Reconstruction of Lightweight 3D Indoor Representations using Color-Depth Cameras

José Mendonça
Instituto Superior Técnico / Universidade de Lisboa, Lisbon, Portugal
jose.mendonca@tecnico.ulisboa.pt

Abstract

The current bandwidth and traffic limits on mobile data connections make it infeasible, for domestic surveillance robots, to regularly send the data acquired on the surveilled environment to mobile devices, ultimately defeating their purpose. In this work we propose developing a methodology to build global, lightweight 3D indoor representations using the range information returned by a low-cost RGB-D camera, so that they are compact enough to be easily transferable to mobile devices. We detail methods to rapidly segment *range images* into planar patches, use them to estimate camera motion and merge them into a global 3D model.

Keywords: RGB-D Cameras, 3D Mapping, Normal Estimation, Microsoft Kinect, Pose Estimation, Planar Segmentation

I. Introduction

Current surveillance robots monitor their owner's homes and collect data, usually as video. However, this data can take up so much space that it is impossible to send it to mobile devices, defeating the purpose of "telepresence". With the development of new sensors, like the *Microsoft Kinect*®, and data processing techniques a solution can be found. It is our objective to develop a methodology to build scene representations for the specific purpose of remote surveillance. The emphasis is put on creating lightweight, human-friendly 3D models passable of being sent and explored in mobile devices.

A. Related Work

The construction of global 3D models using range images is a widely explored topic, with several approaches to it [20], [6]. The planar segmentation of *range images* is also widely explored with examples such as [19], [2], [17]. An important reference for understanding the problem of *surface normal* estimation is [11]. Alternative methods, aimed at *range images* are also presented in [21], [5]. [3] is an important reference to the problem of *point cloud* alignment. An alternative approach, that presents a visual Simultaneous Localization and Mapping (SLAM) method relying on a RGB-D camera is given in [16]. Planar patches may also be used to this end, as demonstrated in [18]. This work is a continuation of that developed by Ricardo Lucas [12] in his MSc thesis.

B. Problem Formulation

In this work, we propose to develop a methodology to build lightweight, global 3D indoor scene representations with data acquired by an RGB-D camera, more specifically, a *Microsoft Kinect*®, mounted on a mobile robot. The objectives are therefore twofold: (i) implementing a methodology to convert range images into alternative, compact, user-friendly scene representations and (ii) develop a method to acquire visual odometry that can be used to merge different scans into single, global scene representations.

II. RGB-D Cameras

RGB-D cameras are complex devices capable of delivering 2D images showing the color and the distance of points in a 3D scene. Simple models can be used to understand their data. In this section, we introduce the *pinhole* camera model and the *Kinect* range sensor (a more detailed explanation of both subjects can be found on the main work [14]).

A. Pinhole *Camera Model*

The *pinhole* camera model describes the mathematical relation between the 3D coordinates of a point in space and its 2D projection on an image, simplifying the camera so that it has a single point aperture and no lenses to focus light.

A light ray, emitted by a point in the 3D scene, goes through the camera's *optical center* and intersects the 2D *image plane* in a single point.

Being $\hat{X}' = [^w X \ ^w Y \ ^w Z \ 1]^T$ a point in the 3D scene on an arbitrary coordinates frame, $\hat{u} = [u \ v \ 1]^T$ the *pixel* projection of the 3D point on the *image plane* and taking into account the camera's *extrinsic parameters* (R, T) and *intrinsic parameters* (K), the model takes the form

$$\boldsymbol{\lambda \hat{u} = K \left[R \quad T \right] \hat{X}'} \tag{1}$$

where λ is an arbitrary scale factor.

From equation (1) and knowing $\boldsymbol{R}$, $\boldsymbol{T}$ and

$$\boldsymbol{K} = \begin{bmatrix} fs_x & & o_x \\ & fs_y & o_y \\ & & 1 \end{bmatrix} \tag{2}$$

where f is the camera's *focal length*, s_x and s_y are the meter to *pixel* horizontal and vertical conversion factors and o_x and o_y are the pixel coordinates of the image's principal point, o, we can obtain the well known perspective projection from 3D coordinates, $[X \ Y \ Z]^T = R \ [^w X \ ^w Y \ ^w Z]^T + T$, to *pixel coordinates*, $u = fs_x X/Z + o_x$ and $v = fs_y Y/Z + o_y$.

Conversely, the transformation from *pixel coordinates* to 3D coordinates, considering a known depth Z, is given by

$$X = \frac{Z}{fs_x}(u - o_x) \ , \quad Y = \frac{Z}{fs_y}(v - o_y). \tag{3}$$

B. Microsoft Kinect®

The *Kinect* is a low-cost *range sensor*, released by *Microsoft* in 2010. It includes a color RGB camera, an Infra-Red (IR) camera and an IR laser projector. It outputs two independent VGA resolution images at a 30 Hz frame rate: an RGB image and a Depth image. The sensor has an angular field of view of 57° horizontally and 43° vertically and nominal depth range limits of roughly 0.8 to 3.5 m. Both the IR projector and camera are used in computing each point's depth value, as derived in [10].

As derived by Khoshelham [9], the *Kinect* exhibits a random *depth* measurement accuracy error that increases quadratically with the distance to the object, having approximately 4 cm of standard deviation at 4 m distance.

III. RDG-D Image Plane Based Segmentation

Taking advantage of the planarity of indoor scenes, we developed a method to rapidly segment *range images* into its main planar patches.

A. Fast Estimation of Surface Normals using Tangent Vectors

A normal to a surface, at a given point, is a vector perpendicular to the tangent plane to that surface at that same point. There are several classic methods to estimate normals as detailed by Klasing [11]. We propose an alternative method that takes advantage of the information coded by the relative position between pixels to make estimations faster.

As illustrated in Figure 1, for each image point we can define a vertical vector, $\vec{V}_o$, by using same column neighbors and an horizontal vector, $\vec{H}_o$, by using same row neighbors. The *surface normal* is then given by

$$^k\vec{N}'_o = {}^k\vec{H}_o \times {}^k\vec{V}_o \tag{4}$$

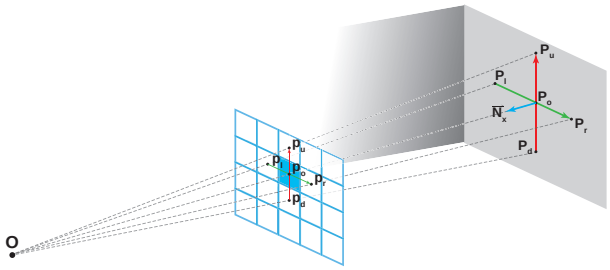


Figure 1: Normal Estimation using Tangent Vectors: a local surface normal estimation may be computed as the cross-product of two orthogonal values, defined by points P_r, P_l, P_u and P_d.

This generates normals that consistently point to the "inside" of the 3D scene. To minimize the effects of measurement errors, this operation is repeated with several groups of neighbors, at different distances from the target *pixel*, and the final *surface normal* is a weighted sum of the individual results. The normal's length is then *normalized*.

B. Random Search for Planar Point Clusters

We use the coordinates and estimated *surface normals* to cluster the points from the *range image*. We select a random unassigned image point, p_k, and take its 3D coordinates and normal as a plane descriptor. They are used to compute the plane's distance to origin along its normal, d_k. From all the other unassigned image points, the ones that belong to the same cluster are those whose: a) distance to origin difference from d_k is under a *distance threshold*, ϵ_d, b) whose angular difference between their normal and the originally selected point's is under a *angle threshold*, ϵ_θ and c) whose distance

to the originally selected point's coordinates is under a *maximum distance*, ϵ_l.

The result of this verifications is a list of points L_k, with all image points that belong to the same surface as p_k. It is only valid if it has a significant number of points, that is, $\#(L_k) \geq minPts$. The whole process is repeated for a total of t times, resulting in a maximum of t valid point clusters found. At the end of each, the points in the list are removed from the group of unassigned points.

1) Cluster Planar Parameters' Estimation: The whole collection of points in each cluster is used to compute its normal and centroid. The cluster centroid, $\hat{P}$, can be computed as the average of the coordinates of all cluster points. The normal, $\hat{N}$, that verifies $||\hat{N}||_2 = 1$, can be computed by minimizing

$$\hat{\boldsymbol{N}} = \arg\min_{\vec{\boldsymbol{N}}} \sum_{k=1}^{l} \left(\left(\vec{\boldsymbol{P_k}} - \hat{\boldsymbol{P}} \right)^{\boldsymbol{T}} \vec{\boldsymbol{N}} \right)^{\boldsymbol{2}} \quad (5)$$

which is a case of the *Total Least Squares* problem, with a well determined solution [13] that can be computed by applying *Singular Value Decomposition* (SVD) over matrix A, where each line is the coordinates of a cluster point minus the coordinates of the centroid. The best normal estimate, is the *right-singular value* of A that corresponds to its smallest *singular value*.

2) Cluster Fusion: For several reasons, often related with measurement errors or the random choice of points, points that should belong in the same cluster are segmented into two or more. We evaluate the difference between the clusters' distance to origin and the angular difference between their normals. If, for any pair of clusters, both are under two user-defined *distance* and *angle threshold*'s, respectively ${}^m\epsilon_d$ and ${}^m\epsilon_\theta$, then both point clusters can be added together, with new *cluster parameters* being computed.

3) Projection to Plane: Since each cluster defines a planar surface, all of its points are projected to the respective plane. This results in a list of coplanar points $\hat{L}$.

C. Construction of Planar Patches

To reduce the amount of data needed, we discard the individual points' information and represent each cluster as a polygon, i.e., a *convex hull*.

1) Computing Plane Base Vectors: Through a change of basis we can convert the points' 3D coordinates to 2D coordinates in the plane they belong to. We compute two *base vectors* of the plane, $\vec{B}_x$ and $\vec{B}_y$ based on the main component of their normals, so that they are all consistent with our 3D models frame.

2) Computing Convex Hulls: By computing the *dot-product* between the points' 3D coordinates and the base vectors we get their metric 2D coordinates on their respective plane. We then use a *convex hull* computing algorithm to get the list of points that define the outer bound of the cluster, $\hat{l}_H$.

3) Rectangular Boundary Fitting: To further compress the amount of data, we use the convex hull points to compute a rectangular boundary that envelops it. This is represented by only 4 vertices, with edges aligned with the plane's base vectors. Its limits are derived from the minimum and maximum values of both the x and y components of the *convex hull* points' coordinates, as illustrated in Figure 2.

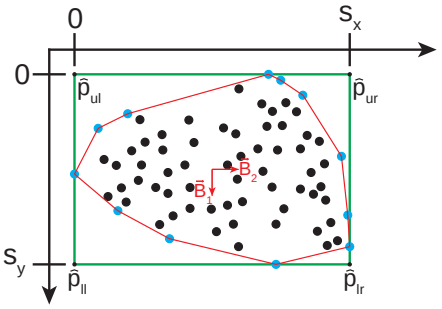


Figure 2: 2D rectangular boundary of a set of points. All the The points (black and blue) are contained inside a rectangular boundary (green), that also envelops the previously computed convex hull (red).

The 2D coordinates of a patch's corners are used to compute the corresponding 3D coordinates by performing a *base vector inversion*.

D. Texture Mapping

The last step in plane segmentation is mapping the texture from the RGB image to the patches. We discretize the patch's rectangular area, according to a *scale factor*, f_s, into *texels*. We then use *backward mapping* to get the RGB values of each *texel* from the correspondent pixel in the *Kinect*'s RGB image. We compute 2D metric coordinates of each *texel* and perform a *base vector inversion* to find its 3D scene coordinates. These are then projected into the original RGB image, returning the respective color value for the *texel*.

We include in the texture a transparency *alpha channel*, that helps identifying the patch areas that were effectively detected by the *Kinect*. Any pixel that successfully gets a valid RGB value assigned has its *alpha* value equal to 1, while all others remain equal to 0.

IV. Construction of Global Models

We now introduce a method to align several individual segmentations and merge them into a global representation. These global models should allow a user to freely navigate the 3D scene and a great reduction in the model size, especially when compared with the traditional approach of global *point clouds*. We take advantage of the planar segments to estimate camera motion, and complement it with visual features-based estimates.

A. Planar Patches based Motion Estimation

Assuming a smooth movement of the sentinel, i.e., with limited rotation and translation between different scan positions, we may postulate that a subset of the elements detected in a scan will also be detected in the subsequent one. Aligning the sets of planes in a sequence of scans requires a 3-step process: a) matching planar patches between consecutive scans, b) computing the rotation between matches, and c) computing the translation.

1) Matching Planar Patches: Assuming we have some knowledge of the sentinel's movement characteristics, we may define two thresholds ${}^{M}\epsilon_d$ and ${}^{M}\epsilon_\theta$, respectively the upper bounds for the sentinel's displacement and rotation.

We take every pair of two patches, one from each scan, and compute the distance and rotation between them. If these are under the thresholds, we compute the rotation between both their normals and apply it to all the patches of the second scan. We check how many pairs of patches are correctly aligned after the rotation, i.e., whose relative orientation is under a second angular threshold ${}^{M}\epsilon_\omega$, lower than ${}^{M}\epsilon_\theta$. Just like a classic RANSAC, we chose the match list with the largest number of matching pairs. This method closely resembles the one introduced by Pathak et al. [15], and since the number of patches in each set is low and only rotations under ${}^{M}\epsilon_\theta$ are accepted, unlikely matches are ignored early in the process speeding up the process.

2) Rotation Estimation: After finding the best match between patches, we compute the best rotation estimate by aligning the corresponding normals. This is equivalent to finding the best rotation between two sets of vectors centered on the origin, which is known as the *Whaba's Problem*, and its solution is well determined by what is known as *Kabsch's Algorithm* [8].

We compute the covariance matrix, ${}^{A}C_B$, between both sets of normal vectors and perform its SVD, yielding ${}^{A}C_B = USV^T$. The optimal rotation between both sets can then be computed as

$$ {}^{\boldsymbol{A}}\boldsymbol{R_B} = \boldsymbol{VDU^T} \tag{6} $$

where D is a diagonal matrix, with all main diagonal elements equal to 1 except for the last which may be -1 to ensure a *right-handed* coordinate system.

3) Translation Estimation: Unlike rotation, we generally need 3 or more mutually non-parallel patches to completely define the translation. We can model this as a minimization problem, in which we try to minimize the sum of squared distances between each pair of patches along their normals. Adapting the minimization problem detailed by Low [7], we may define the minimization problem as

$$ \boldsymbol{T_G} = \arg\min_{\boldsymbol{t}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \left(\left({}^{\boldsymbol{B}}\boldsymbol{P_i^T} - {}^{\boldsymbol{A}}\boldsymbol{P_i^T} + \boldsymbol{t} \right) \cdot {}^{\boldsymbol{B}}\vec{\boldsymbol{N}}_{\boldsymbol{i}}^{\boldsymbol{T}} \right)^{\boldsymbol{2}} \tag{7} $$

where ${}^{A}P_i$ and ${}^{B}P_i$ are the patch centroids and ${}^{B}\vec{N}_i$ is the second patch's normal. This problem may be modeled as a *linear least-squares* minimization problem of the form

$$ \boldsymbol{T_G} = \arg\min_{\boldsymbol{x}} ||\boldsymbol{Ax} - \boldsymbol{b}||_{\boldsymbol{2}}^{\boldsymbol{2}} \tag{8} $$

where

$$ \boldsymbol{A} = {}^{\boldsymbol{B}}\vec{\boldsymbol{N}} \quad \boldsymbol{b} = \begin{bmatrix} {}^{\boldsymbol{B}}\boldsymbol{N_1} \cdot \left({}^{\boldsymbol{A}}\boldsymbol{P_1} - {}^{\boldsymbol{B}}\boldsymbol{P_1} \right) \\ {}^{\boldsymbol{B}}\boldsymbol{N_2} \cdot \left({}^{\boldsymbol{A}}\boldsymbol{P_2} - {}^{\boldsymbol{B}}\boldsymbol{P_2} \right) \\ \vdots \\ {}^{\boldsymbol{B}}\boldsymbol{N_n} \cdot \left({}^{\boldsymbol{A}}\boldsymbol{P_n} - {}^{\boldsymbol{B}}\boldsymbol{P_n} \right) \end{bmatrix} \tag{9} $$

The solution to this problem is then given by

$$ \boldsymbol{Ax} = \boldsymbol{b} \quad \Rightarrow \quad \boldsymbol{x} = \boldsymbol{A^+ b} \tag{10} $$

where A^+ denotes the *Moore-Penrose pseudoinverse* of matrix $A = U\Sigma V^T$ which is given by

$$ \boldsymbol{A^+} = \boldsymbol{V\Sigma^+ U^T} \tag{11} $$

Another translation estimate can be drawn from visual features alone. After converting two subsequent RGB images to grayscale, we use the Speeded Up Robust Features (SURF) algorithm [1] to extract visual features that are common to both and then apply Random Sample Consensus (RANSAC) [4] to filter out the *outlier* features. The estimate, T_V, can be found by first computing the centroids of both sets of 3D feature points and subtracting them.

If the set of matched patches does not contain the required number of mutually non-parallel planes, T_G may not be accurate in all its components. The ideal

solution is to combine both measurements, using as much of T_G as possible and complementing its missing components with information from T_V.

The diagonal elements of the Σ component of matrix A used in equation (10) code how much of matrix A's power is distributed through each of three orthogonal directions. As detailed by Pathak et al. [15], by using only the N most significant elements, we may create a low-rank approximation of matrix A, $\hat{A}$ given by

$$\hat{\boldsymbol{A}} = \sum_{i=1}^{N} \boldsymbol{A_i} = \boldsymbol{U\Sigma_N V^T} \tag{12}$$

where Σ_N is a truncated version of Σ with its $(3-N)$ last diagonal elements replaced by zeros. The valid components of T_G, $\hat{x}$, can then be computed as in (10)

$$\hat{\boldsymbol{x}} = \hat{\boldsymbol{A}}^{+}\boldsymbol{b} = \hat{\boldsymbol{A}}^{+}\boldsymbol{Ax} = \boldsymbol{V\Sigma_N^{+}U^T U\Sigma V^T x} \tag{13}$$

Since U is unitary, $U^T U = I$. Furthermore, since the first N elements of Σ_N^{+} are the inverse of the respective elements of Σ, with the remaining being zeros, we can rewrite equation (13) as

$$\hat{\boldsymbol{x}} = \boldsymbol{V_N V_N^T x} \tag{14}$$

in which V_N is a truncated version of V, with the last $(3-N)$ column vectors replaced by zeros.

Conversely, the remaining components of the solution can be computed by using only the last $(3-N)$ column vectors of V, $\tilde{V}_N$

$$\tilde{\boldsymbol{x}} = \tilde{\boldsymbol{V}}_{\boldsymbol{N}}\tilde{\boldsymbol{V}}_{\boldsymbol{N}}^{\boldsymbol{T}}\boldsymbol{x} \tag{15}$$

Two new matrices thus created,

$$\boldsymbol{M} = \boldsymbol{V_N V_N^T} \qquad \boldsymbol{W} = \tilde{\boldsymbol{V}}_{\boldsymbol{N}}\tilde{\boldsymbol{V}}_{\boldsymbol{N}}^{\boldsymbol{T}} \tag{16}$$

represent respectively the significant and residual directions extracted from the geometric match of planar patches. The best combination of T_G and T_V, T_F can then be computed as

$$\boldsymbol{T_F} = \boldsymbol{MT_G} + \boldsymbol{WT_V} \tag{17}$$

B. Scan Alignment

With the method described above it is possible to compute the rigid body transformation between two subsequent camera positions. The rotations and translations can be used to compute transformation between any two images.

The rotation between two generic poses P_j and P_i is simply the product of all the rotations between pose pairs in between them.

$${}^{i}\boldsymbol{R_j} == \prod_{k=0}^{j-i-1} {}^{i+k}\boldsymbol{R_{i+k+1}} \tag{18}$$

The translation between P_j and P_i can be computed by summing the rotated translations between the poses in between, i.e

$${}^{i}\boldsymbol{T_j} = \sum_{k=0}^{j-i-1} {}^{i}\boldsymbol{R_{i+k+1}}\,{}^{i+k}\boldsymbol{T_{i+k+1}} \tag{19}$$

We consider the first scan to be always at the world's origin and facing directly in the global *Z-axis*' direction. When transforming the patches, not only their coordinates must transformed, but also their normals and base vectors must be rotated.

C. Patch Merging

It is our objective to minimize the size of the final model. For that purpose, we present a method to merge correctly aligned planar patches.

1) Estimating Merged Plane Parameters: Having two matching patches which have already been correctly positioned in their correct place of the 3D scene, their normals and centroids are averaged according to their weight to compute the normal and centroid of the merged patch. We use, as weights, the number of points detected in each of the respective clusters. The number of points of the merged patch is the sum of the number of points of the patches that are being merged.

2) Computing Merged Patch Dimensions: The vertices of each of the patches being merged are projected to the newly defined plane. The new *base vectors* are again computed from the new plane's normal and the 3D coordinates of the projected vertices are converted to 2D planar coordinates. As described before, their individual x and y components are used to compute the outer limits of the new rectangular patch. Additionally, the coordinates of the original patches' vertices inside the new patch are also computed, as illustrated in Figure 3.

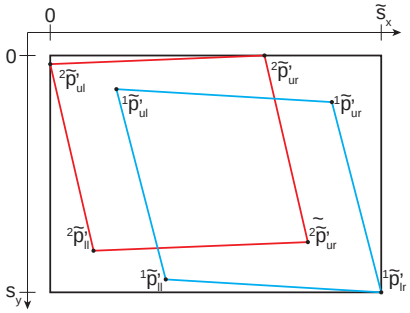


Figure 3: Patch merging example. The boundaries of a new merged patch (black), are defined by the projected vertices of both patches (blue and red).

The 2D coordinates of the merged patch vertices are then projected to the 3D scene, as before, to compute their respective 3D world coordinates.

3) Merging Textures: Merging the textures of both original patches into a single fused patch is a process similar to the *backward mapping* presented before, but that takes into account the number of values ever mapped into every of the final *texels*.

After discretizing the merged patch into *texels*, according to the previously defined *scale factor*, they are mapped back to each of the original patches. This *backward mapping* is used on both the *alpha channel* and texture matrices of both the original patches. Two possibly different RGB values, one for each of the original patches, must be combined to find the actual color value of the resulting merged *texel*. This is achieved by keeping track of the number of values ever mapped into each *texel* in a *multiplicity matrix*, $\hat{M}$.

Matrix $\hat{M}$ is stored with the rest of the patch's information and is projected just as the texture and *alpha channel* matrices. After merging two patches, the resulting *multiplicity matrix* is the sum of the *multiplicity matrices* of both the original patches. When a patch is first created, its matrix $\hat{M}$ is equal to the *alpha channel* matrix. A more detailed explanation of this operation should be consulted on the main work [14].

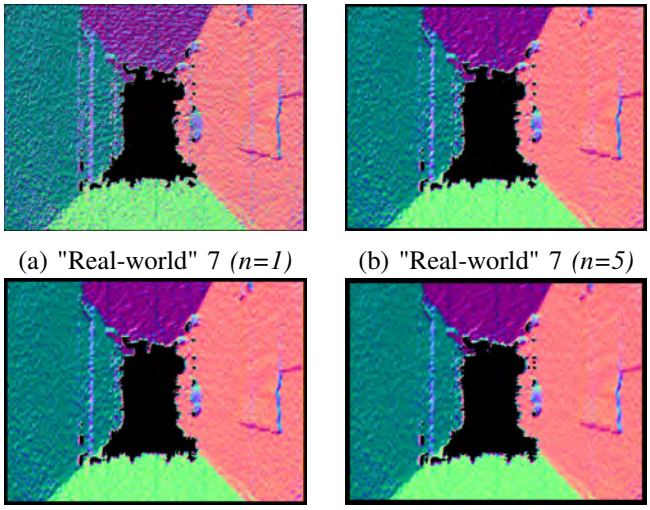
V. Experimental Results

In this chapter we will present and discuss experimental results obtained by using the methodologies previously described. To perform the tests we used a "real-world" dataset, acquired by Lucas [12] using a *Kinect* and a "synthetic" dataset built by Pereira [16], acquired with a "virtual camera" in a Virtual Reality Modeling Language (VRML) world with complex textures. Both datasets feature corridor areas, mainly composed of planar surfaces, i.e, walls, ceiling and floor.

The tests were all performed in a laptop running *Windows 8.1 Pro*, featuring a *quad-core* 64-bit *Intel® Core i7* CPU with a maximum frequency of 1.60 GHz. The methods were implemented and tested using the *32-bit* version of MATLAB R2012 and no parallel processing was used.

A. Normal Estimation Results

The normal estimation process depends on the number of neighbors, n, used during the process which can be defined by the user. Figure 4 illustrates the influence of the choice of n on the normal estimates.



(a) "Real-world" 7 *(n=1)* (b) "Real-world" 7 *(n=5)*

(c) "Real-world" 7 *(n=7)* (d) "Real-world" 7 *(n=10)*

Figure 4: Normal estimation for the "real-world" dataset. Each color is a direct representation of the *XYZ* components of each normal as RGB values.

It is clear that higher values of n create "smoother" surfaces, but also "blur" the edges and create wider areas of unknown values on the borders of the image. The choice of n, as well as the size of the images, s, also influences the time needed to produce estimates. Experimental results reveal that the computation time grows linearly with both values, scaling as $O(sn)$. They are, however, much smaller than those achieved by classic methods, such as the *VectorSVD* implemented by Lucas [12]. For the rest of the experimental results we used $n = 5$, as it provides well balanced results.

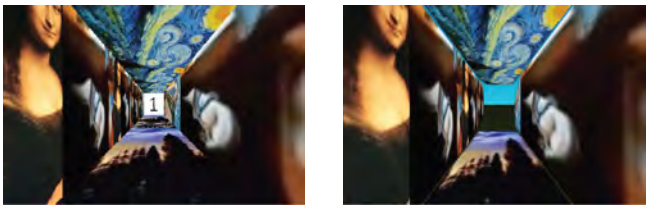
B. Scene Segmentation Results

The parameters that may influence the results of this method are the number of trials, t, the clustering distance and angular error thresholds, ϵ_d and ϵ_θ, the merging distance and angular error thresholds, ${}^m\epsilon_d$ and ${}^m\epsilon_\theta$ and the minimum number of points in each cluster, $minNumPts$. Over many experiments, we arrived at a good combination of parameter values for the used datasets, which are presented in Table I.

Table I: Image segmentation parameter set.

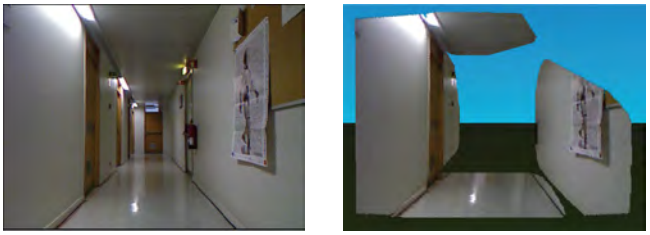
Parameter	t	ϵ_d (mm)	ϵ_θ $(^\circ)$
Value	150	100	15
Parameter	${}^m\epsilon_d$ (mm)	${}^m\epsilon_\theta$ $(^\circ)$	$minNumPts$
Value	300	25	350

The segmentation results for both datasets is drastically different, as can be observed by comparing Figures 5 and 6.



(a) Original (b) Segmentation

Figure 5: Segmentation of "synthetic" image 1, compared to the original "synthetic" image 1.



(a) Original (b) Segmentation

Figure 6: Segmentation of "real-world" image 7 compared to the original "real-world" image 7.

Being error-free, the patches segmented from the "synthetic" dataset contain the majority of points in the *range image*, creating "long" patches that closely resemble the original points-of-view. Even with error added, its effect on segmentation is hardly noticeable. The "real-world" dataset, on the other hand, generates smaller patches slightly more deviated from their right position and orientation.

Each of the representations is stored in Virtual Reality Modeling Language (VRML) file format, with additional Portable Network Graphics (PNG) files for textures with a 1 *pixel* to cm^2 scale, and can be freely rotated and explored.

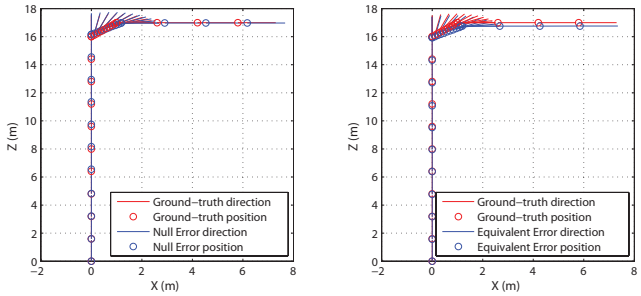
C. Pose Registration Results

By analysis of the camera's movement in the "real-world" and "synthetic" datasets, we can summarize the list of parameters needed for motion estimation in Table II.

Applying the pose estimation method to the "synthetic" dataset results in very good estimates. Figure 7 illustrates the comparison between the camera's *ground-truth* path and the ones estimated with and without

Table II: Odometry estimation parameter set for "real-world" and "synthetics" datasets.

Parameter		${}^{M}\epsilon_d$ (mm)	${}^{M}\epsilon_\theta$ $(^\circ)$	${}^{M}\epsilon_\omega$ $(^\circ)$
Value	***Real-world***	1300	25	10
	Synthetic	1700	25	10



(a) Ground-truth *vs* Null Error estimate (b) Ground-truth *vs* Equivalent Error estimate

Figure 7: Pose comparison between "synthetic" *ground-truth*, *null error* and *equivalent error* estimated paths.

Kinect equivalent error, for the first 27 images. The orientation is very similar between the *ground-truth* and the estimates, in both tests. This is a result of the rotation estimation computed solely from patch comparisons. The position, however, suffers from the accumulation of error, especially from the corner section, where the camera translates and rotates simultaneously.

The "real-world" dataset, with its very feature-poor textures, poses a challenge to the motion estimation algorithm, especially in corner sections. The path estimated by Lucas [12] using ICP is compared to our estimate in Figure 8, using only the first 95 images of the dataset.

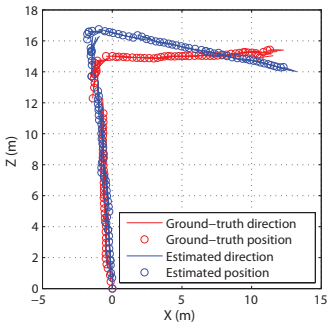


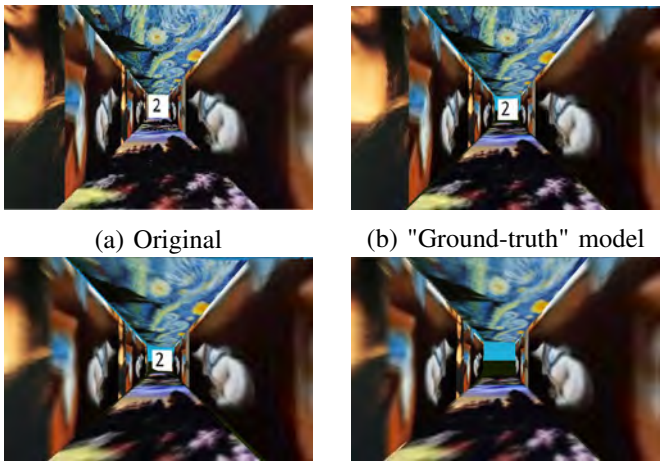
Figure 8: Pose comparison between Lucas' and our estimated paths, for the "real-world" dataset.

It is clear that even though rotation is well estimated in the straight sections, it is overestimated in the corner

section, where the camera traveled and rotated simultaneously. In the straight sections, the deviations transverse to the corridor's length are minimal, but the longitudinal movement of the camera is overestimated, mainly due to the feature-based odometry. In this case, a closed loop around the whole corridor would not be achieved. This indicates the need for the implementation of *loop closure* techniques in future implementations.

D. Global Models from Synthetic RGB-D Data

Combining the scene segmentation and pose registration methods we built global models. Figure 9 illustrates the comparison between the same position views of the original scene, the *ground-truth* odometry based model and estimate-based models with different no error and *Kinect* equivalent error. Only the first 26 of the dataset images were used to build these models.



(a) Original (b) "Ground-truth" model

(c) "Null Error" model. (d) "Equivalent Error" model

Figure 9: Comparison of viewpoints for the "synthetic" dataset between original, *ground-truth*-based model and several estimate-based models with different levels of error added, for image 22.

All the 3 models built are almost indistinguishable from the original image. The only noticeable effect, as the error grows, is a more pronounced "blur" of the textures and the appearance of "secondary patches". Overall, the visual quality of all models is very high.

Applying our method to the first 112 images of the dataset, with Kinect "equivalent" error added, resulted in an almost complete "synthetic" corridor model, very similar to the one from which the images were acquired. Figure 10 illustrates an outside view of the final resulting 3D model.

The model is composed of 13 individual patches, the number strictly necessary to represent the analyzed zone.

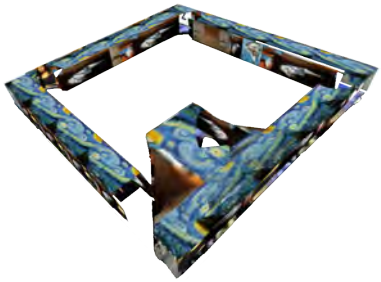
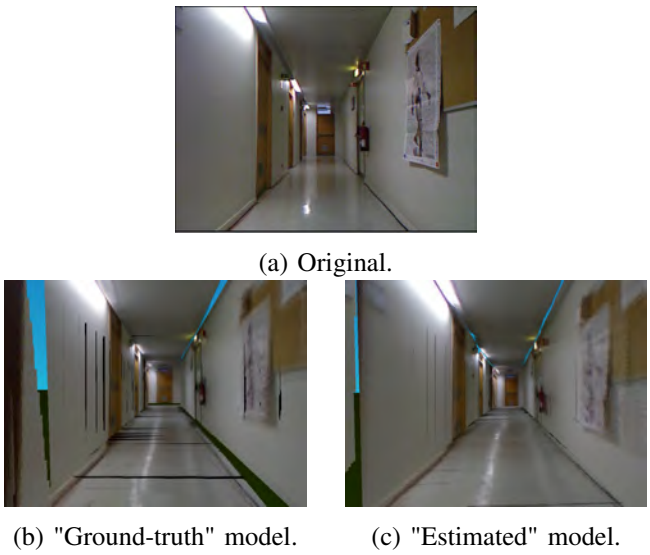


Figure 10: Overview of corridor 3D model, built from the first 112 "synthetic" images with *Kinect equivalent error* added.

The usage of a smaller *texture scale factor*, $f_s = 0.05$, still allows for a good visual quality of the model while keeping its total size to only 2.85 MB. For comparison purposes a point cloud built from the same 112 images, taking into account their resolution, would take up to 140.27 MB of space. Our model, then, represents a compression of approximately 98%.

E. Global Models from Real RGB-D Data

The "real-world" dataset's worse scene segmentation and pose registration results reflect themselves on worse global model constructions, as illustrated in the example of Figure 11.



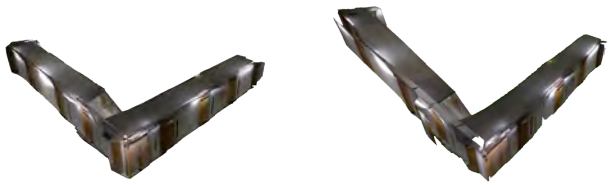
(a) Original.

(b) "Ground-truth" model. (c) "Estimated" model.

Figure 11: Comparison of viewpoints for the "real-world" dataset between original image, *ground-truth*-based model and estimate-based model, for image 7.

The visual quality of the "real-world" models is lower than that of the "synthetic" one. However the general

alignment and texture mapping are successful. The textures suffer from a "blur" effect, especially when relying on estimated odometry.



(a) *Ground-truth* model. (b) *Estimate-based* model.

Figure 12: Comparison between the *ground-truth* and *estimate-based* models, constructed using the first 95 "real-world" images.

The successive merging of patches adds errors in their final positions and orientations. Despite the lower visual quality, we still consider this to be successful representations of the dataset scenario. Overviews of the *ground-truth* and *estimate-based* models are compared in Figure 12. The misalignment of some patches, and the presence of secondary patches is notorious, although the general shape of the corridor is preserved.

Both the *ground-truth* and *estimate-based* models, are under 3.31 MB in total, even though a *texture scale factor* $f_s = 0.1$ was used. The model is built using the first 95 scans in the dataset which would take, in *point-cloud* representation, approximately 167 MB. Again, this represents a compression factor of over 98%.

VI. Conclusions and Future Work

The work described aimed to develop a method to build global, compact scene representations from the data acquired by a low-cost RGB-D camera, to be implemented in robotic home sentinels.

We demonstrated a successful *range image* planar segmentation methodology, capable of detecting and segmenting the main planar surfaces of a corridor-like indoor environment. We also detailed how to successfully estimate camera motion from the segmented planes and combine it with *visual features* based odometry, achieving good estimates for the analyzed datasets.

By combining the scene segmentation and pose estimation methods we were able to build global indoor scene models that are good visual representations of the indoor environments while allowing around high rates of data compression when compared with the traditional *point-cloud* approach.

Future work should focus on implementing the methodology on a real mobile platform with an RGB-D camera and test it *in situ*, to evaluate its strengths, weaknesses and possible improvements. Improvements may include additional metrics to estimate camera motion, improve the quality of patch textures and increase the overall speed to take advantage of the *Kinect*'s full frame rate. Furthermore, new metrics like surface curvature may be taken into account as a way to better segment range images and make the models closer to reality.

References

[1] H. Bay, T. Tuytelaars, and L. Van Gool. SURF: Speeded Up Robust Features. In *Proceedings of the 9^th European Conference on Computer Vision (ECCV 2006)*, pages 404–417, 2006.

[2] J. Biswas and M. Veloso. Fast Sampling Plane Filtering, Polygon Construction and Merging from Depth Images. In *Proceedings of the 2011 Robotics: Science and Systems Conference (RSS 2011)*, 2011.

[3] Y. Chen and G. Medioni. Object Modeling by Registration of Multiple Range Images. In *Proceedings of the 1991 IEEE International Conference on Robotics and Automation (ICRA 1991)*, volume 3, pages 2724–2729, 1991.

[4] M. Fischler and R. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Communications of the ACM*, 24:381–395, June 1981.

[5] D. Holz, S. Holzer, R. Rusu, and S. Behnke. Real-time Plane Segmentation using RGB-D Cameras. In *RoboCup 2011: Robot Soccer World Cup XV*, pages 306–317, 2012.

[6] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. In *Proceedings of the 24^th Annual ACM Symposium on User Interface Software and Technology (UIST 2011)*, pages 559–568, 2011.

[7] Low K. Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration. Technical report, Department of Computer Science, University of North Carolina, 2004.

[8] W. Kabsch. A Solution for the Best Rotation to relate two Sets of Vectors. *Acta Crystallographica Section A*, 32:922–923, September 1976.

[9] K. Khoshelham. Accuracy Analysis of Kinect Depth Data. In *Proceedings of the 2011 International Society for Photogrammetry and Remote Sensing Workshop on Laser Scanning (ISPRS Workshop on Laser Scanning 2011)*, pages 133–138, 2011.

[10] K. Khoshelham and S. O. Elberink. Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications. In *Sensors 2012*, volume 12, pages 1437–1454, 2012.

[11] K. Klasing, D. Althoff, D. Wollherr, and M. Buss. Comparison of Surface Normal Estimation Methods for Range Sensing Applications. In *Proceedings of the 2009 IEEE International Conference on Robotics and Automation (ICRA 2009)*, pages 3206–3211. IEEE, 2009.

[12] R. Lucas. Building World Representations using Color-Depth Cameras. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisbon, Portugal, April 2013.

[13] I. Markovsky and S. Van Huffel. Overview of Total Least-squares Methods. *Signal Processing*, 87:2283–2302, October 2007.

[14] J. Mendonça. Reconstruction of Lightweight 3D Indoor Representations using Color-Depth Cameras. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisbon, Portugal, June 2014.

[15] K. Pathak, N. Vaskevicius, J. Poppinga, M. Pfingsthorn, S. Schwertfeger, and A. Birk. Fast 3D Mapping by matching Planes extracted from Range Sensor Point-clouds. In *Proceeding of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2009)*, pages 1150–1155, 2009.

[16] D. Pereira. Robotic Home Sentinel. Master's thesis, Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisbon, Portugal, April 2014.

[17] J. Poppinga, N. Vaskevicius, A. Birk, and K. Pathak. Fast Plane Detection and Polygonalization in Noisy 3D Range Images. In *Proceeding of the 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2008)*, pages 3378–3383, 2008.

[18] C. Raposo, M. Lourenço, J. Barreto, and M. Antunes. Plane-based Odometry using an RGB-D Camera. In *Proceedings of the 2013 British Machine Vision Conference (BMVC 2013)*, pages 114.1–114.11, 2013.

[19] V. Sanchez and A. Zakhor. Planar 3D Modeling of Building Interiors from Point Cloud Data. In *Proceedings of the 2012 19^{th} IEEE International Conference on Image Processing (ICIP 2012)*, pages 1777–1780, 2012.

[20] K. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A Probabilistic, Flexible, and Compact 3D Map Representation for Robotic Systems. In *Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA 2010)*, 2010.

[21] H. W. Yoo, W. H. Kim, J. W. Park, W. H. Lee, and M. J. Chung. Real-time Plane Detection based on Depth Map from Kinect. In *Proceedings of the 44^{th} International Symposium on Robotics (ISR)*, pages 1–4, 2013.