2020

Exploration of Unknown Environments Using a Tethered Mobile Robot

Danylo Shapovalov
West Virginia University, ds0144@mix.wvu.edu

2020

Exploration of Unknown Environments Using a Tethered Mobile Robot

Danylo Shapovalov

Exploration of Unknown Environments Using a Tethered Mobile Robot

Danylo Shapovalov

Thesis submitted to the
Benjamin M. Statler College of Engineering and Mineral Resources
at West Virginia University

in partial fulfillment of the requirements for the degree of

Master of Science in
Aerospace Engineering

Guilherme A. S. Pereira, Ph.D., Chair
Yu Gu, Ph.D.
Jason N. Gross, Ph.D.

Department of Mechanical and Aerospace Engineering

Morgantown, West Virginia
2020

Keywords: Exploration, Tether, Mobile Robot, Tangle-free

Abstract

Exploration of Unknown Environments Using a Tethered Mobile Robot

Danylo Shapovalov

Exploration with mobile robots is a well known field of research, but current solutions cannot be directly applied for tethered robots. In some applications, tethers may be very important to provide power or allow communication with the robot. This thesis presents an exploration algorithm that guarantees complete exploration of arbitrary environments within the length constraint of the tether, while keeping the tether tangle-free at all times. While a generalized algorithm that can be used with several exploration strategies is also proposed, the presented implementation uses a modified frontier-based exploration approach, where the robot chooses its next goal in the frontier between explored and unexplored regions of the environment. The main modification from standard frontier-based method is the use of a cost function to sort multiple goals in one iteration and pick the cheapest one to go to, minimizing global path length in the process. The cost is calculated in terms of path length with tether constraints accounted for. The basic idea of the algorithm is to keep an estimate of the tether configuration, including length and homotopy, and decide the next robot path based on the length difference between the current tether length and the shortest tether length at the next goal position. The length difference is then used to determine whether it is safe for the robot to take the shortest path to the goal, or whether the robot has to take a different path to the goal in the way that would put the tether back into the most optimal configuration. The maximum length difference that would still guarantee global tangle-free paths has been shown to be the circumference of the smallest expected obstacle in the environment. The presented algorithm is provable correct and was tested and evaluated using both simulations and real-world experiments. Navigation of a planar robot is done with the aid of a Simultaneous Localization and Mapping (SLAM) system, with the data being provided by the on-board LiDAR scanner. The results from conducted experiments have demonstrated that the proposed algorithm results in the total path length increase of anywhere from 30% up to to 80% compared to untethered frontier-based approach, with the exact percentage increase dependent on the complexity of the environment. It is also approximately 6 times shorter than the total path length in a conservative approach of backtracking to the base to avoid tangling.

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgments

I would like to thank my friends and family for continuously supporting and encouraging me on my educational journey. I would also like to thank my research advisor for helping me get to this point, and all the committee members for their time.

This research was made possible by the NASA Established Program to Stimulate Competitive Research, Grant #80NSSC19M0054, and the Statler College of West Virginia University.

1 Introduction

1.1 Problem Overview And Motivation

Commercial mobile robots are ubiquitous in modern times, ranging from simple autonomous house vacuums to complex networks of warehouse and delivery robots. Examples of such robots are shown in Figure 1.1.1. Despite these robots being extremely efficient at their given tasks, they require almost ideal conditions to operate properly. They have to keep track of their on-board batteries and in most cases also maintain a constant wireless communication with their central hub.

In some scenarios, such limitations are simply not practical. In more extreme environments wireless communication might not be possible. In other scenarios, such as disaster responses in mines or collapsed buildings, prolonged mission time with no interruptions might be desired and on-board batteries might no longer be sufficient. Using a tether on the robot can solve most of these problems, providing both power and a data connection. In fact, tethers are already used for some

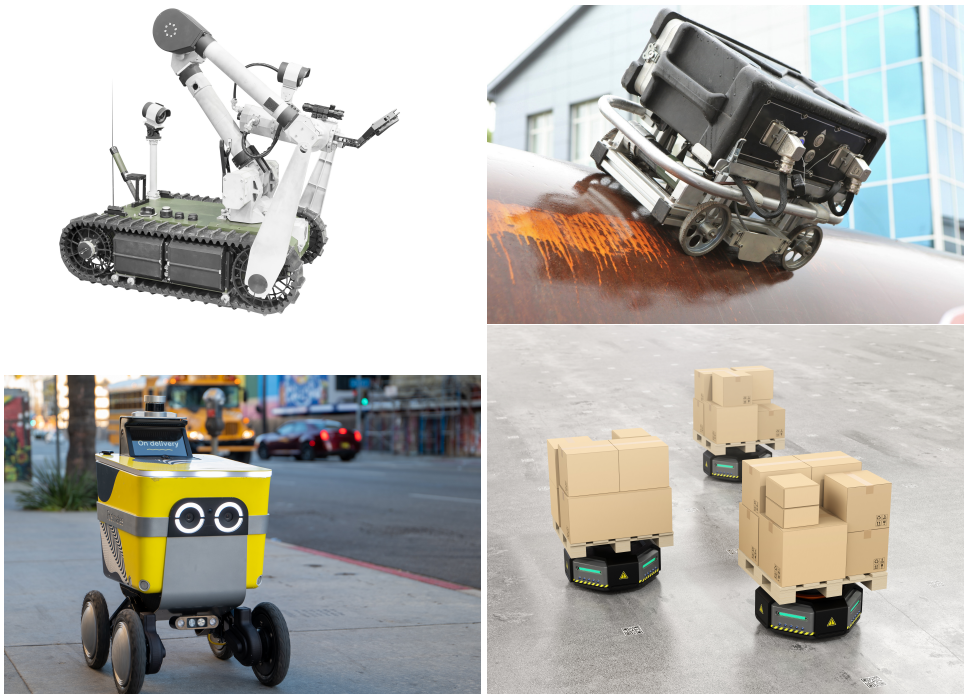


Figure 1.1.1: Examples of modern mobile robots. (Source: shutterstock.com)

(a) (b)

Figure 1.1.2: Examples of (a) underwater tethered robot and (b) land-based tethered robot. (Source (a): shutterstock.com, Source (b): nexxis.com)

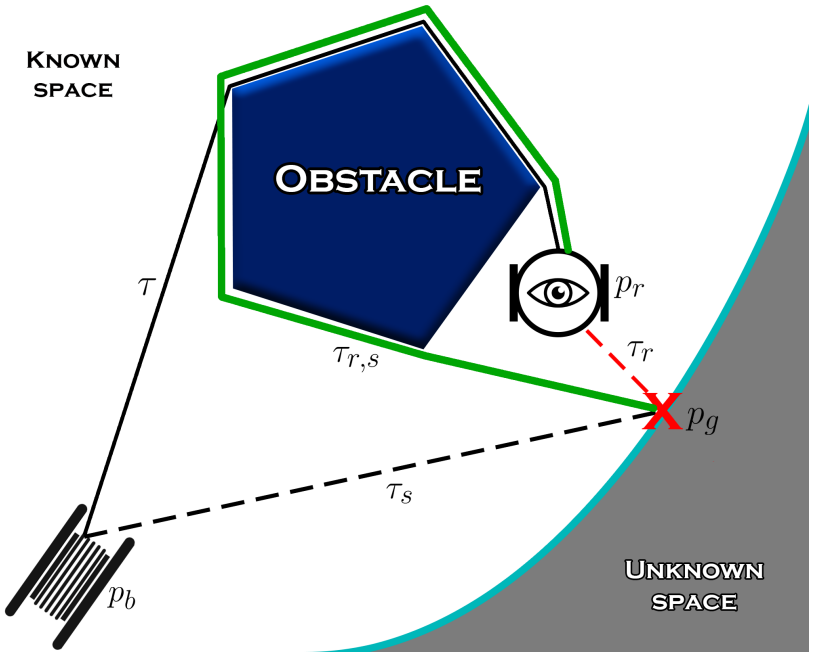


Figure 1.1.3: Visual representation of our proposed approach. The robot at the position p_r with a tether τ connecting it to the base at p_b compares the shortest path to the goal p_g from the base (τ_s) with the one from its current position (τ_r), and takes the path $\tau_{r,s}$ to return the tether to its shortest configuration and avoid further tangling.

underwater robotic missions, and ongoing research is being conducted for land-based platforms as well, with some examples shown in Figure 1.1.2.

However, tethers come with their own drawbacks. An intrinsic limitation of a tethered robot is its operating range, since the robot is only able to go as far as the maximum tether length allows it. Another problem that arises from using a tether on a robot, especially in exploration scenarios with many obstacles in the environment, is having the tether tangle around the obstacles. While there is not much that can be done about the first problem other than getting a longer tether in the first place, it is possible to solve the tangling problem, which is the goal of this thesis.

The approach proposed in this thesis is visually demonstrated in Figure 1.1.3. Assuming a retractable tether and a planar environment, we propose a solution to the tangling problem where the robot constantly keeps track of both its current tether configuration and the theoretical most opti-

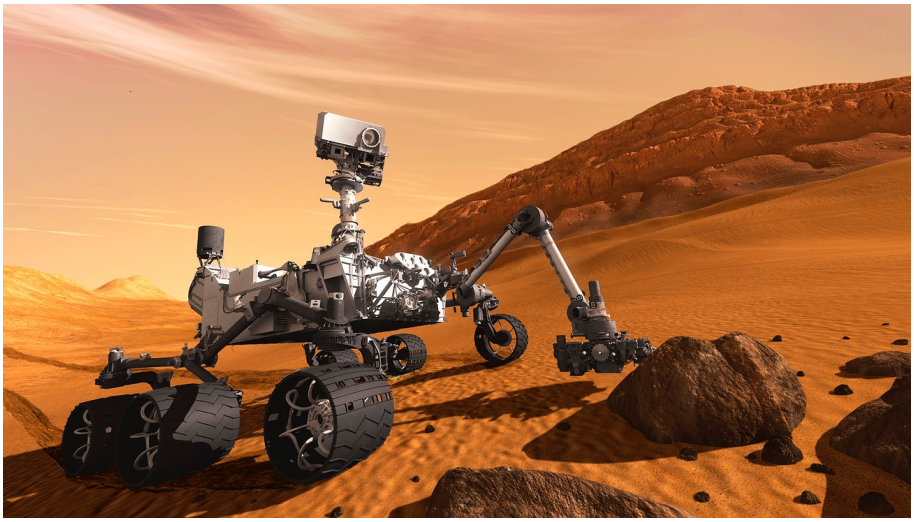


Figure 1.1.4: Artist concept of a NASA Mars rover. (Source: rawpixel.com)

mal configuration. At each step during the exploration, these configurations are being compared, and if the difference becomes too large — the robot returns to the most optimal configuration before there is a chance for the tether to tangle.

The field of robotic exploration is ever evolving, with new solutions being developed regularly. Using autonomous robots for unknown environment exploration is essential in environments where direct human influence is not possible. Such environments can range from something like arctic underwater areas to using autonomous rovers on other planets. Martian rovers are probably the most well-known examples of such, with an artist concept of one shown in Figure 1.1.4.

1.2 Literature Review

Unknown environment exploration is a standard problem in mobile robotics and is fundamental for a lot of applications. Exploration is often performed by robots equipped with sensors that range from simple cameras [25] and sonars to high-end LiDARs. The goal of such exploration platforms is usually to create a map of the environment where the mobile robot is placed into. Assuming that the environment is partially or entirely unknown and the mobile robot platform is autonomous, simultaneous localization and mapping (SLAM) approaches are generally used during the explo-

ration, as continuous and accurate robot localization is essential for building a precise map of the environment [32].

Robotic exploration is typically applied in scenarios where direct human intervention is either impractical or too dangerous. One example of an application in such an environment would be under-ice robotic exploration [2], with another one being fast exploration of underground areas with a team of robots [23]. Fast and efficient underground exploration can also be applied to search-and-rescue operations in environments such as mines [34], potentially saving lives in the process.

There is no single universal method of approaching exploration with mobile robots, with multiple exploration strategies developed over the years. One of the most common methods used for unknown environment is frontier-based exploration [31]. This method is based on the robot continuously selecting and visiting subsequent goals at the borders between known and unknown areas of the environment, with these borders being called frontiers, until the environment is fully mapped and explored. In the end, this approach results in maximizing the available information on the environment. However, other approaches that use available environment information are methods using harmonic functions [6] and potential information fields [30]. An interesting approach that explicitly maximizes the information is [3]. The authors use a laser scanner to construct an Occupancy Grid model [7] of their environment and an extended Kalman filter (EKF) algorithm for their SLAM implementation. The exploration approach presented in their paper was implemented using the frontier-based method for goal selection. However, the general algorithm we are proposing is not limited to one particular approach and any exploration method can be used for goal selection.

Another way to approach exploration is by using a graph-based method [22]. In that particular paper [22], the authors used two robots in collaboration, triangulating their position in the environment based on line-of-sight to each other and their relation to the obstacles around them. Furthermore, more than two robots can potentially be used simultaneously for exploration purposes, with the use of a large number of robots being called swarm robotics. The field of swarm robotics [1] has seen tremendous growth over the recent years, and can be a lot more efficient that single-robot scenarios should the circumstances permit.

Regardless of the exploration strategy used, one of the most important elements for any exploration algorithm is path planning. There are a multitude of path planning algorithms of varying

complexity that have been developed over the years. A commonly used sampling-based path planner is Rapidly-exploring Random Tree (RRT) [17]. RRT works by sampling random points in the environment, trying to connect them to closest prior points and grow a path "tree" that eventually ends up connecting to the goal. An evolution of this algorithm is asymptotically-optimal RRT* [13], which has an extra re-wiring function that optimizes the tree as it is being built. Among others, common grid-based planners include algorithm such as Dijkstra and A* [9], with the latter being used as the shortest path planner in our exploration algorithm.

However, in order to achieve tangle-free exploration when the tether is involved, the robot's path planning algorithm should not only be intelligent enough to account for the tether, but also make decisions that would keep the tether configuration at its shortest possible length for a given configuration. One of the ways to keep track of the tether configuration is using the notion of homotopy. Two tether configurations are said to be within the same homotopy class (i.e. homotopic to each other) if they can be continuously deformed into each other without intersecting an obstacle. It is worth noting that there are several published homotopic path planners. One example is [11], in which the authors have successfully developed a planner that gets a robot to the goal while considering the cable length and its interaction with the obstacles in the environment. Their approach, however, uses only the distance from the initial vertex to distinguish between homotopy classes, which may fail in scenarios where the same distance can represent multiple configurations. This concept was later expanded by the authors of [15]. Instead of using a single metric like the distance, they propose the use of true homotopy invariants, with each possible configuration having its own unique h-signature. A homotopy augmented graph (or h-augmented graph as called by the authors) is then used by the authors to plan tangle-free point-to-point paths.

Some homotopic planners are sampling-based algorithms based on RRT or its asymptotically optimal version, RRT*. Examples of these include H-RRT [10], which partitions the search space in order to constrain the sampling regions, HARRT* [33], which can achieve asymptotic optimality of a generated path within a given homotopy class, and a homotopy-aware kinodynamic planner [24] that proposes an approach that can considers the differential constraints of the robot. Another example of a homotopic path planner is a topology-based multi-heuristic A* [14]. It should be noted that while all the methods mentioned above can successfully generate a path in a given homotopy class, they all rely on the complete knowledge of the environment, which makes their direct use in an exploration scenario without further modifications challenging.

Returning to the topic of exploration, so far all the methods mentioned earlier assumed a freely moving untethered robot. While a tethered robot would have to actively manage the tether, freely moving mobile robots are not without their own drawbacks. The main limiting factor so far for small mobile robots is battery life. Considering that all untethered mobile robots require an on-board power source, energy conservation becomes a big concern for prolonged missions. To address this problem, the authors of [20] have developed an energy-efficient exploration algorithm that reduces the energy consumption of the robot by selecting consequent goals in a way that minimizes repeated coverage of the already mapped area. Such an approach therefore also results in a minimized global path length, a criterion that we will also be aiming to achieve with our approach.

However, there can be multiple advantages for using a tether other than prolonged mission time with no need for on-board power supply. Not only the tether could be used as an anchor for navigating a steep and otherwise inaccessible terrain [18], but it could also serve as a high-speed low-latency data connection to the robot in an environment where wireless communication would be problematic, like underground mines or deep underwater [4]. The main problem created by a tether, especially in environments with a large number of obstacles, is avoiding the tangling of the tether around the obstacles. Presenting a solution to this problem is the goal of this thesis.

The use of tethered robots for space applications has been considered in [12]. The ROSA rover described in the paper uses a 40 meter tether system both as a power supply and a communication link to the main platform. This reduces the weight of the rover that can in turn be used to carry more scientific instruments. The sensors on the tether in combination with a camera on the base platform are also used as a tracking mechanism. The tangling problem was not yet solved in the paper. Another application of a tethered robot for exploration has been extensively discussed in [19]. The author discusses using a tethered robot for exploring extremely steep terrains. Contrary to our solution, during the exploration, the tether gets intentionally wrapped around the obstacles to serve as additional anchor points, which then gets subsequently unwrapped on the robot's return trajectory.

Although we did not find a previous work that solves the tethered exploration problem, tethered coverage, which is a very similar problem, has been solved in [29]. The authors successfully developed an algorithm that uses a tethered robot to fully cover an arbitrary unknown environment, while also keeping the tether in a tangle-free state. This work has later been expanded in [28]. This paper proposes an algorithm for tethered coverage that successfully minimizes the total path length

required to cover the unknown environment. However, because the path required to completely explore the environment is generally far smaller than the path required to cover one, especially with long-range sensors, direct use of these approaches for exploration may not be the best solution.

1.3 Contributions

Considering all the previous work done on tethered robots and their applications, by the authors knowledge, our previous conference paper [26] and journal paper [27] along with this thesis are the first ones that solve the tangle-free exploration problem with tethered robots. The key contributions of these works are as follows:

- Our conference paper [26] proposed the first iteration of our exploration algorithm. The algorithm itself used standard frontier-based exploration method for goal selection and a modified RRT-based planner both for shortest path and homotopic path planning. The algorithm guaranteed tangle-free global paths.
- Simulations and experiments presented in [26] demonstrate the effectiveness of proposed approach, but the real-world experiments used external infrared camera tracking system (VICON) and a simple environment.
- The main exploration algorithm has been generalized and made modular in [27]. The generalized form is not dependent on any specific exploration or path planning method.
- The implementation of the algorithm using a modified frontier-based approach has been presented in [27], along with a new homotopic path optimizer that replaced the previously used RRT-based planner. The algorithm still guaranteed tangle-free global paths, while being a lot more efficient than the previous version.
- The implementation in [27] is fully integrated with SLAM, meaning the robot can now be fully autonomous. This is supported by new experiments using an autonomous robot with SLAM in more complex environments in terms of number and arrangement of obstacles.

1.4 Thesis Outline

The rest of this thesis is divided as follows. Chapter 2 gives a detailed background about homotopy as a whole, then goes over the development history of homotopic path planning used in the exploration algorithm presented in this thesis. Chapter 3 formally defines the problem solved in this thesis, followed by a detailed explanation of both generalized and implemented exploration algorithms proposed. Chapter 4 provides all the collected results, as well as their interpretation. Results from both simulations and real world experiments are discussed. Lastly, Chapter 5 presents a conclusion to this thesis, as well as proposed directions for future research in this area.

2

Homotopic Path Planning

This chapter goes over the development history of the homotopic path planning used in the exploration algorithm proposed in this thesis. The first attempt at homotopy constrained path planning was implemented using RRT planner [17] as the basis. However, there were several problems with this approach, so in the end a homotopic path optimizer has been chosen instead for its simplicity, efficiency and reliability. While using an optimizer instead of a full planner does limit the path planning options, it is not an issue in the tethered exploration scenario.

2.1 Background on Homotopy

Considering the topological nature of the posed path planning problem, it is important to explain the notion of homotopy in that context and describe the way we will be using it. In essence, two paths that start and finish in the same configuration are considered to be homotopic to each other

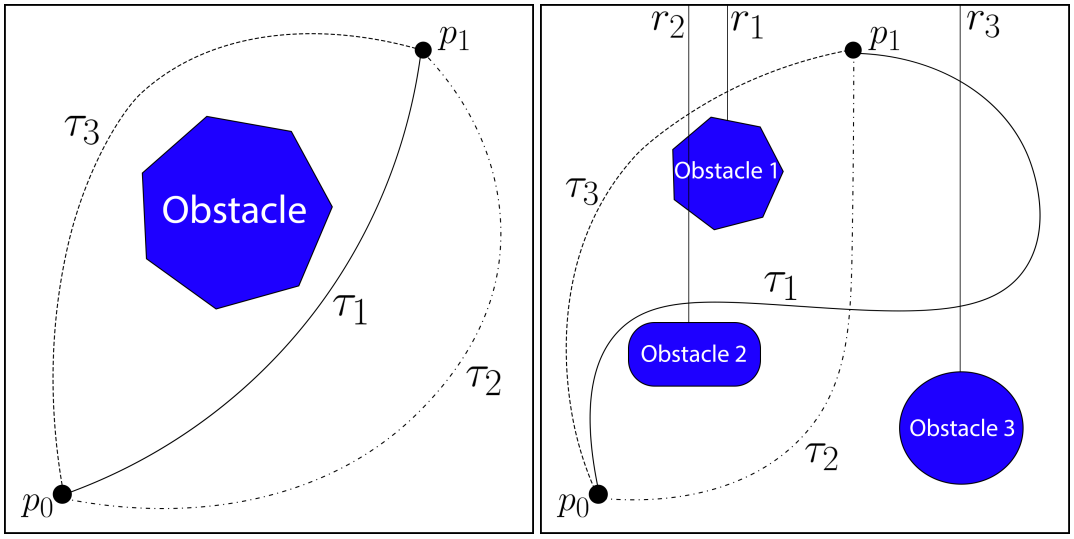


Figure 2.1.1: (a) 3 example paths from p_0 to p_1. Paths τ_1 and τ_2 are homotopic to each other and therefore lie in the same homotopy class, while τ_3 in in a different homotopy class. (b) 3 example paths from p_0 to p_1. h-signature of τ_1 is $r_2 r_3 r_3^{-1}$ which reduces to r_2, h-signature of τ_2 is 0, and h-signature of τ_3 is $r_2 r_1$.

if they can be continuously deformed into each other without crossing any obstacles. Two homotopic paths are said to be in the same homotopy class, which is represented by a unique invariant called h-signature [15]. Examples of homotopic and non-homotopic paths are shown in Figure 2.1.1(a) — paths τ_1 and τ_2 are homotopic to each other and therefore have the same h-signature, i.e. $h(\tau_1) = h(\tau_2)$, where h is a function that computes the homotopy class of a given path. In Figure 2.1.1(a), path τ_1 cannot be deformed into τ_3 without crossing the obstacle, indicating that they are not homotopic and, therefore, do not belong to same homotopy class, i.e. $h(\tau_1) \neq h(\tau_3)$. In the context of path planning for tethered robots, homotopy is an important concept. If, for example, a robot equipped with a retractable tether anchored in p_0 follows the path τ_1 of Figure 2.1.1(a), its tether would assume a shape that is homotopic to τ_1. The taut tether would have the same shape if the robot follows the path τ_2, which is also homotopic to τ_1, as we concluded earlier. Therefore, by constraining the path of the robot to have a specific homotopy class, we can constrain the tether to have the same class. The idea of this thesis is to plan robot paths that make the tether to be homotopic to the shortest possible path from the anchor position to the goal. Since

the shortest path, by definition, does not circulate any obstacles, the tether will also not circulate an obstacle, thus guaranteeing a tangle-free motion.

As proposed in [15], the h-signatures themselves are defined by the rays emanating vertically from the obstacles, and are computed for a path by the order those rays are crossed in. An example is shown in Figure 2.1.1(b). It is important to notice that if a ray is crossed backwards, meaning right to left, a "-1" superscript is added to that particular signature. For example, the h-signature of path τ_3 shown in Figure 2.1.1(b) would be $r_2 r_1$ if the path is taken from p_0 to p_1, but if the same path were to be taken from p_1 to p_0, the h-signature would now be $r_1^{-1} r_2^{-1}$. This is also called inverting the h-signature of a path, so in this example $h(-\tau_1) = h(\tau_1)^{-1} = r_1^{-1} r_2^{-1}$. Another important property of h-signatures is that the same ray crossed back and forth in sequence cancels out. An example of this is path τ_1, whose h-signature would originally be $r_2 r_3 r_3^{-1}$, but because r_3 was crossed back and forth in sequence, its signature cancels out leaving the overall h-signature of τ_1 as just r_2. Such distinction will be important for getting the approximate tether configuration from the total path taken by the robot.

Lastly, it is important to note how it is possible to concatenate h-signatures of two different paths. Suppose we have two paths τ_1 and τ_2, and the end of τ_1 coincides with the start of τ_2. For such paths it is possible to get the overall h-signature $h(\tau_{1,2}) = h(\tau_1) \diamondsuit h(\tau_1)$, with "$\diamondsuit$" being the concatenation operator.

2.2 RRT-Based Homotopic Path Planner

Out of all the path planning algorithms available, we chose RRT as the basis. There are several reasons that influenced our decision. First of all, as discussed in Section 1.2, a lot of work has already been done on homotopic path planning using RRT. Another reason is its simplicity of implementation. Homotopic constraints in our approach were implemented by limiting the tree growth direction, rejecting any branches that would violate the given homotopy constraint. However, that rejection created a different problem that will be discussed in later in Subsection 2.2.2, essentially preventing the planner from finding a path under certain conditions.

Algorithm 1: RRT-based homotopic path planner

input: $p_s, p_g, H, n, \mathcal{C}_f, \mathcal{O}$
$N_{list} \longleftarrow p_s$;
for i = 1,...,n **do**
  $p_{rand} \longleftarrow$ SampleFree($\mathcal{C}_f$);
  $p_{near} \longleftarrow$ Near(p_{rand}, N_{list});
  $p_{new} \longleftarrow$ Steer(p_{rand}, p_{near});
  if CollisionFree($p_{new}, p_{near}, \mathcal{C}_f$) **and** HomotopyCheck($p_{new}$, p_{near}, H, $\mathcal{O}$) **then**
    $p_{parent} \longleftarrow p_{near}$;
    if CrossedRay(p_{new}, p_{parent}, $\mathcal{O}$) = 0 **then**
      $N_{list} \longleftarrow N_{list} \cup \{p_{new}, p_{parent}\}$;
    else
      $N_{list} \longleftarrow \{p_{new}, p_{parent}\}$;
    end
  end
  if CollisionFree(N_{list}[-1], $p_g, \mathcal{C}_f$) **and** HomotopyCheck(N_{list}[-1], p_g, H, $\mathcal{O}$) **then**
    $\tau \longleftarrow$ GeneratePath(N_{list}[-1]);
    if $h(\tau) = H$ **then**
      $\tau \longleftarrow$ OptimizePath($\tau, \mathcal{C}_f, \mathcal{O}$);
      return τ
    end
  end
end
return *None*

2.2.1 Algorithm

The RRT-based homotopic path planner is presented in Algorithm 1. In essence, this algorithm follows a tree growth procedure standard to RRT, but there are two major modifications. The first is the homotopy constraint check in addition to regular collision check, and the second is prior tree cleanup to allow for looping paths that would originally be impossible in a standard RRT planner.

Inputs for this planner require the start and end positions named p_s and p_g respectively, required h-signature of the final path H, maximum allowed number of iterations n, free space $\mathcal{C}_f$, and the obstacle map $\mathcal{O}$. As we are using a grid-based environment, this is not a standard RRT implementation. Specifically, free space $\mathcal{C}_f$ and the obstacle map $\mathcal{O}$ are defined as lists of cell coordinates, with the obstacle coordinates grouped up to represent individual obstacles. The first step in the

algorithm is the initiation of the node list with the starting position p_s in line 1. Lines 2 through 21 are the planner itself, with line 22 returning an empty path should the planner run out of iterations and fail to compute one. Returning to the planner itself, line 3 samples a random node out of the available free space. Notice that because a grid-based environment is used, the sampling is not based on generating a purely random real number coordinate, instead selecting a random free cell in the environment. This allows to omit the collision check for random nodes altogether, further reducing the computation time. Lines 4 and 5 are standard parts of an RRT algorithm, selecting the nearest available node to connect to and subsequently steering towards it respectively.

Lines 6 through 13 are where the main modifications to the tree growth procedure are located. Before adding the new node p_{new} to the node list, line 6 first checks whether the new branch would be both collision-free and not violate the homotopy constraint given the total current path up to p_{new}. Should both conditions be satisfied, line 7 sets the nearest selected node as the parent for p_{new}. Then, line 8 checks if any ray emanating from the obstacles as per the definition of h-signature in [15] was crossed. If no ray was crossed, the algorithm simply adds the new node to the node list in line 9. However, if a ray was crossed, the algorithm resets the node list, only leaving the newly added node in it. Because the node has all the parent history recorded in it up to the starting point p_s, this does not affect the final path generation. It does, however, remove all the other nodes from the selection space, effectively solidifying the path that was generated up to the latest node. This also has an effect of allowing the algorithm to grow a new tree across the area where the old tree was before, effectively allowing the tree growth algorithm to cross the previously generated part of the path should the h-signature require it, which was originally impossible in a pure RRT algorithm.

The last step of the algorithm is the final path generation shown in lines 14 through 20. Line 14 checks if there is a direct path from the last added node to the goal that is both collision-free and doesn't violate the homotopy constraint. Should that succeed — the final path is generated in line 15, and its h-signature is verified in line 16. That latter step is important because even if the final branch connecting the last node to the goal doesn't violate any homotopy constraints on its own, the complete path may still not match the global h-signature required. If it does match however, because RRT doesn't return the shortest path on its own, line 17 first optimizes the generated path before returning it in line 18.

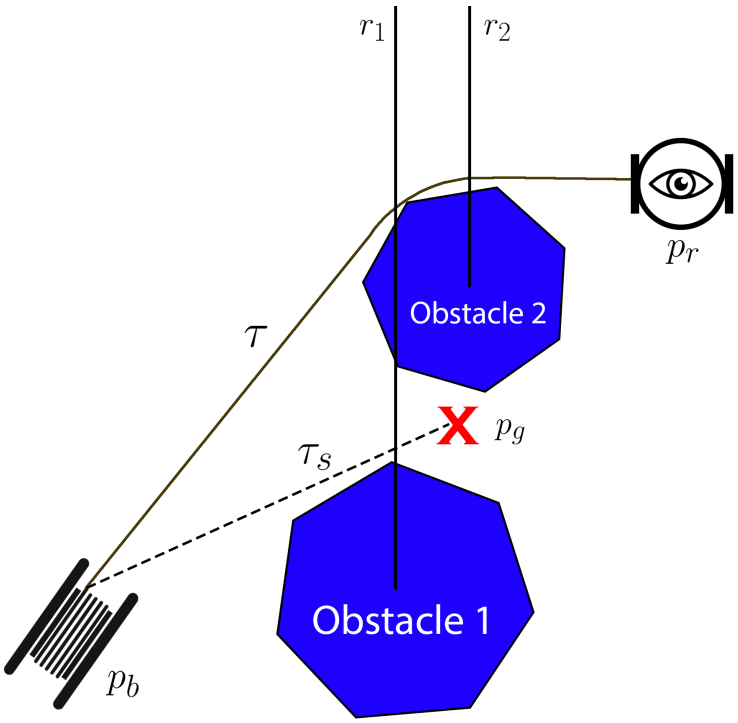


Figure 2.2.1: Visual demonstration of an h-trap. In the case where the robot is asked to go to the goal p_g in the way that would put the tether τ into the shortest configuration τ_s, a required h-signature of such a path would be r_2^{-1} since the ray r_1 is crossed back and forth in sequence.

2.2.2 Encountered Problems

The main problem that we encountered with this RRT-based planner was what we started calling an *h-trap*, which is shown in Figure 2.2.1. Such h-traps are a phenomenon where, with the notion of h-signatures that we're using, a path can end up being impossible to compute due to a ray from one obstacle going through a different obstacle above it when the h-signature requires the robot to loop around the latter. As shown in Figure 2.2.1, in this configuration the path from p_r to p_g that would be required to put the robot back to the shortest tether configuration τ_s would have an h-signature of r_2^{-1} since the ray r_1 would be crossed back and forth in sequence and cancel out. However, because the ray r_1 is technically outside of the h-signature requested, such path would be impossible to compute because the planner rejects branches that violate the homotopy constraint of the path.

As such, the planner would occasionally fail to return the path it was asked for, which required extra mitigation steps in the main algorithm itself. The frequency of h-traps appearing was directly dependent on the complexity of the environment, as more obstacles in the environments naturally meant more chances of h-traps appearing. Therefore, Algorithm 1 in its presented form can not be considered complete, because it can fail to return a path under a given homotopy constraint even if such a path exists.

Other notable problem that stemmed from the planner failing to return a path due to encountered h-traps was performance. While the planner itself has been well optimized for grid-based environments it was meant to be used in, failing to find a path meant that the planner would go through the maximum allowed number of iterations no matter what, increasing its own computation time. On top of that, extra mitigation steps in the main algorithm to counteract the failed planning often involved small physical backtracking of the robot to allow for different planning conditions, resulting in even more delays in the exploration process.

While this problem could be solved by developing a more sophisticated planner, we found that with our particular approach to keep the tether tangle-free, a much more reliable and cost-effective solution would be to simply optimize the data that is already available, namely the total path log and the tether configuration.

2.3 Homotopic Path Optimizer

Assuming that we can get a path in the desired homotopy class using the data already available to the algorithm, namely the total path log and the shortest planned path from the base to the goal, with the next chapter explicitly demonstrating that this is indeed the case, we found that instead of planning a homotopy-constrained path from scratch it would be much simpler to optimize the path data that is already available while keeping the h-signature of the manipulated path unchanged. The optimizer we are using is a simple 2-step process of first shortening the path by removing the extraneous nodes as shown in Algorithm 2, and then subsequently tightening the resulting path to get it closer to the obstacles without colliding, as shown in Algorithm 3. While this approach will not result in an absolutely shortest path possible, the empirically observed difference was not significant. This optimizer is also not very computationally intensive.

Algorithm 2: OptimizeShort algorithm.

```
   input: τ, C_f, O
1  H ⟵ h(τ);
2  i ⟵ 0;
3  j ⟵ EndIndex(τ);
4  while i < EndIndex(τ) do
5      τ_t ⟵ RemoveIntermediateNodes(τ, i, j);
6      if h(τ_t) = H and CollisionFree(τ[i], τ[j], C_f, O) then
7          τ ⟵ τ_t;
8          i ⟵ i + 1;
9      else
10         j ⟵ j − 1;
11     end
12     if j = i then
13         i ⟵ i + 1;
14         j ⟵ EndIndex(τ);
15     end
16 end
17 return τ;
```

Algorithm 3: OptimizeTight algorithm.

```
   input: τ, C_f, O
1  for i in τ[1] to τ[−2] do
2      do
3          τ[i] ⟵ MoveAlongLine(τ[i], τ[i − 1]);
4      while CollisionFree(τ[i + 1], τ[i], C_f, O);
5      do
6          τ[i] ⟵ MoveAlongLine(τ[i], τ[i + 1]);
7      while CollisionFree(τ[i − 1], τ[i], C_f, O);
8  end
9  return τ;
```

Algorithm 2 is a traditional path shortening algorithm [5], with an added modification of also considering the h-signature of the path being optimized. It works by first trying to directly connect the first and last nodes of the input path, and the criteria for a successful connection is for the new path to be both collision-free and have the same h-signature as the original path (line 6). Should that fail, the algorithm then tries to connect the first node to the second-to-last node and so on

until it either succeeds or the ending node comes all the way down to the starting node (line 12), at which point the starting node is moved up by one and the ending node is set to the last node again. This cycle continues until the start node comes all the way up to the end (line 4).

The shorter path optimized in Algorithm 2 is then fed into Algorithm 3 to be further optimized by making that path tighter. This algorithm goes through every node in the path except the first and last ones (line 1). The node that is being currently manipulated is first moved along the line on the path "downstream" from it as far as it can go without resulting in collisions (lines 2 through 4). The same node then undergoes a similar process, now moving it along the line on the path "upstream" from it (lines 5 through 7). This process results in a final path with minimal space wasted between the obstacles.

2.4 Concluding Remarks

In this chapter the concept of homotopy with all the properties of h-signatures has been thoroughly explained, and examples have been provided. Two approaches of generating homotopy constrained paths have been presented, highlighting advantages and drawbacks of both. While the RRT-based homotopic planner is more versatile overall, allowing for any arbitrary start and end points, in its current implementation it ended up too unreliable for the purposes of efficient exploration. Rather than developing a more complex and sophisticated planner, a much simpler solution of using an optimizer was chosen instead. While an optimizer is not as versatile as a proper path planner, for the purposes of tethered exploration it ended up being more than adequate, resulting in a much more efficient and reliable homotopic path acquisition considering the data for the path was already available in the form of total path log and shortest path from the base to the goal.

3

Exploration Algorithm

This chapter presents the main contribution of this thesis, which is an algorithm for exploration of environments using a planar tethered robot. The key parameter in the algorithm, which will be responsible for keeping the tether tangle-free, is the length tolerance ΔL. This parameter is a comparison between the shortest path from the base to the goal and the tether configuration if the robot were to take the shortest path from its current position to the goal. This tolerance is used to determine whether the robot should take the shortest path to the goal or take the path that would put it in the same homotopy configuration as the shortest path from the base.

3.1 Formal Problem Definition

This section defines the problem solved in this thesis. Consider an unknown, planar environment $\mathcal{W} \subset \mathbb{R}^2$ populated with a number of unknown random obstacles of arbitrary shape and size. Exploration, which we consider to be the task of maximizing the information about the environ-

ment to construct a map, is to be done with a planar robot, tethered to the fixed base at position $p_b = (x_b, y_b)$. The robot is expected to start the exploration at the position of the base p_b. We assume that the tether is able to freely rotate around the robot, meaning the rotation of the robot will not cause the tether to be wound up around the robot itself. The tether is assumed to be retractable and always kept in a taut condition, and the maximum tether length $L = L_{max}$ must not be exceeded at any point during the exploration. Our goal is to efficiently cover and map the maximum amount of ground with the footprint of the robot's sensor, which in our case is assumed to be a circle with the radius equal to the field of view of the sensor, while making sure that the tether is kept in a tangle-free configuration at all times. With the tether constraints in mind, it is also possible for the robot to not be able to reach some parts of the environment, in which case the robot should still be able to explore any and all the parts that it can reach.

As briefly mentioned in Chapter 1 and illustrated in Figure 1.1.3, we propose an exploration algorithm that would guarantee that the tether would be kept in a tangle-free state throughout the entire exploration process. This is achieved by keeping track of the tether configuration and not allowing the robot to stray too far from the optimal global tether configuration. However, it is first important to mention how we define tangling, as this definition will be used as the basis for our approach.

Definition 1 (Tangling)**.** *Consider a tethered robot in a planar environment with tether represented by* $\tau(c) \in \mathbb{R}^2, 0 \leq c \leq 1$*, where* $c = 0$ *is the position of the base and* $c = 1$ *the position of the tether's attachment to the robot. If there exist points* c_1 *and* c_2 *along the tether such that* $\tau(c_1) = \tau(c_2)$ *and* $c_1 \neq c_2$*, the tether is considered to be tangled.*

In other words, any taut tether configuration that loops around the obstacles and crosses itself is defined as tangled. However, especially in particularly dense environments, the tether could also get wound up around multiple obstacles in sequence without actually tangling as per Definition 1 but still resulting in a configuration that is very far from the optimum. The algorithm we are proposing eliminates such configurations from the solution space as well.

Algorithm 4: Tether-aware exploration algorithm, general form.

input: $p_b, L_{max}, \Delta L$

1 $\tau \longleftarrow \emptyset$;
2 $p_r \longleftarrow \mathcal{C}_f \longleftarrow \mathcal{O} \longleftarrow$SLAMUPDATE;
3 *Explored* $\longleftarrow$ False;
4 **while not** *Explored* **do**
5 $\quad p_g \longleftarrow$ SELECTGOAL;
6 $\quad$ **if** $p_g = \emptyset$ **then**
7 $\quad\quad p_g \longleftarrow p_b$;
8 $\quad\quad$ *Explored* $\longleftarrow$ True;
9 $\quad$ **end**
10 $\quad \tau_s \longleftarrow$ SHORTESTPATH$(p_b, p_g, \mathcal{C}_f)$;
11 $\quad \tau_r \longleftarrow$ SHORTESTPATH$(p_r, p_g, \mathcal{C}_f)$;
12 $\quad$ **if** $(L(\tau + \tau_r)$ - $L(\tau_s)) > \Delta L$ **or** $L(\tau + \tau_r) > L_{max}$ **then**
13 $\quad\quad h^* \longleftarrow h(\tau)^{-1} \Diamond h(\tau_s)$;
14 $\quad\quad \tau_r \longleftarrow$ SHORTESTHPATH$(p_r, p_g, h^*, \mathcal{C}_f, \mathcal{O})$;
15 $\quad$ **end**
16 $\quad$ FOLLOWPATH(τ_r);
17 $\quad \tau \longleftarrow \tau + \tau_r$;
18 **end**
19 **return** $\mathcal{C}_f, \mathcal{O}$;

3.2 Generalized Form of the Algorithm

The generalized algorithm we are proposing is shown in a flowchart form in Figure 3.2.1. Algorithm 4 presents a more detailed algorithmic version of the flowchart shown in Figure 3.2.1. The algorithm is initialized with the position of the tether anchor (i.e. robot base) p_b, maximum allowed tether length L_{max}, and the length tolerance ΔL. The total path τ is initialized as an empty set before the exploration begins. The position of the robot p_r, the free space $\mathcal{C}_f$ and the obstacle positions $\mathcal{O}$ are expected to be dynamically updated via SLAM in parallel with the main algorithm.

The algorithm itself can be thought of as having different modules that can be replaced with any desired approach. The first such module is goal selection at line 5, which will depend on the exploration approach used in the algorithm. The second, shortest path planning module would then first compute the shortest path from the base to the goal (line 10), and then the shortest path

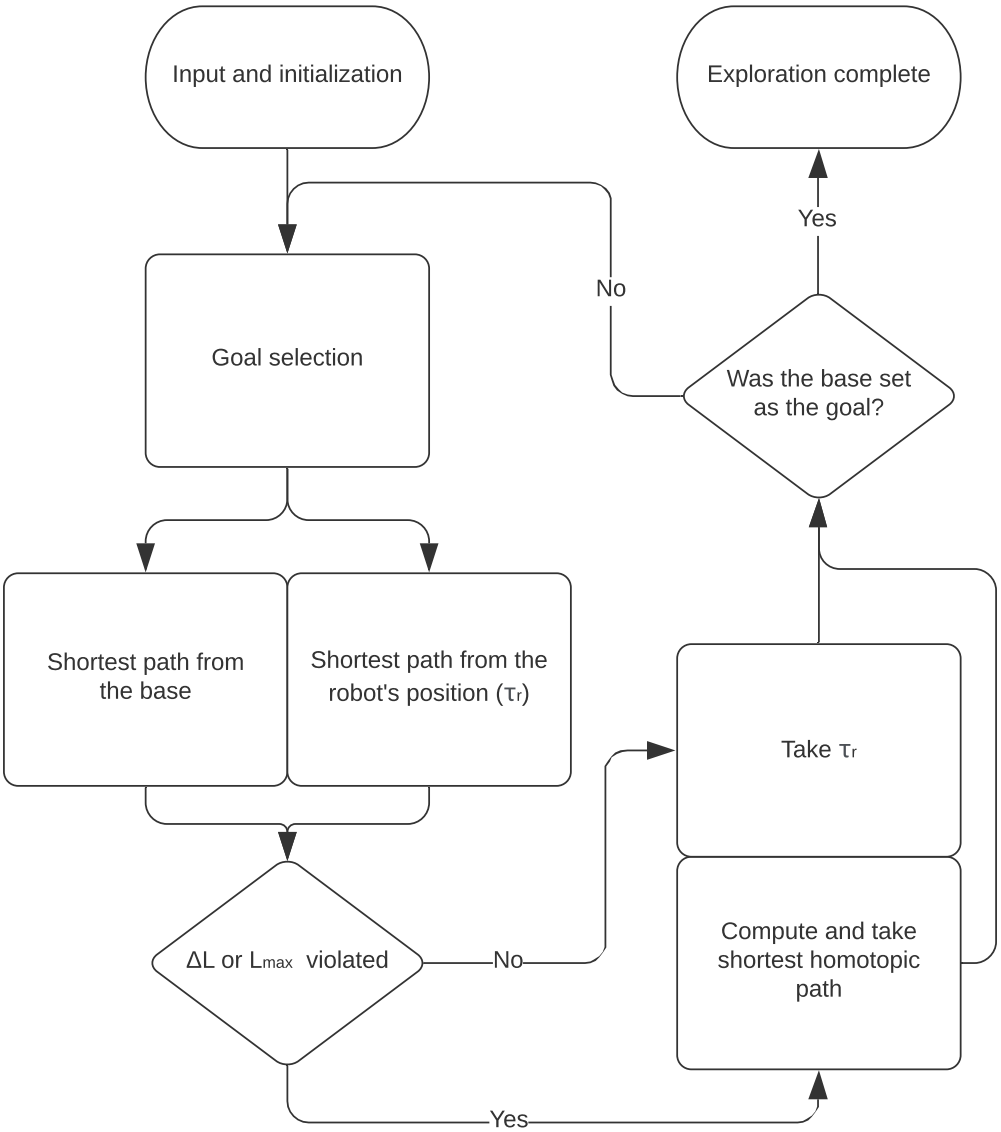


Figure 3.2.1: Flowchart representing the proposed generalized exploration algorithm.

from the robot to the goal (line 11). Line 12 is the main contribution of this algorithm, and it is what's responsible for keeping the tether in a tangle-free configuration. Using the function $L(\cdot)$ to get the estimate of the taut tether configuration in a given path, the algorithm checks both if the difference between the tether configuration if the robot were to take the shortest path τ_r and the optimal configuration in τ_s exceeds the length tolerance ΔL, and if the total path should the robot take the path τ_r exceeds the maximum tether length L_{max}. If any one of those two conditions is violated, in line 13 the algorithm then proceeds to compute the h-signature h^* required to put the robot back in the shortest possible tether configuration. This h-signature would be the same as backtracking to the base and then taking the shortest path τ_s. Notice that the robot will not actually backtrack all the way to the base. Lastly, the third module is the homotopic path planner (line 14) that computes the shortest path from the robot position p_r to the goal p_g satisfying a unique h-signature h^* computed in the previous step. Note that this path replaces τ_r computed in line 11. Finally, line 16 commands the robot to take the computed path τ_r. This path is subsequently added to the total path record τ in line 17.

When the algorithm runs out of available goals to select (line 6), either by completely exploring the environment or by no longer having any goals that can be physically reached with the tether constraint L_{max}, the algorithm concludes the exploration process by selecting the base p_b as the final goal and subsequently returning to it. This process is shown in lines 6 through 9.

3.3 Algorithm Analysis

By properly selecting the length tolerance ΔL the algorithm will guarantee to always keep the tether in a tangle-free configuration by returning to the optimal shortest configuration before the tether has a chance to tangle. However, it is important to discuss a theoretical upper limit on ΔL beyond which a tangle-free global path is no longer fully guaranteed. This upper limit is demonstrated in Theorem 1.

Theorem 1. *Algorithm 4 guarantees tangle-free paths if the length tolerance ΔL is smaller or equal to the circumference of the smallest expected obstacle in the robot's configuration space.*

Proof. Using Definition 1 for tangling and a circular obstacle as an example, the only way a taut tether can cross itself is by looping around the obstacle. The shortest possible way to loop the

tether τ around a circular obstacle so that $\tau(c_1) = \tau(c_2)$ is by encircling the obstacle completely, giving the smallest tangle radius as $2\pi R$, where R is the radius of the smallest obstacle. □

Theorem 1 states that the maximum allowed tolerance value would be $2\pi R$. In the real world, however, the robot would rarely be expected to perfectly encircle an obstacle during the exploration. So depending on the complexity of the environment and density of the obstacles, it is generally safe to pick values for ΔL slightly larger than the theoretical maximum of $2\pi R$. This will be supported by our simulation results in Section 4.1.

3.4 Implementation

It is important to mention that, since Algorithm 4 is generalized, an implementation of that algorithm will not always follow strictly the same structure. The flowchart of our implementation is shown in Figure 3.4.1, with its algorithmic version shown in Algorithm 5. This algorithm uses a modified frontier-based exploration method for the goal selection module and A^* algorithm as the shortest path planner, but the homotopic path planning and goal selection modules are not as clearly separated as they are in Algorithm 4. We are also using a path optimizer instead of a traditional path planner to get the homotopic shortest path, the reasons and algorithm for which were explained previously in Section 2.3.

At its core, Algorithm 5 still uses a similar structure to Algorithm 4, with the exception that the goal selection and path acquisition modules (lines 5-15 in Algorithm 4) are now meshed together. Before the exploration begins, the algorithm initializes the total path array as an empty set in line 1, and requests the robot position p_r, free space $\mathcal{C}_f$, obstacles $\mathcal{O}$ and frontier $\mathcal{F}$ to be updated dynamically in real time through the connected SLAM system. This algorithm uses a frontier-based exploration method as the basis for the goal selection. The main modification in our algorithm is that it samples multiple goal points and then picks the most cost-effective goal to follow, with the cost defined by the length of the path required to reach that goal with the tether constraints accounted for. If the final tether configuration of a proposed path will result in the tether length exceeding the maximum allowed value of L_{max}, the cost for that path is set to ∞, signifying that this particular path would be impossible to take.

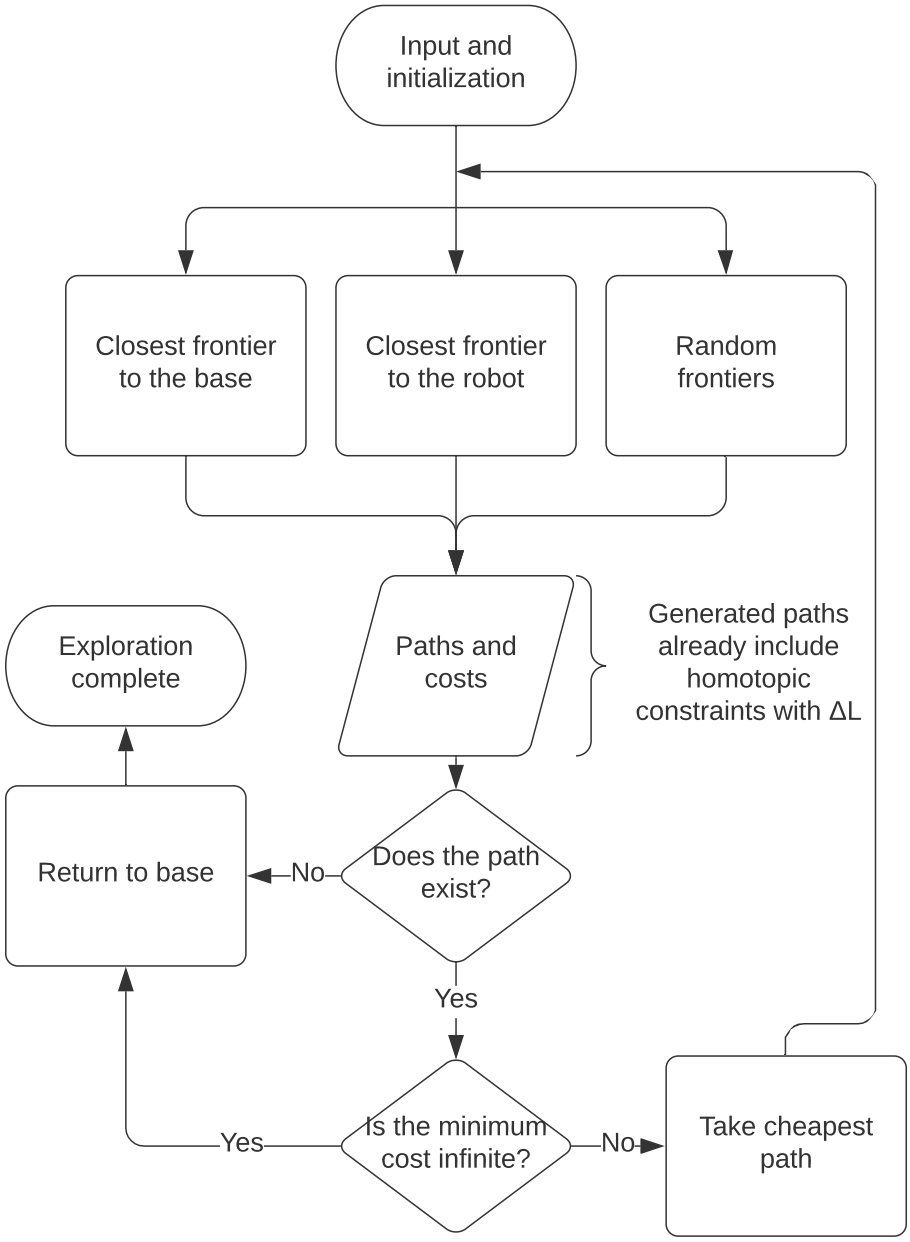


Figure 3.4.1: Flowchart representing our implemented exploration algorithm.

Algorithm 5: Implemented tether-aware exploration algorithm.

input: $p_b, L_{max}, \Delta L, F_{max}$

1 $\tau \longleftarrow \emptyset$;
2 $p_r \longleftarrow \mathcal{C}_f \longleftarrow \mathcal{O} \longleftarrow \mathcal{F} \longleftarrow$ SLAMupdate;
3 *Explored* $\longleftarrow$ False;
4 **while not** *Explored* **do**
5 $\quad \tau_{list} \longleftarrow Cost \longleftarrow \emptyset$;
6 $\quad$ **do**
7 $\quad\quad p_g \longleftarrow$ ClosestFrontier$(p_b, \mathcal{F})$;
8 $\quad\quad [\tau_{list}, Cost] \longleftarrow$ GetPath$(p_b, p_r, p_g, \tau, L_{max}, \Delta L, \mathcal{C}_f, \mathcal{O})$;
9 $\quad$ **while** $Cost[-1] = \infty$ **and** $p_g \neq \emptyset$;
10 $\quad p_g \longleftarrow$ ClosestFrontier$(p_r, \mathcal{F})$;
11 $\quad [\tau_{list}, Cost] \longleftarrow$ GetPath$(p_b, p_r, p_g, \tau, L_{max}, \Delta L, \mathcal{C}_f, \mathcal{O})$;
12 $\quad$ **for** F_{max} **do**
13 $\quad\quad p_g \longleftarrow$ RandomFrontier$(\mathcal{F})$;
14 $\quad\quad [\tau_{list}, Cost] \longleftarrow$ GetPath$(p_b, p_r, p_g, \tau, L_{max}, \Delta L, \mathcal{C}_f, \mathcal{O})$;
15 $\quad$ **end**
16 $\quad$ **if** $p_g = \emptyset$ **or** $\mathit{MIN}(Cost) = \infty$ **then**
17 $\quad\quad p_g \longleftarrow p_b$;
18 $\quad\quad$ *Explored* $\longleftarrow$ True;
19 $\quad\quad \tau_r \longleftarrow$ GetPath$(p_b, p_r, p_g, \tau, L_{max}, \Delta L, \mathcal{C}_f, \mathcal{O})$;
20 $\quad$ **else**
21 $\quad\quad \tau_r \longleftarrow$ CheapestPath$(\tau_{list}, Cost)$;
22 $\quad$ **end**
23 $\quad$ FollowPath(τ_r);
24 $\quad \tau \longleftarrow \tau + \tau_r$;
25 **end**
26 **return** $\mathcal{C}_f, \mathcal{O}$;

At the beginning of each cycle of the main loop, both the list of proposed paths and the list of their associated costs are defined (re-defined in the following cycles) as empty sets in line 5. In lines 6 through 9 the algorithm attempts to find the frontier that is closest to the base that could also be physically reached with the tether length constraint. The loop keeps iterating until it either finds the closest frontier cell that the tether could physically reach or runs out of frontier cells to sample (line 9). The *GetPath* function used to compute both the paths and their respective costs is shown in Algorithm 6 and will be explained later. Lines 10 and 11 execute a similar process, except only the closest frontier cell to the robot is tested. Lastly, lines 12 through 15 pick a number

Algorithm 6: GetPath function.

input: $p_b, p_r, p_g, \tau, L_{max}, \Delta L, \mathcal{C}_f, \mathcal{O}$
$\tau_s \longleftarrow$ ShortestPath$(p_b, p_g, \mathcal{C}_f)$;
$\tau_r \longleftarrow$ ShortestPath$(p_r, p_g, \mathcal{C}_f)$;
if $(L(\tau + \tau_r)$ - $L(\tau_s)) > \Delta L$ **or** $L(\tau + \tau_r) > L_{max}$ **then**
    $\tau_r \longleftarrow \tau^{-1} + \tau_s$;
    $\tau_r \longleftarrow$ OptimizePath$(\tau_r, \mathcal{C}_f, \mathcal{O})$;
end
if $L(\tau + \tau_r) > L_{max}$ **then**
    $Cost \longleftarrow \infty$;
else
    $Cost \longleftarrow$ L(τ_r);
end
return τ_r, $Cost$;

of random frontier cells to test, with the exact number defined by the input variable F_{max}, which in our experiments has been set to 8 random cells, giving a total of 10 cells to sample assuming the cell in lines 6-9 is found immediately. By testing both the closest frontier to the robot and the closest frontier to the base, as well as sampling several cells across the entire frontier, the algorithm ensures that the total path length is kept to a minimum throughout the exploration process.

Overall, lines 5 through 15 can be seen as both the goal selection module and the path generation module of Algorithm 4. Line 16 then checks if the algorithm has no more frontiers to sample or if all the sampled frontiers are impossible to reach, in which case the final goal is selected as the base in line 17 and the path is generated to take the robot there, after which the exploration concludes. Otherwise, in line 21 the algorithm picks the cheapest path that it could take out of all the earlier tested frontiers. Lastly, line 23 commands the robot to take the selected path, and line 24 adds that path to the total path array.

An important point to mention is that, in our implementation, we decided to use simple Euclidean distance to get the closest frontiers instead of a true path length that would be required to get there in order to save on computation time. While it could be argued that using true path length instead of Euclidean distance would result in a more optimal goal selection and therefore shorter total path length required to explore the environment, in our simulations, when we compared the algorithm with Euclidean distance sorting versus the true distance sorting, we did not

see a measurable difference in total path length. This will be demonstrated in Section 4.1.1.

Lastly, the *GetPath* function shown in Algorithm 6 is the actual core of our approach that guarantees global tangle-free paths. As described in Algorithm 4 earlier, lines 1 and 2 of Algorithm 6 generate the shortest paths to the goal from the base and from the robot respectively, which in our implementation is done by the A^* algorithm, which assumes that the environment is represented by a grid. Line 3 here serves the same function as line 12 in Algorithm 4, in that it checks if the tether length should the robot take the path τ_r from line 2 is greater than the most optimal tether configuration from a path acquired in line 1 by more than the length tolerance ΔL, and if that same length should the robot take the path τ_r would be greater than the maximum tether length L_{max}. Should any of these conditions be violated, in lines 4 and 5 the algorithm generates the shortest path that would take the robot to the optimal tether configuration that would have the same homotopy as if the robot would have returned to the base and then took the shortest path τ_s. Note that this is no longer a proper path planner, but a path optimizer, which was explained in detail in Section 2.3. Should the newly generated path τ_r in line 5 still result in the tether length being greater than L_{max}, a condition that is checked in line 7, this would mean that the goal in question is impossible to reach with the imposed tether length limit, and the cost for that path is set to ∞ in line 8. Otherwise, in line 10 the cost is set to the length of the path τ_r computed earlier.

3.5 Concluding Remarks

In this chapter a formal definition of the problem of tangle-free exploration that this work is solving has been presented. The definition of tangling in the way it will be used throughout this study has been formally defined. The main algorithm of the thesis has also been presented. The algorithm has first been given in a generalized form, allowing for any exploration strategy and path planner to be utilized. This is possible due to the defining tangle-free module of the algorithm with the length tolerance ΔL not being dependent on any particular method used for either exploration or path planning. It has also been shown and proven that the maximum theoretical value for ΔL that guarantees tangle-free global paths at all times is equal to $2\pi R$, with R being the radius of the smallest expected obstacle in the environment to be explored.

In practice, since it it not always possible to guess the size of the smallest obstacles expected in the unknown environment, one could always assume a point obstacle to be the smallest and just

take R to be the radius of the robot. However, as will be further discussed in the next chapter, small errors in ΔL that overshoot the theoretical maximum will generally still result in in tangle-free global paths. This is because in the real world, the robot would almost never be expected to take a trajectory that would perfectly encircle the obstacle.

The presented implementation of the algorithm uses a modified cost dependent frontier-based exploration method as the basis for exploration and the A^* planner for shortest path planning. Homotopic path planning is done using a path optimizer instead of a proper path planner, saving on computation times.

4
Results

The approach that we proposed in Algorithm 5 has been implemented using Python programming language and the ROS (Robot Operating System) framework [21]. We used *gmapping* [8] as our SLAM system, and have tested the implementation both in simulations and using a real-world robot.

4.1 Simulations

In order to test our algorithm's both statistical and real-world performance, before implementing it on a real robot, we first implemented the algorithm in a stand-alone Python environment to isolate all the external variables and focus on the algorithm itself. The core algorithm is unchanged in this implementation, but both the robot movement and localization are removed from the calculations and are just supplied to the algorithm in a perfect form. Mapping is also being simulated with a separate function that emulates a sensor reading in a given field of view, with its own com-

putation time excluded from the statistical records. Because both localization and mapping in a full implementation are performed in parallel via SLAM, this approach is still representative of the performance of the algorithm. Gazebo [16] simulations were also performed. These simulations used a full implementation of the algorithm, with both *gmapping* and proper robot movement.

4.1.1 Performance Metrics

To make sure the algorithm gets properly tested in various scenarios, we prepared three randomized grid-based environment types to run the algorithm against, all of which were run with a resolution of 0.5 and the maximum tether length L_{max} set to 40. Typical configurations of these environments as well as example paths taken to explore them are shown in Figure 4.1.1. We do not specify any units in these simulations since for measuring the performance of the system, a 20×20 map with a resolution of 0.5 would be computationally identical to a 40×40 map with a resolution of 1, assuming the field of view and the maximum tether length L_{max} are also scaled appropriately. However, one can assume that all dimensions are in meters, since the same approach is also translated to both the Gazebo simulations and the real-world experiment.

The first "low-density" environment type in Figure 4.1.1 is a 20×20 map with 15-20 random obstacles, each of radius up to 2. The second "high-density" environment type is also a 20×20 map, but now with 50-60 random obstacles, each of radius up to 0.5. Lastly, the third "large" environment type is a 40×40 map with 40-50 random obstacles, each of radius up to 2. The field of view of the simulated sensor was set to 4 for both low and high-density environments, and 10 for large environment. Due to the random nature of this approach, the actual number of discrete obstacles on a given map is generally less than the defined value due to obstacle overlap.

All the environment types mentioned above were simulated with varying length tolerances ΔL to observe the effect it has on tangling rate, computation time, number of iterations it took to explore a given environment, maximum reached tether length, and total path length required to explore the environment. In addition to that, two extra simulations were performed as control samples — one that avoided tangling by simply backtracking the robot to the base after every iteration before selecting the next goal, and one with the ΔL and tether length constraints removed to represent the standard frontier-based approach. The simulations were performed 50 times per value of ΔL to get a representation of average performance that should be expected from a given config-

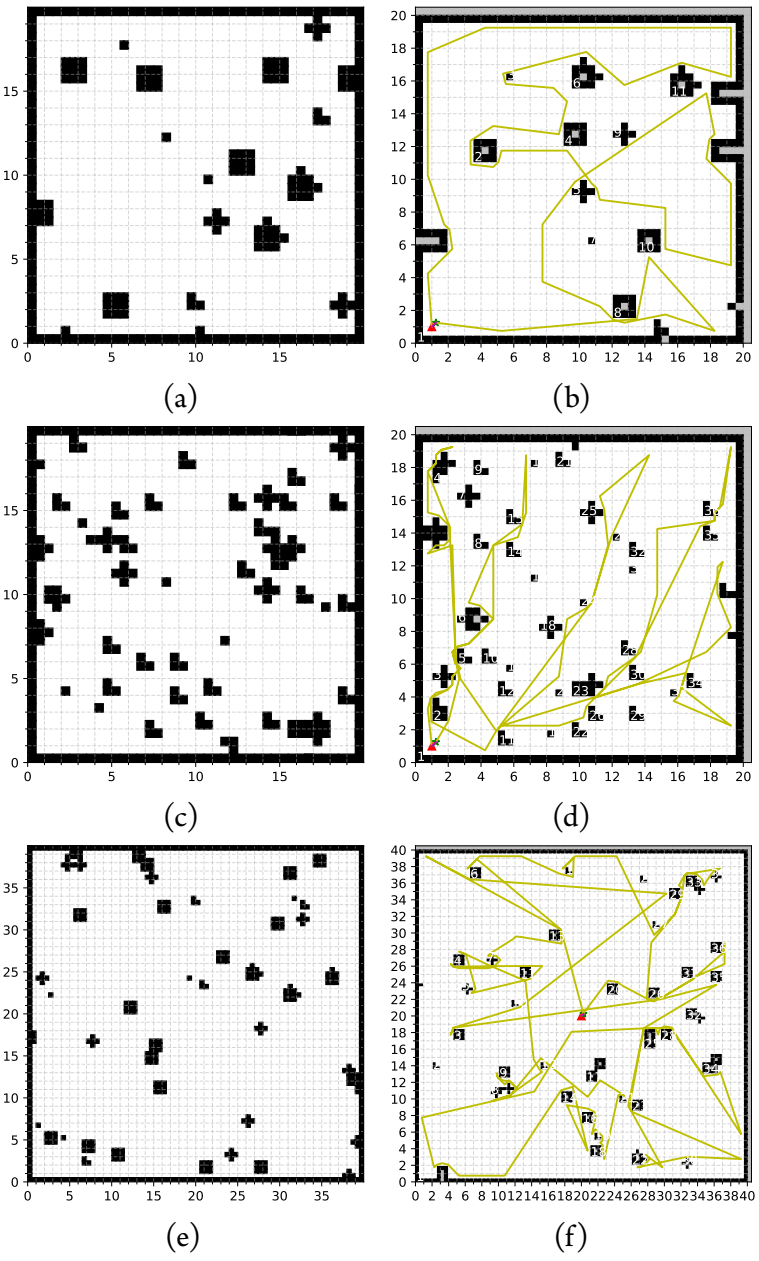


Figure 4.1.1: Typical randomized environments used in simulations. Shown are typical configurations for low-density environment (a), high-density environment (c) and large environment (e), as well as example paths taken to explore them, shown in (b), (d) and (f) respectively.

Table 4.1.1: Simulation data for low density environment. 20×20 map, 0.5 resolution, tether length 40, 15-20 obstacles of radius up to 2. Tether base is at $[1, 1]$, and the field of view is set to 4.

Tolerance ΔL	**Tangle (%)**	**Time per iteration (s)**	**Total time (s)**
Backtrack	0	0.063 ± 0.005	2.611 ± 0.323
0	0	1.000 ± 0.184	37.727 ± 7.629
2πR	0	0.979 ± 0.123	36.335 ± 5.545
4πR	0	1.051 ± 0.153	39.174 ± 6.963
8πR	14	1.120 ± 0.143	42.239 ± 6.951
16πR	28	1.264 ± 0.220	48.757 ± 9.515
∞	44	1.339 ± 0.366	53.098 ± 12.331
∞ $(L_{max} = \infty)$	88	0.270 ± 0.089	10.554 ± 3.745

Tolerance ΔL	**Total iterations**	**Max tether length**	**Total path length**
Backtrack	41.5 ± 2.5	24.129 ± 0.912	1120.101 ± 78.904
0	37.0 ± 2.6	27.742 ± 5.800	187.544 ± 39.293
2πR	37.0 ± 2.8	29.937 ± 4.696	169.000 ± 29.433
4πR	37.2 ± 3.0	32.431 ± 4.264	165.251 ± 19.336
8πR	37.6 ± 2.2	36.147 ± 3.192	170.364 ± 23.007
16πR	38.5 ± 2.1	39.014 ± 0.940	173.233 ± 21.557
∞	39.2 ± 2.8	39.476 ± 0.685	171.970 ± 26.533
∞ $(L_{max} = \infty)$	38.8 ± 2.8	78.129 ± 18.817	126.714 ± 14.293

uration. All of this was run on an AMD Ryzen 9 3950x CPU on a machine running Windows 10. The CPU's reported clock speed during the simulations was 4.2 GHz.

The results from simulations of the low-density environment type are shown in Table 4.1.1. There are several important trends in this table that are worth pointing out. First of all, as we have proved, the length tolerance $\Delta L = 2\pi R$ did indeed result in a tangle-free global path every time. However, since this environment type was not densely populated with obstacles, the value of

$\Delta L = 4\pi R$ also resulted in tangle-free global paths, supporting our claim that depending on the environment type, values of ΔL slightly larger than $2\pi R$ will generally be fine. Beyond that, however, we start to see progressively more tangling the larger ΔL becomes. As expected, the largest tangle percentage was acquired for a simulation type where restrictions on the tether length were removed as well. The shortest maximum tether reached is, as expected, achieved with simple backtracking, and the longest one is where the tether limit was removed. For the simulations with varying ΔL we see an increasing trend, with the maximum tether length leveling off to L_{max} as $\Delta L \longrightarrow \infty$. While the average amount of iterations required to complete one map are all within the margin of error of each other for every simulation parameter with a slight increasing trend (with the exception of "Backtrack" simulation), computation time per iteration is measurably increasing with ΔL, and the total time also increases as a result. This can be explained by the robot straying progressively further from the optimal tether configuration with larger values of ΔL, which in turn requires the path optimizer to work with more complex h-signatures, thus increasing the overall computation time. Lastly, there are several interesting points regarding the total path length taken by the robot. The shortest one is, as expected, the one where both ΔL and the tether length were ignored, and the longest one is when the robot used simple backtracking, resulting in the path length almost 10 times the former. As for the simulations with varying ΔL, we can see that the longest path was for $\Delta L = 0$, with the minimum being achieved somewhere around $\Delta L = 4\pi R$. The value of $\Delta L = 0$ would force the algorithm to always return to the shortest tether configuration, and while this does guarantee a tangle-free global path, this approach is not the most efficient, hence the increased total path length. On the other hand, even if we ignore tangling, increasing ΔL too much would allow the robot to stray too far from the optimal configuration, which in turn would require a longer return path to get back to the optimum, resulting in a longer global path again. The minimum total path length is reached in between these two extremes.

It should be noted that the tangle detection function is only able to return a positive detection based on Definition 1 of tangling. This is why even with $\Delta L = \infty$ the tangling percentage is not nearing 100%, even though the tether may be wound up around multiple obstacles and very far from the optimal configuration. This will be important to keep in mind for the high-density environment results. However, if the robot still fully retracts its tether by the time it returns to the

Table 4.1.2: Simulation data for high density environment. 20×20 map, 0.5 resolution, tether length 40, 50-60 obstacles of radius up to 0.5. Tether base is at $[1, 1]$, and the field of view is set to 4.

Tolerance ΔL	**Tangle (%)**	**Time per iteration (s)**	**Total time (s)**
Backtrack	0	0.076 ± 0.051	3.558 ± 0.383
0	0	2.256 ± 0.470	100.280 ± 24.641
2πR	0	2.440 ± 0.453	107.576 ± 22.351
4πR	6	2.625 ± 0.733	120.867 ± 39.459
8πR	12	2.974 ± 0.607	139.431 ± 31.309
16πR	32	3.174 ± 0.979	154.725 ± 51.098
∞	54	3.241 ± 1.054	153.764 ± 53.731
∞ ($L_{max} = \infty$)	98	1.484 ± 0.554	70.894 ± 29.271

Tolerance ΔL	**Total iterations**	**Max tether length**	**Total path length**
Backtrack	46.5 ± 3.0	24.673 ± 0.754	1269.056 ± 92.467
0	44.2 ± 3.2	26.479 ± 3.282	292.230 ± 42.446
2πR	43.9 ± 2.3	26.721 ± 1.481	236.458 ± 25.719
4πR	45.6 ± 3.1	30.550 ± 2.292	225.613 ± 27.852
8πR	46.7 ± 3.1	35.543 ± 2.435	223.947 ± 21.973
16πR	46.9 ± 3.0	39.178 ± 0.689	215.708 ± 19.908
∞	47.1 ± 3.0	39.200 ± 0.722	218.256 ± 23.976
∞ ($L_{max} = \infty$)	47.2 ± 3.3	102.364 ± 13.341	145.629 ± 13.175

base, such cases do not pose a problem.

Results for a high-density environment demonstrated in Table 4.1.2 show a similar picture, but there are a few key differences. First of all, with the environment being a lot more tightly packed with obstacles, tangle percent for $\Delta L = 4\pi R$ is no longer zero, while $\Delta L = 2\pi R$ still guarantees a tangle-free global path, with a slightly more aggressive tangling rate increase overall. The increasing trend in total iteration number and computation times is now more pronounced, and the computation time itself is noticeably larger due to the more complex h-signatures the algorithm has to

Table 4.1.3: Simulation data for large environment. 40×40 map, 0.5 resolution, tether length 40, 40-50 obstacles of radius up to 2. Tether base is at $[20, 20]$, and the field of view is set to 10.

Tolerance ΔL	**Tangle (%)**	**Time per iteration (s)**	**Total time (s)**
Backtrack	0	0.526 ± 0.031	34.285 ± 4.214
0	0	5.564 ± 0.912	357.424 ± 74.419
2πR	0	5.573 ± 0.697	360.587 ± 59.524
4πR	2	5.942 ± 1.056	393.569 ± 86.561
8πR	20	6.177 ± 0.924	417.102 ± 75.505
16πR	32	6.466 ± 0.862	441.738 ± 64.365
∞	48	6.618 ± 0.869	454.108 ± 68.735
∞ $(L_{max} = \infty)$	96	5.091 ± 1.404	375.764 ± 108.251

Tolerance ΔL	**Total iterations**	**Max tether length**	**Total path length**
Backtrack	65.0 ± 5.1	26.661 ± 0.980	2032.611 ± 173.774
0	63.9 ± 5.1	27.401 ± 1.335	667.252 ± 71.126
2πR	64.5 ± 4.3	28.351 ± 1.771	533.391 ± 53.077
4πR	65.9 ± 4.9	30.658 ± 1.449	516.228 ± 71.644
8πR	67.3 ± 4.0	35.225 ± 1.967	481.901 ± 42.814
16πR	68.3 ± 4.4	39.385 ± 0.522	489.221 ± 38.029
∞	68.5 ± 3.4	39.534 ± 0.384	498.749 ± 43.545
∞ $(L_{max} = \infty)$	73.6 ± 3.9	218.189 ± 23.007	285.281 ± 19.157

deal with. Maximum reached tether length shows a very similar behavior to Table 4.1.1, and with the exception of a generally longer paths, so does the total path length. One difference in total path length is that the minimum is now reached around the $\Delta L = 8\pi R$ mark.

Lastly, the simulation results for a large environment shown in Table 4.1.3 continue to support the general trends established in Tables 4.1.1 and 4.1.2. Total number of iterations and computation times continue to show an increasing trend with ΔL, though the computation times them-

Table 4.1.4: Results using true distance to frontiers at $\Delta L = 2\pi R$.

Environment type	Tangle (%)	Time per iteration (s)	Total time (s)
Low density	0	4.095 ± 0.751	157.243 ± 33.027
High density	0	6.937 ± 1.279	315.000 ± 67.851

Environment type	Total iterations	Max tether length	Total path length
Low density	38.3 ± 2.8	30.322 ± 4.843	171.551 ± 20.323
High density	45.2 ± 2.9	27.432 ± 2.320	239.675 ± 26.978

selves are now also much larger because in addition to more complex h-signatures, A^* now has a larger computation area as well. The maximum tether length shows a very similar trend to both Tables 4.1.1 and 4.1.2, and the minimum for total path length has been reached around the $\Delta L = 8\pi R$ mark similar to Table 4.1.2. While in this particular environment type the computation times per iteration became relatively large, it should be noted that even these times would still be much less than the time it would take the robot to move.

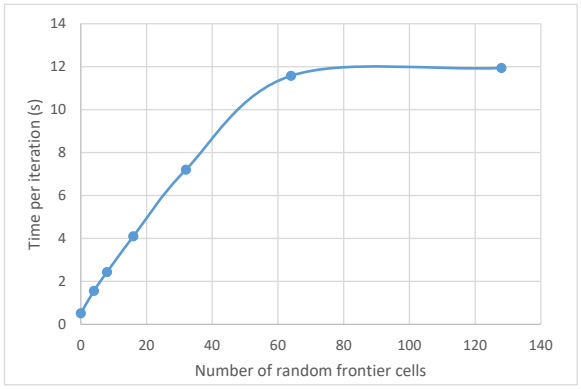
As we have mentioned earlier, in order to save on computation time for all our simulations and experiments we used simple Euclidean distance to sort out closest frontiers instead of getting a proper shortest path length it would take to actually reach them. While this could have hurt the performance in terms of total path length since the closest Euclidean frontier is not necessarily the closest reachable one, our testing did not reveal such a correlation. Table 4.1.4 shows both low and high-density environments at $\Delta L = 2\pi R$ tested again, but this time using proper path distance to frontiers to sort the closest ones instead of simple Euclidean distance. Comparing this data to the corresponding lines in Tables 4.1.1 and 4.1.2 at $\Delta L = 2\pi R$, it is evident that there was no measurable change to the total path length, while the computation time rose significantly. Hence we decided to keep using the Euclidean distance-based sorting method.

Table 4.1.5: Results using varying numbers of random frontier cells at $\Delta L = 2\pi R$ in a high-density environment.

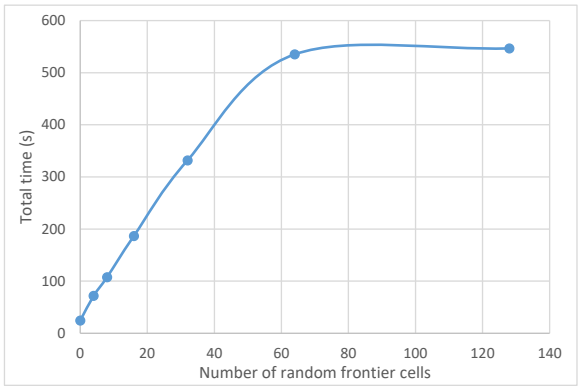
Cells sampled	Time per iteration (s)	Total time (s)	Total iterations
0	0.515 ± 0.128	24.174 ± 7.017	46.6 ± 2.9
4	1.562 ± 0.534	71.740 ± 25.452	45.7 ± 2.8
8	2.440 ± 0.453	107.576 ± 22.351	43.9 ± 2.3
16	4.099 ± 0.841	186.638 ± 43.164	45.4 ± 2.5
32	7.206 ± 1.592	331.824 ± 84.986	45.8 ± 2.9
64	11.576 ± 2.896	535.286 ± 147.438	46.0 ± 2.7
128	11.937 ± 3.171	546.617 ± 159.215	45.5 ± 3.1

Cells sampled	Max tether length	Total path length
0	27.295 ± 2.540	284.044 ± 32.336
4	27.067 ± 1.797	249.882 ± 30.136
8	26.721 ± 1.481	236.458 ± 25.719
16	28.081 ± 2.797	228.557 ± 26.656
32	27.682 ± 2.088	227.961 ± 24.718
64	28.042 ± 2.215	227.405 ± 28.321
128	27.969 ± 2.637	226.805 ± 27.268

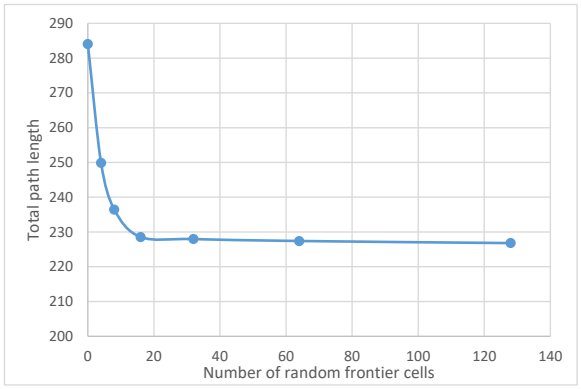
The last important point to our experiment is our use of the number of random frontier cells to sample. While in theory in order to get the absolute best frontier to go to in terms of cost one would need to check all the available frontiers in the map, in practice such an approach would be very computationally inefficient. Another source of inefficiency is that a lot of frontier cells will be right next to each other and there would be little reason to test cells in such close proximity. We have ran an additional test in a high-density environment at $\Delta L = 2\pi R$, varying only the number of random frontier cells sampled. The results of this test are shown in Table 4.1.5. The number of cells sampled goes from 0, meaning no randomness and that only closest cells to the base and to the robot are tested, to 128, which in this testing environment is always more than the total number of frontier cells at any given point. It should also be mentioned that the selection algorithm does not select repeated cells, and should F_{max} exceed the total number of cells it stops once all the cells



(a)



(b)



(c)

Figure 4.1.2: Illustration of the data from Table 4.1.5. The effect of the number of random frontier cells on time per iteration (a), total computation time (b) and total path length (c).

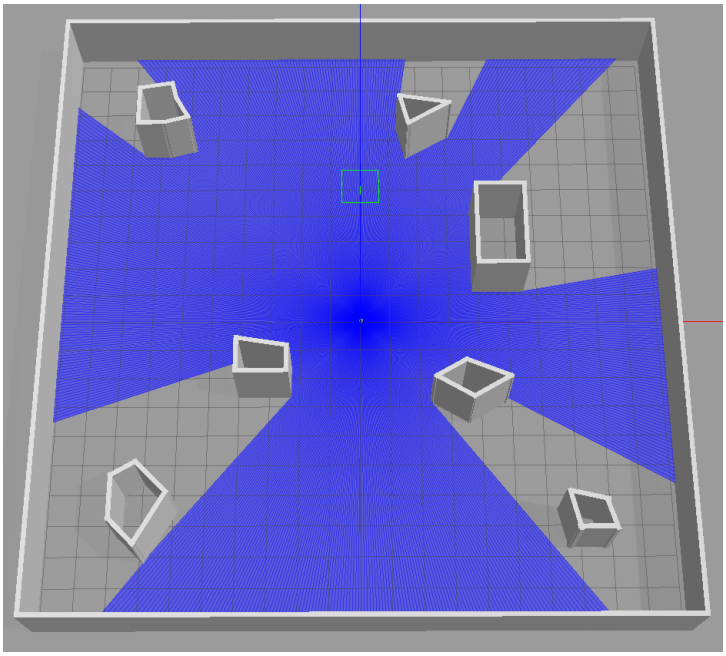
have been sampled.

It is evident from Table 4.1.5 that the number of random cells does not affect neither the total iteration number nor the maximum tether length in a measurable way. The other parameters are affected significantly, and the effect is visually demonstrated in Figure 4.1.2. We can see that both the time per iteration as well as total computation time increase linearly with the number of frontier cells sampled, and saturates at around 70 cells, meaning that in the testing environment this was approximately the maximum number of frontier cells throughout the simulation. This means that computational penalty would be directly proportional to the number of cells sampled. So, by brute-force sampling all the frontier cells in a large high-resolution environment these computation times can easily exceed any reasonable boundaries.

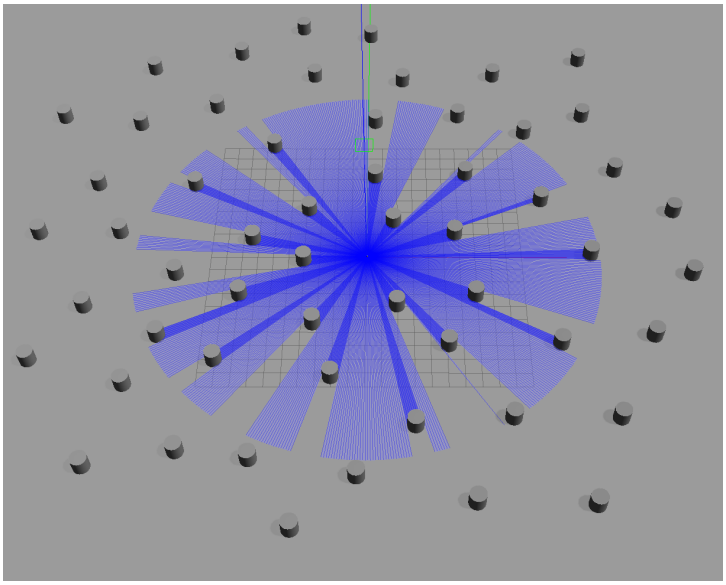
Looking at Figure 4.1.2(c) however, it is clear that such an approach is not necessary. While having no randomness in the algorithm and relying only on the two cells closest to the base and the robot results in a very long total path compared to the rest, the total path length saturates very quickly with the number of random frontier cells, and at 16 cells it has already completely leveled off. Looking back at computation times though, at 8 random cells the algorithm is almost twice as fast as the one at 16 cells, while the total path length is not much longer. We considered this a good balance and hence went with $F_{max} = 8$ for our experiments.

4.1.2 Gazebo Simulations

In order to further test the proposed algorithm in a more realistic scenario before fully implementing it on a real robot, we ran two simulations in the Gazebo simulator. Specifically, we simulated a smaller 20×20 meters "room" type environment as well as a bigger unlimited "forest" type environment. Both environments are shown in Figure 4.1.3(a) and (b), respectively. A model of the iRobot Create mobile robot equipped with a 360-degree planar LiDAR was used in these simulations. While the "room" environment was a more simplistic test to assess the operation of the algorithm itself, the "forest" environment was meant to demonstrate how the algorithm behaves in a more dense field where the tether length is the only constraint on the exploration distance. In both cases maximum tether length L_{max} was set to 15 meters and the length tolerance ΔL was set to 4 meters, with the latter being just over the $2\pi R$ theoretical maximum for the "forest" environ-



(a)



(b)

Figure 4.1.3: Environments used in Gazebo simulations. "Room" type environment is shown in (a), and "forest" type environment is shown in (b).

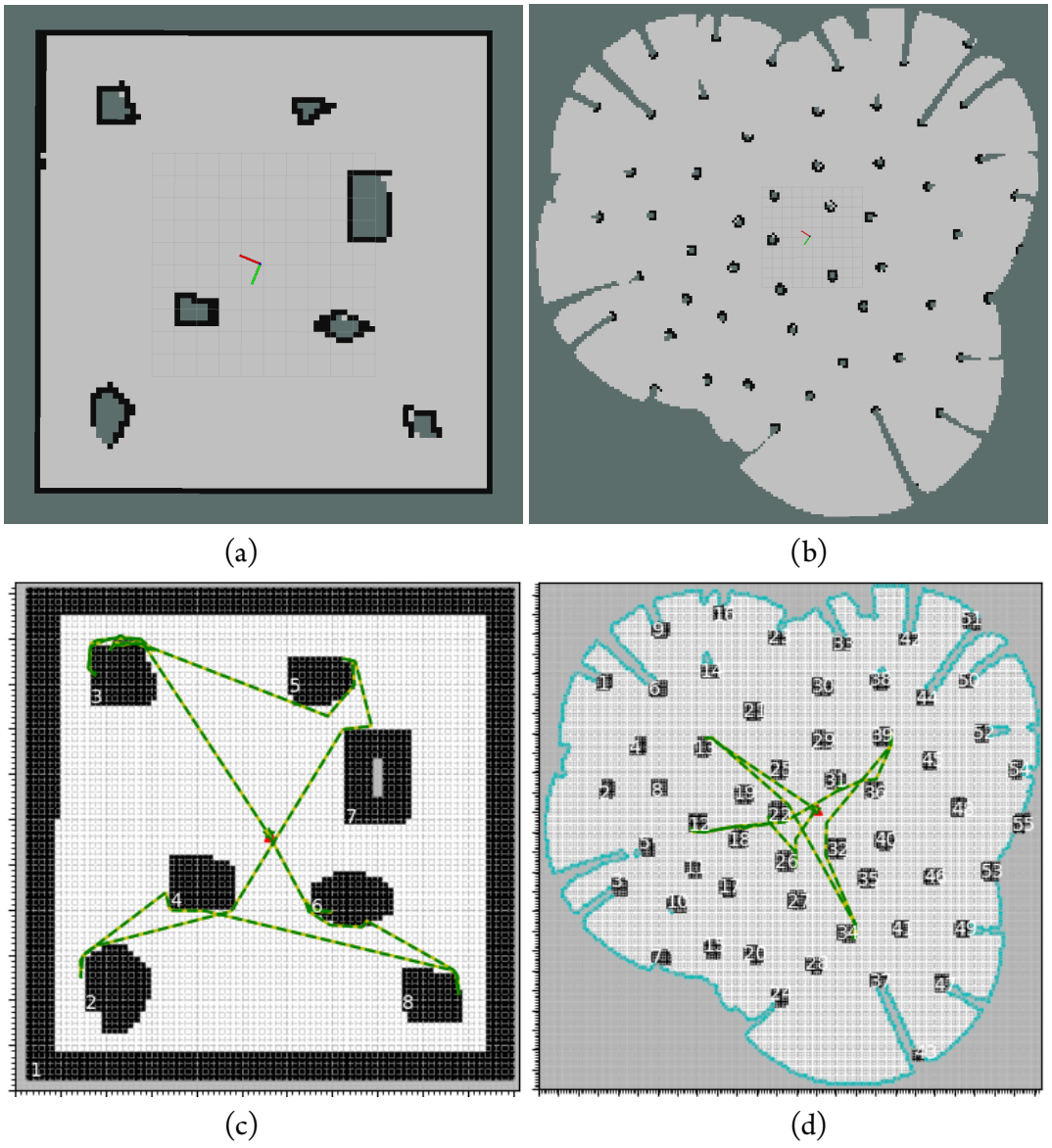


Figure 4.1.4: Results from Gazebo simulations. Maps generated my gmapping for room and forest environments are shown in (a) and (b) respectively, and the processed maps used by the algorithm with expanded obstacles and total paths taken being shown in (c) and (d) respectively.

ment. This further demonstrates the viability of letting ΔL go slightly over $2\pi R$ while still having tangle-free global paths.

Since the algorithm assumes a point robot in its operation, which was not an issue for the artificial simulations of previous subsection, to account for the robot dimension in both Gazebo simulations and real-world experiment the map interpreter in the algorithm creates the robot configuration space by expanding the detected obstacles radially by the radius of the circular robot used. This approach also has a convenient side-effect of expanding the obstacles over what could have been a few extra frontier cells behind an obstacle, saving the robot some unnecessary movement.

Results from Gazebo simulations of both environments can be seen in Figure 4.1.4. The "room" environment was fairly straight-forward, with the robot completely exploring the environment and returning to the base without tangling the tether, as expected. The "forest" environment is a bit more involved, with a much larger number of obstacles and no external borders. However, as shown in Figure 4.1.4(d), the robot still managed to explore all the space it could reach with the tether length constraint, while also keeping the tether tangle-free throughout the exploration process. Videos of these simulations can be seen here: https://www.shorturl.at/bcdDW

4.2 Real World Experiments

4.2.1 Externally Tracked Implementation

Our original methodology, using the RRT-based path planner and having no SLAM integration, was first tested with an iRobot's Create 2 robot using a VICON system for localization. The robot was controlled by a laptop with an Intel Core i3 processor clocked at 1.8GHz and 4 gigabytes of memory that runs Linux Ubuntu 18.04. A spring-loaded retractile tether spool was anchored at the base with the tether attached to the robot. For this demo, the robot sensor was simulated as a circle of 70 cm radius centered on the robot. The maximum tether length was set to be 4 m and the length tolerance ΔL to be 0.8 m, which is smaller than the actual theoretical minimum of approximately 2 m and considers the radius of the robot $(0.17\,\text{m})$ and of the smallest obstacle $(0.15\,\text{m})$. Figure 4.2.2 shows snapshots of the experiment while the robot explores a small 3×3 m environment with a circular and a rectangular object. The robot explores the entire environment and returns to the base by following a tangle-free global path. For the sake of comparison, Figure 4.2.1 shows snapshots of the environment when the robot executes a standard frontier-based strategy $(\Delta L = L_{\max} = \infty)$. The environment is completely explored but the tether ended up tangled

around the obstacles. A video of this experiment can be found at https://youtu.be/2lRHnf43T9g.

While this approach was successful in demonstrating the operation of the main algorithm, the lack of autonomous navigation and a pre-determined map made it just a proof of concept. The originally developed RRT-based planner also made this approach's computation times scale poorly with the size and complexity of the environment. It is also worth noting that at the time of that experiment no cost filtering has been implemented yet, so the robot was always selecting the closest frontier cell as the goal.

4.2.2 SLAM Implementation

For our final experiment we used the same real robot to fully explore our research lab environment with some extra obstacles placed around, this time with full SLAM integration. The picture of the environment is shown in Figure 4.2.3. We have used the same robot, equipping it with the YDLIDAR X4 360-degree plannar LiDAR scanner. The same retractile tether spool was used, anchored to the robot's base and starting point for the exploration. The robot is controlled with the same small laptop computer.

We ran two separate instances of the experiment — one using standard frontier-based exploration, and one using our proposed algorithm. Snapshots of the map building process for these two runs are shown in Figure 4.2.4 and Figure 4.2.5 respectively. In these figures, the robot base is shown as a red triangle and the robot's position is shown as a green star. Computed paths that the robot was required to take are shown in solid yellow, and the robot's true paths as reported by the SLAM system are shown in dashed green. The true path and computed path are very close to each other on the figures, making them hard to separate visually, meaning the robot was following the desired path with minimal deviations. Lastly, the tether approximation that was used throughout the exploration is shown in solid purple. As evident from these figures, the standard frontier-based approach that ignored the tether ended up with major tangling, which is visualized by the tether shown in purple being looped around multiple obstacles across the entire testing environment as the robot returned to the base after concluding the exploration. On the other hand, our proposed

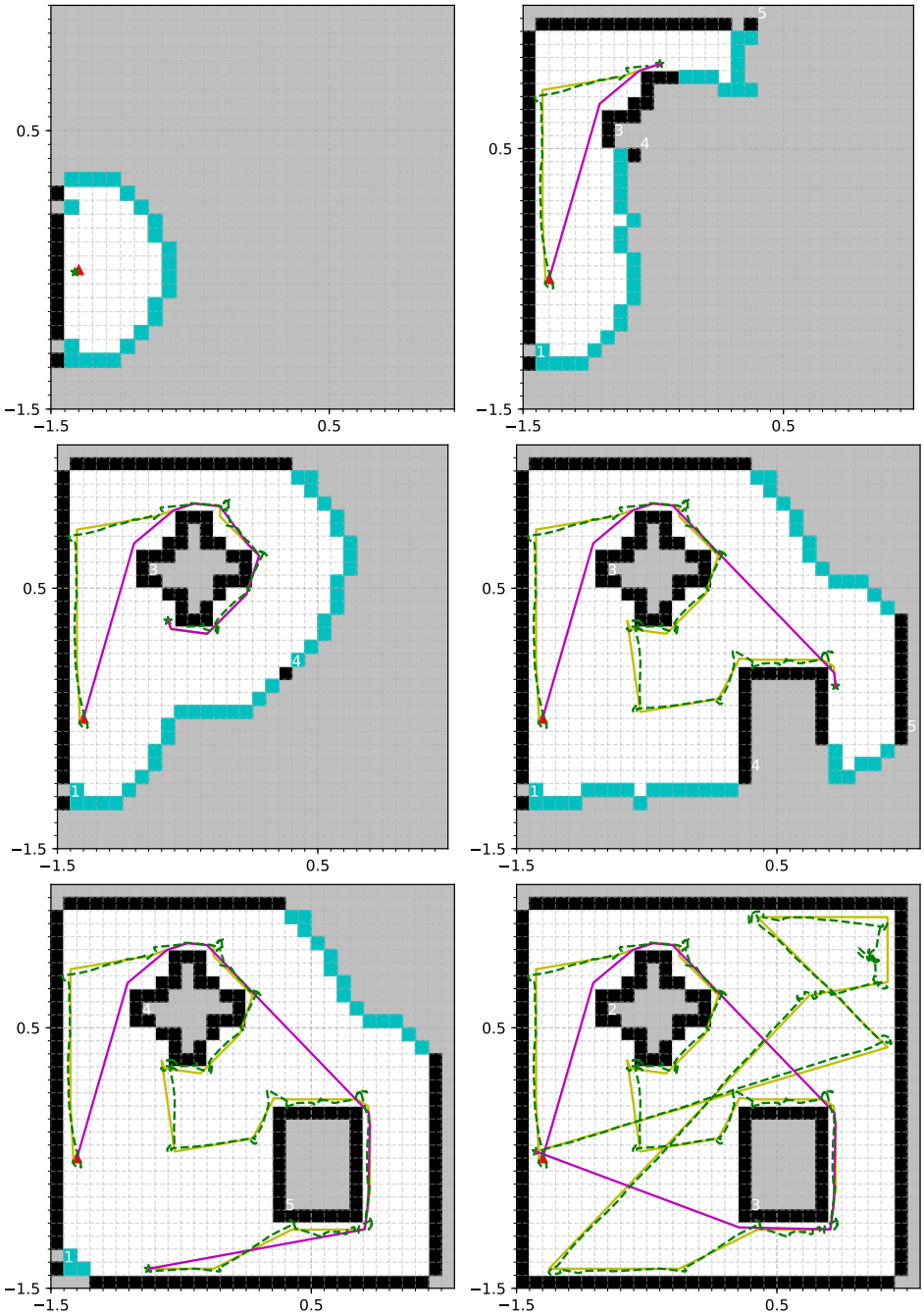


Figure 4.2.1: Snapshots of the externally tracked experiment with standard frontier-based exploration approach used.

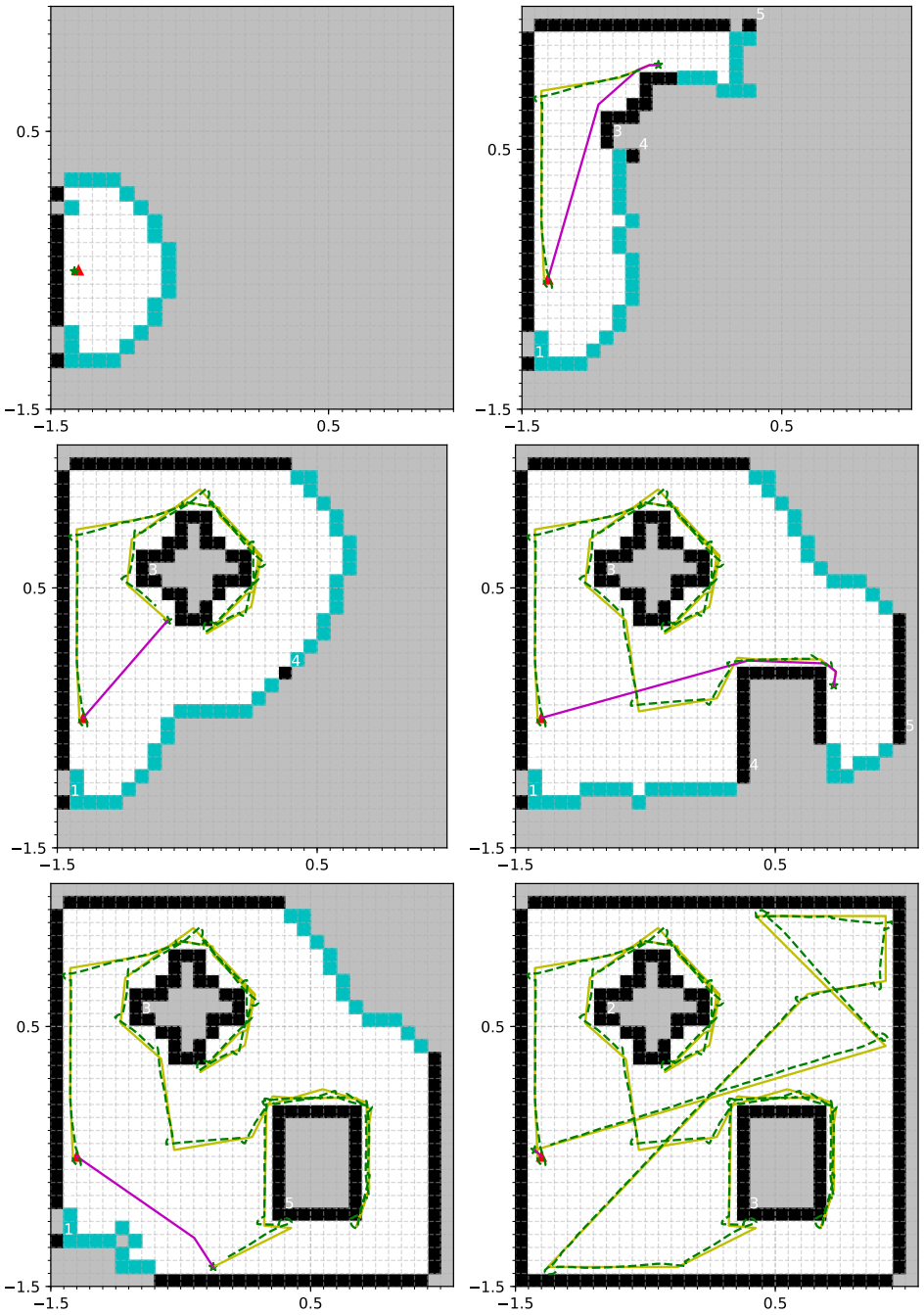


Figure 4.2.2: Snapshots of the externally tracked experiment with our approach used.

Figure 4.2.3: Lab environment used for the experiment.

approach successfully explored the entire environment while keeping the tether in a tangle-free configuration at all times, as the tether is completely retracted by the time the exploration concluded. A video containing similar experiments can be found at https://youtu.be/52bhZoiNjF0.

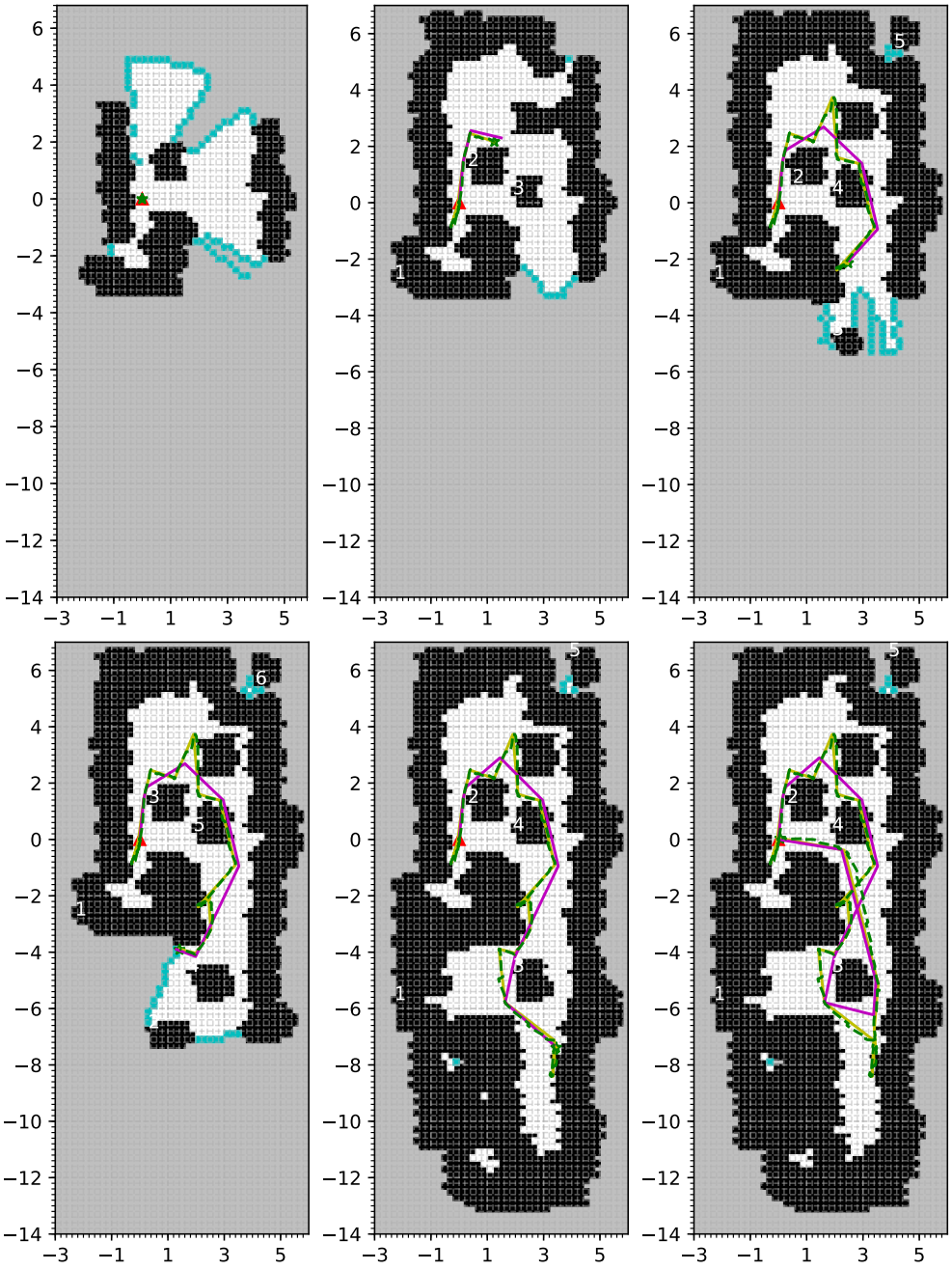


Figure 4.2.4: Snapshots of map building during during the real world experiment using standard frontier-based exploration. The tether approximation during the exploration is shown in purple.

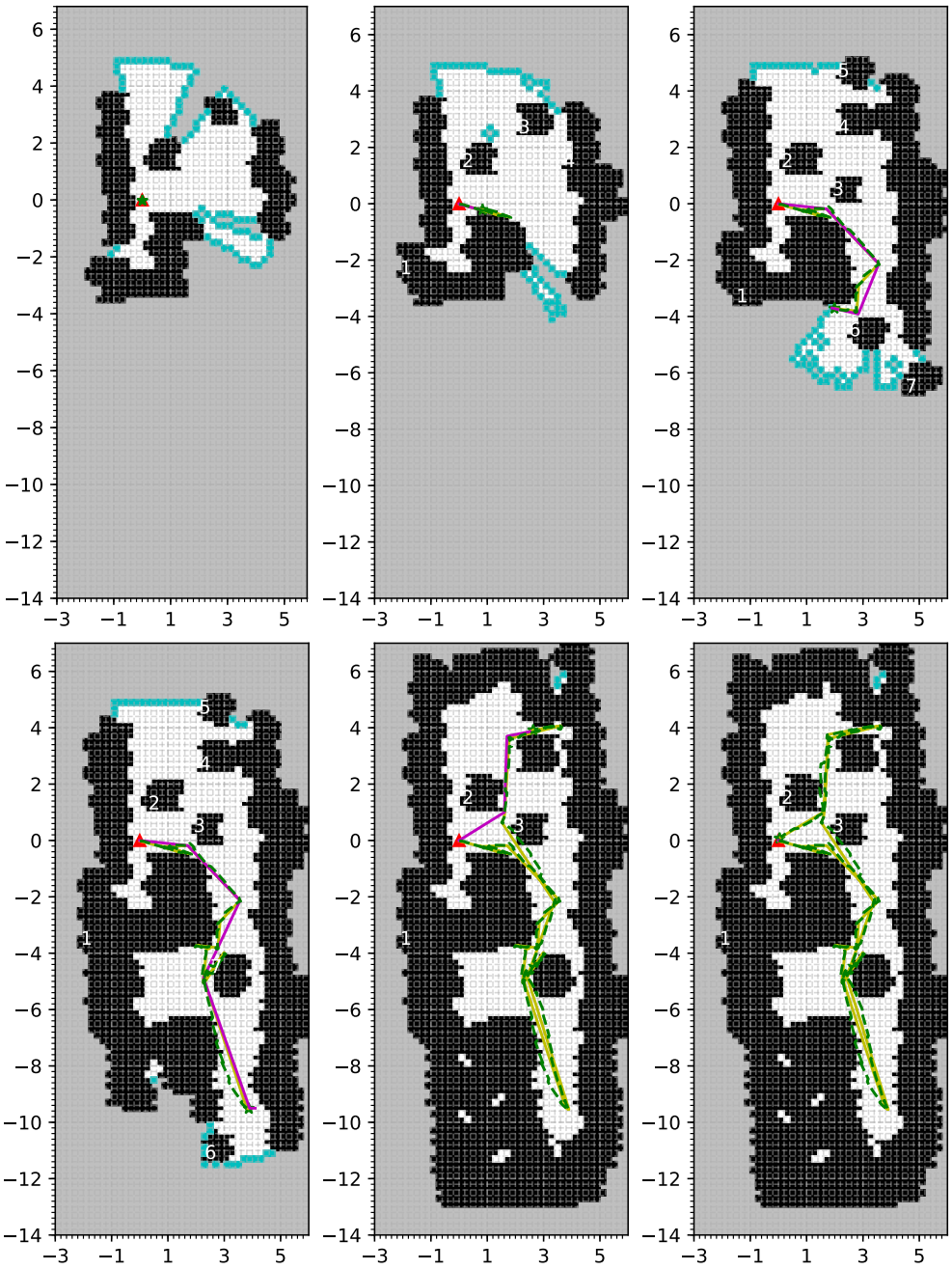


Figure 4.2.5: Snapshots of map building during during the real world experiment using our proposed approach. The tether approximation during the exploration is shown in purple.

5

Conclusions And Future Work

This thesis proposes an algorithm for tangle-free exploration of 2D environments with a tethered mobile robot. The core of the algorithm hinges on the length tolerance value ΔL, which represents the difference between the tether length in proposed total path and the one in the shortest path to the goal, and should ΔL be exceeded — the algorithm computes a path to the goal that will put the tether in the shortest possible configuration. The configuration of the tether is being kept track of by using homotopy invariants, which define a unique path configuration in relation to the obstacles in the environment.

The algorithm was proven correct given the right choice of parameter ΔL, which is used to decide when the robot needs to retracts its tether to avoid tangling. Although our proof determines a maximum value for this parameter in function of the size of the obstacles in the environment being $2\pi R$, where R is the radius of the smallest expected obstacle in the environment, we show experimentally that tangle-free exploration is still possible even with larger errors in this parameter. This is an important characteristic, since we usually do not know the actual size of the obstacles before

starting the exploration of an environment. Our results show the effectiveness of our algorithm in both bounded and unbounded spaces, indicating that it can be directly used for exploring any arbitrary environment with a tethered robot.

There could be multiple applications for such an algorithm where having a tether on the robot is either a convenience or a necessity. One such possible application is disaster response for collapsed buildings or mines. In such scenarios, there will always be a lot of rubble in the environment which can be considered as obstacles, while wireless communication to the robot might be severely hampered. In addition to that, on-board power source of the robot might not be sufficient for long search-and-rescue operations, further facilitating the need for a tether. A lot of such environments, while being fairly large, are still planar, allowing for direct use of our algorithm.

We have initially described a generalized algorithm for our proposed approach. This algorithm is modular, meaning any method can be used for each respective module. These modules include the goal selection, shortest path planning, and homotopic path planning to return the robot to the most optimal tether configuration. Any method or planner can in theory be used for any of those modules, with tangle-free global paths being guaranteed by the ΔL condition.

Our actual implementation of the proposed algorithm is based on a modified frontier-based exploration approach. The main modification is sampling of multiple goals per iteration, subsequently sorting them by their respective costs, with the cost represented as path length required to reach the goal with the homotopy constraints accounted for. This means that in our implementation the goal selection and path planning modules of the generalized algorithm are no longer clearly separated.

The main highlights of our approach are its ease of implementation and its efficiency in terms of global path length with the tether constraint accounted for and the ability to maintain a tangle-free global path. Our testing has demonstrated that the total path length in our approach is 30 to 80% longer than the one in standard untethered frontier-based exploration method depending on the environment size and obstacle density, with a higher percent increase in denser environments. It is also approximately 6 times shorter that the total path length in a conservative method that uses simple backtracking to the base after every iteration as its means of keeping the tether tangle-free. There are however some compromises in our use of a homotopic path optimizer instead of a proper planner. While it offers great efficiency and is perfectly adequate for the purposes of exploration, it is not as versatile as a proper homotopic path planner outside of the proposed scenarios. However,

the RRT-based homotopic path planner that we have initially developed was not reliable enough in complex environments to continue using it.

The main drawback of our approach is its assumption of planar environments. While there are a lot of real-world environments that can be approximated as being planar, like mines and forests, our implemented algorithm without further modifications would fail in true 3D environments, or even should the environment have multiple floors to it. That said, the next logical step in our research would be the extension of our algorithm to 3D environments. This is a complicated task, however. Even though we have successfully tested the algorithm on simulated drones, we still reduced both the drone motion and mapping to the plane (see a video here: `https://youtu.be/A6A7--rLkfo`). While the proposed algorithm itself can work both in 2D and 3D, defining h-signatures and getting accurate tether configuration data becomes much more complex in 3D. Further research is required in this aspect.

References

[1] Tamer Abukhalil, Madhav Patil, and Tarek Sobh. A comprehensive survey on decentralized modular swarm robotic systems and deployment environments. *International Journal of Engineering (IJE)*, 7(2):44, 2013.

[2] Laughlin DL Barker, Michael V Jakuba, Andrew D Bowen, Christopher R German, Ted Maksym, Larry Mayer, Antje Boetius, Pierre Dutrieux, and Louis L Whitcomb. Scientific challenges and present capabilities in underwater robotic vehicle design and navigation for oceanographic exploration under-ice. *Remote Sensing*, 12(16):2588, 2020.

[3] Frederic Bourgault, Alexei A Makarenko, Stefan B Williams, Ben Grocholsky, and Hugh F Durrant-Whyte. Information based adaptive robotic exploration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 1, pages 540–545. IEEE, 2002.

[4] Andrew Bowen, Christopher German, Michael Jakuba, James C Kinsey, Larry Mayer, Dana Yoerger, and Louis L Whitcomb. Lightly tethered unmanned underwater vehicle for under-ice exploration. In *2012 IEEE Aerospace Conference*, pages 1–12. IEEE, 2012.

[5] Howie M Choset, Seth Hutchinson, Kevin M Lynch, George Kantor, Wolfram Burgard, Lydia E Kavraki, Sebastian Thrun, and Ronald C Arkin. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[6] Edson Prestes e Silva Jr, Paulo M Engel, Marcelo Trevisan, and Marco AP Idiart. Exploration method using harmonic functions. *Robotics and Autonomous Systems*, 40(1): 25–42, 2002.

[7] Alberto Elfes. Occupancy grids: A stochastic spatial representation for active robot perception. In *Proceedings of the Sixth Conference on Uncertainty in AI*, volume 2929, page 6, 1990.

[8] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1): 34–46, 2007.

[9] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[10] Emili Hernandez, Marc Carreras, and Pere Ridao. A comparison of homotopic path planning algorithms for robotic applications. *Robotics and Autonomous Systems*, 64:44–58, 2015.

[11] Takeo Igarashi and Mike Stilman. Homotopic path planning on manifolds for cabled mobile robots. In *Algorithmic Foundations of Robotics IX*, pages 1–18. Springer, 2010.

[12] Jamshed Iqbal, Seppo Heikkila, and Aarne Halme. Tether tracking and control of ROSA robotic rover. In *2008 10th International Conference on Control, Automation, Robotics and Vision*, pages 689–693. IEEE, 2008.

[13] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[14] Soonkyum Kim and Maxim Likhachev. Path planning for a tethered robot using multi-heuristic A* with topology-based heuristics. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4656–4663. IEEE, 2015.

[15] Soonkyum Kim, Subhrajit Bhattacharya, and Vijay Kumar. Path planning for a tethered mobile robot. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1132–1139. IEEE, 2014.

[16] Nathan Koenig and Andrew Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2149–2154, Sendai, Japan, Sep 2004.

[17] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical report, Computer Science Dept., Iowa State University, 1998.

[18] Patrick McGarey, François Pomerleau, and Timothy D Barfoot. System design of a tethered robotic explorer (TReX) for 3D mapping of steep terrain and harsh environments. In *Field and Service Robotics*, pages 267–281. Springer, 2016.

[19] Patrick Michael McGarey. *Towards Autonomy and Mobility for a Tethered Robot Exploring Extremely Steep Terrain*. PhD thesis, University of Toronto (Canada), 2017.

[20] Yongguo Mei, Yung-Hsiang Lu, CS George Lee, and Y Charlie Hu. Energy-efficient mobile robot exploration. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 505–511. IEEE, 2006.

[21] Morgan Quigley, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Leibs, Eric Berger, Rob Wheeler, and Andrew Ng. ROS: an open-source Robot Operating System. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA) Workshop on Open Source Robotics*, Kobe, Japan, May 2009.

[22] Ioannis M Rekleitis, Gregory Dudek, and Evangelos E Milios. Graph-based exploration using multiple robots. In *Distributed Autonomous Robotic Systems 4*, pages 241–250. Springer, 2000.

[23] John G Rogers III, Ryan E Sherrill, Arthur Schang, Shava L Meadows, Eric P Cox, Brendan Byrne, David G Baran, J Willard Curtis III, and Kevin M Brink. Distributed subterranean exploration and mapping with teams of UAVs. In *Ground/Air Multisensor Interoperability, Integration, and Networking for Persistent ISR VIII*, volume 10190, page 1019017. International Society for Optics and Photonics, 2017.

[24] Basak Sakcak, Luca Bascetta, and Gianni Ferretti. Homotopy aware kinodynamic planning using RRT-based planners. In *2019 18th European Control Conference (ECC)*, pages 1568–1573. IEEE, 2019.

[25] D Santosh, Supreeth Achar, and CV Jawahar. Autonomous image-based exploration for mobile robot navigation. In *2008 IEEE International Conference on Robotics and Automation*, pages 2717–2722. IEEE, 2008.

[26] Danylo Shapovalov and Guilherme A. S. Pereira. Exploration of unknown environments with a tethered mobile robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 6826–6831. IEEE, 2020.

[27] Danylo Shapovalov and Guilherme A. S. Pereira. Tangle-free exploration with a tethered mobile robot. *Remote Sensing*, 12(23):3858, 2020.

[28] Gokarna Sharma, Pavan Poudel, Ayan Dutta, Vala Zeinali, Tala. Talaei Khoei, and Jong-Hoon Kim. A 2-approximation algorithm for the online tethered coverage problem. In *Robotics: Science and Systems*, 2019.

[29] Iddo Shnaps and Elon Rimon. Online coverage by a tethered autonomous mobile robot in planar unknown environments. *IEEE Transactions on Robotics*, 30(4):966–974, 2014.

[30] Joan Vallvé and Juan Andrade-Cetto. Potential information fields for mobile robot exploration. *Robotics and Autonomous Systems*, 69:68–79, 2015.

[31] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings. 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, pages 146–151. IEEE, 1997.

[32] Brian Yamauchi, Alan Schultz, and William Adams. Mobile robot exploration and map-building with continuous localization. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation*, volume 4, pages 3715–3720. IEEE, 1998.

[33] Daqing Yi, Michael A Goodrich, and Kevin D Seppi. Homotopy-aware RRT*: Toward human-robot topological path-planning. In *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, pages 279–286. IEEE, 2016.

[34] Jingchao Zhao, Junyao Gao, Fangzhou Zhao, and Yi Liu. A search-and-rescue robot system for remotely sensing the underground coal mine environment. *Sensors*, 17(10):2426, 2017.