Driving on Point Clouds: Motion Planning, Trajectory Optimization, and Terrain Assessment in Generic Nonplanar Environments

Philipp Krüsi, Paul Furgale, Michael Bosse, and Roland Siegwart
Autonomous Systems Lab, ETH Zurich, 8092 Zurich, Switzerland
e-mail: philipp.kruesi@mavt.ethz.ch, paul.furgale@mavt.ethz.ch, mike.bosse@mavt.ethz.ch, rsiegwart@ethz.ch

Received 27 August 2015; accepted 11 October 2016

We present a practical approach to global motion planning and terrain assessment for ground robots in generic three-dimensional (3D) environments, including rough outdoor terrain, multilevel facilities, and more complex geometries. Our method computes optimized six-dimensional trajectories compliant with curvature and continuity constraints directly on unordered point cloud maps, omitting any kind of explicit surface reconstruction, discretization, or topology extraction. We assess terrain geometry and traversability on demand during motion planning, by fitting robot-sized planar patches to the map and analyzing the local distribution of map points. Our motion planning approach consists of sampling-based initial trajectory generation, followed by precise local optimization according to a custom cost measure, using a novel, constraint-aware trajectory optimization paradigm. We embed these methods in a complete autonomous navigation system based on localization and mapping by means of a 3D laser scanner and iterative closest point matching, suitable for both static and dynamic environments. The performance of the planning and terrain assessment algorithms is evaluated in offline experiments using recorded and simulated sensor data. Finally, we present the results of navigation experiments in three different environments—rough outdoor terrain, a two-level parking garage, and a dynamic environment, demonstrating how the proposed methods enable autonomous navigation in complex 3D terrain.
© 2016 Wiley Periodicals, Inc.

1. INTRODUCTION

Autonomous navigation in generic, large-scale outdoor environments is essential for safe and efficient operation of robotic systems in many challenging application scenarios. It enables mobile robots to execute or assist in tasks that are repetitive, exhausting, or even dangerous for humans, such as search and rescue, surveillance, exploration, inspection, or transportation. Alongside localization and mapping, motion planning is a key function of any autonomous navigation system. In a generic form, motion planning can be defined as the problem of computing a feasible path connecting two states in the state space of a robotic vehicle. In the case of a ground robot, it enables to automatically find a path guiding the vehicle over the terrain to specified goal pose, while avoiding collisions and other hazards.

The motion of any ground robot is constrained by two factors: the shape of the terrain and the vehicle's physical properties. Neglecting these constraints, i.e. assuming the terrain to be flat and the robot to be holonomic and free of kinematic constraints, is a common approach to simplify the motion planning problem. In state-of-the-art works for outdoor navigation, even when specifically targeted at application in challenging terrain, the three-dimensionality of the terrain and the vehicle's constraints are at best incorporated in planning short, local trajectories, while planning on the large, global scale is conducted under greatly simplifying assumptions (Kelly et al., 2006). However, trajectories that have been computed without consideration of the terrain geometry and the vehicle's limitations can neither be guaranteed to be feasible (safely executable by the robot), nor to be optimal according to any criteria, such as path length or traversal time. The reason for this lies in the fact that the actual, three-dimensional (3D) geometry of such a trajectory is only revealed at execution time, and thus differs from the shape of the trajectory computed by the motion planner, even under the assumption of perfect path-following control.

In this paper, we propose a novel, practical approach to motion planning for nonholonomic ground robots that addresses these issues. Our method is targeted at navigation in generic nonplanar environments, with a focus on rough, unstructured outdoor terrain. It enables planning trajectories of any length that are compliant with curvature and continuity constraints up to a selectable level of

Direct correspondence to: Philipp Krüsi, e-mail: philipp.kruesi@mavt.ethz.ch

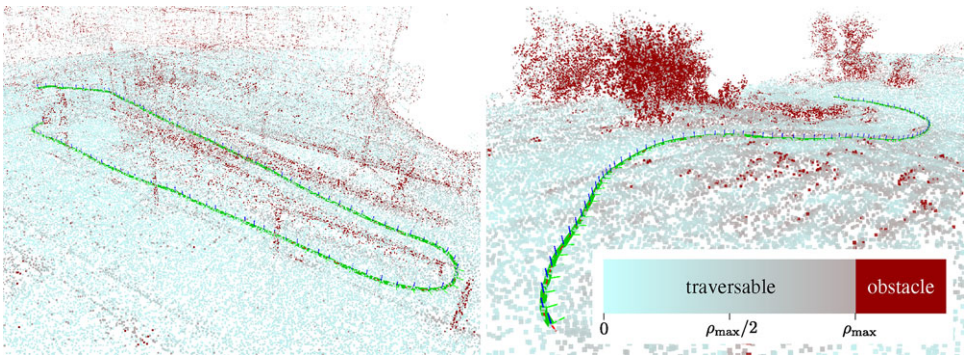

Figure 1. Two example results of our motion planning and terrain assessment framework. The figures show point cloud maps of two different environments, wherein the points are colored according to the estimated terrain roughness, ρ, which is defined in Section 5.2. Points with an assigned roughness value above a certain threshold, ρ_{max}, are considered as obstacles and plotted in dark red. The trajectory on the left is computed in a two-story parking garage, where the start and the goal are located on different levels. On the right, a trajectory is planned from the bottom to the top of a small hill in unstructured, rough terrain.

approximation. Moreover, it inherently considers the shape of the terrain by representing trajectories in the full six-dimensional (6D) space of translations and rotations. Our approach can be seen as a compromise between complex physics-based planning methods (e.g., Howard & Kelly, 2007; Iagnemma, Genot, & Dubowsky, 1999) and highly simplified approaches (such as planning on two-dimensional (2D) grid maps). We approximate the robot as a single rigid body whose motion is (i) subject to certain kinematic constraints, and (ii) restricted by the 3D geometry and the static traversability of the terrain. This allows to incorporate the most important physical properties of the robot and the terrain, while keeping the complexity low enough to perform global planning over considerable distances in short time.

At the core of the proposed methodology is the idea of planning trajectories directly on (unordered) 3D point clouds acquired with range sensors (as shown in Figure 1), avoiding any kind of supplementary terrain reconstruction process, such as computing a height or voxel map, or a mesh representation of the environment. To the best of our knowledge, this is the first work to develop a framework for system-compliant motion planning in true 3D environments using solely point cloud maps. Rather than explicitly reconstructing the 3D terrain and building traversability maps, we propose a *local* terrain assessment method, computing the geometry and the traversability of robot-sized terrain patches *on demand* during motion planning, based on the local distribution of points in the map. This approach has three main advantages: first, point cloud maps are easy to obtain from any kind of range sensor, such as lasers or stereo cameras, often even as a by-product of a localization module. In our robotic setup, we reuse maps built for localization by iterative closest point (ICP) matching. Second, point cloud maps are easy to maintain—map updates simply consist in adding or removing points—and they can be split and merged with minimal computational effort. Third, our point-based terrain assessment method is free of any artificial discretization, which is inherent to classical terrain models, such as digital elevation maps or voxel grids. Instead, we only use the "natural" discretization arising from sensing the terrain surface with a range sensor.

In this work, we consider motion planning as the problem of finding a feasible and in some respects optimal trajectory between a start and a goal pose, given a point cloud map of the terrain, containing both the start and the goal position. Trajectories are represented as dense chains of 6D poses connected by short, planar paths, which are continuous in position, heading, and curvature. Imposing suitable boundary conditions for each of these short segments renders the entire trajectory continuous in heading and enables the planner to account for constraints such as limited linear and angular acceleration. Furthermore, we can incorporate limitations on the turning radius of a nonholonomic vehicle by imposing a limit on the maximum curvature of each trajectory segment. The actual motion planner is a multistage approach. We compute an initial, collision-free trajectory using RRT*, a variant of the rapidly exploring random tree (RRT) algorithm that has been proven to produce asymptotically optimal paths (Karaman & Frazzoli, 2011). For refinement of the result, we propose a novel approach to local trajectory optimization that does not require any gradient information, nor explicit obstacle mapping. The method iteratively optimizes trajectories according to a custom objective function, which can include any quality measures, such as path length, curvature (steering action), or terrain smoothness, while strictly considering all hard constraints.

Since our motion planning framework relies only on local information at any stage of the process (tree expansion, local optimization), we do not have to make any assumptions on the complexity and the topology of the terrain. Unlike existing approaches to outdoor motion planning, which typically expect a 2.5-dimensional (2.5D) geometry

of the terrain, our method enables planning on generic two-dimensional (2D) manifolds (surfaces) embedded in the 3D space. For example, it allows to plan trajectories in multilevel environments or even inside pipe systems for inspection (e.g., using a robot with magnetic wheels such as the *Magnebike*; Tâche et al., 2009), namely without any modifications and without explicit extraction of the environment's topology.

We are convinced that it is impossible to strictly separate motion planning from upstream and downstream processes in autonomous navigation. The nature and the quality of sensing, localization, and mapping largely influence the possibilities of the planner and the suitability of specific techniques. On the other hand, the requirements on the motion control module are highly dependent on the quality of the trajectories computed by the motion planner. In this paper, we develop and discuss the entire data processing pipeline surrounding the motion planning module, starting from sensing the environment to localization, mapping, and finally motion control. We highlight all the different steps necessary for enabling a mobile robot to autonomously move in challenging, large-scale outdoor environments. In particular, we present techniques and algorithms to build and maintain high-quality point cloud maps suitable for motion planning, starting from data acquired by a 3D laser range finder on a mobile robot. In prior work, we have developed a teach and repeat (T&R) framework (Krüsi et al., 2015), based on a highly scalable mapping system using ICP. We extend the latter by integrating techniques for high frequency map updates and for classification of points as dynamic or static. Moreover, we integrate inertial measurements for computing more accurate initial guesses for ICP registration and for undistorting laser readings acquired with a rotating sensor on a moving vehicle. The resulting system allows to build accurate terrain maps along routes of any length, and to keep the same up-to-date over any time horizon.

We evaluate our motion planning and terrain assessment approach and the influence of important parameters using real sensor data acquired in a variety of different environments. Finally, we assess the quality of the entire autonomous navigation system in extensive field tests with a wheeled all-terrain robot. We show that our approach enables safe autonomous navigation in various types of static and dynamic environments, ranging from a multilevel parking garage to rough, unstructured outdoor terrain, without requiring any external localization sources such as GPS.

In summary, this paper makes the following contributions:

1. A practical approach to global motion planning for ground robots in 3D environments (Section 4). The planner improves over the state-of-the-art by combining the following properties: it enables (a) planning directly on point cloud maps using only local terrain assessment, without artificial discretization, (b) planning trajectories compliant with curvature and continuity constraints in the full 6D space of robot poses, and (c) planning on generic 2D manifolds embedded in the 3D space without explicit topology extraction.
2. A method for precise local trajectory optimization in nonplanar terrain (Section 4.6). The algorithm does not require computation of gradients nor explicit obstacle mapping, and it inherently respects hard constraints such as terrain contact, traversability, and kinematic feasibility.
3. A terrain assessment method for computation of the 3D terrain geometry and its traversability based on point cloud maps (Section 5). The approach is computationally inexpensive, free of artificial discretization, and suitable for local application only at specific locations.
4. Integration of the proposed methods for planning and terrain assessment into a complete system for autonomous navigation (Section 6), including an accurate and scalable ICP-based localization and mapping framework. The system enables navigation in large-scale GPS-denied off-road environments, in multilevel facilities, and in dynamic environments, which is experimentally validated in field experiments with a wheeled mobile robot (Section 7).

2. RELATED WORK

2.1. Motion Planning in Nonplanar Maps

Although often neglected, we believe that incorporating the 3D shape of the terrain in motion planning is essential for navigation in nonplanar environments. Trajectories computed with the assumption of flat terrain may turn out to be highly suboptimal or even unfeasible when mapped onto the 3D terrain surface. Because of limited computational resources and/or absence of detailed, large-scale terrain knowledge, 3D information is often incorporated at the *local* level only, while planning on the global scale is conducted with greatly reduced complexity (2D grid map, sequence of waypoints, etc.). One of the earliest work in this domain was the planner implemented on the autonomous land vehicle (ALV) by Daily et al. (1988), in which a sophisticated local planner guided the vehicle along a global path defined by waypoints. Similarly, Cherif (1999) employ detailed physical models and graph search in a discretized state space for local trajectory planning in terms of connecting waypoints given by a global planner. Lacroix et al. (2002) compute local motions by projecting a set of candidate motion primitives onto a DEM and evaluating a robot model at discrete locations along these paths. Instead of sampling candidate paths, Kelly & Stentz (1998) and Kelly et al. (2006) propose to sample the space of motion commands. They generate a set of motion alternatives by model-based simulation of the robot on the 3D terrain, which inherently yields trajectories compliant with kinematic and dynamic vehicle

constraints. Howard & Kelly (2007) take this idea one step further and present an algorithm for computing optimal trajectories connecting two robot states in rough, uneven terrain. They search the control space for the optimal control parameters by numerically linearizing and inverting the mapping from control to vehicle response.

In practical applications—with the exception of exploration in a completely unknown environment—at least some information about the 3D terrain shape beyond the local surroundings is typically available, either from sensor readings acquired during previous navigation runs or from external map sources. Considering this information at the level of *global* motion planning may significantly improve the quality of the results for the reasons explained above. A great deal of the work on global planning in uneven terrain is targeted at planetary exploration. Iagnemma et al. (1999) and Ishigami, Otsuki, & Kubota (2013) use physical vehicle models and discretized elevation maps to assess the feasibility of rover configurations along planned paths. Gingras, Dupuis, Payre, & de Lafontaine (2010) and Rekleitis, Bedwani, Dupuis, Lamarche, & Allard (2013) plan trajectories for rovers in triangular meshes, using the potential field approach or graph search, respectively. Apart from ensuring *safety*—avoiding tipover and slipping hazards—knowledge of the 3D terrain shape on the large scale can be exploited to plan *energy-minimizing* paths (Ganganath, Cheng, & Tse, 2015; Rowe & Ross, 1990; Shum, Morris, & Khajepour, 2015; Sun & Reif, 2005).

The motion of most mobile robots is subject to kinematic constraints, such as restricted steering angles and limited acceleration. While the above global planning algorithms disregard these limitations, which may yield paths that the robot is unable to follow, several researchers have developed methods to consider not only the nonplanarity of the terrain, but also kinematic constraints in the planning phase. Early efforts in this domain include the planner of Gaw & Meystel (1986) using an isolines-based map representation, the algorithms of Siméon & Dacre-Wright (1993) and Siméon (1993), and the approach of Shiller & Gwo (1991), modeling the terrain as cubic B patch and computing trajectories represented as B splines. Guo, Parker, Jung, & Dong (2003) and Arvanitakis, Tzes, & Thanou (2013) first compute a low-dimensional path and incorporate vehicle constraints by fitting smooth curves to the latter. T. M. Howard (2009) establishes a *state lattice* (Pivtoraiko & Kelly, 2005) for global path planning, using the rough terrain trajectory generation algorithm of T. M. Howard & Kelly (2007) for connecting the states in the lattice. Randomized search algorithms, such as rapidly exploring random trees (RRT; LaValle, 1998), probabilistic roadmaps (PRM; Kavraki, Svestka, Latombe, & Overmars, 1996) and their asymptotically optimal variants, RRT* and PRM* (Karaman & Frazzoli, 2011), are inherently suitable for integrating kinematic and dynamic constraints in motion planning (LaValle & Kuffner, 2001). To date, in the context of ground robot navigation, these algorithms have mostly been applied to motion planning in *planar* maps.

Our approach is based on a combination of RRT* search and fine-grained local trajectory optimization, and incorporates both the 3D terrain shape and kinematic vehicle constraints. Furthermore, it does not require explicit terrain surface reconstruction, which allows trajectories to be directly planned on point cloud maps using only local terrain assessment. Regarding this set of characteristics, the methods of Kobilarov & Sukhatme (2005) and Teniente & Andrade-Cetto (2013) are most closely related to ours. Kobilarov & Sukhatme (2005) use a PRM planner while locally approximating the terrain surface by small planar patches. Teniente & Andrade-Cetto (2013) propose a "hybrid randomized" planner called HRA*, which expands a tree of kinematically feasible trajectory segments, combining random sampling with a constraint-aware cost-to-goal heuristic. The trajectory segments are computed by forward-simulating a robot model and projecting the resulting planar path segment to a 3D point cloud map.

All planning algorithms discussed above assume a 2.5D terrain geometry, while our approach extends to more complex 3D environments and enables planning on generic 2D manifolds embedded in the 3D space. Three-dimensional motion planning in the *free space* has been studied extensively in the context of aerial and underwater robotics, but the problem of navigating on complex 3D surfaces with ground robots is of a different nature and has received comparatively little attention. However, many promising applications for mobile robots—such as search and rescue, inspection, or cleaning—clearly require planning beyond 2.5D terrain. Motion planning in multilevel environments can be seen as an intermediate step between 2.5D and full 3D planning. The algorithms developed by Kümmerle, Hähnel, Dolgov, Thrun, & Burgard (2009) and Rusu et al. (2009) enable planning in environments with multiple overlapping floors (such as parking garages). However, they require explicit extraction of the topology, while our planner automatically "discovers" the topologies of any environment directly from point cloud maps. Existing approaches for planning on generic surfaces in the 3D space use graph search in 3D grid maps (Colas, Mahesh, Pomerleau, Liu, & Siegwart, 2013; Hertle & Dornhege, 2013; Menna, Gianni, Ferri, & Pirri, 2014; Stumm, Breitenmoser, Pomerleau, Pradalier, & Siegwart, 2012) or triangular meshes (Breitenmoser & Siegwart, 2012) or wavefront expansion in 3D grid maps (Stoyanov, Magnusson, Andreasson, & Lilienthal, 2010) or triangular meshes (Garrido, Malfaz, & Blanco, 2013; Miró, Dumonteil, Beck, & Dissanayake, 2010). Generating and updating these maps can be computationally expensive, especially in dynamic and changing environments. As our planner only requires *local* terrain information at distinct locations, we can avoid explicit 3D reconstruction by assessing the terrain *on demand* directly from point cloud maps, which are simple and inexpensive to

maintain. Moreover, unlike our algorithm, none of these planners rigorously includes curvature and continuity constraints. The same is true for the approach of Liu & Siegwart (2014), which—similar to our method—plans paths directly on point cloud maps.

2.2. Trajectory Optimization

Because of limited computational resources, *global* motion planning over long distances is often conducted under simplifying assumptions, exploring only a small subset of the entire space of potential trajectories. Local trajectory optimization is a powerful tool for precise refinement of global trajectories in terms of smoothness, risk, energy consumption, or other criteria. Several approaches have been proposed for optimization of *planar* trajectories. Quinlan & Khatib (1993) model trajectories as mass-spring systems called *elastic bands*, which are deformed by artificial forces resulting from obstacles and internal tension to yield smooth and collision-free paths. Lamiraux, Bonnafous, & Lefebvre (2004) and Dolgov, Thrun, Montemerlo, & Diebel (2010) iteratively deform trajectories based on artificial potential fields (Khatib, 1986), which are induced by the obstacles in the environment. Lau, Sprunk, & Burgard (2009) represent trajectories as quintic splines to satisfy kinodynamic motion constraints, and alter the spline control points directed by a certain cost function. Connors & Elkaim (2007) optimize cubic spline trajectories that are not collision free in the beginning. New control points are added where collisions with obstacles are detected. The points are iteratively moved perpendicular to the trajectory until the latter is entirely collision free.

Optimization of *nonplanar* trajectories for ground robots has been studied only marginally. Shiller & Gwo (1991) model the terrain as a cubic B patch and trajectories as B splines mapped to the surface. Optimization of these trajectories amounts to minimizing a cost measure defined as a function of the spline control points. More recently, several approaches have been developed for optimization of high-dimensional trajectories, typically targeted at systems with many degrees of freedom moving in free space, such as robotic manipulators. The *elastic strips* approach of Brock & Khatib (2002) extends the idea of the elastic bands (Quinlan & Khatib, 1993), using a potential-field-based approach to modify a candidate path inside a volume of free space surrounding the path. Zucker et al. (2013) (CHOMP) propose the use of covariant gradient descent for continuous refinement of high-dimensional trajectories. Schulman et al. (2014) tackle the same problem with sequential convex optimization. Kalakrishnan, Chitta, Theodorou, Pastor, & Schaal (2011) (STOMP) and Park, Pan, & Manocha (2012) (ITOMP) avoid computation of cost gradients by using stochastic optimization, enabling incorporation of costs, such as hard constraints, for which well-defined derivatives do not exist. These approaches might in principle be applicable to trajectory optimization for navigation in uneven terrain. However, they require a priori knowledge of the environment around the initial trajectory for computation of distance fields or obstacle potentials. This conflicts with our maxim of representing the terrain surface (and hence the shape and location of obstacles) *implicitly* and without discretization. Our trajectory optimization method does not require any kind of obstacle mapping (but only local terrain assessment at distinct poses), nor computation of gradients. Directed by a freely selectable cost function, the algorithm iteratively optimizes nonplanar trajectories in generic 3D environments, while considering all hard constraints: terrain contact, traversability, and kinematic feasibility.

2.3. Terrain Assessment for Navigation

Safe and efficient navigation in nonplanar environments requires knowledge of the terrain shape for reliably predicting future robot poses and planning feasible trajectories. Several different types of maps have been proposed for reconstruction of 3D terrain. In the context of robotic outdoor navigation, the most widely used maps are 2.5D digital elevation maps (DEMs), built from either laser range data (Daily et al., 1988; Ye, 2003) or stereo vision (Kelly & Stentz, 1998; Lacroix et al., 2002), or by using both laser ranging and vision (Wellington, Courville, & Stentz, 2005). While efficient in terms of computation and storage, grid-based elevation maps are inherently unable to represent overhanging structures (trees, bridges) or multistory facilities. Pfaff, Triebel, & Burgard (2007) introduced an extension to DEMs called multilevel surface maps (MLS maps) to overcome these limitations, allowing for navigation inside multi-level buildings (Kümmerle et al., 2009). Motion planning and navigation in *generic* 3D environments require full 3D terrain reconstruction beyond elevation maps and their variants, for modeling of generic 3D geometries. Commonly used approaches serving this purpose are 3D grid maps, also known as voxel maps (Lacaze, Murphy, & DelGiorno, 2002; Stumm, Breitenmoser, Pomerleau, Pradalier, & Siegwart, 2012), and polygonal meshes (Garrido et al., 2013; Rusu et al., 2009).

Our approach fundamentally differs from these methods: instead of explicitly reconstructing the terrain surface, we infer it locally (on demand) from unordered 3D point cloud maps obtained as a by-product of localization, using nearest-neighbor search and plane fitting based on the approach of Hoppe, DeRose, Duchamp, McDonald, & Stuetzle (1992). Hence, we avoid nearly all computational overhead for building and maintaining terrain maps, while being able to model generic 3D geometries. Moreover, our approach yields a *continuous* terrain representation, as it is free of any artificial discretization (which is inherent to most of the classical methods).

Triangular meshes may be seen as most closely related to our point-based representation. However,

computing a mesh from a point cloud can be computationally expensive, and the process has to be (at least partially) repeated whenever new points are measured, which may happen at a high frequency on a real robotic system. Another difficulty with meshes—and similarly with DEMs and 3D grid maps—arises when multiple (overlapping) maps need to be concatenated, or when the relative transformation between two already connected maps changes (for example, after optimizing a topological/metric graph of maps). These operations require potentially expensive recomputation of large parts of the maps. In contrast, our approach is able to handle these cases with minimal computation. Updating and maintaining point cloud maps is comparatively simple and often carried out externally by a localization module. Concatenating two point clouds boils down to appending one vector of points to another. Similarly, a change in the relative transformation between two point cloud maps only requires transforming a set of points into a new coordinate frame. On the other hand, grid-based approaches may be more efficient in terms of memory usage than dense point clouds. However, we experimentally demonstrate that even large point cloud maps can easily be handled by typical computing equipment on a mobile robot (Section 7). Moreover, the aforementioned properties allow to readily split large terrain maps into smaller "submaps" and to merge them again on demand, such that only a fraction of the entire map—the current region of interest—has to be loaded in memory.

Once the *shape* of the terrain is known, the *traversability* of individual surface patches must be estimated. Traversability analysis for outdoor navigation on the basis of laser measurements and/or (stereo) camera images has been studied in great detail over the past decades. Existing approaches can be classified according to the source data into geometry-based methods (Gennery, 1999; Singh et al., 2000), appearance-based techniques (A. Howard & Seraji, 2001; Khan, Komma, & Zell, 2011), and hybrid approaches (Bellutta, Manduchi, Matthies, Owens, & Rankin, 2000; Santana, Guedes, Correia, & Barata, 2011). The algorithms provide a mapping from sensor or map data to a binary or continuous traversability value. This mapping is either parametric (Lacaze et al., 2002; Langer, Rosenblatt, & Hebert, 1994), learned in offline training (Karumanchi, Allen, Bailey, & Scheding, 2010; Peynot, Lui, McAllister, Fitch, & Sukkarieh, 2014), or learned online while exploring the terrain (Krebs, Pradalier, & Siegwart, 2010). Our approach is a parametric algorithm based on geometric data (point clouds). We compute the *roughness* of the terrain directly on the level of individual map points by means of plane fitting, and determine the traversability of specific robot poses—without any discretization—based on the roughness values of all points contained in a vehicle-sized volume. We refer to Papadakis (2013) for a more extensive survey on traversability assessment in general, and review approaches closely related to ours in the following.

Gennery (1999) and Hamner, Singh, Roth, & Takahashi (2008) fit small planes to point cloud data and compute the terrain roughness based on the residuals of the fit. Their algorithms inherently assume a 2.5D geometry of the environment, while our method is applicable on generic surfaces in the 3D space. Gennery (1999) use weighted least squares in an iterative process to determine the plane fit, while Hamner et al. (2008) subdivide vehicle-sized planar patches into smaller cells for traversability analysis. Our algorithm computes traversability in a noniterative fashion and without introducing any artificial discretization. Manduchi, Castano, Talukder, & Matthies (2005) identify obstacles on the level of individual 3D points obtained from stereo vision. Their algorithm operates directly in the image plane and is thus restricted to application for local obstacle avoidance. The same is true for the approach of Santamaria-Navarro, Teniente, Morta, & Andrade-Cetto (2014) based on point clouds obtained from a time-of-flight camera. They detect obstacles on the basis of local surface normals, which requires dense point clouds. Santamaria-Navarro et al. (2014) further present a method for large-scale traversability classification of point cloud maps. Instead of using a parametric mapping from point distributions to traversability, the model for classification of points is *learned* from training data using Gaussian processes. Lalonde, Vandapel, Huber, & Hebert (2006) segment 3D point cloud maps into three different classes (scatter, linear, surface) using "saliency features" based on point cloud statistics inspired from tensor voting (Medioni, Lee, & Tang, 2000). Their approach is to learn a model of the features distribution by fitting a Gaussian mixture model on hand-labeled training data. Using this model, new data are classified online using a Bayesian classifier. Learning-based approaches in general have the advantage of being adaptable to many different types of environments, at the cost of requiring (potentially tedious) offline training with representative data of the targeted terrain. We experimentally demonstrate that our parametric approach yields good results in a wide variety of different environments (indoor and outdoor, structured and unstructured), using a single hand-tuned set of parameters without terrain-specific adaptations.

2.4. Autonomous Navigation in Nonplanar Terrain

The need for systems capable of moving in uneven and rough terrain has been recognized early in the history of mobile robotics, and autonomous navigation in such environments has received considerable attention since. The Stanford Cart (Moravec, 1980) and the autonomous land vehicle (ALV; Daily et al., 1988) were among the first robots in this domain and demonstrated successful navigation in off-road terrain using stereo cameras and laser scanners, respectively. Daily et al. (1988) introduced a hierarchical control architecture consisting of a mission planner, a map-based planner, and a local planner, which inspired a great deal

of subsequent work. With progress of hardware technology and navigation algorithms, autonomy and deliberative intelligence increased. Langer et al. (1994) presented a system capable of navigating over long distances, using a behavior-based approach for motion planning. Local planning based on evaluation of a set of candidate motions (arcs) was combined with a grid-based global planner by Stentz & Hebert (1995).

A considerable part of the work on autonomous navigation in rough terrain focused on rovers for planetary exploration (Goldberg, Maimone, & Matthies, 2002; Helmick, Angelova, & Matthies, 2009; Lacroix et al., 2002; Maimone, Biesiadecki, Tunstel, Cheng, & Leger, 2006; Rekleitis et al., 2013; Woods et al., 2014). Highly limited resources in terms of computation and energy, along with particularly high demands on reliability, typically yield systems moving at low speeds and using comparatively simple techniques for mapping and planning. Stereo vision combined with a local path planner is a common choice for perception of the environment and obstacle avoidance. Autonomous navigation has been successfully employed for exploration of Mars during NASA's Mars Exploration Rovers (MER) mission (Maimone et al., 2006), although only a small fraction of the entire distance was traveled in autonomous mode.

Research on terrestrial outdoor robots addressed increasing speeds and autonomy over larger distances (Kelly & Stentz, 1998; Kelly et al., 2006; Lacaze et al., 2002; Thrun et al., 2006; Urmson et al., 2006). Kelly & Stentz (1998) introduced a predictive control scheme for local motion planning, representing trajectories implicitly in terms of motion commands and using model-based simulation for trajectory generation in a 2.5D terrain. In follow-up work (Kelly et al., 2006), this approach was refined by adding global planning and terrain classification to the system. Equipped with multiple laser scanners and cameras as well as GPS, their robot demonstrated autonomous navigation in a variety of different environments over several kilometers without human intervention.

More recently, machine learning methods have been applied to autonomous off-road navigation (Bajracharya, Howard, Matthies, Tang, & Turmon, 2009; Konolige et al., 2009; Peynot et al., 2014; Sermanet et al., 2009; Silver, Bagnell, & Stentz, 2010; Sofman et al., 2006). The main idea of these approaches is to *learn* mappings from sensor data to terrain traversability or directly to planning costs and behaviors in offline training, or online from experience. Teach and repeat (T&R) can be seen as a simple form of learning a behavior from demonstration: after a manually controlled *teach* run, the robot is able to *repeat* the learned route autonomously. T&R systems have shown to enable long-range autonomous navigation in unstructured, uneven terrain using a single stereo camera (Furgale & Barfoot, 2010) or a laser scanner (McManus, Furgale, Stenning, & Barfoot, 2013). Some recent T&R systems include obstacle avoidance capabilities (Cherubini & Chaumette, 2013; Krüsi et al., 2015), and Stenning, McManus, & Barfoot (2013) have extended the idea by building a network of reusable paths. In general, T&R approaches increase the robustness of the navigation system and decrease the complexity in terms of required sensor equipment, by sacrificing flexibility and restricting the robot to move along previously driven routes.

Similar to T&R systems, our navigation approach requires an initial survey run for mapping of the environment. Afterward, however, the robot is able to plan and move freely within the mapped area, regardless of previously driven routes. This architecture enables navigation in large-scale outdoor environments without requiring any external localization source, such as GPS, but only using an onboard laser scanner and ICP. While laser-based ICP is a popular choice for localization and mapping in planar indoor environments, it has only rarely been applied for navigation in 3D outdoor terrain to date. Kümmerle et al. (2009) presented a system for navigation in a multilevel parking garage, using a 3D laser and ICP in MLS maps. Colas et al. (2013) and Santamaria-Navarro et al. (2014) used articulated 2D laser scanners for building and localizing within 3D point cloud maps based on ICP scan matching, targeted at search and rescue applications and outdoor navigation, respectively. While these approaches generally assume a static environment, our system enables navigation in dynamic and changing environments with three key functionalities: continuous map updates, real-time detection of nearby dynamic obstacles, and adequately fast replanning of trajectories. Moreover, unlike most existing systems, our approach allows for navigation in real 3D environments, such as multistory buildings, without any special handling and without environment-specific adaptations.

3. PROBLEM STATEMENT

Our goal is to plan trajectories for ground robots in 3D environments on the basis of point cloud maps. The focus of this work lies on *global* motion planning, i.e., on computing feasible trajectories connecting two given robot poses (as opposed to local planning while following a given reference path or direction). We formulate the problem in mathematical terms in Section 3.2. Knowledge about the shape and traversability of the terrain is a fundamental prerequisite for solving the planning problem. Instead of regarding terrain assessment as a separate, upstream process, we treat it as an integral part of motion planning by defining a *terrain assessment function* (Section 3.3) that is invoked by the planner *on demand* to obtain the terrain properties at required locations.

3.1. Preliminaries

We represent poses in the 6D space of rigid body translations and rotations by means of 4×4 transformation matrices belonging to the *special Euclidean group* SE(3). A 6D pose expressed in coordinate frame A is given by the position and

orientation of another coordinate frame, B, with respect to A, and denoted by $\mathbf{T}_{\mathrm{AB}}$,

$$\mathbf{T}_{\mathrm{AB}} := \begin{bmatrix} \mathbf{R}_{\mathrm{AB}} & \mathbf{t}_{\mathrm{AB}} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix} \in \mathrm{SE}(3)\,. \tag{1}$$

The translational part of the pose is expressed by $\mathbf{t}_{\mathrm{AB}} \in \mathbb{R}^3$, which is the vector from the origin of A to the origin of B, expressed in A. The rotational part is encoded in the rotation matrix $\mathbf{R}_{\mathrm{AB}}$ belonging to the *special orthogonal group* SO(3):

$$\mathbf{R}_{\mathrm{AB}} := \begin{bmatrix} \hat{\mathbf{x}}_{\mathrm{AB}} & \hat{\mathbf{y}}_{\mathrm{AB}} & \hat{\mathbf{z}}_{\mathrm{AB}} \end{bmatrix} \in \mathrm{SO}(3)\,. \tag{2}$$

$\mathbf{R}_{\mathrm{AB}}$ will rotate a vector from frame B to frame A, and its columns $\hat{\mathbf{x}}_{\mathrm{AB}}$, $\hat{\mathbf{y}}_{\mathrm{AB}}$, and $\hat{\mathbf{z}}_{\mathrm{AB}}$ are unit vectors defining the axes of B expressed in A. In the remainder of the article, we refer to a *robot pose* as a matrix $\mathbf{T}_{\mathrm{MR}}$, describing the position and orientation of a robot-fixed coordinate frame, R, expressed in a map frame, M. Assuming that the robot model allows the definition of a vehicle-fixed terrain contact plane (e.g., given by wheel contact points), R is defined to be attached to the center of this plane. The x-axis of R is defined to be aligned with the robot's forward direction, the y-axis pointing to the left, and the z-axis upward.

We further define a *planning map* as a set of N_{p} points represented as doubles $(\mathbf{p}, \mathbf{d}_{\mathrm{obs}})$, with $\mathbf{p} \in \mathbb{R}^3$ the Cartesian coordinates of the point and $\mathbf{d}_{\mathrm{obs}} \in \mathbb{R}^3$ the observation direction (a vector pointing toward the sensor at the time of observation). The class of planning maps is denoted by $\mathfrak{M}_{\mathrm{pl}}$. We assume the points to be expressed in a *gravity-aligned* coordinate frame, and to be accurate and dense enough to reproduce the terrain in sufficient detail.[1] In Section 6.2, we present a real-time approach for building maps compliant with these requirements using a mobile robot equipped with a 3D laser scanner.

3.2. Motion Planning

We define the motion planning problem as follows. For a given start pose, $\mathbf{T}_{\mathrm{MS}}$, and a goal pose, $\mathbf{T}_{\mathrm{MG}}$, contained in a point cloud terrain map, $\mathcal{M} \in \mathfrak{M}_{\mathrm{pl}}$, compute a connecting trajectory, $\mathcal{T} : \mathbb{R}_{>0} \to \mathrm{SE}(3)$, defined as a function from path length to 6D robot poses. The trajectory is supposed to comply with the following constraints up to a selected level of approximation:

1. Terrain contact: each pose along the trajectory is required to lie on the terrain surface.
2. Static traversability: at each pose along the trajectory, the terrain roughness, ρ, defined as the height of the largest step within the underlying terrain patch, does not exceed the maximum negotiable step height, $\rho_{\max}$:

$$\rho \leq \rho_{\max} \in \mathbb{R}_{>0}\,. \tag{3}$$

In addition, the robot's roll and pitch angles, ψ and θ, do not exceed given bounds:

$$\begin{aligned} |\psi| &\leq \psi_{\max} \in \mathbb{R}_{>0} \\ \theta &\geq \theta_{\min} \in \mathbb{R}_{<0} \\ \theta &\leq \theta_{\max} \in \mathbb{R}_{>0}\,. \end{aligned} \tag{4}$$

3. Nonholonomic motion constraint: the trajectory can be followed without moving sideways.
4. Curvature: at any point along the trajectory, its curvature, κ, remains below a given limit (the inverse of the minimum turning radius):

$$|\kappa| \leq \kappa_{\max} \in \mathbb{R}_{>0}\,. \tag{5}$$

5. Continuity: there exists a constant, non-zero forward velocity at which the trajectory can be traversed with a *continuous* angular velocity profile. The latter approximately describes the actuation of a real robotic vehicle, where the steering angle or the differential wheel speed cannot change instantaneously.

3.3. Terrain Assessment

Since any ground robot is constrained to move on the terrain surface, three out of six degrees of freedom of its pose are controlled by the local geometry of the terrain. We define the terrain assessment problem as follows. For a given 6D query pose and a specific robot model,

1. find the *closest* 6D robot pose that lies on the terrain surface, and
2. compute a measure of the (static) traversability of the terrain at the resulting pose.

Traversability is defined to depend on both the *roughness* and the *inclination* of the terrain, and to incorporate the static traversability constraints (Eqs. (3) and (4)). Formally, terrain assessment can be defined as a function

$$f_{\mathrm{ta}} : (\mathcal{M}, \mathbf{T}_{\mathrm{MR}}) \mapsto (\mathbf{T}_{\mathrm{M\tilde{R}}}, \tau)\,, \tag{6}$$

with $\mathcal{M} \in \mathfrak{M}_{\mathrm{pl}}$ a planning map, $\mathbf{T}_{\mathrm{MR}} \in \mathrm{SE}(3)$ the query pose, $\mathbf{T}_{\mathrm{M\tilde{R}}} \in \mathrm{SE}(3)$ the resulting pose induced by the terrain, and $\tau \in [0, 1]$ the traversability estimate at this pose (0: untraversable, >0: traversable, 1: maximally traversable). For a given query pose, $\mathbf{T}_{\mathrm{MR}}$, we define the *closest robot pose that lies on the terrain surface*, $\mathbf{T}_{\mathrm{M\tilde{R}}}$, as follows (cf. Figure 2):

1. the origin of $\tilde{\mathrm{R}}$ is the closest point to the origin of R on the z-axis of R that lies on the terrain surface,
2. the z-axis of $\tilde{\mathrm{R}}$ is given by the surface normal $\mathbf{n}$ at the origin of $\tilde{\mathrm{R}}$, and

[1] If the map is too sparse or too noisy, small obstacles cannot be detected. Hence, the required density is dependent on the robot's obstacle negotiation capabilities, defining the minimum size of objects that must be detectable. Similarly, the map must not be polluted by noise larger than the height of traversable terrain discontinuities.

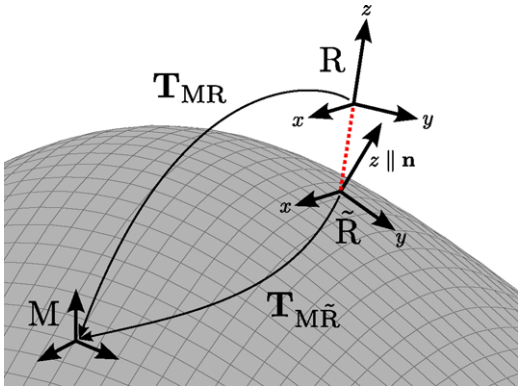

Figure 2. Definition of the geometric part of the terrain assessment problem. For a given query pose, $\mathbf{T}_{\mathrm{MR}}$, we seek $\mathbf{T}_{\mathrm{M\tilde{R}}}$, the closest robot pose on the (unknown) terrain surface along the z-axis of the coordinate frame defining the query pose (R). The poses are expressed as transformation matrices from a robot-fixed coordinate frame (R and $\tilde{\mathrm{R}}$) to the map frame (M).

3. the remaining rotational degree of freedom is determined by the condition of zero yaw between R and $\tilde{\mathrm{R}}$.

We subdivide the terrain assessment function f_{ta} into two separate functions, computing the *geometry* and the *traversability* of the terrain, respectively:

$$f_{\mathrm{ta}}^{\mathrm{p}} : (\mathcal{M}, \mathbf{T}_{\mathrm{MR}}) \mapsto \mathbf{T}_{\mathrm{M\tilde{R}}}$$
$$f_{\mathrm{ta}}^{\tau} : (\mathcal{M}, \mathbf{T}_{\mathrm{M\tilde{R}}}) \mapsto \tau\,. \tag{7}$$

In a first step, $f_{\mathrm{ta}}^{\mathrm{p}}$ computes the terrain-compliant 6D robot pose. Once this pose is found, f_{ta}^{τ} estimates the traversability of the corresponding terrain patch for the given robot model.

4. MOTION PLANNING

In this section, we develop a motion planning framework solving the problem defined in Section 3.2. Our planner computes trajectories directly from unordered 3D point cloud maps by embedding a local terrain assessment method, f_{ta}, as defined in Section 3.3 (the implementation of the latter will be described in Section 5). Trajectories are represented as piecewise continuous functions in the full 6D space of robot poses. The proposed planning method allows computing trajectories over any distances and in environments with highly nonconvex obstacle configurations. The approach is not restricted to 2.5D environments, but extends naturally to generic environments that can be described by 2D manifolds (surfaces) embedded in the 3D space, including multilevel environments or pipe systems.

4.1. Trajectory Representation

The choice of a suitable trajectory model is an important component of a motion planning algorithm. In particular, the model is required to map anticipated robot movements in sufficient detail, such that the constraints defined in Section 3.2 can be enforced. We define a *planner trajectory* as a chain of 6D poses connected to each other by short, planar trajectory segments. As illustrated in Figure 3, a planner trajectory is represented as a sequence of N_{n} *nodes*, $\mathcal{N}_i$, each comprising a 6D pose attached to the terrain surface, $\mathbf{T}_{\mathrm{M\tilde{R}}_i}$, a traversability value, τ_i, a path curvature value, κ_i, and a parameter vector, $\mathbf{t}_i$, defining the planar trajectory connecting the pose to the pose of the next node in the chain. The distance between any two successive nodes is defined to be less than the robot's length. We denote the class of planner trajectories by $\mathfrak{T}_{\mathrm{pl}}$ and define it as follows:

$$\mathfrak{T}_{\mathrm{pl}} := \left\{ (\mathcal{N}_i)_{i=1}^{N_{\mathrm{n}}} \right\}$$
$$\mathcal{N}_i := (\mathbf{T}_{\mathrm{M\tilde{R}}_i}, \tau_i, \kappa_i, \mathbf{t}_i)\,. \tag{8}$$

The poses of the first and the last node of a trajectory $\mathcal{T} \in \mathfrak{T}_{\mathrm{pl}}$ are determined by the projection of the user-defined start and goal poses on the terrain surface,

$$\mathbf{T}_{\mathrm{M\tilde{R}}_1} = f_{\mathrm{ta}}^{\mathrm{p}}(\mathcal{M}, \mathbf{T}_{\mathrm{MS}}),$$
$$\mathbf{T}_{\mathrm{M\tilde{R}}_{N_{\mathrm{n}}}} = f_{\mathrm{ta}}^{\mathrm{p}}(\mathcal{M}, \mathbf{T}_{\mathrm{MG}})\,. \tag{9}$$

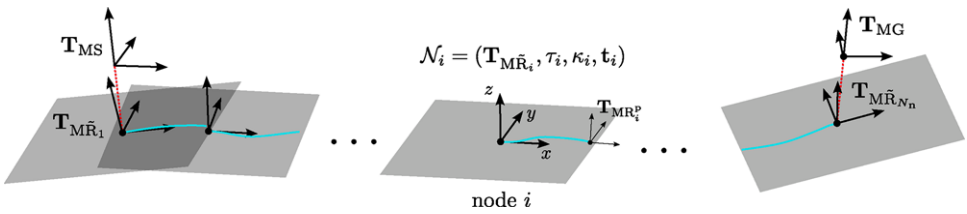

Figure 3. Representation of trajectories for system-compliant motion planning in nonplanar terrain. A *planner trajectory* is defined as a sequence of *nodes*, each defined by a 6D pose, $\mathbf{T}_{\mathrm{M\tilde{R}}_i}$, a traversability value, τ_i, a path curvature value, κ_i, and a planar trajectory connecting the pose to the pose of the next node, $\mathbf{t}_i$, represented as a cubic curvature polynomial in a planar patch locally approximating the terrain surface. Keeping the distance between the nodes sufficiently short (smaller than the robot's length) and defining suitable boundary conditions for the connecting trajectory segments allows to enforce the trajectory to satisfy kinematic vehicle constraints.

The planar connection trajectories between the nodes are represented as cubic curvature polynomials in arc length s, describing trajectories continuous in position, heading, and curvature:

$$\kappa(s) = \kappa_0 + as + bs^2 + cs^3,\ \kappa_0, a, b, c \in \mathbb{R}\,. \tag{10}$$

A trajectory segment of length $s_f \in \mathbb{R}_{>0}$ is defined by a parameter vector

$$\mathbf{t} := \begin{bmatrix} \kappa_0 & a & b & c & s_f \end{bmatrix}^{\mathrm{T}}. \tag{11}$$

As shown by Nagy & Kelly (2001), cubic curvature polynomials inherently satisfy the nonholonomic motion constraint, and they feature sufficient degrees of freedom to connect any two states $\xi_{a,b} = [x, y, \phi, \kappa]^{\mathrm{T}}_{a,b}$ in the space of planar position, heading, and curvature. The boundary conditions at the initial state are implicitly satisfied by representing the trajectory segment in a coordinate frame centered at ξ_a, and choosing $\kappa_0 = \kappa_a$. The other four trajectory parameters are determined by the remaining four constraints, $[\Delta x, \Delta y, \Delta\phi, \kappa_b]^{\mathrm{T}}$, representing the state ξ_b transformed to the coordinate frame centered at ξ_a.

The planar trajectory $\mathbf{t}_i$ between $\mathcal{N}_i$ and $\mathcal{N}_{i+1}$ is computed in the x-y plane of the coordinate frame $\tilde{\mathrm{R}}_i$ attached to the pose $\mathbf{T}_{\mathrm{M}\tilde{\mathrm{R}}_i}$, which approximates the terrain surface at $\mathcal{N}_i$. In uneven terrain, this results in a discontinuity between the pose at the end of $\mathbf{t}_i$, $\mathbf{T}_{\mathrm{MR}_i^{\mathrm{P}}}$, and the pose of $\mathcal{N}_{i+1}$, $\mathbf{T}_{\mathrm{M}\tilde{\mathrm{R}}_{i+1}}$. However, we can enforce the controllable part of the pose, namely the planar position and the heading, to be continuous at this transition point, by choosing appropriate boundary conditions for $\mathbf{t}_i$. Representing the latter in the frame $\tilde{\mathrm{R}}_i$, the initial state is trivially given by

$$\xi_{i,a} = \begin{bmatrix} 0 & 0 & 0 & \kappa_i \end{bmatrix}^{\mathrm{T}}. \tag{12}$$

The terminal state is obtained by projecting the pose $\mathbf{T}_{\mathrm{M}\tilde{\mathrm{R}}_{i+1}}$ to the x-y plane of $\tilde{\mathrm{R}}_i$ along the z axis of $\tilde{\mathrm{R}}_i$ and selecting the terminal heading, $\Delta\phi$, as the angle between the x axis of $\tilde{\mathrm{R}}_i$ and the x-axis of the projected pose. Finally, the curvature at the terminal point is set to the value of the curvature of $\mathcal{N}_{i+1}$:

$$\mathbf{T}_{\tilde{\mathrm{R}}_i\tilde{\mathrm{R}}_{i+1}} = \mathbf{T}^{-1}_{\mathrm{M}\tilde{\mathrm{R}}_i}\mathbf{T}_{\mathrm{M}\tilde{\mathrm{R}}_{i+1}} =: \begin{bmatrix} r_{xx} & \cdot & \cdot & t_x \\ r_{xy} & \cdot & \cdot & t_y \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

$$\xi_{i,b} = \begin{bmatrix} t_x & t_y & \mathrm{atan2}(r_{xy}, r_{xx}) & \kappa_{i+1} \end{bmatrix}^{\mathrm{T}}. \tag{13}$$

$\mathbf{T}_{\mathrm{MR}_i^{\mathrm{P}}}$ is defined at the terminal point of $\mathbf{t}_i$ on the x-y plane of $\tilde{\mathrm{R}}_i$, with the planar position and heading defined by $\xi_{i,b}$. Hence, the transition from $\mathbf{T}_{\mathrm{MR}_i^{\mathrm{P}}}$ to $\mathbf{T}_{\mathrm{M}\tilde{\mathrm{R}}_{i+1}}$ involves no rotation around the z-axis (continuous heading) and no translation along the x- and y-axes (continuous planar position). The inverse problem, finding $\mathbf{T}_{\mathrm{M}\tilde{\mathrm{R}}_{i+1}}$ given $\mathbf{T}_{\mathrm{MR}_i^{\mathrm{P}}}$, can be solved using the terrain assessment function $f_{\mathrm{ta}}^{\mathrm{P}}$, which inherently satisfies the continuity constraint in planar pose according to its definition in Section 3.3.

It can be shown that, in the limit case of a planar environment, this representation renders the entire planner trajectory, $\mathcal{T}$, C^2 continuous in position, C^1 continuous in heading, and C^0 continuous in curvature. For a constant speed, v, the corresponding rotational velocity, ω, follows directly from the path's curvature ($\omega = \kappa v$), and thus is likewise continuous. Hence, the trajectory is compliant with the continuity constraint defined defined in Section 3.2. Moreover, conformance with the maximum absolute curvature constraint Eq. (5) can be enforced by requiring the (analytically computable) maximum curvature of each individual trajectory segment $\mathbf{t}_i$ to remain below the limit.

For investigating the compliance with the constraints in the general case of a nonplanar trajectory, we make the following smoothness assumption: there exists a distance $r_{\mathrm{s,max}} > 0$, such that for any radius r_{s}, $0 < r_{\mathrm{s}} < r_{\mathrm{s,max}}$, it is possible to find an angle $\gamma^{\mathbf{n}}_{\max}(r_{\mathrm{s}}) \ll \frac{\pi}{2}$, such that the change in the orientation of the robot's z-axis (the surface normal computed by the terrain assessment module) within r_{s} around any point is bounded by $\gamma^{\mathbf{n}}_{\max}(r_{\mathrm{s}})$. Moreover, we assume $\gamma^{\mathbf{n}}_{\max}(r_{\mathrm{s}})$ to decrease for decreasing r_{s}. These are valid assumptions even in the presence of discontinuities in the terrain surface, such as large obstacles or walls, since the surface normal provided by $f_{\mathrm{ta}}^{\mathrm{P}}$ is inherently smoothed according to the robot's size. Hence, choosing the maximum distance between nodes in the planner trajectory—referred to as maximum internode distance, $d_{\mathrm{IN}}^{\max}$—smaller than $r_{\mathrm{s,max}}$ guarantees that the angle between the robot's z-axis and the normal of the planar patch containing the respective closest trajectory segment, $\mathbf{t}_i$, never exceeds $\gamma^{\mathbf{n}}_{\max}(d_{\mathrm{IN}}^{\max})$.

At any point along $\mathcal{T}$, the actual path of the robot when executing the trajectory can be seen as the projection of $\mathcal{T}$ to the plane defined by the local surface normal (given by $f_{\mathrm{ta}}^{\mathrm{P}}$ applied to the pose on $\mathcal{T}$). From the definition of $f_{\mathrm{ta}}^{\mathrm{P}}$ and the boundary conditions of the planar trajectory segments follows that at any point (including the transitions between trajectory segments) the respective projected trajectory is still continuous in position and heading, and thus compliant with the nonholonomic motion constraint. However, the curvature of planar paths changes under projection. Hence, the curvature of the projected trajectories may be discontinuous at the transition points, violating the continuity constraint. Between the nodes, the curvature will be continuous, but it may exceed the limit defined by the curvature constraint. However, it can be shown that the curvature at any point along a generic planar path when projected to a plane rotated by an angle γ at most increases by a factor $\frac{1}{\cos\gamma}$ or decreases by a factor $\cos\gamma$. Consequently, the ratio between the curvatures κ_{act} and κ_{plan} of the actual and the planned

trajectory is confined by the following interval at any point along the trajectory:

$$\frac{\kappa_{\text{act}}}{\kappa_{\text{plan}}} \in \left[\cos \gamma^{\mathbf{n}}_{\max}(d^{\max}_{\text{IN}}), \frac{1}{\cos \gamma^{\mathbf{n}}_{\max}(d^{\max}_{\text{IN}})} \right]. \tag{14}$$

Hence, if the internode distance, $d^{\max}_{\text{IN}}$, is chosen sufficiently short, the maximum surface normal variation, $\gamma^{\mathbf{n}}_{\max}(d^{\max}_{\text{IN}})$, and the resulting maximum curvature discontinuities are sufficiently small to conclude that the trajectory satisfies the continuity constraint *approximately*, in the sense that a close trajectory could be found that would satisfy the constraints exactly. The accuracy of the approximation can be increased or decreased by varying $d^{\max}_{\text{IN}}$. The constraint of a maximum absolute curvature of $\kappa_{\max,\text{act}}$ is guaranteed to be satisfied by imposing a slightly reduced limit in the planning phase:

$$\kappa_{\max} = \kappa_{\max,\text{act}} \cdot \cos \gamma^{\mathbf{n}}_{\max}(d^{\max}_{\text{IN}}) . \tag{15}$$

Selecting $d^{\max}_{\text{IN}}$ shorter than the robot's length, compliance with the static traversability constraints Eqs. (3) and (4) can be imposed by requiring each node in $\mathcal{T}$ to satisfy the specifications. Since the traversability in terms of terrain roughness is computed based on all points within the robot's footprint, checking a finite number of suitably spaced poses for traversability suffices to cover the terrain along the entire trajectory. Similarly, the actual roll and pitch angle limits ($\psi_{\max,\text{act}}, \theta_{\min,\text{act}}, \theta_{\max,\text{act}}$) can be guaranteed not to be exceeded along the entire trajectory by only computing the respective values at each node and imposing reduced limits, accounting for the uncertainty in the surface normal between the nodes:

$$\begin{aligned} \psi_{\max} &= \psi_{\max,\text{act}} - \gamma^{\mathbf{n}}_{\max}(d^{\max}_{\text{IN}}) \\ \theta_{\min} &= \theta_{\min,\text{act}} + \gamma^{\mathbf{n}}_{\max}(d^{\max}_{\text{IN}}) \\ \theta_{\max} &= \theta_{\max,\text{act}} - \gamma^{\mathbf{n}}_{\max}(d^{\max}_{\text{IN}}) . \end{aligned} \tag{16}$$

4.2. Approach Overview

The overall motion planning method, illustrated in Figure 4, comprises three different stages: initial trajectory generation using RRTs, RRT*-based global optimization, and fine-grained local optimization. At any point in the entire process, trajectories are represented as defined above and enforced to satisfy all the constraints imposed by the vehicle and the terrain as defined in Section 3.2.

An initial solution is generated by expanding two RRTs, one from the start pose and one from the goal pose, and taking the trajectory resulting from the first successful connection of the trees (Section 4.4). The algorithm enables fast computation of a system-compliant, collision-free trajectory in generic 3D terrain, without prior knowledge of the topology, since the environment is incrementally explored by locally extending the trees.

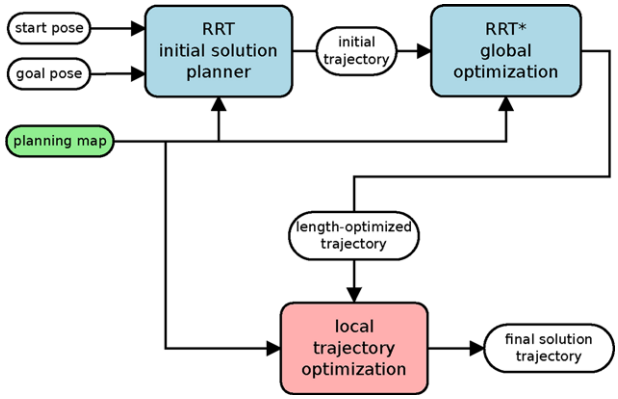

Figure 4. Overview of the motion planning approach. Computing a trajectory connecting two poses involves three consecutive stages: planning of an initial trajectory using RRTs, RRT*-like optimization for path length at the global level, and fine-grained local trajectory optimization according to a user-specified cost function.

However, the resulting trajectory may be suboptimal and contain substantial detours. Hence, we employ an RRT*-like algorithm for optimizing the trajectory in terms of path length on the global level (Section 4.5). Our approach is conceptually similar to the method of Guernane & Achour (2011), allowing modification of the trajectory on the topological level (and thus detection of significant shortcuts), yet without searching the entire configuration space. It can be seen as a compromise between purely local trajectory deformation and planning of a globally optimal trajectory. RRT* is known to yield asymptotically optimal trajectories. In practice, the optimization is carried out until a user-defined termination criterion is reached, representing a compromise between optimality and computation time. Note that RRT* tree expansion—in contrast to our RRT algorithm—requires solving several two-point boundary value problems in every iteration for exactly connecting 6D robot poses, which is a computationally expensive operation, given that the connection trajectories are required to be compliant with all the robot and terrain constraints. Hence, starting with RRT*-based planning from the beginning would increase the time for computation of an initial solution, and thus confine the flexibility in choosing a compromise between computation speed and optimality.

The third stage of our planning method addresses local optimization of the trajectory (Section 4.6). After the RRT*-based optimization on the global level, the trajectory may be reasonably close to the *shortest* path; yet, it may still be suboptimal in terms of steering action (curvature) or risk associated with terrain traversability. While these criteria could theoretically be integrated into the RRT* algorithm, obtaining really precise results would require an unfeasibly high number of iterations. We develop a novel algorithm

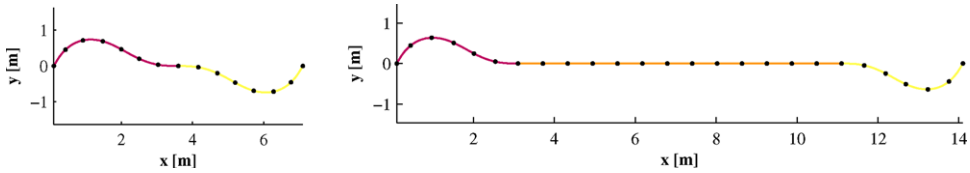

Figure 5. Examples of planar trajectories connecting two poses defined by their position and heading, serving as input for generation of terrain-compliant *planner trajectories* over short distances. The latter are computed by discretizing the planar paths into equally spaced poses (black dots) and projecting them to the terrain surface along the plane normal. Short planar trajectories are represented as single cubic curvature polynomials. For larger distances between the start and the goal pose, one (left) or two (right) intermediate waypoints are inserted to reduce the resulting path length.

for fine-grained local trajectory optimization based on a supplied cost function, typically involving path curvature, length, and terrain roughness. The trajectory is iteratively deformed within the homotopy class of the initial trajectory (the result of the RRT*-based optimization), until a local minimum of the cost function is reached.

4.3. Short-Range Trajectory Generation Based on a Planar Path

Connecting two given 6D poses with a *planner trajectory*, $\mathcal{T} \in \mathfrak{T}_{\text{pl}}$, can be a complex problem, especially if the two poses are far away from each other and the terrain is highly uneven or cluttered with obstacles. However, for short distances (or in smooth and largely traversable terrain), a simple and fast trajectory generation method may be sufficient. In the following, we describe an algorithm for generation of planner trajectories based on projecting a planar trajectory to the 3D terrain surface. The method is a "one shot" approach: first, we generate a trajectory assuming the entire terrain between start and goal pose to be traversable and 2.5D. In a second step, the trajectory is inspected for feasibility and either accepted or declined. We use this approach for two different purposes. First, when generating an initial trajectory using RRTs, for connecting the two trees expanded from the start and the goal pose. Second, in RRT*-based trajectory optimization for computing connecting trajectories between vertices of the tree and newly sampled poses.

The first step of the algorithm consists of projecting the goal pose to the x-y plane of the start pose, and computing a planar trajectory connecting start and goal. We define two distance threshold values, $0 < d_1 < d_2$, and apply three slightly different methods, depending on the distance d_{sg} between start and goal. For short distances ($d_{\text{sg}} < d_1$), the planar trajectory is represented as a single cubic curvature polynomial generated with the method of Nagy & Kelly (2001). For larger distances, this approach can yield suboptimal trajectories in terms of path length, especially for large angles between the heading at either pose and the straight line connecting start and goal. Hence, for $d_1 \leq d_{\text{sg}} < d_2$, we define an intermediate "waypoint" in the middle between start and goal and compute two separate trajectory segments (Figure 5, left). This guides the trajectory closer to the straight-line connection and thus reduces the path length. For long distances, $d_2 \leq d_{\text{sg}}$, the trajectory is split into three parts, represented by two curvature polynomial segments at either end and a straight-line connection in between (Figure 5, right). In any case, the resulting planar trajectory is discretized into a sequence of equally spaced poses: $(\xi_i = [x_i, y_i, \phi_i, \kappa_i]^{\text{T}})_{i=1}^{N_{\text{planar}}}$. The number of discrete poses is chosen such that the distance between the poses is smaller or equal to a selected nominal internode distance, $d_{\text{IN}}^{\text{nom}}$.

Once the planar connection trajectory is established, each pose ξ_i is projected to the terrain surface along the normal of the plane in which the path is represented, using the terrain assessment method f_{ta}. The resulting 6D pose, $\mathbf{T}_{\text{M}\tilde{\text{R}}_i}$, and the traversability estimate, τ_i, represent the respective properties of the sought-after planner trajectory, $\mathcal{T}$, at node i. If all nodes lie on traversable terrain, $\tau_i > 0 \; \forall i \in \{1, \ldots, N_{\text{planar}}\}$, we verify whether the trajectory leads to the correct goal pose in 6D. In non-2.5D environments, this is not obvious; for example, the projected 6D pose at the end of the trajectory could lie on the wrong floor in a multilevel environment. Once this condition is satisfied, the connecting trajectory segments $\mathbf{t}_i$ between each pair of nodes in $\mathcal{T}$ are computed, using the method of Nagy & Kelly (2001). Finally, if the maximum curvature of none of these segments exceeds the threshold $\kappa_{\max}$, trajectory generation is declared as successful. The resulting trajectory is compliant with the vehicle and terrain constraints.

4.4. RRT-Based Initial Trajectory Generation

We generate an initial trajectory connecting the start and the goal pose using RRTs. The resulting trajectory is not yet optimized according to any criterion, but it is collision-free and compliant with the vehicle and terrain constraints. The concept is to generate an initial solution in minimum time and use this as a basis for further optimization according to the available computation time budget.

RRT vertices are defined as 6D poses with attached curvature and traversability values, computed using the terrain assessment method f_{ta}. Vertices are connected among each other by short planar trajectory segments as defined

in Eq. (11), computed to satisfy the boundary conditions given in Eqs. (12) and (13). Hence, the path from the tree's root to any vertex represents a *planner trajectory* according to the definition in Section 4.1. We simultaneously grow two RRTs rooted at the start and the goal pose, which is a standard approach for RRT-based single query motion planning (LaValle & Kuffner, 2001). Whenever two vertices of different trees are close to each other, an attempt is made to connect the trees. The algorithm terminates as soon as a feasible connection is found. Expansion of the trees is conducted according to the standard RRT procedure, essentially consisting of sampling points in the search space and attempting to expand the trees into the direction of the samples. In the following, we describe the peculiarities of our planner with regard to the separate parts of the RRT algorithm.

4.4.1. Sampling Strategy

The strategy for sampling points in a given search space is an important factor for the performance of an RRT-based planner. Randomly sampling points in the state space yields good coverage of the entire environment; yet, finding a path to a specific goal state may be slow. LaValle & Kuffner (2001) propose to bias the samples toward the goal state (or toward the other tree in the case of bidirectional planning), by sampling a fraction of the points from the goal state. In our experiments, this approach yielded good results in general, yet performed poorly in cases where start and goal pose were physically close but could only be connected via a large detour due to the special topology of the environment. A typical example are planning queries in multilevel environments with start and goal at the same location on different floors: despite the two poses being only a couple of meters apart, a connecting trajectory may be as long as several tens or even hundreds of meters, and finding this trajectory requires to explore the environment in all directions.

Instead of sampling randomly from the entire state space or biasing the tree expansion toward the goal, we sample random instances of the points in the planning map $\mathcal{M}$. This approach is motivated by two factors. First, by definition the map points span the entire part of the environment that can be used for planning a trajectory. Expanding the tree beyond the map is not possible, since there is no knowledge about the terrain outside the map boundary. Second, in practical applications, the map $\mathcal{M}$ used for a specific planner query can be extracted from a large-scale terrain map such that it spans only a certain area between the start and the goal of the query, which naturally biases the expansion of each tree toward the respective opposite pose. In our experiments, we have found this sampling strategy to yield good performance in various different types of environments, both in terms of computation time and shortness of the resulting trajectories.

4.4.2. Tree Expansion

After sampling a point $\mathbf{p}_{\text{samp}}$ from the map, the tree is searched for the vertex v_{near} that is closest to $\mathbf{p}_{\text{samp}}$ in terms of Euclidean distance in the 3D position space. Subsequently, we attempt to expand the tree at v_{near} into the direction of $\mathbf{p}_{\text{samp}}$. We avoid any potentially expensive trajectory generation or forward simulation of the vehicle during tree expansion by precomputing a fixed set of candidate planar trajectory segments. Expanding the tree amounts to appending these trajectories at the pose of v_{near} and choosing the most suitable segment in terms of alignment of the trajectory with the line between the v_{near} and $\mathbf{p}_{\text{samp}}$. Defining the expansion segments to be sufficiently short relative to the robot's dimensions, a new tree vertex can be computed by applying the terrain assessment algorithm f_{ta} at the endpoint of the trajectory segment, yielding the 6D pose and the traversability value of the new vertex. If the terrain at this pose is traversable, the new vertex is added to the tree. In any case, the selected trajectory segment is marked as unusable for future expansions of the tree at v_{near}. Once all trajectory segments at a vertex v_i are marked as unusable, v_i is discarded entirely in subsequent tree expansion steps.

The set of expansion trajectory segments consists of cubic curvature polynomials, computed using the method of Nagy & Kelly (2001). To guarantee continuity in curvature in the entire tree and allowing connections between any two instances of the trajectory set, we define the curvature at the start and at the end of each segment to be zero: $\kappa(0) = \kappa(s_f) = 0$. The trajectories are generated by defining discrete end positions on a circle of radius $r_{\text{exp}} = d_{\text{IN}}^{\text{nom}}$ around the start position and introducing a discretization in the terminal heading angle $\Delta\phi$, with a step with of $\delta\phi$. For each discrete $\Delta\phi$ up to $\Delta\phi_{\max}$, we compute a trajectory to each of the end positions on the circle but only keep the one with the smallest maximum curvature along the path. Because the maximum curvature along the trajectories increases with increasing $\Delta\phi$, $\Delta\phi_{\max}$ is determined by the curvature limit $\kappa_{\max,\text{exp}}$, which is chosen smaller than $\kappa_{\max}$ given by the vehicle constraints, in order to produce smoother trajectories. In summary, we generate a set of curvature-limited trajectories, where each trajectory yields a unique change in heading. The specific trajectory set used in our setup is shown in Figure 6. The parameters $\kappa_{\max,\text{exp}} = 1.6\ \text{m}^{-1}$, $r_{\text{exp}} = 0.6$ m, and $\delta\phi = 0.1$ rad yield a set of 13 trajectory segments with terminal heading angles $\Delta\phi$ between -0.6 rad and 0.6 rad.

4.4.3. Tree Connection

During the expansion of the trees, we maintain a matrix of mutual Euclidean distances between any vertex in the start tree and any vertex in the goal tree. In every iteration, we attempt to connect the two trees at each pair of vertices $(v_{\text{start}}, v_{\text{goal}})$ with a mutual distance below a certain threshold (we use $3r_{\text{exp}}$). We compute a connection trajectory

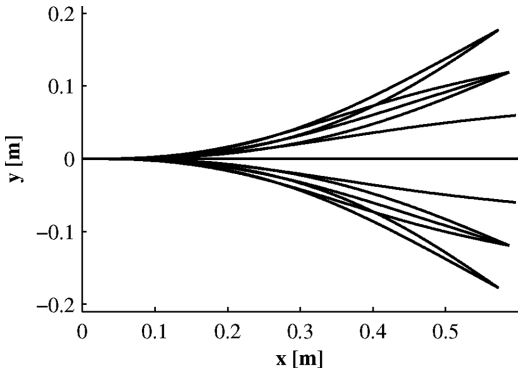

Figure 6. Example of a set of planar trajectory segments for RRT expansion. The curvature of the trajectories is defined to be zero at both the start and the end points for seamless connection of subsequent segments. The maximum feasible change in heading between start and end point, $\Delta\phi_{\max}$, results from an upper bound on the paths' curvature. The trajectories in the set are chosen such that their terminal heading angles are spread over the entire range between $-\Delta\phi_{\max}$ and $\Delta\phi_{\max}$.

with the method outlined in Section 4.3. If it is traversable and compliant with the maximum curvature constraint, the trees have been successfully connected. The final trajectory from the start to the goal pose is obtained as the union of the connection trajectory and the two paths in the trees to v_{start} and v_{goal}, respectively. If the connection fails due to untraversable terrain or unfeasible path curvature, the pair $(v_{\text{start}}, v_{\text{goal}})$ is marked as unfeasible and discarded in future connection trials.

4.5. RRT*-Based Global Trajectory Optimization

The expansion of the RRTs during generation of the initial trajectory is biased toward the respectives goal point by a special sampling strategy (Section 4.4), which in general yields a reasonably short trajectory. Nevertheless, it may contain considerable detours, as—by construction—it is not optimized according to any criteria. We address this issue by including an RRT*-based global trajectory optimization method in the planning process, which is developed in this section. In the following, we only consider the path length as optimization criterion. However, the extension to user-defined cost functions based on traversability, steering action, or other factors is straightforward.

Unlike the classical RRT algorithm, RRT* has been proven to yield asymptotically optimal trajectories (Karaman & Frazzoli, 2011). In essence, RRT* improves over classical RRT by two modifications of the tree expansion process. When adding a new vertex, v_{new}, a set of nearby vertices, V_{near}, is inspected for potential connection to v_{new}. First, the parent of v_{new} is chosen from all vertices in V_{near} with a feasible connection to v_{new} such that the total path cost of v_{new} is minimized. Second, v_{new} is defined as the parent of all vertices in V_{near} that can be reached from v_{new} with a new connection, if the cost of the path via v_{new} is lower than the cost of the previous path. The latter technique is called "rewiring" of the tree.

RRT*-based trajectory optimization starts with initializing the tree from the sequence of vertices defining the trajectory from the start to the goal, as computed by the RRT algorithm of Section 4.4. The RRT* vertices comprise the same properties as RRT vertices, extended by a *cost* value. The latter encodes the path length of the current shortest path in the tree from the start to the respective vertex. Subsequently, the tree is expanded by sampling random points, adding new vertices expanding the tree toward the samples, and applying the RRT* steps (connecting and rewiring), until a user-defined termination criterion is fulfilled. In the following, we describe the characteristics of our approach in detail, referring to the individual steps of the RRT* algorithm.

4.5.1. Sampling Strategy

In every iteration, an attempt is made to expand the tree into the direction of a randomly sampled point. We assume the initial trajectory to represent—at least on the large scale—a reasonable prior for the optimal path. Hence, we confine the sampling of random points to a certain volume around the initial trajectory. After choosing a random vertex of the latter, v_{rand}, a point $\mathbf{p}_{\text{samp}}$ is sampled randomly from a spherical volume around v_{samp}. The radius of this volume is chosen proportional to the distance between start and goal.

4.5.2. Tree Expansion

After finding the nearest neighbor of $\mathbf{p}_{\text{samp}}$ in the current tree, v_{near}, a new vertex, v_{new}, is initialized as follows. First, the line between v_{near} and $\mathbf{p}_{\text{samp}}$ is projected to the planar surface approximation at v_{near}. The prior pose of v_{new} is defined at a fixed distance $r_{\text{exp}}^{\text{RRT*}} > d_{\text{IN}}^{\text{nom}}$ from v_{near} on the projected line. Subsequently, the actual pose of v_{new} on the terrain is computed by f_{ta}. If v_{new} lies on traversable terrain, the set of *near* tree vertices, V_{near}, is found as the set of all vertices within a sphere of radius $r_{\text{near}}^{\text{RRT*}} > r_{\text{exp}}^{\text{RRT*}}$ around v_{new}.

Using the method described in Section 4.3, we attempt to compute a connection trajectory from each vertex in V_{near} to v_{new}. From all feasible trajectories, the one minimizing the total path cost at v_{new} is chosen for connecting the new vertex to the tree, while the other trajectories are discarded. Subsequently, the tree is rewired by generating trajectories from v_{new} to each vertex in V_{near}. If the path from the tree's start via v_{new} to any vertex $v_{\text{near},i} \in V_{\text{near}}$ has lower cost than the current path to $v_{\text{near},i}$, the parent of $v_{\text{near},i}$ is changed to v_{new} and the respective connection trajectory is added as an edge of the tree. Whenever the tree can be rewired at a vertex contained in the initial trajectory, an incremental improvement of the entire trajectory has been achieved.

4.5.3. Termination

The choice of a specific termination criterion for trajectory optimization reflects a trade-off between optimality of the result and computational complexity (runtime). Setting a fixed time limit may be necessary in applications with hard time constraints. However, it will render the quality of the optimization highly unstable, especially if the distance between start and goal varies considerably between different planner queries. A more sophisticated termination criterion should consider how densely the environment has been explored for potential trajectories, allowing to stop the optimization when the chances of further improvement are comparatively low. Following these considerations, we compute the *moving average* of the number of vertices within V_{near} in each iteration k,

$$N_k^{\text{avg}} = \alpha N_k + (1-\alpha) N_{k-1}^{\text{avg}},\ 0 < \alpha \ll 1\,, \tag{17}$$

and terminate the optimization if this value exceeds a selected threshold, $N_{\text{max}}^{\text{avg}}$. Hence, termination is triggered when the space around the initial trajectory is explored by the tree with a certain density, dynamically adapting the assigned computation time according to the trajectory length and the terrain complexity. Finally, the optimized trajectory from the start to the goal is obtained by backtracing, starting at the goal vertex. By construction, this trajectory is a *planner trajectory* belonging to $\mathfrak{T}_{\text{pl}}$, satisfying all the constraints as defined in Section 3.2.

4.6. Local Trajectory Optimization

A *planner trajectory*, $\mathcal{T} \in \mathfrak{T}_{\text{pl}}$, computed using RRTs and RRT*-based global optimization (Sections 4.4 and 4.5) represents a reasonably short, system-compliant connection between the start and the goal pose of the planner query. However, the trajectory may be suboptimal in terms of other criteria, such as steering action (path curvature) or terrain roughness along the path. In this section, we develop an algorithm for precise local optimization of such a trajectory with respect to a cost function, $f_{\text{cost}} : \mathcal{T} \mapsto c \in \mathbb{R}_{\geq 0}$. We assume f_{cost} to be defined only for *feasible* planner trajectories, that is, trajectories that fulfill the traversability and curvature constraints given in Eqs. (3)–(5). Trajectories not satisfying these constraints are defined to have infinite cost, because they are inherently unusable.

State-of-the-art trajectory optimization methods, such as potential fields (Khatib, 1986), elastic strips (Brock & Khatib, 2002), or CHOMP (Zucker et al., 2013), typically rest on tweaking trajectories based on the *gradient* of a cost function. However, in the presence of *hard constraints*, such as terrain contact, kinematic feasibility (maximum curvature), or traversability (obstacles), trajectory cost functions do not exhibit a well-defined gradient in general. This problem is often bypassed by reformulating hard constraints as cost factors and including the latter in the objective function. For example, the traversability constraint can be restated as a differentiable cost, depending on the distance to obstacles. However, we argue that this is an artifact which may distort the optimization: given a certain safety margin accounting for uncertainties, a trajectory close to an obstacle is actually not worse than one that keeps a greater distance. These issues are avoided by specialized trajectory optimization methods capable of incorporating nondifferentiable cost functions, such as sequential convex optimization (Schulman et al., 2014) or stochastic optimization (Kalakrishnan et al., 2011). However, another problem arises in conjunction with the strictly *local* nature of our terrain assessment approach. Existing trajectory optimization approaches require precise knowledge of the environment around the trajectory (e.g., an obstacle map) for computation of distance fields or efficient evaluation of many candidate trajectories. In our framework, quantities such as locations and shapes of obstacles are inherently unknown. Their computation would require expensive, dense application of the terrain assessment function f_{ta}, and inevitably introduce some kind of artificial discretization.

Following these considerations, we propose a trajectory optimization method that does not require gradient information and ensures that the boundaries defined by the hard constraints are respected at any time. Within these boundaries, the trajectory is tweaked according to soft criteria encoded in a cost function, without any influence of the hard constraints. Further, the method is consistent with the principles of the overall planning approach, in the sense that it does not require large-scale terrain reconstruction or obstacle mapping, but only local, on-demand evaluation of f_{ta} at discrete poses around the trajectory.

Without loss of generality, we define the cost of a feasible trajectory to be a function of path length, curvature, and terrain traversability, and present an example implementation of f_{cost}. We compute the cost at the level of individual trajectory segments of $\mathcal{T}$ and denote the cost of the segment between $\mathcal{N}_i$ and $\mathcal{N}_{i+1}$ by c_i:

$$f_{\text{cost}}(\mathcal{T}) := \sum_{i=1}^{N_{\text{n}}-1} c_i\,. \tag{18}$$

We define the cost of a trajectory segment as the weighted sum of path length, curvature, and inverse traversability, integrated along the path. For the sake of computational efficiency, we introduce a simplified calculation method, valid under the following assumptions:

1. The distance between nodes is always kept close to a selected nominal value, $d_{\text{IN}}^{\text{nom}}$ (within the interval $[d_{\text{IN}}^{\text{min}}, d_{\text{IN}}^{\text{max}}]$)
2. The length of the trajectory segment is close to the Euclidean distance between the nodes.
3. The traversability along the trajectory segment is close to the respective values at either node.
4. In terms of curvature, we are interested in pruning *peaks*.

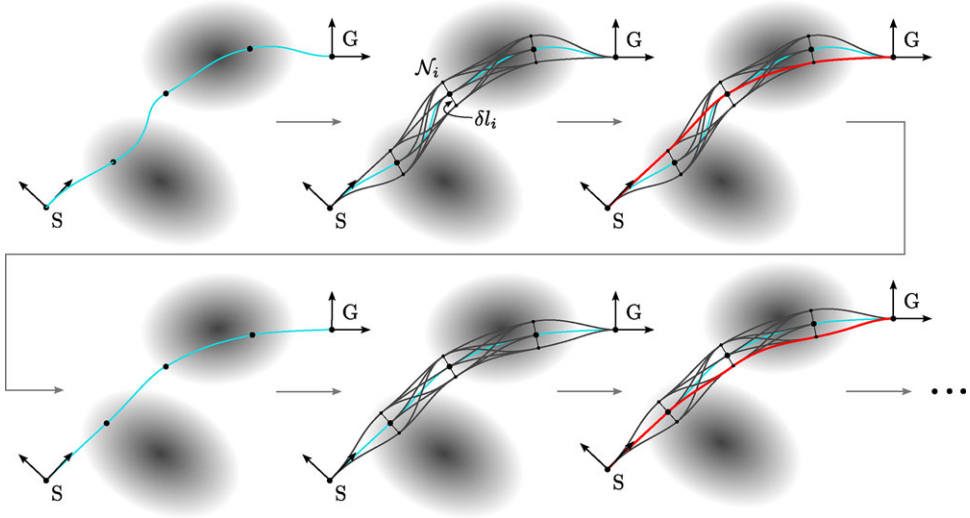

Figure 7. Schematic illustration of the local trajectory optimization method, showing two iterations of the algorithm.[2] The gray ellipses symbolize low traversability regions. In every iteration, the trajectory connecting start (S) and goal (G) is augmented to a *graph*, representing nearby alternative trajectories. This is achieved by adding two *subnodes* at each node $\mathcal{N}_i$ (at a small distance δl_i from the trajectory) and computing a total of nine connecting trajectory segments at each node. Subsequently, the graph is searched for the lowest cost trajectory (the bold, red curve), which represents the starting point for the next iteration. The example illustrates how the trajectory is iteratively moved out of low traversability areas and optimized for short length and low curvature.

We compute c_i as the weighted sum of the Euclidean distance between the two nodes, denoted by d_i^{i+1}, the maximum absolute curvature along the corresponding planar trajectory segment, denoted by $\kappa_{\max}(\mathbf{t}_i)$, and the inverse terrain traversability at $\mathcal{N}_{i+1}$, $1-\tau_{i+1}$, each normalized to a value in the interval $[0, 1]$:

$$c_i := \begin{cases} w_{\text{leng}} \dfrac{d_i^{i+1} - d_{\text{IN}}^{\min}}{d_{\text{IN}}^{\max} - d_{\text{IN}}^{\min}} + w_{\text{curv}} \dfrac{\kappa_{\max}(\mathbf{t}_i)}{\kappa_{\max}} + w_{\text{trav}}(1-\tau_{i+1}) & \text{if } \kappa_{\max}(\mathbf{t}_i) \le \kappa_{\max} \text{ and } \tau_{i+1} > 0 \\ \infty & \text{otherwise.} \end{cases} \tag{19}$$

By choosing a specific set of scalar weights (w_{leng}, w_{curv}, w_{trav}), the result of the optimization can be directed to any desired compromise between short, smooth, and low-risk trajectories.

The basic principle of the trajectory optimization algorithm is illustrated in Figure 7. Directed by the cost function, the trajectory is iteratively deformed in small steps, until a local minimum of f_{cost} is reached. In every iteration, the current trajectory is augmented with an additional pair of nodes at each node $\mathcal{N}_i$, placed at a small lateral offset, δl_i, on the left and on the right of the trajectory. Hence, each node is represented by a set of three *subnodes*. Computing trajectory segments (cubic curvature polynomials) connecting each subnode at $\mathcal{N}_i$ to each subnode at $\mathcal{N}_{i+1}$ along the trajectory (nine connections per node) results in a *graph* encoding 3^{N_n-2} slightly different trajectories from the start pose to the goal pose. Discarding subnodes and connections declared as unfeasible with regard to the traversability and curvature constraints, this graph is searched for the least cost trajectory according to Eq. (18) using standard graph search techniques. The resulting trajectory—by construction again a *planner trajectory* satisfying all predefined constraints—serves as starting point for the next iteration. The algorithm terminates once the trajectory cost cannot be reduced any more by any of the alternative trajectories in the respective graph, indicating that a local minimum of f_{cost} is reached. The four main processes constituting an iteration of the algorithm are detailed in the following.

[2]For the sake of readability, this figure shows the trajectories in a 2D environment. In reality, however, each trajectory segment is computed in a different planar patch, given by the 6D pose of the corresponding (sub)node, as defined in Section 4.1.

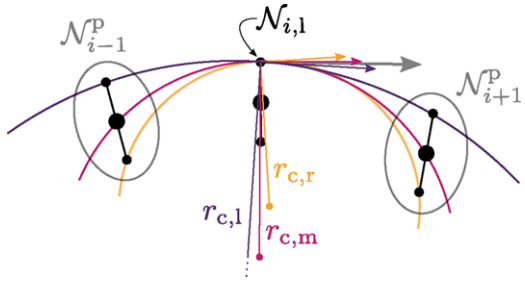

Figure 8. Computation of the heading angle and the curvature value of additional subnodes left and right of the trajectory, using the example of the left subnode at node i, $\mathcal{N}_{i,\mathrm{l}}$. Projecting all neighboring subnode positions to the plane approximating the terrain surface at $\mathcal{N}_{i,\mathrm{l}}$, three circles are computed, each passing through $\mathcal{N}_{i,\mathrm{l}}$ and the respective left, middle, or right subnode positions of the projected predecessor and successor nodes ($\mathcal{N}^{\mathrm{p}}_{i-1}$ and $\mathcal{N}^{\mathrm{p}}_{i+1}$). The heading of $\mathcal{N}_{i,\mathrm{l}}$ is defined such that the pose at the subnode is aligned with the average of the tangent vectors of these circles (the bold, gray arrow), and the curvature of $\mathcal{N}_{i,\mathrm{l}}$ is computed based on the circles' radii, $r_{\mathrm{c,l}}$, $r_{\mathrm{c,m}}$, and $r_{\mathrm{c,r}}$.

4.6.1. Computation of Additional Subnodes

Each node $\mathcal{N}_i$ along the trajectory is augmented with two subnodes, one at the left and one at the right of the trajectory. The lateral offset of these subnodes from the trajectory, δl_i, is dynamically adapted in the range between a node-specific upper limit, $\delta l_{\max,i}$, and a fixed lower bound, $\delta l_{\min}$, according to the curvature value of $\mathcal{N}_i$. At high curvature nodes, small lateral offsets are chosen, reducing the maximum curvature of the trajectory segments connecting the subnodes among each other. As defined in Eq. (8), a node (and thus each subnode) comprises a 6D pose, a curvature, and a traversability value. In the following, the computation of these values is described using the example of the left subnode at the node $\mathcal{N}_i$, denoted by $\mathcal{N}_{i,\mathrm{l}}$. The *pose* of the subnode is initialized on the y-axis of the pose $\mathbf{T}_{\mathrm{M}\tilde{\mathrm{R}}_i}$, at a small distance of δl_i from the origin. Subsequently, the terrain-compliant 3D position and the respective surface normal are computed with $f^{\mathrm{p}}_{\mathrm{ta}}$. These quantities define the subnode pose up to one rotational degree of freedom, namely, the heading angle (rotation around the surface normal). The latter can be freely chosen to make the new subnode conform best with the course of the trajectory. Because predecessors and successors of $\mathcal{N}_{i,\mathrm{l}}$ can be any subnodes of $\mathcal{N}_{i-1}$ and $\mathcal{N}_{i+1}$, respectively, the heading (as well as the curvature) of $\mathcal{N}_{i,\mathrm{l}}$ are chosen such that they represent a reasonable compromise for all potential trajectories passing through this subnode.

Leveraging the observation that a circular arc represents a smooth connection of any three points in the plane, this compromise is found as follows. As illustrated in Figure 8, the positions of all six subnodes of $\mathcal{N}_{i-1}$ and $\mathcal{N}_{i+1}$ are projected to the plane defined by the surface normal at $\mathcal{N}_{i,\mathrm{l}}$, followed by the computation of three circles passing through $\mathcal{N}_{i,\mathrm{l}}$ and the left, middle, and right subnodes of $\mathcal{N}_{i-1}$ and $\mathcal{N}_{i+1}$, respectively. The heading of the pose at $\mathcal{N}_{i,\mathrm{l}}$ is defined such that the x-axis of the pose is aligned with the *average* of the three tangent vectors of these circles at the point where they pass through $\mathcal{N}_{i,\mathrm{l}}$.

Similarly, the *curvature* of the subnode $\mathcal{N}_{i,\mathrm{l}}$ is defined as the average of the three circles' (signed) curvatures. The absolute curvature of each circle is given by the inverse of its radius,

$$|\kappa_{\mathrm{c},k}| = \frac{1}{r_{\mathrm{c},k}},\ k \in \{\mathrm{l}, \mathrm{m}, \mathrm{r}\}\,, \tag{20}$$

and the sign of $\kappa_{\mathrm{c},k}$ is determined based on the location of the circle's center relative to the trajectory (right: negative, left: positive). Finally, the *traversability* value of the subnode $\mathcal{N}_{i,\mathrm{l}}$ is computed by f^{τ}_{ta} based on the previously computed 6D pose.

4.6.2. Adding or Removing Nodes

At the beginning of the trajectory optimization, the distance between all pairs of subsequent nodes roughly amounts to the selected nominal value, $d^{\mathrm{nom}}_{\mathrm{IN}}$, by construction (cf. Sections 4.3 and 4.4). During the optimization, however, the internode distances may increase or decrease depending on the deformation of the trajectory. Both too short and too large distances between the nodes are undesirable: the former result in unnecessary computational effort, and the latter preclude correct enforcement of compliance with the vehicle constraints (cf. Section 4.1). Hence, we select a minimum and a maximum value for the internode distance,

$$\begin{aligned} d^{\mathrm{min}}_{\mathrm{IN}} &:= \frac{1}{2} d^{\mathrm{nom}}_{\mathrm{IN}} \\ d^{\mathrm{max}}_{\mathrm{IN}} &:= \frac{3}{2} d^{\mathrm{nom}}_{\mathrm{IN}}\,. \end{aligned} \tag{21}$$

In the beginning of each iteration of the algorithm (before adding the additional left and right subnodes), the distances between the nodes are examined. Where the above limits are undercut or exceeded, a node is added or removed, respectively, as illustrated in Figure 9. When inserting a node between $\mathcal{N}_i$ and $\mathcal{N}_{i+1}$, its 3D position and the corresponding surface normal are computed by projecting the pose of $\mathcal{N}_{i+1}$ to the x-y plane of $\mathcal{N}_i$'s pose, generating a planar connection trajectory (a cubic curvature polynomial) between the two poses, and applying $f^{\mathrm{p}}_{\mathrm{ta}}$ halfway on this trajectory. The remaining properties of the new pose, namely, heading, curvature, and traversability, are computed similarly as for the additional subnodes (cf. Figure 8), except that only one circle is computed. If the distance between $\mathcal{N}_i$ and $\mathcal{N}_{i+1}$ is shorter than $d^{\mathrm{min}}_{\mathrm{IN}}$, both $\mathcal{N}_{i+1}$ and $\mathcal{N}_{i+2}$ are deleted, and a new node is inserted between $\mathcal{N}_i$ and $\mathcal{N}_{i+3}$ according to the same procedure, yielding a locally equal distribution of nodes.

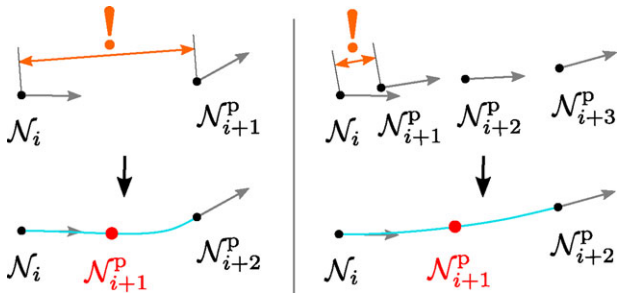

Figure 9. Insertion (left) and removal (right) of trajectory nodes are triggered by too large or too short distances between neighboring nodes. If the distance between $\mathcal{N}_i$ and $\mathcal{N}_{i+1}$ is too large, a new node is inserted halfway between the two nodes, on a trajectory segment generated to connect the pose of $\mathcal{N}_i$ and the pose of $\mathcal{N}_{i+1}$ projected to the x-y plane of $\mathcal{N}_i$, denoted by $\mathcal{N}^{\mathrm{p}}_{i+1}$. If the distance between $\mathcal{N}_i$ and $\mathcal{N}_{i+1}$ is too short, both $\mathcal{N}_{i+1}$ and its successor are deleted, and a new node is inserted between $\mathcal{N}_i$ and $\mathcal{N}_{i+3}$.

4.6.3. Computation of Connecting Trajectory Segments

Connecting all subnodes along the trajectory to all of their potential successor subnodes requires nine trajectory segments between each node $\mathcal{N}_i$ and its respective successor, $\mathcal{N}_{i+1}$. These connecting trajectory segments are represented as cubic curvature polynomials in the x-y plane of the subnode at their origin. The boundary conditions at the start and the end of the segments are set as defined in Eqs. (12) and (13), such that the continuity constraint is satisfied along any path in the resulting graph of trajectories.

The trajectory segments have to be computed using iterative, numerical methods, since no closed-form solutions exist. Given the large number of trajectories required in each iteration of the algorithm, this results in unacceptably long runtimes due to the high computational effort. Hence, we precompute a dense lookup table (LUT) mapping discrete values of the boundary conditions (initial and terminal position, heading and curvature) to the parameters defining corresponding curvature polynomials. The LUT is computed using the approach of Nagy & Kelly (2001) and designed such that it covers the entire range of relevant trajectory segments (depending on the specific application) in sufficient resolution. During trajectory optimization, the only relevant quantity is the maximum curvature of the segments, which is required for computing the trajectory cost. The actual trajectory parameters are only required for the final result. Consequently, only a reduced LUT is loaded at runtime, containing merely the maximum curvature value for each trajectory segment. This drastically decreases the memory consumption and allows to use a truly high resolution LUT. Hence, expensive numerical trajectory generation can be replaced by simple and efficient table lookups, while sacrificing very little accuracy.

Finally, a *cost value* is computed for each trajectory segment, as a preparation for finding the least cost trajectory using graph search. The cost of a trajectory segment between two subnodes, $\mathcal{N}_{i,j}$ and $\mathcal{N}_{i+1,k}$, is computed based on the Euclidean distance between $\mathcal{N}_{i,j}$ and $\mathcal{N}_{i+1,k}$, the maximum curvature of the trajectory segment, and the traversability value of $\mathcal{N}_{i+1,k}$, according to Eq. (19). If the maximum curvature of the trajectory segment exceeds the curvature limit, $\kappa_{\max}$, or if the subnode $\mathcal{N}_{i+1,k}$ is untraversable, the cost of the trajectory segment is set to infinity.

4.6.4. Graph Search and Convergence

The subnodes of the augmented trajectory, along with the (implicit) connections between all subnodes of each pair of successive nodes, form a *graph*, where the cost of edges is defined as the cost value of the respective trajectory segment, as introduced above. The graph encodes a certain number of slightly different trajectories between the start and the goal pose. Among these alternatives, the least cost trajectory, according to Eq. (18) is found by applying Dijkstra's shortest path algorithm (Dijkstra, 1959). It is guaranteed to satisfy all hard constraints, since unfeasible trajectory segments have infinite cost (cf. Eq. (19)).

In the beginning of the optimization, the maximum lateral offset of the subnodes, $\delta l_{\max,i}$, is set to the same value at all nodes ($\delta l_{\max,\mathrm{init}}$). Every time the trajectory "locally converges" at a node $\mathcal{N}_i$—indicated by the least cost trajectory passing through the *middle* subnode of $\mathcal{N}_i$—the maximum lateral offset at this node is slightly reduced, until the selected absolute minimum, $\delta l_{\min}$, is reached. The optimization terminates once the maximum lateral offset values of all nodes have reached the minimum value, that is, $\delta l_{\max,i} = \delta l_{\min}\ \forall i$, and the least cost trajectory passes through the middle subnode at all nodes. This strategy allows to combine quick convergence to the vicinity of a local minimum (using comparatively large steps) with high accuracy in the final approximation of this minimum.

4.7. RESULTS

In this section, we apply the proposed motion planning framework—embedding the terrain assessment method presented in Section 5—to compute trajectories connecting given pairs of start/goal poses, on the basis of point cloud maps (*planning maps*). The latter are built from sensor data recorded with our robot ARTOR[3], using the ICP localization and mapping system presented in Section 6.1. The parameter values listed in Table I reflect the constraints and capabilities of ARTOR[4]. For the sake of comparabil-

[3]Cf. Section 7 for a description of the robot.

[4]Notice that the value of $\kappa_{\max} = 2\ \mathrm{m}^{-1}$ is not directly linked to the physical properties of our robot, which theoretically could even turn on the spot. Hence, we could choose much higher values for the maximum path curvature. However, we believe that smooth trajectories are generally preferable, as they can be followed faster and more precisely than trajectories including sharp turns.

Table I. Parameter values for motion planning.

Category	Description	Symbol	Value
Traversability constraints	robot dimensions (length, width, height)	$\mathbf{d}_{\text{rob}}$	$[1.3, 0.7, 1.2]\,m$
	maximum absolute roll angle	$\psi_{\max}$	0.18 rad
	pitch angle limits (ascending, descending)	$\theta_{\min}, \theta_{\max}$	−0.3 rad, 0.25 rad
	maximum terrain roughness (step height)	$\rho_{\max}$	0.08 m
Curvature constraint	maximum absolute curvature of drivable trajectories	$\kappa_{\max}$	2 m^{-1}
Trajectory representation	nominal distance between two trajectory nodes	$d_{\text{IN}}^{\text{nom}}$	0.6 m
RRT optimization*	distance from parent when adding a new vertex	$r_{\text{exp}}^{\text{RRT}^*}$	1.8 m
	maximum radius for connecting and rewiring	$r_{\text{near}}^{\text{RRT}^*}$	3.6 m
	maximum average number of vertices within $r_{\text{near}}^{\text{RRT}^*}$ (termination)	$N_{\max}^{\text{avg}}$	30
Local traj. optimization	weighting factor of trajectory length	w_{leng}	0.25
	weighting factor of trajectory curvature	w_{curv}	0.25
	weighting factor of terrain traversability	w_{trav}	0.5
	maximum lateral offset of subnodes (initial value)	$\delta l_{\max,\text{init}}$	0.08 m
	minimum lateral offset of subnodes	$\delta l_{\min}$	0.04 m

ity, all results presented in the remainder of the article are based on this same set of parameters, unless otherwise stated.

We first present and discuss the results of a series of example planner queries in different environments. In all of the figures below, the bold, green line represents the final planner result, namely, a trajectory initialized with the RRT planner (Section 4.4), refined with the RRT*-based global optimization method (Section 4.5) and locally optimized with the approach proposed in Section 4.6. The blue lines perpendicular to the trajectory and the terrain represent the z-axis of the poses of each trajectory node. Start and goal poses of the planner queries are marked with S and G, respectively, combined with an arrow indicating the heading direction. The trajectories are plotted in the respective point cloud maps of the environment. The entire maps are colored according to the per-point terrain roughness computed as an intermediate step by f_{ta}^{τ} (cf. Section 5.2; light blue: low roughness, gray: high roughness, dark red: untraversable). Notice that this is for illustration purposes only: in reality, the terrain roughness is computed only locally in places explored by the planner, but never for the entire map, avoiding unnecessary computational overhead.

Figure 10 shows an example planning result in unstructured, nonplanar terrain. The start and goal poses are approximately 40m apart from each other, with the goal located on top of a small hill. The figure shows three different trajectories, representing the results of the three stages of the planner: RRT, RRT*, and local optimization. In the top row, the left image represents a bird's-eye view on the scene, whereas the middle image shows the trajectories from the same perspective as the photograph of the terrain on the right. All of the three trajectories fulfill the defined requirements: they connect the start pose and the goal pose with a piecewise continuous function in the 6D space of robot poses that follows the terrain surface, not exceeding the thresholds in curvature, inclination angles, and terrain roughness. However, the trajectories differ considerably in terms of length, smoothness, and roughness of the traversed terrain. Starting from the initial RRT trajectory, the RRT*-based global optimization results in a significantly shorter trajectory (47 m instead of 53 m), largely eliminating unnecessary detours. The local optimization scheme further improves the trajectory with regard to length, curvature, and traversability, which can be observed in the enlarged views on different trajectory segments in the bottom row of Figure 10. Compared to the result of the RRT* optimization, the final trajectory is shorter (left), smoother (middle), and tweaked to avoid areas of high terrain roughness (right). The local optimization required 32 iterations to converge and reduced the trajectory cost by 9.6%.

Figure 11 shows two more example planner results, one in a highly nonplanar off-road terrain (top), and one in an urban environment (bottom). The example in rough terrain illustrates the planner's awareness of the robot's constraints in terms of maximum inclination. Looking at the top view on the left, it is obvious that avoiding the trees in the middle of the scene on the opposite side would yield a considerably shorter path. However, the slope behind the goal pose (seen in the right part of the photograph) is too steep to be safely traversed by the robot. Forced by the maximum pitch and roll angle thresholds, the planner computes a longer, yet safer trajectory, reaching the goal pose over less steep terrain. Further, the two relatively wide turns at the start and at the goal demonstrate the consideration of the maximum curvature constraint.

The example in the bottom row of Figure 11 shows that the proposed planning framework is able to cope with cluttered environments and find trajectories through

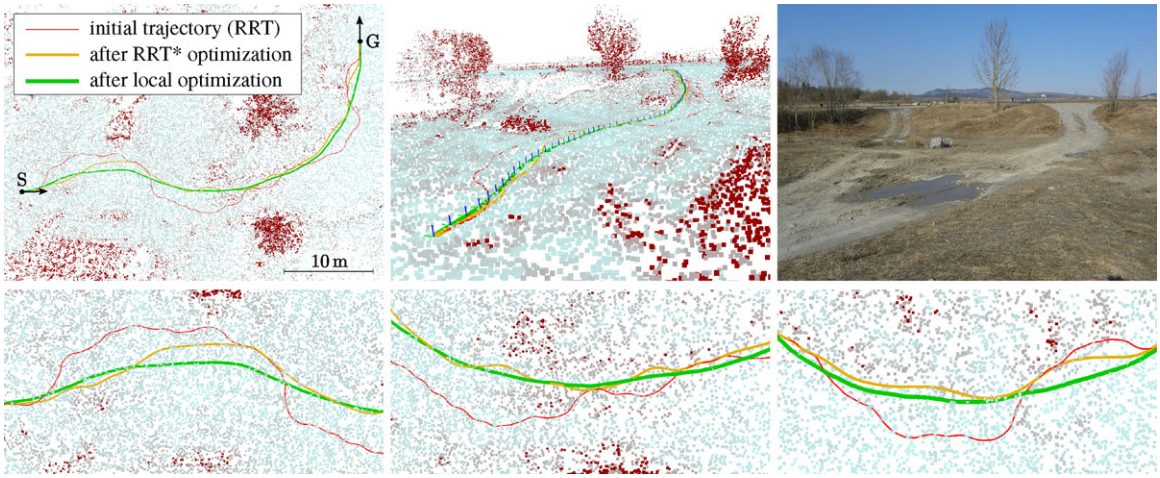

Figure 10. Example planner result in rough, nonplanar terrain, illustrating all three stages of the algorithm (RRT, RRT*, local trajectory optimization) with the respective resulting trajectories. The top row shows the start (S) and the goal (G) of the planner query and the three trajectories from two different perspectives, together with a photograph of the environment. The enlarged views on the trajectories shown in the bottom row highlight the effectiveness of the local optimization algorithm in terms of tweaking the trajectory for short length (left), low curvature (middle), and avoidance of high roughness areas (right).

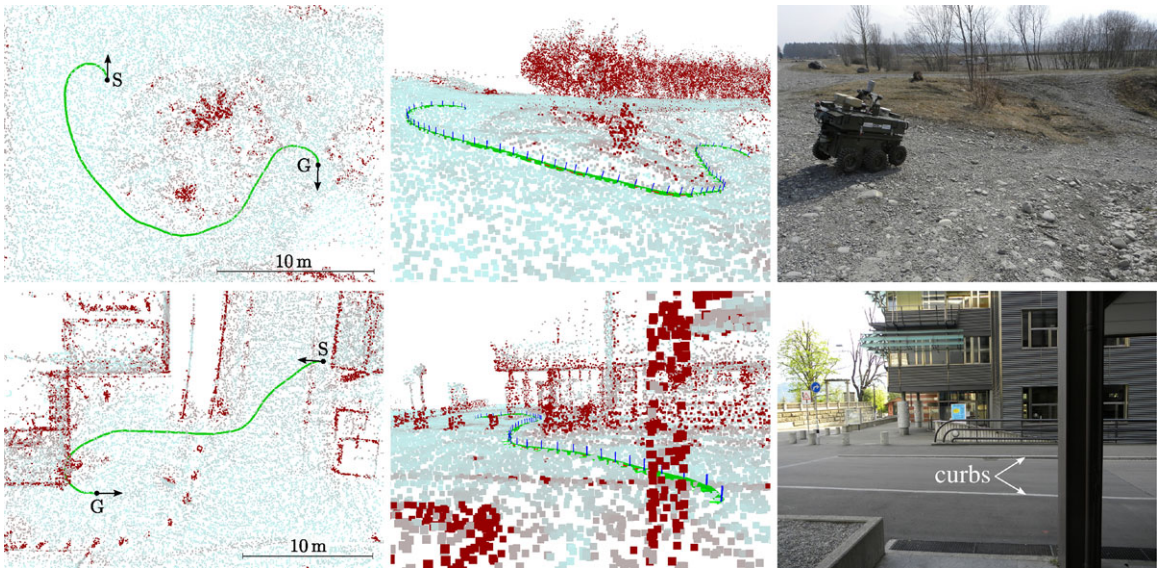

Figure 11. Example planner results in a rough, 3D terrain (top) and in an urban environment (bottom), with bird's-eye view on the result (left), a photograph of the environments (right), and the scene shown from the same perspective as the photograph (middle). The example in the top row demonstrates the compliance of the planner with the maximum inclination constraints: the visibly shortest path via the (too) steep slope behind the goal pose is discarded in favor of a longer, yet safe trajectory. The example in the bottom row shows the planner's ability to find suitable trajectories in cluttered environments and to deal with multiple overlying layers in the map, such as the ground and the ceiling behind the goal pose.

narrow passages. Moreover, it demonstrates an important advantage of our approach, assessing the terrain geometry based only on *local* point cloud data during motion planning: the trajectory can pass through areas with multiple overlying levels in the map (such as the ground and the ceiling behind the goal pose) without any special handling.

The results shown in Figure 12 demonstrate that our motion planning approach inherently enables planning on *generic* 2D manifolds embedded in the 3D space,

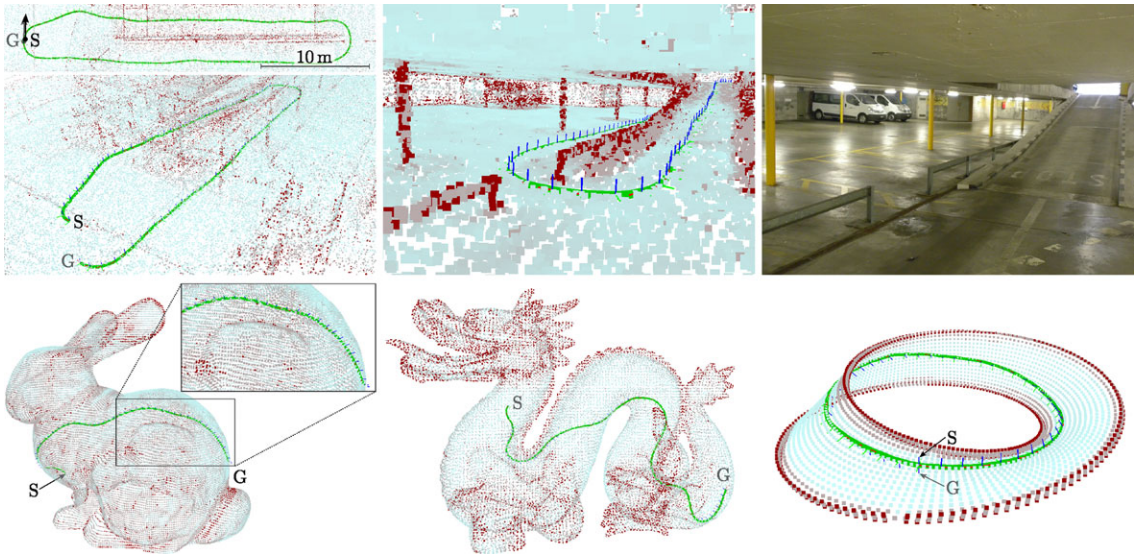

Figure 12. Example results demonstrating the suitability of our planning approach for computing trajectories in generic 2D manifolds in the 3D space, regardless of the complexity and the topology of the environment. All of the environments in this figure cannot be represented in 2.5D maps. Top: planning a trajectory from the upper to the lower floor of a two-story parking garage. Bottom: trajectories computed on artificial point cloud maps (discarding inclination constraints), from the chest to the back on the Stanford Bunny (left), from the neck to the tail on the inside of the Stanford Dragon (middle), and from one to the other side of the surface on a Möbius strip (right).

particularly in environments that cannot be represented in 2.5D maps (such as multilevel buildings or pipe systems), and even on nonorientable surfaces. This functionality results from the strictly *local* nature of all processes involved in planning (terrain assessment, tree expansion, trajectory optimization), together with the ability of 3D point cloud maps to represent arbitrary geometries.

The top row of Figure 12 shows a planning result in a two-level parking garage, where the goal pose is located exactly below the start pose, on the lower level. The planner successfully finds the ramp connecting the two floors and computes a feasible trajectory leading from the upper to the lower level. Notice that the entire planning framework is unaware of the topology of the environment. As in all other examples, the trajectory is computed based on a simple point cloud map, gradually expanding the RRT until a connecting path is found, followed by the trajectory optimization steps. The bottom row of the figure shows planning results in three artificial point cloud maps, demonstrating the planner's ability to compute trajectories on generic 3D surfaces. In these examples, we discard maximum inclination constraints, assuming that the robot is able to hold on to the surface irrespective of gravity (e.g., using magnetic wheels on a metallic structure). In the left image, a trajectory is planned on the Stanford Bunny (Levoy, Gerth, Curless, & Pull, 2015), starting at the chest and ending at the back. The trajectory in the middle image is computed *on the inside* of the Stanford Dragon (Levoy et al., 2015), leading from the neck to the tail. Finally, the rightmost example shows a trajectory computed on a Möbius strip, representing a nonorientable surface. Start and goal poses are located at the same position, with their z-axes pointing in opposite directions. Hence, they lie on opposite sides of the surface. The planner computes a connecting trajectory leading once around the strip. Unlike the theoretical nature of these examples may suggest, the ability to plan trajectories on arbitrarily complex 3D surfaces and topologies is of high practical relevance, for example, in robotic inspection of ship hulls or pipe systems in power plants.

Figure 13 analyzes the local trajectory optimization method (cf. Section 4.6). In particular, the figure shows how the weighting factors of the different components of the trajectory cost (Eq. 19) influence the optimization result. The same initial trajectory is fed to the local optimization algorithm with four different sets of cost weights. From left to right, the trajectory is optimized for short length only, for low curvature only, for high traversability only, and finally using the default set of weights, representing a compromise between the three cost factors. Optimizing the length yields the shortest possible trajectory, yet featuring high curvature turns and keeping only minimal distance to obstacles. The curvature-optimized trajectory exhibits smoother turns, yet passes through high roughness areas. Optimizing for high traversability results in a longer trajectory leading through low terrain roughness areas only, however, featuring many high curvature turns. Simultaneously including all three

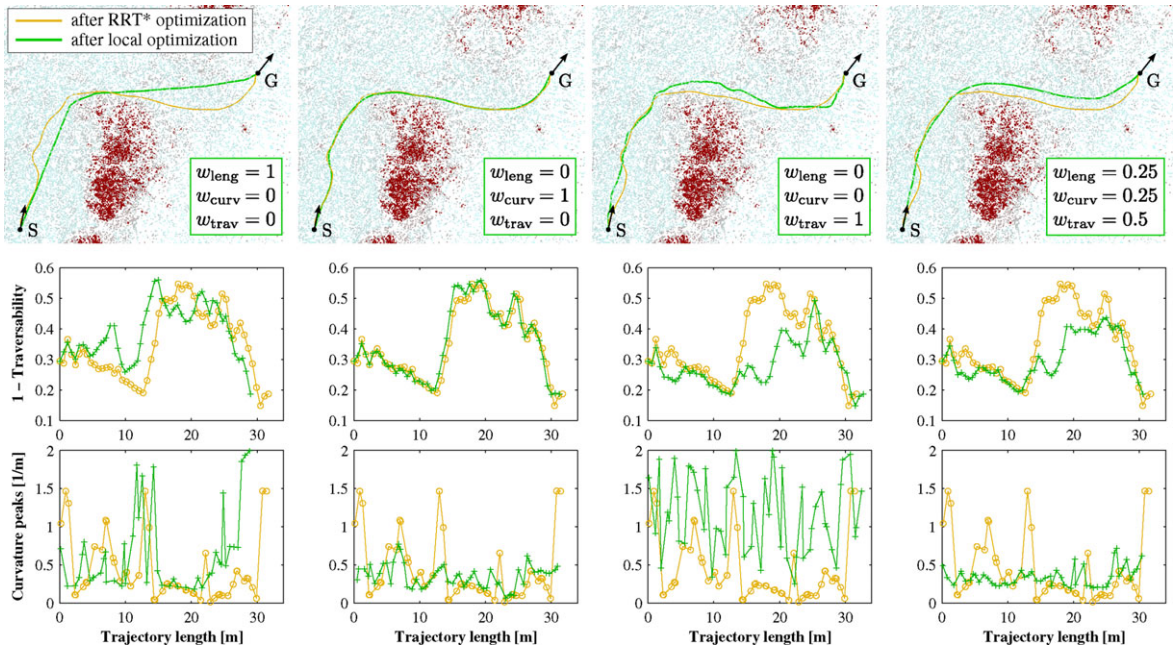

Figure 13. Analysis of the local trajectory optimization. The same initial trajectory is optimized four times with different weighting factors of the path cost components (length, curvature, traversability). From left to right: minimizing trajectory length, minimizing curvature, maximizing traversability of the traversed terrain, and optimizing the trajectory using the nominal weighting factors, taking into account all three factors simultaneously. The latter represents a good compromise between length, smoothness, and high traversability. The plots in the lower two rows show the inverse traversability values, $1 - \tau_i$, of each trajectory node (as defined in Equation 31) and the maximum absolute curvature of each trajectory segment, $\kappa_{max}(\mathbf{t}_i)$, before and after optimization. Notice that—regardless of the chosen weights—the optimization never compromises the trajectories' compliance with the hard constraints ($\tau > 0$ and $|\kappa| \leq 2\,\mathrm{m}^{-1}$).

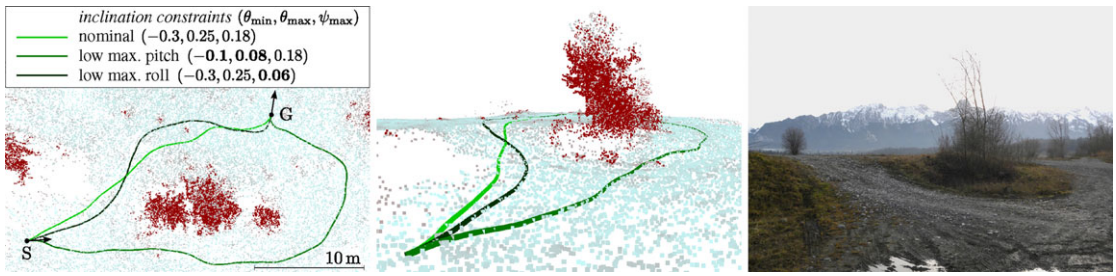

Figure 14. Incorporation of different inclination constraints. The top view on the left and the perspective view in the middle show the results of three identical planner queries, computed once with the nominal constraints, once with low pitch angle limits, and once with a low roll angle limit. The relatively steep slope shown in the left part of the photograph lies on the direct path between start and goal. With the nominal constraints, it can be traversed without difficulty. The low pitch angle limits render this place untraversable and lead to a longer trajectory over less-inclined terrain. In the third case, the planner produces a trajectory aligned with the gradient of the terrain at the steepest part, correctly accounting for the low limit on the maximum absolute roll angle.

cost factors yields a smooth and reasonably short trajectory, avoiding most high roughness patches.

Figure 14 demonstrates the planner's ability to incorporate the robot's inclination constraints (roll and pitch angle limits). The figure shows the results of three identical planner queries (same start and goal) with different inclination constraints. The direct path from the start to the goal includes a relatively steep slope, shown in the photograph on the right. With the nominal constraints, listed in Table I and reflecting the capabilities of our robot ARTOR, this slope can be safely traversed at any angle. Hence, the planner's result is close to a short, straight-line connection of the start and the goal. However, reducing the pitch angle limits to one third of the nominal values renders the slope

impassable. Consequently, the planner computes a much longer alternative trajectory, leading through terrain of low inclination. On the other hand, if the pitch angle limits are kept at their nominal values, but the roll angle limit is lowered to one third of the default, the steep slope remains traversable if negotiated at the right angle. The resulting trajectory in this case is slightly longer compared to the nominal case, yet well aligned with the gradient of the terrain at the steep part, ensuring low absolute roll angle values.

5. TERRAIN ASSESSMENT

Terrain assessment is a fundamental prerequisite for motion planning. Reliable computation of safe and optimal trajectories requires precise knowledge of the properties and condition of the surroundings, especially in challenging outdoor environments. In the following, we develop a terrain assessment method based on point cloud maps, solving the problem defined in Section 3.3. The approach rests on computation of statistics on the geometric distribution of map points in a local neighborhood. Avoiding any additional mapping processes, the terrain is implicitly reconstructed by computing terrain-induced vehicle poses—along with traversability estimates—at specific query poses. Hence, the method is free of any artificial discretization, such as division of the space into grid cells or mesh triangles.

5.1. Pose Computation (f_{ta}^{p})

The closest terrain-compliant robot pose for a given query pose, $\mathbf{T}_{\text{MR}}$, is computed based on the map points in a local neighborhood of its origin, $\mathbf{t}_{\text{MR}}$. Rather than considering all points within a certain distance, we use the K nearest-neighbors of the query point $\mathbf{t}_{\text{MR}}$. This approach is tolerant to comparatively high offsets of the query pose from the actual terrain surface. The number K is tuned according to the density of the maps, and the size of the specific robot, such that the area covered by K points is similar to the size of the robot's footprint. Hence, the orientation of the robot on the terrain can be inferred from the surface normal estimated based on the K nearest-neighbor points. We compute this normal using principal component analysis, as shown by Hoppe et al. (1992). With $\mathcal{N}_K(\mathbf{t}_{\text{MR}})$, the set of indices of the K nearest points to the query point in the map $\mathcal{M} = ((\mathbf{p}, \mathbf{d}_{\text{obs}})_i)_{i=1}^{N_{\text{P}}}$, we compute their center of gravity and the associated covariance matrix:

$$\overline{\mathbf{p}} = \frac{1}{K} \sum_{k \in \mathcal{N}_K(\mathbf{t}_{\text{MR}})} \mathbf{p}_k$$

$$\text{Cov}(\mathbf{t}_{\text{MR}}) = \sum_{k \in \mathcal{N}_K(\mathbf{t}_{\text{MR}})} (\mathbf{p}_k - \overline{\mathbf{p}})(\mathbf{p}_k - \overline{\mathbf{p}})^{\text{T}} \in \mathbb{R}^{3\times 3}. \quad (22)$$

Without regard to sign, the surface normal is given by the eigenvector of the covariance matrix corresponding to the smallest eigenvalue, denoted by $\mathbf{v}_0$. We regard the orientation of the query pose as a prior for the sign of the surface normal, $\mathbf{n}$:

$$\mathbf{n} = \text{sign}\,(\hat{\mathbf{x}}_{\text{MR}} \cdot \mathbf{v}_0)\,\mathbf{v}_0\,. \quad (23)$$

The condition of zero yaw between the query pose (frame R) and the resulting pose on the terrain (frame $\tilde{\text{R}}$) implies that the x-axis of $\tilde{\text{R}}$ is perpendicular to the y-axis of R. Since the z-axis of $\tilde{\text{R}}$ is given by the surface normal, we obtain the rotation of $\tilde{\text{R}}$ as follows:

$$\hat{\mathbf{x}}_{\text{M}\tilde{\text{R}}} = \frac{\hat{\mathbf{y}}_{\text{MR}} \times \mathbf{n}}{\|\hat{\mathbf{y}}_{\text{MR}} \times \mathbf{n}\|}$$

$$\mathbf{R}_{\text{M}\tilde{\text{R}}} = \left[\, \hat{\mathbf{x}}_{\text{M}\tilde{\text{R}}} \;\; \mathbf{n} \times \hat{\mathbf{x}}_{\text{M}\tilde{\text{R}}} \;\; \mathbf{n} \,\right]. \quad (24)$$

Locally approximating the terrain surface by the plane perpendicular to $\mathbf{n}$ containing $\overline{\mathbf{p}}$, the terrain contact point is given by the intersection of this plane with the z-axis of the frame attached to the query pose:

$$\mathbf{t}_{\text{M}\tilde{\text{R}}} = \mathbf{t}_{\text{MR}} + \frac{(\overline{\mathbf{p}} - \mathbf{t}_{\text{MR}}) \cdot \mathbf{n}}{\hat{\mathbf{z}}_{\text{MR}} \cdot \mathbf{n}}\, \hat{\mathbf{z}}_{\text{MR}}\,. \quad (25)$$

The planar surface approximation is only valid in the vicinity of $\overline{\mathbf{p}}$. Hence, we set an upper bound on the distance between $\overline{\mathbf{p}}$ and the terrain contact point and declare the pose as untraversable, if this is exceeded. Otherwise, the sought terrain-compliant 6D robot pose is given by

$$\mathbf{T}_{\text{M}\tilde{\text{R}}} = \begin{bmatrix} \mathbf{R}_{\text{M}\tilde{\text{R}}} & \mathbf{t}_{\text{M}\tilde{\text{R}}} \\ \mathbf{0}^{\text{T}} & 1 \end{bmatrix}. \quad (26)$$

5.2. Traversability Computation (f_{ta}^{τ})

The traversability value of a given terrain-compliant robot pose, $\mathbf{T}_{\text{M}\tilde{\text{R}}}$, is computed based on two factors: the orientation of the vehicle with respect to gravity (roll and pitch), and the roughness of the terrain (obstacles, in a broader sense). The former is given directly by $\mathbf{T}_{\text{M}\tilde{\text{R}}}$, since we use maps expressed in a gravity-aligned coordinate frame. To compute the terrain roughness based on point cloud maps, Hamner et al. (2008) propose to consider the residuals of planes fitted to vehicle-sized patches of the point cloud. Further, these patches are subdivided into smaller cells that are separately analyzed in terms of residuals to the plane fitted to the remaining points in the robot-sized patch, in order to avoid missing small obstacles. We argue that any discretization of the map into cells is problematic, since the composition of points within the cells—and thus the estimated traversability of entire terrain patches—changes with the *positioning* of the grid in the map. However, roughness should be a feature of the terrain with a well-defined, unambiguous value at any point on the surface, independent of specific robot poses or discretization parameters.

Following these considerations, we propose to compute a *roughness* value ρ_i for each map point $\mathbf{p}_i$, essentially using the discretization inherent to the map representation instead of introducing another, artificial decomposition. The

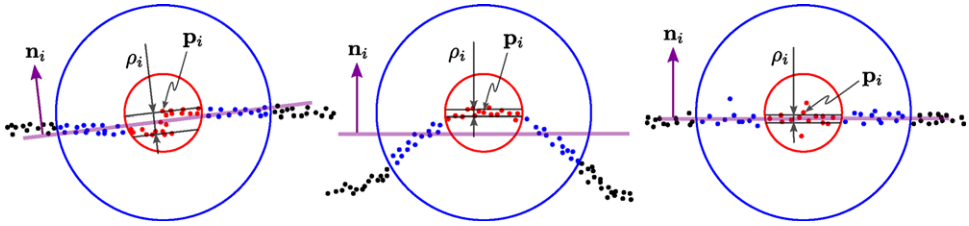

Figure 15. Computation of the per-point terrain roughness value for a map point $\mathbf{p}_i$, illustrated on the basis of cross sections of different terrain maps. The surface is locally approximated by a plane with normal vector $\mathbf{n}_i$, fitted to all points within a sphere of radius r_{plane} around $\mathbf{p}_i$ (outer, blue circle). Only considering the points within a smaller sphere (inner, red circle), the roughness value, ρ_i, is computed based on the maximum step height along the direction of $\mathbf{n}_i$, while discarding points classified as outliers. The method correctly assigns large roughness values to discontinuities in the terrain (left) and low values to smooth surface variations (middle). Further, it is robust to noise in the point clouds thanks to a built-in outlier detection (right).

point roughness values are independent of robot-specific parameters. They are computed on demand only at the required locations, based on the points in a local neighborhood. The terrain roughness of an actual robot pose is subsequently computed based on statistics on the roughness values of all points within a robot-sized volume at this pose.

The computation of the per-point terrain roughness is illustrated in Figure 15. For a given map point, $\mathbf{p}_i$, we first fit a plane to all points within a sphere of radius r_{plane} around $\mathbf{p}_i$. Using the approach introduced in Section 5.1, we obtain a local planar approximation of the surface defined by the center of gravity, $\overline{\mathbf{p}}_i$, and the plane normal, $\mathbf{n}_i$. Roughness can be defined as the presence of discontinuities (steps) in the terrain, which we quantify based on the distribution of the residuals of the plane fit (the distances of the map points to the plane). The most naive approach, namely, computing the mean or the variance of all residuals within the sphere of radius r_{plane}, yields an irresolvable trade-off for the value of r_{plane}. If it is chosen too small, the fitted planes "smooth out" discontinuities: obstacle points do not stand out, as they themselves have a major influence on the plane fit. On the other hand, if r_{plane} is chosen too large, uneven yet smooth terrain (cf. Figure 15, middle) may be classified as untraversable. We address this problem by defining a smaller radius $r_{\text{res}} < r_{\text{plane}}$ and restricting the analysis of the plane fit residuals to points within a distance of r_{res} from $\mathbf{p}_i$. With $\mathcal{B}_{\text{res}}(\mathbf{p}_i)$ the set of indices of all map points within a sphere of radius r_{res} around $\mathbf{p}_i$, their signed distances to the plane are computed as

$$d_{ik} = (\overline{\mathbf{p}}_i - \mathbf{p}_k) \cdot \mathbf{n}_i, \; k \in \mathcal{B}_{\text{res}}(\mathbf{p}_i)\,. \tag{27}$$

The terrain roughness in terms of discontinuities is characterized by the *variance* in $(d_{ik})_{k \in \mathcal{B}_{\text{res}}(\mathbf{p}_i)}$, rather than by the actual distances to the plane. However, point cloud maps may be polluted by noise, such as spurious points above or below the surface (Figure 15, right). We attenuate the influence of noise by considering a fixed fraction $f_\eta < 1$ of all map points as outliers. With $|\mathcal{B}_{\text{res}}(\mathbf{p}_i)|$ the number of points within the sphere of radius r_{res} around $\mathbf{p}_i$, we define

$$N_i := \text{ceil}\left(\frac{f_\eta |\mathcal{B}_{\text{res}}(\mathbf{p}_i)|}{2}\right), \tag{28}$$

and declare the map points associated with the N_i highest and the N_i lowest values in $(d_{ik})_{k \in \mathcal{B}_{\text{res}}(\mathbf{p}_i)}$ as outliers ($N_i = 2$ in the examples of Figure 15). Further defining $\mathcal{O}_{\text{res}}(\mathbf{p}_i)$ as the set of map indices of these outlier points, we finally compute the terrain roughness associated with $\mathbf{p}_i$ as the maximum step height in the direction of the normal of the fitted plane, given by the maximum distance between any two inlier points:

$$\rho_i = |\max_k(d_{ik}) - \min_k(d_{ik})|, \; k \in \mathcal{B}_{\text{res}}(\mathbf{p}_i) \setminus \mathcal{O}_{\text{res}}(\mathbf{p}_i)\,. \tag{29}$$

The overall approach is robust to noisy point clouds and enables to distinguish steps and discontinuities from terrain variation, as illustrated in Figure 15.

The terrain roughness associated with a certain robot pose, $\mathbf{T}_{\text{M}\tilde{\text{R}}}$, is computed based on the point roughness values of all points within a corresponding cuboid of the robot's size, $\mathbf{d}_{\text{rob}} \in \mathbb{R}^3$, enlarged by a certain safety margin. The set of indices of these points is denoted by $\mathcal{C}_{\text{rob}}(\mathbf{T}_{\text{M}\tilde{\text{R}}})$. Points featuring a roughness value higher than the robot-specific maximum roughness threshold, $\rho_{\max}$, are considered as belonging to an untraversable obstacle. However, in order to avoid inflation of obstacles, we only consider a point with $\rho_i > \rho_{\max}$ as an obstacle if its residual to the fitted plane, $d_{i,i}$, actively contributes to the computation of ρ_i, that is, if $d_{i,i} \geq \max_k(d_{ik})$ or $d_{i,i} \leq \min_k(d_{ik})$ for $k \in \mathcal{B}_{\text{res}}(\mathbf{p}_i) \setminus \mathcal{O}_{\text{res}}(\mathbf{p}_i)$. Otherwise, we set $\rho_m = \rho_{\max}$, marking the point as traversable. Finally, the robot pose is classified as untraversable ($\tau = 0$) if any point within the cuboid has a roughness greater than $\rho_{\max}$. Otherwise, the terrain roughness of the pose is computed as the mean point roughness value:

$$\rho_{\text{cub}} = \frac{1}{|\mathcal{C}_{\text{rob}}(\mathbf{T}_{\text{M}\tilde{\text{R}}})|} \sum_{m \in \mathcal{C}_{\text{rob}}(\mathbf{T}_{\text{M}\tilde{\text{R}}})} \rho_m \; \in [0, \rho_{\max}]\,. \tag{30}$$

Further, the roll and pitch angles of the pose with respect to a gravity-aligned coordinate frame, denoted by ψ and θ, respectively, are computed from $\mathbf{T}_{\mathrm{M}\tilde{\mathrm{R}}}$. The robot-specific thresholds for these angles are selected in order to guarantee safety (avoid tipover and slipping hazards) and feasibility (avoid overcharging the robot's propulsion system). The range of admissible roll angles is chosen symmetric around zero: $|\psi| \leq \psi_{\max}$. Since the robot's capabilities of negotiating slopes may be different for ascending and descending, we use independent lower and upper bounds for the pitch angle: $\theta_{\min} \leq \theta \leq \theta_{\max}$. The pose is classified as untraversable ($\tau = 0$) if one of these limits is violated. Otherwise, the final traversability value is computed using a weighted sum of the terrain roughness and the roll and pitch angles, normalized by their respective maximum values:

$$\tau = 1 - w_{\text{rough}} \frac{\rho_{\text{cub}}}{\rho_{\max}} + w_{\text{roll}} \frac{|\psi|}{\psi_{\max}} + w_{\text{pitch}} \max\left(\frac{\theta}{\theta_{\min}}, \frac{\theta}{\theta_{\max}}\right) \in [0, 1]\,. \tag{31}$$

5.3. Multilevel Environments

In multilevel environments, using nearest-neighbor search for extracting the locally relevant points for terrain assessment can yield incorrect results, since points belonging to the *ground* may be close to points associated to the *ceiling* of the floor below. In the point cloud maps, this typically becomes manifest in two planar layers of points, separated by a distance equal to the floor thickness. For correct pose and traversability computation, ground floor points need to be separated from ceiling points, such that the latter can be discarded for plane fitting and roughness estimation. Hence, we slightly modify our terrain assessment approach for application in multilevel environments.[5]

Instead of explicitly extracting the topology of the terrain on a global level, we leverage the knowledge of the *observation direction* of each point for locally extracting the relevant points. As illustrated in Figure 16, we estimate the local surface normal, $\mathbf{n}_i^{\text{surf}}$, at each map point in question, based on a small number of nearest neighbors. The directional ambiguity is resolved by orienting the normal along the observation direction, $\mathbf{d}_{\text{obs},i}$, that is, by enforcing $\mathbf{n}_i^{\text{surf}} \cdot \mathbf{d}_{\text{obs},i} > 0$. When computing the terrain-compliant robot pose (f_{ta}^{P}), only those points among the K nearest neighbors are considered whose local surface normal forms an angle of less than $\frac{\pi}{2}$ with the normal of the query pose:

$$\hat{\mathbf{z}}_{\text{MR}} \cdot \mathbf{n}_i^{\text{surf}} \overset{!}{>} 0\,. \tag{32}$$

Similarly, for estimating the terrain roughness of a map point $\mathbf{p}_i$ (f_{ta}^{τ}), a neighboring point $\mathbf{p}_k$ within r_{plane} and r_{res} is only considered if its local surface normal forms an angle of less than $\frac{\pi}{2}$ with the surface normal at $\mathbf{p}_i$:

$$\mathbf{n}_i^{\text{surf}} \cdot \mathbf{n}_k^{\text{surf}} \overset{!}{>} 0\,. \tag{33}$$

This methodology allows to correctly extract map points belonging to the ground plane, since the surface normals of ceiling points are oriented in the opposite direction. Consistent with our overall approach, the separation method is based only on local computations, avoiding any global mapping operations, such as clustering of points or explicit plane extraction.

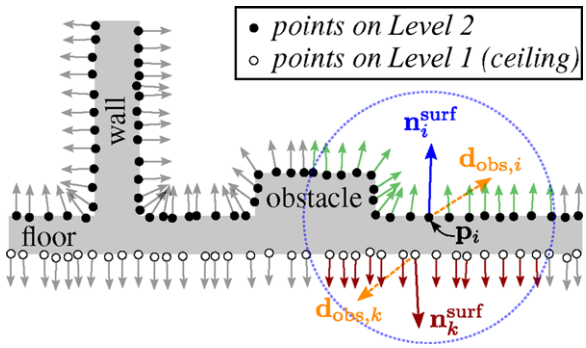

Figure 16. Separation of ground points from ceiling points in multilevel environments. The figure shows the cross section of an exemplary point cloud map, where the floor consists of two layers of points, representing the ground of Level 2 and the ceiling of Level 1, respectively. Leveraging the knowledge of the observation direction ($\mathbf{d}_{\text{obs}}$) of each map point, the local surface normals ($\mathbf{n}^{\text{surf}}$, solid arrows) are oriented to point *out of the surface*. When estimating the terrain roughness of a map point, $\mathbf{p}_i$, based on all points within a certain distance (dotted circle), only those points are considered whose local surface normals form an angle of less than $\pi/2$ with the normal at $\mathbf{p}_i$. Hence, points belonging to the ceiling of the lower floor are discarded.

5.4. Results

In the following, we apply the proposed terrain assessment method to point cloud maps (*planning maps*) recorded in various different outdoor environments. The maps are built from sensor data acquired with our robot ARTOR, using the ICP-based localization and mapping framework presented in Section 6.1. Table II lists the specific parameter values used for this evaluation. Unless otherwise stated, all results

[5]Technically, the same problem exists with walls or other thin objects in single-level environments. However, the standard approach is typically sufficient, since these objects are classified as obstacles in any event. Nevertheless, the approach proposed in the following similarly helps to extract the relevant points those cases as well.

Table II. Parameter values for terrain assessment.

Category	Description	Symbol	Value
Pose computation	number of nearest-neighbor points	K	100
Traversability computation	radii for plane fit and analysis of residuals	$r_{\text{plane}}, r_{\text{res}}$	1.5 m, 0.45 m
	fraction of points considered as noise	f_η	0.3
	weighting factor of terrain roughness	w_{rough}	0.6
	weighting factors of inclination angles	$w_{\text{roll}}, w_{\text{pitch}}$	0.2, 0.2

presented throughout the article are based on this set of terrain assessment parameters.

The analysis in this section focuses on the per-point terrain roughness, representing the basis—along with the inclination angles—for computing the traversability of specific robot poses. In the following figures, each map point is colored based on its assigned roughness value, ρ_i, according to the color map shown in Figure 1. Points with a roughness exceeding the threshold ($\rho_i > \rho_{\max}$) are considered as obstacles and plotted in dark red. The remaining points represent traversable areas. Their color indicates the roughness of the terrain at the respective location, varying from light blue (low roughness) to gray (high roughness).

Figure 17 shows example results of the terrain roughness computation in an unstructured outdoor environment, along with photographs of the respective scenes. Obstacles such as rocks, trees, and bushes are reliably classified as untraversable, while gravel paths and patches of grass are largely rated traversable. Within the traversable parts of the terrain, rough areas are accurately distinguished from smoother regions: in the rightmost example, the uneven, grassy area in the background appears considerably darker (i.e., of higher roughness) than the even gravel surface in the foreground. Further, the results demonstrate that the roughness computation is independent of the larger-scale terrain geometry and, thus, suited for application in general nonplanar environments. For example, the steep, yet smooth slopes and the respective transitions to flat areas in the two middle scenes are rated equally traversable than the planar surface in the rightmost example.

Figure 18 shows results of the terrain roughness computation in a structured, urban environment. The flat and smooth roads are consistently labeled with very low roughness, whereas large objects such as buildings, bollards, or vehicles are clearly classified as obstacles. Curbs of typical sidewalks are slightly too high for our robot to safely drive over. The first three scenes contain various different curbs, all of which are correctly marked as untraversable. In the second example from the left, the curbs on both sides of the road are flattened at the location of the crosswalk. The corresponding roughness values exactly reflect this feature: the curbs are classified traversable on both ends of the crosswalk and untraversable everywhere else. The rightmost example shows a map recorded on the lower level of a two-story parking garage. The roadway—including the ramp at the entrance—is rated perfectly traversable, while the small fence in the foreground and the various pillars in the background are identified as obstacles.

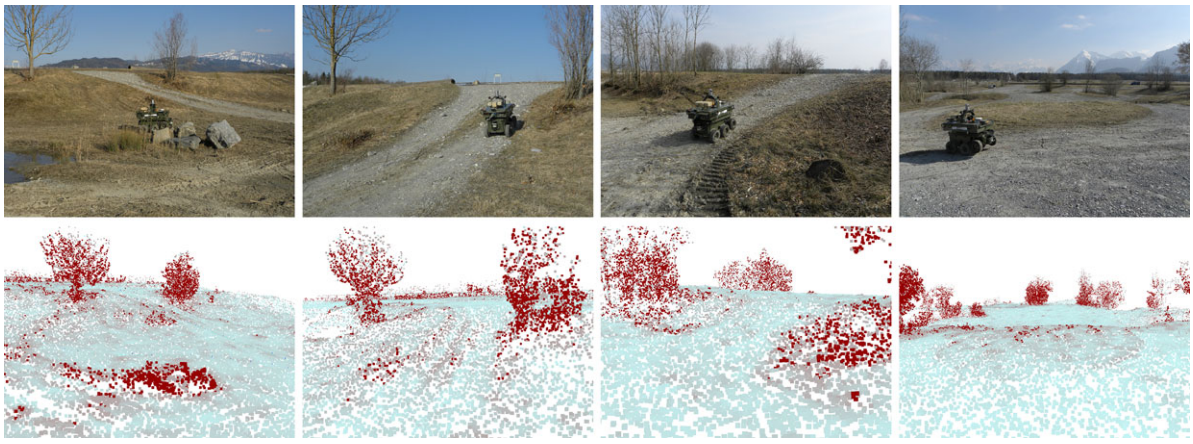

Figure 17. Terrain assessment results in an unstructured, nonplanar environment: photographs of four different scenes (top), and the corresponding point cloud maps colored according to the computed per-point terrain roughness (bottom). Trees, bushes, rocks, and other highly uneven areas are classified as obstacles (dark red), while gravel paths and patches of grass are generally rated traversable (light blue to gray).

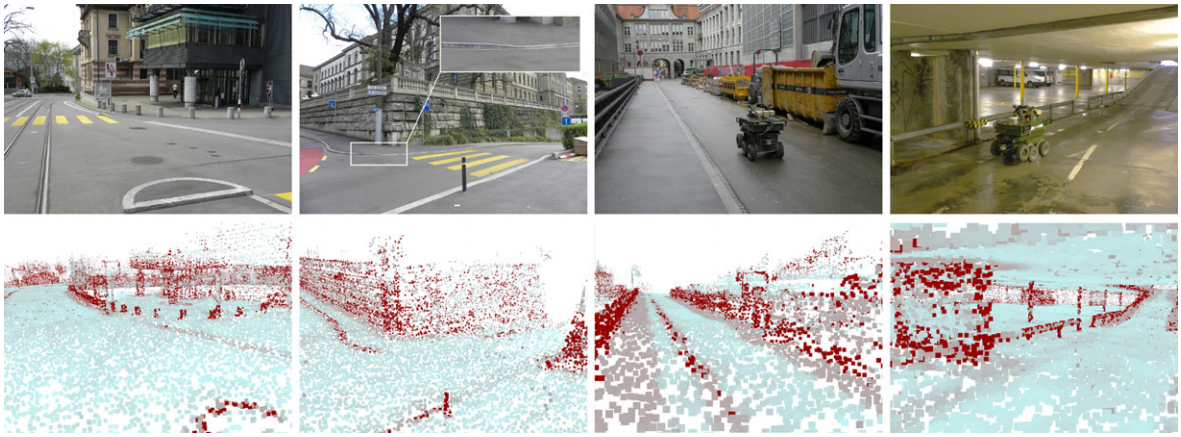

Figure 18. Terrain assessment results in an urban environment: photographs of four different scenes (top), and the corresponding point cloud maps colored according to the computed per-point terrain roughness (bottom). Obstacles such as walls, bollards, and curbs are identified as untraversable (dark red), while flat roads and sidewalks are classified as traversable (light blue to gray), typically with very low roughness values (light blue).

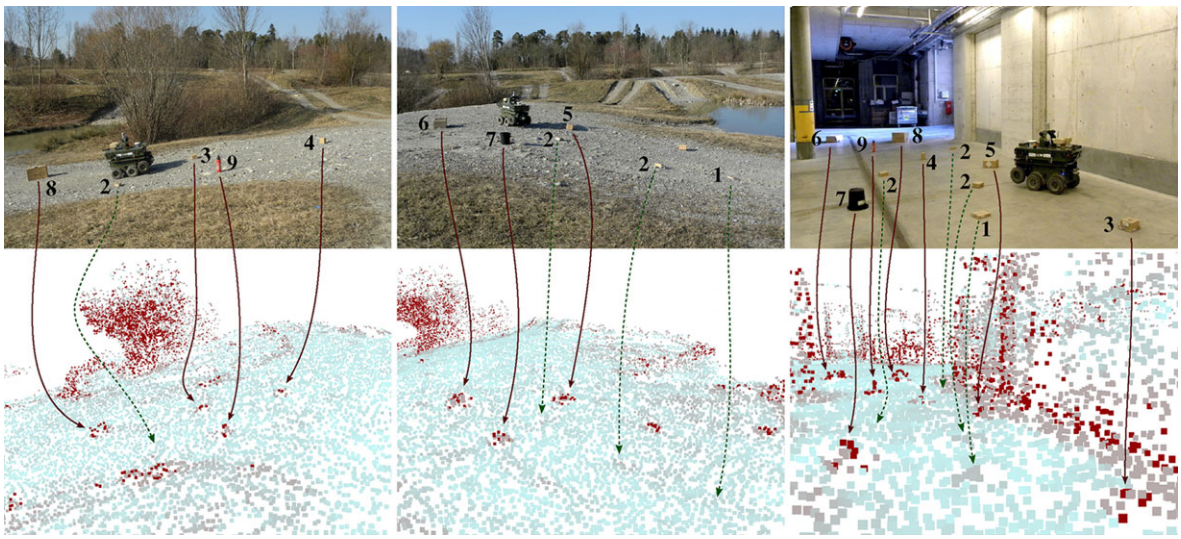

Figure 19. Systematic analysis of the obstacle detection on the basis of terrain roughness: photographs of three different scenes (top), and the corresponding point cloud maps colored according to the computed per-point terrain roughness (bottom). Eleven objects of known size (labeled according to their height, 1: lowest, 9: highest) were placed in an unstructured, nonplanar environment (left and middle), and in an urban environment on a smooth, even surface (right). The arrows point to the locations in the maps representing the different objects. The results show that—independent of the environment—objects of Types 1 and 2 are rated traversable, while the rest are classified as obstacles. This is consistent with the fact that only the objects of Types 1 and 2 exhibit a height less or equal to the maximum roughness threshold $\rho_{\max}$.

A systematic analysis of the terrain roughness computation and the resulting classification in obstacles and traversable points is shown in Figure 19. Eleven objects of known size were randomly placed in two different environments, in unstructured terrain on a hilly gravel area (left and middle), and on a smooth, paved surface inside a parking garage, representing an urban environment (right). The different types of objects were numbered from 1 to 9 according to their height, in ascending order. The set comprised a variety of different cardboard boxes (1—0.04 m, 2—0.08 m, 3—0.12 m, 4—0.15 m, 5—0.16 m, 6—0.2 m, 8—0.35 m), a bucket (7—0.3 m), and a small fire extinguisher (9—0.4 m). The heights of all objects except those of Types 1 and 2 exceed the maximum roughness threshold $\rho_{\max}$ (0.08 m). Hence, objects of Types 1 and 2 should be classified as traversable, and the rest should be identified as obstacles.

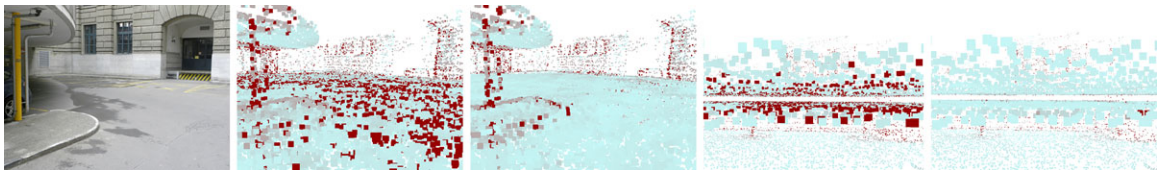

Figure 20. Special handling of multilevel environments in terrain assessment: photograph of the top level of a two-story parking garage, and the corresponding point cloud maps colored according to the computed roughness values (first pair: same perspective as photograph; second pair: cross-section views of the floor, with ground points at the top and ceiling points at the bottom). The left images of each pair show the roughness values computed ignoring the map's multilevel characteristics, spuriously classifying the area as untraversable. The right images of each pair show the resulting roughness when locally separating ground and ceiling points as described in Section 5.3, yielding a correct traversability classification.

In the photographs of Figure 19, each object is labeled with the number defining its type. The bottom row of the figure shows the corresponding point cloud maps, colored according to the computed terrain roughness, with arrows pointing to the locations of the objects in the maps. The results show that the obstacle detection is independent of the surrounding terrain and consistent with the maximum roughness threshold: in both different environments, all objects of Types 3 to 9 are classified as obstacles, while those of Types 1 and 2 (as well as the surrounding terrain surface) are rated traversable. Moreover, the analysis demonstrates that the proposed method allows to detect small obstacles, despite the comparatively low resolution of the point cloud maps.

Figure 20 demonstrates the necessity of specially handling maps recorded in multilevel environments (cf. Section 5.3). The photograph shows a scene on the top level of a two-story parking garage, where the entire paved area should clearly be classified as traversable. The point cloud maps on the right of the photograph cover parts of both the upper and the lower level of the garage. The first two images show the scene from the same perspective as the photograph, while the two rightmost images show cross-section views of the floor, with points on the top belonging to the ground of the upper level and points on the bottom belonging to the ceiling of the lower level. Each pair of images shows the resulting point roughness of the same scene, once without accounting for the multilevel nature of the environment (left), and once locally separating ground and ceiling points, using the approach described in Section 5.3 (right). The results show that the latter is crucial for correct traversability estimation in this kind of environment: ground and ceiling points are as close to each other that they would otherwise be interpreted as a single, rough surface, yielding the entire area labeled untraversable.

The results presented in this section demonstrate that our approach to terrain roughness computation and the resulting obstacle detection are applicable in a large variety of different environments. The method performs equally well in unstructured, 3D off-road terrain and in structured, urban environments, without adapting any parameters to the specific type of terrain. In general, the binary classification in traversable and untraversable points (resulting from the roughness) allows to reliably detect even small obstacles, while negotiable terrain irregularities, such as smooth slopes, patches of grass, or uneven gravel paths are still rated traversable. In rare cases, we observed obstacle points classified as traversable (false negatives) as well as points on traversable terrain labeled as obstacles (false positives). The latter can occur in places where points originating from dynamic objects have not (yet) been removed, and in areas with dense and tall, yet compressible vegetation. False negatives can appear due to locally insufficient resolution of the map in conjunction with very small or thin obstacles. However, if the maps are continuously updated during navigation (as they are in our framework; cf. Section 6.1), resolution irregularities—and thereby the probability of undetected obstacles—are constantly reduced, especially in the vicinity of the robot (where obstacle detection is most important). For safety-critical applications such as real-time local collision avoidance (cf. the safety layer of our navigation system; Section 6.3), the default resolution of the maps can be increased to avoid any false negatives in the obstacle detection. During the experiments reported in Section 7, our robot autonomously drove several kilometers in diverse environments. Using the proposed terrain assessment method, it was able to identify and avoid all critical hazards without any operator interaction.

Being a purely geometric approach, our terrain assessment method does not allow reliable detection of hazards such as water or deep mud. In our experiments, we observed that puddles and ponds either appeared as holes in the map (no points measured due to specular reflection of the laser beams) or as flat areas (water surface measured), probably depending on the turbidity of the water. On our testing ground, the water was typically rather cloudy in the small puddles and clear in the bigger ponds, yielding correct traversability classification. However, reliable detection of water and mud in generic environments would require the integration of additional sensors and specific terrain classification algorithms.

6. AUTONOMOUS NAVIGATION SYSTEM

In this section, we describe how the developed algorithms for motion planning and terrain assessment are integrated with a localization and mapping framework and a motion control module to build a complete system for autonomous navigation. We regard autonomous navigation of a robotic vehicle as the problem of moving from the current pose to a user-defined goal pose without any help of human operators, in an environment that has previously been explored and mapped, yet without using external localization sources such as GPS. Moreover, we do not assume the environment to be static. Throughout this section, we highlight the different measures taken to cope with dynamic scenarios and changes in the environment.

We first present an ICP-based localization and mapping framework in Section 6.1, and an approach for obtaining high-resolution, gravity-aligned point cloud maps suitable for terrain assessment and motion planning in Section 6.2. In theory, a trajectory computed according to the algorithm in Section 4 already includes the velocity commands to make a robot execute the desired maneuver: for a given forward velocity, v, the corresponding angular velocity is obtained as $\omega = v\kappa(s)$, where $\kappa(s)$ is the curvature of the respective trajectory segment $\mathbf{t}_i$ (given directly by the representation of $\mathbf{t}_i$ as curvature polynomial) at the particular point on the path. In practice, however, various sources of errors and inaccuracies necessitate a separate system for motion control. In Section 6.3, we develop a path-tracking controller with an integrated safety layer for high-rate collision checking, enabling accurate, safe, and reliable execution of planned trajectories. Finally, we give an overview of the complete autonomous navigation system and the interactions between all building blocks in Section 6.4.

6.1. Localization and Mapping

We have presented and evaluated an ICP-based framework for localization and mapping in prior work (Krüsi et al., 2015). The framework incorporates a hybrid topological/metric map representation. Metric information is stored in spatially bounded *submaps* arranged in an undirected graph, where edges represent relative transformations. This layout is known to result in a highly scalable system (Bosse, Newman, Leonard, & Teller, 2004). Localization drift—typically growing with traveled distance—does not result in inconsistencies in the map, since the global topology and the local metric information remain correct without global optimization. The input to the system are point clouds acquired by a high-rate spinning laser scanner, a Velodyne HDL-32E. The sensor rotates around the vertical axis with a frequency of 10 Hz, producing data packets of 12×32 range measurements of up to 100 m at approximately 1,800 Hz, or roughly 700,000 points per second. We use the open-source registration library `libpointmatcher` (Pomerleau, Colas, Siegwart, & Magnenat, 2013) for building point cloud maps and localizing within the same based on ICP. We refer to our previously published work (Krüsi et al., 2015) for a detailed description and discussion of the parameters for the different `libpointmatcher` modules, which were tuned to enable accurate, real-time localization and mapping with the given sensor setup.

Our current system is based on this approach but includes three main enhancements that are essential for using the point cloud maps for motion planning: 1) undistortion of laser scans using inertial measurements, 2) identification of dynamic elements and extraction of the static part of the environment, and 3) continuous map updates. The framework operates in two different modes (cf. Figure 21): exploration of an unknown environment or navigation in a previously mapped area. In the exploration phase, the topological/metric map is built: using incremental localization, a sequence of local submaps along the driven route are generated and stored in a database. When navigating a known environment, the robot localizes against submaps in the graph, avoiding accumulation of error in the pose estimate over time.

In both modes, we first assemble the laser scans of each 360-deg turn of the sensor into monolithic point clouds. We build undistorted point clouds by taking into account the motion of the vehicle during the scanning process. On the basis of inertial and odometry measurements, we compute the incremental motion of the sensor between the acquisition of any two data packets and transform the point measurements into a single coordinate frame for each turn. The composition of all incremental transforms along a full turn yields an estimate of the vehicle's motion since the completion of the last 360-deg point cloud. Undistorted point clouds and motion estimates, the latter being used to update the initial guess for registration, are fed to an ICP-based *incremental localization* module. This module simultaneously builds and localizes within a *sliding map*, that is, a map of limited size that moves with the robot. Technically, this is implemented using a global map with a limited number of points and a limited point density (max. 40,000 pts and 20 pts/m^3 in our setup). We tested this setup in extensive field tests involving navigation in the kilometer range in both structured and unstructured environments. It was found to yield highly accurate incremental localization, with typical position errors below 0.2 % of distance traveled.

In the exploration mode, a graph of overlapping submaps is built. As illustrated in Figure 21 on the right, point clouds are added to the most recent submap after being transformed to the submap's coordinate frame, using the pose estimate provided by the incremental localization module. Swapping submaps is triggered by the distance traveled and the overlap between the current scan and the submap. Whenever the former exceeds a certain threshold or the latter falls below a minimum value, the current submap is saved to a database and a new map is

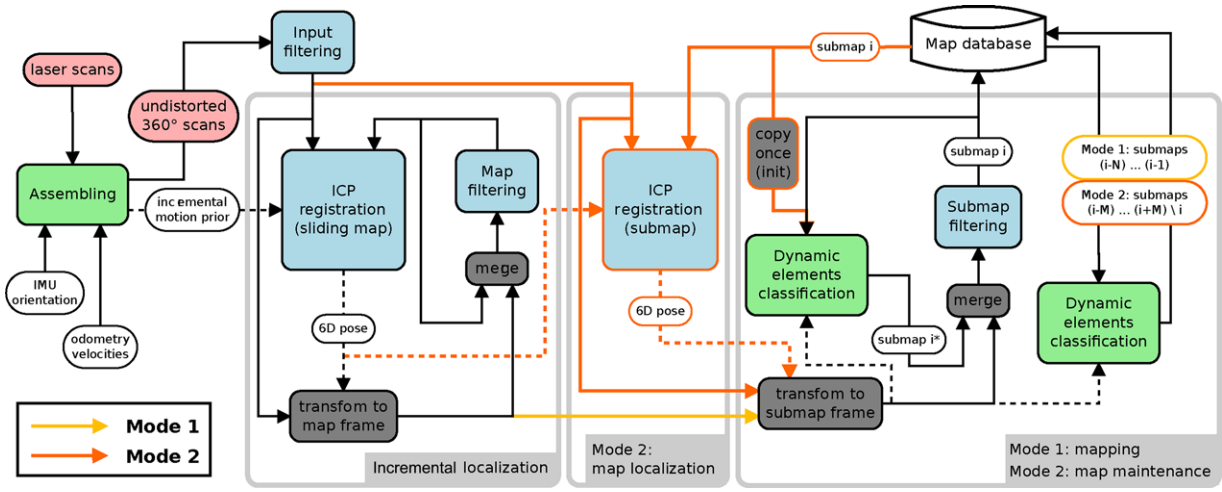

Figure 21. Structure of the ICP-based localization and mapping system for exploration of an unknown environment (Mode 1), and for navigation in a previously mapped environment (Mode 2). Incoming laser measurements are assembled to monolithic 360-deg point clouds, accounting for the motion of the sensor during acquisition. The *incremental localization* module maintains a size-limited map moving with the robot, ensuring continuous operation even in highly dynamic environments. In Mode 1, the *mapping* module builds a graph of spatially bounded submaps connected by relative transformations. In Mode 2, the *map localization* module registers scans and the respective closest submap in the database, providing drift-free localization. In parallel, the submaps in the database are continuously updated by the *map maintenance* module.

initialized. In our system, we set the thresholds to 10 m and 84 %, respectively. The *submap filtering* module subsamples the maps to enforce a maximum point density per cubic meter (10 in our setup), yielding a uniform distribution of points in the maps. Apart from adding and removing points, we continuously estimate their probability of being dynamic, that is, of originating from a mobile object, using the approach presented in (Pomerleau, Krüsi, Colas, Furgale, & Siegwart, 2014). The quality of this estimate increases with the number of observations of a certain point. Hence, we update the points' probability of being dynamic not only in the most recent submap, but simultaneously in prior submaps, as long as they overlap with the sensor's field of view (FOV).[6]

When navigating in previously mapped environments, the *map localization* module loads submaps from the database for registration of incoming laser scans. In prior work, we have shown that this approach yields reliable and accurate map-based localization, with typical localization errors (with respect to the current submap) in the range of a few centimeters (Krüsi et al., 2015). In every iteration, the currently relevant submap is chosen from three alternatives—1) the submap used in the previous iteration, 2) its predecessor, and 3) its successor in the graph—as the one whose origin is spatially closest to the robot's current position. In addition, a *relocalization* functionality enables switching to any spatially close submap in the graph, thus allowing the robot to move freely in the mapped environment without being restricted to following the previously driven route. Relocalization is triggered when the overlap between the incoming laser scans and the currently loaded submap falls below a certain threshold. In this case, all submaps within a given radius around the robot's position are searched for the one providing the largest overlap.

To account for dynamic environments, the localization module is surrounded by modules for *incremental localization* and *map maintenance*. The former allows to handle even large changes in the environment, in which the overlap of the current scan with the recorded map is too low for reliable registration. The approach is to maintain a *sliding map* along the route, built from the most recent laser scans (exactly similar to the method applied when exploring the environment for the first time). In normal operation, the resulting pose estimate is used to estimate the incremental motion for map-based localization. If the latter fails, however, the system continues to operate, using the pose estimate given by incremental localization as final result. In this phase, every scan is nevertheless fed to the *map localization* module, in order to restart operation as soon as registration with the map is successful again. In parallel to localization, the *map maintenance* module continuously updates the submaps in the database. Whenever a new submap is loaded, the map data are copied to a separate process and updated as in the *mapping* module during exploration. Simultaneously, the map points' probability of being dynamic is updated in a window of submaps around the current reference submap,

[6]Because the maximum distance between two submaps is considerably smaller than their spatial size (given by the sensor range), multiple maps overlap at each point along the route.

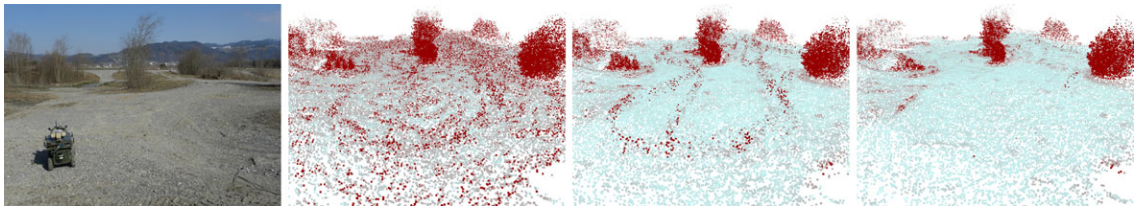

Figure 22. Influence of undistorting laser scans and identifying dynamic elements on terrain assessment. The three point cloud maps (colored according to the computed terrain roughness) represent the scene shown in the photograph on the left. They were built from the same raw laser data recorded while driving a short loop, with two people walking in the laser's FOV. The leftmost map was built without accounting for the sensor's motion (no undistortion) and without removing dynamic points. The middle map was established using our scan undistortion algorithm but again without excluding dynamic points. Finally, the rightmost map shows the result of the entire pipeline, including undistortion of laser scans and removal of points classified as dynamic. It is evident that both of these techniques are indispensable for traversability classification: only in the rightmost map, the entire gravel area is correctly identified as traversable.

and highly dynamic points are deleted periodically. Overall, this amounts to adding new points to the submaps if new objects come to the FOV of the sensor, and removing points belonging to objects that have disappeared and are no longer observable. When localization switches to the next submap, the updated submap is written to the database, replacing its older version.

6.2. Extraction of Maps for Motion Planning

Point cloud maps provide the basis for our terrain assessment and motion planning algorithms. The core idea of our approach is to reuse the maps built for localization purposes. In the following, we describe a simple and computationally inexpensive methodology for extraction of high-resolution, gravity-aligned *planning maps* $\mathcal{M} \in \mathfrak{M}_{\text{pl}}$ as defined in Section 3.1 from the graph of submaps produced by the localization and mapping system (Section 6.1).

During the initial exploration phase, we compute the orientation of each submap with respect to a gravity-aligned coordinate frame using an inertial measurement unit (IMU) and save the respective rotation parameters in the map database. *Planning maps* are generated *on demand* for every planning request, ensuring up-to-dateness of the information. In a first step, the graph of submaps is searched for the path between the robot's current position and the vertex closest to the goal. Subsequently, all submaps along this path are transformed to the gravity-aligned coordinate frame attached to the submap at the start position. Since the submaps' spatial size is considerably larger than the average distance between two maps, their concatenation results in a map with greatly increased density. This is a positive effect, as the single submaps are typically too sparse for reliable terrain assessment. However, point density is a trade-off between accuracy and computational complexity of terrain assessment. Hence, we limit the number of points in the planning maps by restricting each of the submaps to a sphere of radius r_{max} around its center, before they are merged to a single map. Discarding points far away from the submaps' center points simultaneously reduces map errors due to inaccuracies in the relative transformations between the submaps (induced by localization drift in the exploration phase). In our setup, the nominal distance between submaps is $d_{\text{nom}} = 10$ m, and we cut the submaps at a radius of $r_{\text{max}} = 20$ m. These two parameters can be tuned to control the density of the resulting maps according to the specific requirements. With submaps at around 10 pts/m^3, the given values yield planning maps containing typically 30 to 40 pts/m^3.

Figure 22 shows example point cloud maps computed according to this methodology. Moreover, it demonstrates the importance of undistorting laser scans acquired with a moving vehicle and identifying dynamic elements for extraction of the static part of the environment (cf. Section 6.1). Our robot was steered along one loop in the environment shown in the photograph on the left, with two people walking in the laser's FOV, one in front of the robot, and one behind. It is obvious that the large gravel area in the foreground should be rated largely traversable, because of its smoothness and the absence of obstacles. The remainder of the figure shows three slightly different point cloud maps of the photographed scene (built from the same raw data), colored according to the computed terrain roughness (cf. Section 5). The first map (left) was built neglecting the laser sensor's motion during data acquisition and without identifying dynamic elements. The consequences of these simplifications are evident: traces of moving objects and off-surface points (noise) due to distorted laser scans essentially result in the entire area being classified as untraversable. The second map (middle) was assembled using our undistortion method based on IMU and odometry measurements but still without removing points originating from dynamic objects. The resulting roughness estimation represents a large improvement over the example without scan undistortion: the gravel area is mostly rated traversable, with the exception of the traces of the two people. The last map (right)

was built with both undistortion of scans and identification of dynamic points enabled. The terrain roughness was computed discarding points with a probability of being dynamic above the threshold $p_{\text{dyn,max}} = 0.3$. The result is a clean map, in which the flat, smooth part of the terrain is rated traversable, and only trees and bushes remain as obstacles.

6.3. Motion Control

We assume the environment for navigation to be subject to both high-frequency and low-frequency changes over time. The latter are handled by continuously updating the map database (cf. Section 6.1) and thus automatically considered by the motion planner, regularly replanning trajectories based on the most recent map information. High-frequency changes caused by dynamically moving other agents are detected by the classification of map points as dynamic or static. However, planning trajectories to circumnavigate highly dynamic objects would require very high replanning rates and specialized collision avoidance methods, which are outside the scope of this work. Hence, we use the following strategy for autonomous navigation in dynamic environments:

- The motion planner only considers the *static* part of the map, denoted by $\mathcal{M}_{\text{stat}} \in \mathfrak{M}_{\text{pl}}$, for computing a trajectory, $\mathcal{T}$, connecting the start and the goal pose. $\mathcal{T}$ is replanned as frequently as possible. $\mathcal{M}_{\text{stat}}$ is obtained by discarding points with a probability of being dynamic above the threshold $p_{\text{dyn,max}} = 0.3$ according to the most recent classification.
- A *path-tracking controller* is used to make the robot follow the resulting trajectory.
- On top of this, a *safety layer* operating at high frequency ensures collision-free navigation. The course of the robot is forward-simulated for a given time horizon and checked for potential collisions, using a map consisting of both $\mathcal{M}_{\text{stat}}$ and the current laser scan. The robot is stopped in the case of an imminent collision.
- A special *recovery behavior* allows to free the robot in situations in which the motion planner cannot find any feasible trajectory.

This setup is well suited for navigation in moderately dynamic environments. Collisions with moving objects are avoided by temporally slowing down or stopping, without leaving the planned trajectory; lower-frequency changes in the environment are eventually integrated in the map and taken into account by the planner, adapting the trajectory accordingly. In the following, we describe the building blocks of the motion control system in more detail.

6.3.1. Path-Tracking Controller

We define path-tracking control as the problem of computing linear and angular velocity controls, v_{ctrl} and ω_{ctrl}, that make a robot follow a given trajectory.[7] Our path-tracking algorithm is a nonlinear controller using state feedback linearization. The control inputs are given by the singed lateral offset from the reference trajectory, ϵ_{L}, the error in the heading angle with respect to the closest point on the trajectory, ϵ_{H}, and the curvature of the closest point on the trajectory, κ_{t}. For a given robot pose, $\mathbf{T}_{\text{MR}}$, and a reference trajectory, $\mathcal{T}$, these terms are computed as follows. After finding the respective closest trajectory segment, $\mathbf{t}_{i_c}$, the pose $\mathbf{T}_{\text{MR}}$ is projected to the plane in which $\mathbf{t}_{i_c}$ is represented, resulting in a planar pose, $\xi_{\text{p}} := [x_{\text{p}}, y_{\text{p}}, \phi_{\text{p}}, \cdot]$, expressed in the coordinate frame attached to the trajectory node $\mathcal{N}_{i_c}$. The reference pose on the trajectory, $\xi_{\text{t}} := [x_{\text{t}}, y_{\text{t}}, \phi_{\text{t}}, \kappa_{\text{t}}]$, expressed in the same coordinate frame, is found as the closest point to ξ_{p} on $\mathbf{t}_{i_c}$. The lateral offset, ϵ_L, is given by the Euclidean distance between ξ_{p} and ξ_{t}, with a negative sign if ξ_{p} is *on the right* of the reference trajectory. The heading error is obtained as $\epsilon_{\text{H}} = \phi_{\text{t}} - \phi_{\text{p}}$[8].

Depending on the curvature of the reference trajectory and the current lateral offset, the linear target velocity is dynamically adapted within the user-defined range, $[v_{\text{min}}, v_{\text{max}}]$, where $0 < v_{\text{min}} < v_{\text{max}}$. The speed is set to the maximum value if the reference trajectory is a straight line and the robot is perfectly following this path. For increasing curvature and path-tracking error, the velocity is gradually reduced,

$$v_{\text{ctrl}} = v_{\text{max}} - (v_{\text{max}} - v_{\text{min}}) \left(\frac{|\kappa_{\text{t}}|}{\kappa_{\text{max}}} + \frac{|\epsilon_{\text{L}}|}{\epsilon_{\text{L,max}}} - \frac{|\kappa_{\text{t}}||\epsilon_{\text{L}}|}{\kappa_{\text{max}}\epsilon_{\text{L,max}}} \right), \quad (34)$$

where $\epsilon_{\text{L,max}} \in \mathbb{R}_{>0}$ is a parameter for the maximum tolerable lateral tracking error. The rotational velocity is computed according to a control law similar to the one of Marshall, Barfoot, & Larsson (2008),

$$\omega_{\text{ctrl}} = v_{\text{ctrl}}\kappa_{\text{t}} + \frac{-k_{\text{L}}\epsilon_{\text{L}} - k_{\text{H}} v_{\text{ctrl}} \sin \epsilon_{\text{H}}}{v_{\text{ctrl}} \cos \epsilon_H}, \quad (35)$$

where k_{L} and k_{H} are positive constants (controller gains). Unlike Marshall et al. (2008), who assume the reference path to be a straight-line, we include a feedforward term, $v_{\text{ctrl}}\kappa_{\text{t}}$, representing the required rotational velocity in the case of zero tracking error.

[7] Controlling the vehicle motion to comply with the desired velocities is assumed to be accomplished by a low-level controller, which is outside the scope of this article.

[8] Note that ϵ_{L} and ϵ_{H} in general are not continuous at the transitions between two trajectory segments of $\mathcal{T}$ (unless the robot perfectly follows the trajectory or the environment is planar). While this is an unfavorable characteristic of the algorithm, it has shown to be of no practical relevance in our experiments, since the discontinuities are typically small for small values of ϵ_{L} (which are enforced by the controller).

Using this controller for precise goal acquisition at the end of the trajectory requires two modifications when the robot is close to the goal. First, $v_{\max}$ is set to a low value once the goal is near, allowing for timely stopping at the end. Second, the reference trajectory is prolonged with a short straight line segment at the goal pose. This avoids any undesirable behavior of the controller that may arise from a suddenly ending reference path.

6.3.2. Safety Layer

The task of the safety layer is to monitor the environment at a high frequency and to stop the robot in the event of an imminent collision with dynamic objects. The basic concept consists in forward-simulating the course of the robot up to a certain instant of time in the future, using a simple motion model steered by the path-tracking controller. The time horizon of the simulation, denoted by T_{sim}, is adapted to the current speed and the vehicle's braking capabilities. The simulated path is checked for traversability with a point cloud map composed of both the static part of the environment, $\mathcal{M}_{\text{stat}}$, and the most recent 360-deg laser scan. The latter is subsampled according to a certain maximum density threshold, yielding an even distribution of points in the resulting map.

The motion model for forward-simulation respects the limitations of the robot's powertrain, namely, the maximum linear and angular velocities, $v_{\max}$ and $\omega_{\max}$, the maximum linear and angular acceleration (or deceleration), $\dot{v}_{\max}$ and $\dot{\omega}_{\max}$, and the system-inherent delay, Δt_{d}, defined as the typical time duration elapsing after a change in the desired robot velocities, v_{ctrl} and ω_{ctrl}, until this change affects the actual robot motion. We simplify the model by assuming that, after a change in $(v_{\text{ctrl}}, \omega_{\text{ctrl}})$, the robot continues moving during Δt_{d} with its current velocities $(v_{\text{curr}}, \omega_{\text{curr}})$ and afterward changes the latter instantaneously according to the desired values, but respecting the acceleration and speed limits. Further assuming the controller rate to coincide with the system delay, safety layer collision checking amounts to executing $N_{\text{sim}} = \frac{T_{\text{sim}}}{\Delta t_{\text{d}}}$ iterations of a simulation loop, each consisting of the following operations:

1. Forward-simulate the robot pose for the duration of Δt_{d} according to $(v_{\text{curr}}, \omega_{\text{curr}})$. In the first iteration, these velocities are set to the actual current robot motion estimated by odometry. The forward-simulated pose, denoted by $\mathbf{T}_{\text{MR}}$, is given as the end point of a circular arc segment in the x-y plane of the current pose, $\mathbf{T}_{\text{M}\bar{\text{R}}}$.
2. Update $\mathbf{T}_{\text{M}\bar{\text{R}}}$ by computing the terrain-induced 6D pose corresponding to $\mathbf{T}_{\text{MR}}$ and the associated terrain traversability using f_{ta}. If the terrain is untraversable, terminate and command the robot to stop.
3. Compute the velocity controls $(v_{\text{ctrl}}, \omega_{\text{ctrl}})$ by applying the path-tracking controller at the pose $\mathbf{T}_{\text{M}\bar{\text{R}}}$.
4. Update $(v_{\text{curr}}, \omega_{\text{curr}})$ according to $(v_{\text{ctrl}}, \omega_{\text{ctrl}})$ and the acceleration and speed limits:

$$\begin{aligned} \Delta v &= \min(|v_{\text{ctrl}} - v_{\text{curr}}|, \dot{v}_{\max}\Delta t_{\text{d}}) \cdot \text{sign}(v_{\text{ctrl}} - v_{\text{curr}}), \\ \Delta\omega &= \min(|\omega_{\text{ctrl}} - \omega_{\text{curr}}|, \dot{\omega}_{\max}\Delta t_{\text{d}}) \cdot \text{sign}(\omega_{\text{ctrl}} - \omega_{\text{curr}}), \\ v_{\text{curr}} &\leftarrow \min(v_{\text{curr}} + \Delta v, v_{\max}), \\ \omega_{\text{curr}} &\leftarrow \min(|\omega_{\text{curr}} + \Delta\omega|, \omega_{\max}) \cdot \text{sign}(\omega_{\text{curr}} + \Delta\omega). \end{aligned} \tag{36}$$

If the simulation terminates without detection of potential collisions, the control inputs to the real robot are given by the values of v_{ctrl} and ω_{ctrl} as computed in the first iteration of the simulation. Hence, the path-tracking controller inherently accounts for the system delay.

6.3.3. Recovery Behavior

In rare cases, this motion control framework can lead to situations in which the robot cannot proceed in forward motion any more. For example, this can happen when a moving object suddenly blocks the path of the robot and does not go out of the way anymore. To continue navigation in such events, we integrate a simple recovery behavior that is triggered when the motion planner is unable to compute any feasible trajectory. The robot is commanded to slowly move backward on a straight-line path, while the safety layer is permanently active and the motion planner continuously attempts to compute a feasible trajectory. As soon as the latter is successful, navigation continues according to the regular scheme. The recovery behavior is demonstrated in the experiment described in Section 7.4.

6.4. System Overview

Figure 23 gives an overview of the autonomous navigation system, showing its building blocks and the data flow between the different modules. The system relies on measurements from three sensors mounted on the robotic vehicle: a 3D laser scanner, an IMU, and wheel encoders. The corresponding point clouds, the vehicle's orientation with respect to gravity (computed from the IMU measurements either by an internal filter or by an additional software module), and the odometry velocities are fed to the *localization and mapping* module (cf. Figure 21, Mode 2), which provides map-based localization of the vehicle in 6D (at 10 Hz in our setup), and continuously updates its internal maps based on the incoming laser measurements. In addition, it delivers a point cloud map for motion planning (cf. Section 6.2) when requested by the *motion planning* module. This map consists of an unordered set of 3D points as defined in Section 3.1; it does not contain any information about the trajectory driven when mapping the environment.

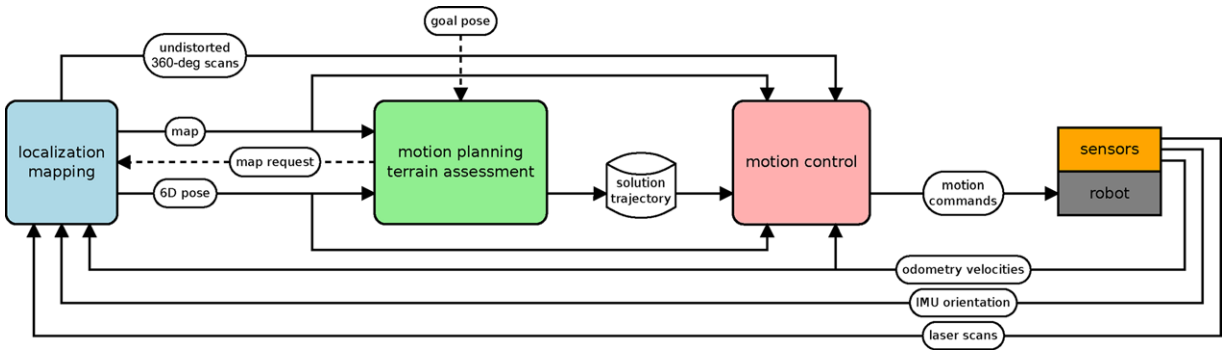

Figure 23. Autonomous navigation system. The three building blocks of the system are represented by a module for map-based localization and map maintenance (*localization and mapping*), a module for planning trajectories from the current robot pose to the desired goal pose (*motion planning and terrain assessment*), and a module for high-rate path tracking and hazard avoidance (*motion control*). On the basis of measurements from onboard sensors (laser, IMU, encoders), the overall system computes suitable motion commands to navigate the robot safely and efficiently to a user-specified goal pose.

The motion planning module computes a trajectory, $\mathcal{T}$, connecting the current robot pose with a *goal pose* (specified by the operator or a mission planning module), using the planning and terrain assessment methods described in Sections 4 and 5. The trajectory is replanned as frequently as possible, since changes in the environment may require the trajectory to be adapted. To speed up planning in simple cases (short distance between start and goal, smooth and largely traversable terrain), we initially try to connect the start and the goal pose by computing a planar connection trajectory and projecting it to the terrain surface, using the approach described in Section 4.3. If this results in a feasible trajectory, we directly proceed with local optimization, omitting the potentially expensive steps of sampling-based initial trajectory generation (RRT) and global optimization (RRT*). Moreover, if the goal pose has not changed since the last successful trajectory computation, the planner first inspects whether the previous solution trajectory is still feasible with the latest map data. If this is the case, the previous solution is reused as initial trajectory and only optimized locally.

Finally, it is the task of the *motion control* module (cf. Section 6.3) to compute suitable motion commands for the robot to make it follow the reference trajectory provided by the motion planning module, and to ensure safety by stopping the vehicle in case of an imminent collision. Unlike the planner, the *motion control* module is designed to operate at a fixed frequency (10 Hz in our setup), which is crucial for stable control and reliable detection of potential hazards.

7. EXPERIMENTS

We analyze our motion planning and terrain assessment framework in an offline experiment involving a large number of random planner queries, using a point cloud map recorded in a rough terrain environment (Section 7.1). Subsequently, we present the results of extensive field tests, demonstrating the suitability of the proposed methods and algorithms for practical application in outdoor robotic navigation. In particular, we tested the autonomous navigation system (as summarized in Section 6.4) in three different scenarios, involving a total of more than 4 km of autonomous driving: 1) long-range waypoint navigation in rough terrain (Section 7.2), 2) navigation in a two-story parking garage (Section 7.3), and 3) navigation in a dynamic environment (Section 7.4). Table III lists the values of the motion control parameters used for the navigation experiments. The motion planning and terrain assessment parameters were kept at the values listed in Table I and Table II, respectively. All experiments presented in the following have been conducted with the same setup of the navigation system and the same set of parameter values, without any environment-specific adaptations.

The navigation experiments were conducted with our robot, ARTOR, a skid-steered electrically powered vehicle (shown in Figure 24 and in numerous photographs throughout the article). The relatively compact dimensions of ARTOR (length: 1.3 m, width: 0.7 m) enable passing through standard doors and other narrow passages, while the six-wheeled configuration yields good performance in rough and steep outdoor terrain. The robot is equipped with an array of different sensors, of which only three are relevant for our navigation system: 1) a 3D laser scanner (Velodyne HDL-32E) on top of the vehicle, 2) an IMU (Xsens MTi) mounted below the laser, and 3) wheel encoders. ARTOR is fitted with an onboard computer, featuring an eight-core Intel Core i7 processor, running Ubuntu Linux and the Robot Operating System ROS. Sensor data are provided as ROS messages, and the different components of the navigation system, implemented in C++, communicate with each other via ROS messages and ROS services. During

Table III. Parameter values for motion control.

Category	Description	Symbol	Value
Powertrain constraints	maximum linear and angular velocities	$v_{\max}$, $\omega_{\max}$	1 m/s, 1 rad/s
	maximum linear and angular acceleration/deceleration	$\dot{v}_{\max}$, $\dot{\omega}_{\max}$	0.8 m/s^2, 0.8 rad/s^2
	system-inherent delay	Δt_{d}	0.4 s
Path-tracking controller	minimum linear velocity	$v_{\min}$	0.3 m/s
	maximum-tolerated lateral tracking error	$\epsilon_{\mathrm{L,max}}$	1.5 m

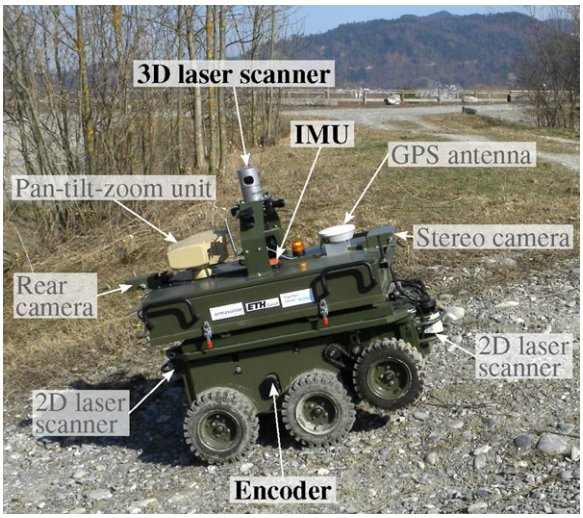

Figure 24. Our robot ARTOR, a skid-steered electrically powered vehicle. The three wheels on either side of the vehicle are driven synchronously by a single motor. ARTOR is equipped with an array of sensors, of which only three are used by our autonomous navigation system: a 3D laser scanner (Velodyne HDL-32E), an IMU (Xsens MTi), and two encoders measuring the wheel speeds.

autonomous navigation, all computations are executed on the onboard computer.

7.1. Quantitative Offline Evaluation

For a quantitative analysis of our motion planning and terrain assessment framework, we conducted an experiment comprising 5,000 different random planner queries. We present a statistical evaluation of the results, focusing on two aspects: 1) the runtimes (computational complexity) of the three different stages of the planner and 2) the effectiveness of the two trajectory optimization methods (RRT*-based and local optimization). For this evaluation, we used the point cloud map recorded for the navigation experiment in Section 7.2, spanning an area of approximately 40,000 m^2 in the unstructured, nonplanar terrain shown in Figure 27. For each planner query, the respective *planning map*, $\mathcal{M}$, was extracted from the graph of submaps with the approach outlined in Section 6.2. The 5,000 pairs of start and goal poses for the planner queries were chosen according to the following procedure, ensuring the *theoretical* existance of a feasible trajectory between each pair:

1. Select a random point on the approximately 1km long mapping path (the path driven for recording the map).
2. Define the start pose at this point, with the heading angle randomly chosen between $-\pi$ and π.
3. Choose a random distance, d_{SG}, between 10 m and 100 m, and a random direction (forward/backward).
4. Define the goal pose at the distance d_{SG} from the start pose *on the mapping path*, depending on the chosen direction, with a random heading angle between $-\pi$ and π.
5. Check the traversability of the terrain in small areas around both poses (ensuring enough space for turning). Discard the pair of poses if the terrain is untraversable.

The motion planner successfully found a feasible trajectory in all 5,000 queries of the experiment, resulting in 5,000 different trajectories between 3.5 m and 121 m in length, distributed as shown in Figure 25 on the left. The remaining diagrams in Figure 25 demonstrate the effectiveness of the two trajectory optimization methods. The RRT*-based optimization on the global level resulted in a path length reduction of up to 77.6 %, with an average over all trajectories of 12.2 %. The local optimization yielded trajectories improved by up to 22.9 % in cost (as defined in Eq. (18)), with an average of 8.2 %.

Figure 26 analyzes the computational complexity of the three different planner stages, on the basis of the runtimes measured during computation of the 5,000 trajectories.[9] The experiment was conducted with a C++ implementation of the planning framework, on a single core of a 2.6-G Hz Intel Core i7 3720QM processor. The results show that—in all three stages—the runtime and its variance increase approximately proportionally with the length of the computed trajectory. In average, the initial trajectory generation with

[9]Notice that these runtimes include all computations for terrain assessment, since the latter is embedded in the planner and only executed *on demand* at specific query points.

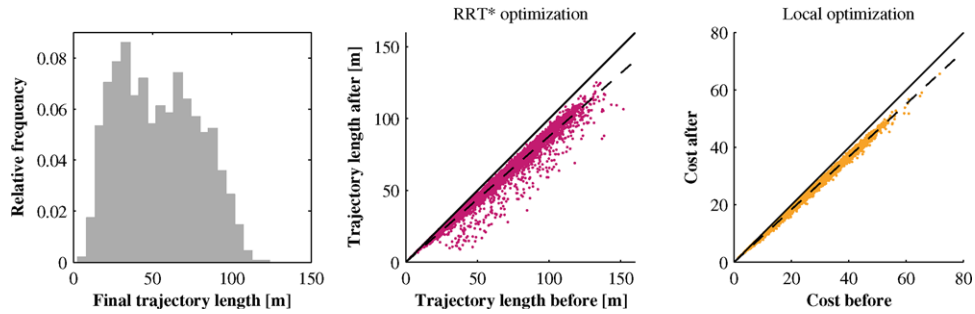

Figure 25. Statistical evaluation (5,000 random planner queries) of the effectiveness of the trajectory optimization algorithms: distribution of the length of the final trajectories (left), trajectory length reduction resulting from the RRT*-based optimization (middle), and reduction in the trajectory cost (including length, curvature, and traversability as defined in Eq. (18)) achieved by the local trajectory optimization (right). The dots represent individual trajectories, and the dashed lines indicate the mean improvement in length and cost of 12.2 % and 8.2 %, respectively. The outliers in the middle plot demonstrate the importance of trajectory optimization at the global level: the RRT*-based optimization algorithm is able to remove large detours that may be included in paths computed by the RRT planner.

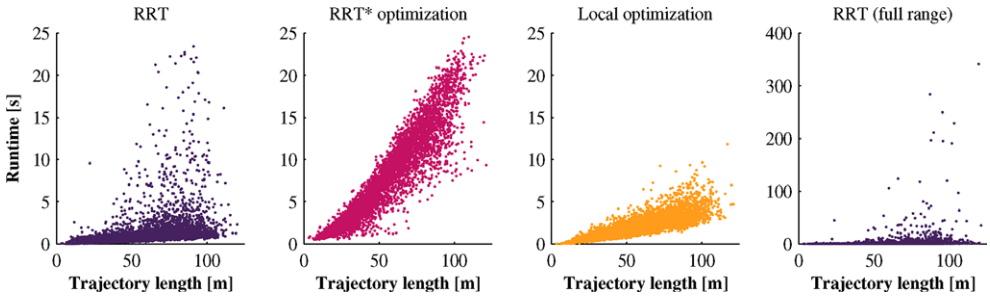

Figure 26. Statistical evaluation (5,000 random planner queries) of the planner's computational complexity. The runtimes of all three stages increase roughly proportionally with the length of the computed trajectory. Generation of the initial trajectory (RRT) is generally the least expensive step (13.2 ms/m median runtime per meter trajectory). However, the inherent randomness of the RRT planner leads to a certain number of outliers, in rare cases yielding high runtimes, as shown in the rightmost chart. The runtimes of RRT*-based optimization (median: 131 ms/m) and local optimization (median: 31.6 ms/m) depend on the chosen termination criteria, reflecting the selected compromise between optimality and computational complexity.

the RRT planner is the least computationally expensive of the three phases. However, the distribution of the RRT runtimes contains a considerable amount of outliers, reflective the randomness inherent to the approach and the dependency of the required runtime on the complexity of the particular terrain (for example, narrow passages typically require disproportionately many tree expansions to be detected). The trajectory optimization algorithms, especially the RRT*-based optimization, require more time than the initial path generation. However, these runtimes depend on the specific, user-defined termination parameters, reflecting a selectable compromise between optimality and computational complexity (the initial trajectory already represents a valid solution to the planner query, satisfying all relevant constraints). Concerning the RRT*-based optimization, we have chosen a conservative value for the termination criterion, which yields a high probability of eliminating *all* larger detours in the path but may result in relatively high average runtime.

7.2. Long-Range Waypoint Navigation in Rough Terrain

The experiment described in this section was targeted at assessing the suitability of the autonomous navigation system, particularly of the motion planning and terrain assessment framework, for navigation over long distances in nonplanar, unstructured outdoor terrain. It was conducted in a military testing ground for off-road vehicles in Thun, Switzerland, featuring these exact characteristics. The area consists of numerous small hills and valleys, overlaid with a network of rough gravel paths, surrounded by trees, bushes, and patches of grass, as shown in Figure 27.

The first step consisted of a *mapping run*: we manually steered the robot along an approximately 1-km-long route, while running the ICP-based localization and mapping system of Figure 21 in exploration mode (Mode 1) to create the initial graph of point cloud maps. Subsequently, we defined 11 waypoints spread over the entire mapped area, each

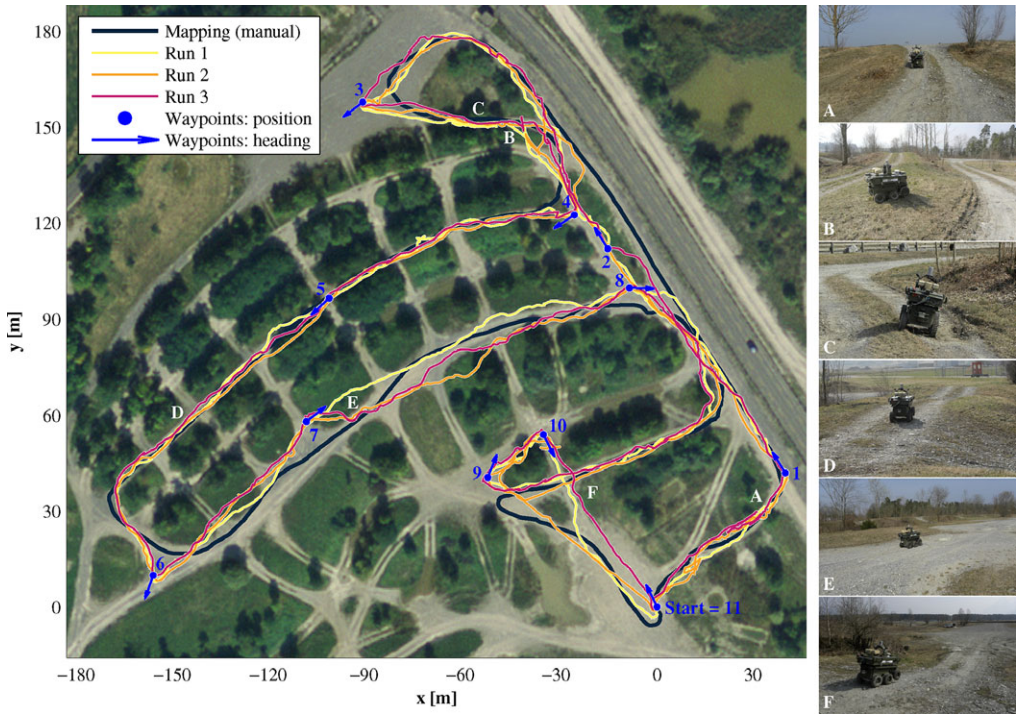

Figure 27. Waypoint navigation experiment in rough terrain. The robot was instructed to autonomously navigate to 11 waypoints, using previously recorded point cloud maps. The left image shows the path driven for mapping the environment and the autonomously driven paths during the three runs of the experiment (measured with a GPS receiver on the robot). Although the current navigation system requires all waypoints to lie within the mapped area, it still allows navigation to places that are reasonably far away from the path of the mapping run (e.g., waypoint 10). As shown in the photographs on the right (the letters mark the locations where they were taken), the environment along the driven routes consisted of steep slopes (A), patches of grass (B), rough, uneven gravel paths (C), hilly terrain (D), as well as wide, open areas (E, F). *Source aerial image: Bundesamt für Landestopografie swisstopo (Art. 30 GeoIV): 5704 000 000.*

consisting of a 3D position and a heading angle, represented in the map's coordinate frame. The robot's task was defined as navigating to all 11 waypoints in sequential order, starting at the same location as in the mapping run and using the navigation system as described in Section 6.4. To avoid any operator interaction during the experiment, we implemented a simple program for automatic switching between the waypoints and sending the respective goal coordinates to the planner.

We conducted three runs of the same experiment, resulting in a total of 33 individual start-to-goal navigation tasks between the waypoints. Figure 27 shows an aerial view of the environment, with the poses of the waypoints and the paths driven by the robot as measured by GPS.[10] Table IV summarizes the most important results of the experiment. In total, the robot autonomously drove a distance of more than 3km, with an average speed slightly above 0.5 m/s. In each of the three runs, it reached all 11 waypoints, reliably avoiding all hazards present in the environment, such as sheers, obstacles (trees, bushes, rocks), and too-steep paths. The localization and mapping system, as well as the planning and terrain assessment module, operated flawlessly over the course of the entire experiment. Although working reliably most of the time, the motion control module turned out to be the most critical part of the system, causing three situations that required short manual interventions (one in

[10]Notice that we did not use GPS data for localization but for evaluation purposes only.

Table IV. Waypoint navigation experiment in rough terrain: results.

Run ID	Distance Traveled (m)	Duration (min)	Average Speed (m/s)	Operator Interventions (duration (s))
1	1085	35.1	0.51	1 (13)
2	1130	35.3	0.54	1 (7)
3	1090	32.7	0.56	1 (6)

each run, discussed below). However, neither of these cases posed any danger to the robot or the environment.

The root of the motion control problems lies in the robot's slow and imprecise low-level motor control (working at only 10 Hz, with a typical system delay of 0.4 s). As a result—despite the efficient path-tracking controller—the robot cannot always follow the planned path with the desired accuracy, especially in slippery, uneven terrain. We have incorporated this characteristic by defining a high maximum lateral path-tracking error ($\epsilon_{L,max} = 1.5$ m), allowing relatively large deviations from the planned trajectory without replanning the latter (which can be computationally expensive due to the large distance between the waypoints). As a consequence of this, the robot may drive through areas that have not been checked for traversability by the planner. While this does not compromise safety[11], it may yield situations where the robot is stopped by the safety layer due to an obstacle blocking the path, requiring a reverse maneuver (recovery behavior) and replanning of the trajectory to the goal.

Over the course of the experiment, we observed this behavior several times. In most cases, the robot reversed for a few meters and continued normal operation once the planner had computed a new trajectory. However, two situations required short operator interventions (one in Run 1 and one in Run 2), since the recovery behavior (reversing on a straight line) did not bring the robot to a place from where the planner could find a feasible trajectory for continuation in forward motion. In Run 3, we observed a different failure case: although commanded to drive forward by the motion control module, the robot did not move, potentially due to a temporary error in the low-level motor control system. In all of these three cases, we manually steered the robot for a few seconds (cf. Table IV), moving it to a traversable area where it continued navigating autonomously.

The large distances between the waypoints in the experiment (up to more than 100 m) resulted in relatively high runtimes of the motion planner. In most cases, the robot stopped for several seconds when a waypoint was reached, until the trajectory leading to the next waypoint was computed. Subsequently, since the environment was largely static, the trajectories could typically be followed in continuous motion. Only in rare cases, due to small changes in the environment or inaccurate path tracking, the current planner trajectory had to be recomputed, which could cause the robot to stop temporarily.

7.3. Navigation in a Two-Story Parking Garage

Autonomous navigation in urban environments involves different challenges than navigation in unstructured off-road terrain. While traversability assessment may be less complex due to smooth and largely planar driving surfaces, man-made structures may include complex topologies and narrow passages, posing a challenge for motion planning and control. We conducted an experiment in a parking garage at the ETH campus in Zurich, Switzerland, demonstrating the suitability of our approach for autonomous navigation in structured, urban environments, including multilevel topologies. The garage consists of two floors, each containing several rows of parking spots, numerous pillars, and some small fences. The entrance to the garage is located on the upper floor, which is connected to the lower level by two narrow single-lane ramps, one for each driving direction, as shown in the photographs of Figure 28.

With the exception of the nature of the environment, the setup of the experiment was similar to the one described in Section 7.2. In the *mapping run*, we manually steered the robot along the officially prescribed route for cars: starting at the upper level, we drove down via the first ramp, did one full round at the lower level, and finally returned to the upper level using the second ramp. For autonomous navigation, we defined the start pose at the entrance of the garage on the upper level. The robot was instructed to navigate to three waypoints (defined by position and heading), the first one located on the lower level, approximately below the starting point, the second one on the other end of the lower level, and the third one again on the upper level.

We conducted three runs of the experiment, each with the exact same setup. Figure 28 shows a top view on a point cloud map of the parking garage, with the poses of the waypoints and the paths driven by the robot. Table V lists the most important results of the experiment. The robot successfully completed each run without any operator interaction, reaching all predefined waypoints. The complex, two-level topology and the general narrowness of the garage rendered both planning and execution of trajectories more challenging than in spacious outdoor terrain, resulting in a slightly lower average speed compared to the previous experiment. In the middle of the lower floor, a small fence with two narrow openings (shown in the photographs C an H in Figure 28) separates the two traffic lanes. In all three runs, the planner successfully detected these openings, resulting in significantly shorter trajectories between the start and Waypoint 1 and between Waypoints 2 and 3 compared to the path driven in the mapping run.

The environment was largely static during the experiment, and most of the parking spots were empty. From this point of view, some of the paths driven on the relatively wide road between Waypoint 1 and 2 appear slightly suboptimal and too twisty. This is partly due to the limited amount of computation assigned to trajectory optimization (RRT*) and partly due to imprecise path tracking resulting from the shortcomings of the low-level motor control discussed in Section 7.2. Moreover, similar to the experiment in rough terrain, the latter yielded several situations that required the robot to stop and plan a new trajectory, sometimes combined

[11] The *safety layer* is always active and stops the robot in case of an imminent collision.

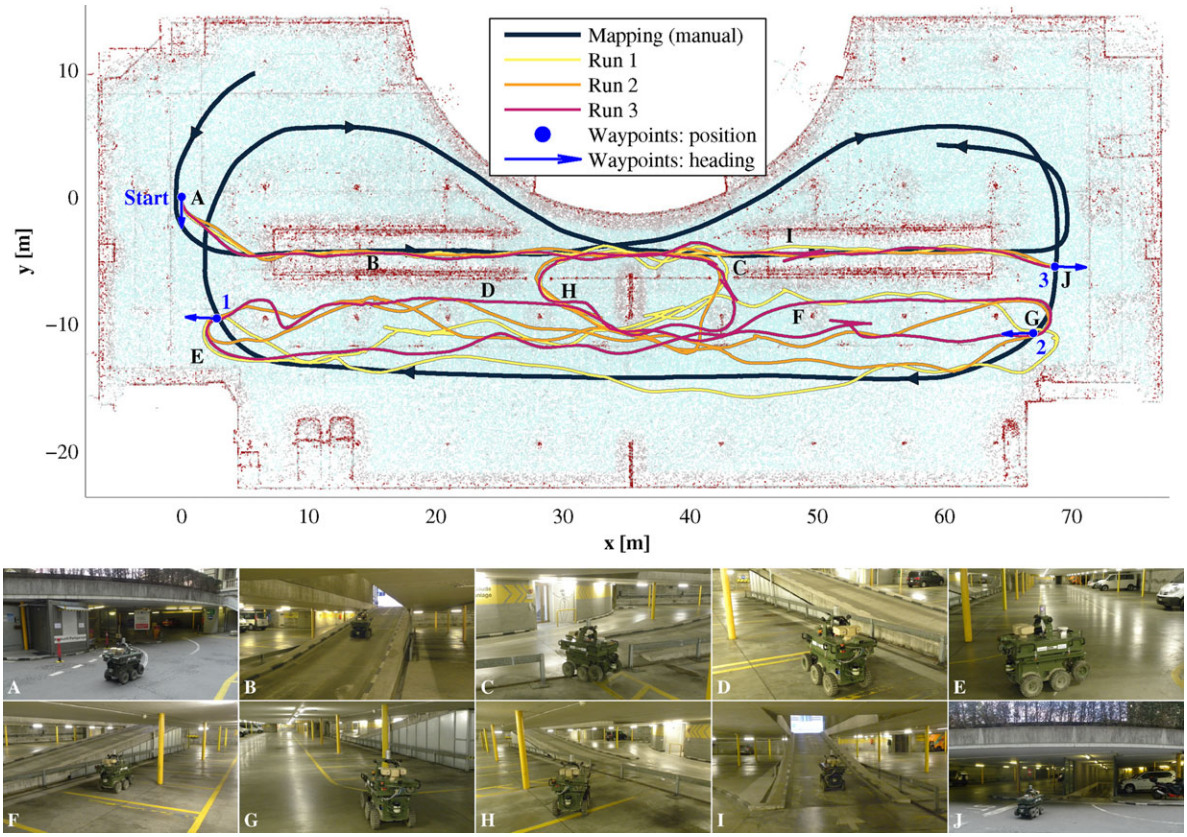

Figure 28. Waypoint navigation experiment in a two-story parking garage. Starting at the entrance of the garage on the upper level, the robot was instructed to navigate to three waypoints (1 and 2 on the lower floor, 3 on the upper floor). The figure shows a top view on a point cloud map of the environment, with the manually driven path for initial mapping (along the official route through the garage), and the paths driven autonomously during the three runs of the experiment (measured by the ICP-based localization system). The photographs show the robot in different places along a typical route in sequential order, with the letters marking the locations in the map where the pictures have been taken.

Table V. Waypoint navigation experiment in two-story parking garage: results.

Run ID	Distance Traveled (m)	Duration (min)	Average Speed (m/s)	Operator Interventions (duration (s))
1	301	14.5	0.35	0 (0)
2	272	10.9	0.42	0 (0)
3	287	12.7	0.38	0 (0)

with a reverse maneuver (recovery behavior). However, all these cases were handled fully autonomously.

7.4. Navigation in a Dynamic Environment

Dynamic environments represent a challenge for all different components of an autonomous navigation system. Localization must be robust to outliers, the mapping framework needs to constantly update the maps of the environment, the planner must be able to dynamically adapt or recompute trajectories when new information is available, and the motion control layer needs to react quickly to moving objects in order to avoid collisions. We conducted an experiment in a dynamic environment, demonstrating the suitability of our autonomous navigation system for application in places that change over time and may be populated by other moving agents. In particular, we show that continuously updating the point cloud maps and estimating the points' probability of being dynamic (Section 6.1) allow the planning and terrain assessment module to react to changes in the environment by recomputing the trajectory if necessary. Moreover, we demonstrate how the safety layer and the recovery behavior embedded in the motion control module (Section 6.3) enable safe autonomous navigation even in the presence of quickly moving other agents.

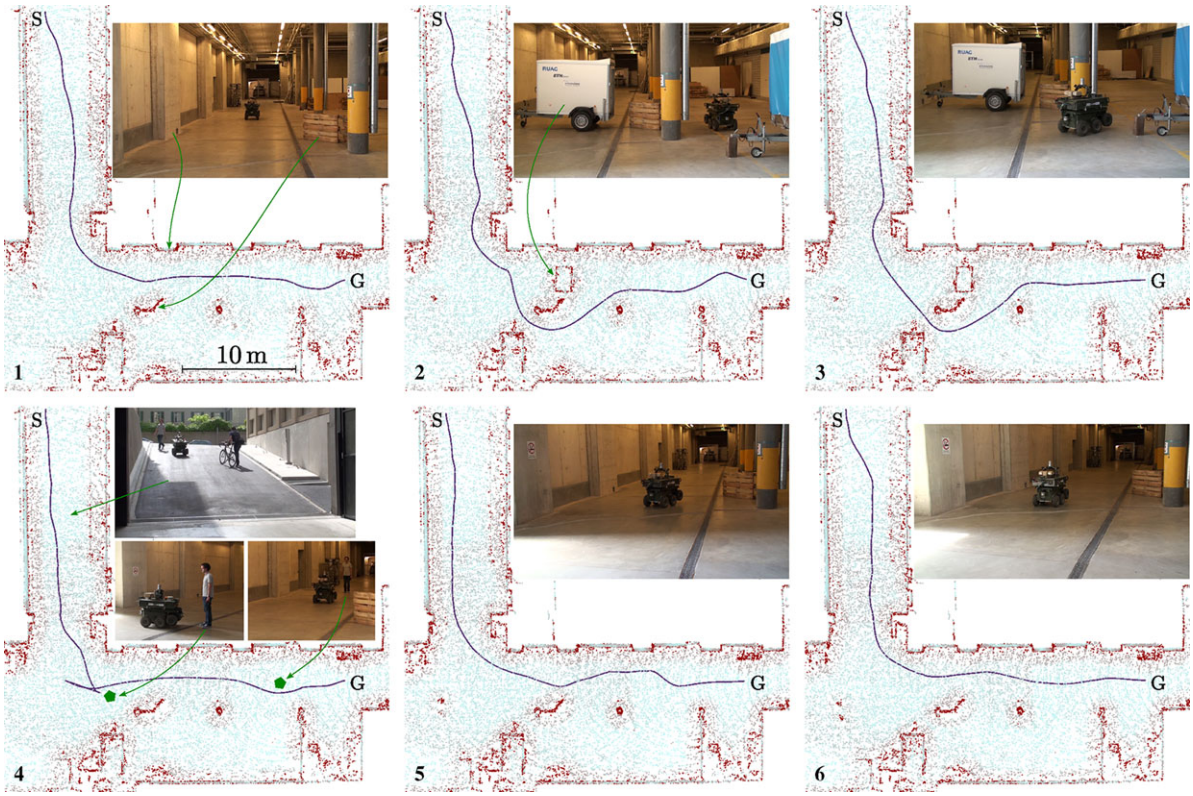

Figure 29. Navigation experiment in a dynamic environment. The robot was tasked to navigate six times to a given goal pose (G) inside a parking garage, starting outside on top of the garage's access ramp (S), while various modifications were made in the environment. The figure shows the six trajectories driven by the robot (as measured by ICP localization), each drawn on the *planning maps* of the place as estimated *at the end of the respective run* (colored according to the computed terrain roughness). As shown in the photographs, we blocked the path with a trailer for the second and third runs. In the fourth run, a person was walking around and twice blocked the robot's path (at the locations marked with a pentagon). Runs 1, 5, and 6 were conducted in static environment without additional obstacles. The resulting trajectories—all driven completely autonomously—demonstrate how continuous map updates combined with real-time collision checking (safety layer) allow the system to react appropriately to changes in the environment.

The experiment was conducted in an underground parking garage at the ETH campus in Zurich, Switzerland. The robot was repeatedly tasked to navigate to a predefined goal location inside the garage, starting outside at the top of a wide ramp representing the access to the garage. In total, we conducted six runs of the experiment, each with the same start and goal locations but making various modifications in the environment. An initial graph of point cloud maps was created during a manually controlled mapping run. During the autonomous runs—all completed without any operator interventions—these maps were continuously updated by our ICP-based localization and mapping module. Hence, each run started with the maps representing the most recent estimate of the environment's geometry.

Figure 29 shows the trajectories driven in the six runs of the experiment, each plotted in a *planning map* representing the static part of the environment as estimated *at the end of the respective run*, along with corresponding pictures of the environment. For the first run, the environment was not modified with respect to the state during the mapping run. The robot drove down the ramp, smoothly turning to the left by 90 degrees, and proceeding on a largely straight path to the goal. For the second run, we blocked the previously driven path by placing an obstacle (a small trailer) between a large pillar and the wall of the garage. This modification was not visible to the robot unless it entered the garage (at the end of the access ramp). Hence, the planner initially computed a similar path than in first run. However, once the trailer appeared in the FOV of the laser, the map was updated, and the planned trajectory was identified as unfeasible. The robot stopped for a short moment (at the kink in the trajectory after the entrance to the garage) until the planner had computed an alternative trajectory to the goal, avoiding the newly added obstacle through a narrow

passage on the right. Between the second and the third runs, the environment was not modified. However, at this time, the trailer was recorded in the map (visible in the figures), allowing the planner to compute a feasible trajectory for the entire route from the beginning. As a consequence, the robot reached the goal on a smoother trajectory and without stopping on the way.

For the fourth run, we removed the trailer again. Instead, a person was constantly walking around in the FOV of the laser and twice deliberately blocked the robot's path. The first time, this was done immediately in front of the robot. The safety layer detected the imminent collision and stopped the vehicle, and the recovery behavior was triggered, since the person did not move out of the way and the robot was too close to drive around in forward motion. It reversed, computed a new, collision-free trajectory, and continued navigation. Close to the goal, the person blocked the way once more; this time a few meters ahead of the robot. Hence, the planner had enough time to compute an avoidance maneuver, allowing the robot to proceed without stopping. In the meantime, the trailer had (at least partially) been removed from the map, allowing the robot to drive to the goal on a similar path than in the first run. Finally, we conducted two more runs without any additional obstacles (the same setup as in the first run), targeted at demonstrating that objects are eventually completely removed from the maps when they have disappeared from the environment. We assumed that after the fourth run (dynamic environment) the maps might still be polluted by some traces of the walking person. However, it turned out that the respective points were classified reliably as highly dynamic already during the experiment, resulting in a clean map of the static part at the end of the run. Hence, the trajectories of the robot in the two last runs were—as desired—very similar to the one in the first run.

8. CONCLUSION

This paper presented a complete system for autonomous navigation of ground robots in arbitrarily complex 3D environments. At the core of the approach is a motion planner combining sampling-based planning and local trajectory optimization to compute system-compliant trajectories in the full 6D space of robot poses. The proposed "on demand" terrain assessment module enables planning directly on 3D point cloud maps, without explicit surface reconstruction, and without any artificial discretization of maps or trajectories. We have integrated these algorithms with ICP-based localization and mapping, path-tracking control, and real-time local obstacle detection, resulting in a complete system for autonomous navigation in GPS-denied environments. The effectiveness and practicality of the proposed methods has been demonstrated in field tests involving several kilometers of autonomous driving in largely different environments: rough, unstructured outdoor terrain, a two-level parking garage, and a dynamic urban environment spanning outdoor and indoor areas.

Though the planning and terrain assessment algorithms are connected to each other by the motivation of this work—enabling navigation in 3D environments based on point cloud maps—their application is not restricted to the one shown in this article. The motion planner could be used with any other terrain assessment method providing the same functionality as ours, namely, mapping a 6D query pose to a terrain-compliant pose and a corresponding scalar traversability value. The proposed trajectory optimization algorithm is a general approach for local refinement of ground robot trajectories on 3D terrain, with the advantages of respecting important system constraints and not requiring computation of optimization gradients. Finally, the terrain assessment method can be seen as a general approach for identification of geometry and traversability of surfaces represented as point clouds.

Planning trajectories over long distances can be computationally expensive due to the high degree of detail. In our current setup, the full global trajectory is replanned if a part of it is detected to be in collision after a map update. In the future, we plan to address this issue by implementing an intelligent replanning functionality, updating the trajectory only *locally* where required. On the level of the localization and mapping system, we intend to globally optimize the graph of submaps in offline processing to further increase the metric consistency of the map on the large scale. Simultaneously, additional connections (loop closures) could be created between submaps that are metrically close and connected by traversable terrain, but topologically far from each other in the initial graph. The resulting *network* of submaps would increase the efficiency of planning, since the set of submaps required for a specific query could be determined more precisely. Finally, we plan to integrate the developed terrain assessment and motion planning algorithms with a system for autonomous exploration of completely unknown environments, which could be used to replace the initial, manually controlled mapping run, or to extend an existing graph of submaps during navigation.

ACKNOWLEDGMENTS

This work is supported by the armasuisse S+T UGV research program. We would like to thank Renaud Dubé, Timo Hinzmann, and Jörn Rehder for their help with the experiments, and Hannes Sommer for his advice concerning mathematical formulation.

APPENDIX: VIDEO ATTACHMENT

The video accompanying this article shows our robot navigating autonomously in different environments, while visualizing the point cloud maps, the terrain roughness estimation, and the planned trajectories. In particular, the video

includes the first run of the waypoint navigation experiment in rough terrain (Section 7.2), the third run of the experiment in the two-story parking garage (Section 7.3), and Runs 1 to 5 of the experiment conducted in a dynamic urban environment (Section 7.4).

REFERENCES

Arvanitakis, I., Tzes, A., & Thanou, M. (2013). Geodesic motion planning on 3D-terrains satisfying the robot's kinodynamic constraints. In Proceedings of the Annual Conference of the IEEE Industrial Electronics Society (IECON), Vienna, Austria.

Bajracharya, M., Howard, A., Matthies, L. H., Tang, B., & Turmon, M. (2009). Autonomous Off-road navigation with end-to-end learning for the LAGR Program. Journal of Field Robotics, 26(1), 3–25.

Bellutta, P., Manduchi, R., Matthies, L., Owens, K., & Rankin, A. (2000). Terrain perception for DEMO III. In Proceedings of the IEEE Intelligent Vehicles Symposium, Dearborn, MI.

Bosse, M., Newman, P., Leonard, J., & Teller, S. (2004). Simultaneous Localization and map building in large-scale cyclic environments using the Atlas framework. International Journal of Robotics Research, 23(12), 1113–1139.

Breitenmoser, A., & Siegwart, R. (2012). Surface reconstruction and path planning for industrial inspection with a climbing robot. In Proceedings of the International Conference on Applied Robotics for the Power Industry (CARPI), Zürich, Switzerland.

Brock, O., & Khatib, O. (2002). Elastic strips: A framework for motion generation in human environments. International Journal of Robotics Research, 21(12), 1031–1052.

Cherif, M. (1999). Motion Planning for All-Terrain Vehicles: A Physical Modeling Approach for Coping with Dynamic and Contact Interaction Constraints. IEEE Transactions on Robotics and Automation, 15(2), 202–218.

Cherubini, A., & Chaumette, F. (2013). Visual navigation of a mobile robot with laser-based collision avoidance. International Journal of Robotics Research, 32(2), 189–205.

Colas, F., Mahesh, S., Pomerleau, F., Liu, M., & Siegwart, R. (2013). Path planning and execution for search and rescue ground Robots. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, Tokyo, Japan.

Connors, J., & Elkaim, G. (2007). Manipulating B-spline based paths for obstacle avoidance in autonomous ground vehicles. In Proceedings of the ION National Technical Meeting, San Diego, CA.

Daily, M., Harris, J., Keirsey, D., Olin, K., Payton, D., Reiser, K., ... Wong, V. (1988). Autonomous cross-country navigation with the ALV. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Philadelphia, PA.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische Mathematik, 1(1), 269–271.

Dolgov, D., Thrun, S., Montemerlo, M., & Diebel, J. (2010). Path planning for autonomous vehicles in unknown semi-structured environments. International Journal of Robotics Research, 29(5), 485–501.

Furgale, P., & Barfoot, T. D. (2010). Visual teach and repeat for long-range rover autonomy. Journal of Field Robotics, 27(5), 534–560.

Ganganath, N., Cheng, C.-t., & Tse, C. K. (2015). A constraint-aware heuristic path planner for finding energy-efficient paths on uneven terrains. IEEE Transactions on Industrial Informatics, 11(3), 601–611.

Garrido, S., Malfaz, M., & Blanco, D. (2013). Application of the fast marching method for outdoor motion planning in robotics. Robotics and Autonomous Systems, 61(2), 106–114.

Gaw, D., & Meystel, A. (1986). Minimum-time navigation of an unmanned mobile robot in a 2-1/2D world with obstacles. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), San Francisco, CA.

Gennery, D. B. (1999). Traversability analysis and path planning for a planetary rover. Autonomous Robots, 6(2), 131–146.

Gingras, D., Dupuis, E., Payre, G., & de Lafontaine, J. (2010). Path planning based on fluid mechanics for mobile robots using unstructured terrain models. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Anchorage, AK.

Goldberg, S. B., Maimone, M. W., & Matthies, L. (2002). Stereo vision and rover navigation software for planetary exploration. In Proceedings of the IEEE Aerospace Conference, Big Sky, MT.

Guernane, R., & Achour, N. (2011). Generating optimized paths for motion planning. Robotics and Autonomous Systems, 59(10), 789–800.

Guo, Y., Parker, L. E., Jung, D., & Dong, Z. (2003). Performance-based rough terrain navigation for nonholonomic mobile robots. In Proceedings of the Annual Conference of the IEEE Industrial Electronics Society (IECON), Roanoke, VA.

Hamner, B., Singh, S., Roth, S., & Takahashi, T. (2008). An efficient system for combined route traversal and collision avoidance. Autonomous Robots, 24(4), 365–385.

Helmick, D., Angelova, A., & Matthies, L. (2009). Terrain adaptive navigation for planetary rovers. Journal of Field Robotics, 26(4), 391–410.

Hertle, A., & Dornhege, C. (2013). Efficient extensible path planning on 3D terrain using behavior modules. In Proceedings of the European Conference on Mobile Robots (ECMR), Barcelona, Spain.

Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., & Stuetzle, W. (1992). Surface reconstruction from unorganized points. In Proceedings of the 19th annual conference on Computer Graphics and Interactive Techniques (SIGGRAPH), Chicago, IL.

Howard, A., & Seraji, H. (2001). Vision-based terrain characterization and traversability assessment. Journal of Robotic Systems, 18(10), 577–587.

Howard, T. M. (2009). Adaptive model-predictive planning for navigation in complex environments (PhD dissertation). Carnegie Mellon University, Pittsburgh PA.

Howard, T. M., & Kelly, A. (2007). Optimal rough terrain trajectory generation for wheeled mobile robots. International Journal of Robotics Research, 26(2), 141–166.

Iagnemma, K., Genot, F., & Dubowsky, S. (1999). Rapid physics-based rough-terrain rover planning with sensor and control uncertainty. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Detroit, MI, USA.

Ishigami, G., Otsuki, M., & Kubota, T. (2013). Range-dependent terrain mapping and multipath planning using cylindrical coordinates for a planetary exploration rover. Journal of Field Robotics, 30(4), 536–551.

Kalakrishnan, M., Chitta, S., Theodorou, E., Pastor, P., & Schaal, S. (2011). STOMP: Stochastic trajectory optimization for motion planning. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China.

Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. International Journal of Robotics Research, 30(7), 846–894.

Karumanchi, S., Allen, T., Bailey, T., & Scheding, S. (2010). Non-parametric Learning to aid path planning over slopes. International Journal of Robotics Research, 29(8), 997–1018.

Kavraki, L. E., Svestka, P., Latombe, J.-C., & Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation, 12(4), 566–580.

Kelly, A., & Stentz, A. (1998). Rough terrain autonomous mobility: Part 2. An active vision, predictive control approach. Autonomous Robots, 5(2), 163–198.

Kelly, A., Stentz, A., Amidi, O., Bode, M., Bradley, D., Diaz-Calderon, A., ... Warner, R. (2006). Toward reliable off road autonomous vehicles operating in challenging environments. International Journal of Robotics Research, 25(5-6), 449–483.

Khan, Y. N., Komma, P., & Zell, A. (2011). High resolution visual terrain classification for outdoor robots. In Proceedings of the IEEE International Conference on Computer Vision (ICCV), Barcelona, Spain.

Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. International Journal of Robotics Research, 5(1), 90–98.

Kobilarov, M. B., & Sukhatme, G. S. (2005). Near time-optimal constrained trajectory planning on outdoor terrain. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Barcelona, Spain.

Konolige, K., Agrawal, M., Blas, M. R., Bolles, R. C., Gerkey, B., Solà, J., & Sundaresan, A. (2009). Mapping, navigation, and learning for off-road traversal. Journal of Field Robotics, 26(1), 88–113.

Krebs, A., Pradalier, C., & Siegwart, R. (2010). Adaptive rover behavior based on online empirical evaluation: Rover-terrain interaction and near-to-far learning. Journal of Field Robotics, 27(2), 158–180.

Krüsi, P., Bücheler, B., Pomerleau, F., Schwesinger, U., Siegwart, R., & Furgale, P. (2015). Lighting-invariant adaptive route following using iterative closest point matching. Journal of Field Robotics, 32(4), 534–564.

Kümmerle, R., Hähnel, D., Dolgov, D., Thrun, S., & Burgard, W. (2009). Autonomous driving in a multi-level parking structure. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Kobe, Japan.

Lacaze, A., Murphy, K., & DelGiorno, M. (2002). Autonomous mobility for the Demo III experimental unmanned vehicles. In Proceedings of AUVSI, Lake Buena Vista, FL.

Lacroix, S., Mallet, A., Bonnafous, D., Bauzil, G., Fleury, S., Herrb, M., & Chatila, R. (2002). Autonomous rover navigation on unknown terrains: Functions and integration. International Journal of Robotics Research, 21(10-11), 917.

Lalonde, J.-F., Vandapel, N., Huber, D. F., & Hebert, M. (2006). Natural terrain classification using three-dimensional ladar data for ground robot mobility. Journal of Field Robotics, 23(10), 39–861.

Lamiraux, F., Bonnafous, D., & Lefebvre, O. (2004). Reactive path deformation for nonholonomic mobile robots. IEEE Transactions on Robotics, 20(6), 967–977.

Langer, D., Rosenblatt, J. K., & Hebert, M. (1994). A behavior-based system for off-road navigation. IEEE Transactions on Robotics and Automation, 10(6), 776–783.

Lau, B., Sprunk, C., & Burgard, W. (2009). Kinodynamic motion planning for mobile robots using splines. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), St. Louis, MO.

LaValle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning. Report No. TR 98-11, Iowa State University, Ames, IA.

LaValle, S. M., & Kuffner, J. J. Jr., (2001). Randomized kinodynamic planning. International Journal of Robotics Research, 20(5), 378–400.

Levoy, M., Gerth, J., Curless, B., & Pull, K. (2015). The Stanford 3D scanning repository. http://graphics.stanford.edu/data/3Dscanrep.

Liu, M., & Siegwart, R. (2014). Navigation on point-cloud—a Riemannian metric approach. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China.

Maimone, M., Biesiadecki, J., Tunstel, E., Cheng, Y., & Leger, C. (2006). Surface navigation and mobility intelligence on the Mars Exploration Rovers. In A. Howard & E. W. Tunstel (Eds.), Intelligence for space robotics (pp. 45–69). San Antonio, TX: TSI Press.

Manduchi, R., Castano, A., Talukder, A., & Matthies, L. (2005). Obstacle Detection and Terrain classification for autonomous off-road navigation. Autonomous Robots, 18(1), 81–102.

Marshall, J., Barfoot, T. D., & Larsson, J. (2008). Autonomous underground tramming for center- articulated vehicles. Journal of Field Robotics, 25(6-7), 400–421.

McManus, C., Furgale, P., Stenning, B., & Barfoot, T. D. (2013). Lighting-invariant visual teach and repeat using appearance-based Lidar. Journal of Field Robotics, 30(2), 254–287.

Medioni, G., Lee, M.-S., & Tang, C.-K. (2000). A computational framework for segmentation and grouping. Amsterdam: Elsevier.

Menna, M., Gianni, M., Ferri, F., & Pirri, F. (2014). Real-time autonomous 3D navigation for tracked vehicles in rescue environments. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Chicago, IL.

Miró, J. V., Dumonteil, G., Beck, C., & Dissanayake, G. (2010). A Kyno-dynamic metric to plan stable paths over uneven terrain. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Taipei, Taiwan.

Moravec, H. P. (1980). Obstacle avoidance and navigation in the real world by a seeing robot rover (PhD dissertation). Stanford University, Stanford, CA.

Nagy, B., & Kelly, A. (2001). Trajectory generation for car-like robots using cubic curvature polynomials. In Proceedings of the International Conference on Field and Service Robotics (FSR), Helsinki, Finland.

Papadakis, P. (2013). Terrain traversability analysis methods for unmanned ground vehicles: A survey. Engineering Applications of Artificial Intelligence, 26(4), 1373–1385.

Park, C., Pan, J., & Manocha, D. (2012). ITOMP: Incremental trajectory optimization for real-time replanning in dynamic environments. In Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), Atibaia, S ao Paulo, Brazil.

Peynot, T., Lui, S.-T., McAllister, R., Fitch, R., & Sukkarieh, S. (2014). Learned stochastic mobility prediction for planning with control uncertainty on unstructured terrain. Journal of Field Robotics, 31(6), 969–995.

Pfaff, P., Triebel, R., & Burgard, W. (2007). An efficient extension to elevation maps for outdoor terrain mapping and loop closing. International Journal of Robotics Research, 26(2), 217–230.

Pivtoraiko, M., & Kelly, A. (2005). Efficient constrained path planning via search in state lattices. In Proceedings of the International Symposium on Artificial Intelligence, Robotics, and Automation in Space (iSAIRAS), Munich, Germany.

Pomerleau, F., Colas, F., Siegwart, R., & Magnenat, S. (2013). Comparing ICP variants on real-world data sets. Autonomous Robots, 34(3), 133–148.

Pomerleau, F., Krüsi, P., Colas, F., Furgale, P., & Siegwart, R. (2014). Long-term 3D map maintenance in dynamic environments. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, China.

Quinlan, S., & Khatib, O. (1993). Elastic bands: connecting path planning and control. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Atlanta, GA.

Rekleitis, I., Bedwani, J.-L., Dupuis, E., Lamarche, T., & Allard, P. (2013). Autonomous over-the-horizon navigation using LIDAR data. Autonomous Robots, 34(1-2), 1–18.

Rowe, N. C., & Ross, R. S. (1990). Optimal grid-free path planning across arbitrarily contoured terrain with anisotropic friction and gravity effects. IEEE Transactions on Robotics and Automation, 6(5), 540–553.

Rusu, R. B., Sundaresan, A., Morisset, B., Hauser, K., Agrawal, M., & Latombe, J.-C. (2009). Leaving flatland: Efficient real-time three-dimensional perception and motion planning. Journal of Field Robotics, 26(10), 841–862.

Santamaria-Navarro, A., Teniente, E. H., Morta, M., & Andrade-Cetto, J. (2014). Terrain classification in complex three-dimensional outdoor environments. Journal of Field Robotics, 31(1), 42–60.

Santana, P., Guedes, M., Correia, L., & Barata, J. (2011). Stereo-based all-terrain obstacle detection using visual saliency. Journal of Field Robotics, 28(2), 241–263.

Schulman, J., Duan, Y., Ho, J., Lee, A., Awwal, I., Bradlow, H., ... Abbeel, P. (2014). Motion planning with sequential convex optimization and convex collision checking. International Journal of Robotics Research, 33(9), 1251–1270.

Sermanet, P., Hadsell, R., Scoffier, M., Grimes, M., Ben, J., Erkan, A., ... LeCun, Y. (2009). A multirange architecture for collision-free off-road robot navigation. Journal of Field Robotics, 26(1), 52–87.

Shiller, Z., & Gwo, Y.-R. (1991). Dynamic motion planning of autonomous vehicles. IEEE Transactions on Robotics and Automation, 7(2), 241–249.

Shum, A., Morris, K., & Khajepour, A. (2015). Direction-dependent optimal path planning for autonomous vehicles. Robotics and Autonomous Systems, 70, 202–214.

Silver, D., Bagnell, J. A., & Stentz, A. (2010). Learning from demonstration for autonomous navigation in complex unstructured terrain. International Journal of Robotics Research, 29(12), 1565–1592.

Siméon, T. (1993). Motion planning for a non-holonomic mobile robot on 3-dimensional terrains. In C. Laugier (Ed.), Geometric reasoning for perception and action (Vol. 708, pp. 38–50). Lecture Notes in Computer Science. Berlin: Springer.

Siméon, T., & Dacre-Wright, B. (1993). A practical motion planner for all-terrain mobile robots. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Yokohama, Japan.

Singh, S., Simmons, R., Smith, T., Stentz, A., Verma, V., Yahja, A., & Schwehr, K. (2000). Recent progress in local and global traversability for planetary rovers. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), San Francisco, CA.

Sofman, B., Lin, E., Bagnell, J. A., Cole, J., Vandapel, N., & Stentz, A. (2006). Improving robot navigation through self-supervised online learning. Journal of Field Robotics, 23(11-12), 1059–1075.

Stenning, B. E., McManus, C., & Barfoot, T. D. (2013). Planning using a network of reusable paths : A physical embodiment of a rapidly exploring random tree. Journal of Field Robotics, 30(6), 916–950.

Stentz, A., & Hebert, M. (1995). A complete navigation system for goal acquisition in unknown environments. Autonomous Robots, 2(2), 127–145.

Stoyanov, T., Magnusson, M., Andreasson, H., & Lilienthal, A. J. (2010). Path planning in 3D environments using

the normal distributions transform. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Taipei, Taiwan.

Stumm, E., Breitenmoser, A., Pomerleau, F., Pradalier, C., & Siegwart, R. (2012). Tensor-voting-based navigation for robotic inspection of 3D surfaces using lidar point clouds. International Journal of Robotics Research, 31(12), 1465–1488.

Sun, Z., & Reif, J. H. (2005). On finding energy-minimizing paths on terrains. IEEE Transactions on Robotics, 21(1), 102–114.

Tâche, F., Fischer, W., Caprari, G., Siegwart, R., Moser, R., & Mondada, F. (2009). Magnebike: A magnetic wheeled robot with high mobility for inspecting complex-shaped structures. Journal of Field Robotics, 26(5), 453–476.

Teniente, E. H., & Andrade-Cetto, J. (2013). HRA*: Hybrid randomized path planning for complex 3D environments. In Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS), Tokyo, Japan.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., ... Mahoney, P. (2006). Stanley: The robot that won the DARPA Grand Challenge. Journal of Field Robotics, 23(9), 661–692.

Urmson, C., Ragusa, C., Ray, D., Anhalt, J., Bartz, D., Galatali, T., ... Struble, J. (2006). A robust Approach to High-Speed Navigation for Unrehearsed Desert Terrain. Journal of Field Robotics, 23(8), 467–508.

Wellington, C., Courville, A., & Stentz, A. (2005). Interacting Markov random fields for simultaneous terrain modeling and obstacle detection. In Proceedings of Robotics: Science and Systems (RSS), Cambridge, MA.

Woods, M., Shaw, A., Tidey, E., Van Pham, B., Lacroix, S., Mukherji, R., ... Chong, G. (2014). Seeker autonomous long-range rover navigation for remote exploration. Journal of Field Robotics, 31(6), 940–968.

Ye, C. (2003). A new terrain mapping method for mobile robots obstacle negotiation. In Proceedings of the UGV Technology Conference at the 2003 SPIE AeroSense Symposium, Orlando, FL.

Zucker, M., Ratliff, N., Dragan, A. D., Pivtoraiko, M., Klingensmith, M., Dellin, C. M., ... Srinivasa, S. S. (2013). CHOMP: Covariant Hamiltonian optimization for motion planning. International Journal of Robotics Research, 32(9-10), 1164–1193.