# Leaving Flatland: Toward Real-Time 3D Navigation

Benoit Morisset*, Radu Bogdan Rusu†, Aravind Sundaresan*,
Kris Hauser‡, Motilal Agrawal*, Jean-Claude Latombe‡ and Michael Beetz†
* Artificial Intelligence Center, SRI International, Menlo Park, CA 94025 USA
† Intelligent Autonomous Systems, Technische Universität München, München, Germany
‡ Computer Science Department, Stanford University, Stanford, CA 94305, USA

***Abstract*— We report our first experiences with Leaving Flatland, an exploratory project that studies the key challenges of closing the loop between autonomous perception and action on challenging terrain. We propose a comprehensive system for localization, mapping, and planning for the RHex mobile robot in fully 3D indoor and outdoor environments. This system integrates Visual Odometry-based localization with new techniques in real-time 3D mapping from stereo data. The motion planner uses a new decomposition approach to adapt existing 2D planning techniques to operate in 3D terrain. We test the map-building and motion-planning subsystems on real and synthetic data, and show that they have favorable computational performance for use in high-speed autonomous navigation.**

## I. Introduction

A major area of research for unmanned ground vehicles (UGVs) will be the development of perception, planning, and control systems that will enable UGVs to autonomously navigate in the same environments and at the same speeds as humans. These capabilities are necessary for deploying UGVs to assist humans in complex tasks such as rescue, scouting, or transportation. The goal of our "Leaving Flatland" project is to study the key challenges of closing the loop between autonomous perception and action in complex environments.

Typical state-of-the-art mobile robot systems use 2D discrete representations of the world, such as *occupancy maps* [1]–[3], to plan the robot's motion. Occupancy maps define obstacles in a crude binary sense that overly simplifies the complex interactions between the robot and the environment (see Fig. 1). For example, obstacles can be context-specific, such as a ledge that acts as an obstacle when approached from a lower level, but acts as the ground while the robot is on it. These shortcomings can be addressed by *elevation maps* [4], which work well for pre-surveyed natural terrain. However, both these representations are inherently 2D. 2D elevation maps have been combined with a 3D occupancy grid to allow a humanoid robot to crawl under low ceilings [5]. This approach still does not address environments with multiple overlapping levels, such as tunnels or multi-story buildings. Other researchers have addressed 3D mapping and reconstruction for mobile robots [6]–[9] using laser range finders and stereo cameras. However, these approaches have not been demonstrated to work in real-time or in a fully autonomous robot system.

We have applied our mapping and navigation algorithms

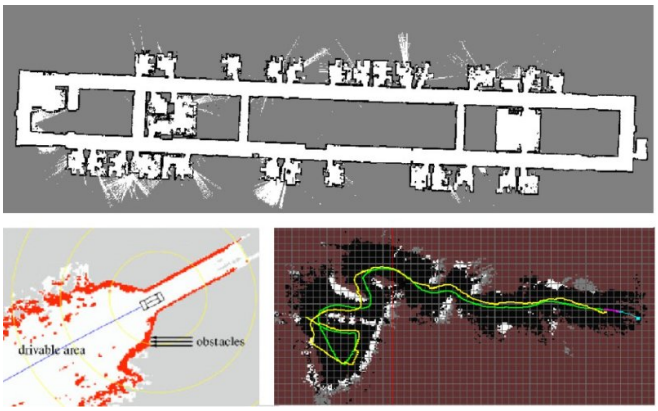



Fig. 1. Classical occupancy maps in mobile robotics. Top: Centibots [1], left: Grand Challenge [2], right: Learning Applied to Ground Robots [3]

to the RHex mobile robot [10]. RHex is a highly capable six-legged walking robot developed by Boston Dynamics, which can run at high speed on flat ground, crawl over rocky terrain, and climb stairs. We use stereo cameras to build a 3D model of the environment that is updated as RHex moves. Periodic updates are sent to a motion planner, which generates 3D trajectories to be executed by the robot. These trajectories specify a sequence of primitive gait commands to be sent to the robot's on-board controller. The planner is typically asked to reach a nearby goal within the robot's visual range. This goal is usually chosen by either a human operator or some supervisory behavior. Long-range goals are achieved by frequent replanning. To help the planner select the most appropriate gait for a given terrain type, we notate the 3D model elements with semantic labels. The motion planner then decomposes the terrain into locally 2D regions, and uses the semantic labels to select fast 2D planning techniques within each region.

We have tested the core system components in outdoor and indoor environments. Experiments show that the 3D mapping module can be operated in real-time, averaging about 250 ms for each map update. The motion planner is queried approximately once or twice a second depending on the complexity of the terrain. We hope to integrate these components into a complete system in the near future. As a preliminary step toward this integration, we present an extension of our system to a smart teleoperation application.

## II. System Overview

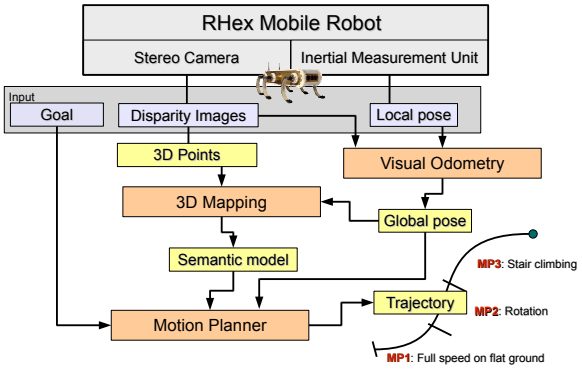



Fig. 2. The general architecture of Leaving Flatland.

Figure 2 presents the overall architecture of our system. The two sensing devices (a stereo camera and an Inertial Measurement Unit) are attached to RHex. The robot localization and point cloud registration are based on Visual Odometry [11], [12] which takes angle estimates from the IMU besides the intensity and disparity images. The 3D Mapping module uses the robot pose estimate from the VO module to register the point cloud data from each frame to the global reference frame and build a polygonal map annotated with semantic labels. The labeled polygonal map and a user-designated goal constitute the inputs for the Motion Planner module. Our motion planner uses a new decomposition technique, which decomposes the 3D workspace into locally 2D regions. This enables the use of established techniques for mobile robot planning on elevation maps. It also enables the use of fast planning techniques on flat ground and reserves more complex techniques for uneven terrain. The output of the motion planner is a sequence of gaits to be sent to the robot. We use gaits that have been designed specifically for RHex, including high-speed running on flat and uneven terrain [10] and stair climbing gaits [13].

## III. 3D Mapping

The 3D mapping module builds a polygonal model of the world using disparity images obtained from a stereo camera attached to the robot. Figure 3 presents the complete 3D mapping pipeline. A disparity image is computed from the stereo images and is used to produce point cloud data (PCD) in the camera coordinate frame. A Visual Odometer is used to estimate the camera motion between consecutive frames (Section III-A). The camera pose with respect to a fixed global coordinate system is then used to register the PCD from each frame to the global coordinate system. A Polygonal Modeling step is applied in which planar patches are used to approximate the underlying surface model. Each local model is merged with the global model in the Polygonal Growing step and the global model is refined (Section III-A). Finally, the polygonal patches in the model are annotated with semantic labels.

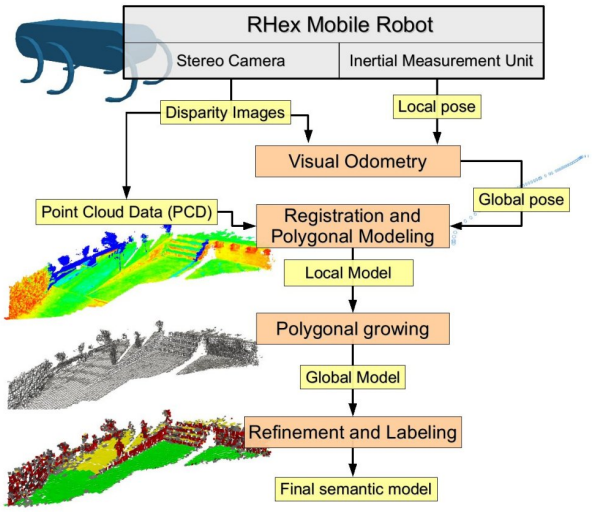



Fig. 3. The 3D mapping system pipeline.

### A. Visual Odometry and Point Cloud Registration

The accuracy of the global map depends directly on the precision with which we evaluate the position of each separate PCD. This process is called *registration* and its accuracy is determinant for the quality of the final model. Instead of employing a classical registration framework (e.g., Iterative Closest Point), our solution employs a Visual Odometer (VO).

Our VO system derives from recent research by the authors and others on high-precision VO [14]. Distinctive features are extracted from each new frame in both the left and the right image. The features in the left image are matched to those in the right image and also to the features extracted in the previous frame. From these uncertain matches, we recover a consensus pose estimate using a RANSAC method [15]. Several thousand relative pose hypotheses are generated by randomly selecting three matched non-colinear features, and then scored using pixel reprojection errors. If the motion estimate is small and the percentage of inliers is large enough, we discard the frame, since composing such small motions increases error. This pose estimate is then refined further in a sparse bundle adjustment (SBA) framework [16]. When we have IMU data available, we further fuse the VO results with this IMU data.

Several type of features can be employed. An important consideration is using features that are stable across viewpoint changes. While SIFT [17] and SURF [18] are the features of choice, they are not suitable for real-time implementations (15 Hz or greater). Instead, we use a novel multiscale center-surround feature called CenSurE [12]. We have shown that CenSurE has the requisite stability properties, is extremely efficient to compute and works quite well for visual odometry applications [12].

Given the pose of the robot in an arbitrary global coordinate system, each PCD computed in the robot's camera

frame can be registered to extend the global model. Figure 4 illustrates the Visual Odometry registration process for two individual local PCD obtained from two different view points.

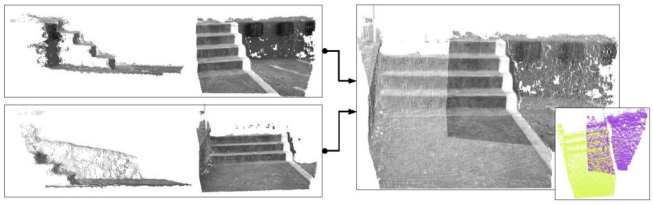


Fig. 4. Point cloud registration using Visual Odometry. The two images on the left represent two different PCDs, and the image on the right illustrates the resulting aligned model.

Because the VO algorithm computes new camera poses at about 10 Hz, the variation between two consecutive poses is typically small, and therefore it would be highly inefficient to update the 3D model with the same frequency. Therefore, we only update the model if the change in the camera pose since the last update is greater than a threshold (we use 0.1 m for the translation and 0.1 rad for the rotation). Figure 5 shows the final result after aligning approximately 1600 stereo frames.

Fig. 5. Globally aligned point cloud model for a dataset composed of approximately 1600 stereo frames.

### B. Polygonal Modeling and Data Simplification

A purely point-based map grows in size as more frames are captured by the camera (e.g., the model in Fig. 5 contains millions of points) and therefore storing and transmitting such models would quickly become unmanageable. Furthermore, the motion planner needs to perform many geometric queries on the model, such as distance and collision tests. Therefore, a simple and suitable model is critical to the efficiency of the entire system and towards this end we maintain and update a polygonal surface representation. This simplification reduces the size of the model by orders of magnitude, from several million points to tens of thousands of polygons.

For each new PCD frame, we construct a local octree, whose cells are aligned with some global underlying grid of resolution $h$. We restrict the octree to divide cells no smaller than $h$. We assign $h$ to roughly determine the minimum size of the model's polygons. For each cell in the local octree, we examine the set of points $S$ contained within. We then fit a plane $p$ to $S$ using the RMSAC (Randomized M-Estimator SAmple Consensus) [19] method. We then project the inliers $\tilde{S}$ to $p$, and then compute a reweighted convex polygon of the projected points.

Fig. 6. Point Cloud model simplification using polygonal fitting. From left to right: overlapping aligned point cloud views, combination PCD model and polygonal model, and the resulting polygonal model.

Fig. 7. Reducing the model complexity through merging and duplicate removal: before (left) and after (right).

To incrementally merge multiple polygonal models, we use the following procedure (illustrated in Fig. 6). For each overlapping octree cell, we compare the colliding polygons based on their relative orientation and distance. If the polygons are similar, we construct a new polygon that is the "average" of the two polygons weighted by their inlier support.

For efficiency purposes, we only consider the cells updated with the latest frame. We perform simple outlier removal by erasing lone polygons, i.e, polygons that have no neighbors in the scene. Because a single location can be observed several times during the mapping process, some surfaces may generate duplicate polygons in adjacent cells (see Fig. 7). We identify these duplicates by searching for parallel polygons among the neighboring polygons. If a coplanar polygon is found, the two polygons are merged. Experimentally, we observed that this procedure reduced the global number of polygons by a factor of 2.5 on average.

### C. Semantic Labeling

The motion planner manipulates several types of motion primitives, each of them adapted to a specific terrain type. To help the motion planner to select the appropriate series of motion primitives along the trajectory, some semantic information is added to the polygonal model. Our approach segments the terrain using rules that only examine the local terrain geometry. This helps achieve real-time performance. More sophisticated terrain types might be addressed by the methods of [20]–[23], which account for more global geometric relationships.

We predefined the following four labels (see Figure 8): **Level**- polygons that form an angle less than $15^\circ$ with the horizontal plane; **Ground**- Level polygons that are 5cm from the ground plane or are attached to **Ground** polygons; **Vertical**- polygons that form an angle greater than $75^\circ$ with the horizontal plane; **Steps**- Groups of adjacent polygons that form a step- (i.e., belong to planes that are parallel or perpendicular to the horizontal plane); **Unknown**- everything else.

After refining the global model, we label each polygon in an absolute sense based on its orientation and position with respect to the ground plane. We then propagate the ground plane by the process of relabeling **Flat** polygons that are adjacent to **Ground** polygons as **Ground** polygons in an iterative manner. With this ground propagation, a surface with a smooth inclination (up or down) is still considered flat ground. RHex can traverse small slopes and unevenness with ease, so ignoring minor irregularities helps speed up the motion planner. Figure 9 presents the refined

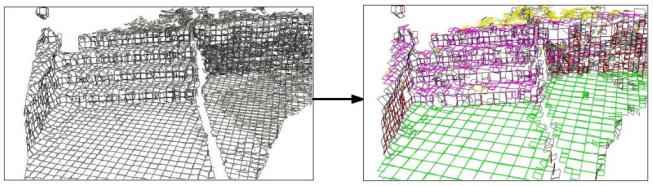


Fig. 8. Semantic labeling of polygonal surfaces based on heuristic geometrical rules. The labeled classes are: flat and level terrain (green and yellow), vertical structures (red), steps (purple), and unclassified or unknown (gray).

semantically annotated polygonal model of the point cloud dataset presented in Figure 5.

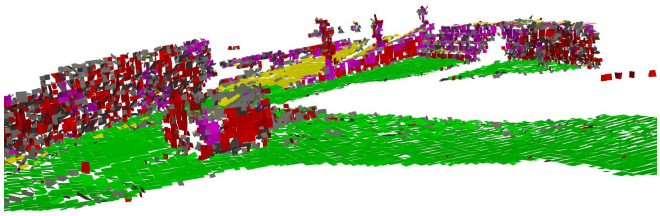


Fig. 9. Semantically annotated polygonal model for a dataset ("dumpster") consisting of approximately 1600 stereo frames.

### D. 3D Mapping Performance

*1) Computation Time.:* To evaluate performance, we monitored the computation time for each component of our 3D Mapping module. The components are labeled as follows:

- *Stereo + VO* - produces the disparity image from the stereo system and computes the global pose estimate;
- *PCD→LocalOctree* - produces the local PCD from the disparity image and creates/updates the local octree;
- *GlobalOctree growing* - expands the local octree's boundaries when needed;
- *LocalOctree→Polygon→GlobalOctree* - generates the polygonal model and merges it with the existing model;
- *GlobalOctree→GlobalModel* - refines the model (polygonal merging, polygonal growing, outlier removal);
- *Semantic Labeling* - labels the model.

Figure 10 presents the results for the dataset that correspondings to the model presented in figure 5. The majority of the model-building components have low computational requirements. Almost consistently, *Stereo + VO* requires the majority of resources. To address this, we implemented the

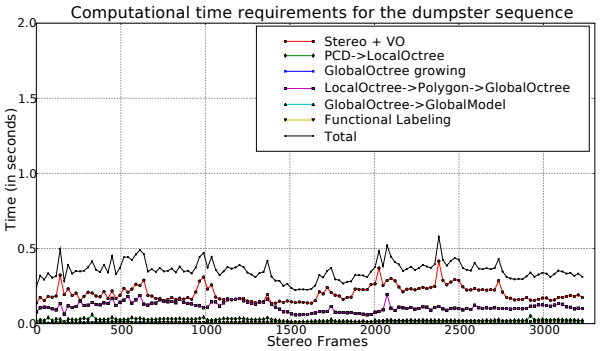



Fig. 10. Computational time of each 3D Mapping component for the dataset presented in Figs. 5 and 9.

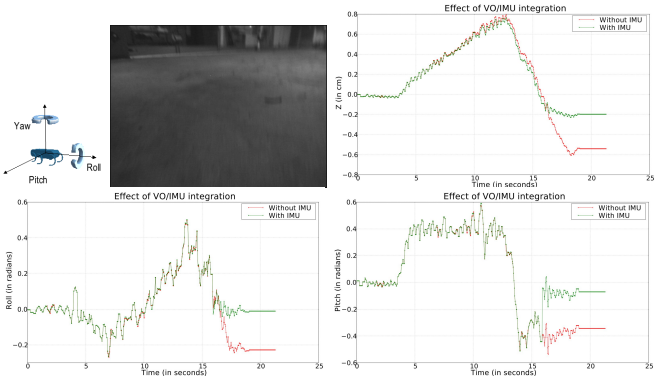



Fig. 11. Global pose corrections by integrating an IMU into the VO system (top right and bottom row) for a blurred frame (top left).

*Stereo + VO* component in a separate process from the other (model-building) components. By concurrently running the two processes, the global map can be updated every 250 ms on average.

*2) Accuracy.:* As previously explained in Sec. III-A, the accuracy of the global map depends on the quality of the Visual Odometry pose estimate. Some images can be blurred primarily due to rapid rotational accelerations of the robot (see the top-left part of Fig. 11). In addition, some areas of the environment do not contain enough features for the VO module. If several failures from VO occur consecutively, the system could lose track of robot pose and the integrity of the 3D model could be compromised.

These effects can be attenuated by integrating an Inertial Measurement Unit (IMU) and using it to fill in when VO fails to estimate a pose. In this version of our work, we did not integrate the translational information given by the IMU. We simply entered the angle data to update the last pose computed by VO. Figure 11 presents the corrections in roll (bottom left) and pitch (bottom right) before and after the IMU integration with VO. Even if some error in pitch persists, the drift on $z$ is reduced by approximatively 65% (top right).

## IV. Motion Planning

The motion-planning subsystem is called periodically to generate a trajectory that connects the current position of the robot to a user-designated target. The planner is given a map of the environment built by the 3D Mapping system. Its output is a trajectory that consists of a sequence of gait primitives to be executed by the robot, and their intermediate milestones. We seek a trajectory with low (but not necessarily optimal) execution cost. We define execution cost as a function of local terrain characteristics such as slopes and unevenness, as in [4].

### A. Overview

Usually, our planner treats RHex like a mobile robot on a 2D elevation map [4]. As mentioned in the Introduction, the elevation map approach cannot represent environments with multiple overlapping levels. Therefore, our planner starts by decomposing the 3D environment model into *regions* that are locally 2D, using a grid-based approach. Furthermore, when planning on uneven outdoor terrain, we must perform frequent stability tests in addition to collision tests. These tests are computationally moderately expensive, and are unnecessary on flat terrain. Similar simplifications can be made when planning on stairs and steep slopes. Thus, our planner chooses planning techniques that are specialized to the certain types of ground.

To implement this idea, the planner has a repertoire of *primitive planners* that are used only when the robot interacts with certain semantic types of terrain. The planner uses a two-phase algorithm to generate multi-region paths. First, the planner builds a large graph of configurations, with edges representing candidate primitive planning queries. This graph is used in the second phase to help choose low-cost primitive planning queries that are likely to yield a path to the goal. Several single-mode trajectories are then composed to produce a trajectory from the start to the goal.

### B. Terrain Decomposition

The terrain decomposition step decomposes the empty workspace into volumetric cells, which are bounded on the top and bottom by terrain surfaces (or no surface at all), and bounded on the sides by vertical faces. A *region* is defined as a collection of cells whose upper and lower terrain surfaces have the same semantic label, and are connected along boundary faces. The lower surface of a region can be represented as an elevation map. If two regions have connected boundary faces, we say they are *adjacent*. The graph of regions is called the *region graph*.

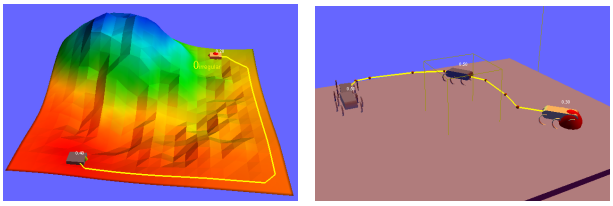
This structure could potentially be computed exactly from the polygonal model (Figs. 12a and b), but for large models of tens of thousands of polygons, this would be computationally expensive, and the structure would become extremely large. Instead, we use an approximation in which the polygons are projected and rasterized onto a x-y grid. The total running time of the approximate decomposition is $O(CK \log K)$, where $C$ is the number of grid cells, and $K$ is the maximum number of polygons overlapping a single grid cell.

The approximation is computed as follows. Consider a 2D grid cell $c$. Let $p_1, \ldots, p_k$ be the polygons whose projection onto the x-y plane intersect $c$. For each polygon $p_i$, let $[a_i, b_i]$ be the vertical range of the intersection between $p_i$ and $c$. Let $S = \bigcup_{i=1}^{k} [a_i, b_i]$ be the vertical range of $c$ intersected by surfaces, and $F = \mathcal{R} \setminus S$ be the vertical range which is free. For each connected component $f$ of $F$, we output a cell, whose top and bottom surfaces are identified by the polygon defining $f$'s upper and lower bound (Fig. 12c).

A flood-fill algorithm is used to determine coherent regions and their adjacencies (Fig. 12d). Additionally, we reduce the total number of regions by aggregating regions that only differ by their upper surface. This reduces the decomposition's complexity, which speeds up the multi-region planner. During this process, we refrain from incorporating regions that cause the x-y projection of the aggregated region to self-overlap.

### C. Primitive Planners

A mode is simply a set of regions in which a primitive planner can be applied. A mode is considered invalid if its regions are disconnected, or they overlap in the x-y plane. A primitive planning query is specified with a (valid) mode, a start configuration, and a goal configuration. The query terminates when the planner either produces a feasible trajectory, determines that no path exists, or exceeds a user-specified time limit.

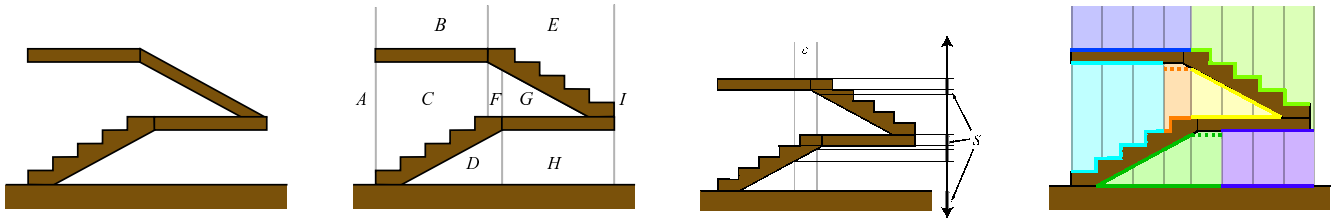


(a) Unlabeled terrain planner (b) Flat ground planner.

Fig. 13. Examples of paths planned by primitive planners. The cost function in (a) is increasing from red to blue

We use the following primitive planners, classified by the type of region(s) on which they are applicable:

- *Unlabeled terrain*. We use an A* search on a discretized $(x, y, \theta)$ state space [4]. Successors are generated by enumerating all gait primitives. At each generated configuration, we check for collisions with the region's ceiling, and compute the execution cost. We use a straight-line distance heuristic. The planner finds the trajectory with minimum cost (see Fig. 13a).
- *Flat ground*. We use a sample-based motion planner called SBL [24], which builds trees bidirectionally in configuration space to connect the start and goal. SBL is typically much faster than A* search and uses random sampling to avoid discretization artifacts. We use an $(x, y, \theta)$ configuration space, and propagate either car-like paths or turn-and-drive paths, depending on the




(a) An example terrain with three levels.

(b) An exact decomposition.

(c) Approximate decomposition of the cell $c$.

(d) Regions of the approximate decomposition.

Fig. 12. Illustrating the terrain decomposition step for a 2D slice of terrain.

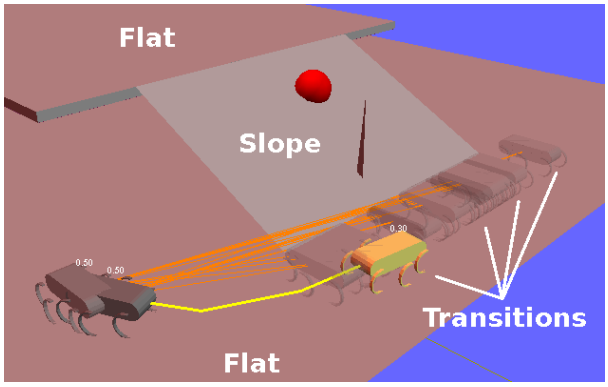



Fig. 14. Configuration space sampling at the junction of 2 different regions (transitions).

user's preference (see Fig. 13b). The robot's elevation, pitch, and roll are constant, so we do not need to consult an elevation map to assess stability. We check for collisions with the region's ceiling.

- *Stairs* or *steep slopes*. We restrict the robot's heading to either face up or down the slope, and use the stair-climbing gait primitive. The planner examines a 1D line in configuration space for collisions.
- *Flat ground and stairs*, or *flat ground and steep slopes*. The same as above.

The unlabeled terrain planner is also used as a backup when no other primitive planner is applicable.

### D. Multi-Region Planner

Each primitive planner can only be applied to modes whose regions are connected and do not overlap in the x-y plane (otherwise, this is considered an invalid mode). It is the multi-region planner's responsibility to select (valid) modes for primitive planner queries, and to choose fast planners when possible. For example, even though the unlabeled terrain planner can be applied anywhere, it is much slower on flat ground than the flat ground planner.

It is the multi-region planner's responsibility to select (valid) modes, initial and final configurations for primitive planner queries, and to choose the fastest planner for the given query. We use a two-phase algorithm, which is similar to the one used in motion planning for a climbing robot [25].

In the first phase, the algorithm builds a network of configurations including the start and goal configurations, as well as sampled configurations that transition between modes. We call this the *transition graph*. Each edge in the transition graph corresponds to a potential primitive planning query. Rather than making the query immediately, the edge is assigned a heuristic cost that includes an estimate of the planning cost and the cost of execution. The second phase of the algorithm refines the transition graph, making primitive planning queries corresponding to each edge.

The transition graph $\mathcal{G}$ is built using a search. To expand a configuration $q$ at a given mode $m$, we consider all modes $m'$ obtained by either 1) adding a region $r'$ adjacent to some region $r$ in $m$, or 2) removing a region in $m$. For each valid $m'$, we sample a set $T$ of configurations that transition from $m$ to $m'$, and add them to $\mathcal{G}$ (Fig. 14). If $m$ contains the goal configuration $q_g$, then we add $q_g$ to $T$. For each transition configuration $q' \in T$, we connect an edge between $q$ and $q'$. The process is repeated for some number of iterations, and at least until $\mathcal{G}$ contains a path to $q_g$.

To sample a configuration $q'$ that transitions from $m$ to $m'$, we use the following procedure. First, consider the case where $m'$ was obtained by adding a region $r'$ to $m$. To maximize the amount of terrain covered by simpler primitive planners, we try to find $q'$ that is still in $m$, but is close to $r'$. This can be done easily by sampling $q'$ in the cells in the terrain decomposition that border both $r'$ and a region $r$ in $m$, and then shifting $q'$ so that it is not supported by $r'$. If $q'$ violates the constraints of either $m$ or $m'$, it is rejected. Now consider the case where $m'$ removes a region $r$ from $m$. If the robot at configuration $q$ is supported by $r$, we sample $q'$ in the cells in the terrain that border both $r$ and a region $r'$ in $m'$, and shift $q'$ out of $r$. Otherwise, we simply set $q' = q$.

The second phase tries to find trajectories that correspond to edges in $\mathcal{G}$. For each edge $e = (q, q')$ in $\mathcal{G}$, we set the edge cost $c_e$ equal to a heuristic $\tilde{c}(q, q')$ that estimates the sum of the planning cost and execution cost. We then perform Dijkstra's algorithm to find the lowest cost path from start to goal. This path is then refined by executing single-mode planning queries along each *unsolved* edge $e$ along the path. If a single-mode trajectory is successfully found between $q$ and $q'$, we mark $e$ as *solved*, and replace $c_e$ with the trajectory's actual execution cost. If not, we remove $e$ from the graph. This procedure is repeated until a feasible

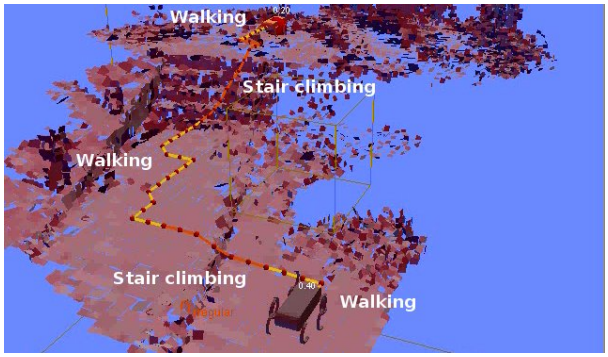



Fig. 15. Motion Planning example based on semantic polygonal labels for the "dumpster" dataset.

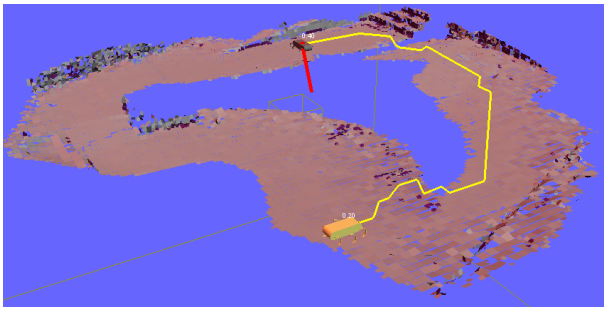


Fig. 16. Example of intuitive control using a joystick. The user describes a general direction (red line), and the motion planner adapts it to a correct trajectory (yellow polyline).

trajectory to the goal is found.

We also adaptively improve estimates of the heuristic edge costs. For each primitive planning query, we measure the computation time, and use it to update the planning cost estimate. For a primitive planner $p$, the planning cost $c_p(q, q')$ between $q$ and $q'$ is assumed to be a linear model $a_p d(q, q') + b_p$. We improve our estimates of $a_p$ and $b_p$ over time, using a recursive least squares estimation. We use a similar procedure to adaptively estimate execution cost.

### E. Experiments

We tested the planner with and without semantic labeling. First, we tested a problem with a uniformly flat terrain of over 20,000 triangles, with a point target 5 m away from the robot's initial position. The decomposition step takes approximately 220 ms. If the terrain is unlabeled, the multi-region planning stage took 184 ms to generate a path (averaged over 10 runs). Labeled as flat ground, it took 53 ms. We then tested a two-level problem consisting of flat ground and stairs, with a point target on a second level, approximately 2m above the robot. The decomposition step took approximately 200 ms. Multi-region planning took 886 ms on unlabeled terrain, and 134 ms on labeled terrain. Similar speedups were observed in several other experiments.

Experiments on terrain generated from the 3D Mapping module (e.g., Fig. 15) show that the planner can generally produce a path to targets up to approximately 5m away from the robot's initial position in less than a second. The running time can vary significantly depending on the size of the model, the shape of obstacles, and the performance of heuristics.

## V. Application : Assisted Teleoperation

Teleoperating a robot is a difficult task, especially when the robot is not in sight. The feedback from the robot's cameras is an insufficient source of information to guarantee collision-free navigation and the appropriate selection of motion primitives by a user. Our system, in real time, fulfills these objectives for the user. In order to combine the user's commands with our autonomous navigation system, we added some intuitive controls to task the motion planner. A user's simple direction, given via a joystick, is automatically translated to a complex trajectory in the 3D model with the correct series of motion primitives (Fig. 16). The user can also control the aggressiveness of the motion planning to modify the level of risk taken by the robot (Fig. 17).

### A. Intuitive Control

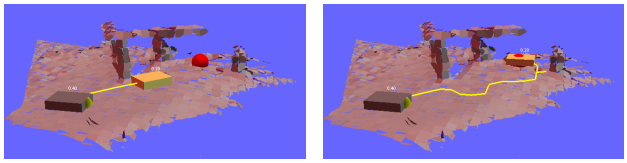
The motion planner described above assumed that a goal configuration was specified. This level of task specification is inefficient for a human operator, who most likely would be using a joystick or console to command the robot. Here, we describe how the planner can be adapted to handle more intuitive tasks, like a point target or line following (see Fig. 16).

We assume the user's input is translated into an early termination penalty function $p(q)$ defined over configuration space, which penalizes the planner if it chooses to terminate planning at a path ending in $q$. For example, a point target $t$ would define a reward function $p(q) = a||t - x(q)||$, where $a$ is a positive constant and $x(q)$ is the center of the robot at configuration $q$. The function can be shifted uniformly by an arbitrary constant without affecting the planner.

We implement early termination as follows: When building the transition graph $\mathcal{G}$, we do not add the goal configuration $q_g$ as a node in $\mathcal{G}$. Rather, we add a *virtual edge* $e$ between each transition configuration $q$ and a *virtual goal* configuration. We assign the cost $c_e = p(q)$ to each of these edges. We also allow the planner to early terminate within a mode, not necessarily at a transition. To do this, we augment $\mathcal{G}$ with configurations sampled inside a single mode. The second phase of the planner proceeds as usual, except that virtual edges are treated as solved, and the virtual goal is treated as the de facto goal.

### B. Aggressiveness Modulation

The early termination penalty function relies on the definition of a scale factor $a$. We denote this factor the *aggressiveness*. A high aggressiveness value causes the planner to: 1) produce trajectories with higher execution costs, and 2) spend more time trying to find a path before terminating. A low aggressiveness will make the planner act greedily and reactively, and be more risk-averse. Figure 17 illustrates an example.



(a) Low aggression (b) 5x higher aggression

Fig. 17. Plans to reach target on unlabeled terrain with different levels of aggression.

## VI. Conclusion

We present a comprehensive system for real-time 3D mapping and motion-planning for mobile robots. The robot is localized using Visual Odometry techniques and the mapping sub-system builds a labeled polygonal model using point cloud data in realtime. The semantic labeling helps the motion planner choose appropriate gaits and planning techniques. The motion planner uses a new decomposition technique to convert a 3D model into locally 2D regions. Existing mobile robot planning techniques can be used within each region, and our planner uses a repertoire of techniques specialized for certain terrain types, such as flat ground and stairs. We demonstrate that the mapping and planning subsystems exhibit favorable computational performance, which makes the overall system suitable for fast, real time 3D navigation.

Our short-term goals include system integration work and testing on the RHex robot in realistic reconnaissance and search-and-rescue scenarios. We plan to improve the mapping and localization modules by incorporating loop-closure techniques to globally align maps gathered over long periods of time. We plan to improve the accuracy of the 3D map by better polygon fitting, especially when environment features are not aligned to grid cell boundaries. We also plan to further develop our preliminary work on assisted teleoperation.

## Acknowledgments

The authors thank Edward Van Reuth and DARPA for support on the "Leaving Flatland" project (contract #FA8650-04-C-7136). This work was partially supported by the CoTeSys (Cognition for Technical Systems) cluster of excellence at the Technische Universität München.

## References

[1] R. Vincent, F. Dieter, J. Ko, K. Konolige, B. Limketkai, B. Morisset, C. Ortiz, D. Schulz, and B. Stewart, "Distributed multirobot exploration, mapping, and task allocation," in *Special Issue on Multi-Robot Coverage, Search, and Exploration*, D. A. Shapiro and D. G. A. Kaminka, Eds. Annals of Math and Artificial Intelligence (AMAI), 2008.

[2] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney, "Winning the darpa grand challenge," *J. of Field Robotics*, 2006.

[3] K. Konolige, M. Agrawal, R. C. Bolles, C. Cowan, M. Fischler, and B. P. Gerkey, "Outdoor mapping and navigation using stereo vision," in *Int. Symp. on Exp. Robotics (ISER)*, Rio de Janeiro, Brazil, 2006.

[4] T. Simèon and B. Dacre-Wright, "A Practical Motion Planner for All-terrain Mobile Robots," in *IEEE/RSJ Int. Conf. Intel. Rob. Sys*, Yokohama, Japan, 1993.

[5] J.-S. Gutmann, M. Fukuchi, and M. Fujita, "3d perception and environment map generation for humanoid robot navigation," *The International Journal of Robotics Research*, vol. 27, no. 10, pp. 1117–1134, 2008.

[6] Y. Liu, R. Emery, D. Chakrabarti, W. Burgard, and S. Thrun, "Using EM to Learn 3D Models of Indoor Environments with Mobile Robots," in *ICML*, 2001, pp. 329–336.

[7] P. Biber, H. Andreasson, T. Duckett, and A. Schilling, "3D Modeling of Indoor Environments by a Mobile Robot with a Laser Scanner and Panoramic Camera," in *IEEE/RSJ Int. Conf. on Intel. Rob. Sys.*, 2004.

[8] J. Weingarten, G. Gruener, and R. Siegwart, "A Fast and Robust 3D Feature Extraction Algorithm for Structured Environment Reconstruction," in *Int. Conf. on Advanced Robotics*, 2003.

[9] J. Diebel and S. Thrun, "An Application of Markov Random Fields to Range Sensing," in *Advances in Neural Information Processing Systems 18*, Y. Weiss, B. Schölkopf, and J. Platt, Eds. Cambridge, MA: MIT Press, 2006, pp. 291–298.

[10] U. Saranli, M. Buehler, and D. E. Koditschek, "Rhex: A simple and highly mobile hexapod robot," *Int. J. of Robotics Research*, vol. 20, no. 7, pp. 616 – 631, July 2001.

[11] M. Agrawal and K. Konolige, "Real-time localization in outdoor environments using stereo vision and inexpensive GPS," in *Int. Conf. on Pattern Recognition*, 2006.

[12] M. Agrawal, K. Konolige, and M. R. Blas, "Censure: Center surround extremas for realtime feature detection and matching," in *ECCV (4)*, ser. Lecture Notes in Computer Science, D. A. Forsyth, P. H. S. Torr, and A. Zisserman, Eds., vol. 5305. Springer, 2008, pp. 102–115.

[13] E. Z. Moore and M. Buehler, "Stable stair climbing in a simple hexapod robot," in *Int. Conf. Climbing and Walking Rob.*, Karlsruhe, Germany, 2001.

[14] K. Konolige, M. Agrawal, and J. Solà, "Large scale visual odometry for rough terrain," in *Proc. International Symposium on Robotics Research*, November 2007, p. To appear.

[15] M. Fischler and R. Bolles, "Random sample consensus: a paradigm for model fitting with application to image analysis and automated cartography," *Commun. ACM.*, vol. 24, pp. 381–395, 1981.

[16] C. Engels, H. Stewnius, and D. Nister, "Bundle adjustment rules," *Photogrammetric Computer Vision*, September 2006.

[17] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[18] T. T. Herbert Bay and L. V. Gool, "Surf: Speeded up robust features," in *European Conference on Computer Vision*, May 2006. [Online]. Available: http://www.vision.ee.ethz.ch/ surf/

[19] P. Torr and A. Zisserman, "MLESAC: A new robust estimator with application to estimating image geometry," *Computer Vision and Image Understanding*, vol. 78, pp. 138–156, 2000.

[20] O. M. Mozos, R. Triebel, P. Jensfelt, A. Rottmann, and W. Burgard, "Supervised Semantic Labeling of Places using Information Extracted from Laser and Vision Sensor Data," *Rob. and Aut. Syst.*, vol. 55, no. 5, pp. 391–402, May 2007.

[21] D. Schröter, T. Weber, M. Beetz, and B. Radig, "Detection and Classification of Gateways for the Acquisition of Structured Robot Maps," in *Proc. of 26th Pattern Recognition Symposium (DAGM), Tübingen/Germany*, 2004.

[22] A. Nuechter, H. Surmann, and J. Hertzberg, "Automatic Model Refinement for 3D Reconstruction with Mobile Robots," in *Proc. of the 4th IEEE Int. Conf. on Recent Advances in 3D Digital Imaging and Modeling (3DIM'03)*, Banff, Canada, October 2003.

[23] R. B. Rusu, Z. C. Marton, N. Blodow, M. Dolha, and M. Beetz, "Towards 3D Point Cloud Based Object Maps for Household Environments," *Robotics and Autonomous Systems Journal (Special Issue on Semantic Knowledge)*, 2008.

[24] G. Sánchez and J.-C. Latombe, "On Delaying Collision Checking in PRM Planning: Application to Multi-Robot Coordination," *Int. J. of Rob. Res.*, vol. 21, no. 1, pp. 5–26, 2002.

[25] T. Bretl, S. Lall, J.-C. Latombe, and S. Rock, "Multi-Step Motion Planning for Free-Climbing Robots," in *WAFR*, Zeist, Netherlands, 2004.