International Conference on Intelligent Robots and Systems (IROS), Nice, France, IEEE Press, 2008

Fast Plane Detection and Polygonalization in noisy 3D Range Images

Jann Poppinga, Narunas Vaskevicius, Andreas Birk, and Kaustubh Pathak

***Abstract*— A very fast but nevertheless accurate approach for surface extraction from noisy 3D point clouds is presented. It consists of two parts, namely a plane fitting and a polygonalization step. Both exploit the sequential nature of 3D data acquisition on mobile robots in form of range images. For the plane fitting, this is used to revise the standard mathematical formulation to an incremental version, which allows a linear computation. For the polygonalization, the neighborhood relation in range images is exploited. Experiments are presented using a time-of-flight range camera in form of a Swissranger SR-3000. Results include lab scenes as well as data from two runs of the rescue robot league at the RoboCup German Open 2007 with 1,414, respectively 2,343 sensor snapshots. The $36{\cdot}10^6$, respectively $59{\cdot}10^6$ points from the two point clouds are reduced to about $14{\cdot}10^3$, respectively $23{\cdot}10^3$ planes with only about 0.2 sec of total computation time per snapshot while the robot moves along.**

I. Introduction

Mobile robots are increasingly employed in challenging domains where the environment can not anymore be captured with standard 2D maps. There is hence an increasing amount of work on 3D mapping, e.g., [1], [2], [3], [4], [5], [6], [7]. In addition to the data acquisition problem, also dubbed 6D-SLAM [1], one particular challenge for 3D mapping is to keep the amount of data reasonable and to allow efficient processing with well-known techniques, especially from computational geometry. The pure generation of precise 3D point clouds is hence not sufficient, but it is also necessary to extract 3D surface models. This is a well known and extensively studied problem in computer graphics [8], [9], [10], [11], [12], [13]. But these techniques rely on high quality datasets. As pointed out by Hähnel, Burgard and Thrun in [14], the related approaches of using local analysis of the normals of trimeshed data are doomed to fail when using sensors suited for mobile robots. They suggest to use plane fitting and present an approach based on a sweeping plane technique where the assumptions about the environment are exploited.

Here a very fast approach to surface extraction for 3D maps is presented, which is completely independent from any environment constraints. It consists of two stages. In the first, a new plane fitting algorithm is used where the standard mathematical formulation of the problem is revised to an incremental version, which is particular fit for range images. This allows hence efficient region growing, i.e., the computation of the next best fit when an additional data point is considered. Second, the segmented regions are polygonalized exploiting the fact that mobile robots capture the 3D data as a sequence of 3D range images. Results with real world data are presented. Note that Hähnel, Burgard and Thrun in [14] already convincingly showed that 3D laser scanner data is too error-prone for local normal analysis as used in standard approaches from computer graphics. Here, we demonstrate our approach with a sensor type with even significantly higher noise rates than 3D laser scanners.

This is motivated by the fact that 3D laser scanners are slow in the data acquisition. They are typically based on 2D scanners that are driven with some servo-mechanism to cover an additional dimension [15][3][16]. This takes in the order of seconds. Better alternatives with respect to update frequencies are stereo cameras or time-of-flight cameras like the Swissranger SR-3000 [17], which is used for the work presented here. They provide update rates of 25 Hertz and higher, but at the cost of data, which is significantly more error prone than a 3D laser scanner. The SR-3000 is mounted on our *rugged robot* or rugbot (figure 1), a system for rescue robotics applications [18], [19].

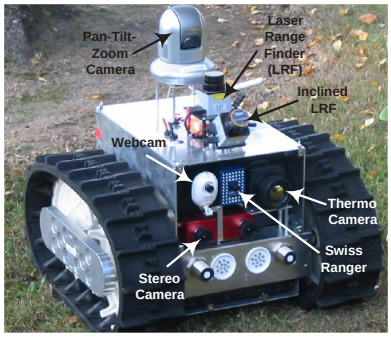


Fig. 1. The autonomous version of a Jacobs *rugbot* - from rugged robot - with some important onboard sensors, especially the swissranger and the stereo cam, pointed out.

II. Plane Fitting as Region Growing

In this section we describe the principle of our algorithm for identifying regions of points that lie on one plane. We extend the work presented in [14] by modifying algorithm and re-formulating the underlying mathematics to an incremental version, which allows a highly efficient implementation. The algorithm proceeds as follows. We take a random point p_1 and its nearest neighbor p_2 from point cloud data PC. This is our initial set of points - region Π (Algorithm 1, Line 4 - 5). Then we try to extend this region by considering points in increasing distance from set Π. Now suppose point p' is such that the distance between it and the region is less than

This work was supported by the German Research Foundation (DFG).

The authors are with the Dept. of Electrical Engineering and Computer Science, Jacobs University Bremen, 28751 Bremen, Germany. (j.poppinga, n.vaskevicius, a.birk, k.pathak)@jacobs-university.de

the distance δ. Then if the mean square error (MSE) to the optimal plane Ω of the region $\Pi \cup p'$ is less than ϵ and if the distance between the new point and the optimal plane Ω is less than γ, then p' is added to the current region Π. We grow this region until no points can be added (Algorithm 1, Line 6 - 11). Afterwards if the region size is more than θ we add it to the set of regions R, else we treat those points as unexplained and add them to the set R' (Algorithm 1, Line 12 - 16). This is repeated until each point from PC is either in R or in R'. Note that each data point can be assigned to several regions as it could lie in the intersection line of two or more planes.

Our approach for regions identification is not sensitive to local noise and therefore it is suitable for the data produced by fast but error-prone 3D range sensors. Actually, the complexity of the model and sensitivity to noise can be controlled by using aforementioned parameters (δ, ϵ, γ, θ). The meaning and impact of parameters is quite intuitive, however selecting the right parameters can play a crucial role in the production of a good model. Fortunately, a single calibration step is sufficient, i.e., the parameters can be tuned for one sensor on a single point cloud and then they can be used from then on without adaptation.

Although the algorithm is easy to understand, a naive implementation of it would lead to a high computational complexity and unreasonable work time even for small point clouds of a size around 10^4. Here, a special incremental approach is presented, which is discussed later in the implementation section II-A.

Algorithm 1 Regions identification

1: $R \leftarrow \emptyset$
2: $R' \leftarrow \emptyset$
3: **while** ($PC \setminus (R \cup R') \neq \emptyset$) **do**
4: `select points` p_1 `and` p_2 in $PC \setminus (R \cup R')$
5: $\Pi \leftarrow \{p_1, p_2\}$
6: **while** (new point can be found) **do**
7: `select nearest neighbor` p' `with distance` $d(\Pi, p') < \delta$
8: **if** (MSE$(\Pi \cup \{p'\}) < \epsilon$ && $d(\Omega, p') < \gamma$) **then**
9: $\Pi \leftarrow \Pi \cup p'$
10: **end if**
11: **end while**
12: **if** (size$(\Pi) > \theta$) **then**
13: $R \leftarrow R \cup \Pi$
14: **else**
15: $R' \leftarrow R' \cup \Pi$
16: **end if**
17: **end while**

First, some background for finding the optimal plane for a point set is introduced. Suppose we have a set of 3D points $p_i = (x_i, y_i, z_i)$, $i = \overline{1, n}$ and we want to find the best fitting plane for this data. The optimal plane can not be simply the regression plane as we have a measurement error in all three coordinates, therefore the orthogonal distances have to be considered. So the goal is to minimize the sum of squared distances to the plane. It can be shown that this is the Eigen vector problem.

The mass center of the given data is defined as:

$$m = \frac{1}{n} \cdot \sum_{i=1}^{n} p_i \tag{1}$$

Using this, the following matrix C is defined:

$$C = \begin{pmatrix} \Gamma_n(x,x) & \Gamma_n(x,y) & \Gamma_n(x,z) \\ \Gamma_n(y,x) & \Gamma_n(y,y) & \Gamma_n(y,z) \\ \Gamma_n(z,x) & \Gamma_n(z,y) & \Gamma_n(z,z) \end{pmatrix}$$

here $\Gamma_n(\phi, \psi) = \sum_i^n (\phi_i - m_\phi)(\psi_i - m_\psi)$.

Then the normal vector of the optimal plane is equal to the Eigen vector n which corresponds to the smallest Eigen value of the matrix C. Suppose the resulting vector is normalized then the bias element - the plane is described by $n_x \cdot x + n_y \cdot y + n_z \cdot z + bias = 0$ - of the optimal plane can be calculated:

$$bias = -(n_x \cdot m_x + n_y \cdot m_y + n_z \cdot m_z)$$

Note that the point m - the mass center (equation 1), is on the optimal plane. The normal vector and the bias element fully define this plane. To find the best fitting plane we need to calculate the Eigen values and the Eigen vectors for the matrix C. This operation is crucial as it has to be performed whenever a new point is investigated, therefore it should be efficient. Therefore, the numerically optimized methods of the GNU Scientific Library (GSL) are used for this purpose [20].

A. Efficient Implementation

As mentioned before, it would be very time consuming to use a naive implementation for region growing algorithm 1. Here, we introduce a minimization of the computational complexity for this.

The important part of the region growing algorithm is the nearest neighbor selection for a current region (algorithm 1, line 7), which is performed every time the inner loop starts. For that we use a priority queue `Q` with the minimum distance on the top. Whenever we add a new point to the region we investigate k unvisited nearest neighbors nb_t, $t = \overline{1, k}$ of this point and if distance $d(p, nb_t) < \delta$ we add nb_t to the priority queue. This allows to extract nearest neighbor for the region just by calling `Q.TOP()`. The question is how to find the k nearest neighbors for a given point. It can be done quite easy with the assumption that the point cloud data is produced from one scan. Then all points are aligned in the grid and can be interpreted as a range image. Then it makes sense to define the 8 neighbors in a range image as k nearest neighbors a the given point.

The last optimization is the most significant one. It deals with the optimal plane calculation. Suppose the matrix C would be calculated from the start every time a new point is added to the region. This would mean that one would need to traverse every point in the current region leading to a huge overhead. Here a way to an incremental update of the matrix C is presented, which takes previous calculations into account. Lets define $S(n)$ to be the sum of n points and $m(n)$

the mass center of those points. Suppose we want to add a new point p_{n+1}. Then the update formula for $\Gamma_{n+1}(\phi,\psi)$ is:

$$\begin{aligned}\Gamma_{n+1}(\phi,\psi) &= \Gamma_n(\phi,\psi)+\phi_{n+1}\psi_{n+1} \\ &\quad -m_\phi(n+1)S_\psi(n+1)+m_\phi(n)S_\psi(n)\end{aligned}$$

here $\phi, \psi \in \{x, y, z\}$. The formula can be derived by simply calculating the difference $\Gamma_{n+1} - \Gamma_n$. The mass center can be easily updated and combined with the updated matrix C, thus providing all information for fast calculation of the best fitting plane. The only thing we need to track is the sum of the points.

The plane mean square error for the given point set can also be calculated incrementally. Here we need to evaluate following expression:

$$MSE(k) = \frac{1}{k}\sum_{i=1}^{k}(n\cdot p_i + d)^2$$

where n is the normal vector of the plane and d is a bias element. Expanding this equation gives us a form which is suitable for incremental calculation:

$$\begin{aligned}MSE(k) &= \frac{1}{k}\sum_{a,b\in\{x,y,z\}}\left(n_a n_b \sum_{i=1}^{k} p_a^{(i)} p_b^{(i)}\right) \\ &\quad +\frac{2\cdot d}{k}\sum_{a\in\{x,y,z\}} n_a S_a(k) + d^2\end{aligned}$$

Here we need to track one more element. This is the product matrix:

$$P_{a,b} = \sum_{i=1}^{k} p_a^{(i)} p_b^{(i)}$$

where $a \in \{x, y, z\}$ and x corresponds to 1, y - 2, z - 3 ($P_{x,y}$ is the same element as $P_{1,2}$). This matrix can easily be updated and therefore we have a fast MSE evaluation.

All optimizations described in the previous section lead to a really fast algorithm. Suppose we have a point cloud data of size n. All operations inside the loops are performed in constant time except the nearest neighbor search for the current region which has logarithmic complexity, as we use a priority queue. Now assume that there are few points lying in the intersection of the planes, which means that most of the points belongs to exactly one region. Then the complexity of the algorithm can be considered to be $O(n \cdot log(n))$. The memory complexity is clearly to be linear - $O(n)$.

III. Polygonalization

The used time-of-flight camera delivers regular point clouds in Cartesian coordinates, which are generated from range images. The latter's structure, i.e. especially the neighborhood information is however still available. Note that the same holds for other common 3D range sensors like stereo cameras. This property is exploited here. The following polygonization algorithm is hence not applicable to general point clouds, but it allows faster processing. The input for the algorithm are the regions found by the plane fitting algorithm. The algorithm works on the 2D coordinates, i.e., the points' positions in the range image. Once a polygonization is found, it is transferred to a global 3D coordinate model.

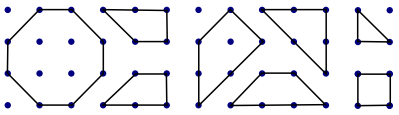


Fig. 2. Some examples for convex polygons on a grid.

The purpose of the algorithm is to find a set of convex polygons covering the same area as the set of triangles that is produced by connecting all triplets of points neighboring in the distance image. This set is supposed to have a triangular decomposition with as few triangles as possible.

1) Decomposition into convex polygon: All convex polygons on a grid are a (possibly degenerated) octagon (see figure 2 for some examples). It can furthermore be observed that it can be decided whether a polygon is convex when scanning it row by row. So an algorithm can be deceived, which reads the input regions row by row and gradually builds convex polygons, starting a new polygon whenever the current one would become non-convex. This obviously has the advantage of linear runtime.

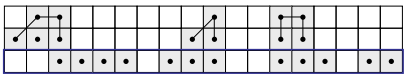
The algorithm: (keep an array of convex polygons P) In each line i:

- mark all polygons in P unused
- find sections of continuous pixels
- for all sections s:
  - is P's leftmost unused element pl close to s?
  - no:
    - ∗ if pl occurs earlier in the line than s, close pl, remove it from P, try s again with the next polygon
    - ∗ if s starts before pl, start a new polygon with s, insert it into P before pl, mark it used

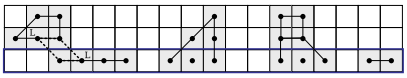
    Either way: insert linking polygon if necessary
  - else: if s can be added to pl without making it nonconvex:
    add s to pl, mark pl used.
  - else: close pl, replace it in P by a new polygon, marked used, started with the last line in pl if possible. If not, start with s and insert linking polygon
  - no more unused elements in P: start new polygons with the remaining sections
- close all left over open polygons

An example iteration of the algorithm is shown in figure 3. As can be seen, there are a number of things to consider as compared to the naive implementation:

- When a polygon cannot be continued with a segment because it would become non-convex, the new polygon must not be started on the current line, but on the line before that to keep the region whole.
- In this case, if the old polygon and the segment differ to much, beginning a polygon with the segments on the



(a) Before processing the last line: three open regions



(b) After processing the last line: two closed regions, four open ones and one link polygon (dashed)

Fig. 3. An example iteration of the algorithm: The polygons in P are depicted on top of the pixels which are part of the region of the range image currently being processed

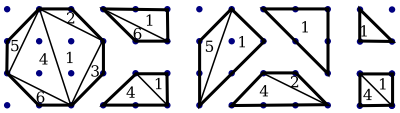


Fig. 4. Triangulation of convex grid polygons. The numbers on the leftmost polygon indicate all possible occurring triangles. On the other polygons, some of these have been left out based on a simple decision tree using at most two tests for equality per triangle. Note that all vertices are points on the grid.

previous and on the current line may not be possible. In this case, a link polygon has to be inserted to keep the region whole. This does not use the beginning and end points of the segments (see dashed polygon in figure 3).

- This may also occur when the segment on the current line was too far away from the nearest polygon to begin with.

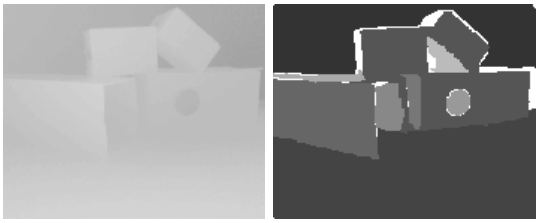
Triangulation of convex polygons on a grid is mostly trivial. The basic mode of polygonization can be seen in the leftmost polygon in figure 4. Depending on how degenerated the octagon is, some of the triangles can be left out (ibid.).

The major advantage of the scan line algorithm is its high speed. In fact, 64 % of the processing time is used for projecting the points of the ideal plane of their region and related pre-processing. Further 30 % are needed for the scan line algorithm and a mere 6 % for the actual polygonization.

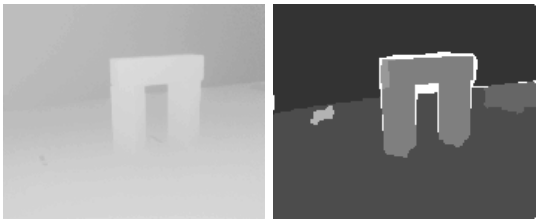
IV. Experiments and Results

Several typical results of the approach on single range snapshots are shown in figure in figures 5 and 6. The input point clouds (shown as range images on the left of figure 5) consist of 25,344 points each. As illustrated in figure 5 on the right, proper planes are fitted into the data. The planes are then turned into proper 3D polygons, which are each shown from two different perspectives in figure 6. As shown in table I, the complete runtimes for processing are in the order of 200 msec, i.e., an update rate of 5 Hz is possible.

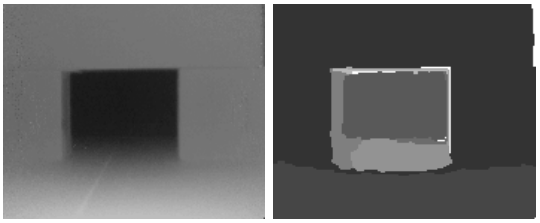
The approach was also tested with large scale data sets for 3D mapping. The data sets were collected at the RoboCup



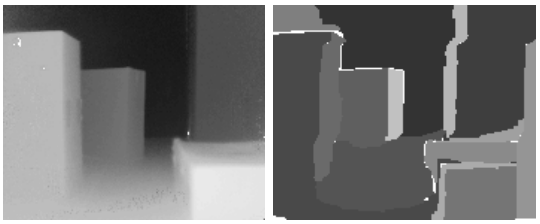
(a) Heaped boxes



(b) An arc



(c) A tunnel



(d) Several boxes

Fig. 5. Input point clouds as distance images (left) and their partition into regions based on 3D plane fitting (right).

German Open 2007[1] (see figure 7. It encompasses recordings of all of Rugbot's sensors, including odometry, gyroscope and SwissRanger time-of-flight 3D camera. The robot is capable of state-of-the-art localization [21], which is used to put the locally extracted polygons into a global 3D model in a straight forward manner. The resulting global model, which is of secondary interest for this paper where we concentrate on the local extraction of surfaces, can be viewed as X3D-model and as flight through movie on the same website as the dataset. The test data consists of two parts, each representing an exploration of the arena lasting roughly between 15 and 20 minutes of robot driving time. The first set contains 1414 point clouds, the second one of 2343. The related global point clouds consist of 35 million, respectively 59 million points.

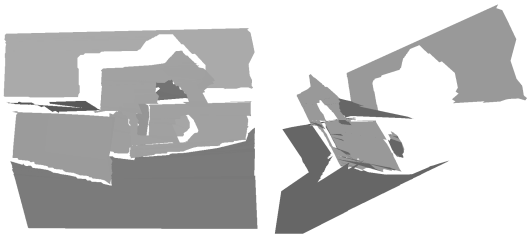
As shown in table II, the approach can generate the surface

[1]The data is available for download: `http://robotics.jacobs-university.de/datasets/`

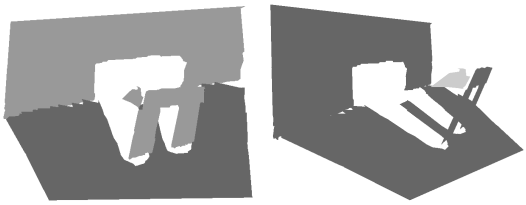
TABLE II

RESULTS ON REAL WORLD DATA (ALL TIMES IN SECONDS, ALL RESULTS ARE AVERAGES)

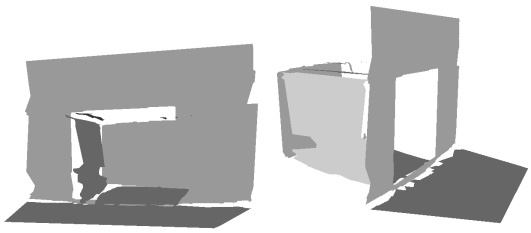
Data Set	#PC	#points	Region grw. reg./PC	Region grw. time	Polygonization #points	Polygonization ratio	Polygonization time	Results total time
run1	1414	35,836,416	10.58	0.195	1944.29	0.077	0.011	0.217
run2	2343	59,380,992	9.80	0.194	1815.35	0.072	0.011	0.215



(a) 1773 points, 9 regions, compression ratio 0.0700



(b) 1095 points, 6 regions, compression ratio 0.0432



(c) 1152 points, 8 regions, compression ratio 0.0455

(d) 2271 points, 14 regions, compression ratio 0.0896

Fig. 6. The generated 3D polygons viewed from two different angles

TABLE I

PERFORMANCE FOR EXAMPLES SHOWN IN FIGURES 5 AND 6

Sample	Compression ratio	Region growing time [s]	Polygonization time [s]	Total time [s]
5(a)	0.0700	0.19	0.01	0.21
5(b)	0.0432	0.21	0.02	0.23
5(c)	0.0455	0.19	0.01	0.21
5(d)	0.0896	0.20	0.01	0.23

Fig. 7. The NIST rescue arena at the RoboCup German Open (left) and two Jacobs robots exploring it (right).

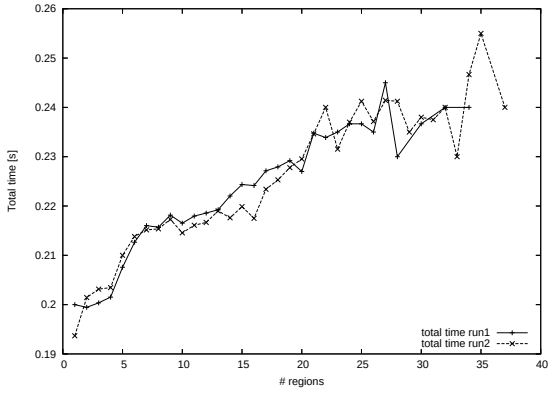


Fig. 8. Processing time in dependence of the number of planes per point clouds

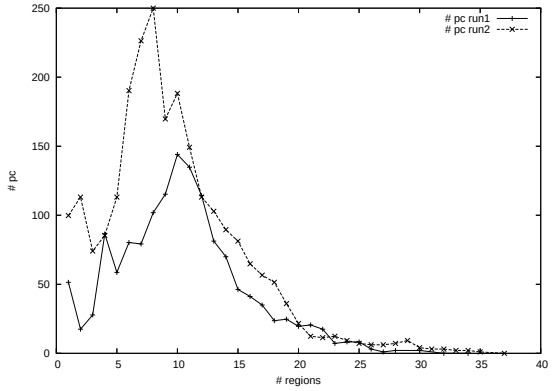


Fig. 9. Number of point clouds with different numbers of planes

models with about 5 Hz. The region growing by incremental plane fitting takes on average only about 200 msec for each of the clouds with about 25,000 points. The following polygonalization turns the regions within only about 11 msec into proper surface models. The two datasets include 1414, respectively 2343 range images leading to global point clouds with 35 million, respectively 59 million points. The exact performance of the approach depends on the number of planes in each range image as illustrated in figures 8. The higher the number of planes in the point cloud, the longer the runtime; note the linear relation. As shown in figure 9, the majority of range images consists of about 10 planes.

V. Conclusions

A fast approach to surface extraction from 3D point clouds was presented. Instead of processing a global model, the sequential nature of the 3D data acquisition on mobile robots is exploited. Concretely, the neighborhood relation of pixels in range images is used. The data used in the presented work is delivered by a Swissranger SR-3000 time-of-flight camera, but the same principle can be applied to range data from other sensors like stereo cameras. A novel incremental plane fitting algorithm is used to segment the points of a range cloud into regions that lie on a common plane. This step is very fast - only about 200 msec for clouds with about 25,000 points - and it is very robust against the high noise rates of the SR-3000. The region growing is followed by a polygonalization, which also exploits the neighborhood relation and turns the regions within only about 11 msec into proper surface models. The speed and robustness of the approach is illustrated with lab scenes and two runs with a Jacobs robot in the rescue league at the RoboCup German Open 2007. The datasets include 1414, respectively 2343 range images leading to global point clouds with 35 million, respectively 59 million points, which were reduced in realtime to surface models with three orders of magnitude less data.

Acknowledgments

The authors gratefully acknowledge the financial support of *Deutsche Forschungsgemeinschaft* (DFG).

Please note the name-change of our institution. The Swiss Jacobs Foundation invests 200 Million Euro in **International University Bremen (IUB)** over a five-year period starting from 2007. To date this is the largest donation ever given in Europe by a private foundation to a science institution. In appreciation of the benefactors and to further promote the university's unique profile in higher education and research, the boards of IUB have decided to change the university's name to **Jacobs University Bremen**. Hence the two different names and abbreviations for the same institution may be found in this article, especially in the references to previously published material.

References

[1] A. Nuechter, K. Lingemann, J. Hertzberg, and H. Surmann, "6d slam – mapping outdoor environments," in *IEEE International Workshop on Safety, Security, and Rescue Robotics (SSRR)*. IEEE Press, 2006.

[2] A. Howard, D. F. Wolf, and G. S. Sukhatme, "Towards 3d mapping in large urban environments," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sendai, Japan, 2004.

[3] H. Surmann, A. Nuechter, and J. Hertzberg, "An autonomous mobile robot with a 3d laser range finder for 3d exploration and digitalization of indoor environments," *Robotics and Autonomous Systems*, vol. 45, no. 3-4, pp. 181–198, 2003.

[4] S. Thrun, D. F. D. Haehnel, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. Whittaker, "A system for volumetric robotic mapping of abandoned mines," in *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, Taipei, Taiwan, 2003.

[5] D. Hähnel, W. Burgard, and S. Thrun, "Learning compact 3D models of indoor and outdoor environments with a mobile robot," *Robotics and Autonomous Systems*, vol. 44, no. 1, pp. 15–27, 2003.

[6] J. Davison and N. Kita, "3d simultaneous localisation and map-building using active vision for a robot moving on undulating terrain," in *IEEE Conference on Computer Vision and Pattern Recognition*, Hawaii, Dec 8-14, 2001.

[7] Y. Liu, R. Emery, D. Chakrabarti, W. Burgard, and S. Thrun, "Using em to learn 3d models of indoor environments with mobile robots," in *18th Conf. on Machine Learning*, Williams College, 2001.

[8] J. Klein and G. Zachmann, "Nice and fast implicit surfaces over noisy point clouds," in *SIGGRAPH 2004, Sketches and Applications*, 2004.

[9] ——, "Proximity graphs for defining surfaces over point clouds," in *Eurographics Symposium on Point-Based Grahics (SPBG'04)*, 2004, pp. 131–138.

[10] N. J. Mitra and A. Nguyen, "Estimating surface normals in noisy point cloud data," in *Proceedings of the nineteenth annual symposium on Computational geometry*. ACM Press, 2003, pp. 322–328.

[11] N. Amenta, M. Bern, and M. Kamvysselis, "A new voronoi-based surface reconstruction algorithm," in *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*. ACM Press, 1998, pp. 415–421.

[12] C. L. Bajaj, F. Bernardini, and G. Xu, "Automatic reconstruction of surfaces and scalar fields from 3d scans," in *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*. ACM Press, 1995, pp. 109–118.

[13] P. Alfeld, "Scattered data interpolation in three or more variables," in *Mathematical methods in computer aided geometric design*. Academic Press Professional, Inc., 1989, pp. 1–33.

[14] D. Hähnel, W. Burgard, and S. Thrun, "Learning compact 3d models of indoor and outdoor environments with a mobile robot," *Robotics and Autonomous Systems*, vol. 44, no. 1, pp. 15–27, 2003.

[15] O. Wulf and B. Wagner, "Fast 3d-scanning methods for laser measurement systems," in *International Conference on Control Systems and Computer Science (CSCS14)*, 2003.

[16] O. Wulf, C. Brenneke, and B. Wagner, "Colored 2d maps for robot navigation with 3d sensor data," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3. IEEE Press, 2004, pp. 2991–2996 vol.3.

[17] R. Lange and P. Seitz, "Solid-state time-of-flight range camera," *Quantum Electronics, IEEE Journal of*, vol. 37, no. 3, pp. 390–397, 2001.

[18] A. Birk and S. Carpin, "Rescue robotics - a crucial milestone on the road to autonomous systems," *Advanced Robotics Journal*, vol. 20, no. 5, pp. 595–695, 2006.

[19] A. Birk, K. Pathak, S. Schwertfeger, and W. Chonnaparamutt, "The IUB Rugbot: an intelligent, rugged mobile robot for search and rescue operations," in *IEEE International Workshop on Safety, Security, and Rescue Robotics (SSRR)*. IEEE Press, 2006.

[20] G. H. Golub and C. F. Van Loan, *Matrix Computations (Johns Hopkins Studies in Mathematical Sciences)*. The Johns Hopkins University Press, October 1996.

[21] A. Birk, S. Markov, I. Delchev, and K. Pathak, "Autonomous rescue operations on the IUB Rugbot," in *IEEE International Workshop on Safety, Security, and Rescue Robotics (SSRR)*. IEEE Press, 2006.