Cloud Hopping; Navigating in 3D Uneven Environments via Supervoxels and Control Lyapunov Function

Fetullah Atas 1 Grzegorz Cielniak 2 Lars Grimstad 1

Abstract—This paper presents a novel feedback motion planning method for mobile robot navigation in 3D uneven terrains. We take advantage of the supervoxels representation of point clouds, which enables a compact connectivity graph of traversable regions on the point cloud maps. Given this graph of traversable areas, our approach navigates the robot to any reachable goal pose using a control Lyapunov function (cLf) and a navigation function. The cLf ensures the kinodynamic feasibility and target convergence of the generated motion plans, while the navigation function optimizes the resulting feedback motion plans. We carried out navigation experiments in real and simulated 3D uneven terrains. In all circumstances, the experimental findings show that our approach performs superior to the baselines, proving the approach’s efficiency and adaptability to navigate a robot in challenging uneven 3D terrains. The proposed method can also navigate a robot with a particular objective, e.g., shortest-distance or least-inclined path. Finally, we provide an open-source implementation of the proposed method to benefit the robotics community.

Index Terms—Uneven Terrain Navigation, Outdoor Robotics, Motion Planning.

I. INTRODUCTION

Robots are increasingly being used in high-risk areas characterized by uncertainties. For example, a high-speed unmanned aerial vehicle (UAV) in a dynamic environment, a legged robot navigating unstructured terrain, or a mobile robot performing agricultural duties in uneven terrain are all subject to some uncertainties that only reveal themselves on runtime. These uncertainties include slippery or rough terrain, windy weather, and changes in robot dynamics due to the aging of the robot components. A popular strategy to control robots subject to such uncertainties uses a decoupled approach for consecutive planning and control steps. In this strategy, a motion planner develops a set of obstacle-free open-loop trajectories. Then a controller stabilizes the system around the trajectory to drive the robot to the desired state. While this strategy works well for systems with minor uncertainties (e.g., robots deployed in indoor environments), issues arise when uncertainties or noises develop substantially due to system dynamics or highly dynamic environments. Early motion planning (sampling-based) algorithms such as RRT and PRM did not account for system dynamic constraints. Hence it was not theoretically possible to provide guarantees for the robot to stabilize around a given open-loop trajectory.

Feedback motion planning has been introduced to deal with uncertainties, bringing more reliable solutions than open-loop trajectories. The system can evaluate itself while navigating and adjust its actions accordingly, thanks to instant feedback. A funnel metaphor has been widely popularized in literature to describe the procedure of feedback motion planning. A feedback motion plan is imagined as a set of sequentially composed funnels steering the states towards a goal region. In this work, we consider a robot operating in an uneven 3D terrain navigation scenario where interaction between the robot and rough terrain cannot be modeled accurately in the planning phase. Hence it is unclear whether the robot can follow a planned trajectory. We propose a new feedback motion planning strategy to overcome difficulties arising from robots’ unpredictable interaction with uneven 3D terrain. Our approach uses point cloud maps of an environment which are used to determine traversable regions of the terrain. A navigation function combined with a control Lyapunov function (cLf) is used to drive the robot to a reachable goal region in a near-optimal manner. The proposed navigation function includes a term for the optimality of resulting motion plans. This proved to make our approach flexible regarding the expected behavior from the robot. In contrast to previous methods that mainly experimented within simulations, we deploy a feedback motion planning algorithm to a real robot navigating in uneven terrain, demonstrating its potential in real-world scenarios. The specific contributions of our work include the following:

- We propose a novel feedback motion planning scheme for navigation in real uneven environments using supervoxels and cLf.
- We show that our method outperforms state-of-the-art sampling-based planners through experiments in simulated and real-world uneven terrains. We demonstrate that our approach is highly efficient and flexible.
- We provide an open-source implementation of our approach to the robotics community.

1 Norwegian University of Life Sciences {fetullah.atas, lars.grimstad}@nmbu.no
2 University of Lincoln gcicielniak@lincoln.ac.uk

https://github.com/NMBURobotics/vox nav

1
In this section, we take a brief look into existing work in sampling-based and feedback motion planning, respectively. Sampling-based planning has gained traction in recent decades because of its effectiveness and usefulness in planning for complex systems. The PRM [13] and RRT [15] algorithms and their dozens of variants have paved the way for sampling-based motion planning. The authors in [12] investigated sampling-based planners’ completeness and optimality properties to further understand formal guarantees. Some sampling-based planners were later improved to account for non-holonomic constraints [22]. A wide range of sampling-based planners have been proposed; a library called Open Motion Planning Library (OMPL) [29] contains implementations of key sampling-based planners. The majority of these planners are based on RRT* [12] and PRM* [12] planners (e.g., RRTX [21], LazyPRM* [5], InformedRRT* [9], and so on), but a newer collection of planners uses a structure that contains both graphs and trees (e.g., BIT* [8], ABIT* [28], AIT* [27]). Some methods utilize parallelized approaches (e.g., CFOREST [20], AnytimePartShortening (APS) [17]), in which many planners execute concurrently on different threads while planners tell each other of milestones reached, leading to better performance overall. Sampling-based planners are used to generate feasible open-loop trajectories, the trajectories are then fed to a controller so that the robot achieves the desired goal safely.

Feedback motion plans are often more reliable when employed instead of open-loop plans [16]. Mason et al. introduced the “funnel” metaphor for the feedback policies that drive a large set of initial conditions to final conditions [19]. There is a strong connection between Lyapunov functions and these motion planning funnels, as both concern the stability of a system around some point [6]. Tedrake et al. [31] introduced LQR-trees, a method that constructs a sparse tree of LQR-stabilized trajectories that probabilistically ensures that all initial states capable of reaching the goal will do so eventually. Park et al. [24] proposed a feedback motion planning approach via non-holonomic RRT*. The proposed algorithm generates feedback policies that result in asymptotically optimal plans while respecting the non-holonomic constraints of a unicycle robot. Ege et al. [7] proposed a feedback motion planning method for unmanned surface vehicles (USVs) via RRT.

The validation or computation of feasible Lyapunov functions for complex nonlinear systems has become possible thanks to optimization techniques such as Sum-of-Squares (SOS). Majumdar et al. propose funnel libraries for real-time feedback motion planning via SOS optimization [18]. The method first computes a library of funnels and different system maneuvers and then uses the calculated funnel libraries to compose feedback plans sequentially on run-time.

Authors in [2] proposed a feedback motion strategy for a differential drive robot based on a network of obstacle-free regions, which is similar to our approach in terms of using a connectivity graph. First, within the environment, a graph or tree structure composed of safe zones (boxes, rectangles) is created. Following that, a local feedback policy generates the appropriate control actions to move the robot to a different (connected) area while strictly ensuring that the robot never exceeds the bounds of the current active region until it reaches the borders of the next zone. The method was tested with a simulated robot in 2D environment.

Unlike probabilistic sampling-based planners, we base our strategy on an optimal navigation function, resulting in more predictable and deterministic planner behavior (see Tab. I). Our method uses a point cloud map, which has various information about the traversability cost of terrains [3]. Connecting a navigation function with realistic point cloud maps allows our approach to be easily configured so that the resulting feedback motion plans have desirable properties such as shortest-distance, least-inclination, or least-rough. Additionally, we validate our approach on real uneven terrains using a real robot.

III. Approach

Our approach is divided into two stages. First, we build supervoxels from point cloud maps and then calculate cost-to-go values over the supervoxels using Dijkstra or A* algorithm. In the second stage, we use cost-to-go to determine the best path to take, the CLf then computes feasible control commands accordingly until a termination condition is met. In the remainder of this section, we formally define the feedback motion planning problem and describe details for both stages of our approach.

A. Problem Formulation

A discrete feedback plan is defined with state \( \tilde{x} \) and control \( \tilde{u} \) history sets at stage \( t \). These are defined as:

\[
\tilde{x} = (x_1, x_2, ..., x_t), \quad \tilde{u} = (u_1, u_2, ..., u_t).
\]

Based on Eq. 1 we define the following components for a feasible discrete feedback plan:

1) A finite state space \( X \).
2) A control space \( U(x) \) for each state \( x \in X \).
3) A state transition function \( f \) that produces next state \( f(x, u) \in X \).
4) A set of stages denoted by \( t \) that begins at \( t = 0 \) and continues indefinitely, and a goal set \( X_G \in X \).
The feedback motion planning task is defined as a function \( \phi \) that maps every state to control: \( X \rightarrow U \). Once the goal is reached, a defined termination action is activated \( \phi(x) = U_F \), if \( x \in X_G \).

To consider the optimality property in the above definition, we define an additional navigation function \( \theta \) \[16\]. The navigation function \( \theta \) escapes local minima by having a property that maps every state to control; \( X \rightarrow \theta(x) \) in which \( \theta(x') < \theta(x) \).

\[
\theta(x) = \min_{u \in U(x)} (l(x, u) + \theta(f(x, u))), \quad x' = f(x, u^*). \tag{2}
\]

In Eq. 2 \( l(x, u) \) is cost term (see Subsec. III-C) and \( \theta \) is a feedback plan by selecting proper control \( u \).

### B. Control-Lyapunov Function

An ordinary Lyapunov function is typically used for stability analysis of a non-linear system. An extension to the ordinary Lyapunov function was introduced by Sontag \[26\], known as Control-Lyapunov Function (cLf), in which the cLf tests whether a dynamic system is stabilizable to a set point by applying a feasible control input \( u(x) \). This paper proposes a discontinuous cLf based on a controller originally introduced in \[1\] for a uni-cycle robot model. The modified cLf controller ensures kinodynamic feasibility and guaranteed convergence to the target state with smooth trajectory in all quadrants W.R.T robot body frame.

\[
\begin{bmatrix}
\dot{x} \\
\dot{y} \\
\dot{\sigma}
\end{bmatrix} =
\begin{bmatrix}
sin \sigma & 0 \\
0 & \cos \sigma \\
0 & 1
\end{bmatrix}
\begin{bmatrix}
v \\
w
\end{bmatrix}.
\tag{3}
\]

For the convenience of stabilizing the robot to a pose, the error dynamics are represented in polar coordinates:

\[
e = \sqrt{(y_g - y)^2 + (x_g - x)^2},
\]

\[
a = \tan(y_g - y, x_g - x),
\]

\[
\lambda = \sigma_g
\]

Where terms \( e \), \( a \), and \( \lambda \) are explained and illustrated in Fig. 2 and

\[
\dot{e} = -v \cos(a),
\]

\[
\dot{a} = -w + v \frac{\sin(a)}{e},
\]

\[
\dot{\lambda} = v \frac{\sin(a)}{e}.
\tag{5}
\]

Based on the new dynamics in polar coordinates, a candidate Lyapunov function is constructed in quadratic form.

\[
V = V_1 + V_2 = \frac{1}{2} e^2 + \frac{1}{2} (a^2 + \lambda^2).
\tag{6}
\]

To ensure asymptotic stability of this system, the control terms \( v \) and \( w \) are required to be chosen such that the derivative of Lyapunov function \( V \) is semi-definite negative with \( V < 0 \) and \( V(0) = 0 \). The authors \[1\], proposed \( v \) and \( w \) as suitable for driving robot states \( (x, y, \lambda) \) to zeros from any initial state. Nonetheless, during tests on a real robot and simulation, the ensuing behavior with originally proposed \( v \) and \( w \) can be unappealing due to jerks in some circumstances, see, for instance Fig. 5a. Inspired by the approach in \[30\], we deal with these situations by dividing state space into different subsets. Hence the resultant cLf is discontinuous. According to the modified scheme, depending on the quadrant in which the target pose resides in, dynamically allocated control inputs \( v \) and \( w \) are chosen:

\[
v = \begin{cases} 
  k_1 e \cos a, & \text{if } (x - x_g) < 0 \\
  -k_1 e \cos a, & \text{otherwise}, 
\end{cases}
\tag{7}
\]

\[
w = \begin{cases} 
  k_2 a + (\cos(a + \pi) \sin(a + \pi)) / a + \pi, & \text{if } (x - x_g) < 0 \text{, and } (y - y_g) < 0 \\
  k_2 a + (\cos(a - \pi) \sin(a - \pi)) / a - \pi, & \text{if } (x - x_g) < 0 \text{, and } (y - y_g) > 0 \\
  k_2 a + (\cos(a) \sin(a)) / a, & \text{otherwise}.
\end{cases}
\tag{8}
\]

This leads to improved robot behavior due to smoother convergence, see Fig. 5b. The controllers are substituted in navigation function described in Subsec. III-B as \( U(x) = [v, w]^T \). We explain effects of control gains \( k_1 \) and \( k_2 \) in Sec. IV. Our algorithm can be configured to operate with state feedback in run-time and with a state propagation function based on Eq. 3, Eq. 7 and Eq. 8. It is feasible to obtain a whole motion plan before plan execution using the state propagation setup. The resulting feedback motion plans are nearly identical for both run-time and propagated state setups can be seen in Fig. 4. The state propagation setup is handy for
Fig. 3: Figure depicting the construction of supervoxels on the point cloud maps. In the left, a cost-regressed point cloud map of a real environment is presented, in the middle, supervoxels are overlaid onto only traversable regions of the point cloud map. On the right-hand side, the connectivity graph of supervoxels is visualized.

rapid experimentation of the planner’s behavior as presented in our experiments introduced in Subsec. IV-C.

Fig. 4: Resulting feedback motion plans for the run time states (blue) and the propagated states (yellow) setups.

C. Supervoxels for Connected Safe Regions

Point cloud maps are commonly used in 3D robotic applications. It is also the most often used output format for the state-of-the-art SLAM algorithms; our approach builds on point cloud maps. Our approach explores supervoxels [23] as an alternative way of marking reachable regions for navigation over uneven 3D terrains via feedback motion planning. Supervoxels were proposed as an intermediate representation for a dense underlying point cloud to lower the computational requirements by segmentation algorithms, which are analogous to superpixels for image segmentation. Supervoxels are suitable for navigation functions described in Subsec. III-A, as they provide a connectivity graph over the reachable regions of a point cloud map. We refer you to [23] for further details on supervoxel construction for point cloud segmentation. Our previous work established point clouds as an environment representation for navigation over uneven terrains [3]. As a result, a traversability measure was regressed across the point clouds, allowing reachable portions of the point cloud to be derived. We regress the cost values to each local terrain patch, and this cost defines a traversability measure for each patch. We select three critics to determine a cost value associated with each local patch:

- Tilt of the slope within a patch.
- Average point deviation from the plane of a patch used to determine roughness.

Fig. 5: (a) A case where navigating to a pose fails due to jerks in control. Start pose $X_S = (0, 0, 3.14)$ and goal pose $X_G = (10, 0, 0)$. (b) control succeeds with new discrete cLf.

Fig. 6: Cost values are encoded with RGB channels to the terrain point cloud maps.

- Ground clearance to the robot’s bottom chassis.

For example, in Fig. 3 red points on the left picture are designated as non-traversable regions, and no supervoxels are created on these regions. Given the connectivity graph of supervoxels, it is straightforward to compute cost-to-go over state space $X$ with Dijkstra’s algorithm working backward from $X_G$. Given the cost-to-go over the regions and cLf derived in Eq. 7 and Eq. 8, we establish an optimal navigation function $\theta$. We illustrate the process of computing cost-to-go over the reachable regions in Fig. 7, which the cost-to-go value from each supervoxel to $X_G$ is computed over a small region. The area covered by a supervoxel is controlled with two parameters, seed resolution $p_{sr}$ and resolution $p_r$; we discuss these parameters further in Sec. IV
IV. Experiments

We conduct experiments in 3D uneven terrains with varied slopes and roughness in both simulated and real environments. Buildings, poles, trees, and other items are examples of objects in the environment. The maps are around 300x300 meters in size. The simulated environment consists of more steep hills and more significant variance in terms of inclination. Our method relies on point cloud maps for the occupancy information of environments. Hence we construct point cloud maps of real environments with a SLAM method named LIO-SAM [23]. For the real environment map construction, we use an Ouster OS1-64 LiDAR and MTi-30-2A8G4 Xsens IMU, while for simulation, we use Gazebo [14] simulator and simulated sensors. We extract point cloud maps from Gazebo simulated environment through a software plugin. As for the robot platform, we use the Thorvald II [11] robot to carry out experiments. We consider two metrics: the length of the resulting paths and the execution time. Tab. I is created with these two evaluation metrics in mind.

Throughout the experiments, we denote a robot pose with \((x, y, \sigma)\) where the \(x, y\) are the position in meters and \(\sigma\) is the yaw angle in radians. Since we operate in 3D, there is an additional \(z\) component. However, this state is not directly controlled. Therefore we do not include it in the navigation pose. \(z\) is computed by \(x\) and \(y\) with a function; \(z = f(x, y)\).

A. Qualitative Performance Evaluation

In the following, we qualitatively assess the performance of the proposed method with experiments in real and simulated environments. For instance, in [Fig. 1] the robot’s start pose is at \((35.2, -11.6, 3.1)\) and goal pose is at \((-16.0, 26.0, 0.2)\). The robot arrives at the destination successfully; the path depicted in blue is the resultant feedback plan. Similarly, in [Fig. 9] the robot navigates from a similar start pose (as in [Fig. 1]) but to a closer goal pose, resulting in a near-optimal path depicted in blue color. For [Fig. 1] and [Fig. 9] we refer to optimality in sense of Euclidean distance. However, with the availability of traversability costs (represented with RGB colors in point cloud maps) introduced in our previous work [3], we can re-adjust the sense of optimality to refer to the traversability cost in point cloud maps. This means that our feedback motion planning algorithm can be conveniently configured such that it can navigate robot with one of the following properties:

- least inclined path.
- shortest path.
- least rough path.

In [Fig. 8] (a), we configure the cost term \(l(x, u)\) of Eq. 2 such that it accounts for traversability costs. The resulting motion plans avoid higher-cost regions, resulting in an optimal motion plan where less-inclined, less-rough terrain segments are preferred rather a short distance. Hence the resultant motion plans tend to navigate through greener regions. This allows the robot to navigate through inclined terrains safely by avoiding steep hills and rough terrain patches. In the scenario depicted in [Fig. 8] the elevation variance between start and goal poses is approximately 24 meters. Therefore, it is crucial for robots not to shortcut the path but also to consider other factors such as inclination. Within the same map, we select euclidean distance as the objective and run the same experiment for the same start-goal poses. For the euclidean distance objective, we achieve a much shorter feedback plan as depicted in [Fig. 8] (b).

B. Tuning Parameters

Our algorithm’s tuning is simple as only a few configuration parameters exist. For supervoxels, two parameters, namely seed resolution \(p_{sr}\) and resolution \(p_r\) are used. These two parameters specify the size of supervoxels (see [Fig. 7]) and they affect the overall smoothness of the feedback plan, in experiments we find the best values for these parameters to be \(p_{sr} = 1.0\) and \(p_r = 0.4\) for the used robot model. In Eq. 7 and Eq. 8 we introduced control gain parameters \(k_1\) and \(k_2\). A higher value of \(k_1\) leads to more lazy behavior (longer paths), while a higher value of \(k_2\) leads to quicker convergence towards the goal. We use the term lazy as the feedback plans with high \(k_1\) values tends to be lengthier and non-straight. Given these observations, we choose \(k_2\) to be higher than \(k_1\), resulting in more straight feedback plans. In the presented results, \(k_1\) was chosen as 1.0 and \(k_2\) 5.0. We also provide customizable objective selection in our software implementation to optimize ensuing feedback plans, allowing us to achieve plans with the various characteristics itemized above.

C. Comparison to Sampling-based planners

As baselines, we choose some of the state-of-the-art sampling-based planners. These planners have been evaluated in high-dimensional spaces and proven effective and practical in difficult real-world situations. We argue that the proposed method is more efficient than existing sampling-based planners. Hence it is critical to test its performance against them. Our previous work investigated the efficacy of sampling-based planners for uneven 3D terrains [4]. We compare the proposed method to top performers such as AnytimePathShortening (APS), CFOREST, AIT*, and others based on the findings in our previous work [4]. We create a benchmark of planning
problems consisting of 20 unique planning problems. Each planner is requested to solve 20 different planning problems five times, resulting in a total of 100 runs. The results of 100 runs are presented in Tab. I. Although we run a dozen successful experiments on a real robot in a real environment (see Fig. 8 and Fig. 9), it is laborious to perform 100 experiments of long navigation (over 40 meters) tasks on a real for a benchmark. To speed up the process and increase the number of experimental samples, we use a state propagation setup described at the end of Subsec. III-B.

Our strategy consistently produced shorter paths in significantly less time. Optimal sampling-based planners require a timeout parameter where they use all allowed time to construct a valid path with minimal cost, in this case, shortest-length. To observe the behavior of sampling-based planners with different time constraints, we set the timeout to 10 and 20 seconds consecutively, see Tab. I. The planners can improve their performance due to the available time in the 20-second setup. To establish a baseline, we run the APS planner for 60 seconds; our method almost achieves APS’s performance in less than one second. The planning problems were randomly generated from both real maps (see Fig. 1) and simulated maps (see Fig. 8).

### Table I: Means of acquired path lengths with 10 sec—timeout from 100 runs. A lower mean value indicates that planners achieve better performance for both metrics (shorter path in a shorter time).

| Planners   | Length Mean (20 sec.) | Length Mean (10 sec.) |
|------------|-----------------------|-----------------------|
| RRT*       | 65.2 ± 9.6            | 75.7 ± 13.9           |
| PRM*       | 51.2 ± 2.2            | 67.0 ± 12.0           |
| AIT*       | 52.0 ± 3.7            | 58.8 ± 8.3            |
| CFOREST    | 51.6 ± 3.2            | 58.3 ± 6.9            |
| APS        | 50.7 ± 1.8            | 54.1 ± 6.1            |
| APS (60 sec.) | 48.7 ± 0.5         | -                     |
| Ours (0.65 sec.) | 49.5 ± 1.3        | -                     |

The results in Tab. I show that the proposed approach provides shorter paths in significantly less time. More specifically, the approach generates 8.56% shorter paths than the next best-performing planner (APS) in just 6.56% of the time consumed by sampling-based planners. We observe that the best-performing baseline planner (APS) requires more than 40 seconds to outperform the proposed method. The proposed method consistently generates kinodynamically feasible shorter paths in less than a second. The planning problems were randomly generated from both real maps (see Fig. 1) and simulated maps (see Fig. 8).

### V. Conclusion

This paper introduced a new optimal feedback motion planning approach based on supervoxels, cLf, and a navigation function. We demonstrated the method’s efficient performance on a real robot in an uneven 3D environment and its superior performance to some of the state-of-the-art sampling-based planners through extensive experiments. We demonstrated that our method is highly efficient and adaptable to real-world scenarios. The method guides a robot across uneven 3D terrains with various objectives, such as taking the shortest or least-inclined path. A limitation of the proposed algorithm is that the robot gets too close to obstacles in sharp corners as the obstacle clearance is not explicitly incorporated into the algorithm. We plan to extend our approach to deal better with obstacle clearance in the future. We also plan to integrate dynamic obstacles into the scene for better use of already available instantaneous state feedback.
