Regular Article

STEP: Stochastic Traversability Evaluation and Planning for Risk-Aware Off-road Navigation; Results from the DARPA Subterranean Challenge

Anushri Dixit*,1, David D. Fan*,2, Kyohei Otsu2, Sharmita Dey2, Ali-Akbar Agha-Mohammadi2 and Joel Burdick1

1California Institute of Technology
2NASA Jet Propulsion Laboratory, California Institute of Technology

Abstract: Although autonomy has gained widespread usage in structured and controlled environments, robotic autonomy in unknown and off-road terrain remains a difficult problem. Extreme, off-road, and unstructured environments such as undeveloped wilderness, caves, rubble, and other post-disaster sites pose unique and challenging problems for autonomous navigation. Based on our participation in the DARPA Subterranean Challenge, we propose an approach to improve autonomous traversal of robots in subterranean environments that are perceptually degraded and completely unknown through a traversability and planning framework called STEP (Stochastic Traversability Evaluation and Planning). We present 1) rapid uncertainty-aware mapping and traversability evaluation, 2) tail risk assessment using the Conditional Value-at-Risk (CVaR), 3) efficient risk and constraint-aware kinodynamic motion planning using sequential quadratic programming-based (SQP) model predictive control (MPC), 4) fast recovery behaviors to account for unexpected scenarios that may cause failure, and 5) risk-based gait adaptation for quadrupedal robots. We illustrate and validate extensive results from our experiments on wheeled and legged robotic platforms in field studies at the Valentine Cave, CA (cave environment), Kentucky Underground, KY (mine environment), and Louisville Mega Cavern, KY (final competition site for the DARPA Subterranean Challenge with tunnel, urban, and cave environments).

Keywords: subterranean robotics, control, motion planning, extreme environments

1. Introduction

Robust autonomous traversal over extreme, hazardous terrain is an open problem with many applications to extra-terrestrial (Ono et al., 2015), disaster-struck (Ali-akbar Agha-mohammadi et al., 2022), and subterranean environments (Kalita et al., 2018). The robots operate in uneven
and highly risky terrain based on the use of noisy sensor measurements and localization uncertainty. Autonomous motion planning in such conditions requires a framework that can account for the traversability risk arising from sources like rubble, sudden drops, muddy/slippery areas, while also considering the uncertainty in the robots estimates. The framework must be able to make tractable reformulations of the nonconvex constraints arising from these risks so that it can plan reactive motions in real time.

In this work, our evaluation of a terrain’s traversability accounts for the hazards from the uneven terrain (geometric, confidence-aware analysis) and type of terrain (semantic analysis), see Figure 1. We will treat this traversability cost as a random variable and account for the uncertainty in the traversability cost using Conditional Value-at-Risk (CVaR). Using the CVaR assessment of the terrain, we obtain a geometric path and a low-level model predictive control (MPC) plan that accounts for the risk in the terrain.

Most traversability analyses are dependent on the types of sensors used to obtain information of the terrain, and they are based on geometry, appearance, or proprioceptive methods (Papadakis, 2013). Geometry-based methods often rely on building a 2.5D terrain map that is used to extract features such as maximum, minimum, and variance of the terrain height and slope (Goldberg et al., 2002). Planning algorithms for such methods take into account the stability of the robot on the terrain (Hait et al., 2002). In (Otsu et al., 2020; Ghosh et al., 2018), the authors estimate the probability distributions of states based on the kinematic model of the vehicle and the terrain height uncertainty. Furthermore, a method for incorporating sensor and state uncertainty to obtain a probabilistic terrain estimate in the form of a grid-based elevation map is considered in (Fankhauser et al., 2018). Learning-based traversability frameworks are becoming increasingly popular. In (Gan et al., 2022), the authors propose an inverse reinforcement learning-based method for learning the traversability cost from raw exteroceptive and proprioceptive sensor measurements. In (Cai et al., 2022), the authors learn a speed-map representation given semantic information, while in (Manderson et al., 2020), the authors use a self-supervised learning-based terrain roughness classifier. In (Fan et al., 2022), the authors propose a neural network architecture for learning the distribution of traversability costs arising from geometric risks using the point cloud obtained from a LiDAR in a supervised manner. In this paper, we build upon the traversability analyses...
of classical geometric methods while using semantic information from a LiDAR and incorporating the uncertainty of these methods for risk-aware planning (Agha-Mohammadi et al., 2017; Thakker et al., 2020; Fan et al., 2021).

Risk can be incorporated into motion planning using a variety of different methods, including chance constraints (Ono et al., 2015; Wang et al., 2020), exponential utility functions (Koenig and Simmons, 1994), distributional robustness (Xu and Mannor, 2010; Wei et al., 2023), and quantile regression (Fan et al., 2020; Dabney et al., 2018). Risk measures, often used in finance and operations research, provide a mapping from a random variable (usually the cost) to a real number. These risk metrics should satisfy certain axioms in order to be well-defined as well as to enable practical use in robotic applications (Majumdar and Pavone, 2020). Conditional value-at-risk (CVaR) is one such risk measure that has this desirable set of properties, and is a part of a class of risk metrics known as coherent risk measures (Artzner et al., 1999). Coherent risk measures have been used in a variety of decision-making problems, especially Markov decision processes (MDPs) (Chow et al., 2015; Ahmadi et al., 2020; Ahmadi et al., 2021b; Ahmadi et al., 2021a). Coherent risk measures have been used in a MPC framework when the system model is uncertain (Singh et al., 2018; Dixit et al., 2022), when the uncertainty is a result of measurement noise or moving obstacles (Hakobyan et al., 2019; Dixit et al., 2021), and when the uncertainty arises from both measurement and process noise (Dixit et al., 2023). Our work extends CVaR risk to a risk-based planning framework that utilizes different sources of traversability risk (such as collision risk, step risk, slippage risk, etc.) Moreover, this paper introduces the first field-hardened and theoretically grounded approach to traversability assessment and risk-constrained planning using CVaR metrics. Using CVaR to assess traversability risks allows us to dynamically tune the entire system’s behavior—from aggressive to highly conservative—by changing a single value, namely the risk level. When the value of the risk level is small, the behavior of the traversability assessment is equivalent to an expectation-based assessment, which is similar to the geometric traversability evaluation in the past literature. Conversely, when the risk level is high, we account for the worst possible value of our terrain assessment given the uncertainty in our measurements.

Model Predictive Control has long been used to robustly control more complex systems, including time-varying, nonlinear, or MIMO systems (Camacho and Alba, 2013). While simple linear PID controllers are sufficient for simpler systems, MPC is well-suited to more complex tasks while being computationally feasible—MPC can account for the nonlinear robot dynamics while making multi-step predictions of the trajectory followed by the robot, and incorporating complex state and control constraints such as CVaR constraints, obstacle avoidance constraints, and nonholonomic constraints. Hence, MPC provides an optimization-based framework to mimic a suboptimal infinite-horizon controller while being computationally tractable. In particular, we employ Sequential Quadratic Programming (SQP), which iteratively solves locally quadratic subproblems to converge to a globally (more) optimal solution (Boggs and Tolle, 1995). This approach reduces computational costs and has the flexibility to handle a wide variety of costs and constraints (Schulman et al., 2014; Lew et al., 2020; Morgan et al., 2014; Augugliaro et al., 2012; Foust et al., 2020). A common criticism of SQP-based MPC methods (and nonlinear MPC methods in general) is that they are susceptible to local minima. We address this problem by incorporating a trajectory library [which can be predefined and/or randomly generated, e.g., as in (Kalakrishnan et al., 2011)] to use in a preliminary trajectory selection process. This approach finds more globally optimal initial guesses for the SQP problem to refine locally. Another common difficulty with risk-constrained nonlinear MPC problems is ensuring recursive feasibility (Löfberg, 2012). We sidestep this problem by dynamically relaxing the severity of the risk constraints while penalizing CVaR in the cost function.

In this work, we propose STEP (Stochastic Traversability Evaluation and Planning), which pushes the boundaries of the state-of-the-practice to enable safe, risk-aware, and high-speed ground traversal of unknown environments. Specifically, our contributions include:

1. Uncertainty-aware 2.5D traversability evaluation, which accounts for localization error, sensor noise, and occlusion, and combines multiple sources of traversability risk.
2. An approach for combining these traversability risks into a unified risk-aware CVaR planning framework.
3. A highly efficient MPC architecture for robustly solving nonconvex risk-constrained optimal control problems.
4. A suite of recovery behaviors to account for fast response to failure scenarios.
5. Risk-based gait adaptation for quadrupedal robots (in our case, the Boston Dynamics Spot platform).
6. Real-world demonstration of real-time CVaR planning on wheeled and legged robotic platforms in unknown and risky environments.

This work significantly extends our previous work (Fan et al., 2021) to incorporate semantic and confidence-aware traversability risk sources in addition to geometric ones to provide a complete suite of traversability analyses using LiDARs. Further, we provide recovery behaviors in our hierarchical planning framework to account for high-risk situations that the geometric and kinodynamic planners fail to account for. Lastly, we also provide risk-aware gait selection techniques for quadrupedal robots. This hierarchical framework is tested extensively in multiple subterranean environments.

2. Risk-Aware Traversability and Planning

2.1. Problem Statement

Let $x_k$, $u_k$, $z_k$ denote the robot’s state, action (or control input), and observation (or sensory measurement) at the $k$th time step. A path $x_{0:N} = \{x_0, x_1, \ldots, x_N\}$ is composed of a sequence of poses. A policy is a mapping from state to control $u_k = \pi(x_k)$. A map is represented as $m = (m^{(1)}, m^{(2)}, \ldots)$, where $m^i$ is the $i$th element of the map (e.g., a cell in a grid map or a feature in a feature-based map). The robot’s dynamics model captures the physical properties of the vehicle’s motion, such as inertia, mass, dimension, shape, and kinematic and control constraints:

$$
\begin{align*}
    x_{k+1} &= f(x_k, u_k) \\
    g(u_k) &> 0 \\
    h(m, x_k) &> 0
\end{align*}
$$

where $g(u_k)$, $h(m, x_k)$ are vector-valued functions that encode the control and state constraints/limits, respectively. The inequalities hold elementwise.

Following (Papadakis, 2013), we define traversability as the ability of a ground vehicle to stably settle over a terrain region under an admissible state that satisfies constraints. We represent traversability as a cost, i.e., a continuous value computed using a terrain model, the robotic vehicle model, and kinematic constraints, which represents the degree to which we wish the robot to avoid a given state:

$$
r = R(m, x, u)
$$

where $r \in \mathbb{R}$, and $R(\cdot)$ is a terrain assessment model. This model captures various unfavorable events, such as collision, getting stuck, tipping over, and high slippage, to name a few. Each mobility platform has its own assessment model to reflect its mobility capability.

Associated with the true traversability value is a distribution over possible values based on the current understanding about the environment and robot actions. In most real-world applications where perception capabilities are limited and noisy, the true value can be highly uncertain. To handle this uncertainty, consider a map belief, i.e., a probability distribution $p(m|x_{0:k}, z_{0:k})$, over a possible set $\mathcal{M}$. Then, the traversability estimate is also represented as a random variable $R : (\mathcal{M} \times \mathcal{X} \times \mathcal{U}) \rightarrow \mathbb{R}$. We call this probabilistic mapping from map belief, state, and controls to possible traversability cost values a terrain assessment model.

A risk metric $\rho(R) : R \rightarrow \mathbb{R}$ is a mapping from a random variable to a real number, which quantifies some notion of risk. To assess the risk of traversing along a path $x_{0:N}$ with a policy $\pi$, we
wish to define the cumulative risk metric associated with the path, \( J(x_0, \pi) \). To do this, we need to evaluate a sequence of random variables \( R_{0:N} \). To quantify the stochastic outcome as a real number, we use the dynamic, time-consistent risk metric given by compounding the one-step risk metrics (Ruszczynski, 2014):

\[
J(x_0, \pi; m) = R_0 + \rho_0(R_1 + \rho_1(R_2 + \cdots + \rho_{N-1}(R_N)))
\]

where \( \rho_k(\cdot) \) is a one-step coherent risk measure at time \( k \). This one-step risk gives us the cost incurred at time-step \( k + 1 \) from the perspective of time step \( k \). Any distortion risk measure compounded as given in Equation 5 is time-consistent [see (Majumdar and Pavone, 2020) for more information on distortion risk metrics and time-consistency]. We use the Conditional Value-at-Risk (CVaR) as the one-step risk metric:

\[
\rho(R) = \text{CVaR}_\alpha(R) = \inf_{z \in \mathbb{R}} \mathbb{E} \left[ z + \frac{(R - z)_+}{1 - \alpha} \right]
\]

where \((\cdot)_+ = \max(\cdot, 0)\), and \( \alpha \in (0, 1) \) denotes the risk probability level. CVaR provides the expected cost of the random variable in the \( 1 - \alpha \) tail of the distribution, hence it is able to account for the low probability, high risk events. We note that the results in this paper are also easily extended to other tail risk measures such as Entropic Value-at-Risk (Ahmadi-Javid, 2012) and total variation distance-based risk (Dixit et al., 2022).

We formulate the objective of the problem as follows: Given the initial robot configuration \( x_S \) and the goal configuration \( x_G \), find an optimal control policy \( \pi^* \) that moves the robot from \( x_S \) to \( x_G \) while 1) minimizing time to traverse, 2) minimizing the cumulative risk metric along the path, and 3) satisfying all kinematic and dynamic constraints. For a quadrupedal robot like the Boston Dynamics Spot robot, the framework must additionally also select the best gait type based on the risk accrued while moving from \( x_S \) to \( x_G \) as a part of the optimal policy \( \pi^* \).

### 2.2. Hierarchical Risk-Aware Planning

We propose a hierarchical approach to address the aforementioned risk-aware motion planning problem by splitting the motion planning problem into geometric and kinodynamic domains. We consider the geometric domain over long horizons, while we solve the kinodynamic problem over a shorter horizon. This is convenient for several reasons: 1) Solving the full constrained CVaR minimization problem over long timescales/horizons becomes intractable in real time. 2) Geometric constraints play a much larger role over long horizons, while kinodynamic constraints play a much larger role over short horizons (to ensure dynamic feasibility at each time step). 3) A good estimate (upper bound) of risk can be obtained by considering position information only. This is done by constructing a position-based traversability model \( R_{pos} \) by marginalizing out non-position-related variables from the terrain assessment model, i.e., if the state \( x = [p_x, p_y, x_{other}]^\top \) consists of position and nonposition variables (e.g., orientation, velocity), then

\[
R_{pos}(m, p_x, p_y) \geq R(m, x, u) \quad \forall x_{other}, u.
\]

**Geometric Planning:** The objective of geometric planning is to search for an optimistic risk-minimizing path, i.e., a path that minimizes an upper bound approximation of the true CVaR value. For efficiency, we limit the search space only to the geometric domain. We search for a sequence of poses \( x_{0:N} \) which ends at \( x_G \) and minimizes the position-only risk metric in Equation 5, which we define as \( J_{pos}(x_{0:N}) \). The path optimization problem can be written as

\[
x_{0:N}^* = \arg \min_{x_{0:N}} \left[ J_{pos}(x_{0:N}) + \lambda \sum_{k=0}^{N-1} \|x_k - x_{k+1}\|^2 \right] \quad \text{s.t.} \quad \phi(m, x_k) > 0
\]

\[4:182–210\]
where the constraints $\phi(\cdot)$ encode position-dependent traversability constraints (e.g., constraining the vehicle to avoid obstacles and prohibit lethal levels of risk) and $\lambda \in \mathbb{R}$ weighs the tradeoff between risk and path length.

**Kinodynamic Planning:** We then solve a kinodynamic planning problem to track the optimal geometric path, minimize the risk metric, and respect kinematic and dynamics constraints. The goal is to find a control policy $\pi^*$ within a local planning horizon $T \leq N$ which tracks the path $X^*_{0:N}$. The optimal policy can be obtained by solving the following optimization problem:

$$
\pi^* = \arg \min_{\pi \in \Pi} \left[ J(x_0, \pi) + \lambda \sum_{k=0}^{T} \|x_k - x_k^*\|^2 \right]
$$

subject to:

$$
x_{k+1} = f(x_k, u_k) \quad (11)
$$

$$
g(u_k) \succ 0 \quad (12)
$$

$$
h(m, x_k) \succ 0 \quad (13)
$$

where the constraints $g(u)$ and $h(m, x_k)$ are vector-valued functions that encode controller limits and state constraints, respectively.

**Recovery from Unfavorable States:** Recovery planning is a particular type of planning problem where the initial state is not safe. For example, the robot might need to start planning from the state where it touches walls with its bumpers, or the state where the body is tilted on top of rubble. The recovery from those unfavorable states involves finding a safe control without violating the safety constraints under smaller margin conditions.

3. **STEP for Unstructured Terrain**

This section discusses how we compute traversability risk and efficiently solve the risk-aware trajectory optimization problem. At a high level, our approach takes the following steps (see Figure 2): 1) Aggregate sensor measurements to create an uncertainty-aware map using some source of localization with uncertainty. 2) Perform ground segmentation to isolate the parts of the map that the robot can potentially traverse. 3) Compute risk and risk uncertainty using geometric properties of the point cloud (optionally, include other sources of risk, e.g., semantic or other sensors). 4) Aggregate these risks to compute a 2.5D CVaR risk map. 5) Solve for an optimistic CVaR minimizing path over long ranges with a geometric path planner. 6) Solve for a kinodynamically feasible trajectory, which minimizes CVaR while staying close to the geometric path and satisfying all constraints.

![Figure 2. Overview of system architecture for STEP. From left to right: Odometry aggregates sensor inputs and relative poses. Next, Risk Map Processing merges these point clouds and creates a multilayer risk map. The map is used by the Geometric Path Planner and the Kinodynamic MPC Planner. An optimal trajectory is found and sent to the Tracking Controller, which produces control inputs to the robot. The robot is also able to react quickly to potentially mission-ending scenarios because the tracking controller overrides the optimal control input when one of the recovery behaviors is activated.](image-url)
3.1. Point Cloud Processing and Mapping

Multisensor Merging: Our point cloud pipeline starts from merging point clouds from different sensors. One robot can have multiple units of the same sensor to increase coverage, or have heterogeneous sensors that produce point clouds using different mechanics (e.g., active LiDAR, RGBD cameras, passive stereo cameras). After applying sensor-specific filters that remove noise or body occlusion, these point clouds are merged using extrinsic calibration information. If the sensors are not time-synchronized, we use odometry to compensate motion offset.

Temporal Point Cloud Merging: The merged point clouds are aggregated over a fixed time window to construct a local point cloud map. We maintain a pose graph on the time window to incorporate history updates in the odometry estimate (e.g., loop closures). Based on the latest pose graph, we reconstruct a full aggregated point cloud. We annotate each point by the time offset to the latest point cloud. This allows us to propagate odometry uncertainty to each point in the fusion phase.

Ground Segmentation: The aggregated point cloud is segmented into ground and obstacle points using 3D point cloud segmentation techniques. We leverage the work in (Himmelsbach et al., 2010) that allows efficient ground segmentation based on line fitting in the cylindrical coordinates. We extended the work to also handle challenges prevalent in subterranean environments, such as low ceiling or negative obstacles. The ground segmentation is critical for elevation mapping in an occluded environment where the ground is not observed by the sensors and the measurements of walls/ceiling make false ground planes.

Elevation Mapping: We construct a 2.5D height map using the segmented ground points. After splitting points to each grid map cell and sorting by time stamps, we apply a Kalman filter to estimate the height of the ground. We set up the filter to place more weight on the recent measurements, which are less affected by the odometry error. Figure 3 shows the visualization of ground height estimation with uncertainty. Note that there is higher level of uncertainty for the areas that have few measurements. These mapping uncertainties are used to adjust confidence in a later traversability estimation process.

3.2. Modeling Assumptions

Among many representation options for rough terrain, we use a 2.5D grid map in this paper for its efficiency in processing and data storage (Fankhauser and Hutter, 2016). The map is represented as a collection of terrain properties (e.g., height, risk) over a uniform grid.
For different vehicles we use different robot dynamics models. For example, for a system that produces longitudinal/lateral velocity and steering (e.g., legged platforms), the state and controls can be specified as

\[
x = [p_x, p_y, p_\theta, v_x, v_y, v_\theta]^\top
\]

\[
u = [a_x, a_y, a_\theta]^\top
\]

while the dynamics \(x_{k+1} = f(x_k, u_k)\) can be written as

\[
\begin{bmatrix}
v_x \cos(p_\theta) - v_y \sin(p_\theta) \\
v_x \sin(p_\theta) + v_y \cos(p_\theta) \\
k_v x + (1 - k) v_\theta \\
a_x \\
a_y \\
a_\theta
\end{bmatrix} = 
\]

\[
\begin{bmatrix}
v_x \cos(p_\theta) - v_y \sin(p_\theta) \\
v_x \sin(p_\theta) + v_y \cos(p_\theta) \\
k_v x + (1 - k) v_\theta \\
a_x \\
a_y \\
a_\theta
\end{bmatrix}.
\]

We let \(k \in [0, 1]\) be a constant that adjusts the amount of turning-in-place the vehicle is permitted. In differential drive or Ackermann steered vehicles, we can remove the lateral velocity component of these dynamics. However, our general approach is applicable to any vehicle dynamics model. (For a differential drive model, see Appendix A.) Note that this dynamics model is not meant to capture complex dynamics of legged mobility, but rather to provide a simplified, conservative model of the robot’s motion at the body placement level. This allows for separating footstep/gait planning from risk-aware planning. This approach is computationally less complex than planning using the full dynamics of the legged platform over complex terrain. Future work will involve more tightly coupling these optimization problems.

### 3.3. Terrain assessment models

The traversability cost is assessed as the combination of multiple risk factors. These factors are designed to capture potential hazards for the target robot in the specific environment (Figure 4^1). Such factors include:

- **Collision**: quantified by the distance to the closest obstacle point.
- **Step size**: the height gap between adjacent cells in the grid map.
- **Tip-over**: a function of slope angles and the robot’s orientation.
- **Sensor uncertainty**: sensor and localization error increase the variance of traversability estimates.
- **Negative obstacles**: detected by checking the lack of measurement points in a cell.
- **Slippage**: quantified by geometry and the surface material of the ground.

#### 3.3.1. Geometry-based risk sources

Geometry-based risk sources include collision with obstacles, too large step sizes in terrain, and impassable slopes. These geometry-based risks are constructed using geometric analysis of elevation map estimates and LiDAR point cloud points. We construct these risks per grid cell with the following methods. First, using the ground estimates from the ground segmentation pipeline, we obtain a lower bound on the height of the ground, i.e., the height at which the robot would place its foot or wheel if occupying that cell with its foot/wheel. Above this ground estimate, we can determine the relative height of other occupying LiDAR points in the temporally merged point cloud with respect to the ground. Points that occur above or below a certain height threshold that outline the body of the robot are treated as obstacles, and the corresponding grid cells at these

---

^1The risk map color notation in this figure holds for all the figures in the paper.
Figure 4. Multilayer geometric risk analysis, which first aggregates recent point clouds (top). Then, each type of analysis (slope, step, collision, etc.) generates a risk map along with uncertainties (middle rows). These risks are aggregated to compute the final CVaR map (bottom). Spot robot shown for scale. The risk map [colored by risk level—white: safe ($r \leq 0.05$), yellow to red: moderate ($0.05 < r \leq 0.5$), black: risky ($r > 0.5$)].

points are marked as untraversable. Similar analysis is performed for step size risk, which checks the height gap between adjacent cells in the elevation map. For adjacent cells that exhibit too high a height gap, these cells are marked as untraversable. Finally, checking the normals of the elevation map (Fankhauser and Hutter, 2016) gives an overall estimation of slope (which is the size of the normal calculation averaged over the approximate size of the robot), and areas with too high slope values are marked as high risk or untraversable.

Note that in all these analyses, the uncertainty of the elevation map plays a large role. Sensor and localization uncertainty corrupts elevation measurements to a varying degree, proportional with distance from the robot. Therefore, we adapt various detection or risk thresholds with distance from the robot to obtain a more robust result. For example, for the Husky robot, ground clearance beneath the robot belly is 10 cm. Detecting a 10 cm step size in elevation at 100 m away requires an angular localization accuracy of $< 0.06$ deg. This may be infeasible, and therefore adapting the step size threshold with distance helps to reduce spurious detections at longer ranges.

3.3.2. Confidence-based risk sources
One example of confidence-based risk sources is the detection of negative obstacles. To detect negative obstacles, we estimate areas in the point cloud that have no returns, and we use 2D ray-tracing to find “gap” areas that are not occluded by obstacles, small steps, or upward slopes. However, if this is the only criterion for negative obstacle detection, many false positives for negative obstacles are observed. One such instance is shown in Figure 5 when the robot turns the corner into a new room from a narrow passageway, and in the time taken for the LiDAR returns from all areas of the new room to reach the sensor, false-positives of negative obstacles are detected. To address this, we account for the confidence in the gaps in the point cloud by estimating whether these
STEP: Stochastic traversability evaluation and planning for risk-aware off-road navigation

Figure 5. Confidence-based risk analysis: the scene is illustrated through the point-of-view (POV) of a third-person (top left) and robot (top middle and top right). The aggregated point cloud (bottom left) has regions of no returns from the area on the left side of the robot. These holes in the point cloud are marked as negative obstacles in the risk layer (bottom middle) only when there are no returns from these regions despite them being unoccluded and sufficiently covered by laser strike pattern. This risk layer is aggregated with the geometric risk layers to compute the final CVaR map (bottom right).

areas have been sufficiently covered so far, i.e., whether the robot has sensed the area from different positions and for long enough to ensure that the gaps in the point cloud are not caused by the sensor and environment configuration; see Algorithm 1 for a detailed description of the method to check whether a given region passes the coverage check.

3.3.3. Semantics-based risk sources
LiDAR intensity returns are different on different materials at the same distance from the sensor. In (Hui et al., 2008), the authors demonstrate the efficacy of terrain classification using LiDAR intensity and height returns. We account for changes in the terrain features based on changes in the intensity of LiDAR returns. Areas that comprise a lot of mud and water have low intensity of returns. This enables detection of muddy regions where the robot may get stuck or fall down. Areas with deep water levels are similarly detected as negative obstacles because the LiDAR returns are very sparse. Notably, negative obstacles combined with low intensity of returns allows us to detect lethal regions with deep water levels. Figure 6 shows an instance of the water hazard detection when the robot is near a water puddle. Note that in Figure 5, on the contrary, while there are holes in the LiDAR point cloud, they are not surrounded by low intensity returns and hence are classified as negative obstacles.

3.4. Traversability Cost
To efficiently compute the Conditional Value-at-Risk (CVaR) traversability cost for \( l > 1 \) risk factors, we assume each risk factor \( R_l \) is an independent random variable that is normally distributed, with mean \( \mu_l \) and variance \( \sigma_l \). We take a weighted average of the risk factors to obtain the terrain assessment model, \( R = \sum_l w_l R_l \), which will also be normally distributed as \( R \sim N(\mu, \sigma^2) \). Let \( \varphi \) and \( \Phi \) denote the probability density function and cumulative distribution function of a standard
Algorithm 1. Algorithm for computing whether a given area is unoccluded and sufficiently covered, i.e., $m_{\text{cover MSE}} > 0$

Input: $m_{\text{unocc}}$: Grid map of unoccluded grid points from the current robot state $x_t$ (using ray-tracing),
$x_t = [p_x, p_y, \theta, v_x, v_y, v_{\theta}]^T$: Current robot state,
$d_{\text{cover}}$: Threshold for sufficient coverage (in meters).

Output: $m_{\text{unocc}}^N$: Number of times each grid cell in the grid map has been observed unoccluded by the robot so far,
$m_{\text{cover}} = [m_{\text{cover}}, m_{\text{cover MSE}}]^T$: Multilayer map indicating whether each grid cell being evaluated for presence of negative obstacles has been observed unoccluded from different points to ensure sufficient coverage.

1: # Initialize mean x and y robot positions as well as the mean squared deviation for all grid cells
$m_{\text{cover}} = \{0, 0, 0\}$

2: for $i = 0$ to $|m_{\text{unocc}}|$ do

3: # Update $m_{\text{unocc}}^N$ if the $i^{th}$ grid point is unoccluded, i.e., $m_{\text{unocc}}^{(i)} = 1$
$m_{\text{unocc}}^N = m_{\text{unocc}}^N + m_{\text{unocc}}^{(i)}$

4: # Update mean x and y robot positions from where each grid cell has been observed unoccluded
$\{m_{\text{x}}, m_{\text{y}}\} = \{m_{\text{x}}^{(i)} + \frac{n_{\text{unocc}}^{(i)} m_{\text{unocc}}^{(i)}}{m_{\text{unocc}}^{(i)}}, m_{\text{y}}^{(i)} + \frac{n_{\text{unocc}}^{(i)} m_{\text{unocc}}^{(i)}}{m_{\text{unocc}}^{(i)}}\}$
if $m_{\text{unocc}}^{(i)} = 1$

5: # Update mean squared deviation of the average robot positions from which a given grid cell is observed to check if it crosses the threshold for sufficient coverage, $d_{\text{cover}}$
$m_{\text{cover MSE}} = \begin{cases} 
(p_x - m_{\text{x}})^2 + (p_y - m_{\text{y}})^2 - (d_{\text{cover}})^2 & \text{if } m_{\text{unocc}}^{(i)} = 1 \\
- (d_{\text{cover}})^2 & \text{if } m_{\text{unocc}}^{(i)} = 0
\end{cases}$

6: end for

Figure 6. Semantics-based risk analysis: the robot camera (top left) shows a puddle of water in front of the robot. The aggregated point cloud (top right) has regions of no returns from the area in front of the robot. These holes in the point cloud are marked as water in the risk layer (bottom left) when the holes in the point cloud are present with low intensity returns in the area near the hole. This risk layer is aggregated with the geometric risk layers to compute the final CVaR map (bottom middle and bottom right) with different levels of robustness $\alpha$. We clearly see the effects on the risk map, where higher values of $\alpha$ result assigning high cost to unknown regions.
normal distribution, respectively. The corresponding CVaR is computed as

$$
\rho(R) = \mu + \sigma \frac{\phi(\Phi^{-1}(\alpha))}{1 - \alpha} \quad (16)
$$

The CVaR cost accounts for the expected cost in the $(1 - \alpha)$ tail of the distribution of the terrain assessment model, $R$, thus also accounting for high-risk, low probability events. We construct the $R$ such that the expectation of $R$ is positive to keep the CVaR value positive, $\mu \geq 0$.

Construction of the mean and variance of each risk factor depends on the type of risk. For example, collision risk is determined by checking for points above the elevation map, and the variance is derived from the elevation map variance, which is mainly a function of localization error. In contrast, negative obstacle risk is determined by looking for gaps in sensor measurements. These gaps tend to be a function of sensor sparsity, so the risk variance increases with distance from the sensor frame.

### 3.5. Risk-aware Geometric Planning

To optimize Equations 8 and 9, the geometric planner computes an optimal path that minimizes the position-dependent dynamic risk metric in Equation 5 along the path. Substituting Equation 16 into Equation 5, we obtain

$$
J_{pos}(x_{0:N}) = \mu_0 + \sum_{k=1}^{N} \left[ \mu_k + \sigma_k \frac{\phi(\Phi^{-1}(\alpha))}{1 - \alpha} \right]. \quad (17)
$$

(For a proof, see Appendix B.)

We use the $A^*$ algorithm to solve Equation 8 over a 2D grid. $A^*$ requires a path cost $g(n)$ and an admissible heuristic cost $h(n)$, given by

$$
g(n) = J_{pos}(x_{0:n}) + \lambda \sum_{k=0}^{n-1} \|x_k - x_{k+1}\|^2, \quad (18)
$$

$$
h(n) = \lambda \|x_n - x_G\|^2. \quad (19)
$$

For the heuristic cost, we use the shortest Euclidean distance to the goal, which is always less than the actual path cost. The parameter \(\lambda\) is a relative weighting between the distance penalty and the risk penalty, and it can be thought of as having units of (traversability cost / m). Our experiments use a relatively small value, which means we are mainly concerned with minimizing traversability costs.

### 3.6. Risk-aware Kinodynamic Planning

The geometric planner produces a path, i.e., a sequence of poses. We wish to find a kinodynamically feasible trajectory that stays near this path while satisfying all constraints and minimizing the CVaR cost. To solve Equations 10–13, we use a risk-aware kinodynamic MPC planner, whose steps we outline (Figure 7, Algorithm 2, Appendix C).

**Trajectory library**: Our kinodynamic planner begins with selecting the best candidate trajectory from a trajectory library, which stores multiple initial control and state sequences. The selected trajectory is used as an initial solution for solving a full optimization problem. The trajectory library can include 1) the trajectory accepted in the previous planning iteration, 2) a stopping (braking) trajectory, 3) a geometric plan following trajectory, 4) heuristically defined trajectories (including v-turns, u-turns, and varying curvatures), and 5) randomly perturbed control input sequences.

**QP Optimization**: Next, we construct a nonlinear optimization problem with costs and constraints (Equations 10–13). We linearize the problem about the initial solution and solve iteratively in a sequential quadratic programming (SQP) fashion (Nocedal and Wright, 2006). Let $\{\delta x_k, \delta u_k\}_{k=0:T}$ denote an initial solution. Let $\{\delta x_k, \delta u_k\}_{k=0:T}$ denote deviation from the initial solution. We introduce the solution vector variable $X$:

$$
X = [\delta x_0^T \ldots \delta x_T^T \delta u_0^T \ldots \delta u_T^T]^T. \quad (20)
$$
Algorithm 2. Kinodynamic MPC Planner (sequences \{\text{var}\}_{k=0:T} are expressed as \{\text{var}\} for brevity)

\textbf{Input:} current state \(x_0\), current control sequence (previous solution) \(\{u^*\}^{(j)}\)
\textbf{Output:} replanned trajectory \(\{x^*\}^{(j+1)}\), replanned control sequence \(\{u^*\}^{(j+1)}\)

\begin{align*}
\# \text{ Initialization} \\
1: & \{x^*\} = \text{updateReferenceTrajectory}() \\
2: & \{u^*\}^{(j)} = \text{stepControlSequenceForward}(\{u^*\}^{(j)}) \\
3: & l = \text{generateTrajectoryLibrary}(x_0) \\
4: & [\{x^c\}, \{u^c\}] = \text{chooseCandidateFromLibrary}(l) \\
\# \text{ Loop process} \\
5: & \text{for } i = 0 \text{ to qp_iterations } \text{do} \\
6: & [\{\delta x^*\}, \{\delta u^*\}] = \text{solveQP}(\{x^c\}, \{u^c\}, \{x^r\}) \\
7: & [\gamma, \text{solved}] = \text{lineSearch}(\{x^c\}, \{\delta x^*\}, \{u^c\}, \{\delta u^*\}) \\
8: & u^*_k = u^*_k + \gamma \delta u^*_k, \forall k = 0 : T \\
9: & \{x^c\} = \text{rollOutTrajectory}(x_0, \{u^c\}) \\
10: & \text{end for} \\
11: & \text{if solved then} \\
12: & \{x^*\}^{(j+1)}, \{u^*\}^{(j+1)} = \{x^c\}, \{u^c\} \\
13: & \text{else} \\
14: & \{x^*\}^{(j+1)}, \{u^*\}^{(j+1)} = \text{getStoppingTrajectory}() \\
15: & \text{end if} \\
16: & \text{return } \{x^*\}^{(j+1)}, \{u^*\}^{(j+1)}
\end{align*}

We can then write Equations 42–45 in the form

\begin{align*}
\text{minimize} & \quad \frac{1}{2} X^T P X + q^T X \\
\text{subject to} & \quad l \leq AX \leq u
\end{align*}

where \(P\) is a positive-semidefinite weight matrix, \(q\) is a vector to define the first-order term in the objective function, \(A\) defines inequality constraints, and \(l\) and \(u\) provide their lower and upper limit (see Appendix E). In the next subsection (Section 3.6.1), we describe these costs and constraints in detail. This is a quadratic program, which can be solved using commonly available QP solvers. Our implementation uses the OSQP solver, which is a robust and highly efficient general-purpose solver for convex QPs (Stellato et al., 2020).

**Linesearch:** The solution to the SQP problem returns an optimized variation of the control sequence \(\{\delta u^*_k\}_{k=0:T}\). We then use a linesearch procedure to determine the amount of deviation \(\gamma > 0\) to add to the current candidate control policy \(\pi\): \(u_k = u_k + \gamma \delta u^*_k\) (see Appendix F).
3.6.1. Optimization Costs and Constraints

**Costs:** Note that the cost (Equation 10) includes the CVaR risk. To linearize the cost for representation in a QP, we compute the Jacobian and Hessian of \( \rho \) with respect to the state \( x \). We efficiently approximate these terms via numerical differentiation.

**Kinodynamic constraints:** Similar to the cost, we linearize the system dynamics (Equation 11) with respect to \( x \) and \( u \). Depending on the dynamics model, this may be done analytically.

**Control limits:** The constraint function \( g(u) \) in Equation 12 limits the range of the control inputs. For example, in the six-state dynamics case, we limit maximum accelerations: \( |a_x| < a_x^{\max}, |a_y| < a_y^{\max}, \) and \( |\theta| < \theta^{\max} \).

**State limits:** Within the constraint function \( h(m, x) \) in Equation 13, we encode velocity constraints: \( |v_x| < v_x^{\max}, |v_y| < v_y^{\max}, \) and \( |\theta| < \theta^{\max} \). We also constrain the velocity of the vehicle to be less than some scalar multiple of the risk in that region, along with maximum allowable velocities:

\[
|v_\theta| < \gamma_\theta \rho(R_k),
\]

\[
\sqrt{v_x^2 + v_y^2} < \gamma_v \rho(R_k).
\]

This constraint reduces the robot’s kinetic energy in riskier situations, preventing more serious damage.

**Position risk constraints:** The function \( h(m, x_k) \) also encodes constraints on robot position and orientation to prevent the robot from hitting obstacles. The general form of this constraint is

\[
\rho(R_k) < \rho^{\max}.
\]

To formulate the constraint Equation 25, we locate areas on the map where the risk \( \rho \) is greater than the maximum allowable risk. These areas are marked as obstacles, and they are often highly nonconvex. To obtain a convex and tractable approximation of this highly nonconvex constraint, we decompose obstacles into nonoverlapping 2D convex polygons, and we create a signed distance function that determines the minimum distance between the robot’s footprint (also assumed to be a convex polygon) and each obstacle (Schulman et al., 2014). Let \( A, B \subset \mathbb{R}^2 \) be two convex sets, and define the distance between them as

\[
\text{dist}(A, B) = \inf\{\|T\| \mid (T + A) \cap B \neq \emptyset\}
\]

where \( T \) is a translation. When the two sets are overlapping, define the penetration distance as

\[
\text{penetration}(A, B) = \inf\{\|T\| \mid (T + A) \cap B = \emptyset\}.
\]

Then we can define the signed distance between the two sets as

\[
\text{sd}(A, B) = \text{dist}(A, B) - \text{penetration}(A, B).
\]

We then include within the function \( h(m, x_k) \) a constraint to enforce the following inequality:

\[
\text{sd}(A_{\text{robot}}, B_i) > 0 \quad \forall i \in \{0, \ldots, N_{\text{obstacles}}\}.
\]

Note that the robot footprint \( A_{\text{robot}} \) depends on the current robot position and orientation: \( A_{\text{robot}}(p_x, p_y, \theta) \), while each obstacle \( B_i(m) \) is dependent on map information (see Figure 8).

**Orientation constraints:** To constrain the robot’s orientation on sloped terrain so as to prevent the robot from rolling over or performing dangerous maneuvers, we add constraints to the function
Figure 8. Left: Computing convex to convex signed distance function between the robot footprint and an obstacle. Signed distance is positive with no intersection and negative with intersection. Right: Robot pitch and roll are computed from the surface normal rotated by the yaw of the robot. Purple rectangle is the robot footprint with surface normal $n^w$. $g$ denotes gravity vector, $n^r_{x,y,z}$ are the robot-centric surface normal components used for computing pitch and roll.

$h(m, x_k)$ that limit the roll and pitch of the robot as it settles on the surface of the ground. Denote the robot’s position as $p = [p_x, p_y]^T$ and the position/yaw as $s = [p_x, p_y, p_\theta]^T$. Let $\psi$ and $\phi$ denote the robot’s pitch and roll in its body frame. Let $\omega = [\psi, \phi]^T$. The constraint will have the form $|\omega| < \omega_{\text{max}}$. Let the surface normal vector at point $p$ be denoted by $n^w = [n^w_x, n^w_y, n^w_z]^T$ in the world frame. Let $n^r = [n^r_x, n^r_y, n^r_z]^T$ be the surface normal in the body frame, where we rotate by the robot’s yaw: $n^r = R_\theta n^w$ (see Figure 8), where rotation matrix $R_\theta$ represents a rotation about the angle $\theta$ about the world $z$ axis. Then, we define the robot pitch and roll as $\omega = g(n^r)$, where

$$\omega = g(n^r) = \begin{bmatrix} \text{atan2}(n^r_x, n^r_z) \\ -\text{atan2}(n^r_y, n^r_z) \end{bmatrix}. \tag{30}$$

Creating a linearly constrained optimization problem requires a linear approximation of the constraint:

$$|\nabla_s \omega(s) \delta s + \omega(s)| < \omega_{\text{max}}. \tag{31}$$

The linearization is realized by finding the gradients of the orientation constraints with respect to position and yaw separately (see Appendix D).

**Box Constraint:** Note that if $\delta x$ and $\delta u$ are too large, linearization errors will dominate. To mitigate this effect, we also include box constraints within Equations 12 and 13 to maintain a bounded deviation from the initial solution: $|\delta x| < \epsilon_x$ and $|\delta u| < \epsilon_u$.

**Adding Slack Variables:** To further improve the feasibility of the optimization problem, we introduce auxiliary slack variables for constraints on state limits, position risk, and orientation. For a given constraint $h(x) > 0$ we introduce the slack variable $\epsilon$, and we modify the constraint to be $h(x) > \epsilon$ and $\epsilon < 0$. We then penalize large slack variables with a quadratic cost: $\lambda \epsilon^2$. These slack variables are incorporated into the QP problem (Equations 21 and 22).

### 3.7. Risk-Driven Gait Selection

We considered two platforms for testing our traversability framework—a Clearpath Husky (wheeled robot) and Boston Dynamics Spot (quadrupedal legged robot). In the case of the Spot, multiple locomotion gaits are available, namely walking gait (standard operation), stair climbing gait (reduces the robot speed and pitches the robot to better see the stairs while descending), and crawling gait (three feet touch the ground at all times, the most stable gait).\(^2\) Furthermore, the height of the

\(^2\)Further descriptions of the gaits can be found here: https://support.bostondynamics.com/s/article/Operating-Spot
Figure 9. Stair detection with plane-fitting method. Images show a sequence of stair traversal from top left to bottom right. Blue and green planes indicate planes detected above and below the body of the robot. The robot walks down backwards down the stairs (facing left in the top left figure).

robot can also be dynamically adjusted; the height reduction to fit in small spaces is henceforth called the crouch gait. These locomotion gaits provide mobility for the Spot in areas that would not be traversable with the standard walking gait.

- **Stair gait selection**: Stair detection was performed using geometric analysis of LiDAR and camera disparity point-cloud data. To detect stairs, we use a plane fitting method similar to that of (Schwarze and Zhong, 2015). We compute normals and perform clustering to identify contiguous surfaces in the point cloud of similar slope. We then isolate flat and vertical planes to detect potential stair surfaces. After finding candidate stair surfaces, the larger stair direction and overall plane is determined. Finally, the intersection of the ground plane that the robot currently sits on and the plane of the stairs is used to determine the start of the stairs. Once the start of the stairs (either up or down) is determined, the robot can align itself to this location and begin a stair walking gait.

- **Crawl gait through slip detection**: A learning-based slip prediction module was implemented in (Dey et al., 2022) that leveraged the joint state information of the quadruped at each time instant, see Figure 10. We train an ensemble model to predict the probability of slip, \( p(\text{slip}) \), at each time instant, based on the robot limb kinematics and kinetics. Slip-annotations from multiple field trials across different terrains (e.g., rocky, sandy, muddy terrains) formed the ground truth for the model training. During slip prediction using the trained model, if the \( p(\text{slip}) > 0.5 \), the crawling gait mode is enabled for the robot to traverse the potentially slippery terrain. For more details on the implementation of the slip-predictive model, see (Dey et al., 2022).

- **Crouch behavior in low ceiling areas**: If the robot detects a region with low ceiling that is within 16 cm of the robot height (the limit to which the robot can bend), the robot enables crouch gait, which reduces its height by 16 cm, see Figure 11. Once the robot passes through the low ceiling area, it switches back to walking gait.
3.8. Recovery Behaviors

Recovery behaviors ensure fast, reactive actions to hazardous, mission-ending situations such that the robot can recover quickly and safely. When a recovery behavior is activated, the robot temporarily pauses following the commands from the geometric and kinodynamic planning pipeline to get to a safe state from where it can resume the hierarchical planning pipeline (see Figure 2).

**Tilt recovery:** The tilt recovery behavior tracks the odometry received from the state estimation pipeline for the wheeled robot. The odometry is a combination of wheel odometry and inertial measurements. It consists of the pose and twist measurements. If the robot’s pitch exceeds a threshold, the robot backtracks at a low speed along the same path that got the robot into the lethal pitch state.

**Wiggle behavior:** We monitor the robot’s velocity obtained from wheel and inertial odometry to ensure that the robot is not stuck during the mission. If the robot does not move for a certain time period while the MPC provides robot movement commands, the robot is considered to be
stuck. In that case, we deploy a wiggle behavior that moves the wheels of the robot in such a way that, unintruded, the robot would travel in a figure-8 pattern. The allows the robot to wiggle back and forth until it is not stuck anymore. If the wiggle behavior fails to ensure recovery, the robot is considered stuck and the mission ends.

**Escape lethal:** If the robot finds itself trapped in a lethal zone for some period of time, it is useful to have a behavior that attempts to escape the high risk regions with minimum risk. This is accomplished in the following way. First, a goal location is chosen that is not in a lethal area, close to the robot, and which minimizes the amount of lethal area the robot would need to traverse to reach the goal in a straight line. Next, the risk threshold of the traversability cost maps is changed to allow traversal through high risk map cells, which would normally be treated as constraints or obstacles. Then, the goal is sent to the global and local planners, and executed on the robot. If the goal is reached, the recovery behavior is ended, but if the goal is still not reachable, the risk threshold can be further adjusted to allow more and more risky behavior in an attempt to escape. In this way, the CVaR risk threshold is useful for adapting the risk profile of the robot in real time.

### 3.9. Dynamic Risk Adjustment

The CVaR metric allows us to dynamically adjust the level and severity of risk we are willing to accept. Selecting low $\alpha$ reverts to using the mean cost as a metric, leading to optimistic decision making while ignoring low-probability but high cost events. Conversely, selecting a high $\alpha$ leans towards conservatism, reducing the likelihood of fatal events while reducing the set of possible paths. We adjust $\alpha$ according to two criteria: 1) Mission-level states, where depending on the robot’s role, or the balance of environment and robot capabilities, the risk posture for individual robots may differ. 2) Recovery Behaviors, where if the robot is trapped in an unfavorable condition, by gradually decreasing $\alpha$, an escape plan can be found with minimal risk. These heuristics are especially useful in the case of risk-aware planning, because the feasibility of online nonlinear MPC is difficult to guarantee. When no feasible solution is found for a given risk level $\alpha$, a riskier but feasible solution can be quickly found and executed.

### 4. Experiments

In this section, we report the experimental and field performance of STEP. We first present a comparative study between different adjustable risk thresholds in simulation on a wheeled differential drive platform. Then, we demonstrate real-world performance using a wheeled robot deployed in an abandoned subway filled with clutter, and a legged platform deployed in a lava tube environment.
4.1. Simulation Study

To assess statistical performance, we perform 50 Monte-Carlo simulations with randomly generated maps and goals. These results were also included in our previous work (Fan et al., 2021). Random traversability costs are assigned to each grid cell. The following assumptions are made: 1) no localization error, 2) no tracking error, and 3) a simplified perception model with artificial noise. We give a random goal 8 m away and evaluate the path cost and distance. We use a differential-drive dynamics model (no lateral velocity).

We compare STEP using different $\alpha$ levels. Figure 13 shows the distribution of paths for different planning configurations. The optimistic (close to mean-value) planner $\alpha = 0.05$ typically generates shorter paths, while the conservative setting $\alpha = 0.95$ makes long detours to select statistically safer paths. The other $\alpha$ settings show distributions between these two extremes, with larger $\alpha$ generating similar paths to the conservative planner and smaller $\alpha$ generating more time-optimal paths. Statistics are shown in Figures 14 and 15.

Figure 13. Path distributions from four simulated runs. As seen in the figure legend, the risk level $\alpha$ spans from 0.05 (close to mean-value) to 0.95 (conservative). Smaller $\alpha$ typically results in a shorter path, while larger $\alpha$ chooses statistically safe paths.

Figure 14. Distance vs risk tradeoff from 50 Monte-Carlo simulations. Left: Distributions of path distance. Right: Distributions of max risk along the traversed paths. Box plot uses standard quartile format where the orange line denotes the mean value and dots denote the outliers.
STEP: Stochastic traversability evaluation and planning for risk-aware off-road navigation

Figure 15. Distance vs success of reaching goal tradeoff from 50 Monte-Carlo simulations for different values of $\alpha = \{0.1, 0.5, 0.95\}$. Left: Likelihood of reaching goal within some path distance. Right: Average risk along the traversed paths vs the time taken to reach the goal. Plot uses standard quartile format wherein the dots that lie outside the shaded regions are outliers.

4.2. Hardware Results

We deployed STEP on two different robots (wheeled and legged) in different challenging environments (an abandoned subway, a lava tube, a limestone mine, and a combination of different subterranean environments in the DARPA Subterranean Challenge). The robot was equipped with custom sensing and computing units, and driven by JPL’s NeBula autonomy software (Ali-akbar Agha-mohammadi, et al., 2022). Three Velodyne VLP-16s were used for collecting LiDAR data. Localization was provided onboard by a LiDAR-based SLAM solution (Palieri et al., 2020; Ebadi et al., 2020). The entire autonomy stack runs on an Intel Core i7 CPU. The typical CPU usage for the traversability stack is about a single core.

4.2.1. Previous Results

Earlier versions of STEP (without semantic and confidence-aware risk sources, gait adaptations, and the entire suite of recovery behaviors) were tested in multiple field locations. For the sake of completeness, we have included these results, published in (Fan et al., 2021), from:

1. An abandoned subway filled with industrial clutter in Downtown Los Angeles, CA in Figure 16,
2. Valentine Cave in Lava Beds National Monument, Tulelake, CA in Figure 17.

4.2.2. Results from the Kentucky Underground and LA Subway

We tested the various new parts of our traversability pipeline during fully autonomous runs in our tests at Kentucky Underground Storage (KU), Wilmore, KY. The results of the cost map for confidence-based and semantics-based risk are seen in Figures 5 and 6. We also tested the tilt recovery behavior on a pile of rubble in KU by forcing the robot into a hazardous tilt position; the robot was able to recover from this risky configuration, as seen in Figure 12.

We also tested new results in an abandoned subway with multiple levels filled with industrial clutter in downtown Los Angeles, CA. For example, the LA Subway environment served as a test bed for the stair gait and crouch gait behaviors that are shown in Figures 9 and 11, respectively.

4.2.3. Results from the DARPA Subterranean Challenge

The traversability framework was deployed in the DARPA Subterranean Challenge held in Louisville Mega Cavern, Louisville, KY. The course consisted of three different environments—tunnel, urban, and cave. Figure 18 shows the course map and different sections that the robot explored. The
Figure 16. Traversability analysis results for Husky in an abandoned subway experiment. Top left to right: Risk maps at three varying risk levels: $\alpha = 0.1, 0.5, 0.9$, respectively. Colors correspond to CVaR value [white: safe ($r \leq 0.05$), yellow to red: moderate ($0.05 < r \leq 0.5$), black: risky ($r > 0.5$)]. Also shown are the most recent LiDAR measurements (green points). Bottom left and middle: Front and right onboard cameras observing the same location. Bottom right: Completed top-down map of the environment after autonomous exploration. Bright dots are pillars, which are visible in the camera images. The effect of these risk analyses results in intuitive outcomes—for example, a low pile of metal, while probably traversable, should be avoided if possible. When the region has inadequate sensor coverage, the risk will be high. When the robot is closer and the sensor coverage is good, then the CVaR cost will decrease, yielding a more accurate risk assessment. This results in more efficient and safer planning when compared to deterministic methods.

Figure 17. Traversability analysis results for the Valentine Cave experiment. From left to right: Third-person view, elevation map (colored by normal direction), risk map [colored by risk level—white: safe ($r \leq 0.05$), yellow to red: moderate ($0.05 < r \leq 0.5$), black: risky ($r > 0.5$)], and planned geometric/kinodynamic paths (yellow lines/red boxes).
Figure 18. The layout of the competition course with snapshots of the systems build of the areas traversed by the robots that have interesting terrain and high difficulty.

sections showcased in Figure 18 have a high difficulty rating as listed in the course layout guide posted by DARPA. The terrain challenges of each of the sections are listed below:

Region A: An office-like area that consists of narrow corridors (5’ x 8’) and small rooms (8’ x 8’) for the robots to explore.

Region B: A warehouse-like area with a lot of shelving and clutter imitating an industrial warehouse after an earthquake.

Region C: A connection between the urban and tunnel part of the course. The stairs act as a negative obstacle for the wheeled robots. The presence of this negative obstacle in a narrow corridor makes the drop harder to detect.

Region D: A constrained passage with ground, wall, and ceiling obstacles—with vertical pipes and debris.

Region E: A narrow cave opening mimicking a region that humans have to crawl through. The ground slopes upwards and the ceiling slopes downwards, creating issues for ground segmentation and low-ceiling detection.

Region F: A small limestone cave with rubble and loose rock piles.

Figure 19 shows the results of the traversability analysis and the geometric and kinodynamic plans of the robot in the different scenarios listed above. We see that the robot is able to plan safe paths in all of these scenarios and successfully navigate the different terrain hazards.

5. Discussion

We presented our approach to Stochastic Traversability Evaluation and Planning and the performance of our method in the DARPA Subterranean Challenge. The method was successfully able to...
account for uncertainty in the a priori unknown environment and to navigate various types of terrain. In this extension of (Fan et al., 2021), we accounted for semantic and confidence-aware risk sources in addition to the previously presented geometric risk sources. Additionally, we allowed for changes in gaits of quadrupedal robots based on the terrain risk observed and information obtained from proprioceptive sensors. We added a large suite of recovery behaviors to robustify our traversability stack in case of unaccounted for but mission-ending situations that the rest of the framework cannot account for.
There are many avenues of improvement of our framework. STEP accounts for traversability risk arising from static environments, but it does not make predictions for dynamic obstacles (Wei et al., 2023). Hence, the robot cannot plan reactive motions to avoid falling debris or account for sudden movement of rubble underneath it. The traversability analysis is also the same for all speeds of the robot. Dynamic traversability analysis will allow for a larger range of motion. The LIDAR intensity-based semantic terrain analysis provides us with a coarse classification of muddy terrain vs regular terrain, but using a vision-based semantic classification in addition to the LIDAR intensity information will provide a richer understanding of the surface over which the robot is traversing.

Acknowledgments

The work was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004), and Defense Advanced Research Projects Agency (DARPA).

ORCID

Anushri Dixit https://orcid.org/0000-0002-9698-2189
David D. Fan https://orcid.org/0000-0002-8261-1045
Kyohei Otsu https://orcid.org/0000-0002-9363-8817
Sharmita Dey https://orcid.org/0000-0001-8058-4867
Ali-Akbar Agha-Mohammadi https://orcid.org/0000-0001-5509-1841
Joel Burdick https://orcid.org/0000-0002-3091-540X

References


# A. Dynamics model for differential drive

For a simple system that produces forward velocity and steering (e.g., differential drive systems), we may wish to specify the state and controls as

\[
x = [p_x, p_y, p_\theta, v_x]^T
\]

\[
u = [a_x, v_\theta]^T
\]

(32) (33)

For example, the dynamics \( x_{k+1} = f(x_k, u_k) \) for a simple differential-drive system can be written as

\[
x_{k+1} = x_k + \Delta t \begin{bmatrix} v_x \cos(p_\theta) \\ v_x \sin(p_\theta) \\ \gamma v_x + (1 - \gamma) v_\theta \\ a_x \end{bmatrix}
\]

(34)

where \( \gamma \in [0, 1] \) is a constant that adjusts the amount of turning-in-place the vehicle is permitted.

# B. Dynamic risk metric using CVaR for Normal distributions

\[
J(x_0, \pi) = R_0 + \rho_0 (R_1 + \rho_1 (R_2 + \cdots + \rho_{T-1}(R_T)))
\]

\[
= R_0 + \rho \left( R_1 + \rho \left( R_2 + \cdots + \rho \left( R_{T-1} + \mu_T + \sigma_T \left( \frac{\Phi^{-1}(\alpha)}{1-\alpha} \right) \right) \right) \right)
\]

\[
= \sum_{i=0}^{T} \rho_i \mu_i + \sigma_i \left( \frac{\Phi^{-1}(\alpha)}{1-\alpha} \right)
\]

C. Kinodynamic Planning Diagram

See Figure 20.

# D. Gradients for Orientation Constraint

We describe in further detail the derivation of the orientation constraints. Denote the position as \( p = [p_x, p_y]^T \) and the position/yaw as \( s = [p_x, p_y, p_\theta] \). We wish to find the robot’s pitch \( \psi \) and roll \( \phi \) in its body frame. Let \( \omega = [\psi, \phi]^T \). The constraint will have the form \( |\omega(s)| \leq \omega_{\text{max}} \). At \( p \), we compute the surface normal vector, calling it \( n^w = [n^w_x, n^w_y, n^w_z]^T \), in the world frame. To convert the normal vector in the body frame, \( n^r = [n^r_x, n^r_y, n^r_z]^T \), we rotate by the robot’s yaw: \( n^r = R_\theta n^w \) (see Figure 8), where \( R_\theta \) is a basic rotation matrix by the angle \( \theta \) about the world z axis:

\[
R_\theta = \begin{bmatrix}
\cos p_\theta & \sin p_\theta & 0 \\
-\sin p_\theta & \cos p_\theta & 0 \\
0 & 0 & 1
\end{bmatrix}
\]

(35)

Let the robot pitch and roll vector \( \omega \) be defined as \( \omega = g(n^r) \), where

\[
\omega = g(n^r) = \begin{bmatrix}
\arctan2(n^r_x, n^r_z) \\
-\arctan2(n^r_y, n^r_z)
\end{bmatrix}
\]

(36)
STEP: Stochastic traversability evaluation and planning for risk-aware off-road navigation

Figure 20. Diagram of kinodynamic MPC planner, which begins with evaluating various paths within a trajectory library. The lowest cost path is chosen as a candidate and optimized by the QP solver.

Creating a linearly constrained problem requires a linear approximation of the constraint:

\[ |\nabla_s \omega(s) \delta s + \omega(s)| \leq \omega_{\text{max}} \]  

(37)

Conveniently, computing \( \nabla_s \omega(s) \) reduces to finding gradients with respect to position and yaw separately. Let \( \nabla_s \omega(s) = [\nabla_p \omega(s), \nabla_\theta \omega(s)]^T \). Then

\[
\nabla_p \omega(s) = (\nabla_n \cdot g)(R_\theta) (\nabla_p n^w) 
\]

(38)

\[
\nabla_\theta \omega(s) = (\nabla_n \cdot g) \left( \frac{d}{d\theta} R_\theta \right) (n^w) 
\]

(39)

where

\[
\nabla_n \cdot g = \begin{bmatrix} \frac{n^e}{(n^e)^2+(n^z)^2} & 0 & -\frac{n^z}{(n^e)^2+(n^z)^2} \\ 0 & -\frac{n^e}{(n^e)^2+(n^z)^2} & \frac{n^z}{(n^e)^2+(n^z)^2} \end{bmatrix} 
\]

(40)

and

\[
\nabla_p n^w = \begin{bmatrix} \frac{\partial n^w}{\partial p_x} & \frac{\partial n^w}{\partial p_y} \\ \frac{\partial n^w}{\partial p_x} & \frac{\partial n^w}{\partial p_y} \end{bmatrix} 
\]

(41)

The terms with the form \( \frac{\partial n^w}{\partial p} \) amount to computing a second-order gradient of the elevation on the 2.5D map. This can be done efficiently with numerical methods (Fankhauser and Hutter, 2016).

**E. Converting nonlinear MPC problem to a QP problem**

Our MPC problem stated in Equations 10–13 is nonlinear. To efficiently find a solution, we linearize the problem about an initial solution, and we solve iteratively, in a sequential quadratic programming (SQP) fashion (Nocedal and Wright, 2006). Let \( \{\hat{x}_k, \hat{u}_k\}_{k=0,\ldots,T} \) denote an initial solution. Let \( \{\delta x_k, \delta u_k\}_{k=0,\ldots,T} \) denote deviation from the initial solution. We approximate Equations 10–13 by
a problem with quadratic costs and linear constraints with respect to \( \{ \delta x, \delta u \} \):

\[
\{ \delta x^*, \delta u^* \} = \arg \min_{\delta x, \delta u} \sum_{k=0}^{T} \| \hat{x}_k + \delta x_k - x_k^* \| Q_k + \lambda J(\hat{x}_k + \delta x_k, \hat{u}_k + \delta u_k)
\]

\[
\text{s.t. \quad \forall k \in [0, \ldots, T]:}
\]

\[
\dot{x}_{k+1} + \delta x_{k+1} = f(\dot{x}_k, \hat{u}_k) + \nabla_x f \cdot \delta x_k + \nabla u f \cdot \delta u_k
\]

\[
g(\hat{u}_k) + \nabla_u g \cdot \delta u_k > 0
\]

\[
h(m, \dot{x}_k) + \nabla_x h \cdot \delta x_k > 0
\]

where \( J(\dot{x} + \delta x, \hat{u}_k + \delta u_k) \) can be approximated with a second-order Taylor approximation (for now, assume no dependence on controls):

\[
J(\dot{x} + \delta x) \approx J(\dot{x}) + \nabla_x J \cdot \delta x + \delta x^T H(J) \delta x
\]

and \( H(\cdot) \) denotes the Hessian. The problem is now a quadratic program (QP) with quadratic costs and linear constraints. To solve Equations 42–45, we introduce the solution vector variable \( X \):

\[
X = \begin{bmatrix} \delta x_0^T & \cdots & \delta x_T^T & \delta u_0^T & \cdots & \delta u_T^T \end{bmatrix}^T
\]

We can then write Equations 42–45 in the form

\[
\min \frac{1}{2} X^T P X + q^T X
\]

\[
\text{subject to \quad } l \leq A X \leq u
\]

where \( P \) is a positive-semidefinite weight matrix, \( q \) is a vector to define the first-order term in the objective function, \( A \) defines inequality constraints, and \( l \) and \( u \) provide their lower and upper limit.

**F. Linesearch Algorithm for SQP solution refinement**

The solution to the SQP problem returns an optimized control sequence \( \{ u_k^* \}_{k=0:T} \). We then use a linesearch routine to find an appropriate correction coefficient \( \gamma \), using Algorithm 3. The resulting correction coefficient is carried over into the next path-planning loop.

**Algorithm 3.** Linesearch Algorithm

**Input:** candidate control sequence \( \{ u_k^c \}_{k=0:T} \), SQP solution \( \{ \delta u_k^* \}_{k=0:T} \)

**Output:** correction coefficient \( \gamma \)

**Initialization**

1. initialize \( \gamma \) by default value or last-used value
2. \( [c, o] = \text{getCostAndObstacles}(\{ u_k^c \}_{k=0:T}) \)

**Linesearch Loop**

3. for \( i = 0 \) to max_iteration do
4. for \( k = 0 \) to \( T \) do
5. \( u_k^{c(i)} = u_k^c + \gamma \delta u_k^* \)
6. end for
7. \( [c^{(i)}, o^{(i)}] = \text{getCostAndObstacles}(\{ u_k^{c(i)} \}_{k=0:T}) \)
8. if \( (c^{(i)} \leq c \text{ and } o^{(i)} \leq o) \) then
9. \( \gamma = \min(2\gamma, \gamma_{max}) \)
10. break
11. else
12. \( \gamma = \max(\gamma/2, \gamma_{min}) \)
13. end if
14. end for
15. return \( \gamma \)