015. "Past, Present and Future of Robotic Tunnel Inspection." *Automation in Construction* 59: 99–112.

Morlana, J., J. D. Tardós, and J. M. Montiel. 2024. "Topological Slam in Colonoscopies Leveraging Deep Features and Topological Priors." In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, 733–743. Springer.

Morris, A., D. Silver, D. Ferguson, and S. Thayer. 2005. "Towards Topological Exploration of Abandoned Mines." In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2117–2123. IEEE.

Muravyev, K., A. Melekhin, D. Yudin, and K. Yakovlev. 2025. "Prism-Topomap: Online Topological Mapping With Place Recognition and Scan Matching." *IEEE Robotics and Automation Letters* 10, no. 4: 3126–3133.

Pereira, G., C. Duarte, D. Marques, H. Azpúrua, G. Pessin, and G. Freitas. 2021. "Towards a Simple Navigation Strategy for Autonomous Inspection of Ducts and Galleries." In *2021 Latin American Robotics Symposium (LARS), 2021 Brazilian Symposium on Robotics (SBR), and 2021 Workshop on Robotics in Education (WRE)*, 336–341. IEEE.

Placed, J. A., J. J. G. Rodríguez, J. D. Tardós, and J. A. Castellanos. 2022. "Explorb-Slam: Active Visual Slam Exploiting the Pose-Graph Topology." In *Iberian Robotics Conference*, 199–210. Springer.

Prados Sesmero, C., S. Villanueva Lorente, and M. Di Castro. 2021. "Graph Slam Built over Point Clouds Matching for Robot Localization in Tunnels." *Sensors* 21, no. 16: 5340.

Qi, C. R., H. Su, K. Mo, and L. J. Guibas. 2017. "Pointnet: Deep Learning on Point Sets for 3d Classification and Segmentation." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*: 652–660.

Ren, Z., L. Wang, and L. Bi. 2019. "Robust Gicp-Based 3d Lidar Slam for Underground Mining Environment." *Sensors* 19, no. 13: 2915.

Ribas, D., P. Ridao, J. Neira, and J. D. Tardos 2006. "Slam Using an Imaging Sonar for Partially Structured Underwater Environments." In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5040–5045. IEEE.

Rizzo, C., D. Tardioli, D. Sicignano, L. Riazuelo, J. L. Villarroel, and L. Montano. 2013. "Signal-Based Deployment Planning for Robot Teams in Tunnel-Like Fading Environments." *International Journal of Robotics Research* 32, no. 12: 1381–1397.

Romeo, A., and L. Montano. 2006. "Environment Understanding: Robust Feature Extraction From Range Sensor Data." In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 3337–3343. IEEE.

Ross, M., and J. Murray. 2004. "Occupational Respiratory Disease in Mining." *Occupational medicine* 54, no. 5: 304–310.

Rossi, C., A. Caro Zapata, Z. Milosevic, R. Suarez, and S. Dominguez. 2023. "Topological Navigation for Autonomous Underwater Vehicles in Confined Semi-Structured Environments." *Sensors* 23, no. 5: 2371.

Saroya, M., G. Best, and G. A. Hollinger. 2020. "Online Exploration of Tunnel Networks Leveraging Topological CNN-based World Predictions." In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 6038–6045. IEEE.

Simonyan, K., and A. Zisserman. 2014. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In *International Conference on Learning Representations (ICLR 2015)*.

Tampier, C., M. Mascaró, and J. Ruiz-delSolar. 2021. "Autonomous Loading System for Load-Haul-Dump (LHD) Machines Used in Underground Mining." *Applied Sciences* 11, no. 18: 8718.

Tardioli, D., L. Riazuelo, D. Sicignano, et al. 2019. "Ground Robotics in Tunnels: Keys and Lessons Learned After 10 Years of Research and Experiments." *Journal of Field Robotics* 36, no. 6: 1074–1101.

Tardioli, D., D. Sicignano, L. Riazuelo, A. Romeo, J. L. Villarroel, and L. Montano. 2016. "Robot Teams for Intervention in Confined and Structured Environments." *Journal of Field Robotics* 33, no. 6: 765–801.

Wang, H., C. Wang, C. Chen, and L. Xie. 2020 F-Loam: Fast Lidar Odometry and Mapping." In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE.

Worley, R., and S. R. Anderson. 2025. "Hybrid Metric-Topological Localization for Robots in Pipe Networks." *Journal of Field Robotics* 42, no. 3: 806–826.

Xue, W., R. Ying, Z. Gong, R. Miao, F. Wen, and P. Liu. 2020. "Slam Based Topological Mapping and Navigation." In *2020 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, 1336–1341. IEEE.

Yang, X., X. Lin, W. Yao, H. Ma, J. Zheng, and B. Ma. 2022. "A Robust Lidar Slam Method for Underground Coal Mine Robot With Degenerated Scene Compensation." *Remote Sensing* 15, no. 1: 186.

## Supporting Information

Additional supporting information can be found online in the Supporting Information section.
Supplementary Information