3D Graph-based Vision-SLAM Registration and Optimization

Doaa M. A.-Latif, Mohammed A.-Megeed Salem, H. Ramadan and Mohamed I. Roushdy

Abstract—Mapping and localization of robots in an unknown environment is a complicated but essential task for navigation and further operations. This has made researchers eager to solve this problem and accordingly many techniques have been investigated using different types of sensors. In this paper we address the Simultaneous Localization and Mapping (SLAM) problem using colored and depth images. We present an overview of the most known techniques with focus on the graph-based mapping, along with a comparison of different algorithms used in registration and optimization. The system is tested on a standard 3D datasets of indoor environment.

Keywords—Graph SLAM, Optimization, Registration, RGB-D sensor, Kinect.

I. INTRODUCTION

Robots have a great impact on our life, they have many forms and are used everywhere, they can be found in automotive, medical, manufacturing and space industries. The nature of the robot operation scenario requires that it builds a map of the environment and at the same time localize itself in this map. The problem becomes hard if the operating robot is mobile autonomous in an unknown environment. However, localization is needed for environment mapping, and available maps are necessary for self localization. This problem is known as Simultaneous Localization and Mapping (SLAM). In the localization task the robot assumes it has a map of the environment and that it only has to answer the question, where am I? In the mapping task it assumes that its exact location is known and that it only has to answer the question; what does the world look like?

The SLAM problem can be adapted and combined with other techniques to solve many problems in different domains including the medical one. It can have a great impact on the visually impaired if used to provide a description of the environment and an independency in navigation especially if used in indoor environments. The mapping task is very challenging because usually the data provided by the sensors are noisy and sometimes they interfere with the map updating process resulting in false matches between the previously known features and the newly measured ones. The environment representation is another important challenge. The robot has to represent its surrounding environment in a way that is easily processed and updated and in the same time it should be informative and capture as much information as possible of the surrounding environment.

This paper presents an extended survey of current state-of-the-art of techniques used for vision-based SLAM for indoor environment. We present focus on the graph-based map registration and optimization [34]. The rest of the paper is organized as follows. In the following section II we discuss the different types of sensors used for SLAM and we justify the advantage of optical sensors over the other types. Section III provides an overview of the different SLAM processing techniques along with a survey of the related work. The tested methods are presented in section IV. Results are detailed and discussed in section V, followed by conclusion in section VI.

II. SENSING

Various type of sensors has been used to generate 3D Maps. Classical examples include, laser [1][2], stereo cameras [3][4][31][32] monocular cameras [5][6][7][30], and recently RGB-D sensors[9][10][8]. The type of sensor used affects how the data is processed and the information in the generated map greatly.

A robot is usually equipped with internal sensors that are used to perceive its position, pose and motion parameters. The main internal sensors are odometry and compasses. Compasses are often used to detect headings of mobile robots. Odometry estimates the robot's trajectory from the summation of wheel speeds; it is widely used for localization. Internal sensors can provide inaccurate data so they are always used with some other type of sensor to correct the drifts, there are even some approaches that don't use the internal sensors at all.

Laser range finder is traditional sensor which is widely used in robotics. It uses a laser beam to measure the distance to an object. They produce 3D point clouds that are suitable for frame-to-frame alignment and for dense 3D reconstruction but the produced point clouds are not as rich in visual information as the ones produced by color cameras moreover they are bulky and expensive compared to cameras.

Cameras can be used to reconstruct a depth map from at least two images showing a 3D scene from different
observation points. They have become more and more important in the robotics field, due to its remarkable characteristics such as low cost, small size, low weight and low energy consumption. Unfortunately when it comes to 3D mapping it requires extensive processing to extract dense depth information from color cameras data alone, especially in indoor environments with very dark or sparsely textured areas.

RGB-D cameras are sensing systems that capture RGB images along with per-pixel depth information. They rely on either active stereo or time-of-flight sensing to generate depth estimates at a large number of pixels. The most well-known RGB-D camera is the Microsoft Kinect [11]. They provide an alternative to the expensive and bulky Laser range finders. It provides 3D point clouds with depth information and at the same time captures rich visual information of the environment. It also gives a means to create maps of an environment with known scale (most camera-based visual mapping algorithms can only create maps with unknown scale). The Microsoft Kinect is able to grab RGB images of 640x480 pixels. It has a frame rate of 30Hz and an angular field of view of 57 degrees horizontally and 43 degrees in the vertical axis and the depth sensor range is between 1.2m and 3.5m. RGB-D sensors provide a trade of between accuracy and complexity. They provide rich visual scenes when compared to laser sensors and at the same time simplify the calibration and rectification processes when compared to monocular or stereo cameras.

III. SLAM PROCESSING PARADIGMS

There have been many approaches to solve the SLAM problem, most of which can be categorized into two main paradigms: filtering and optimization based approaches [12].

A. Filtering approaches

The most popular filtering approaches are the extended Kalman filter (EKF) and particle filters. EKF were used in [13][14][15][33], it stores the robot pose and the environment feature positions in one state vector and uses an error covariance matrix to store the uncertainties of these state estimates along with cross correlation terms between features and poses. Particle filters were used in [16][17][18]. It maintains multiple map hypotheses, each conditioned on a stochastically sampled trajectory through the environment.

Filtering approaches were used widely over the past years due to the fact that the data provided by the robot sensors suffer from noise and inconsistency and these approaches can model different sources of noise and their effects on the measurements.

B. Optimization-based approaches

Optimization (Graph)-based approach usually uses an underlying graph structure to represent the robot measurements. The graph nodes represent the robot poses and the measurement acquired at this position and the edges represent a spatial constraint relating two robot poses. A constraint usually consists of the relative transformations between the two poses. These transformations are either odometry measurements between sequential robot positions or are determined by aligning the observations acquired at the two robot locations. Once the graph is constructed the optimization process starts to find the configuration of the robot poses that best satisfies the constraints. Graph-based SLAM contains two chore tasks:

1. Graph construction concerned with constructing the graph from the raw sensor measurements.
2. Graph optimization concerned with determining the most likely configuration of the poses given the edges of the graph.

The process is summarized graphically in Fig. 1. The graph construction is usually called front-end and it is heavily sensor dependent, while the second part is called back-end usually it relies on an abstract representation of the data.

Fig. 1 The front-end and back-end of the SLAM process.
SURFELS reduce the generated map size by a factor of 32 and present a better map quality for viewing, but unfortunately they affect the computation time preventing the whole system from operating in real time. In [8] a graph-based SLAM system is built using the oriented FAST and rotated BRIEF (ORB) as a feature detector and descriptor. The poses were calculated using RANSAC and further refined using the Generalized Iterative Closest point (GICP). Finally global optimization was achieved using the General (Hyper) Graph Optimization g'o.

IV. GRAPH-BASED SLAM

The approach used to complete this study follows the related work presented above. Fig. 2 provides an overview of the system. In the front-end the graph is constructed as the camera moves, new areas are discovered and new poses are added to the graph. When adding a new pose registration is required to align the data together. After a while small errors in registration accumulate resulting in inconsistency in the generated map. This is obvious if the robot visited a previously mapped place, the error will result in presenting the same place twice in the map. Here comes the need for the back-end to adjust the accumulative error and align the complete data sequence.

A. SLAM Front-end

The SLAM front-end consists of two parts the Registration and the Loop closure.

1) Registration

The registration step to align consecutive data frames. The alignment is usually done by estimating an approximate transformation between the consecutive frames and then refining this initial estimate. The approach used in this study is similar to the one presented in [4]. It can be summarized into 3 main steps:

1. Computing the correspondence between successive frames:
   a) Find 2D feature correspondence between RGB Images.
   b) Reject bad correspondence.
   c) Transform the 2D features to their equivalent 3D features.

2. Estimate the initial alignment of the frames.

3. Refine the alignment.

First step we applied four combinations of feature detectors and descriptors:

1. Speeded Up Robust Features (SURF) [35].
2. Features from Accelerated Segment Test (FAST) [36] & Binary Robust Independent Elementary Features (BRIEF)[37].
3. Oriented FAST and Rotated BRIEF (ORB)[38].
4. Binary Robust Invariable Scalable Keypoint (BRISK)[39].

These were used to detect and describe the features in the RGB images obtained from the Kinect device. Then after matching the corresponding features we rejected the bad correspondences using the symmetry test and the fundamental matrix.

The symmetry test is so simple features are matched from the first image to the second image i.e for every detected feature in the first image find its possible match in the second image. Then the features are matched from the second image to the first image i.e for every detected feature in the second image find its possible match in the first image, and finally the two matching lists are compared and the non-symmetric items are rejected.
The fundamental matrix is a 3 × 3 matrix. It is the algebraic representation of epipolar geometry. It relates the corresponding points between two stereo images \( x_i \leftrightarrow x_i' \). If a point in space is viewed in one image as \( x_i \) and in the second as \( x_i' \), the two points will be related together with the fundamental matrix \( F \) transforms the point \( x_i \) in an image into an epipolar line \( l' = Fx_i \) in the second image. For each point \( x_i \) in one image, there exists a corresponding epipolar line \( l' \) in the other image. Any point \( x_i' \) in the second image matching the point \( x_i \) must lie on the epipolar line \( l' \).

Second we estimated an initial rigid transformation (rotation and translation) that will align the points in the first point cloud to the second point cloud using Singular Value Decomposition which can be summarized in 3 steps:

1. Find the centroids of both point clouds \( A,B \).
2. Translate both point clouds to the origin then find the optimal rotation (\( R \)) using (SVD).
3. Combine \( R \) and the centroids into a single 4x4 transformation matrix.

The centroids are just the average point and can be calculated as follows:

\[
P = \begin{bmatrix} x \\ y \\ z \end{bmatrix}
\]  

(1)

\[
\text{centroid}_A = \frac{1}{N} \sum_{i=1}^{N} P_A^i
\]  

(2)

To find the optimal rotation we first re-center both point clouds so that both centroids are at the origin. This removes the translation component, leaving only the rotation to deal with, after that we calculate \( H \), and using SVD to find the rotation

\[
H = \sum_{i=1}^{N}(P_A^i - \text{centroid}_A)(P_B^i - \text{centroid}_B)^T
\]  

(3)

\[
[U, S, V] = \text{SVD}(H)
\]  

(4)

\[
R = VU^T
\]  

(5)

Then combine the rotation and centroids in one matrix

\[
C_A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} - \text{centroid}_A
\]  

(6)

And the rigid transformation matrix \( T \) is calculated as \( T=C_BR_{new}C_A \).

Third we refine the alignment between registered point clouds. Iterative Closest Point (ICP) was used to minimize the alignment difference. Since its introduction in [19] there has been many variants introduced affecting each step in the algorithm. ICP starts with two point clouds and an initial guess for their relative transformation, and iteratively refines the transformation by repeatedly generating pairs of corresponding points and minimizing an error metric. It terminate when the change in the mean square error falls below a pre-set threshold. The Generalized iterative closest point (GICP) is one of many variants of the ICP algorithm [20]. GICP introduces a change in the last step of standard ICP, which is the error minimization function. It attaches a probabilistic model to it and keeps the rest of the algorithm unchanged so as to reduce complexity and maintain speed. Correspondences are computed using Euclidean distance and \( kd \)-trees are used in the look up of closest points.

2) Loop closure

Registration between successive clouds produce satisfactory results but for only small or moderate distances, as noise and errors in depth values and in pair-wise cloud alignment cause the estimated pose to drift over time resulting in an erroneous map. This is obvious when a long path is mapped, eventually returning to a previously visited location. The cumulative error in cloud alignment results in a map that has two representations of the same region in different positions. This is known as the loop closure problem. Loops are represented as edges between nodes that are not temporally adjacent. Once a loop is detected the new correspondence between nodes can be used as an additional constraint in the graph. Fig. 3 shows an example. The question now is how to detect the loop in an efficient way. There has been many loop closing techniques presented in the literature and according to [22] they can be classified into 3 main categories.

1. **Map to map** as the name indicates this is used in approaches that depends on building small sub-maps of the environment, the correspondence between sub-maps is investigated taking into account the visual appearance and the relative position between the sub-maps.

2. **Image to map** the most recent image captured is compared with the built map features looking for correspondence. The pose of the camera is determined relative to a map of point features by finding correspondences between the image and the features in the map.
3. **Image to Image** the correspondence is investigated between images of the world being mapped; the most recent image is compared with previously captured images looking for matches.

In our approach a simple image to image loop detection approach is used. A set of key nodes is stored with the features detected and described in the registration step are kept for each key node. When a new observation is processed its location is examined if it is close to any of the key nodes it is further checked Fig. 4 shows an example if the number of matches passes a certain threshold in our case 40 features then this node is considered as a loop closing node and a constraint represented as an edge is added to the graph connecting the two matched nodes.

**B. SLAM back-end**

The SLAM back-end role is to optimize the map reducing the error by optimizing the underlying graph structure provided by the front end, this graph is composed of n vertices storing the observation at certain poses, and edges representing the neighbour relations between these poses. Global optimization techniques try to estimate optimally all poses to build a consistent map of the environment. It can be considered as choosing the best solution that minimizes an error function from all the feasible solutions.

If we assume that \( X = (x_1, \ldots, x_n)^T \) is a vector describing the robot poses where \( x \) describe the pose of node \( i \) and \( Z_i \) is the mean and \( \Omega_{ij} \) is the information matrix of the transformation matrix that aligns the observation at node \( i \) and node \( j \) together. Let \( \hat{Z}(x, x) \) be the estimated measurement at nodes \( x \) and \( x \). The log likelihood \( l \) of \( z_i \) can be calculatedas \( l_i(x) = [z_i - \hat{Z}(x, x)]^T \Omega_{ij} [z_i - \hat{Z}(x, x)] \). If we consider \( e(x, x) \) as an error function that computes the difference between the estimated measurement \( \hat{Z} \) and the real measurement \( Z \) such that \( e_i(x, x) = z_i - \hat{Z}(x, x) \).

The function value of \( F(x) \) at the minimum is not important, what matters is the value of the variable \( x^* \) where that minimum occurs, further details about graph optimization in SLAM can be found in [12][23].

In our study we have compared the performance of 3 different global optimizers:

1. **Georgia Tech Smoothing and Mapping**: Known also as GTSAM [26]. It is a C++ library based on factor graphs. A factor graph consists of factors connected to variables. The factors represent probabilistic information on the unknown random variables in the estimation problem.

2. **Hierarchical Optimization on Manifolds**: Known also as HOG-Man [24] it applies Gauss-Newton with sparse Cholesky factorization that considers a manifold representation of the state space to better deal with the camera rotations.

3. **General (Hyper) Graph Optimization**: Known also as g2o [25]. It is a C++ framework for performing the optimization of nonlinear least squares problems that can be embedded as a graph or in an hyper-graph.
The following open source projects were used to implement different techniques subject to our comparison: The point cloud library (PCL) [27] was used in the transformation estimation and refinement. It is a large scale, open source project for 2D/3D image and point cloud processing, the Open source computer vision library (OpenCV) [28] which has been used in feature detection, description and matching. The different algorithms used in this comparison were tested on the Computer vision and pattern recognition group (CVPR) [29] datasets. These datasets contain the color and depth images of a Microsoft Kinect sensor along with the ground-truth trajectory. Fig. 5 provides an example of the data provided by the Kinect in the Freiburg1_room dataset.

![Fig. 5 The color and depth images of the Freiburg1_room dataset image courtesy to the CVPR group [29].](image)

V. RESULTS

All the tests were performed using an Intel Core i7-3610QM CPU @ 2.30GHz _ 8 running a 32 bit Linux 3.2.0. We used four of the CVPR group [29] datasets in our tests: Freiburg2_xyz which contains data for debugging translations where the Kinect was moved along the principal axes in all directions, Freiburg2_desk which captures the details of a typical office scene with two desks, a computer monitor, keyboard, phone, chairs, etc. with the Kinect moving around two tables so that the loop is closed, Freiburg1_room which has been recorded through a whole office environment and is well suited for evaluating how well a SLAM system can cope with loop-closures, and Freiburg2_pioneer slam which was recorded from a Kinect mounted on top of a Pioneer robot which was joysticked through a maze of tables, containers and other walls, so that several loops have been closed for map building.

In the registration module we have tested the feature correspondence step by combining different detectors and descriptors. We calculated the percentage of good matches by counting the number of matches after rejecting bad correspondence with the fundamental matrix. Fig. 6 shows the result. All the tested algorithms produce close results between 40%-60% with BRISK getting the most scores on all the datasets tested. Details of the tests results were presented in our earlier work [21].

![Fig. 6 Average percentage of good matches on the 4 tested datasets.](image)

We have used the CVPR group evaluation tools to compare the global optimization algorithms. Two error metrics have been used: the Absolute Trajectory Error (ATE), and the Relative Pose Error (RPE). The ATE is useful for measuring the performance of visual SLAM systems. It measures the absolute trajectory error by comparing the difference between
the estimated and the ground-truth path after associating them using the timestamps. It also computes the mean, median and the standard deviation of these differences. The RPE is useful for measuring the drift of visual odometry systems. It computes the error in the relative motion between pairs of timestamps. More details about the evaluation metrics are in [29].

Fig. 9 The result trajectory compared to the groundtruth of the Freiburg1_room dataset using g'.

Table 1 The Absolute Trajectory Error on the Freiburg1_room dataset in meters.

<table>
<thead>
<tr>
<th>ATE</th>
<th>GTSAM</th>
<th>HO-Man</th>
<th>g'0</th>
</tr>
</thead>
<tbody>
<tr>
<td>rmse</td>
<td>0.083895</td>
<td>0.079756</td>
<td>0.079022</td>
</tr>
<tr>
<td>mean</td>
<td>0.074727</td>
<td>0.070692</td>
<td>0.068790</td>
</tr>
<tr>
<td>median</td>
<td>0.069797</td>
<td>0.065114</td>
<td>0.064971</td>
</tr>
<tr>
<td>std</td>
<td>0.038135</td>
<td>0.036927</td>
<td>0.038891</td>
</tr>
<tr>
<td>min</td>
<td>0.015323</td>
<td>0.010718</td>
<td>0.009273</td>
</tr>
<tr>
<td>max</td>
<td>0.156559</td>
<td>0.163685</td>
<td>0.154001</td>
</tr>
</tbody>
</table>

Table 2 The Relative Pose Error on the Freiburg1_room dataset.

<table>
<thead>
<tr>
<th>RPE</th>
<th>GTSAM</th>
<th>HO-Man</th>
<th>g'0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relative Translation Error</td>
<td>0.089116</td>
<td>0.077349</td>
<td>0.078440</td>
</tr>
<tr>
<td>Relative Rotation Error</td>
<td>4.346548</td>
<td>3.628420</td>
<td>3.773067</td>
</tr>
</tbody>
</table>

Table 3 The Average Absolute Trajectory Error and Relative Pose Error on the tested four datasets.

<table>
<thead>
<tr>
<th>Average rmse</th>
<th>GTSAM</th>
<th>HO-Man</th>
<th>g'0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Absolute trajectory error</td>
<td>0.158508 m</td>
<td>0.160902 m</td>
<td>0.157364 m</td>
</tr>
<tr>
<td>Relative Translation error</td>
<td>0.109753 m</td>
<td>0.085200 m</td>
<td>0.081775 m</td>
</tr>
<tr>
<td>Relative Rotation error</td>
<td>2.436107 deg</td>
<td>5.969135 deg</td>
<td>4.279843 deg</td>
</tr>
</tbody>
</table>

Table 3 The Average Absolute Trajectory Error and Relative Pose Error on the tested four datasets.

The difference between the estimated trajectory and the ground-truth using g'0 as an optimizer can be viewed clearly in Fig.9. The average error on the tested four datasets is described in Table 3 in which all three produce similar scores but g'0 performs slightly better.

VI. CONCLUSION

In this paper a description of the graph-based approach using a RGB-D camera as its only sensor was presented. Our system factored the SLAM problem into front-end and the back-end providing a comparison of different registration and global optimization techniques using a standard dataset. The SLAM problem has a long history with many approaches but yet there are still some open problems including solving SLAM for extended periods or life-long SLAM. We have also started investigation in Multi-robot cooperation in map building using multiple sensors each exploring a different part of the environment [40].

REFERENCES


