Special Issue: DARPA Subterranean Challenge, Advancement and Lessons Learned from the Finals (DARPA SubT Final)

Regular Article

ArtPlanner: Robust Legged Robot Navigation in the Field

Lorenz Wellhausen and Marco Hutter[*]
Robotic Systems Lab, ETH Zürich, Switzerland

Abstract: Due to the highly complex environment present during the DARPA Subterranean Challenge, all six funded teams relied on legged robots as part of their robotic team. Their unique locomotion skills of being able to step over obstacles require special consideration for navigation planning. In this work, we present and examine ArtPlanner, the navigation planner used by team CERBERUS during the Finals. It is based on a sampling-based method that determines valid poses with a reachability abstraction and uses learned foothold scores to restrict areas considered safe for stepping. The resulting planning graph is assigned learned motion costs by a neural network trained in simulation to minimize traversal time and limit the risk of failure. Our method[1] achieves real-time performance with a bounded computation time. We present extensive experimental results gathered during the Finals event of the DARPA Subterranean Challenge, where this method contributed to team CERBERUS winning the competition. It powered navigation of four ANYmal quadrupeds for 90 minutes of autonomous operation without a single planning or locomotion failure.

Keywords: navigation, legged robots, learning, planning, subterranean robotics

1. Introduction

Navigation planning for legged robots has distinct challenges that are not present for other types of robots. While flying robots attempt to avoid any contact with the environment, ground robots by definition require contact with the ground to locomote. Compared to other types of ground robots, which have a constant contact patch with the ground, legged robots can overcome obstacles by lifting their legs. Most traditional navigation planning approaches assume a single traversability value for any given terrain patch, which they check against the footprint of the robot (Wermelinger et al., 2016). These approaches are limiting for legged robots because their ability to change their footprint and choose contact locations with the environment is deliberately not accounted for. Defining a traversability value for such a highly articulated systems is extremely challenging due to the high dimensionality of the problem.

[*]This work was supported by the Swiss National Science Foundation (SNF) through the NCCR Robotics and project 188596, and the EU Horizon 2020 research and innovation programme under grant agreements No 780883, No 852044 and No 101016970. It has been conducted as part of ANYmal Research, a community to advance legged robotics.

[1] http://github.com/leggedrobotics/art_planner

Received: 19 May 2022; revised: 18 October 2022; accepted: 23 February 2023; published: 23 March 2023.

Correspondence: Lorenz Wellhausen, Robotic Systems Lab, ETH Zürich, Switzerland, Email: lorenwel@ethz.ch

DOI: https://doi.org/10.55417/fr.2023013

Figure 1. Team CERBERUS won the DARPA Subterranean Challenge Finals with four ANYmal quadrupeds deployed during the Prize Run. The navigation planner presented in this work guided all four robots safely during the hour-long mission.

To find a stable configuration for a legged robot, we need to find a base pose where the terrain is within the range of motion of each limb. We call this reachability checking, where a robot is represented as one collision volume for its torso, and one reachability volume for each of its limbs (Tonneau et al., 2015). When checking the feasibility of a given robot pose, the torso volume is expected to be collision-free, while collision is enforced for the reachability volumes, to ensure that the robot is able to make environment-contact with its legs. While this approach can be used to find all theoretically feasible poses, basic reachability does not account for the dynamics of the robot, and in practice most locomotion controllers do not use the full range of motion. This issue can be alleviated somewhat by restricting geometry which is considered safe for contacts, and heuristics on the range of motion used by a specific locomotion controller. This mostly produces feasible paths, but the cost of walking over challenging terrain and the risk of stepping close to obstacles and edges is not considered (Wellhausen and Hutter, 2021).

In this work, we consider the path planning problem for legged robots on a robocentric height map, which is of fixed size, centered around the robot. Since we are interested in a navigation planner that can work in possibly unknown environments, as shown in Figure 1, it will use information from an on-board mapping pipeline that is continuously updated as the robot moves. We therefore require a fast update rate for our planner to keep up with map updates. Building a planning graph incrementally by maintaining the graph between planning queries can speed up planning in static environments where most map updates will not invalidate the planning graph. However, this approach necessitates graph maintenance between map updates, which adds heuristics and algorithmic complexity to identify updated graph elements. The extreme challenges posed by the DARPA Subterranean Challenge (SubT), such as smoke and dynamic obstacles, necessitate a more robust solution.

1.1. Contributions

In this work, we therefore introduce ArtPlanner, a navigation planner for legged robots that uses geometric reachability checking to find valid poses and a learned motion cost to find optimal paths that are safe and practically feasible. This combination is enabled by a novel graph construction method: it creates a new planning graph every time the map is updated. To keep the computation time low, we lazily sample candidate pose vertices until we reach a limit of the number of poses, the number of edges in the graph, or the sampling time. Because we plan on a robocentric map of limited size, this allows us to densely sample the planning map while providing a limit on the sampling duration. We then validate all graph edges at once by applying a locomotion risk threshold, leveraging massively parallel execution of a cost prediction network on the GPU. This allows for consistent and fast planning times. The resulting paths are collision-free and can be followed safely by the used locomotion controller.

When robotic algorithms are deployed on hardware, they have to deal with the imperfections of the real world and interact with other, possibly flawed parts of a robot software and hardware stack. It is therefore beneficial to apply certain heuristics and make tweaks to part of the stack. These "small tricks" are often glossed over, but they can be essential to making a system robust in practice. For us, this was the case when computing the height map, which has the biggest impact on the outcome of planning decisions. We detail three important components of our height map processing pipeline, and we show why each of them played an important role during the DARPA Subterranean Challenge (SubT) Finals.

We extensively evaluated ArtPlanner during SubT, and we present results gathered on four legged robots during the Finals, for ArtPlanner and GBPlanner2, the planner used for exploration (Kulkarni et al., 2022). We provide detailed analysis of the challenges faced by ArtPlanner, and how it managed to overcome these adverse conditions. In addition, we compare our method to other state-of-the-art planners on the data gathered during the Finals, and we show why other methods would not have been robust enough for SubT.

Finally, we provide the source code of our implementation at https://github.com/leggedrobotics/art_planner.

1.2. Related Work

Navigation planning for mobile robots is a vast field of research with many different approaches.

Most navigation approaches for mobile robots use a geometric environment representation as their basis for planning (Wermelinger et al., 2016; Krüsi et al., 2017; Chavez-Garcia et al., 2017; Oleynikova et al., 2017). They use various terrain representations for planning, most commonly 2.5D height maps (Wermelinger et al., 2016; Chavez-Garcia et al., 2017), point clouds (Krüsi et al., 2017), or truncated signed-distance fields (TSDFs) (Oleynikova et al., 2017). Because planning in full 3D representations is currently computationally prohibited (Tonneau et al., 2018), we chose to work with 2.5D height maps as the environment representation. Most planning approaches compute a single geometric traversability value per terrain patch (Wermelinger et al., 2016; Krüsi et al., 2017) as a measure for how easily the terrain can be traversed, irrespective of robot orientation. Thereby, they neglect the much higher mobility that legged robots provide due to their ability to step over obstacles. An overview of different traversability analysis approaches can be found in a recent survey article (Borges et al., 2022). While we have argued in previous work (Wellhausen et al., 2019; Wellhausen et al., 2020) that purely geometric approaches are not sufficient for navigation in natural outdoor environments, approaches relying on semantic information exhibit the same issues as traditional geometric approaches. They predict, either implicitly (through semantic segmentation of the environment) (Rothrock et al., 2016; Bradley et al., 2015; Otsu et al., 2016; Valada et al., 2017) or explicitly (Kim et al., 2006; Barnes et al., 2017; Hirose et al., 2018), a traversability label. However, we can instead reinterpret traversability labels as foothold feasibility labels and use them to enhance geometric planning. While full kino-dynamic planning over long horizons would be the most general and accurate planning method, applying these methods in real-time is not tractable on current computational hardware (Tonneau et al., 2018; Fernbach et al., 2017; Winkler et al., 2018). Dynamic planning using a reduced robot model has recently shown promising results (Norby and Johnson, 2020), but it has not been evaluated in deployment scenarios. Other work on navigation planning specifically for legged robots either only considers cases of obstacle avoidance on flat terrain (Zhao et al., 2018; Harper et al., 2019), or it does additional contact planning, which pushes computational complexity past the real-time mark (Belter et al., 2019; Lin and Berenson, 2017; Reid et al., 2020). Approaches that learn traversability (Chavez-Garcia et al., 2017) or motion cost (Guzzi et al., 2020; Yang et al., 2021a) are powerful, but they are either too slow due to the sequential querying of neural networks during sampling-based planning (Guzzi et al., 2020), or they struggle in tight spaces where precise motion checking is necessary (Yang et al., 2021a). In previous work (Wellhausen and Hutter, 2021), we have used reachability planning with a learned foothold score to achieve real-time performance for legged navigation planning. While the resulting paths were generally feasible, the employed shortest path cost did not sufficiently account for locomotion risk on challenging terrain or close to obstacles. For further related work of other navigation planners used during SubT, please refer to Section 1.3.1.

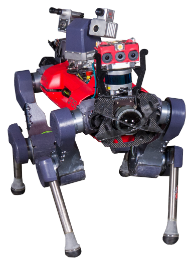
ArtPlanner uses a reachability-based robot representation (Tonneau et al., 2015) and learned foothold scores (Wellhausen and Hutter, 2021) with batched motion cost computation (Yang et al., 2021a). This is the first work that combines geometric collision checking and learned motion costs in a navigation planner for legged robots.

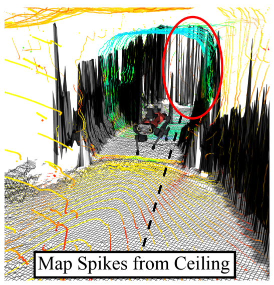
1.3. DARPA Subterranean Challenge

The DARPA Subterranean Challenge (SubT) was a robotics challenge initiated by the Defense Advanced Research Projects Agency (DARPA) in 2018. Its goal was to expedite development of robotic systems to rapidly map, navigate, and search complex underground environments such as human-made tunnel systems, an urban underground, and natural cave networks. Three *Circuit* events were held focusing on the aforementioned underground environments to qualify for the *Finals* event in Louisville, KY, which combined all environments into a single, purpose-built course. Eight qualified teams, six of which were DARPA-funded, competed on September 21–23, 2021 to explore the unknown course to find, locate, and identify a number of artifacts. For every artifact that was identified correctly and localized accurately, a point was awarded. Two preliminary rounds of 30 minutes were followed by the all-deciding 60 minute *Prize Run*. Only a single human supervisor was allowed to remotely interact with the robots once they entered the competition area, and communications were severely limited, requiring a high level of autonomy from the robots. The composition of the robot teams was not prescribed by DARPA, and approaches changed over the four years of the competition. In the end, all funded teams brought at least one legged robot to the *Finals*. Our team, CERBERUS (Tranzatto et al., 2022b), which won the competition, focused our ground robot efforts exclusively on legged robots from the beginning and brought four ANYmal-C (Hutter et al., 2016) quadrupeds to the *Finals*.

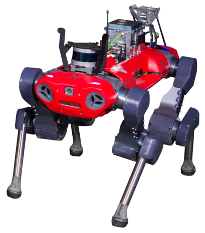
1.3.1. Navigation Planners in the SubT Challenge

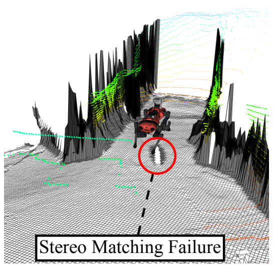
The approaches for ground robot navigation used during SubT are diverse. Team CoStar used a 2D multilayer risk map to assess the terrain and planned the robot base path using a risk-aware kinodynamic MPC planner (Fan et al., 2021; Thakker et al., 2021). Team CSIRO Data61 used heuristic height map features to assess terrain traversability. They also used the concept of virtual surfaces (see Section 2.4.1) to compute a lower bound for terrain inclination, which was used to avoid negative obstacles (Hines et al., 2021). Additionally, they used a deep reinforcement learning policy to control their tracked robots through narrow gaps (Tidd et al., 2021). Team Explorer developed a kinodynamic local planner (Chao et al., 2021) as well as a viewpoint-based planner using a polygonal representation of the environment (Yang et al., 2021b). However, both works do not explicitly state how the traversable regions and obstacles were computed. Team CTU-CRAS-NORLAB computed traversable regions in a height map based on the neighboring cell height difference and planned on using the A* algorithm (Bayer and Faigl, 2019; Bayer and Faigl, 2020). Team MARBLE used an Octree representation of the environment and computed traversability based on the surface normal of extracted ground voxels (Ohradzansky et al., 2021). They employed a graph-based global planner combined with a reactive local controller. All other funded teams besides team CERBERUS used a team of heterogeneous ground robots and therefore did not explicitly consider the capabilities of legged robots.


(a) Explorer specification



(b) Explorer sensor data and height map


(c) Carrier specification



(d) Carrier sensor data and height map (depth camera data not shown)

Figure 2. Modified ANYmal-C robots used during the competition.

1.4. ANYmal Hardware

For the *Finals*, team CERBERUS customized four ANYbotics ANYmal-C (Hutter et al., 2016) robots with two different specifications. All robots carried a Velodyne VLP-16 puck lidar for localization and mapping. For computations, they had two PCs with an Intel i7-8850H CPU as well as an Nvidia Jetson Xavier for GPU processing, connected via ethernet. An *explorer* specification robot is shown in Figure 2(a). They were equipped with two Robosense BPearl dome lidars placed at the front and the rear of the robots and a pan-tilt head on their back which housed various sensors and a lighting system for artifact detection. Figure 2(c) pictures a *carrier* robot. They retained the four Realsense D435 depth cameras placed on each side of the base-spec ANYmal and were equipped with a mechanism for dropping WiFi mesh nodes.

In the context of this work, the most essential distinction between the two is the sensors used for height mapping of the environment. The *explorer* robot used its two dome lidars for height mapping, which return highly accurate and reliable readings. However, its scan pattern is sparse and requires sweeping the ground through robot motion to obtain a dense height map. Furthermore, its 180 degree field-of-view (FoV) means that the ceiling is also frequently observed, which can lead to issues with 2.5D height mapping (see Section 3.3.1). The *carrier* robot used its four depth cameras for height mapping, which combined provide 360 degree coverage around the robot. Unfortunately, their somewhat small stereo baseline meant depth accuracy dropped below a usable range around 2 m from the robot. In addition to their limited range, their active stereo matching technology can also lead to outliers on reflective and low-texture surfaces as well as missing data in environments with high lighting contrast. On the *carrier*, we therefore also used the Velodyne VLP-16 puck lidar to get reliable height readings farther from the robot. Examples for the sensor data and the resulting height map for both specifications at the entrance to the Cave section of the *Finals* course are shown in Figure 2(b)+2(d). We used the same navigation planner with a single set of parameters for both robot specifications such that our method had to deal with their different height map characteristics.

1.5. Navigation Stack

In the context of SubT, ArtPlanner was embedded into a larger navigation stack, shown in Figure 3, to provide capabilities for autonomous exploration, and to actually follow computed paths. These components were connected by a behavior tree to provide robustness and enable direct goal input to the navigation planner from the operator. Details of this are beyond the scope of this work, and they can be obtained from the overview article of team CERBERUS (Tranzatto et al., 2022a).

Our graph-based exploration planner (Dang et al., 2019; Kulkarni et al., 2022) (GBPlanner2) plans to maximize information gain along the robot path. It operates on a global 20 cm voxel-size volumetric map (Oleynikova et al., 2017) which is too sparse to capture the terrain in sufficient detail for legged robot navigation. Knowing that we had ArtPlanner to provide safe navigation, we

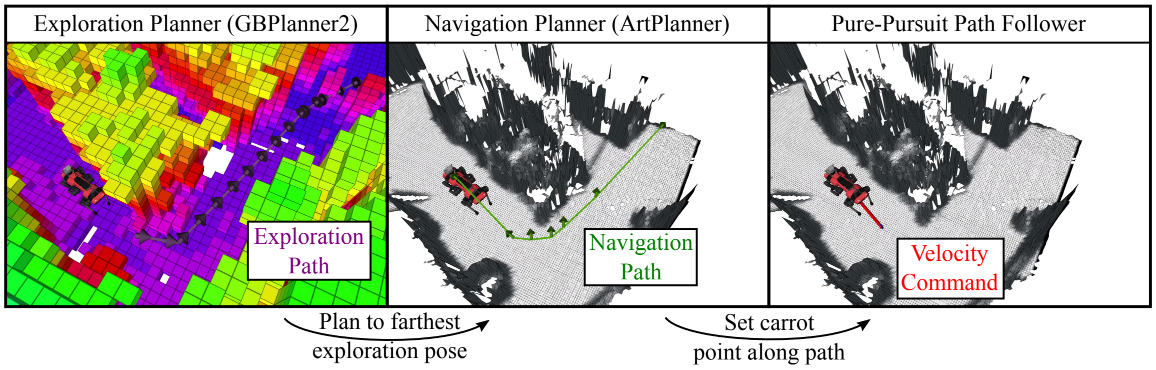

Exploration Planner (GBPlanner2)
Navigation Planner (ArtPlanner)
Pure-Pursuit Path Follower
Exploration Path
Navigation Path
Velocity Command
Plan to farthest exploration pose
Set carrot point along path

Figure 3. Team CERBERUS' navigation stack. An exploration planner (Kulkarni et al., 2022) plans on a coarse volumetric map to maximize information gain. ArtPlanner then plans to the farthest reachable exploration pose on a more fine-grained robocentric height map. Navigation paths are tracked using a pure-pursuit path follower.

also tuned the parameters of GBPlanner2 to be optimistic and favor exploration gain over safety considerations. Specifically, this meant we used a small collision volume for validity checking without any safety margin to fit through narrow openings. Additionally, similar to the concept of virtual surfaces (Hines et al., 2021), GBPlanner2 is allowed to output so called "hanging vertices," where vertices in free space are not supported by any ground surface underneath, but only unobserved space (see Section 3.2.2). We therefore need our navigation planner to refine the exploration path in cases where its low-resolution map causes suboptimal or risky paths, and to stop the robot if the path is completely infeasible. For this, we plan on a local height map that is centered at the current robot positions and moves with it.

When ArtPlanner receives a new exploration path, it iterates through the path poses in reverse, starting with the farthest one, and tries to plan to each pose. This is done to maximize the planning distance, such that ArtPlanner can optimize the path and circumvent any obstacles that might have been missed by the exploration planner. We continuously repeat this at a target planning rate of 0.5 Hz. When we successfully plan to an exploration path pose, all path poses that precede it are considered reached and are not used for future planning iterations. Planning of the GBPlanner2 is triggered in two cases: (i) if the exploration path is infeasible and therefore contains no pose to which we can plan, or (ii) if the last pose in the exploration path has been reached.

The resulting navigation path is tracked using a pure-pursuit PID path follower. It considers both the horizontal distance as well as the angular distance when setting the carrot point along the path. This is done to ensure accurate tracking in tight spaces and speedy locomotion when moving forward. Since we used a single robust and perceptive locomotion controller (Miki et al., 2022a) we did not have to consider any gait or locomotion controller switching.

2. Method

ArtPlanner uses reachability-based pose validity checking with learned foothold scores and computes path costs using a learned motion cost module. Because a high-quality map of the environment is crucial for good path planning, we perform additional processing on the input height map to increase its quality and improve safety.

2.1. Reachability Planning

To check for validity of sampled robot poses, we use a reachability-based approach (Tonneau et al., 2015). It is based on the notion that the ground support surface needs to be reachable for the robot's legs, while the torso remains collision-free. To check for this condition, we decompose the robot body into volumes representing torso collisions and leg reachability, as shown in Figure 4(a).

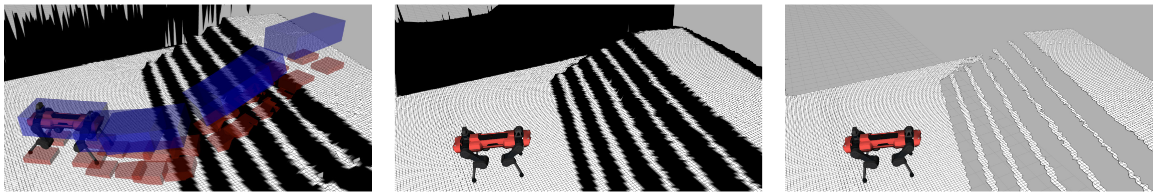

(a) Reachability Abstraction **(b)** Torso Collision World **(c)** Reachability Collision World

Figure 4. We use a reachability abstraction of the robot (a). Blue boxes need to be contact-free in the torso collision world while red boxes need to be in contact with the reachability collision world. The torso collision world (b) considers all geometry perceived by the mapping algorithm. Regions considered unsuitable for footholds are colored in black. This geometry is removed (c) when checking reachability volumes for collision.

To account for geometry that should not be used as a support surface (i.e., walls), we trained a convolutional neural network (CNN) to predict a foothold score based on height map information, and we used this to constrain the regions considered for valid footholds (Wellhausen and Hutter, 2021). Because the CNN is very small, with few parameters, we can train it with only 20 hand-labeled samples. The foothold score is indicated with black and white in Figure 4. Geometry that has a low foothold score is disregarded for collision checking of the limb reachability volumes but is still considered for the torso, visualized in Figure 4(b) and 4(c). A valid pose is therefore a pose where all reachability volumes are in contact with valid geometry, while the torso volume is not in collision with any geometry.

Inspired by the sampling-based LazyPRM* (Hauser, 2015) algorithm, ArtPlanner only checks the sampled pose for validity when adding a new node to the graph, but it does not immediately check the validity of newly added edges. Edge checking is performed using batched motion cost computation when a path is queried, as discussed in Section 2.2. We use a custom sampling scheme that uses a 2D pose sample augmented to a 3D pose using map information and biases sampling towards regions with low node density (Wellhausen and Hutter, 2021).

2.2. Learned Motion Cost

A simple shortest-path cost can lead to risky paths that require the robot to step close to obstacles and edges. Due to the difficulty of analytically determining locomotion cost and risk, other recent work has proposed using a learned neural network to compute cost (Guzzi et al., 2020). Building upon these works, we use a neural network that computes the energy and time required and failure risk associated with moving from a query location in the height map to a given relative 2D pose (Yang et al., 2021a).

The inputs to the network are a patch of the height map centered around the robot, the current yaw orientation of the robot, as well as the 2D goal pose relative to the robot. The network outputs the time c_t and energy c_e required to move to the target pose as well as the motion risk c_r, which is the probability that this transition fails. The network is trained using data generated in simulation, where the robot is spawned in a random location on a randomly generated height map, and a goal pose in a certain range around the robot is randomly selected. We then naively give a directional command to the robot to move towards this goal pose while the robot experiences external disturbances. This is repeated a number of times to compute the failure probability. Time and energy values are averaged over all attempts that successfully reached the goal.

The risk value is used to determine validity of planning graph edges, while the graph edge cost is computed as a weighted sum of time c_t, energy c_e, and risk c_r:

$$c = w_t \cdot c_t + w_e \cdot c_e + w_r \cdot c_r \tag{1}$$

c_t and c_e are normalized with their respective maximal value in the training dataset, whereas c_r expresses the probability of failure. Battery runtime of the ANYmal robots was not of concern, and energy consumption c_e was shown to be highly correlated with time c_t, so we chose to disregard this cost term. Since any navigation error would be potentially mission-ending during the SubT *Finals*, we put a high weight on risk. Consequently, our chosen cost weights were $w_t := 1$, $w_e := 0$, and $w_r := 5$.

2.3. Graph Construction

We construct our planning graph using lazy sampling, where only the added vertices but not edges are immediately checked for validity. When querying a solution in a regular LazyPRM* graph, an A* search is performed, and all edges that are part of the optimal solution are checked for validity. Any invalid edges are removed from the graph, and the process is repeated until either all edges returned by the A* search are valid (success), or the start and goal vertices are no longer connected (failure). This makes the path query time highly nondeterministic since the number of A* search iterations strongly depends on the complexity of the environment, which determines the number of invalid edges. As discussed in our previous work (Wellhausen and Hutter, 2021), this can lead to excessive planning times, which make real-time applications impractical or impossible. While our previously suggested graph expansion method (Wellhausen and Hutter, 2021) can compensate for this issue in most cases, during extensive real-world testing in preparation for the SubT challenge we still encountered some instances of long planning times in difficult environments where the number of invalid edges in the graph is exceedingly large. In addition, when used with a learned motion cost, classical sampling-based planning requires inefficient, sequential querying of the motion cost network. This drastically increases planning times (Guzzi et al., 2020).

To achieve consistent and low planning times, we therefore leverage batched edge cost computation using our motion cost network, detailed in Figure 5. Every time a new planning query is received and the map has updated, we build a new planning graph by lazily sampling until the graph exceeds either N valid vertices, M unchecked edges, or we have sampled for more than T seconds. The number of graph edges determines the batch size for the motion cost query, which in turn determines the inference time of the motion cost network. Therefore, the upper bound of unchecked edges M limits the maximal computation time used by the cost query. The other two termination criteria prevent unnecessarily long and dense sampling in certain environments where either the traversable area of the map is very small (T limit) or the geometry allows every node to only have a small number of neighbors (N limit), like in narrow corridors.

A batched query of the cost network is then performed for every edge in the graph at once, which can be executed efficiently on GPU. Edges that exceed a risk threshold R are removed, and valid edges are assigned a cost, according to Equation 1.

We now have a fully validated graph with assigned edge costs, and consequently a single A* search will return the optimal path between two query nodes.

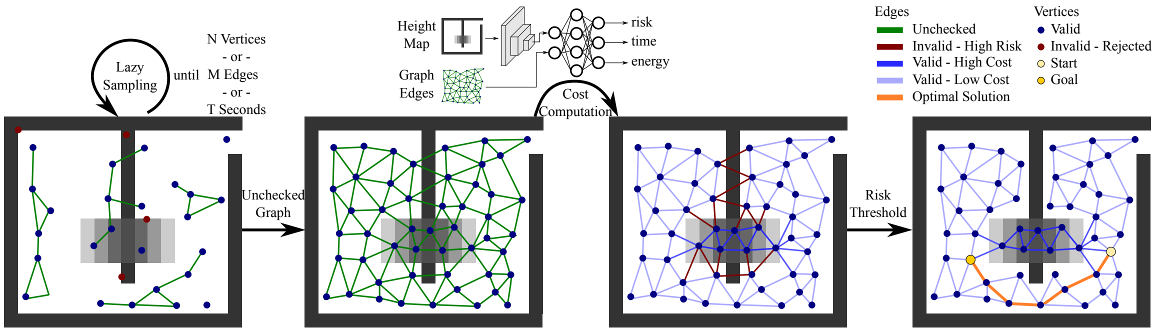


Figure 5. Our planner lazily samples graph poses, only checking state but not edge validity. Once a sampling limit is reached, the motion cost for every graph edge is computed in a batched fashion. The graph is pruned by applying a motion risk threshold, and the optimal path can be found using an A* search.

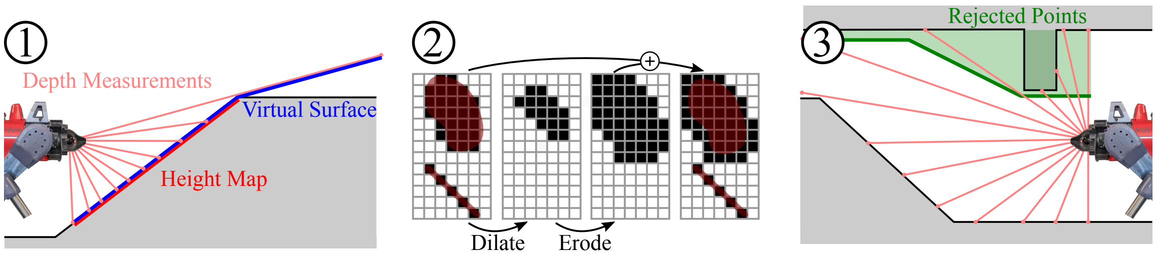


Figure 6. ① We use depth measurement rays to infer virtual surfaces above the robot in unobserved map space. ② By first dilating steppable map terrain and then eroding it, we compute a safety threshold around dangerous terrain (red). ③ We reject ceiling points when computing the height map using a height threshold which changes with distance from the robot, to handle both low openings and steep inclines.

2.4. Height Map Processing

The height map provided to the planner is its only source of information about the environment. Obtaining a high-quality height map is therefore paramount for safe planning. To achieve this, we perform additional processing steps to improve map quality and increase planning safety.

2.4.1. Virtual Surfaces

The depth sensors on our legged robots are mounted about half a meter above the ground. Especially on rough terrain and inclines, this can often lead to large occlusions in the height map. To combat this issue, we compute virtual surfaces (Hines et al., 2021) as the upper bound of a height value in unobserved cells, shown in Figure 6 ①.

We cast a ray from the sensor origin to every point in the observation point cloud. For every cell this ray passes through, we know that its height value cannot be larger than the height at which the ray passes through the cell. This way, we can fill unknown map regions using sensor information, instead of simply filling them using an image inpainting algorithm or considering them untraversable.

However, the low sensor placement means that when we approach a negative obstacle, like a cliff, its virtual surface will appear to have a very small inclination, until we are very close to it. This introduces additional risk, if the planner and path follower are not able to stop the robot in time, once the severity of the drop appears in the virtual surface. We therefore decided to only use virtual surfaces for planning if they are above sensor height. This way, we can benefit from them if we are walking up inclines (see Section 3.2.3), but also stay safe in the presence of negative obstacles (see Section 3.2.2).

2.4.2. Safety Threshold

We rely on the learned motion cost network to compute the path risk to avoid walking too close to dangerous obstacles. While this works well if the environment is mapped well, negative obstacles often do not appear in the height map, as discussed above in Section 2.4.1. We can therefore not rely on the motion cost network to keep a safe distance from negative obstacles.

To combat this issue, we apply a safety margin to disallow stepping too close to edges, illustrated in Figure 6 ②. It is implemented using image erosion on the foothold score layer of the height map, which reduces the steppable map region by a safety margin. This has the additional benefit of also removing small isolated steppable patches from the map, which could only be overcome by solving a stepping-stone problem. To avoid unnecessarily inflating small obstacles like rails, which the robot can easily step over, we do not inflate unsteppable regions below a certain size. In practice, this is done by performing an image dilation of smaller radius, before doing the erosion.

This safety threshold was crucial on the Subway Station of the *Finals* circuit, as shown in Section 3.2.2.

2.4.3. Ceiling Point Filter

We use a 2.5D height map representation for planning. While this is sufficient for ground robot navigation in most environments, it can be problematic in the tight underground spaces encountered during SubT. Low ceilings mean that they are frequently observed by the depth sensors, which causes spikes in the height map, as shown in Figure 2(b). However, we cannot simply discard all points above a fixed height, since this would either prevent us from planning up slopes or from passing underneath low overhangs.

We therefore use a rising height threshold to filter points (Miki et al., 2022b), shown in Figure 6 ③. It filters points just above robot height close to the robot, and linearly increases the height threshold up to a maximum at larger distances. This setup caused map spikes in parts of the course with low ceilings, which slowed us down, but these crucially never stopped us from exploring. It allowed us to pass underneath very low overhangs, and to plan up slopes, even when encountered together, as detailed in Section 3.3.1.

3. Experimental Results

ArtPlanner was deployed on all four ANYmal-C ground robots of team CERBERUS during all runs of the SubT *Finals*. It used a height map of size 8 m × 8 m with a 4 cm resolution. We only cover results related to the navigation planner presented in this work. For further details on the general performance, we refer to our overview article (Tranzatto et al., 2022a).

We deployed all four ground robots during the *Prize Run*, which were directed by the supervisor to explore different areas of the course. All ground robots successfully made it to the end of the competition and we did not observe a single path planning or locomotion failure, which could have been provoked by bad path planning. Our planner was active for 90 minutes between all robots, which accounts for 88.94% of all robot motion. We gracefully navigated the narrow doorways and small rooms in the Urban section, passed through the Tunnel section with obscuring fog, and made it through the narrowest and roughest part of the Cave section. The only case where ArtPlanner did not follow the exploration path over traversable terrain happened at the stairs leading to the subway station. This resulted from our operational decision to use older, well-tested motion cost network weights, which produced elevated risk levels on stairs. We preferred these weights over newer, untested weights, which perform well on stairs, since stairs were not a prominent feature in the *Finals* course. We only collided with the environment a single time, getting caught on a narrow pole due to path following delays, discussed in further detail in Section 3.3.2. The only other issue encountered by our planning method was slow progression due to artifacts in the height map, as outlined in Section 3.3.1. Nevertheless, we safely explored large parts of the competition course, as shown in Figure 7.

3.1. Comparison

Directly following GBPlanner2's exploration path (Dang et al., 2019; Kulkarni et al., 2022) during the competition would almost certainly have led to major issues. It did not sufficiently account for traversability characteristics of the terrain, planning directly over high rails shown in Figure 7 ①. It even completely missed some smaller obstacles, like the traffic cones shown in Figure 7 ②. This could have possibly been avoided with more conservative tuning of the exploration planner, but this would have significantly impeded exploration.

To show that ArtPlanner was crucial to performing well in the challenging environment of the SubT *Finals*, we compare it to other navigation planning methods, using data collected during the Prize Run. Figure 7 shows the motion cost and collision rate for all robots and all planners as a heat map overlayed over the top-view of the competition course.

3.1.1. Other Planners

Naturally, only ArtPlanner was running during the *Finals*. We recorded all paths planned during the *Finals* to evaluate performance postevent. To compare with other methods, we played back the state

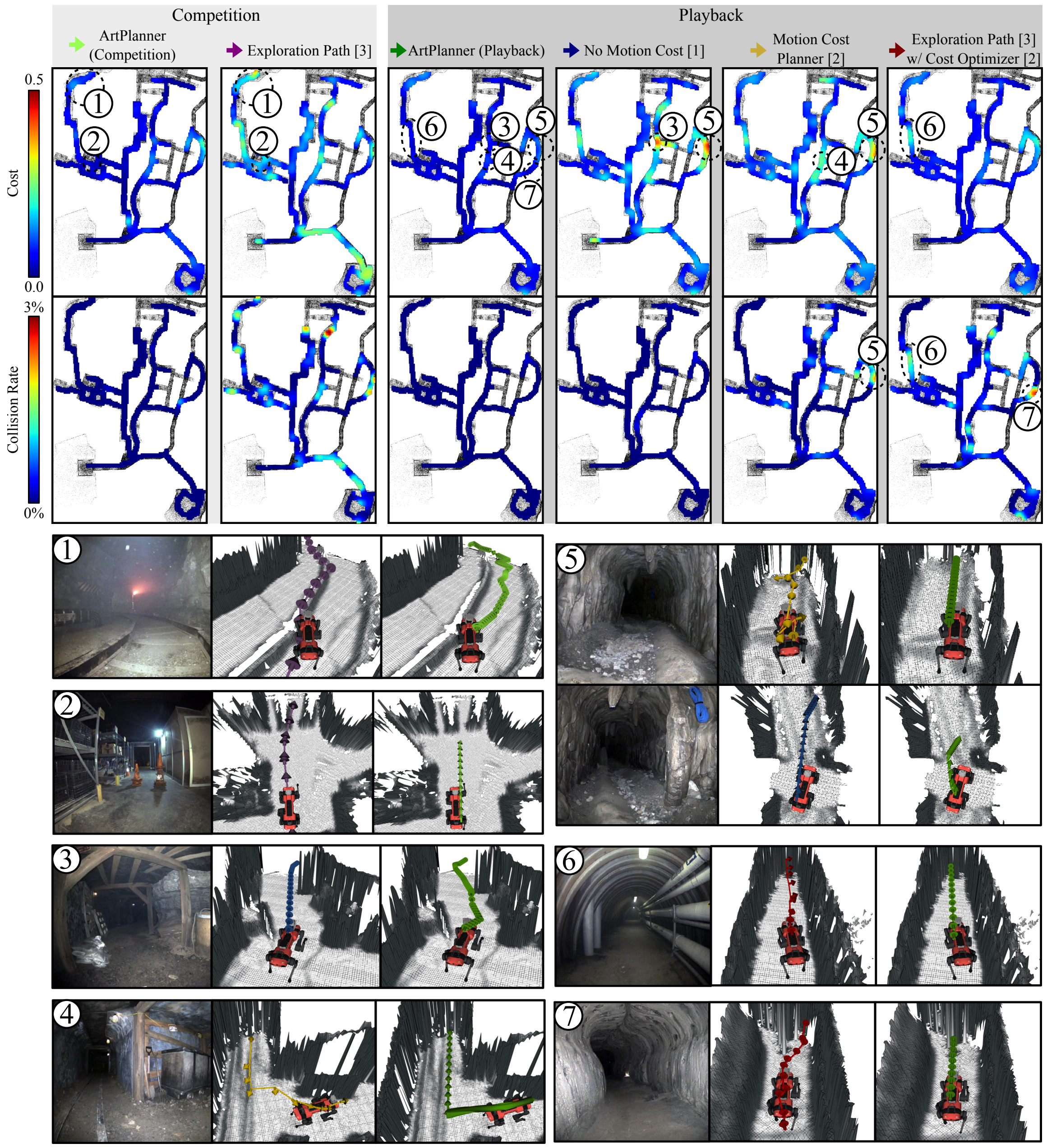


Figure 7. Comparison of path costs and path collision rate for different planners during the SubT Finals Prize Run. The two leftmost columns show data from the planners as run on the robots during the competition, while the other columns show real robot data played back. Data are smoothed with a Gaussian filter with a standard deviation of 2 m for better visibility. ① The *Exploration Path* was highly risky, whereas ArtPlanner avoids the high rail track. ② The *Exploration Path* was infeasible as it ignored the obstructing traffic cones. ③ With *No Motion Cost* the path was very close to obstacles, whereas ArtPlanner kept a safe distance. ④ Due to a bad initial guess, the *Motion Cost Planner* turned on the train tracks, whereas ArtPlanner turned beforehand and walked straight on the tracks. ⑤ Due to a bad initial guess, the *Motion Cost Planner* performed an unnecessary turn, which caused the path cost optimizer to produce a colliding path in the narrow cave tunnel. ⑥ The initial exploration path was in collision with height map artifacts, leading to suboptimal behavior of the path optimizer. ⑦ In trying to avoid map noise, the path optimizer pushed the path into the tunnel wall.
[1] (Wellhausen and Hutter, 2021) [2] (Yang et al., 2021a) [3] (Kulkarni et al., 2022)

estimation, localization, and height map data recorded during the *Finals* and input the exploration paths commanded during the *Finals* to all planners. This allows us to somewhat reproduce the behavior of the first two components of the navigation stack, as shown in Figure 3, and therefore evaluate path quality. Note, however, that this means planning will always start from the competition robot pose, which is a result of following our planner. Therefore, planning always starts from a safe state, and other intricacies that can lead to issues (see Section 3.3.2) cannot be reproduced. However, if planners show problematic behavior even in this simplified setup, they are even more likely to fail during real deployment. To have a fair comparison to the other methods, we also reran ArtPlanner on *Finals* data.

We evaluate the following methods:

- *ArtPlanner (Competition)*: Our method as run in the loop, on the robots, during the competition.
- *GBPlanner2*: Exploration planner (Kulkarni et al., 2022) as run during the competition.
- *ArtPlanner (Playback)*: Our method using data from the competition played back.
- *No Motion Cost*: A reachability-only planner (Wellhausen and Hutter, 2021), which we fed the height map processed as proposed in this work.
- *Motion Cost Planner*: Planner based purely on the motion cost network (Yang et al., 2021a). It first computes a raw path, by querying motions in a fixed graph pattern, with heuristics to determine robot orientation. This raw path is then optimized with gradient-based optimization using the motion cost network.
- *Exploration Path w/ Cost Optimizer*: GBPlanner2 (Kulkarni et al., 2022) already outputs a path that is mostly generally feasible, but it optimizes for information gain and plans on a coarser map. We therefore fed the exploration path directly into the motion cost optimizer (Yang et al., 2021a) to obtain a refined navigation path.

3.1.2. Methodology

We used the motion cost network to compute path costs for all path segments inside the height map. The same network architecture and weights were used for evaluation and in the planners which use the cost network themselves.

Since the planners mentioned above have different characteristics, implementation details, and produce paths of different lengths, we took extra considerations to guarantee a fair comparison. We check height map collisions only for the torso of the robot, analogous to the blue boxes shown in Figure 4(a). We also reduced the size of these boxes by 10 cm in all dimensions compared to the size used by the reachability planners. This was done to account for the more accurate collision models used when training the motion cost network. Finally, we disregarded the first pose of all output paths, since it corresponds to the current robot pose. This means that it has to be valid, and any detected collisions would be erroneous. The reachability planners account for this by sampling a valid pose in a small region around the start, but the motion cost planners do not have that functionality. When evaluating the *Exploration Path*, we only consider path poses from the one closest to the current robot pose until the current goal pose of the ArtPlanner (see Section 1.5). This prevents excessive collision rates caused by height map artifacts, which do not appear in the volumetric map used by GBPlanner2. Note that all these concessions disadvantage ArtPlanner, and the comparison would have been even more in our favor without them.

When computing cost and collisions, we used the height map at the time when the paths are published, not the maps that they are planned on. This is a more realistic evaluation, which takes into account effects caused by different planning speeds of the methods. We use the same risk cutoff of 0.5 for all methods and the same cost function mentioned in Section 2.2.

All playback experiments were run on a Desktop computer with an Intel i7-8700K CPU, a Nvidia RTX 2080 GPU, and 32 GB of RAM.

Table 1. The collision rate, motion cost, and motion risk for all compared methods over all robots. For the path collision rate, the left column indicates the percentage of paths where any pose was in collision. The right column shows the percentage of paths where over $\frac{1}{3}$ of all poses were in collision. This serves as an indicator of how many paths would produce severe collisions that could become dangerous for the robot. For motion cost and motion risk, the columns indicate their mean and 95th-percentile.

	Collision Rate [%]		**Motion Cost**		**Motion Risk**	
	Any	**Severe**	**Mean**	**95%**	**Mean**	**95%**
ArtPlanner (Competition)	4.50	0.05	0.22	1.00	0.03	0.17
Exploration Path	10.93	2.44	0.77	4.34	0.12	0.74
ArtPlanner (Playback)	6.26	0.34	0.21	0.96	0.03	0.16
No Motion Cost	5.24	0.16	0.48	2.82	0.08	0.49
Motion Cost Planner	25.44	0.86	0.41	1.92	0.06	0.32
Exploration Path w/ Cost Optimizer	37.17	1.70	0.34	1.44	0.05	0.24

3.1.3. Motion Cost

To analyze the planner performance with respect to motion cost, we aggregate the cost of all individual path segments spatially. We visualize this as a heat map, overlaid over a top-down view of the *Finals* course map, shown in Figure 7. We also show motion cost and motion risk statistics in Table 1.

ArtPlanner consistently output paths with low motion costs in all regions of the map, as shown in Figure 7 ①-⑦. During the competition, ArtPlanner produced paths with slightly higher motion costs in very narrow parts of the course, most likely due to the lower compute budget available onboard the robot. The *Exploration Path w/ Cost Optimizer* generally also performed well in terms of cost and is able to reduce the high motion cost of the raw *Exploration Path* in most cases. There are no hot spots with high costs, and it only produced elevated costs in two cases: in the very narrow cave section, and in one case in which the exploration path intersected map artifacts, which led to suboptimal optimization behavior (Figure 7 ⑥). Interestingly, the full *Motion Cost Planner* produced higher cost paths in more regions than the *Exploration Path w/ Cost Optimizer*, even though it considers the motion cost at all planning stages. This is due to the orientation heuristics in the first planning stage, which can produce raw paths with motions that the gradient-based optimizer cannot fix, like 360-degree turns. This caused the path to rotate on challenging terrain (Figure 7 ④) and in narrow spaces (Figure 7 ⑤). Unsurprisingly, the reachability planner with *No Motion Cost* produced multiple dangerous, high cost hot spots, frequently planning very close to obstacles (Figure 7 ③+⑤) and over rough terrain.

3.1.4. Collision Rate

Table 1 shows the collision rates computed for each planner. In addition to computing the general collision rate of each planner, which is the ratio of paths that have any collision, we also computed a severe collision rate. We did this to diminish the effect of slight path collisions, which likely would not be problematic for the robot, or are caused by map artifacts or state estimation drift. A path is in severe collision if more than $\frac{1}{3}$ of its poses are in collision. For this, we considered only paths with at least three poses to avoid the outsized impact short paths would have on this metric.

All reachability planners performed at safe levels with respect to path collisions. Interestingly, ArtPlanner deployed during the competition performed even better than during playback. This is most likely related to differences in message timing between recorded competition data and playback, but not the actual underlying performance. Note that collision rates are not zero because we checked collisions against height maps at the time paths are published. This means that map changes, artifacts, and state estimation drift can lead to some map collisions which are unproblematic during operation. As mentioned before, we only had a single real collision during the event, and in that case the obstacle was missing from the map (see Section 3.3.2). The *Exploration Path* only has a slightly higher general collision rate at 10.93% but the highest rate of severe collisions at

2.44%. This is due to the fact that the exploration planner performs collision checking in a coarser, volumetric map. This map captures most larger obstacles, which leads to a low general collision rate. Small obstacles, however, can be completely missed, which leads to severe collisions in these instances. *Motion Cost Planner* and *Exploration Path w/ Cost Optimizer* have very high general collision rates at 25.44% and 37.17%, respectively, but comparatively much lower severe collision rates, at 0.86% and 1.70%. This is due to the fact that, when training the cost network, collisions with the environment are allowed, as long as the robot still reaches its intended goal. This means paths frequently slightly graze some obstacles without severely colliding. While mostly not fatal, in practice even slight bumps into obstacles should be avoided since they can get the robot stuck on protruding elements or damage its payload. The higher severe collision rate of the *Exploration Path w/ Cost Optimizer* was caused by the colliding initial exploration path, which the gradient-based cost optimizer could not fix.

Taking a look at the collision heat maps in Figure 7 allows us to determine in which situations these planners struggled. Figure 7 ② shows an instance in which the *Exploration Path* passes through obstructing traffic cones. Note that this does not show up as a severe hot spot on the heat map due to *Exploration Path* pruning for collision checking, as discussed in Section 3.1.2. The *Motion Cost Planner* generally had few collisions, except in a narrow cave section, where the optimized path performs a turn due to bad initialization (Figure 7 ⑤). The *Exploration Path w/ Cost Optimizer* collisions were generally caused by the optimizer not dealing well with height map artifacts (Figure 7 ⑥+⑦).

3.1.5. Planning Time

Since we did not log planning times of our planner during the competition, and GBPlanner2 has a different scope from the other navigation planners, we compare the planning times of all methods during playback. Planning times are shown as box plots in Figure 8. Our chosen target update rate was to publish a new path every 2 seconds, and the real-time threshold for our $8\,\mathrm{m} \times 8\,\mathrm{m}$ map at a locomotion speed of $0.9\,\mathrm{m\,s^{-1}}$ was 4.44 s. The real-time threshold (Wellhausen and Hutter, 2021) is the time the robot requires to reach the edge of the height map at maximal speed.

We set ArtPlanner's maximal sampling time T to 2 seconds, which does not factor in map processing and motion cost query time. Therefore, ArtPlanner can exceed the target time if the maximum sampling time is reached, with a maximal planning time of 4.65 s. Consequently, we achieved the target time in 75% of cases, exceeding the real-time threshold only a single time, by

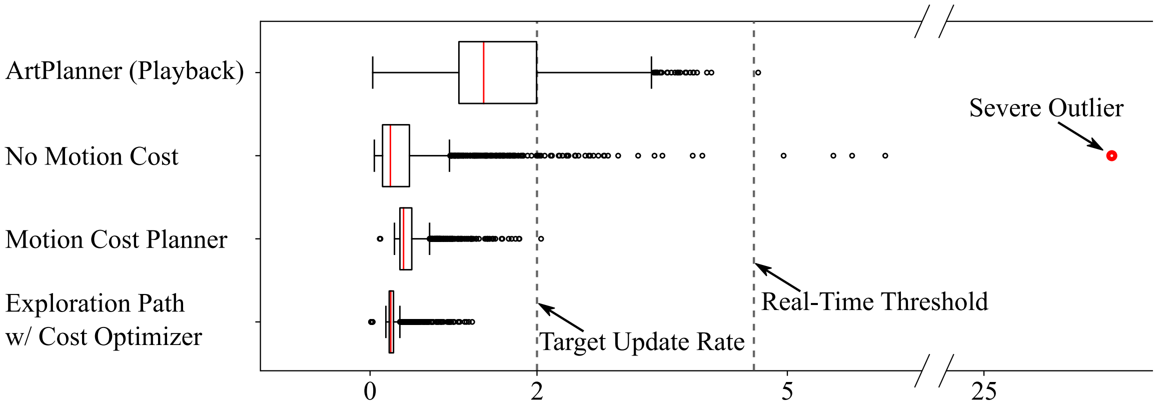


Figure 8. While ArtPlanner plans are slower than the compared methods on average, its computation time is bounded. This avoids severe planning time outliers that the *No Motion Cost* planner produces in rare cases. The *Motion Cost Planner* is fast because it checks a fixed planning graph pattern and therefore does not perform any pose sampling. The *Cost Optimizer* only does a few steps of gradient-descent, which makes it the fastest method in our comparison.

0.21 s. The *No Motion Cost* planner has fast query times in the median. Note that these times do not include the graph sampling time, because this method uses a persistent graph that is built over time, in-between planning queries. If we added this sampling time, the median performance would be comparable to ArtPlanner. Unfortunately, this method can produce severe planning time outliers on rare occasions due to the graph validation at query time, as discussed in Section 2.3. We observed a time of 26.53 s in the data we used to generate Figure 8 but even longer times in other instances. This shows the benefit of our graph validation method through batch motion cost query. We can limit our planning time and can therefore avoid the planner being unresponsive for a long period of time. The *Motion Cost Planner* uses a fixed planning graph and therefore only has to perform a batch motion cost query, but no sampling. Since the optimization stage also always performs a fixed number of iterations, planning is fast and the target time can be reached in all cases. Since the *Exploration Path w/ Cost Optimizer* only deploys the optimization stage of the *Motion Cost Planner*, it is even faster.

Overall, all evaluated planners would be fast enough for real-time operation in the nominal case. However, the worst-case time of the *No Motion Cost* planner is too long to deploy it in a competition environment.

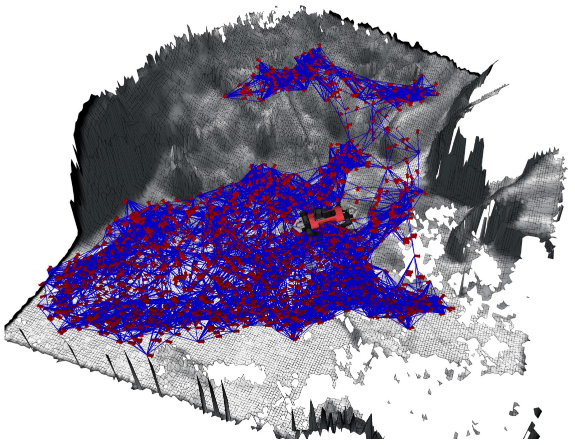
3.2. Component Analysis

We now investigate the effect individual components of our method had on the performance during the *Finals*.

3.2.1. Motion Cost Analysis

We used the motion cost in two ways: to prune the planning graph based on motion risk, and to optimize the cost function for both risk and time.

Since the terrain during the *Finals* was generally flat and easy to traverse for our locomotion controller (Miki et al., 2022a) in most sections, risk pruning did not have a large effect. We observed the biggest effect in the large hall of the Cave section, which housed a rocky incline (bottom right of the map in Figure 7). Since the inclination of the terrain was in principle low enough to climb it, GBPlanner2 wanted to explore up the slope, even though the actual terrain was too rough and rocky to overcome. Using just reachability checking, our sampler found individual valid poses on the slope and connected them, as shown in Figure 9(a). Here, risk pruning identified that moving on


(a) Without risk pruning

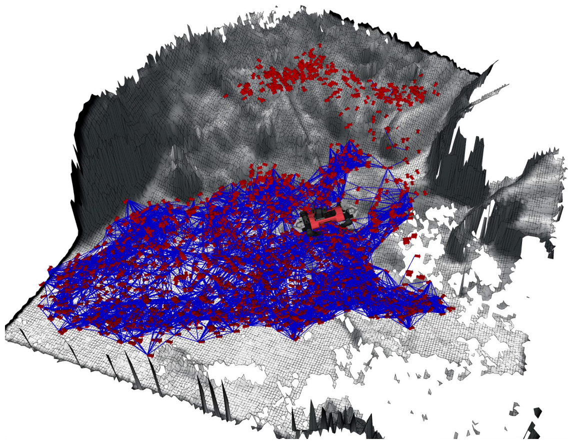

(b) With risk pruning

Figure 9. Sampled valid poses shown in red with blue graph edges connecting them. Using reachability checking, valid robot poses are still found on a steep and rocky incline in the cave section. Pruning graph edges based on motion risk prevents the planning graph from spanning this risky area.

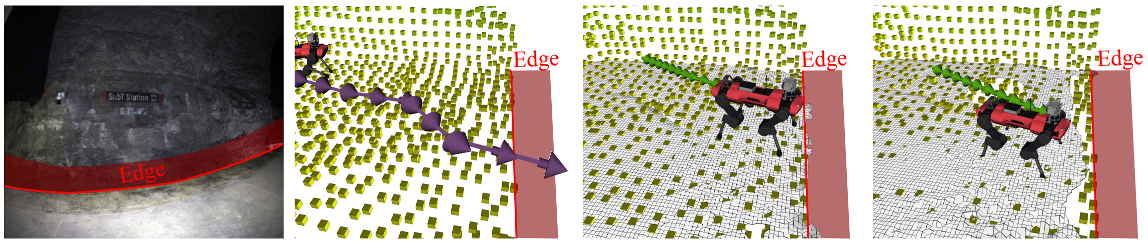


Figure 10. (a) The SubT Station platform had a sharp edge with a significant drop. (b) GBPlanner2 was tuned to be optimistic and planned over the edge of the SubT Station platform. (c) Without a foothold safety margin, the robot would have stepped onto and possibly over the platform edge. (d) With a foothold safety margin, the final path pose is a safe distance from the platform edge. Yellow cubes represent our lidar map to indicate the actual location of the edge. The ANYmal model in (c)+(d) does not indicate the current robot pose, but rather is placed at the final pose of the path to better show how close the robot would have stepped to the edge.

this slope was too risky and removed these edges, as shown in Figure 9(b). Without risk pruning, the robot would have tried to scale this slope, which would in the best case have led to lost time, and in the worst case to loss of this robot.

Figure 7 ③+⑤ shows that the cost function generally led to safer paths, which kept a safe distance from obstacles. This can be attributed to the risk term c_r in the cost function. The time cost c_t had a negligible impact compared to the shortest path of the *No Motion Cost* planner, since the terrain present during the *Finals* was uniform enough such that the shortest path generally was also the fastest. However, c_t was necessary to condition the planning problem, since a pure risk cost would have led to large detours to achieve minor risk improvements.

3.2.2. Safety Threshold Analysis

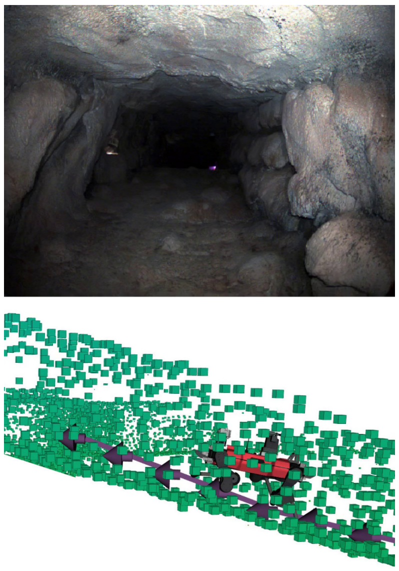
The safety threshold was introduced to handle negative obstacles. One carrier robot reached the Subway Station in autonomous exploration mode during the first Preliminary Run of the *Finals*. The Subway Station had a sharp drop with a wall a few meters behind, pictured in Figure 10(a). As discussed in Section 1.5, the exploration planner was tuned to be optimistic, and planned to explore into the free space above the train tracks [Figure 10(b)]. With the wall visible behind the platform, both image inpainting and virtual surfaces would have simply created flat ground or a gentle slope such that we could not rely on motion cost for safety. Reachability checking generally prevents the planner from planning over the edge; however, without a safety threshold, the planned final pose is dangerously close to the edge [Figure 10(c)]. With the safety threshold applied, the robot only plans to a safe distance from the edge [Figure 10(d)].

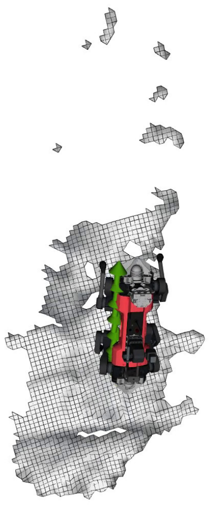
The safety threshold therefore most likely prevented a severe fall of the robot during the first Preliminary Run, which would have caused heavy damage to the payload on top of the robot and possibly the robot itself.

The safety margin parameters were well tuned and generally did not cause the robot to be overly cautious. We only observed the safety margin to come into effect around the tall railroad tracks shown in Figure 7 ①. These were just on the edge of being traversable and tall enough to cause occlusions in the height map. This affected a few unsmooth but still safe paths when the exploration path crossed these tracks, but mostly caused our planner to prefer staying in between the rail tracks, which was the safest approach anyway.

3.2.3. Virtual Surfaces Analysis

As discussed in Section 2.4.1, we only use virtual surfaces above sensor height, due to safety concerns with negative obstacles, demonstrated in Section 3.2.2. Since the *Finals* course was mostly planar, and did not have multiple levels like the Urban Circuit, the only time virtual surfaces came into effect were in the cave section (located just to the left of Figure 7 ⑤). Here, the course had a two-sided


(a) Onboard image and Lidar map.


(b) Steppable region with height map only.

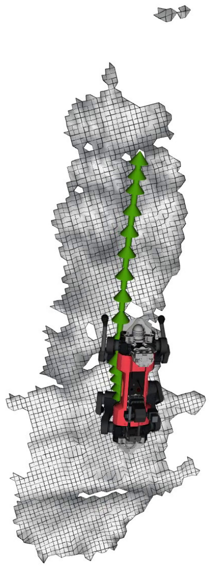

(c) Steppable region with virtual surfaces.

Figure 11. (a) The cave section had a narrow incline leading up to a Cube artifact. (b) Due to the low sensor height above the uneven ground, the height map is too sparse to plan uphill. (c) Using virtual surfaces, the plannable area is vastly increased.

ramp with a very low ceiling height, which led to a platform with a Cube artifact, pictured in Figure 11(a). Due to the rough terrain on the incline, the height map contained many holes in front of the robot. This reduced the size of the steppable area so much that planning would have become almost impossible, as shown in Figure 11(b). Here, Figure 11(c) shows how the virtual surfaces allowed us to plan up the incline, reach the platform, and score the cube artifact.

3.3. Issues

Although our planner performed very well during the *Finals*, it still encountered some issues related to other parts of the navigation stack.

3.3.1. Height Map Spikes

As mentioned in Section 2.4.3, obtaining a clean height map in environments with low ceilings was challenging, and we tuned our ceiling point filter to also work with inclines and stairs, which exacerbated the issue. This slowed our progress quite a bit in the cave section, where the ceiling was especially low. Unfortunately, we reached these sections with our explorer robots, which recorded many ceiling points very close to the robot, due to their dome lidar configuration [see Figure 2(a) and 2(b)].

Although this slowed our progress, we never got stuck. We demonstrate this behavior with a narrow cave opening, which is immediately followed by an incline, shown in Figure 12. Even when the robot was less than a meter from the opening (Figure 12 ①), the height map showed a straight wall. Our planner could therefore only plan right up to the hallucinated wall, which then shifts forward slightly (Figure 12 ②) and allows the robot to plan a bit farther. In Figure 12 ③, the slope aligns with the distance-dependent height threshold of our ceiling point filter, and we can plan forward farther. Once we are fully on the incline (Figure 12 ④), the negative slope at the rear of the robot causes the fake wall to reappear right behind it. Nonetheless, the robot was able to plan up the slope and back down fully autonomously.

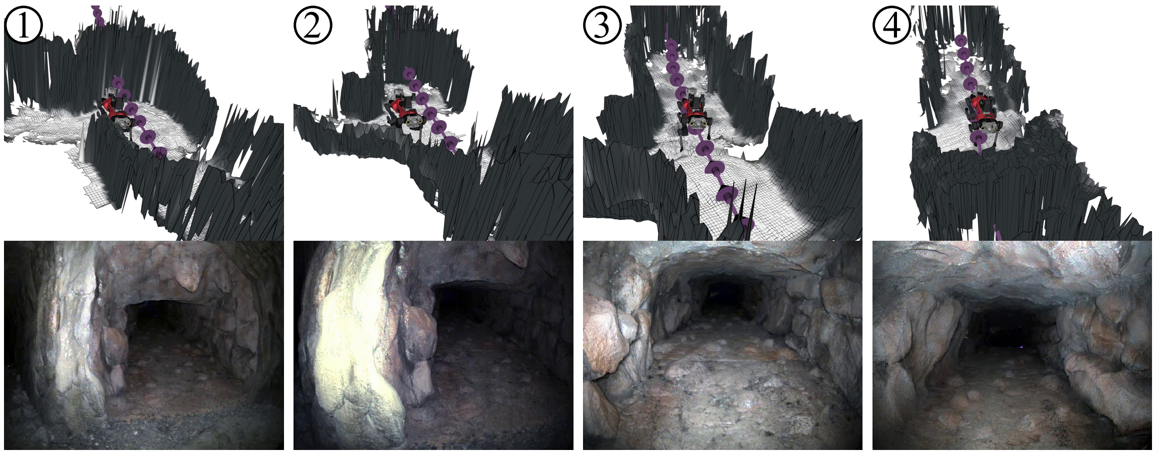


Figure 12. Corrupted height maps from ceiling lidar returns frequently slowed down progress by reducing the distance the planner can plan ahead. ① The exploration path leads into a low corridor which is observed as a wall in the height map. ② The fake wall shifts forward as the robot gets closer. ③ The incline reduces ceiling hits in front of the robot, which helps move the fake wall farther from the robot. ④ Once fully on the slope, ceiling measurements immediately behind the robot produce another fake wall.

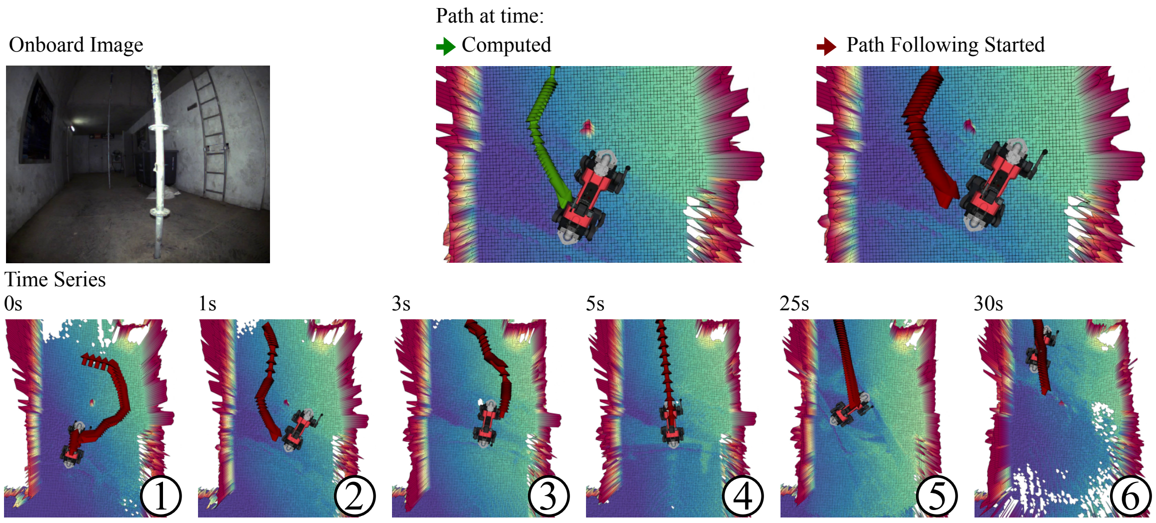


Figure 13. A 500 ms time delay between computing a path and starting to follow it led to imperfect path following. As a result, one robot briefly got stuck on a narrow scaffolding pole in the urban section. ① The robot initially planned to pass the pole on the right but ② replanned to use the shorter path on the left. ③ Since the robot already moved, the next path went right again, which got the robot stuck on the pole. ④ The pole got removed from the map as it was now too close to be perceived and, consequently, paths through the pole were planned until ⑤ the robot drifted to the left and ⑥ finally got unstuck.

3.3.2. Path Follower

Planners rely on their paths being tracked accurately. The path follower is therefore an important component of the navigation stack. Our pure-pursuit path follower generally performed well, tracking paths precisely in narrow spaces and moving swiftly in more open spaces. However, during data analysis for this work, we discovered that there was a nonconstant delay between the planner publishing a path and the path follower starting to track it, which we observed to be up to 500 ms. The top right of Figure 13 shows an example of the difference in robot position between the plan being published and that being followed. This, in combination with height map issues and the unsmooth nature of sampling-based planning, caused one explorer to get stuck on a scaffolding pole in the Urban section for about 20 seconds. The time series of this event is shown at the bottom of Figure 13:

① When the robot approaches the pole, the initial plan is to circumnavigate it on the right, which is slightly longer than on the left. ② Just as the robot starts moving, a new path is published, which passes the pole on the left. ③ This causes the robot to move so close to the pole that it cannot be perceived and disappears from the height map. Since the robot had followed the original path to the right, the next path goes right again and gets the robot stuck on the pole. ④ Because the pole is now missing from the height map, all new paths go straight through it. ⑤ After another 20 seconds, the robot has drifted to the left of the pole and ⑥ finally gets unstuck.

This event demonstrates the complexity induced by interactions of the many components that make up a robotic system. While every path planned before the pole disappeared from the map was safe, this was no guarantee that the actual robot motion was safe. In the end, the robot only continued due to our exceptionally robust locomotion controller (Miki et al., 2022a).

4. Conclusion

In this work we presented ArtPlanner, a navigation planner for legged robots, which was deployed by team CERBERUS during the DARPA Subterranean Challenge (SubT) *Finals* to win the competition. Its reachability robot abstraction allows for precise maneuvering in tight spaces, while its learned motion cost makes sure the path is safe and cost-optimal. We presented extensive real-world evaluations of the planner performance during the SubT *Finals* and compared to other state-of-the-art navigation planners. We showed that ArtPlanner outperforms the other methods and that without ArtPlanner we likely would have encountered navigation issues during the competition. Our planner never failed to plan and never produced an infeasible or unsafe path, and all deployed ground robots remained operational throughout the competition.

4.1. Limitations

We build a new planning graph every time the planning map is updated and only allow for a limited sampling budget. This means that the algorithm proposed in this work is not probabilistically complete. This is a limitation of this work, but it has shown to be unproblematic in practice, since ArtPlanner runs fast enough to sample its fixed-size map densely, while maintaining real-time update rates. Batch risk querying for edge validation is compatible with our previously used persistent graph (Wellhausen and Hutter, 2021), which is probabilistically complete. However, the error-prone and potentially expensive accounting for updated graph edges would still be necessary. We found this tradeoff of theoretical guarantees for practical reliability to be a good choice for our use-case.

4.2. Future Work

The main issues encountered during the SubT *Finals* were related to the 2.5D height map representation and the interaction with the path follower. Moving to a full 3D representation would be desirable; however, until now these methods could not run in real-time with the level of detail necessary for legged robot navigation. Increased computational power of edge computing devices and advanced GPU-accelerated algorithms could soon enable real-time mapping at necessary resolutions on mobile robots. Another direction for research is to make the exploration planner itself robot-traversability aware. Recently proposed 3D motion cost networks (Frey et al., 2022) could prevent instances where the exploration planner proposes a path that the robot cannot follow. Finally, reinforcement learning (RL) has the potential to remove the limitations inherently produced by any map representation and can at the same time replace the path follower. Recent results in sim-to-real training of RL-policies show the potential for training a navigation agent in simulation, which would allow us to go directly from sensor measurements to locomotion command.

Acknowledgments

We would like to thank the following people for their help in the context of this work. Samuel Zimmermann for writing the code, which interfaced our planner with the rest of the ANYmal software stack; Samuel Zimmermann, Marco Tranzatto, Gabriel Waibel, and Timon Homberger for frequent testing in the run-up to the *Finals*; Takahiro Miki for implementing the ceiling point filter; and Marko Bjelonic for providing feedback on the manuscript. We thank the entire team CERBERUS for exceptional performance over the four years of the competition.

ORCID

Lorenz Wellhausen https://orcid.org/0000-0001-5148-754X
Marco Hutter https://orcid.org/0000-0002-4285-4990

References

Barnes, D., Maddern, W., and Posner, I. (2017). Find your own way: Weakly-supervised segmentation of path proposals for urban autonomy. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 203–210. IEEE.

Bayer, J., and Faigl, J. (2019). On autonomous spatial exploration with small hexapod walking robot using tracking camera intel realsense t265. In *2019 European Conference on Mobile Robots (ECMR)*, pages 1–6.

Bayer, J., and Faigl, J. (2020). Speeded up elevation map for exploration of large-scale subterranean environments. In Mazal, J., Fagiolini, A., and Vasik, P., editors, *Modelling and Simulation for Autonomous Systems*, pages 190–202. Springer International Publishing.

Belter, D., Wietrzykowski, J., and Skrzypczyński, P. (2019). Employing natural terrain semantics in motion planning for a multi-legged robot. *Journal of Intelligent & Robotic Systems*, 93(3-4), 723–743.

Borges, P., Peynot, T., Liang, S., Arain, B., Wildie, M., Minareci, M., Lichman, S., Samvedi, G., Sa, I., Hudson, N., et al. (2022). A survey on terrain traversability analysis for autonomous ground vehicles: Methods, sensors, and challenges. *Field Robotics*, 2(1), 1567–1627.

Bradley, D. M., Chang, J. K., Silver, D., Powers, M., Herman, H., Rander, P., and Stentz, A. (2015). Scene understanding for a high-mobility walking robot. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1144–1151. IEEE.

Chao, C., Hongbiao, Z., Howie, C., and Ji, Z. (2021). Tare: A hierarchical framework for efficiently exploring complex 3d environments. In *Robotics: Science and Systems XVII*, Virtual. Robotics: Science and Systems Foundation.

Chavez-Garcia, R. O., Guzzi, J., Gambardella, L. M., and Giusti, A. (2017). Image classification for ground traversability estimation in robotics. In *International Conference on Advanced Concepts for Intelligent Vision Systems*, pages 325–336. Springer.

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

Fan, D. D., Otsu, K., Kubo, Y., Dixit, A., Burdick, J., and Agha-Mohammadi, A.-A. (2021). Step: Stochastic traversability evaluation and planning for risk-aware off-road navigation. *Robotics: Science and Systems XVII.*, pp. 1–21.

Fernbach, P., Tonneau, S., Del Prete, A., and Taïx, M. (2017). A kinodynamic steering-method for legged multi-contact locomotion. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3701–3707. IEEE.

Frey, J., Hoeller, D., Khattak, S., and Hutter, M. (2022). Locomotion policy guided traversability learning using volumetric representations of complex environments. *arXiv:2203.15854*.

Guzzi, J., Chavez-Garcia, R. O., Nava, M., Gambardella, L. M., and Giusti, A. (2020). Path planning with local motion estimations. *IEEE Robotics and Automation Letters*, 5(2), 2586–2593.

Harper, M. Y., Nicholson, J. V., Collins, E. G., Pusey, J., and Clark, J. E. (2019). Energy efficient navigation for running legged robots. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6770–6776. IEEE.

Hauser, K. (2015). Lazy collision checking in asymptotically-optimal motion planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2951–2957. IEEE.

Hines, T., Stepanas, K., Talbot, F., Sa, I., Lewis, J., Hernandez, E., Kottege, N., and Hudson, N. (2021). Virtual surfaces and attitude aware planning and behaviours for negative obstacle navigation. *IEEE Robotics and Automation Letters*, 6(2), 4048–4055.

Hirose, N., Sadeghian, A., Vázquez, M., Goebel, P., and Savarese, S. (2018). Gonet: A semi-supervised deep learning approach for traversability estimation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3044–3051. IEEE.

Hutter, M., Gehring, C., Jud, D., Lauber, A., Bellicoso, C. D., Tsounis, V., Hwangbo, J., Bodie, K., Fankhauser, P., Bloesch, M., et al. (2016). Anymal-a highly mobile and dynamic quadrupedal robot. In *IROS*, pages 38–44. IEEE.

Kim, D., Sun, J., Oh, S. M., Rehg, J. M., and Bobick, A. F. (2006). Traversability classification using unsupervised on-line visual learning for outdoor robot navigation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 518–525. IEEE.

Krüsi, P., Furgale, P., Bosse, M., and Siegwart, R. (2017). Driving on point clouds: Motion planning, trajectory optimization, and terrain assessment in generic nonplanar environments. *Journal of Field Robotics*, 34(5), 940–984.

Kulkarni, M., Dharmadhikari, M., Tranzatto, M., Zimmermann, S., Reijgwart, V., De Petris, P., Nguyen, H., Khedekar, N., Papachristos, C., Ott, L., Siegwart, R., Hutter, M., and Alexis, K. (2022). Autonomous teamed exploration of subterranean environments using legged and aerial robots. In *IEEE International Conference on Robotics and Automation (ICRA)*.

Lin, Y.-C., and Berenson, D. (2017). Humanoid navigation in uneven terrain using learned estimates of traversability. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 9–16. IEEE.

Miki, T., Lee, J., Hwangbo, J., Wellhausen, L., Koltun, V., and Hutter, M. (2022a). Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62), eabk2822.

Miki, T., Wellhausen, L., Grandia, R., Jenelten, F., Homberger, T., and Hutter, M. (2022b). Elevation mapping for locomotion and navigation using gpu. *arXiv:2204.12876*.

Norby, J., and Johnson, A. M. (2020). Fast global motion planning for dynamic legged robots. *2020 IEEE International Conference on Intelligent Robots and Systems (IROS)*.

Ohradzansky, M. T., Rush, E. R., Riley, D. G., Mills, A. B., Ahmad, S., McGuire, S., Biggie, H., Harlow, K., Miles, M. J., Frew, E. W., Heckman, C., and Humbert, J. S. (2021). Multi-agent autonomy: Advancements and challenges in subterranean exploration. *arXiv:2110.04390*.

Oleynikova, H., Taylor, Z., Fehr, M., Siegwart, R., and Nieto, J. (2017). Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1366–1373. IEEE.

Otsu, K., Ono, M., Fuchs, T. J., Baldwin, I., and Kubota, T. (2016). Autonomous terrain classification with co-and self-training approach. *IEEE Robotics and Automation Letters*, 1(2), 814–819.

Reid, W., Fitch, R., Göktoğan, A. H., and Sukkarieh, S. (2020). Sampling-based hierarchical motion planning for a reconfigurable wheel-on-leg planetary analogue exploration rover. *Journal of Field Robotics*, 37(5), 786–811.

Rothrock, B., Kennedy, R., Cunningham, C., Papon, J., Heverly, M., and Ono, M. (2016). Spoc: Deep learning-based terrain classification for mars rover missions. In *AIAA SPACE 2016*, page 5539. AIAA.

Thakker, R., Alatur, N., Fan, D. D., Tordesillas, J., Paton, M., Otsu, K., Toupet, O., and Agha-mohammadi, A.-a. (2021). Autonomous off-road navigation over extreme terrains with perceptually-challenging conditions. In *Experimental Robotics*, pages 161–173, Cham. Springer International Publishing.

Tidd, B., Cosgun, A., Leitner, J., and Hudson, N. (2021). Passing through narrow gaps with deep reinforcement learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3492–3498. IEEE.

Tonneau, S., Del Prete, A., Pettré, J., Park, C., Manocha, D., and Mansard, N. (2018). An efficient acyclic contact planner for multiped robots. *IEEE Transactions on Robotics*, 34(3), 586–601.

Tonneau, S., Mansard, N., Park, C., Manocha, D., Multon, F., and Pettré, J. (2015). A reachability-based planner for sequences of acyclic contacts in cluttered environments. In *International Symposium on Robotics Research (ISSR 2015)*.

Tranzatto, M., Dharmadhikari, M., Bernreiter, L., Camurri, M., Khattak, S., Mascarich, F., and Alexis, K. (2022a). Team cerberus wins the darpa subterranean challenge: Technical overview and lessons learned. *submitted to Field Robotics*.

Tranzatto, M., Mascarich, F., Bernreiter, L., Godinho, C., Camurri, M., Khattak, S., Dang, T., Reijgwart, V., Loeje, J., Wisth, D., et al. (2022b). Cerberus: Autonomous legged and aerial robotic exploration in the tunnel and urban circuits of the darpa subterranean challenge. *Field Robotics*, 2, 274–324.

Valada, A., Vertens, J., Dhall, A., and Burgard, W. (2017). Adapnet: Adaptive semantic segmentation in adverse environmental conditions. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4644–4651. IEEE.

Wellhausen, L., Dosovitskiy, A., Ranftl, R., Walas, K., Cadena, C., and Hutter, M. (2019). Where should i walk? predicting terrain properties from images via self-supervised learning. *IEEE Robotics and Automation Letters*, 4(2), 1509–1516.

Wellhausen, L., and Hutter, M. (2021). Rough terrain navigation for legged robots using reachability planning and template learning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6914–6921. IEEE.

Wellhausen, L., Ranftl, R., and Hutter, M. (2020). Safe robot navigation via multi-modal anomaly detection. *IEEE Robotics and Automation Letters*, 5(2), 1326–1333.

Wermelinger, M., Fankhauser, P., Diethelm, R., Krüsi, P., Siegwart, R., and Hutter, M. (2016). Navigation planning for legged robots in challenging terrain. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1184–1189. IEEE.

Winkler, A. W., Bellicoso, C. D., Hutter, M., and Buchli, J. (2018). Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robotics and Automation Letters*, 3(3), 1560–1567.

Yang, B., Wellhausen, L., Miki, T., Liu, M., and Hutter, M. (2021a). Real-time optimal navigation planning using learned motion costs. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE.

Yang, F., Cao, C., Zhu, H., Oh, J., and Zhang, J. (2021b). FAR planner: Fast, attemptable route planner using dynamic visibility update.

Zhao, Y., Chai, X., Gao, F., and Qi, C. (2018). Obstacle avoidance and motion planning scheme for a hexapod robot octopus-iii. *Robotics and Autonomous Systems*, 103, 199–212.

How to cite this article: Wellhausen, L., & Hutter, M. (2023). ArtPlanner: Robust legged robot navigation in the field. *Field Robotics, 3*, 413–434.