Graph Optimization and Probabilistic SLAM of Mobile Robots using an RGB-D Sensor

João Carlos Virgolino Soares[1] and Marco Antonio Meggiolaro[2]

***Abstract*— Simultaneous Localization and Mapping (SLAM) is one of the most important problems in mobile robotics, as it is a crucial step to achieve autonomy. It consists in creating a map of the environment using only sensor information, and simultaneously using this information to estimate the pose of the robot. This problem needs a probabilistic formulation to handle the uncertainty present in sensor measurements and robot motion. The Graph-SLAM is a popular probabilistic approach for SLAM based on maximum likelihood estimation and non-linear least squares optimization. This work is subdivided into two main parts. First, this paper presents the development of a pose-graph optimization tool for MATLAB that works for both 2D and 3D problems. Second, it presents the development of a full RGB-D SLAM system for indoor environments, implemented as a Robot Operating System (ROS) package. The proposed methodology is evaluated using benchmark datasets available in the literature and compared with state-of-the-art methods.**

Student level: MSc
Defended in March 26th, 2018
Submission to CTDR

I. INTRODUCTION

Autonomous mobile robots have a wide range of applications, including autonomous vehicles, industrial robots and unmanned aerial vehicles. Autonomous mobile navigation is a challenging subject due to the high uncertainty and nonlinearity inherent to unstructured environments, robot motion and sensor measurements. In several situations the robot needs a map of the environment to perform autonomous navigation, such as indoor scenarios where there is no GPS information available. The robot also needs to estimate its own pose with respect to the global coordinate system. However, usually there is no prior information about the environment and the robot needs to create the map, using only sensor measurements, and simultaneously estimate its pose. This problem is known as Simultaneous Localization and Mapping (SLAM), and is one of the most important subjects in mobile robotics.

For many years the laser range-finders were the most popular sensor for SLAM, due to its high range and precision. However, RGB-D sensors are an interesting alternative, due to a lower price and for the availability of color and depth information [19]. In 2011, Huang et al. [1] developed FOVIS, a visual odometry system for RGB-D sensors, tested in a micro air vehicle. However, the first published SLAM system for RGB-D sensors was proposed by Henry et al. [26] in 2012.

Due to the high urcertainty inherent to sensor measurements, a probabilistic formulation of the SLAM problem is necessary. Probabilistic algorithms deal with this uncertainty explicitly representing it using probability theory.

There are three main probabilistic approaches for SLAM: kalman filters, particle filters and graph-based. The Graph-SLAM method consists in representing the states of the robot with a graph, that is optimized using a non-linear least squares formulation to generate a maximum likelihood solution for the trajectory of the robot, which minimizes the errors caused by motion estimation. It has several advantages in comparison with kalman filters and particle filters, including performance and accuracy [12] [32].

The Graph-based approach for SLAM was proposed in 1997 by Lu and Milios [10]. However, their approach was infeasible to perform in real-time [29] [12], and it was heavily dependent on the initial estimate [16]. Several real-time implementations were developed afterwards, such as TORO [11] and g^2o [29], implemented in C++, that have both computational speed and precision. Wagner et al. [31] implemented in 2011 a MATLAB framework to make the technique accessible to non C++ developers, the Manifold ToolKit. Their framework is extended to multi-sensor calibration problems, with a SO(3) exponential map for 3D optimization.

One of the most important steps in a SLAM system is the loop closure problem, which consists in detecting if the robot is in a previously visited region [26], enabling the graph optimization and allowing the robot to understand the real topology of the environment [4]. Henry et al. [26] introduced the concept of keyframes for loop closure detection.

This work has two main contributions. First, the development of a standalone pose-graph optimization tool for MATLAB that can work for both 2D and 3D SLAM problems, specifically designed to pose-graph optimization with a pose-quaternion representation, using advanced algebraic concepts to assure efficiency and convergence. Second, the development of a full RGB-D SLAM system for static indoor environments, combining the robust visual odometry of FOVIS with a efficient keyframe-based loop closure detector, implemented as a Robot Operating System (ROS) [25] package. The proposed methodology is evaluated using benchmark datasets available in the literature and compared with state-of-the-art methods.

This work was supported by CAPES

The authors are with Dept. of Mechanical Engineering, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, Brazil.

[1]E-mail: virgolinosoares@gmail.com

[2]E-mail: meggi@puc-rio.br

The paper is organized as follows. The section 2 presents the formal definition of the SLAM problem. In Section 3 is presented the development of the pose-graph optimization tool with the respective numerical evaluation. The Section 4 details the proposed RGB-D SLAM system. Section 5 details the experiments and results. In Section 6 is shown the results of the numerical evaluation using real world datasets and a comparison between the proposed methodology and state-of-the-art methods. Finally, in section 7 are shown the final remarks.

II. Problem Definition

The SLAM problem is divided into two main steps: front-end and back-end [4]. The front-end processes sensor information and creates a graph using a motion estimation method, such as visual odometry. The back-end uses probability theory to optimize the graph given the measurement errors. In Fig. 1 is shown the diagram of a Graph-SLAM system.

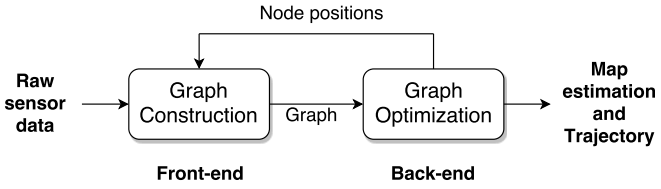

Fig. 1. Graph-SLAM system

In the Graph-SLAM formulation, the nodes of the graph are the state variables $x = [x_1, \ldots, x_T]$ that represent the poses of the robot in a plane or in space, depending if the problem is two-dimensional or three-dimensional. The edges are measurement relations locally affected by Gaussian noise. It is assumed that the robot has odometry information and range measurements. The relative transformation between two poses x_i and x_j is called the predicted measurement $\hat{z}_{ij}$. The real observations are represented by z_{ij}. The error function is computed by the difference between the measurement prediction and the real measurement [12].

When the robot performs a movement, going from position i to position j, a node x_j is created and also an edge e_{ij} between x_i and x_j. An edge is also created if the robot detects a loop, in other words, revisits a previous known location. The front-end is heavily sensor-dependent. Different techniques are used if the robot has laser scanners or RGB-D sensors, for example.

Once the graph is constructed, the objective of the back-end is to find the trajectory of the robot (the configuration of the nodes) that best explains these measurement constraints created by visual odometry and loop closures. Therefore, the objective is to find maximum belief of the state x, given the measurements z [4], as stated by Eq. (1).

$$\hat{x} = \underset{x}{\operatorname{argmax}}\, p(x|z) \tag{1}$$

Using Bayes' theorem, it becomes the likelihood of measurements given the state:

$$\hat{x} = \underset{x}{\operatorname{argmax}}\, \frac{p(z|x)p(x)}{p(z)} \tag{2}$$

Assuming that is not prior information and the measurements are independent, the problem factorizes into:

$$\hat{x} \propto \underset{x}{\operatorname{argmax}} \prod_{k=1}^{n} p(z_k|x) \tag{3}$$

Assuming that every measurement is locally Gaussian, the likelihood of the measurements will also be Gaussian:

$$\hat{x} = \underset{x}{\operatorname{argmax}} \prod exp(-e_{ij}^T(x_i, x_j, z_{ij})\Omega_{ij}e_{ij}(x_i, x_j, z_{ij})) \tag{4}$$

where e_{ij} is the error vector that states the difference between the measurement predicted by odometry and the actual measurement, and Ω is the information matrix associated with each measurement. Taking the logarithm to transform the product into a sum:

$$\hat{x} = argmin \sum_{ij} e_{ij}^T(x_i, x_j, z_{ij})\Omega_{ij}e_{ij}(x_i, x_j, z_{ij}) \tag{5}$$

Therefore, the graph-based probabilistic formulation is analogous to a non-linear least squares optimization problem [18]. A first order Taylor expansion around the initial guess $\check{x}$ is applied to approximate the error function, as stated in eq. (6).

$$e_{ij}(\check{x} + \delta x) \approx e_{ij} + J_{ij}\delta_x \tag{6}$$

where J_{ij} is the Jacobian of the error function computed in $\check{x}$. Thus, the cost function F of an observation between nodes i and j can be obtained rewriting a parcel of the sum in Eq. (5) with the local approximation of Eq. (6).

$$F_{ij}(\check{x} + \delta x) \approx (e_{ij} + J_{ij}\delta_x)^T\Omega_{ij}(e_{ij} + J_{ij}\delta_x) \tag{7}$$

The global cost function can be found with the sum of all local approximations:

$$\begin{aligned} F(\check{x} + \delta x) = \sum F_{ij}(\check{x} + \delta x) \approx \\ \sum (e_{ij}^T\Omega_{ij}e_{ij} + 2e_{ij}^T\Omega_{ij}J_{ij}\delta_x + \delta_x^T J_{ij}^T\Omega_{ij}J_{ij}\delta_x) \end{aligned} \tag{8}$$

Eq. (8) can be minimized solving the following linear system:

$$H\delta_x = -b \tag{9}$$

where

$$H = \sum J_{ij}^T\Omega_{ij}J_{ij} = J^T\Omega J \tag{10}$$

$$b = \sum J_{ij}^T\Omega_{ij}e_{ij} = J^T\Omega e \tag{11}$$

The solution for one iteration is then obtained by adding the increments to initial guess, as stated in Eq. (12).

$$x^* = \breve{x} + \delta_x \tag{12}$$

III. A Pose-Graph Optimization tool for MATLAB

This section details the implementation of a graph-based back-end framework developed for MATLAB.

A. 2D Implementation

The poses of the robot are given by a translation vector $t_i = [x_i, y_i]$ and a rotation angle θ_i, which represents the orientation of the robot.

$$p = [t_i, \theta_i] \tag{13}$$

Each measurement between the nodes i and j is given by z_{ij}, stated in Eq. (14):

$$z_{ij} = [t_{ij}, \theta_{ij}] \tag{14}$$

The rotations of the robot are expressed with 2x2 rotation matrices, as shown in Eq. (15).

$$R_i = \begin{pmatrix} cos(\theta_i) & -sin(\theta_i) \\ sin(\theta_i) & cos(\theta_i) \end{pmatrix} \tag{15}$$

The error function is given by Eq. (16).

$$e_{ij} = z_{ij}^{-1} \begin{pmatrix} R_i^T (t_j - t_i) \\ \theta_j - \theta_i \end{pmatrix} \tag{16}$$

The Jacobian matrix, composed by the derivate of the error function in terms of each pose, is naturally sparse, because the error function of each measurement only depends on the values of two nodes, as shown in Eq. (17).

$$J_{ij} = \begin{pmatrix} 0 \dots 0 & \frac{\partial e_{ij}}{\partial x_i} & 0 \dots 0 & \frac{\partial e_{ij}}{\partial x_j} & 0 \dots 0 \end{pmatrix} \tag{17}$$

The formulation is described using pseudocode notation in algorithm 1.

B. 2D Dataset Evaluation

The implementation was evaluated using datasets available in the literature. The Intel Dataset, chosen for the 2D evaluation, is a benchmark dataset with real data acquired at the Intel Research Lab in Seattle, consisting of a graph with 1228 poses and 1505 constraints created from raw measurements of wheel odometry and laser range finder. All nodes are represented by an ID, x, y and θ values, which correspond to the initial odometry poses.

All edge lines have the format: "IDfrom IDto x y θ I11 I12 I22 I33 I13 I23". The first two numbers "IDfrom IDto" correspond respectively to the ID of observing and observed nodes i and j. The x and y values compose the translation vector between nodes, and θ correspond to the rotation angle between nodes. The numbers "I11 I12 I22 I33 I13 I23" are the 6 top triangular elements of the 3x3 information matrix corresponding to each measurement. The symmetric information matrix is stated in Eq. (18).

Algorithm 1 Pose Graph Optimization

```
procedure READ GRAPH
    x ← vertices
    z ← edges
    Ω_ij ← information matrices
endprocedure
while not converged do
    preallocate H and b
    for all measurements do
        compute error function e_ij
        compute Jacobians of the error function with respect to the nodes i and j
        compute the contribution of this measurement to H
        compute b
    endfor
    δ_x ← solve(Hδ_x = −b)
    x += δ_x
endwhile
```

$$\Omega_{ij} = \begin{bmatrix} I11 & I12 & I13 \\ I12 & I22 & I23 \\ I13 & I23 & I33 \end{bmatrix} \tag{18}$$

In Fig. 2 is shown the corrupted initial pose-graph. The poses of the robot are represented by the blue dots, and the red lines are the measurement constraints, derived from loop closures. Fig. 3 shows that the graph converges to the real trajectory of the robot after the optimization. In Fig. 4 is shown a comparative image of the same dataset optimized by a method called MOLE2D, developed by Carlone and Censi [22].

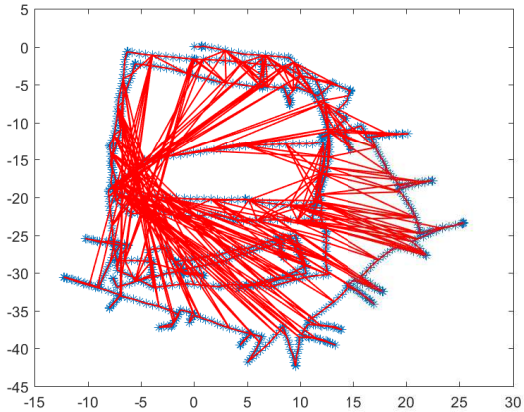

Fig. 2. Intel - Initial corrupted pose-graph

In Fig. 5 is shown the logarithmic global error per iteration. In only four iterations the system was able to optimize the entire graph, which shows the robustness of this implementation.

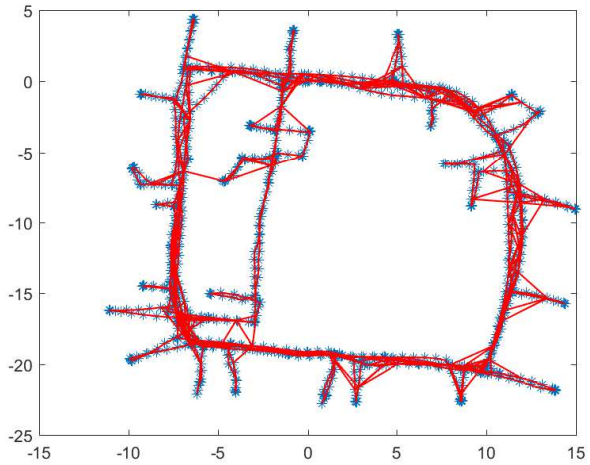

Fig. 3. Intel Optimized pose-graph

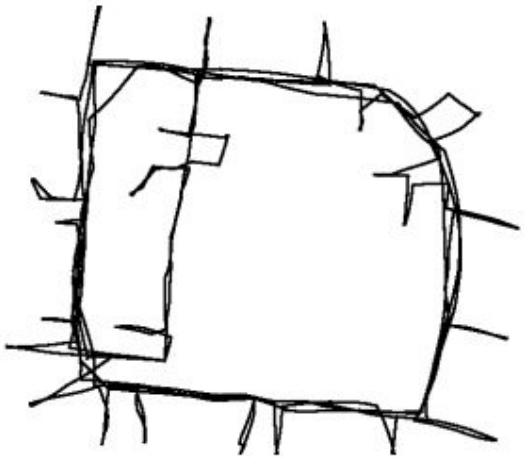

Fig. 4. MOLE 2D Optimization

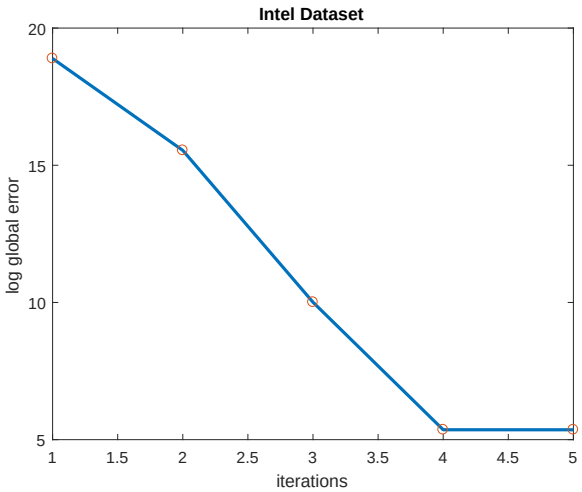

Fig. 5. Intel dataset - Global error per iteration

The objective of the second 2D test is to evaluate if the system is able to optimize a graph with oversized number of constraints. This dataset contains 10000 poses and 64311 constraints. The initial corrupted configuration is shown in Fig. 6. In Figs. 7 and 8 is shown a comparison between the result of the present work and the result obtained with LAGO, an algorithm developed by Carlone et al. [23]. The global error is shown in Fig. 9. Even with a larger number of constraints, the system is able to optimize the graph.

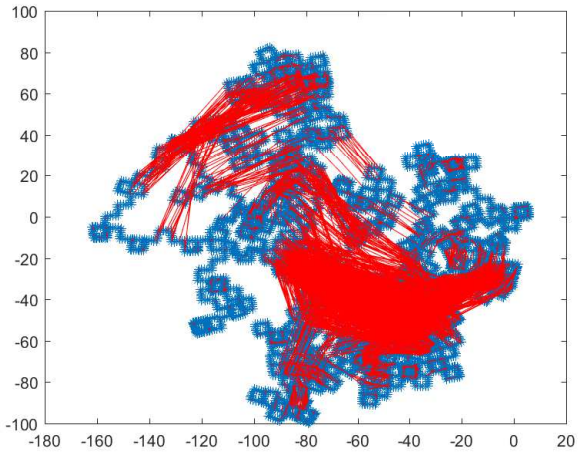

Fig. 6. Initial corrupted pose-graph

C. 3D Implementation

The orientation of the robot is easily represented in the 2D case with a single normalized angle. In other hand, it is more problematic in the three-dimensional case due to the variety of possible parameterizations and their corresponding drawbacks. For instance, orientation can be represented by euler angles, rotation matrices or unit quaternions. Euler angles are subject to singularities. When two of the three rotation axes are aligned, a DOF is lost, which is called the gimbal lock problem. To overcome this problem, a solution would be the use of an over-parametrized representation, such as unit quaternions or rotation matrices. Rotation matrices are problematic for imposing six non-linear constraints in the optimization to ensure orthogonality and unit length of the columns. In other words, to ensure it remains in SO(3) [9].

Quaternions are more suitable for optimization problems than rotation matrices due to the number of constraints that need to be maintained at every iteration. First described by W. R. Hamilton in 1843 [34], quaternions can be seen as a generalization of complex numbers, with a real part and three different imaginary parts [21]. In Eq. (19) is shown a general form of a quaternion.

$$q = q_x i + q_y j + q_z k + q_r \tag{19}$$

where

$$i^2 = j^2 = k^2 = ijk = -1 \tag{20}$$

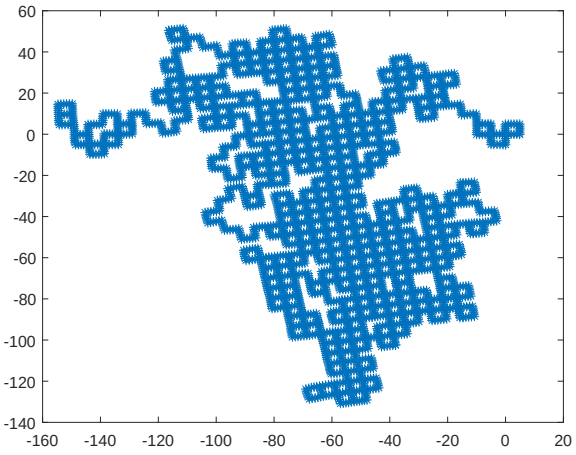

Fig. 7. Optimized pose-graph

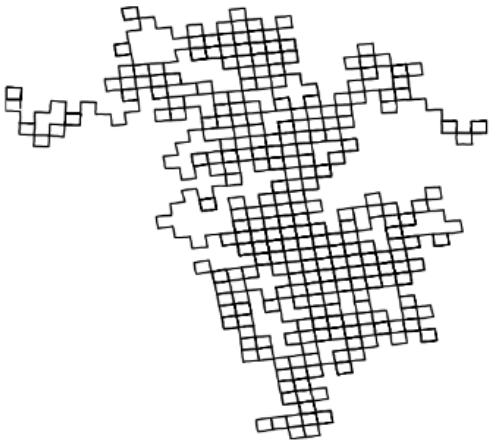

Fig. 8. LAGO's pose-graph

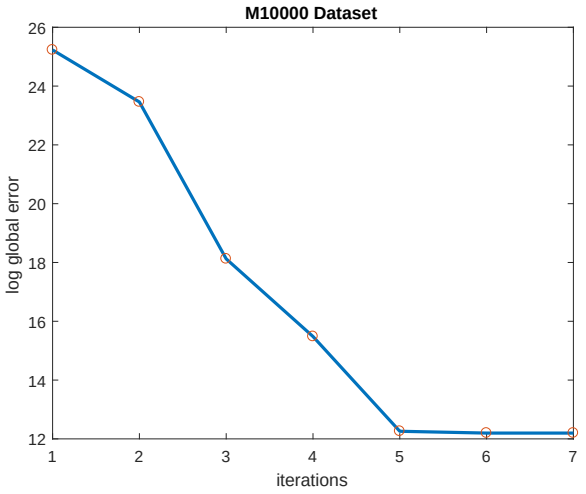

Fig. 9. Global error per iteration

and $q_x, q_y, q_z, q_r \in \mathbb{R}$. Thus, it is a sum of a scalar q_r and a vector part $q_v = [q_x, q_y, q_z]$.

The unit quaternions, given by Eq. (21), are a sub group of quaternions that are used to represent rotations. They satisfy the condition $\|q_u\| = 1$. A full quaternion belongs to the $\mathbb{R}^4$ space. However, the unit quaternion belongs to a subspace of $\mathbb{R}^4$ called $\mathbb{S}^3$, which represents the unit sphere in $\mathbb{R}^4$ [15] [2].

$$q_u = \frac{q}{\|q\|} = \frac{1}{\sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2}} q \tag{21}$$

A unit quaternion just need to maintain its unit lenght throughout the optimization process. However, the addition of this constraint degrades the performance of the algorithm [9]. The problems about the quaternion parameterization occur because rotations have three DOF and the quaternion can change in four directions. Since estimation algorithms, in general, expect variables from euclidean vector spaces [5], the goal is to use a representation with three parameters, such as euler angles, but without singularities. However, there is no SO(3) parameterization with only three parameters that has no singularities [5].

To overcome these problems, the state is globally represented by a unit quaternion, but local perturbations around the current state have a minimal representation, ideally behaving as an euclidean space [5]. This parametrization is related to manifold theory and exponential maps [18].

A manifold is a mathematical space that, in a global scale, is not an euclidean space, but can be locally approximated by one [17]. In other words, "every point of a manifold has a neighborhood that can be mapped bidirectionally to $\mathbb{R}^n$" [6].

The space of rotations and S^3, the unit quaternions, are manifolds and can be locally mapped to a euclidean space. Therefore, the parameterization problem can be dealt with using unit quaternions to represent the orientation of the state, and defining a operator $\boxplus$ that maps a local variation in the euclidean space to a variation on the manifold [12].

The $\boxplus$-method, developed by Hertzberg [5], defines the mapping functions between the manifold and the euclidean spaces, which are called the exponential and logarithmic maps [18]. The operator $\boxplus$, stated in Eq. (22), represent the exponential map, which performs a rotation around axis δ with an angle $\|\delta\|$, accoding to Hertzberg et al. [6].

$$p \boxplus \delta = p \; exp\Big(\frac{\delta}{2}\Big) \tag{22}$$

The operator $\boxminus$, stated in Eq. (23), represent the logarithmic map, which computes the rotation from p to q. The global difference in manifold space is mapped to a local perturbation in euclidean space [12].

$$q \boxminus p = 2 \cdot log(p^{-1} q) \tag{23}$$

The exponential and logarithmic functions are given by the Eqs. (25) and (27), respectively, considering a quaternion with real part w and vector part u [2] [5].

The exponential map function maps a vector v into a unit quaternion q:

$$exp: \mathbb{R}^3 \rightarrow \mathbb{S}^3 \tag{24}$$

$$exp(v) = q = \begin{cases} \left[sin(\|v\|)\frac{v}{\|v\|}, cos(\|v\|)\right] & \text{for} \quad \|v\| \neq 0 \\ [0,0,0,1] & \text{for} \quad v = 0 \end{cases} \tag{25}$$

The logarithmic map function maps a unit quaternion q into a vector v:

$$log: \mathbb{S}^3 \rightarrow \mathbb{R}^3 \tag{26}$$

$$log(q) = v = \begin{cases} 0 & \text{for} \quad u = 0 \\ \frac{atan(\|u\|/w)}{\|u\|}u & \text{for} \quad u \neq 0, w \neq 0 \\ \frac{\pi/2}{\|u\|}u & \text{for} \quad w = 0 \end{cases} \tag{27}$$

Thus, the pose of the robot is represented by a translation vector and a unit quaternion:

$$x_i = [x, y, z, q_x, q_y, q_z, q_r] \tag{28}$$

The error function can be approximated as:

$$e_{ij} = e_{ij}(\breve{x} \boxplus \delta_x) \backsimeq e_{ij} + J_{ij}\delta_x \tag{29}$$

where e_{ij} is the difference between the predicted and the actual measurement, as stated in Eq. (30)

$$e_{ij} = \hat{z}_{ij} \boxminus z_{ij} \tag{30}$$

The Jacobian is given by Eq. 31.

$$J_{ij} = \frac{\partial e_{ij}(\breve{x} \boxplus \delta_x)}{\partial \delta_x} \tag{31}$$

However, now the Jacobian matrix is computed numerically, according to [5]. A small perturbation is applied for each degree of freedom.

$$J_{ij} = \frac{e_{ij}(x \boxplus dv_j) - e_{ij}(x)}{d} \tag{32}$$

where e_{ij} is the error function, d is a small positive scalar and v_j is the unitary vector corresponding to the DOF.

Thus, the incremental addition to the initial guess is defined by the exponential map:

$$x^* = \breve{x} \boxplus \delta_x \tag{33}$$

The operator $\boxplus$ first converts the rotational part of δ_x to a full quaternion and then apply the transformation to $\breve{x}$ [12] [3]. The formulation is described using pseudocode notation in algorithm 2.

Algorithm 2 3D Pose Graph Optimization

procedure READ GRAPH
  $x \leftarrow$ vertices
  $z \leftarrow$ edges
  $\Omega_{ij} \leftarrow$ inf matrices
endprocedure
while not converged **do**
  preallocate sparse J
  preallocate e
  given scalar d
  for all measurements m **do**
    compute error function e_{ij}
    $e_{ij} \leftarrow \Omega_{ij} e_{ij}$
    $z_{ij} \leftarrow$ measurement m
    procedure COMPUTE THE JACOBIAN
      for each dependant random variable rv **do**
        for $k = 1 : dof(e)$ **do**
          $x_{ie} \leftarrow x_i \boxplus de_k$
          $e_d \leftarrow (x_{ie}^{-1} x_j) \boxminus z_{ij}$
          $e_d \leftarrow \Omega_{ij} e_d$
          $J \mathrel{+}= \frac{e_d - e_{ij}}{d}$
        endfor
      endfor
    endprocedure
    $e \mathrel{+}= e_{ij}$
  endfor
  $H \leftarrow J^T J$
  $b \leftarrow J^T e$
  $\delta_x \leftarrow$ sparsesolve$(H\delta_x = -b)$
  $x = x \boxplus \delta_x$
endwhile

D. 3D Dataset Evaluation

For the 3D evaluation, the nodes are listed in the format "ID x y z q_x q_y q_z q_w", which corresponds to the number of the pose and its respective 3D position and orientation in unit quaternion representation. All edge lines have the format: "IDfrom IDto x y z q_x q_y q_z q_w I11 ... I66". The first two numbers "IDfrom IDto" correspond to the ID of observing and observed nodes i and j. The x, y and z compose the translation vector between nodes, and q_x, q_y, q_z, q_w is the unit quaternion corresponding to the rotation between the two nodes. The numbers I11 ... I66 are the 21 top triangular elements of the 6x6 information matrix, stated in Eq. (34).

$$\Omega_{ij} = \begin{bmatrix} I11 & I12 & I13 & I14 & I15 & I16 \\ & I22 & I23 & I24 & I25 & I26 \\ & & I33 & I34 & I35 & I36 \\ & & & I44 & I45 & I46 \\ & \ddots & & & I55 & I56 \\ & & & & & I66 \end{bmatrix} \tag{34}$$

The evaluated dataset for the 3D case represents the movement of a robot on a surface of a sphere. The graph

has 2500 poses and 4949 contraints. Fig. 10 shows the initial graph configuration, a sphere corrupted by noise. Figure. 11 shows that the system is able to correctly optimize the graph, displaying the optimized sphere. Figure 12 shows the global error per iteration of the evaluation in logarithmic scale.

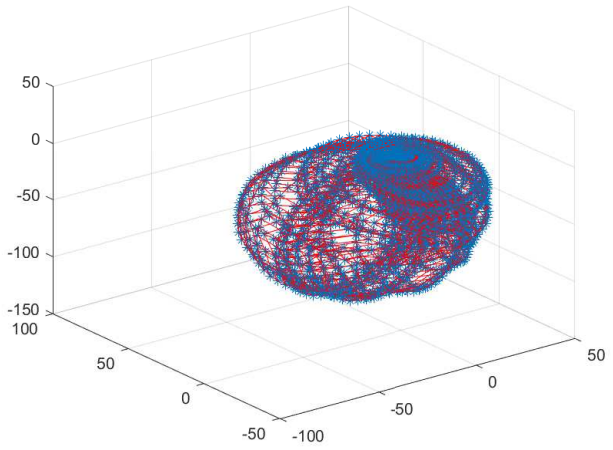

Fig. 10. Initial pose-graph

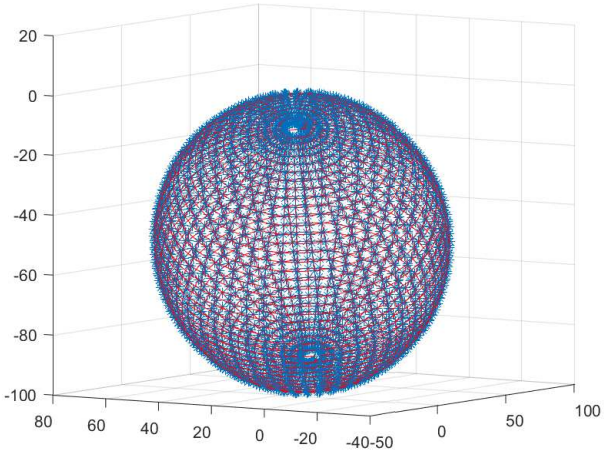

Fig. 11. Optimized pose-graph

IV. RGB-D SLAM System

This section presents the implementation of a complete RGB-D SLAM system, detailing the proposed methodology and software libraries used.

A. Proposed Methodology

The ROS [25] framework is built as a large number of small programs, called nodes, that communicate one another through messages, carried by topics [19]. Each node either subscribes to topics or publishes them. The proposed SLAM system is implemented as a ROS package, with a C++ main ROS node and auxiliary header files. The final output of the system is the global point cloud map and the optimized trajectory of the robot. Fig. 13 shows the

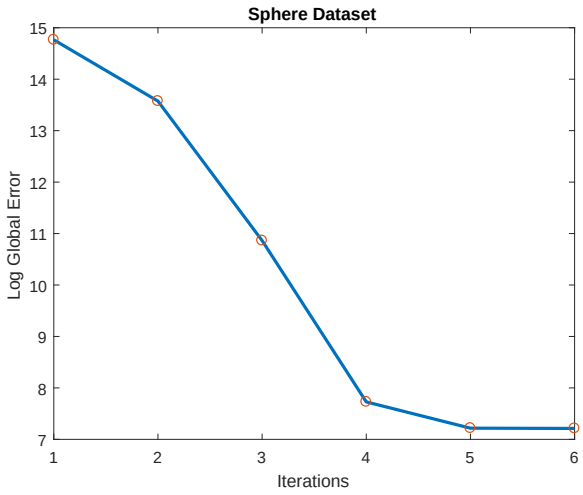

Fig. 12. Sphere dataset - Global error per iteration

schematic overview of the proposed approach. The RGB-D data is used to estimate visual odometry using FOVIS, to create point clouds, and to detect loop closures for graph construction. The g^2o framework is used to store the nodes and edges of the graph and to perform the optimization.

B. Data Acquisition

An RGB-D camera is composed by two sensors: a color camera and an infrared camera, both modeled as a pinhole camera, shown in Fig. 14. The camera model is used to map information from world coordinates to image coordinates, and to define the position of the camera in world coordinates.

The center of the camera is the point C, which is the center of the euclidean coordinate system $[X_c, Y_c, Z_c]$, showed in Fig. 14. The image plane is located at $Z_c = f$, which is the focal length of the camera. The principal point P is where the image plane meet the Z axis, and has the coordinates c_x and c_y in the image plane.

This model is used to map a point in space with coordinates **X'** $= [X', Y', Z']$ to a point $[fX'/Z' + c_x, fY'/Z' + c_y]$ on the image plane [28], which can be written as stated in Eq. (35).

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x & 0 \\ 0 & f & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X' \\ Y' \\ Z' \\ 1 \end{bmatrix} \tag{35}$$

where the matrix carrying the focal length and principal point parameters is called the calibration matrix, or intrinsic matrix, and $[u, v, 1]$ are the coordinates of the mapped point in the image plane, written in homogeneous coordinates.

The extrinsic parameters are composed by a 3x3 rotation matrix and a translation vector, defined in Eq. (36), and define where the camera is located in world frame coordinates, mapping a point $[X, Y, Z, 1]$ in world frame homogeneous coordinates to the camera coordinate frame.

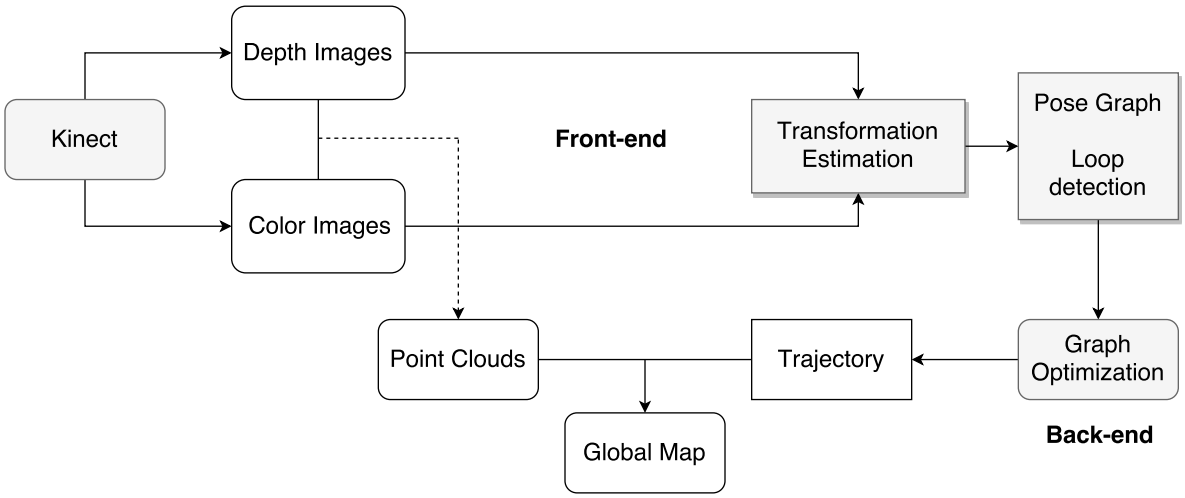

Fig. 13. Schematic overview of the proposed approach

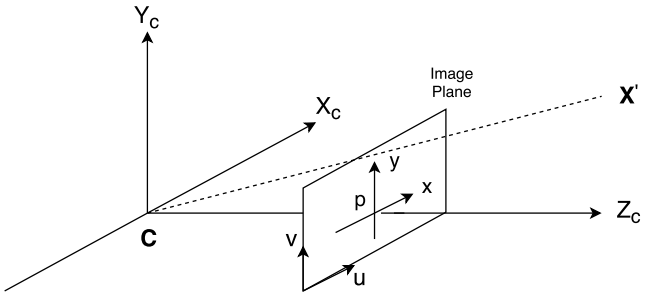

Fig. 14. Pinhole Camera Model

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{36}$$

Both extrinsic and intrinsic parameters are obtained through calibration, and are used to register the depth images to the same coordinate system of the RGB images.

The main ROS node receives color and depth images provided by the RGB-D sensor in form of ROS topics. A point cloud is generated combining color and depth images, using the ROS library *depth image proc* [27] and the Eqs. (37) (38) and (39). These equations map a point (v, u) in the image plane of the camera to a 3D coordinate of the scene. The Point Cloud Library (PCL) [30] is used to store, visualize and manipulate the point clouds.

$$Z = depth\_image[v, u] \tag{37}$$

$$X = (u - c_x)\frac{Z}{f_x} \tag{38}$$

$$Y = (v - c_y)\frac{Z}{f_y} \tag{39}$$

A voxel grid filter from PCL is applied to reduce the number of points used to create the global map due to the large amount of memory required by the point cloud representation. In Fig. 15 is shown a point cloud with the full set of measured points and in Fig. 16 is shown the same point cloud after the filtering process.

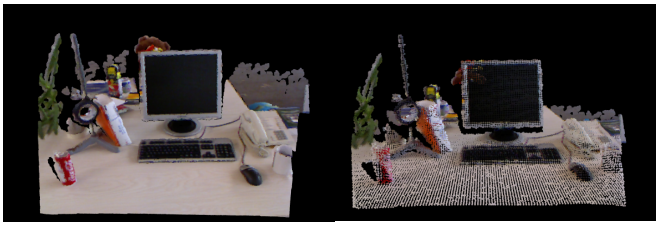

Fig. 15. Point Cloud

Fig. 16. Downsampled Cloud

C. Visual Odometry

The FOVIS library [1] is used to obtain a pose estimation. The FOVIS ROS node subscribes to the corresponding topics: rgb camera information, depth camera information, depth image and color image. Afterwards, it publishes the odometry topic, sent to the main node. Each pose i of the robot has the format stated in Eq. (40).

$$\mathbf{x}_i = [x_i, y_i, z_i, q_x, q_y, q_z, q_r] \tag{40}$$

where x_i, y_i, z_i are the euclidean coordinates of the robot and q_x, q_y, q_z, q_r compose the unit quaternion that represents the orientation of the robot.

D. Keyframe selection and Loop Closure

Ideally, to detect a loop closure, it would simply require a comparison between the current frame and all past frames. However, it is computationally infeasible [26]. To overcome this problem, only a subset of frames, called keyframes, is selected to be compared.

The first frame is sellected as a keyframe and is matched against the next frames. When the number of matched

inliers is below a threshold, it means the robot has made a significant movement and a new keyframe is chosen. Every new keyframe is matched againt the previous ones. However, two keyframes are only compared if their global pose is close enough, given a threshold.

Every time a new keyframe is detected, a node is added to the graph with the corresponding position and orientation. If the number of matched inliers between the current keyframe and a past keyframe is above a threshold, called loop closure inliers threshold, then a loop is detected and an edge is added to the graph. This edge is composed by the transformation between the two frames and the associated information matrix, that comprises the uncertainty of the measurement [19].

The ORB features [7] are used in this methodology. Besides their efficiency, they provide good invariance to changes in illumination and motion. The implementations from the OpenCV library [13] are used for the feature detection and matching. In Fig. 17 is shown the matched points between two keyframes. To be robust againt outliers, an rejection filter is applied using the fundamental matrix and the RANSAC method [24].

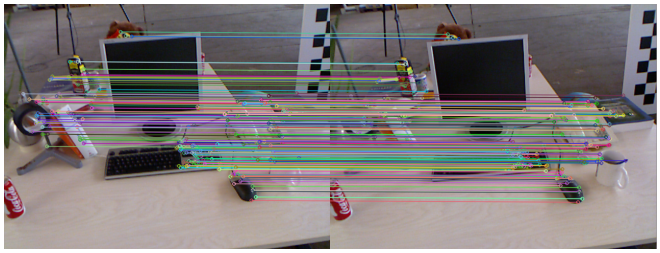

Fig. 17. Feature Matching

The PCL implementation of the ICP algorithm is used to determine the relative transformation between two keyframes with near global pose, using their corresponding point clouds. The Algorithm 3 details the procedure of keyframe selection and loop closure detection in pseudocode notation.

E. Graph Optimization

The g^2o framework [29] is used to register the graph and perform the optimization when a loop is detected. The information matrix for every measurement is the identity matrix multiplied by the number of inliers.

F. Global Map

The global map is constructed through the alignment of all stored point clouds. The PCL function "transformPointCloud" applies a given transformation to a point cloud. This function is used to align all keyframe point clouds with their respective optimized poses. Finally, each one is added to the global map at the same coordinate system. This formulation is described using pseudocode notation in algorithm 4.

V. EXPERIMENTS

This section presents the qualitative results obtained with the proposed methodology, using a differential drive commercial robot iRobot Create, equiped with a Microsoft Kinect v2 and a laptop. All the experiments were conducted in the Robotics Laboratory from the Pontifical Catholic University of Rio de Janeiro. In Fig. 18 is shown the assembled robot performing SLAM.

Algorithm 3 Keyframe Selector and Loop detection

for every new frame i **do**
  $P_i \leftarrow i^{th}$ fovis pose
  detect features
  feature match(features i, features i-1)
  $n\_inliers \leftarrow$ outlier rejection
  if $n\_inliers < threshold$ **then**
    $keyframe_j \leftarrow currentframe_i$
    graph vertex j $\leftarrow P_j$
    compute $T_{j-1,j}$ between keyframes (FOVIS)
    add edge
    compare current and past keyframes
    if loop detected with keyframe k **then**
      compute $T_{k,j}$ between keyframes (ICP)
      add edge
    endif
  endif
endfor

Algorithm 4 Global Map Construction

initialize global map
$keyframes \leftarrow$ get keyframes
$poses \leftarrow$ get poses
for all keyframes **do**
  $pointcloud_k \leftarrow$ get keyframe pointcloud
  $aligned\_cloud = transform(keyframe_k, pose)$
  global map += $aligned\_cloud$
endfor

Fig. 18. Robot performing SLAM

In the following experiment, the robot maps a entire room at once. The Figs. 19 and 20 are parts of the same map. The robot was able to create a map of the laboratory in real

time. Some problems were encountered with high motion speed or rough movements. However, the graph optimization overcome major misalignment problems caused by odometry drift. Despite some misalignments, the map is still consistent due to the loop closure detection and optimization.

Fig. 19. Point cloud map

Fig. 20. Point cloud map 2

VI. NUMERICAL EVALUATION

To numerically analyze the proposed methodology with experimental data, it is necessary to have the ground truth trajectory of the robot in the environment. However, this would require motion detector systems or other external measurement device. Thus, the numerical evaluation of the system is made using the RGB-D benchmark [20] from Technical University of Munich, which provides datasets of color and depth image sequences of a kinect sensor, under different conditions. All sequences have a corresponded ground-truth trajectory, obtained with a high precision motion capture system. All tests were conducted in a notebook with an Intel Core i7 6700 HQ processor with 2.60 GHz and 16 GB of RAM, running Ubuntu Linux 14.04 LTS and ROS Indigo.

A. Evaluation Metrics

This evaluation employs the Absolute Trajectory Error (ATE) to compare the estimated trajectory with the provided ground-truth trajectory. The ATE compares absolute distances between both trajectories and evaluates the global consistency [20].

The mean, minimum, maximum and root mean square errors are evaluated for each Dataset. Given the trajectory estimate with translational components $\hat{x} = [\hat{x}_1, \cdots, \hat{x}_n]$, and the ground truth trajectory with translational components $x = [x_1, \cdots, x_n]$, the root mean square error (RMSE) is given by Eq. (41).

$$RMSE = \left(\frac{1}{n} \sum_{i=1}^{n} \|\hat{x}_i - x_i\|^2 \right)^{1/2} \tag{41}$$

The ground truth trajectories have the format "timestamp tx ty tz qx qy qz qw", where timestamp is the time of each pose in unix epoch time, "tx, ty, tz" is the translation vector, and "qx qy qz qw" is a unit quaternion. The estimated trajectories and the ground truth trajectories are aligned using the timestamps of each pose [20].

B. Dataset Evaluation

The first sequence is called "fr1 room" and corresponds to a movement of $15.989m$ of ground-truth trajectory length, $0.334m/s$ of average translational velocity and $29.882deg/s$ of average angular velocity. In Fig. 21 is shown the comparison between the estimated trajectory and the ground-truth trajectory. In Fig. 22 is shown the resulted global point cloud map, and in Tab. I is shown the comparison between the ATE error using only the visual odometry of FOVIS and using the proposed methodology.

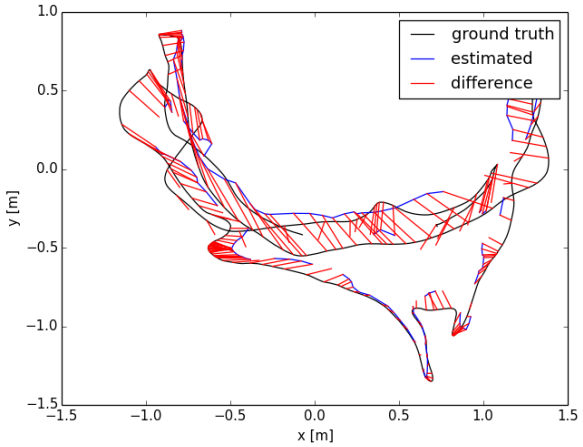

Fig. 21. fr1-room - Optimized

TABLE I
ATE EVALUATION OF THE FR1 ROOM DATASET

Error (m)	FOVIS	This work
RMSE	0.2807	0.1987
Mean	0.2432	0.1722
Min	0.0256	0.0159
Max	0.6446	0.4072

The second evaluation is the "fr3 long office household" sequence, with $21.455m$ of ground-truth trajectory length, $0.249m/s$ of average translational velocity and $10.188deg/s$ of average angular velocity. In Fig. 23 is shown the comparison between the estimated trajectory and the ground-truth trajectory. In Figs. 24 and 25 is shown, respectively,

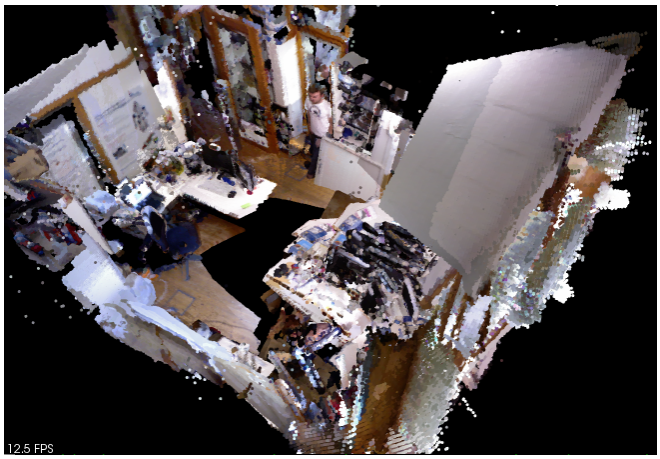

Fig. 22. fr1-room - Point Cloud Map

the initial point cloud and the resulted global point cloud map. In Tab. II is shown the comparison between the ATE error using only the visual odometry of FOVIS and using the proposed methodology.

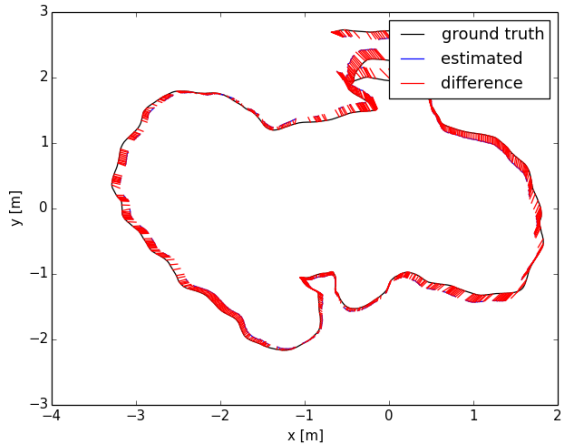

Fig. 23. fr3-long-office - Optimized

TABLE II
ATE EVALUATION OF THE FR3 LONG OFFICE DATASET

Error (m)	FOVIS	This work
RMSE	0.2717	0.1238
Mean	0.2329	0.1101
Min	0.0381	0.0156
Max	0.5375	0.2685

Both results showed that the system was able to create consistent maps of indoor environments, and the loop closure and graph optimization provided a considerable improvement of the FOVIS trajectory estimation. The root mean square, maximum, minimum and mean errors were all minimized. The errors in the first evaluation are larger than the second one due to the higher translational and angular velocities of the camera.

Fig. 24. fr3-long-office: Initial Point Cloud

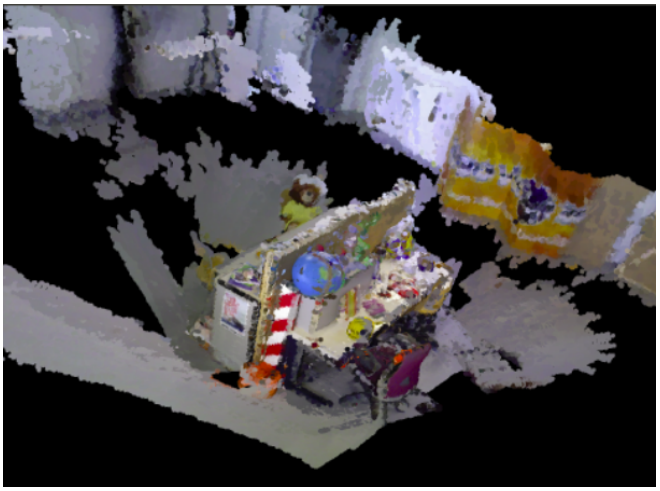

Fig. 25. fr3-long-office: Point Cloud Map

The Tables III and IV show comparisons between the proposed implementation and other methods from the literature, using the ATE RMSE metric and the datasets freiburg room, with 3Hz of frame rate, and freiburg long office household, with 30 Hz of frame rate. The first comparison, shown in Tab. III, is made with the first version of the rgbdslam method, developed by Endres et al. [8] and FOVIS [1]. This work outperformmed both methods.

TABLE III
COMPARATIVE RESULTS FOR THE FR1 ROOM DATASET

Method	ATE RMSE (m)
This work	0.1987
RGB-D SLAM	0.2190
FOVIS	0.2807

The present system is also compared with FOVIS and LSD-SLAM [14], a keyframe-based SLAM system for monocular cameras developed by Engel et al. in 2014. This work showed satisfatory results for a real time performance,

achieving lower errors than both methods.

TABLE IV

COMPARATIVE RESULTS FOR THE FR3 LONG OFFICE DATASET

Method	ATE RMSE (m)
This work	0.3064
LSD-SLAM	0.3853
FOVIS	0.5144

VII. CONCLUSIONS

This work presented a pose-graph optimization tool for MATLAB for 2D and 3D trajectories, and an RGB-D SLAM system for mobile robots. The pose-graph tool was able to optimize graphs with a large number of constraints and a considerable initial error in both 2D and 3D cases. The SLAM system was able to perform real-time, creating consistent maps and achieving an acceptable global error in localization. The SLAM implementation was tested in a low cost platform, using only open source software and affordable hardware, and it has a vast applicability. The system also outperformed state-of-the-art methods from the literature.

Future works include a C++ implementation of the pose-graph optimization tool, to allow its use in real time SLAM implementations. For the RGB-D SLAM system is proposed the use of a bag of words formulation for place recognition in the loop closure problem. Another important improvement is to extend the implementation to the use in dynamic environments, which would considerably increase the applicability of the system.

REFERENCES

[1] 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, in Int. Symposium on Robotics Research (ISRR), 2011.

[2] A. Ude, Filtering in a unit quaternion space for model-based object tracking, Robotics and Autonomous Systems, 1999, vol. 28, pp. 163–172.

[3] A. Ude, Nonlinear least squares optimisation of unit quaternion functions for pose estimation from corresponding features, Proceedings of IEEE Fourteenth International Conference on Pattern Recognition, 1998, vol. 1, pp. 425–427.

[4] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid and J. Leonard, Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age, IEEE Transactions on Robotics, 2016, vol. 32, pp. 1309-1332.

[5] C. Hertzberg, A Framework for Sparse, Non-Linear Least Squares Problems on Manifolds, 2008.

[6] C. Hertzberg, R. Wagner, U. Frese, and L. Schröder, Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds, Information Fusion, 2013, vol. 14, pp. 57–77.

[7] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, ORB: An efficient alternative to SIFT or SURF, in Proceedings of IEEE international conference on Computer Vision (ICCV), 2011, pp.2564–2571.

[8] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, An evaluation of the RGB-D SLAM system in Proceedings of IEEE International Conference on Robotics and Automation (ICRA), 2012, pp. 1691–1696.

[9] F. Grassia, Practical Parameterization of Rotations Using the Exponential Map, Journal of Graphics Tools, 1998, vol. 3, pp. 29–48.

[10] F. Lu and E. Milios, Globally consistent range scan alignment for environment mapping, Autonomous Robots, 1997, vol. 4, pp. 333–349.

[11] G. Grisetti, C. Stachniss, and W. Burgard, Nonlinear constraint network optimization for efficient map learning, Trans. Intell. Transport. Sys., 2009, Vol. 10, No. 3, pp. 428–439.

[12] G. Grisetti, R. Kümmerle, C. Stachniss and W. Burgard, A Tutorial on Graph-Based SLAM, IEEE Intelligent Transportation Systems Magazine, 2010, vol.2, pp. 31-43.

[13] Itseez, Open Source Computer Vision Library, https://github.com/itseez/opencv, 2015.

[14] J. Engel, T. Schöps, and D. Cremers, LSD-SLAM: Large-Scale Direct Monocular SLAM, in Proceedings of Computer Vision – ECCV 2014: 13th European Conference, pp. 834–849.

[15] J. Gallier, Notes on differential geometry and Lie groups, 2012.

[16] J. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In Proceedings of IEEE International Symposium on Computational Intelligence in Robotics and Automation (CIRA), 1999.

[17] J. Lee, Introduction to Smooth Manifolds, 2001.

[18] J. Soares, and M. Meggiolaro, A Pose-Graph Optimization Tool for MATLAB, X Congresso Nacional de Engenharia Mecânica (CONEM), 2018.

[19] J. Soares, and M. Meggiolaro, Keyframe-based RGB-D SLAM for Mobile Robots with Visual Odometry in Indoor Environments using Graph Optimization, IEEE 15th Latin American Robotics Symposium LARS, 2018 (Submitted).

[20] J. Sturm, N. Engelhard, F. Endres, W. Burgard and D. Cremers, A benchmark for the evaluation of RGB-D SLAM systems, in 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012.

[21] K. Horn, Some notes on unit quaternions and rotation, 2001.

[22] L. Carlone, and A. Censi, From angular manifolds to the integer lattice: Guaranteed orientation estimation with application to pose graph optimization, CoRR, 2012.

[23] L. Carlone, R. Aragues, J. Castellanos, and B. Bona. A fast and accurate approximation for planar pose graph optimization. The International Journal of Robotics Research, 33(7):965–987, 2014.

[24] M. Fischler and R. Bolles, Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, Commum ACM, 1981, vol. 24, pp. 381 - 395.

[25] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler and A. Ng, ROS: an open-source Robot Operating System, in Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA), Workshop on Open Source Robotics, 2009.

[26] P. Henry, M. Krainin, E. Herbst, X. Ren and D. Fox, RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments, The International Journal of Robotics Research, 2012, vol.31, pp. 647-663.

[27] P. Mihelich, depth_image_proc, http://wiki.ros.org/depth_image_proc

[28] R. Hartley, and A. Zisserman, Multiple view geometry in computer vision, Cambridge university press, 2003.

[29] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige and W. Burgard, g2o: A General Framework for Graph Optimization, in Proceedings of IEEE International Conference on Robotics and Automation (ICRA), 2011.

[30] R. Rusu and S. Cousins, 3D is here: Point Cloud Library (PCL), in Proceedings of IEEE International Conference on Robotics and Automation (ICRA), 2011.

[31] R. Wagner, O. Birbach and U. Frese, Rapid development of manifold-based graph optimization systems for multi-sensor calibration and slam. In 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3305–3312.

[32] S. Thrun and M. Montemerlo. The graphslam algorithm with applications to large-scale mapping of urban structures. International Journal on Robotics Research, 25:403–430, 2005.

[33] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard and J. McDonald, Robust real-time visual odometry for dense RGB-D mapping, in Proceedings of IEEE International Conference on Robotics and Automation (ICRA), 2013, pp. 5724-5731.

[34] Y. Jia, Quaternions and Rotation, 2015.