Visual Simultaneous Localization And Mapping for Humanoid Robots with Dense Techniques

Nikolaos Tavoularis

Masters' of Science degree in Computer Science and Engineering

University of Crete
School of Sciences and Engineering
Computer Science Department
Voutes University Campus, 700 13 Heraklion, Crete, Greece

Thesis Advisor: Prof. *Panos Trahanias*

This work has been conducted at the Computational Vision and Robotics Laboratory (CVRL) of the Institute of Computer Science (ICS) of the Foundation for Research and Technology–Hellas (FORTH).

The work has been supported by a scholarship from the Foundation for Research and Technology - Hellas (FORTH).

University of Crete
Computer Science Department

Visual Simultaneous Localization And Mapping for Humanoid Robots with Dense Techniques

Thesis submitted by
Nikolaos Tavoularis
in partial fulfillment of the requirements for the
Masters' of Science degree in Computer Science

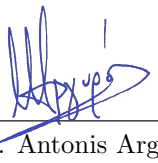
THESIS APPROVAL

Author: 

Nikolaos Tavoularis

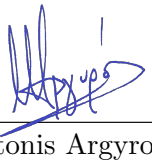
Committee approvals: 

Prof. Panos Trahanias, Thesis Supervisor



Prof. Antonis Argyros, Committee Member

Dr. Manolis Lourakis, Research Director, Committee Member

Departmental approval: 

Antonis Argyros
Professor, Director of Graduate Studies

Heraklion, July 2020

Visual Simultaneous Localization And Mapping for Humanoid Robots with Dense Techniques

Abstract

One of the hardest problems in modern robotics is the estimation of robot's pose while simultaneously creating the map of the environment, termed *Simultaneous Localization and Mapping* (*SLAM*). This problem is often confronted with the use of visual sensors such as RGB and depth cameras and it is mentioned in the literature as *visual SLAM* (*vSLAM*). In addition, solely determining the robot's pose is called *Visual Odometry* (*VO*). vSLAM algorithms can be divided into two different groups, dense algorithms that have to deal with a huge amount of data and sparse algorithms that handle less data but they have to choose them meticulously. vSLAM is particularly hard due to sensor's noise, blurriness of the input image, approximation errors and limited resources. The problem is even harder when it is applied to humanoids. Humanoids' bipedal movement cause sudden acceleration of the camera's motion and extra blurriness in the image.

This work starts with an evaluation of vSLAM systems and their properties. Then it follows an evaluation of novel feature extraction algorithms that use neural network in order to extract descriptors from 3D data. Algorithms such as *3DMatch* [56] and *3DSmoothNet* [10] were compared with more conventional methods such as *SIFT*. Next it proposes an algorithm that produces better pose estimation and map representation for humanoids. This is achieved with the combination of two systems, *Kinect Fusion* [13] SLAM system and *State Estimation RObot Walking* (*SEROW*) [35] which fuses VO with kinematics. Moreover it provides a lightweight implementation that runs on embedded devices such as jetson tx2. Finally this work studies the combination of dense methods with sparse features in a unified graph representation.

Ταυτόχρονος Προσδιορισμός της Πόζας και Χαρτογράφηση για Ανθρωποειδή Ρομπότ με χρηση Πυκνών Τεχνικών

Περίληψη

Ένα από τα δυσκολότερα προβλήματα της σύγχρονης ρομποτικής είναι το πρόβλημα του προσδιορισμού της πόζας του ρομπότ μαζί με την ταυτόχρονη χαρτογράφηση του περιβάλλοντός του (*Simultaneous Localization and Mapping - SLAM*).Το πρόβλημα αυτό συχνά αντιμετωπίζεται με χρήση οπτικών αισθητήρων όπως κάμερες βάθους ή χρώματος και αναφέρεται στην βιβλιογραφία ως ταυτόχρονος εντοπισμός της θέσης και χαρτογράφηση με οπτικά μέσα, *visual SLAM (vSLAM)*. Αντίστοιχα, ο αποκλειστικός εντοπισμός της πόζας ονομάζεται *Οπτική οδομετρία*, *Visual Odometry* (*VO*). Οι αλγόριθμοι vSLAM χωρίζονται κυρίως σε δύο ομάδες ανάλογα με το μέγεθος των δεδομένων που χρησιμοποιούνται: α) Στους πυκνούς αλγορίθμους, που χρησιμοποιούν μεγάλο όγκο απο σημεία στο χώρο, και β) στους αραιούς αλγορίθμους που μειώνουν τον αριθμό των σημείων, επιλέγοντας μόνο κάποια χαρακτηριστικά σημεία του χώρου. Γενικά, το πρόβλημα του vSLAM είναι ιδιαίτερα δύσκολο λόγω υψηλού θορύβου στους αισθητήρες, θολότητα στις εικόνες εισόδου, σφάλματα στους προσεγγιστικούς αλγορίθμους, περιορισμένοι υπολογιστικοί πόροι κ.τ.λ. Το πρόβλημα όμως είναι ακόμα πιο δύσκολο όταν εφαρμόζεται σε ανθρωποειδή ρομπότ. Η δίποδη βάδιση των ανθρωποειδών ρομπότ προκαλεί ξαφνική επιτάχυνση της κίνησης της κάμερας μαζί με αυξημένη θολότητα στην εικόνα.

Η παρούσα εργασία ξεκινάει με μια ανάλυση των υπαρχόντων συστημάτων vSLAM και των ιδιοτήτων τους. Έπειτα ακολουθεί μια σύγκριση καινοτόμων αλγορίθμων όπως το *3DMatch* [56] και *3DSmoothNet* [10], οι οποίοι εξάγουν χαρακτηριστικά σημεία με χρήση νευρονικών δικτύων. Οι αλγόριθμοι αυτοί μελετούνται σε σχέση με πιο συμβατικές τεχνικές οπως ο αλγόριθμος των *SIFT*. Εν συνεχεία παρουσιάζεται μια μέθοδος που επιτυγχάνει καλύτερη εκτίμηση της πόζας μαζί με πιο ακριβή χαρτογράφηση. Αυτό γίνεται με τον συνδυασμό δυο συστημάτων, του vSLAM συστήματος *Kinect Fusion* [13] και του *State Estimation RObot Walking* (*SEROW*) [35] που συνδυάζει οπτικη οδομετρια με κινηματική. Επίσης παρέχεται μια χαμιλή σε υπολογιστικούς πόρους υλοποίηση που επιτυγχάνει εκτέλεση σε πραγματικό χρόνο σε ενσωματωμένες συσκευές όπως το jetson tx2. Τελος μελετάται ο συνδιασμος πυκνών αλγοριθμων μαζι με αραιά χαρακτηριστικά σημεία σε μια ενοποιημένη αναπαρασταση γράφου.

Acknoweledgments

This thesis would never have become a reality without the noted contributions of many individuals and institutions, to whom I'm deeply indebted. First and foremost I would like to express my gratitude to my supervisor Prof. Panos Trahanias, who advised me and supported me throughout this thesis and also gave me the wonderful opportunity to work at the Computational Vision and Robotics Laboratory (CVRL).

Two members of CVRL played an important role during the pursuit of the thesis technical work. Dr. Manos Hourdakis offered ample guidance and supervision for my work, while Dr. Stelios Piperakis greatly supported me with his advice and contributions.

The focused comments and suggestions of my thesis committee members, namely Professor Antonis Argyros and Dr. Manolis Lourakis, greatly contributed to the final written document and presentation of my thesis.

It would have been an omission not to mention CVRL and all its members for their support and help. Also I am highly thankful to the Foundation for Research and Technology - Hellas (FORTH) for its financial support and provision of resources and equipment. In addition, I would also like to thank the Computer Science Department of the University of Crete for offering me the opportunity to study in this vibrant environment and for all the knowledge and experience I gained during this graduate degree.

Finally I would like to thank my aunt Miranta Papadaki who supported and inspired me in this endeavor.

Nikolaos Tavoularis

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Simultaneous Localization and Mapping

In autonomous navigation, agents calculate their desired trajectory in order to reach their destination. For that task, the knowledge of its current position is crucial. The process of determining agent's position with respect to its environment is known as *localization*. In addition, in order to plan an efficient trajectory, agents need to know the world that surrounds them. The world is represented either as a map in a two dimensional plane or as a more complex structure in three dimensions. The problem of building and maintaining such a map is called *mapping*. Frequently, localization and mapping problems coincide. While a robot navigates in an unknown environment, it has to build the map of the surroundings and, in addition, it has to determine its location within it. The problem of solving Localization and mapping simultaneously is called *Simultaneous Localization and Mapping* or *SLAM*. The first SLAM algorithms focused on fussing together information from robots' sensors and control system into a global probabilistic model. Shortly SLAM was spread to multiple applications in computer vision and virtual reality where the use of cameras is essential. This created the problem of *Visual SLAM* which deals with the SLAM problem with only the use of cameras. In addition, solely determining robot's pose with cameras is called Visual Odometry (VO). Usual SLAM algorithms calculate VO at first, and then integrate the visual information into a map.

1.2 Algorithms Categorization

Existing SLAM algorithms can be classified into two major categories, according to the number of points that they extract from camera images. Algorithms that use only a small subset of points or pixels from a camera image are called sparse, while algorithms that use most or all of the camera information are called dense.

1.2.1 Sparse Algorithms

Sparse algorithms extract a set of distinguished points from images called features. Features are characteristic points such as edges, corners etc. that are extracted through feature extraction techniques. Additionally to points' location, a high dimensional descriptor is calculated in order to distinguish each specific point. Such points form a sparse representation of the world and they are tracked from consecutive images in order to determine the camera's displacement. There are two highly used algorithms for feature tracking, probabilistic filters and bundle adjustment.

Probabilistic Filters: In this approach, features are treated as unknown variables with a mean value and uncertainty modeled as a covariance matrix. Then, probabilistic filters such as Kalman Filters are applied in order to track them between multiple frames. These algorithms work well with a relatively small number of features, since increased number of features require more filters and highly increased computations. An example of such an algorithm is MonoSLAM [8].

Bundle Adjustment: Bundle adjustment is a technique that minimizes the re-projection error of features from the 3D space to camera frame between multiple images [54]. Figure 1.1 displays the bundle adjustment process between three images (C_1, C_2, C_3).

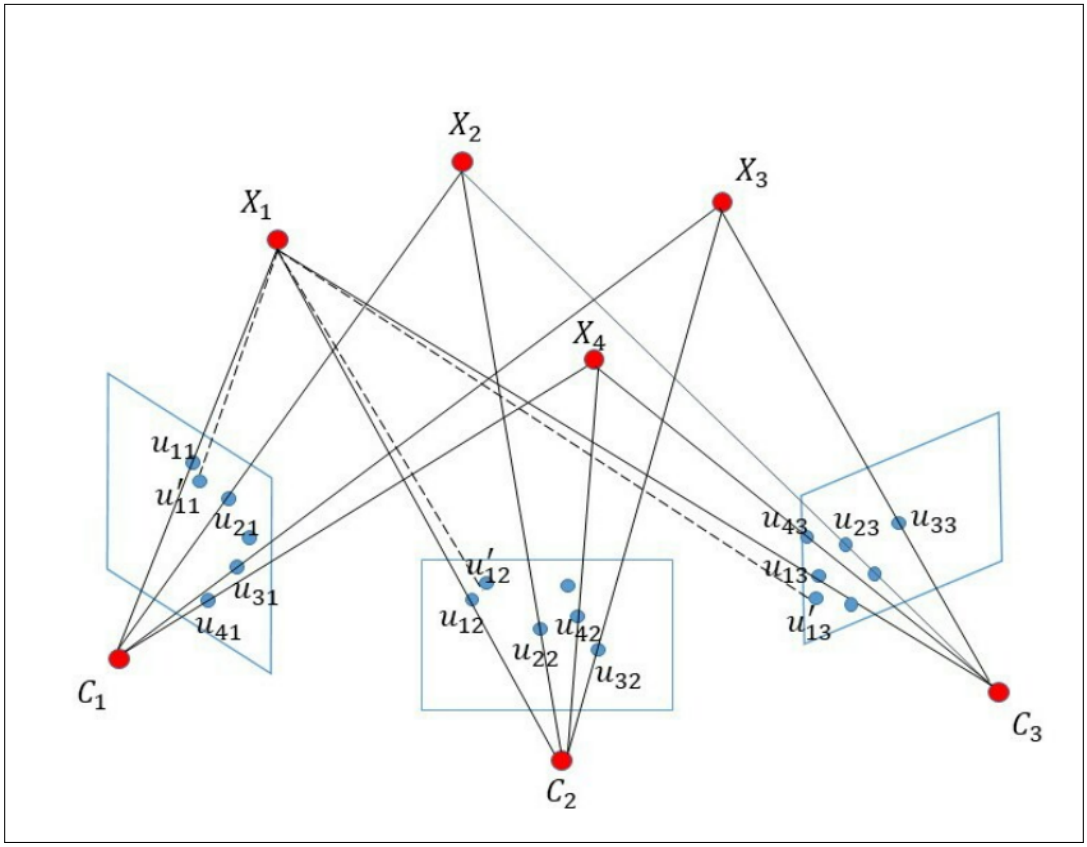


Figure 1.1: Bundle Adjustment.

Formally Bundle adjustment can be described as a non linear least squares problem by the following formula:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{m} (u_{ij} - p(C_j, X_i))^2$$

Where u_{ij} is the feature i in camera frame j, X_i is the 3D coordinates of feature i and p is a function that projects the given feature in camera frame j.
One generic Bundle Adjustment framework that supports a wide range of parametrization, is *Sparse Bundle Adjustment* or *SBA* [27]. SBA is written in ANSI C but it also supports linking with *C++* and *MATLAB* through *MEX-files*. An example of a SLAM algorithm that utilizes Bundle Adjustment is *PTAM* [18].

Generally bundle adjustment suffers from memory and efficiency requirements in large scale reconstruction and for that reason it is used only in sparse algorithms.

1.2.2 Dense Algorithms

In contrary to sparse algorithms, dense algorithms do not extract characteristic points but they use most or all the pixels of the image. This approach highly increases the amount of processing data but it is relieved of feature extraction and therefore it is not sensitive to feature selection.

Iterative Closest Point: The most common algorithm for calculating camera's displacement from dense points clouds is Iterative Closest Point (ICP). [2]. ICP finds the rigid transformation that aligns a source point cloud towards a target point cloud by minimizing an error function. There are two error metrics commonly used, *Point-to-point* and *Point-to-plane* metric. Point-to-point metric defines the error function as the sum of the square distances between the corresponding points, while point-to-plane metric minimizes squared distances between a point and the tangent plane. More specifically the point-to-point metric is formulated in equation (1.1) bellow, while equation (1.2) is a depiction of the point-to-plane metric:

$$J = \sum_{i=1}^{n} (R \cdot P_i + T - Q_i)^2 \tag{1.1}$$

$$J = \sum_{i=1}^{n} \left((R \cdot P_i + T - Q_i) \cdot n_i \right)^2 \tag{1.2}$$

where:

$$\begin{aligned} R &= \text{rotation} \\ T &= \text{translation} \\ P_i &= \text{source point} \\ Q_i &= \text{target point} \\ n_i &= \text{tangent vector of point } i \end{aligned}$$

Accordingly, ICP attempts to find the optimal R_{opt}, T_{opt} that minimize its error function:

$$[R_{opt}|T_{opt}] = \arg \min_{[R|T]} J(R, T)$$

This problem is a nonlinear least-squares optimization problem and solving it is often computationally involved. In addition the correspondences between source points P_i and target points Q_i are needed. ICP simply matches a source point with its nearest one in the target point cloud. This however may not be the optimal match and outliers may deteriorate it further. For that reason, ICP is applied iteratively and points that their

distance exceeds a threshold are excluded. In every iteration ICP calculates the rigid transformation between the inliers that minimizes the error function. The problem of determining the rigid transformation between two sets of corresponding 3D points is known as *Absolute orientation*. ICP usually assumes a small relative orientation between the two point clouds and approximates the solution with a linear system of equations than can be solved by *Singular Value Decomposition* (*SVD*) [24]. Hence ICP converges iteratively toward the optimal solution and it is often applied in a coarse to fine pyramid scheme. Nevertheless, if the small rotation assumption does not hold, ICP may not converge or may be trapped in a local minimum.
Additionally to SVD other methods have also been proposed for the Absolute Orientation problem. *Lourakis* proposed an efficient solution to the absolute orientation problem [22] and later together with *Terzakis* reviewed the performance of various direct methods for Absolute Orientation [23].

1.2.3 Loop Closure

A common problem in visual odometry methods is the accumulation of error. Most algorithms calculate the displacement of the camera from its previous position. Such calculations are in most cases erroneous and as the camera moves, this error accumulates. Loop Closure techniques add extra constraints to pose estimation in order to bound the accumulated error. A widespread method for loop closure is the implementation of a Pose Graph.

Pose Graph: A Pose Graph, like a regular graph, contains nodes and edges. Nodes are divided into two categories. Nodes that have been associated with robot's poses (Pose nodes) and nodes that have been associated with landmarks (landmark nodes). Pose nodes represent robot's poses taken from different points in time while landmark nodes indicate a distinct location in the world. Furthermore edges express constrains between nodes. Hence an odometry measurement is a constraint from a previous location to the next one and it is represented by an edge on the graph. The value of this edge contains the displacement as well as the uncertainty of the measurement.
Figure 1.2 illustrates a Pose Graph with pose nodes (blue nodes) and landmarks (green nodes).

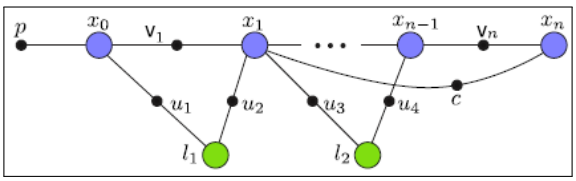


Figure 1.2: Pose Graph, blue nodes are poses while green nodes are landmarks.

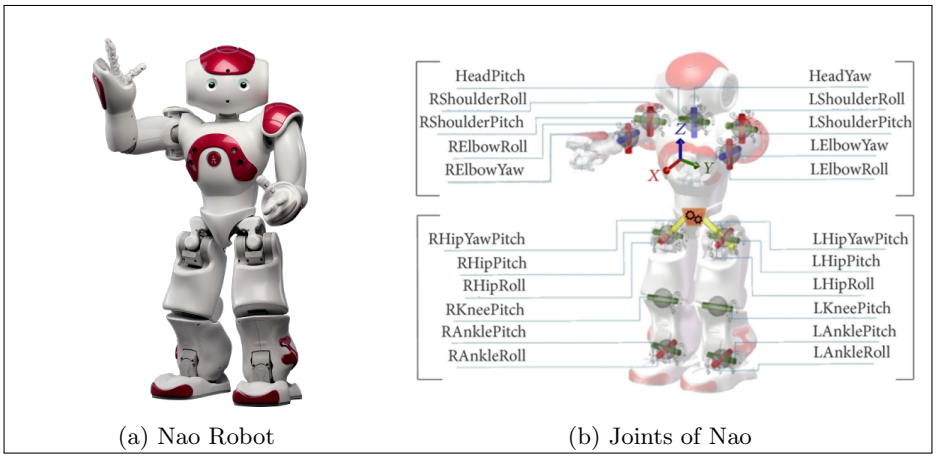
When a Pose Graph contains circles, the values of the edges can be optimized in order to minimize uncertainty.
There are two widely used implementations of Pose Graph: General Graph Optimization (G2O) [20] and Incremental Smoothing and Mapping (ISAM) [15].

1.3 vSLAM in Humanoids

The environment that we live in, is created and modified specially for human needs. Accordingly, mobile robots are often inadequate for completing tasks in this environment. For that reason humanoid robots have been developed that resemble the actual human body and can interface better within our environment. Moreover humanoid robots *feel* more familiar to us improving human to robot interaction.
Generally, humanoids have a torso, a head, two arms, and two legs. Also they are characterized by their bipedal locomotion. The bipedal movement of humanoids is calculated by the inverse kinematics. However, calculating the inverse kinematics is a complex problem, due to high degrees of freedom, and can not be solved analytically. Furthermore, regular kinematics suffer from slippage in bipedal gait, discontinuous ground contacts and actuation errors. As a result, localization solely based on kinematics is prone to error. Hence a visual SLAM system can increase localization accuracy.

1.3.1 Nao Robot

One widely used humanoid is the Nao Robot (Figure 1.3). Nao is preferred by multiple laboratories due to its compact size and relatively low cost. It is developed by SoftBank robotics and since 2008 was used in RoboCup. It has 57.3 centimeters height and 25 degrees of freedom. In addition, Nao contains nine tactile sensors, eight pressure sensors, sonar, gyroscope and two cameras. However, despite its variety of features, its sensors are of low quality because of its low cost and they are often inadequate for SLAM purposes. Specifically, the gyroscope is of low quality and produces measurements in low frequency. In addition, the overlapping section between the field of view of the cameras is very narrow, making the cameras insufficient for stereo matching.



(a) Nao Robot (b) Joints of Nao

Figure 1.3: Characteristics of the Nao Robot.

1.4 Thesis Contribution

This work addresses the SLAM problem and its application in humanoid robots focusing on RGB-D sensors and embedded devices. It begins with a performance evaluation of various SLAM systems and their components in simulated experiments. The purpose of this evaluation is to study the performance of SLAM algorithms in order to use them in embedded GPUs and humanoid robots. The above evaluation showed that Kinect

Fusion performs well with respect to achieved frame-rate and memory usage and thus, it is well suited for on-board execution. Then it follows an evaluation of geometric feature descriptors extracted from convolutional neural networks. Sparse features are used later in this work for an implementation of a loop closure mechanism. In this evaluation, *3DMatch* [56] and *3DSmoothnet* [10] were studied along with *SIFT* features [25]. Based on the obtained evaluation results, this thesis subsequently proposes two modifications of Kinect Fusion. The first modification combines Kinect Fusion with State Estimation RObot Walking (SEROW). SEROW provides a humanoid robot's state estimation that fuses multiple information including VO. The proposed modification improves at the same time state estimation and map representation and achieves real time execution on a *Jetson tx2*. The second modification enhances Kinect Fusion with a loop closure mechanism. The mechanism incorporates VO from Kinect Fusion with sparse features into a common Pose Graph. It achieves to bound the accumulated error but runs offline.

1.5 Thesis Structure

The structure of this thesis is as follows. Related work is first presented in Chapter 2. Then, Chapter 3 contains the evaluations of SLAM system and feature descriptors. Chapter 4 describes the modifications of Kinect fusion with both SEROW and Pose Graph and Chapter 5 includes the results of the modified Kinect Fusion. The thesis concludes with Chapter 6 that summarizes our work and provides directions for future work.

Chapter 2

Related Work

2.1 Simultaneous Localization and Mapping Algorithms

A turning point on SLAM systems was the creation of Microsoft Kinect sensor with RGB and Depth information. Since then, many algorithms exploited the new technology to utilize depth for pose estimation and map creation, which is also the case in the current work. The first algorithm that used Kinect sensor was *Kinect Fusion* [13]. Kinect fusion performs solely on depth data, consequently it works in variable lighting conditions. It stores the map in a dense volumetric representation and it achieves high frame rate due to its GPU based implementation. The main disadvantage of Kinect fusion is the pre-allocation of memory resulting in a fix map size. Many algorithms have been proposed to expand the map size dynamically and also reduce the accumulated error.
Fioraio et al. [9] propose a system of multiple volumes. Volumes contain at maximum k frames so the drift inside a volume should be minimal. Then they are aligned together through an energy minimization procedure on their zero-level set.
Another algorithm that is based on Kinect fusion is *InfiniTAM v3* [17]. InfiniTAM creates sub-maps every k frames. These sub-maps may overlap and the camera may be positioned in more than one. The ray-tracing process is carried out in every sub-map that contains the camera. Hence the relative positions of the camera also align the sub-maps together. InfiniTAM achieves real time execution by implementing hashing techniques in voxel grid, increasing ray-tracing performance and reducing memory usage. Additionally [32] implements voxel hashing that allows arbitrary large grids. However it does not deal with the accumulated estimation error.
Moreover *Whelan* et al. [49] implemented a system that divides volume into *slices*. As the camera moves, volume map is sifted by removing the most distant slice and inserting an empty one at the end of the map. Slices form a circular buffer so no extra memory allocation is required. The data of the removed slice are stored in form of surfels. Finally camera's location during a sift is stored in a Pose Graph and a loop closure mechanism is developed with the use of SURF features.
Another dense SLAM algorithm is *Elastic Fusion* [50]. Elastic Fusion represents the world with *Surface elements* or *surfels*. Similar to ICP, Elastic Fusion calculates VO by solving an energy minimization problem which considers both geometry and photometry of the scene. In addition, it performs local and global loop closure by applying non-rigidly deformation in surfels elements.
In addition there are multiple sparse slam algorithms in the literature. One such algorithm is *ORB-SLAM2* [31]. ORB-SLAM2 works on multiple cameras (monocular, stereo and RGB-D) and utilizes *Oriented FAST and Rotated BRIEF* or *ORB* features [41]. Moreover

it implements a loop closure mechanism based on a Pose Graph. Parallel to a main thread, a second thread is running that detects loops and optimizes the graph.
Another sparse algorithm that operates with monocular cameras is *MonoSLAM* [8]. MonoSLAM creates a probabilistic map with state estimates for every feature as well as their uncertainty. In every frame, it applies Extended Kalman Filters in order to predict the camera's motion and features displacement. MonoSLAM achieves real time execution by limiting the features number to around 100.
One SLAM algorithm that has been used in augmented reality is *Parallel Tracking and Mapping (PTAM)* [18]. PTAM works on monocular cameras and separates tracking and mapping into two different threads. One thread tracks camera's motion while the other optimizes the map using bundle adjustment.
Recently a modification of PTAM, *Stereo Parallel Tracking and Mapping* (*S-PTAM*) [37] has been proposed in order to extent PTAM capabilities into stereo cameras and avoid the bootstrapping problem in monocular systems.
An additional monocular algorithm is *VINS-Mono* [39]. VINS-Mono overcomes the lack of depth information by fusing IMU with visual odometry creating the Visual-Inertial Odometry (VIO). VINS-Mono tracks corner features with a sparse optical flow algorithm and rejects outliers with RANSAC. Additionally to VIO, it provides a loop detection mechanism with *bag-of-word* place recognition as well as a loop closure and re-localization module based on Pose Graph optimization.
Finally *BundleFusion* [7] is a recent slam system that combines sparse and dense methods. It utilizes SIFT features for an initial coarse global alignment and then it refines this alignment with a geometric and photometric energy minimization problem. This approach achieves instantaneous and globally consistent relocalization.

2.1.1 Datasets

Besides SLAM algorithms, there are multiple datasets to test and assess their performance. The Technical University of Munich provides the *TUM* [46] datasets with RGB-D data from a variety of environments such as an office environment, an industrial hall etc. RGB-D information was captured from a Microsoft Kinect which was either handheld or mounted into a *Pioneer 3* mobile robot. Along with the camera data, the ground truth trajectory of the sensor was obtained from an eight cameras motion-capture system.
Moreover *Imperial College London and National University of Ireland Maynooth (ICL-NUIM)* [11] datasets simulates camera motion in synthetically generated environments. ICL-NUIM includes RGB-D data as well as ground truth trajectories and surface models. RGB-D information includes real world artifact by modeling sensor's noise in RGB and depth. In addition due to the synthetically generated nature of the dataset, the ground truth trajectory and models are perfect.
Another recently published collection of datasets is *OpenLORIS* [43]. OpenLORIS provides visual, inertial and odometry data from Intel's *RealSense* cameras mounted on a mobile robot. It also includes several scenes from an office environment, a household, a market etc. In addition it captures scene changes such as day-night shifts, human motion and others. The ground truth trajectory of OpenLORIS was captured by either *OptiTrack* motion capture system or by offline LiDAR SLAM, based on the *Hokuyo* laser scans.

2.1.2 Evaluation Software

SLAM algorithms can be evaluated with the *Slambench2* [3] software. Slambench2 is a platform for SLAM systems evaluation that provides implementation of the most popular

SLAM algorithms in addition with a unified framework for bench-marking them. It was developed by a collaboration between University of Edinburgh, Imperial College and University of Manchester under the umbrella of the PAMELA project. Among others, Slambench2 records the processing time, memory usage and error in pose estimation. Moreover it provides tools for importing algorithms and datasets. Finally, in 2019, Slambench2 was succeed by *Slambench3* [4]. Slambench3 includes implementations of the latest SLAM algorithms and supports dynamic SLAM and scene understanding with convolutional neural networks.

2.1.3 Simultaneous Localization and Mapping in Humanoids

Visual SLAM is especially challenging in humanoids. While wheeled robots can navigate using a 2D representation of the world, humanoids require a 3D structure in order to plan their footsteps. Moreover their bipedal movement causes sudden accelerations due to contact switching. This on its part, results in visual motion blurriness that deteriorates the performance of SLAM systems. Furthermore, humanoid robots have limited power and computational resources. Consequently the computational efficiency of the algorithms together with an optimized implementation is necessary.
For such reasons, multiple studies focus especially on humanoid robots and how to integrate a SLAM system into them.
Kagami et al. [16] use stereo images together with a localization module to produce a 2.5D map of the environment. The map consists of a 2D grid with extra height information.
Stasse et al. developed a system for humanoids that integrates walking and vision in order to increase autonomy [44]. They based their SLAM system on Extended Kalman Filters (EKF) and sparse feature extraction. The authors in [51] attempted to implement a sparse visual SLAM algorithm in Nao robot but they encountered several difficulties. Specifically they found out that Nao cameras are subject to strong motion blurriness and also they are inadequate for stereo vision since they do not have a shared field of view. In addition, Nao's CPU is limited and can not run heavy procedures such as Kalman filters. Finally, feet slippage and hip design cause systematic deviation in the walk and odometry drifting. The authors however achieved to track orientation changes and build a visual compass with only one camera.
Maier et al. [26] overcame Nao's cameras problems by mounting an external Asus xtion RGB-D camera on Nao. They used depth and kinematics in order to produce a high-resolution map with height and uncertainty information for every cell. Later this map was used to produce collision-free footsteps and whole-body motions.
In addition, the authors in [33] produce a more accurate odometry for Nao robot using EKF and fusing multiple sources including IMU, kinematics and VO. Briefly, joint encoders are used in the prediction step of EKF while VO from *PTAM* SLAM algorithm and IMU are used as correction measurements.
Complementary to sparse algorithms, also dense SLAM systems have been integrated into humanoid robots due to their good performance in environments with poor features. In [57] the authors propose a dense method for map reconstruction working in dynamic environments with human motion in it. They based their method on human body detection through deep learning and a graph-based segmentation that separates moving objects from a static environment.
In [42] the authors combine *Elastic Fusion* with kinematics and IMU. They expand the energy function of Elastic Fusion in order to consider photometric, geometric and kinematic errors. Finlay, they embedded and tested their system in the Valkyrie humanoid robot.

2.1.4 Simultaneous Localization and Mapping for Specific Purposes

In addition to SLAM algorithms for Humanoid Robots, other SLAM algorithms have been proposed for specific systems. For instance *Hourdakis* et al. [12] proposed a SLAM system for Planetary Rovers. Their system detects geological elements such as boulders either on the ground or in orbit and use them to update VO reducing pose drift. Moreover *Lentaris* et al. [21] focus on FPGA implementation of VO also for Planetary Rovers. They perform a hardware and software codesign that achieves one order of magnitude faster execution. Furthermore, *Panteleris* et al. [34] proposed a SLAM system motivated by the *c-Walker* device. The latter is a smart walker platform that aims to safely guide elderly people though public spaces and crowded areas. The proposed system detects and tracks independently moving objects while progressively building the map of the environment. Also it achieves near real time performance on ARM based embedded systems.

2.2 Feature Extraction

Sparse algorithms depend on local feature tracking between consecutive images. Generally local features are pieces of information that describe distinct areas either on the image plane or in the 3D world. They consist of key-points and descriptors. Key-points define feature locations while descriptors characterize the corresponding feature uniquely. Features can be extracted either from RGB or depth information of the scene. Most of the algorithms that extract features from RGB assume that the photometry of the scene does not change, and that the object reflects the light evenly to every direction. This assumption however does not always hold true. For that reason geometric features that operate solely on depth have been introduced, although in various cases they are prone to noise.

2.3 Photometric Features

The *Scale-Invariant Feature Transform* or *SIFT* [25] was one of the first feature extraction algorithms introduced in the literature. SIFT features rely on the Difference of Gaussians (DoG) applied on grayscale images and they produce rotation and scale invariant key-points and descriptors. Later Mikolajczyk et al. [29] performed PCA analysis on SIFT descriptors and showed that they have high distinctiveness. In addition, Yan Ke et al. [52] proposed *PCA-SIFT* that reduce the size of the relevant descriptor from 128 dimensions to 36.
Another feature algorithm that outperforms SIFT in execution time is *Speeded-Up Robust Features* (*SURF*) [1]. SURF uses an approximation of the determinant of Hessian that can be calculated with very few operations, thus improving performance. Similarly, *Binary Robust Independent Elementary Features* (*BRIEF*) [5] introduces a binary descriptor that outperforms SURF. Moreover similarities between multiple BRIEF descriptors are calculated with the Hamming distance instead of euclidean. This greatly decreases the needed calculations and at the same time increases performance. Nevertheless BRIEF does not provide a key-point extraction method. Later *Oriented FAST and rotated BRIEF* (*ORB*) [41] features combined BRIEF with FAST key-point detector [40] in order to provide a fast and free of license feature extraction mechanism. The ORB descriptor was used in *Orbslam* [30] SLAM algorithm.

2.4 Geometric Features

Geometric features extract key-points and descriptors from depth data. They do not consider the photometry of the scene, hence perform well under different lighting conditions. Nevertheless depth data are usually more noisy and come in low resolution.
Kovács et al. [19] propose a method for corner detection in simple objects. In addition they provide a descriptor generation algorithm. This method however was tested only in simple synthetic scenes. *Normal Aligned Radial Feature* (*NARF*)[45] provides a geometric feature technique for key-points and descriptor extraction from range images. NARF selects key-points in position where the surface is stable so the estimation of the normal is robust and also there are sufficient changes in the immediate vicinity.
Another common approach is to use neural networks for feature extraction. 3DMatch [56] is a neural network model that learns a local descriptor from 3D patches. It selects random key-points and produces local patches around them. Then a convolutional neural network calculates the actual descriptor. 3DMatch also aligns the local patches by using the RANSAC algorithm. A different model that calculates local descriptors with convolution networks is *3DSmoothNet* [10]. Contrary to 3DMatch, it uses a different representation for the local patches that achieves better rotation invariance and more dense matrices.

2.5 Robot Operating System - ROS

All algorithms described here are independent from the sensors hardware and can perform under different platforms. Thus, a hardware abstraction is necessary for their efficient implementation. *Robot Operating System* or *ROS* offers such abstraction for robotics applications. ROS is widely used in research and industry. It is a mid-ware that stands in a layer above the actual operating system, and bellow the processes. It provides low level device code and hardware abstraction by a message passing mechanism. Processes in ROS are called nodes and they communicate with each other using streaming topics, services, and the Parameter Server. Topics are the most common way of communication. They are data-streams with a message type and a topic name. So arbitrary nodes may publish messages in a topic and subscriber nodes access such messages. With that mechanism, ROS achieves the hardware abstraction as stated above. A device driver is an independent node that streams information into a topic. For instance a camera driver streams images in the camera topic. Therefore, any subscriber node accesses camera's images without any knowledge of the hardware.
Software in ROS is organized in packages. A ROS package contains ROS nodes, source code, libraries, executables etc. In addition it has build instructions and dependencies. A package may depend on other packages for their compilation or node execution. That forms a complicated build system. Fortunately, ROS provides its own tool for compilation and resolving dependencies called catkin. Finally, ROS provides its own emulation system called *Gazebo*.

Chapter 3

Performance Evaluations

This chapter contains the results of the performance evaluations conducted in this work. In Section 3.1, SLAM algorithms and their components are tested with the purpose to be used in embedded devices. Then in Section 3.2 feature descriptors are tested for a latter implementation of a loop closure mechanism. Specifically *3DMatch* and *3DSmoothNet* feature descriptors that are extracted from convolutional neural networks are compared with more conventional methods such as SIFT.

3.0.1 System Description

All evaluations were executed on a desktop computer with the following specifications:

CPU:	intel i7-8700K
CPU frequency:	3.7Ghz (Turbo boost 4.7Ghz)
CPU cores:	6
Threads per core:	2
RAM:	32Gb
gpu:	GeForce GTX 1070
gpu memory:	8Gb
gpu Cores:	1820

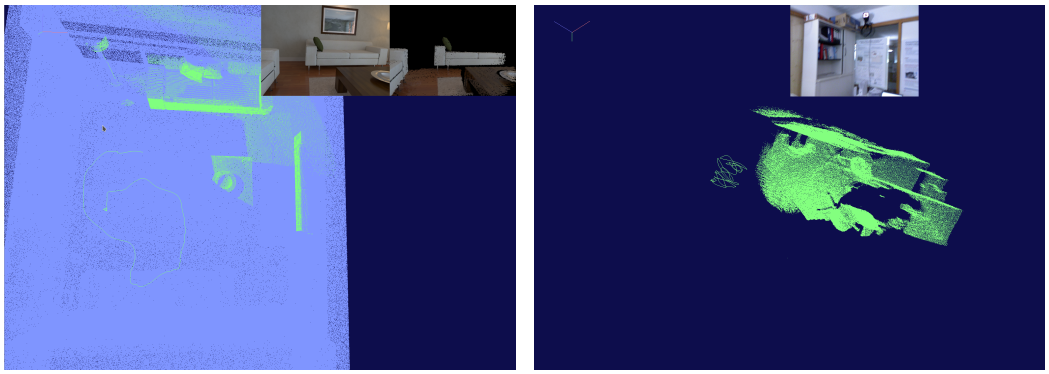
3.1 Evaluation of Simultaneous Localization and Mapping Systems

In this section we study the performance and requirements of different components that inhere in various SLAM systems. Specifically we conducted three sets of experiments. In the first set we studied the performance and the quality of the results of Visual Odometry, implemented by ICP on depth data. For that reason the *Kinect Fusion* algorithm implementation was used. Next we tested the performance of VO when extra photometric together with geometric constrains are applied. For this particular case, *Elastic fusion* was used. Finally we examined the requirements of loop closure on sparse algorithms using Orbslam2. All the experiments were contacted using the Slambench2 platform [3].

3.1.1 Datasets

In all the evaluation experiments, two different datasets were used:

- *Living_room* trajectory 2 dataset from ICL-NUIM [11].
- *Freiburg1_360* dataset from TUM [46].



(a) Living Room Dataset. (b) Freiburg1_360 Dataset.

Figure 3.1: Snapshots from the two employed datasets.

Living_room: This dataset contains RGB-D data from a common living room. The data are synthetically generated and hence the ground truth trajectory is perfect. In addition the noise of the sensor has been modeled in both RGB and depth. In this dataset, the camera is moving slowly in a large circle. Figure 3.1a displays a screenshot of Slambench2, where the green line is the ground truth trajectory of the camera. This is a characteristic example of a domestic area and we expect good performance of SLAM algorithms and accurate map representation.

Freiburg1_360: This is a common office environment. However it contains multiple rotations which are expected to stress the algorithms. As in the above case, the green line in figure 3.1b demonstrates the ground truth trajectory of the camera.

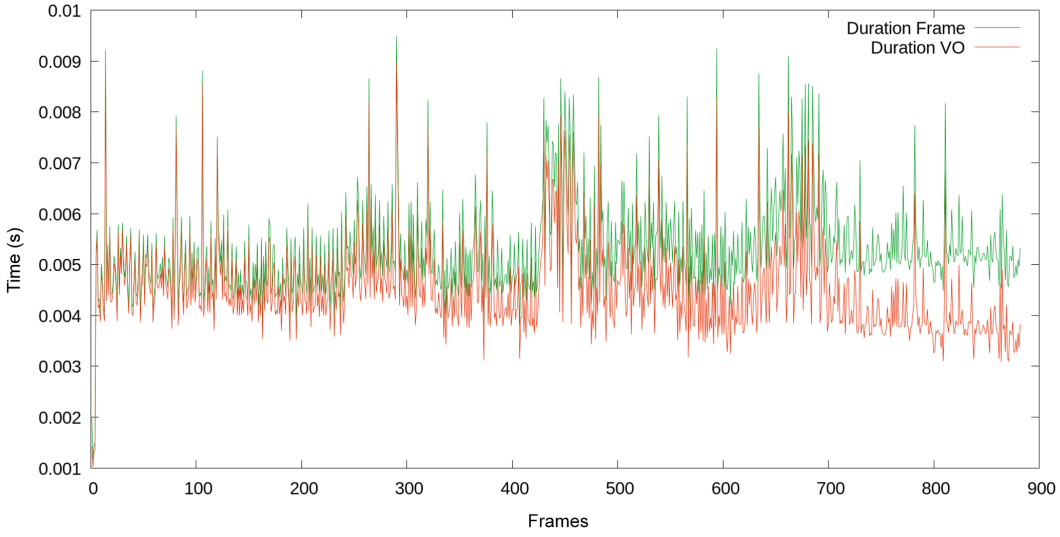
3.1.2 Performance Evaluation for the Case of Depth Data

In this case we study the requirements of Visual Odometry (VO) as it was implemented by point-to-plane ICP. ICP was operating on depth data taking into account solely the geometry of the scene. For this particular experiment we use Kinect Fusion and its Slambench 2 OpenCL implementation. We measured per frame and VO processing time, memory usage and pose estimation error. The absolute error of the graphs represent the translation error per frame in meters while the mean error represents the average error until the corresponding frame

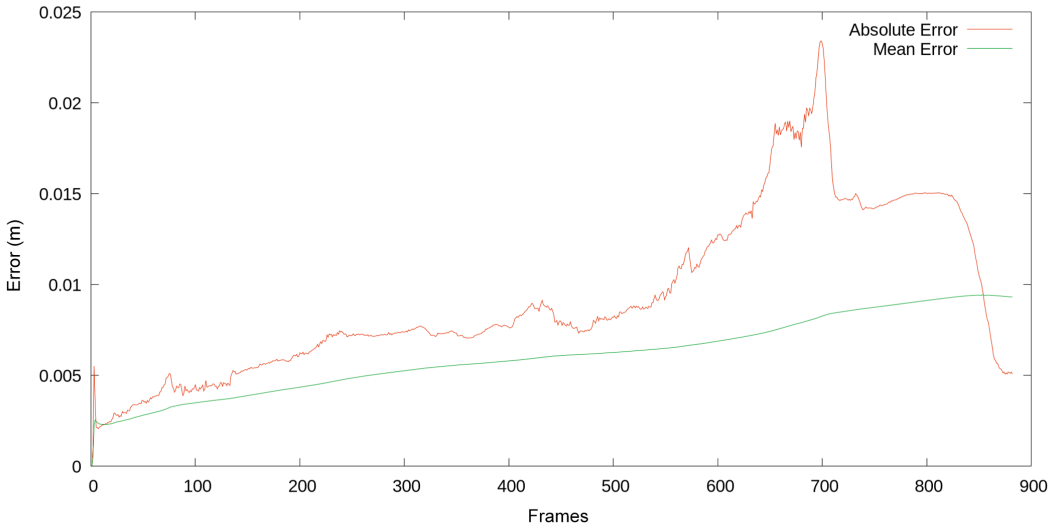
Figures 3.2a and 3.3a show the execution time per frame (green line) and VO (red line). The x axis represents the frame number while the y axis is the time in seconds. From the graph it is clear that the execution time of each frame is mainly influenced by VO. Furthermore the measured frame-rate was **188,7** frames/s. Considering that regular cameras run at 30 frames pes second, Kinect Fusion can apparently achieve real time execution. Nevertheless Slambench is a simulator and is not concerned about system issues that may deteriorate the frame rate, like camera drivers, memory copying of images etc.

Figures 3.2b and 3.3b represent the absolute and mean translation error of pose estimation per frame. Generally the error is increasing. Nevertheless in the room dataset, we detect a drop in the absolute error after the camera has finished a cycle.

Finally diagrams 3.4 present the memory usage of Kinect Fusion, which is pre-allocated and it is common for every dataset.

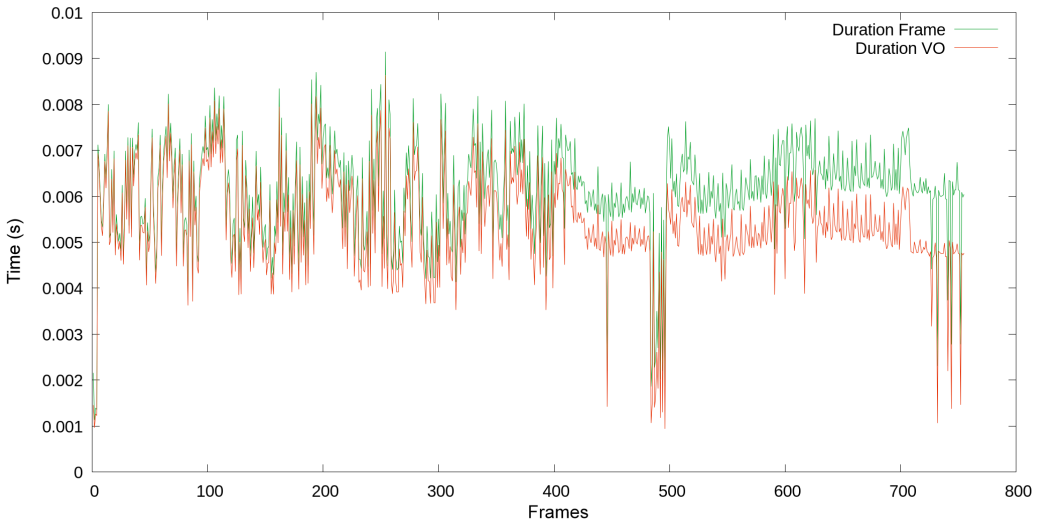


(a) VO and Frame processing time.

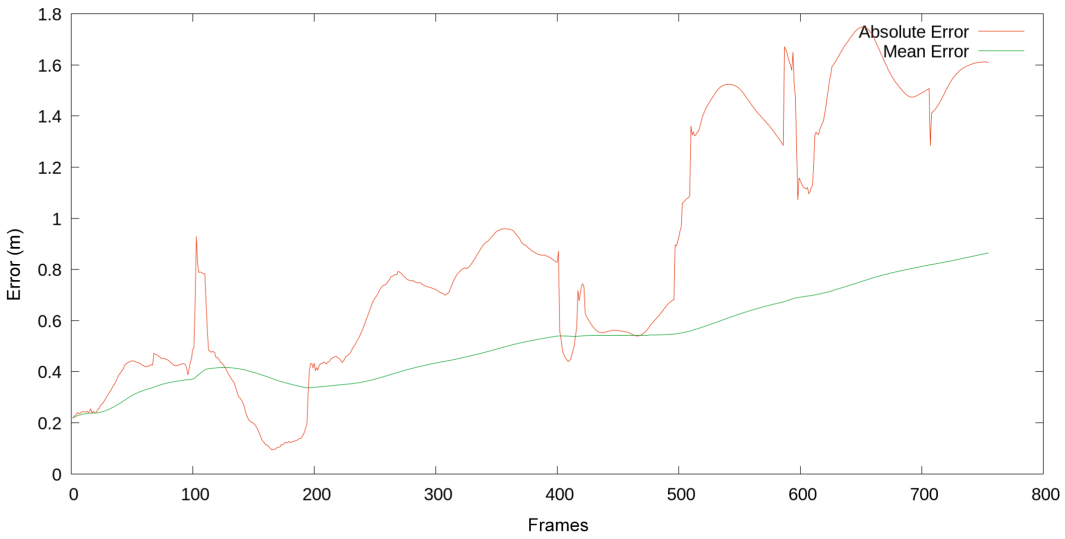


(b) Pose estimation error (Translation).

Figure 3.2: Processing Time and translation error from Kinect Fusion in Living Room Dataset.



(a) VO and Frame processing time.



(b) Pose estimation error (Translation).

Figure 3.3: Processing Time and translation error from Kinect Fusion in Freiburg1_360 Dataset.

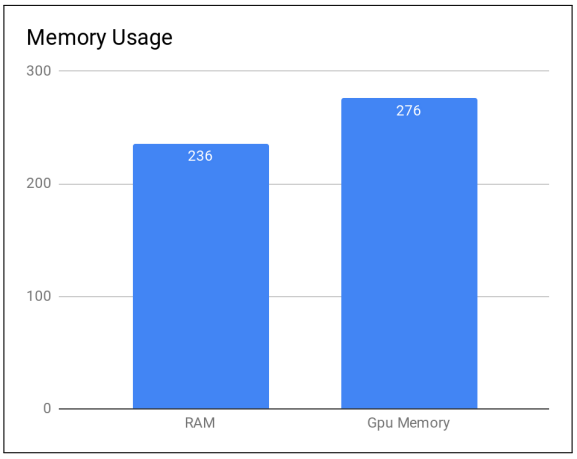
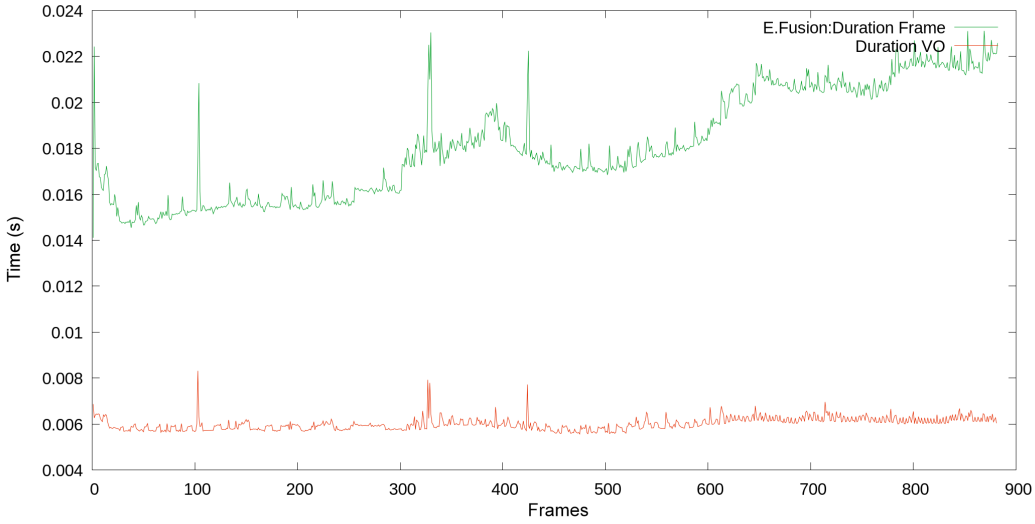


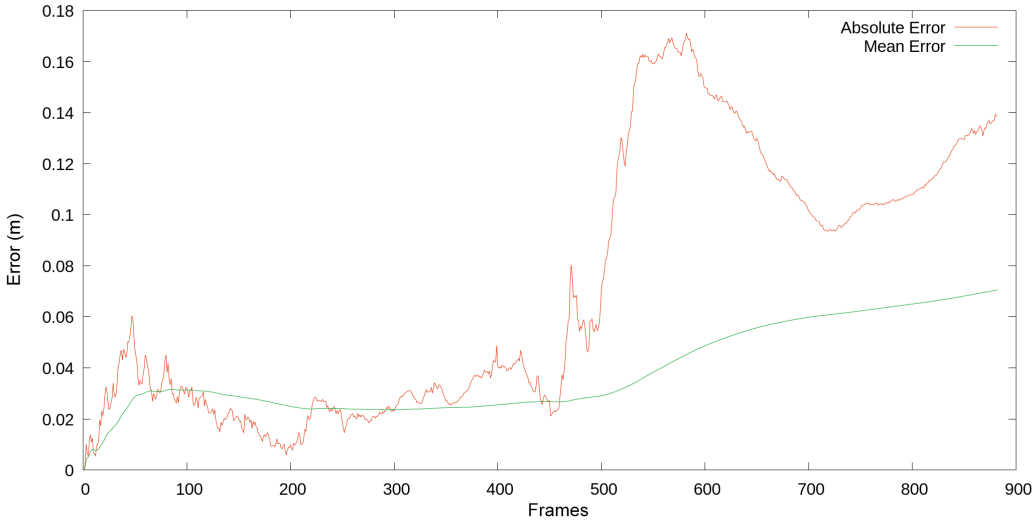
Figure 3.4: Kinect Fusion memory usage.

3.1.3 Performance Evaluation of in the Case of Photometric and Depth Data

At the second case we study the extra computational cost that photometric constrains introduce to ICP. For this scenario, we use the cuda implementation of Elastic Fusion in Slambench2. Similarly to previous case, we measure the processing time per frame, the processing time of VO per frame as well as the memory usage and pose estimation error. As previously, the absolute error is the translation error in meters and the mean error is the average error until the corresponding frame.
Figures 3.5a and 3.6a demonstrate the execution time of Elastic Fusion. As in the previous case, the green line denotes the processing time per frame, while the red line is the processing time of Visual Odometry. While the VO processing time remains relative low, the frame processing time is much higher. The average frame-rate is **55.1** frames/s.
Comparing VO from Kinect and Elastic fusion, the performance difference is insignificant, therefore the extra photometric constrains do not deteriorate the execution time. However the total frame rate of elastic fusion is much lower, probably due to algorithmic differences and not optimized implementation. Moreover the memory usage, as shown in figure 3.7, is much higher. In addition, Figures 3.5b and 3.6b show the translation error of Elastic Fusion. In the Room dataset, Elastic Fusion performs worst than Kinect Fusion. However in the Freiburg1_360 Dataset, Elastic Fusion manages to reduce the translation error probably due to its loop closure mechanism.

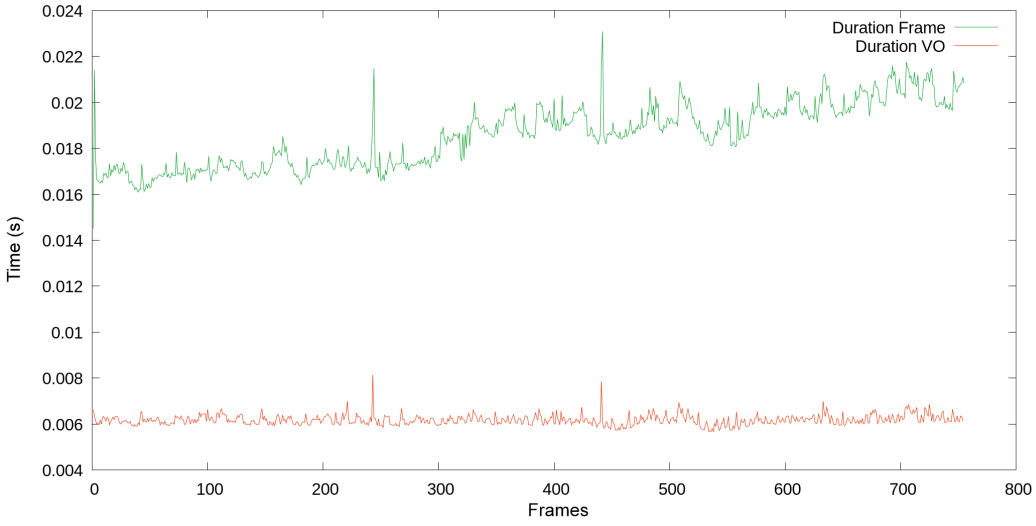


(a) VO and Frame processing time.

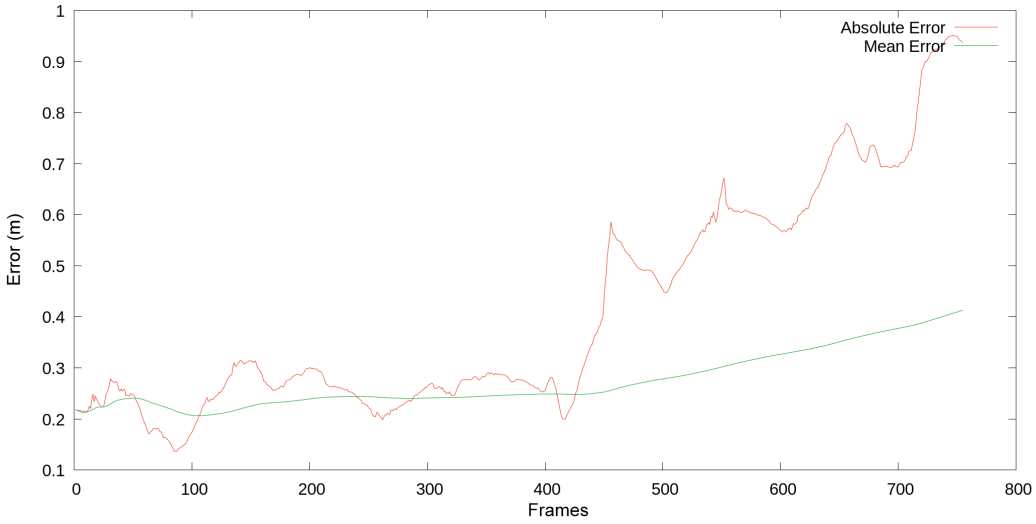


(b) Pose estimation error (translation).

Figure 3.5: Processing Time and translation error from Elastic Fusion in Living Room Dataset.



(a) VO and Frame processing time.



(b) Pose estimation error (translation).

Figure 3.6: Processing Time and translation error from Elastic Fusion in Freiburg1_360 Dataset.

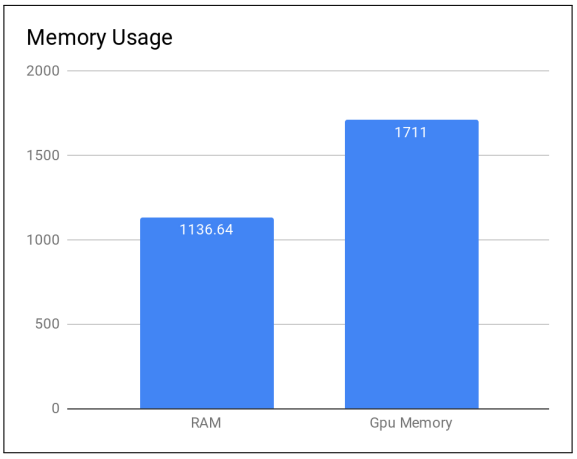


Figure 3.7: Elastic Fusion memory usage.

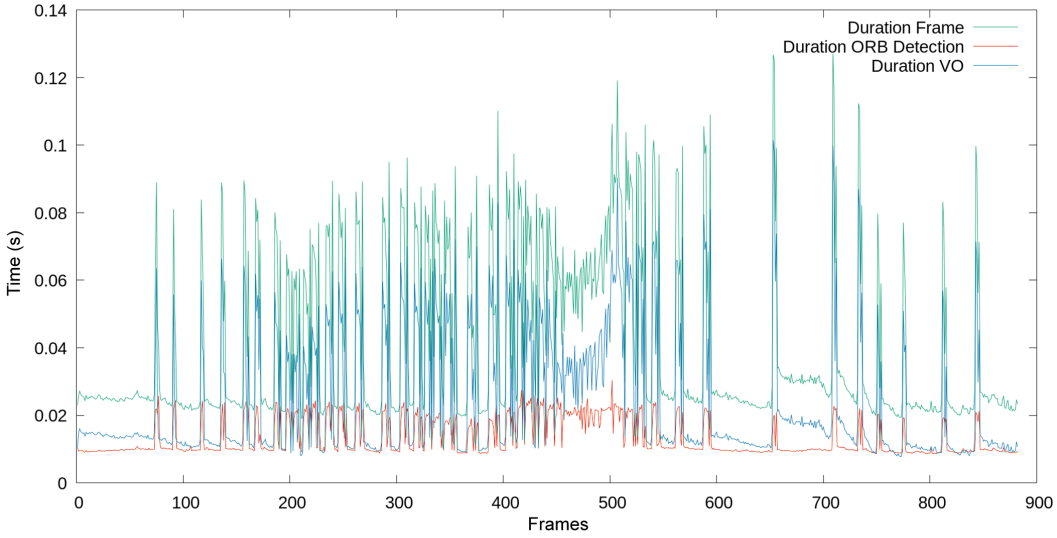
3.1.4 Performance Evaluation of Pose Graph and Loop Closure

In this case we study the requirements of Pose Graph and loop closure mechanism using *ORB-SLAM2* and its Slambench2 implementation. We measured the processing time of VO and ORB detection as well as the total processing time per frame. Furthermore we measured the performance of loop closure which runs on a separate thread. For that reason the duration of loop closure detection was measured in addition to the Pose Graph optimization time. Finally Figures 3.8c and 3.9c present the translation pose estimation error.

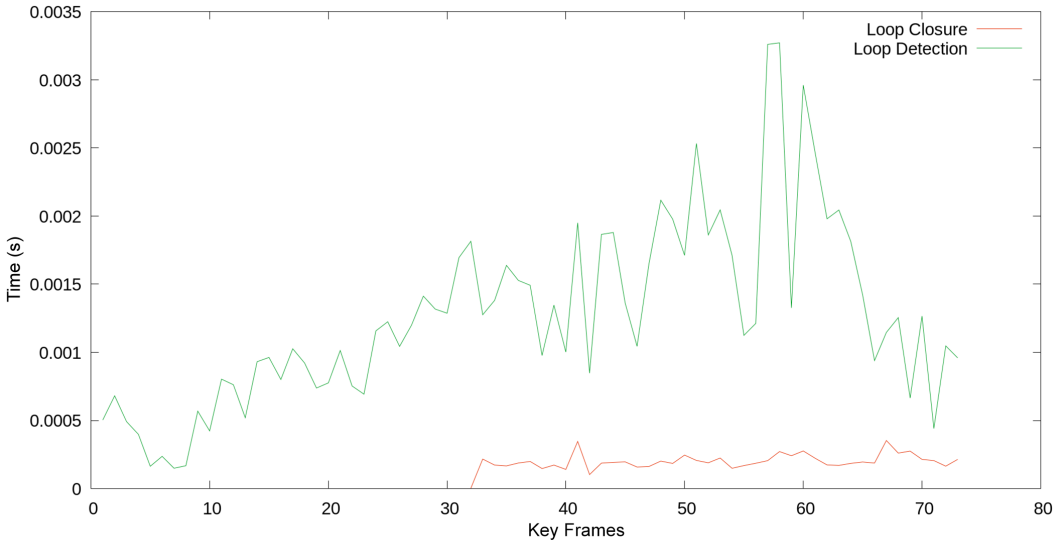
Plots 3.8a and 3.9a demonstrate the total execution time per frame (green line), processing time of ORB detection (red line) and processing time of VO (blue line). Frame proceeding time is mainly the sum of ORB detection and VO proceeding time. This implementation of ORB-SLAM2 is based on CPU, and for that reason the execution time is much slower than the others algorithms.

Moreover graphs 3.8b and 3.9b depict loop closure's performance. The green line is the time lapsed for loop detection whereas the red line is for graph optimization. The x axis in these graphs is the number of key frames. We can observe that the loop detection is more time consuming that the actual graph optimization.

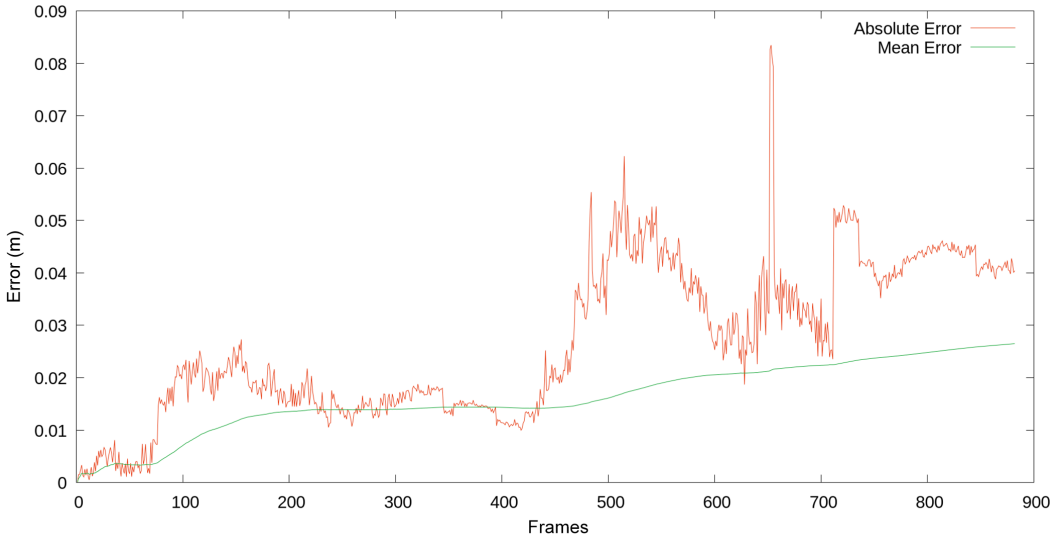
Examining the pose estimation error in figures 3.8c and 3.9c, we notice that the loop closure mechanism limits the accumulated error in the Freiburg1_360 dataset.



(a) ORB Detection, VO and Frame processing time.

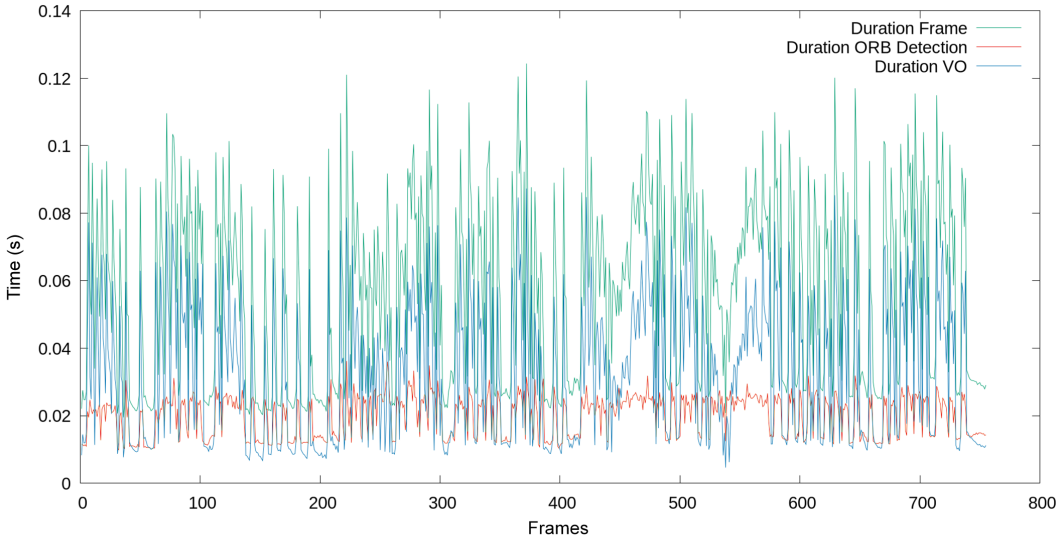


(b) Duration of loop detection and graph optimization.

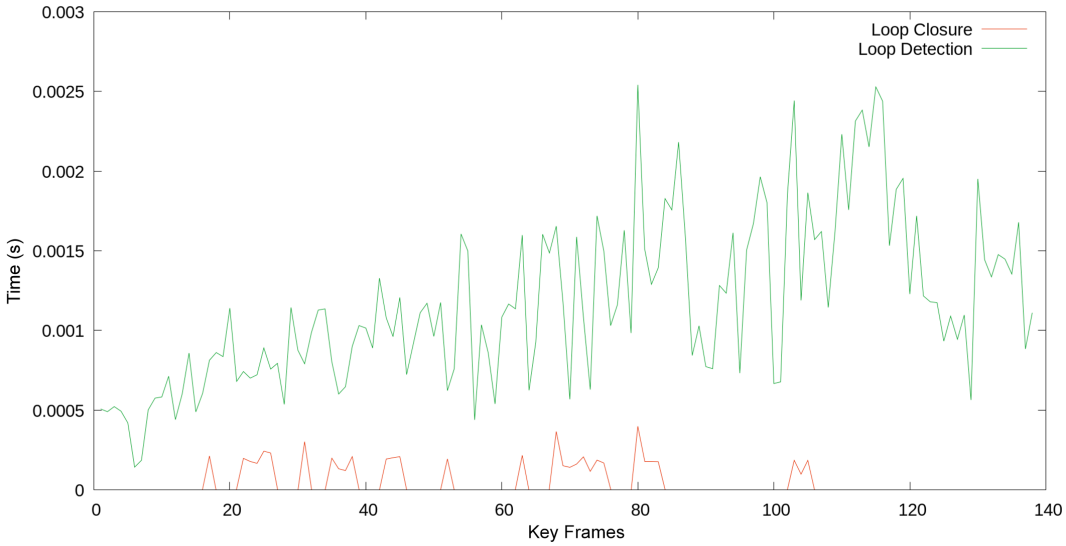


(c) Pose estimation error (translation).

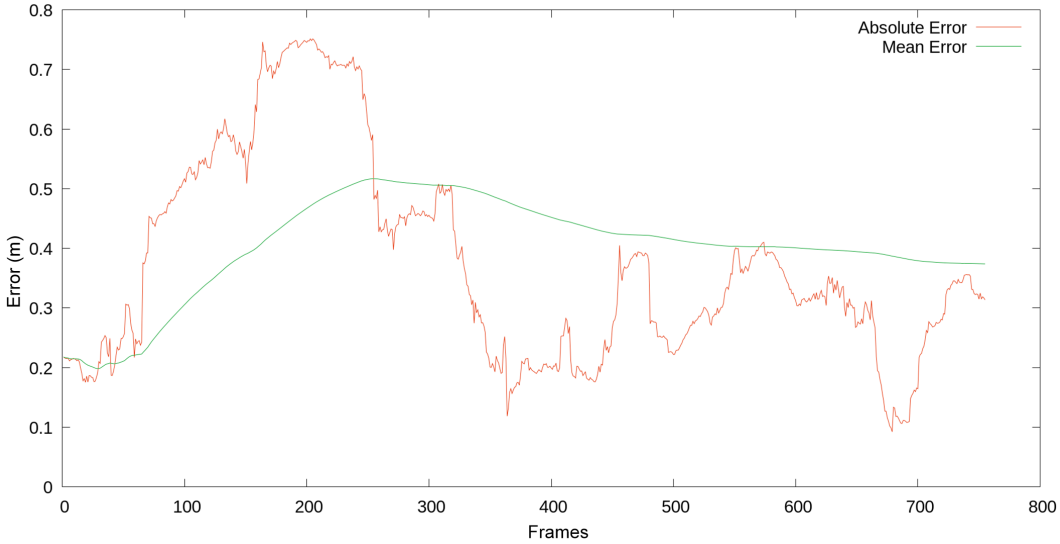
Figure 3.8: Processing Time and translation error from ORB-SLAM2 in Living Room Dataset.



(a) ORB Detection, VO and Frame processing time.



(b) Duration of loop detection and graph optimization.



(c) Pose estimation error (translation).

Figure 3.9: Processing Time and translation error from ORB-SLAM2 in Freiburg1_360 Dataset.

3.1.5 Summary

The performance of the dense algorithms (Kinect Fusion and Elastic Fusion) indicate that high frame-rates can be achieved with parallel GPU implementations even in large data sizes. Kinect Fusion can accomplish a very high frame rate and good memory usage due to its thorough optimization, making it well suited for on-board execution. Furthermore extra photometric constrains do not negatively affect the performance of ICP. Lastly, loop closure mechanisms can reduce the accumulated error but the loop detection problem is computationally harder than the actual graph optimization.

3.2 Evaluation of Aspects Related to Features

This chapter compares *3DMatch* and *3DSmoothNet* geometric descriptors in addition with the well known *Scale-Invariant Feature Transform* (SIFT) features from photometry [25]. Both *3DMatch* and *3DSmoothNet* choose key-points randomly from dept data. Here we also tested a combination of *3DSmoothNet* with Harris Corners as key-point extraction mechanism instead of a random selection. Moreover we preform Principal Component Analysis (PCA) to find the principal components that hold more than 90% of the information. Also we compute the *Distinctiveness* of descriptors as the sum of the eigenvalues of the principal components, *Mikolajczyk* et al. [28] have also used this metric to measure the Distinctiveness of SIFT descriptors.

3.2.1 3DMatch: Learning Local Geometric Descriptors from RGB-D Reconstructions

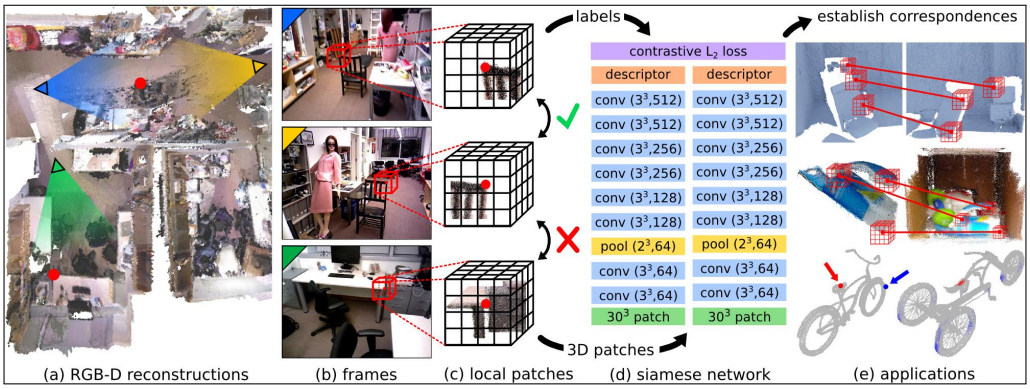


Figure 3.10: 3DMatch Feature Descriptors (from [56]).

3DMatch [56] is a model that learns a local descriptor for establishing correspondences between partial 3D data. It creates a voxel representation of the world using the *Truncated Distance Function* (TDF). TDF is the absolute value of the *Truncated Signed Distance Function* (see section 4.2). This representation loses the distinction between free and occupied space but the largest gradients between voxel values are concentrated around the surfaces. The voxel grid is aligned with respect to the camera view, hence this may deteriorate the rotation invariant property of the descriptor. 3DMatch selects random key points in the grid, and creates local patches of $30 \times 30 \times 30$ voxels around them, as shown in figure 3.10. Then these patches are passes into a 3D convolution network. For every local patch, 3DMatch creates a descriptor with 512 elements. Finally the local patches are aligned using the RANSAC algorithm.

3.2.2 3DSmoothNet

3DSmoothNet [10] utilizes convolutional networks in order to calculate local descriptors from depth data. It employs a novel voxelization method called *Smoothed Density Value* (SDV). SDV relies on the *Local Reference Frame* (LRF) in order to create rotation invariant descriptors. LRF produces a rotation matrix based on the covariance of the points inside a neighborhood. Then these points are rotated and thus a rotation invariant representation is achieved. Next 3DSmoothNet creates the actual voxel grid, whose elements are computed using a Gaussian smoothing kernel. Gaussian smoothing results in a more

dense grid than other voxel representations and in addition reduces the noise of the data. Finally, 3DSmoothNet passes this grid into a stack of multiple convolutional layers, producing a descriptor with 32 elements. 3DSmoothNet also chooses key-points randomly and aligns the point clouds using the RANSAC algorithm.
Figure 3.11 depicts the pre-processing operations that 3DSmoothNet performs on the data. Image (*a*) shows the initial input data. Then 3DSmoothNet calculates the LRF matrix (image (*b*)) and rotates the data accordingly (image (*c*)). Finally in image (*d*), 3DSmoothNet creates the vocalized representation and applies the aforementioned Gaussian smoothing (image (*e*)).

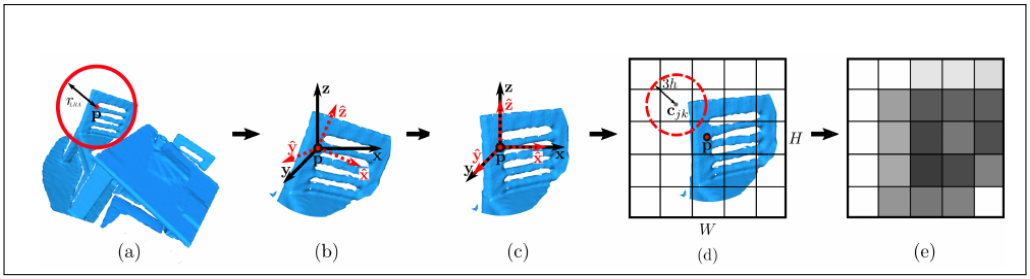


Figure 3.11: (a)Input image. (b) LRF (c) Data rotation (c) Voxelization (c) Smoothing (from [10]).

3.2.3 Scale-Invariant Feature Transform - SIFT

Scale-Invariant Feature Transform or SIFT features are well tested and highly used in computer vision [25]. The SIFT algorithm is based on the Difference of Gaussians (DoG) to extract key-points and feature descriptors. Initially, it creates a pyramid scheme of octaves and scales. Each octave has half the size of the previous one. Within an octave, there are multiple blurred versions of the image, produced by Gaussian smoothing at different widths (scales). Then the subtraction of the images between consecutive scales forms the Difference of Gaussians or DoG. Afterwards, key-points are extracted on the maxima and minima values of the DoG images. Moreover the eigenvalues of the Hessian matrix of DoG provide information about the orientation of the descriptor, thus accomplishing rotation invariance. Figure 3.12, illustrates the multiple layers of the image and the DoG representation.

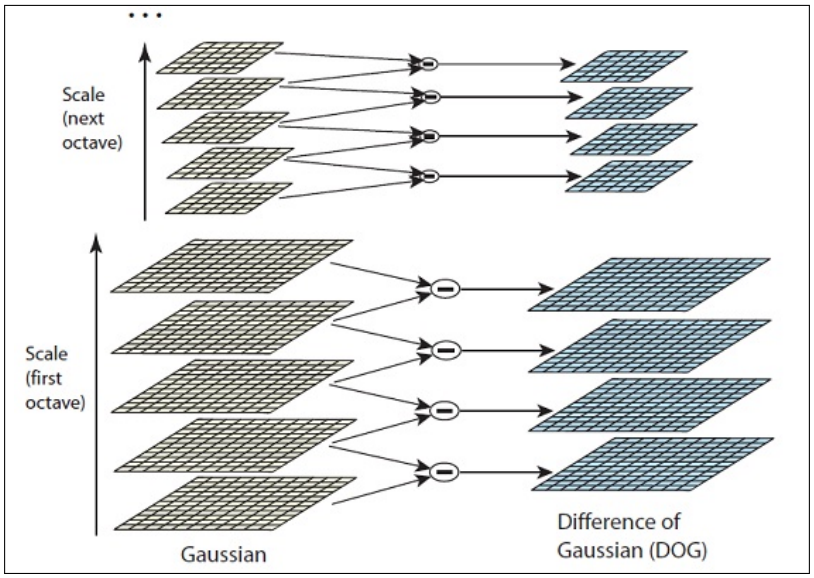
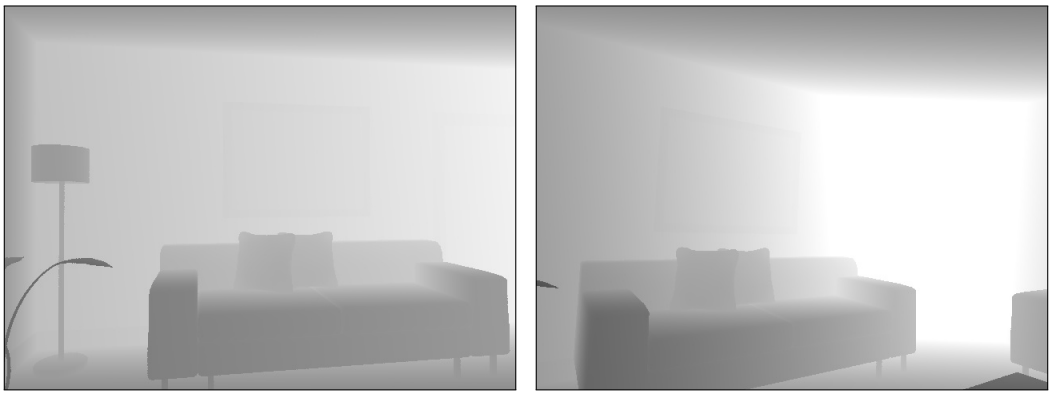


Figure 3.12: Octaves, Scales and Difference of Gaussian (from [25]).

3.2.4 Experimental Setup

The experiments described in this section were conducted with two images from the Room dataset (see section 3.1.1). The images display a sofa in a slightly rotated angle. Figure 3.14 shows these RGB data of these images while Figure 3.13 shows the depth information. The depth of these data is aligned to RGB. Additionally, the camera displacement between these two images is known from the ground truth trajectory and it is indicated in Table 3.1.



(a) Sofa:position 1 (b) Sofa:position 2

Figure 3.13: Depth data of sofa from two different angles.

(a) Sofa:position 1 (b) Sofa:position 2

Figure 3.14: RGB data of sofa from two different angles.

Table 3.1: Ground truth displacement

x(m))	y (m)	z (m)	roll(deg.)	pitch(deg.)	yaw(deg.)
-0.519	0.021	0.092	-2.779	22.944	2.020

We conducted four experiments and we tested the execution time, distinctiveness of descriptors as well as the transformation between these two images. The transformation was calculated either with RANSAC or kNN. In case of RANSAC, the algorithm chooses random points from the key-points set of the first image. The initial correspondences between these points and the key-points of the second image are detected with the Nearest Neighbor method applied to descriptors. RANSAC also excludes outliers by checking if the two point clouds build polygons with similar edge lengths. Specifically it checks if the lengths of any two arbitrary edges within the first set of key-points is similar to the edge formed by the corresponding points of the second key-point set. For the implementation of RANSAC the *Open3D* library was used.
In case of kNN, the two Nearest Neighbors were detected. Then a point was considered as inlier if the ratio of first Nearest Neighbor to the second one was less than a threshold value, in this case 0.7. This method assures that the descriptor distances between the inliers are much closer than the other points. Then the final transformation was calculated with SVD. For this method the *OpenCV* library was used.

3.2.5 Local Descriptor Consistency

In this section we studied the consistency of the local descriptors from 3DMatch and 3DSmoothNet. Initially, random key-points were selected from the first image. Then these key-points were transformed with the ground truth transformation. Points outside the common field of view were excluded. With this approach we obtained the ground truth correspondence between key-points. Figure 3.15 shows the selected key-points with red dots.

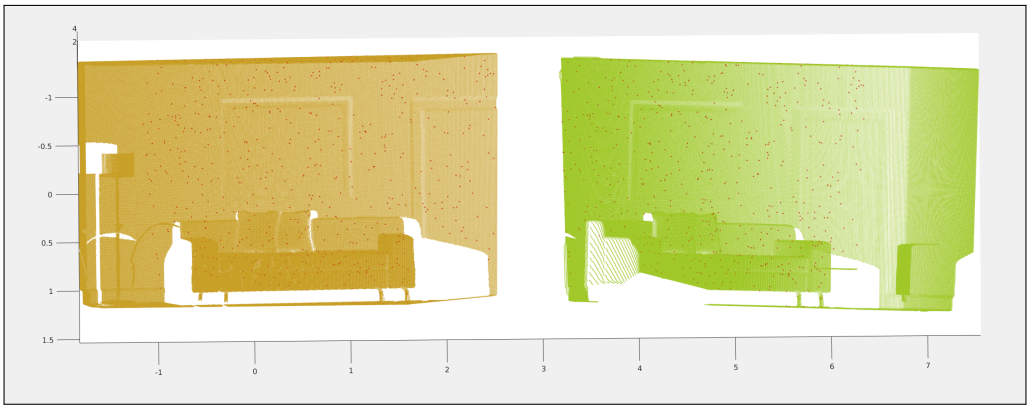


Figure 3.15: Key-point on the first image was chosen randomly. Then it was transformed with gt transformation at the second image.

Generally we expect the descriptors from the corresponding key-points to be close with respect to their Euclidean Distance. In addition we measured the average distance of the whole descriptor set. Finally we performed Principal Component Analysis in the descriptors of both algorithms. We determined the number of Principal Components with more than 90% variance, as well as the summation of the eigenvalues as proposed in [28]. The table bellow (Table 3.2) contains the results of these measurements.

	Correspondence dist.(avg)	total dist.(avg)	PCA (90%)	PCA(sum)
3DMatch	0.121	0.511	4 / 512	0.153
3DSmoothNet	0.328	0.866	18 / 32	0.498

Table 3.2: Euclidean distance of descriptors and PCA.

As expected, the descriptors of the corresponding key-points are closer than the average distance. Moreover PCA shows that *3DMatch* has very few principal components, only 4 from a set of 512. Thus, despite the huge size of the descriptor vector, there is no much information in it. On the contrary *3DSmoothNet* performs better with 18 principal components out of a set of 32.
Next we determine again the correspondence between the key-points with Nearest Neighbor (NN) and RANSAC and compare it against the ground truth. Figures 3.16 and 3.17 contains a visualization of the correspondences generated from the Nearest Neighbor algorithm of 3DMatch and 3DSmoothNet respectively. Moreover, Figures 3.18 and 3.19 display the correspondences from the RANSAC algorithm. For visualization purposes, only a sub-set of 100 key-points was used.

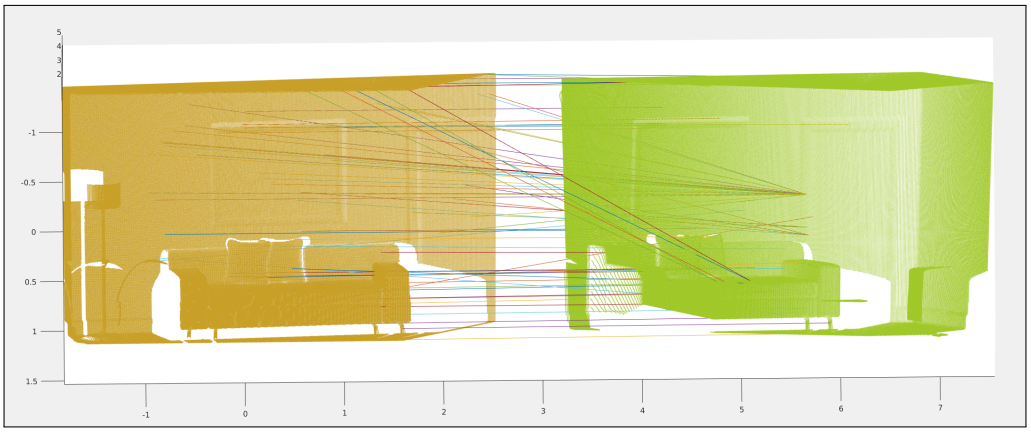


Figure 3.16: 3DMatch - Nearest Neighbor (100 points).

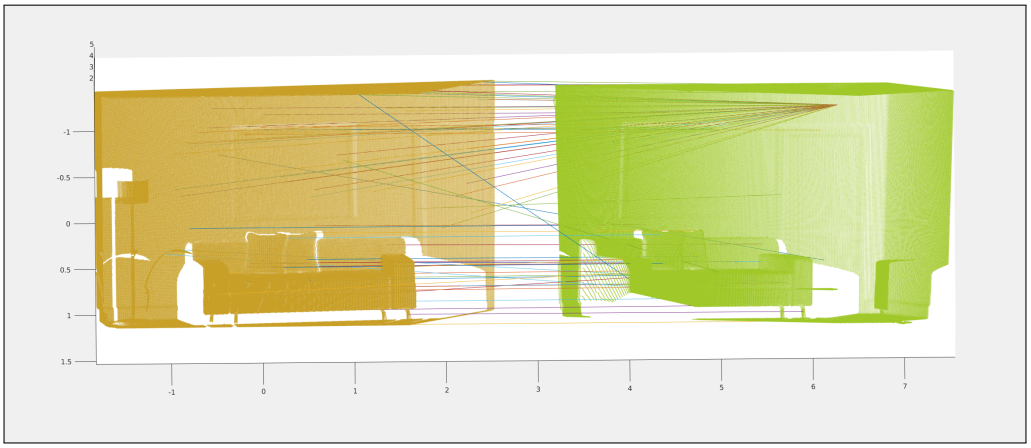


Figure 3.17: 3DSmoothnet - Nearest Neighbor (100 points).

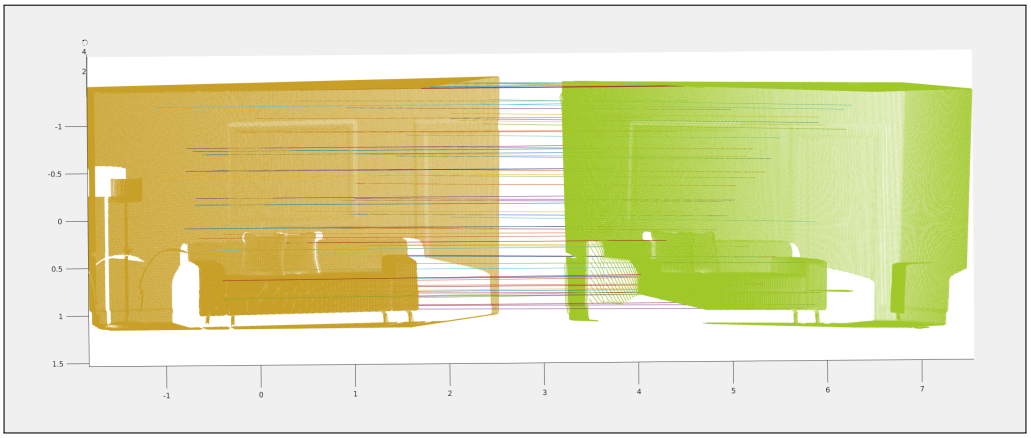


Figure 3.18: 3DMatch - RANSAC (100 points).

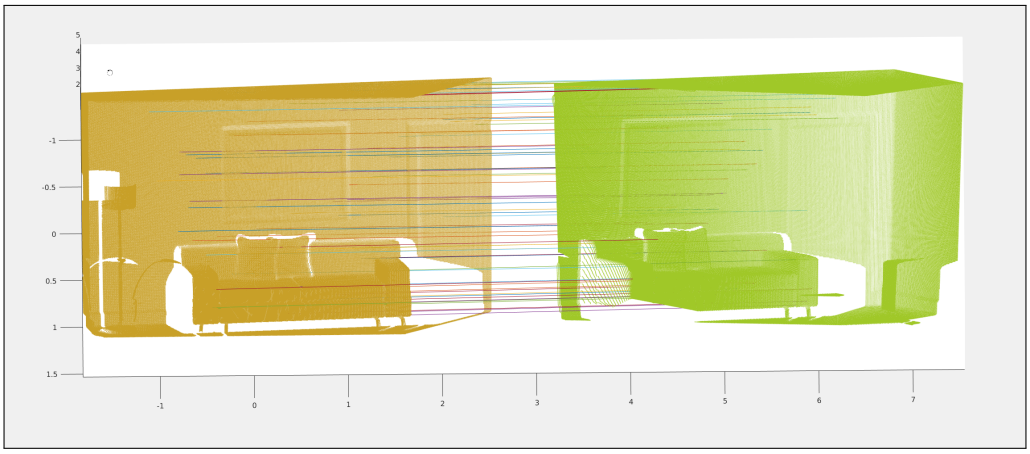


Figure 3.19: 3DSmoothnet - RANSAC (100 points).

Table 3.3 shows the success rate of the two algorithms. Evidently, Nearest Neighbor fails to find accurate correspondences. On the contrary, RANSAC performs well with both 3DMatch and 3DSmoothNet.

	NN	RANSAC
3DMatch	13.99%	97.19%
3DSmoothNet	28.81	98.32%

Table 3.3: Nearest Neighbor and RANSAC success rate.

3.2.6 Correspondence with Random Key-points

In this experiment we calculated two random sets of key points for the two images. For every set, 1000 key-points were chosen uniformly. Then the camera displacement was calculated using the RANSAC algorithm. Table 3.4 contains the camera transformation error compared to ground truth. In addition Table 3.5 indicates the percentage of inliers of RANSAC (fitness) as well as the execution time of feature extraction (average execution time of two frames). In case of *3DSmoothNet*, a heavy pre-processing operation takes place on the CPU for the calculation of LRF and SDV representation. Hence we measured the execution time of LRF and SDV separately.

	x	y	z	roll(deg.)	pitch(deg.)	yaw(deg.)
3DMatch	0.012	0.005	0.001	-0.693	0.285	0.032
3DSmoothNet	0.000	0.000	0.010	-0.014	0.000	-0.555

Table 3.4: Transformation error (m).

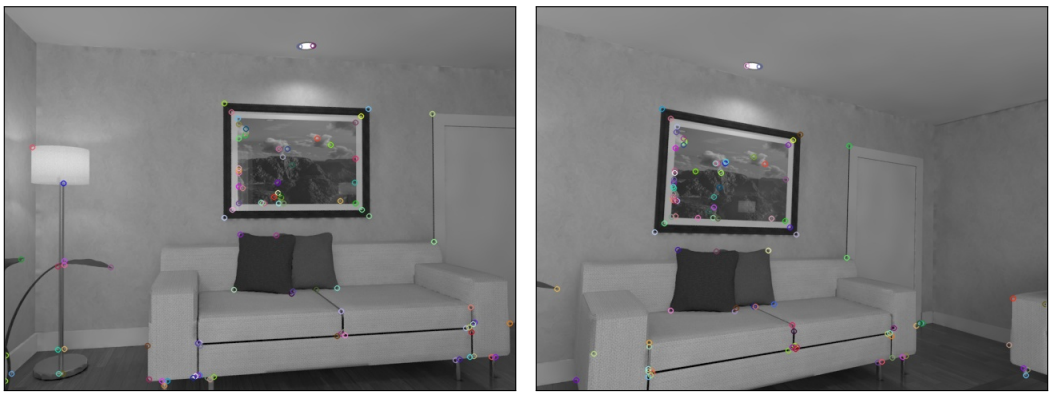
	LRF	SDV	CNN	TOTAL	Fitness
3DMatch	-	-	3402	3402	25.9 %
3DSmoothNet	5350	5382	2129	12861	27.0 %

Table 3.5: Operation time (ms) and RANSAC fitness

Considering the transformation error, 3DSmoothNet is slightly better in both, translation and rotation. However, the LRF and SDV calculations of the 3DSmoothNet are computational heavy and even slower than the actual neural network. This makes 3DSmoothNet's execution time more that 12 seconds. Nevertheless, higher parallelization can be achieved with GPU implementation on LRF and SDV calculations.
Regarding the fitness of RANSAC, both show very low percentage of inliers (25.9% and 27.0% respectively), which is due to random key-point selection. Random selection offers no guarantee that there would be any correspondence. For that reason we have to select a large amount of key points, deteriorating execution time and yet accomplish very small fitness. Small fitness also makes RANSAC prone to error. In addition it is unclear if small fitness is due to bad alignment or due to minimum key-points correspondence because of randomness. For those reasons, it is essential to employ a better key-points extraction technique.

3.2.7 Harris Corners

The above experiments show that random key-point selection mechanism is not appropriate. In this experiment we extracted Harris corners and used them as key-points. Harris corners were extracted from RGB and then, the corresponding depth vertex was used as key-point. The implementation of the extraction was based on the *OpenCV* library and specifically the *good feature to track* method [14] with Harris corners parameter activated. The number of corners that were found was **100**.
We evaluated this method only on 3DSmoothNet due to better PCA results. The next images (Figure 3.20) display the selected key-points.



(a) Harris Corneres - position 1 (b) Harris Corneres - position 2

Figure 3.20: Harris Corners of two images.

The final transformation between these two images was computed with RANSAC, similarly to the previous experiment. Table 3.6 shows the transformation error of RANSAC against the ground truth. In addition Table 3.7 contains the execution time of feature extraction and PCA results.

x	y	z	roll(deg.)	pitch(deg.)	yaw(deg.)
0.004	0.006	0.001	0.012	-0.115	-0.110

Table 3.6: Transformation error (meters).

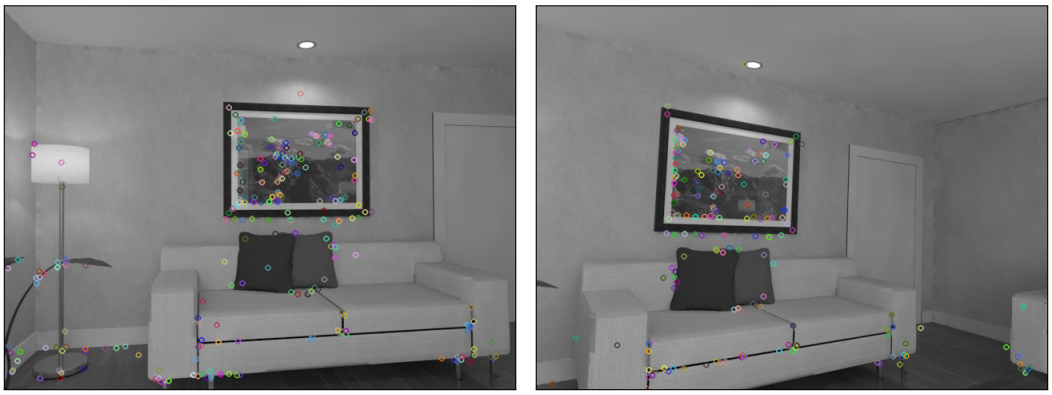
Harris	LRF	SDV	CNN	TOTAL	Fitness	PCA (90%)	PCA(sum)
6.4	666	713	822.79	2208.19	70 %	14 / 32	0.630

Table 3.7: Operation time (ms), RANSAC, PCA.

The results indicate a slightly worse transformation error and fewer principal components. The error however remains considerably low. Moreover, we observe a major increase of the fitness of RANSAC with way less key-points. Fewer key-points also lead to better execution time. Nevertheless Harris Corners are based on RGB information, hence the feature extraction mechanism is no more pure geometric.

3.2.8 SIFT Features

At the previous experiment we created descriptors by combining photometry and geometry of the scene. If, however, we utilize the photometry of the scene, then SIFT is a reliable and well tested descriptor. In this section we tested the SIFT descriptors using the same images. For every key-point in the image plane, we took the associated vertex in 3D space. Then we found the correspondence between these points with two different methods: RANSAC and kNN. Regarding the kNN method, we found the two nearest neighbors of a key-point and we added this point to the matching set only if their distance ratio was below a threshold, in our case 0.7. Then the final transformation was calculated with *SVD*.
Figure 3.21 displays the selected key-points. In addition Figures 3.22 and 3.23 show the correspondences found with kNN and RANSAC respectively. Moreover Table 3.8 contains the transformation error of kNN and RANSAC and Table 3.9 contains the percentage of inlier with these two methods. Finally, Table 3.10 shows the number of principal components with more than 90% variance and the distinctiveness of the SIFT descriptors.



(a) SIFT - position 1 (b) SIFT - position 2

Figure 3.21: SIFT features of two images.

	x	y	z	roll(deg.)	pitch(deg.)	yaw(deg.)
kNN	0.004	0.023	0.009	-0.308	-0.184	-0.263
RANSAC	0.001	0.007	0.000	0.166	0.159	0.133

Table 3.8: Transformation error (m).

	Fitness
kNN	39.92%
RANSAC	67.37%

Table 3.9: kNN and RANSAC fitness.

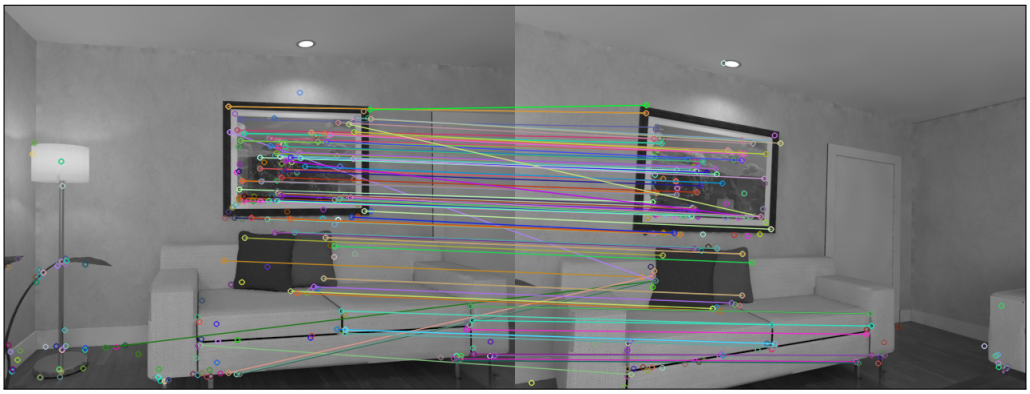


Figure 3.22: kNN correspondence.

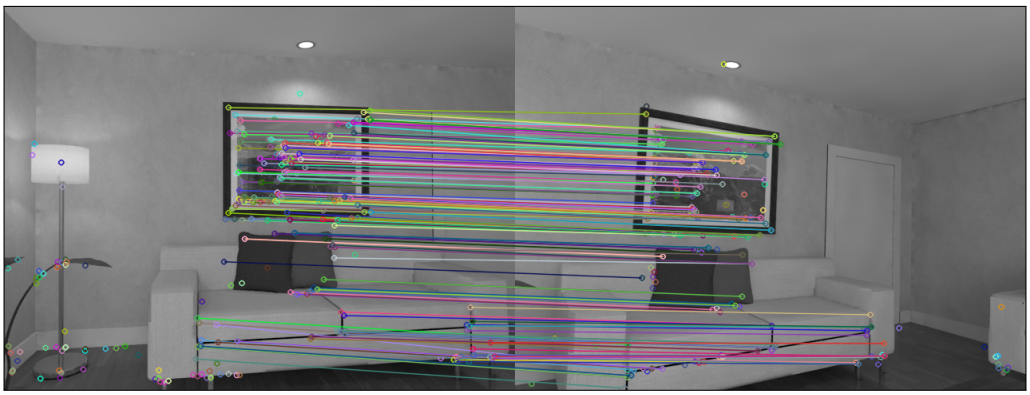


Figure 3.23: RANSAC correspondence.

	PCA (90%)	PCA(sum)
SIFT	36 / 128	1.4520e+05

Table 3.10: Principal components.

This experiment shows that kNN performs well in this task. kNN excludes a large amount of key-points and thus, it suffers of small fitness (39.93%). Nevertheless the remaining points achieve a relative small transformation error. RANSAC also performs well with more inliers. Considering the Principal Component Analysis, SIFT descriptors have the 90% of their variance in 36 principal components. This indicates that they have more information than the previous tested methods. Finally the average feature extraction time was **43.4ms** making SIFT extraction faster than *3DMatch* and *3DSmoothNet*.

3.2.9 Summary

The following tables contain a summary of the results of the experiments in this section. Table 3.11 shows the transformation error of all the tested methods and Table 3.12 contains the execution time for each descriptor. Finally, Table 3.13 includes the results of PCA and the distinctiveness for each descriptor.

	x	y	z	roll(deg.)	pitch(deg.)	yaw(deg.)
3DMatch	0.012	0.005	0.001	-0.694	0.285	0.032
3DSmoothNet	0.000	0.000	0.010	-0.014	0.000	-0.555
3DSmoothNet+Harris	0.004	0.006	0.001	0.012	-0.115	-0.110
SIFT kNN	0.004	0.023	0.009	-0.308	-0.185	-0.263
SIFT RANSAC	0.001	0.007	0.000	0.166	0.159	0.133

Table 3.11: Transformation error (m).

3DMatch	3DSmoothNet	3DSmoothNet+Harris	Sift
3401.7	12860.7	2208.19	43.4

Table 3.12: Total operation time (ms).

	Size	PCA (90%)	PCA(sum)
SIFT	128	36	1.4520e+05
3DMatch	512	4	0.1527
3DSmoothNet	32	18	0.4976
3DSmoothNet+Harris	32	14	0.6295

Table 3.13: Distinctiveness of descriptors (PCA).

Generally, both neural network algorithms are much slower than conventional methods such as SIFT and it is difficult to include them in a real time SLAM system. In addition, they show small distinctiveness and few principal components. For that reason, Nearest Neighbor fails to find accurate correspondences. However, a better key-point extraction mechanism in combination with RANSAC, may produce accurate estimations of camera displacement.

Chapter 4

Kinect Fusion: Proposed Modifications

This section introduces two modifications of Kinect Fusion. Kinect Fusion was the vSLAM system employed in this Thesis due to the achieved high frame-rate and low memory usage, as indicated in Section 3.1. The first modification combines the VO from Kinect Fusion with kinematics using the SEROW state estimation system [35]. This mechanism reduces the pose estimation error and achieves real time execution on embedded devices. The second modification of Kinect Fusion incorporates VO together with SIFT features into a common Pose Graph. This modification manages to reduce the accumulated error at the expense of offline execution.

4.1 Kinect Fusion

Kinect Fusion was the first algorithm that used the Microsoft Kinect sensor and its depth capabilities. It approaches the SLAM problem in a depth frame-to-model method and it achieves real time execution due to its GPU based implementation. The map representation of Kinect Fusion is solely stored in GPU with a dense volumetric representation as described in section 4.2.
The workflow of Kinect Fusion, as shown in Figure 4.1, can be separated into four procedures:

- Preprocessing
- Tracking (VO)
- Integration
- Raycasting

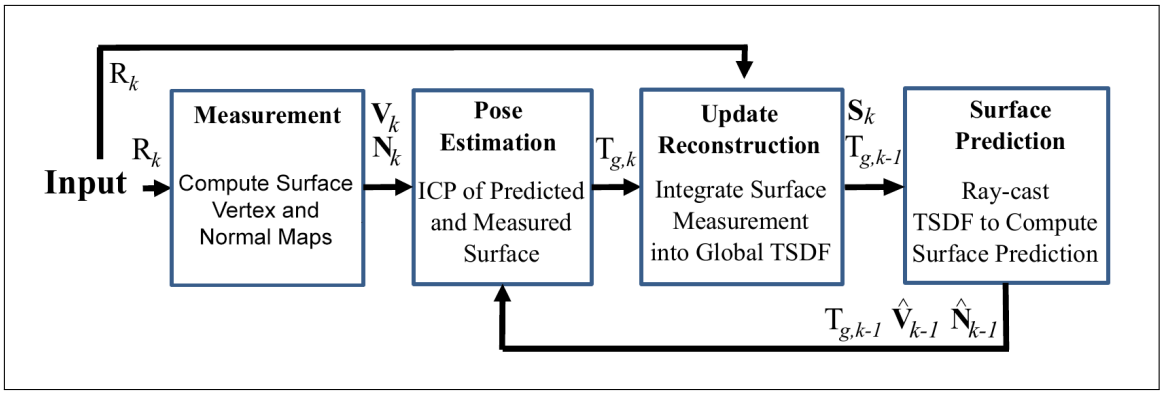


Figure 4.1: Kinect Fusion workflow (from [13]).

Initially Kinect Fusion pre-processes the input data by applying a bilateral filter kernel for noise reduction. Then it produces the surface vertexes and normal maps for input data. Next, Kinect Fusion executes the tracking process by aligning the input data with the model's vertices and normal map from the previous camera position. The alignment is found with a point-to-plane ICP in coarse to fine pyramid scheme. After tracking, the new camera pose is calculated. Using this pose, Kinect Fusion integrates the input data into the model. Afterwards the algorithm commits raycasting in order to determine the new model vertices and normals.

4.2 Truncated Signed Distance Function - TSDF

Truncated Signed Distance Function (TSDF) is a volumetric scene representation that allows integration of multiple depth images taken from different viewpoints. It has several benefits like time and space efficiency, representation of uncertainty and incremental updates. Moreover it supports high parallelization, thus it is well suited for GPU implementation. TSDF is an extension of *Singed Distance Function* (*SDF*) proposed in [6]. The Signed Distance Function consists of a three dimensional grid divided into voxels. Each voxel has two relevant values, the signed distance function sdf_i and its weight w_i.
$sdf_i(x)$ is the distance of the center of voxel x from the nearest surface, towards the direction of the measurement. Hence positive values denote the free space, while negative values are behind or inside the objects. Furthermore, the zero levels of SDF form the actual surface. So it is clear that large distances (either negative or positive ones) do not contribute to surface reconstruction. For that reason, such values can be truncated in order to maintain numerical stability and reduce memory footprint. These *truncated sdf* values form the *Truncated Signed Distance Function* (*TSDF*). The following formula describes the *truncation* of signed distance function.

$$tsdf_i(x) = \min(1, \max(-1, sdf_i(x))) \tag{4.1}$$

where:

$$i = \text{is the sequence number of measurement.}$$

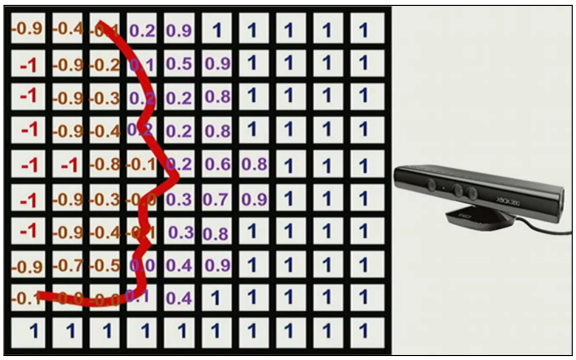


Figure 4.2: TSDF volume representation

As mentioned above, voxels also have a weight value. Weights indicate the uncertainty of the voxel and can be used to fuse multiple measurements together. A voxel x holds the $TSDF_i(x)$ and $W_i(x)$ values for frame i. When a new measurement of x arrives, these values have to be updated. The new measurement has a new $tsdf_i(x)$ and also uncertainty $w_i(x)$. The following formula fuses a new measurement to voxel grid.

$$TSDF_i(x) = \frac{W_{i-1}(x) \cdot TSDF_{i-1}(x) + w_i(x) \cdot tsdf_i(x)}{W_{i-1}(x) + w_i(x)}$$
$$W_i(x) = W_{i-1}(x) + w_i(x) \tag{4.2}$$

A common approach for the uncertainty is to set $w_i(x) = 1$ for every voxel update. This simple technique averages measurements over time. This is also the approach that follows Kinect Fusion. In addition the initialization of voxel grid is important. Usually the $TSDF_0$ is set to one to mark all the voxels in the grid as empty and W_0 is set to zero.
For surface reconstruction the zero level of TSDF must be found. This can be done by raycasting from a given camera viewpoint. Rays pass through voxels until there is a change in sign. Then the actual zero crossing is calculated via interpolation.
More information about the TSDF and its parameters can be found in [48].

4.3 Kinect Fusion and SEROW

The *State Estimation Robot Walking* or *SEROW* [35] is a home-made software module that estimates the state of the Center of Mass (CoM) for humanoid robots. It incorporates multiple sources including VO in order to produce more accurate estimations. The current section proposes an algorithm that combines Kinect Fusion with SEROW.
SEROW incorporates VO independently from the method that produces it. In this case we chose Kinect Fusion since it is well suited for embedded devices and its structure allows separation of VO and Mapping. SEROW fuses VO form Kinect Fusion with kinematics improving the accuracy of pose estimation. Subsequently, the refined poses are used for mapping, achieving better map representation. This in turn leads to even more accurate estimation and so on. Moreover, the whole system is lightweight and can run on embedded devices such as Jetson tx2, maintaining high frame rates. Finally, we tested and evaluated this system in real world experiments. This work has been published in [36] and the implementation is publicly available in [47].

4.3.1 State Estimation RObot Walking - SEROW

Estimating the state of CoM is essential for stable walk and gait planning in humanoids. Generally gait planning is a very difficult task due to the large number of degrees of freedom and the highly non-linear rigid-body dynamics of humanoids. For that reason, the dynamics are often approximated with mass concentrated models and consequently the estimation of CoM is crucial. SEROW estimates accurately the 3D position of CoM as well as its velocity and acting external forces.

Briefly, it fuses measurements from different sources such as joint encoder, IMU, Force/Torque and VO in a modular and probabilistic way.
Formally it estimates the following vector:

$$x_t = [{}^b v_b \quad {}^w R_b \quad {}^w p_b \quad b_w \quad b_a]^T \tag{4.3}$$

where:

$${}^b v_b \quad = \text{linear velocity}$$
$${}^w p_b \quad = \text{base position with respect to world frame } w$$
$${}^w R_b \quad = \text{base rotation with respect to world frame } w$$
$$b_w, b_a = \text{IMU biases, in the base frame b.}$$

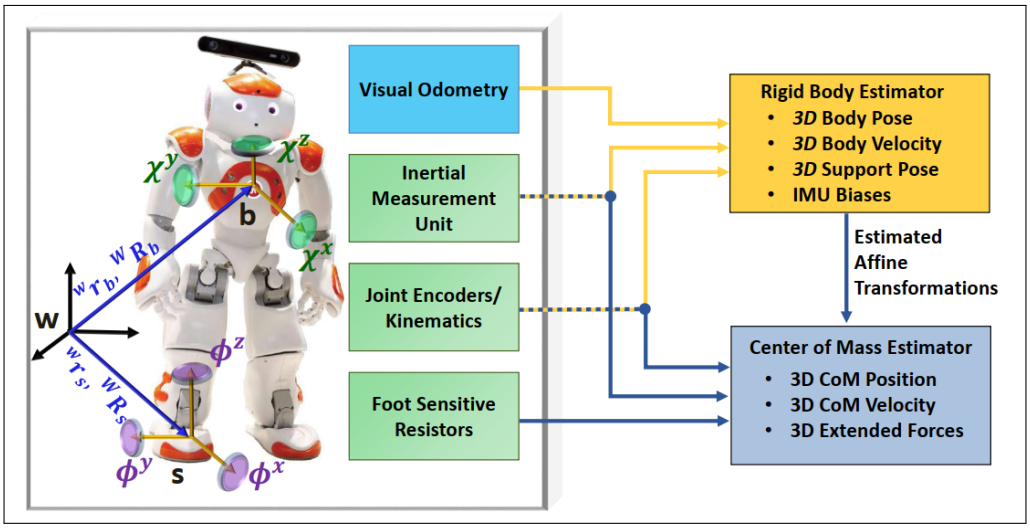


Figure 4.3: SEROW architecture (from [35]).

The estimation scheme of SEROW (Figure 4.3) consists of two cascading Extended Kalman Filters, which furnish it with low computational complexity and hence appropriate for on-board execution.

4.4 Architecture

In order to combine SEROW with Kinect Fusion we have to separate the second into two different functionalities.

- Visual Odometry (VO).

- Mapping.

Then SEROW stands between them as shown in Figure 4.4.

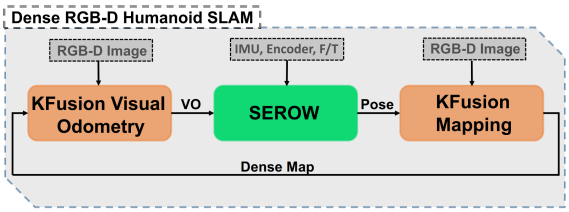


Figure 4.4: Kinect Fusion and SEROW architecture.

VO operates in the usual way by aligning the input data against the model's vertices and normals. Next the output of VO is directed to SEROW which produces a more accurate estimation that is used later for map integration and raycast. With that technique we have successfully enhanced vSLAM with information from IMU, encoder, and F/T measurements.

4.4.1 Implementation

For this work we port the implementation of Kinect Fusion from Slambench2 [3] into ROS. Although there are other implementations of Kinect Fusion, we showed in chapter 2 that this implementation achieves high accuracy and frame rate. Meanwhile SEROW is already implemented in ROS. Therefore a standalone ROS node was developed for Kinect Fusion independent of SEROW. Figure 4.5 depicts the ROS structure with nodes as rectangles and topics noted as arrows.

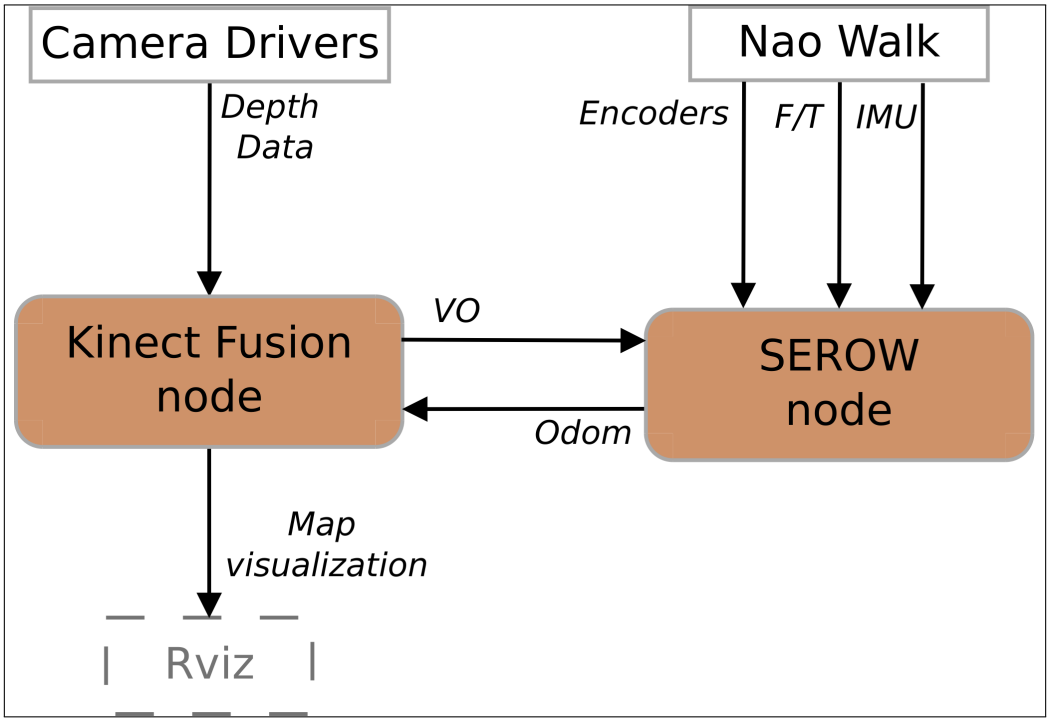


Figure 4.5: ROS nodes and topics as arrows.

4.5 Kinect Fusion with Pose Graph

ICP based algorithms accumulate error over time. The algorithm presented in this section bounds the accumulated error by incorporating a Pose Graph in dense data. Poses from ICP are integrated in the graph as nodes. Furthermore, in every key frame sparse features are extracted from images and added in the graph as landmark nodes. SIFT features are used for this purpose due to their robustness and established means to estimate their uncertainty [55]. The correspondence between SIFT features from two consecutives key-frames is found with the Nearest Neighbor algorithm and then a novel method named *TEASER++* [53] is applied to remove outliers. TEASER++ was published recently and outperforms RANSAC. Finally after optimization, depth information is removed from the map, and re-integrated with the new poses.

4.6 Pose Graph

Pose graph consists of pose and landmark nodes as well as edges connecting them. An edge represents a measurement or a displacement from the source node to the target one. Together with the measurement, a covariance matrix represents its uncertainty. Consecutively, at every frame a new node is added to the graph connected thought the ICP measurement. Moreover, at every key-frame, SIFT features are extracted and incorporated into the graph as landmarks.
Figure 4.6 offers a visualization of a Pose Graph. The blue nodes are the camera poses connected with VO measurements. In addition the green nodes are landmarks extracted by the SIFT algorithm.

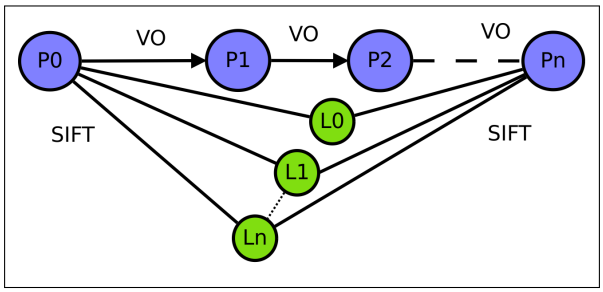


Figure 4.6: Pose Graph: blue nodes are poses while green nodes are landmarks.

Also the uncertainty of the above measurements is needed in order to integrate them in the Pose Graph. Generally we assume that the measurements follow a normal distribution, hence their uncertainty is formalized as a covariance matrix. Sections 4.6.1 and 4.6.2 describe the calculations of ICP covariance and SIFT covariance, respectively. For the implementation of the Pose Graph the *ISAM* [15] library was used.

4.6.1 Covariance of Visual Odometry

The covariance of visual odometry can be calculated based on the second order derivative of the error metric of ICP. *Prakhya* et al. [38] provide closed-form calculations for both point-to-point and point-to-plane metrics.
The general formula is:

$$cov(x) \approx \left(\frac{\partial^2 J}{\partial x^2}\right)^{-1} \cdot \left(\frac{\partial^2 J}{\partial z \partial x}\right) \cdot cov(z) \cdot \left(\frac{\partial^2 J}{\partial z \partial x}\right)^{T} \cdot \left(\frac{\partial^2 J}{\partial x^2}\right)^{-1} \tag{4.4}$$

where:

$$\begin{aligned} z &= \text{input measurement} \\ x &= \text{output vector} \\ cov(z) &= \text{covariance of sensor} \\ J &= \text{error function} \end{aligned}$$

Hence setting J with the formula of point-to-plane ICP (see chapter 1 equation 1.2) gives the required covariance. Nevertheless this covariance is relevant only for the inliers and only if ICP converges to a solution. Therefore this is not an appropriate tool for convergence test.

4.6.2 Covariance of Landmarks

SIFT features have been employed as landmark. Consequently the covariance of SIFT features is required for the landmark's measurements which can be estimated according to *Zeisl* et al. [55]. Initially the operating response function is defined as $D(x, c_i)$. This function detects features at the neighborhood of point x and at scale c_i. Then the covariance is estimated as the inverse Hessian matrix of D.

$$\Sigma = H^{-1} = \pm \begin{bmatrix} D_{xx}(p,\sigma) & D_{xy}(p,\sigma) \\ D_{xy}(p,\sigma) & D_{yy}(p,\sigma) \end{bmatrix}^{-1} \tag{4.5}$$

The above formula calculates the covariance at scale σ_i. Then this covariance has to be propagated at the initial image. Lets denote it as $\Sigma^{(0)}$. Then:

$$\Sigma^{(0)} = \Sigma \left(\frac{res(D(\bullet, \sigma_0))}{res(D(\bullet, \sigma_i)}\right)^2 \tag{4.6}$$

where:

$$res = \text{Rescaling in case of different resolution between scales.}$$

Subsequently, this covariance has to be propagated from the image plane to three dimensions:

Let:

$$\begin{aligned} x_k, y_k, z_k &= \text{Coordinates of point } k \text{ to 3 dimensions} \\ x_k^p, y_k^p &= \text{Coordinates of pixel } k \text{ on image plane} \\ Z_k &= \text{Depth of pixel } k \end{aligned}$$

The following formula computes the 3D coordinates from pixels and depth:

$$\begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix} = \begin{bmatrix} \frac{x_k^p - C_x}{f_x} \cdot Z_k \\ \frac{y_k^p - C_y}{f_y} \cdot Z_k \\ Z_k \end{bmatrix}$$

Thus:

$$\begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix} = \begin{bmatrix} \frac{Z_k}{f_x} & 0 & 0 \\ 0 & \frac{Z_k}{f_y} & 0 \\ 0 & 0 & Z_k \end{bmatrix} \cdot \begin{bmatrix} x_k^p \\ y_k^p \\ 1 \end{bmatrix} + \begin{bmatrix} -\frac{C_x}{f_x} \cdot Z_k \\ -\frac{C_y}{f_y} \cdot Z_k \\ 0 \end{bmatrix} \quad \Longrightarrow$$

$$\begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix} = \begin{bmatrix} \frac{Z_k}{f_x} & 0 \\ 0 & \frac{Z_k}{f_y} \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_k^p \\ y_k^p \end{bmatrix} + \begin{bmatrix} -\frac{C_x}{f_x} \cdot Z_k \\ -\frac{C_y}{f_y} \cdot Z_k \\ Z_k \end{bmatrix} \tag{4.7}$$

Nevertheless depth measurements are prone to noise:

$$\begin{aligned} \hat{Z}_k &= Z_k + \mathcal{N}(0, \sigma) \quad \Longrightarrow \\ Z_k &= \hat{Z}_k - w_z \end{aligned} \tag{4.8}$$

Then from equations (4.7) and (4.8):

$$\begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix} = \begin{bmatrix} -\frac{\hat{Z}_k - w_z}{f_x} & 0 \\ 0 & -\frac{\hat{Z}_k - w_z}{f_y} \\ 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_k^p \\ y_k^p \end{bmatrix} + \begin{bmatrix} -\frac{C_x \cdot \hat{Z}_k}{f_x} \\ -\frac{C_y \cdot \hat{Z}_k}{f_y} \\ \hat{Z}_k \end{bmatrix} + \begin{bmatrix} \frac{C_x}{f_x} \cdot w_z \\ \frac{C_y}{f_y} \cdot w_z \\ -w_z \end{bmatrix}$$

Assuming that:

$$\frac{w_z}{f_x} \ll 1, \quad \frac{w_z}{f_y} \ll 1 \quad \Longrightarrow$$

$$\begin{bmatrix} x_k \\ y_k \\ z_k \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{\hat{Z}_k}{f_x} & 0 \\ 0 & \frac{\hat{Z}_k}{f_y} \\ 0 & 0 \end{bmatrix}}_{A} \cdot \begin{bmatrix} x_k^p \\ y_k^p \end{bmatrix} + \begin{bmatrix} -\frac{C_x}{f_x} \cdot \hat{Z}_k \\ -\frac{C_y}{f_y} \cdot \hat{Z}_k \\ \hat{Z}_k \end{bmatrix} + \underbrace{\begin{bmatrix} \frac{C_x}{f_x} & 0 & 0 \\ 0 & \frac{C_y}{f_y} 0 & \\ 0 & 0 & -1 \end{bmatrix}}_{L} \cdot w_z$$

The previous model is linear, thus the uncertainty is propagated according too:

$$P_k = A_k \cdot Q_p \cdot A_k^T + L_k \cdot Q_z \cdot L_k^T$$

Where:

$$\begin{aligned} Q_p &= \text{Pixel uncertainty} \\ Q_z &= \text{Depth uncertainty (sensors depending)} \end{aligned}$$

The above formulas propagate the covariance of SIFT from image plane to 3D space. On the contrary, in the case of Pose Graph, uncertainties in feature correspondences are, by construction, not supported.

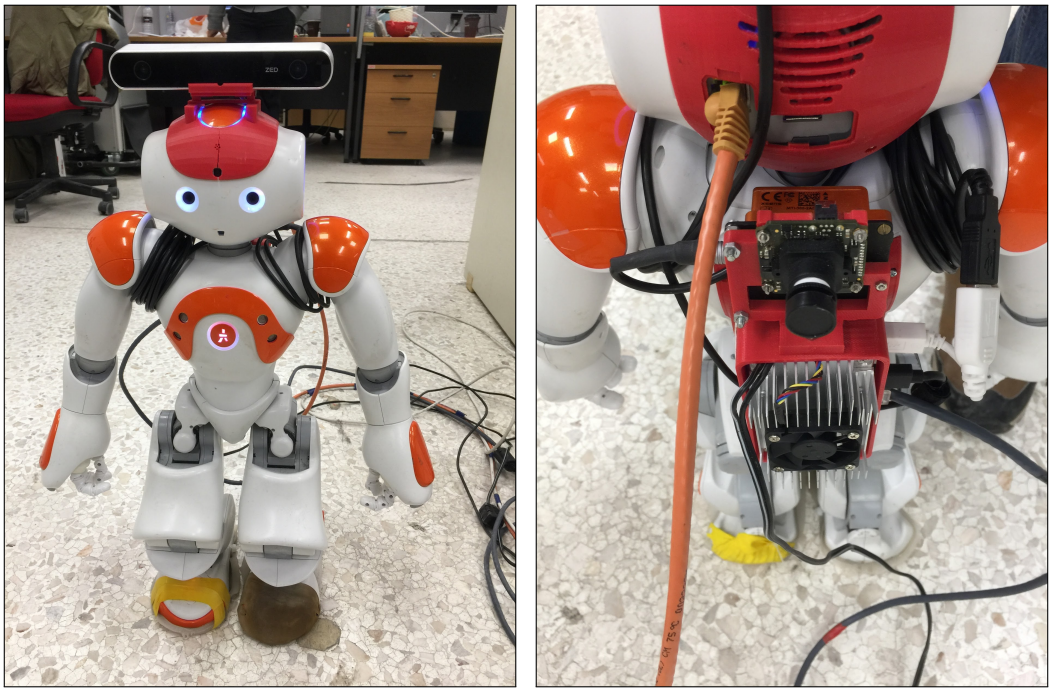
4.6.3 Map Fixes

The Pose Graph provides new corrected estimations of camera poses. These corrections needs to be propagated in the map. This is accomplished by removing data (*de*-integrate them) and integrating them again with the correct pose. We remove the data similarly with [9] by the following formula:

$$TSDF_i(x) = \frac{W_{i-1}(x) \cdot TSDF_{i-1}(x) - w_i(x) \cdot tsdf_i(x)}{W_{i-1}(x) - 1}$$

4.7 Robotic Platform

Both systems (Kinect Fusion with SEROW and Kinect Fusion with Pose Graph) were evaluated in real life experiments. In particular, we conducted experiments with a Nao Robot enhanced with an *Asus Xtion* camera (Figure 4.7a). In addition, Kinect Fusion and SEROW run on an *Nvidia Jetson tx2*, placed on the back of the Nao robot (Figure 4.7b). This system was independent of external resources allowing Nao to move complete autonomous. On the contrary, Kinect Fusion with Pose Graph was evaluated offline on a desktop computer.



(a) Nao with xtrion camere (b) Nao with Nvidia Jetson tx2

Figure 4.7: Nao Robot Configuration.

Chapter 5

Experimental Results

In the current Chapter the results of the two proposed modification of Kinect Fusion are presented. We conducted separate experiments for Kinect Fusion with SEROW and Kinect Fusion with Pose Graph. Kinect Fusion with SEROW was tested in real world experiments in a lab environment while Kinect Fusion with Pose Graph was tested in Gazebo simulator and in the Nao robot.

5.1 Results from Kinect Fusion and SEROW

The lab environment shown in Figure 5.1 was used in our evaluation experiment. In the latter, we recorded and compared the output of three different sources, Kinect Fusion, kinematics and Kinect Fusion with SEROW. Figures 5.2 and 5.3 depict the estimated robot position and orientation for each output. Moreover the final position of the robot was measured with conventional means (metre and protractor). The actual drift of every output from this position is included in Table 5.1.

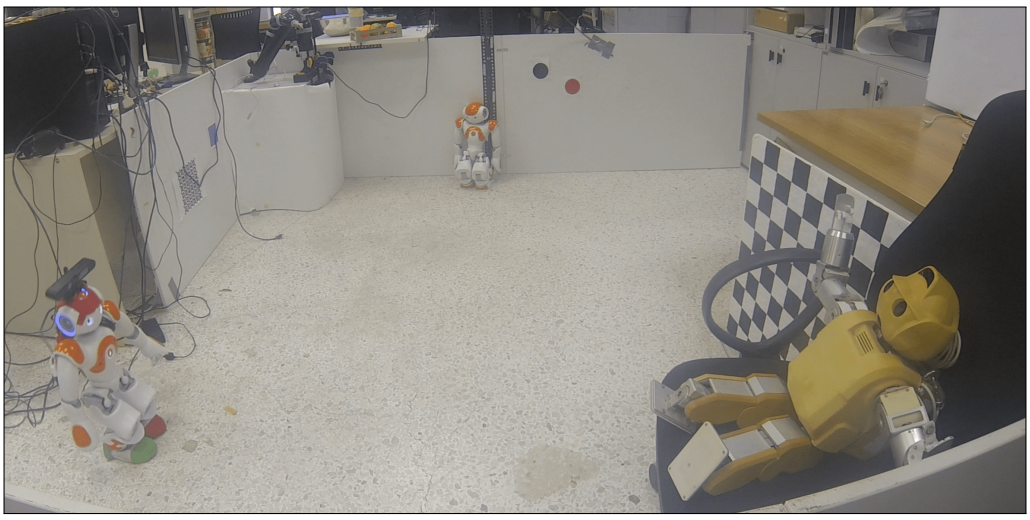


Figure 5.1: Navigation environment.

We observe that Kinect Fusion differs from kinematics, especially in the vertical axis. In addition, Table 5.1 indicates that Kinect Fusion and Kinematics are prone to drift. Due to robot's bipedal motion, the camera experiences sudden acceleration on the vertical

axis, which degrades the performance of Kinect Fusion in this axis. On the contrary, feet slippage, deteriorates the estimation of kinematics on the ground plane (x and y axes). Evidently, the combination of Kinect Fusion with SEROW produces more accurate estimates for all involved quantities. It achieves to correct the divergence of Kinect Fusion in the vertical axis and also reduces the final pose drift of Kinematics. Correction on pose estimation also results to a better map representation. Figure 5.4 draws the final world map, as it has been produced by Kinect Fusion with SEROW. The resulting representation indicates an accurate map of the environment. Finally Kinect Fusion with SEROW achieved frame rate of **29 fps** running on jetson tx2 with all the visualizations disabled.

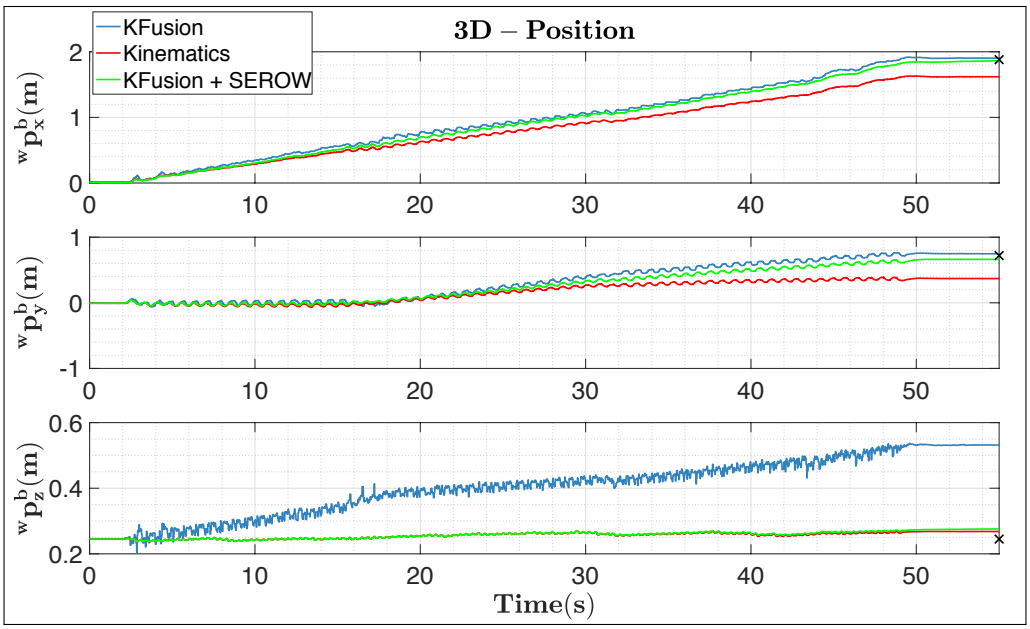


Figure 5.2: Position error.

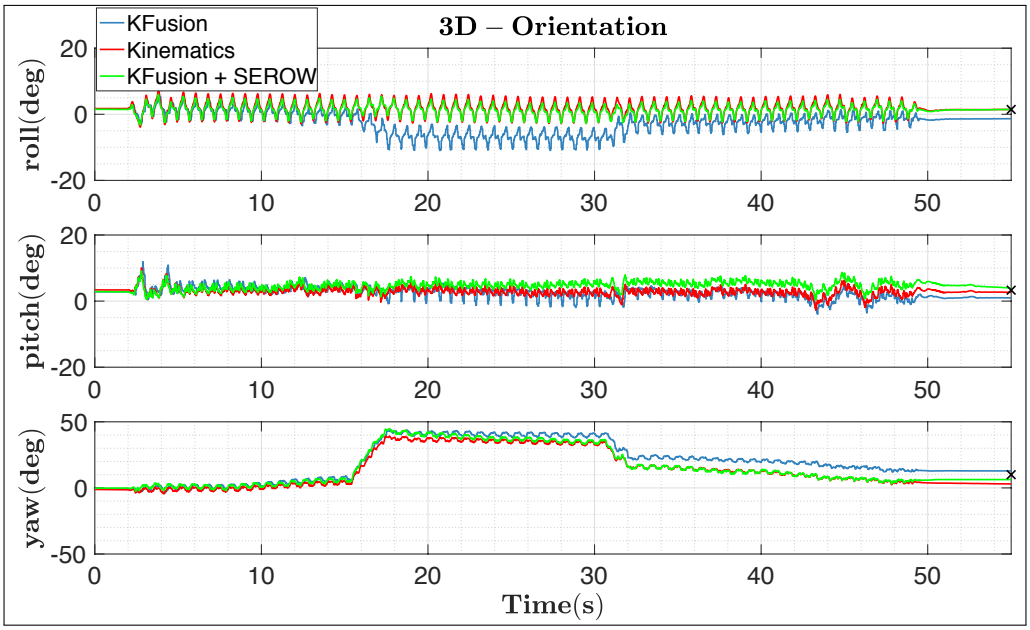


Figure 5.3: Orientation error.

	${}^{w}p_x(m)$	${}^{w}p_y$	${}^{w}p_z$	roll (deg)	pitch (deg)	yaw (deg)
Kinect Fusion	0.025	0.047	0.287	3	2	2
Kinematics	0.261	0.331	0.023	< 0.5	1	7
KF+SEROW	0.011	0.038	0.031	< 0.5	1	3

Table 5.1: Pose Drift: Precision 1 mm, 1 deg.

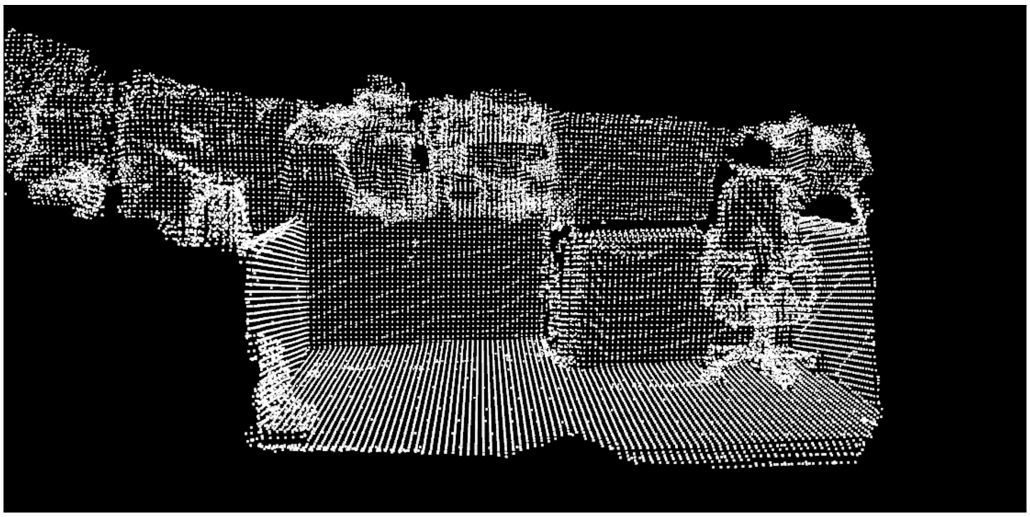


Figure 5.4: Resulting map.

5.2 Results from Kinect Fusion with Pose Graph

To assess the performance of Kinect Fusion with Pose Graph, we conducted two types of experiments: a) simulated experiments in Gazebo simulator, b) real world experiments with the Nao robot. We measured the error per frame before and after the optimization as well as the average error for every trajectory. In case of the Nao robot we did not have the actual ground truth trajectory and we compared the results with the odometry measurements.

5.2.1 Simulated Experiments

The experiments in this section were conducted in the Gazebo simulator. The simulated robotic platform was a *TurtleBot2* with asus xtion RGB-D camera, while the virtual navigated environment was an office room (Figure 5.5).
In this environment, the robot followed two different trajectories as shown in Figures 5.6 and 5.7. The yellow lines are the ground truth trajectories, the green lines are the trajectories before the optimization which are equivalent to Kinect Fusion while the red lines are the optimized trajectories. We can observe that the optimized trajectories are closer to ground truth than Kinect Fusion. The actual improvements can be observed in Figures 5.8 and 5.9 which indicate the translation error. The blue lines represent the error of Kinect Fusion while the green lines the error of the optimized trajectories. Generally there is an significant drop of the error, however in the beginning of each trajectory, Kinect Fusion has not yet diverged. In these frames, the error of the optimized trajectory is greater than Kinect Fusion. This is due to the fact that Pose Graph tends to spread the error across the frames. Nevertheless Table 5.2 shows that the average error is greatly reduced in the optimized trajectories. Complimentary to translation error, we measured the rotation error which was less than 0.2 degrees on each trajectory.
Finally Figure 5.10 shows the map projection to the camera frame for the two trajectories, before and after the optimization. The visual results look better in surfaces such as the painting in the wall (trajectory 1) or the two chairs (trajectory 2). On the contrary they are worse in small surfaces such as the coffee table (trajectory 1) or the library (trajectory 2). As noted above, this is due to the fact that Pose Graph spreads the error across the frames reducing the number of details in the map.

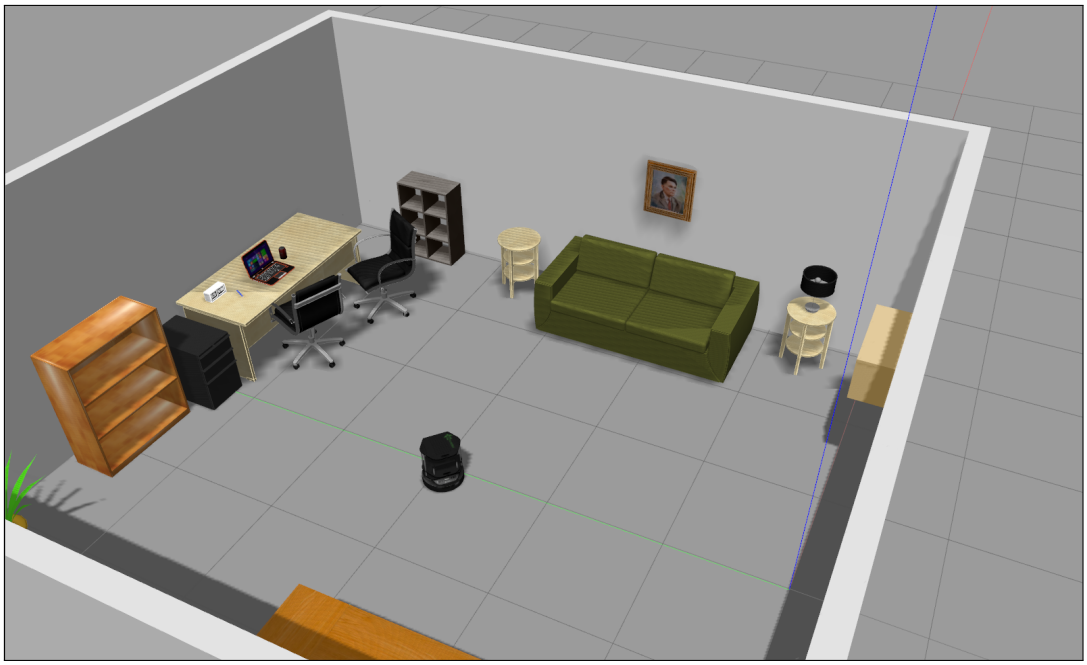


Figure 5.5: Office environment in Gazebo.

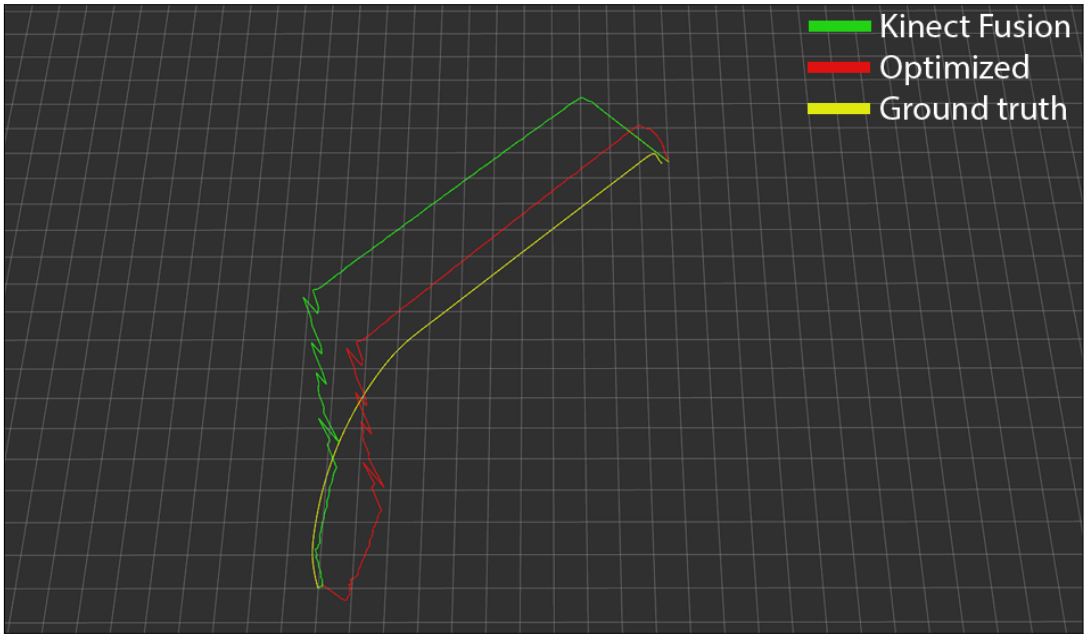


Figure 5.6: Trajectory 1.

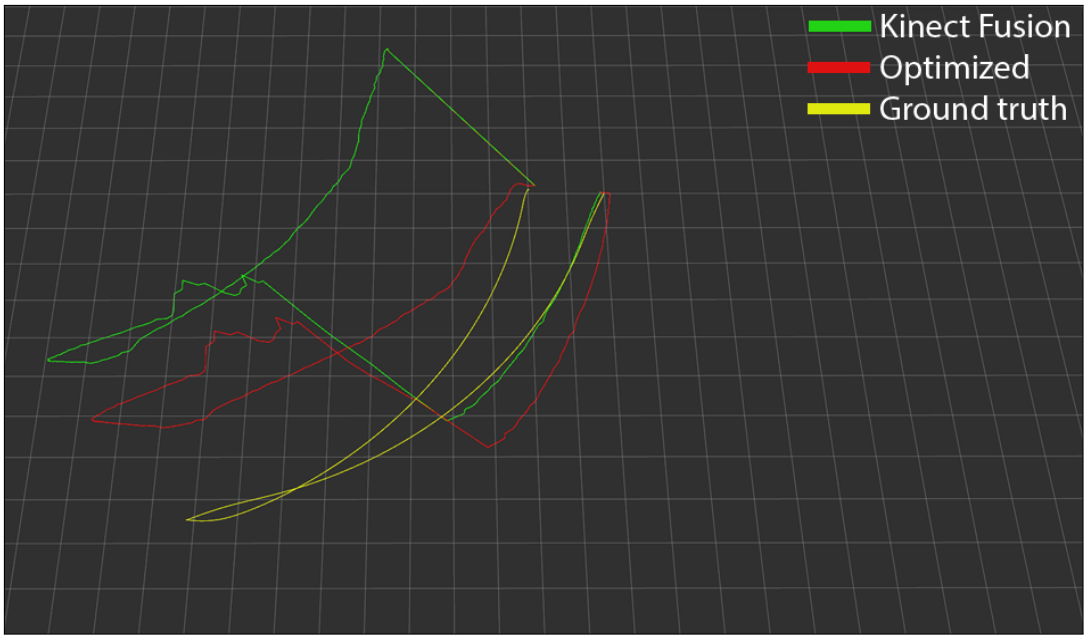


Figure 5.7: Trajectory 2.

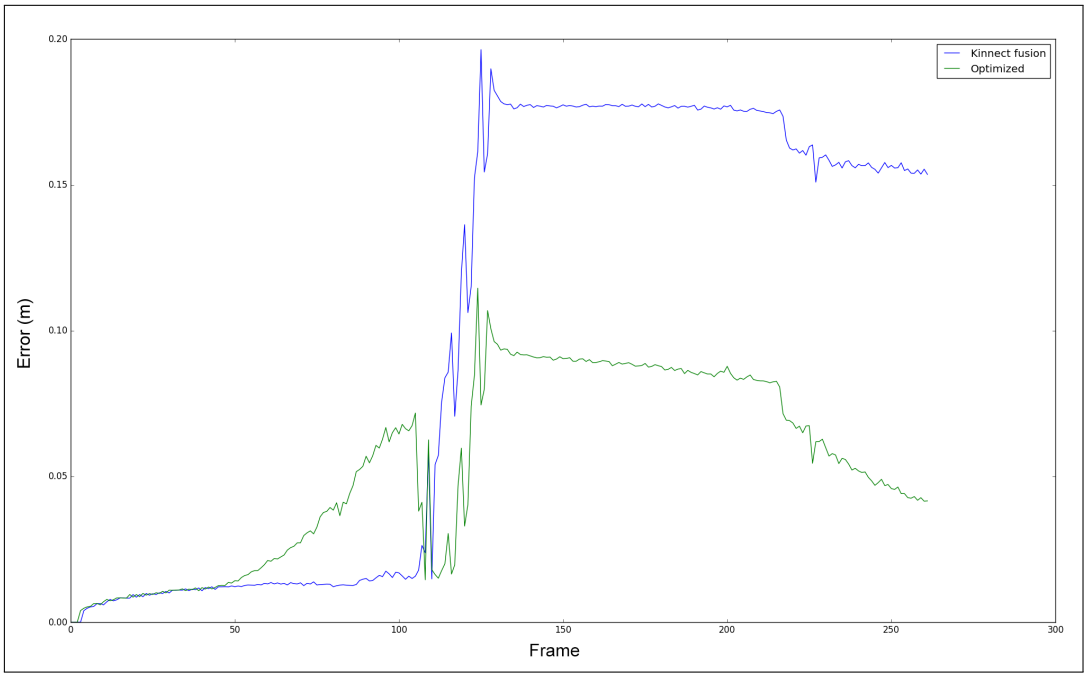


Figure 5.8: Translation error in Trajectory 1 (m).

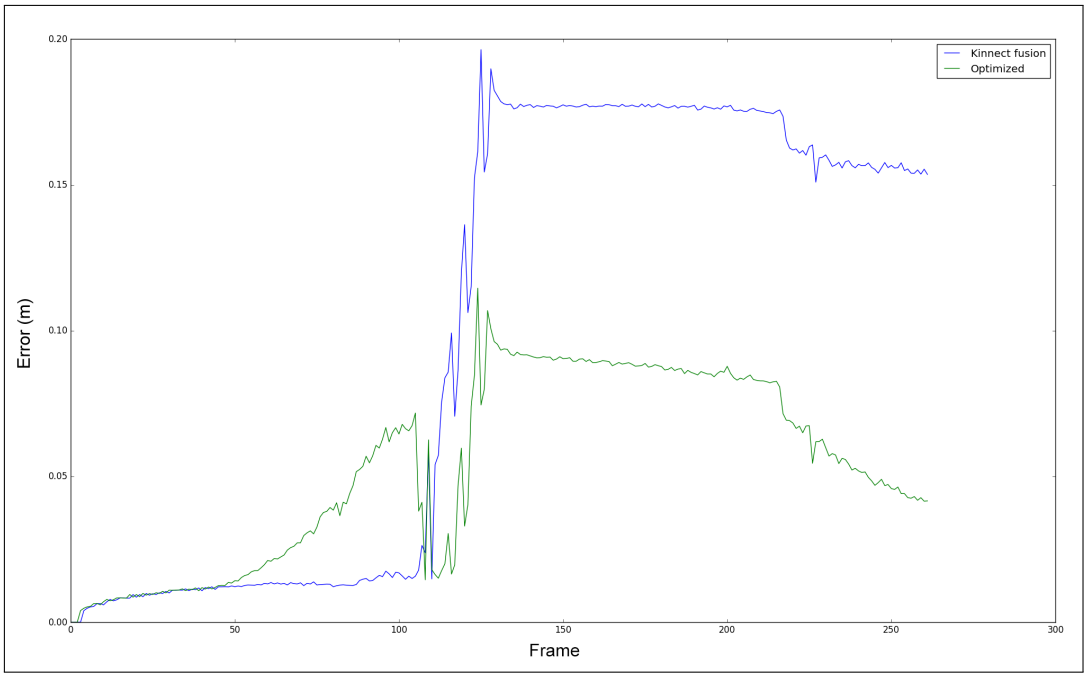
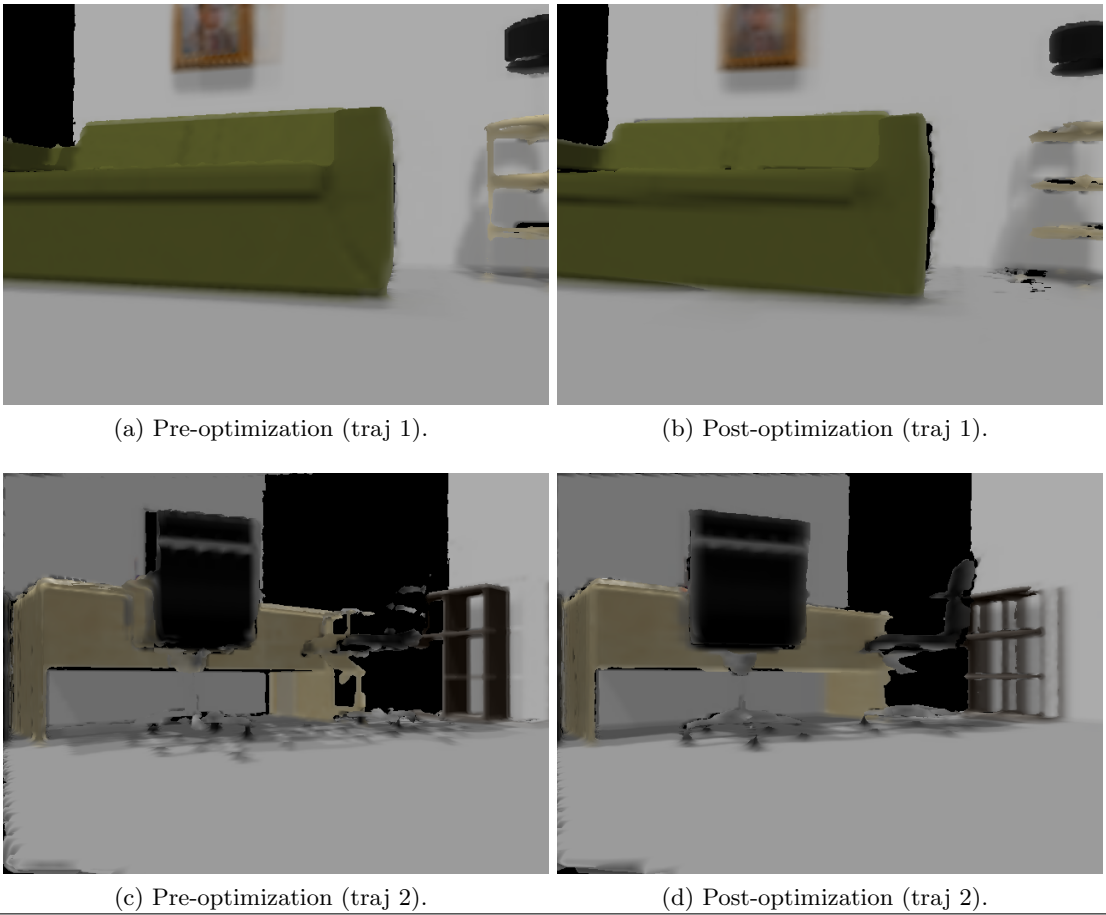


Figure 5.9: Translation error in Trajectory 2 (m).



(a) Pre-optimization (traj 1).

(b) Post-optimization (traj 1).

(c) Pre-optimization (traj 2).

(d) Post-optimization (traj 2).

Figure 5.10: Map projection before and after optimization.

	Trajectory 1	Trajectory 2
Kinect fusion	0.099	0.183
optimized	0.053	0.087

Table 5.2: Average translation error (m).

5.2.2 Experiments with the Nao Robot

In addition to simulated experiments, we also conducted experiments in the real world with the Nao Robot and an Asus Xtion camera as described in Section 4.7. In these experiments we did not have the actual ground truth trajectory of the camera, therefore we used the odometry of the Nao robot as baseline.
Figure 5.11 shows such a trajectory of the Nao Robot. The yellow line in this graph represents the odometry measurement. The green line represents the estimated trajectory of Kinect Fusion while the red line the optimized trajectory. Images in Figure 5.12 display the projection of the map to camera frame before and after the optimization. As can be observed, there is a significant visual improvement of the image. Moreover Figure 5.13 depicts the deviation in meters from odometry. The average deviation of Kinect fusion was **0.078** while the deviation of the optimized trajectory was **0.060**. We can observe a drop of the error, however it is not as high as in the simulated experiments. Nevertheless, it is noted that the comparison with Nao's odometry is not an accurate metric.

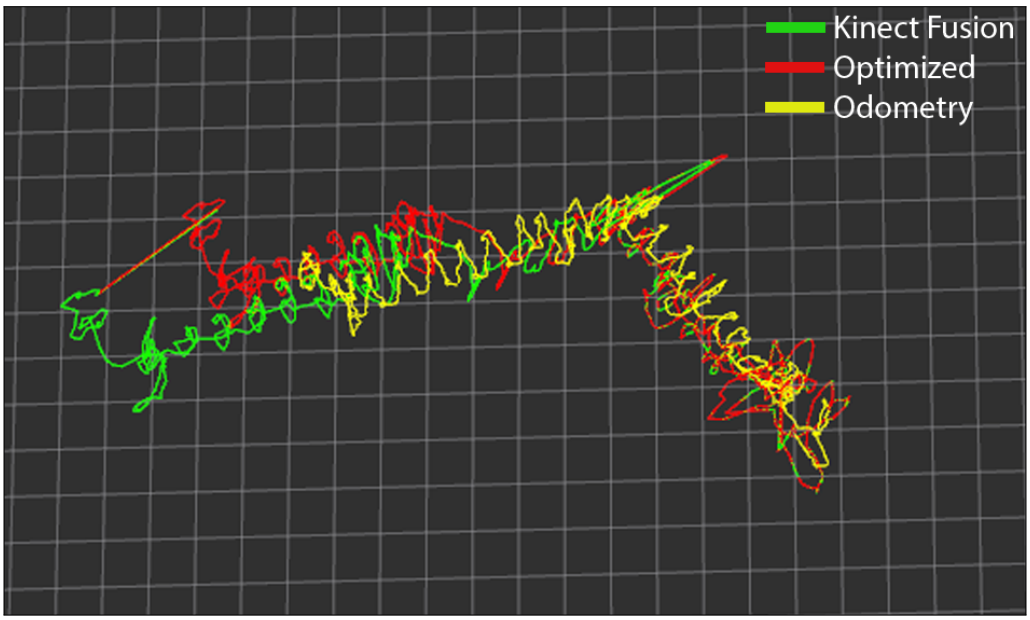
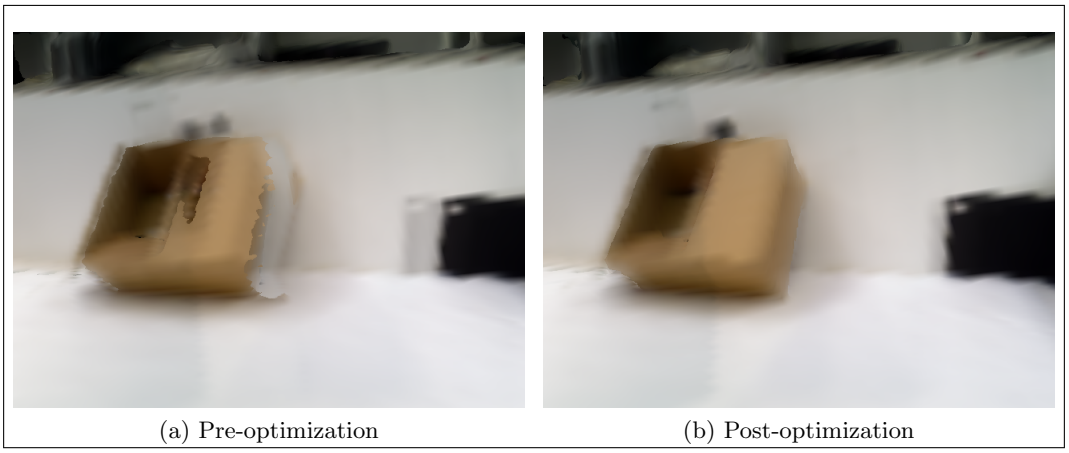


Figure 5.11: Trajectory of Nao (cell size: $25cm^2$).



(a) Pre-optimization (b) Post-optimization

Figure 5.12: Map projection before and after optimization.

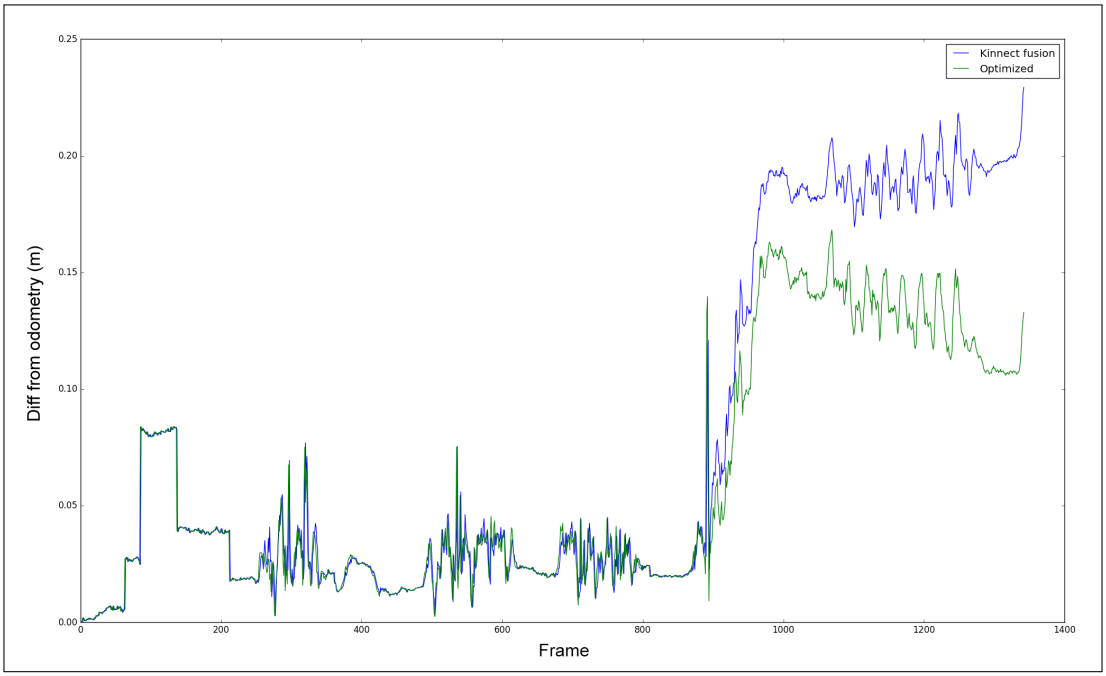


Figure 5.13: Trajectory 1.

Chapter 6

Conclusions

6.1 Summary

This thesis addresses the SLAM problem in the case of humanoid robots. It begins with an evaluation on SLAM systems and their properties. This evaluation shows that *Kinect Fusion* [13] is well suited for on-board execution due to its high frame rate and low memory usage. In addition, the extra photometric constraints that *Elastic Fusion* [50] introduces do not negatively affect the operation time of VO. Moreover, loop closure of *ORB-SLAM2* [31] can bound the accumulated error. Nevertheless the loop detection problem is more computationally heavy than the actual loop closure and Pose Graph optimization.
There follows a comparative study of *3DMatch* [56] and *3DSmoothNet* [10] geometric descriptors. Both descriptors are extracted from random key-points using convolutional neural networks and they are compared to more conventional methods such as SIFT [25]. The results show that 3DMatch has very few principal components (only 4 out ot 512). 3DSmoothnet performs better with 18 principal components out of 32 and finally SIFT ends up with 32 out of 128. Generally, both methods are slower than SIFT and need an excessive amount of random key-points in order to find an accurate alignment between two point clouds. Finally, a combination of Harris Corners from RGB with 3DSmoothnet descriptors from depth can greatly reduce the necessary key-points and consequently reduce operation time.
The evaluation phase indicated that Kinect Fusion is suitable for embedded devices. Hence, this work continued with two proposed modification of Kinect Fusion. The first proposed modification takes advantage of the SEROW state estimation system [35]. SEROW fuses multiple sources including VO, kinematics and odometry. The introduced modification operates by propagating VO to SEROW for better pose estimation. Then this refined pose is used for new data integration into the map. This approach achieves better pose estimation and map representation. However it does not include any loop closure mechanism and hence it is prone to the accumulated error. This method was tested in real world experiments with the Nao robot and accomplished real time execution on a Jetson tx2.
The second modification establishes a loop closure mechanism by combining Kinect Fusion and SIFT features into a common Pose Graph. Poses from Kinect Fusion are incorporated into the Pose Graph together with SIFT features as landmarks. We evaluated this algorithm with simulated and real word experiments. The experiments showed reduction of error per frame in every case. This algorithm however runs offline in a desktop computer.

6.2 Future Work

Both algorithms presented here were based on Kinect Fusion. Kinect Fusion has limited map size as it pre-allocates the memory. This work did not deal with this problem. Nevertheless, voxel hashing methods similar to one applied here [32] can address this issue. In addition Kinect Fusion with SEROW do not contain any loop closure mechanism or re-localization technique.
Regarding Kinect Fusion with Pose Graph, at the moment this method does not contain a key-frame extraction mechanism. Such mechanism can be developed with a bag of words on input images. Furthermore, other feature extraction methods can be tested together with the Pose Graph. 3DSmoothNet with Harris corners is a promising alternative, however the covariance of these features has to be calculated. Besides that, Pose Graph runs offline and it is not suitable for real time execution. Finally Pose Graph can be combined with kinematics and odometry. Both measurements can be added as pose nodes in the graph, an issue that requires further research.

Bibliography

[1] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. SURF: Speeded Up Robust Features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[2] P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. In *in IEEE Transactions on Pattern Analysis and Machine Intelligence*, volume 14 no. 2, pages 239–256, 1992.

[3] Bruno Bodin, Harry Wagstaff, Sajad Saeedi, Luigi Nardi, Emanuele Vespa, John H Mayer, Andy Nisbet, Mikel Luján, Steve Furber, Andrew J Davison, Paul H.J. Kelly, and Michael O'Boyle. SLAMBench2: Multi-Objective Head-to-Head Benchmarking for Visual SLAM. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, May 2018.

[4] M. Bujanca, P. Gafton, S. Saeedi, A. Nisbet, B. Bodin, M. F. P. O'Boyle, A. J. Davison, P. H. J. Kelly, G. Riley, B. Lennox, M. Luján, and S. Furber. SLAMBench 3.0: Systematic Automated Reproducible Evaluation of SLAM Systems for Robot Vision Challenges and Scene Understanding. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6351–6358, 2019.

[5] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal" Fua. BRIEF: Binary Robust Independent Elementary Features. In *Computer Vision – ECCV 2010*, pages 778–792, 2010.

[6] Brian Curless and Marc Levoy. A Volumetric Method for Building Complex Models from Range Images. In *SIGGRAPH*, pages 303–312, 1996.

[7] Angela Dai, Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Christian Theobalt. BundleFusion: real-time globally consistent 3D reconstruction using on-the-fly surface re-integration. *ACM Transactions on Graphics*, 36:1, 07 2017.

[8] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. MonoSLAM: Real-Time Single Camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):1052–1067, 2007.

[9] N. Fioraio, J. Taylor, A. Fitzgibbon, L. Di Stefano, and S. Izadi. Large-scale and drift-free surface reconstruction using online subvolume registration. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4475–4483, 2015.

[10] Z. Gojcic, C. Zhou, J. D. Wegner, and A. Wieser. The Perfect Match: 3D Point Cloud Matching With Smoothed Densities. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5540–5549, 2019.

[11] A. Handa, T. Whelan, J. McDonald, and A. J. Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1524–1531, May 2014.

[12] E. Hourdakis and M. Lourakis. Countering Drift in Visual Odometry for Planetary Rovers by Registering Boulders in Ground and Orbital Images. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'15)*, pages 111–116, October 2015.

[13] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera. In *UIST '11 Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM, October 2011.

[14] Jianbo Shi and Tomasi. Good features to track. In *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, 1994.

[15] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Trans. on Robotics (TRO)*, 24(6):1365–1378, December 2008.

[16] S. Kagami, K. Nishiwaki, J. J. Kuffner, K. Okada, M. Inaba, and H. Inoue. Vision-based 2.5D terrain modeling for humanoid locomotion. In *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, volume 2, pages 2141–2146 vol.2, 2003.

[17] Olaf Kähler, Victor Adrian Prisacariu, and David W. Murray. Real-Time Large-Scale Dense 3D Reconstruction with Loop Closure. In *ECCV 2016*, pages 500–516, 2016.

[18] G. Klein and D. Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 225–234, 2007.

[19] Viktor Kovács and Gábor Tevesz. Corner Detection and Classification of Simple Objects in Low-Depth Resolution Range Images. 57:9–17, 2013.

[20] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. G2o: A general framework for graph optimization. pages 3607 – 3613, 06 2011.

[21] G. Lentaris, I. Stamoulias, D. Soudris, and M. Lourakis. HW/SW Co-design and FPGA Acceleration of Visual Odometry Algorithms for Rover Navigation on Mars. In *IEEE Trans. on Circuits and Systems for Video Technology 26(8)*, pages 1563–1577, 2016.

[22] M. Lourakis. An efficient solution to absolute orientation. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 3816–3819, 2016.

[23] M. Lourakis and G. Terzakis. Efficient Absolute Orientation Revisited. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5813–5818, 2018.

[24] Kok-Lim Low. Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration. Technical report, 2004.

[25] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[26] D. Maier, C. Lutz, and M. Bennewitz. Integrated perception, mapping, and footstep planning for humanoid navigation among 3D obstacles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2658–2664, 2013.

[27] Lourakis M.I.A. and Argyros A.A. SBA: A Software Package For Generic Sparse Bundle Adjustment. In *ACM Trans. on Mathematical Software (TOMS)*, pages 36(1):2:1–30, March 2009.

[28] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. In *in Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 257–263, 2003.

[29] Krystian Mikolajczyk, Cordelia Schmid, and Andrew Zisserman. Human Detection Based on a Probabilistic Assembly of Robust Part Detectors. In *Computer Vision - ECCV 2004*, pages 69–82, 2004.

[30] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[31] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[32] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3D Reconstruction at Scale using Voxel Hashing. *ACM Transactions on Graphics (TOG)*, 2013.

[33] Giuseppe Oriolo, Antonio Paolillo, Lorenzo Rosa, and Marilena Vendittelli. Vision-based Odometric Localization for humanoids using a kinematic EKF. *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*, pages 153–158, 2012.

[34] Paschalis Panteleris and Antonis A. Argyros. Vision-Based SLAM and Moving Objects Tracking for the Perceptual Support of a Smart Walker Platform. In *Computer Vision - ECCV 2014 Workshops*, pages 407–423, 2015.

[35] S. Piperakis, M. Koskinopoulou, and P. Trahanias. Nonlinear State Estimation for Humanoid Robot Walking. *IEEE Robotics and Automation Letters*, 3(4):3347–3354, Oct 2018.

[36] Stylianos Piperakis, Nikolaos Tavoularis, Emmanouil Hourdakis, Dimitrios Kanoulas, and Panos Trahanias. Humanoid Robot Dense RGB-D SLAM for Embedded Devices. In *IEEE International Conference on Robotics and Automation (ICRA): 3rd Full-Day Workshop "Towards Real-World Deployment of Legged Robots"*, 04 2019.

[37] Taihú Pire, Thomas Fischer, Gastón Castro, Pablo De Cristóforis, Javier Civera, and Julio Berlles. S-PTAM: Stereo Parallel Tracking and Mapping. *Robotics and Autonomous Systems*, 93:27–42, April 2017.

[38] S. M. Prakhya, L. Bingbing, Y. Rui, and W. Lin. A closed-form estimate of 3D ICP covariance. In *2015 14th IAPR International Conference on Machine Vision Applications (MVA)*, pages 526–529, 2015.

[39] Tong Qin, Peiliang Li, and Shaojie Shen. VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[40] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *Proc. 9th European Conference on Computer Vision (ECCV'06)*, 2006.

[41] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: an efficient alternative to SIFT or SURF. pages 2564–2571, 11 2011.

[42] R. Scona, S. Nobili, Y. R. Petillot, and M. Fallon. Direct visual SLAM fusing proprioception for a humanoid robot. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1419–1426, 2017.

[43] Xuesong Shi, Dongjiang Li, Pengpeng Zhao, Qinbin Tian, Yuxin Tian, Qiwei Long, Chunhao Zhu, Jingwei Song, Fei Qiao, Le Song, Yangquan Guo, Zhigang Wang, Yimin Zhang, Baoxing Qin, Wei Yang, Fangshi Wang, Rosa H. M. Chan, and Qi She. Are We Ready for Service Robots? The OpenLORIS-Scene Datasets for Lifelong SLAM. *arXiv preprint arXiv:1911.05603*, 2019.

[44] Olivier Stasse, François Saïdi, Kazuhito Yokoi, Bjorn Verrelst, Bram Vanderborght, Andrew Davison, Nicolas Mansard, and Claudia Esteves. Integrating Walking and Vision to Increase Humanoid Autonomy. *I. J. Humanoid Robotics*, 5:287–310, 06 2008.

[45] 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.

[46] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580, 2012.

[47] Nikolas Tavoularis. Kfusion ros. `https://github.com/tavu/kfusion_ros`.

[48] Diana Werner, Ayoub Al-Hamadi, and Philipp Werner. Truncated Signed Distance Function: Experiments on Voxel Size. volume 8815, pages 357–364, 10 2014.

[49] Thomas Whelan, Michael Kaess, Hordur Johannsson, Maurice Fallon, John J. Leonard, and John McDonald. Real-time large scale dense RGB-D SLAM with volumetric fusion. *The International Journal of Robotics Research*, 34:598–626, April 2015.

[50] Thomas Whelan, Renato F Salas-Moreno, Ben Glocker, Andrew J Davison, and Stefan Leutenegger. ElasticFusion: Real-time dense SLAM and light source estimation. *The International Journal of Robotics Research*, 35(14):1697–1716, 2016.

[51] Emilie Wirbel, B. Steux, Silvère Bonnabel, and Arnaud de La Fortelle. Humanoid robot navigation: From a visual SLAM to a visual compass. In *2013 10th IEEE*

International Conference on Networking, Sensing and Control, ICNSC 2013, pages 678–683, 04 2013.

[52] Yan Ke and R. Sukthankar. PCA-SIFT: a more distinctive representation for local image descriptors. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 2, pages II–II, 2004.

[53] Heng Yang, Jingnan Shi, and Luca Carlone. TEASER: Fast and Certifiable Point Cloud Registration. *ArXiv*, abs/2001.07715, 2020.

[54] Christopher Zach. Robust Bundle Adjustment Revisited. In *Computer Vision – ECCV 2014*, pages 772–787, 2014.

[55] Bernhard Zeisl, Pierre Georgel, Florian Schweiger, Eckehard Steinbach, Nassir Navab, and GER Munich. Estimation of Location Uncertainty for Scale Invariant Feature Points. *Proceedings of the British machine vision conference*, 01 2009.

[56] A. Zeng, S. Song, M. Nießner, M. Fisher, J. Xiao, and T. Funkhouser. 3DMatch: Learning Local Geometric Descriptors from RGB-D Reconstructions. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 199–208, 2017.

[57] T. Zhang, E. Uchiyama, and Y. Nakamura. Dense RGB-D SLAM for Humanoid Robots in the Dynamic Humans Environment. In *2018 IEEE-RAS 18th International Conference on Humanoid Robots (Humanoids)*, pages 270–276, 2018.