Rheinische
Friedrich-Wilhelms-
Universität Bonn

Institute for Computer Science
Department VI
Autonomous Intelligent Systems

Rheinische
Friedrich-Wilhelms-Universität Bonn

Master thesis

Calibration and Simultaneous Localization and Mapping with Multiple RGB-D Cameras

Author:
Pablo Antonio Aponte Del Real

First Examiner:
Prof. Dr. Sven Behnke

Second Examiner:
Dr. Dirk Schulz

Advisor:
Dr. Jörg Stückler

Submitted: March 11, 2015

Declaration of Authorship

I declare that the work presented here is original and the result of my own investigations. Formulations and ideas taken from other sources are cited as such.

It has not been submitted, either in part or whole, for a degree at this or any other university.

Location, Date

Signature

Abstract

Reconstruction of a 3D scene while simultaneously finding the location of the camera(s) is one of the most important tasks in robotics. The resulting map of the scene can be used in other tasks like path finding, navigation and object interaction. Currently several techniques can perform this task with impressive results, but most of them require to process a big amount of captured frames to be able to create a complete map.

Our approach relies on the *Multi-Resolution Surfel Maps (MRSMap)* by Stückler and Behnke, 2014. This technique is extended to use several cameras instead of one, permitting the creation of more accurate maps, but also introducing the need for correct extrinsic parameters. For the calculation of this requirement, we propose an extension of the pose graph optimization used in the MRSMap algorithm. The graph will change the current edges to hyper-edges, in order to include the calibration nodes that represent the offset of the different sensors. This way, it is possible to simultaneously compute the extrinsic parameters without loosing the efficiency of the algorithm, permitting it to continue working in real time.

Finally, to improve the optimization phase even further, we detect possible undefined registrations (e.g., a white featureless wall) via the covariance matrix that models the uncertainty of each registration. Once it is detected, its effect in the pose graph optimization is reduced by increasing the information matrix of the edge that represents it.

This novel ideas were tested in different environments, giving the best results with datasets with rotational movement, as well as an improvement in comparison to the maps obtained with the system that works only with one camera and in the same environment.

Contents

1 Introduction

The use of RGB-D cameras in robots has been increasing rapidly over the last years, since it became inexpensive after the release of the Kinect by Microsoft. With these sensors, a robot may perceive the environment in three dimensions with a high accuracy and in real time. Since the depth and color information is captured and processed by the camera, there is no need to use computationally expensive algorithms, such as stereo vision or structure from motion, to generate a point cloud.

Taking advantage of this new perception of the surroundings, a new range of algorithms is being developed. One of the most popular new algorithms is the Simultaneous Localization And Mapping using RGB-D cameras (RGB-D SLAM from now on), which finds an accurate estimation of the position of the camera(s) and at the same time, creates a 3D map of the environment. This algorithm is really useful in the area of robotics, since the output can be used in several and different algorithms, e.g. path planning and object interaction. It is specially useful in those cases where the robot needs to operate without any prior knowledge of its surroundings and fully autonomously.

Being such an important task in robotics, RGB-D SLAM has received the attention of many experts in the area and nowadays we can find several techniques to perform it, e.g. the ones from Stückler and Behnke (2014), Scherer and Zell (2013), Bylow et al. (2013) and Newcombe et al. (2011). Even though all of them differ from each other, they have one common aspect: All of them use only one camera.

The notion of using several sensors instead of only one to obtain a 3D map representation is not new either, as we can see in the works of Yang et al. (2014) and Heng, Lee, et al. (2014). In both cases, visual simultaneous localization and mapping (vSLAM) is used to achieve the task. This means that it uses only color information to register the images and create the map. Until the moment of the creation of this thesis, there is no known work that utilizes multiple RGB-D sensors to do SLAM, making the results of this project important for the scientific community.

1.1 Problems

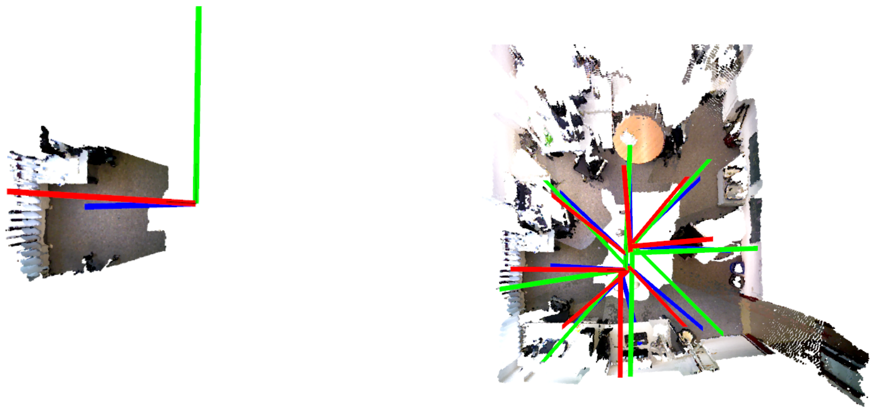
As stated before, the RGB-D cameras got popular when their price went down due to the release of the Kinect in 2010 for Xbox 360 by Microsoft. In 2012, other companies release their own RGB-D sensors at a competitive price, such is the case of Asus Xtion from Asus, the Carmine from Primesense and a version of the Kinect that could be connected directly to the computer. Since then, several researchers have focus their efforts to exploit the capabilities of these sensors, but due to the noise produced by the cameras or other algorithm limitations, they found unexpected behaviours. It is important to find a solution to this problems in order to obtain a good result.

1.1.1 Hardware

It is important to take in consideration the hardware limitations of the sensors. For example, it is necessary to remember that these devices have a lot of noise in the depth information captured, and for that reason, several algorithms rely on statistical average or filters to reduce it. Also, depending on the technology the camera uses, e.g. time of flight or structured light, it may not be possible to position the devices to share the point of view without adding more noise to the depth information. If the cameras are used in outdoors experiments, the captured images will contain too much noise and it will give unexpected or random result, in which case it is recommended to avoid this setting if possible.

The lighting in the environment also has an effect on this camera, not only in the depth information but also in the color data, thus, it is important to keep a similar illumination setting while using this kind of devices. As a final remark, it is important to note the maximum distance and minimum distance that the sensor works with, since it is possible to find cameras that may not work with distances bigger than 2 meters, for instance, making the process of scanning a large room a time consuming task.

Most of the hardware problems can be avoided, but it is important to know them before using these devices, since they may generate enough noise in the data to make it unusable with the majority of the algorithms.



(a) Output of the first 8 frames using one camera.

(b) Output of the first frame of each of the 8 cameras.

Figure 1.1: Contrast between the first eight frames of the reference camera and the first frame of each of the eight cameras.

1.1.2 Software

The current algorithms that perform the RGB-D SLAM task have also some limitations. For instance, in the aforementioned algorithms, the pose difference between the current frame that is being process and any of the previous ones, have to be small in order to be able to register the frame correctly and to find the accurate position of the camera, if not, it will lose track of the sensor and it will create an incorrect map.

Another problem with the majority of the RGB-D SLAM algorithms is the time needed to create a complete map of the surroundings. As mentioned before, the difference between two frames must be small in order to get a correct registration. This limitation will affect the robot's speed, and in consequence, the time needed to obtain a map of the environment. Also, if the the sensor is moving too fast, it may generate motion blur in the color and depth data.

We propose to solve this problem by using several cameras to shorten the time needed for the robot to scan the complete range of 360°. As it is shown in figure 1.1, using an eight camera system already covers a big percentage of the map, and the robot will need to rotate a considerable smaller amount of degrees to obtain a complete scan. With only one sensor, the robot will need to process more frames to be able to obtain the same map as the robot with multiple cameras.

Another possible problem that may exist in some of the systems is the accumulation of the error in big datasets. This error is usually corrected via loop closures and constraints, but it is possible that the error is already too big before a loop closure occurs and the algorithm might not detect it; therefore, it will not apply the constraints associated with the loop. With a multi-camera approach, this problem can be avoided, since the device will have a field of view that covers almost the 360° of the field of view of the surroundings, in consequence, the field of view of one of the sensors will overlap with one of the frames captured by another device, creating a loop closure and creating the respective constraints that will reduce the error. This way, a multi-camera system is less prone to accumulate error; therefore, it will create a more accurate map and give a better estimate of the position of the sensors.

Even though a multi-camera system may fix the aforementioned problems, it also introduces a new one that must be solved. To obtain the correct registration between the frames of one camera to another, it is necessary to calculate the pose difference between the two sensors. In other words, it is needed to find the extrinsic parameters of all the devices with respect a reference point. Afterwards, this will be used as an offset to correctly estimate the position of the new frame, and then, register it and refine it. Without it, the algorithm will create a piece of the map with each camera, but will not be able to associate it correctly with respect to the piece generated by another sensor; therefore it will fail to create the map and find the correct pose of the device.

1.2 Calibration with multiple cameras

As it was mentioned before, to be able to use a multi-camera approach correctly, it is necessary to obtain the extrinsic parameters needed to calibrate the system. To achieve this task, it is possible to find several different approaches, but with some similarities between one another, and it is possible to generalize and group them. Even though not all of the existing calibration methods use RGB-D devices, it is possible to consider them, since they use multiple sensors and their approach can be adapted and used with other types of sensors.

Geiger et al. (2012) and Furgale et al. (2013) use a pattern such as the one that is observable in Figure 1.2, to find the extrinsic parameters between the sensors. Despite having good results, the need of a visual cue makes the system to be dependent on a human interaction with the environment. With the approach discussed in this project, the robot will operate fully autonomously; therefore the

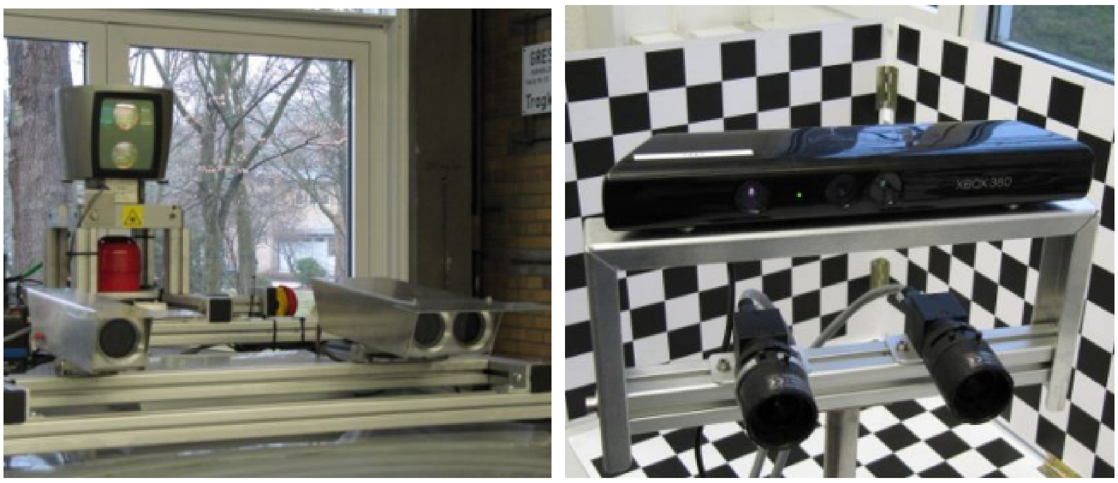


Figure 1.2: Experimental set-up with multiple cameras and a checkboard pattern from Geiger et al. (2012)

system may work in places where the human interaction may not be possible, e.g. disaster zones or other planets.

Other possibility is to use the visual odometry to complete the task as Heng, Li, et al. (2013), Yang et al. (2014) and Heng, Lee, et al. (2014) did. This approach will only use the color information, thus it can be used with RGB-D cameras omitting the depth data. Relying only on the color image, the system may be prone to more error due to illumination changes or featureless places that may contain depth features, such as columns in a white wall.

Finally, there is a third approach that is consider the best option for the current project. The task will be completed by minimizing the error of the edges in a pose graph, that will include nodes that represent the calibration parameters. This is possible to achieve by extending the work of Kuemmerle, Grisetti, and Burgard (2011), to a multi-camera system.

1.3 Approach

The proposed system will use one of the aforementioned RGB-D SLAM algorithms as its base. It will be extended and adapted to use multiple sensors, instead of a single one. This should reduce the time needed to obtain a complete and accurate map of the surroundings. As explained before, this will required the calculation of the extrinsic parameters. To compute these transformations, the system will use the g2o library that has been proven in the work of Kuemmerle, Grisetti, Strasdat, et al. (2011) and Kuemmerle, Grisetti, and Burgard (2011) to be a useful algorithm to obtain these parameters with one sensor.

The RGB-D SLAM algorithm chosen is the Multi-resolution surfel maps (MRSMAP from here on) from Stückler and Behnke (2014) from the University of Bonn. This state-of-the-art algorithm not only performs RGB-D SLAM, but it also uses g2o for the optimization phase; therefore, to integrate our approach is just necessary to modify this phase and this way it should not take more computational time, and it will be possible to use it in real time.

In a multi-camera system such as the one proposed, if one of the camera fails due to an undefined registration between the previous frames and a new frame that is featureless, i.e. a white wall, it may result in an error in the whole system, since the majority of the frames are connected to each other. Also, it may create walls or objects in places where they do not exist in reality, making the resulting map unusable. To avoid this problem, we propose an undefined registration detector.

The undefined registration detector will find which key frames are considered to have a low amount of features and give a false positive during the registration phase of the MRSMAP algorithm. This key frames are marked, and their edges will have an increased covariance until they obtain a good registration with another frame. To find out which registrations are good or not, the eigenvalues of the covariance matrix are used, and the maximum value must fall between two thresholds for it to be considered as a good registration.

The detector may generate maps with missing frames, since it may be possible that some frames never find a good registration and thus they are never included in the final representation of the map. In most of the results, these gaps were not big enough to be considered a serious problem.

1.4 Evaluation

For the evaluation of the system, the entropy of the map sharpness will be calculated as proposed by Droeschel et al. (2014) to quantify the correctness of the map. It is considered a useful tool to evaluate maps that do not have a ground truth to compare, such as the ones used in this project. Also, the extrinsic parameters obtained in each calibration will be compared to each other using the same camera structure and showing that the parameters will be the approximately the same, independent of the environment used.

One of the data sets will be tested with a different number of cameras. This way, it is possible to demonstrate the benefits of a multi-camera system in contrast of a single-camera one. To be able to compare both types of systems, the amount of frames needed to generate a complete map of the surroundings will be measured. In other words, the system will count the number of frames needed for all the cameras to do a loop closure with either the starting frame of another camera or its starting frame, using a dataset where the robot rotates around its own axis.

It is important to evaluate the other novelty introduced in this project; the undefined registration detector. To be able to test it adequately, it is necessary to use a dataset with one or more cameras observing an object that may generate such problem, e.g. white wall. For this evaluation, the extrinsic parameters found in the calibration datasets will be used as input and the entropy of the resulting map will be calculated to determine the correctness of the map.

The project has two ways of showing the resulting map, either using the point clouds generated from the images, or using the CPU TSDF library from Miller, 2015 to create a mesh. This library uses truncated signed distance function from Curless and Levoy (1996), to create a volume representation and then, it uses a straightforward implementation of marching cubes based on the work of Lorensen and Cline (1987), to create a mesh representation of the volume. In figure 1.3 , it can be appreciated a resulting map of the system using the library to create a mesh. Also, the mesh can be created using the Fast Fusion algorithm from Steinbruecker et al. (2014) that is publicly available and gives a good result.

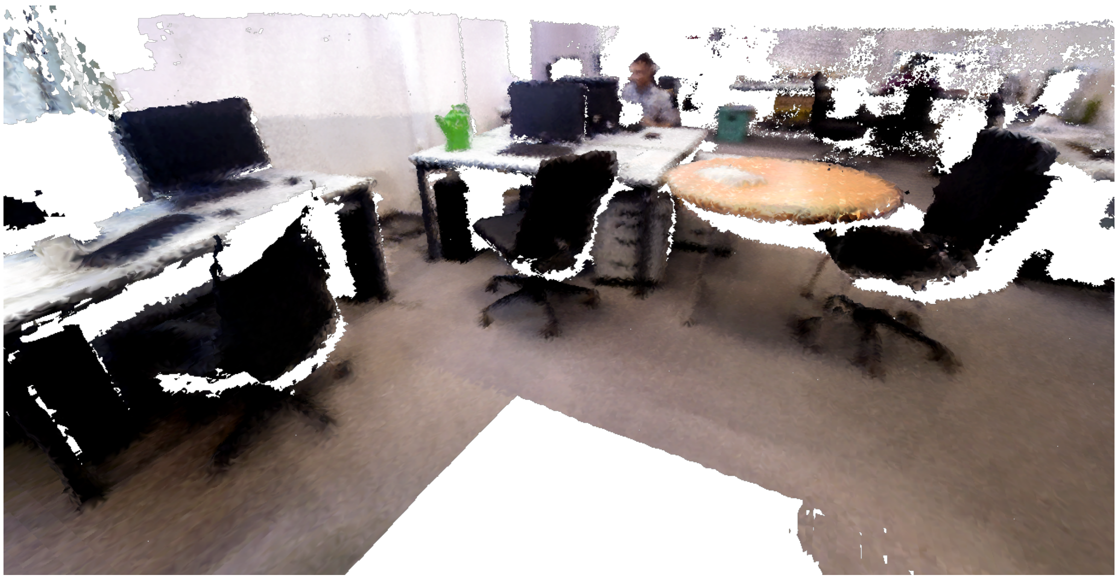


Figure 1.3: Mesh representation of the output using CPU TSDF library.

2 Related Work

2.1 RGB-D SLAM

As described by (Thrun et al., 2005), SLAM is a problem where "the robot acquires a map of its environment while simultaneously localizing itself relative to this map". This problem arises when the robot does not have any prior knowledge of the environment. The solution to this problem is extremely useful, since a robot can be sent to an unknown location and still be able to explore it completely while performing another task, such as object manipulation or path planning to reach a target. For this purpose, several techniques can be used, each depending on the sensors installed on the robot.

If the SLAM task is performed using the data obtained by a RGB-D camera,it is called RGB-D SLAM. It can be accomplish using different state-of-the-art approaches, such as e.g. KinectFusion from Newcombe et al. (2011), Efficient Onboard RGB-D SLAM of Scherer and Zell (2013), and Multi resolution surfel maps from Stückler and Behnke (2014).

KinectFusion is an algorithm developed by Microsoft's research team to show the capabilities of their RGB-D camera, Kinect. It is important to note that the Kinect was one of the first low-cost RGB-D cameras available in the market, designed by Microsoft for the XBOX gaming console to use augmented reality in their games; the scientific community saw a great value in such devices, and started using it for research in robotics and computer vision. KinectFusion uses Truncated Signed Distance Function (TSDF from now on) from Curless and Levoy (1996) to store the data that represents the environment. The key idea of this algorithm is to find and track the position of the camera using Iterative Closest Points algorithm, and later on, to fuse the new data into the TSDF volume by averaging the new values with the previously ones obtained. Using an incremental average of the values, the algorithm can reduce the noise of the captured data, or in some cases, it may remove or add objects that have moved or disappeared. Two of the drawbacks of this algorithm are the necessity to use the graphics processing unit (GPU) to perform the task in real time and that it only uses the depth information, giving as a result a colorless map.

Since the work of Newcombe et al. (2011) generated better results than other state-of-the-art techniques, more researchers tried a similar approach. For example, Bylow et al. (2013) use a similar approach, but in this case, the pose of the camera was obtained by minimizing the values given by the TSDF that represents the map when it was queried using the current image information. This project also achieved results that outperformed the competition at the moment of its publication, but also had the same problem as the KinectFusion; as it needed a GPU to achieve real-time performance. Also, as before, the algorithm only uses the depth data to compute the camera poses, but this time, the color information is also used to generate the mesh, giving a complete colored map of the environment.

The efficient on-board RGB-D SLAM of Scherer and Zell (2013) from the University of Tübigen, is able to run on small computers onboard of a micro aerial vehicle, separating the two tasks of the RGB-D SLAM algorithm into different threads. The first thread is the localization and tracking thread, that will receive the camera data and find an accurate estimate of the pose using a motion model, sparse optical flow and finally a Random Sample Consensus (RANSAC) from Fischler and Bolles (1981). The second thread is in charge of creating the map using the data produced by the first thread. It will create keyframes for the poses estimated in the other thread, refine the pose using a bundle adjustment, match it with more keyframes to create edges in a pose graph that later on will be optimized.

A similar approach to the one explained before is the Multi Resulution Surfel Map developed by Stückler and Behnke (2014) from the University of Bonn. This approach integrates and stores color and depth measurements in surfels inside a multi-resolution map representation. They register the motion of the camera using key frames and then they optimize it using the pose graph optimization. This approach will be explained in more detail in section 3.1.

After carefully considering several options, we decided to use MRSMAP as base for our approach, since the code is freely distributed with a BSD License, it uses graph pose optimization and it is a state-of-the-art technique with impressive results. It also uses the depth and color information in contrast to KinectfFusion. Although we decide to base our approach on this algorithm, it is possible to apply it to other SLAM algorithms.

2.2 Multi-Camera calibration

Transforming a monocular algorithm to a multi-camera solution can become a difficult task. As mentioned before, in section 1.1, some problems must be tackled while doing this transformation, but the most important one is the calibration between the different cameras.

Nowadays, it is possible to find several ways to realize the calibration of multiple cameras, depending not on just the method, but also on the parameters that must be calibrated. In most of the cases it is desirable to compute the extrinsic parameters, but in some cases it is also necessary to calculate the difference in time of the frames being captured between the different cameras, as can be seen in the work of Furgale et al. (2013).

2.2.1 Time Synchronization

Other researches try an approach that is more simple, i.e. Heng, Li, et al. (2013) used a master camera that triggers the other cameras at the same time, and packs all the images with the same timestamp. This can prove to be harder to achieve in RGB-D cameras, since they have two streams of data, one for the color image and one for the depth image, and both of them do not come synchronized. In this case, the idea of Heng, Li, et al. (2013) may be more difficult to do, since we will need to open the device and do some modifications, so both cameras can be trigger with the master camera. It is important to note that this may ruin the intrinsic calibration of the device, thus, giving wrong registrations between color and depth images.

In our approach we associate the depth and image data of each frame that are closer in time to each other and assume that the time in between it is depreciable; therefore, it would not affect the results. Sturm et al. (2012) also explains this time difference between streams and provides a script that does the same as we intend to do. Although, they provide the average time difference between the streams, it was done using a Kinect Camera, while we use Asus xtion cameras, and may have a different time delay.

Once the stream synchronization is solved, it is used to calculate the time difference between each of the cameras, or create a workaround as the aforementioned solution of Heng, Li, et al. (2013). Another solution could be the one described by Amplianitis et al. (2014), that uses different USB buses for each camera, calculates the average latency and then creates a timer of a value, greater than this latency. The downside of this approach, is that the frame rate will go down in one third, up to 19 frames per second as explained by Amplianitis et al. (2014). Since we consider this task not trivial and that it will consume too much time, we prefer to assume that the images can be in different times; therefore, they will not be consider in the same pose as the camera reference with the respective calibration transform, but they will be considered to be close to it.

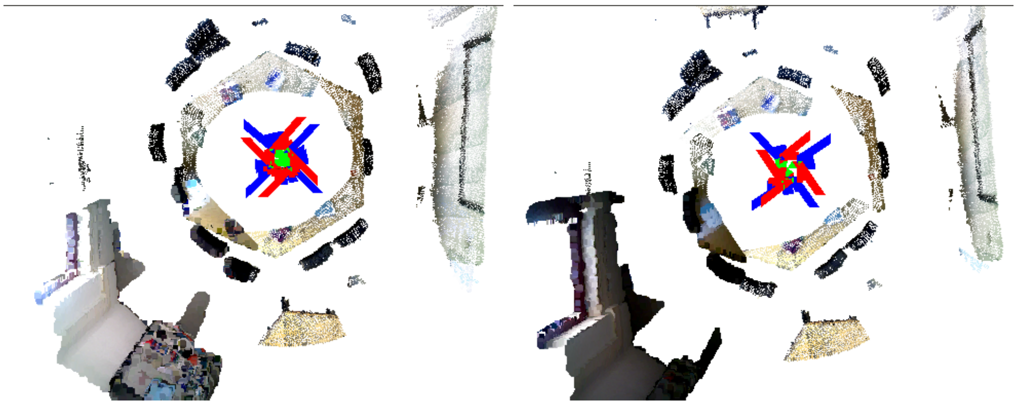
2.2.2 Extrinsic Parameters

As the time synchronization, this task should not be considered trivial, and there are many approaches to solve it, such as the works of Geiger et al. (2012), Heng, Li, et al. (2013), Fernandez-Moral et al. (2014) and Kuemmerle, Grisetti, Strasdat, et al. (2011). The general idea of this task, is to find the transform that describes the difference between one camera and a reference point. Depending on the approach, this reference point may vary, since some of them use one of the sensors, some use the centroid between the sensors, and some other projects may use an arbitrary point in the map, i.e. the base of the robot.

To understand further how important it is to calculate correctly the extrinsic parameters, we can observe figure 2.1, which shows our system, using four cameras. In the left side we see, that using the correct extrinsic parameters in a feature rich environment, the error without performing loop closure is small enough to be detected by the naked eye. In the right side, we can see, that a miscalculation in the extrinsic parameters may yield gaps big enough to deteriorate the system results, since it may not be possible to find a loop closure with such a big difference.

Geiger et al. (2012) algorithm uses checkerboard patterns, requiring human interaction to achieve the calibration. Another drawback is that the cameras must share the same field of view, that in some cases it is not possible, e.g. an omnidirectional camera set-up. The idea behind this algorithm, is to use several checkerboard patterns scattered in a room. The different cameras will take a single shot of the environment, making sure that all of the devices have an overlapping field of view. Then, the algorithm will detect the different checkerboard patterns, find the correspondences between images and finally recover the parameters via non-linear optimization.

The approach of Fernandez-Moral et al. (2014) needs the cameras to look at the same plane, e.g. the floor, and through this plane, find the correspondences between the different images and then the extrinsic parameters. This idea is quite



(a) Output with the correct extrinsic parameters **(b)** Output with wrong extrinsic parameters

Figure 2.1: Difference between good and bad extrinsic parameters.

useful, since it does not require any special pattern and can be applied to cameras that do not share the same field of view. Although the results of this algorithm are quite good and it has only one special requirement, with the approach used in this thesis it is possible to do the calibration while performing the RGB-D SLAM, thus saving time.

To find the calibration parameters, Heng, Li, et al. (2013) uses monocular visual odometry with sliding window bundle adjustment for each camera. They find feature point correspondences between the images of different cameras and the landmark features. To avoid drift, they also use loop closure. The obtained results are good, but as every algorithm it has its drawbacks. First, they use GPU to keep make it work in real time. Also, they only rely on the color information that is prone to give more error.

As explained before, in section 1.2, it is possible to determine the extrinsic parameters using graph pose optimization. Several algorithms already use graph optimization as a final step, and by adding some nodes and turning the edges into hyper-edges, it is possible to calculate this parameters without adding much computational effort, similar to the work of Kuemmerle, Grisetti, and Burgard (2011). The idea behind it will be explained in higher detail in section 3.2.

3 Fundamentals

3.1 Multi Resolution Surfel Maps

This algorithm was researched by Stückler and Behnke (2014) from Bonn University and as its name indicates, it is a RGB-D SLAM technique that uses multi-resolution strategies for registering the images and surfel maps to represent the RGB-D data. One of the most important characteristics is that it runs using only CPU, in contrast to other RGB-D SLAM algorithms as the one from Newcombe et al. (2011) that requires also the GPU to be able to achieve the result in real-time. Also, the use of GPU can limit the algorithms greatly, since the amount of memory available for such devices is usually limited and can not be expanded, and the quantity tends to be significantly smaller in comparison to the ones available for the CPU.

3.1.1 Surfel Maps

A surfel map is represented using octrees, where each node of the octree have one or more surfels, since the same object may be seen from different perspective, thus, the necessity of having more than one surfel per node. A surfel is a way of integrating depth and color images, saving in it the statistics of the color and spatial distributions of the points in the volume.

As stated before, the algorithm relies only on the CPU; therefore, it is needed to be efficient, and for that reason, a single pass update scheme for the statistics was implemented. To avoid numerically instability, a minimum sample of 10 points and no more than 10000^2 points is required to create the surfel and to stop updating it respectively.

Finally, it is important to note that the color space that is being use to model distribution of the color is a variant of the HSL color space defined as L$\alpha\beta$ as in equation 3.1.1. This way the chrominance (α, β) is separated from the luminance

(L).

$$
\begin{aligned}
&L := 1/2(max\{R,G,B\} + min\{R,G,B\}),\\
&\alpha := R - \frac{1}{2}(G+B),\\
&\beta := \frac{\sqrt{3}}{2}(G-B)
\end{aligned}
\tag{3.1}
$$

Also, a shape-texture descriptor is created for the neighbourhood of each surfel. To create this descriptor, the histogram of the three angles between the surfel and its 26 neighbors is taken in account.

3.1.2 Image Aggregation and registration

To insert a new image, the idea will be to update the statistics for each node in the octree that contains the point that is being process in their volume, and this will be done for every point in the image. This can be slow, but they create a faster way, that exploits a property of the camera, by knowing that neighbours in the image are likely to be from the same octree node and by changing the resolution of the tree to adapt to the noise; therefore the processing time can be lowered since it will need several thousands of insertions instead of the hundreds of thousands require in the naive way.

To do the registration of MRSMaps it is required to do two steps. First, it is necessary to find an association between the surfels from both maps, and once it is found, it is possible to determine the transformation between both maps that maximizes their matching likelihood.

For the first step, they iterate through each node associating each surfel, but if the children of the node have already been associated, then it prevents the parent to be associated to avoid unneeded associations or associations within coarse resolutions. The euclidean distance between shape and texture descriptors is used to accept or reject associations, depending on an arbitrary threshold.

Once the algorithm do the correct associations from the previous step, it is possible to determine the transformation using the proposed probabilistic model. This way, they find the most probable pose that maximizes the likelihood of observing the current frame in the target map.

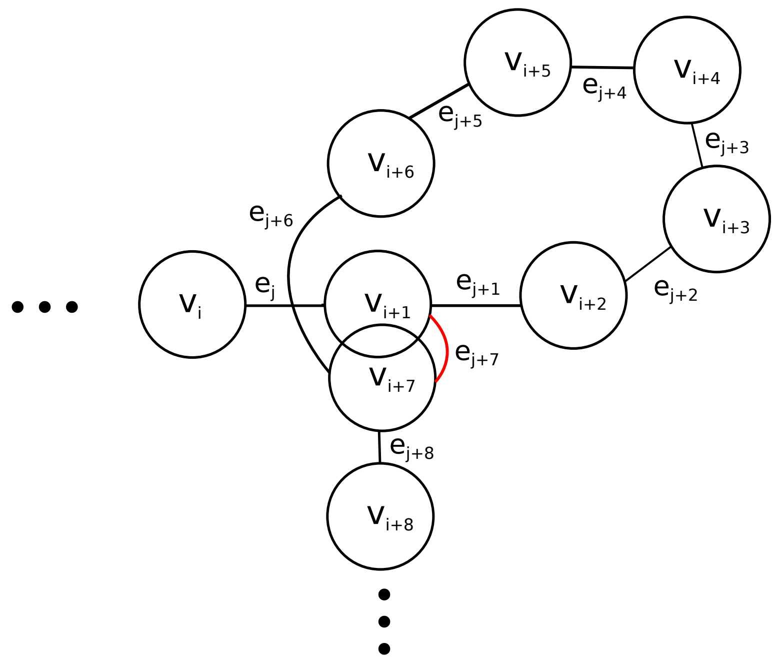


Figure 3.1: Example of the pose graph in MRSMaps. The red edges denote constraints added by similarity between nodes, and the black edges are added from the new key view to its reference key view.

3.1.3 Pose optimization

Even though the transformation obtained is the most probable one, it requires a optimization step in two stages to improve even further the results. First, their approach applied a fast approximate Levemberg Marquardt Optimization to the surfel asociations, followed by the second stage that is a fine registration using Newton's method directly in the logarithm of the observation likelihood described in the previous section.

The images are always registered towards a reference key view. If the camera moves far enough from this reference, a new key view is created and it becomes the new reference. Every time a new key view is created, a spatial constraint is also created between the new key view and the reference one. Then, the constraints will be added as an edge, and the key view as a node in the graph $G = (V, E)$ of key views. In figure 3.1, it is possible to observe an example of the pose graph created. In this case the set of nodes $V = [v_1, ..., v_i + 1, ...v_n]$ and the set of edges $E = [e_1, ..., e_i + 1, ...e_m]$.

It is important to note, that for each edge that is created and added to the graph, an estimation of the transformation uncertainty between the two nodes is required. It can be estimated using the observation covariance between the two maps of the images that the nodes represent. The covariance will show the uncertainty along unobservable dimensions.

To reduce even further the error, it is possible to add more spatial constraints. This constraints will come from key frames that are really close to each other, giving the possibility to do loop closure. It is important to note, that the new constraints must be created between two frames that overlap, or else the optimization may diverge. To make sure of this requirement, the constraint matching likelihood must pass a threshold to be added as an edge in the graph. This new constraint can be observed in figure 3.1 as the edges in color red.

Once the graph is formed, and all the possible edges and nodes are added, it can be optimized through a general graph optimization tool. In this case, MRSMap uses *g2o* from Kuemmerle, Grisetti, Strasdat, et al. (2011), and the main idea is to reduce the error by a sparse Cholesky decomposition. It is important to highlight, that the pose graph optimization stage, will take in account all the current poses, and possibly moving none, some or all of the current poses so far, giving a more accurate transformation for each frame.

3.2 Graph Optimization and Calibration

A good number of problems in computer science can be represented as a graph, and from this set of problems, we can find many that, to be solve, requires to find the minimum of a formula like equation 3.2, as it is described in the work of Kuemmerle, Grisetti, Strasdat, et al. (2011).

$$x^* = \underset{x}{\operatorname{argmin}} \sum_{\langle i,j\rangle \in C} e(x_i, x_j, z_{ij})^T \Omega_{ij} e(x_i, x_j, z_{ij}) \tag{3.2}$$

Where, x_i is a generic parameter, Ω represents the information matrix of a constraint relating two parameters x_i and x_j, and z is the mean matrix from the same constraint. To be more precise, in a graph, the x_i parameters represent the nodes of such graph and the edges will be the constraints between a pair of nodes, with the information and mean matrices to describe such edges.

3.2.1 Approaches

Although our approach its based on MRSMaps that uses the work of Kuemmerle, Grisetti, Strasdat, et al. (2011) for the graph optimization, this is not the only one to use graph optimization. We can also see the works of Scherer and Zell (2013) that uses in one of his last step, a pose graph created by his team, and then optimizing it using Hierarchical Optimization for Pose Graphs on Manifolds, also known as HOG-MAN, from Grisetti et al. (2010).

Other projects as the one from Olson et al. (2006), are designed to optimize a poor SLAM output. Even though the results are really good, the time required to process all the poses is small for offline optimization, but it will take too long to do it online. It is possible to take advantage of this algorithm, as a final optimization step to generate a better map of the environment.

Basing his work in g2o, Sünderhauf and Protzel (2012) introduces a innovative idea, switching constraints, that will make the optimization more robust to outliers, permitting the SLAM algorithm output to have some frames with a degree of error. The idea behind this new constraints, is to add a new variable that will enable or disable constraints. This will make the topology of the graph variable, permitting a better optimization. Although, it is possible to enable or disable a constraint, it is also possible to attenuate or intensify the constraints using this new variables.

3.2.2 Calibration with pose graph optimization

As explained before in section 2.2.2, nowadays exists several methods to do camera calibration, but only one of the aforementioned methods does it simultaneously with the graph pose optimization. Even though Kuemmerle, Grisetti, and Burgard (2011) uses only one sensor in his approach and he uses two dimensions instead of three, it is possible to extended to several sensors as it will be explained in chapter 5.

The idea behind Kuemmerle, Grisetti, and Burgard (2011) project is to find the extrinsic parameters of the sensor used with respect to the robot used in the experiments. To do such task, first, the graph must be changed to a hypergraph, and introduce a new calibration node. To explain better this notion, we can observe figure 3.2, that is a model of the hypergraph used for his approach. In this figure, it is possible to see, that the constraints represented by edges in the graph, will be changed to hyperedges; therefore, all the constraints will have a calibration node in between the two nodes. It is important to note, that this calibration node

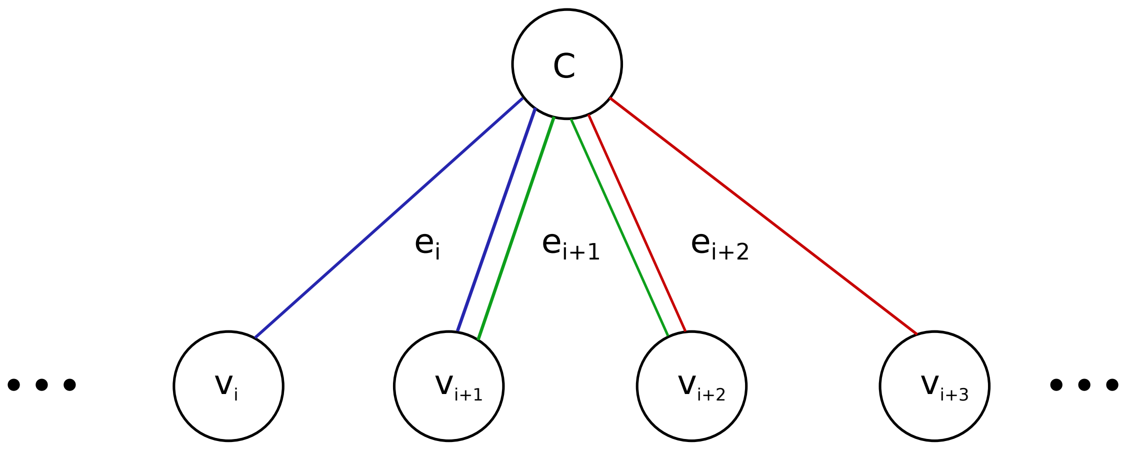


Figure 3.2: Example of the pose hypergraph in g2o. Each hyperedge have a different color, and all are connected via the calibration node.

will be the same for all the constraints; therefore, if the pose that represents the calibration node is changed, it will affect all the other nodes.

This approach did not use RGB-D sensors, but instead, they used a laser sensor and the odometry of the wheels in a forward kinematics model to obtain the estimated pose of the robot, hence the use of 2D vectors for the position of the sensor. Even so, it is possible to extrapolate this idea to the 3D geometry and also to a multi-camera system.

3.3 Undefined Registrations

In some cases, the RGB-D SLAM algorithm may encounter a match between two frames, that is almost perfect, but in reality, the camera have been translated much more than the estimated transform indicates. In this project, these cases are considered as undefined registrations.

Undefined registrations usually occur in place where one or more of the dimensions is not constrained, e.g. white walls with or without seeing the floor. In other words, it means a sequence of frames, that do not have any features in the color or depth image, or if they have goes along one of the dimensions, e.g. the corner between the wall and the floor. Since these frame are almost identical, the algorithm will perceive it as a identical match and the sensor position will not move until it finds a frame that is well defined.

It is of utmost importance to detect these undefined registrations, or the results

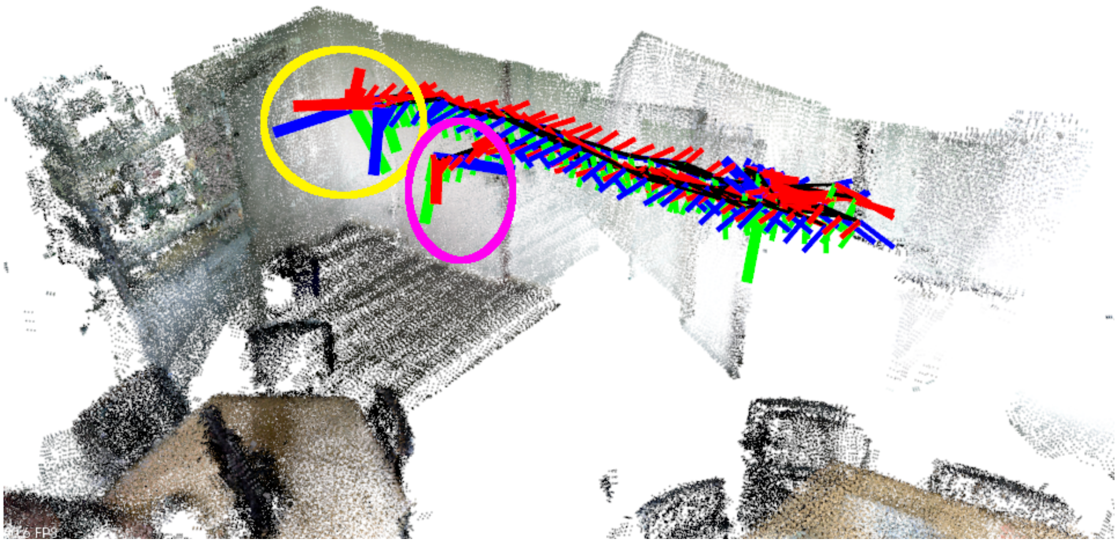


Figure 3.3: Undefined registration error in a four camera rig. One of the camera stayed behind (circled in magenta), while the other ones continued without problems (circled in yellow).

can be greatly deteriorated. For instance, if a camera is looking towards a wall and moves along it, for the algorithm it will be as if the camera was in the same position, giving as a result a shorter wall in a map representation and also, it the error may increase up until a point, where the loop closure will not be possible. In figure 3.3, it is possible to observe this example in a four camera system, in which one of the cameras stayed behind, since the algorithm compute the pose and gave unknowingly an undefined registration. Also, this figures shows, that even if the rest of the cameras do not capture an undefined registration, the whole result will be ruined by the camera that did.

Stückler and Behnke (2014) already gives us an indirect hint on how to detect this issue. They estimate for each constraint the uncertainty along each dimension with the covariance of the observation. Using this covariance it is possible to detect an undefined registration. The method to do so, will be explained in detail in section 5.3.

4 Methodology

Our approach will extend the ideas of Stückler and Behnke (2014) and Kuemmerle, Grisetti, and Burgard (2011) to use multiple cameras and to work simultaneously. To achieve this, the solution can be separated in three subtasks, i.e. extending MRSMaps to work with multiple cameras, integrate the simultaneous calibration to the system and detect and process undefined registrations correctly. The first task will permit the system to work with more than one camera, but it will create the need of the second task, the calibration. With the calibration, the correct offset is generated and the system will be fully functional, but it will permit improvements that were not possible with a monocular approach, such as the third task mentioned before.

4.1 Multi camera MRSMaps

To be able to use multiple cameras with the MRSMap algorithm of Stückler and Behnke (2014), we utilize a divide and conquer approach. Each of the camera will have their own reference key frame and will register the new images with the ones of the same camera, creating for each of them a small piece of the map, as show in figure 4.1. After the registration, the frame will merge with the global MRSMap. This way it is possible to use the frames of any camera to obtain a more accurate registration. If the new frame is consider to be far enough of the previous reference frame, it will create a key frame and a node in the graph. This node will be connected with a constraint to the reference key frame of that camera. To unite the different parts of the map, constraints can be created between frames that are close to each other, independently of the camera used to capture it. This will generate the same effect as a loop closure in the single camera MRSMap, making both sections of the map to merge and create a complete map.

Having constraints and nodes of different cameras interconnected, will permit the algorithm to optimize the poses of the cameras all together. This will generate a coherent optimization and consequently a more accurate map of the environment.

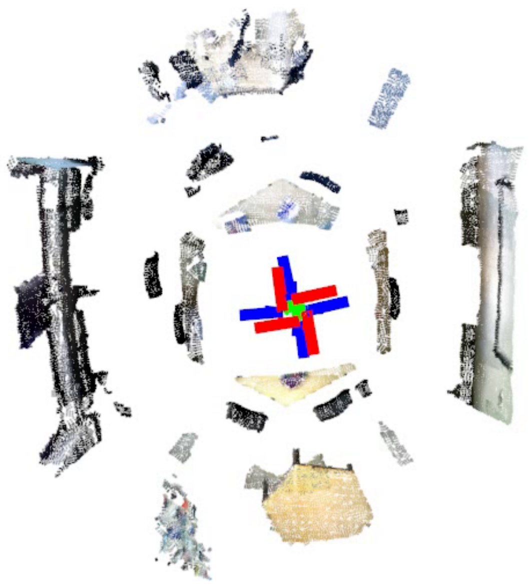


Figure 4.1: Example of a four camera system integrated with MRSMaps, showing how the map is divided in four sections, one for each camera

Even though creating the constraints between different cameras sounds simple, it requires an extension to the graph and edges, that will be explained in higher detail in the next section.

4.2 Calibration and Simultaneous Localization and Mapping

Until now, most of the RGB-D SLAM algorithms that have a pose graph optimization phase will seek to minimize the error in the graph. Having a set of nodes representing the poses $X = \{x_1, ..., x_n\}$ and a set of measurements z_i^j that will express the difference between the pose i and j, it is possible to calculate the error of the whole pose graph as shown in equation 4.1 and 4.2. In here, the e represents the error of each individual and Ω_{ij} represents the information matrix of the constraint, in the case of MRSMap, it is also the covariance of the registration.

$$E = \sum_{i,j \subseteq C} e(x_i, x_j, z_{i,j})^T \Omega_{ij} e(x_i, x_j, z_{i,j}) \tag{4.1}$$

$$e(x_i, x_j, z_{i,j}) = z_{i,j}^{-1} \cdot X_i^{-1} \cdot X_j \tag{4.2}$$

This error will measure how good the poses i and j fit the measurement $z_{i,j}$. In the best case scenario, it will be a perfect fit and the formula will return 0. The idea of the pose graph optimization is to obtain the smallest error possible and which can be achieved by finding the poses x that best fit the measurement value. The uncertainty that is recorded in the constraint will determine how big or small the changes of the original pose x_i and x_j can be.

Using a similar approach to the one done by Kuemmerle, Grisetti, and Burgard (2011), the algorithm will use a formula derived from the pose graph error equation 4.1 and 4.2. First, it is necessary to extend it to cope with the new node that represents the sensor's offset with respect to a reference point. Since the new graph will require a new element involved in the calculation of the error of each constraint, the edge need to be extended and converted to an hyper-edge. An example of this hyper-graph can be found in figure 3.2 in section 3.2.2. As it is expected, this will generate a change in the formula used to calculate the error in a pose graph, as it is shown in equation 4.3.

$$e(x_i, x_j, C, z_{i,j}) = z_{i,j}^{-1} \cdot (X_i \cdot C)^{-1} \cdot (X_j \cdot C) \tag{4.3}$$

Although these changes will make possible the calculation of the extrinsic parameter of a sensor, in our approach more devices are involved. Since every node representing a frame may have a constraint with another node, it is necessary to change the edges and the graph once again. In a constraint between frames captured by different cameras, two nodes from the sensors will be involve in addition to the initial nodes representing the frames. As before, the formula to calculate the error will change again as it is shown in equation 4.4.

$$e(x_i, x_j, C_i, C_j, z_{i,j}) = z_{i,j}^{-1} \cdot (X_i \cdot C_i)^{-1} \cdot (X_j \cdot C_j) \tag{4.4}$$

If the constraint is between two nodes of the same camera, even though it is possible to use the error calculation in equation 4.3, the equation 4.4 was used. Also, it is important to note that in the graphs that involves the nodes representing the sensors, only one node per sensor is being used. In other words, all the constraints created will use the calibration nodes; therefore, all the poses will influence the pose of the sensors. To understand this approach even further, it is possible to observe the figure 4.2 , that contains an exemplar of the graph described. In

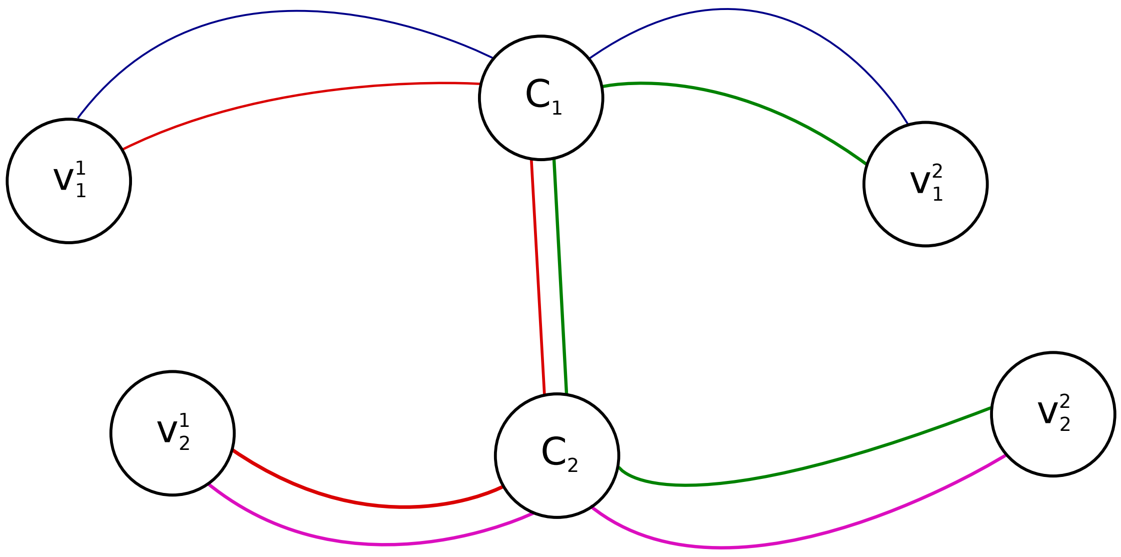


Figure 4.2: Example of the multi-cam pose hypergraph in our project. Each hyper-edge have a different color.

here, the subscript denotes the camera number, the superscript the time-step, V_i^j is the vertex representing a frame from camera i in time-step j and the sensor are represented by the nodes C_i. Also, to be able to follow the hyper-edges, each of them was drawn using a different color.

4.3 Undefined Registration Detector

First, it is important to understand the meaning of an undefined registration to be able to detect it. In some cases, the MRSMap algorithm, as well as other RGB-D SLAM algorithms, will register a new frame to any of the previous frames incorrectly, but giving a high likelihood match score. This means that most of the surfels associated will give a high probability of being the correct match, even though it is not from the same place. In most of the cases, this is due to the fact that both frames are either identical, or they do not have any feature in their color and depth image, for the algorithm to be able to separate them correctly. As an example, a white wall segment may give an exact match in depth and color information when it is compare with another segment of the same wall, since it is very similar.

This errors are not invisible to the RGB-D SLAM algorithms, but it is hard to avoid them or to fix them, since there is no more information to correct the camera pose. Using a pose graph optimization may help if it is a small amount of

	Biggest Eigen Value
Mean	2.9202e-09
Standard Deviation	3.2062e-09

Table 4.1: Mean and standard deviation of the eigenvalues.

frames that generate such an error, and if other frames create a constraint with this problematic ones. If several frames of this kind are captured and processed, then it is not possible with one sensor to avoid or correct this error. Nonetheless, the problem can be fixed, or at least estimate with a higher precision the correct pose of the sensor, if several cameras are involved. Once it is detected, instead of using the uncertain pose to register future frames, an estimated pose that comes from the extrinsic parameters of the cameras will be used.

As mentioned before, this error is not completely invisible to the algorithms, for instance, the MRSMap algorithm already have a method to model the uncertainty of each dimension of each registration using a covariance matrix that can be use as a base for the undefined registration detector. This matrix models the error for the 6 dimensions used in the algorithm, i.e. x, y, z spatial dimensions and three of the four the quaternion's components that represent the angle. In our approach, only the error from the x, y, z dimensions is used to simplify the calculations and reduce time. If the values of this matrix are used without any process, it is still possible to determine if one of the dimensions have a big uncertainty and thus giving a wrong registration, but the wanted value could be produced in any direction, inclusive one that combines more than one of the chosen dimensions. As a consequence, to determine the dimensions in which the uncertainty in bigger, it is necessary to do the eigen-decomposition of the covariance matrix.

The resulting eigenvectors will describe in which direction the values of the covariance matrix are bigger and the eigenvalues the magnitude of such vectors. To determine if a registration is undefined or not, it is necessary to examine the eigenvalues. If the biggest value falls outside of two thresholds, then it may be consider an undefined registration. To calculate such thresholds, our approach will assume a normal distribution of the biggest eigenvalue and through a statistical analysis it is possible to find out the mean and standard deviation of such distribution, as shown in table 4.1. The data from the table comes from a feature rich sequence of frames of one of the datasets used in the experiments. Also, this information was compared with the one obtained from featureless frames, and from images with only one distinguishable feature, e.g. a column in a white wall.

It is well known, that in statistics, 99.8% of the values under the curve can be found within three times the standard deviation in a normal distribution. Due to this reason, we decided to use as threshold the interval of three times the standard deviations and centered it in the mean, giving a lower probability of classifying incorrectly the registration. If too many frames are incorrectly classified, it may lead to unpredictable behaviour in the optimization phase.

4.3.1 Marking

Once an undefined registration is detected, it is necessary to make this nodes and the constraints associated to it differently. One of the ways thought by us, was to isolate the node in the graph, and fix its position so it won't be taken in account during the optimization. Even though, it proved itself useful in some experiments, it may create a rupture in the graph and it will make the isolated pieces of the graph to drift apart and diverge, creating unusable maps.

Instead, we decided to add the edge from the new key frame to the previous reference key frame, in spite of the undefined registration. To be able to avoid the effect of the undefined registration in the optimization phase and in the registration of new images, we softened the constraints that are generated with the previous frame. This way, during the optimization phase, this constraint will have enough force to keep the frame in the correct position and not to generate more drift, but at the same time the edge will not force the previous frames to move towards it. This is done by modifying the covariance information of the frame.

Also, the pose estimate given by the undefined registration is not used, but instead, the estimated pose of the calibration parameters with respect to the reference camera is used. This way, the position of the new frame will not be affected by featureless sections, e.g. white walls, and the future frames will have a more accurate starting position to estimate their transformation.

4.3.2 Unmarking

In some cases, it is possible to find a good registration for a frame that has been previously marked as undefined. This could happen in the case that a new frame overlaps part of the undefined section, and the rest of the image is feature rich, which allows the system to find a new good registration with the marked node, hence the need to remove the mark of the previously undefined frame, since it is already integrated correctly in the map.

Once the mark is removed, it is necessary to permit it to change with the pose graph optimization; therefore, the edge with the modified covariance is recalculated and the real covariance is used this time. Due to this, it will generate a bigger influence in the graph optimization and act as if it was never marked. Also, it will be possible to add constraints between this node and other nodes in the graph without the need to do another undefined registration check, since it is already associated correctly with another frame.

It is important to note that not all of the nodes that are marked will be unmarked; therefore, they will continue to have an estimated pose using the calibration nodes and the last reference camera pose in the moment of its creation. Since this poses are dependant of all the frames that shares them, it is safe to assume that the estimated pose, will be close to the correct pose using this method. This will circumvent an important problem that affects most of the RGB-D SLAM systems.

5 Implementation

As mentioned in the previous chapter, to implement the calibration and simultaneous localization and mapping with multiple cameras, the solution was divided in three tasks. First, the program should be able to work correctly in a multi-camera environment. For this it is required to modify the selected RGB-D SLAM algorithm, i.e. MRSMap (Stückler and Behnke, 2014), to work with more than one device, as explained in section 5.2. Since it has an open source code, we use the latest version available as a base for the implementation. Once it is able to cope with multiple sensors, it is necessary to be able to determine, in a autonomous way, the extrinsic parameters of each of the sensors with respect to a reference point. After obtaining such parameters, the system should also be able to use them either as initial guess to fine-tune the values by doing the calibration process once more, or to use it in a exploration task in which these parameters will be fixed and no change will be possible.

To obtain a better result in difficult cases in which a camera can not capture a feature-rich scene, it was necessary to implement an undefined registration detector. This detector will find frames that have a high uncertainty in one or more dimensions, but which have a good match with another frame. This will contribute greatly by avoiding errors in the pose estimation, and later on, in the creation of the map.

5.1 Capturing tool

Prior to the implementation of the proposed system, it is necessary to create a system to capture the data from the eight cameras simultaneously and then it should either save the data to a hard disk or transfer the information directly to the algorithm for it to be processed. In this case, the first option was used to be able to reproduce the experiments with the same data. This tool was created using the OpenNI as the camera drivers and the Robot Operating System (ROS) to create and manage the threads through nodelets.

To save the data without loosing any frame, each camera was controlled by a different thread. Another thread was in charge of saving the data to a Solid State Drive, which has a faster writing rate. As mentioned before, the color and depth images do not have an explicit association to each other, but to be able to find the best possible pair, the timestamp of each of the images is saved. When the data is needed again, to find the best match, the algorithm will find the depth image that is the closest to each of the color ones. Using this method, it is possible to repeat a depth image without affecting the result, but it will obtain a more accurate association.

5.2 Multi-camera RGB-D SLAM

Before starting the implementation it is necessary to choose the RGB-D SLAM algorithm to be used as base for our approach. It is important that the method is efficient, without any specific hardware constraint and with results competitive with the current state-of-the-art techniques. Also, to cope with the time limitations of this project, it is preferable if the algorithm is publicly available as an open source project, to be able to modify the code and to add our changes. Finally, to keep the efficiency of the system, it is important that it already has a pose graph optimization phase. Having those conditions in mind, MRSMAp from Stückler and Behnke (2014) was selected, since it met the requirements and also uses an open source graph optimization library that is possible to modify, and it contains several functions and data-types that makes it feasible to ingrate without problems our approach.

Even though MRSMap was selected as our RGB-D SLAM algorithm, it is possible to use this approach with any other option, but if it doesn't have an optimization phase using pose graph optimization, it will incur greatly in the processing time.

5.2.1 Integration with MRSMaps

One of the most important tasks, and the first one that needs to be tackled, is the one of integrating the MRSMaps algorithm with our multi-camera approach. To achieve this, it was necessary to extend first the reference key notion explained in section 3.1.2; therefore, instead of having one reference key frame in which all the subsequent frames will register too, we use one reference key frame for each camera that is being used, and only the frames of that camera will register to it.

This way, the map will be divided in sections, one for each camera. Once the sensors start moving, the pieces will act as separate instances of the MRSMap algorithm until they reach the section of another camera and perform a loop closure, thus, unifying the map. In other words, it will create a new constraint between the cameras that overlap each other, tying both sections in the graph with an edge, and optimizing all the poses at the same time. It is important to note, that due to the field of view of the camera, a multi-camera system with eight cameras will cover the area almost completely, and with a small rotation to the robot, it will perform the loop closure in all the sections, in consequence, it will create a complete map of the environment.

To complete this task, the key frames must be modified in order to be able to store the information of the camera that captures the frame. This is important, since it is needed to find the correct reference camera and also, to be able to create the correct constraint. The creation of the constraints will be explained in full detail in section 5.2.2.

5.2.2 Extrinsic Parameters

Once the integration of the multi-cam system with the algorithm of the MRSMaps starts, it is imperative to know the extrinsic parameters of the cameras, since the integrated system will need to know the offset of the cameras to do a correct registration of the images. It is possible to obtain such parameters with any other method and use it directly as an offset for each frame, but due to normal wear and tear of the hardware, this parameters can deviate and give an incorrect result.

With our approach it is possible to accurately obtain the extrinsic parameters of the camera at the same time it creates a complete 3D map of the environment, without the need of an specific set-up. To integrate the acquisition of this parameters with the MRSMaps algorithm, we extend the hyper-graph explained in section 3.2.2 from Kuemmerle, Grisetti, Strasdat, et al. (2011), and substitute the graph used in MRSMaps with this extension.

5.2.3 Calibration nodes and edges

To extend calibration hyper-graph, it is necessary to adapt it first as to work in a multi-camera environment. As explained in section 5.2.1, each camera will create new nodes, however, the offset needed in the previous stage will be taken in account using hyper-edges between the nodes, that pass through the calibration nodes. Similar to the ones used by Kuemmerle, Grisetti, and Burgard (2011) in their project, but with a slight difference. Since our project uses multiple cameras, and we may require connections between one node from one camera to another, we implement a new type of hyper-edge that will go from one node to another passing through the calibration node of the first camera and then through the calibration node of the second camera. This is necessary since the two offsets of the two cameras are required.

To implement this graph we used the latest version of g2o from Kuemmerle, Grisetti, Strasdat, et al. (2011). Since this is a novel idea, a new type of edge is created, extending the current hyper-edges of the author and adding the correct calculation of the error to it. Even though g2o do not have an implementation of the monocular calibration system of Kuemmerle, Grisetti, and Burgard (2011) in 3D, it has the necessary 3D vertex for the hyper-graph; therefore there was no need to modify the current nodes.

This approach is capable of handing any number of cameras, although it was only tested with four and eight cameras. To use a different number of cameras, it is only required to change the number of calibration nodes, one for each sensor. Also, it is important to give an approximate guess of the position of the cameras. It is not necessary to give a very accurate one, usually the one specified by the manufacturer of the sensor structure is enough.

5.3 Undefined registration detector

In the majority of the RGB-D SLAM algorithms, if the camera passes near a featureless place, i.e. white wall, the algorithm looses track of the camera and generates a map with errors. This problem may happen due to the algorithm registering an frame that shows one section of the featureless place to another frame that shows another section of the same place, but both sections are so similar that are registered as if it was in the same place. Since it gives a good match, most of the algorithms do not detect this error.

Even though it may be possible to detect such undefined registrations, in a single camera infrastructure, it may prove very difficult to avoid and to continue tracking of the camera correctly. When using multiple cameras to perform the RGB-D SLAM, it is possible to avoid the frames that can be considered undefined and calculate through the other cameras a probable position of the camera with the undefined registration for the frames that can not be registered correctly.

5.3.1 Detecting

The most important part of this task is the detection of the undefined registrations. We consider the use of the covariance of the registration of the frame that models the uncertainty of the constraint in each dimension. Since this covariance is already calculated during the creation of the new key frame, it is possible to use it again without incurring in additional computation time.

The covariance given from the MRSMaps algorithm is a six by six matrix, that models the six dimensions that are measured. Our approach will use only a section of this matrix, the top-left three by three block, that represents the covariance in the planes x, y and z. This way, it is possible to reduce the computations needed afterwards, having enough information of the registration to classify it correctly.

Our approach will obtain the eigenvalues of the aforementioned covariance matrix section via eigendecomposition with the library *Eigen*. Then, the biggest eigenvalue is tested to see if it fits in a interval, if not, it will be considered undefined. To keep track of the key frames with undefined registrations, the structure is modified to save such information and other important details, such as the information needed to move the node after the optimization phase. This mark will enforce a check when another frame tries to associate to the one with the undefined registration. This way, it is possible to prevent the creation of future undefined registrations against this frame.

A second mark is associated to the constraint from this key frame to the reference key frame. This is necessary since the edge created will have a modified covariance. Later one, if this node manages to get a good registration with another node, the marks will be removed, and the covariance of the edge will be recalculated.

6 Experiments

The experiments done for this project demonstrate the capabilities of the system and the improvements in performance against a monocular system. It is important to remember that until the submission of this thesis, the author is not aware of any other method that performs simultaneous calibration, localization and mapping with multiple RGB-D cameras; therefore, it may not be compared to other state-of-the-art methods that do such task. Even so, it is possible to compare other aspects such as execution time with different amount of cameras, calibration parameters between different datasets with the same set-up and the entropy of the resulting map sharpness.

To be able to reproduce the experiments, it is necessary to divide them in two stages. During the first stage, the program will only capture the datasets and save them directly into the disc, permitting a frame rate of 30 frames per second (fps). Once the complete dataset is recorded, it is possible to load it and use them as input for the system. This way it is also possible to control the number of cameras being used with the same data.

6.1 Hardware

To carry out the experiments, it is indispensable to use a RGB-D cameras to capture the datasets. All the cameras used in this project are Asus Xtion [1], that is a lightweight RGB-D sensor, smaller and more accurate than the Kinect. With eight devices such as this one it is possible to cover almost completely the 360° with a set-up such as the one in figure 6.1, since each of the camera have a vertical field of view of 45°. Once again, having more cameras may give some difficulties to record the datasets keeping the real time frame rate of 30 fps. This problem can be avoided either by saving the data directly into a Solid State Drive as Portable Network Graphics (PNG) images or saving it into the memory and later on to the hard disk. For this project, it was decided to use the first option, since some

[1]www.asus.com/Multimedia/Xtion/

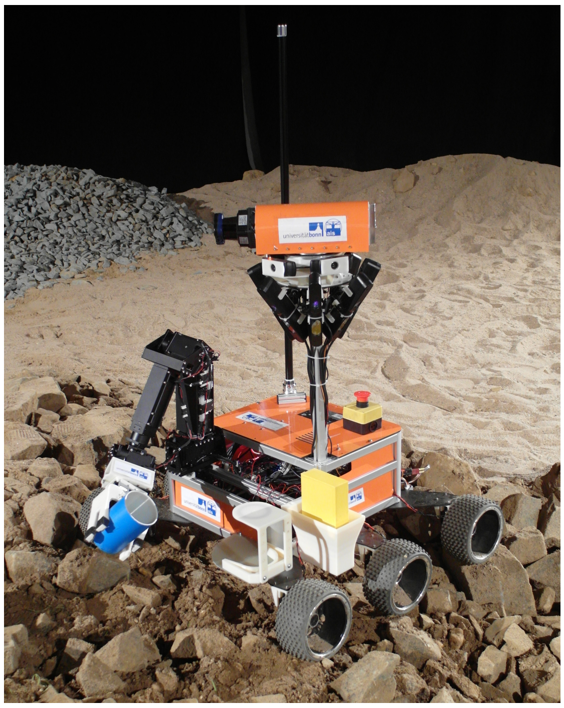


Figure 6.1: Explorer robot with eight RGB-D sensors used for the experiments.

datasets may consume big quantities of memory and making the system unstable in the process, before being able to save them to the hard disk.

The hardware used to perform and evaluate the proposed experiments of this projects is the explorer robot from the Autonomous Intelligent Systems department of the University of Bonn, shown in figure 6.1. This machine have an *Intel Core i7-4770K* processor with a speed of 3.50*GHz* and 32 GB of Random-Access Memory (RAM). As mentioned before, it also has Solid State Drive and eight Asus Xtion cameras connected to a USB card with different BUS each to avoid a data bottleneck.

6.2 Proposed Experiments

To test and evaluate the new features and improvements of the system, it is necessary to do different type of tests in different environments. Furthermore, the generated maps should be evaluated or compared with ground truth. Since it is not feasible to generate ground truth in different environments with the available tools, the entropy of the map is calculated using the tool implemented by Droeschel et al. (2014). Later on, the extrinsic parameter obtained in different environments

Exp. Number	Room	Type	Observation
1	Robotics Lab	Calibration	Rotational movement
2	Robotics Lab	Walking	—
3	Computer Graphics Lab	Calibration	Rotational movement
4	Computer Graphics Lab	Undefined Reg.	Rotational movement
5	Computer Graphics Lab	Undefined Reg.	Translating along the wall
6	Computer Graphics Lab	Walking	—
7	Horsaal 3 (center)	Undefine Reg.	Rotational movement
8	Horsaal 3 (wall)	Undefine Reg.	—
9	Conference Room	Undefine Reg.	Rotational movement
10	Conference Room	Undefine Reg.	Translating along the wall

Table 6.1: The proposed experiments rooms and types.

will be compared to demonstrate the robustness of the project.

To show the improvement of the system in time consumption, the time used and the number of frames needed to obtain a complete map with different amount of cameras will be compared. To determine when a map is completed, the system will detect when a node representing a starting frame of a camera, has a constraint with the newest frame of another camera or in the case of only having one sensor, the newest frame of its own. This experiment must be done with the *Calibration Dataset* explained in detail in the section 6.2.1.

Finally, the undefined registration detector has to be tested, and to do so, the data used must contain a featureless section. Once tested, the generated map will be examined and analysed for the qualitative results and later on, it will be evaluated using the aforementioned tool to calculate the map sharpness entropy.

6.2.1 Datasets

Different types of experiments require different types of datasets. In this project three classes of datasets are used. The first one will be the ***calibration dataset***. To perform this capture, the robot will rotate on its axis in a feature rich environment, until it completes with each camera a 360° scan of the surroundings. It is important to note that the movement of the robot is done by a person using a cart; therefore, the movement will not only be rotational, but it will also have a slight translation movement that will not influence the results of the experiments.

The second type of dataset, called *undefined dataset*, will be used to test the undefined registration detector implemented in this project, thus the need for at least one camera to pass through an object that will trigger the detector, i.e. a white wall. After passing through this object, it should capture another scenes with more features, giving it the chance to register some of the marked frames to fully test the capabilities of the system to detect, avoid and recover from this problematic scenes.

Finally, to demonstrate the performance of the system in a real case scenario, the *walking dataset* was created. In here, the robot will capture the environment of a large room, including objects, walls, tables, and other items found in a room. Also, the robot will move through the room in a undefined way, but trying to return to the original position at the end of the dataset. This way, the system will have the opportunity to create a loop closure.

6.2.2 Environments

To be able to test the project completely, it is important to use different environments and observe how the algorithm behaves within these different conditions. The datasets used in the experiments were taken in four rooms of the *Landesbehördenhaus* building from the University of Bonn. The first room is *Hörsaal 3*, which is a lecture room that contains tables, chairs and the most important characteristic, i.e. a white wall without any feature. In this environment, it is possible to test the undefined registration detector, since the white wall segment and also the identical tables and chairs generate undefined registrations that must be avoided.

The next two rooms are used for the calibration datasets; therefore, they must be feature rich environments. The *Robotics Lab* and the *Computer Graphics Lab* have several tables, chairs, and objects such as monitors, books, and computers, giving the possibility to find features in the depth and color images. These two cases can be considered the best scenarios for the robot to determine accurately the calibration parameters.

To enforce a white wall segment in the *Computer Graphics Lab*, the blackboards attached to the wall were removed temporarily. This way, the undefined registration detector can be tested in a different environment, in which the other cameras should capture a feature rich scene. Furthermore, the detector will be tested in a third environment differently from the rest. The *Conference Room* has a unique feature that is not possible to capture inside the other rooms. It posses a white wall segment, that has also a white column in the middle of it, giving an undefined registration in half of the cases, since it may give depth features in some of the frames.

7 Results

Once the aforementioned experiments are completed, the algorithm will output a file with the poses of each point cloud used, and also, the image files involved in the creation of those point clouds. This way, it is possible to do a comparison of the poses with another algorithm or parameters. Also, it is possible to create the 3D map with this poses, either overlapping the point clouds and creating one global point cloud, or by using any meshing tool such as *CPU_TSDF* of Miller (2015) or *FastFusion* of Steinbruecker et al. (2014). After using both methods, we decide to continue using FastFusion, since it creates smoother and complete surfaces in shorter time. However, the other method helped to reduce the noise that comes from the point clouds.

Once the 3D map of the environment is done, it is possible to evaluate the results. At first, a qualitative analysis is performed on the results to confirm visually that the map corresponds to the scene and that the algorithm fulfilled its purpose. To go even further in the qualitative evaluation, straight lines are overlapped over the mesh to help the user to visualize and compare the correctness of the objects and walls in the map. Also, each intersecting line forms a 90° angle, to compare objects such as tables that must have such angles. Finally, some measurements are also done to the mesh, and compared with real life measurements, to test the accuracy of the maps.

Once the quality of the map have been assessed, it is important to confirm these results with the necessary quantitative evaluation. Using the tool created by Droeschel et al. (2014), it is possible to quantify the entropy of the sharpness of the map. This tool is of great utility for this project, since it gives the possibility to measure and compare the results for datasets without the proper ground truth. Since this project introduces the novelty of using multiple RGB-D cameras to perform SLAM, there is no public dataset with a ground truth that can be used; therefore, we created new datasets, but without ground truth.

For the calibration datasets it is important to measure not only the entropy of the map, but also the correctness of the extrinsic parameters calculated. To perform this evaluation, we compare the difference between each of the parameters

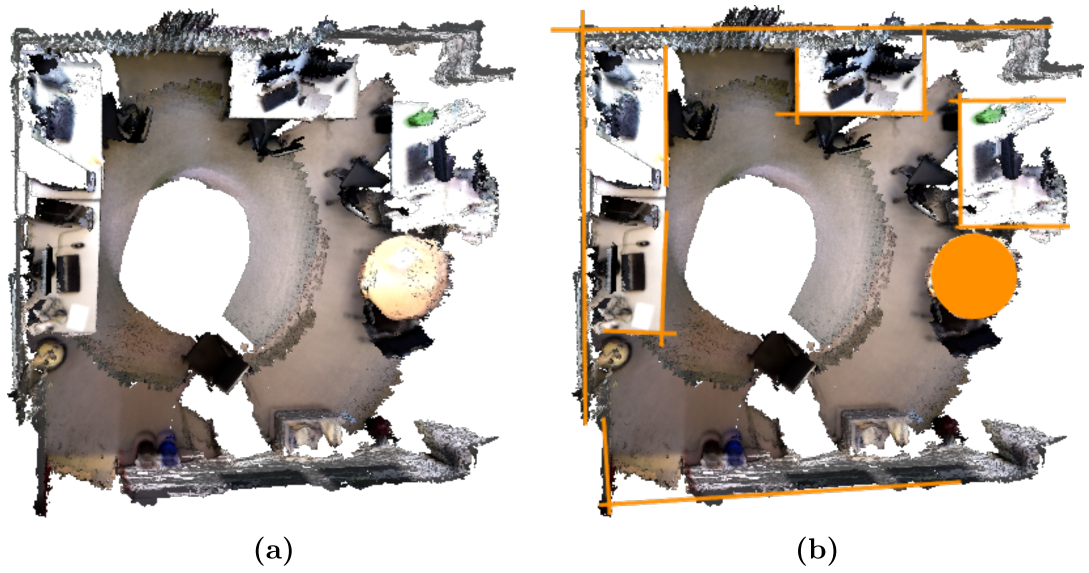


Figure 7.1: CSLAM with the Graphics Lab dataset. The orange objects in 7.1b, show the correctness of the map.

obtained and calculate the sum squared error of it. It is important to note, that the difference is the matrix difference, and then, the identity matrix is subtracted and finally the squared values of each of the matrix's cells is summed together and averaged to obtain the mean square error of the transform.

7.1 Qualitative Results

7.1.1 Calibration Datasets

To show the qualitative results, the mesh representation of the 3D map is being used. For visual aid, straight lines have been drawn over the walls of the results, to demonstrate the accuracy of the basic structures of the objects inside the map. As it can be seen in figure 7.1, the lines completely overlap the walls without any indication of them going in a different direction; therefore, the generated map have straight walls.

Straight walls is just one of the qualifications we look forward to when creating a map. Another very important feature that the map must have, is the correct angles between these straight lines. As we mentioned before, the created lines have 90° angles in each of the intersections. Again, in figure 7.1 we observe that several angles measured in the images are indeed right angles, and consequently proving

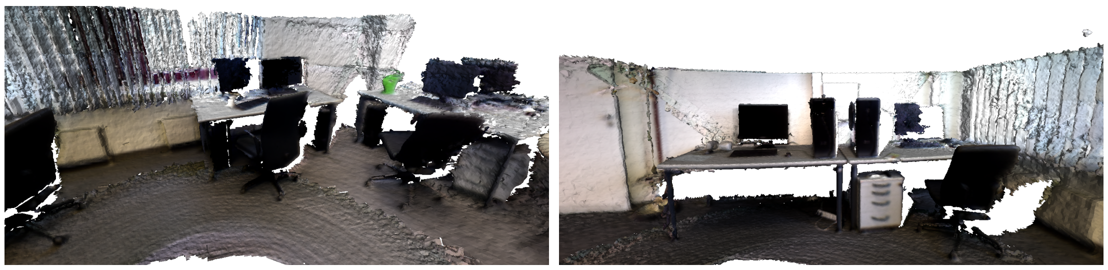


Figure 7.2: Mesh of the resulting map of the CSLAM with the Graphics Lab dataset.

Object	Measurement Real	Measurement Mesh
Round Table	1 meter	1.03 meter
Door width	1 meter	0.97 meter
Rect. Table width	1.60 meter	1.59 meter
Rect. Table depth	0.70 meter	0.71 meter
Black board	1 meter	1.02 meter

Table 7.1: Object measurements for the graphics laboratory dataset

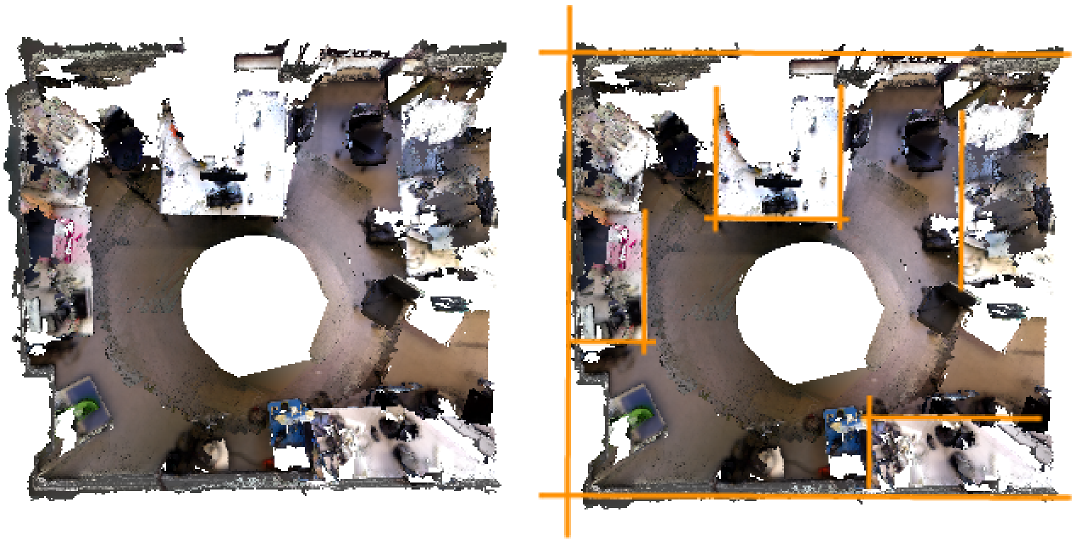
that the objects with perpendicular lines are correctly represented in the map.

In the figure 7.1, we can also observed a round table that has the same diameter in every arc passing through the center of it. Once again, we can show the map's accuracy, overlapping a circle on top of it, and showing that it fits without problem. Other objects, with non-geometric form can also be distinguished and easily classified by any person, such as the ones shown in figure 7.2.

Finally, we decide to do some measurements, for instance the diameter of the table is 1 meter long, and as we can observe in table 7.1, the value in real life is also 1 meter. In the same table, we can observe other different objects, with a similar result. Showing the correct measurements demonstrates that the objects have the correct scale, thus, the accuracy of the map can be consider good.

In figure 7.3 we observe the second calibration dataset, the one taken in the robotics laboratory. As before, we can observe that the generated map have straight walls and the rectangular objects as well. Also, the right angles can be seen via the intersections of this lines that were drawn on top of the mesh. Once again, it is possible to observe the measurements taken in table 7.2, to observe the accuracy of the representation of the objects in the map.

It is important to remember that both of this datasets were captured using a



(a) (b)

Figure 7.3: CSLAM with the Robotics Lab dataset. The orange objects in 7.3b, show the correctness of the map.

Object	Measurement Real	Measurement Mesh
Door width	1 meter	1.01 meter
Rect. Table width	1.60 meter	1.56 meter
Rect. Table depth	0.70 meter	0.73 meter
White board	1.50 meter	1.51 meter

Table 7.2: Object measurements for the robotics laboratory dataset

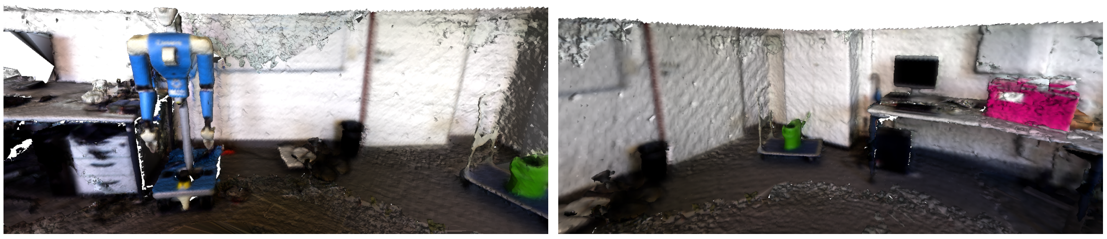


Figure 7.4: Mesh of the resulting map of the CSLAM with the Robotics Lab dataset.

rotational movement of the robot on top of the cart to obtain a good view of the environment; therefore, some parts of the map will be missing, since the robot can not see such places during the capturing of the data. The most important one is the one occluded by the cart in the middle of the room with a circular form. Since the cart is bigger than the robot, the hole in the floor looks like an oval. Also, places behind the chairs, monitors, and other objects will not have a mesh representation, since those surfaces are not visible during the whole dataset. Even though, this missing parts do not harm the results as we have seen already in the figures and tables.

7.1.2 Walking Datasets

As with the calibration datasets, the mesh representation of the algorithm was calculated, although it did not show the expected results. Having a closer observation to the SLAM algorithm visualization, we notice that the main problem is that the dataset contains several long segments, in which the camera number 4 and in some cases the camera 0 will not observe any features other than the floor. Even though we created the undefined registration detector to counter effect this kind of problem, it prove useless when the segment do not find any correspondence with another cameras key-frames and it lags behind.

To be more precise, in this case, the robot started far from the nearest feature and due to the angle of the device, it captures a big section of the floor. In the consecutive images, it detected some undefined registrations, giving a higher covariance and permitting it to move easily in case that it was needed during the optimization phase, but some of the frames manage to find a good registration with another of this frame, removing the status, and aligning itself. This could

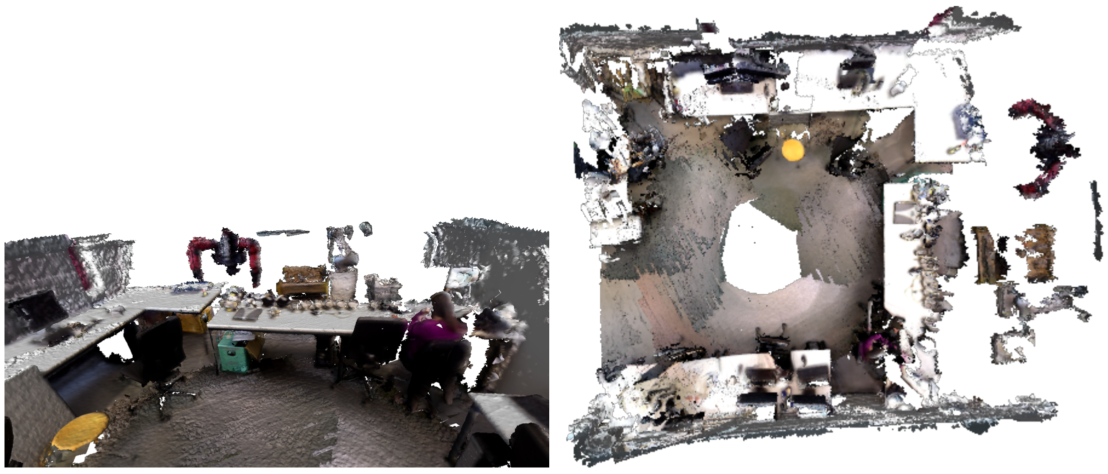


Figure 7.5: Resulting mesh of the CSLAM with the Graphics Lab walking dataset.

be avoided if the camera is able to see more features during its exploration. Also, it is possible to avoid having a camera in the back to avoid drift, or have a more irregular trajectory.

With that said, it was possible to obtain an accurate representation of part of the map in a section of the dataset where the robot turns, as it can be seen in the figure 7.5. Due to the small quantity of frames, the mesh representation looks a little bit more coarse, but it is possible to still be able to use it, since it is possible to distinguish several objects and, as in the calibration sets, the tables and walls, show straight lines and right angles.

In the case of the other dataset, the one in the robotics laboratory, it was not possible to obtain a long enough section of frames to obtain an accurate representation of the map, since it contains two sections of the room, one with a big open space, that has a football field for the robots, but at this time it was empty and the dataset was captured far enough of the main features. The other section of this laboratory, can be seen in the calibration dataset results and in figure 7.3. The back camera started far enough from the table and walls to trigger this problem, making it not possible to obtain an accurate result. Although it was possible with just a few key-frames, to obtain a visualization of the football field, showing that the calculated extrinsic parameters were accurate enough. It is possible to clearly see this through the straight lines drawn in the football field in figure 7.6.

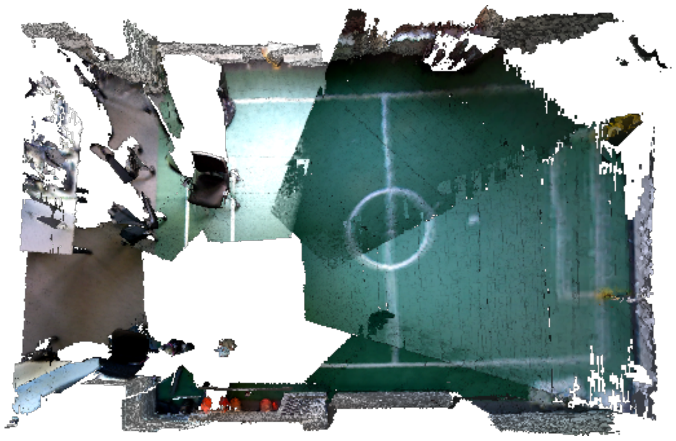


Figure 7.6: Resulting mesh of the CSLAM with the Robotics Lab walking dataset.

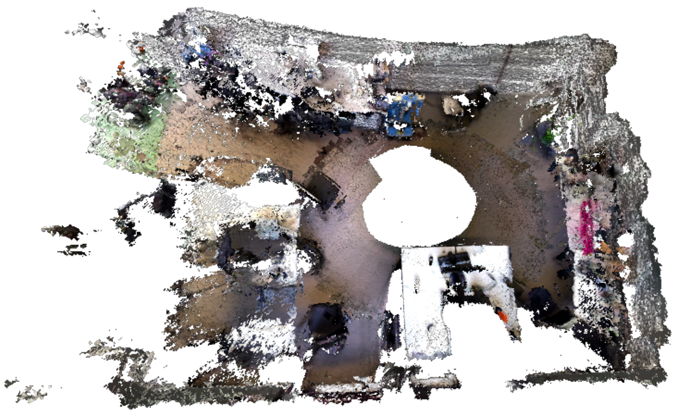
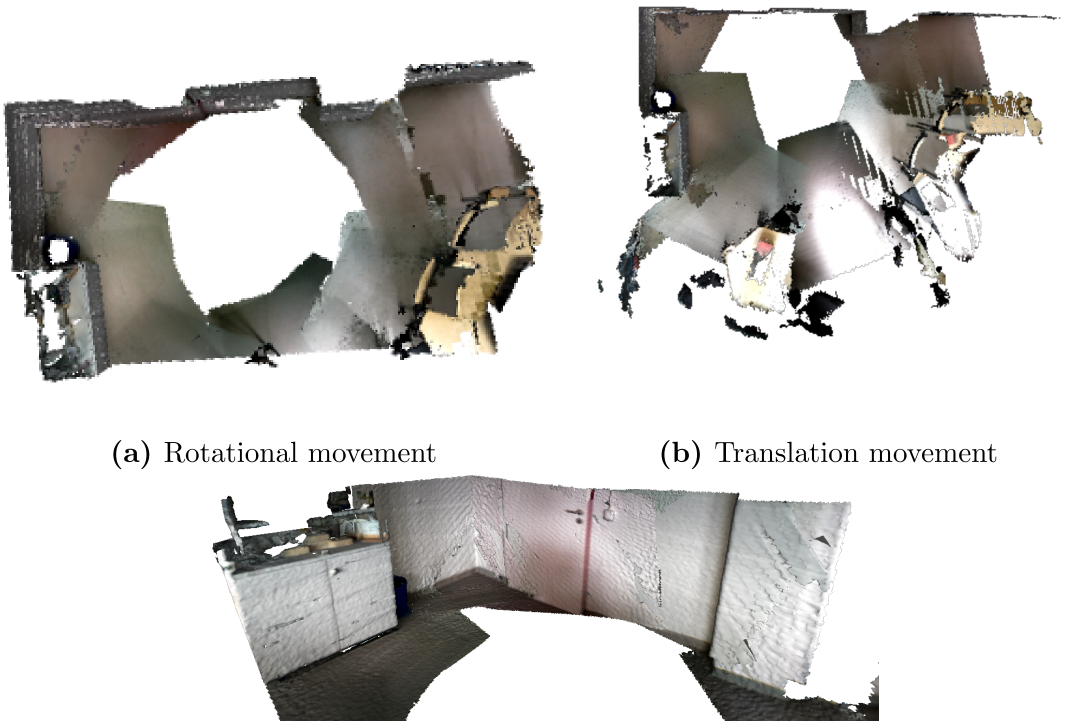


Figure 7.7: Resulting mesh of the CSLAM with the Robotics Lab dataset without the undefined registrations detector.

7.1.3 Undefined Registration Datasets

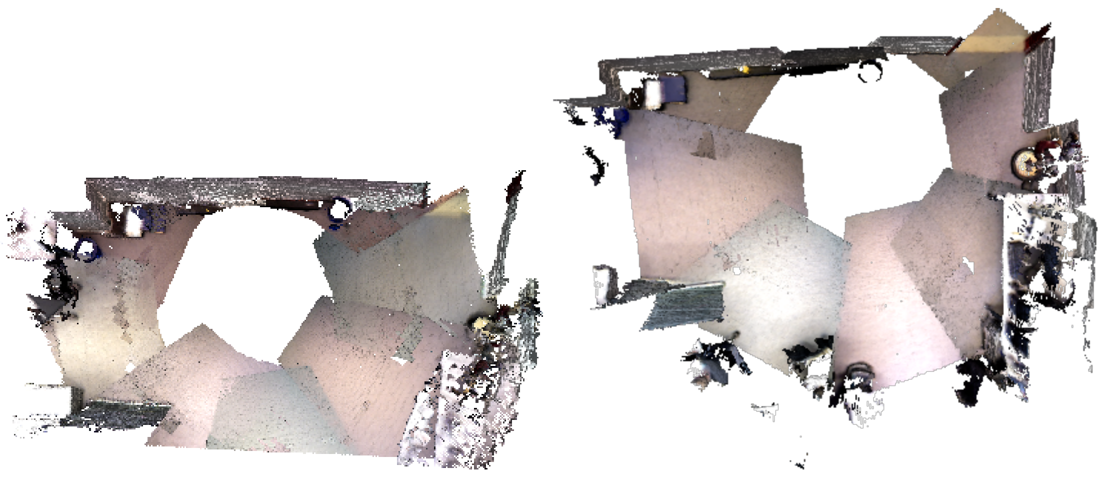
The undefined registrations are not simply registrations in sections with white walls or completely featureless places, it may occur in places where the noise damage the depth perception like objects far from the robot. This scenario occurred more than once in the Robotics laboratory calibration dataset, and it was promptly detected and counter by the algorithm. To show as an example, the same dataset was used without the undefined registration, and this time the results were not as good as the ones explained in section 7.1.1, and can be seen in figure 7.7.

In the graphics laboratory calibration dataset some undefined registrations were also encountered, but in this case, without this detection it did not have a deterring effect as in the other calibration dataset.


(a) Rotational movement (b) Translation movement

(c) Rotational movement side view

Figure 7.8: Mesh of the resulting map of the CSLAM with the Conference room datasets.



(a) Rotational movement **(b)** Translation movement

Figure 7.9: Mesh of the resulting map of the CSLAM with the Graphics Lab datasets near the wall.

Finally, we can observe in figure 7.8 and 7.9 the map from the conference room dataset and the graphics laboratory. Both sets are close to a white wall, but the conference room also has a column and a closed door. In this case, the error mentioned before occurred, but not in the section of the wall, and that is why we truncated the mesh, to examine the section of the wall. We did in both cases two datasets, one for a rotation movement, and the other for a translation movement. In both cases, undefined registrations were encountered and managed correctly, creating a complete and straight wall as shown in the images. Before, in section 3.3, we show a similar dataset, but with four cameras. In that case, one of the camera lags behind due to part of the wall being captured and wrongly registered. Also, in figure 7.8 it is observable the difference of color between the cameras due to the illumination. In the part farther from the wall, the floor looks almost white due to this effect.

As we mentioned before, finding and managing undefined registration does not secure a successful result, but it may help greatly with some sections. The Horsaal 3 experiments had too many undefined registrations, and it was impossible to achieve a successful result.

Camera	t_x	t_y	t_z	q_x	q_y	q_z	q_w
1	-0.01448	0.09631	0.02734	0.3597	0.08025	-0.2773	0.8873
2	-0.1649	0.02459	-0.144	0.601	-0.03981	-0.4636	0.6498
3	-0.1752	0.04716	-0.2139	0.7645	-0.005034	-0.5407	0.3509
4	-0.1636	0.1029	-0.02054	0.8561	-0.02698	-0.5134	-0.05351
5	-0.01923	0.2662	0.07208	0.7803	-0.05447	-0.4739	-0.4044
6	-0.03415	0.1185	-0.009551	-0.5764	0.08186	0.3496	0.7341
7	0.004486	0.1374	0.01044	-0.2891	0.09309	0.1609	0.9391

Table 7.3: Calibration parameters obtained in the Robotics Lab dataset

Camera	t_x	t_y	t_z	q_x	q_y	q_z	q_w
1	-0.003669	0.006705	0.01356	0.3128	0.08189	-0.2261	0.9189
2	-0.06972	-0.09536	-0.07841	0.5651	-0.02274	-0.4518	0.6899
3	-0.1211	-0.0397	-0.1301	0.7607	0.002553	-0.508	0.404
4	-0.1488	0.03046	0.03819	0.859	-0.02163	-0.5114	-0.006827
5	-0.06501	0.1146	0.01537	0.7918	-0.04451	-0.4793	-0.376
6	-0.05627	-0.02383	0.07303	-0.6035	0.08429	0.3739	0.6992
7	-0.0328	0.1477	-0.02664	-0.2948	0.09169	0.2148	0.9266

Table 7.4: Calibration parameters obtained in the Graphics Lab dataset

7.2 Quantitative Results

7.2.1 Calibration Datasets

One of the most important aspects of this thesis is the calibration parameters obtained while performing the RGB-D SLAM. To demonstrate the accuracy of the parameters, first, the generated maps should be consistent, as shown in 7.1. Additionally, the algorithm should be able to reproduce such results in different environments with the same camera set-up; therefore, we performed two tests in different feature-rich environments and compared the generated extrinsic parameters, as it can be observed in the tables 7.3, 7.4, and 7.5.

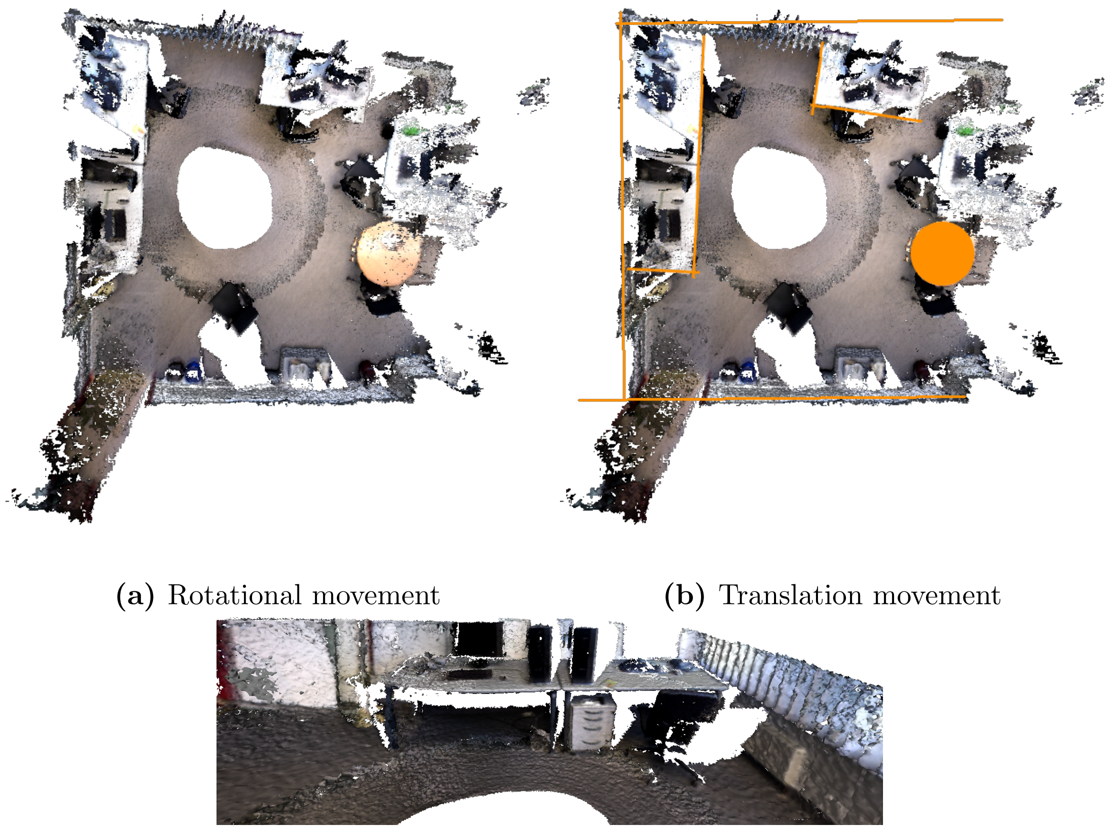
As mentioned in the previous chapters, the seven values of the extrinsic parameters shown in the tables represent the t_x, t_y, t_z values of the translation with respect to the worlds origin and the q_x, q_y, q_z, q_w values of the quaternion that represents the rotation of the camera.

Camera	t_x	t_y	t_z	q_x	q_y	q_z	q_w
1	0.05264	-0.05488	0.05052	0.9753	-0.0009596	-0.01102	-0.2206
2	0.0988	0.1247	0.04931	0.9773	-0.0006695	-0.003705	-0.2117
3	-0.01282	0.131	-0.01295	0.9839	-0.004633	-0.006161	-0.1787
4	-0.04542	0.06663	-0.0491	0.9859	-0.0006975	-0.003218	-0.1675
5	-0.02571	0.1554	-0.05895	0.9921	0.0001672	-0.0009158	-0.1257
6	-0.1193	-0.06468	-0.09575	0.9869	-1.709e-05	-0.004716	-0.1612
7	-0.02221	0.04035	-0.02739	-0.002838	-0.00297	0.9798	0.2001

Table 7.5: Difference between the calibration parameters in both scenes.

Camera	Mean Squared Error
1	0.00608147
2	0.00604256
3	0.00546144
4	0.00296344
5	0.00409099
6	0.00532826
7	0.00306612

Table 7.6: Error of the difference between the calibration parameters in both scenes.



(a) Rotational movement **(b)** Translation movement

(c) Rotational movement side view

Figure 7.10: Mesh of the resulting map with only one camera and MRSMap

7.2.2 Difference between single and multiple camera

One of the most important differences in our approach is the use of multiple cameras. To decide which approach has a better performance, we check the resulting map, compare their entropies and also compare the amount of frames needed for each of them to complete a full map in a only rotational dataset. To determine the moment in which the map was fully created, we track the loop closures generated, and once all cameras have at least one loop closure with the camera that follows it or with the first frames register by itself, then it can be considered as a full map.

The measurements were executed in the calibration dataset in the graphics laboratory, since it was the one with the best qualitative results. The single camera mesh can be seen in figure 7.10 and even though the walls look straight, we can observe some deviations in the angles of the wall, specifically in the one where the windows are located. Also, the mesh looks rough, since it does not get enough points per place and the quality of it is reduced.

Number of Cameras	Entropy
One camera	-2.73297
Eight cameras	-2.79559

Table 7.7: Difference of the entropy in both cases.

To reach the end of the loop, it required 680 frames for the single camera system. In the case of the multiple sensor one, it took 783 frames to generate a complete map, that is 103 frames more than the required frames with only one camera. It took longer to generate all the loop closures, probably due to the difference of colors between the camera, since the white balance varied a lot from sensor to sensor. Even though the single camera generated the complete loop faster, the multi camera system covered almost all the map with 50 frames as it is the case of the robotics lab walking dataset shown in figure 7.6.

Finally, the entropy between them will give the final point of comparison. As we can see in table 7.7, the entropy of the map generated by the system with eight cameras is smaller than the one generated by the original monocular approach. With these values, and the previous comparisons we can conclude that the multi-camera system have a better performance compared to the original approach, but still have some issues that must be addressed to make the system even better.

8 Conclusions

We presented an efficient improvement to the MRSMaps algorithm that calculates the calibration parameters simultaneously to the already existent RGB-D SLAM. These parameters were computed successfully and later on they were used in the following SLAM experiments. Not all of them were as successful as expected, although, thanks to the multiple cameras and the correct calibration, it was possible to obtain a map representation of the surroundings.

The extrinsic parameters gave consistent results on the experiments, not only in the qualitative, but also in the quantitative results. The multi-camera system introduced in this thesis also proved to work as expected, and even with the limitations and problems, it managed to create a map that contains accurate data comparable to the actual one and that it may be used by other algorithms. Comparing to the original monocular system, our approach show a significant improvement with respect to the creation of the map, since it clearly built a more accurate map as explained in the previous sections.

Using the eight cameras in an omnidirectional set-up and the resulting extrinsic parameters obtained via this system, it was possible to build a map from the surroundings with just a few frames. As we saw in Figure 7.6, where only 50 frames were used, the representation can be accurate and it can be produced in a short amount of time. These maps can be used in cases where the robot may not move too much or where it may need a map promptly. Since it is created without needing many frames, it can be used to create a new map every time it is needed to cope with objects that may change place in the environment. This way, the map can be restarted and updated without loosing much time, and at the same time, giving a more accurate representation of the current state of the surroundings.

The results obtained show that indeed, the multi-camera system can be useful in a big variety of cases and that this approach may prove useful for future projects. The approach of this thesis for a multi-camera system, yields good results in some datasets; therefore, it is possible to reuse this project and create additional constraints for it to work in the other cases where it had unexpected behaviours, or do changes to the undefined registration detector to permit it to detect this problematic registrations and avoid it as it have been done in this thesis.

8.1 Limitations and problems

The undefined registration detector found in almost all the datasets these problematic frames and change the transformation to the estimated by the calibration nodes and avoid using these frames incorrectly. Even though it showed an improvement in some datasets, i.e. calibration in the robotics lab, it failed to constraint it correctly to the other good frames in datasets were the loop closure between this frames and the ones from another camera that had good registration was possible. Also, it found some good registrations between frames that only had floor on it, possibly, due to the color of the floor, making it drift even more.

As it was mentioned before, we tried unsuccessfully to temporarily remove the node that represents an undefined registration from the graph. This gave as result, a broken pose graph in some cases where the frame never finds a good registration. Then the optimization phase will create a bigger gap in between the frames that are not connected; therefore it will create even more drift and will not work as it is intended. A similar problem occurs if the covariance is increased with a really big number instead, like $1e+06$ instead of the value used in this project, i.e. $1e+03$.

8.2 Recommendations for future projects

As recommendations to projects based on this thesis, we suggest to explore other ways to detect more accurately the undefined registrations, specifically for the sequences that have only sections of the floor on it. It is possible to continue using the eigenvalues of the covariance matrix, since they model the uncertainty of the registrations, but instead of using two thresholds to classify the registrations, it is possible to use another method such as nearest neighbour classifier, trained with data from different types of images, e.g. white walls, floor sections and feature-rich sections.

Changes in the illumination may generate captures with different colors between the cameras, making it harder to find a loop closure between the sections generated by different cameras. Keeping a similar white balance or a illumination in the room may generate better results. Also, reflective materials may induce a change in the white balance of the camera and it is even possible that they add more noise to the depth image.

The current algorithm could benefit from data that contains more features, for that it is recommended to have objects that generate features 1 meter away from the robot and less than 3 meters away to obtain better results. For datasets in which the robot moves in a straight line, some extra constraints should be added in the back and in the front of the robot, since they will have big sections with only floor and this may generate problems during the execution of the program or as we suggested before, changing the undefined registration detector may help. A similar approach can be used for datasets with cameras viewing sections with no features, which could benefit from some constraint.

As a final remark, it is better to use a robot with the cameras in a height similar to an adult human, circa 1.75 meters. This will give better results, since more features are visible. Also, do not forget to avoid moving objects such as the parts of the robot or living beings that are moving, since this may trigger registrations with the wrong transformations.

List of Figures

Bibliography

[1] K. Amplianitis, M. Adduci, and R. Reulke. “Calibration of a Multiple Stereo and Rgb-D Camera System for 3d Human Tracking”. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 1 (Mar. 2014), pp. 7–14 (cit. on pp. 17, 18).

[2] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers. “Real-Time Camera Tracking and 3D Reconstruction Using Signed Distance Functions”. In: *Proceedings of Robotics: Science and Systems.* Berlin, Germany, June 2013 (cit. on pp. 7, 16).

[3] B. Curless and M. Levoy. “A Volumetric Method for Building Complex Models from Range Images”. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH ’96. New York, NY, USA: ACM, 1996, pp. 303–312. ISBN: 0-89791-746-4 (cit. on pp. 13, 15).

[4] D. Droeschel, J. Stuckler, and S. Behnke. “Local multi-resolution representation for 6D motion estimation and mapping with a continuously rotating 3D laser scanner”. In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on.* May 2014, pp. 5221–5226 (cit. on pp. 12, 40, 44).

[5] E. Fernandez-Moral, J. Gonzalez-Jimenez, P. Rives, and V. Arevalo. “Extrinsic calibration of a set of range cameras in 5 seconds without pattern”. In: *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on.* Sept. 2014, pp. 429–435 (cit. on p. 18).

[6] M. A. Fischler and R. C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography”. In: *Commun. ACM* 24.6 (June 1981), pp. 381–395. ISSN: 0001-0782. URL: http://doi.acm.org/10.1145/358669.358692 (cit. on p. 16).

[7] P. Furgale, J. Rehder, and R. Siegwart. “Unified temporal and spatial calibration for multi-sensor systems”. In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on.* Nov. 2013, pp. 1280–1286 (cit. on pp. 10, 17).

[8] A. Geiger, F. Moosmann, O. Car, and B. Schuster. “Automatic camera and range sensor calibration using a single shot”. In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on.* May 2012, pp. 3936–3943 (cit. on pp. 10, 11, 18).

[9] G. Grisetti, R. Kummerle, C. Stachniss, U. Frese, and C. Hertzberg. "Hierarchical optimization on manifolds for online 2D and 3D mapping". In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on.* May 2010, pp. 273–278 (cit. on p. 24).

[10] L. Heng, G. H. Lee, and M. Pollefeys. "Self-Calibration and Visual SLAM with a Multi-Camera System on a Micro Aerial Vehicle". In: *Proceedings of Robotics: Science and Systems.* Berkeley, USA, July 2014 (cit. on pp. 7, 11).

[11] L. Heng, B. Li, and M. Pollefeys. "CamOdoCal: Automatic intrinsic and extrinsic calibration of a rig with multiple generic cameras and odometry." In: *IROS.* IEEE, 2013, pp. 1793–1800. URL: `http://dblp.uni-trier.de/db/conf/iros/iros2013.html#HengLP13` (cit. on pp. 11, 17–19).

[12] R. Kuemmerle, G. Grisetti, and W. Burgard. "Simultaneous Calibration, Localization, and Mapping". In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* San Francisco, CA, USA, Sept. 2011 (cit. on pp. 11, 12, 19, 24, 27, 29, 37).

[13] R. Kuemmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. "G2o: A general framework for graph optimization". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on.* May 2011, pp. 3607–3613 (cit. on pp. 11, 18, 23, 24, 36, 37).

[14] W. E. Lorensen and H. E. Cline. "Marching Cubes: A High Resolution 3D Surface Construction Algorithm". In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques.* SIGGRAPH '87. New York, NY, USA: ACM, 1987, pp. 163–169. ISBN: 0-89791-227-6 (cit. on p. 13).

[15] S. Miller. *CPU_TSDF.* Accessed 31.01.2015. 2015. URL: `https://github.com/sdmiller/cpu_tsdf` (cit. on pp. 13, 44).

[16] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. "KinectFusion: Real-time dense surface mapping and tracking". In: *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on.* Oct. 2011, pp. 127–136 (cit. on pp. 7, 15, 16, 20).

[17] E. Olson, J. Leonard, and S. Teller. "Fast Iterative Optimization of Pose Graphs with Poor Initial Estimates". In: 2006, pp. 2262–2269 (cit. on p. 24).

[18] S. A. Scherer and A. Zell. "Efficient Onboard RGBD-SLAM for Fully Autonomous MAVs". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013).* Tokyo Big Sight, Japan, Nov. 2013 (cit. on pp. 7, 15, 16, 24).

[19] F. Steinbruecker, J. Sturm, and D. Cremers. "Volumetric 3D Mapping in Real-Time on a CPU". In: *Int. Conf. on Robotics and Automation.* Hongkong, China, 2014 (cit. on pp. 13, 44).

[20] J. Stückler and S. Behnke. "Multi-Resolution Surfel Maps for Efficient Dense 3D Modeling and Tracking". In: *Journal of Visual Communication and Image Representation* 25.1 (Jan. 2014), pp. 137–147 (cit. on pp. 3, 7, 12, 15, 16, 20, 26, 27, 34, 35).

[21] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. "A Benchmark for the Evaluation of RGB-D SLAM Systems". In: *Proc. of the International Conference on Intelligent Robot Systems (IROS).* Oct. 2012 (cit. on p. 17).

[22] N. Sünderhauf and P. Protzel. "Switchable constraints for robust pose graph SLAM." In: *IROS.* IEEE, 2012, pp. 1879–1884. ISBN: 978-1-4673-1737-5. URL: `http://dblp.uni-trier.de/db/conf/iros/iros2012.html#SunderhaufP12` (cit. on p. 24).

[23] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents).* The MIT Press, 2005. ISBN: 0262201623 (cit. on p. 15).

[24] S. Yang, S. A. Scherer, and A. Zell. "Robust Onboard Visual SLAM for Autonomous MAVs". In: *2014 International Conference on Intelligent Autonomous Systems (IAS-13).* Padova, Italy, July 2014 (cit. on pp. 7, 11).