Real-Time Estimation of Planar Surfaces in Arbitrary Environments Using Microsoft Kinect Sensor

Francesco Castaldo[1], Vincenzo Lippiello[2],
Francesco A.N. Palmieri, and Bruno Siciliano

[1] Seconda Universitá degli Studi di Napoli, Dipartimento di Ingegneria Industriale e dell'Informazione, Via Roma 29, 81031 Aversa (CE), Italy
[2] Universitá degli Studi di Napoli Federico II, Dipartimento di Ingegneria Elettrica e Tecnologie dell'Informazione, via Claudio 21, 80125 Napoli, Italy

Abstract. We propose an algorithm, suitable for real-time robot applications, for modeling and reconstruction of complex scenes. The environment is seen as a collection of planes and the algorithm extracts in real time their parameters from the 3D point cloud provided by the Kinect sensor. The execution speed of the procedure depends on the desired reconstruction quality and on the complexity of the surroundings. Implementation issues are discussed and experiments on a real scene are included.

Keywords: Microsoft Kinect, Real-time 3D Reconstruction, Planes extraction, Point Cloud.

1 Introduction

Real time reconstruction of the geometry of a unknown environment is a topic that is recently subject of very active research for the design of truly autonomous robots. The availability of low cost sensors (such as cameras), that are still able to guarantee an acceptable level of precision, is giving rise to many projects that aim at designing autonomous mobile robots on wheels or helices (as quadricopters).

In robotics, SLAM (Simultaneous Localization And Mapping) [1] [2] is a large area of research in which an ever growing number of algorithms are derived to fuse data from different sensor modalities with the objective of constructing an internal map of the environment and for self-localization. To control navigation and task execution is much simpler when the map of the surroundings is known to the robot. Unfortunately, unless we deal with very specialized applications, the general problem of operating in an unknown and unstructured environment remains the challenge of much of the current research effort [2].

The difficulties are also related to the computational complexity of any algorithm devised for this purpose. In a really autonomous system all the computations required to localize the robot and to reconstruct the map of the environment, must be done on board. Often the robot may have a limited processing

A. Petrosino (Ed.): ICIAP 2013, Part II, LNCS 8157, pp. 552–561, 2013.

power and to operate in real-time, any algorithm implemented on board must have a limited computational complexity.

In this work we have focused on the reconstruction of what could be considered to be the "most important" objects present in the environment during navigation. Our reference robot is a flying quadricopter for which necessary relevant information are: floor and roof recognition; height from the ground; position of significant obstacles, etc. All this data is crucial for navigation and collision avoidance. Furthermore the algorithms have to run on low-class processors. Our description of the environment is based on planes [3] that are detected and parametrized in real time. The focus of our work is on algorithms that can provide a reasonable level of accuracy within stringent computational constraints.

The sensor we have used in this work is the Microsoft Kinect [9] [10] camera, a peripheral born as an accessory of the Microsoft Xbox360 console, but increasingly used in robotics. The Kinect represents a good compromise between precision, easiness of use and cost.

In Section 2 we provide a rapid survey on the state of the art regarding our problem, introducing the elementary steps of our algorithm. In Section 3 we present in detail the algorithm itself. Section 4 contains experiments and results from data acquired from real and complex scenes framed by the sensor. The last section is about general considerations and indications for future developments.

2 Planes Reconstruction

3D scanning systems usually returns to a *point cloud*, i.e. a set of discrete points in a 3D coordinate system (generally Cartesian) that is an approximate copy of real objects framed by the sensor. We are not interested in precise 3-D reconstruction [4] [5] (as the ones needed in virtual reality or graphics contexts), because operation control algorithms are based on a parametrized representation of the environment. The objective is clearly to provide the flight control module with sufficient information to elaborate its course and to avoid possibly dangerous objects. The challenge is then to extract synthetic scene information in a very short time. Even though the environment typically navigated by a robot (an example in Figure 1) is formed by objects of different sizes and shapes, the most massive and relevant ones are floor, roof, tables, closets, i.e. structures with regular surfaces. The shape of these objects may grossly be associated to that of a parallelepiped, a geometric structure formed by planes. Therefore the problem can be reasonably reduced to the reconstruction of planar surfaces.

Many algorithms are present in the literature that aim to localize and extract planar surfaces from points belonging to the same planes. The problem can be seen as an instance of the well-known problem of fitting data to one or more models [6], [7], [8]. These approaches are very precise, but present very poor performances, making them unusable for our objective of quasi real-time execution. Therefore we set up a procedure that, using standard, well-known and fast algorithms. manage to extract what we need to know from the point cloud in very short time.

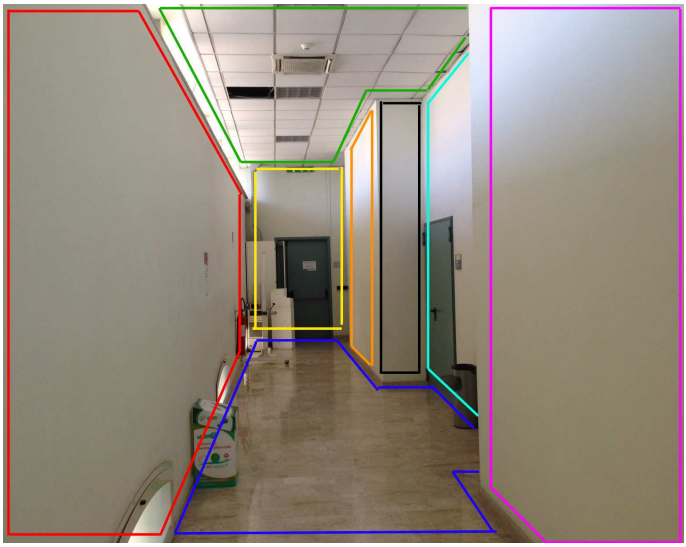

Fig. 1. A typical environment navigable by the robot. The most relevant objects can be described by planes.

3 The Algorithm

The steps of our algorithm is conceptually divided in six steps:

- Acquisition of Point Cloud from Kinect Sensor
- Edge Extraction
- Noise Reduction using Connected-Component Labeling
- Raw clustering of points
- Extraction of planar surfaces
- Refining of identified planes

Each of these operations will be addressed in detail in the following paragraphs.

3.1 Acquisition of Point Cloud from Kinect Sensor

Our front-end sensory system is the *Kinect for Xbox360,* an inexpensive motion sensing input device based on range camera technology developed by Israeli company PrimeSense. The Kinect provides on top of RGB real-time images, a real-time disparity map of the environment. With a simple transformation the data can be converted in real-time in depth and 3D coordinates giving a truly precious real-time 3D-film of the environment.

Kinect's camera has a field of view of 58° horizontally, 45° vertically, and 70° diagonally. The spatial x/y and depth z resolution are respectively 3mm and 1cm (at the distance of 2 meters from the sensor). Kinect operation range is between

0.8 and 3.5 meters, and it is due in part to the dynamic range of the infrared (IR) sensor. The idea behind the scene reconstruction algorithm performed by Kinect [9] [10] consists of utilizing projection of a laser speckle pattern on the surface of an object. The image data grabbed from the IR image unit (embedded in the system and not accessible) is indicative of the objects' relative positions with respect to a reference image. Kinect's embedded algorithm provides only an 11-bit normalized disparity image D collinear to the RGB data from its standard vision camera.

More specifically at each time frame n, we have four $N \times M$ matrices $R(n)$, $G(n)$, $B(n)$ and $D(n)$ representing Red, Green, Blue and *Disparity* respectively. Assuming a Cartesian reference system as the one shown in Figure 2, disparity information can be translated into the 3D coordinate matrices $X(n)$, $Y(n)$ and $Z(n)$ (depth) by the transformations

$$T_{ij}(n) = \frac{100}{-0.00307 D_{ij}(n) + 3.33},$$
$$X_{ij}(n) = \lambda(i - 480/2)(T_{ij}(n) + d),$$
$$Y_{ij}(n) = \lambda(640/2 - j)(T_{ij}(n) + d),$$
$$Z_{ij}(n) = T_{ij}(n),$$

with $i = 1, ..., N$, $j = 1, ..., M$, $d = -10$ and $\lambda = 0021$. Coefficient values have been computed experimentally, as neither PrimeSense nor Microsoft have ever provided specific details about calibration. The set of 3D coordinates at time n, $X(n)$, $Y(n)$ and $Z(n)$ is estimates of objects' external surface points and is named *3D Point Cloud* $P(n) = (X(n), Y(n), Z(n))$.

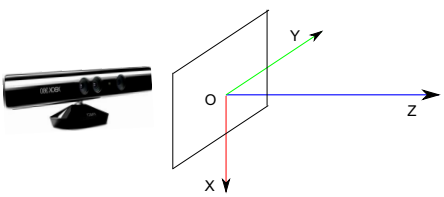

Fig. 2. References axes for the point cloud. Z-axis is in the direction of the Kinect's "eye," Y-axis on the left and X-axis downward in order to form a clockwise tern.

The disparity-to-depth transformation is based on well-known triangulation concepts [11], i.e. an approximate inverse proportionality between distance and disparity.

3.2 Edge Extraction

Given the point cloud $P(n)$, our purpose is to obtain an estimate of the objects' edges. To this end we perform horizontal and vertical edge-detection by convolving two 3×3 *Sobel* [12] [13] operators S_h and S_v with $X(n)$, $Y(n)$ and $Z(n)$, obtaining the six images

$$(X_h(n), Y_h(n), Z_h(n)) = ((X * S_h)(n), (Y * S_h)(n), (Z * S_h)(n)),$$

$$(X_v(n), Y_v(n), Z_v(n)) = ((X * S_v)(n), (Y * S_v)(n), (Z * S_v)(n)),$$

where "$*$" denotes 2D convolution. The binary edge-map image

$$E(n) = [E_{ij}(n)]_{j=1,..M}^{i=1,..N},$$

with $E_{ij}(n) = 0$ (no edge) and $E_{ij}(n) = 1$ (edge), is obtained as

$$E_{ij}(n) = u(|X_{h_{ij}}(n)|+|Y_{h_{ij}}(n)|+|Z_{h_{ij}}(n)|+|X_{v_{ij}}(n)|+|Y_{v_{ij}}(n)|+|Z_{v_{ij}}(n)|-T_e),$$

where $u(\xi)$ is the step function and T_e is an experimentally-determined threshold. If T_e is too low, the contour-map image becomes too noisy, while high values of T_e result in loss of narrow contours. Note that even though the operations on the images are 2D, and the result is a single matrix, the edges represent information from the 3D world.

3.3 Noise Reduction Using Connected-Component Labeling

In order to avoid loss of important scene details (such as boundary lines between different planes), we keep threshold T_e of Subsection 3.2 very high, and get, as expected, a very noisy image. To reduce the noise we use the well-known *Connected-Component Labeling* [14] algorithm, that scans a binary image and constructs a non-binary matrix

$$L(n) = [L_{ij}(n)]_{j=1,..,M}^{i=1,..N}$$

with different values $L_{ij}(n) = l$, $l = 1, ..., N_L$ for each identified connected region. We use this approach into the following scheme:

1. Apply connected-component labeling on the binary image $E(n)$ to obtain $L(n)$.
2. Construct from $L(n)$ an histogram $H(l, n)$ in which for each label l the number of points of $L(n)$ with that label value are counted.
3. For each $l = 1, ..., N_L$, if $u(H(l, n) - T_{cc}) < 0$, (where $u(\xi)$ is the step function and T_{cc} is another experimentally-determined threshold), for each i and j by which $L_{ij}(n) = l$, set $E_{ij}(n) = 0$.

In other words we check for each label if the number of points marked with that label is above or below a fixed threshold. Noisy points are generally scattered all along the image, therefore they will belong to small-numbered connected region and consequently "zeroed" at the end of this step. T_{cc} represents the trade-off between precision (high value of T_{cc}) and level of detail (low value of T_{cc}).

3.4 Raw Clustering of Points

Here we accomplish planar surfaces identification. We cannot simply discriminate between closed zones in the image because, despite our efforts, sometimes we still get fragmented and open contours. Our solution implies the use of a $d \times d$ square $Q_d(n)$ that is moved onto the image. The algorithm checks if the image under the square is empty and, if so, fills it with a label l_i. Adjacent and empty zones are filled with the same label. Iterating this procedure a new non-binary matrix $E(n)$ that represents the surfaces identified in the scene is obtained. This solution overcomes the problem of fragmented image, because a square of a certain size will not be able to "slip" through the holes.

The most important parameter in this step is the dimension d of $Q_d(n)$. A small square leads to great precision but also great computational load and implies the possibility that the square could enter through the holes.

To have the actual clustering, we organize the points $P_{ij}(n)$ from the point cloud in a new structure $C(n) = \{C^l(n)\}^{l=1,\ldots,N_L}$, with $C^l(n)$ that contains

$$C^l_{ij} = ((X_{ij}(n), Y_{ij}(n), Z_{ij}(n)) \in plane_l),$$

with l index of the plane and N_L number of detected planes. We associate at each label l the correspondent points from the point cloud, using matrix $E(n)$ obtained above.

3.5 Extraction of Planar Surfaces

The purpose of this step is to estimate the interpolating planes from 3D points belonging to $C^l(n)$. For each plane l we want to extract the correspondent plane parameters [3]: $\mathbf{n}$ (normal vector), $\widehat{\mathbf{n}}$ (unit normal vector), d (perpendicular distance from the plane to the origin), and the barycenter p_b. We can arrange our equation set in matrix form

$$\mathbf{n}^T\mathbf{P} + \mathbf{q} = \mathbf{0}, \tag{1}$$

where $\mathbf{n}$ is the (not-unitary) surface normal vector, $\mathbf{P}$ is a matrix containing the set of points belonging to a plane l (it is a $3 \times n_p$ matrix, with n_p number of points) and $\mathbf{q}$ is the (unitary) vector of non-normalized distance. Transposing and arranging (1), we obtain

$$\mathbf{P}^T\mathbf{n} = -\mathbf{q}^T,$$

that is in the form $\mathbf{Ax} = \mathbf{b}$. The equation system can be resolved finding the least-squares solution [15] for the over-determined system. Such solution can be found using the singular value decomposition, as shown in [11]. In this way we compute $\mathbf{n}$. At the same time, we compute $p_b = \frac{1}{n_p}\sum_{i=1}^{n_p} p_i$, with p_i points from the $\mathbf{P}$ matrix. From $\mathbf{n}$ we calculate the unit vector $\widehat{\mathbf{n}} = \frac{\mathbf{n}}{||\mathbf{n}||}$, and then calculate $d = -\widehat{\mathbf{n}}^T p_b$. This procedure is repeated for each identified plane l , in order to extract the parameters of all the planar surfaces identified in the scene.

3.6 Refining of Identified Planes

The clustering quality obtained at this stage can be increased taking care of the problem of wrongly-merged planes. This situation is likely to happen with distant objects in the scene: the sensor loses the contour between planes, that are associated within the same label by the moving square of step 3.4. A possible solution is to put on a standard RANSAC (RANdom SAmple Consensus) [7] procedure. By firstly extracting planes parameters from a small number of random points (chosen with a proximity criterion), and by checking for each plane if the number of outliers (points that do not belong to that plane) exceeds a fixed threshold T_{RANSAC}, we can iteratively begin to divide the plane itself into usually two or three planes, associating in the process each point to its rightful owner. A point $\mathbf{p}_i$ belongs to a plane if $d_e = \widehat{\mathbf{n}}^T \mathbf{p}_i + d < T_p$, with T_p an adjustable threshold. A point is discarded if it does not belong to any of the planes identified at this stage.

The execution time of this procedure is dependent on the scene complexity. A complicated scene could require a discrete number of iterations before an effective splitting of the planes, but normally the algorithm adjusts the scene in only few steps.

At this stage the planes are defined, and a last little increase of quality can be achieved discarding for each plane false points (following a criteria, i.e. points whose distance is above a 3σ threshold) and computing one last time the planes parameters.

4 Experimental Results

The algorithm has been tested with data coming from complex scenes framed by the Kinect. We will extract planes from the scene shown in Figure 3 on the left. The scene under consideration is representative of a common environment navigable by a robot. There are obstacles, walls and the floor is framed.

The chosen thresholds are $T_e = 28$, $T_{cc} = 50$, $d = 15$, $T_p = 5$, $T_{RANSAC} = 20$. Figure 4 shows the operation performed in Step 2 (left), 3(center) and 4(right), where from the point cloud we identify planar surfaces of the scene. Figure 5 is about the raw clustering of points into planar surfaces, and then the refining of the reconstruction via RANSAC.

The algorithm returns the parameters of planes composing the scene. This information can be effectively available to the navigation algorithm. For instance, the height from the ground, important for a flying robot that varies its own quota, can be estimated by checking each of the normal vectors of the planes. The floor is the plane with a normal vector pointing in the negative X-direction. If more than one plane meets this requirement (i.e. tables), we can pick the one with the lowest barycenter value. Other useful information about the scene (identification of roofs, distance from walls, etc.) can be easily obtained from our data, and can be used by the flight control module of a robot to securely navigate an arbitrary and unknown scenario.

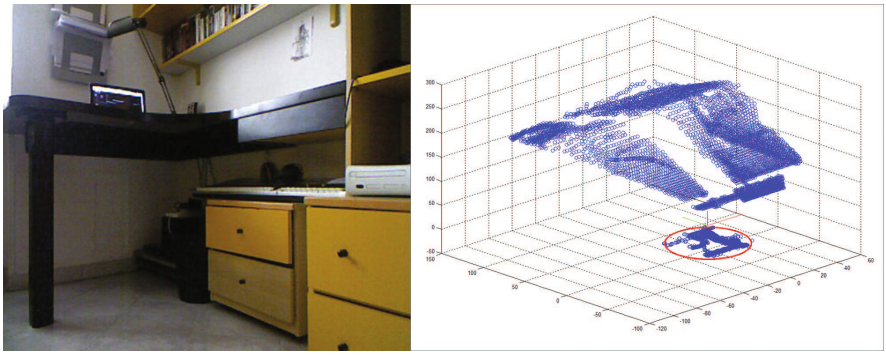

Fig. 3. Left image: the scene under consideration. Right plot: the point cloud extracted by the Kinect sensor. Due to light conditions and scene complexity, the sensor returns also few false points (the ones surrounded by the ellipse, that have a negative Z value). These points will be discarded during the execution.

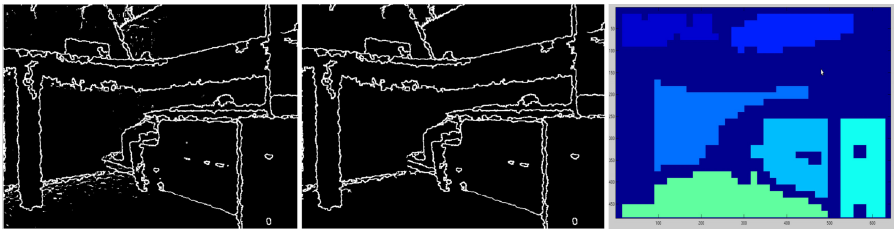

Fig. 4. An edge image (on the left) and a noise-free edge image (in the center). The threshold's choice is crucial to avoid loss of important contours. The image on the right shows the identified planes.

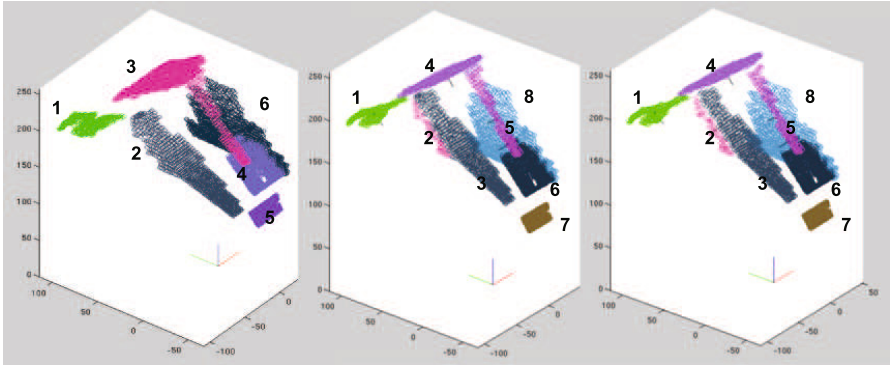

Fig. 5. Initial 3D reconstruction of scene planar surfaces (on the left). Wrongly-merged planes are split and identified with different labels as result of RANSAC (in the middle). The last operation is the discarding of points whose distance is above 3σ (on the right).

A C++ implementation of this algorithm has showed fast execution times ($\simeq$ 100 ms on low-class dual-core processors), that makes this procedure usable in real-time scenarios. It is important to notice that the procedure is also highly-parallelizable, because it performs matrix operations in which each element is computed independently from the others. The approach presented here is frame-by-frame i.e. each scene needs the output of the previous one. Future work will be devoted to prediction-based strategies in which previously available information can be just updated.

5 Conclusions

This paper has proposed a new fast clustering algorithm for planes identification. Many different approaches in literature have tried to achieve the same objective, but focusing on the entire set of 3D points. Such "brute-force" methods can lead to good results, but with heavy computational load. Our algorithm instead tries to reduce the execution time by performing first simple clustering, and then increasing the reconstruction quality.

The balance between execution time and quality is crucial to meet the requirements of real-time surroundings estimation. Each of the algorithm's steps uses standard computer vision algorithms, but each choice has carefully made keeping in mind the trade-off between performances and computation load. Other choices are obviously possible and will be further investigated in future papers. For instance, the RANSAC phase (one of the algorithm's bottlenecks) could be modified, or we could use different approaches.

Even the planar hypothesis could be modified. We could consider other types of structures (cylinders for example) and modify the algorithm to identify and reconstruct different geometric forms.

Another important element to focus on is thresholds' choice. Arbitrary environments differ in number and type of objects, light conditions, etc. Therefore thresholds' values have to be tuned accordingly to the scene. A possible future development involves the creation of an algorithm that adaptively chooses the thresholds.

References

1. Leonard, J.J.: Durrant-Whyte: Simultaneous Map Building and Localization for an Autonomous Mobile Robot. In: IEEE/RSJ International Workshop on Intelligent Robots and Systems, IROS 1991, vol. 3, pp. 1442–1447 (1991)
2. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotic (Intelligent Robotics and Autonomous Agents). The MIT Press (2005)
3. Gellert, W., Gottwald, S., Hellwich, M., Kastner, H., Kunstner, H.: VNR Concise Encyclopedia of Mathematics, 2nd edn. Springer (1990)
4. Hoppe, H., DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W.: Surface Reconstruction from Unorganized Points. In: ACM SIGGRAPH 1992 Proceedings, pp. 71–78 (1992)

5. Cazals, F., Giesen, J.: Delaunay triangulation based surface reconstruction: Ideas and algorithms. Rapport de recherche. INRIA (2004)
6. Vidal, R.: A Tutorial on Subspace Clustering. IEEE Signal Processing Magazine 28, 52–68 (2011)
7. Martin, A., Fischler, B.R.C.: Random Sample Consensus: A Paradigm for Model Fitting with Application to Image Analysis and Automated Cartography. Comm. of the ACM 24(6), 381–395 (1981)
8. Zuliani, M., Kenney, C.S., Manjunath, B.S.: The multiRANSAC algorithm and its application to detect planar homographies. In: Proceedings of the IEEE International Conference on Image Processing, Genova, IT (2005)
9. Method and System for Object Reconstruction (2007)
10. Depth Mapping using Projected Patterns (2008)
11. Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision, 2nd edn. Cambridge (2003)
12. Main, R.: Study and Comparison of Various Image Edge Detection Techniques. In: Image Processing 2009, vol. 147002(3), pp. 1–12. Citeseer (2009)
13. Shafiri, M., Fathy, M., Mahmoud, M.T.: A classified and comparative study of edge detection algorithms. In: Proceedings of the International Conference on Information Technology: Coding and Computing, pp. 117–120 (2002)
14. Klaus, B., Horn, P.: Robot Vision. McGraw-Hill (1986)
15. Wall, M.E., Rechtsteiner, A., Rocha, L.M.: Singular value decomposition and principal component analysis. In: Berrar, D.P., Dubitzky, W., Granzow, M. (eds.) A Practical Approach to Microarray Data Analysis, pp. 91–109. Kluwer, Norwell (2004)