Optimization-Based Terrain Analysis and Path Planning in Unstructured Environments

Ueli Graf*† , Paulo Borges*, Emili Hernández*, Roland Siegwart†, and Renaud Dubé†
*Robotics and Autonomous Systems Group, CSIRO's Data61, Brisbane, Australia
Email: {Paulo.Borges, Emili.Hernandez}@data61.csiro.au
†Autonomous Systems Lab, ETH Zurich, Switzerland, Email: {grafue, rsiegwart, rdube}@ethz.ch

***Abstract*— Accurate environment representation is one of the key challenges in autonomous ground vehicle navigation in unstructured environments. We propose a real-time optimization-based approach to terrain modeling and path planning in off-road and rough environments. Our method uses an irregular, hierarchical, graph-like environment model. A space-dividing tree is used to define a compact data structure capturing vertex positions and establishing connectivity. The same unique underlying data structure is used for both terrain modeling and path planning without memory reallocation. Local plans are generated by graph search algorithms and are continuously regenerated for on-the-fly obstacle avoidance inside the scope of the local terrain map. We show that implementing a hierarchical model over a regular space division reduces graph edge expansions by up to 84 %. We illustrate the applicability of the method through experiments with an unmanned ground vehicle in both structured and unstructured environments.**

I. Introduction

Navigating off-road environments is a key challenge for unmanned ground vehicles (UGVs). While smooth, road-like environments can be traversed using only planar scanning for obstacle avoidance, this does not always apply to rough and unstructured terrains as variations in terrain inclination can be perceived as obstacles. One strategy to address this issue is to classify obstacles using 3D scanners and to capture the estimated traversability in a planar costmap [1]. Body vibrations introduced by non-planar terrain can distort environment scans if the ego-motion is not properly estimated. Using scan alignment for terrain assessment where precise sensor tracking is unavailable has been proposed but can be computationally expensive without strong observation filtering and requires map post-processing to deal with outliers and enable meaningful planning [2].

In this paper, we propose a hierarchical environment model, an approach for efficient model optimization, and a search-based local planner that enable navigation in rough terrain without any observation preprocessing. Our approach sets limits on the admissible terrain resolution to ensure that flat terrain observed under vibration or sensor noise is not modelled as a rough surface. The admissible resolution can be chosen according to the size and capabilities of the UGV, ensuring that only relevant information is captured in the terrain model. Compared to observation clustering-based approaches, this negates the need for any arbitrary observation filtering and iterative scan alignment procedures.

The different hierarchy levels in our terrain model correspond to terrain resolution levels and the recursive structure can be stored compactly as a tree. This enables dynamic resolution changes, simultaneously allowing for fine-grained obstacle resolution and sparse open space representation, greatly reducing the complexity of both model optimization and path planning as illustrated in Fig. 1. We present a graph search implementation that is suitable for generating local paths in real time during exploration. The proposed approach leverages a unique set of vertices to simultaneously express the terrain model at varying resolutions, to state the terrain optimization problem, and to solve the graph search planning problem, as illustrated in Fig. 2. The applicability to automated driving is demonstrated in a large industrial environment that includes overhanging structures as well as on- and off-road terrain. Successful on-the-fly obstacle avoidance in rough environments is presented. In summary,

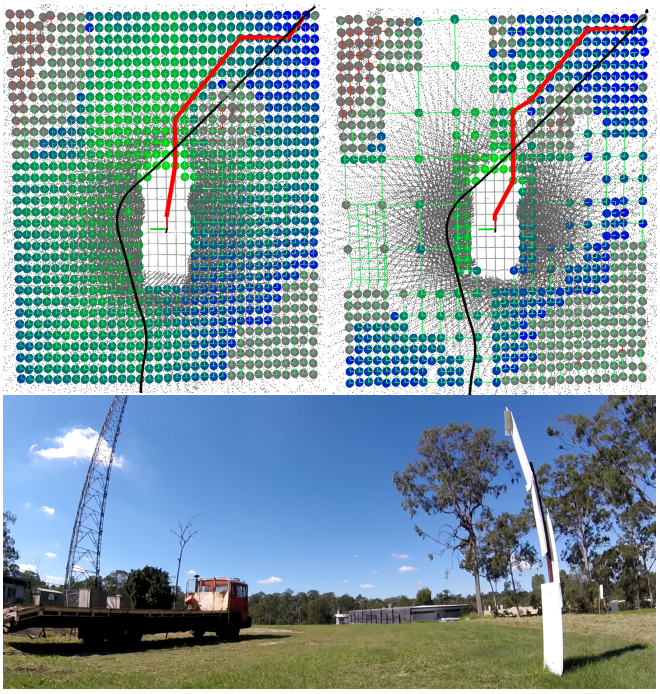

Fig. 1. Top-down view of our terrain analysis and path planning approach in an off-road environment (top) and corresponding first person view (bottom). The hierarchical grid can be used both at full resolution and with an adaptive resolution (top left and top right, respectively). Mesh vertex colour from green to blue indicates cost to arrive (low to high), whereas gray indicates infinite or non-explored cost. The black path indicates the global plan, whose intersection with the local map in the top right corner is the only reference point the local planner is aware of. The red path indicates the local plan found in the graph. Both the white pole and the large vehicle are detected as obstacle regions as indicated by gray vertices.

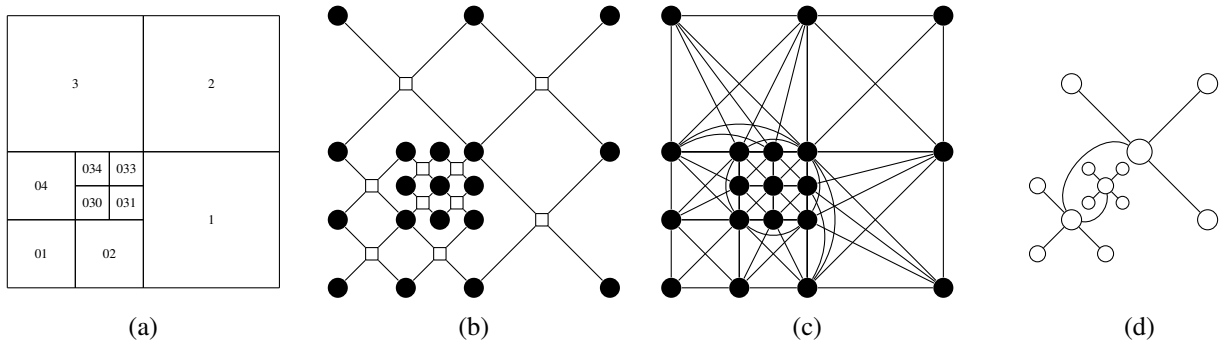

Fig. 2. Exemplary terrain model expressed as a space-dividing cell map (a), optimization graph with vertices (dots) and associated observations (squares) (b), and path planning graph with vertices and all admissible paths (c). The underlying data structure of (a)-(c) is the tree illustrated in (d), whose nodes (circles) contain all vertices and whose structure uniquely determines all edges in the other representations.

this paper presents the following contributions:

- A novel graph-based, multi-resolution terrain model suited for real-time optimization.
- A unified and reusable underlying data structure for terrain modelling and path planning.
- Demonstration of the applicability of our approach through experiments in both structured and unstructured environments.

The remainder of this paper is structured as follows: We give an overview of related research in Sec. II. Sec. III lays out the methods developed in this work while experimental results are introduced in Sec. IV. Conclusions and future work opportunities are presented in Sec. V.

II. Related work

Garrido et al. [3] use a triangular mesh as basis of a map representation. Additional information (height, spherical variance and surface gradient) is captured in a weighting matrix which is subsequently used for planning using the fast marching method. This requires specific extraction of additional features from every environment cell while our approach is capable of using the terrain modelling output directly for planning. Breitenmoser and Siegwart [4] show that terrain extraction using either implicit or explicit observation meshing by triangulation requires filtering and mesh post-processing even when driving on smooth surfaces, a complication that would only be amplified when meshing rough terrain under heavy sensor vibration while driving.

Kümmerle et al. [1] demonstrate a local planner capable of navigating multi-story parking garages. It finds obstacles in the observations and then considers traversability in 2D only which disregards all information about inclination and terrain roughness below some obstacle threshold. In contrast to this, our planner is capable of minimizing not only for minimal distance, but also for minimal change in robot body orientation which can be advantageous especially when navigating non-planar and rough environments.

Krüsi et al. [5] identify traversable points in a similar fashion and implement a sampling-based planner. The distinction between initial plan generation and subsequent optimization is made. The method is applicable to any terrain but it can fail when dynamic obstacle avoidance is required and the local optimizer demands a new initial plan. In contrast to this, the local plans computed by our method are always recomputed in their entirety and planning times are consistently fast at the cost of universal applicability. Our algorithm performs collision checks against all environment cells that are occupied by part of the robot body at any stage when moving between discrete locations and also against all observations in real-time. The true robot body size is taken into account in these checks and no templates for performing collision checks are required.

III. Methods

Our local path planning pipeline for unstructured environments, depicted in Fig. 3, consists of three main modules: the *terrain model*, the *optimizer*, and the *local path planner*. The terrain model receives as an input environment observations and populates an underlying data structure. The optimizer modifies the parameters of the model to find a best fit given the observations. The local path planner attempts to find a path on the optimized terrain model (specifically, a graph representation of the underlying tree) towards a reference path or waypoint given by a global planner.

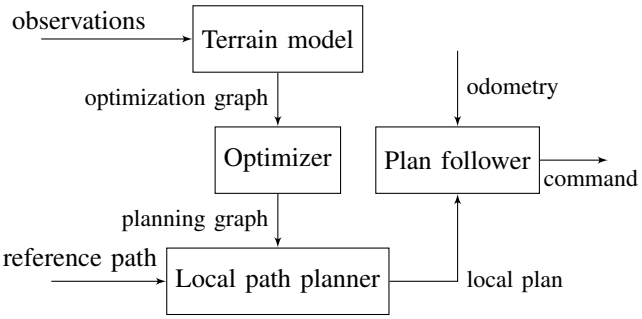

Fig. 3. System diagram of the proposed optimization-based terrain modelling and path planning pipeline.

A. Terrain model

In the construction of our terrain model, we assume that traversable terrain can be captured by a mesh. We attempt to find the best positions of all vertices in this mesh, such that the surface of all cells outlined by these vertices best fits some set of observations. An intuitive special case of such

a mesh would be an initial simple regular grid of squares where each node in the grid can move in space to adapt to observations. Regular grids, however, can be inefficient as there is no mechanism to capture an area composed of many cells as flat space. This is not optimal for optimization and planning since a lot of computation could be saved by capturing flat areas at a lower resolution. While our model can be thought of as a mesh covering the environment, our implementation represents it as a graph that uses an underlying space dividing tree for compact and efficient storage of vertex positions and connectivity information.

1) Model Formulation: We represent the environment by a set of cells c_i, each element of which is uniquely described by the *set of vertices* V_i that describe its outline polygon. The members of V_i do not necessarily lie on a common plane. The position of the vertices are the model parameters to be optimized. The cells are arranged such that they form a tessellation when projected onto a horizontal plane. The terrain model is constructed by recursive expansion of a space-dividing tree where tree nodes correspond to cells c_i in the environment and also hold their respective sets of outline vertices V_i. The tree root corresponds to the scope of the local map. During model construction, tree leafs are split recursively into b (*branching factor*) children until the tree depth is equal to a *maximum tree depth* l that depends on available computing power and desired resolution. During this process, leafs are subdivided by virtue of cutting planes. New vertices are generated at intersections between cuts or between cuts and existing cell borders.

While this process is similar in concept to building a space-dividing k-d-tree [6], multiple cuts per tree level are allowed and all nodes of the resulting tree hold references to the vertices of their outline polygons. Tree nodes might refer to the same vertex even across tree levels as every new cell captured as a new tree leaf is bound by both members of the set of vertices of its parent tree node and new vertices. An illustration of how new cells are constructed from their parent and how vertices are related is shown in Fig. 4. Vertices that are shared between adjacent cells or between tree levels correspond to the same allocated memory. After expanding to l tree levels, the number of cells at the highest resolution is given by b^{l-1}, while the total number of cells across all levels is $\sum_{l'=1}^{l} b^{l'-1}$.

Since tree nodes remain valid environment cells after construction of their children, this data structure enables dynamic resolution changes by introducing the set of *active cells* C. The set C contains all cells that are considered during terrain optimization and its members are not required to be taken from a single level of the space dividing tree. Instead, any node in the tree and its associated cell is *active* if its parent and all its children are *inactive*. This requirement implies that $1 \leq |C| \leq b^{l-1}$, meaning that the smallest possible set of active cells contains only the root node of the tree while the largest possible set contains exactly all the existing tree leaves.

2) Observation association: In this context, an *observation* refers to a single point in the local environment perceived by a terrain scanning device, represented as a position vector $y \in \mathbb{R}^3$. While traversing terrains, the terrain model is periodically updated using sets of such observations denoted by Y. When Y is combined with the terrain model, every observation $y_k \in Y$ is associated with exactly one cell in the map by projecting all y_k and all map vertices into a horizontal plane and finding the containing cell for each observation. We introduce the set of *associated* observations Y_i which is given by all $y_k \in Y$ that lie in the interior of the terrain cell c_i after projection. This can be done efficiently since the terrain model is captured as a space-dividing tree.

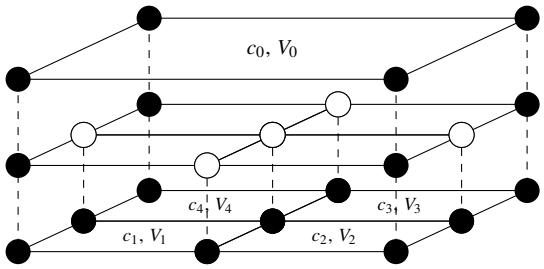

Fig. 4. Initial cell subdivision example. A parent cell c_0 outlined by its set of vertices V_0 is split into four children cells c_1-c_4 with sets of vertices V_1-V_4 by adding vertices at the side midpoints and in the center. Dashed lines indicate coinciding vertices. The four vertices of the parent cell V_0 are shared across both resolution levels. In this example, nine vertices are used to define five cells at two levels of resolution.

B. Optimizer

In order to optimize the terrain model parameters with respect to a set of observations Y, a cost is evaluated per cell c_i and its set of associated observations Y_i and the total cost incurred by the map is defined as the *root mean square* (RMS) value over all costs. In the following, for brevity, we demonstrate the cost evaluation for an individual cell only. The cost R for cell c_i is

$$R = \sqrt{\frac{1}{|Y_i|} \sum_{y_k \in Y_i} \left(n^T \Omega_y (y_k - \bar{v})(y_k - \bar{v})^T \Omega_y^T n \right)} \qquad (1)$$

where n is a plane normal vector estimate obtained from the vertex positions V_i, $\bar{v} = \frac{1}{|V_i|} \sum_{v' \in V_i} v'$ is the mean position of all vertices outlining the polygon and Ω_y is a weighting matrix. When evaluating (1), the summation over the outer product $(y_k - \bar{v}) \otimes (y_k - \bar{v})$ can be expensive as $|Y_i|$ can be large. In a simplistic solution, the cost function must be evaluated once for every cell after every minimization step and twice for every degree of freedom of every vertex of every cell when evaluating cost derivatives numerically. This behaviour is undesirable, especially when the set of associated observations Y_i does not change over many minimization steps. The evaluation is simplified by introducing estimates for the sample mean $\hat{\mu}_y \in \mathbb{R}^{3\times 1}$ and the covariance matrix $\hat{\Sigma}_y \in \mathbb{R}^{3\times 3}$ as

$$\hat{\mu}_y = \frac{1}{|Y_i|} \sum_{y_k \in Y_i} y_k, \quad \hat{\Sigma}_y = \frac{1}{|Y_i|} \sum_{y_k \in Y_i} (y_k - \hat{\mu}_y)(y_k - \hat{\mu}_y)^T. \quad (2)$$

Using this result, the outer product can be expressed as

$$\begin{aligned}\sum_{y_k\in Y_i}&(y_k-\bar{v})(y_k-\bar{v})^T\\ &=\sum_{y_k\in Y_i}\left(y_k y_k^T\right)+|Y_i|\left(\bar{v}\bar{v}^T-\bar{y}\bar{v}^T-\bar{v}\bar{y}^T\right)\\ &=|Y_i|\cdot(\hat{\Sigma}_y+\hat{\mu}_y\hat{\mu}_y^T)+|Y_i|\cdot(\bar{v}\bar{v}^T-\hat{\mu}_y\bar{v}^T-\bar{v}\hat{\mu}_y^T).\end{aligned}\tag{3}$$

Result (3) allows for rewriting the cost function without using individual point-to-plane distances. We write

$$R=\sqrt{n^T\Omega_y\left(\hat{\Sigma}_y+\hat{\mu}_y\hat{\mu}_y^T+\bar{v}\bar{v}^T-\hat{\mu}_y\bar{v}^T-\bar{v}\hat{\mu}_y^T\right)\Omega_y^T n}\tag{4}$$

which corresponds to optimizing with respect to an observation distribution estimate. This completely separates the summations from variables that are influenced by the optimization, which are the mean vertex position $\bar{v}$ and the plane normal vector n. It is important to note that introducing the estimates in (2) does not constitute an approximation but an exact result. In our approach, the summations over a total of $|Y_i|$ terms need only be evaluated once during optimization initialization and are cached for successive cost and cost derivative evaluations.

The mean vertex position $\bar{v}$ and the plane normal vector n are dependent on the positions of the members of V_i, which are subject to optimization. They must therefore be updated after every cost function minimization step. In order to estimate n, we define an over-parametrized plane model as $ax+by+cz-d=0$ where $a,b,c,d\in\mathbb{R}$. Since we are only estimating the plane normal direction, we can choose an arbitrary value for d. for example $d=1$. This describes a plane with normal vector $n=\frac{p}{\sqrt{p^Tp}}$ where $p=\begin{pmatrix}a & b & c\end{pmatrix}^T$ is tangent to an origin-centered sphere at $r_0=\frac{1}{\sqrt{p^Tp}}n=\frac{p}{p^Tp}$. The distance $e_j\in\mathbb{R}$ of a vertex position $v_j\in V_i$ from the plane with normal n and member r_0 is found by projection to the plane normal vector according to

$$e_j=(v_j-r_0)^Tn=\frac{v_j^Tp-1}{\sqrt{p^Tp}}.$$

Our approach assumes that shortest distances e_j of vertices from the idealized plane with normal vector n are distributed normally with zero mean and finite variance $\sigma^2\in\mathbb{R}$ and thus introduces a new random variable

$$f_j=e_j\sqrt{p^Tp}=v_j^Tp-1\sim\mathcal{N}(0,\sigma^2\cdot p^Tp)\in\mathbb{R}.$$

Since f_j is zero-mean and its variance is linear in p^Tp, it is used to obtain the maximum likelihood estimate $\hat{p}^{\mathrm{ML}}$ for the value of p. We define the vector of model parameters $\beta=p$ and cast to a least-squares estimation problem by introducing $X=\begin{pmatrix}v_1 & v_2 & \dots & v_{|V_i|}\end{pmatrix}^T\in\mathbb{R}^{|V_i|\times 3}$ and $z=\begin{pmatrix}1 & 1 & \dots & 1\end{pmatrix}^T\in\mathbb{R}^{|V_i|\times 1}$. The maximum likelihood estimator $\hat{\beta}^{ML}$ in the presence of zero-mean Gaussian disturbances f_j is exactly the argument that minimizes the sum of squared residuals

$$\begin{aligned}\hat{\beta}^{ML}&=\arg\min_\beta\sum_{v_j\in V_i}f_j(v_j)^2=\arg\min_\beta\sum_{v_j\in V_i}\left(v_j^Tp-1\right)^2\\ &=\arg\min_\beta\left((X\beta-z)^T(X\beta-z)\right)=(X^TX)^{-1}X^Tz\\ &=\left(\sum_{v'\in V_i}v'v'^T\right)^{-1}\left(\sum_{v'\in V_i}v'\right).\end{aligned}$$

This has a unique solution, since vertices are never collinear by construction and thus X^TX is always invertible. The maximum likelihood plane normal vector estimator is then given by

$$\hat{n}=\frac{\hat{\beta}^{\mathrm{ML}}}{||\hat{\beta}^{\mathrm{ML}}||}.\tag{5}$$

C. Local path planner

Graph search algorithms are used to plan a path on the underlying representation of the local terrain model. The planning graph is implied directly by the optimization result: Every active vertex, that is, any vertex $v_m\in\bigcup_{i|c_i\in C}V_i$, corresponds to a vertex in the planning graph and shares an edge with another vertex $v_n\in\{\bigcup_{i|c_i\in C}V_i\}\setminus v_m$ if there is a cell $c_i\in C$ of which they are both members, that is, $\exists c_k\in C\mid v_m\in V_k\wedge v_n\in V_k$.

1) Planning cost: The robot heading at each node is defined according to the position of its predecessor or - if available - its successor in the current plan. An orientation matrix for a given 3D position t and heading h is estimated by finding the set of all cells that contain a robot ground contact point $r_{g,j}(t,h)$, denoted by C_{gc}. From this set of cells, a local ground normal direction is estimated by using (5) over all vertices outlining the cells in C_{gc}. The robot heading is orthogonalized with respect to the estimated normal vector $\hat{n}$ using a Gram-Schmidt process according to

$$\hat{h}_\perp=\frac{h-\hat{n}h^T\hat{n}}{||h-\hat{n}h^T\hat{n}||}.$$

The full estimated robot orientation matrix is thus given by

$$\hat{T}(t,h):=\begin{pmatrix}\hat{h}_\perp & \hat{n}\times\hat{h}_\perp & \hat{n}\end{pmatrix}\in\mathrm{SO}(3).$$

The cost-to-go from position t_i, direction d_i to position t_j, direction d_j is a function of both the Frobenius norm of the change in orientation $||\hat{T}(t_j,h_j)-\hat{T}(t_i,h_i)||_{\mathrm{F}}$ and the Euclidean distance $||t_j-t_i||_2$. The L^1 norm of the relative positions of target t and initial state s given by $||t_t-t_s||_1$ is used as a normalizing constant for the euclidean distance. The maximum value of the relative orientation of $\hat{T}_i=\hat{T}(t_i,h_i)$, $\hat{T}_j=\hat{T}(t_j,h_j)$ can be found in closed form as

$$\max_{\hat{T}_i,\hat{T}_i\in\mathrm{SO}(3)}||\hat{T}_j-\hat{T}_i||_{\mathrm{F}}=\max_{\hat{T}_i,\hat{T}_i\in\mathrm{SO}(3)}||\mathbf{I}_3-\hat{T}_j^T\hat{T}_i||_{\mathrm{F}}=2\sqrt{3}.$$

The full cost function is thus defined as

$$f(t_i,h_i,t_j,h_j):=\frac{||t_j-t_i||_2}{||t_t-t_s||_1}+\frac{||\hat{T}_j-\hat{T}_i||_{\mathrm{F}}}{2\sqrt{3}}.$$

Since both addends in f are linear in norms, subadditivity is satisfied by definition and f is admissible for defining a consistent heuristic as $h(t_i,h_i):=f(t_i,h_i,t_t,h_t)$.

Using f, Dijkstra's algorithm (Dijkstra [7]) can be used to find an optimal path from s to t. Inclusion of h enables the use of A* graph search (Hart et al. [8]), which has proven to significantly speed-up convergence in our implementation, as shown in Fig. 5.

2) Obstacle detection: After optimizing the terrain with respect to a set of observations Y, a binary classifier performs an initial traversability assessment. The classifier works on the distribution of the observations and the positions of the vertices of the cell they are associated with, and serves the purpose of reducing the planning search space. A support vector machine with a radial basis function kernel was trained on data recorded with a UGV.

The estimated first, second and fourth moments of the distribution of shortest distance between individual observations and the idealized cell surface were chosen as classification features since they can be expressed in terms of estimates that were cached already for efficient optimization. The third moment was not used since it did not significantly improve classifier performance compared to the fourth moment. Deviating from their respective definitions, we estimate the second and fourth moments about zero instead of the sample mean and normalize all moments with powers of the variance about the sample mean. Our approach assumes a normal distribution and estimates the excess kurtosis with respect to $3\tilde{\mu}_{e,2}^2$ where $\tilde{\mu}_{e,2}$ is the normalized second moment. This approach allows us to reuse results derived in (3) and can be conveniently expressed as

$$\tilde{\mu}_{e,1} = \frac{\frac{1}{N}\sum_{k=1}^{N} e_k}{\sqrt{n^T \hat{\Sigma}_y n}} = \frac{n^T(\hat{\mu}_y - \bar{v})}{\sqrt{n^T \hat{\Sigma}_y n}}$$

$$\tilde{\mu}_{e,2} = \frac{\frac{1}{N}\sum_{k=1}^{N} e_k^2}{n^T \hat{\Sigma}_y n} = \frac{n^T(\hat{\Sigma}_y + \hat{\mu}_y\hat{\mu}_y^T + \bar{v}\bar{v}^T - \hat{\mu}_y\bar{v}^T - \bar{v}\hat{\mu}_y^T)n}{n^T \hat{\Sigma}_y n}$$

$$\tilde{\mu}_{e,4} = \frac{\frac{1}{N}\sum_{k=1}^{N} e_k^4 - 3\left(\frac{1}{N}\sum_{k=1}^{N} e_k^2\right)^2}{(n^T \hat{\Sigma}_y n)^2} = \frac{\frac{1}{N}\sum_{k=1}^{N} e_k^4}{(n^T \hat{\Sigma}_y n)^2} - 3\tilde{\mu}_{e,2}^2.$$

This yields a set of dimensionless (and thus scale-invariant) features. Finally, we collapse all features into the interval $[0,1]$ by defining our feature vector as

$$r = \begin{pmatrix} 1 - e^{-|\tilde{\mu}_{e,1}|} & 1 - e^{-|\tilde{\mu}_{e,2}|} & 1 - e^{-|\tilde{\mu}_{e,4}|} \end{pmatrix}.$$

Our experiments showed that including the normalized excess kurtosis as a feature improves obstacle detection accuracy, measured as ratio of actual obstacles that are correctly classified as obstacles, from 61 % to 95 % while reducing the number of support vectors required by a factor of three. Similarly, the ratio of classified obstacles that are actual obstacles improves from 51 % to 81 %.

IV. Experiments

In this section we present experimental evaluation of our proposed approach and compare it with existing methods. We also discuss optimization performance considerations and the effects of adaptive resolution and heuristic-based graph search algorithms and show a sample output of our local planning pipeline.

A. Practical implementation details

We have evaluated our approach on a John Deere Gator TE that was automated at CSIRO. The vehicle is equipped with a Velodyne PUCK VLP-16 LiDAR mounted approximately 1 m above the vehicle. The base of the sensor is tilted by 45 deg from the ground plane and continuously revolves around the vehicle vertical axis in order to maximize the field of view. All evaluations were performed using a computer with an Intel® Core™ i7-6500U CPU and 16 GiB DDR4-2133 main memory. The operating system was Ubuntu 16.04. Our implementation relies on Eigen [9] for linear algebra and g2o [10] for graph optimization and planning.

B. Point cloud planner

The proposed approach was compared against the *Point Cloud Planner* (PCP) presented in Krüsi et al. [5] since it attempts to solve a similar problem (navigating rough environments with or without prior knowledge of the map) with a similar sensor setup (focus on 3D LiDAR). While our proposed approach relies on a locally consistent global positioning system (that may or may not maintain its own map) and completely rebuilds the local environments, the PCP plans paths directly on a full global 3D map, if available, and up to the observation horizon. To ensure that the planning range for both planners was similar, both used the same set of observations and target inside the map range of our proposed planner. The PCP did not have a prior map of the environment.

Since the PCP relies on the output (and the observation filtering) of a SLAM system based on iterative closest point scan alignment, the time spent filtering observations and updating and post-processing the map were not included for the PCP in this comparison, whereas for our proposed approach, map updates (optimization) and path planning were also considered. Observation association times were in the order of 1 ms and were subsequently neglected. The timing results are shown in Tab. I. To rule out any effects of the SLAM system on the PCP results, these times were obtained by commissioning the planner with finding a path with the vehicle in a static position in a static environment. PCP first computes an initial path (*PCP–initial*) and then optimizes it (*PCP–static*) whenever a new environment scan is registered and the map is updated. Our approach does not make that distinction and finding a path in a static environment takes the same amount of time (*Proposed–static*), no matter whether a previous solution is available. We also show the execution time of our proposed approach when traversing the same environment instead of scanning it statically (*Proposed–dynamic*). In that case, execution times are slightly longer since body vibrations and other sources of noise require a higher map resolution.

The results show that the proposed constrained terrain model is capable of consistently higher path update rates, enabling on-the-fly path re-planning and obstacle avoidance when used on a UGV.

TABLE I

TIMING OF PCP AND THE PROPOSED APPROACH

Algorithm	Execution times (ms)			Total
	Optimization	Obstacles	Planning	
PCP–initial	-	-	1504	1504
PCP–static	-	-	430	430
Proposed–static	110	17	17	144
Proposed–dynamic	141	39	19	199

C. Optimization performance

We compared the optimization performance of the proposed constrained optimization clustering approach introduced in Sec. III-B. Execution times of our optimizer were compared against optimizing the terrain with respect to all individual observations and against computing the unconstrained least squares solution, which corresponds to optimizing without any constraints. In the experiment, 1024 terrain observations were generated by sampling an underlying plane model and disturbing the result with noise in the true plane normal direction. The observations are then associated to one environment cell with four vertices with random initial positions. The timing results are shown in Tab. II.

TABLE II

TIMINGS OF DIFFERENT TERRAIN MODEL OPTIMIZATION TECHNIQUES

Algorithm	Execution time (ms)	Improvement (%)
Single observation	28.5	0
Clustered observation	1.38	95.1
Least squares	0.106	99.6

D. Planner sample output

The performance of the proposed graph search planner was evaluated in an environment with obstacles as shown in Fig. 1. Visualized in red are paths found by the A* graph search algorithm on the full resolution and the adaptive resolution meshes, the latter exploits the tree hierarchy to dynamically resolve areas around obstacles. The current goal is the intersection of the global plan (black) with the local map border at the top right. The local planner shows desirable behavior compared to the global planner by successfully cutting corners when allowed by the perceived terrain but choosing a longer path to evade an obstacle on the global reference trajectory. In our experiments, the UGV was able to navigate a distance of over 600 m without intervention.

E. Graph search evaluation

The numbers of expanded nodes when traversing this environment were recorded for comparison. The mesh optimizer was run in full resolution and adaptive resolution mode on the same inputs, with paths being generated using both Dijkstra's and the A* algorithm. The results are shown in Fig. 5. Three situations should be distinguished when moving towards a target:

1) The target is outside the map and the intersection of the boundary with the global plan is a feasible position.

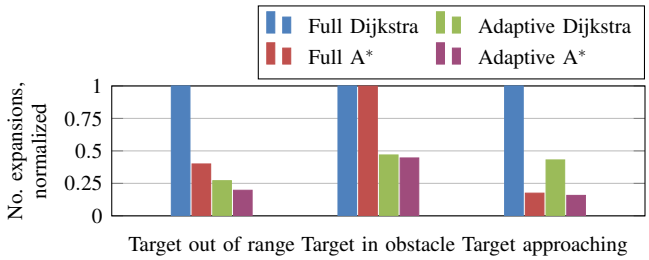

Fig. 5. Comparison of the number of edge expansions when searching for an optimal path to a given target, normalized with the number obtained for using Dijkstra's algorithm on the full resolution map. The other algorithms are using either adaptive resolution, A* search or both.

2) The intersection point or the end of the global plan are in the local map but the containing cell represents an obstacle, the current goal is thus unreachable.
3) The global target is inside the local map and reachable.

The first is the generic case. In our experiments 80.39 % less edges were expanded in the graph search when using A* on an adaptive resolution mesh instead of running Dijkstra's algorithm on the full resolution mesh.

In the second case, the advantage of A* is less pronounced since early termination is not possible when the global target is never achieved. Instead, the complete mesh must be searched for the best possible alternative path in a heuristic sense. The advantage of adaptive resolution is still obvious, with 55.44 % less explored edges on average.

The third case with the target approaching demonstrates the advantage of using a cost heuristic. The main speed-up is achieved by finding a feasible path to the target very quickly. Once a cost has been assigned to the target, most nodes can be rejected based on their cost heuristic. Overall, 84.27 % average less number of edges were explored, while even the full-resolution A* search saves 82.54 %.

V. CONCLUSION

In this work we proposed a novel optimization-based approach to local path planning for UGVs navigating in both structured and rough environments. We developed a hierarchical, graph-based, multi-resolution terrain model and demonstrated the use of a single underlying data structure for both terrain modelling and path planning. We trained a distribution-based binary classifier to reduce the planning search space and included a cost heuristic to speed up convergence. We demonstrated the applicability of our approach in real-world experiments involving on- as well as off-road environments. We showed how clustered optimization and dynamic resolution contributes to lower processing (and, thus, reaction) times. In our experiments, out proposed local planner successfully takes shortcuts with respect to a global reference path when it is safe to do so while obstacles on the reference path trigger an evasive maneuver.

Future work could focus on a fully probabilistic optimization, incorporating a 3D sensor noise model and covariance estimation per environment cell. Additionally, an iterative resolution-refining planner exploiting the full tree model hierarchy could improve planning performance.

References

[1] R. Kümmerle, D. Hähnel, D. Dolgov, S. Thrun, and W. Burgard, "Autonomous driving in a multi-level parking structure," in *Proc. - IEEE Int. Conf. Robot. Autom.*, 2009, pp. 3395–3400.

[2] P. Krüsi, B. Bücheler, F. Pomerleau, U. Schwesinger, R. Siegwart, and P. Furgale, "Lighting-invariant adaptive route following using iterative closest point matching," *J. F. Robot.*, 2015.

[3] S. Garrido, M. Malfaz, and D. Blanco, "Application of the fast marching method for outdoor motion planning in robotics," *Rob. Auton. Syst.*, vol. 61, no. 2, pp. 106–114, 2013.

[4] A. Breitenmoser and R. Siegwart, "Surface reconstruction and path planning for industrial inspection with a climbing robot," *2012 2nd Int. Conf. Appl. Robot. Power Ind.*, pp. 22–27, 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6473354

[5] P. Krüsi, P. Furgale, M. Bosse, and R. Siegwart, "Driving on Point Clouds: Motion Planning, Trajectory Optimization, and Terrain Assessment in Generic Nonplanar Environments," *J. F. Robot.*, vol. 34, no. 5, pp. 940–984, 2017. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21700

[6] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, 1975.

[7] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numer. Math.*, 1959.

[8] P. E. Hart, N. J. Nilsson, and B. Raphael, "Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths"," *ACM SIGART Bull.*, 1972.

[9] G. Guennebaud and B. Jacob, "Eigen v3," http://Eigen.Tuxfamily.Org, 2010.

[10] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *Proc. - IEEE Int. Conf. Robot. Autom.*, 2011.