Navigating underground environments using simple topological representations

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

***Abstract*— Underground environments are some of the most challenging for autonomous navigation. The long, featureless corridors, loose and slippery soils, bad illumination and unavailability of global localization make many traditional approaches struggle. In this work, a topological-based navigation system is presented that enables autonomous navigation of a ground robot in mine-like environments relying exclusively on a high-level topological representation of the tunnel network. The topological representation is used to generate high-level topological instructions used by the agent to navigate through corridors and intersections. A convolutional neural network (CNN) is used to detect all the galleries accessible to a robot from its current position. The use of a CNN proves to be a reliable approach to this problem, capable of detecting the galleries correctly in a wide variety of situations. The CNN is also able to detect galleries even in the presence of obstacles, which motivates the development of a reactive navigation system that can effectively exploit the predictions of the gallery detection.**

I. Introduction

Underground scenarios are notoriously difficult to navigate autonomously as they present many characteristics that make them very challenging for common navigation strategies as shown, for example, in [1] or [2].

One problem present in underground environments is the lack of global external localization aids such as GPS. This makes the agent very dependent on sensors like odometers and inertial measuring units, that are susceptible to the accumulation of error during operation. To worsen things, the floors of these scenarios are most commonly very irregular and loose, which increases the slippage on the wheels, which, in turn, increases the errors in the odometry and inertial measurements. All of these make precise localization in these environments exceptionally difficult.

Also, these settings tend to be in complete darkness and, if lit, the lightning tends to be very irregular, with some sparse bright spots surrounded by darkness. Furthermore, in the case of mines or other human-made settings, the extraction of rocks and minerals fills the environment with dust, deteriorating visibility even further. This is a hindrance to vision-based approaches, that have to deal with poorly illuminated and low visibility images, and often force the agent to carry its own light source. In [3], the authors elaborate on the challenges of using cameras for robotics in underground environments and in [4], the effect of dust on LiDAR sensors is studied, with the conclusion that reliable readings can still be obtained in the presence of dust.

Another challenging aspect of underground scenarios is that they usually consist in long, featureless corridors and intersections between these corridors (Fig. 1). The lack of clear and distinct features (specially in LiDAR readings) makes SLAM systems struggle, as they need them to iteratively reduce the errors caused by odometry and inertial measurements. Long corridors also complicate the loop-closure methods used in most SLAM systems, further increasing difficulty. In [3] the authors show how different state-of-the-art SLAM systems struggle to give accurate pose estimation in these kinds of scenarios. At the same time, these scenarios are some of the most hazardous workplaces in the modern world. Some of the dangers associated with them are very low quality air, risk of collapse, repetitive and monotonous work and continuous exposure to loud noises. All these reasons make underground operations one of the fields where automation could have the greatest positive impact.

Other underground scenarios are very difficult for humans to traverse, like vertical shafts or flooded sections in caves. And, finally, there are some scenarios not accessible at all to humans, like long-term nuclear waste storage sites [5]. In these situations, robots capable of navigating these scenarios are of great help to many people; from research oriented groups like geologists or biologists, to rescue teams that have to operate in collapsed infrastructure or engineers in charge of revising critical infrastructure not accessible to people (oil pipelines, sewage systems, etc.).

In this work, we propose a navigation system based on high-level topological information of scenarios like the ones previously described, especially those composed of long corridors that intersect each other. The goal is to make a robot capable of navigating from a starting point to an objective given as a set of high-level directives (e.g. "take the second on the left and then the third on the right"), which is closer to how humans move in similar environments, and thus without the need for precise geometric localization, nor representation of the scenario.

To do so, the robot must recognize the features encountered along its path, such as intersections and corridors, in order to execute the high-level instructions at the right time, thus following the desired path. This feature recognition is accomplished by processing the on-board 3D LiDAR data through a convolutional neural network, whose details will be explained later on. Additionally, a specialized reactive navigation algorithm for obstacle avoidance has been devel-

[1]Universidad de Zaragoza, [2]Centro Universitario de la defensa, Zaragoza, [3]Instituto de Investigación en Ingeniería de Aragón. Email: {lcano, amosteo, dantard}@unizar.es}

This work was supported by the project PID2019-105390RB-I00 founded by Spanish MICINN.

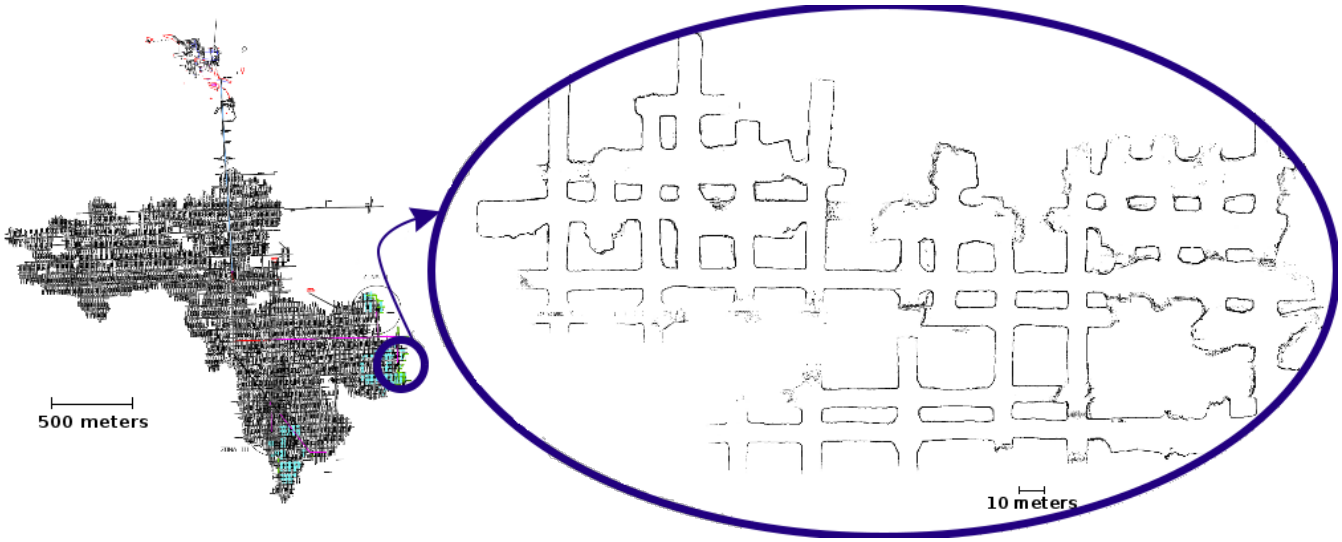


Fig. 1: Map of a portion of the Santa Marta mine (Spain).

oped that takes advantage of the network output.

In this scheme, we assume the existence of a topological representation of the tunnel network from which the already mentioned directives are extracted when a plan is requested.

The paper structure is based on the three main subsystems of the proposed navigation approach: the topological representation of the environment (section II), the gallery detection system (section III) and the instruction-based navigation (IV-A). Finally, in section V, we detail the tests used to validate this approach, and the results obtained.

A. *Related work*

For all the reasons presented on the previous section, many of the usual navigation strategies struggle in underground settings. This has motivated the development of many different approaches that tackle a variety of use cases in subterranean scenarios.

In the context of industrial tunneling and excavating machines, very precise absolute location is often required, while flexibility is not an important factor. In [6] the use of a theodolite is combined with a measurement device mounted in a tunnel boring machine so that very precise orientation information of the machine can be obtained. This approach provides great accuracy, at the cost of requiring continuous line of sight between the boring machine and the theodolite operator, and the system has to be set up for each gallery. A different approach that also achieves remarkable accuracy is [7], where they manage to maintain an accurate pose estimation for a duration of over a week using a localization system based exclusively on inertial and odometry measurements, given that the robot operates at low speeds.

In the field of autonomous Micro Aerial Vehicles (MAVs) cameras are a popular sensor given their low weight. In [8], [9] the authors propose a navigation approach based on controlling the heading of the MAV, so it advances along the axis of the tunnel. To do this they use a CNN that classifies the images of the drone as facing to the center, the left or the right of the tunnel or detects the center of the tunnel in the image. In [10], a topological approach is used to navigate corridor-like environments. Using a convolutional neural network they classify the front-facing camera images into different topological features (like turns, intersections or dead-ends), and use that information to navigate.

On the topic of reactive navigation in confined environments, specifically corridors, in [11] the Hough transform is used to detect the corridors present in laser-scan data. In [2] the author describes a centering algorithm that helps UGV vehicles to traverse corridors safely.

Topological and metric-topological representations have already been extensively used in underground robotics to help with navigation, localization and exploration of subterranean environments. In [2], edges and vertices are extracted from 2D geometrical representations of mines, and then used to navigate based on that graph, using a centering algorithm. In [12] the authors process on-board laser-scan data using Principal Component and Linear Discriminant analysis, and then use Bayes Decision theory on the results to classify the laser-scan readings into features common in underground environments, like intersections, dead-ends or corridors. In [13] a metric-topological map based on common features (corridors, intersections) and laser-scan measurements is used to enable the haul-dump vehicles operating inside a mine to self-localize and path-plan. The authors also develop a movement-graph, based on the topological map, that allows the robot to execute complex maneuvers inside very limited space. In [14], a graph representation is used for the global path-planning in drone exploration tasks in underground environments.

The common topological features present in underground environments can be exploited in more ways than navigation and localization. In [15] the authors develop a world-prediction technique based on convolutional neural networks that exploits the topological features of the known environ-

ment to predict the unknown parts. They then use these predictions to more efficiently explore the scenario.

A different approach to topological navigation is presented in [16] and [17]. In these approaches the topological map is a graph structure where each node corresponds to an image of the environment, and the edges of the graph represent the real world connectivity. Neural networks are used to localize the robot inside the topological graph, and to generate the commands to move from node to node.

II. Rationale

Most human-made underground environments consist of a set of tunnels that intersect each other (Fig. 1). For a person, if given a map of the mine and an objective to reach inside it, intermediate objectives would not be of the kind "walk 200 meters then go right". Instead, they would probably take the form of "skip one intersection and go to the right on the second intersection".

This reasoning implies that precise geometric information of the tunnels is not a hard requirement for navigation and, thus, it should be possible for a robot to navigate using only a graph representation of the environment. A possible representation consists in defining a graph in which nodes are features (intersections, dead ends, corridors) connected by edges. In this case, a high-level plan would consist in a sorted list of nodes (features) the robot has to visit by moving through corridors before reaching the final node.

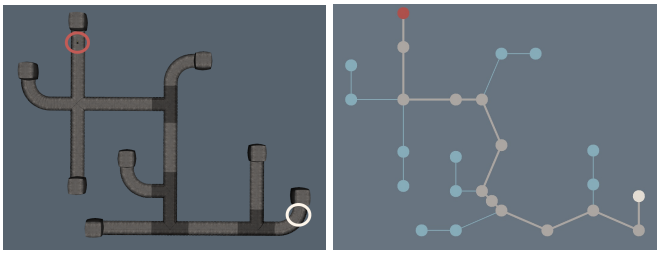
However, to put the plan into practice, at least two aspects must be taken into account. On the one hand, the robot must be able to recognize when it has reached a new node, where it must fetch the next instruction of the plan (for example, take the exit on the left). On the other hand, the robot must be able to execute that instruction; for example, navigating to the right exit of the intersection, and entering the following corridor without crashing and avoiding obstacles at the same time.

A. Topological Representation of a Tunnel Network

In this work we represent a gallery network as a graph in which vertices (nodes) represent features extracted from the environment, linked by edges when there is a corresponding real-world connection.

More precisely, a feature represents a contiguous area of the map where the same set of exits is available to reach other areas. Exits are meant not geometrically but topologically, such as the two possible ways in any location within a corridor.

Fig. 2.a) shows a sample environment while fig. 2.b) shows the corresponding undirected graph that, for visualization purposes, is presented with a layout similar to the environment, and in which each node has been associated with a feature. In this case, it is possible to identify intersections, T-intersections, dead ends and corridors with four, three, one and two edges (i.e. connections) respectively. Our scheme is not limited to these features, given that any number of connections to a given node can be included in the graph. Notice also that curves and straight corridors are not distinct features, as they both have two exits. Likewise, a long gallery with many curves is represented by a unique node, no matter if it is 20 meters, or 4 kilometers long.



(a) (b)

Fig. 2: Tunnel network in Gazebo (a) with origin position (red, top-left) and goal position (white, bottom-right), along with the corresponding topological representation (b) and generated node path. Topological instructions to follow the path: $[1, -1, -1, 1, 1, 2, 1, 2, -1, 1, -1]$ (as explained in Sec. II-B).

B. Generation of the navigation instructions

Topological instructions are meant to indicate which exit the robot must take when reaching a new topological node. This implies that for each node in the topological path of the robot, a navigation instruction must be specified. Finding a sequence of instructions on the graph is done for example using Dijkstra's algorithm [18]. Fig. 2.b) shows in gray the path computed to go from the red node to the white one.

Once the path is known, it can be converted into topological instructions given that each node knows the relative ordering of their neighbours, computed from the exit angle of each edge. In particular, in our approach, instructions are positive and negative integer numbers, assigned with respect to the gallery the robot is coming from when it reaches an intersection: the exits on the right are numbered with a positive integer number while those on the left with a negative one as shown in Fig. 3.a). In this way, the instructions for a specific path will be a list of integer numbers that specify the chosen exit in each node as shown in Fig. 3.b).

In this way we can handle any kind of intersection, be it a 4-way, a bifurcation, lateral gallery, etc., no matter the number of exits and their geometric coordinates, using the same type of instruction. It also means that an instruction may have different meanings in different intersections. For example, a *-1* instruction in a 4-way intersection means to go left, but that same instruction in a 3-way intersection where there is a gallery in front and one on the right would mean to go forward.

III. Gallery Detection

As the topological instructions specify what gallery to take at each intersection, it is necessary for the correct operation of this method to reliably detect the presence of intersections and exits. For this, a CNN has been used, as it gives us

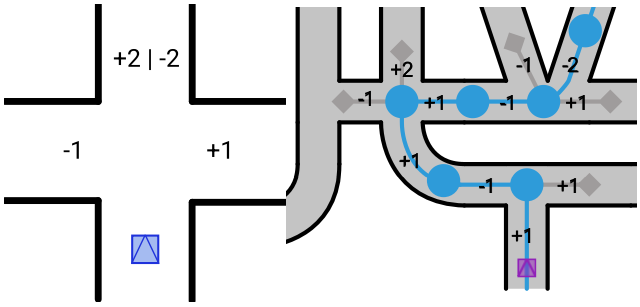


Fig. 3: Numbering of exits in a node. (a) Example of an intersection. (b) Sequence of instructions that accomplish a particular topological path.

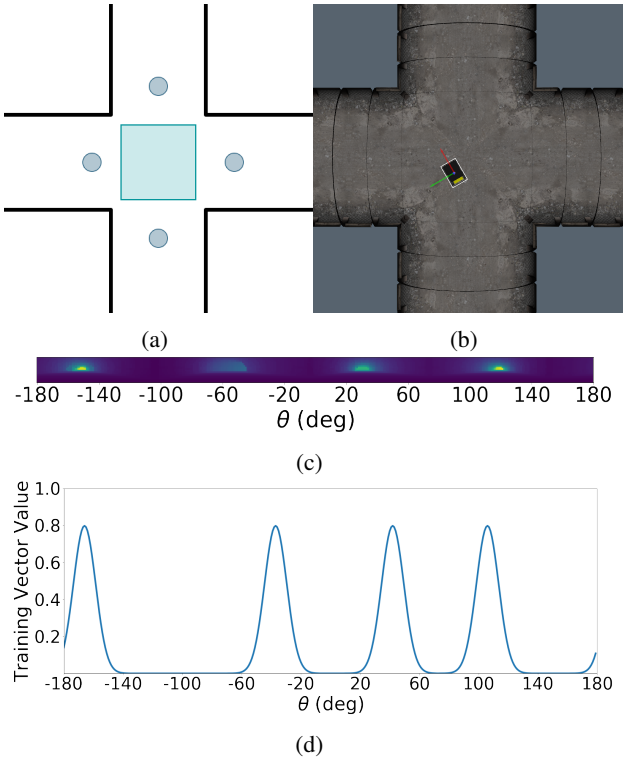


Fig. 4: Training sample from the central area of a 4-way intersection tile. (a) Diagram of the training area with the placement of the exits represented by circles. (b) Robot placed in a random point inside the training area in simulation. (c) Depth image seen by the robot. (d) Training label generated from the position of the robot and exits.

more flexibility and generalization than other methods (for example, Hough transform methods can identify a limited set of parameters). The goal is to detect the presence of exits at intersections and their relative angular position in order to be able to make the decision about what exit the robot needs to take and so follow the path instructions.

The CNN takes as input a depth image generated from the point-cloud readings of an on-board LiDAR. The resulting image is 360 pixels wide by 16 pixels high, with the intensity of each pixel being proportional to the distance read by the LiDAR. Before inputting the image to the CNN, it is normalized by dividing it by its highest value. The expected output of the network is a vector of 360 floating-point numbers that indicates the presence of exits as peaks, centered around the angle at which the exit is found.

A. *Network Architecture*

The CNN used in this work differs significantly from other, more established architectures, used for image recognition. The reason behind this is that the output is closer to a generative model than it is to a classification model.

The network is composed by a convolutional block with 5 convolutional layers, with kernels of size $[3, 3]$ and a fully connected block with 4 layers. All layers, both convolutional and fully connected, use ReLU activation. In the convolutional block there are 3 Max Pooling layers, after the first, third and fifth layers. Additionally, batch normalization has been used after each of the convolutional layers.

Circular padding is applied on the lateral edges of the images before each convolutional layer, while no padding is applied on the upper and lower edges. In this way, the cylindrical shape of the input image is taken into account by the convolutional block.

B. *Data collection and training*

The training data was obtained in simulation, using Gazebo [19] and the 3D models provided for the DARPA Subterranean challenge [1]. These 3D models consist in sections of tunnels and caves that can easily be tiled together to form complete tunnel networks. To automatically generate the data, an area has been defined for each tile and a set of points have been defined which represent the exits that can be observed from that area, as shown in Fig. 4.a). To obtain each training sample, the robot is automatically placed in a random position and orientation inside the cited area (Fig. 4.b) and an image is obtained as described at the beginning of this section (Fig. 4.c).

To obtain the corresponding label, the relative position of the exit points to the robot is calculated. Then, for each one, a Gaussian-shaped peak is placed in a 360-long array, centered in the index closest to the corresponding angular position of the exits. In this way we end up with a 360-long array of zeros, except at the indices corresponding to angles where a gallery is present, around which there is a Gaussian-shaped peak (Fig. 4.d).

With this method, a dataset of over 89 thousand training samples was generated, of which $90\,\%$ was used for training.

For the training, we have used the Adam optimizer [20] with a learning rate of 0.0001, and batch size of 256. We trained the network for 16 epochs.

Also, to further improve the generalization capabilities of the network, multiple data augmentation techniques were used; Gaussian noise with variance of up to $2\,\%$ of the

[1] www.subtchallenge.com

maximum distance, cutting out squares in random parts of the image, rotation of the image of up to 5 degrees around the center of the image, horizontally flipping the image (and vector) and vertical shifts of the image of up to 2 pixels.

C. Output post-processing

The output of the neural network (shown in polar coordinates in Fig. 5.b) for the sample situation in Fig. 5.a) is not immediately useful for the purpose of navigation because of noise. For this reason, the output vector of the neural network is processed in order to obtain the angles at which the exits are located. The first step of this process is to apply a max-filter to the output of the neural network. This filter applies a window around each element of the array, and if said element is the maximum inside the window, it keeps its value, otherwise setting it to 0 (Fig. 5.c).

The previous step guarantees that the central values of the peaks in the original vector are present in the filtered vector. However, it is also possible that some spurious values are still present. To isolate only the values that correspond to exits, all values that are under 0.3 times the highest value are also discarded as shown in Fig. 5.d). After this step, only a few values in the vector are different from 0, the indices of which directly translate into the angles where exits are located.

The exits detection system operates at the same frequency as the LiDAR system (10 Hz).

IV. Navigation

Navigation in our method consists of moving among graph nodes (that represent features). The transition between nodes is triggered by a change in the exits detected during the movement of the robot. Given that the features are not defined explicitly (a feature is simply something with a certain number of exits), the transition is triggered by the change of the configuration of the exit themselves. For example, if the robot is moving along a corridor, it will detect only two exits, namely the ones at its front and back. When a T-intersection is reached, two lateral exits will appear and eventually the front one will disappear.

On the other hand, as the robot moves through an intersection, the different galleries that it detects change their relative angle in the robot view. For example, if the robot is navigating along a corridor and encounters a gallery to the right, it will first detect an exit towards the front-right but, as the robot keeps advancing, the exit will progressively move towards the rear-right.

If the system were not to recognise this moving gallery as the same unique gallery from one measurement to the next one, it would trigger a node transition every time the detection angle of the gallery changed, which leads to the need for gallery tracking.

A. Gallery Tracking

To circumvent this problem it is necessary to track the observed exits across time. To this end, we store all tracked exits as a tuple of two elements: the angle at which the gallery is being detected, and a value that summarises the confidence level. The latter is an integer that goes from 0 to 10, and represents how consistent in time a particular exit has been. The confidence level of the tracked exits is updated with every prediction.

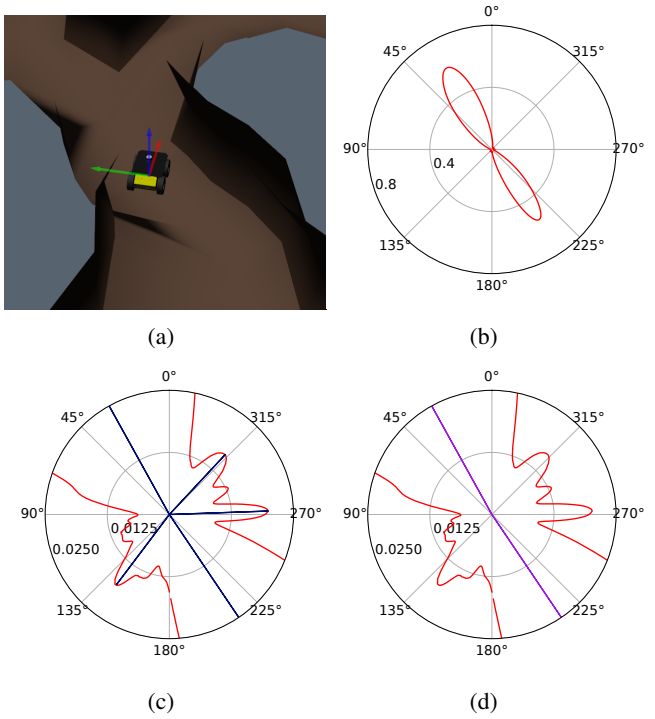


Fig. 5: Post-processing of the neural network output. (a) Situation of the robot in the simulation. (b) Output of the CNN displayed in a polar plot. (c) Initial filtering of the output of the neural network (blue) where some incorrect peaks pass through. (d) Final filtering after discarding the small values from the first filtering (purple).

As detailed in the previous section, the result of the gallery detection system is a list of angles for the candidate exits. When a new list of angles is received, we iteratively match tracked exits to the closest currently observed exit angle. If the closest angle is within 20º of the tracked exit angle, it is interpreted as the exit having moved slightly, but still being observed. If this is the case, the tracked exit is updated by

1) increasing its confidence by 1, and
2) updating its angle to the detected one, which is removed from the list.

If the closest angle is not within 20º of the tracked exit, it is interpreted as the exit no longer being observed, so the tracked exit is updated by

1) decreasing its confidence by 1, and
2) not updating its angle, and so, not deleting the closest angle from the angle list.

After iterating over all currently tracked exits there are two possible scenarios: either the list of angles is empty, or it is not. In the former case, all observed exits have been associated to already tracked exits. Otherwise, one or more

observed exits are not being tracked, and so, a new tracked exit is created for each of the remaining angles, all of them starting with a confidence of 1.

Among all detected exits, two of them are specially important for navigation, that we label *followed exit* and *tail exit*. The *followed exit* dictates the navigation direction of the robot, and is chosen by following the topological instructions. The *tail exit* is used as an anchor in transition periods as will be explained later on.

B. Node transition

After updating the tracked exits, the following scenarios are possible: 1) A tracked gallery reaches the maximum confidence for the first time: This is interpreted as the robot entering a new topological node, either entering an intersection from a corridor, or getting away from a dead-end. When this happens, a node transition is triggered. 2) If a tracked exit that had previously reached the maximum confidence reaches a confidence of 0, then it is considered as if that gallery has been left behind, and so, it means that a new topological node has been reached. This triggers a node transition and the tracked exit is deleted. 3) If a tracked exit that has not reached the maximum confidence at any time reaches a confidence of 0, it is considered an spurious detection, and so it is deleted, but no node transition is triggered. 4) If none of the previous cases apply, it is interpreted as the robot staying in the same node and no node transition is triggered.

If there is no node transition, no further action is required, and the robot keeps going towards the same *followed exit*. If, instead, a transition has been triggered, it means that the robot has entered a new node, and a new topological instruction has to be fetched.

C. Transition period

According to the cases just expounded, there are two causes for a node transition: either a tracked exit has disappeared, or it has reached maximum confidence for the first time. When entering a new node, instances of both causes can take place in a short period of time. For example, when entering a T-intersection from the bottom, the exit in front of the robot disappears, while two exits, one per side, would appear. This would trigger up to three transitions for the same node.

To avoid this problem, we have defined a transition period during which all the triggers after the first one are ignored and the next instruction is not fetched until this period is over, when new and old exists have consolidated.

During this period the robot maintains its followed heading. This guarantees that the robot keeps advancing until it can fetch the next command, and solves the problem of the *followed exit* disappearing (as in the example of the T-intersection).

Once the transition period is over, it is necessary to select the next exit to take according to the corresponding topological instruction. As detailed in section II, these instructions are generated in relation to the previous node, which in our case, is pointed to by the *tail exit*. This makes choosing the next gallery straightforward:

1) Order the N tracked exits by their angle
2) Obtain the index j of the *tail exit*
3) The index of the next gallery to follow is $(j + i + N) \bmod N$.

Once the new *followed exit* is chosen, the robot exits the transition state, and starts heading in the direction pointed to by the chosen corridor. Regarding the *tail exit*, as long as the *tail exit* confidence does not reach 0, it is not changed. However, if it reaches 0 at some point, the tracked exit closest to the opposite angle of the followed corridor is chosen as the new *tail exit*.

D. Velocity Command generation

The output of the gallery detection and path instructions subsystems is, ultimately, an angle in the robot coordinate frame, pointing in the direction to be followed. This angle must be translated into a velocity in order to generate the robot motion. However, the robot must avoid obstacles at the same time it tries to follow the angle provided by the detection system.

Given that we start directly from an angle, we devised a simple reactive navigation approach tailored to this kind of input and the narrow corridors found in our particular scenarios. Other reactive navigation methods based on vector field histograms or dynamic windows could also work after some adaptation.

This method generates velocity commands from the desired movement angle and the laser-scan reading of the horizontal beam of the LiDAR. To do so, a value is computed for every angle, with the highest value determining the immediate direction for the robot to follow.

The value of each angle is obtained by multiplying two weights:

1) The *exit direction weight* gives more weight the closer an angle is to the *followed exit*. To obtain this weight we first generate a 360 long array, whose index corresponds to an angle, and the value at index i is $1 - |J - i|/180$, being J the index corresponding to the angle of the selected exit.
2) The *obstacle avoidance weight* evaluates angles on the closeness to obstacles. Initially, maximum weight is given to angles with obstacles farther than a maximum threshold, or without obstacles (in our case set to 5 meters). Then, a radial filter is applied in which a parameterizable safety distance is considered to both sides of every reading, keeping the minimum weight for the angle at the center of each corresponding sector. We call this final weight the *inflated laser scan*.

To better illustrate this process, an example is given in Fig. 6, where the *followed exit* is the one at the front, but an obstacle is in the way. Fig. 6.b) shows how the direction weight gives preference to the angles around the direction of the *followed exit*. In Fig. 6.c), the effect of the safety distance is shown, as the angles close to obstacle end up

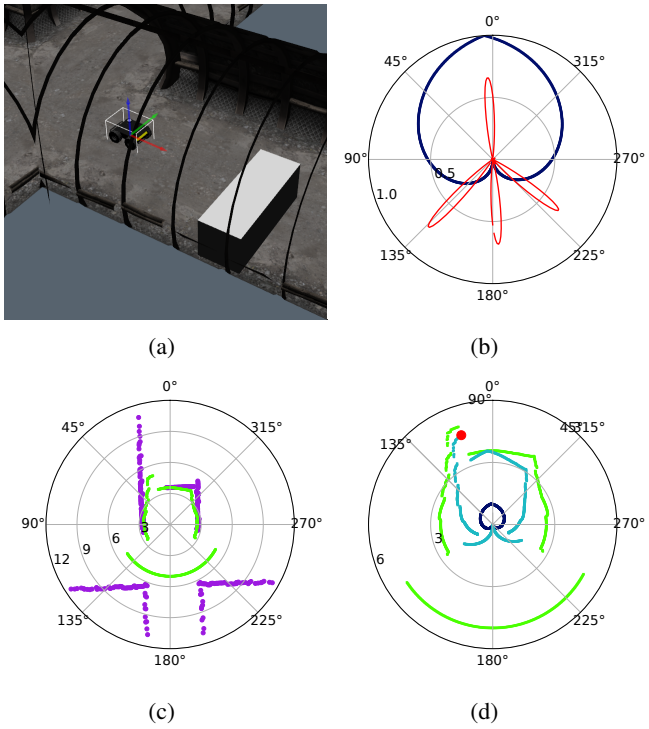


Fig. 6: Heading angle calculation. (a) Situation of the robot in simulation, the red arrow is the x-axis. (b) CNN output (red) and exit direction weight (blue). (c) Laser scan (purple) and inflated laser scan (green). (d) Multiplication of the direction weight (dark blue) by the inflated laser-scan (green) to obtain the total angle value (light blue), the angle with highest value is the selected heading angle (red).

with the same weight as the obstacle itself, ensuring that the highest-value angle is far-enough from the obstacles to avoid collisions. By multiplying both weights we obtain the final value for each angle (Fig. 6.d).

After obtaining the desired heading direction (ϕ), the velocity commands are calculated as follows:

$$\omega = \frac{\phi}{|\phi|} \times \min(|\phi|, \Omega) \tag{1}$$

$$v = \max(V \times (\frac{min(d,2)}{2} - \frac{|\omega|}{\Omega}), 0) \tag{2}$$

Where Ω is the maximum angular velocity and V is the maximum linear velocity of the robot. Finally, d is the minimum distance detected by the laser scan in the $15°$ in front of the robot.

The resulting velocity commands ensure that, if the objective angle is far from the heading angle, the angular velocity is maximal, while the linear velocity is 0. This is important in our case, because it ensures that when the *followed exit* changes in an intersection, the robot will not exit the topological node while turning.

V. Results

To test the complete navigation stack, a custom setup has been used, based in the procedural generation of tunnel networks.

To generate the tunnel networks, first a set of high-level instructions (of the sort "go left", "go right"...) is generated. Based on these instructions, a Gazebo world is put together using the DARPA SubT challenge tiles as building blocks. During world generation the corresponding topological representation of the network is also created.

For every testing run, the robot is placed in the dead-end of a straight section of tunnel, pointing towards the only exit. The corresponding topological node for the initial position of the robot is known. However, the destination node of the robot is decided randomly between all the dead-ends present in the map. This means that the instructions used to generate the world are not necessarily the same as the navigation instructions for the robot; we ensure that the plan requires traversing the same number of intersections, so the difficulty is not lessened. Once the destination node is chosen, the navigation instructions for the robot are generated and the navigation begins. Once the robot has followed all of the instructions, the testing system checks whether the robot has reached the intended objective, and logs the result of this check for posterior analysis.

Using the described testing method, 60 testing runs have been performed in scenarios with 3, 4 and 5 intersections, for a total of 180 testing runs of which 177 have been successful, yielding a completion rate of $98.3\,\%$.

The method has been also tested in scenarios where obstacles were introduced (1 or 2 per tile) obtaining a global success rate of $73\,\%$ over 60 runs despite the fact that the network was not trained taking into account this situation. One of these successful runs is shown in Fig. 7 where the robot has to traverse a complex tunnel network to reach its destination following instructions computed beforehand.

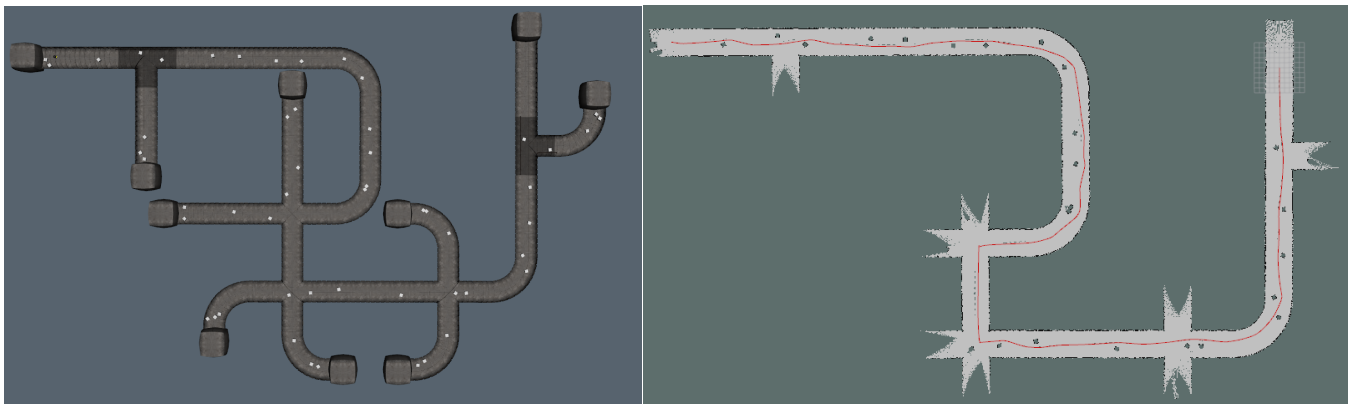
VI. Conclusions

This work presented a method for the navigation of robots in tunnel-like environments, relying on topological information of the kind "skip two crossings and then go left" familiar to humans. The method does not require global geometric localization, as the topological information and a local reactive navigation are enough to follow a given plan.

A convolutional neural network is used to constantly detect available exits even in the presence of obstacles. The post-processed CNN output reliably provides the topological information needed by the plan-following system to work, triggering transitions in the graph that represents the environment, derived from the number of exits at each node. Simulations in randomly generated tunnel mazes show the effectiveness of the proposal, achieving success rates of $98.3\,\%$ in uncluttered scenarios, regardless of the complexity of the maze. Future work will test the system on real-world scenarios and will aim to improve the performance of the method in more cluttered scenarios.

References

[1] D. Tardioli, D. Sicignano, L. Riazuelo, A. Romeo, J. L. Villarroel, and L. Montano, "Robot teams for intervention in confined and structured environments," *Journal of Field Robotics*, vol. 33, pp. 765–801, 2016.



(a) (b)

Fig. 7: Example of a training run. (a) Possible training scenario that has 5 intersections and 2 obstacles per tile. In (b), resulting robot trajectory (red) with origin at top-right and destination at top-left. The topological instructions followed are $[1, 1, 1, 2, 1, 1, -1, -3, -1, -2, -1]$. (b) Occupancy grid generated with gmapping during a testing run [21] using the perfect simulation odometry.

[2] M. T. Ohradzansky, A. B. Mills, E. R. Rush, D. G. Riley, E. W. Frew, and J. Sean Humbert, "Reactive control and metric-topological planning for exploration," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 4073–4079.

[3] D. Tardioli and J. Villarroel, "Odometry-less localization in tunnel-like environments," in *2014 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, 2014, pp. 65–72.

[4] T. G. Phillips, N. Guenther, and P. R. McAree, "When the dust settles: The four behaviors of LiDAR in the presence of fine airborne particulates," *Journal of Field Robotics*, vol. 34, pp. 985–1009, 8 2017.

[5] J. S. Kim, S. K. Kwon, M. Sanchez, and G. C. Cho, "Geological storage of high level nuclear waste," *KSCE Journal of Civil Engineering*, vol. 15, 2011.

[6] C. Gugg and P. O'Leary, "Robust machine vision based displacement analysis for tunnel boring machines," in *Conference Record - IEEE Instrumentation and Measurement Technology Conference*, vol. 2015-July, 7 2015, pp. 875–880.

[7] S. Han, X. Ren, J. Lu, and J. Dong, "An orientation navigation approach based on INS and odometer integration for underground unmanned excavating machine," *IEEE Transactions on Vehicular Technology*, vol. 69, pp. 10 772–10 786, 10 2020.

[8] S. S. Mansouri, C. Kanellakis, G. Georgoulas, and G. Nikolakopoulos, "Towards MAV navigation in underground mine using deep learning," in *2018 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2018, pp. 880–885.

[9] S. S. Mansouri, P. Karvelis, C. Kanellakis, D. Kominiak, and G. Nikolakopoulos, "Vision-based MAV navigation in underground mine using convolutional neural network," in *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1. IEEE, 2019, pp. 750–755.

[10] A. Garcia, S. S. Mittal, E. Kiewra, and K. Ghose, "A convolutional neural network vision system approach to indoor autonomous quadrotor navigation," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2019, pp. 1344–1352.

[11] J. Larsson, M. Broxvall, and A. Saffiotti, "Laser-based corridor detection for reactive navigation," *Industrial Robot*, vol. 35, no. 1, 2008.

[15] M. Saroya, G. Best, and G. A. Hollinger, "Online exploration of tunnel networks leveraging topological cnn-based world predictions," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 6038–6045.

[12] A. Romeo and L. Montano, "Environment understanding: Robust feature extraction from range sensor data," in *2006 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2006, pp. 3337–3343.

[13] M. Mascaró, I. Parra-Tsunekawa, C. Tampier, and J. Ruiz-Del-solar, "Topological navigation and localization in tunnels—application to autonomous load-haul-dump vehicles operating in underground mines," *Applied Sciences (Switzerland)*, vol. 11, 2021.

[14] T. Dang, F. Mascarich, S. Khattak, C. Papachristos, and K. Alexis, "Graph-based path planning for autonomous robotic exploration in subterranean environments," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 3105–3112.

[16] N. Savinov, A. Dosovitskiy, and V. Koltun, "Semi-parametric topological memory for navigation," *arXiv preprint arXiv:1803.00653*, 2018.

[17] D. S. Chaplot, R. Salakhutdinov, A. Gupta, and S. Gupta, "Neural topological SLAM for visual navigation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 12 875–12 884.

[18] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[19] C. Aguero, N. Koenig, I. Chen, H. Boyer, S. Peters, J. Hsu, B. Gerkey, S. Paepcke, J. Rivero, J. Manzo, E. Krotkov, and G. Pratt, "Inside the virtual robotics challenge: Simulating real-time robotic disaster response," *Automation Science and Engineering, IEEE Transactions on*, vol. 12, no. 2, pp. 494–506, April 2015.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2017.

[21] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, 2005, pp. 2432–2437.