KTH Computer Science
and Communication

Building a 3D map from RGB-D sensors

VIRGILE HÖGMAN

Master's Thesis in Computer Science

Supervisor: Alper Aydemir
Examiner: Stefan Carlsson

Computer Vision and Active Perception Laboratory
Royal Institute of Technology (KTH), Stockholm, Sweden

TRITA xxx yyyy-nn

Abstract

For a mobile robot exploring an unknown static environment, localizing itself and building a map at the same time is a chicken-or-egg problem, known as Simultaneous Localization And Mapping (SLAM). When a GPS receiver cannot be used, such as in indoor environments, the measurements are generally provided by laser rangefinders and stereo cameras, but they are expensive and standard laser rangefinders offer only 2D cross sections. However, recently there has been a great interest in processing data acquired using depth measuring sensors due to the availability of cheap and performant RGB-D cameras. For instance, the Kinect developed by Prime Sense and Microsoft has considerably changed the situation, providing a 3D camera at a very affordable price.

In this study, we will see how a 3D map based on a graphical model can be built by tracking visual features like SIFT/SURF, computing geometric transformations with RANSAC, and applying non-linear optimization techniques to estimate the trajectory. This can be done from a sequence of video frames combined with the depth information, using exclusively the Kinect, so the field of applications can be wider than robotics.

Referat

Skapa en 3D karta från en RGB-D kamera

En robot som utforskar en okänd statisk miljö, där den inte bara ska lokalisera sig utan också skapa en karta på samma gång, måste hantera det problem som benämns Simultaneous Localization And Mapping (SLAM). När en GPS-mottagare inte kan användas, typiskt i en inomhus-miljö, brukar laseravståndsmätare och kameror andvändas som sensorer. En laseravståndsmätare är dyr och erbjuder information enbart i 2D. Kameror kräver mycket databehandling eftersom de inte tillhandahåller djupinformation. Det har på sistone vuxit fram ett stort intresse för behandling av data från RGB-D kameror, dvs kameror som utöver den vanliga bilden ger djupinformation. Nyligen blev Kinect, utvecklad av Prime Sense och Microsoft, tillgänglig. Kinect erbjuder en RGB-D data till ett väldigt attraktivt pris.

I denna rapport studerar vi hur man kan skapa en 3D karta baserad på en grafisk model genom att spåra visuella landmärken från tex SIFT/SURF, beräkna geometriska transformationer med RANSAC, och applicera icke-linjära optimeringstekniker för att skatta hur sensorn rört sig. Vi visar hur vi kan använda en vanlig Kinect till detta utan några andra sensorer på roboten som annars ofta är fallet. Detta innebär att tillämpningsområdet kan vara bredare än robotik.

Acknowledgments

> “ The most exciting phrase to hear in science, the one that heralds new discoveries, is not ’Eureka!’ but ’That’s funny...’ ”
>
> — Isaac Asimov

My first thanks go to my supervisor Alper Aydemir, for carefully following my progress and always giving me some useful suggestions to solve the problems raised all along this work. I am grateful for all the time he spent in discussions, experiments and practical issues. I also want to warmly thank Giorgio Grisetti for his most valuable advices, Patric Jensfelt and John Folkesson for their help, and my examiner Stefan Carlsson. My deepest thoughts go to my family and friends, especially to my father Hugues for constantly showing interest in my activities, and for his wise support.

This work was achieved at Computer Vision and Active Perception Lab, at the Royal Institute of Technology of Stockholm, a very pleasant ambient to work in, for the quality and the atmosphere of the school by itself, and above all for the persons I encountered, thanks to their availability, involvement, and their ease to share their knowledge. Special thanks to André Susano Pinto for his friendship, his smart ideas, and the nice moments spent together during his stay. Staff, students, and visitors, this time would not have been the same without Alessandro Pieropan, Ali Mosavian, Andrzej Pronobis, Cheng Zhang, Christian Smith, Gert Kootstra, Johan Ekekrantz, Josephine Sullivan, Magnus Burènius, Miroslav Kobetski, Niklas Bergström, Oscar Danielsson, Renaud Detry, Lazaros Nalpantidis, Victoria Matute Arribas, and many others.

Contents

Chapter 1

Introduction

To navigate in an unknown environment, a mobile robot needs to build a map of the environment and localize itself in the map at the same time. The process addressing this dual problem is called Simultaneous Localization And Mapping (SLAM). In an outdoor environment, this can generally be solved by a GPS which provides a good accuracy for the tasks the robot can take on. However, when moving indoor or in places where the GPS data is not available, or not reliable enough, it can become difficult to estimate the robot's position precisely and other solutions have to be found.

The main problem raised with SLAM comes from the uncertainty of the measurements, due to the sensory noise or technical limitations. Probabilistic models are widely used to reduce the inherent errors and provide satisfying estimations. While this process is generally based on data provided by sensors such as laser scanners, combined with the odometry, Visual Simultaneous Localization And Mapping (VSLAM) focuses on the use of camera, as illustrated in figure 1.1.

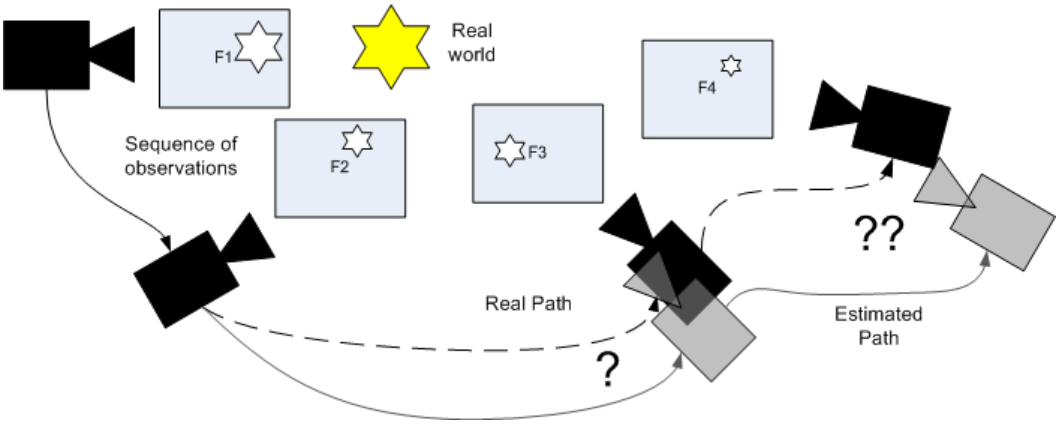


Figure 1.1: Concept of Visual SLAM. The poses of the camera (hence, the robot for a fixed camera) are determined from video data. The estimations generally drift with respect to the real trajectory, and the uncertainty grows over time.

1.1 Context

Currently, most of robotic mapping is performed using sensors that offers only a 2D cross section of the environment around them. One reason is that acquiring high quality 3D data was either very expensive or had hard constraints on the robot movements. Therefore, research has mainly focused on laser scanners to solve the SLAM problem, although there are some methods making use of stereo and monocular cameras [14] [23]. However, recently there has been a great interest in processing data acquired using depth measuring sensors due to the availability of cheap and efficient RGB-D cameras. For instance, the Kinect Camera developed by Prime Sense and Microsoft has considerably changed the situation, providing a 3D camera at a very affordable price. Primarily designed for entertainment, it has received a warm welcome in the research community, especially in robotics.

In the past, to solve the inherent problem of drift and provide a reliable estimation of the camera poses, most of the projects have used techniques such as Extended Kalman Filtering (EKF) or particle filters [25]. More recently, several methods rely on pose graphs to model and improve the estimations [24] [9] [18] [8]. Some projects making use of both RGB-D data and graph optimization [11] [5] illustrate well the interest for such an approach.

The work presented in this thesis is done at the Computer Vision and Active Perception Lab (CVAP) at KTH, the Royal Institute of Technology, in Stockholm. Since 1982, the research at CVAP focuses on the topics of computer vision and robotics.

1.2 Goals

The main goal of this thesis is to build a 3D map from RGB and depth information provided by a camera, considering a 6 Degrees-of-Freedom (DOF) motion system. The hardware device used for the experimentations is the Microsoft Kinect but this work could be extended to any system providing video and depth data.

The VSLAM process can be described as estimating the poses of the camera from its data stream (video and depth), in order to reconstruct the entire environment while the camera is moving. As the sensory noise leads to deviations in the estimations of each camera poses with respect to the real motion, the goal is to build a 3D map which is close, as much as possible, to the real environment.

One objective is to have an overview of the different methods and techniques, so the problems can better be identified and then analyzed more deeply after doing some experiments. However, this work will focus on the use of visual features. Though a VSLAM system is intended to be used in real time, some of the processing may be done without considering the performance as a main priority, and could therefore be delayed. The rendering of the scene is also out of scope of this study.

Figure 1.2: Microsoft Kinect mounted on a mobile robot platform at CVAP (KTH)

1.3 Thesis outline

The rest of the document is structured as follows:

Chapter 2 presents the background and the underlying concepts that are most commonly used in this area, with features and methods commonly used in computer vision, and general notions about SLAM.

Chapter 3 presents the feature matching between a couple of frames, and how a 3D transformation can be computed from these associations using the depth data.

Chapter 4 describes how a map can be built, by estimating the camera poses through the use of a pose graph, refining them through the detection of loop closures, and finally performing a 3D reconstruction.

Chapter 5 presents the experimentations, the software, how the data was acquired, and finally the results, with examples of maps generated from different datasets.

Chapter 6 presents the conclusions and the future works with some suggestions of possible improvements.

Chapter 2

Background

In this chapter, we first present the Kinect camera, describe some of the features and methods commonly used in computer vision, and finally give a short introduction about SLAM concepts.

2.1 Microsoft Kinect

As mentioned in the introduction, the hardware used in this work is the Kinect, a device developed by PrimeSense, initially for the Microsoft Xbox 360 and released in November 2010. It is composed by an RGB camera, 3D depth sensors, a multi-array microphone and a motorized tilt. In this work, only the RGB and depth sensors are used to provide the input data.

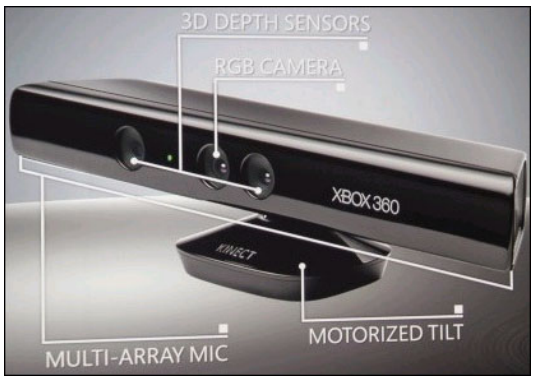


Figure 2.1: The Kinect sensor device (courtesy of Microsoft)

Main characteristics of the Kinect:

- The RGB sensor is a regular camera that streams video with 8 bits for every color channel, giving a 24-bit color depth. Its color filter array is a Bayer filter mosaic. The color resolution is 640x480 pixels with a maximal frame rate of 30 Hz.

- The depth sensing system is composed by an IR emitter projecting structured light, which is captured by the CMOS image sensor, and decoded to produce the depth image of the scene. Its range is specified to be between 0.7 and 6 meters, although the best results are obtained from 1.2 to 3.5 meters. Its data output has 12-bit depth. The depth sensor resolution is 320x240 pixels with a rate of 30 Hz.
- The field of view is 57° horizontal, 43° vertical, with a tilt range of ± 27°.

The drivers used are those developed at OpenNI[1] (Open Natural Interface), an organization established in November 2010, launched by PrimeSense and joined by Willow Garage, among others. The OpenNI framework offers high level functions which are mainly oriented for the gaming experience, such as gesture recognition and motion tracking. It aims to provide an abstraction layer with a generic interface to the hardware devices. In the case of the Kinect, one advantage is that the calibration of the RGB sensor with respect to the IR sensor is ensured, so the resulting RGB and depth data are correctly mapped with respect to a unique viewpoint.

2.2 Features

In this work, we focus on tracking some parts of the scene observed by the camera. In computer vision, and more specifically in object recognition, many techniques are based on the detection of points of interests on object or surfaces. This is done through the extraction of *features*. In order to track these points of interests during a motion of the camera and/or the robot, a reliable feature has to be invariant to image location, scale and rotation. A few methods are briefly presented here:

Harris Corner A corner detector, by Harris and Stephens [10]

SIFT Scalar Invariant Feature Transform, by David Lowe [16]

SURF Speeded Up Robust Feature [1]

NARF Normal Aligned Radial Feature [22]

BRIEF Binary Robust Independent Elementary Feature [3]

There are two aspects concerning a feature: the *detection* of a keypoint, which identifies an area of interest, and its *descriptor*, which characterizes its region. Typically, the detector identifies a region containing a strong variation of intensity such as an edge or a corner, and its center is designed as a keypoint. The descriptor is generally computed by measuring the main orientations of the surrounding points, leading to a multidimensional *feature vector* which identifies the given keypoint. Given a set of features, a matching can then be performed in order to associate some pairs of keypoints between a couple of frames.

[1] http://www.openni.org/

The features listed previously can be summarized in table 2.1.

Feature	Detector	Descriptor
Harris Corner	Yes	No
SIFT	Yes	Yes
SURF	Yes	Yes
NARF	Yes	Yes
BRIEF	No	Yes

Table 2.1: Overview of some existing features (non-exhaustive list)

2.2.1 Harris Corner

Known as the Harris corner operator, this is one of the earliest detector, as it was proposed in 1988 by Harris and Stephens [10]. The notion of corner should be taken in a wide sense as it allows to detect not only corners, but edges and more generally, keypoints. It is done by computing the second moment matrix (or auto-correlation matrix) of the image intensities, describing its local variations. One of the main limitation with the Harris operator, at least in its original version, concerns the scale invariance as the matrix should be recomputed for a different scale. Therefore, we will not give further details about this method.

2.2.2 SIFT feature

The Scalar Invariant Feature Transform (SIFT) is a method presented by David Lowe [16], now widely used in robotics and computer vision. This is a method to detect distinctive, invariant image feature points, which easily can be matched between images to perform tasks such as object detection and recognition, or to compute geometrical transformations between images.

The main idea of the SIFT method is to define a cascade of operations following an increasing complexity, so that the most expensive operations are only performed to the most probable candidates.

1. The first step relies on a pyramid of Difference-of-Gaussian (DoG) in order to be invariant to scale and orientation.

2. From this, stable keypoints are determined with a more accurate model.

3. The image gradient directions is then used to assign one or more orientations to the keypoints.

4. The local image gradients are then transformed to be stable against distortion and changes in illumination.

Detector One characteristic of the SIFT detector is to be scale invariant. The scale invariance ensures that a keypoint can be detected in a stable way at different scales, which is a fundamental requirement for a mobile robot. As the platform of the camera moves, the areas containing the points of interest will appear larger or smaller, relatively to a scale factor. Most of these concepts concerning the study of the scale space are based on the works of Lindeberg [15]. The general idea is shown in figure 2.2. First, the scale space is defined as the convolution of an image I with a Gaussian G with variance σ.

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

where

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

The DoG is the difference between two layers in scale space along the σ axis:

$$\begin{aligned} D(x, y, \sigma) &= (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \\ &= L(x, y, k\sigma) - L(x, y, \sigma) \end{aligned}$$

This provides a close approximation to the scale-normalized Laplacian of Gaussian $\sigma^2\nabla^2 G$, as shown by Lindeberg [15]:

$$\sigma\nabla^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x, y, k\sigma) - G(x, y, \sigma)}{k\sigma - \sigma}$$

and therefore:

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k - 1)\sigma^2\nabla^2 G$$

The σ^2 factor allows the scale invariance. Then, the localization of the keypoints is done by finding the extrema in the DoG, which approximates the Laplacian $\sigma^2\nabla^2 G$. Lowe shows that the remaining factor $(k - 1)$ does not influence the location.

Descriptor In order to provide a good basis for the matching, the descriptor has to be highly distinctive and invariant to remaining variations such as change in illumination or 3D viewpoint. The SIFT descriptor is based on a histogram of local oriented gradients (HoG) around the keypoint. It is stored as a 128 dimensional feature vector (4x4 descriptor with 8 orientation bins). The figure 2.3 illustrates this concept with a smaller 2x2 descriptor.

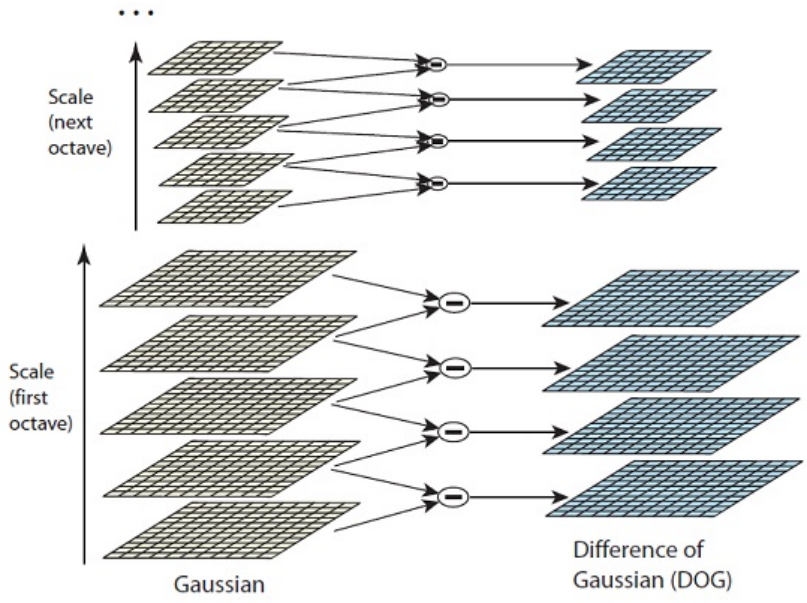


Figure 2.2: Difference of Gaussians at different scales (courtesy of David Lowe). For each octave of the scale space, the image is convolved with a Gaussian, which variance σ is multiplied each time by a constant factor. Two consecutive results are substracted to give the DoG which approximates the Laplacian of Gaussian. After each octave, the image is downsampled by two, and the process is repeated doubling the variance σ. The initial value of σ can be changed accordingly to the type of application.

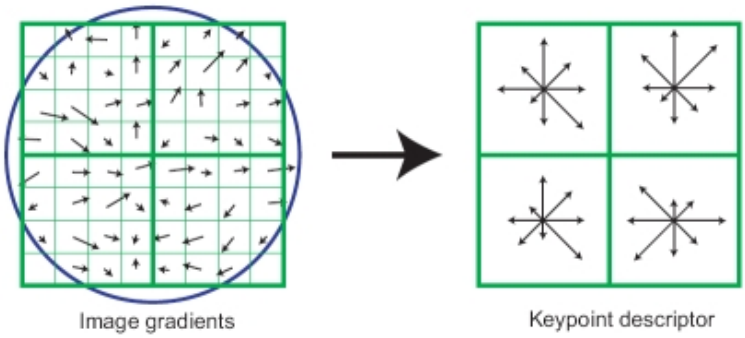


Figure 2.3: SIFT descriptor (courtesy of David Lowe). Illustration of how the gradient directions are accumulated into orientation histograms around the center of the region defined by the keypoint. The gradients are first weighted by a Gaussian window represented by the circle. From the 8x8 sample, a 2x2 descriptor is computed this way, where each bin covers a 4x4 subregion.

Matching Once the keypoints are found and the descriptors computed, the next step is to perform fast searches on the features in order to identify candidate matching, to associate some pairs of features between the two sources. Refer to the method described in [16] for further details.

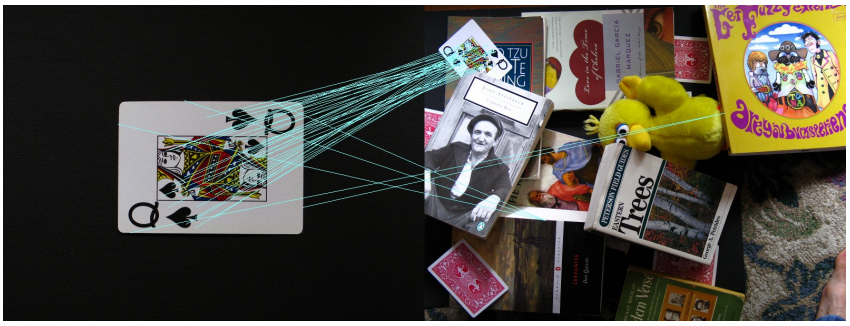


Figure 2.4: SIFT features matched between the two images (courtesy of Rob Hess)

For this work, we use the SIFT library developed by Rob Hess [12]. Additionally to the feature extraction, this library provides a useful function to perform the initial matching through a kd-tree. There is also a set of template functions that can be used to compute geometrical transformations with RANdom SAmple Consensus (RANSAC) (method described in 2.3.1), but the given interface is written for 2D operations and therefore will not be used.

2.2.3 SURF feature

The Speeded Up Robust Feature (SURF) provides a robust detector and descriptor [1], that can be used in computer vision tasks like object recognition or 3D reconstruction. It is partly inspired by the SIFT descriptor, both are using local gradient histograms. The main difference concerns the performance, lowering the computational time through an efficient use of integral images for the image convolutions, Hessian matrix-based detector (optimized through approximations of the second order Gaussian partial derivatives, see figure 2.5), and sums of approximated 2D Haar wavelet responses for the descriptor (see figure 2.7). The standard version of SURF is several times faster than SIFT and claimed by its authors to be more robust against different image transformations than SIFT.

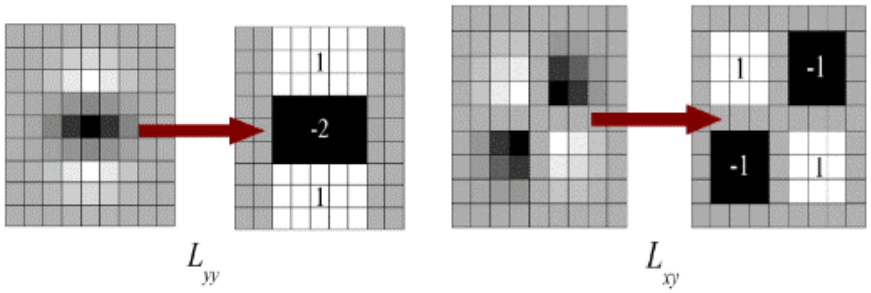


Figure 2.5: SURF detector relies on approximation of the Gaussian second order derivatives.

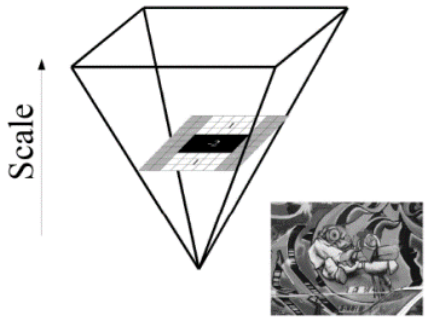


Figure 2.6: SURF scale space is built by changing the filter size.

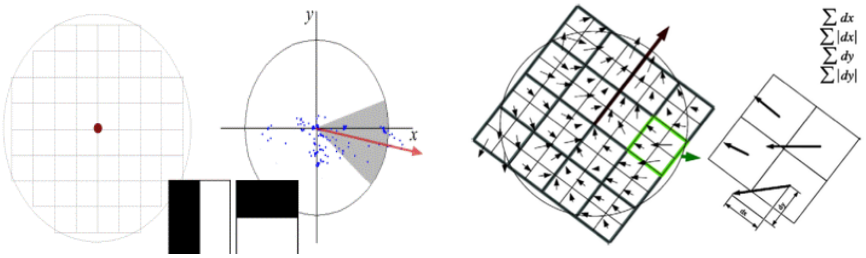


Figure 2.7: SURF keypoint orientation is found by summing up the response of Haar wavelets with a sliding window of 60 degrees. The descriptor is finally computed by splitting the region around the keypoint into 4x4 subregions and computing Haar wavelet responses weighted with a Gaussian kernel.

2.2.4 NARF feature

The Normal Aligned Radial Feature (NARF) is presented by Bastian Steder in [22]. It is meant to be used on single range scan obtained with 3D laser range finders or stereo camera. This feature is available in the Point Cloud Library (PCL) [21], which is part of the Robot Operating System (ROS), but also released as a standalone library. Its detector looks for stable areas with significant change in vicinity, that can be identified from different viewpoints. The descriptor characterizes the area around the keypoint by calculating a normal aligned range value patch and finding the dominant orientation of the neighboring pixels.

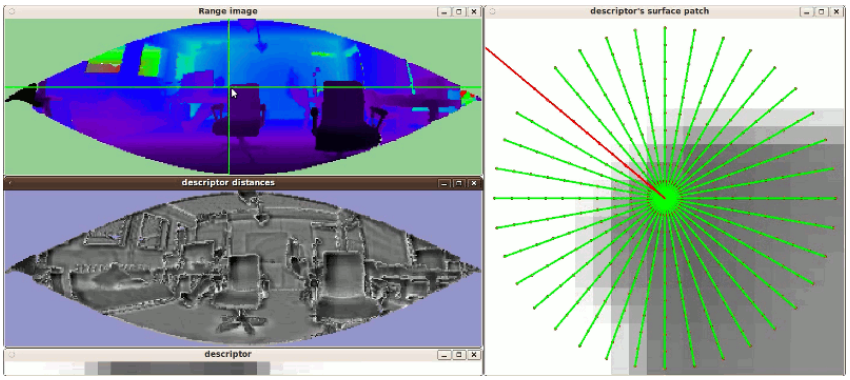


Figure 2.8: NARF feature (courtesy of Bastian Steder). From the range image, a descriptor is computed at the given keypoint localized by the position of the green cross. The right figure shows the surface patch containing the corner of the armchair and the dominant orientation which is pointed by the red arrow. Each component of the descriptor is associated to a direction given by one of the green beams. The bottom left picture shows the values of this descriptor. The middle left figure shows the distance to this descriptor for any point in the scene, and therefore their similarity with the current keypoint. The darkest areas are the closest, identifying the areas which are the most similar to the left corner of the armchair.

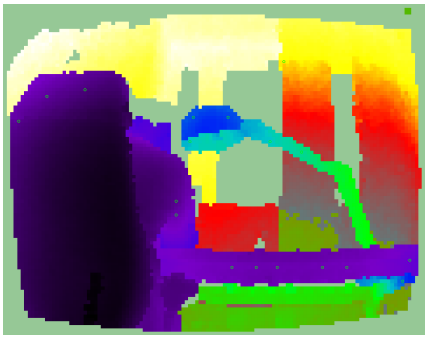


Figure 2.9: NARF keypoints computed on a desk

2.2.5 BRIEF feature

The Binary Robust Independent Elementary Feature (BRIEF) presented in [3] is an efficient alternative for the descriptor, based on binary strings computed directly from image patches, and measures the Hamming distance instead of the L_2 norm commonly used for high dimension descriptors. As the binary comparison can be performed very efficiently, the matching between several candidates can be done much faster. The use of a BRIEF descriptor supposes the keypoints are already known, this can be done with a detector such as SIFT or SURF. A deeper study is available in the PhD thesis of Calonder [4], who is the main author of the BRIEF features. This document presents comparative evaluation of BRIEF, SIFT and SURF. The main interest of this descriptor resides in its performance.

2.3 Computing a transformation

Once the features have been computed on the whole image, the goal is to track them during the movement of the camera. This is done by associating them between different frames. Here, we only consider the matching for a couple of frames, the process can then be repeated on the whole sequence of frames. To associate several pairs of features is not straightforward, as the descriptors are not exactly the same between two different frames, first as a consequence of the movement of the camera and also because of the sensory noise. The best matching then consists in finding the correct associations with a good belief. Most of the algorithms work with different steps, first from a sparse level where the hypothesis is wide, to eliminate the most obvious mismatches at lower computational cost, and then refined. The matches that fit to the model are called the *inliers*, while the matches being discarded are called the *outliers*.

2.3.1 RANSAC

The RANdom SAmple Consensus (RANSAC) is an iterative method, widely known in computer vision, to estimate the parameters of a transformation given a dataset. It was first published in 1981 by Fischler and Bolles [6]. Generally, by using a least square approach, the parameters which satisfy the whole dataset can be found, but this is likely not the optimal solution when there are some noisy points. Hence, the idea is to find the parameters which are valid for *most* of the points, a consensus, by discarding the noisy points. For each iteration, a very small number of samples are *randomly* selected to define a model, representing one hypothesis. This model is then evaluated for the whole dataset with an error function. This is repeated several times by choosing new samples for each iteration and keeping the best transformation found. After running this for a fixed number of steps, the algorithm is guaranteed to converge to a better transformation (with a lower error), but it does not necessarily find the best one, as all the possibilities have not been tested. The general algorithm is presented at page 14.

Algorithm 1 General RANSAC

Require: Dataset of points
  $bestModel$, $bestInliers \leftarrow \emptyset$
  Define the number of iterations N
  for $iteration = 1$ to N **do**
    $samples \leftarrow$ Pickup k points randomly
    Compute $currentModel$ from $samples$ (base hypothesis)
    $inliers \leftarrow \emptyset$
    for all points **do**
      Evaluate $currentModel$ for the point and compute its error
      if $error < threshold$ **then**
        $inliers \leftarrow inliers + point$
      end if
    end for
    Count number of inliers and compute mean error
    if $currentModel$ is valid **then**
      Recompute $currentModel$ from $inliers$
      if $currentModel$ better than $bestModel$ **then**
        $bestModel \leftarrow currentModel$
        $bestInliers \leftarrow inliers$
      end if
    end if
  end for
  return $bestModel$, $bestInliers$

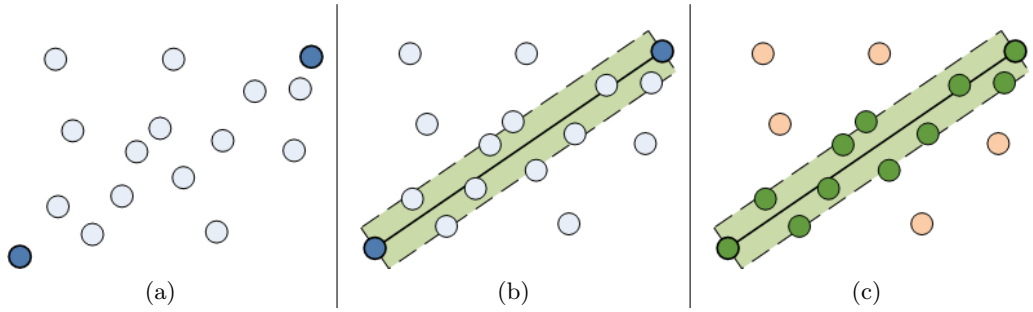


Figure 2.10: Illustration of a RANSAC iteration, for a 2D dataset. (a) First, k items are chosen randomly among the set (here k=2). (b) From these points, a model is defined. Here, a line is drawn between the 2 chosen points, with an area of validity defined by a given threshold. (c) The model is then evaluated by measuring the error for each point, here by computing the distance to the line. This separates the dataset into two subsets: the inliers, in green (fitting to the model) and the outliers (ignored). The ratio of inliers relatively to the total number of items can give a numerical evaluation of the model ("goodness"). The initial model is generally recomputed taken all the inliers into account.

The main difficulty with the RANSAC algorithm is to define the number of iterations and the thresholds. For the number of iterations, it is possible to define the ideal value according to a desired probability. Considering a sample of k points, randomly chosen, if p denotes the probability that at least one of the sample set does not include an outlier, u the probability of observing an inlier, we have then:

$$1 - p = (1 - u^k)^N$$

It can be more convenient to define $v = 1 - u$ as the probability of observing an outlier. From this, we can obtain the required number of iterations:

$$N = \frac{log(1 - p)}{log(1 - (1 - v)^k)}$$

For the threshold, it depends of the problem and the application, generally this is done empirically. There are many variants of the RANSAC that try to overcome this problem, such as the MLESAC method [26] using M-estimators in order to find the good parameters in a robust way.

2.3.2 ICP

The Iterative Closest Point (ICP) is an algorithm presented by Zhang [27]. It iteratively revises the rigid transformation needed to minimize the distance between two sets of points, which can be generated from two successive raw scans. Considering the two sets of points (p_i, q_i), the scope is to find the optimal transformation, composed by a translation t and a rotation R, to align the source set (p_i) to the target (q_i). The problem can be formulated as minimizing the squared distance between each neighboring pairs:

$$min \sum_i ||(Rp_i + t) - q_i||^2$$

As any gradient descent method, the ICP is applicable when we have in advance a relatively good initial guess. Otherwise, it is likely to be trapped into a local minimum. One possibility, as done by Henry [11], is to run RANSAC first to determine a good approximation of the rigid transformation, and use it as the first guess for the ICP procedure.

2.4 General concepts for SLAM

As described in the introduction, the SLAM problem can be defined as looking for the best estimation of the robot/camera poses, by reducing the uncertainty due to the noise affecting the measurements. With the probabilistic approach, one way is to use methods such as Expectation Maximization, described for example in [25]. The underlying idea is to compute a most likely map, and every time the estimation of the robot pose is known with more accuracy, the previously computed map is updated and the process is repeated. The momentary estimation of the position, or $belief$, is represented by a probability density function.

By denoting x_t the state of the robot at time t (representing the robot's motion variables such as the poses), z_t a measurement (for example camera images or laser range scans), and u_t the control data (denoting the changes of state), the belief over the state variable x_t is given by:

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t})$$

This posterior is the probability distribution over the state x_t at time t, conditioned on all past measurements $z_{1:t}$ and all past controls $u_{1:t}$.

In the case of VSLAM with the Kinect, we consider the video sequence as a collection of frames, defined by a flow of RGB and depth data. Here, each measurement z_t is a couple of RGB and depth frames at time t . The total number of frames then depends on the duration of the sequence and the effective frame rate, which is bounded by the maximum rate of the Kinect which is 30 Hz.

2.4.1 Filtering vs Smoothing

To solve the SLAM problem, the literature presents different approaches that can be classified either as filtering or smoothing. Filtering approaches model the problem as an on-line state estimation where the state of the system consists in the current robot position and the map. The estimate is augmented and refined by incorporating the new measurements as they become available. The most common techniques are the Extended Kalman Filtering (EKF) and the particle filters [25]. To highlight their incremental nature, the filtering approaches are usually referred to as on-line SLAM methods. The drawback for these techniques is the computational cost for the particle filter as it maintains several hypothesis of the state variables, which can grow fast as the robot moves, while the EKF only keeps one hypothesis.

Conversely, smoothing approaches estimate the full trajectory of the robot from the full set of measurements. These approaches address the so-called full SLAM problem, and they typically rely on least-square error minimization techniques. Their performance could also have been an issue in the past, but now they are an interesting alternative. In this work, we will therefore study the results that can be obtained from a graph-based approach. In contrast to the online techniques, the pose graph is said to be offline or a "lazy" technique as the optimization processing is triggered by specific constraints.

2.4.2 Pose Graph

The SLAM problem can be defined as a least squares optimization of an error function, and it can be described by a graph model, combining graph theory and probabilistic theory. Such a graph is called a pose graph, where the nodes (or vertices) represent a state variable describing the poses of the cameras, and the edges represent the related constraints between the observations connecting a pair of nodes. Not only the graph representation gives a quite intuitive and compact representation of the problem, but the graph model is also the base for the mathematical optimization, where the purpose is to find the most-likely configuration of the nodes. Here, we present different frameworks:

GraphSLAM In probabilistic robotics [25], beliefs are represented through conditional probability distributions. A belief distribution assigns a probability (or density value) to each possible hypothesis with regards to the true state. Belief distributions are posterior probabilities over state variables conditioned on the available data. The idea behind GraphSLAM [24] is to represent the constraints, defined by the likelihoods of the measurements and motions models, in a graph. The computation of the posterior map is achieved by solving the optimization problem, taking all the constraints into account.

TORO Tree-based netwORk Optimizer [9]. A least square error minimization technique based on a network of relations is applied to find maximum likelihood (ML) maps. This is done by using a variant of gradient descent to minimize the error, by taking steps proportional to the negative of the gradient (or its approximation). It is used in the 3D dense modeling project led by Henry [11].

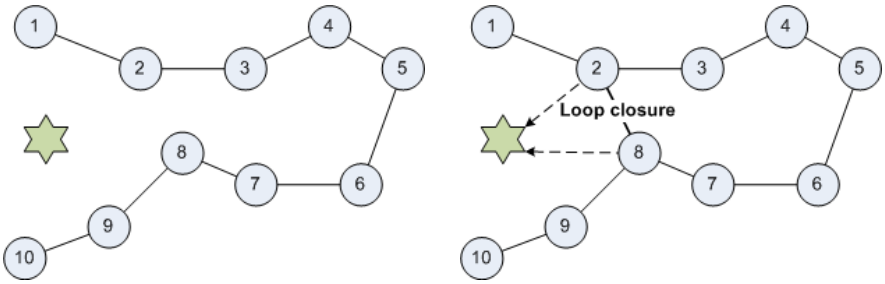
HOG-Man Hierarchical Optimizations on Manifolds for Online 2D and 3D mapping [8]: this approach considers different levels. Solving the problem on lower levels affects only partially the upper levels, and therefore it seems to be well adapted to handle larger and more complex maps. It has been used in the RGBD-6D-SLAM project [5].

g^2o General Framework for Graph Optimization [18], by the same authors of HOG-Man [8]. g^2o is a framework which gives the possibility to redefine precisely the error function to be minimized and how the solver should perform it. Some classes are provided for the most common problems such as 2D/3D SLAM or Bundle Adjustement. The HOG-Man concepts are likely to be integrated in g^2o. This is the method chosen to solve the graph problem in this work.

2.4.3 Loop closure

In order to minimize the error, the graph optimization relies on constraints between the nodes. One interesting event occurs when the robot moves back to a position already visited in the past. This is refered as the *loop closure* detection. To illustrate it, let us take an example. Imagine you are visiting a new city and you have just arrived at your hotel. Without taking a map, you would like to explore the neighborhoods but you still want to keep track of the path you follow with a sparse definition of the environment (mentally, or drawing it on a paper). After walking for a while around a few blocks, you don't know any longer exactly where you are. However, by following a circular path, or a squared path considering standard blocks, you will probably pass again by places you have already visited. By recognizing these places, you are now able to estimate more precisely the path you actually followed and correct the related map.

The loop closure detection relies basically on this idea. Without making an assumption on the path followed by the robot, and simply by keeping the history of the past frames, it is possible to check if the current frame matches with some of the previous ones. If the current observation contains enough similarities with another observation seen in the past, a transformation can be computed between these frames and a new constraint can be inserted from it. This can be repeated several times. With these new constraints, the cumulated error of the graph can then be considerably reduced. Theoretically, all the poses could be linked together this way, if such a relation exists.



(a) (b)

Figure 2.11: Loop closure detection. (a) The poses are linked together by standard edges, each one represents a transformation. (b) The observation of a common point of interest already seen in the past, if significant, can trigger the creation of a new link between two poses that were previously separated in the whole sequence.

2.4.4 Graph optimization

The detection of loop closures leads to new constraints that can be inserted into the graph as new edges, linking the related poses. The optimization of the graph consists in finding the optimal configuration of vertices to respect all the given constraints, as illustrated in figure 2.12.

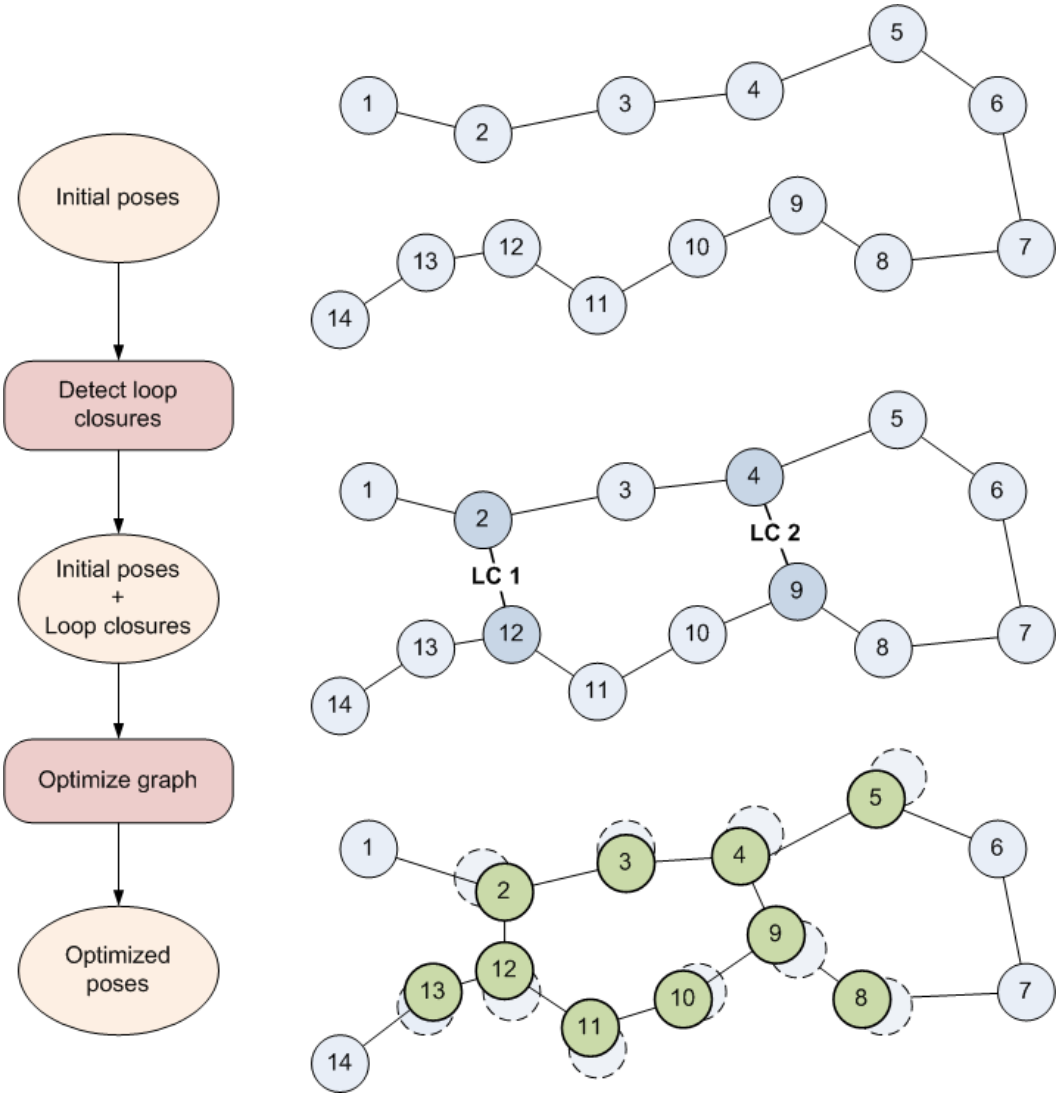


Figure 2.12: Overview of the graph optimization procedure

2.5 Summary

This chapter gives an overview of some of the main concepts used for SLAM and more specifically VSLAM, defining the outlines of a possible approach to solve the problem. The features allow to track some points of interest and can be used to evaluate a unitary move between a couple of frames. By combining them together, an estimation of the trajectory of the camera could be estimated. With the use of a pose graph, and after detecting loop closures, the graph could be optimized in order to reduce the global drift error. From the corrected camera poses, a map can then be built.

While the performance is not the main issue of this work, it is still possible to make a distinction between the operations that can be performed *online* (while the robot is moving) and the operations done *offline* (after a given sequence of moves). While the SLAM addresses both Localization and Mapping, the solution based on the graph implies the detection of loop closures to give a correct result. Therefore, the dual problem of building the map and localizing the robot at the same time is more likely to be done *offline*. This is one of the main constraint resulting of this approach. However, once the map is built, the single operation of localization could be done on the basis of the relevant features that have been used during the initial SLAM operation. This kind of approach could allow the robot to build a map as a first step, only for this purpose. Once the map is built, the localization and further operations could be done, but this mainly depends on the tasks the robot has to fulfill.

Chapter 3

Feature matching

In the context of this work, the goal is to track some keypoints precisely and to evaluate the motion of the camera from the keypoint coordinates. In this chapter, we present how the extraction of features is performed and how the keypoints are matched, in order to compute a rigid transformation between a couple of frames.

3.1 Feature extraction

The first question is the choice of the features, starting from those listed in table 2.1. As the performance is not an issue yet, we will rather consider the well known SIFT and SURF, for which many implementations exist, but also the more recent NARF [22]. The BRIEF descriptor can then be introduced later if necessary, assuming the detector is already given. One important difference is that SIFT and SURF are computed from the RGB data in 2D, while the NARF keypoints are computed from the 3D point cloud, involving both RGB and depth data. Before looking at scale and rotation invariance, one important condition that can easily be verified is the stability of the keypoints if the camera does not move.

We first tested the NARF descriptor provided by the PCL. The keypoints showed to be unstable while the camera was not moving. These variations could be explained by the noise of the depth data of the Kinect, affecting the NARF detector. A better tuning may give more stable results, but apparently this feature was rather designed for range scans such as laser range finders or stereo cameras which are generally more accurate, or less sensible to noise than the Kinect sensor. For this reason, we preferred to work with the 2D image, first using SIFT feature which is widely used in the community and known to be robust. Here, we tested the library developed by Rob Hess [12]. In order to keep only features that are interesting for the next steps of the processing, the features are removed if the depth data on this point is not available (occlusion) or if the depth data is outside a fixed range of depth distance. Here, a maximum range of 5 meters is used.

3.2 Comparison of SIFT and SURF

After testing simple extractions and getting satisfying results with the SIFT feature, the same experiments were repeated using the SURF features. The implementation provided in OpenCV[1] was used. We noticed a considerable gain in performance for a similar number of features. Globally, for a scene requiring about 1000ms with SIFT, the extraction of the SURF feature takes about 250ms, meaning that the time is reduced by a factor of 4 which is significant.

Both algorithms have various parameters that will result in more or less features at the cost of computational time. In particular, for SIFT it concerns the number of scales k computed for each octave (see figure 2.2) and the variance σ of the Gaussian. Here, the default values of the SIFT library [12] are used, which are also the standard values recommended by David Lowe ($k = 3$, $\sigma = 1.6$). Refer to the method [16] for the description of these parameters. For SURF, another parameter is the threshold of the Hessian. Only features with Hessian larger than this value are extracted.

In order to compare both methods, we first tried to find a configuration of parameters giving similar number of features, using a dataset of 6474 frames described in chapter 5 (see also the description of the hardware and software configuration). For an image of 640x480 pixels, the number of octaves set in the SIFT Library is 6. By setting the same number of octaves at 6 and a Hessian threshold of 300 for SURF, we get a very close number of features. Then, we compare with the default settings which implies a number of octaves of 3 and a Hessian threshold of 500. The results are summarized in the table 3.1.

	Avg nb features	Avg time (in ms)	Avg time by feature (in ms)
SIFT 6	745	894	1.407
SURF 6/300	711	345	0.582
SURF 3/500	384	178	0.586

Table 3.1: Comparison of SIFT and SURF features, averaged for 6474 frames. For a similar number of features, SURF is almost 3 times faster than SIFT. For half number of features, there is a gain of factor 5.

[1] http://opencv.willowgarage.com/

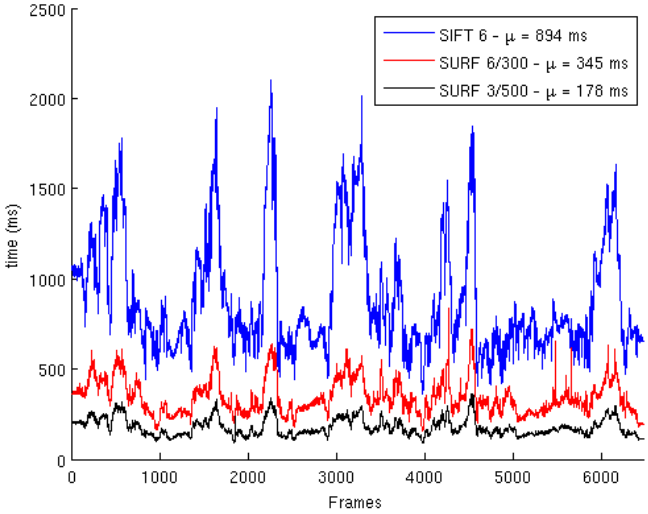


Figure 3.1: Comparison of SIFT and SURF – time for computing features. SURF is about 2.6 times faster than SIFT for a similar number of features. For different SURF parameters, the time is linearly proportional to the number of features (x2 faster for x2 less features).

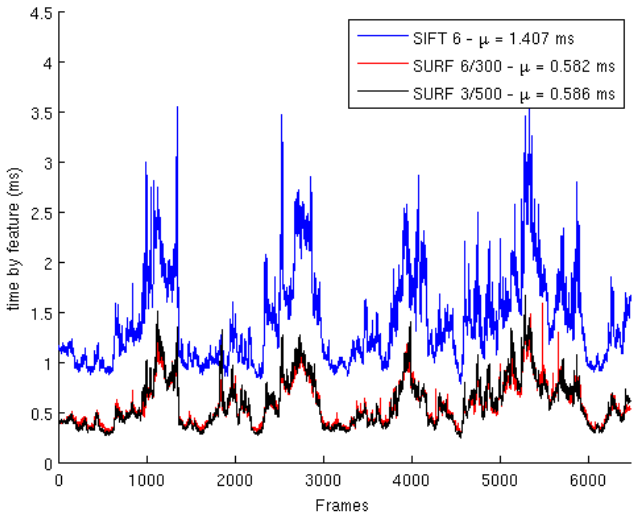


Figure 3.2: Comparison of SIFT and SURF – average time by feature. SURF is fastest and it does not depend on the given parameters.

3.3 Initial Matching

Now that the features can be computed, it is possible to look for their matching between a couple of frames. The initial matches can be found through a nearest neighbor search, using the Euclidean distance computed on the descriptors of these features. The SIFT Library [12] proposes an implementation of the Best-Bin-Search described by Lowe [16], based on kd-tree. For each feature in the target frame, the two nearest neighbors are searched in the source frame. If these neighbors are close enough (with respect to a fixed threshold), then the result is considered to be valid. This initial search is done on the 2D features. To increase robustness, matches are rejected for those keypoints where the ratio of the nearest neighbor distance to the second nearest neighbor distance is greater than a given threshold, set to 0.5 here. After this, the depth information is used to keep the features that are close enough to the camera, as the accuracy of the depth information is not considered reliable enough above a given range (around 5-6 meters).

Figure 3.3: Initial matching of SIFT features

3.4 Estimation of the 3D transformation

From the matching pairs, it is possible to find a rigid transformation binding the two sets of points, i.e. the operation that projects each feature point of a source frame to the corresponding point in the target frame. In our case, this transformation is a perspective projection for 6 degrees of freedom, composed by a rotation matrix and a translation vector in 3 dimensions. This transformation can be written by using a 4x4 matrix and homogeneous coordinates. We can then project a point P where $P = (x, y, z, 1)^T$ simply by applying this transformation matrix to the point:

$$P' = T_{transformation}\, P$$

The points defined by the initial matching pairs need to be converted in 3 dimensions. For this, we need to define a proper coordinate system. To make the next steps easier (graph optimization and scene reconstruction) and avoid further conversions, we keep the same coordinate system for all the work, defined as follows:

$$\begin{cases} x : depth,\ positive\ forward \\ y : height,\ positive\ upwards \\ z : width,\ positive\ to\ the\ right \end{cases}$$

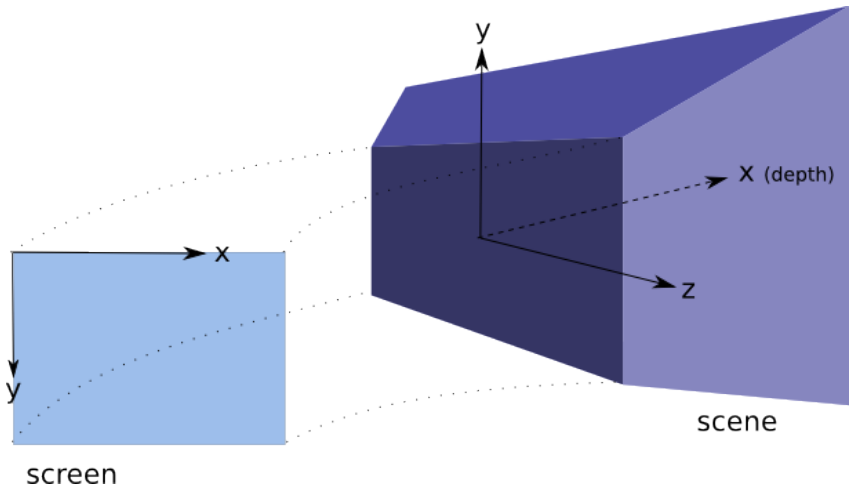


Figure 3.4: The two different coordinate systems, from the "screen" (as seen by the sensors RGB-D) to the 3D scene in the real world.

From the depth value (given in mm by OpenNI) and the screen coordinates (with RGB color and a resolution of 640x480 pixels), it is possible to compute the points coordinates in 3D. Let (u,v) be the point coordinates in pixels, we have then:

$$P(x, y, z) \begin{cases} x = depth \\ y = -(v - 480/2) * depth/focal\_length \\ z = (u - 640/2) * depth/focal\_length \end{cases}$$

Considering the initial matches, the next step is then to find a transformation matrix, that gives a satisfying projection for *most* of these points. Ideally, we could simply compute a transformation by a least square method for all these points, each matching pair defining an equation. The minimum number of pairs required for this would be three, and the more points, the better defined the system would be. However, because of the uncertainty due to the sensory noise and the feature matching by itself, a single point which location is incorrectly estimated could lead to a significant error in the final transformation. Therefore, a better transformation can be found iteratively with the RANSAC method described in the background (2.3.1). In this context, we can precise the algorithm:

Algorithm 2 Find the 3D transformation with RANSAC

Require: initial pairs of 3D points (origin, destination)
  $bestTransform$, $bestInliers \leftarrow \emptyset$
  Define the number of iterations N
  for $iteration = 1$ to N **do**
    $samples \leftarrow$ pickup randomly k pairs (origin, destination)
    Compute $currentTransform$ from $samples$
    $inliers \leftarrow \emptyset$
    for all pairs of points **do**
      $projected_i \leftarrow$ projectPoint($origin_i$, $currentTransform$)
      $error \leftarrow$ computeDistance($projected_i - destination_i$)
      if $error < threshold$ **then**
        $inliers \leftarrow inliers + pair_i$
      end if
    end for
    Count number of inliers and compute mean error
    if $currentTransform$ is valid **then**
      Recompute $currentTransform$ from $inliers$
      if $currentTransform$ better than $bestTransform$ **then**
        $bestTransform \leftarrow currentTransform$
        $bestInliers \leftarrow inliers$
      end if
    end if
  end for
  return $bestTransform$, $bestInliers$

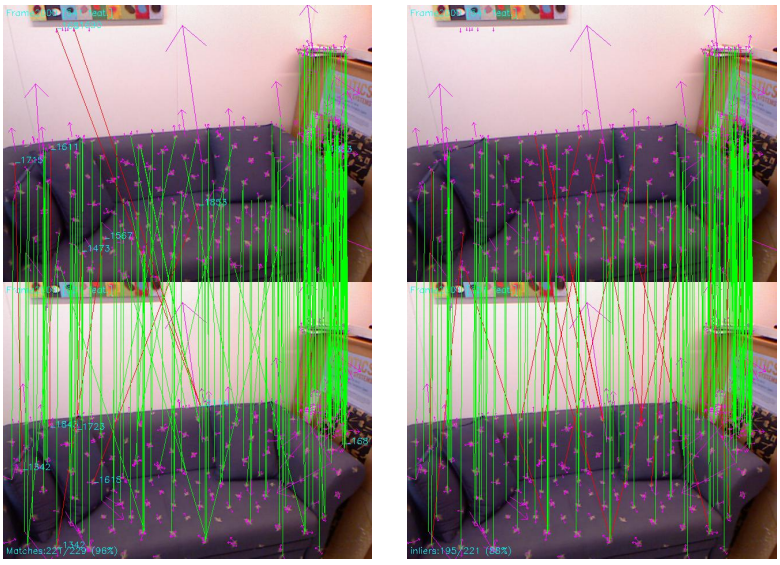
As mentioned in the section 2.3.1, it is possible to estimate the optimal value of N to satisfy a target probability of inliers ratio. In this work we use a fixed value given by parameter of N=20. It may be tuned more accurately with a deeper study of the RANSAC procedure. Generally, if the quantity of features is high, the probability of having an outlier is lower at this step, as most of the mismatches have already been excluded by the initial matching.

To compute a transformation (an hypothesis), only the k samples from the initial matches are used each time. The 3D points are converted to homogeneous coordinates and the transformation is given by solving the corresponding equation from the known constraints which are defined by the chosen points. This is can be done through a Singular Value Decomposition (SVD) of the covariance data. To do this, we use the minimal number of points to solve this equation, k=3.

To evaluate a transformation, each sample taken from the initial matches (source) is then projected according to this transformation. A 3D vector is computed from the difference between the projected point and the real point taken from the matching point (destination). The error is the norm of this vector.

3.5 Analysis

Some preliminary experiments were conducted to determine how to find a satisfying transformation between a couple of frames. In this context, the unitary move between two frames should be generally small if we consider a limited motion of the camera (by amplitude and speed). The larger the move, the more difficult it becomes. To do this, a program was written to perform the feature extraction with the initial matching and the RANSAC iterations, taking a couple of RGB-D frames as an input.



(a) Initial matching (b) After RANSAC

Figure 3.5: Matching of SIFT features: (a) initial matching from KNN search (b) after running RANSAC. Some of the initial matches drawn in diagonal (green) in the left figure are excluded and become outliers (red) after RANSAC.

Characterizing a transformation There are at least 3 parameters:

- Mean error: the norm of the error vector. Lower is better.
- Number of inliers: the absolute number of inliers. Higher is better.
- Ratio of inliers: the relative number of inliers with respect to the initial matches. Higher is better.

The main difficulty is to find the best balance among these 3 values, by putting some thresholds. Setting too high constraints would lead to the impossibility to find a transformation satisfying all the criteria. This is true especially when there are not enough features detected. Setting too low constraints helps to find a transformation even when there are less features or the measurements are noisy, but the resulting transformation will be less precise. The main risk here is to set too loose constraints so the inliers would include some mismatches. If the criteria are too permissive, this would result in an invalid model, leading to incorrect associations, as shown in figure 3.6.

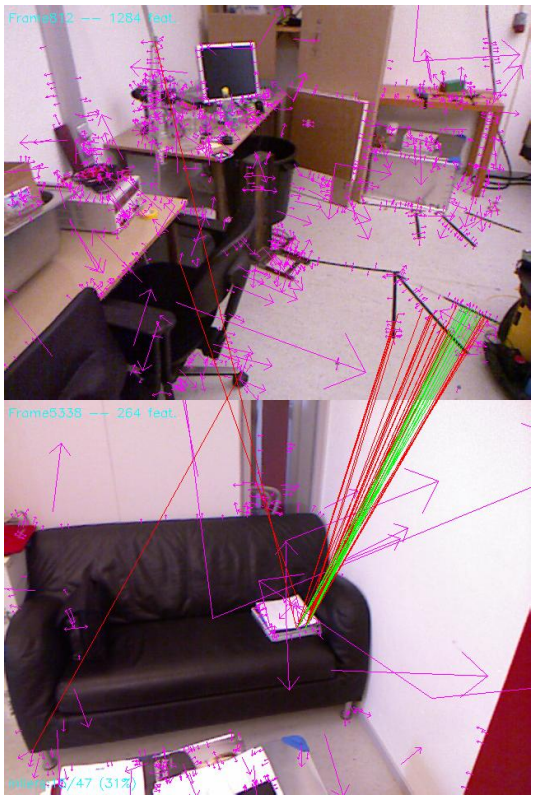


Figure 3.6: Example of an incorrect model. The matches are incorrect, as the scenes and objects are completely different. But the green lines are considered to be inliers, due to the similarity of the dark lines. Here, the threshold defining the minimum ratio of inliers with respect to the initial matches is too low.

Quality of a transformation A good transformation should be valid for most of the given points. But this definition is not enough, as only considering the number of inliers is not necessarily the best choice. Another criterion could be the spread of the inliers. If many inliers are concentrated on a small area, they don't give much added value with respect to the global transformation of the scene. A better transformation may be found including more distant points, if their projection error is just slightly above the threshold. This could be measured by computing the mean position of the inliers and, from this, the standard deviation, or more simply the variance of the inliers.

Let N be the number of inliers and p_i be the i-th inlier vector. We can then compute the mean vector μ and the variance σ^2 with their standard definitions:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} p_i \qquad \sigma^2 = \frac{1}{N}\sum_{i=1}^{N} (p_i - \mu)^2$$

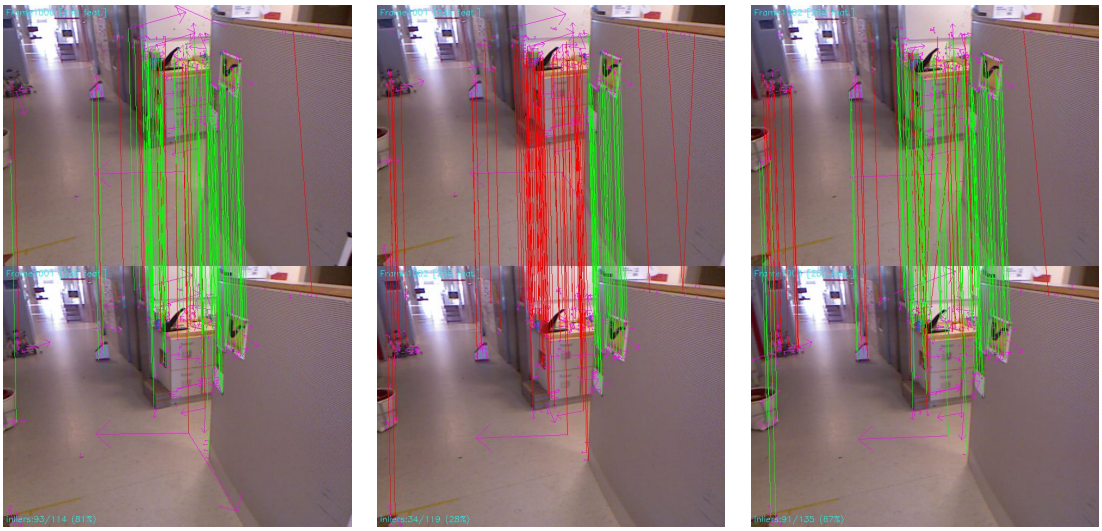


Figure 3.7: Sequence showing different distributions of the inliers. Note how the inliers are grouped in the case shown in the middle.

inliers/matches	ratio	$\sigma^2(2D)$	$\sigma^2(3D)$
93/114	81%	0.345409	1.2232
34/119	28%	0.0203726	0.0519737
91/135	67%	0.476902	1.42809

Table 3.2: Ratio of inliers shown in figure 3.7, variance of the keypoints in 2D (without depth information) and 3D.

The variance may be used to detect these situations. Instead of using the ratio of inliers as the criterion to measure the quality of the model, the variance could be used, not only as an evaluation of the goodness ("score") but also as a criterion for the selection of the k elements of each sample. For example, the initial elements would have above a minimal distance from the mean, set by a threshold. This concept could not be developed in the given time, but it could be studied in future works.

3.6 Summary

In this chapter, we presented a method to compute a rigid transformation from a couple of frames given their RGB-D data. This transformation represents the motion of the camera between two consecutive frames, for 6 degrees of freedom. This was done through the following steps:

1. first, the SIFT/SURF features are extracted from each RGB frame (using only 2D);
2. an initial matching is performed with the use of a kd-tree, and the depth information is integrated to compute the feature positions in 3D;
3. from this set of pairs of features, a transformation is computed by running a RANSAC algorithm.

The figure 3.8 illustrates how the RGB and depth input data can be used to compute the sequence of 3D transformations, and subsequently, the initial estimation of the poses.

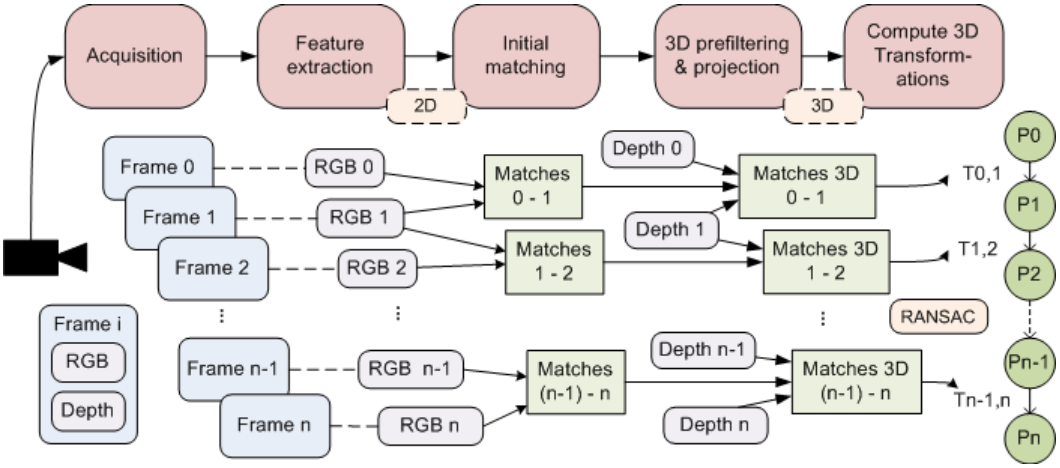


Figure 3.8: Processing of the RGB-D data

Chapter 4

Building a map

In the previous chapter, we saw how to determine a rigid transformation between a couple of frames. From each single transformation, an estimation of the poses of the camera can be computed. Now we can combine them to proceed with a sequence of frames. Each pose is then inserted into the graph by converting its representation from the camera matrix into a graph node. From the loop closures, new constraints can be inserted in the graph. The graph can then be optimized to minimize an error function, and the poses are updated according to the new vertices given by the graph after optimization, reducing the global drift.

4.1 Estimating the poses

Knowing the initial pose, the first step is to determine an estimation of any pose after a succession of transformations. Considering a finite sequence of frames $[frame_0; frame_N]$, let P_k be the pose at rank $k \in [0; N]$. For 6DOF, composed by 3 axis of rotation and translation, it can be represented by a 4x4 matrix with homogeneous coordinates, where R is a 3x3 rotation matrix and t a translation vector:

$$P_k = \begin{bmatrix} & & & t_x \\ & R & & t_y \\ & & & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For $i > 0$, we have the transformation T^i_{i-1} that binds the position P_{i-1} to the position P_i. Each transformation T can also be represented by a similar 4x4 matrix. If P_0 determines the initial position, we can then compute the position P_i by combining all the transformations like:

$$P_k = \prod_{i=k}^{1} T^i_{i-1} \, P_0 \tag{4.1}$$

As the matrix product is not commutative, it is essential to follow the correct order when multiplying the matrices. For example, for the pose P_3 we have:

$$P_3 = T_2^3 \, T_1^2 \, T_0^1 \, P_0$$

Generally, the initial position is defined by the initial orientation of the camera for each axis, stored in the initial rotation matrix R_0, and the initial translation vector $t_0 = (x_0, y_0, z_0)^T$, as follows:

$$P_0 = \begin{bmatrix} & & & x_0 \\ & R_0 & & y_0 \\ & & & z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

As an arbitrary choice, we can define the initial position to be at the origin of the coordinate system, which is given by the identity matrix I_4.

$$P_0 = I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

4.2 Initializing the graph

The equation 4.1 gives an estimation of the pose that can be inserted into the g^2o graph. However, a sequence of video frames can contain an important number of similar poses, especially if the robot slows down, or even stops. Inserting a node for each transformation would lead to many redundant poses. It makes good sense to insert a new pose only after a significant change from a given pose. To distinguish them among all the others that can potentially be inserted in the graph, these poses are called the *keyposes*.

A simple solution is to define a keypose only after having achieved a move, with a given amplitude defined by a threshold. By cumulating the unitary moves between each couple of frames in the sequence, the creation of a new key pose is triggered once a given distance in translation or a given angle in rotation has been reached, with respect to the last key pose. The thresholds are set by parameters, here 0.1m for the distance and 15°for the rotation (both values consider the variations on the 3 axis cumulated together). Another solution would be to compare the points of interests between the frames and trigger the creation of a new keypose when the number of points in common falls below a given threshold, but it is not given that the absolute number of points is representative enough to describe the amplitude of the move.

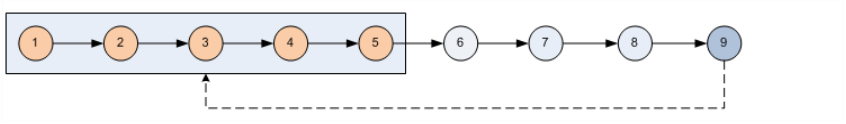
For each keypose, a node is inserted in the graph with an edge linking the new keypose with the previous one. A standard edge represents the rigid transformation between the two keyposes. Between each keypose, they may be a more than a couple of frames. To compute the corresponding transformation, the resulting matrix is found by multiplying all the intermediate matrices for each couple of frames.

4.3 Loop closures

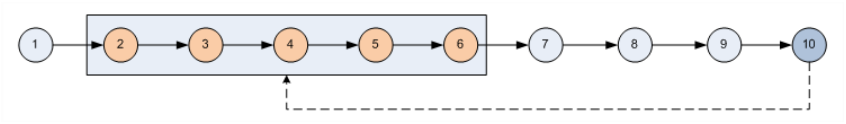
The detection of loop closures can be translated into additional constraints in the graph, as illustrated in figure 2.12. Each time a loop closure is triggered between two keyposes, a new edge is inserted into the graph. Here, the detection of the loop closure is done by computing a transformation between the two frames, as described in section 3.4. If there are enough inliers to make the transformation valid, a loop closure can be inserted.

Following the order of the sequence, the detection of a loop closure can be done by comparing the most recent frames with previous ones. The naive approach would be to check for the whole set of frames, comparing the current frame with all the previous frames. However, this can be highly time consuming, as the time necessary for the current frame would grow fast. For example, for 4 frames, there would be 6 checks to perform by looking in the past (respectively 1,2,3 checks for each frame). For N frames, there would be $(N-1) * N/2$ checks to perform. Clearly, this can become an issue if the check is not fast enough. First, it can be optimized by storing the past features into a memory buffer. A preliminary check exclusively done on the RGB data, for example with color histograms, could also be used to discard most of the negative candidates. Then, a more accurate verification would involve the features for the remaining candidates.

In this work, for reason of performance, it was necessary to reduce the list of candidates to check. Without making any assumption on the past frames, the first idea is to define a *sliding window* with a fixed size of k frames in the past. The loop closure is of interest when it closes a loop with frames that are sufficiently distant in the temporal sequence, as the goal is to reduce the drift over time. For this reason, from a given frame, it is more appropriate to check with older frames in the past rather than most recent ones. By defining a number of excluded frames and the size of the sliding window, an efficient method can be implemented.



(a)



(b)

Figure 4.1: Illustration of a sliding window of 5 frames, where the last 3 frames are ignored. (a) For the current frame n.9, the frames 1-5 are checked. (b) For the next frame n.10, the window slides one step forward. The frames 2-6 are checked.

However, these parameters are hard to define without a prior knowledge of the followed path. To extend the search, but still limiting the number of checks, these frames could be selected randomly in a much larger window, first assuming a constant probability density function. But the goal is to select the past frames that are more likely to match with the current frame, giving a higher priority to the oldest frames. This could be done defining a probability density function with a higher probability for the oldest frames. This probability could also be defined considering the knowledge of the context, in terms of features, or in terms of estimated position, preselecting the candidates with the highest likelihood.

Here, we limited the search by using a candidate list of N frames, ignoring the most recent frames (the size can be set by a parameter), and a probability density function giving priority to the oldest frames. To select the N candidate frames, the function giving the probability to select a candidate can be defined by $p(i) = i/S$, where $S = (N + 1) * N/2$. For a sliding window of 3 frames, the first frame would have the probability 3/6, the second 2/6 and the third 1/6. These values can then be normalized for a total probability of 1.

To improve the quality of the loop closure detection, once a candidate frame is found, the check is extended to its neighbors, looking for a better transformation. The ratio of inliers is used to measure the quality of the transformations and to compare them. We consider only a forward search, meaning that the following neighbors will be checked, as illustrated in figure 4.2.

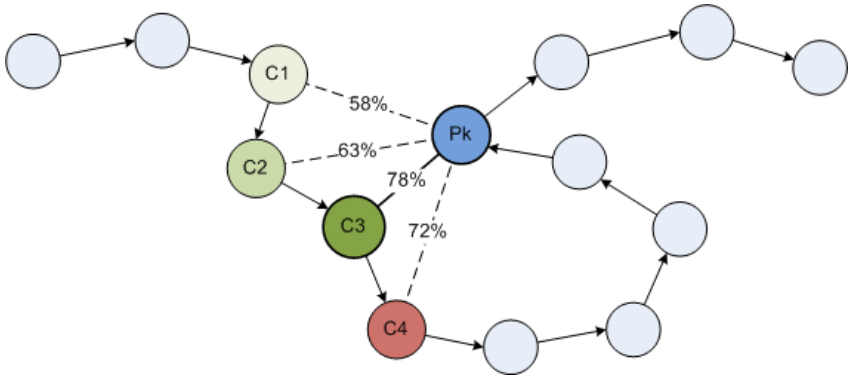
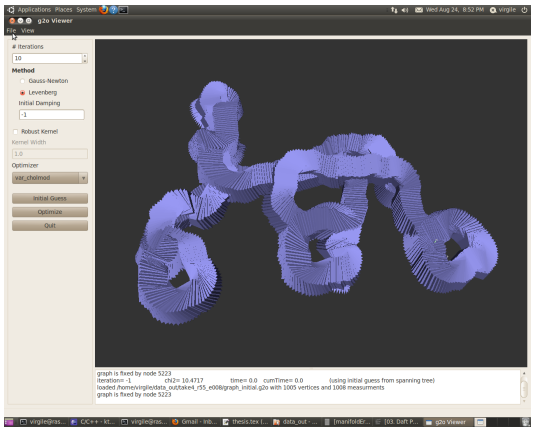


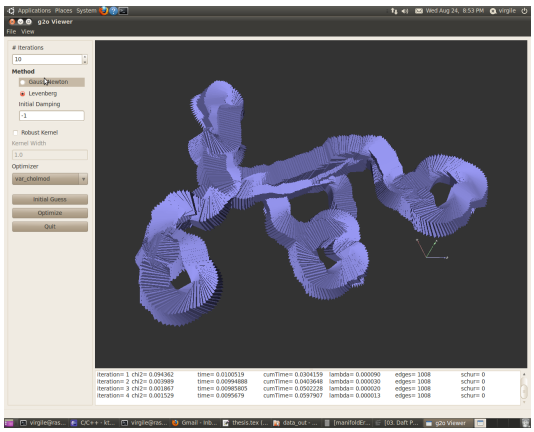
Figure 4.2: Loop closure with forward search for the pose P_k. First, a valid transformation is found with candidate C1. The check is then performed with the next frame C2, giving a better transformation. The search continues forward, and founds a better transformation with C3. Finally, the quality of the transformation with C4 is lower, so the process stops. The loop closure is inserted between P_k and C3.

4.4 Optimizing the graph

Once the graph has been initialized with the poses and the constraints from the loop closures, it can be optimized. Various methods are available in g^2o, both for the optimization process and the solving problem. The method used here is Levenberg-Marquardt with a linear Cholmod solver. The optimization is done with a predefined number of steps. Once the graph has been optimized, the vertices are corrected and the new estimations of the camera poses can be extracted. Similarly to the insertion, the inverse operation is done to convert the information from the graph vertices to 4x4 matrices.



(a) initial graph



(b) optimized graph

Figure 4.3: Visualization of the graph with the g2oviewer, (a) non optimized, (b) after 10 iterations, for a map of 4 rooms (experiments are described in chapter 5).

4.5 Scene reconstruction

Finally, once the camera poses are determined with a good belief, the reconstruction of the scene can be performed. For each frame, a 3D point cloud can be generated from the RGB and depth data, providing the colour information for each point. However, in order to reconstruct the whole scene, they have to be combined together from a unique point of view. This process is called the point cloud *registration*. Once the camera positions are known, each point cloud is transformed by projecting all its points according to the corresponding camera position, given by the equation 4.1. This is done relatively to the first pose, which is predefined. The transformed point clouds are then concatenated together to build the scene.

This method is simple, the major issues are that it leads to duplicate points, and it does not take variances of illumination into account. Some considerations about memory are described in the experiments, in section 5.5. However, the point cloud is not necessarily the best representation of the scene. For many applications, it would be preferable to define some dense surfaces, such as the *surfels* presented in [17] and used in the work of Henry [11], but this would require further study and processing and it is out of the scope of this work.

4.6 Summary

In this chapter, we presented how the poses can be estimated, first by cumulating each transformation between a couple of frames. Through the use of a pose graph and the loop closures, their positions can be corrected over time. Finally, the map can be built from a scene reconstruction, by registration of the point clouds with the given positions of the cameras.

Chapter 5

Experiments

After studying the feature extraction and matching, the graph optimization and the reconstruction, some experiments could be done. A software was developed, which goal is to acquire data from the Kinect, compute the 6DOF camera poses representing the trajectory and orientation of the robot, and produce a 3D map. The following sections describes how the input data was acquired, and how the output is generated. The next sections present the maps obtained from different datasets, first from KTH, for different sizes (1 room, 2 and 4 rooms with corridor). The last section shows the results obtained with data from other universities.

5.1 System overview

The program can perform the following tasks:

- acquires a stream of RGB-D data from the Kinect;
- performs extraction and matching of SIFT/SURF features;
- computes the relative transformations with RANSAC;
- computes the initial poses and translates them into g^2o nodes and edges;
- detects the loop closures and inserts the corresponding edges into the graph;
- optimizes the graph with g^2o and extracts the updated camera poses;
- reconstructs the global scene by generating a point cloud datafile (*.pcd).

The sequence of actions, which have been described in the previous chapters, can be summarized in the figure 5.1. Note: the use of RGB and depth data is more detailed in figure 3.8.

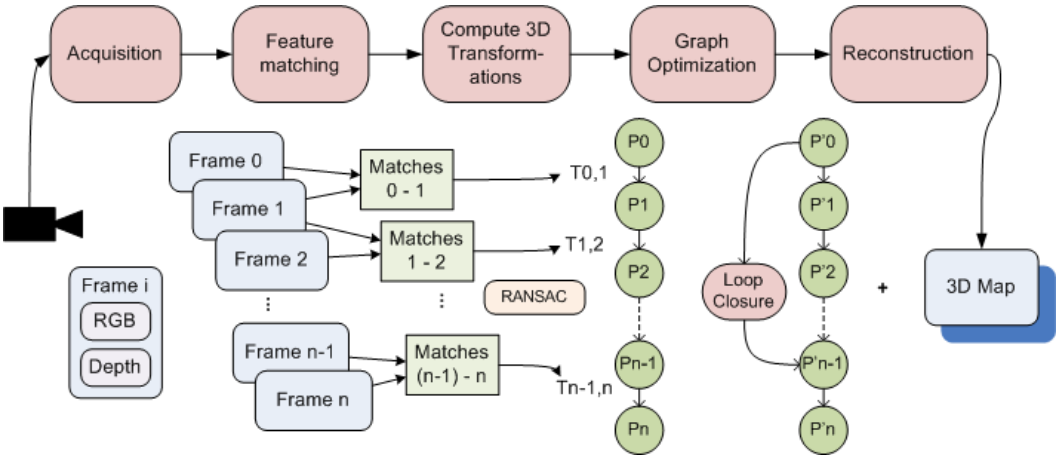


Figure 5.1: Overview of the system

In order to repeat the experiments with different parameters, the RGB-D stream is saved and the program is able to replay a given sequence from a list of files. Two types of experiments were conducted:

- on the fly, by moving the Kinect sensor by hand, with a feedback in real time;
- offline, by recording a data stream with a mobile robot, and processing it afterwards.

In the first case, the program performs the feature extraction and computes the transformations in real time. When the motion is too fast, or when the number of features is too low, the synchronization can be lost if no valid transformation is found between the last two frames. The sequence is then suspended and a message is displayed, so the user can move the camera back and lock again with the last valid frame, to resume the sequence. The detection of the loop closures, the graph optimization, and the reconstruction are done in a next step. Similarly, all the processing could have been done on the robot as well, but this would require further installations, and here the goal was to generate a set of reference data, as described in the section 5.2.

The main program was executed on a PC equiped with an Intel Core™ i3–540 CPU (3.06 GHz, 4MB cache) and 3GB of RAM, running on Linux Ubuntu 10.04 (kernel 2.6.35.7). When the acquisition and matching are done on the fly, the system is able to process about 3-4 frames per second using SURF, which is already enough to let the user walk slowly in a room, with a real time feedback. Most of the time is spent for the feature extraction (100-200ms). Some possible improvements are given in the final chapter with the conclusions.

5.2 Data acquisition

To build some reusable datasets, the experiments were carried out at CVAP (KTH) on a Pioneer III wheeled robot, Dora the explorer (see figure 5.2). The Kinect camera is mounted at 1.4 m above the floor, and the robot is also equiped with a Hokuyo URG laser scanner. Each frame saved by the Kinect leads to a couple of RGB and depth files taken at the same time. Additionally, the data from the laser scans and the odometry were saved, and then used to build a reference path, as shown in figure 5.10. The acquisition of the raw data by the robot is not part of the main program built in this work, but with the tools that were developed at CVAP. Clearly, this implies the format of the RGB-D data is specified.

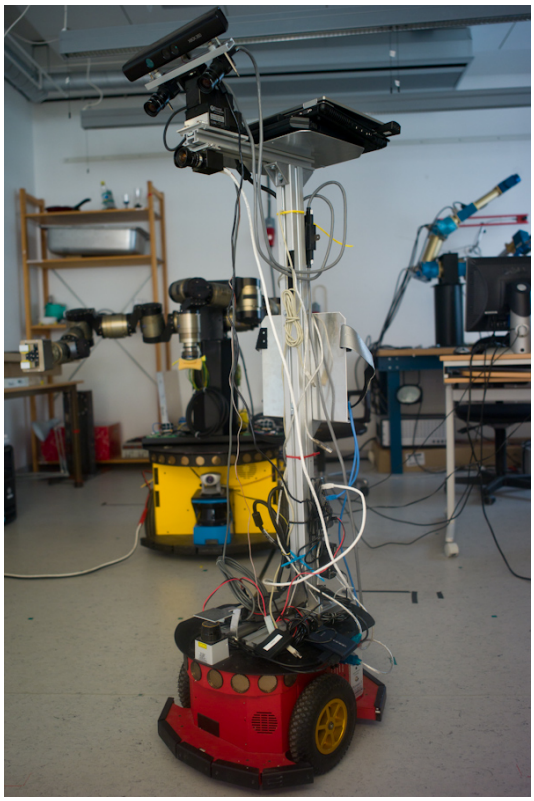


Figure 5.2: Dora, the explorer

At KTH, the main dataset took place in environment with 4 different rooms connected by a corridor (see map in figure 5.10, page 49), resulting in 6474 frames (RGB-D datafiles) with several possibilities of loop closures. Other universities participating on common projects could provide data at the same format with similar robots. This allowed to test the system with data provided by the universities of Birmingham and Ljubljana.

University	Dataset	Rooms	Frames
KTH	7th floor	1	3082
KTH	7th floor	2	2697
KTH	7th floor	4	6474
Birmingham	Lab3	1	1342
Birmingham	Robotic Lab	1	2878
Ljubljana	Office1	1	2088
Ljubljana	Office2and3	2	5687

Table 5.1: Description of the different datasets. Note the KTH set of 1 room contains more frames than for 2 rooms, due to the robot moving at a slower speed.

5.3 Software implementation

The code is available as an open source project[1]. It is released with a LGPL v3 license. The number of dependencies has been limited to a reasonable set of open-source libraries. Unlike the RGBD-6D-SLAM project [5], it is not based on ROS framework, and PCL is mainly used for the reconstruction at the final step. The main dependencies are the following:

- g2o: the graph optimization
- OpenNI: to acquire the Kinect data
- OpenCV: open source library used to visualize the intermediate results with bitmaps (frame matching) and compute the SURF features
- SIFT Library by Rob Hess [12]: for the SIFT extraction and initial matching
- Eigen3: for the geometric support with transformations and matrices
- Point Cloud Libary [21], standalone distribution (cminpack, Flann, Eigen3, OpenNI): for the transformations and the export to point cloud datafiles
- Boost: support library, used here to access the filesystem

[1] http://code.google.com/p/kth-rgbd/

5.4 User interface

To use the program in real time, a simple graphical user interface was developed. The main window displays two frames with their features. The upper frame is the current frame, and the lower frame is the last valid frame. If the sequence is out of synchronization, meaning that no valid transformation between the last two frames, this layout allows the user to move back by comparing visually the current frame with the last valid one. Then, the sequence is resumed automatically as soon as there is a valid transformation. The quality of the transformation is also displayed, that corresponds to the ratio of inliers.

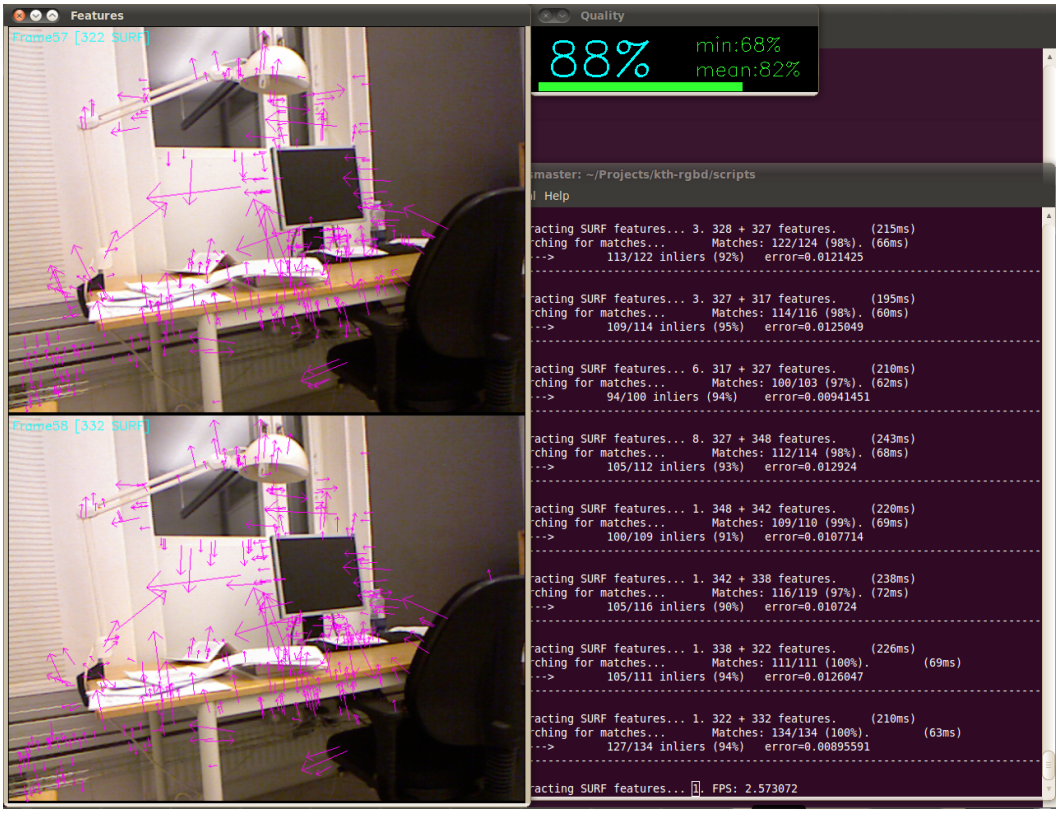


Figure 5.3: user interface, displaying the current and last frames with their features

5.5 Data output

As a result, the program exports the positions of the cameras, and the map represented by a point cloud (PCL). For reasons of performance, it is necessary to control the memory storage by limitating the size of the final point cloud. Theoretically, each point cloud can be composed of $640 \times 480 = 307,200$ points. Practically, the effective number of points is lower, as the depth information is not available for each point (due mainly to the range limit and to the occlusions considering the difference of point of view between the IR and RGB sensors). A standard point cloud is composed of about 200,000 points. For each pixel, 24 bits are used for the positions (x,y,z) and 24 bits for the colors R,G,B (3×8 bits), padded to 8 bytes for alignment in memory. Therefore, a colored cloud point occupies around $200k \times 8$ = 1.6MB in memory. For a scope of visualization, it is reasonable to reduce the amount of information. Each point cloud is first subsampled simply by removing points taken randomly with a given probability (according to a fixed ratio, set as a parameter depending on the global size of the scene).

Generally, only one output file is produced but this can still lead to a big file in case of a large map. In this case, the total number of points can be important and a too high rate of subsampling would result in a low quality map. By setting a threshold for the size of the final point cloud file, it can be split in several subfiles each time the threshold is reached during the reconstruction of the global map. The result is then divided in different files. This allows to keep a low subsampling rate, and still to visualize a portion of the global scene with a good definition. For the visualization of the point clouds, the standard viewer provided in PCL is used.

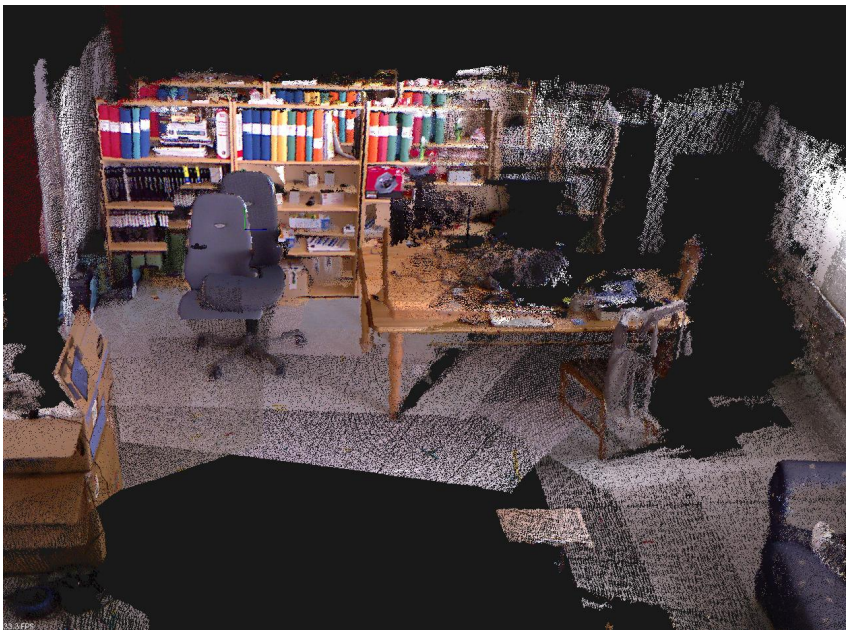
5.6 Map at CVAP – one room

This map is built from a sequence containing one room (see table 5.1), the graph is optimized with one loop closure, which is triggered after having gone through the room and back, close to the initial position.

(a) from initial graph

(b) from optimized graph

Figure 5.4: Map at CVAP with 1 room – (a) Without graph optimization. (b) With loop closure. Note how the corner of the table and the chair appears to be doubled in the top figure, and how it is corrected in the bottom figure.



(a) from initial graph

(b) from optimized graph

Figure 5.5: Map with 1 room, details of the chair – (a) Without graph optimization. (b) With loop closure. The reduction of the drift is clearly noticeable.

5.7 Map at CVAP – two rooms and corridor

This map is built from a sequence containing two rooms separated by a corridor (see table 5.1). The graph is optimized with two loop closures, the first one with an internal loop in the living room, and the other one at the end of the sequence, after coming back in the first room close to the initial position.

Figure 5.6: Map with 2 rooms, optimized with two loop closures. The rooms are clearly displayed and correctly oriented. The main issue concerns the dividing wall in the corridor.

5.8 Map at CVAP – four rooms and corridor

This map is built from a sequence containing four rooms separated by a corridor (see table 5.1), first without graph optimization.

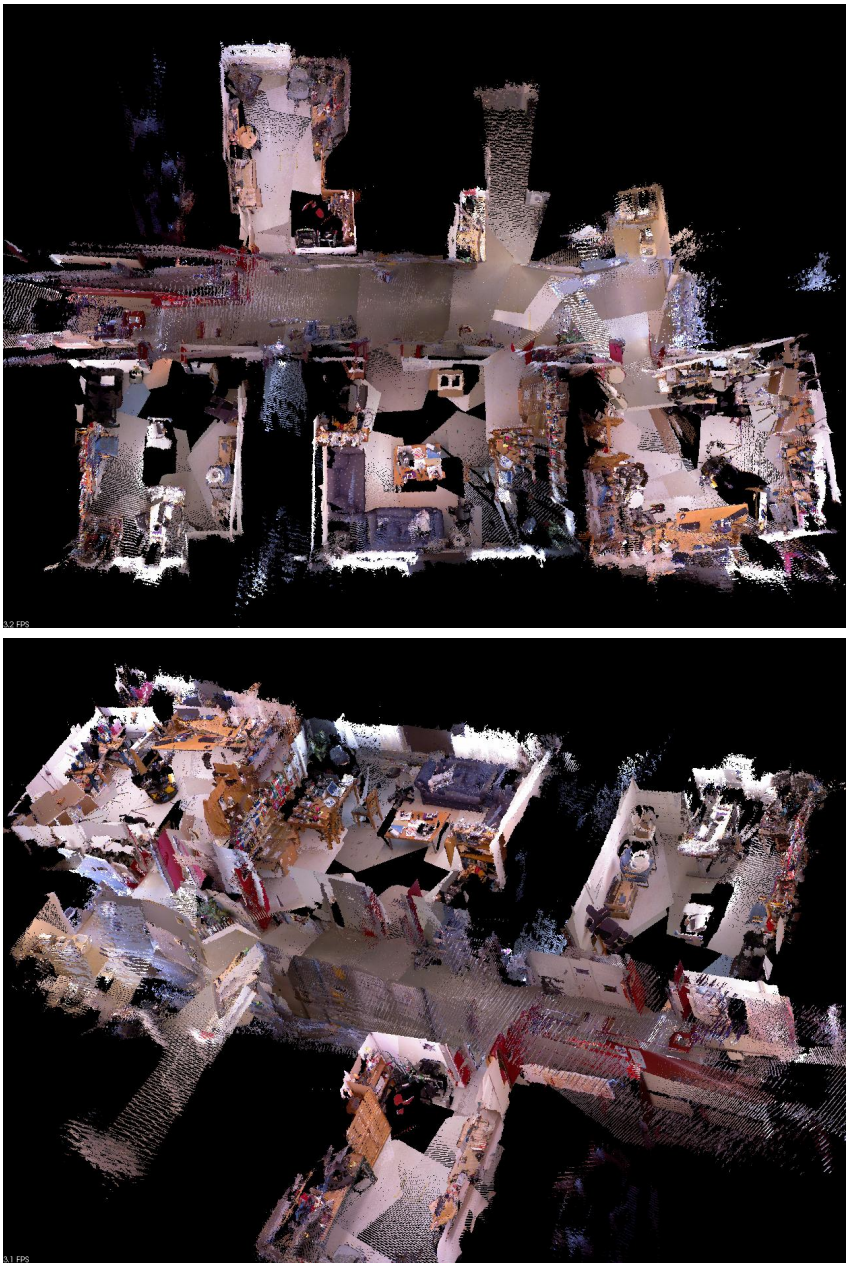


Figure 5.7: Map with 4 rooms and corridor, not optimized. The main issue that can be noticed at this scale is the relative orientation of the first room (first figure, bottom right), which looks misaligned, and its desk appears to be doubled.

For the following map, the graph is optimized with four loop closures, triggered with a loop in three rooms room and the other one at the end of the sequence, after coming back in the first room close to the initial position.

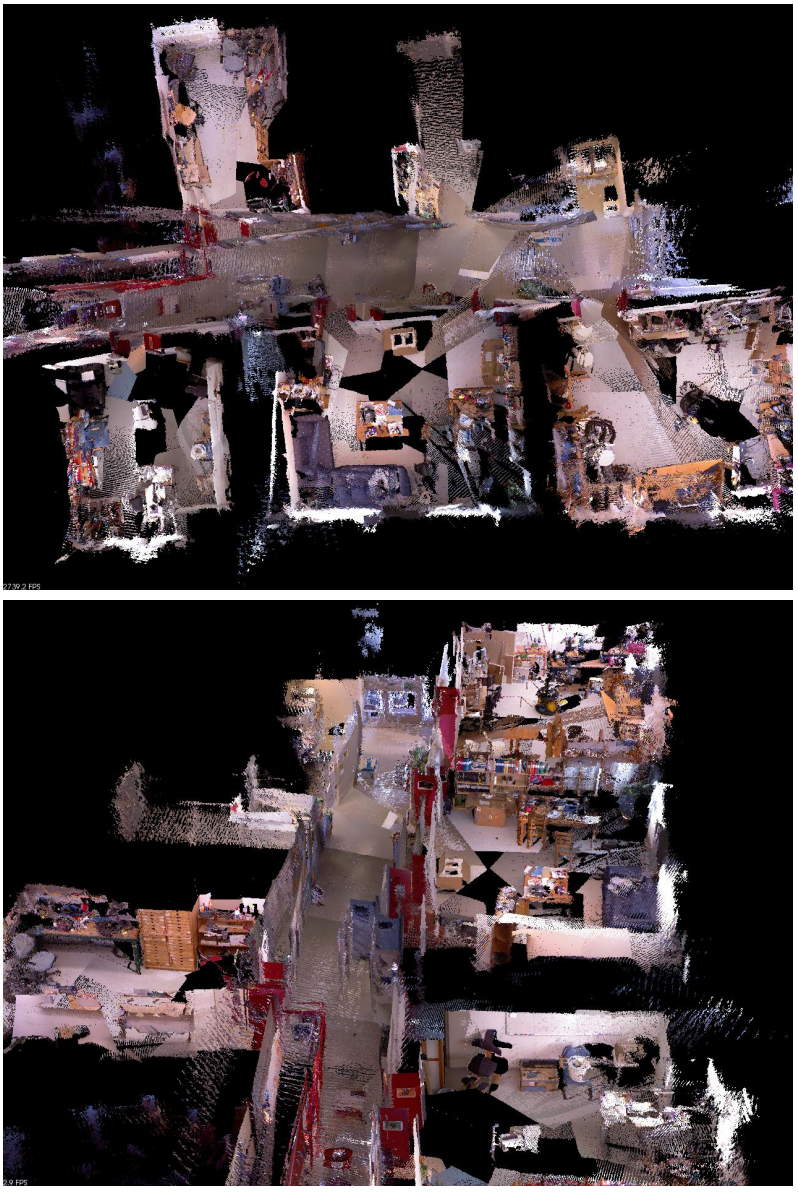
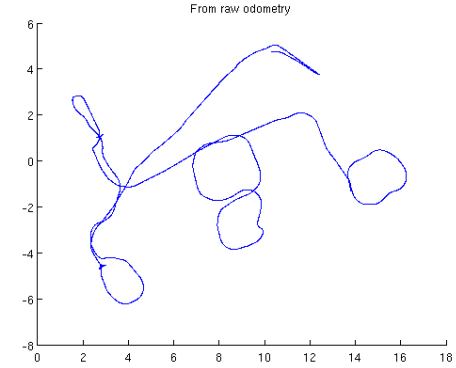
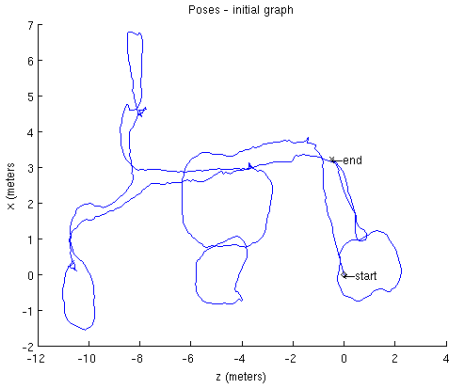


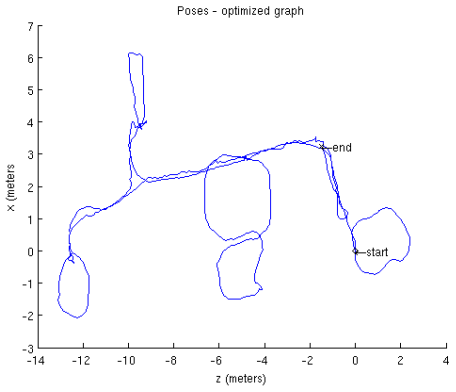
Figure 5.8: Map with 4 rooms and corridor, optimized with several loop closures. The alignment of the first room looks better (first figure, bottom right), but the living room (bottom center) shows some misaligned frames. This is a trade-off with respect to the global error.



(a)



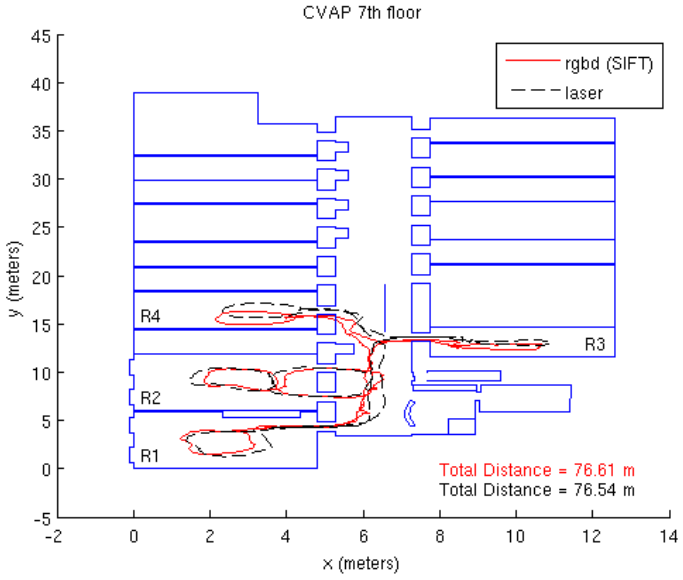
(b)



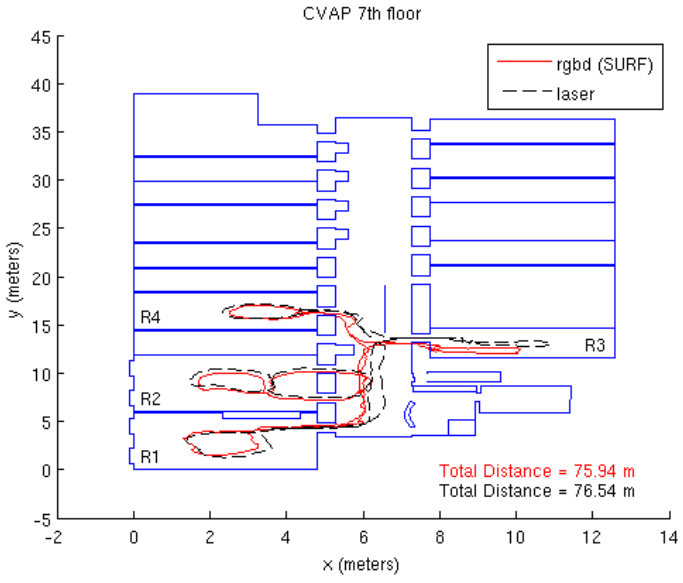
(c)

Figure 5.9: Estimation of the poses – (a) From standard odometry, raw data. Note the drift, as the start and end points should be pretty close. (b) With this system, initial graph. (c) With the optimized graph. There is a slight variation.

To analyze the quality of the localization, we compared the resulting path with one close to the ground truth, generated by a SLAM system developed at CVAP, based on EKF using the laser scan data [7].



(a)



(b)

Figure 5.10: Trajectory overlayed on the reference map of the 7th floor, using (a) SIFT features. (b) SURF features. In both cases, the drift is noticeable, but the results are relatively close. The total distance gives an indication of the accuracy.

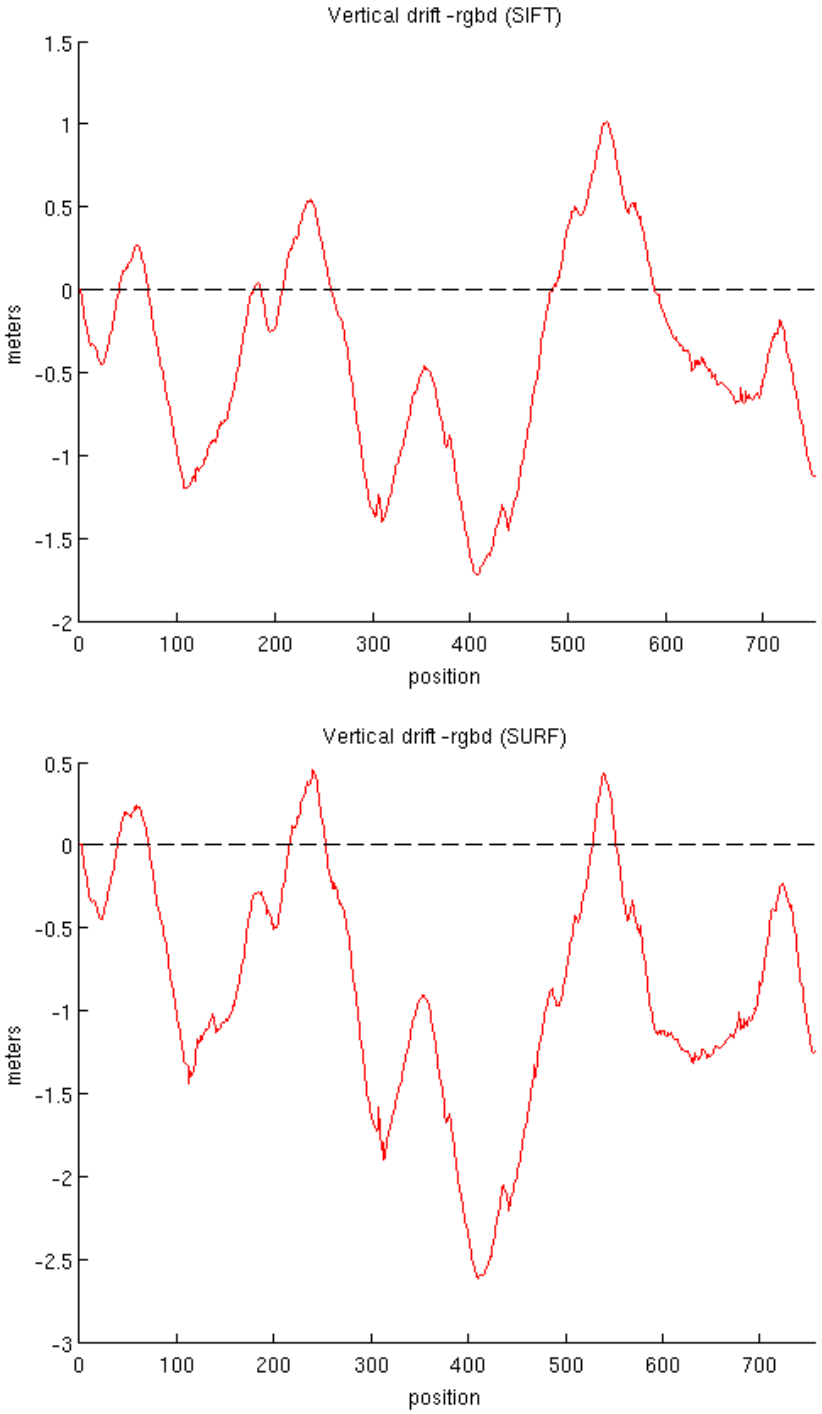


Figure 5.11: Analysis of the vertical drift. The Y-coordinates have been shifted to the origin for the starting position. Considering the camera is fixed on a support, the value should stay close to zero. This issue needs to be further investigated.

5.9 Map from other universities

For this map, the *Lab3* dataset provided by the University of Birmingham was used. No particular issues were encountered.

Figure 5.12: Birmingham Lab3

For this map, the *Robotic Lab* dataset provided by the University of Birmingham was used. An issue was raised due to the people moving during the passing of the robot, leading to some inaccuracies in the transformations. The system is not able to handle such situations, affecting the feature matching, and therefore the overall quality of the map.

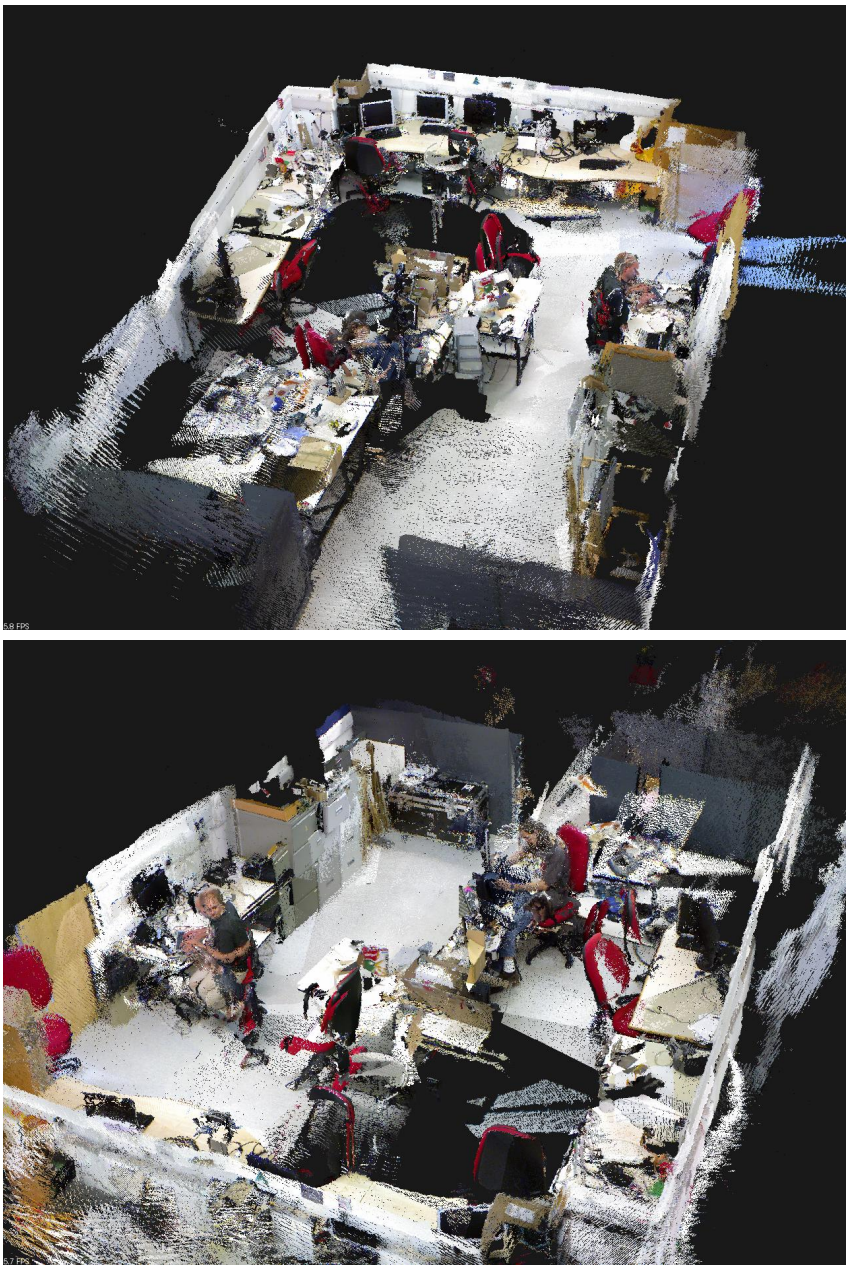


Figure 5.13: Birmingham Robotic Lab

For this map, the *Office1* dataset provided by the University of Ljubljana was used. It could only be processed partially, the sequence was interrupted for a lack of features. This is one intrinsic limitation of the method presented in this work.

(a)

(b)

Figure 5.14: Ljubljana office1 – (a) Map without closure loop. (b) Not enough features could be extracted at this point. This occurs typically when flat surfaces are predominant in the scene.

For this map, the *Office2and3* dataset provided by the University of Ljubljana was used. As for the previous dataset, it could only be processed partially, the sequence was interrupted for a lack of features.

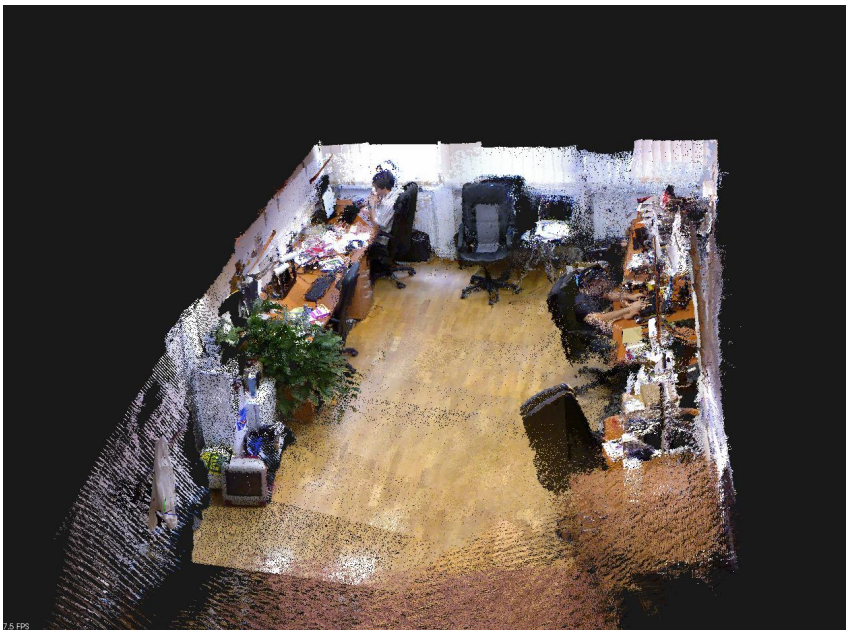


Figure 5.15: Ljubljana office2 – map without closure loop

5.10 Summary

These experiments show that 3D maps can be built with this system and illustrate the interest of the loop closure to get a good correction. Though the results are satisfying for small maps, there can still be a noticeable drift as the size of the map grows. Here, some misalignments can be perceived with 4 rooms. The most obvious limitations were also encountered, showing that the process can be interrupted as soon as there are not enough visual features to detect a valid transformation, and how moving elements can alter the final result.

Chapter 6

Conclusions and Future Works

With this work, we could have a good insight into the general problem of SLAM, and more specifically of VSLAM with feature extraction and matching. The approach can be described by the following steps:

1. Feature extraction: SIFT/SURF extraction in 2D, preselection with depth;
2. Feature matching: nearest neighbors on the feature space, using kd-trees;
3. Transformation: computed with RANSAC on the 3D keypoints;
4. Graph optimization: based on the g^2o framework, a better graph can be computed with new constraints by detecting loop closures;
5. Scene reconstruction: registration of the points clouds accordingly to the estimation of the keyposes, and concatenate them to build the global map.

Given this, it is now possible to imagine some improvements of the current work, both in terms of quality of the results and overall performance.

6.1 Quality improvements

- Finding a better transformation: the RANSAC method returns a rigid transformation that is not necessarily the best. The parameters may be tuned in a more efficient way to give more accurate results, or defined with a variant like MLESAC [26]. The tuning also concerns the initial matching done with the SIFT/SURF features. The ICP [27] method should give a refined solution. The output of the RANSAC loop could be used as an initial guess for ICP.
- Graph optimization: it could be improved by formulating more accurately the optimization problem and refining the minimization method (possibly with g^2o by deriving new classes and defining appropriate error functions), and also by exploiting the possibilities of modularity, for example with hierarchical levels corresponding to different scales for bigger maps. New criteria could

be defined for loop closures, by introducing some knowledge about the past frames (features or estimated position more likely to give a loop closure with the current frame) or by matching only a portion of scene that does not necessarily give a valid transformation in the current system.

- Additional sensors: in this work, the only sensor used was the Kinect providing RGB-D data. One possibility would be to use others sensors and combine them to get a better result in a multi-modal or hybrid system, especially when the transformations computed by the method presented in this work are less reliable.

6.2 Performance improvements

Currently, most of the time is passed on the feature extraction. Here are a few alternatives :

- GPU implementation: a feature extraction optimized for the hardware of the graphic cards would dramatically lower the computational time to enable real-time applications. Various GPU implementations exist for both SIFT and SURF. In addition, some solutions take advantage of the parallel computing on specific platforms, such as CUDA™ for the NVIDIA graphic cards. The features extraction could then be performed in 15-20ms [2].

- BRIEF: combined with a keypoint detector such as SURF or SIFT, the BRIEF descriptors [3] could give an interesting alternative, being more efficient to compute. However, this would imply to rewrite the matching code, by doing binary string comparison. See also the PhD thesis of Calonder [4].

- ICP: though the standard algorithm could increase the computational time, some efficient variations exist [20].

6.3 Other approach

With the KinectFusion project, Microsoft Research presents a solution performing reconstruction without any feature, though it achieves real-time performance as demonstrated[1] at SIGGRAPH 2011. The camera tracking is based on a GPU implementation of ICP on the point clouds generated from the depth data. The related papers [13] [19] describe how an accurate dense reconstruction can be performed in real-time, which sounds promising in the future for many applications related to computer vision, not only mapping but also segmentation and object recognition. A new version of the Kinect camera working at shorter range will also extend the possibilites.

[1] KinectFusion: http://www.youtube.com/watch?v=quGhaggn3cQ (last access: Jan 2012)

Acronyms

BRIEF Binary Robust Independent Elementary Feature. 6, 13, 21, 56

CVAP Computer Vision and Active Perception Lab. 2, 39, 49

DOF Degrees-of-Freedom. 3, 31, 37

DoG Difference-of-Gaussian. 7, 8

EKF Extended Kalman Filtering. 2, 16, 49

HoG histogram of local oriented gradients. 8

ICP Iterative Closest Point. 15, 55, 56

NARF Normal Aligned Radial Feature. 6, 12, 21

PCL Point Cloud Library. 12, 21, 40, 42

RANSAC RANdom SAmple Consensus. 10, 13, 15, 26, 30, 37, 55

ROS Robot Operating System. 12, 40

SIFT Scalar Invariant Feature Transform. 6–8, 10, 13, 21, 22, 30, 40, 55, 56

SLAM Simultaneous Localization And Mapping. 1, 2, 16, 17, 20, 49, 55

SURF Speeded Up Robust Feature. 6, 10, 13, 21, 22, 30, 38, 40, 55, 56

SVD Singular Value Decomposition. 27

VSLAM Visual Simultaneous Localization And Mapping. 1, 3, 16, 20, 55

References

[1] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-Up Robust Features (SURF). *Comput. Vis. Image Underst.*, 110:346–359, June 2008.

[2] M. Björkman. A CUDA implementation of SIFT, Computer Vision and Active Perception Lab. http://www.csc.kth.se/~celle/. (last access: Jan 2012).

[3] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. BRIEF: Binary Robust Independent Elementary Features. In *European Conference on Computer Vision*, September 2010.

[4] Michael Calonder. Robust, High-Speed Interest Point Matching for Real-Time Applications (PhD Thesis), 2010.

[5] Nikolas Engelhard, Felix Endres, Jürgen Hess, Jürgen Sturm, and Wolfram Burgard. Real-time 3D visual SLAM with a hand-held RGB-D camera. In *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum*, Västerås, Sweden, April 2011.

[6] Martin A. Fischler and Robert C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. *Commun. ACM*, 24(6):381–395, 1981.

[7] J. Folkesson, P. Jensfelt, and H.I. Christensen. The M-Space Feature Representation for SLAM. *Robotics, IEEE Transactions on*, 23(5):1024 –1035, oct. 2007.

[8] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg. Hierarchical optimization on manifolds for online 2D and 3D mapping. *In Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Anchorage, AK, USA, 2010.

[9] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard. A Tree Parameterization for Efficiently Computing Maximum Likelihood Maps using Gradient Descent. Atlanta, GA (USA), 2007.

[10] C. Harris and M. Stephens. A Combined Corner and Edge Detector. In *Proceedings of the 4th Alvey Vision Conference*, pages 147–151, 1988.

[11] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments, Delhi, India, 2010.

[12] Rob Hess. An Open-Source SIFT Library. http://blogs.oregonstate.edu/hess/code/sift/. (last access: Jan 2012).

[13] Shahram Izadi, Richard A. Newcombe, David Kim, Otmar Hilliges, David Molyneaux, Steve Hodges, Pushmeet Kohli, Jamie Shotton, Andrew J. Davison, and Andrew Fitzgibbon. KinectFusion: real-time dynamic 3D surface reconstruction and interaction. In *ACM SIGGRAPH 2011 Talks*, SIGGRAPH '11, pages 23:1–23:1, New York, NY, USA, 2011. ACM.

[14] Kurt Konolige and Motilal Agrawal. FrameSLAM: From Bundle Adjustment to Real-Time Visual Mapping. *IEEE Transactions on Robotics*, 24(5):1066–1077, 2008.

[15] Tony Lindeberg. *Scale-Space Theory in Computer Vision*. Kluwer Academic Publishers, Norwell, MA, USA, 1994.

[16] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[17] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus H. Gross. Surfels: surface elements as rendering primitives. In *SIGGRAPH*, pages 335–342, 2000.

[18] Rainer Kuemmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g2o: A General Framework for Graph Optimization. *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[19] Richard Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Andrew Fitzgibbon. KinectFusion: Real-Time Dense Surface Mapping and Tracking. In *10th IEEE International Symposium on Mixed and Augmented Reality (Proceedings of ISMAR 2011)*, October 2011.

[20] S. Rusinkiewicz and M. Levoy. Efficient Variants of the ICP Algorithm. *3D Digital Imaging and Modeling, International Conference on*, 0:145, 2001.

[21] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[22] B. Steder, R. B. Rusu, K. Konolige, and W. Burgard. NARF: 3D Range Image Features for Object Recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Taipei, Taiwan, 2010.

[23] Hauke Strasdat, J. M. M. Montiel, and Andrew J. Davison. Scale Drift-Aware Large Scale Monocular SLAM. In Matsuoka, Yoky and Durrant-Whyte, Hugh F. and Neira, José, editor, *Robotics: Science and Systems*. The MIT Press, 2010.

[24] S. Thrun and M. Montemerlo. The GraphSLAM Algorithm With Applications to Large-Scale Mapping of Urban Structures. *International Journal on Robotics Research*, 25(5/6):403–430, 2005.

[25] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*. Intelligent robotics and autonomous agents. The MIT Press, August 2005.

[26] Philip H. S. Torr and Andrew Zisserman. MLESAC: A New Robust Estimator with Application to Estimating Image Geometry. *Computer Vision and Image Understanding*, 78(1):138–156, 2000.

[27] Zhengyou ZHANG. Iterative Point Matching for Registration of Free-form Curves, 1992.