Unconstrained 3D-Mesh Generation Applied to Map Building

Diego Viejo and Miguel Cazorla

Robot Vision Group
Departamento de Ciencia de la Computación e Inteligencia Artificial
Universidad de Alicante
E-03690, Alicante, Spain
Phone +34 96 590 39 00, Fax +34 96590 39 02
{dviejo,miguel}@dccia.ua.es

Abstract. 3D map building is a complex robotics task which needs mathematical robust models. From a 3D point cloud, we can use the normal vectors to these points to do feature extraction. In this paper, we will present a robust method for normal estimation and unconstrained 3D-mesh generation from a not-uniformly distributed point cloud.

1 Introduction

Map building is a main task in robotics. This task estimates a map from sensor data (sonar, laser, 3D data, ...). Our work is focused in 3D map building. We use data from a stereo camera and our goal is to obtain geometric primitives (such as planes, cylinders, and so on) from this data.

Some previous approaches has been proposed to 3D map building. Some of them ([1], [2]) use a laser pointing upwards and taking data every small interval of time. The triangulation between points turns out fast and easy. In [3] an additional step is applied in order to reduce the error between two consecutive steps. However, our main problem is the use of a stereo camera. In environments with low texture the camera does not provide points and the point set is not uniform. So, we have to address the "holes" inside data. We propose to use a (robust) triangulation algorithm which takes into account these holes from data.

Our work will be guided from some assumptions. We assume that our robot will work in a indoor (structured) environment. For this reason, we can guess that the environment is formed from geometric primitives, like (mainly) planes or cylinders. Extracting these primitives might help in the overall process. In order to obtain these primitives we propose to use the normals obtained from a triangulation of the point set. First, we apply a 3D pose registration method, so that the odometry error is reduced. Then, with the rectified point set, we triangulate it taking into account the limitations of our sensor, which is our main contribution. Finally, we calculate the normals from the triangulation in a robust way.

The rest of the paper is organized as follows: In Section 2, we describe the experiment set, the hardware used and the method for pose registration. Section 3

A. Sanfeliu et al. (Eds.): CIARP 2004, LNCS 3287, pp. 241–248, 2004.

describes the method developed for the estimation of normals. Section 4 introduces the triangulation method which takes into account the non-uniformity of data and, finally, some conclusions are drawn in Section 5.

2 Obtaining a 3D Point Set from a Stereo Camera

The experiment was realized in the faculty of Economics Sciences at the University of Alicante. In Figure 1 (right) we show the building plant and the approximate path performed by our robot. Our robot Frodo (see Figure 1 left) is a Magellan Pro from RWI with a tri-stereo Digiclops camera (see [4] for more details).

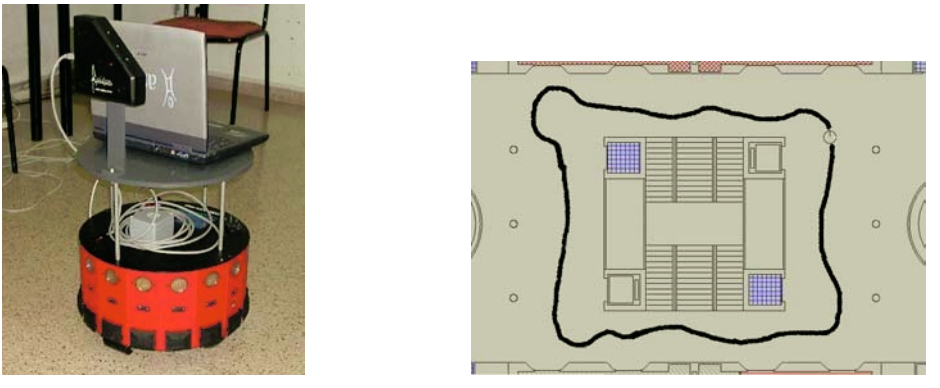

Fig. 1. Left: Robot and stereo camera used in the experiment. Right: Building plant where the experiment was realized and path followed by the robot.

The camera provides us a 3D point set and the intensity value at each point. While the robot is moving it takes 3D images approximately every 1 meter or 15 degrees. We have the odometry information provided by the encoders of the robot. This information yields an approximation of the actions realized by the robot $A = \{a_1, a_2, \ldots, a_n\}$, where each action $a_i = (x_i, y_i, \theta_i)$. We can obtain a reconstruction of the environment (map building) using this odometry information. To do that, provided that the coordinates of the points are local to every position of the robot, we must apply a transformation to the coordinates of the points using such information.

Figure 2 (left) shows a zenithal view of the plant of this environment. Note that points both from the floor and the roof have been removed which allows to observe accurately the reconstruction. We can observe that certain problems existing in the reconstruction come from odometry errors. In order to minimize this we can use a pose registration method applied to two consecutive point sets. The classic algorithm for 3D pose registration is ICP: Iterative Closest Point [5]. ICP calculates, iteratively, the best transformation between two point sets in two

steps. The first one, given an initial transformation, it calculates correspondences between points using a distance criterion after applying the transformation. In the second one, the latter correspondences are used to estimate the best transformation in terms of minimum squares. The latter transformation is retained for a new first step. ICP ensures convergence to a local minimum. A modified algorithm [6] provides better results in terms of efficiency and it reduces the odometry error. This algorithm rather than determining the correspondences by one-way form, it uses random variables to estimate the probability that a given point in one set matches another one in the other set. Figure 2 (right) shows the result of applying such algorithm.

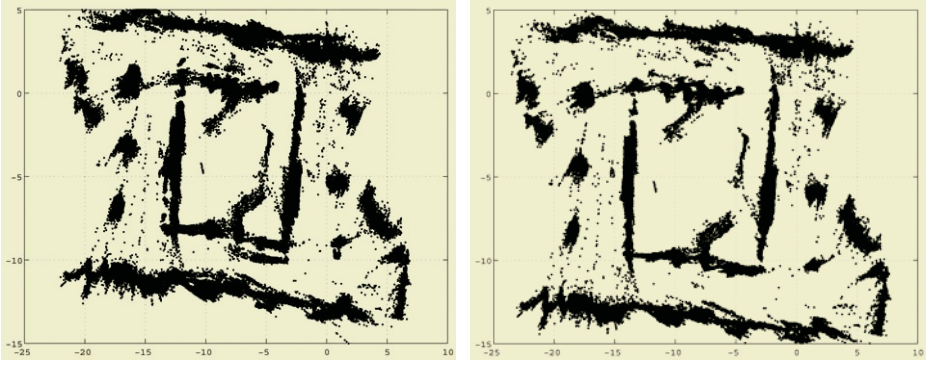

Fig. 2. Zenithal view of the floor of the reconstructed environment. Left: using odometry. Right: Using ICP-EM.

3 Normal Classification

The aim of this work is to extract geometric primitives from a cloud of 3D points, which have been obtained from the environment in the latter step. As we set before, we assume that our environment can be described from planar surfaces. We estimate the normal vectors of the underlying surfaces on each of the 3D points extracted from the environment. A later geometric primitive recognition will depend on the robustness of this normal estimation step.

Normal estimation relies on the method described in [7]: given a triangle mesh it returns the normal at each vertex of this mesh. The basic idea is to select a region from the mesh around a vertex. This region is specified in terms of geodesic neighborhood of the vertex. Each triangle T_i in this neighborhood casts a vote N_i which depends on the normal vector of the plane that contains the triangle. To avoid the problem that normals with opposite orientation annihilate each other, N_i is represented as a covariance matrix $V_i = N_i N_i^t$. Votes are collected as a weighted matrix sum $\mathbf{V}_v$ with

$$\mathbf{V}_v = \sum w_i V_i = \sum w_i N_i N_i^t \tag{1}$$

$$w_i = \frac{A_i}{A_{max}} \exp(-\frac{g_i}{\sigma}) \tag{2}$$

where A_i is the area of T_i, A_{max} is the area of the largest triangle in the entire mesh, g_i is the geodesic distance of T_i from v, and σ controls the rate of decay. Figure 3 shows the effects of the rate of decay. The lower σ the higher the smoothness of the normals obtained.

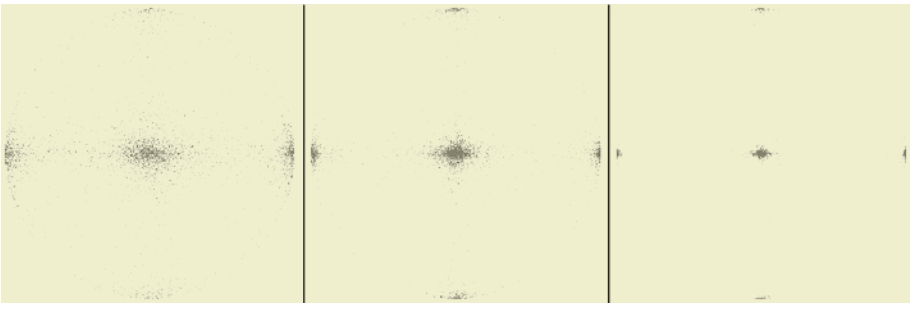

Fig. 3. Normals projection on a semi-sphere using several rates of decay, from left to right: $\sigma = 5/3cm, \sigma = 5/9cm$ and $\sigma = 5/15cm$.

With these equations we obtain knowledge about the variance instead of loosing information about normal sign orientation. This knowledge allows us to draw conclusions about the relative orientation of the vertex. So, we can decompose $\mathbf{V}_v$ using eigen-analysis and then classify vertex v. Since $\mathbf{V}_v$ is a symmetric semidefinite matrix, eigen-decomposition generates real eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$ with corresponding eigenvectors E_1, E_2, end E_3. With this information we can define the saliency map [8] as:

$$\begin{aligned} S_s &= \lambda_1 - \lambda_2, \\ S_c &= \lambda_2 - \lambda_3, \\ S_n &= \lambda_3 \end{aligned} \tag{3}$$

And so, we propose the following vertex classification scheme for the eigenvalues of $\mathbf{V}_v$ at each vertex:

$$\max S_s, \varepsilon S_c, \varepsilon\eta S_n = \begin{cases} S_s : \text{surface matches with normal } N_v = E_1; \\ \epsilon S_c : \text{crease junction with tangent } T_v = E_3; \\ \epsilon\eta S_n : \text{no preferred orientation} \end{cases} \tag{4}$$

where $0 \leq \varepsilon \leq \infty$ and $0 \leq \eta \leq \infty$ are constants that control the relative significance of the saliency measures. Given such a classification we are interested in vertices $N_v = E_1$, and we can filter the rest. Using vector voting, we can calculate the normal of a 3D point from a triangle mesh. However, prior to obtain normals we must compute the triangulation. Our approach is addressed to generate a Delaunay triangulation from 3D points, which is explained in the next section.

4 Triangle Mesh Generation

The Delaunay triangulation of a set of 3D points is a well-known topic in the literature. Many of these methods [9, 10] obtain the triangulation for a 3D solid. Nevertheless, our problem cannot be solved by these approaches since we cannot assume that our set of points comes from a 3D solid. Mainly, our data comes from walls of corridors and rooms where the robot moves. Walls are, in fact, 2D surfaces in a 3D space. Others approaches [12, 13] are used to build topographic maps from a piece of terrain. These methods also build a mesh from a 2D surface in a 3D space, but really what they do is to project the 3D points over a horizontal plane, and so, final calculation is a 2D Delaunay triangulation.

Our approach wants to resolve a Delaunay triangulation of clouds of 3D points without either any consideration about sensor geometry or data coming from a closed volume. On the other hand, due to the nature of the problem that we want to solve, we introduce a constraint about the maximum size of the triangles in the mesh in order to maintain openings between walls or between a wall and any nearby objects from the environment. To solve this, we use a *divide & conquer* (D&C) schema that is based on the recursive partition and local triangulation of the point set, follow by a merging phase where the resulting triangulations are joined.

In general, D&C methods work well in 2D spaces, but nevertheless, to do the same in 3D space is not a simple task because merging is simple in 2D [11] but hard to design in more than two dimensions. DeWall [14] proposed an interesting triangulation method that uses a D&C strategy by reversing the order between the solutions of sub-problems and the merging phase. Instead of merging partial results, it applies a more complex dividing phase which partitions a set and builds, as first step, the merging triangulation.

In order to build our mesh (see figure 4), we use this idea and then we incorporate the constraints imposed by the problem. First, DeWall uses a tetrahedron as geometric basic entity to build a mesh from a set of 3D points that form a closed volume. The supposition of closed volume is not fulfilled in our case and it would be a source of problems due to the high noise that we must handle. For this, we are going to use the triangle as geometric basic entity to calculate the triangulation, in spite of the tetrahedron. This idea arises from the fact that we are triangulating points from 2D planes inside a 3D environment and, in general, to build a triangulation from 2D points the geometric basic entity is the triangle. In addition, we have to consider the fact that the triangle size constraint has a consequence: it might happen that certain parts of the space remain unconnected and, therefore, not be triangulated. For this reason, before the triangulation process, we compute clusters of connected points which will be the input to this triangulation process.

Our mesh generation approach is as follows: First, we use a plane α to split the space in two half spaces. Then, we compute the merging triangulation Σ_α using the splitting plane α. The technique used to build the Σ_α is a slight variation on an incremental construction algorithm: a starting triangle is founded and then Σ_α is built by adding a new triangle at each step. To find the first triangle we

```
function Triange_Builder (P: point_set, L_α: side_list): triangle_list
    var f: side; , L_l, L_r: side_list;
        t: triangle; Σ: triangle_list; α: splitting_plane;
    begin
        if L_α = Ø then
            t:=FindFisrtTriangle(P, α);
            Σ := Σ ∪ t;
            for each f': f'∈ Sides(t) do
                if IsIntersected(f', α) then Insert(f', L_α);
                if InHalfSpaceL(f', α) then Insert(f', L_l);
                if InHalfSpaceR(f', α) then Insert(f', L_r);
        while L_α! = Ø
            f:=Extract(L_α);
            t:=FindTriangle(f, P);
            if t ≠ null then
                Σ := Σ ∪ t;
                for each f': f'∈ Sides(t) AND f' ≠ f do
                    if IsIntersected(f', α) then Insert(f', L_α);
                    if InHalfSpaceL(f', α) then Insert(f', L_l);
                    if InHalfSpaceR(f', α) then Insert(f', L_r);
        /*Recursive Triangulation*/
        if L_l ≠ Ø then Σ := Σ ∪ Triangulator(P, L_l);
        if L_r ≠ Ø then Σ := Σ ∪ Triangulator(P, L_r);
        Triange_Builder:=Σ;
    end.
```

Fig. 4. Algorithm Triange_Builder computes 3D Delaunay triangulation.

select the nearest point p_1 to plane α. Then, we select a second point p_2 such that it is the nearest point to p_1 on the other side of α. From p_1 and p_2 we search the point p_3 such that the *circum-circle* around p_1, p_2 and p_3 has the minimum radius r_i. The center of the *circum-circle* are extracted and we can compute a sphere with center c_i and radius r_i. Finally, to accept p_3 as the point which complete the triangle, it has to fulfill a pair of conditions. First, the Delaunay condition: no one point is inside the sphere; second, point p_3 is near enough from p_1 and p_2 to accomplish the triangle size constraint.

The rest of Σ_α is built from the first triangle. We label each triangle side as it lies completely contained in one of the two half-spaces (left or right) defined by α or it is intersected by the plane. We use three lists L_l, L_r and L_α to insert the triangles sides depending on their label. We extract a side from L_α and search for a new triangle, but now we must consider that this side comes from an exiting triangle. Each side of a triangle has a plane β which is perpendicular to the triangle. β divides the space into two half-spaces, therefore, we just search the next triangle in the valid half-space. The sides of the new triangle are now inserted in the corresponding list.

In Figure 5(left) we show a detail of a 3D point set. Figure 5(Center) shows the result of applying our proposed algorithm to generate the triangulation of

the points: unconnected zones appear due to the absence of information and the constraint of the maximum size of triangle imposed. In Figure 5 (right) normal vectors calculated from the triangle mesh are shown.

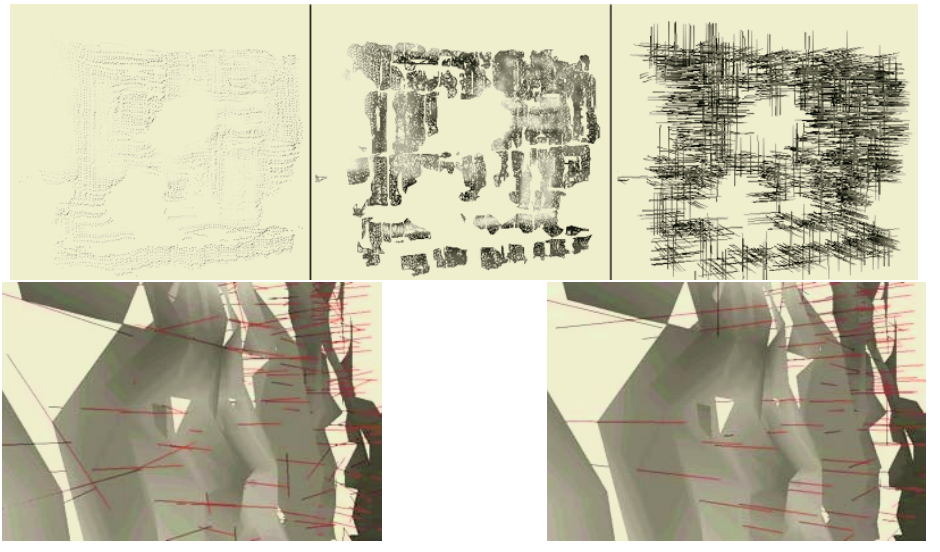

Fig. 5. Detail of normal estimation process. Top: Left: source 3D points. Center: triangle mesh. Right: resulting normals. Bottom: Left: normals before normalization. Right: normals after normalization.

5 Conclusions and Future Work

The line of work that we are following tries to obtain a 3D map of an environment. In this paper we improved the method of normals estimation, endowing it with higher robustness. For this, it has been necessary to develop a method to build a mesh of triangles from the point set. The method proposed obtains a triangulation from the point set, taking into account that our data are not uniformly distributed. The work is completed by the previous phase of reconstruction of the environment.

As continuation of this work we try to address the SLAM problem in order to improve the reconstruction of the 3D map. In addition we want to build a robust system of constraints that allows us to obtain geometric primitives (planes, cylinders, boxes, and so on) from the normals obtained.

Acknowledgments

This work has been supported by grant TIC2002-02792 funded by *Ministerio de Ciencia y Tecnología* and FEDER.

References

1. S. Thrun, W. Burgard, and D. Fox: A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. In Proc. of the IEEE International Conference on Robotics and Automation (ICRA) (2000)
2. S. Thrun and D. Hähnel and D. Ferguson and M. Montemerlo and R. Triebel and W. Burgard and C. Baker and Z. Omohundro and S. Thayer and W. Whittaker: A System for Volumetric Robotic Mapping of Abandoned Mines. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) (2003)
3. H. Surmann and A. Nuchter 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. **45** (2003) 181-198
4. J.M. Saez and F. Escolano: A Global 3D Map-Building Approach Using Stereo Vision. In Proceedings of IEEE International Conference on Robotics and Automation (ICRA) (2004)
5. P. Besl and N. McKay: A method for registration of 3-d shapes. IEEE Trans. On Pattern Analysis and Machine Intelligence, **14** (1992) 239-256
6. M. Cazorla and B. Fisher: Characterizing local minima in 3d registration methods. Not yet published, (2004)
7. D. L. Page, Y. Sun, A. F. Koschan, J. Paik and M. A. Abidi: Normal vector voting: crease detection and curvature estimation on large, noisy meshes. Graphical Models, Special Issue on Larte Triangle Mesh Models, **64** (2002) 199-229
8. G. Medioni, M. Lee, and C. K. Tang. A Computational Framework for Segmentation and Grouping, Elsevier Science Ltd., Amsterdam (2000)
9. E. Mücke: A Robust Implementation for Three-dimensional Delaunay Triangulations. In Proceedings of the 1st International Computational Geometry Software Workshop (1995)
10. K. Hormann and M. Reimers: Triangulating Point Clouds with Spherical Topology. Curve and Surface Design (2003) 215-224
11. D. T. Lee and B.J. Schchter: Two algorithms for constructing a Delaunay triangulation. Int. J. of Computer and Information Science, **9** (1980) 219-242
12. M. de Berg, M. van Kreveld, M. Overmars and O. Schwarzkopf: Computational Geometry, Algoritms and Applications. Ed. Springer (1991) 181-183.
13. G. Petrie and T.J.M Kennie: Terrain modelling in Survey and Civil Engineering. Computer Aided Design, **19**, number 4 (1987).
14. P. Cignoni, C. Montani and R. Scopigno: DeWall: a fast divide and conquer Delaunay triangulation algorithm. Ed. Computer-Aided Design **30** (1998) 333-341