Incremental compact 3D maps of planar patches from RGBD points

Juan Navarro and José M. Cañas

Universidad Rey Juan Carlos, Spain

Abstract. The RGBD sensors have opened the door to low cost perception capabilities for robots and to new approaches on the classic problems of self localization and environment mapping. The raw data coming from these sensors are typically huge clouds of 3D colored points, which are heavy to manage. This paper describes a premilinary work on an algorithm that incrementally builds compact and dense 3D maps of planar patches from the raw data of a mobile RGBD sensor. The algorithm runs iteratively and classifies the 3D points in the current sensor reading into three categories: close to an existing patch, already contained in one patch, and far from any. The first points update the corresponding patch definition, the last ones are clustered in new patches using RANSAC and SVD. A fusion step also merges 3D patches when needed. The algorithm has been experimentally validated in the Gazebo-5 simulator.

Keywords: 3D point cloud, mapping, computer vision

1 Introduction

In late 2010 Microsoft introduced the Kinect sensor device with its Xbox 360 game console. Since them many Kinects have been sold, and other low cost RGBD sensors have appeared like Asus Xtion and Kinect-2. Opening the SDK to manage these devices has brought applications for them in other areas like robotics [11] , health, security and industrial applications. Now they are about to be included in mobile phones and tablets for general public applications (Apple bought PrimeSense in 2013, Google is developing its Tango project [1] and both have interesting prototypes).

In robotics, maps are useful for robot navigation and even self localization. Taking into account the information contained in maps a robot can plan its paths and make better movement decisions. Many map building techniques have been developed to create maps from robot sensors . They gather noisy sensor data and merge them into a more abstract, reliable and compact description of the 2D or 3D environment. In the last years many SLAM techniques ([9][4]) have been developed that simultaneously cope with the Localization and Mapping problems for mobile robots, using different sensors as input. In particular many recent SLAM works focus on the use of RGBD sensors ([10][3][5]).

[1] https://www.google.com/atap/project-tango/

Many 3D spatial primitives have been proposed in the literature to represent the 3D maps of scene or objects: 3D Points (dense point clouds), regular 3D voxels, octree maps [6], Surfels [2], Signed Distance Fusion (SDF) [8], patch volumes [7] and planar patches [1] among others.

This paper presents a preliminary work on an algorithm that builds incremental 3D maps from 3D point clouds using planar patches as compact spatial primitive. The continuous 3D pose estimation of the sensor is outside the scope of this paper, and we assume that a localization algorithm is working in the background. We focus on the incremental map construction and update.

2 Incremental 3D planar patches map building

We have designed and developed an algorithm for building a 3D map of the environment surrounding a mobile RGBD sensor using its readings. We define the map as a collection of 3D planar patches. A *planar patch* is defined as an area of a plane bounded by a contour (Fig. 1). Each planar patch (patch with hourglass shape, in green) is described by the general equation of the corresponding plane (square plane, in red), its *contour* (yellow line of the hourglass shape) and associated *cell image* (black and white binary image).

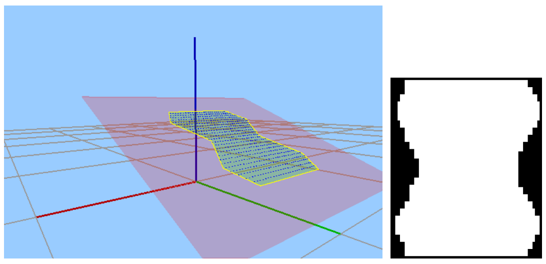

Fig. 1. Example of 3D points, a planar patch in 3D and its cell image.

A *cell image* gives us information on the distribution of points within the planar patch. It representes an image of the occupied and empty zones after dividing the area of each planar patch into fixed-size cells (pixels).

2.1 Design

The map building algorithm runs iteratively following the next steps:

1. Data acquisition. At the beginning of each iteration the current 3D position of the RGBD sensor and the current depth image are collected. The depth image is transformed into 3D point cloud applying a depth sampling and translation to absolute coordinates.

2. Point Classification. Each 3D point from the current cloud is classified into one of the following three categories according their relationship with the existing planar patches: (a) Belonging to one planar patch; (b) Close to one planar patch, and (c) Far from any existing patch, and so, not explained.
3. With the points belonging to one of existing planar patch we do nothing.
4. Patch redefinition. Each set of points close to an existing planar patch are used to redefine its contour.
5. The mapping algorithm generates (several) new planar patches that group unexplained points. This stage involves several steps:
   (a) RANSAC algorithm is used to group the unexplained points in one or more tentative planar patches.
   (b) Each tentative patch is evaluated with a *coherence test* to ensure that holds four criteria to be considered valid. If the patch meets all criteria it will be considered valid and added to the map. If the patch fails any of the first three criteria, then its points will be returned to RANSAC algorithm to continue considering them. If the patch only fails in the connexity criterion then it will be refined into several smaller refined patches. This situation is typical of very large patches with holes inside and in the case of several noncontiguous but coplanar areas.

   The RANSAC algorithm is iterative itself and ends when the number of unexplained points is less than a certain amount, or if it exceeds a certain number of iterations without extracting any valid new planar patch.
6. Fusion step. Finally the algorithm explores all possible pairs of existing patches to check whether some of them can be joined together. The generated map is a vector of planar patches and a vector of three-dimensional unexplained points.

The algorithm has several thresholds, which were finally selected after testing in the experiments. It has been programmed as a C++ component in the open-source *JdeRobot* framework[2] .

2.2 Data acquisition

In each iteration of the algorithm, the position and orientation of the device is captured first, and then the depth image from the RGBD sensor. A sampling is performed when translating the Depth image into a 3D point cloud. Without sampling the density of 3D points near the sensor is greater than the density of distant points. With our sampling all the Depth pixels are separated into layers according its distance to the sensor, and a different sampling rate is used on each layer to preserve an homogeneous 3D density of points between close and far areas. Fig. 2 shows a raw point cloud (left) and the sampled one (right).

So far the points are computed in coordinates relative to the sensor, but now the algorithm computes their absolute position incorporating the current 3D position and orientation of the RGBD sensor. Due to this, the algorithm can segment the planes from various RGBD sensors (or a mobile one) as all the points lie in the same absolute frame of reference.

[2] http://jderobot.org

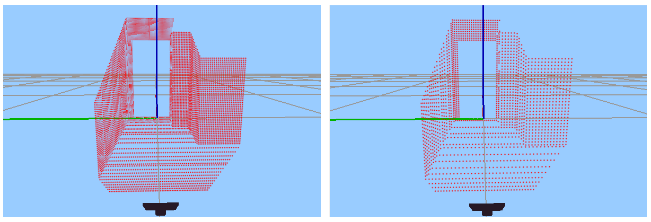

Fig. 2. Raw point cloud from the RGBD sensor and the sampled one.

2.3 Point classification

Second step of each iteration classifies the 3D points into three categories according to their relation with the existing planar patches until the previous iteration (Fig. 3):

1. *Belonging to a planar patch.* These points do not give us any new information, only confirm some existing patch.
2. *Close to a planar patch.* These points will be useful to extend and redefine the planar patches which they are associated to.
3. *Unexplained by any existing planar patch.* The algorithm will try to group them in new planar patches.

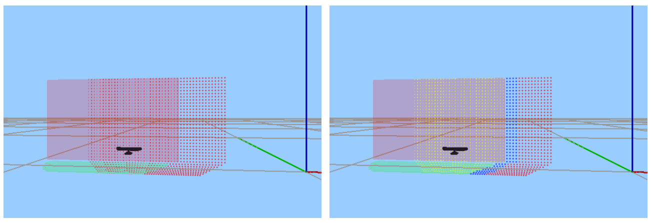

Fig. 3. Existing planar patches, point cloud (left) and their classification (right): close (yellow), nearby (blue) and unexplained (red).

For each point of the cloud we will calculate its normal distance to all planar patches. For those points whose normal distance is less than a configurable threshold, we also calculated the minimum lateral distance to the contour of the corresponding planar patches. If this lateral distance is less than another threshold, we consider that point *belonging* to the plane. If not, if the lateral distance is less than a second threshold, we will consider *close* to the planar patch. One point may fall into different categories with respect to several planar patches and then, it is classified according to the planar patch whose distance is smallest. The points that do not fit into any of the previous categories are classified as *unexplained by any planar patch.*

2.4 Redefinition of patches

With the set of points that are close to an existing planar patch, we extend this patch by redefining its cell image and its contour, but without affecting the equation of its plane (Fig. 4).

For the redefinition of the cell image, first we add the new points to the corresponding cells, adding new rows and/or columns of cells at the edges of the image where necessary. Then we employ *dilate* and *erode* functions of OpenCV to apply dilation a predefined number of iterations, erosion the same number of iterations plus one, and another iteration of expansion (more information about this in the subsection 2.5). This preprocessing is necessary to join slightly separated areas, and to eliminate the isolated areas.

Once the cell image is preprocessed, we obtain the new contour of the image using the *findContours* function of OpenCV. The row and column coordinates of the cells that form the contour of the image are translated into absolute 3D coordinates, and a Principal Component Analysis of the new cell image is performed. Initially, we did use PCL to obtain the contour by calculating the Concave Hull of the points that support a patch, but for some concave patches the contours were inadequate.

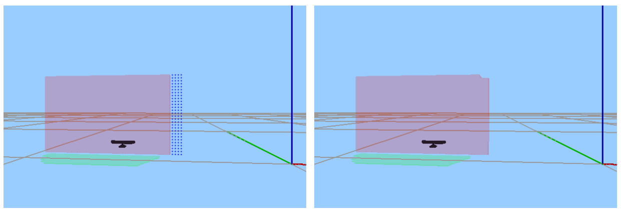
Fig. 4. (Left) Wall patch and close points. (Right) Redefined, wider, patch.

2.5 Generation of new patches

With the point cloud of *unexplained by any existing patch* we seek for new patches that may explain these points using the RANSAC *(RANdom SAmple Consensus)* iterative algorithm. At each iteration it randomly selects three points of the cloud as candidates to form a plane. It classifies the remaining points into *inliers* (low distance to that candidate plane) or *outliers* (big distance), according to normal distance of each point relative to the candidate plane. Also, it gets the average distance associated with the inlier points. It stores the plane with the highest number of inliers, and in case of a tie, the plane that has the lower average distance. When a plane remains as the best for a number of iterations (which is customizable), RANSAC presents it as tentative patch.

Some of the tentative patches obtained by RANSAC are unsatisfactory. For instace, it extracts only one tentative patch several patches that are coplanar but not connected. Therefore we implemented a *coherence test* to study the validity of these tentative patches before incorporating them into the map. If they include separate coplanar clusters, the algorithm refines the tentative patch into smaller patches which must also pass the coherence test themselves.The coherence test consists of four criteria:

1. Have a sufficient number of points that support it.
2. Exceeds a nonlinearity threshold, to ensure that the distribution of the points does not correspond to a line.
3. The size of its surface is large enough.
4. The occupied cells in the corresponding cell image are connected.

After verifying that meets the first criterion, the algorithm builds the *cell image* associated to the tentative patch to evaluate the remaining criteria. For generating the cell image, a Principal Component Analysis (PCA) is performed on the cloud of points of this patch , using a *Singular Value Decomposition (SVD)*. The PCA extracts the three perpendicular directions of greater variability in the distribution of points . With this analysis we obtain the three principal singular vectors with their associated eigenvalues, although we are only interested the first two. We also get the centroid of the distribution.

Using the centroid as the origin and the two principal singular vectors (which are unitary) as basis vectors, we pass the points to this new two-dimensional base. Looking for the minimum and maximum values in both directions, we save the minimum value and the geometric distance in each direction. Using the geometric distance, along with the size chosen for the side of the cells we get the number of cells in each direction, and we round it upwards.

With the points in two-dimensional format we generate a raw cell image with the occupied cells (Fig. 5(a)). Then we perform a preprocessing using the *dilate* and *erode* functions of *OpenCV*, consisting in a number of iterations of dilation (Fig. 5(b)), the same number plus one of erosion (Fig. 5(c)), and other of dilation (Fig. 5(d)), to join groups of cells near each other but eliminate isolated cells.

To prevent that dilation damaged the real perimeter estimation, during preprocessing we added an edge of pixels that simulate empty cells. At the end we will cut the edges leaving only one pixel around (Fig. 5(e)), necessary when the calculating of the contour with *findContours* function of OpenCV is performed.

To check the linearity criterion, we compare if the geometric distance corresponding to the second singular vector exceeds a certain threshold. For the minimum size criterion, we compare the product of the two geometric distances with a threshold which represents the minimum area to be considered.

And for the criterion of connectivity, we perform a growing region algorithm on the cell image using *findContours* and *floodFill* functions of *OpenCV* (Fig. 6). We use as seed of region growth the first vertex of each contour. If only one region is obtained, the patch meets the criterion. If there are more regions then the tentative patch will be refined in a number of patches equal to the number of obtained regions, which are formed by the points in the cells of each region.

If a tentative patch holds the four criteria then is considered valid, and its contour is calculated using the *findContours* function of *OpenCV* on its cell im-

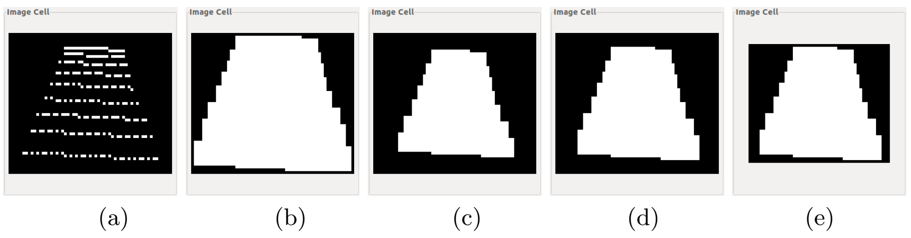

Fig. 5. Preprocessing of cell image: (a) Initial image. (b) First dilation. (c) Erosion. (d) Second dilation. (e) Cut of the edge.

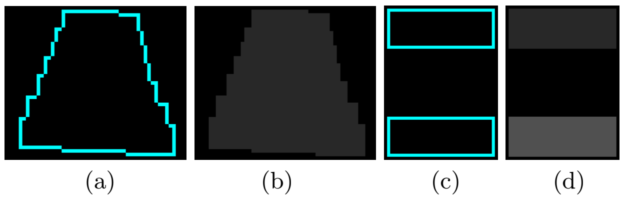

Fig. 6. Connectivity criterion: (a) *findContours* and (b) *floodFill*, for a connected patch. (c) and (d) for a non-connected patch.

age (Fig. 7). The two-dimensional coordinates of the cells that form the vertices are converted to absolute three-dimensional coordinates. Then we store the planar patch on the map, defined by its equation of the plane, its contour and its cell image.

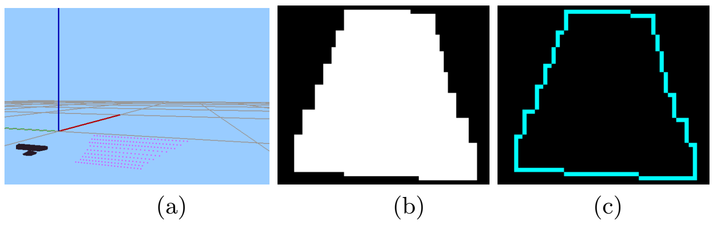

Fig. 7. (a) Points of a tentative patch (b) Cell image (c) Contour.

If a tentative patch fails during any of the first three criteria, it is discarded directly and their points are returned to the RANSAC algorithm to continue taking them into account in the search for new tentative patches. If it only fails the ultimate criterion, because it is formed by non-connected groups (Fig. 8), it will be refined into several smaller tentative patches. The failed coherence test delivers the sets of points corresponding to the tentative refined smaller patches.

These refined patches are passed to a new coherence test for themselves. This time, those which meet the four criteria will be incorporated into the map, those failing any of the criteria will be discarded and their points returned to the RANSAC algorithm.

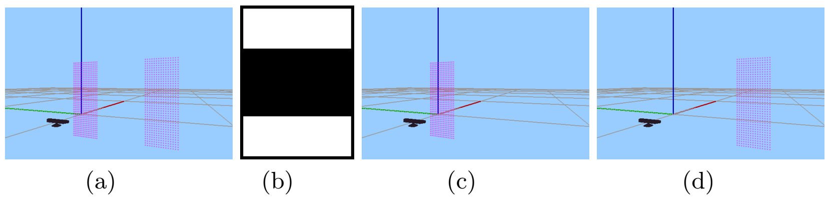

Fig. 8. (a) Points of non-connected patch. (b) Cell image of non-connected patch. (c) Points of first refined patch. (d) Points of the second refined patch.

Since RANSAC is an iterative algorithm we need a stop condition. We used any of these two: first, if at the beginning of an iteration the number of remaining unexplained points is less than a threshold; second, if the number of iterations exceedes a certain maximum without extracting any valid patch. Fig. 9 shows the result of applying this algorithm.

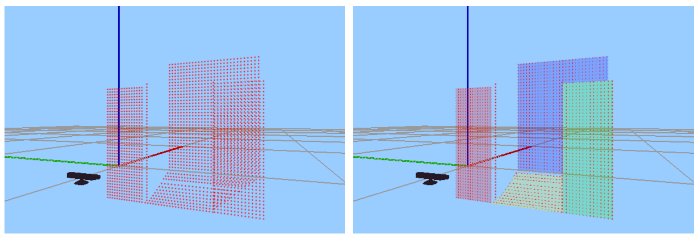

Fig. 9. Generation of new planar patches: (left) Initial points, (right) Obtained patches.

2.6 Fusion of patches

The final step of the iterative mapping algorithm is the patch fusion. We explore the planar patches inside the map to check whether some of them correspond to different areas of a single real larger patch and can be joined together. We compare each planar patch with all others. For each pair of patches we first check their parallelism calculating the cross product of their normals to plane equations. If they can be considered parallel, we studied their coplanarity through its normal distance. To calculate the normal distance between two planar patches we get the minimum distance of the contour points of each planar patch to the plane equation of the other patch. If one of the two minimum distances is less than a threshold we consider the two patches coplanar.

For a pair of coplanar patches, we also calculate the minimum lateral distance between them. If any vertices of the contour of a planar patch is contained within the contour of the other, the lateral distance is zero. If the lateral distance is not zero, we calculate the minimum distance between all of the vertices of the contour of each patch to all the sides the contour of the other patch. If this lateral distance is less than a threshold we consider this pair of patches amenable of fusion.

When two patches are amenable of fusion we redefine the first one using the second . To redefine a planar patch by fusion with another patch, we first obtain a point cloud formed by the vertices of the contours of both patches and two sets of synthetic points within their cell images. Generating synthetic points from the cell image consists in generating one point in three-dimensional coordinates per each occupied cell With that point cloud we obtain the equation of the plane that best fits it using SVD (Singular Value Decomposition). We also define a new cell image using that point cloud, and with such cell image we get the contour of the planar patch resulting from the fusion. This new patch is compared with successive patches of the map for the remaining of the fusion step. This allows to join several patches that correspond to the same one in a single pass of the fusion algorithm. The Fig. 10 shows a fusion between planar patches.

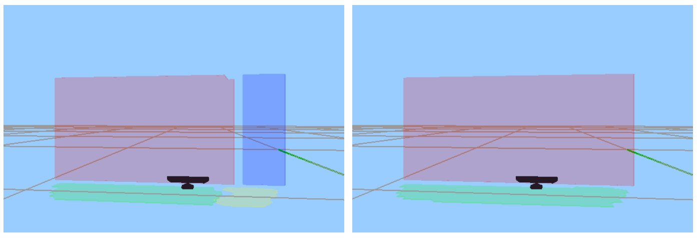
Fig. 10. Fusion of patches: (left) Scene before, (right) scene after.

3 Experiments

This section describes some experiments made with the software implementation of the developed algorithm in a simulated environment. To perform these experiments we used a scenario consisting of an apartment in Gazebo simulator. In particular, the world `GrannyAnnie` of the Rockin@Home[3] project, shown in Fig. 11. The experiments test the algorithm output positioning the simulated

[3] http://rockinrobotchallenge.eu/home.php

sensor at different locations of the apartment. For this experiments, before run the algorithm in each location, we wait until the 3D sensor location reading is stabilized.

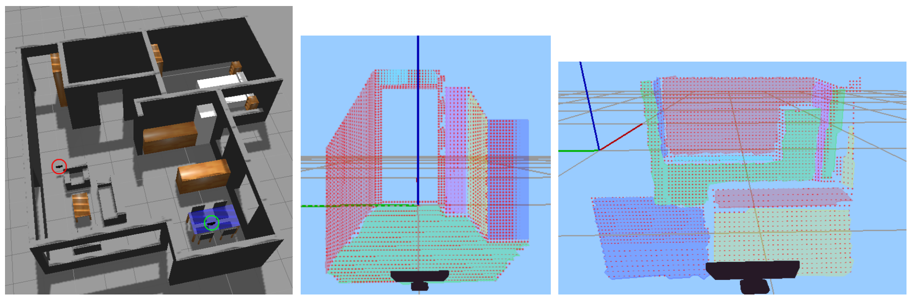

Fig. 11. Apartment GrannyAnnie in Gazebo. Patches and point cloud for the hall and for the kitchen.

The used hardware is a desktop PC with a processor Intel Core 2 Quad CPU Q6600 @ 2.40GHz(x4) and 2x2GB(4GB) DDR2 @ 800MHz of RAM memory. The Gazebo simulator and our component simultaneosly run in this computer. The minimum, average and maximum times for each iteration of our algorithm are 27 ms, 77 ms and 304 ms, respectively. These numbers are obtained after multiple runs of the algorithm working with a cloud of 1500 $\sim$ 3500 points and up to 25 patches. We expect similar costs for the real scenario but we have not tried yet because not have solved the reliable 3D location.

3.1 Typical execution from static position

In the first experiment the sensor is located oriented to the hall with a shelf (red circle, at the left in Fig. 11). The center of the same figure shows the point cloud. The variable depth sampling rate allows a good distribution of points in all patches, resulting in 3361 points. It also shows the 6 patches resulting from the algorithm. The right side of the Fig. 11 shows the point cloud (in this case of 3046 points) and the obtained planar patches (10) from the mapping algorithm when the sensor is located at the kitchen. In both situations the obtained patches are very satisfactory and correspond pretty well to the scenario.

3.2 Incremental map from several positions

In this experiment the ability of the algorithm to incrementally modify its planar patch collection has been tested. The resulting map coming from incorporating two readings from two different sensor locations was compared to real scenario. We placed the sensor in two positions inside of the bedroom, keeping overlapping areas between the two observations (Fig 12). This scenario has furniture that includes patches of different sizes.

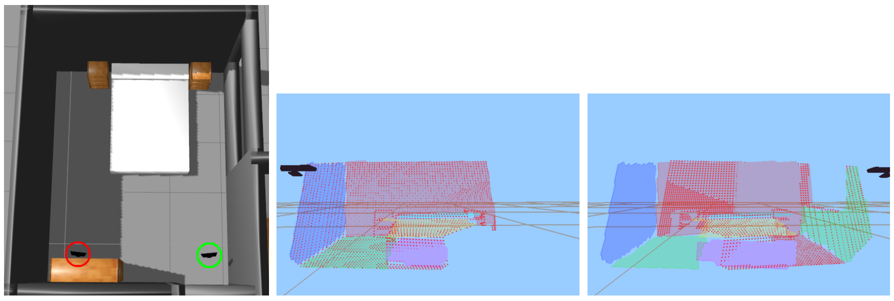

Fig. 12. Bedroom of the apartment in Gazebo and planar patches and point cloud for the bedroom in positions 1 and 2.

First, we placed the sensor in front of the bed and left (sensor, with the red circle, in the left of Fig. 12), facing slightly clockwise and towards the ground. The center of the same figure shows the cloud with 3453 points and the 6 patches obtained by the mapping algorithm.

Second, we placed the sensor in front of the bed, but this time right (sensor, with the green circle, in the left of Fig. 12), facing slightly counterclockwise and towards the ground. In this position, several iterations of the algorithm were executed, allowing the redefinition of existing patches from the previous position . In the right image of the same figure we have the cloud with 3117 points, and the 9 patches resulting of this process. We can see how the patches of the background wall (red), the top of the bed (yellow), and the part of the foot of the bed (magenta) have extended their surfaces absorbing points .

4 Conclusions

In this paper a preliminary work with an incremental 3D map building algorithm from RGBD sensor data has been presented. It uses planar 3D patches as spatial primitive, and faces typical problems when working with noisy low cost RGBD devices like gap filling, extension with close points and segment fusion. The algorithm computes point-plane associations and so classifies new coming points as belonging to an existing patch, close to one of them or far from any. Close points redefine patch spatial equation and its contour. Far points are clustered together using RANSAC an a coherence planar test.

The generated 3D maps are compact, dense and simple, with tens of patches instead of millions of points. They can be used in 3D path planning and robot navigation. The algorithm is incremental. It has been experimentally validated in a cutting edge simulator, Gazebo-5, showing promising results.

We have assumed that the 3D localization was already solved by a different algorithm, so all the 3D raw data are located in absolute coordinates. We want to test the algorithm with noise and false readings in both the real depth data and the received 3D position from a real 3D localization algorithm.

Acknowledgments

This work has been supported by the Spanish Government DPI2013-40534-R Grant, supported with Feder funds. The research leading to these results has also received funding from the RoboCity2030-III-CM project (S2013/MIT-2748), funded by Programas de Actividades I+D en la Comunidad de Madrid and cofunded by Structural Funds of the EU.

References

1. C. Erdogan and M. Paluri and F. Dellaert, *Planar Segmentation of RGBD Images using Fast Linear Fitting and Markov Chain Monte Carlo*, 2012 Ninth Conference on Computer and Robot Vision (CRV), pp 32-39, 2012.
2. P. Henry and M. Krainin and E. Herbst and X. Ren and D. Fox, *RGB-D Mapping: Using Kinect-style Depth Cameras for Dense 3D Modeling of Indoor Environments*, International Journal of Robotics Research (IJRR) 31:5, 2012.
3. F. Endres and J. Hess and Jürgen Sturm and Daniel Cremers and Wolfram Burgard, *3D Mapping with an RGB-D Camera*, IEEE Transactions on Robotics, VOL. 30, NO. 1, January 2012.
4. F. Endres and J. Hess and N. Engelhard and J. Sturm and D. Cremers and W. Burgard, *An Evaluation of the RGB-D SLAM System*, in Proc. of the IEEE Int. Conf. on Robotics and Automation, (ICRA), 2012
5. F. Endres, J. Hess, J. Sturm, D. Cremers, W. Burgard, *3D Mapping with an RGB-D Camera*, IEEE Transactions on Robotics, 2014.
6. J. Sückler and S. Behnke, *Multi-Resolution Surfel Maps for Efficient Dense 3D Modeling and Tracking*, Journal of Visual Communication and Image Representation 25(1):137-147, Springer, January 2014.
7. P. Henry and D. Fox, *Patch Volumes: Segmentation-based Consistent Mapping with RGB-D Cameras*, 2013 International Conference on 3D Vision, 2013.
8. R.A. Newcombe and S. Izadi and O. Hilliges and D. Molyneaux and D. Kim and A.J. Davison and P. Kohli and J. Shotton and S. Hodges and A. Fitzgibbon, *Kinect-Fusion: Real-Time Dense Surface Mapping and Tracking*, 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR), October, 2011.
9. R.A. Newcombe and S. Lovegrove and A.J. Davison, *DTAM: Dense Tracking and Mapping in Real-Time*, IEEE International Conference on Computer Vision, ICCV 2011, pp 2320–2327, Barcelona, 2011.
10. C. Audras and A.I. Comport and M. Meilland and P. Rives, *Real-time dense appearance-based SLAM for RGB-D sensors*, in Australian Conference on Robotics and Automation, 2011.
11. L.V. Calderita and L.J. Manso and P. Nuñez, *Assessment of PrimeSense RGB-D camera for using in robotics: Application to human body modeling*, Proc. of XIII Workshop of Physical Agents, WAF 2012, September 2012.