Efficient Traversability Analysis for Mobile Robots using the Kinect Sensor

Igor Bogoslavskyi Olga Vysotska Jacopo Serafin Giorgio Grisetti Cyrill Stachniss

***Abstract*— For autonomous robots, the ability to classify their local surroundings into traversable and non-traversable areas is crucial for navigation. In this paper, we address the problem of online traversability analysis for robots that are only equipped with a Kinect-style sensor. Our approach processes the depth data at 10 fps-25 fps on a standard notebook computer without using the GPU and allows for robustly identifying the areas in front of the sensor that are safe for navigation. The component presented here is one of the building blocks of the EU project ROVINA that aims at the exploration and digital preservation of hazardous archeological sites with mobile robots. Real world evaluations have been conducted in controlled lab environments, in an outdoor scene, as well as in a real, partially unexplored, and roughly 1700 year old Roman catacomb.**

I. Introduction

Autonomous outdoor navigation is an active research field in robotics. In most navigation scenarios, the classification of terrain into traversable and non-traversable areas plays an important role. Failing to stay on roads or other traversable surfaces can introduce wheel slippage, which in turn leads to errors in the odometry, and the risk of getting stuck or of destroying the platform. Therefore, the ability to robustly detect traversable areas is important for safe navigation, especially in fully autonomous settings.

Our work is motivated by a project for the autonomous exploration and digital preservation of hard-to-access archeological sites such as catacombs. Catacombs are old Roman burying places used between the 2nd and 5th century in Italy. Even today, they are partially unexplored due to the high risk of entering them. First, most sites are unstable and can collapse. Second, most of the (non-ventilated) catacombs yield a high concentration of radioactive radon gas so that humans are only allowed to stay in these sites for 15 min-30 min to prevent serious health issues. Thus, robots are an excellent tool for the exploration, mapping, and digital preservation of such sites. To achieve that, the robots have to operate and explore the space in a completely autonomous fashion and for this task, robust traversability analysis is an essential prerequisite.

The main contribution of this paper is an accurate, fast to compute, and comparably easy to implement traversability analysis approach for mobile robots. Our system operates on the depth images of a Microsoft Kinect or an ASUS Xtion pro live camera and analyzes the visible area in front of the robot at 10 fps-25 fps on a notebook computer without using the graphics processing unit (GPU). Not relying on GPUs has the advantage of requiring less energy, which is a relevant issue for small-scale autonomous robots. Our approach has been implemented and evaluated in several sites including a real catacomb. An example of a labeled depth image is shown in Fig. 1.

Igor, Olga, and Cyrill are with the University of Freiburg, Institue of Computer Science, 79110 Freiburg, Germany. Jacopo and Giorgio are with the La Sapienza University of Rome, Dept. of Systems and Computer Science, 00185 Rome, Italy. This work has partly been supported by the European Commission under FP7-600890-ROVINA.

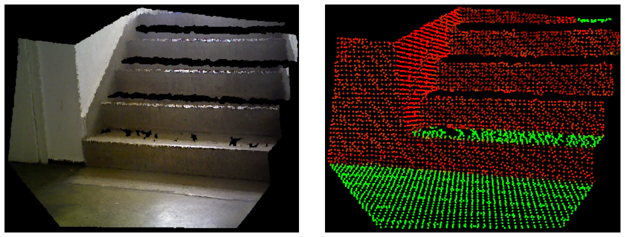

Fig. 1. Example or a stair case observed with a Kinect and a corresponding labeling into traversable (green) and non-traversable (red) areas.

II. Related Work

Estimating traversable areas is essential for most navigation tasks and thus has been investigated intensively in the past. For example, Rasmussen [16] proposes an approach to trail following for mobile cross country robots. The robot investigates the local variance of depth measurements, structural texture, and contrast to identify and follow a trail. The work closest to our approach is probably the work of Renner et al. [18] that aims at estimating environment properties such as positive obstacles, flexibility, shape, dimensions, slope, etc. using a camera and a PMD depth sensor. Besides visual information such as texture, color, and variance in contrast, they also consider surface normals and steps to identify obstacles and create a polar, robot-centric model based on which they navigate. Similar to Renner et al., De Cubber et al. [5] address outdoor terrain traversability using a PMD and a stereo camera. They estimate a ground plane and seek for pixels that have a high probability of belonging to the ground plane. Then, they use the color information of other pixels to classify all image pixels as traversable.

Other approaches perform a semantic scene analysis to support navigation [17], [19]. Ren et al. [17] propose an approach for indoor scenes using a Kinect. They compute a combination of color and depth features using kernel descriptors and achieve a high labeling performance by combining Markov random fields with segmentation trees.

Katramados et al. [12] present a real-time approach for traversable surface detection using a monocular camera mounted on robot. Based on the currently assumed to be traversable location, the system searches similar areas in the image given color and texture features. The approach of Maier et al. [13] combines a monocular camera with sparse laser range data on a humanoid to identify obstacles on the ground plane. They infer the traversability information based on the vision data after learning a classifier from sparse laser information.

Terrain types have also been classified using vibration sensors [4], [20]. Here, the vibration measurements are usually analyzed in the Fourier domain. Brooks and Iagenemma [4] use a combination of PCA and LDAto classify terrain and Weiss et al. [20] use SVMs. Vibration-based approaches typically offer highly accurate classification results. The drawback of such methods, however, is that only the terrain the robot is moving on can be classified and not the terrain in front of the robot.

There exist approaches that apply self-supervised learning to classify terrain and detect obstacles. A number of methods have been developed that exploit local terrain knowledge to predict surface terrain in the far range. These near-to-far approaches use color information [9], [8], 3D geometry information [14], or texture information [1]. Self-supervised learning using laser and vison data is also used by Dahlkamp et al. [6] in a vision-based road detection system. Finally, traversability analysis is also the motivation for several approaches that aim at detecting vegetation such as grass. They typically use laser remission values [22], laser range data [10], [21], and also combinations with vision [3], [7].

III. Our Approach

The main objective of our work is developing an accurate and fast to compute traversability analysis system for mobile robots operating in catacomb-like environments. A central focus lies on the online capabilities of the system on a standard notebook computer without requiring a GPU so that the traversability analysis can be computed and integrated into the model on the fly. Our approach considers only the depth image. Our motivation is that first, the underground environments are completely dark and RGB data is basically useless. Second, the depth cue is in general more informative for estimating the traversability compared to RGB data.

Our approach can be split up into two main steps. After a preprocessing step, our system first estimates the local traversability based on a single depth image. This step takes into account the navigation capabilities of the vehicle. Second, the integration of the single-image traversability estimates into a local traversability map.

Note that we assume the depth images to be locally registered, i.e., the traversability analysis does not account for any pose uncertainty of the vehicle. In practice, we use an incremental ICP-based matching approach that uses the point clouds and normal vectors and can be executed on the fly. The approach can also be used with global methods such as graph-based SLAM and obtain a globally consistent model. This is straight forward if the local traversability estimates are stored in the nodes of the SLAM graph. After the optimization, the global traversability map can be rendered from the local views in the global frame. Global mapping, however, is not the scope of this paper.

A. Traversability for a Mobile Robot

Given a typical robotic platform such as a Pioneer 2AT or a Mesa Element platform, the traversability is mainly governed by three factors. First, a maximum step height limits the vehicle from climbing steps higher than 5 cm-15 cm (depending on the wheels/tracks and the exact setup). Second, the maximum slope the robot can climb or descend. The exact figures depend on the weight distribution of the platform and its sensors. In our case, the maximum slope was 15 deg. A third factor, that limits the traversability is the height of the platform, which may prevent the robot from driving into low niches or similar places. Additional factors may impact the traversability of a platform such as mud or water – such surfaces are, however, hard or even impossible to be identified with a depth sensor such as the Kinect and are therefore not considered here.

Our approach only considers the environment and not explicitly the shape of the platform such as its width and height. The question if the robot physically fits into an area that is labeled as traversable has to be done by the planner itself as the state of the robot, especially its orientation, influences that. Thus, this decision is done by the planner. Therefore, also currently unreachable places can be classified as traversable as the planner will not expand these states.

Based on the constraints described above, the tasks of estimating the traversability consists of analyzing the environment covered by the sensor and to identify height constraints, steps, and slopes. Before providing further details on that, we briefly describe how to efficiently compute surface normals directly from the depth image and introduce the sparse data structure that we use to store the 3D data. Both support the three detection steps and are key for fast online processing.

B. Fast Normal Computation from a Depth Image

Our approach exploits information about the surface normals of the perceived environment. Thus, the first operation we perform is to compute a normal vector for each 3D point using the depth image. For a point p, we can compute its normal by considering a Gaussian distribution with parameters μ and Σ, which models the distribution of the neighboring points of p. The eigenvector of Σ that corresponds to the smallest of the three eigenvalues provides the direction of the normal.

We apply a fast method for normal extraction that exploits the structure of the input similar to [11], and takes advantage of the underlying hardware. We can rewrite the computation of the mean and the covariance matrix as:

$$\mu \;=\; \frac{1}{N}\underbrace{\sum_i p_i}_{P} \;=\; \frac{1}{N}P \qquad (1)$$

$$\Sigma \;=\; \frac{1}{N}\underbrace{\sum_i p_i p_i^T}_{S} - \mu\mu^T \;=\; \frac{1}{N}S - \mu\mu^T, \qquad (2)$$

where N is the number of neighbors of p. If we know the terms P and S, we can compute the covariance matrix and thus the normals in constant time. By exploiting the fact that we are using a depth image as the input, we can realize the computation of P and S in constant time too through the use of integral images.

The integral image $\mathcal{I}(i,j) \to (P_{ij}, S_{ij})$ maps the pixel coordinates (i,j) to the tuple (P_{ij}, S_{ij}) such that

$$P_{ij} \;=\; \sum_{k=1}^{i}\sum_{l=1}^{j} p_{kl} \qquad S_{ij} \;=\; \sum_{k=1}^{i}\sum_{l=1}^{j} p_{kl}p_{kl}^T, \qquad (3)$$

where p_{kl} corresponds to the 3D point, which is observed by the pixel (k,l). Thus, this integral image is a 2D array where the position (i,j) contains the sum P_{ij} and the squared sum S_{ij} of all other points visible in the rectangular area between $(1,1)$ and (i,j).

Exploiting the fact that the terms S and P support addition and subtraction, we can determine S and P for any rectangular region (i_1,j_1) to (i_2,j_2) in the integral image by

$$\begin{aligned}\mathcal{I}(i_1,j_1,i_2,j_2) \;=&\\ &\mathcal{I}(i_2,j_2) + \mathcal{I}(i_1,j_1) - \mathcal{I}(i_1,j_2) - \mathcal{I}(i_2,j_1). \qquad (4)\end{aligned}$$

By processing the depth image from the top left to the lower right corner, each computation exploits the result of the previous step leading to a constant time computation of P_{ij} and S_{ij} for each pixel.

Note that the computation of the normals through the integral image is an approximation. The reason is that only rectangular image regions can be queried with Eq. (4) and the queries are performed in image coordinates and not in world coordinates. The appropriate size of the region, however, can easily be determined given the distance of the query point to the camera.

The implementation of the algorithm described above can furthermore exploit the SSE extensions of Intel CPUs by operating on packed structures of four floats. The points are stored as homogeneous vectors $p = (x,y,z,1)^T$. The terms P and S are stored as a 4-dimensional vector and as a 4×4 matrix respectively. Adding points to an accumulator P using this representation, results in a homogeneous component that contains the number of points. This has the effect of removing conditional instructions and of performing the evaluation almost entirely in the SSE subsystem of the CPU. This provides a speed up of a factor of 3 compared to an implementation of the same algorithm without SSE instructions.

C. Sparse 3D Map

The second operation we perform is to store the measured endpoints and normals in a 3D data structure. In theory, a raw point cloud could be used but they have the disadvantage that finding neighboring points is expensive. As finding neighbors is explicitly required later on, we propose to approximate the points using a grid-like structure. As our data is sparse compared to the number of cells of a full 3D grid, we store the data in a sparse grid that is realized via a two-layered hash-table like structure.

First, a hash-table is used to index points in the x-y plane in world coordinates. The key of the hash-table is the discretized (x,y) coordinate of points that can, given the range and resolution of the Kinect sensor, be modeled by a single 32 bit integer. This hash table acts as a sparse 2D grid as only those cells are instantiated for which 3D points with a corresponding (x,y) coordinate exists. For every non-empty (x,y) grid cell, we initiate a red-black tree, which is a self-balancing binary search tree that allows for quickly accessing elements and for processing them in a sorted order. The key of each red-black tree is the discretized z coordinate of the endpoints to be stored. From an implementation point of view, this may sound complex but note that the C++ standard template library implements a red-black tree within std::map and thus can directly be used without additional efforts. The same holds for the hash table (std::unordered_map).

The overall data structure models a sparse 3D grid that allows us to store discretized (x,y,z) triplets and, thanks to the red-black tree, allows for parsing the z coordinate efficiently in an ordered way. This will be used to compute steps for (x,y) cells and to estimate if the height constraints of the platform are violated.

In sum, the points of the point cloud that is computed from the depth image are added to the sparse 3D grid. For each 3D cell, we compute the average 3D coordinate and normal on the fly. In our current implementation, we use a discretization of 4 cm. As a result of that, we obtain a (deterministically) subsampled point cloud that is stored in a sparse 3D grid. The following two operations can be conducted efficiently: accessing any cell including neighboring cells and iterating over the sorted z-coordinates of all cells for a given (x,y) coordinate.

D. Accounting for the Vehicle Height

The easiest criterion that impacts the traversability of the terrain is the height constraint of the vehicle. For every (x,y) location in our sparse 3D map M, we start from the lowest measured z value and compute the free space in z direction to the next obstacle. Any obstacle for which the difference in the z coordinate to the previous obstacle is larger than the height of the platform can be discarded as it does not constrain the motion of the vehicle. All obstacles for which this distance is smaller than the height of the platform are maintained in M and will, in the subsequent steps, be analyzed according to their step height and slope.

E. Efficient Step Detection

Based on the sparse 3D map M, we can efficiently query the neighboring points for any 3D coordinate. By analyzing the height differences between points stored in neighboring cells, we can quickly check for large steps that the robot cannot traverse. The neighbors of a point p up to a distance of d can be written as

$$\mathcal{N}(p,d) \quad = \quad \{q \mid q \in M \wedge ||p-q|| \leq d\}. \tag{5}$$

For each instantiated grid cell $p \in M$, we inspect the z coordinates of the neighbors

$$q_z \quad = \quad \{z \mid (x,y,z) \in \mathcal{N}(p,d)\} \tag{6}$$

and test if the coordinate is larger then z_{max}, which is the maximum step that the vehicle can climb or descend. The decision about traversability is then done by the following function

$$a^{(p)}_{step} \quad = \quad \begin{cases} 1 & \exists\, q \in q_z \,:\, ||p_z - q|| \leq z_{max} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

The expression is equal to 1 if there is any non-traversable step at p and $a = 0$ otherwise. In our implementation, the neighbor distance d was set to 10 cm and the maximum step height z_{max} that the robot can traverse was 10 cm as well.

F. Robust Slope Detection

Besides steps, there is a second criterion that is important for deciding if terrain is traversable or not: Up to which degree can the robot climb slopes?

For computing the slope of a local area, we consider local normal vectors which are efficiently computed as explained in Sec. III-B. Based on the normal vector n and the gravity vector n_g, a straight forward test allows us to estimate the traversability of a perceived surface based on the slope as

$$n_g \cdot n \quad \leq \quad \cos(\alpha_{max}), \tag{8}$$

where $\cdot$ is the scalar product and α_{max} the maximum slope the robot can handle. In our setup, the gravity vector n_g is obtained from a standard IMU. We use an XSens MTi, which provides the gravity vector up to an error of approx. 0.5 deg.

The test in Eq. (8) allows us to efficiently detect normal vectors that yield a steeper slope than the navigation capabilities of the robot. However, testing only for the slope is not enough. Consider a small step that the robot can traverse. The vertical surface of the step will create normal vectors that are orthogonal to the gravity vector and thus report a steep slope that cannot be traversed. Mathematically, that surface is correctly labeled as 90 deg slope but it should not affect the traversability labeling as long as the step is small enough to be traversed by the robot. Thus, we are only interested in slopes that have a minimum extension in the x-y plane in order to be classified as a slopes, which cannot be traversed.

To achieve this, we apply a rather standard erosion-dilation filter [2] with a 5-cross structuring element to our traversability map. Let $a^{(x,y)}_{slope}$ be the traversability label for the position (x, y) where 0 refers to traversable and 1 to non-traversable. The erosion step updates the estimate

$$\begin{aligned} a^{(x,y)}_{slope} \quad \leftarrow \quad & \max\big(a^{(x,y)}_{slope}, a^{(x-d,y)}_{slope}, a^{(x+d,y)}_{slope}, \\ & a^{(x,y-d)}_{slope}, a^{(x,y+d)}_{slope}\big), \end{aligned} \tag{9}$$

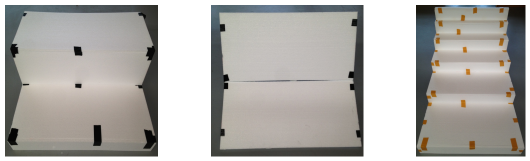

Fig. 3. Photos of the individual test objects: two steps of different size, a connected shallow and steep slope, steps of changing size.

and is followed by the dilation step

$$\begin{aligned} a^{(x,y)}_{slope} \quad \leftarrow \quad & \min\big(a^{(x,y)}_{slope}, a^{(x-d,y)}_{slope}, a^{(x+d,y)}_{slope}, \\ & a^{(x,y-d)}_{slope}, a^{(x,y+d)}_{slope}\big). \end{aligned} \tag{10}$$

Here, the scalar d describes the distance in which the neighbor considered (as in the previous subsection on step detection). As a result of the erosion-dilation filtering, small slopes such as steps are filtered out while "real slopes", i.e., larger areas with a steep inclination angle, are maintained.

Finally, the traversability $a^{(x,y)}$ is obtained by combining the traversability extracted from slopes $a^{(x,y)}_{slopes}$ and steps and height constraints $a^{(x,y)}_{steps}$ by

$$a^{(x,y)} \quad = \quad \max\big(a^{(x,y)}_{slope}, a^{(x,y)}_{step}\big). \tag{11}$$

In sum, this approach provides a traversability estimate given a single depth image.

G. Traversability Map Estimation

Let z_t be such a traversability estimate of a local area obtained from a single Kinect image taken at time t. As this estimate is not free of errors, we integrate multiple of such measurements into one model. We achieve that by employing a static state binary Bayes filter that integrates the information for every non-empty cell i in M.

Following the work of Moravec [15], we can compute a recursive update formula for $P(a^i \mid z_{1:t})$ as

$$\begin{aligned} & P(a^i \mid z_{1:t}) \;= \\ & \left[1 + \frac{1-P(a^i \mid z_t)}{P(a^i \mid z_t)} \frac{1-P(a^i \mid z_{1:t-1})}{P(a^i \mid z_{1:t-1})} \frac{P(a^i)}{1-P(a^i)}\right]^{-1} \end{aligned} \tag{12}$$

In order to gain efficiency, one can furthermore use the log-odds formulation of Moravec, so that the operations in Eq. (12) are realized via addition and subtractions in the log-odds space.

To apply the Bayes filter, we need to specify the inverse observation model $P(a^i \mid z_t)$. As the depth resolution decreases quadratically with increasing distance from the sensor, we use the inverse sensor model (assuming a prior of $P(a^i) = 0.5$)

$$P(a^i \mid z_t) \quad = \quad 0.5 + \frac{2a^i - 1}{2 + l^2}, \tag{13}$$

where l is the distance between the camera and the measured cell (0.7 m-4 m) and $a^i \in \{0, 1\}$. With this filter, the individual traversability estimates that are computed per depth image can easily be integrated into one model.

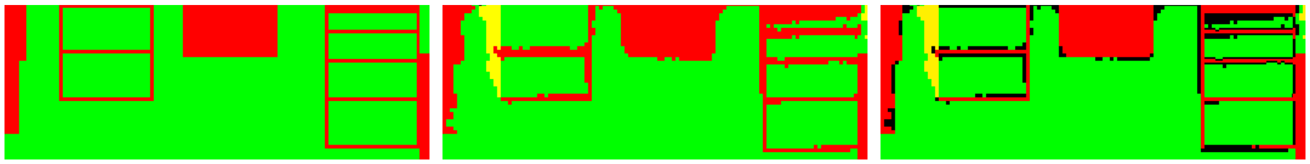

Fig. 2. Ground truth comparison using the objects pictured in Fig. 3. Left: ground truth labeling. Middle: our approach. Right: overlayed images (truth above estimate). Green cells are labeled as traversable, red refers to non-traversable and yellow to cells for which not enough observations have been made to allow for a confident labeling. Wrongly labeled pixels are highlighted in black. Note that most error are due to discretization effects.

CPU	640×480 pixels			320×240 pixels		
	normals	trav.	fps	normals	trav.	fps
i7	38 ms	55 ms	10.7	18 ms	22 ms	25
i5	85 ms	70 ms	6.6	40 ms	25 ms	15.3

Fig. 4. Average timings of the normal computation and traversability analysis and the overall rate for two different resolutions of the depth image.

IV. Experiments

The experimental evaluation is designed to show the capabilities of our traversability estimation system in different environments. Throughout all our experiments, we used either a Microsoft Kinect (catacomb scenes) or an ASUS Xtion pro live (office and outdoor scenes) installed on a mobile robot. We only used the depth information for the traversability estimate, the RGB information is only used for visualizations.

A. Timing Experiments

The first experiment is designed to illustrate that our method runs online on a notebook computer without GPU usage and can process the incoming depth images at high framerate. We tested our system on two standard notebooks, one equipped with a 2.3 GHz i7 processor and one with an i5-2410M processor. Fig. 4 illustrates the results. As can be seen, we achieve framerates between 10 fps and 25 fps on an i7 notebook depending on the depth image resolution. Thus, the environment in front of the robot can be analysed on the fly allowing for autonomous navigation and exploration. Even on the i5, the data can be analyzed with 15 fps on a 320×240 pixel depth image.

B. Ground Truth Comparison

The next experiment is designed to analyze the error of our traversability estimate. It is non-trivial to provide a ground truth analysis outside a simulator, but we put our best efforts to achieve a near ground truth evaluation by observing custom-made structures with known 3D geometry and compare the traversability estimates pixel by pixel with the geometric model. Fig. 2 illustrated the objects and the results. The right image shows the overlay of the ground truth (left) and estimated (middle) maps. All errors based on a pixel-by-pixel comparison are highlighted in black. In this experiment, 7.2% of the cells are wrongly labeled if considered independently. However, nearly all wrongly labeled cells occur at the borders of the obstacles and are between one and two cells sizes away from the real obstacle. Most of these errors actually result from discretization errors or slight smoothing effects at steps when computing the normal. In addition to that, 1.9% of the cells were not observed sufficiently to allow for an appropriate labeling. This occurs if $P(a^i \mid z_{1:t})$ has a value close to the prior (here 0.5). These cells are colored yellow.

This experiment shows that our approach provides an accurate labeling when ignoring the discretization errors. This can typically be done on most real navigation settings and the system classifies well for our application—exploring an unknown catacomb with a mobile platform.

C. Traversability Estimated Obtained in Different Scenes

Finally, we would like to show initial results obtained with our system in real world scenes. First, this includes the deployment of a prototype robot based on a Pioneer2 AT system in the Catacombe di Priscilla in the underground of Rome, see Fig. 5. The robot was steered through the environment, incrementally aligning the depth images of a Kinect and building the traversability map. Fig. 6 illustrates a fraction of the traversability map showing the traversable and non-traversable areas. Fig. 7 and 8 illustrate two selected places showing the RGB image from the Kinect and the local traversability map.

Finally, Fig. 9 shows the results obtained on the Freiburg computer science campus outdoors on a cloudy day. As can be seen, the small rocks are traversable for the outdoor platform but not the big rock and two steps. Finally, the motivating example in Fig. 1 illustrates a labeled depth image (without the integration into a traversability map).

D. Limitations

We also experienced limitations of our system. There are situations in which parts of the environment are traversable only if the robot steers in a specific way, otherwise not. An example are track-like obstacles that are too high to be traversed but are at the same time low enough to fit between the wheels of the robot. Another example is a v-shaped valley in which the slopes are traversable except a corner-shaped bottom, which is not non-traversable (it is depended on the position of the wheels on the platform). These situations are subject to future work.

V. Conclusion

Traversability information is important for autonomous mobile robots. This paper presents a system for estimating the local traversability for a mobile robot based on Kinect images online. Our approach can process the depth data at 10 fps-25 fps on a standard i7 laptop computer without a GPU and allows for robustly identifying the areas in front of the sensor that are safe for navigation. The component presented here is one of the building blocks of the EU

Fig. 5. Pictures of our robot and of the Catacombe di Priscilla used as a test environment for our approach.

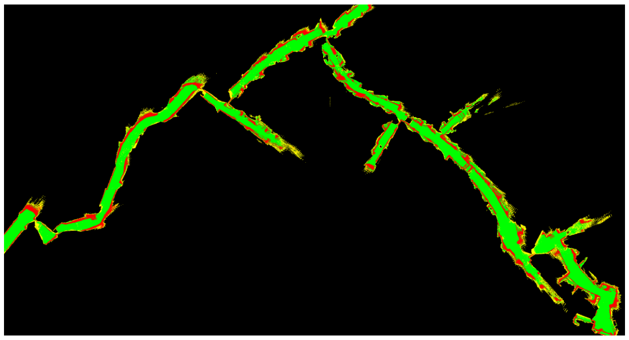

Fig. 6. Fraction of the explored space of the Catacombe di Priscilla.

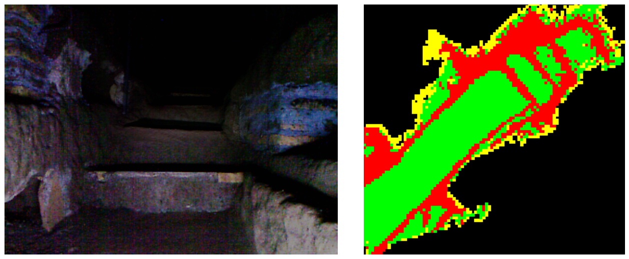

Fig. 7. A stair case experienced during the mapping of a catacomb site that is not traversable for the Pioneer robot. Left: RGB image from the Kinect. The image is dark as the onboard light was not powerful enough to appropriately illuminate the scene. Right: traversability estimate.

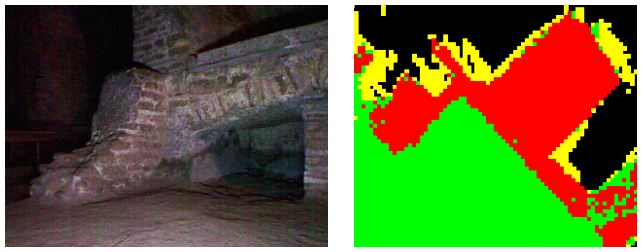

Fig. 8. A situation in which the ground level is flat and traversable but the height of the platform prevents the robot from entering the niche. Left: RGB image from the Kinect. Right: traversability estimate.

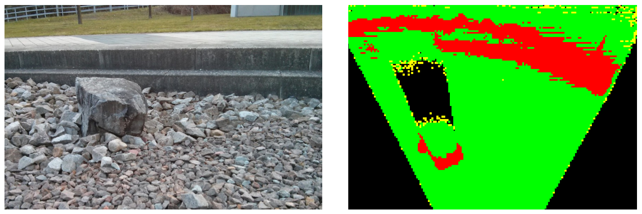

Fig. 9. Example of an outdoor scene on the Freiburg campus observed by a rough terrain robot. The left image shows a photo and the right one shows the local traversability map (distorted). The black region corresponds the area behind the obstacle that was not visible.

project ROVINA that aims at the exploration and digital preservation of hazardous archeological sites with mobile robots. As we showed in our experimental evaluation, our approach is able to reliably estimate the traversability in different environments, ranging from lab to outdoor scene as well as in a real partially unexplored, and nearly 1700 year old Roman catacomb in the underground of Rome.

References

[1] A. Angelova, L. Matthies, D. Helmick, and P. Perona. Dimensionality reduction using automatic supervision for vision-based terrain learning. In *Proceedings of Robotics: Science and Systems*, 2007.

[2] A. Bovik. *Handbook of Image and Video Processing*, chapter 2. Elsevier, 2005.

[3] D. Bradley, R. Unnikrishnan, and J. Bagnell. Vegetation detection for driving in complex environments. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2007.

[4] C.A. Brooks, K. Iagnemma, and S. Dubowsky. Vibration-based terrain analysis for mobile robots. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 3415–3420, 2005.

[5] G. De Cubber, D. Doroftei, H. Sahli, and Y. Baudoin. Outdoor terrain traversability analysis for robot navigation using a time-of-flight camera. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2011.

[6] H. Dahlkamp, A. Kaehler, D. Stavens, S. Thrun, and G. Bradski. Self-supervised monocular road detection in desert terrain. In *Proc. of Robotics: Science and Systems (RSS)*, Philadelphia, USA, 2006.

[7] B. Douillard, D. Fox, and F. Ramos. Laser and vision based outdoor object mapping. In *Proc. of Robotics: Science and Systems*, 2008.

[8] G. Grudic, J. Mulligan, M. Otte, and A. Bates. Online learning of multiple perceptual models for navigation in unknown terrain. In *Field and Service Robotics*, pages 411–420. Springer, 2008.

[9] M. Happold, M. Ollis, and N. Johnson. Enhancing supervised terrain classification with predictive unsupervised learning. In *Proceedings of Robotics: Science and Systems*, 2006.

[10] M. Hebert and N. Vandapel. Terrain classification techniques from ladar data for autonomous navigation. In *Proc. of the Collaborative Technology Alliances conference*, College Park, MD., 2003.

[11] S. Holzer, R.B. Rusu, M. Dixon, S. Gedikli, and N. Navab. Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2012.

[12] I. Katramados, S. Crumpler, and T.P. Breckon. Real-time traversable surface detection by colour space fusion and temporal analysis. In *Proc. International Conference on Computer Vision Systems*, 2009.

[13] D. Maier, C. Stachniss, and M. Bennewitz. Vision-based humanoid navigation using self-supervised obstacle detection. *The Int. Journal of Humanoid Robotics (IJHR)*, 10, 2013.

[14] L. Matthies, M. Turmon, A. Howard, A. Angelova, B. Tang, and E. Mjolsness. Learning for autonomous navigation: Extrapolating from underfoot to the far field. *J. of Machine Learning Research*, 1, 2005.

[15] H.P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, pages 61–74, 1988.

[16] C. Rasmussen. Kinects for low- and no-sunlight outdoor trail-following. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012.

[17] X. Ren, L. Bo, and D. Fox. Indoor scene labeling using rgb-d data. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012.

[18] A. Renner, T. Foehst, and K. Berns. Perception of environment properties relevant for off-road navigation. 2009.

[19] C. Stachniss, O. Martínez-Mozos, A. Rottmann, and W. Burgard. Semantic labeling of places. In *Proc. of the Int. Symposium of Robotics Research (ISRR)*, San Francisco, CA, USA, 2005.

[20] C. Weiss, H. Frohlich, and A. Zell. Vibration-based terrain classification using support vector machines. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

[21] D.F. Wolf, G. Sukhatme, D. Fox, and W. Burgard. Autonomous terrain mapping and classification using hidden markov models. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2005.

[22] K.M. Wurm, H. Kretzschmar, R. Kümmerle, C. Stachniss, and W. Burgard. Identifying vegetation from laser data in structured outdoor environments. *Journal of Robotics & Autonomous Systems*, 2012.