Building World Representations using Color-Depth Cameras

Ricardo Mileu Cavalheiro Lucas
Instituto Superior Técnico, Lisbon, Portugal
ricardolucas@ist.utl.pt

***Abstract*—Clouds of points acquired by color-depth cameras are commonly fused with Iterative Closest Point (ICP) like algorithms. ICP is known to be effective to perform the fusion of the clouds but slow to converge when the clouds are not close to each other. Besides the computational time issue, experiments show that a good initial guess of the transformations is important for an effective fusion. This dissertation proposes the use of a *Manhattan world* hypothesis to estimate the camera orientation and, therefore, simplify the ICP problem to estimate mostly just 3D translations. Additionally it is proposed the estimation of translations by the registration of principal planes in order to achieve more robustness to local minima. The proposed algorithm is tested in a large indoor environment and yields promising results by combining accurately the visual and depth information, and therefore effectively mapping the scenario.**

I. Introduction

Depth cameras have been studied and developed for several years in computer vision and robotics. However, these sensors have been too expensive to be widely used. Recently, inexpensive cameras appeared in the market encompassing both depth and RGB color sensors. These cameras, usually called RGB-D cameras, have an important role in the development of many applications such as Simultaneous Localization And Mapping (SLAM), people tracking or object recognition.

The main goal of this dissertation is to build a colored three dimensional representation of an indoor scenario, which has many practical applications. A more technical application of these 3D scenarios is the use of a RGB-D camera to perform a camera calibration based in image lines and in the the Direct Linear Transformation [1], where the 3D lines can be acquired using a textured three dimensional representation. This also brings new possibilities to entertainment. For example, future gamers could create their own playing scenarios by acquiring data with a low-cost RGB-D camera.

In this dissertation, a Microsoft Kinect camera is carried along the corridors in order to acquire color and depth images of the scenario. The images are taken with steps ranging from 0.3 to 1 meters, which are large steps comparing to RGB-D SLAM methods, where the acquisition and fusion is in real time (limited by the camera frame rate) and, therefore, the steps are very small. Despite these large steps between consecutive point clouds (group of 3D points) there are about 70% of overlapped points after the reconstruction.

Further in this dissertation is described an initial attempt to create a complete three dimensional representation that was unsuccessful and presented another method that returns better results.

A. Related Work

One of the most studied problems in computer vision and robotics is the SLAM problem. Shen et al. [2] successfully presented a stochastic differential equation based exploration algorithm for mapping single and multi-floor buildings with a Microsoft Kinect mounted at the top of an aerial vehicle. Huang et al. [3] uses a quadrotor helicopter with an RGB-D camera on board that provides SLAM, using the 3D model of the scenario for trajectory planning. Wurm et al. [4] presented an approach for modeling 3D environments using probabilistic occupancy estimation and a tree based representation named OctoMaps.

The most popular method for 3D shapes registration between two point clouds is the Iterative Closest Point (ICP) that was originally proposed by Besl and McKayin [5]. Nowadays, this method has many variations. Armesto et al. [6] developed a generalization of the Metric-Based Iterative Closest Point (MbICP) to match three dimensional data. Segal et al. [7] proposed a generalized ICP which attaches a probabilistic model to the minimization step of the algorithm, taking into account the surface information of both clouds of points.

The Robot Operating System (ROS) [8], which is an open-source software, implemented in C++, Python and other languages, provides hardware abstraction, visualizers and others, and covers multiple subjects in robotics. In particular ROS allows acquiring and processing RGB-D data from Microsoft Kinect like cameras [9].

B. Problem Formulation

The 3D modeling is possible given a dataset with several depth and RGB color images obtained with a RGB-D camera. The faced problem is to combine all the acquired data in one global colored point cloud, without incorrect overlap of the points. This requires a camera calibration to match depth and color information and an additional processing over both images.

This dissertation presents two approaches to perform the reconstruction. Both have in common the Iterative Closest Point algorithm, but one is essentially based on image features and the other is based on surface information and edge points.

II. Color-Depth Cameras

In this section is described the internal calibration method of the camera and the relation between 3D points and their respective projection onto the image plane based on the pin-hole camera model. It is also presented the color-depth camera

model that allows changes between coordinate systems. The computation of these transformations enables the combination of data obtained with both RGB and depth cameras. Lastly is provided an overview of the calibration result for a single image.

A. Pin-hole Camera Model

The pin-hole camera model maps the 3D to the 2D projective spaces. Given the projection matrix $\mathbf{P}$ and the homogeneous coordinates of one 3D point $\mathbf{M}$, it is possible to project these points in the image plane,

$$\lambda m = PM \quad \Leftrightarrow \quad \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \ [R \,|\, t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1}$$

where the screen coordinates (u, v), in pixels, are the projection of the 3D points (X, Y, Z) in the image plane, λ is an arbitrary scale factor, $\mathbf{P} \in \mathbb{R}^{3\times4}$ is the projection matrix that describes the mapping of a pin-hole camera, $\mathbf{K} \in \mathbb{R}^{3\times3}$ is a upper triangular matrix called the intrinsic parameters matrix or camera matrix, $\mathbf{R} \in \mathbb{R}^{3\times3}$ is the rotation matrix and $\mathbf{t} \in \mathbb{R}^{3\times1}$ is the translation. The entries of the matrix $\mathbf{K}$ are the intrinsic camera parameters, which are independent of the camera position and orientation [10] but are affected by a zoom operation.

Assuming that modern cameras have rectangular pixels [11] (the skew parameter is zero), the relation between screen coordinates measured in pixels (u, v), and the correspondent value in meters $(\mathbf{x},\mathbf{y})$ is given by:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} fs_u & 0 & o_u \\ 0 & fs_v & o_v \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{2}$$

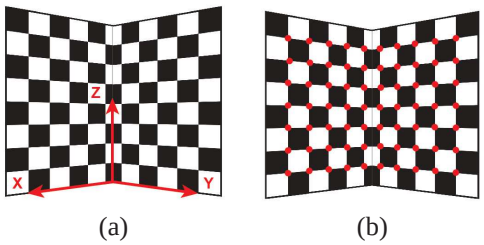
where f is the focal length, which is the distance, in millimeters, between the center of projection and the image plane, s_u and s_v are the conversion factors that converts from meters to pixels (so fs_u and fs_v are in pixels), o_u and o_v are the coordinates, also in pixels, of the principal point (center of the image) in the image plane.

B. Pin-hole Camera Calibration

Relatively to the necessary camera calibration, in order to obtain the intrinsic and extrinsic camera parameters, there are several methods. One of the most popular is the calibration toolbox implemented in Matlab by Jean-Yves Bouguet [12] based on the calibration method proposed by Zhang [13].

In equation 1, where $\mathbf{P}$ is unknown, the set of 3D points (X, Y, Z) can be acquired by creating a referential and select a set of points. The used reference frame in this dissertation and the selected points are shown in figure 1. The two dimensional screen coordinates (u, v) in the image plane are also needed and they are directly obtained by selecting the same points in the image plane.

Given $\mathbf{M} = [X\ Y\ Z\ 1]^T$ and $\mathbf{m} = [u\ v\ 1]^T$ allows computing the projection matrix $\mathbf{P}$, which has a known 3×4 dimension:



(a) (b)

Fig. 1. (a) World reference frame. (b) Selected points.

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix}. \tag{3}$$

From equation 1 results $2n$ linear equations:

$$u_i = \frac{p_{11}X_i + p_{12}Y_i + p_{13}Z_i + p_{14}}{p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}} \tag{4}$$

$$v_i = \frac{p_{21}X_i + p_{22}Y_i + p_{23}Z_i + p_{24}}{p_{31}X_i + p_{32}Y_i + p_{33}Z_i + p_{34}} \tag{5}$$

Each equation has eleven independent variables, meaning that a minimum of eleven equations are needed, so at least six points (n≥6) are required to compute $\mathbf{P}$. From equations 4 and 5 one can obtain a homogeneous equations system with the form $Ap = 0$, where

$$p = [p_{11} \quad p_{12} \quad \dots \quad p_{34}]^T \tag{6}$$

As shown in [10], $\mathbf{p}$ is the eigenvector associated to the smallest eigenvalue of A^TA. With the vector $\mathbf{p}$, the projection matrix can be built and described as

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} = [KR \quad Kt] = [A\ B] \tag{7}$$

where $\mathbf{A} \in \mathbb{R}^{3\times3}$ corresponds to the first three columns of $\mathbf{P}$ and $\mathbf{B} \in \mathbb{R}^{3\times1}$ is the last column.

Since $\mathbf{R}$ is a rotation matrix and, therefore, an orthogonal matrix and $\mathbf{K}$ is an upper triangular matrix, the $\mathbf{A}$ matrix can be factorized into a product of an orthogonal matrix and an upper triangular matrix using a QR decomposition [14]. To obtain the inverse, i.e. the product of an upper triangular matrix by an orthogonal matrix, the decomposition must be applied to $\mathbf{A}^{-1}$. However, this returns $\mathbf{K}^{-1}$ and $\mathbf{R}^{-1}$ (equation 8) matrices which must be inverted.

$$\mathbf{Qr}(A^{-1}) = K^{-1}R^{-1} \tag{8}$$

The translation $\mathbf{t}$ is obtained multiplying the matrix $\mathbf{K}^{-1}$ by the matrix $\mathbf{B}$, which is the known last column of $\mathbf{P}$, $t = K^{-1}\,[p_{14}\ p_{24}\ p_{34}]^T$. Finally the matrix $\mathbf{K}$ must be normalized $\hat{K} = K/\,\|K\|$.

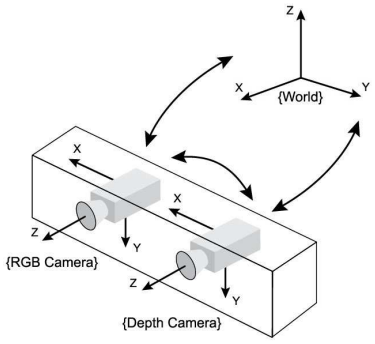


Fig. 2. Coordinate Systems.

C. Color-Depth Camera Model

This section describes Color-Depth cameras and, in particular, the Microsoft Kinect.

There are three different coordinate systems on which the point cloud can be represented: RGB camera, depth camera and world coordinate systems (see figure 2). In order to associate color to each point of the point cloud, it is necessary to find the transformations that allow changes between the coordinate systems.

The rotations, ${}^{rgb}R_w$ and ${}^{d}R_w$, and the translations, ${}^{rgb}t_w$ and ${}^{d}t_w$, obtained with the calibration process, allow the transformation of 3D points from the world coordinate system $\{W\}$ to both image $\{RGB\}$ and depth $\{d\}$ camera coordinate systems, ${}^{rgb}M$ and ${}^{d}M$, respectively:

$$ {}^{rgb}M = {}^{rgb}R_w \ {}^{w}M + {}^{rgb}t_w \tag{9} $$

$$ {}^{d}M = {}^{d}R_w \ {}^{w}M + {}^{d}t_w \tag{10} $$

where ${}^{a}M = [X \ Y \ Z]^T \in \mathbb{R}^{3\times N}$ are 3D points in the coordinate system $\{a\}$, ${}^{b}R_a$ and ${}^{b}t_a$ denote the rotation and translation from $\{a\}$ to $\{b\}$ coordinate systems.

Using equations 9 and 10 it is possible to make the correspondences between the three coordinate systems,

$$ {}^{rgb}M = \underbrace{{}^{rgb}R_w \ {}^{d}R_w^{-1}}_{{}^{rgb}R_d} \ {}^{d}M + \underbrace{{}^{rgb}t_w - {}^{rgb}R_w \ {}^{d}R_w^{-1} \ {}^{d}t_w}_{{}^{rgb}t_d} \tag{11} $$

$$ {}^{d}M = \underbrace{{}^{d}R_w \ {}^{rgb}R_w^{-1}}_{{}^{d}R_{rgb}} \ {}^{rgb}M + \underbrace{{}^{d}t_w - {}^{d}R_w \ {}^{rgb}R_w^{-1} \ {}^{rgb}t_w}_{{}^{d}t_{rgb}} \tag{12} $$

In general, one pixel (u, v) in the RGB image does not correspond directly to the same pixel (u, v) in the depth image and vice-versa (see figure 3). The first step to find the correspondence $(u, v)_d \leftrightarrow (u, v)_{RGB}$ is to get the conversion from the depth image pixels to meters using the pin-hole projection model (equation 2) [10], where $\mathbf{K}$ is the intrinsic parameters matrix ${}^{\mathbf{d}}\mathbf{K}$, obtained during the calibration

$$ \begin{cases} u = fs_u \ x + o_u \ [pixel] \\ v = fs_v \ x + o_v \ [pixel] \end{cases} \Rightarrow \begin{cases} x = \frac{u - o_u}{fs_u} \ [m] \\ y = \frac{v - o_v}{fs_v} \ [m] \end{cases} . \tag{13} $$

Now it is possible to obtain the 3D points in the depth image coordinate system, ${}^{\mathbf{d}}\mathbf{M}$, using equation 14, where $\mathbf{x}$ and $\mathbf{y}$ are computed above and the $\mathbf{Z}$ axis values of each point are known as they are the depth in meters obtained with the Kinect:

$$ {}^{d}M = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} Zx \\ Zy \\ Z \end{bmatrix} \in \mathbb{R}^{3\times N} \tag{14} $$

The 3D points in the RGB image coordinate system, ${}^{\mathbf{rgb}}\mathbf{M} = [X \ Y \ Z]^T$, are obtained by applying the rigid transformation ${}^{rgb}R_d \ {}^{d}M + {}^{rgb}t_d$ (equation 11). The next step it is just reversing the process, reconverting the 3D points in 2D and then convert from meters to pixels, where the intrinsic camera parameters are now related to ${}^{\mathbf{rgb}}\mathbf{K}$.

In order to transfer just the essential data, Cartesian X, Y, Z points are usually acquired by the camera $Z(u, v)$ where u, v are image points i.e. perspective projection (projective) coordinates.

The data completion problem consists on finding the 3D coordinates $(X(u, v), \ Y(u, v), \ Z(u, v))$ from $Z(u, v)$ and the intrinsic parameters matrix K. The general case can be written as

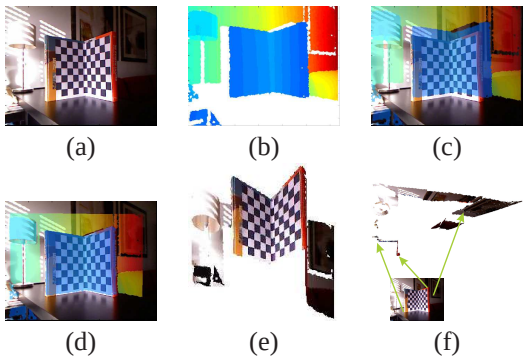
$$ \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \Leftrightarrow \lambda K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{15} $$

Let $A = K^{-1}$ and $m = [u \ v \ 1]^T$, then $\lambda \ A(3,:) \ m = Z$ and thus $\lambda = Z/(A(3,:) \ m)$, where $A(i,:)$ denotes the i^{th} line of the matrix A.

$$ \begin{cases} X = \lambda \ A(1,:) \ m = Z \ A(1,:)m/(A(3,:)m) \\ Y = \lambda \ A(2,:) \ m = Z \ A(2,:)m/(A(3,:)m) \end{cases} \tag{16} $$

In this case the K matrix expression is known and is shown in equation 2, then

$$ \begin{cases} X(u, v) = \left(\frac{u}{fs_u} - \frac{o_u}{fs_u} \right) Z(u, v) \\ Y(u, v) = \left(\frac{v}{fs_v} - \frac{o_v}{fs_v} \right) Z(u, v) \end{cases} . \tag{17} $$



(a) (b) (c) (d) (e) (f)

Fig. 3. Calibration result. RGB and Depth images (a,b). Direct superposition of RGB and Depth images before and after calibration (c,d). RGB image superimposed on the depth data (point cloud), inferior-view and top-view (e,f).

Figure 3 illustrates the calibration of a color-depth camera. One obtains a correct superposition of the Depth and RGB data after calibration and perspective correction (see figure 3(d)).

Figure 3(e) represents the three dimensional point cloud of the used calibration pattern in the depth camera coordinate system (${}^{d}M$) with the corresponding color information for each pixel obtained by mapping the pixels from the RGB image to the depth image. The perspective of figure 3(f) allows assessing qualitatively the precision of the calibration by displaying a top view of the calibration pattern. The observed angle between the faces of the calibration pattern matches, approximately, the true value of 90^o.

III. Building Global 3D Models

This section describes standard methods for building global 3D representations of scenarios imaged by moving RGB-D cameras. Fusing multiple clouds of points is a central problem. The Iterative Closest Point algorithm (ICP), described in detail in this section, is a commonly used methodology for fusing clouds of points. Matching features observed in the RGB images allow fusing clouds of points separated by distances much larger than the ones typically found, for example, in Simultaneous Localization And Mapping (SLAM).

A. Iterative Closest Point

The Iterative Closest Point (ICP) algorithm computes the optimal rigid transformation, i.e. rotation and translation, between two point clouds (algorithm 1). ICP is essentially composed by three main steps, namely (i) matching points between the clouds, (ii) finding the rigid transformation given the matched points, (iii) applying the transformation and repeating the process. The steps are run cyclically until convergence, i.e. the computed transformation does not change significantly.

ICP starts with an initial estimation for the transformation T between the first two clouds. Assuming that one wants to register point cloud $B \in \mathbb{R}^{3\times n}$ with point cloud $A \in \mathbb{R}^{3\times k}$. The matching consists in finding for each point of $T.B$ the nearest point in A.

Finding a transformation between matched points usually involves minimizing a sum of point-to-point Euclidean distances. Considering the case of three dimensional data, there are some alternative cost functions. Chen and Medioni [15] proposed a point-to-plane distance, minimizing the error based on the local surface normal of one scan. As mentioned before, Segal et al. [7] proposed a plane-to-plane alternative, which use probabilistic techniques to increase the robustness of the algorithm and take into account the surface normal information from both scans. With the first estimation of the transformation computed, the iterative process repeats until T converge.

The ICP algorithm has several implementations and variations for matching points and minimization strategies between point clouds. Given a set of points $M = [p_1, p_2, ..., p_n] \in \mathbb{R}^{3\times n}$ and one query point q, the matching consists on finding the closest point to q in the dataset M. There are several methods for this Nearest Neighbor Search (NNS), all with different approaches and different computation complexity but with a common objective, find the nearest neighbor.

The used matching method is based on a K-Dimensional Tree (k-d tree) which is a binary search tree with n leaves. This method initially creates a space partitioning data structure representing a k-d tree and, in the case of 3D data, the k-d tree structure is composed with 3D points split alternatingly by X, Y and Z axis by finding the median of the points. The point correspondent to the first coordinates median is the root of the k-d tree [16].

Algorithm 1 Standard ICP

Require: $T_0 \leftarrow \begin{bmatrix} R_0 & t_0 \\ 0 & 1 \end{bmatrix}$
Require: $A \leftarrow [a_1, a_2, ..., a_k]$
Require: $B \leftarrow [b_1, b_2, ..., b_n]$
$T \leftarrow T_0$
while T not converged **do**
  for $i \leftarrow 1$ to n **do**
    $p_i \leftarrow T.b_i$
    $\hat{p}_i \leftarrow$ find closest point of $T.b_i$ in A
    d $\leftarrow \|\hat{p}_i - p_i\|$
    if $d > d_{max}$ **then**
      $w_i \leftarrow 0$
    else
      $w_i \leftarrow 1$
    end if
  end for
  $T \leftarrow \arg_T \min\{\sum_{j=1}^{n} w_j.\|p_j - \hat{p}_j\|^2\}$
end while
Output: T

Having all points matched allows computing the rigid transformation. There are essentially three cases of interest: point-to-point, point-to-plane and plane-to-plane, which are detailed in the next paragraphs. Some of these cases have a local surface normal associated to each point, which allows to characterize the surface where the points are located.

1) Point-to-Point: This method is based on the Single Value Decomposition (SVD) approach, which is robust, easy to implement and usually the fastest method. To compute the rotation from one point cloud ($\mathbf{B}$) to another ($\mathbf{A}$), it is necessary to re-center both point clouds by removing their centroids μ, which removes the translation component, where $\mathbf{N}$ is the number of points of each point cloud

$$\mu = \begin{bmatrix} \frac{\sum_{i=1}^{N} x_i}{N} & \frac{\sum_{i=1}^{N} y_i}{N} & \frac{\sum_{i=1}^{N} z_i}{N} \end{bmatrix}^T . \tag{18}$$

With only the rotation to deal, this is the orthogonal Procrustes problem, which consists on finding the orthogonal matrix ($\mathbf{R}$) that most closely maps the point cloud $\mathbf{B}$ to $\mathbf{A}$ by minimizing the Frobenius norm in equation 19, where $R^T R = I$ and $det(R) = 1$:

$$R^* = \arg_R \min \|A - RB\|_F \tag{19}$$

In order to compute the rotation, the orthogonal Procrustes problem can be solved using the SVD of the matrix $A^T B$ [17]. To compute the translation from the point cloud $\mathbf{B}$ to $\mathbf{A}$ it is just apply the rotation $\mathbf{R}$ to the centroid of $\mathbf{B}$ and then subtract this product from the centroid of $\mathbf{A}$ as shown in equation 20

$$t = \mu_A - R\ \mu_B \tag{20}$$

2) Plane-to-Plane: This method minimizes the distances between planes, taking advantage of indoor scenes and it requires surface information from the two point clouds. It is necessary to compute the normal vector associated with every point using the same method as in point-to-plane, i.e. considering the k-nearest neighbors. This method minimizes a weighted distances cost

$$T^* = \arg_T \min\{\sum_{i=1}^{N} w_i.\|T.b_i - a_i\|^2\} \quad (21)$$

where $\mathbf{d} = \|T.b_i - a_i\|$ is the distance, $\mathbf{w} = (M\ x)^2$ is the weight,

$$w_i = (M\ x_i)^2 + 1 = M^2(1 - < n_a, n_b >)^2 + 1 \quad (22)$$

where $\mathbf{M^2}$ is the maximum weight and $\mathbf{x_i} = 1 - |< n_a, n_b >|$ is the variable that defines if the planes of each point are similar. Since the inner product between two normal vectors $\in [-1, 1]$ returns ≈ 0 if they have different directions, ≈ 1 if they have the same direction and ≈ -1 if they are opposite, in order to make the inner product $\in [0, 1]$ it's necessary to use the module (planes are similar if normal vectors have either the same direction or the opposite), subtracting it to 1 to penalize points belonging to different planes, giving them an higher weight.

B. Using Texture Information

In case one has sparsely acquired point clouds, ICP falls easily in faraway local minima. In order to get close to the correct registration of two point clouds, one can incorporate a prior step matching visual features.

The registration of consecutive RGB-D images can be based on the Scale Invariant Feature Transform (SIFT) algorithm, developed by David Lowe [18]. Figure 4 shows matched SIFT features between two images of our dataset.

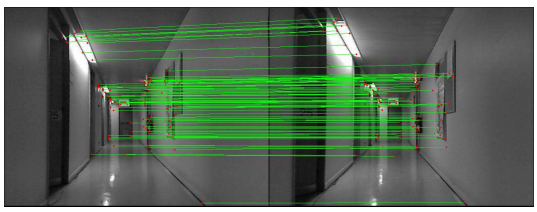


Fig. 4. Example of correspondent features between two consecutive images.

Given image features matched in consecutive point clouds, and knowing the 3D coordinates of all matched (image) points, the rigid transformation is computed using the point-to-point method, resulting in a rough estimation for the rotation and translation between both point clouds.

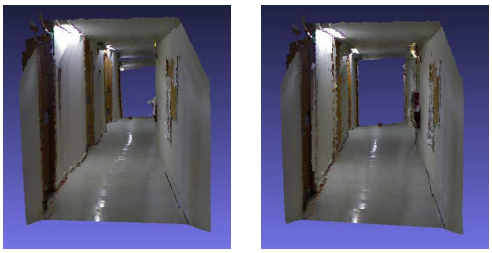
The 3D points are obtained from the completion of depth information associated to the matched image points. Finally, as before, the rigid transformation charactering the camera motion is computed as a Procrustes problem [17].

The repetitive nature of our scenario, corridors of white walls with similar doors, implies that the registration of visual features is prone to return many outliers, i.e. points that are not well matched. In order to decrease the number of outliers, and therefore obtain better estimates of the inter-cloud transformations, the Random Sample Consensus (RANSAC) algorithm [19] is used.

In a first experiment we targeted fusing the point clouds associated to the 40 RGB-D images acquired along a corridor which is about 16 meters long. Since the steps that the camera took are large, between 0.5 and 1 meters, texture based registration was necessary to provide initial inter cloud registration guesses for the ICP algorithm. Figure 5 shows the result of fusing the 40 clouds of points. The ICP algorithm was based in Point-to-Point registration.

Fig. 5. Forty fused point clouds along one corridor.

It is possible to observe that in figure 5 the planar floor appears bended. This effect occurs typically when successive transformations between point clouds are computed, and small registration errors are accumulated along many transformations. In order to find a better estimation for the rigid transformation, the ICP based in Plane-to-Plane registration was also applied in a second experiment. Figure 6(b) shows an improved fusion when compared with figure 6(a) (the same reconstruction as in 5, but in a different perspective).



(a) (b)

Fig. 6. (a) Thirty-two point clouds with only Point-to-Point minimization. (b) Thirty-two point clouds with Point-to-Point and Plane-to-Plane minimization.

The plane-to-plane method requires computing a normal vector for each point of both point clouds. This is a computational bottleneck of ICP based in Point-to-Point registration.

C. Conclusion

In this section were described some approaches to create three dimensional representations by fusing 3D point clouds using ICP based methods. Two alternative methods, point-to-point and plane-to-plane, were then considered to compute the transformation between the consecutive point clouds. The correspondence of points between RGB images was successfully computed by matching SIFT features. Finding correspondences using plane-to-plane matching was found to produce better results due to using more information about the surfaces, as noted in other published works.

Despite the methods described in this section allowed to obtain global clouds by fusing local clouds of points, the resulting reconstructions were found to have a number of issues. The ICP is known to be prone to local minima, especially when T_0 is far from T. The accumulation of small registration errors, associated to local minima, turned out visible after fusing a number of clouds. More precisely, the floors of the reconstructed corridors have been found to be clearly curved. In addition, our datasets were acquired in a modern office scenario that is texture-poor, and therefore difficult to register using image features.

The proposed methodologies in this section are therefore not ideal for our datasets. In the next section we propose a method where depth data provides the first estimate of the inter-point-clouds transformation. In particular we explore the *Manhattan world* hypothesis, where a good initial guess for the transformation is provided by detecting principal planar directions and matching them along consecutive point clouds.

IV. Fusing Clouds of Points in a Manhattan World

In the previous sections we described how to obtain a textured 3D representation from color and depth images. In particular, the last section proposed an attempt to fuse all clouds of points in one global point cloud but the achieved results were not the expected, leading to a visible accumulation of small registration errors. The main difficulty experienced was to get a good initial estimation for the transformation between point clouds, especially in the corners of the corridor due to the insufficient texture. In this section is described an approach based in the so called *Manhattan world* scenes [20], [21], where all surfaces in the world are aligned with three main directions.

A. Global coordinate system

The assumption that the scene can be described as a *Manhattan world* provides a natural coordinate system. More precisely, one can choose the normals to the three main surface orientations to build a Cartesian coordinate system. The first RGB-D image defines the naming (X, Y, Z) of the three axis of the coordinate system. This naming can be maintained by registering the three main directions of consecutive images. Hence, the first step is to compute the three main directions of the surfaces of the first RGB-D image.

In order to characterize surfaces of the point cloud, one can compute a surface normal, v_i, for each point, x_i. Considering a neighborhood around x_i allows formulating a least squares normal estimation problem [22]:

$$(v_i, d_i) = \arg_{v,d} \min \sum\nolimits_j |x_{ij}^T v - d|^2 \tag{23}$$

where d_i denotes distance of the (local) plane to the origin, and x_{ij} represent the j^{th} point of the N nearest neighbors of x_i (in this thesis the used value was $N = 100$). More precisely, one builds the covariance matrix, $Cov(x_i) := \sum_j (x_{ij} - o_i).(x_{ij} - o_i)^T$, and computes v_i from the smallest eigenvalue of $Cov(x_i)$ obtained with Singular Value Decomposition (SVD).

Local surface normals allow clustering the point cloud into three main (surface) orientations (directions). In other words, one defines a Cartesian coordinate system using Principal Component Analysis (PCA), i.e. the Karhunen-Love transformation. The PCA is obtained with the SVD of the surface normals. The eigenvectors corresponding to the largest eigenvalues of the covariance matrix define the principal directions [23], [24], [25].

Let $\{W\}$ denote the world coordinate system. The first RGB-D image is used to set $\{W\}$ as

$${}^w T_1 = \begin{bmatrix} {}^w R_1 & {}^w t_1 \\ 0 & 1 \end{bmatrix} \tag{24}$$

where ${}^w R_1$ denotes the orientation of the camera, in its first location, relative to $\{W\}$. The origin of $\{W\}$ is set equal to the position of the camera in its first location, i.e. ${}^w t_1 = [0\ 0\ 0]^T$.

The orientation of the camera relative to the world is computed using the PCA,

$${}^w R_1 = \text{PCA}_{1:3}(V_{t=1}) \tag{25}$$

where $V_{t=1} = [v_1\ v_2\ \cdots\ v_N]$ are the local surface normals of the first RGB-D image, and $\text{PCA}_{1:3}(V_{t=1})$ denote the first three PCA components of $V_{t=1}$. The columns of the camera orientation, ${}^W R_1 = [n_1\ n_2\ n_3]$, describe the orientations of the axis of $\{W\}$, namely $(n_1,\ n_2,\ n_3)$ denote the (X, Y, Z) directions or, in our experiments, the North-to-South, sky-to-ground and West-to-East directions. Figure 7 illustrates the segmentation objective. It is necessary to find three principal directions, n_1, n_2 and n_3 (see in figure 7 the areas marked in red, green and blue, respectively).

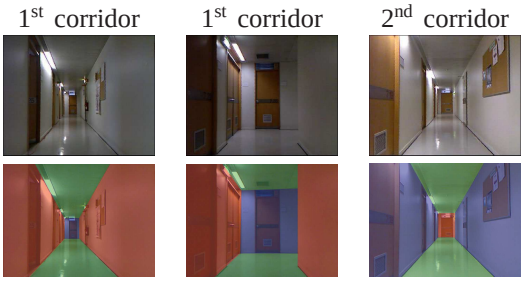


Fig. 7. Segmentation idea of the corridor scenario.

B. Camera orientation

As described in the previous section, the global coordinate system is defined by the PCA of the local normals of the point cloud corresponding to the first image. The main idea for maintaining the consistency of the orientation is that the camera rotates smoothly along time.

PCA can be used to obtain three principal directions for all color-depth images acquired in the scenario [1]. However, the three principal directions do not necessarily match the ones found for the first image. More in detail, one has to take into account two potential issues: (i) the order of the three principal directions may be permuted, (ii) the principal directions may be characterized by symmetric vectors.

[1] Some RGB-D images may have lesser than three principal directions. This special case is studied later.

Let the 3-tuple of indexes (i, j, l) denote permutations, i.e. (i,j,l) $\in$ { (1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,1,2), (3,1,2), (3,2,1) }. The 3-tuple defines a permutation matrix $\Pi_{i,j,l} = [e_i \ e_j \ e_l]^T$, where $e_1 = [1 \ 0 \ 0]^T$, $e_2 = [0 \ 1 \ 0]^T$ and $e_3 = [0 \ 0 \ 1]^T$. Let ${}^w\tilde{R}_{k+1}$ denote a PCA based estimation of the camera orientation at time $k+1$ combined with a permutation of the PCA vectors $\Pi_{i,j,l}$

$$ {}^w\tilde{R}_{k+1} = \text{PCA}_{1:3}(V_{k+1}) \ \Pi_{i,j,l}. \tag{26} $$

Using the smooth rotation assumption, the right permutation can be found by comparing distances of vectors

$$ (i, j, l)^* = \arg_{i,j,l} \min \ d\left({}^wR_k, \ {}^w\tilde{R}_{k+1}\right) \tag{27} $$

where $d(A, B) = tr(I_3 - \text{abs}(A^T B))$ is a distance function comparing two matrices (two collections of three vectors) based on the matrix trace and absolute value of the entries. After finding the permutation (i, j, l) that minimizes the distance, one has an estimation of the rotation matrix , which may still contain sign ambiguity in its column vectors. In order to remove the sign ambiguity in the columns of ${}^w\tilde{R}_{k+1}$, we compute

$$ {}^wR_{k+1} = {}^w \ \tilde{R}_{k+1} \ s({}^w\tilde{R}^T_{k+1} \ {}^wR_k) \tag{28} $$

where $s(X)$ denotes a signs correction diagonal matrix, more precisely $s(X) = \Lambda(sgn(\Lambda(X)))$, $sgn(.)$ returns the signal of each element of a vector and $\Lambda(.)$ denotes an operation converting a set of values to a diagonal matrix or, vice-versa, converts a diagonal matrix into a set of diagonal values. Algorithm 2 summarizes the procedure to compute the camera orientation along time given the previous orientation and the actual principal component analysis.

Algorithm 2 Compute the camera orientation ${}^wR_{k+1}$

Require: ${}^wR_k$ of image k
Require: PCA_{k+1} of image $k+1$
1: **for** $i = 1$ to 6 **do**
2: Permute columns of PCA_{k+1}
3: Compute the orientation distance between PCA_{k+1} and ${}^wR_k$
4: **end for**
5: ${}^w\tilde{R}_{k+1} \leftarrow$ Permutation of PCA_{k+1} with the smallest distance to ${}^wR_k$
6: Signs correction in ${}^w\tilde{R}_{k+1}$ (see Eq.28)
Output: ${}^wR_{k+1}$

C. Camera translation

Computing the rotation (${}^wR_c$) helps to compute the translation ${}^wt_c$. In particular, the main directions of normals allow computing 1D histograms of point clouds along the main directions. These histograms describe distances from the center of the camera to each point of the cloud (x-axis) and the number of points at that distance (y-axis). Distance d can be computed by projecting the point onto the respective surface normal $d = p^T n$ (see figure 8). The objective is to minimize the following functional cost

$$ {}^wt_c = \arg_t \ \min\{\sum_{i=1}^{k} \left\| {}^1M_i - ({}^2M_i + t) \right\|^2\} \tag{29} $$

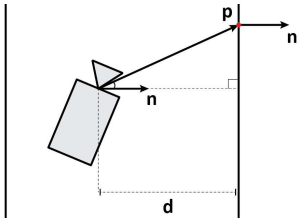


Fig. 8. Distance from the center of the camera to a point, $d = p^T n$.

where ${}^1M$ and ${}^2M \in \mathbb{R}^{k\times 3}$ represent two consecutive point clouds.

The surface normals representing the three main directions of a point cloud, in the coordinate system of the camera, are obtained by transposing the respective rotation matrix ${}^wR_c$. So ${}^wR_c^T = [n_1|n_2|n_3]$ are the surface normals used to compute the histograms.

Figure 9 shows an example of one 1D histogram. Usually there are two distinct peaks per histogram, representing two different sides. If there is just one peak, then there is one single plane observed in the respective principal direction. A typical example is the plane of the front wall observed near the end of a corridor.

Considering the two histograms represented in figure 9, one component of the translation ${}^wt_c$ is computed from the distance between the peaks corresponding to the two images. In other words, each peak represents the points distance (in that specific direction) to the camera, where the difference between that distances is, in fact, the translation between the two point clouds in that direction.

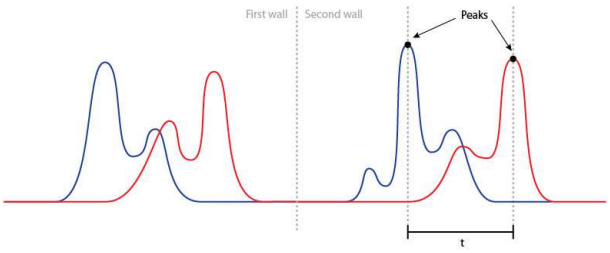


Fig. 9. Histogram representing one direction (each color represents a different image).

Considering a pair of consecutive images (38 and 39), in figure 10 is shown the correspondent RGB-D point clouds with and without the computed translation (${}^wt_c$) applied.

D. Lesser than three principal directions

The three principal planes (directions) are not always observable during the data acquisition along the corridor. Some camera positions do not allow observing, for example, the end of the corridor because it is far away, or does not observe a wall (or floor or ceiling) because it is not in the field of view. We consider that one direction is observed if at least 10% of the points in the point cloud have that direction.

Altogether the *Manhattan* cases can be summarized into four distinct cases, namely (i) all three directions are observed,

(a) (b)

Fig. 10. Applying the translation ${}^{w}t_c$. (a) Textured point clouds before applying ${}^{w}t_c$. (b) Textured point clouds after applying ${}^{w}t_c$. Computed using histogram registration.

(ii) only two main directions are found, (iii) only one direction is found, (iv) there are no directions found. When all the three directions are observed, is still possible to compute the rotation ${}^{w}R_c$ based on all the surface normals. This also allows the computation of the translation ${}^{w}t_c$ using one dimensional histograms, one per direction and then using ICP just to fine tune the computed transformation.

When one observes just two directions it is possible to compute the 3D rotation. Despite missing the third direction, it is implicit. The third direction is the vector normal to both known direction vectors and thus can be computed as the cross product. For example, if one observes the directions X and Y, the missing vector correspondent to the unobserved direction (Z in this case) is computed as

$$n_z = n_x \otimes n_y \tag{30}$$

where $\otimes$ denotes the cross product.

There is only one observed direction when, for example, the camera is near the wall, preparing to begin the rotation to the next corridor. Again, the result of the PCA is incomplete, since it only computes one valid direction. This case can be completed with the information of the previous principal directions and with the signal correction. The translations of the two unknown directions are computed using the ICP algorithm, which in this case receives the rotation and just one translation as initialization parameters.

When there are one or two directions unobserved, even after computing them, it is not possible to compute all the components of the translation (based on 1D histograms). In this case, the unknown components of the translation vector are initialized as zero and are computed later, during the ICP based fine tuning.

The last case, where all three main directions are unobserved does not occur in our indoor scenario. The solution in this case would be running the ICP algorithm without any initial guess of the transformation. In other words, one would return to a simple ICP methodology.

E. Camera motion fine tuning

The transformation ${}^{w}T_c$ turns possible to initialize a fine tuning ICP algorithm with very similar point clouds. In other words, one has a good initial estimative to start the ICP process, which avoids far away local minima.

Before entering a standard ICP algorithm, in many cases one has image (RGB) edge points, which can help registering consecutive point clouds. The edge points pixels obtained in the RGB image can be converted to three dimensional points in the camera coordinate system and then used in a small number of points ICP formulation.

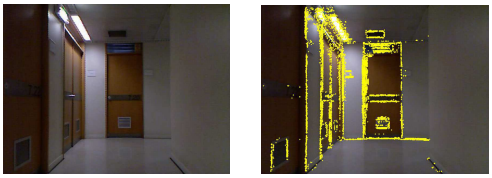


Fig. 11. Original image and the respective edge points.

Matching the 50 nearest neighbors was initially chosen to bootstrap the fine tune ICP. K-dimensional trees are used to find the nearest neighbor in a cloud for each point of the other cloud.

The last step of the fine tuning is to use the ICP algorithm again for the complete point cloud. In particular, we consider just points within 5 meters depth and start the algorithm with the previous estimated transformation.

V. Experiments

This dissertation focus on building three dimensional representations of man-made scenarios. In particular, this section documents the proposed algorithm for fusing point clouds assuming a *Manhattan world*. The algorithm is tested on a real dataset composed by 182 color and depth images acquired using the Microsoft Kinect with small steps, approximately half a meter, made along the floor.

Having acquired an RGB-D image, the surface normal information for each point can be computed. This is essential to characterize the *Manhattan world* with the three principal directions. Figure 7 shows the labeling idea. One labels each 3D point according to the surface normal $n = [n_x \ n_y \ n_z]^T$. Since the 3D points are in the camera coordinate system, it is necessary to apply the rotation ${}^{w}R_c$ to each surface normal, in order to adjust the orientation. The projection of these points into the color image plane (equation 31) results in image pixels (u, v), where the Red, Green and Blue values that define the color of each one are defined by the surface normal values. More specifically $(R, G, B) = (n_x, n_y, n_z)$.

$$\lambda m = \lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = {}^{RGB}K \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}. \tag{31}$$

Figure 12 shows the label color propagation along the time. As expected, the front wall of the first corridor is labeled with blue (in this case, East direction), which is the same direction and thus the same color of the left/right walls of the second and fourth corridors. This corresponds to the labeling objective of the proposed fusion algorithm based in the *Manhattan world* hypothesis. This test in particular shows that the computed rotation matrices, ${}^{w}R_c$, represent effectively the camera orientation.

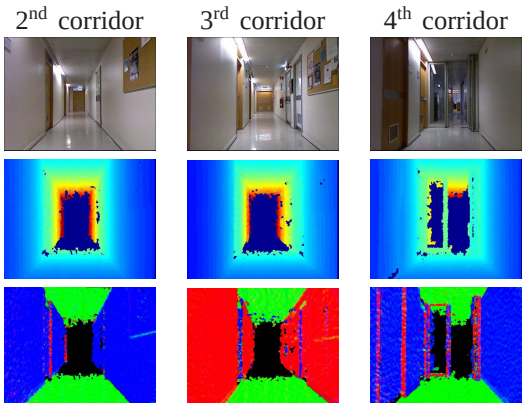


Fig. 12. Original RGB image (first row), original depth image (second row) and normal map of the floor (third row).

In order to assess qualitatively the correction of the computed camera motion, one can superimpose the computed trajectory over the floor plan. Starting at the beginning of the first corridor and applying the successive computed transformations, the obtained result is shown in figure 13, where each red dot represents one taken image. It shows also that the path is within the corridors as supposed to be, however the fourth corridor is not complete, this is because we abruptly rotate the camera almost in 180 degrees to acquire data from the Institute for Systems and Robotics (ISR) reception on the left of the fourth corridor, which is a wide open space that changes drastically the indoor environment, leading this method to stop working properly because of the principal component analysis result in this scenario.

Also, in the end of the first corridor there is a large deviation between two images due an insertion of a subset composed with a few images until the end of this corridor, since the original dataset had some images that were too close to the wall, resulting on a white image (shows only the wall) with no edge points for processing. This subset was taken closest to the left wall and it was inserted with a slight advance comparing to the last used image of the original dataset, which explains the deviation of the path.

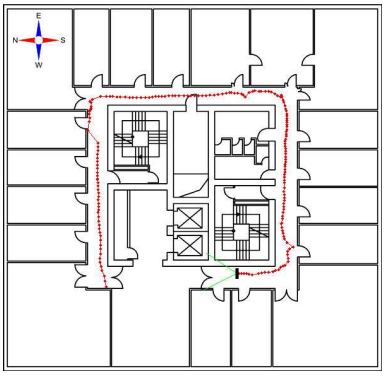


Fig. 13. Simulation over the floor plan.

With the transformations (${}^{w}T_{c}$) computed to all point clouds, the next step is to compute a colored global cloud combining all the images, resulting on a three dimensional reconstruction of the floor. Figure 14 shows the obtained result. It is visible some points that come out of the corridor, these points are some data that the Kinect captured through the glass windows in the top of each door or, sometimes, it is a simple opened door. Also there is a gap in the right wall, at the beginning of the last two corridors, this is due to the fact that when the camera rotates to the right, there is also some translation associated and when the right wall finally appears in the camera sensor range, the camera had already advanced in that corridor.

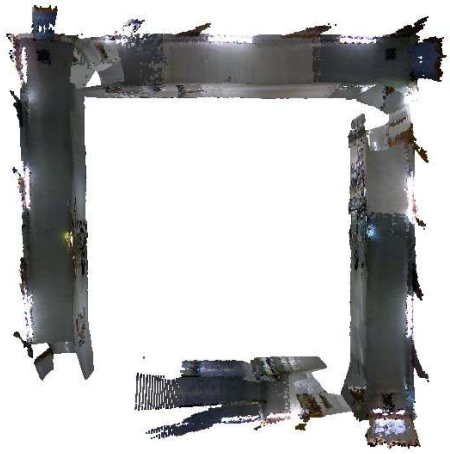


Fig. 14. Top view of global point cloud. Fusion of 182 point clouds. Approximately 47 million points.

Figure 15 shows the comparison between a computed textured point cloud (composed by several images) and one RGB image of the second corridor. The different tones in the textured point cloud are due to the different exposure and color balance of each image.

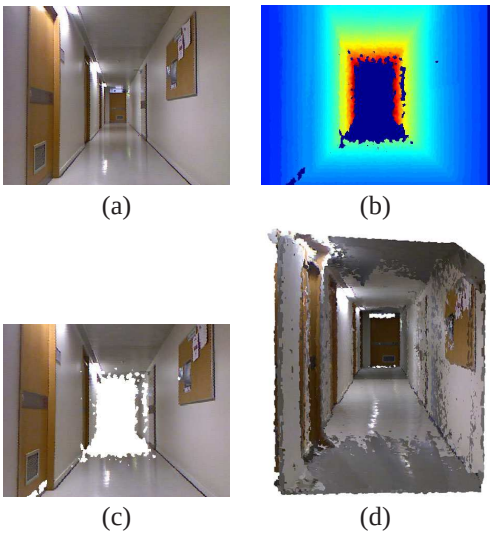


Fig. 15. Second corridor. (a) RGB image. (b) Depth image. (c) Textured depth image. (d) Fused textured point cloud (52 RGB-D images).

VI. Conclusion and Future Work

The work described in this dissertation aims to develop and test a method that can build three dimensional representations

of large scenes using color-depth cameras. In particular we have considered representing a level of a building which has been densely observed by a color-depth camera, more precisely the Microsoft Kinect.

The first step of the process consisted in the calibration of the Kinect camera. We have assumed that both the color and depth cameras forming the Kinect camera are represented by the pin-hole model. Calibration has been computed by matching color and depth features found in a chess texture calibration pattern.

Given the camera calibration already computed, the first approach considered was based in the texture of the RGB images. Texture allows a first rough matching of point clouds which can therefore be fine tuned using the 3D points of the cloud with an ICP like algorithm. This made possible to compute a global point cloud of each corridor separately. However, the accumulation of the small errors associated to the registration of consecutive point clouds actually lead to a visible global bending of the complete reconstructed corridor. In addition, near the end of each corridor the scenario has been found to be texture-poor (mostly white walls), making hard to join the corridors.

One interesting aspect explored in this thesis is that lacking scene texture can, to some extent, be mitigated by using depth information. For example, in a case of white walls forming a corridor corner, one has 3D features, namely orthogonal planes, which form a very precise registration information. Hence, we considered a novel approach, based on the assumption of a *Manhattan world* scenario, i.e. a scenario where most of the surfaces are aligned by three main directions. In this approach one obtains information about the orientation and translation by registering principal planes along consecutive color-depth images. In addition, the extracted edge points of the RGB image help to improve the computed transformation (translation), resulting in a more reliable one. The assumed *Manhattan world* hypothesis turned out to be a good alternative in texture-poor regions of the environment. Considering the fact that in this dissertation there was not acquired odometry information associated to each image, the *Manhattan world* hypothesis actually provided and effective alternative.

Comparing to RGB-D SLAM methods, the *Manhattan world* hypothesis based reconstruction approach allows considering larger steps between point clouds. For instance, in a SLAM method working at 30 frames per second, using a color-depth camera mounted in a robot that is moving at a speed of 10 centimeters per second, each step is about 0.33 centimeters. These are steps much finer than the ones used in this dissertation, which range between 0.3 and 1 meters.

Considering future work, one of the aspects to improve is the computational time of the current implementation. In particular the estimation of surface normals, currently based in finding 3D nearest neighbors within a point cloud, may be replaced by computing derivatives of the depth images. Also for future work, one of the main goals is to dynamically detect the surrounding environment (assessing on line the validity of the *Manhattan world* hypothesis) and combine/extend the proposed approaches to operate outdoors.

REFERENCES

[1] M. Silva, R. Ferreira, and J. Gaspar, "Camera calibration using a color-depth camera: Points and lines based dlt including radial distortion," *Workshop Color-Depth Camera Fusion in Robotics, held with IROS*, 2012.

[2] S. Shen, N. Michael, and V. Kumar, "3d indoor exploration with a computationally constrained mav," 2011.

[3] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual odometry and mapping for autonomous flight using an rgb-d camera," 2011.

[4] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems," 2010.

[5] P. Besl and N. McKay, "A method of registration of 3-d shapes," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 12, no. 2, pp. 239–256, 1992.

[6] L.Armesto, J. Minguez, and L. Montesano, "A generalization of the metric-based iterative closest pointtechnique for 3d scan matching," *IEEE International Conference on Robotics and Automation*, 2010.

[7] A. Segal, D. Haehnel, and S. Thrun, "Generalized icp," *In Proc. of Robotics: Science and Systems (RSS)*, 2009.

[8] E. Berger, K. Conley, J. Faust, T. Foote, B. Gerkey, J. Leibs, M. Quigley, and R. Wheeler, "Ros," http://www.ros.org/.

[9] R. B. Rusu, T. Foote, P. Mihelich, and M. Wise, "Ros kinect," http://www.ros.org/wiki/kinect.

[10] J. Marques, "Image processing and vision, course notes," p. 137148, 2007.

[11] B. Zhang and Y. F. Li, *Automatic Calibration and Reconstruction for Active Vision Systems*, 2012, vol. 57.

[12] J.-Y. Bouguet, "Camera calibration toolbox for matlab," http://www.vision.caltech.edu/bouguetj.

[13] Z.Zhang, "Flexible camera calibration by viewing a plane from unknown orientations," 1999.

[14] W. Gander, "Algorithms for the qr-decomposition," 1980.

[15] Y. . Chen and G. Medioni, "Object modeling by registration of multiple range images," *Proc. of the 1991 IEEE International Conference on Robotics and Automation*, 1991.

[16] H. Kjer and J. Wilm, "Evaluation of surface registration algorithms for pet motion correction," 2010.

[17] G. Golub and C. V. Loan, *Matrix Computations*, 1996, vol. 3.

[18] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," 2004.

[19] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with apphcatlons to image analysis and automated cartography," vol. 24, 1981.

[20] J. M. Coughlan and A. L. Yuille, "Manhattan world: Compass direction from a single image by bayesian inference," *International Conference on Computer Vision ICCV*, 1999.

[21] A. Martins, P. Aguiar, and M. Figueiredo, "Orientation in manhattan: Equiprojective classes and sequential estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 5, 2005.

[22] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," *In Proceedings of ACM SIGGRAPH*, pp. 71–78, 1992.

[23] R. E. Madsen, L. K. Hansen, and O. Winther, "Singular value decomposition and principal component analysis," 2004.

[24] H. Murase and S. K. Nayar, "Visual learning and recognition of 3-d objects from appearance," 1994.

[25] N. Winters, J. Gaspar, G. Lacey, and J. Santos-Victor, "Omni-directional vision for robot navigation," *IEEE WS on Omnidirectional Vision*, 2000.