A Toolkit for Automatic Generation of Polygonal Maps – Las Vegas Reconstruction

Thomas Wiemann, Kai Lingemann
Universität Osnabrück

Andreas Nüchter,
Jacobs University Bremen

Joachim Hertzberg
Universität Osnabrück and DFKI Robotics Innovation Center, Osnabrück Branch

Abstract

In this paper we present a new open source software package for automatic generation of polygonal 3D maps from point cloud data for robotic purposes called "Las Vegas Reconstruction Toolkit" [11]. The implemented algorithms focus on minimizing both the computation costs and optimization of the number of polygons in the generated maps. Furthermore, we present two application examples: 6D self localization and scene interpretation.

1 Introduction

Recently, the focus in robotic mapping has begun to shift from planar 2D maps towards 3D environment mapping. 3D maps outperform 2D maps for many purposes, such as obstacle avoidance, object recognition and scene understanding. The introduction of new 3D sensors, especially Microsoft Kinect, gave an additional boost to this development. Nowadays, 3D point clouds are available in several variants and qualities: From noisy sparse clouds delivered by 3D cameras and tilted or rotated 2D lasers scanners to dense clouds gathered from terrestrial laser scanners, which achieve very high resolutions with a count of several (hundred) million points per cloud.

Point clouds do not yield a continuous surface representation, just a possibly very dense sampling. Mathematical descriptions like B-Splines or polygonal meshes are far more suitable representations for 3D scenes for several reasons. With increasing scanner resolution, the amount of collected point cloud data becomes unhandy. These representations can compress the amount of data needed to represent a surface significantly. On the other side of the spectrum there are sensors that deliver comparably sparse point clouds. Suitably fitted continuous surfaces can fill up the space between the points and can thus result in a more complete representation. The Las Vegas Surface Reconstruction Toolkit delivers a set of tools for mesh based surface reconstruction for robotic applications. The software focuses on data compression and geometry preservation. An overview of the implemented methods is given in the remainder of this paper.

2 Related Work

Mesh based approaches create triangle meshes to approximate the scanned surfaces. The de-facto standard is Marching Cubes, introduced by Lorensen et al. [8]. This algorithm sub-divides the scanned volume into cubic cells. For each cell the intersections between the cell edges and the surface are calculated. Pre-calculated surface patterns are then used to generate a local triangle mesh approximation. To interpolate the intersections, implicit continuous surface representations like planes or splines are fitted to the local data using least squares fits [1, 7]. A comprehensive survey of related research is given in [9]. Another approach,"Growing Cells Meshing", uses a neural network with interactive learning to generate the triangle mesh [2].

Marching Cubes based reconstructions are included in several publicly available software packages like Meshlab or the Point Cloud Library (PCL) [12]. The Meshlab Marching Cubes reconstruction uses APSS [5] and RIMLS [10] projections to determine the isosurface of the point sets. These methods can yield good results for the right parameter set, but the computation time needed is very high. Meshlab also includes implementations of surface simplification algorithms [4]. To optimize the model, the edges causing minimal error to the topology are removed iteratively. Since after each edge removal new vertices have to be inserted into the mesh, the initial topology may be altered. Besides the reconstruction, PCL features a greedy triangulation for surface extraction, which delivers a triangulation of a given scene, but not necessarily a continuous surface representation.

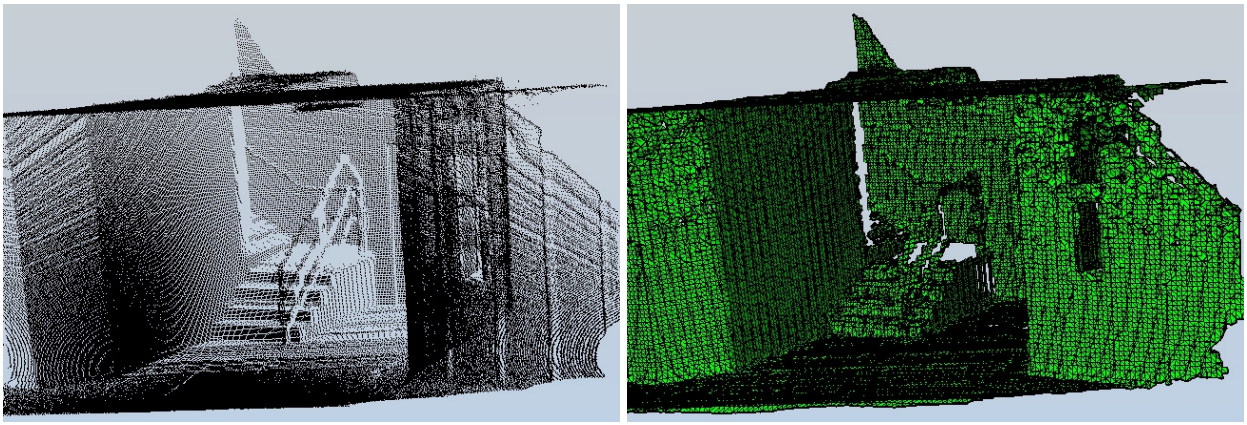

Figure 1: Initial mesh generation. The left picture shows a point cloud taken with a tilted SICK scanner. On the right the generated triangle mesh.

With the Las Vegas Reconstruction Toolkit we present a new software package for polygonal map creation. The integrated functions are implemented with a robotic background in mind, thus they are optimized for execution speed and data compression. The main focus for the reconstruction is to create surface representations that can be used in robotic contexts like localization (via ray tracing) or scene interpretation (using spatial reasoning), rather than other constraints that are discussed in the computer graphics community like triangle quality, well formed topology in meshes, i.e. no t-edges, and so on. Our software primarily exploits the inherent planar scene structure that is found in many robotic applications.

3 The Las Vegas Surface Reconstruction Toolkit

The Las Vegas Reconstruction Toolkit provides an open source C++-library of several algorithms for polygonal map generation. Currently Linux and Mac OS are supported, a Windows port is in development. The library structure is strictly modular, so that new features can be integrated easily. It comes with a detailed API documentation and example programs for surface reconstruction and a viewer. The reconstruction components can roughly be divided into three main components: Surface reconstruction, mesh optimization and texture generation. Functional details together with example data sets are presented in the following sections.

3.1 Surface Reconstruction

The map generation process consists of two steps: Initial mesh generation and mesh optimization. The initial surface reconstruction in Las Vegas is based on Marching Cubes with Hoppe's distance function [7]. The idea of Hoppe's approach is to assign a tangent plane $T(p_i)$ to each data point using a local least squares fit to the k nearest points (k-*neighborhood*). These fit planes are represented by the center of gravity o_i of the k-neighborhood and the surface normal n_i:

$$T(\mathbf{p}_i) = \mathbf{o}_i \cdot \mathbf{n}_i.$$

The signed distance of any spatial point $\mathbf{p}$ is defined as

$$d_T(\mathbf{p}) = s(\mathbf{p}) \cdot d(\mathbf{p}, T),$$

where $d(\mathbf{p}, T)$ is the distance of this point to the nearest tangent plane

$$d = (\mathbf{p} - \mathbf{o}_i) \cdot \mathbf{n}_i$$

and $s(\mathbf{p})$ is the sign of the signed distance function according to the relative position of $\mathbf{p}$. This sign is determined by using the orientation of the normal of the tangent plane. If $\mathbf{p} \cdot \mathbf{n}_i > 0$, then $s(\mathbf{p}) = +1$, otherwise $s(\mathbf{p}) = -1$. The whole process is illustrated in Fig. 2.

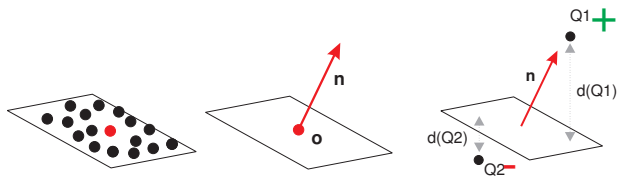

Figure 2: The construction of the signed distance function as described by Hoppe. For each data point, a so called "tangent plane" is calculated by a least square fit to its k nearest points (left). Each plane is defined by its centroid $\mathbf{o}$ and surface normal $\mathbf{n}$ (middle). The signed distance of a query point is the distance of its projection onto the nearest tangent plane $d(\mathbf{q_i})$ and itself. The sign depends on which side of the surface the query point is (right).

Currently two Marching Cubes variants are implemented: Standard Marching Cubes and Marching Tetraeder. Usually the Marching Cube implementation delivers good results, but the generated meshes can show holes even in

dense data. The Marching Tetraeder version delivers better results in such cases, but produces more triangles. Fig 1 displays two exemplary reconstructions that were created from a scan recorded with a tilting SICK LMS 200 laser scanner. The left image shows the captured point cloud, the image on the right displays the initial surface reconstruction using standard Marching Cubes. Note that the generated surface contains far more triangles than necessary. This number is reduced in the mesh optimization step.

For the evaluation of the distance function within the Marching Cubes algorithm, point normals have to be estimated. We implemented normal estimation methods using RANSAC or least squares to fit local planes to the the data points using a set of k nearest neighbors (k-neighborhood). The normal of this plane defines the point normal. To achieve consistence, all normals are flipped towards the scene center. The normal estimation methods can be exchanged. Besides our implementation we also integrated the normal estimation from PCL.

Working with point clouds from rotating 2D laser scanners results in line or arc-shaped artifacts of data points at larger distances. These artifacts will cause, in turn, incorrect results in the fitting process since the orientation of the calculated plane is solely dependent on local noise. To cope with this problem, we integrated a dynamic adaptation of the k-neighborhood to optimize the normal quality. Since higher k values result in extended runtimes, we aim to find a k with a value as small as possible that still allows an accurate approximation. Therefore, we adapt k dynamically to the data density. To detect ill formed k-neighborhoods, we analyze the shape of their bounding boxes. Critical configurations will result in elongate bounding boxes. If we detect such a configuration, k is increased until this shape criterion is fulfilled (cf. Fig 3). Since the laser data suffers from a lot of sensor noise, the resulting normals are still fluctuating to some degree, although their basic alignment is consistent. To reduce this effect, we average all normals with their neighbors. Fig. 4 displays the improvements of our method over non-adaptive estimation with fixed k-neighborhood.

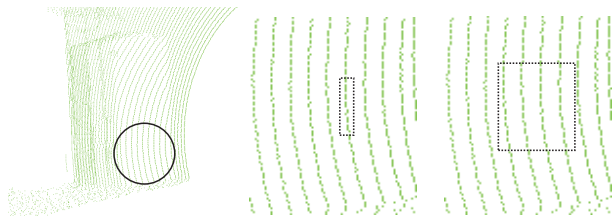

Figure 3: Adaption of the number of nearest neighbors in regions with low point density. If k is too small, all data points will be on a straight line. In this case the fit of a plane will fully depend on the local noise of the data. To ensure a good fit, we analyze the shape of the bounding box.

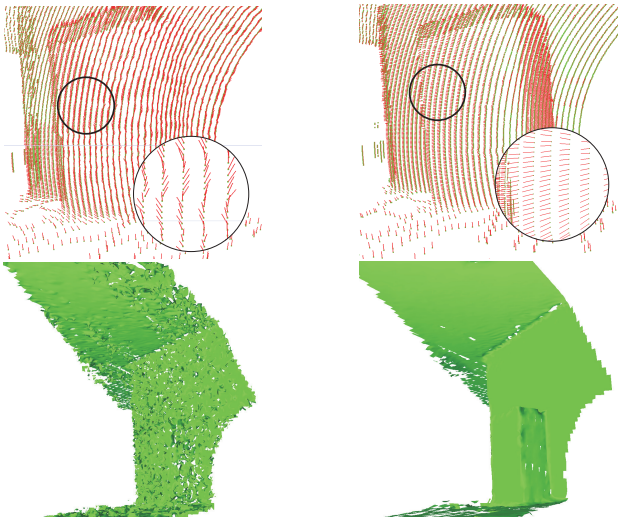

Figure 4: Normal estimation. Top row: Normal estimation results with k-adaption (right) and without k (left). Second row: Influence on the mesh quality. Consistent normals result in connected surfaces (right) while inaccurate estimations produce holes in the triangle mesh (left).

3.2 Mesh optimization

After an initial mesh is created, a set of optimization routines can be applied. The most effective one in terms of compression is the connection of planar patches in the created mesh. Human made environments contain many planar surfaces [3]. In these scenarios, the region-growing approach can help to reduce the number of triangles drastically. The generated meshes are stored in a half edge data structure that allows us to find adjacent normals of any triangle in the mesh in constant time. Region-growing is done by checking if the surrounding triangles of an arbitrarily chosen start triangle have a similar surface normal. As long as the normal of a neighbor triangle does not differ more than a user defined threshold from the start triangle, a new search is started recursively from this triangle. This process is carried on, until a bend in the surface is detected (cf. algorithm in Fig. 6). The edge between such tow triangles marks a boundary of a planar region. All these contour edges are saved and fused using a line following procedure to create an optimal polygonal representation of such a region.

Practically, the normal threshold for planar fusion has to be chosen quite big (up to $15°$), so noise in the point cloud data can result in unevenness in the generated surface. Therefore we optimize the extracted planes by shifting all triangle vertices into the common plane. After this process, the contours of the extracted planar regions are tessellated to reduce the number of triangles. The effects of this optimization procedure on the mesh presented in Fig. 1 are shown in Fig. 5. For applications where no triangulation is needed, the tesselation step can be skipped.

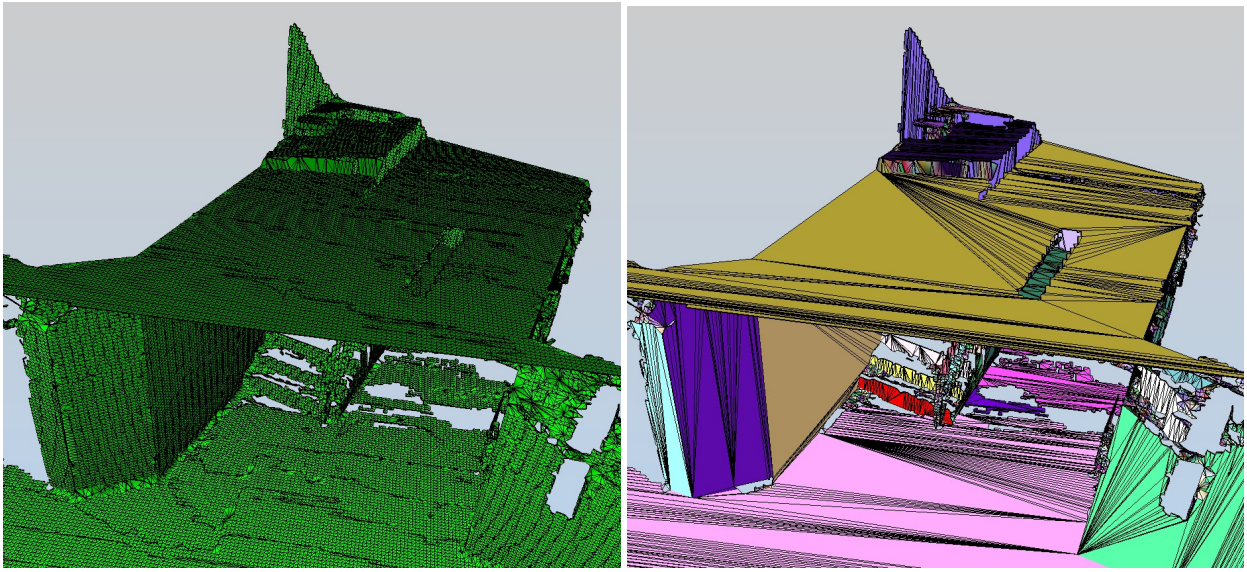

Figure 5: Planar mesh optimization. The left picture shows the modified mesh, after all vertices of connected regions were moved into their common plane. On the right the extracted regions and the newly triangulated mesh.

Figure 6: The mesh simplification algorithm. Faces within the mesh that have similar surface normals are detected. The border edges of these planar areas are fused to polygons.

```
function SIMPLIFY
    for all faces do
        current face ← visited
        FUSE(current normal, current face, currentList)
        borderLists ← currentList
        CREATEPOLYGON(border list)
        currentList ← empty
    end for
end function

function FUSE(start normal, current face, list of borders)
    current face ← visited
    for all neighbors of current face do
        angle ← start normal · neighbor normal
        if angle < ϵ and neighbor not visited then
            FUSE(start normal, neighbour, listOfBorders)
        else
            list of borders ← border edge to neighbor
        end if
    end for
end function
```

Besides the planar optimization we have included functions to filter artifacts from outliers in the laser scanner data. We call this function "Remove Dangling Artifacts" (RDA). The RDA algorithm is also based on region growing that disperses over all faces connected by a shared edge. The number of all connected faces represents the size of the artifact and can be used to determine whether the artifact is large enough to be of importance and can remain in the mesh. Regions that are too small are deleted from the mesh, cf. Fig. 8.

Another optimization feature is hole filling. The hole filling algorithm performs a contour tracking on the mesh and collects all holes up to a certain size which is given by the number of edges in the contour. In a second step the edges of each hole are collapsed using an edge collapse operation until there are only three edges left per hole. The remaining triangle holes are closed by adding new faces to the mesh which close those holes. Fig. 7 displays an application example.

4 Performance and Accuracy

Since an error metric for polygonal maps is difficult to define, we present some key figures for our maps. The running times and compression rates for two example data sets are shown in Table 1. The performance bottleneck in the reconstruction process is the normal estimation, whose performance in turn depends on the speed of the used k-neighbor search algorithm. We integrated several kd tree implementations (flann, STANN, nabo and ANN). In our experiments, flann had the best execution performance. The results in Table 1 were achieved using this library for k-search.

Table 2 shows the running times for different edge removal techniques. In most cases mesh simplification is done by iteratively collapsing the edges in a mesh, whose removal causes the lowest error to the stored geometry. To identify such edges, different error metrics exist. The most used are the quadric error metrics presented by Garland and Heckbert [4]. To compare our approach with such methods, we iteratively removed edges from the initial mesh, until a compression ratio similar to our method was achieved.

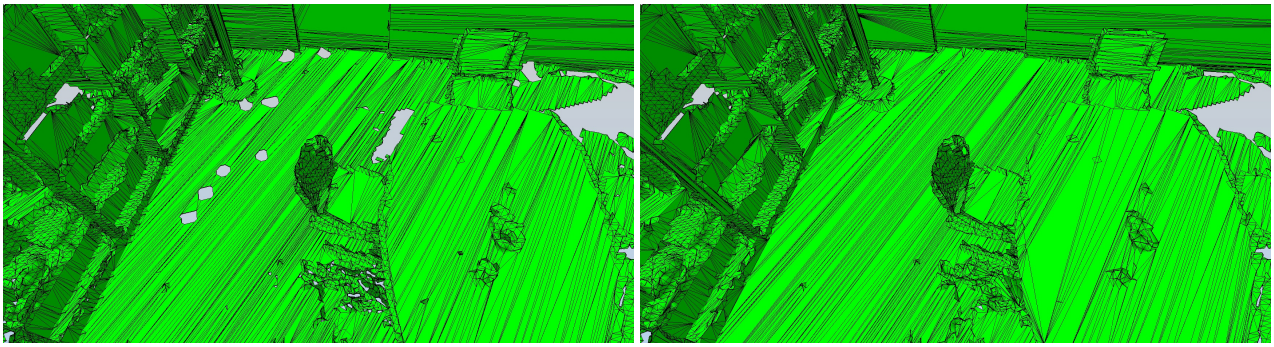

Figure 7: Exemplary result of the "Hole Filling" function. Before (left) and after (right) application.

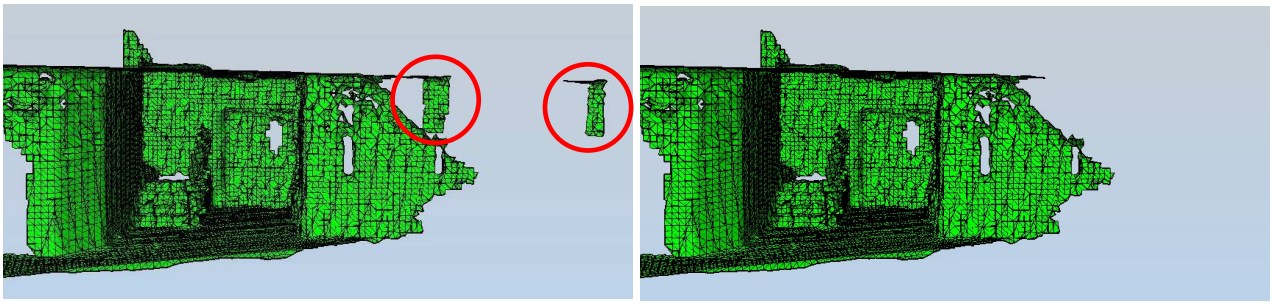

Figure 8: Exemplary result of the "Remove Dangling Artifacts" function. Non-connected regions up to a given size (left, marked red) are deleted from the reconstruction.

Table 1: Running time and compression rates for two different data sets. One was the single scan shown in Fig 1. The other was a registered data set consisting of 10 single scans.

	#Points	Initial Faces	#Polygons	Time
1	271,288	66,374	23,670	1.21 s
10	1,834,599	44,740	3,029	2.25 s

Table 2: Run time comparison between our optimization algorithm and other mesh reduction methods (removal of the shortest edges and using quadric error metrics [4]).

	Map Gen.	Shortest	Quadric	Compression
1	0.25 s	1.02 s	2.37 s	65 %
10	0.47 s	1.72 s	2.35 s	39 %

To evaluate the accuracy of the generated map, we have compared the reconstructed geometry shown in Fig. 9 with manual measurements in the original environment of ceiling height, wall width and height of a room and door width. Furthermore we inspected the parallelity of the floor, ceiling and walls in the reconstruction. The tests were done using the vertex distance measurement tool in Meshlab. The results are given in Table 3. The reconstructed values show a deviation of about 3 to 4 cm from the original values due to interpolation errors and noise in the original scans. Compared to the size of the mapped area these inaccuracies are negligible.

Table 3: Comparison of the original and reconstructed geometries.

	Ceiling	Width	Depth	Door Width
Original	2.99 m	5.89 m	7.09 m	0.94 m
Reconstr.	2.96 m	5.85 m	7.06 m	0.90 m

5 Practical Usability of the Generated Maps

We have tested the usability of the polygonal maps created with the Las Vegas toolkit for localization in different contexts. One example is the LiSA (Life Science Assistant) robot. The used robot is equipped with several laser scanners mounted in different orientations. All these sensors use the same polygonal map. The localization is done using MCL localization. The sensor models are generated via raytracing in the polygonal map. Fig. 9 demonstrates the used method and the improvement of the self-localization. More details can be found in [13].

Another successful application was real time 6D pose tracking using a PMD time-of-flight camera. For this experiment, a pinhole camera model and ray tracing was used to generate an expectation of the incoming sensor data. The expected data was matched with the real sensor data using ICP to determine the transformation between the expected pose and the real pose to correct the initial estimation. More details and preliminary results in [14].

Besides localization we used the extracted planes for scene

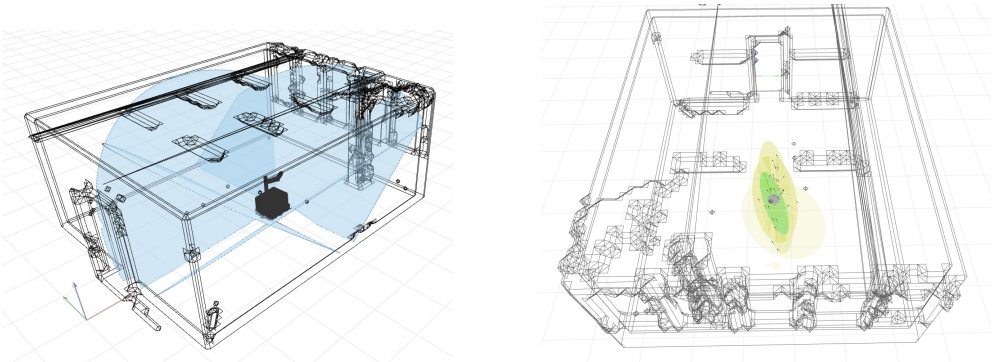

Figure 9: Application example: Ray tracing for localization in polygonal maps. For this purpose, only the extracted polygons were used, not the retesselated map. The used robot us equipped with several laser scanners mounted at fixed angles and pointing at different directions. A Ray tracing technique is used to calculate a sensor model (left). The right figure shows the improvement of the self localization. The yellow area marks the estimated pose error without the use of the tilted scanners. The green area shows the result using the additional information from these scanners. The localization improves considerably [13].

interpretation [6]. Using an ontology of spatial relationships between planes together with an analysis of their size and orientation, we were able to detect furniture in the scanned scenes and replaced them with CAD-models (cf. Fig. 10).

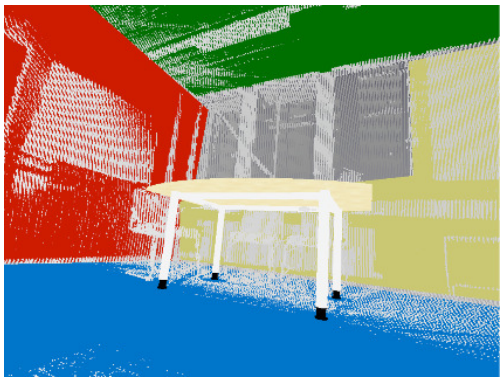

Figure 10: CAD objects that were replaced in the point clouds using spatial analysis of the planes that were extracted by the Las Vegas Toolkit [6]

6 Conclusion

This paper has presented a novel software library to extract polygonal maps from 3D point clouds. Such point clouds are common output of 3D mapping systems. As several approaches from the field of computer graphics cannot be used in robotics, due to sensor noise and time constraints, we developed and implemented robust methods to create practically usable 3D polygon maps. The mapping algorithm exploits the inherent scene structure of indoor environments that typically contain a large number of planar surfaces.

References

[1] M. Alexa, J. Behr, and D. Cohen-Or et al. Computing and rendering point set surfaces, 2002.

[2] H. Annuth and C.-A. Bohn. Smart growing cells. In *Conference on Neural Computation*, 2010.

[3] R. B. Fisher. Applying knowledge to reverse engeniering problems. In *Geometric Modeling and Processing*, 2002.

[4] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. *Computer Graphics*, 31(Annual Conference Series), 1997.

[5] G. Guennebaud and M. Gross. Algebraic point set surfaces. In *ACM SIGGRAPH 2007 papers*, 2007.

[6] M. Günther, T. Wiemann, S. Albrecht, and J. Hertzberg. Model-based object recognition from 3d laser data. In *German Conference on Artificial Intelligence KI 2011*, 2011.

[7] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. *Computer Graphics*, 26(2), 1992.

[8] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM SIGGRAPH*, 1987.

[9] T. S. Newman and H. Yi. A survey of the marching cubes algorithm. *Computers & Graphics*, 30(5), 2006.

[10] A. C. Öztireli, G. Guennebaud, and M. Gross. Feature preserving point set surfaces based on non-linear kernel regression. *Computer Graphics Forum*, 28(2), 2009.

[11] Las Vegas Surface Reconstruction. `http://www.las-vegas.uni-osnabrueck.de`.

[12] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *Conference on Robotics and Automation*, 2011.

[13] S. Stiene and J. Hertzberg. Virtual range scan for avoiding 3d obstacles using 2d tools. In *Conference on Advanced Robotics*, 2009.

[14] J. Wülfing, J. Hertzberg, K. Lingemann, A. Nüchter, S. Stiene, and T. Wiemann. Towards real time robot 6d localization in a polygonal indoor map based on 3d tof camera data. In *Symposium on Intelligent Autonomous Vehicles*, September 2010.