博士論文 (要約)

Dense 3D SLAM Using Multi-Resolution Volumetric Mapping and Real-Time Agile Tracking
(多解像度立体地図生成と実時間敏捷追跡による稠密3D SLAM)

カチリ　ユセフ

論文の内容の要旨
Abstract

Title of dissertation :

Dense 3D SLAM Using Multi-Resolution Volumetric Mapping and Real-Time Agile Tracking (多解像度立体地図生成と実時間敏捷追跡による稠密 3D SLAM)

Name of the Author :

KTIRI Youssef (カチリ　ユセフ)

Simultaneous Localization and Mapping lies at the heart of fully autonomous mobile robotic systems. Last years have seen a prominent number of contributions to the field backed by the recent advent of cheap yet reliable commodity sensors. Successful SLAM systems lay a solid ground for achieving challenging tasks such as disaster area exploration where network latencies, non availability of previously acquired maps of any sort, severe viewing conditions ask for robots to show high degrees of autonomy and less reliability on a remote human agent. Key aspects of such SLAM systems include high speed robust tracking, online volumetric map construction, dynamic obstacles handling and reliability in challenging environments which can exhibit geometrical or photometric features scarceness. If the 2D SLAM problem has widely been tackled during the past decade, 3D SLAM and its increased load of information brings in additional challenges where memory consumption can quickly grow out of the system boundaries and straightforward tracking methods fail to keep up with real-time needs. In the present work, we derive a solution to the full 3D SLAM problem which complies with mobile robotic systems and their tight requirements. First, we proposed a map representation with associated stepping and traversal iterators. The map bases on a limited depth octree data structure which allocates all necessary memory beforehand to avoid online data allocation latencies and guarantee memory contiguity. Memory is managed internally and allows concurrent reading and modification on multi-core hardware. Our map representation allows us to derive fast insertion, freeing, raycasting and neighbor search algorithms. The enhanced speed we obtain is crucial to be able to build

highly detailed maps online and in real-time. The memory compression is also such that large workspaces and maps can be handled. The map is essentially multiscale. The multiscale property is used by all algorithms for speed-ups but also as different points have different noise amplitude, mapping proceeds by inserting each point at the correct scale hence avoiding corruptions of more precise voxels with less precise data. Then, we proposed a real-time agile tracker which builds on the association of a direct optimization based dense photometric tracker and a model based geometric tracker. The geometric tracker builds on our map iterators to extract at high speed the exact nearest neighbor in a 3D neighborhood around candidate points and run subsequent ICP optimization. This tracker shows large basin of attraction to the minimum cost solution with a marked convexity and hence converges in few iterations only. The geometrical tracker can recover from relatively large six dimensional sensor displacements and return results at high speed. These two conditions guarantee convergence under fast and dexterous sensor motions. The photometric tracker complements the geometric tracker's behavior and adds more stability and robustness against environments with poor geometric features. The photometric tracker show tighter basin of attraction but, with good initialization from the geometrical tracker counterpart, can yield subpixel motion estimates in few iterations only. A sensor model associates each point at the input stage with a proper variance derived from a normal distribution approximation. The point noisiness is taken into account during the tracking stage to yield more noise resilient estimates. Our front-end methods are used in association with a back-end routine with runs loop closure detection and optimization and hence allows to scale up the system for large environments. Finally, all system components are blended in an architecture which solves the full 3D SLAM problem at high speed. The architecture blends tracking, sensing, map insertions, map freeing, drawing, loop detection and optimization in a concurrent way and such that, at each moment, distinct threads request different computational resources on the CPU or the GPU side. The architecture as it has been designed allows solving the full 3D SLAM problem and rendering highly detailed 3D maps in real-time without enforcing any high-end computing power re-

quirements. Experimental results show how our framework provides with a fast and reliable solution to the 3D SLAM problem and can be used as a backbone for mobile robots operating under the most challenging conditions.

Acknowledgments

I would like to thank all the people who supported me and have gotten me here. First of all, I would like to thank my professors Okada sensei and Inaba sensei, whose guidance have brought this work to accomplishment. I also would like to thank the Rotary Foundation, the Rotary clubs from Bretagne and Ueno, the Shundou Foundation, the Sojitz Foundation for their support and the great moments I spent within them. Finally, my eternal gratitude for mom, dad, my family, friends and loved ones whose love and support made my life so wonderful.

目 次

第1章

Introduction

Reconstruction of 3D models of small or large workspaces from static or mobile sensory devices has been an active area of research for the past decades. Such process lies at the intersection of multiple fields such as computer graphics, robotics or computer vision. Applications can range from 3D medical reconstruction of a body part anatomy from scans, augmented reality for gaming industry, scanning of damaged buildings and infrastructures like bridges or power plants, automatic mesh acquisition of a human body or an object, building indoor or outdoors maps for autonomous robots navigation and so on.

In either static or dynamic settings, accurate reconstruction needs special handling since measurements acquired from single or multiple sensory sources are inevitably entailed with noise due to factors ranging from structural misalignment, too high or too low temperatures, electro-magnetic interference, environment reflectance, direct sunlight exposure, motion blur, saturation, scaling errors, quantization and so on. For such a fundamental reason, sensory measurements are crossed with multiple observation from the same or multiple sources then integrated into one statistically consistent estimate. The process of combining these multiple measurements into a more accurate estimate is called sensor data fusion and is central to all processes which rely on sensory data to infer knowledge about the world. A well understood example in the literature is Inertial Measurements Units (IMUs) which today are ubiquitous in mobile devices technology. These sensors are made of two principal components. The first one is an accelerometer (usually perpendicularly aligned 3-axis accelerometers) which measures acceleration. The second component is a gyroscope which measures the angular speed around the gyroscope axis. These two sensors provide with redundant measurements. By sensing the direction of the gravity vector on a static setting the accelerometers can infer the roll and pitch of a the sensor while the gyroscope can recover similar information by integrating the angular speed. However, each sensor is entailed which a characteristic source of noise and hence used alone will provide a poor estimate of the anglar position. Accelerometers provide slower response and can get affected by the noise coming from acceleration due to the body motion which adds to the gravity value. Gyroscopes through the integration process involved in computing angles are prone to drift but provide a faster response. They are also not prone to errors due to motion acceleration. Based on such observations it is clear how it is particularly appealing to fuse the information provided by both sensors into one consistent estimate. This is done by statistical tools like Kalman filters[1][2][3][4] or less computationally expensive complementary filters [5]. Dynamic configurations introduce a fundamentally more complex problem to solve since sensor positions at various timestamps also needs to be inferred. This can be considered as a chicken and egg problem since accurate map estimates need precise motion increment values while accurate sensor localization requires accurate maps to be match against. The specific problem of recovering models and camera positions from sequences of camera images is known as structure from motion within the computer vision community. In robotics the more general process of recovering the position of a mobile agents using its intrinsic sensor along with a map of its surrounding environment is denoted Simultaneous Localization and Mapping (SLAM) and will be the focus of the present work. It is considered as the fundamental key to enable full or partial navigation in previously unknown environments. Robotic mobile agents are seldom guaranteed the availability of maps to use for navigation purposes. Robots need to build such map estimate from scratch as they navigate through obstacles and in an incremental manner. This is particularly the case when exploring mines, disaster areas or any other scenarios where no map is available. SLAM takes on its full sense in indoors, densely populated outdoor scenarios like forests or dense urban areas where direct satellite line of sight is not guaranteed or the signal cannot be distinctly received and hence during which no global localization tool like GPS can be based on. In such scenarios, a robot needs to solely rely on its intrinsic sensory information in order to extract the necessary knowledge about how to navigate, its current position, where and how to move to target points in space.

SLAM has been an active research area and subject to a great load of research in literature. It has also been subject to radical evolutions throughout the past decades. In the recent years, it has particularly been boosted by the advance in parallel computing solutions such as GPGPUs, which makes it possible to deal with increasingly dense and large chunks of data in real-time, and also by the advance in laser and imagery sensors technology. Among the most important breakthroughs is the advent of cheap and high quality RGB-D cameras such as the Microsoft Kinect sensor which allows to reconstruct complete 3D colored slices of the world at high frame rates. Such sensors made the 3D world easily accessible for a broader range of the research community. As a consequence, research which can reconstruct very large slices of the environment with high accuracy has been steadily maturing during the past few years.

SLAM approaches can greatly vary with the sensor technology in use like 2D laser, IMU, cameras, RGB-D cameras and so on. Cameras can be a very attractive choice since they provide a cheap and dense solution to perform SLAM tasks. They are today the preferred choice for applications where weight and compactness are critical like aerial robots [6][7][8][9][10][11]. Cameras can however only sense projections of the 3D world points on its camera image plane and hence have an inherent depth ambiguity. The depth of the points projected on the camera plane can be recovered using a multi-camera configuration like a stereo-camera system. In stereo vision two cameras with known relative position are used and hence recovering a pixel in one image and the other allow to compute the depth via triangulation. Stereo vision has been one of the most popular and earliest solutions adopted to provide mobile robots with 3D world recognition capabilities. Another solution to recover depth data from cameras without losing the benefits of using single camera over a more complex hardware setup consisting of multiple cameras, is to use multi-view depth map estimation. Multi-view stereo (MVS) aims at recovering depth data by matching and obtaining dense correspondences from a sequence of images acquired during separated timestamps. MVS comes however at increasing computational complexity since each pixel needs to run an optimization step to compute the most consistent depth value from cross observation and matching in multiple frames. Another important problem remaining in this respect is recovering the scale of a scene. Since the baseline distance between the acquired images is not exactly known, the depth map generated will only be true up to a scale factor. This scale factor can be recovered by introducing an object with known dimensions in a scene, moving the camera in a bootstrapping step with a known distance or using cameras in association with other lightweight sensors like IMUs, which combined with cameras can provide more accurate information on the nature of the sensory system motion. Camera and IMU association, termed Vision Aided Inertial Navigation System (VINS), has become an increasingly popular solution for weight constrained systems like UAVs. An in-depth observability consistency and accuracy analysis of VINS has been tackled in [12]. The association also allows faster motion estimation [11].

Stereo vision can be compared to more recent RGB-D sensors which have gained immense popularity. Stereo vision systems can fail in textureless regions or regions where texture patterns are repeated and hence where it is ambiguous to recover the exact pixel matching. They are also considered in general a more expensive alternative than consumer grade RGB-D sensors counterpart. RGB-D sensors on the other hand, are more sensitive to material reflectance. In some cases surfaces can induce light path distortion effects and hence inaccuracies. Another shortcoming of RGB-D cameras is that the angle of view provided by current sensors is still too narrow and hence using an RGB-D camera alone for navigation purposes require improved treatment via robust algorithms in order to limit drift effects. However, on the bright side, they also provide a viable solution in complete dark scenarios and work regardless of the textured or textureless property of the environment. Moreover, RGB-D cameras do not suffer from scale or depth ambiguity and hence provide a readily usable solution for 3D localization and mapping scenarios. RGB-D sensors often come in the form of an association of an RGB camera with an infrared camera which by pattern emission or time-of-flight principles can recover the depth information. As a result, RGB-D cameras are considered to be an easy solution to create colored 3D point cloud of the environment, which can in turn be directly fed to a tracker in order to compute incremental motions or to a mapper in order to add new data to an incrementally built map. All these good properties of RGB-D sensors added to affordable prices explain the immense literature which has been making central use of RGB-D cameras to perform 3D localization and mapping after the recent birth of the Microsoft Kinect sensor [13][14][15][16][17][18].

2D lasers solutions have also been widely adopted during the past decade to perform SLAM even though its impact in mobile robotics has been shadowed by the recent popularity of RGB-D sensors. 2D lasers have been behind some of the first systems which could map very large areas and run for very long time. They have very appealing properties consisting in tens of meters range of detection and broad angle of view while RGB-D sensors often provide measurements below ten meters only and with a narrow viewing angle at the moment. However, these sensors can only sense 2D slices of the real 3D environment and usually show centimeter accuracy against few millimeters only at very close ranges for cameras. Their simplicity and low density make it a good candidate to use on mobile robots which have to navigate in planar areas like office floors. Systems which have been extensively using lasers are omnipresent in the robotics literature and range from wheeled robots [19], humanoid robots [20] or UAVs [21][22][23]. Rotating 2D lasers are an alternative to bypass the 2D slicing limitation. By using a 2D laser on a tilting or rotating platform like the PR2 robot one can acquire 3D point cloud of the world knowing the exact motion of the moving platform. This howver require the robot to operate in a halt-scan-move scenario which adds on the needed time to complete tasks and hinders the reactivity of the robot.

Other SLAM approaches make use of Radio Signal Strength (RSS) in buildings where the RSS map is discriminative enough to perform localization. Some other approaches like FootSLAM [24][25] use IMUs only on pedestrians to recreate a navigable foot map and localize with respect to this map. In general one can assume that any sensor which can provide repeatable and discriminative enough data can be fed to a statistical framework which can build a map representation or world model then compute the sensor position with respect to such map.

The above discussion shades the light on the importance of smart use of multiple sensor configurations. An increased number of sensors in use will require additional calibration steps to setup the relative knowledge between the multiple sensors in use. Such calibration step will still inherently be prone to noise which will eventually propagate into the online map and motion estimate as well. Then, measurements from multiple sensory sources are not guaranteed to be perfectly synchronized due to transmission and processing delays. This problem is referred to as Out of Sequence Measurements (OOSMs) and has been the source of a broad range of approaches in literature which try to limit or predict its impact on further system estimates [26][27][28][29][30][31].

SLAM approaches can also be classified by the number of agents they can handle. The simpler case considers a single mobile agent with intrinsic sensing capabilities moving in its environment. Such robot does not hold any knowledge about other mobile agents nor do these other mobile agents participate in any decision making scenario. A more complex problem arises when each mobile agent seeks knowledge about other mobile robots locations or actions in order to plan time efficient actions or avoidance, optimal resource allocation such as worker robots in a warehouse, or even active cooperation like cooperative object carrying. Many examples have been provided in the literature to illustrate these scenarios and how to define the multi-robot SLAM problem [32][33][34][35][36][37]. In the same way multiple sensors can possess orthogonal properties which make them interesting to use in synergy, robots can also show complementary properties. For example [38] uses an aerial robot with a ground robot to explore damaged buildings. While the aerial robot can show improved mobility in space and can avoid ground obstacles easily, it suffers from severe battery limitations and hence time of flight constraints which makes it difficult to explore very large areas. Ground robots on the other hand can be much more energy efficient but show less ability to bypass large obstacles. The cooperation between these two types of robots in [38] consist in carrying the aerial robots as long as no obstacle is found and using an aerial robot when the obstacles can not be cleared by the ground robot. Both robots relative position are known and the maps built by each robot are fused into one map estimate which gives the overall rendering of the damaged building. Multi-robots exploration requires each robot to maintain a minimal knowledge about other robots. Such mutual knowledge is not always provided at start or can be lost at some point during the mission. The robots hence need a way to discover relative positions either by comparing mutual maps or by assigning rendez-vous or by discovering common landmarks. Multi-agent SLAM systems can show much greater complexity and accurate multi-agent map reconstruction can require optimization not only on the mobile robot's own trajectory but also on other agents trajectory [37].

SLAM problems can also differ through the inherent dimension of the problem they try to solve. 2D SLAM approaches project the more general six dimensional (if a minimal representation is used) localization problem on a 2D plane of choice. In such configuration, the SLAM requirements are to estimate the XY position on the plane along with an orientation angle and an estimate of the up-to-date map. Needless to say that lower dimensional problems can usually be solved in shorter time. 2D SLAM is a well understood problem and life-long systems over large areas have already been provided in recent robotic literature. The obvious disadvantage of 2D SLAM is that they constraint the problem to a plan and hence all information carried out of such plan become unavailable which usually limits the navigation to perfectly planar floors. This assumption is clearly violated when using aerial robots or navigation through uneven grounds. 3D SLAM approaches on the other hand usually try to build some sort of 3D representation of the environment (either dense or sparse) along with the 6D position of the mobile agent. The problem is more computationally expensive as dense three dimensional sensory data is an order of magnitude more expensive to handle. With such aspect in mind, the reconstruction of full and accurate 3D maps of large environments in real-time still remains a challenge even with the availability of modern hardware. An interesting approach lies at the boundary between 3D and 2D SLAM systems and is called 2.5D SLAM or Manhattan world assumption. In a 2.5D SLAM the main assumption is such that the world is made in majority of regular structures like walls. Such knowledge around the world makes it possible to extract additional information without the availability of complete 3D sensing capabilities. For example using a 2D laser with an IMU (i.e plane and plane orientation sensing) makes it possible to recover a 3D representation if we can forecast how points lying off the ground will project on average on the ground plane. Such 2.5D SLAM retains the complexity of a 2D SLAM problem with the possibily to build full 3D dense maps. Of course if the assumption is violated or is not respected by a majority of obstacles lying in the scene then the computed localization and map estimates will be poor. An example of a SLAM process using IMU and 2D laser data on an aerial robot is shown figure 1.

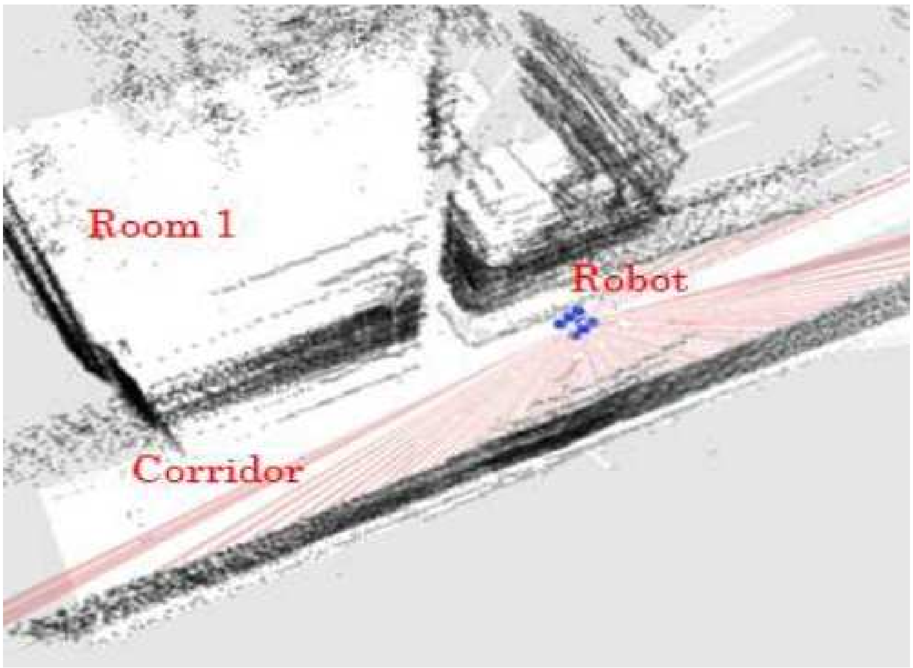


図 1.1: Mapping with 2.5D assumption

The map estimate built throughout a SLAM process can take on many forms. They can represent a dense form of the world like occupancy grids [39][40] or signed distance function based grids [41]. In other cases, maps can consist of a sparse representation of the world. For instance the knowledge about the positions of well chosen landmarks or features is enough to localize in a given environment [42]. A landmark is considered to be good if it shows some properties like repeatability under changes in viewpoint, lighting conditions or scale and discriminative power which allows to infer with high probabilities the identity of the landmark being observed. If a sparse representation only is enough to localize accurately in the environment, a sparse representation makes it harder to identify obstacles and plan obstacle avoidance policies accordingly. A dense model on the other hand can store rich representation about the world but such attribute comes at the expense of a more involved process like raytracing to fuse new data with past map models.

Based on the above discussion, map representations can be sparse or dense but tracking algorithms in use can also show dense or sparse properties. This brings us to a fundamental difference between SLAM approaches which can be labeled either dense or feature based. Feature based SLAM approaches extract form the set of all sensory input only a small subset to represent knowledge about the world called features. For instance, from a 100000 points made point cloud the feature number is usually set to two order of magnitude less to about a 1000 representative points. The advantage of such procedure can a be immediately understood as the dimensionality of the problem to solve is reduced to only a subset of the full problem. As it has been pointed out before, these features need to verify certain properties which mainly are robustness against sensor dependent transform like affine transforms for camera, lighting changes which can occur depending on viewing directions, scale and also need to show enough discriminative power. To do so, at a detection stage a feature based algorithm starts by extracting candidate points, usually corners or blob features with some of the most successful approaches including Shi-Tomasi [43], FAST [44] for corner detectors, SURF [45] and SIFT [46] for blob detectors. The feature detection phase constructs a set of candidate points. In order to match these detected features across frames it is essential to construct a discriminative knowledge about each feature. Such knowledge is called a descriptor and aims at allocating a discriminative vector to each feature. Such information is usually extracted from some properties like gradients or intensity values in a neighborhood around the candidate feature point. Examples of popular feature descriptor include SURF, SIFR, BRISK [47] or FREAK [48]. All the descriptors share the common objective of constructing a knowledge which can show robustness against viewpoint change, scale an so on. Computing descriptors can be very computationally demanding and can induce an overall slowing of the SLAM pipeline. For such reasons, some approaches use less expensive computation schemes such as small binary feature tests like FERN [49] or simpler averages like sum of squared intensity difference or sum of absolute intensity difference around the target point. Once correspondences have been established, the last step of a feature based tracking pipeline is to compute the incremental camera motion which minimizes an energy function given the matching previously obtained. To do so, it is important to consider that the matching process is not perfect and false associations can occur. Since the subset of extracted features is relatively small, such false correspondences can yield serious errors during the motion estimation step hence the need to adopt a strategy to deal with these so called outliers. Dealing with outliers can either consist in lowering the impact of these mismatching on the final output results or discriminating outliers and inliners then using the subset of inliners only to estimate the incremental camera motion. The first option consists in assigning variable weights on each residual to use in the total energy function with higher weights being assigned to associations which are more likely to be true. The second option is more popular and usually bases on Random Sample Consensus (RANSAC) or other voting schemes in order to discriminate the set of inliners from the set of outliers and retain only the inliners to compute the updated motion estimate. Dense approaches on the other hand use all or a fraction of the input sensory data without any computationally demanding discrimination of sensory data as feature and non feature points. Doing so, dense approaches make use of all the information available and which can be missed by feature based approaches. In this regard, dense approaches usually yield superior estimates in term of accuracy, but as it has been stated before have to often handle two order of magnitude more data. Since every pixel or point at the sensory input stage only requires a simple computation compared with feature based approaches, dense approaches are usually good candidates for parallelization and have been recently benefiting from the advances in parallel computing technologies. As a consequence, dense approaches have steadily replaced feature based approaches in the most recent literature.

Finally, the remaining important distinction to make between SLAM approaches bases on which statistical framework they use to represent and propagate uncertainty inherent to the incremental motion estimation process. A popular approach during the first half of the past decade consisted in the so called filtering process. In its most popular form, a filter compresses all past data observations and intermediate estimates into one up-to-date average and expresses the uncertainty involved in the process in a covariance matrix. Such compression can be seen as a lossy procedure where past trajectory or knowledge can not be recovered but lies all averaged and compressed in the most recent few first orders moments. Popular filters used in robotics literature include Gaussian filters and more specifically Kalman filters derivatives like linear, unscented, extended Kalman filters or information filters along with Monte Carlo based approaches. Gaussian filters have been behind the first large scale successful SLAM approaches such as [42]. A shortcoming of gaussian based filter is first the requirement on the uncertainty to follow a normal distribution, which is not always the case, then their computational complexity scaling quadratically with the number of state variables. Early filtering based approaches represented joint probability over the mobile agent and world landmarks such that at each update step, a new estimate of both the map and robot state needs to be recomputed. Kalman Filter based approaches were rapidly replaced by less expensive particle filter based systems [50][51][52][53] where the distribution over the map and robot space is represented by a finite sample of particles. Each particle then represents an estimate of a robot state and associated map. Of course, more particles represent a more accurate system which samples more closely the real distribution but this comes at the expense of an increasing computational complexity. Particle filters can be considered more flexible to use for mobile robotics since they can model random distribution and can be tuned according to the available computational resources. An early review of probabilistic derivation of SLAM problems can be found in [54][55]. The second framework which has superseded filtering based approaches in recent literature is based on graph optimization. The processing pipeline adds a new node to a graph each time a new estimate of the world state is computed along with an edge representing the uncertainty of the incremental motion. As such, the graph represents a discretization of the full SLAM problem. It stores all past data which hence can be revisited, recomputed and propagated again. This process can yield an ever growing graph and hence will end in a loss of real-time capabilities of the system after some time. Uncertainty can be minimized upon loop closures i.e when the robot navigates through previously visited areas. In this case, an edge loops back to a previously created node and such closing back represents an important constraint to reduce the uncertainty over the whole loop trajectory. Loop detection in robotics literature can be based on matching visual appearance [56][57][58] or any other frame to frame matching algorithm such as laser scan-matching. In case of frame to frame matching, since the number of node candidates to match against can be high, the matching can be performed at lower levels then the transform between present and past nodes refined upon first detection. Recent insights in sparse linear algebra in association with the full SLAM derivation has led to increasingly fast and efficient graph optimization approaches. The proposed literature for graph optimization is immense and has reached a state of maturity [59][60][61][62][63][64][65] with g2o being probably the most used solution in recent literature [66]. An overview of graph based SLAM formulation is provided in [67]. An example of a 2D map of the 7th floor of Building 2 of the University of Tokyo closed with laser scan-matching and optimized through iSAM [60] is shown in figure 1.

Finally, the pioneering paper by Klein and Murray [68] showed how SLAM can depart from the expensive joint probabilistic estimate of robot state and map at each update and that the SLAM processing pipeline could be divided into two distinct processes : the first one labeled front-end SLAM performs tracking ie computes the robot state estimate at higher frame rates while the second process labeled back-end SLAM updates map estimates at lower frame rates. The back-end SLAM also maintains a graph to perform global optimization upon loop closure. New nodes are added to the graph only when enough motion is detected or the quality of

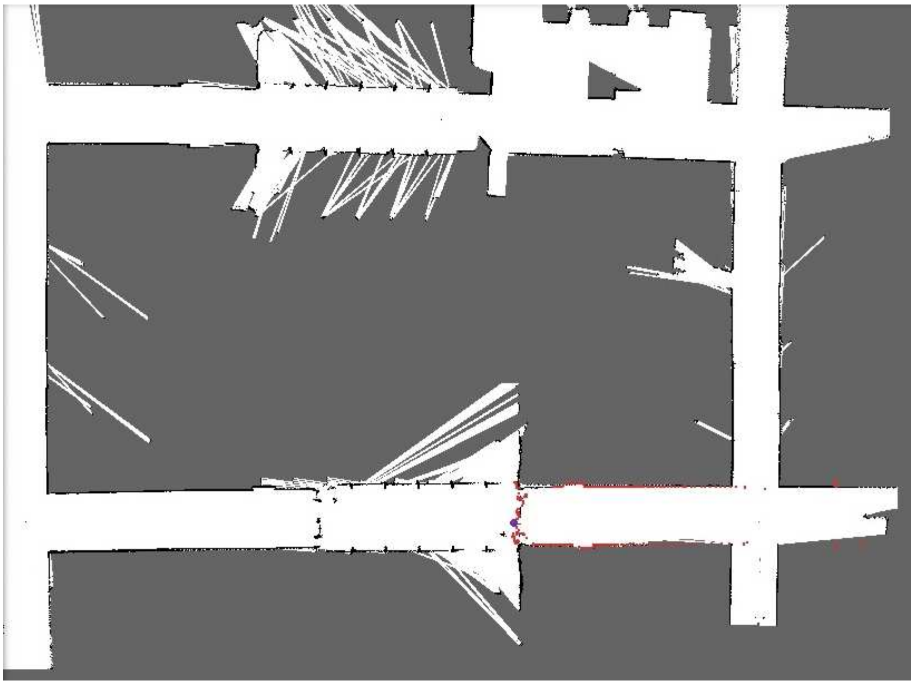


図 1.2: Large 2D environment with loop closing

tracking becomes poor. This last framework became mainstream in most recent SLAM systems.

We have provided in this chapter a taxonomy of different SLAM approaches and how they have evolved throughout the past two decades. In next chapter, we present an in detail review of the most relevant SLAM literature for our work. An outline of the present thesis is provided at the end of next chapter.

第2章

Related Work

Laser based approaches have been the corner stone of many of the life long SLAM approaches in the past decade. As it has been previously pointed out, lasers have excellent range and angle angle of view properties. The only limitation is most of the high speed and affordable laser solution available such as Hokuyo [69] or Sick [70] provide range measurements only relative to a plane. Tilting lasers or rotating heads can provide a full point cloud but these systems introduce latencies and are very sensitive to fast motions. Other technologies like Velodyne sensors [71] provide above a million points 3D point cloud but such solution, if ideal for outdoor navigation in large environments, is still not dense enough to model fine details compared to high resolution camera based solutions. Visual SLAM on the other hand is still a highly active research area in Simultaneous Localization and Mapping, robotics and computer vision community and has been sustained by the recent blooming of UAV research and smartphones market which both require lightweight and compact sensing technologies to be used. In this context, cameras can be regarded as an attractive sensor choice for on-board robotics sensing especially for systems where weight considerations are of essence. They also provide a cheap, dense and high speed solution for the Simultaneous Localization and Mapping process. Camera sensors map 3D real world models to 2D projections on the camera image plane. Doing so, a single frame captured from a camera alone fails to capture the real world 3D geometry without any knowledge on the visualized scene, effectively introducing a scale ambiguity. 3D geometry can be recovered within a stereo or multiple camera setup where the relative transform between the different camera sensors is provided or must be inferred by multiview stereo through overlapping successive views. The second option is particularly attractive since it takes advantage of the compactness and low power alternative using a camera alone represents, which takes on it full sense on recent tablets and smartphones. Monocular SLAM comes at the expense of more complex algorithms which have to deal with depth map estimation and denoising from multiview stereo in addition to the more fundamental tracking and mapping issues. The robotics literature with the recent gain in popularity of UAV navigation and UAV camera based stabilization systems have proposed multiple and different visual odometry and visual SLAM pipelines such as [9][11][7][72]. Visual odometry approaches can be regarded as a subcomponent of full SLAM systems since they consider the robust estimation of the egomotion of an agent in two or multiple frames without consideration of the global consistency in the joint full map and trajectory estimate. For small workspaces, robust and accurate visual odometry approaches can prove fast and remarkably converge to a repeated map model without introducing serious drift. RGB-D camera recently introduced to the consumer grade market can be considered as an even more remarkable sensory solution for mapping indoor scenes since they provide directly the depth information which has to be computed through complex processing in the case of Monocular SLAM. RGB-D based approaches have been increasingly gaining more maturity during the past years and show great similarity with recent monocular SLAM approaches when it comes to the tracking step. In the following section, we give a review of some of the most relevant SLAM approaches as well as some of the applications on real-time robotics systems.

2.1 Filtering Based Approaches

If early approaches in the 50s tackled reconstructing 3D structures from successive 2D images in an offline fashion, later approaches provide online computational capability as a central attribute. Early real-time approaches include the work from Davidson [42]. Davidson's work is particularly interesting since it proved robust in larger areas and for longer time than previously established. In an initialization step, the metric scale is estimated by introducing an object of known dimensions in the scene. The SLAM process relies on sparse salient features which first order uncertainty along with position and velocity uncertainty is constantly updated through an Extended Kalman Filter with a constant velocity motion model. Landmarks are dynamically inserted and deleted from the map. A new landmark is not immediately inserted to the map upon first observation but the depth estimate of the feature is quantized in a closed range along the camera ray. The new landmark is effectively used in the EKF process only when the estimate on depth uncertainty becomes below a threshold. Finally, given the number of landmarks N inserted in the state vector the computational time requires is in $O(N^2)$. The system, labeled MonoSLAM, has been successfully applied in [73] to the HRP-2[74] robot navigation. The need of a separate initialization process for new features has been suppressed in Montiel et al. [75][76] by using a unified parametrization which does not require any special initialization treatment of new landmarks before integration in the EKF process. By using an explicit parametrization of the inverse depth, depth estimates spanning finite and infinite values for low parallax motions show gaussian uncertainty and hence can seamlessly be integrated through the EKF process. Features at infinity can effectively contribute to the estimate of the bearing of the sensor and take on finite values when enough parallax has been recovered. Eade and Drummond[77] on the other hand take advantage of the independence of landmarks pose estimates with known camera trajectory to derive a FastSlam[78] particle filters based monocular SLAM system where each particle represents jointly one estimate of the position and an estimate of each landmark position. The system runs at frame rate with a computational complexity for M particles and k landmark observations of $O(Mk)$ which is significantly lower than the $O(N^2)$ computational complexity of EKF based systems. Eade and Drummond also use an inverse depth parametrization. Kalman filters are used independently in each particle's map to estimate the inverse depth parameter and a new landmark is inserted in a particle's map when the corresponding XYZ has small enough uncertainty. Grisetti et al. later introduced the Gmapping framework [52][53] which is probably still one of the most used SLAM approaches to date and the default navigation support for robots like PR2 by WillowGarage [79]. Gmapping considers the problem of mapping using grids and Rao-Blackwellized particle filters. Each particle stores the state information along with a grid map estimate. Gmapping proposes two improvements : the first one draws the particles from an improved proposed distribution based on the sensor likelihood while more traditional approaches use the odometry model as the proposal distribution. The sensor likelihoood can be established using any scan-matching based approach for laser based SLAM but the idea can be generalized for other sensors. We recall here that scan-matching is the mean by which a laser range data can be aligned with either a previous scan or a map. The second improvement Gmapping proposed consists in reducing the particles depletion risk and keeping the richness of the particles representation by triggering a resampling step only when the dispersion of the weight is above a threshold. In such case particles are resampled according to their importance weight. Since resampling is performed only when really needed, the risk of throwing away good particles is reduced.

2.2 Graph Optimization Based Approaches

Eade and Drummond [80] proposed to use local sub-maps updated through non linear optimization called nodes which are incrementally inserted to a global graph. Global optimization takes place upon new edge and node insertion. It allows to refine the state estimates through loop closing and shared local landmarks constraints. The local optimization effectively updates both a local mean estimate of local features and an uncertainty matrix acting just like a filtering EKF counterpart. The local update process uses an inverse depth parametrization to limit non linearity effects which come through using XYZ parametrization. A new node is inserted if not enough landmarks are retrieved or too much non linearity is detected (via the trace of the Hessian matrix).

Klein and Murray's PTAM [68] can be considered as one of the major breakthroughs in the SLAM literature. PTAM introduced the idea that tracking and mapping can be split and run on two different threads with two different updating frequencies. Doing so, tracking is no more probabilistically attached to the mapping component, which is a major departure from filtering approaches of the same period and achieved unprecedented degree of robustness and agility. The mapping thread does not need to perform at tracking rate and is run through bundle adjustment of carefully selected keyframes. One main difference between PTAM and Eade and Drummond [80] is that only selected keyframes are used to construct the final map while in the second approach nodes are further filtered. In [68] features comprise sparse textured patches as well as the keyframes where they first appear. The sparcification of the selected keyframes is a key element into real-time update of the bundle adjustement step. During the tracking process, patches are retrieved with comparing zero mean SSD scores of Fast features in an area around the projected image plane location. The camera is tracked from the most recent keyframe through robust non linear optimization based on Tukey M-estimator by minimizing the reprojection error of 3D map points and corresponding patches in the input image. Depth information on map points is retrieved by founding correspondences between two adjacent keyframes by zero mean SSD comparison along the epipolar line. The initial map scale is initialized through a user guided bootstrapping procedure where the initial two keyframes are supposed to have a metrically fixed baseline. One main shortcoming happens when tracking is lost and a map is started over. In such scenario, a scale consistent with the first estimate might not be recovered. PTAM has been further improved in [81] where the robustness of the system during fast motion has been improved by using edglets which show more robustness than pixel patches under motion blur. [81] also added an inter-frame rotation estimation to the tracking procedure. A modified version of PTAM has been successfully used in the european project sFly [6] to achieve autonomous navigation in GPS denied environments for Micro Aerial Robots. sFly relies on PTAM to provide a high frequency (30Hz) non drifting (as opposed to optical flow based approaches also used for UAVs platforms) solution which is necessary to achieve good position control for the MAVs. It is interesting to note that an additional EKF is used to fuse the output of PTAM with an on-board IMU data in order to provide a scale metric state estimation of the position of the robot since PTAM alone provides outputs which are scale ambiguous and which depend on the bootstrapping step. Methods which associate IMU to additional sensors like cameras or lasers are discussed in further section. The list of modifications to PTAM has been described in [10]. First of all, in order to minimize the computational complexity, the number of keyframes used is set to a maximum. Setting the number of keyframes to infinite represents the original PTAM approach while setting it to two represents a pure visual odometry incremental approach. Then, only features from higher pyramidal levels are retained which induces drastic speed-ups during keyframe creation. Moreover, a speed controller allows to stabilize the UAV upon failure and limits the jumps in scale estimate during reinitialization. Finally, an inverted data structure is implemented which limits the number of points which need to be reprojected only to the visible keyframes.

2.3 Dense Monocular Methods

An excellent summary of Feature based front-end SLAM pipeline in computer vision can be found in [82] and [83]. However, with parallel computing solutions becoming more mainstream and providing the latest mathematical inside in dense SLAM approaches, dense methods have steadily replaced the feature based pipeline. When feature based methods discriminate features and non feature data, dense methods on the other hand tend to use all data available hence lowering the risk of missing importance information. This provides both enhanced tracking accuracy and dense reconstructions of the 3D world which has superior utility for planning and navigation compared to sparse maps generated by feature based approaches. In this section we review some of these most successful dense approaches.

Newcombe et al. introduced DTAM in [84] one of the pioneering approaches in recent years which uses every pixel to perform live reconstruction and tracking. DTAM starts with a bootstrapping step which relies on features tracked from stero frames in order to initialize the first keyframe. Passed initialization, the system solely relies on a dense derivation. On the mapping side, a keyframe is associated with a cost volume where each pixel in the keyframe maintains a limited number of depth candidates included in a restricted interval. Each new frame acquired in the neighborhood of the keyframe adds in a total cost data average based on a photometric error. An exhaustive search over all possible depth candidates can solve the minimization problem but textureless regions will still have poor results. In order to compute depths in textureless regions more accurately, DTAM takes the hypothesis that such regions have smooth depth variation and proposes to add a Huber norm regularization term to the problem. The whole is further solved using a primal-dual approach. On the other hand, tracking uses a coarse-to-fine scheme in a Lucas-Kanade way to estimate the motion. The registration starts by estimating the rotation through frame to frame alignment. The full 6D pose is then refined by projecting the dense model on the image. Robustness is achieved by selecting only pixels which photometric error is below a threshold. DTAM provided the first real-time capable full dense reconstruction system and has achieved superior accuracies when compared to other feature based methods.

Forster et al. proposed SVO [72] a semi-direct approach which removes the need to explicitly run the feature detection-feature matching step on each frame. SVO makes use of two parallel steps namely a real-time tracker and a mapper which adds keyframes and initialized new depth data. The tracker starts by computing the frame to frame transform by minimizing the photometric based reprojection error of point maps on both frames. This provides outliers free good first guess of the camera transform between two consecutive frames. A second step then proceeds to frame to keyframe alignment while a last steps performs motion only and structure only bundle adjustment. Local bundle adjustment can additionally be performed. Everytime a new keyframe is inserted, a feature detector selects interesting points to consider for the tracking step. Each feature depth is computed by using a filter. The process starts by assigning an average scene depth value and matching pixels along an epipolar line computed through the tracking result lying on an area defined around the initialized depth value and which has the best patch to patch correlation. The depth value is then computed through triangulation. Once the depth filter has converged, the 3D point is added to the map and subsequently used for tracking purposes.

Engel et al. introduced LSD-SLAM [85][86]. LSD-SLAM is a semi-dense approach in the sense that only points close to strong enough gradient regions are considered which partly removes the complexity DTAM has to go through to smooth regions with uniform texture. LSD-SLAM builds on three components. The first component is a tracker which tracks live frames against keyframes by defining a photometric cost function and running a robust Gauss Newton optimization step on Lie manifolds which uses Huber weights. The second component is a depth estimation component which is used to refine i.e add new pixels or replace the current keyframe if too much motion has been accumulated. Keyframes are initialized by projecting points from a nearby keyframe. New depth measurements are added through pixel-wise stereo comparisons and merged with a filtering based approach. A propagated prior allows to constraint the search interval. To ensure scale awareness, each keyframe is scaled to have a mean depth equal to unity. Each keyframe is added to a third component which maintains a pose graph. Constraints are added by direct image alignment between the first keyframe and neighbouring keyframe in the graph. Such alignment is performed to solve a similarity transform based problem to take into account accumulated scale deviations. Once the best candidate to loop closure has been found a solution to the global optimization problem is computed using g2o [66].

2.4 RGB-D based methods

One of the early approaches to solve the full 6D SLAM problem has been proposed by Nuchter et al. [87]. In their approach they use a tilting SICK 2D range laser mounted on a mobile robot in a stop-scan-go to acquire 3D scans. Successive 3D point clouds are first aligned combining odometry and an octree alignment heuristic to provide a first guess of the 6D robot pose. Then, a point-to-point ICP step is performed to refine the pose estimate. Data association during the ICP scan matching has been sped up using KD-tree search methods. Finally, global optimization via loop closure detection has also been implemented. Subsequent approaches in the literature tried to emancipate the 6D SLAM problem from the need of readily available odometry, such as wheel encoders, which is the case when mapping with hand held camera. Researchers have also proposed abundant literature to increase scan-matching speed, to limit memory consumption, increase robustness to outliers or efficiently solve the problem on multicore machines and GPGPUs via adequate parallelization. In [88][89] Liu and Chen et al. used a human operated backpack where several 2D laser cameras and IMU have been mounted. Lasers and IMus are used in ICP based scan-matching to calculate the incremental pose between successive nodes inserted in a graph. Other constraints which compute the tranforms between distant camera images are also inserted. Optimization uses Levenberg-Marquardt algorithm which minimizes a Sampson reprojection error. Loops are detected using FAB-MAP[57] and the graph is further optimized using TORO [63]. The recent advent of cheap and high quality RGB-D sensor such as the Microsoft Kinect or more recently Microsoft Kinect2 has strongly stimulated 3D reconstruction and 6D tracking methods. RGB-D sensors have been since, and more than ever, playing an important role in many intersecting field such as computer vision, mobile robotics or computed graphics with applications ranging from Simultaneous Localization and Mapping in GPS denied environments, textured mesh reconstruction like live human 3D modeling [90] or augmented reality [13][15]. A review on Kinect-like RGB-D sensors and some of the related recent reconstruction methods can be found in [91].

Henry et al. [92][93] has proposed one of the first approaches using a hand held Microsoft Kinect sensor named RGB-D SLAM. The method starts by constructing a sparse 3D point cloud of SIFT features detected from the RGB frame and transformed into 3D points using the depth information. RANSAC is then run to compute an initial transform estimate between two adjacent frames. Based on this first visual based estimate, ICP refinement is run by finding neighbors between a source point cloud and a target point cloud through fast kd-search based on euclidean distance. ICP minimizes a point-to-point error for the sparse features and a point-to-plane error for the dense points associations. In [93] reprojection error (referred to RE-RANSAC) is proved to provide better tracking result and is used instead of 3D point euclidean distance (refered to EE-RANSAC) for 3D features error metric. The non linear optimization is solved using Levenberg-Marquardt algorithm. New keyframes are inserted in a global graph if the number of 3D SIFT features becomes below a threshold. Each time a new keyframe is detected, an attempt to visually match SIFT features from previous keyframes takes place. Keyframes which are matched against are those present in a neighborhood of the current pose and those which have similar appearance using a vocabulary tree [94]. Global optimization uses TORO [63]. Finally a surfel map [95] integrates all points from all keyframes into one consistent and concise representation of the world. A surfel consists of a location, a surface orientation, a patch size and a color and stores confidence about each surfel. Surfels with low confidence are subsequently pruned from the map.

Endres et al. [96][97] also rely on matching sparse visual features from frame to a subset of past keyframe and updating the motion estimate through RANSAC. They again build a pose graph which is further optimized upon loop closure via g2o[66]. As a final step, a global map is created offline by concatenating all point cloud data from keyframes into a volumetric octree based map by means of Octomap [40]. The system runs at about 10Hz and shows few centimeters RMSE error.

Similarly, in [98] Hunag et al. introduce the FOVIS RGB-D based SLAM approach for UAVs. They also rely on sparse features to compute the motion increment. Since speed is of essence in their application, they rely on FAST feature detection coupled with an 80bytes descriptor made of brightness values of neighboring pixels extracted from different pyramidal levels of the input image. An initial position guess is estimated with direct minimization of pixels from neighboring frames. This position is further refined using nonlinear least square optimization of the sum of square errors of the feature descriptors. Motion increments are computed by matching against keyframes to allow slow accumulation of drift. The visual odometry step runs on an on-board processor while loop closure detection global graph optimization run on a remote machine. The MAV transmits RGB-D data to an off-board laptop, which detects loop closures, computes global pose corrections, and updates a 3D log-likelihood occupancy grid map. The visual odometry process of FOVIS runs at 30Hz.

St端ckler and S. Behnke [14][99][100] have used octrees allocated on the CPU side to represent a surfel map which stores joint shape and color distribution in a multi-resolution fashion. Each node of the octree stores up to six surfels depending on the viewing direction and where each surfel updates two sufficient statistics to recover the mean and covariance of stored 3D points. Then, each surfel is associated with a shape and texture histograms based descriptor describing the local neighbourhod in order to improve the quality of data associations between surface in subsequent registration step. Also, in order to speed-up the data association step, the 26 neighbouring nodes are precomputed. The approach also uses the RGB-D sensor model to set the maximum mapping depth depending on the depth value. Each map insertion then consists of two order of magnitude less data to insert. Keyframe to current view registration is performed first using a Levenberg-Marquardt non linear least squares minimization then the solution is refined through Newton's method by adding second order derivatives. A global graph is also built and further optimized upon constraint detection via g2o[66]. The method has been shown to perform at 10Hz on a resolution at 5 cm. Memory storage requires some 50Mb for a chair model.

Tykkala et al. [15][101] have developped an incremental approach which minimizes the photometric error between sensory images and selected keyframes. The approaches extract points with salient gradients then minimizes a photometric error with a weighted gauss-newton non linear optimization. The weight are computed with a Tukey weighting function. The tracking step takes about 20 ms on a low end GPU. In [101] the method is augmented with post-processing watertight poligonization step from input point clouds using the Poisson method.

Steinbrucker et al. in [102] depart from the sparse features estimation approaches and write down a direct minimization of the linearized photometric error between consecutive frames. Since such reprojection error is non convex, they derive a resulting linearized energy term which is solved in a coarse to fine approach to cope with large camera motions. This approach proved fast (10Hz on CPU) given small frame to frame motions. Kerl et al. introduced DVO, an impressive fast and high quality keyframe to frame dense approach. In [103] Kerl provides a probabilistic derivation to estimate the frame to frame RGB-D camera motion by direct minimization of the photometric error. Such derivation allows to extract the role of motion prior and

and the sensor model. Using a suitable motion prior allows to track faster camera movements while a suitable sensor model distribution allows to cope more efficiently and minimize the impact of outliers on the final estimate. DVO proceeds on a coarse to fine scheme with a Gauss-Newton solving approach. The weight assigned to the residuals are derived from a t-distribution following a robust estimation scheme. Extensive experimentation showed how t-distribution outperforms Tukey based one in modeling the residuals and hence yields improved accuracy even in the presence of dynamic objects in the scene. [103] can be seen as a generalization of [102]. DVO in [104] has been extended to optimize both intensity and depth error using keyframe to frame direct minimization. A new keyframe is inserted in a global pose graph when an entropy based ratio between selected frames is below a threshold. Candidates keyframes to loop detection are taken in a radius around the current keyframe and matched against in a coarse resolution. Upon loop closure constraint detection, the graph is optimized using g2o[66]. DVO showed improved tracking performance compared to other state-of-art systems. Also, the combination of both photometric and depth error minimization yields superior results and each of the errors used alone and provides more robustness in case of environments lacking either texture or structure. Steinbruecker et al. [105] used DVO to reconstruct and optimize the camera trajectory and generate a keyframe based map of an entire office. They presented a multiscale SDF stored as an octree on the GPU side in order to fuse all keyframes in one volumetric map bearing in mind memory requirements. The camera sensor model is taken into consideration when selecting the octree's level to write into and an efficient octree representation on the GPU is thoroughly described. The approach needs about 200 MBytes to store an environment with the size of a room.

Newcombe pioneered the KinectFusion method in [13][106]. The approach reconstructs in real-time high quality dense mesh representations of small workspaces. Memory allocations and computations are all run on GPGPUs. The original approach uses geometric data only to calculate pose increments and update a dense map. The dense map representation used by KinectFusion is a bounded 3D voxel uniform grid at fix resolution which represents a Signed Distance Function [41]. In practice, the map only represents a truncated signed distance function TSDF which represents values for an interval around the zero crossing surface only. Each voxel stores the current signed distance value to the zero surface as well as the accumulated weight. A map is incrementally generated by projection of the map voxels on the current image and updating a weight through a bounded running average. Bounds on the weight allow smooth handling of dynamic scenarios as well. Rendering the current camera view is done through raycasting the SDF. Sensor pose estimate is recovered through dense ICP. Point correspondences are generated through fast projective data association between the sensory image and a map surface predicted by raycasting. Grossly incorrect correspondences are eliminated by setting a threshold on euclidean distance and normals angles. The minimizing criteria is the point-to-plane distance. The energy function is linearised around the previous pose estimate and a solution is obtained via Cholesky decomposition. KinectFusion by using frame to model tracking could close small loops without use of any global optimization scheme and shows superior results in terms of accuracy compared to the frame to keyframe approach. The whole pipeline on a room sized environment can be reconstructed at 30Hz on a high end GPU. Zeng et al. [107] extended the original KinectFusion method by representing the TSDF as an octree allocated on the GPU instead of the more memory demanding uniform grid. Doing so their approach requires 10 times less memory and runs slightly faster than the original KinectFusion. Whelan on the other hand in [108] introduced Kintinuous an extension to KinectFusion for larger scale environments through a different approach. Whelan uses SDF volumes which vary dynamically as the camera enters larger unexplored space. As the camera pose shifts away from the center of the SDF, previously mapped region is extracted in the form of a mesh representation and added to a pose graph. New unmapped area is then inserted. The graph loop detection bases on DBoW [58] and pose graph optimization is handled by isam [60]. In [109][110] the approach is further augmented by adding color as well as a photometric constraint in the camera pose optimization. The photometric constraint is expressed between two consecutive RGB-D frames in order to avoid SDF sampling and unusual coloring due to changing lighting conditions artifacts and consist in minimizing the difference of intensity values between correspondences found by projective data association. The ICP based geometric error and the frame to frame resulting photometric error are combined through a weighted sum. Adding both pose graph and map deformation steps upon loop closure constraints, Kintinuous shows high quality reconstructed maps over larger scale maps and overcome two main limitations of Newcombe's KinectFusion which are bounded volume reconstruction and poor quality tracking in planar areas (given enough texture).

Bylow et al. [16][111] also used TSDF to represent the live constructed map and a frame to model tracking approach to estimate the camera motion. They use a direct minimization on the signed distance function extracted by projecting the sensor cloud in the TSDF. They proved that such direct minimization outperforms the ICP and projective association based approach by KinectFusion especially in the case of faster motion. Their approach uses depth data only and hence is a pure geometric method.

2.5 Discussion

Early structure from motion or SLAM approaches were mainly incremental systems which estimated successive odometry while loop closing was performed offline. The successive gaussian filtering (Extended Kalman Filter, Unscented Kalman filter, Information Filter...) based approaches were considered the first real-time systems robust against drift which can operate in relatively large areas. Gaussian filters approaches were gradually outclassed by faster monte-carlo based ones. Particle filters hence proved a more flexible and practical solution for many robotics applications where real-time speed and problem size was of essence. Recent insights in the structure of SLAM graph optimization and its relation with sparse linear algebra provided with clues on how to dramatically improve the execution speed in batch and incremental modes [59][66].

The fundamental question which remains here is how do filtering based approaches compare to graph optimization based ones ? Such analysis has been carried out by Stradat et al. in [112]. They investigate some of the most successful filtering and bundle adjustement for Monocular SLAM based methods but their finding can be generalized to other forms of SLAM systems. The conlusion they reached to is that bundle adjustment is superior to filtering approaches in almost all cases and independently of the scene structures and the nature of motion. Hence with full SLAM solutions computable in near real-time more recent systems abandoned the full probabilistic state propagation and instead compute a solution by separating two different process : a tracking process which computes incremental odometry and a mapping process which adds new data to a map and runs global optimization such as it has been pioneered in [68]. These two processes have also been conveniently labeled as front-end SLAM and back-end SLAM in further systems. Almost all recent approach follow the front/back-end separation and run global optimization in background while a certain form of odometry is used to compute the incremental motion.

On the front-end side, most of the approaches following the advent of cheap RGB-D sensing technologies were feature based at first. Feature based approaches can be attractive since they allow to map the initial problem from 100000 points made clouds usually dealt with to a two order of magnitude reduced problem but potentially miss valuable data. If feature based applications can yield enough camera tracking accuracy for most of robotics application, the map created conventionally contains sparse points only, which comes short for robotics systems which need to perform subsequent planning, object recognition and manipulation tasks. Later approaches use a dense formulation on all or most of available sensory data in the input stage to estimate odometry. The formulation can either be written for image intensities or depth map values hence yielding a photometric or geometric approach respectively. Dense approaches as opposed to sparse feature based approaches use all available data and hence yield approaches with superior accuracies averaging an increased time required for processing data. The last time constraint does not hold true anymore in the past years since the problem can easily be parallelized which yield important speed-ups on modern hardware, while the feature descriptor computation step, even with the availability of GPU based implementation like CUDA-SIFT, remains the speed limiting factor. This is why some approaches have combined keypoints detection routines with a rather cheaper patch of neighboring pixels from which correspondences are found using classical metrics such as sum of squared differences (SSD) or sum of absolute differences (SAD). Dense approaches usually try to minimize a sort of reprojection error. With RGB-D sensors, both depth image and RGB image can be used for odometry computation. Expressing an intensity error yields accurate motion estimates with environments with enough texture while expressing a depth error yields accurate motion estimates in environments with important geometrical features. The depth term also adds robustness against sudden changes in pixel intensity values due to auto-exposure. Such dense optimization schemes yield good results for certain ranges and types of camera motions but ICP based methods can perform better for certain faster and larger camera motions. Moreover, model based front-end approaches proved superior accuracies compared with frame to frame or even less drift prone frame to keyframe methods. This can be easily understood by means of which frame to model matching is equivalent to frame to all past keyframes matching which is naturally superior in comparison. Then, the live reconstructed 3D model usually updates a weight parameter which incrementally denoises the registered sensor inputs at each new sensory data integration. Maintaining a live reconstructed map however asks for choosing clever data structures to preserving fast points access while limiting memory requirements. An important limitation of model based approaches is such that the model can usually not be efficiently updated in case of important change in past data which is usually the case of loop closing update. In such case, most of the time, a simple solution consists in recreating the maps by reinserting all frames over again. Also, ICP based method depend on which scheme is used to find neighbors. KinectFusion relies on projective data association which limits the approach to small incremental displacements and works well for specific kind of motions in general. Moreover, expressing a frame to frame depth geometric solution can be seen seen as performing a point-to-plane ICP with projective data association.

In SLAM application, the underlined map can take many forms in the literature : a set of landmarks, sparse feature cloud, successive keyframes stored in a graph, point clouds, volumetric voxel based volumes such as TSDF volumes or surfel volumes. Such maps can be characterized by the memory compression ratios, stored data, explicit or implicit information storage and so on. Most approaches relied on keyframe representation in the form of graph to perform incremental odometry estimate while the reconstruction of volumetric maps is performed in an offline batch step. Keyframes as such represent redundant information as the intersection between successive keyframes is most often non null. Storing keyframes indefinitely can quickly limit the space which can be mapped at once. The availably of a dense volumetric map is of essence for most of robotics applications which are required to interact with the environment or change it. Robots acting redundantly in common areas like work offices or homes can rely on a preprocessing mapping step then perform 3D navigation on previously acquired data. For the later case, offline batch reconstruction is enough to guarantee operation success. For robots which act in previously unknown environments, live dense reconstruction is necessary. [113] and [114] run an odometry estimation thread inside a UAV's processor while the live reconstruction is performed by a remote base station. KinectFusion derived approaches use TSDF representation to generate a dense representation of the workspace. The map in a TSDF form presents advantages over probabilistic occupancy maps counterparts since surface can be readily generated by looking for the zero crossing interface. Occupancy maps on the other hand have longer history for robotics applications since they represent a more natural way to express free, unknown and mapped space which in turn makes easier to use for planning applications. OctoMap [40] is an example of a widely spread occupancy grid solution for robotics applications. Octomap is particularly attractive since it models free, unknown and occupied space in a compressed octree based data structure. Regular grids provide the fastest way to partition dense volumetric maps but the memory requirements for mapping 3D environments with resolution down to few millimeters renders mapping volumes larger than a room infeasible in practice. Further solutions like Kintinuous proposed rolling volumes to shift a working buffer as the camera moves and store slices not visible anymore on the disk. If such approach added flexibility to KinectFusion, it still proves difficult to support revisiting of previously seen areas. Moreover RGB-D camera usually show an accuracy quadratically decreasing with the depth of the image and hence with increased covariance associated with further points [115]. Such far points are usually associated with grid cells at higher scales which is harder to model using regularly spaced grids. KinectFusion does not take such point into consideration which makes it more prone to error during large scale changes. Such operation increasingly adds on the map corruption. Pre-cited approaches like [105][100] use octree based structures to store the map and effectively associate further points from the camera center with cells at lower levels essentially modelling larger scales. Tree based approaches provide higher compression ratios at the expense of increasing time to insert and access points. More fundamentally, neighbors search on a tree can be computationally expensive which can limit the number of points usable in real-time. [100] typically uses thousands of points to achieve real-time processing while a common RGB-D sensor point cloud is made up of two order of magnitude more points. [105] allocates the tree structure on the GPU. In order to respect the memory contiguity constraint to maximize memory fetches on GPU devices, each tree node consists in a cubic volume called brick that subdivides into 256 cells which can yield a waste of memory allocated for non occupied space. OctoMap [40] uses an octree scheme with one global root. This tyically yields a tree with non negligible depth. Cloud insert time using raycasting at 5mm takes 25 seconds for 100000 rays which one again shows how tree based approaches are more challenging to use as a real-time solution for online live reconstruction.

2.6 Thesis Outline

Throughout this thesis, we derive a complete and practical framework to solve the full SLAM problem in 3D with focus on time and memory efficiency. Our framework specially targets robotics application and hence aims at providing a complete solution to build live maps and track robot position with high accuracy, which is the first essential brick of a fully autonomous robotic system that also runs subsequent recognition and planning algorithms. Our framework does not require the availability of high-end GPU to run and we only require the availability of point clouds (or depth map with known camera parameters such as RGB-D camera inputs). The issue of estimating depth for monocular camera through multiview stereo is not tackled and we refer the reader to appropriate literature like [84]. Given a point cloud at the input stage we use the sensor noise characteristic to compute the scale each point is to be mapped to. We also extract normal data and intensity gradient when possible. We then build a multi-level pyramid expressing our data at changing scales. We use a parallel mapping and tracking approach. Moreover, the pipeline assigns GPU and CPU resources to run in parallel. The tracking stage bases in its most fundamental aspect on a frame to model ICP tracking where the model consists of an up-to-date world map representation stored efficiently in an octree-like data structure. The ICP tracker can be augmented by frame to keyframe photometric tracker to account for planar areas which can lack geometric features but can be rich in textured regions. Doing so, the tracker ensures model tracking enhanced accuracy and robustness in scenarios where either geometric or photometric data becomes scarce. The mapping stage inserts the new point cloud data in the online map. The stage also checks for loop closures and runs graph optimization upon detection of a closing loop constraint. After optimization, the map is rebuilt. The online stage is appended by an offline stage where point decimation is performed. The online reconstructed model consist in an octree-like structure while the final model can be stored in various formats such as octrees or point clouds. The outline of the present work is as follows : in an introductory chapter we presented a taxonomy of SLAM problems while chapter 2 provided the most relevant references to our work and how they compare to each other. We provide an outline of our research in the end of chapter 2. Chapter 3 describes pre-processing steps needed at the input stage. These include RGB-D camera calibration, noise model computation, multiple sensors calibration as well as the point cloud online processing pipeline. This pipeline can include steps such as depth reconstruction, undistortion, filtering, color projection, normals and gradients estimation as well as pyramid building. Chapter 4 describes our map memory efficient data structure and the mapping component of the system. Chapter 5 derives the set of tracking formulas and algorithms and which break into model based ICP traking and direct optimization approaches. Chapter 6 introduces the architecture of the system as well as the back-end solution we adopted. It also presents a set of experiments we performed to evaluate the suitability of our approach. Every chapter is associated with a set of experiments and finishes with a conclusion. Comparison with other state-of-art approaches are given when appropriate.

第3章

Pre-processing

Sensory data can be provided through multiple means like IMUs, cameras, RGB-D cameras, lasers, GPS, wheel odometry and so on. Here, we separate between two classes of sensors, those which provide direct odometry and hence a guess or motion prior like GPS or IMU and those like lasers and cameras which directly provide a partial observation of the world and can be used to provide more accurate tracking results through more sophisticated algorithms. These two roles can be more easily understood through the following factorization which writes an incremental solution of the full SLAM problem. The objective of our system is to maximize the probability estimate of the joint trajectory and map which can be factorized :

$$p(x_{1:t}, m_{1:t}|z_{1:t}, u_{1:t-1}) = p(m_t|x_{1:t}, z_{1:t})p(x_{1:t}|z_{1:t}, m_{1:t-1}, u_{1:t-1}) \tag{3.1}$$

where u represents the system's odometry and the observations of the world are denoted z. m represents the map incrementally acquired while x is the mobile agent's trajectory. From this factorization, we can see how the full system can be solved by first tracking, then from the up-to-date tracked position along with the most recent sensor measurement the map can be expanded. The posterior over the trajectory $p(x_{1:t}|z_{1:t}, m_{1:t-1}, u_{1:t-1})$ can be further written :

$$p(x_{1:t}|z_{1:t}, m_{1:t-1}, u_{1:t-1}) \propto p(z_t|x_{1:t}, z_{1:t-1}, m_{1:t-1}, u_{t-1})p(x_{1:t}|z_{1:t-1}, m_{1:t-1}, u_{1:t-1}) \tag{3.2}$$

with :

$$p(x_{1:t}|z_{1:t-1}, m_{1:t-1}, u_{1:t-1}) \propto p(x_t|x_{1:t-1}, z_{1:t-1}, m_{1:t-1}, u_{t-1})p(x_{1:t-1}|z_{1:t-1}, m_{1:t-1}, u_{1:t-2}) \tag{3.3}$$

In an incremental scenario, we assume that most recent observations don't depend on older ones. This is known as the Markovian assumption. The Markovian assumption serves well the incremental SLAM purposes but can be easily violated when dynamic objects are introduced to the scene. Nevertheless it provides an important implication to further simplify the equations above and yields a better understanding of the process incurred. Bearing the Markovian assumption in mind, the whole solution can be hence expressed as :

$$p(x_{1:t}|z_{1:t}, m_{1:t-1}, u_{1:t-1}) \propto p(z_t|x_t)p(x_t|x_{t-1}, u_{t-1})p(x_{1:t-1}|z_{1:t-1}, m_{1:t-1}, u_{1:t-2}) \tag{3.4}$$

The equation above shows how the a new posterior on the trajectory and for the incremental SLAM is computed at each step by integrating a motion model along with the sensor likelihood $p(z_t|x_t)$. Approaches for exploiting such likelihood are explained in chapter 5. The motion model $p(x_t|x_{t-1}, u_{t-1})$ can be obtained from odometry measurements. In our case, a speed based motion model is used by default when no direct or only partial direct odometry is used (like in the case of IMU data which computes only a fraction of the state vector). A much simpler model to adopt is one which takes only white noise into consideration. Motion models if correctly derived allow to cope with motions at even higher speed. The equation also shows how direct odometry measurements and partial world observations interact to derive trajectory posteriors. In the following, we review two classes of prerequisites : offline calibration steps for estimating sensors noise characteristics, sensor intrinsic or extrinsic parameters estimation then online pre-processing in order to build denoised and suitable data for the tracker to use.

3.1 Generic Sensor Representation

Sensors have different characteristics, noise models and parameters. Still, the set of most common parameters across different sensor categories can be expressed in a generic enough way which is the approach we take hereafter. Each generic sensory vector input stores vertex $(v_x, v_y, v_z) \in \mathbb{R}^3$ coordinates, normal vector $(n_x, n_y, n_z) \in \mathbb{R}^3$, intensity gradient vectors $(g_x, g_y, g_z) \in \mathbb{R}^3$, RGB with intensity data $(r, g, b, i) \in \mathbb{N}^4$ then grayscale intensity and scale level $(gr, l) \in \mathbb{N}^2$. Each type of data is only allocated as needed, as different types of sensors can provide all or only a subset of the precited data. The sensory vectors are stored in a 2D map form in order to take advantage of sensor neighboring information if available such as in the case of an RB-D sensor or camera. In addition, each sensor has intrinsic parameters, extrinsics parameters which describe the sensor position in a global coordinates system then noise parameters which we limit to offset and scale noise parameters. Once again, data which is not relevant to a given sensor class is discarded and set to default values. Finally, the data stack is sampled at the different pyramids levels to enable speed-ups during the tracking stage. The default sensory information is acquired on the CPU side through usual links like USB or ethernet. The data is then sent on the GPU side where all preprocessing operations take stage. These operations are also implemented to be generic enough and include bilateral filtering, edge filtering, data truncating, denoising, utndisortion, vertex computation, normal data computation, RGB and intensity projection, grayscale intensity and scale computation, grayscale intensity gradient computation then finally pyramid extraction. Finally, data which holds only partial or poor information such as no RGB data or a noisy normal vector is set to non valid. These operations act on pixels or a pixel's neighborhood and therefore highly benefit from the parallelization power of GPUs. Once the sensory pipeline is finished the data is sent back to the CPU side to be fed to the tracking stage. Some sensors however need a separate pipeline since they don't fit in the generic sensor representation. This is notably the case for IMU or MARG sensors for which a separate data fusion step between magnetic data, accelerometer data and gyro data is needed.

As it has been pointed out before, direct odometry measurements can be provided by multiple sources such as velocity models, gyros, accelerometers or other forms. These different sources supply with partial or whole information about the trajectory increments and can sometimes be redundant as it is the case with accelerometers and gyros. This is why a fusion center shown in blue is necessary to compute one estimate from all these sensory inputs. The data fusion center runs two separate Kalman Filters, one for positional estimates and the second for angular estimates. When trajectory vector elements are not observed at the sensor stage a white noise is used as a prediction step while a speed model is used at the observation step of the filter. For the angular filter a quaternion based formulation is used. The complete derivation of the filters is straightforward and abundant in the literature. We refer the reader to appropriate references like [116].

3.2 World Observations

In this section we handle the case where sensory input is an observation of the world. This include the case of 2D laser sensors, RGB cameras and so on. Raw data as such provides vertex data. The neighboring relationship between successive points can also yield precious information about the underlying surface. Then, color information if available can be crossed with vertex data to provide with colored point clouds. Such data is preprocessed and fed directly to the tracker in order to increment accurate updates of the positions given a guess provided by direct odometry. The whole pipeline is described hereafter using the Microsoft Kinect 2 sensor as an example of input sensory data.

3.2.1 Calibration

The Offline calibration step is a prerequisite to any computing involving cameras. The aim of this step is to compute intrinsic parameters corresponding to focal lengths and center position but also, for RGB-D cameras for instance, the extrinsic parameters which allow to project vertex data extracted from the depth data using intrinsic parameters onto the RGB camera frame in order to associate each vertex with its corresponding RGB data. In the case of the Microsoft Kinect2 sensor we also include an intensity level which associates the underlying surface vertex with an appropriate reflectivity value. For multiple sensors scenario, the relationship between each of the sensors involved in the sensing loop also needs to be recovered. When multiple sensors look at the same area these extrinsic parameters can be extracted by matching common parts of the image data accross the different sensor frames. This is the case in a classic stereo camera pair setup. However, his relationship can be harder to recover when sensors point at divergent parts of the world. For the case of cameras and single RGB-D cameras, we base our calibration process on a classic implementation of chess board based calibration as widely used in the computer vision community. Figures 3.2.1 and 3.2.1 show respectively a chess pattern as seen from the RGB camera and infrared camera. Multiple samples at different viewing angles allow to compute a best fit for the intrinsic parameters. For RGB-D cameras, we validate a chess pattern only when it is simultaneously viewed by both RGB and infrared camera. Doing so, a best fit for the stereo transform between the two camera frames can be estimated. This is shown in the figure below :

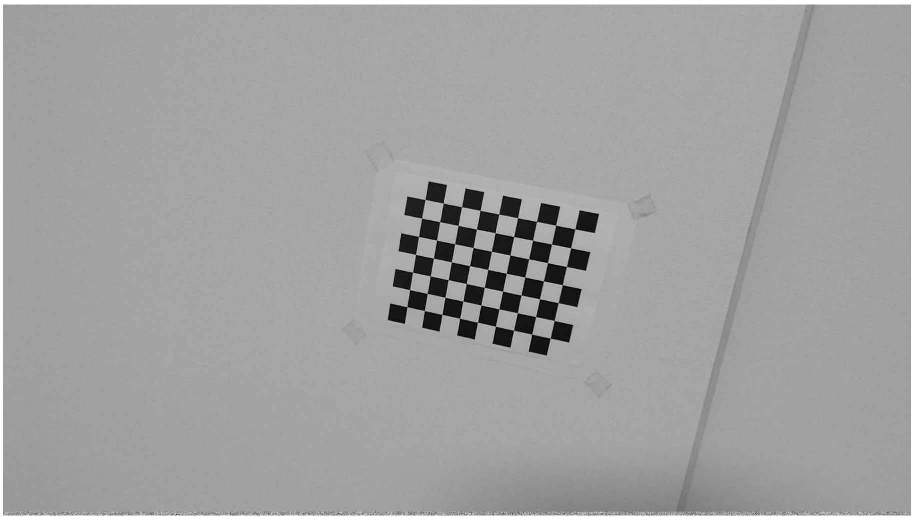


図 3.1: Chess Pattern as viewed from the RGB camera

Note that the Miscrosoft Kinect 2 at an initialization step streams factory calibrated intrinsics which are usually good enough to work directly with. Once this step has been completed, each sensor is ready to use but the multi-sensors setup requires a step further. If the setup is convergent a classic cross frame matching is performed and the transform between each sensor pair can easily be recovered. The divergent setup is however slightly more difficult to deal with. Since sensors don't point at the same region of the world, immediate cross matching cannot be performed. In order to recover the extrinsics between divergent sensors we tested

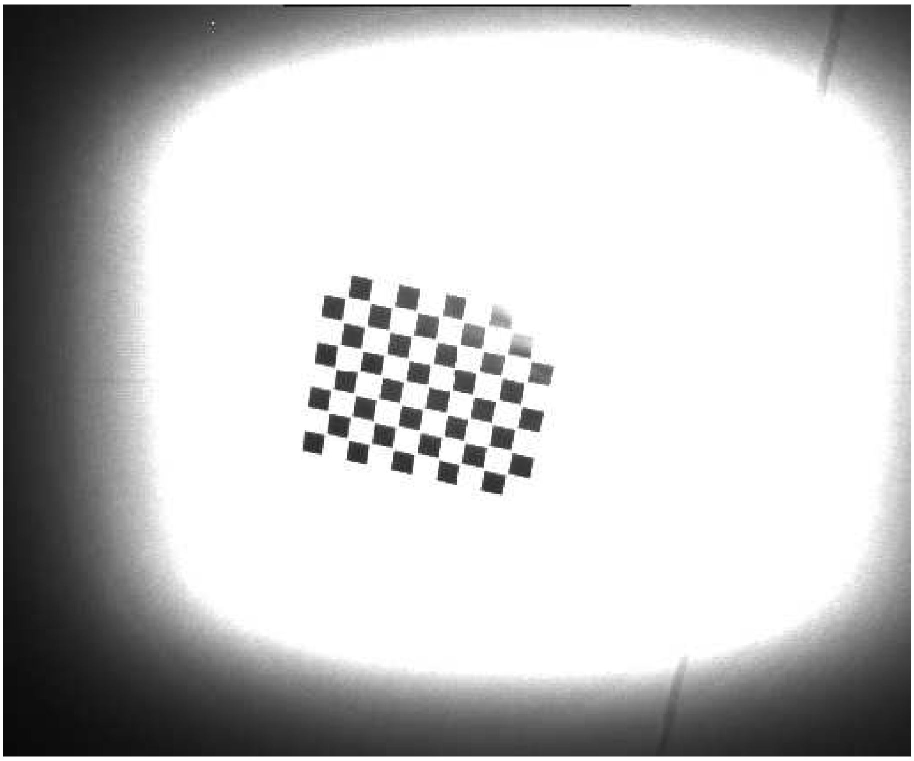


図 3.2: Chess Pattern as viewed from the IR camera

two procedures which are shown in the figures below. In the first one the second camera is static and starts by initializing a static frame. The frame is temporally filtered. The first camera on the hand initializes then builds in an incremental slam fashion a frame graph until an overlapping region with the second camera's frame is found. The graph is made of multiple successive frames and associated transforms. Once the overlapping region is detected extrinsics can be computed in a straightforward fashion. The alignment can be performed in a geometrical fashion using ICP or by using direct frame to frame optimization using color data or both depending on which data is available in the input stage. As such the procedure does not need particular environments or setups to complete but any environment should provide reasonable estimates. Needless to say that environments at smaller scales will provide with best accuracies. Note that larger point clouds can accumulate larger drift values which in return can affect the accuracy of the extrinsic parameters to recover.

The experiment is repeated and results appropriately averaged. The procedure mentioned above consist in a minimization between one frame from one sensor and a open graph made up of multiple successive frames. Since the graph is open, as it expands through time it adds on drift which hence affects the quality of extrinsics estimate. In a second procedure we tried to limit the impact of drift by constructing a closed graph. The graph consist of two branches. The first one consist of an open graph from the first sensor to the second while the second branch consist of the open graph from the second sensor to the first one. The cross frame transforms are computed through ICP or direct optimization. Once the graph is completed a loop closure constrain is inserted and the whole graph optimization takes place. Note that the inter frame transform covariance is properly scaled to a factor to account for sensors with different precisions.

3.3 Sensor Model

Multiple factors can corrupt the data provided by sensors. Understanding the impact of noise and sensory errors on the quality of sensory output and hence on the reconstruction algorithms in general is of primary importance and has been an important issue in the literature. We give below a brief description of the most common of such errors and their source :

- Maximum range : Sensor have a range where they behave almost linearly. Out of that range the maximum or minimum output voltage is returned.
- Nonlinearity : The sensitivity to the changes in a measurable variable is supposed to be constant, hence describing a linear relation. In practice, this is never the case.
- Null-shift error : describes the constant bias error when the sensor is static.
- Misalignements errors : These come from two sources. The first is the small additive angles that exist between sensory components. The second comes from misalignement of the whole sensor with the containing board.

- Cross-Axis sensitivity : Describes how much of ouput is seen on an axis given an input on a diferent axis.
- Quantization errors.
- Technology related noise such as photon noise in a CCD camera and so on. Multiple sources with different noise distribution and levels can add up and which are usually represented by a gaussian noise.

Moreover, some of the sensor properties can change through time or with temperature like sensitivity or zero-bias. In a robust system, this change should also be taken into consideration to derive a full sensor model. Additionally, the underlying technology in use can bring additional effects such as rolling shutter distortion on CMOS sensors, optical distortions, distance ambiguity on phase shift based time-of-flight sensors and so on while fast motions can provide with even noisier signals. The quality of a sensor data is directly impacted by the degree of noise inside the sensor itself but can can also be affected by the scene viewed. For instance, for an IMU higher accelerations can corrupt more strongly the angular estimates while surfaces with low diffuse reflection or high specular reflection can result in a poor laser data and distortion in comparison with the real world geometry.

Figure 3.3 shows an example of data acquired with a kinect 2 sensor. The environment consists of a white lambertian like wall along with a black smooth monitor screen. The apparent noise levels from the figure is clearly more important for the screen points. Finally another limitation comes from the quantization resulting as sensors use a finite number of sensory elements to sense the world geometry or photometric properties. This can consist in a fixed image resolution for a camera or a finite number of dots on the IR pattern used by the Microsoft Kinect Sensor and so on. As the depth of the scene increases the expected results get quantized to closed achievable sensory resolution. Finally, we make an important distinction between the precision and the accuracy of a sensor. The precision describes the variability of the sensory result and is directly impacted by the random noise inside the sensor while accuracy is how does the sensory data compare to the ground truth which

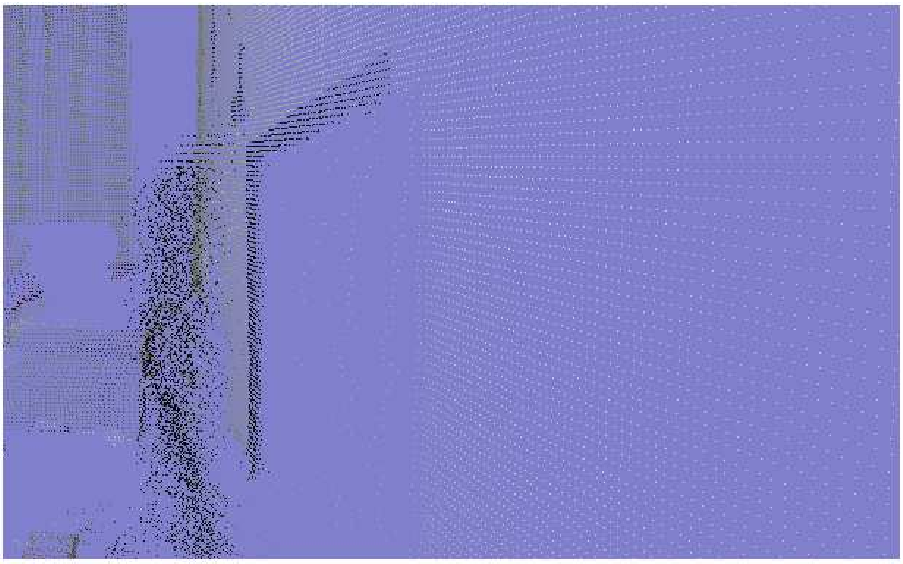


図 3.3: Point cloud captured with a Kinect 2 sensor

is affected by systemic errors like poor intrinsic or parameter calibration or can be impacted by the environment itself. The precision of a sensor is usually characterized by a distribution. It is important to note that measured data and output data can be different values. For instance, a Microsoft Kinect measures disparity while we are interested in the accuracy and the precision of the output 3D point cloud. From the disparity measure an estimate of the depth Z is computed then using a pinhole camera projection model the values of X and Y coordinates are extracted. Even if a thorough study of input sensor noise model is performed, extrapolating the result on the output value, which most of the case is linked to the input with a complex non linear relationship, is difficult and the most simple assumption which uses a linear model on the input data and a linear approximation of the input-output transformation does most of the time not hold in real. Moreover since each sensor is fundamentally different in terms of technology and input-output relationship, various works have tried to derive a model for each sensor in particular like stero cameras [117], Microsoft Kinect [115] or Swiss Ranger [118]. The Kinect2 sensor works with the time-of-flight principle. It send a signal and receives it back with a phase shift.

This phase shift is linearly related to the distance to the obstacle. In order to make one frame (one 512*424 frame) Kinect2 captures 10 different 512*424 images : 3 different frequencies with 3 different initial phases and one with the projector off. These 10 frames are sent through USB 3.0. The phase shift relates to the computed depth through an approximately linear relationship but in order to derive the phase shift the input image signals are combined in a non linear relationship fashion.

A precise modeling of the entire sources of noise is usually a hard problem. For the present work we write a simple linear model which sums up the most important error contributions commonly met within most of sensors. We write our model directly on the output. It takes into account the zero-bias, the scale then a normally distributed noise. Even such a simple model where the fixed parameters are correctly estimated and the precision correctly quantized can largely contribute to the accuracy of each sensor and to the accuracy of the system as a whole. In our system, the sensor model first contributes to derive results as close as possible to the ground truth but also selects the appropriate scale to write the map and which accounts for the precision at each point data. Conversely, when such model in neglected, both trajectory, accuracy and memory can largely be impacted as more map cells are needed to store a noisy signal. Also, assuming a linear model with normally distributed noise, section 4 describes how new data are fused throughout time inside a single cell effectively resulting in a smooth signal. Kinect2 works according to time of flight principle. It send a signal and receives it back with a phase shift. This phase shift is linearly related to the distance to the obstacle. The signal input s_0, s_1, s_2 is related to the phase through the relation :

$$\phi_i = arctan(\frac{f(s_0, s_1, s_2)}{g(s_0, s_1, s_2)}) \tag{3.5}$$

where f and g are linear. Furthermore, the corrected depth is recovered through an approximately linear relation. Supposing that the input signal variance is known the impact on the output signal V_o can be recovered trough the relation :

$$\sigma_o^2 = \frac{\partial V_o}{\partial V_i}^2 \sigma_i^2 \tag{3.6}$$

Thus, if the input signal is assumed to be normally distributed, the input-output relationship is usually non linear and the normal distribution for the ouput does not hold anymore. However since the variance of the true underlying distribution is finite, the filtered output signal converges to a normal distribution. We hence write the output sensory value as:

$$V_o = V_0 + K * (V_i) + w(D, t) \tag{3.7}$$

where the term $w(D, t)$ is an overestimating normally distributed noise which denotes the variability of the sensor and V_i is the non corrected sensory output. For most of sensors in presence, we hence proceed to comparing the sensor data to ground truth values to derive the null shift and the scale factor after what the variance of the underlying normally distributed noise assumption is estimated. For the Kinect2 sensor, we record the values at center and extremes of the sensor facing planar surfaces ranging from white rough planes to dark smooth surfaces. The figure below represents the results for a white rough wall surface which we can assume close enough to the lambertian case. A hundred sample data has been taken for the center pixel and for the corner pixel of the kinect2 sensor and the variability of the signal has been recorded at different to trace the evolution to the system noise at different distance values from the target.

Figure 3.4 shows four graph of the center pixel and corner pixel of the image data provided by the Kinect 2 sensor with and without mean filtering. We notice that corner pixels induce noisier outputs compared to the center pixels. The position to the plane is of another impacting factor as the emitted signal strength falls quadratically with the distance to the object and hence is more prone to noise at longer ranges. We truncated the data to 5m as a maximum measured range. The standard deviation varied from few millimeters to few centimeters for longer distances to the plane and for corner pixels. We also noticed that smooth object induces on

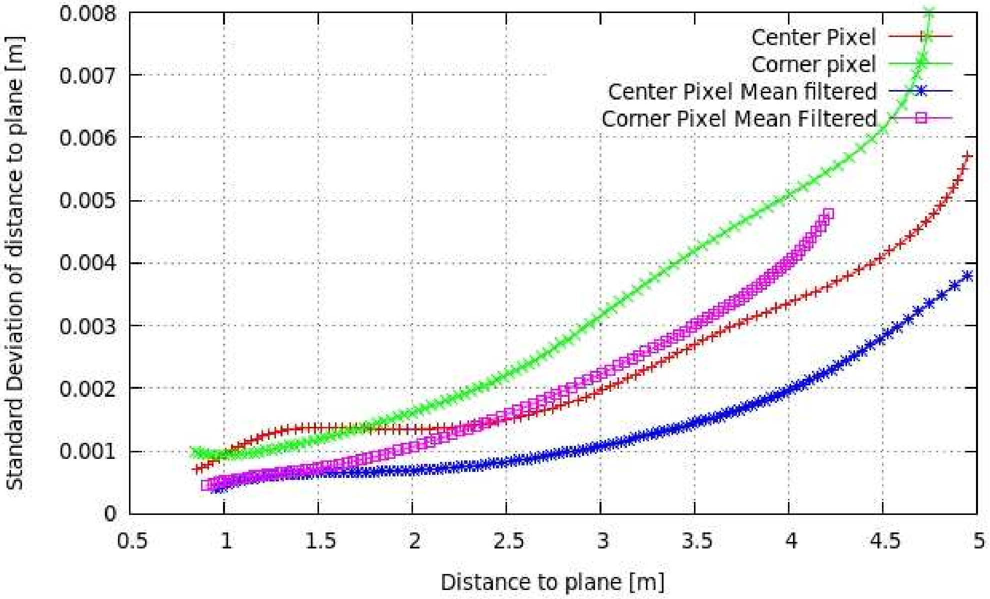


図 3.4: Standard deviation of the distance to plane

average twice as much noisy signals while the curves obtained were acquired with a material which has close diffuse properties to lambertian surfaces. From these observations, when using the Kinect2, sensor ranges under 2 meters are mapped at the highest possible resolution while ranges between 2 and 4 meters fall under the next resolution. Ranges beyond 4m are conservatively mapped at even lower resolution. The graph also highlights how proper filtering can dramatically increase the precision of the sensor but in practice, and especially since the present system is to be implemented on robotic platforms, fast motions prevent from using temporal filtering hence the conservative values we took to assign the resolution of each point in the cloud based on the range distance measured.

3.3.1 Input data

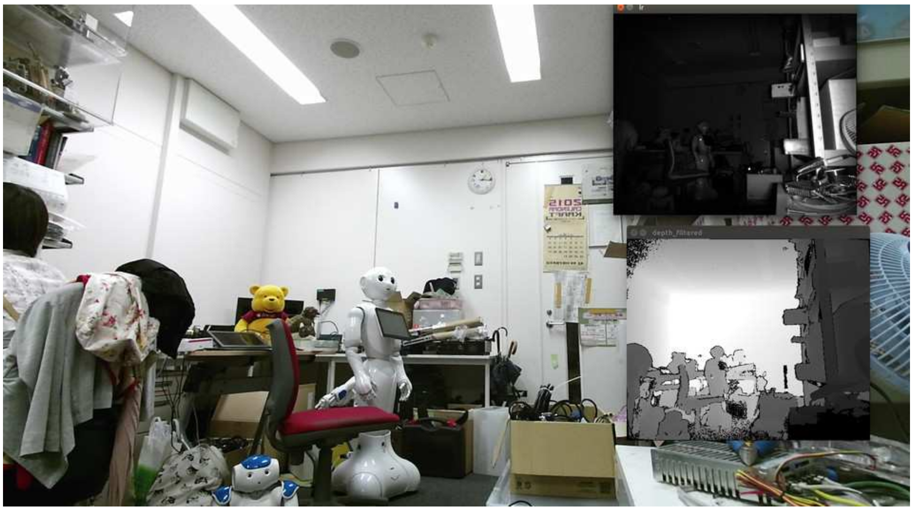


図 3.5: Input Filtered data from an RGB-D sensor

Figure **??** shows an example of data extracted after bilateral filtering from the Kinect2 sensor. Kinect2 works according to time of flight principle. It sends a signal and receives it back with a phase shift. This phase shift is linearly related to the distance to the obstacle. In order to make one frame (one 512*424 frame) Kinect2 captures 10 different 512*424 images : 3 different frequencies with 3 different initial phases and one with the projector off. These 10 frames are sent through USB 3.0 and transformed into meaningful IR and depth data as shown in the Figure. Kinect2 Also sends a higher resolution JPEG compress RGB data captured from an onboard RGB camera which is uncompressed and from which point cloud coloration is performed. The raw data directly extracted from the Kinect 2 sensor is shown in figure 3.3.1

As it often the case with time-of-flight based sensors, edge regions are considered high noise regions which brings the need to implement an appropriate edge filter in addition to a bilateral filter to fill the wholes.

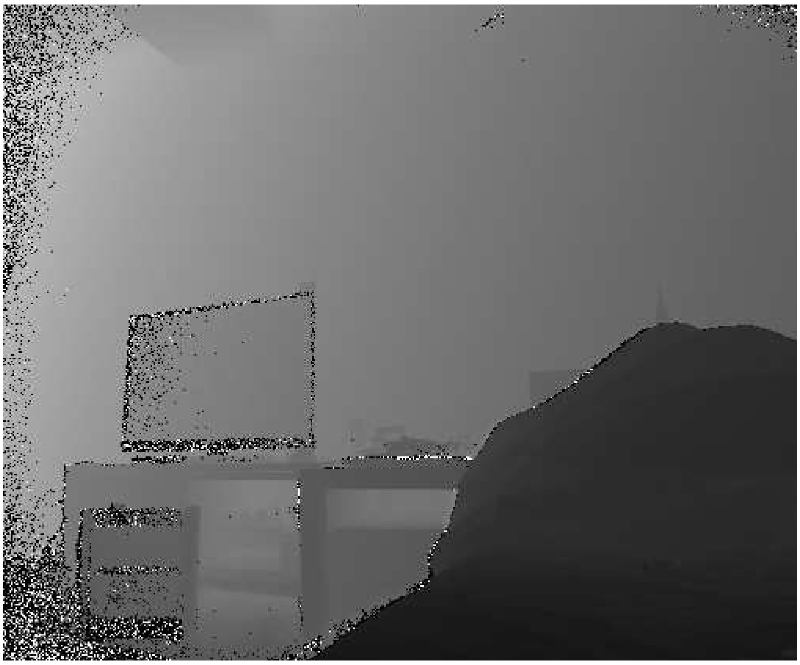


図 3.6: Raw input from Kinect 2 sensor

3.3.2 Filtering data

Bilateral filtering is one of the most popular methods to reduce noise in input. The operation performed on raw depth data takes the simple form :

$$I(\mathbf{u}) = \frac{\sum_{i \in N} w_i I_i}{\sum_{i \in N} w_i} \tag{3.8}$$

Such that :

$$w_i = exp(\frac{-(I(\mathbf{u}) - I(\mathbf{u_i}))}{\sigma_I^2})exp(\frac{-||\mathbf{u} - \mathbf{u_i}||}{\sigma_U^2}) \tag{3.9}$$

where the variance of the space and intensity gaussian kernels are empirically chosen. The result of applying bilateral filtering to the input is shown in figure 3.3.2

Next step consists in removing the saturated pixels and reducing the noise around the edges through edge filtering and truncating to min and maximum values which ensures limited and noise attenuated depth data. The influence of edge filter only is shown in figure 3.3.2

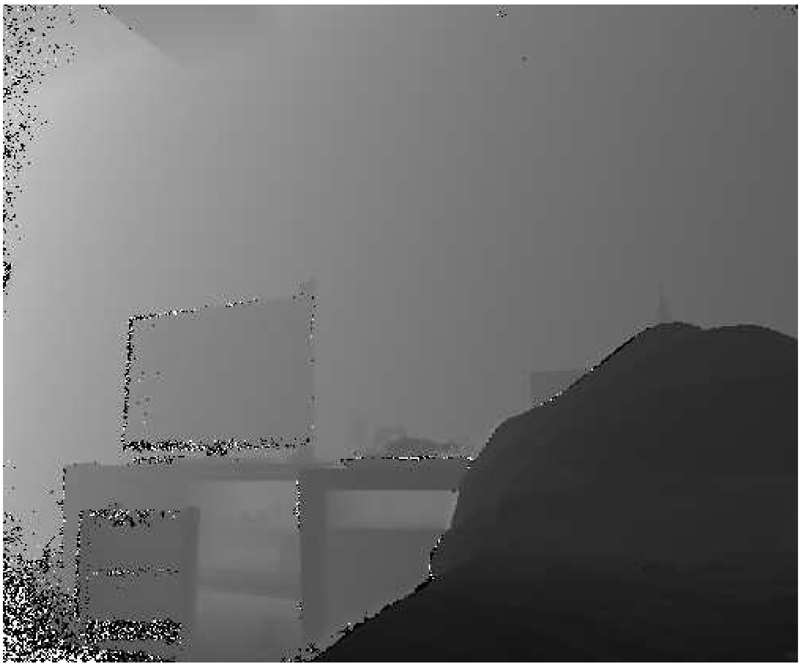


図 3.7: Bilateral filtered raw data

The filters discussed here are spatial filters as they operate on a neighborhood to attenuate the noise in the current frame. Previous section showed how a temporal filter can reduce dramatically the noise. To illustrate the importance of a statistical filter like a mean filter figure 3.3.2 shows the discrepancy from the mean computed on the environment shown in figure 3.3.2. Darker colors in figure 3.3.2 denote higher discrepancy values. We can notice that corner pixels or pixels with far range show higher noise values as expected. Hence for mapping applications, for instance where a map of has to be created and saved for robot to use during later operation as a base for localization, a mean filter is implemented and allows to build more accurate and noise free maps. Such operation however requires the operator to map in a stop and go fashion as movement during the filtering process can induce errors which hinder the objectives we first pursued to justify the use of a temporal filter. For online SLAM this step is skipped. Another process to reduce the impact of noise over time is described in chapter 4 of the present work.

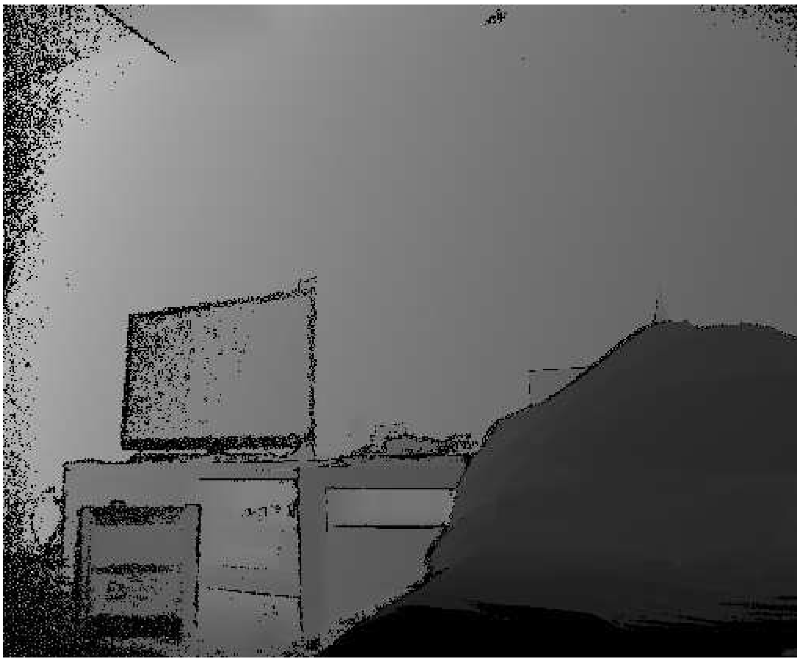


図 3.8: Edge filtered raw data

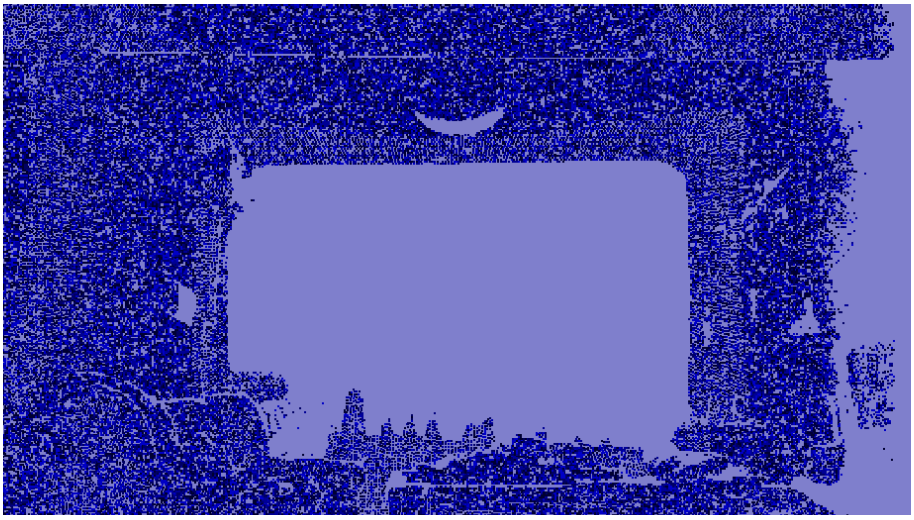


図 3.9: Mean Filter applied to test environment

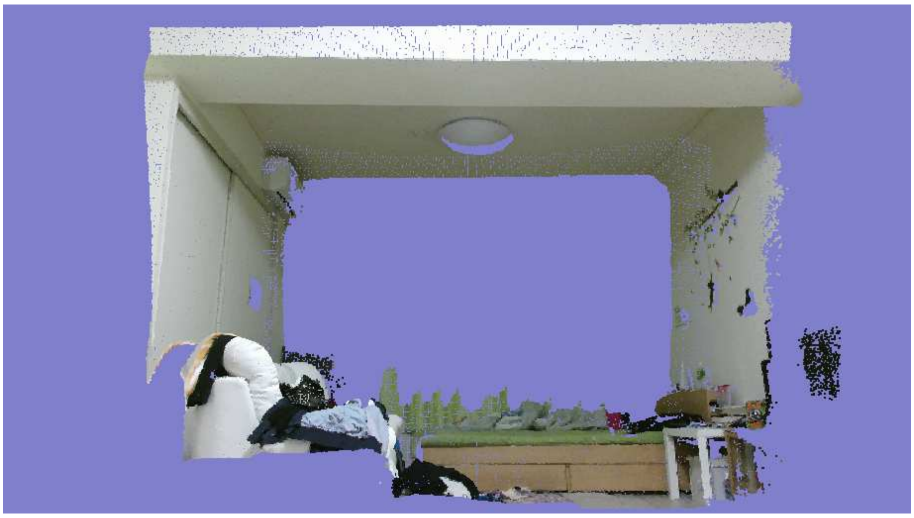


図 3.10: Test environment for mean filter

3.3.3 Vertex Extraction

Depth data along with calibrated intrinsic camera parameters allows to reconstruct a point cloud. For that, given a point $\mathbf{u}$ associated with depth z, the vertex point $\mathbf{p} \in \mathbb{R}^3$ associated with the pixel is defined by π^{-1} :

$$\pi^{-1}(\mathbf{u}) = \mathbf{p} = (\frac{z * (u - cx)}{fx}, \frac{z * (v - cy)}{fy}, z) \tag{3.10}$$

The point cloud which results from such operation is shown in figure 3.3.3

3.3.4 Normals Extraction

Multiple methods have been proposed in the literature to compute normals from a point cloud among which computing normals directly from depth as the cross vector $\mathbf{p}(u+1, v)X\mathbf{p}(u, v+1)$, using Principal Component Analysis on the covariance $\sum_{i \in N}(\mathbf{p}(\mathbf{u}) - \mathbf{p}(\mathbf{u}_i))(\mathbf{p}(\mathbf{u}) - \mathbf{p}(\mathbf{u}_i))^T$, integral images or difference of normals. In the present work we use the PCA based method. The normal is associate to the eigenvector corresponding to the smallest eigenvalue α which is extracted from the

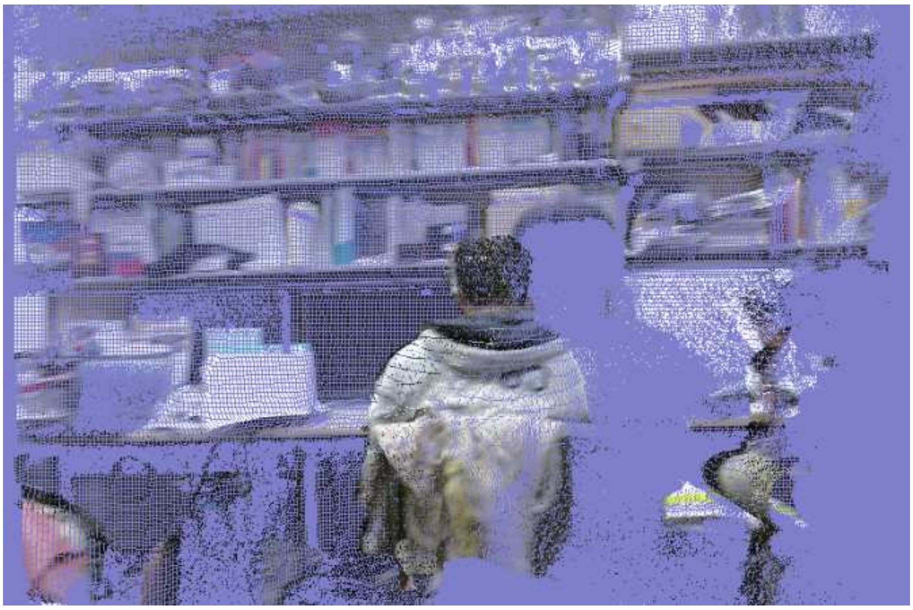


図 3.11: Colored Point Cloud

covariance matrix A. The eigenvector lies on the perpendicular to the image $A-\alpha I$. An example of the estimated normals are shown in figure 3.3.4.

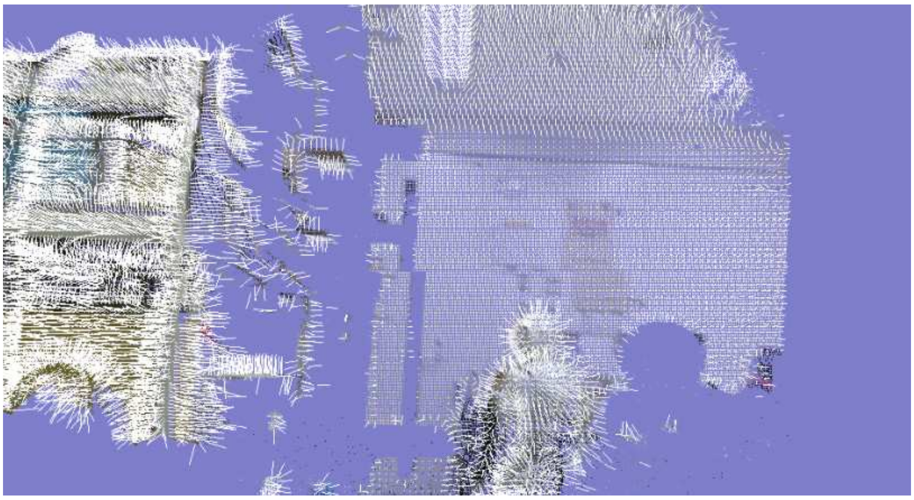


図 3.12: Normal Estimation

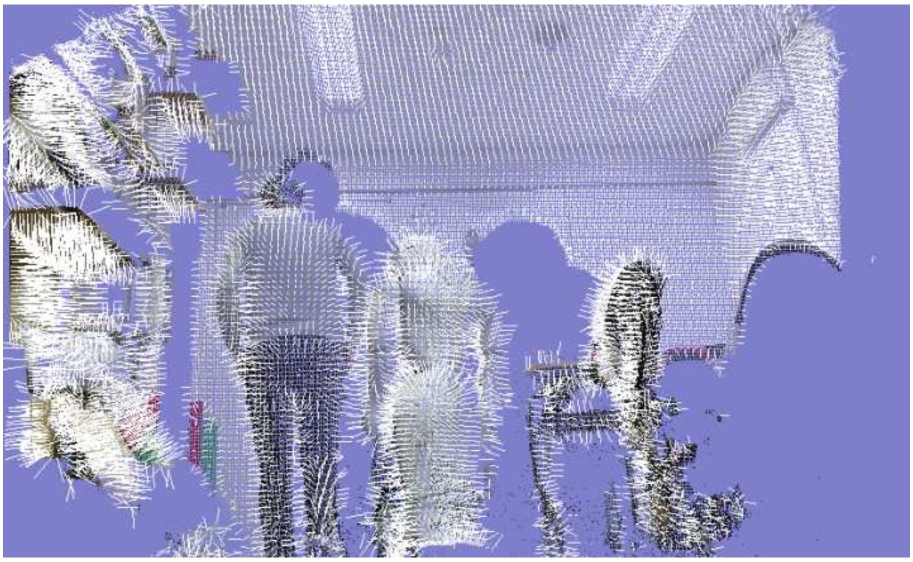


図 3.13: Normal Estimation

3.3.5 RGB Projection

First let's define the point $\mathbf{p} = (x, y, z) \in \mathbb{R}^3$ and let's denote π the function which projects the 3D point $\mathbf{p}$ with its associated pixel $\mathbf{u}$. For camera model following classic convention (x axis to the right and z pointing out of the camera) with a local center of reference at the optical center (cx, cy) and with focals (fx, fy) we have :

$$\pi(\mathbf{p}) = \mathbf{u} = (\frac{fx * x}{z} + cx, \frac{fy * y}{z} + cy) \tag{3.11}$$

The point $\mathbf{p}$ is first transformed to $\mathbf{p}'$ in the local RGB camera frame centred on the RGB camera center. It is then projected on the RGB camera image in order to assign the corresponding RGB triple using $\pi(\mathbf{p}')$. The result of RGB coloration is shown is figure 3.3.5.

3.3.6 Grayscale and Scale

Each vertex in the input point cloud is assigned an RGB value and normal data. Moreover, a grayscale value is computed. This grayscale value will be central in

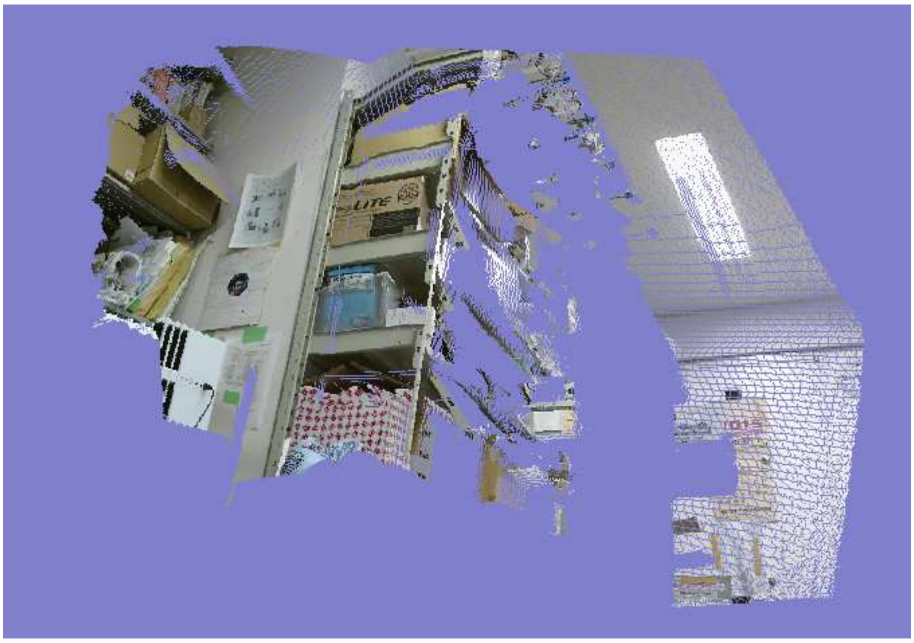


図 3.14: Colored Point Cloud

the photometric tracking stage later described. A sensor model precomputed at an offline stage maps a point $\mathbf{p}$ to a corresponding level l. In our model lower levels are assigned to less accurate vertices. This is important to ensure that noisier data does not corrupt data acquired at closer ranges which usually provides superior accuracy as it has been demonstrated in a previous section.

3.3.7 Intensity Gradients

Normals data is central to geometrical tracking introduced in a later chapter. For photometric tracking a prerequisite is to compute the intensity gradient which essentially provide direction and intensity of photometric data variation. These are computed using a classic Sobel filter. An example is provided in figure 3.3.7.

図 3.15: Intensity Gradients

3.3.8 Pyramid of Data

Finally the last step is to compute sensory data at lower scales. These lower scale version are central to making the tracker able to cope with larger and faster motions. A second important role of the pyramid is such that data at lower accuracy level is compared against pyramid data at lower scales. A pyramid reconstruction of our sensory stage is shown in the figures below. A pyramid example is shown for three different levels and for multiple components of our sensory data.

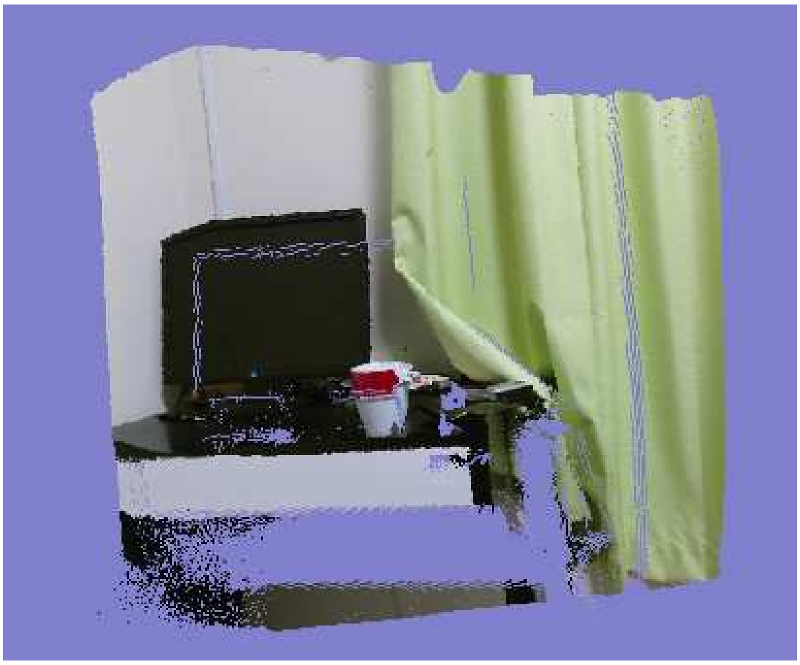


図 3.16: Point Cloud at level 0

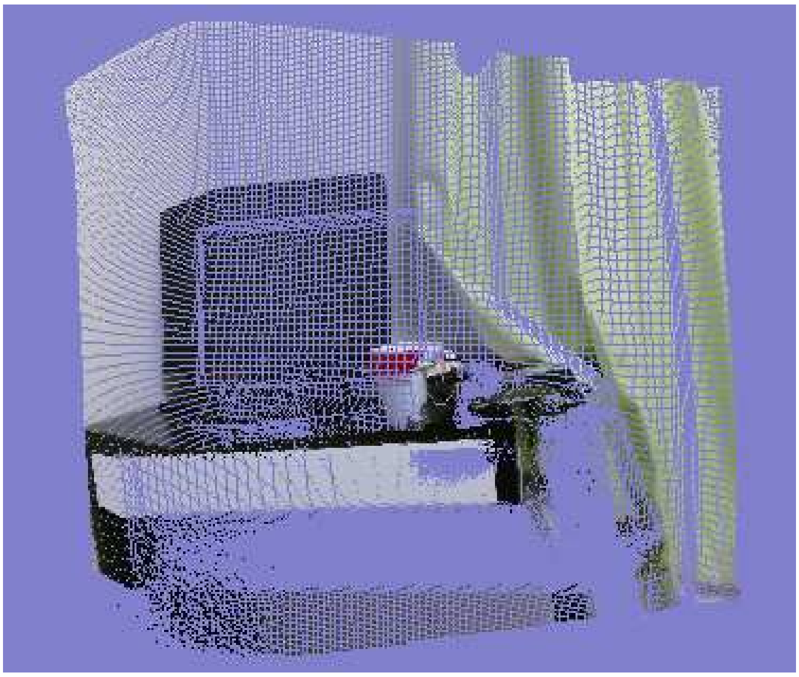


図 3.17: Point Cloud at level 1

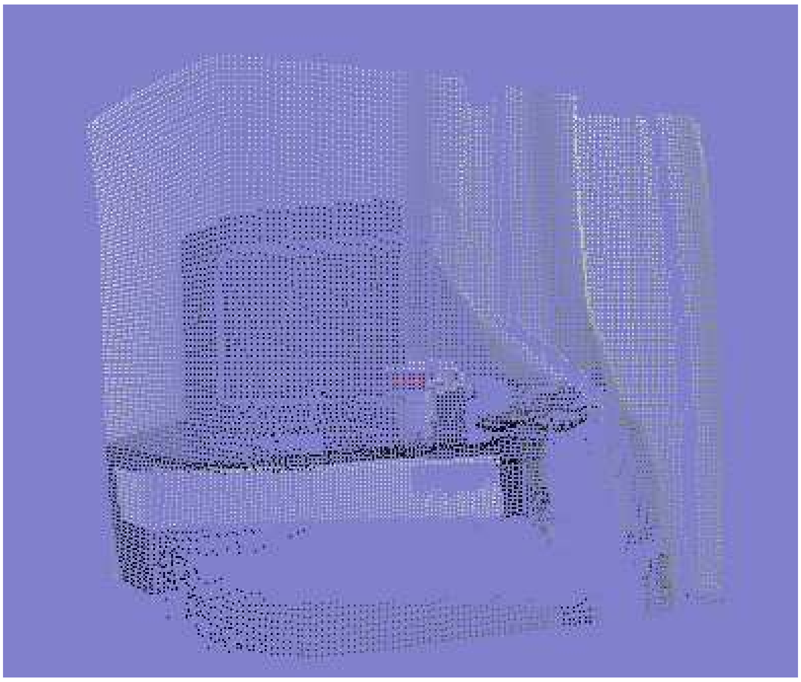


図 3.18: Point Cloud at level 2

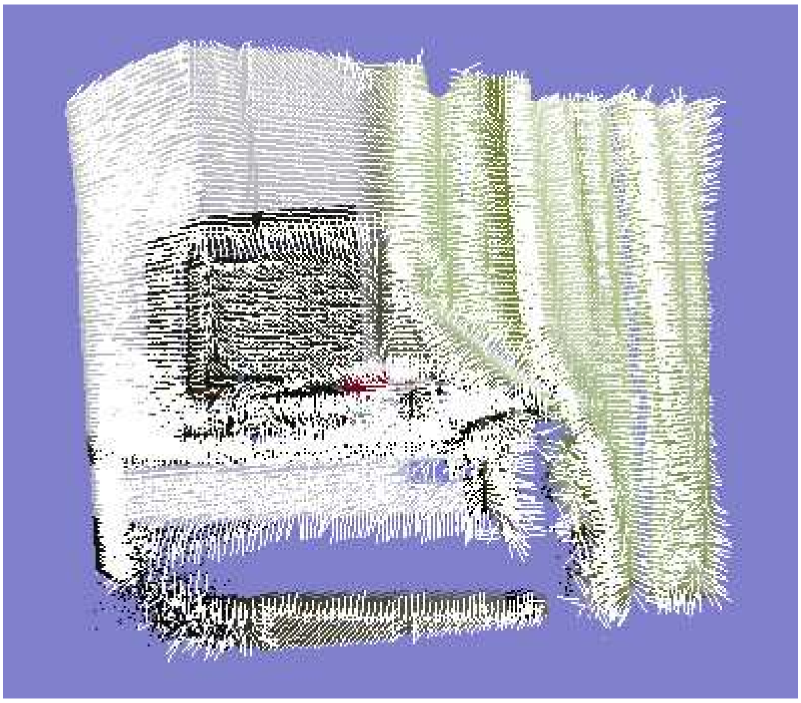


図 3.19: Point Cloud Normals at level 0

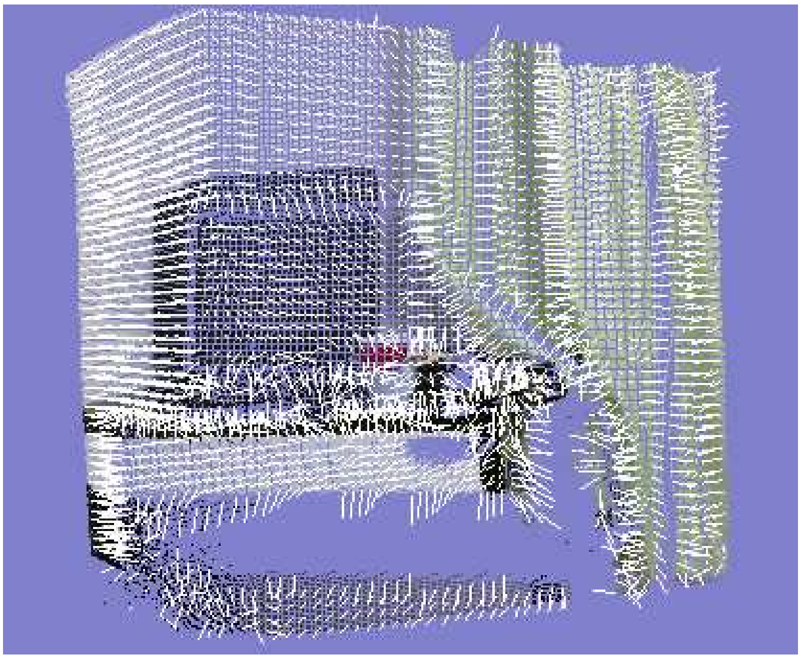


図 3.20: Point Cloud Normals at level 1

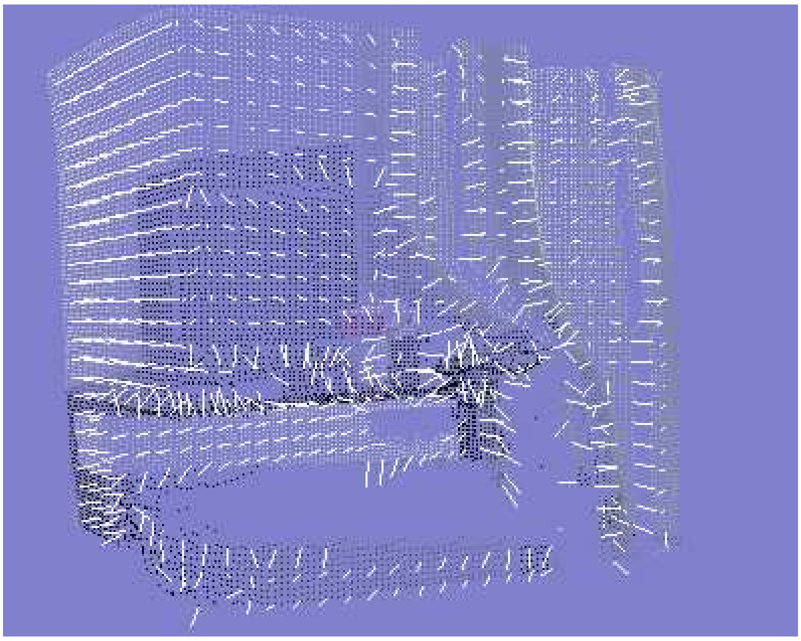


図 3.21: Point Cloud Normals at level 2

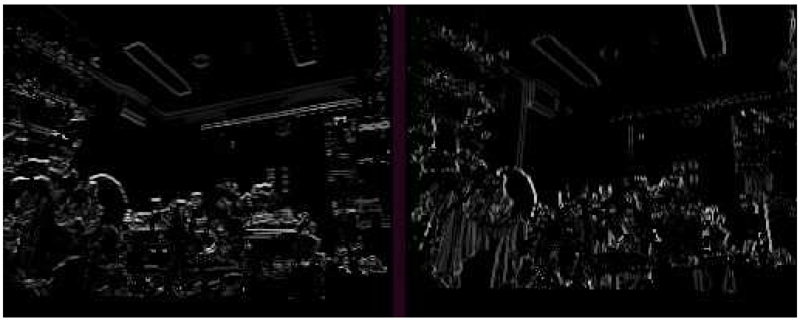


図 3.22: Intesity Gradients at level 1

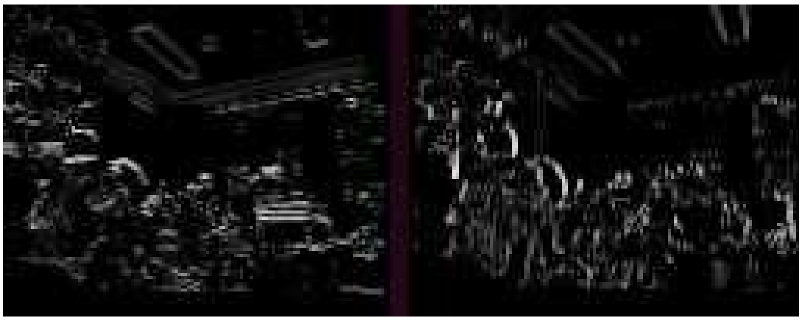


図 3.23: Intensity Gradients at level 2

Such pyramid data is built for the whole pipeline. The pyramid form of data along with the direct odometry based guess forms the output of the pre-processing stage and hence the input for the tracking and mapping stages.

3.4 Conclusion

The present chapter has discussed all preprocessing operations needed to form a denoised data as input for the tracking and mapping stage. The discussion has considered all aspects of the pipeline focusing on the role of odometry and observation in the sensing pipeline. A model of sensory input and a simple linear model of sensory errors has also been proposed and a precision test for the case of the Kinect2 sensor has been acquired. Such model allows to assign each point in the cloud to an appropriate scale to preserve poor estimates from corrupting more accurate ones. Then, the different stages of the sensor preprocessing pipeline have been explained and the importance of proper filtering has been stressed. Next chapter will describe our map representation, map stepping and map fusion algorithms.

第4章

Multi-Resolution 3D volumetric Map

The ability of creating 3D maps of surrounding environments has been one of the most attractive research areas in recent years with application spanning a large scoop such as augmented reality, video gaming, architecture, navigable space for mobile robots or for the visually impaired. For large environments, map creation is usually paired with solving a tracking problem to extract the sensors position first, then holding such position information, create a mapping of the surroundings. In the robotics community such combination is known as SLAM as it has been described in the first chapters of the present work. In the previous section, we established how the input to the mapping stage is a pyramid of data comprising vertex, normals, color, scale and gradient attributes. Our goal is, given successive and overlapping input pyramid data, how to fuse all estimates into one consistent map. As it has been pioneered in the work by Klein and Murray [68], mapping can be run parallel to the tracking step at lower frequencies. Moreover as it widespread in the keyframe based approaches, mapping cleverly chosen sparser data and tracking against such data can induce less drift than mapping every frame coming along. This is especially the case when the mobile robot is idle. Thus, the mapping process runs at an order of magnitude lower frequency than its tracking counterpart. However, the longest running path of our system comprises tracking and mapping in series even though most of our system will be performing tracking alone. The longest running loop of our system determines its reactivity and hence mapping at high speed is a primary objective we pursue in this section. Another central issue about mapping, especially for the robotic field, is whether a volumetric map is generated or not. A volumetric map allows to create a 3D map representation where the distance between any two points in the map is readily extractable or can be identified as unknown to the system yet. This contrasts with the pure keyframe representations like [104] where the map consist in a succession of interconnected graph nodes storing the input sensory data. Obstacle avoidance on such map representation can be more cumbersome and dynamic obstacles can be handled poorly or not at all which limits its use for live navigation of mobile agents. Also, note that nodes in a graph can store redundant information which impacts both the memory consumption and needs for a process to determine how to retrieve information from multiple redundant sources. A more memory efficient representation is to maintain a global volumetric map model where old information is fused with new one which guarantees no redundancy and allow to readily extract the necessary information from the map without the need of any disambiguation process. Another advantage of volumetric maps is the ability to extract parts of the map to visualize or the ability to traverse the world which is necessary for planning applications. However, even if model maps are more memory efficient, on the other hand they can require important computational resources to insert new point clouds which become even more of a problem when the map model needs to be reconstructed in a live scenario. Volumetric maps can be created in real-time such as [13] or offline like [96][97]. An advantage of real-time availability is to be able to use model to frame tracking based algorithms which often show higher accuracy and are less prone to drift. Also, the ability to inspect or plan on the fly using the reconstructed map is another prerequisite to achieve autonomous navigation. Volumetric maps require to specify two fundamental aspects : the underlying map representation i.e which data is going to be stored in the map then which data structure is to use in order to achieve a good trade-off between speed and memory management. With respect to the second aspect of a map description, tree based structures can provide good map compression but yield increased latency. Trees can also be used to represent efficiently maps at different scales. Plain arrays have also been used but they require important memory resources at high resolution mappings which constraint most of them to small workspaces. Plain arrays can make data available for access at optimal speed. This is especially the case for the original KinectFusion approach [13]. A solution to the workspace limitation shortcoming has been proposed by [108] who has used rolling volumes which store back to drive parts of the buffer which lie far from the sensor view. A shortcoming of this approach is that revisiting previously mapped areas becomes more difficult to implement and the global field of view of the agent has not been larger in essence. This makes it harder to implement on a real robot which can show pseudo random movements and trajectories in the space. A popular example of trees is the Octomap [40] framework which stores a multi-scale volumetric 3D map in the form of a voxel octree. The whole tree has a single root which breaks into 8 children until reaching the minimum voxel size allowed. Octomap discriminates free occupied and unknown space. Such distinction is very precious for planning applications. Octomap's unique root model makes it harder to use efficient parallelization on tree operations. Moreover, Octomap acts relatively slowly and hence can be challenged when mapping hundred thousands rays at sensor update speed.

The underlying representation stored in the map can be either implicit like Signed Distance Function (SDF) or Truncated Signed Distance Function (TSDF) volumes [41] or explicit like occupancy grids [39] and surfel maps [95]. Most SDF based maps applied to SLAM problems store the distance to the surface along with a weight factor. The weight factor updates as data is redundantly fed to the same voxel cell. The SDF stores minimum data and hence SDF based approaches have generally a smaller footprint. In SDF volumes, data cannot be readily read from voxels but has to be extracted by averaging neighboring cell information. This includes surface position and orientation. Such operation can account for important latencies especially if the data structure associated with the SDF module has high latency stepping such as trees. Surfel maps on the other hand store necessary surface data at each voxel in an explicit fashion. Vertex position, normals, RGB and probability of occupancy all have to be stored in an explicit way. Doing so, they have higher memory requirements. On the bright side, data is readily available and does not need interpolation to be usable.

In this chapter we describe our main map structure which holds the key to online reconstruction of large volumes without any workspace limitation and with still conserving real-time update and access attributes. We thoroughly test the computational capabilities and the memory cost associated with our map structure. The tests have been conducted on a million data at for high resolution down to 5mm. They all showed how our approach allows real-time operation and outperforms other "naive" ones. On the memory cost side, we map a large room and record the memory requirements for the structure as well as different offline storage format. The online memory cost is such that a large building can be mapped at high resolution especially for the case of mobile robots which do not require full resolution to achieve their tasks. Next chapter will describe our front-end tracking algorithms and how they take the most of our structure to enable fast, agile and robust tracking.

第5章

Direct and Model Based Tracking

Front-end approaches come in different flavors in the literature following which sensing technology they base on, the range of movement required to cope with and the desired accuracy. For laser based method this step called scan-matching aims at aligning a laser scan with a previous scan or an online map. It divides in ICP [119] based methods [120], correlative methods [121], greedy hill-climbing methods [122] or direct optimization like Hector Mapping [123]. ICP has been arguably the most popular of these approaches and can, with good initialization, converge to an accurate solution in few iterations. ICP based methods suffer from two main constraints. The first one, as already pointed out, requires initialization of the optimization process in a convex neighborhood of the solution. Further initializations can result in false solutions or can be blocked in local minimums. The second constraint remains in the nearest neighbor search performed at the beginning of each iteration of the algorithm. This step is seen as the most time consuming point of the algorithm and can without appropriate structures or search strategies hinder real-time performance. In order to speed up the nearest neighbor step, well established method use KD-trees or regular trees like octrees or hierarchical trees or dynamically configure at runtime which partitioning can yield more speed-up such as the approach taken in the widely used open-source implementation FLANN [124]. Strategies for speed-up include limiting the neighbor search to a neighborhood around the input point, requiring only an approximate nearest neighbor, or searching the neighbor on a common projection of the 3D space. A comparison of ICP variants is provided in [125] and a comparison of nearest neighbor search implementations can be found in [126]. The ICP algorithm can be used to align any set of ordered or non ordered point clouds. They provide a good compromise between speed, range and precision and will be the preferred class of algorithm we base our fame to model tracking on. Monocular SLAM approaches on the other hand have for a long time primary used a feature based tracking like PTAM [68] where corner or blob detectors extract interesting points in an input image, descriptors are computed for all or a set of these features, associations are computed and an outliers robust optimization scheme like RANASAC iterate to find the best transform between two successive frames. Associations can be found at higher speeds using a fast search scheme like vocabulary trees [94] or indexes. Direct optimization methods on the other hand go back to Lucas and Kanade's gradient descent based approach but with the advent of more affordable paralell programming have only recently increasingly superseded feature based approach in literature. Some very recent successful approaches include DTAM [84], DVO [103][104], SVO [72] and LSD-SLAM [127][85][86]. An early description of visual tracking methods is provided in [128] while a complete description of recent feature based tracking pipeline is provided in the excellent set of tutorials by Scaramuzza and Fraundorfer [82][83]. Direct optimization methods in contrast to feature based approaches use all or most of the data available ie all or a large set of pixels in the input stage without computing a discriminating descriptor for each point to include in the optimization step. By doing so, they retain most of the important information which in contrast can be skipped by labeling only a small fraction of entry points as features and hence provide higher tracking accuracies. Dense approaches like DTAM use all pixels available while most recent methods guarantee more speed-up using only pixels associated with strong gradient which has been labeled as semi-dense methods like [127] or semi-direct scheme like SVO [72] combining direct alignment with light feature based refinement.

This chapter introduces four tracking methods, two of which are geometric methods while the two others are photometric. These algorithms can further be classified as direct optimization based and model based. All the proposed approaches are dense in the sense that no feature extraction scheme is extracted but all pixels from the input stage are used as such. We propose a full derivation of each of these algorithms as well as strategies for additional speedups and robustness. We have tested their speed and convergence properties which mainly showed the superiority of the model based geometric tracker while the direct optimization photometric tracker proved the most valuable photometric alternative. This approach has essentially described front end approaches. A SLAM system made solely of front end algorithms can behave well locally but starts to accumulate serious drift and global errors as the sensor moves away from the starting point and explores large environments. The solution to the full 3D SLAM problem is discussed in next chapter.

第6章

Full 3D SLAM

We describe in this chapter our full system architecture as well as how to solve the full 3D SLAM. In the last chapter, we introduced four tracking approaches. Among these four, the model based geometric tracker and the direct photometric optimization tracker proved more performing than the other algorithms. We therefore chose to use them in cascade to allow our system even increased robustness in the case of systems with poor geometric features or poor photometric features. Used alone a geometric tracker alone can fail in flat areas where not enough or noisy geometric features as shown in figure 6. For the same scenario, if enough photometric features can be extracted from the scene, using the geometric and the photometric trackers in cascade allows correct tracking as shown in figure 6. Conversely, as shown in figure 6, using the photometric tracker alone in areas with poor photometric features can result in bad quality tracking and in consequence failed mapping while the geometric counterpart can converge nicely 6. For such reasons, and especially for systems which need to guarantee a degree of robustness against failure like disaster area exploring robots, using both tracking schemes in cascade is important.

6.1 Back-end SLAM

As it has been shown in last chapter, front-end SLAM systems are characterized with an accuracy and a precision. The accuracy of the system can be impacted by the systemic errors like poor calibration, approximations, or by the environment itself. As the camera moves further away, dynamic objects, occlusions, decreasing overlapping can deprive the incremental tracking of precious information and hence result in erroneous estimates. The tracking result is also entailed with variable noise as the process is impacted with various random noise sources. As new views are added by the mapper, the incremental accuracy is such that the process seems to map accurately but when the camera moves further and drift accumulates, comparison of the last and first frames indicates a clear global error. Such phenomenon is exacerbated when the mobile agent loops back to previously seen areas and the error is such that the tracker cannot possibly recover a mapping good enough to

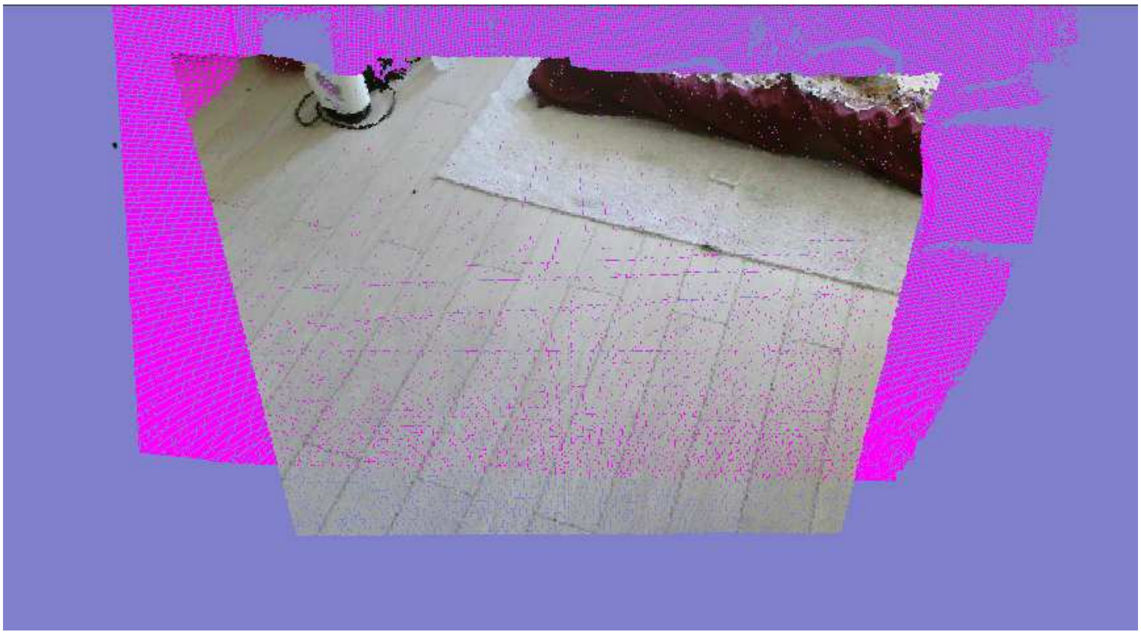


図 6.1: Tracking with a geometric tracker alone. In environments with little geometric features failure can happen.

transit from the first and last views. Loop closure if not properly handled can create enormous corruption in the map, and eventually cause global localization failure. This is shown in figures 6.1.1 and 6.1.1. Figure 6.1.1 shows an experiment conducted in a room of 25m2. The camera traverses the room and loops back. Figure 6.1.1 shows the reconstructed environment with an apparent loop closure artifact in the center of the image. The artifact is zoomed further in figure 6.1.1. The amplitude of the error in this case is small and is not likely to seriously corrupt the map, cause high tracking errors or complete loss. The experimentation section presents a second case where the trajectory has been modified to pass through more error prone views and the resulting loop closure error is drastically more severe.

6.1.1 Loop Detection

In order to solve the loop closing problem two essential components are necessary. The first one is a loop detector. The loop detection in literature has taken

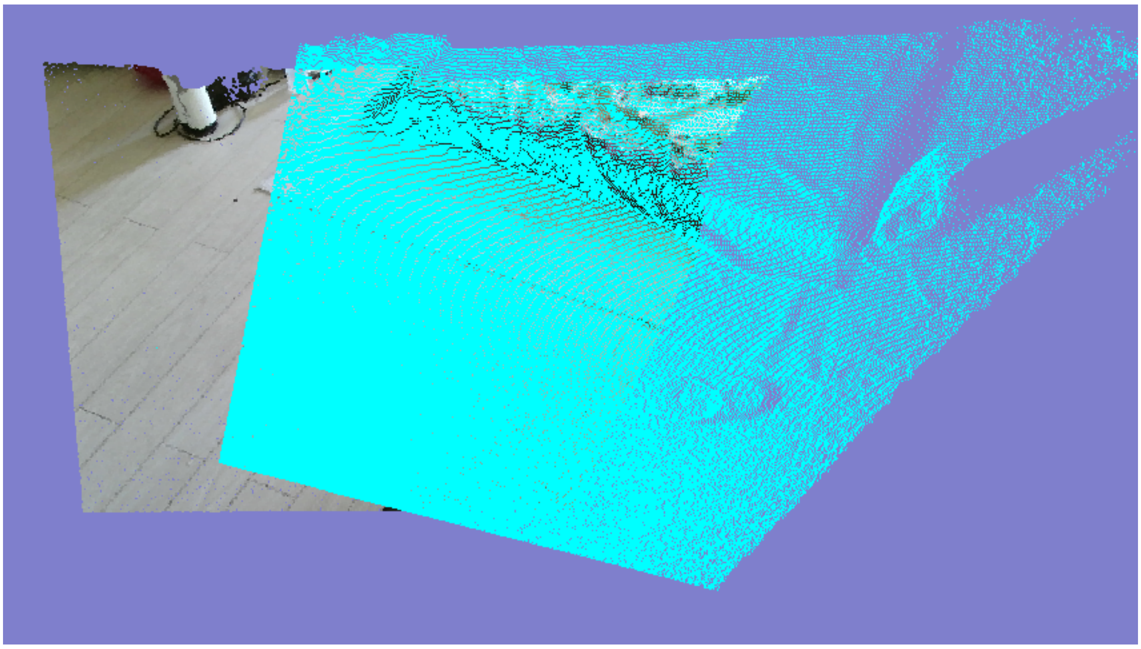


図 6.2: Tracking with a geometric tracker and a photometric tracker. In environments with little geometric features tracking converges.

many forms such as matching visual appearance [56][57][58] or any other frame to frame matching algorithm such as laser scan-matching. Classifier based approaches which create a dictionary of visual words can scale to very large environments. The matching step essentially labels each frame to match against using visual words only and hence a simple word proximity criteria can prune away most of the candidates and return a small number of candidates only. If the detection step is fast and scales nicely with the number of frames, a major drawback is the possible necessity of training the classifier either once by collecting samples from the environment to map and hence the most likely features or by reconfiguring the classifier on the fly during the online operation as new features come in. Without a sparse classification the return frames can be large and the benefit of the approach hindered by poor configuration. The need to reorganize can either impact the online operation or work against the present work's objectives where environments are supposed to be completely random and unknown. For this reason we choose a frame-to-frame

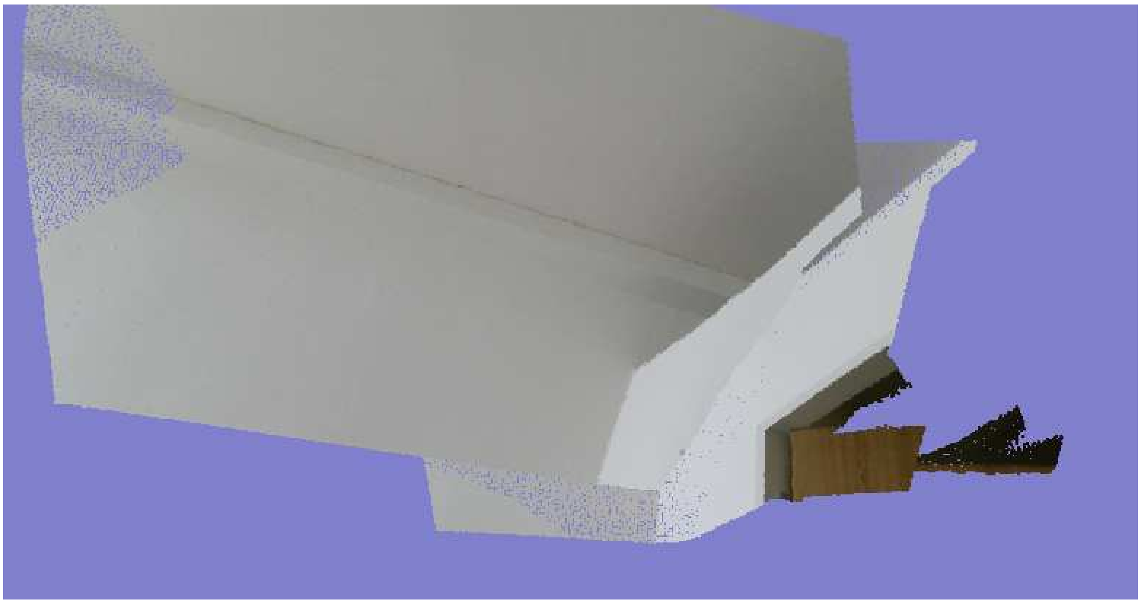


図 6.3: Tracking with a photometric tracker alone. In environments with little photometric features failure can happen.

approach and use the photometric matcher to complete the match as it has showed an attraction basin large enough to find correspondences from frame far apart from each other. As a consequence, in addition to the online map update by the front-end SLAM counterpart, the back-end SLAM manages a graph. The graph essentially stores the sensory data from all sensory input along with a position estimate and an information matrix. The detection step consists in scanning all the frame from the past data and returns the best match candidates. These candidates are further scanned in order to compute a more precise transform. The candidate with the highest score is selected to be the closing view. If the score is above a threshold the loop closure is further confirmed. Note that the lack of convexity of the photometric matcher calls for a higher number of iterations to scan potential candidates than actually required for the tracker. Each single scan runs for a fraction of a millisecond on multi-core CPU and hence limits in practice the number of frames to scan to some 100 frames for strict real-time requirements.

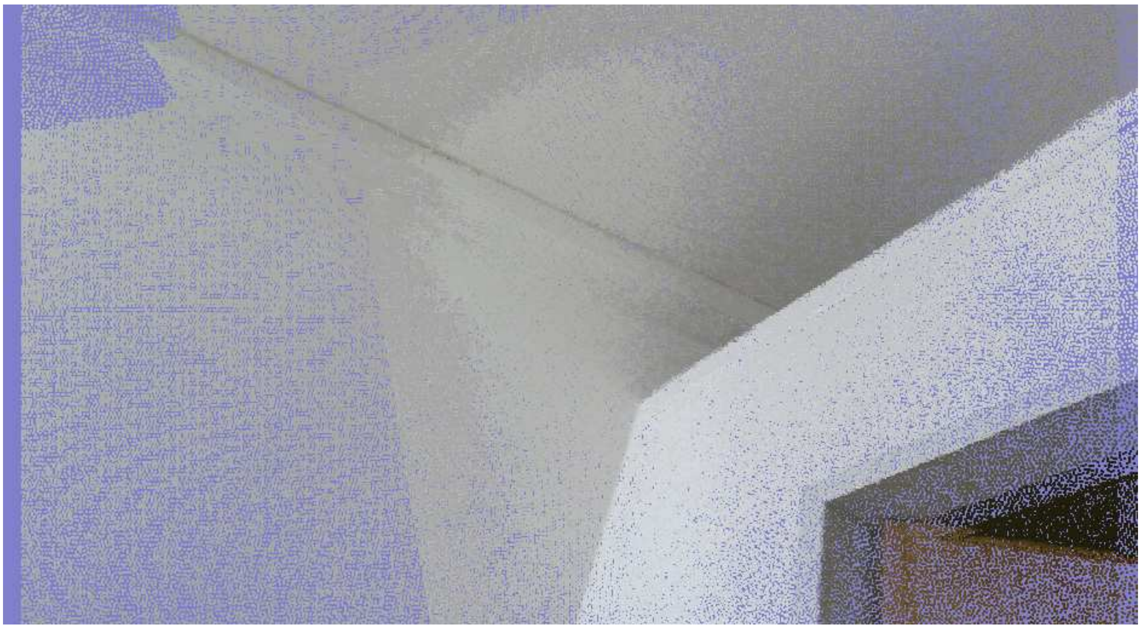


図 6.4: Tracking with a geometric tracker alone. In environments with little photometric geatures and enough geometric features tracking converges.

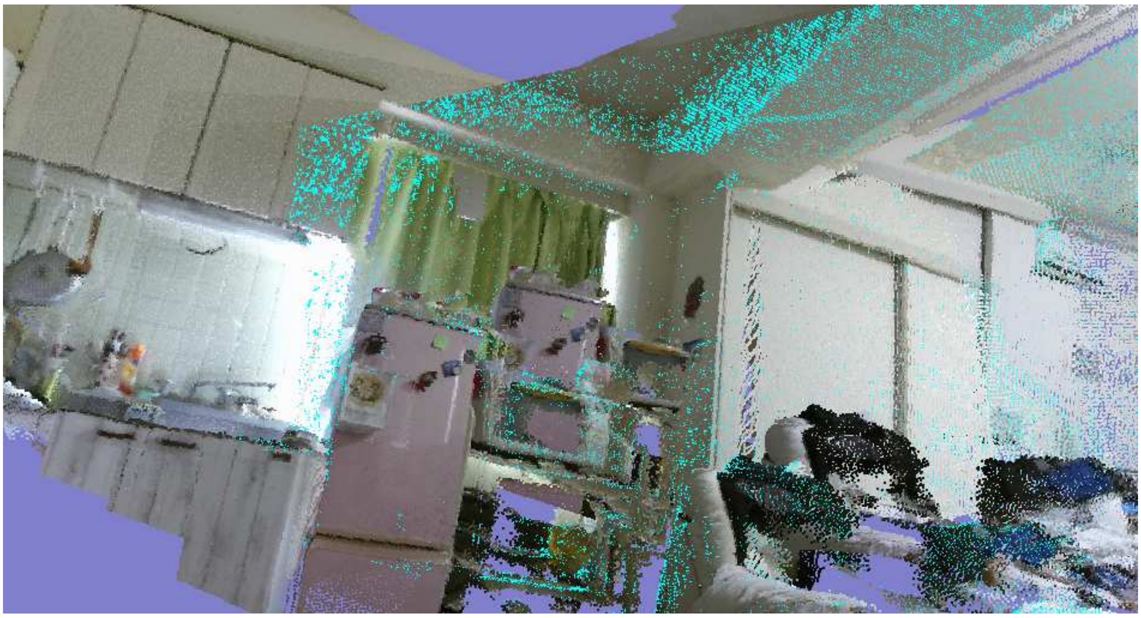


図 6.5: Looping back trajectory. Loop closure issue occurs.

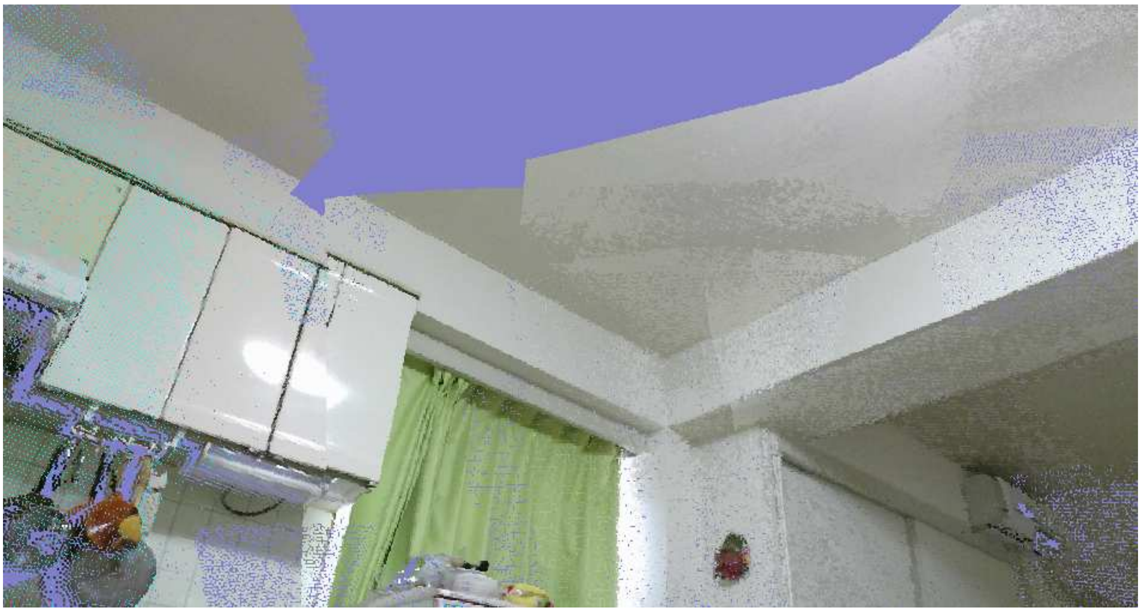


図 6.6: Looping back trajectory. Loop closure issue occurs.

6.1.2 Loop optimization

Recent insights in sparse linear algebra in association with the full SLAM derivation has led to increasingly fast and efficient graph optimization approaches. The proposed literature for graph optimization is immense and has reached a state of maturity [59][60][61][62][63][64][65] with g2o being probably the most used solution in recent literature. For the present case we use the most recent g2o [66] framework to solve the graph optimization problem. Upon loop closure detection, and if a certain threshold in the quality of the detection holds, the loop optimization takes place. The information matrix associated with each edge can be expressed as :

$$I = (J^T J)^{-1} \tag{6.1}$$

where the matrix J denotes the jacobian of the tracking problem. This matrix essentially captures the quality of the tracking. Increments with fewer associations will essentially have less information and hence less weight in the optimization process. Also note how the weights defined in chapter 5 improve the quality of optimization as noisier data are assigned smaller weights which in return translate in less information. As such, branches which are likely to be corrected are first those which yield fewer number of associations then those on which noisy points prevail. Note how each dimension of our six dimensional tracking vector act with a certain degree of independence as the number of associations found can be high but contribute in one degree of freedom only. In such case the information matrix will change in consequence to accommodate for the fact that one degree of freedom is noisy while the other are computed with a higher degree of confidence. The graph optimization essentially results in new position estimates for each frame stored in the graph while the online map stores all past data in a fused form. The fact that data has been fused in the online map means that rolling back in the past is impossible. In this regard, when the graph is optimized the sub map stored in the limited depth structure is erased. Since the local submaps are essentially allocated sequentially the erasing operation requires a stack pointer manipulation and a contiguous array erase only and hence runs fast. A reinsertion operation then takes place where all the frame sensory data are inserted all over using the corrected position estimates. Since the insertion operation is fast as it has been proven in chapter 4, a redrawing of a whole room completes in about a second. On the overall the whole detection, optimization and redrawing of local submaps takes about a little more than a second to finish. During this time the robot could have moved away which can, after returning to normal tracking operation account for a too much displacement for the system to track correctly. In this case we can either call for stationary behavior on the part of a mobile robot for one second time or reduce the navigation speed. For such a reason, during optimization and redrawing, the model based geometric tracker is disabled and only the photometric tracker is kept running as a local copy of the last keyframe is saved and tracking is performed against. Moreover, since we use photometric matching for loop detection, we are guaranteed that areas of matching are also areas where photometric features are strong enough and hence where a photometric tracker alone can achieve good accuracy. Doing so, during the time required for optimization and redrawing the system can still be reactive enough. During this short period the number of iterations assigned to the photometric tracker is increased.

6.2 Experimentation

6.2.1 Room Experiment

We use a calibrated kinect2 sensor for the present experiment. We walk with a handheld camera through a 25 m2 room and run our system. The current scenario contains one loop only. The experiment was recorded at human natural speed. A step by step reconstruction is shown in figure 6.2.1.

The upper view of the whole open loop reconstruction is shown in figure 6.2.1. The figure shows an overall good reconstruction with small incremental drift merely discernible. However as the drift accumulates and the front-end SLAM loops back to previously visited areas, a loop closure artifact becomes apparent. 6.2.1 shows a zoomed view on the loop closure point. The error is severe and even if it does not result on a tracking loss, continuing the experiment will lead to overwriting the previously stored map and more serious drift. Note that the corrupted map also results in an increased memory consumption in order to store additional noisy points. Therefore, upon loop closure, the loop detection routine detects a loop closure and computes a transform between the last acquired frame and the closing frame. This transform is added to the graph as a new constraint. The optimization step takes place and new position estimates are computed. Note that using the geometric tracker alone or the photometric tracker alone result in a different trajectory. For example figure 6.2.1 shows the loop closure point using a geometric only for tracking. As it can be seen the amplitude of the error is higher which shows the benefit of using a combination of a geometric and a photometric tracker.

The full result of the whole process is shown in figure 6.2.1. The figure shows some views taken from the reconstructed environment. Note how the environment looks coherent without apparent major distortions. Some RGB color artifacts appear in the scene as a result of the change in luminosity during the experiment. This phenomenon can further be dealt with by taking into account the angle of view during the RGB data fusion. The data shown is rendered as a point cloud. Further meshing can also take place to provide with even more visually appealing result. For mobile robots the reconstruction shown has greater value.

Finally, figure 6.2.1 shows the runtime profile of the experiment. The computational time of each of the major threads is recorded at each frame. All the experiments have been run on a laptop computer with a intel i7-4900MQ CPU and a low end Nvidia Geforce GT 730M GPU. As the figure shows that the system guarantees a 20Hz runtime on this low end configuration. It is important to highlight that the sum of all thread computational time is far more superior than the 50ms required for our system. This shows the benefits of our parallel architecture which runs different threads on different resources which guarantees both higher update speeds and optimal use of our system resources. The system can be separated in two main update steps. The first one runs the sensor and the mapper in parallel while the second runs the tracker and the drawer in parallel. Each two threads in each step executes mainly on either CPU or GPU side. As a result, the update time can roughly be expressed as :

$$\Delta t = max(\delta t_{sensor}, \delta t_{mapper}) + max(\delta t_{tracker}, \delta t_{drawer}) \tag{6.2}$$

The sensor thread requires a 20ms in rounded up average to compute the sensor pyramid. This can be contrasted by the 12ms only required by the CPU in parallel to register new point clouds in the map. With slightly better GPU the processing time of the sensor thread can easily drop to about 10ms only. The tracking time is the sum of the required time to complete each of the geometric and photometric tracker. The photometric tracker used for this experiment has been run on one core only whereas it can be trivially parallelized on multiple cores for additional speedup. With four cores we expect a bit more than twice as a speedup. Moreover, with finer parameter tuning the execution time can drop further as the number of iterations has been used without parameter optimization in mind. The mapper has a 12 ms running time which is fast enough for most applications. The drawer's execution time varies from few milliseconds to 15ms. Note that most of the time the drawer has little to run. The sudden increase in the drawer time happens when new areas which are not stored yet in buffer need to be drawn. For incremental drawing few areas of the image only are refreshed whereas during initialization and looping back large chunks of the map are modified. The drawing time stays fast enough. We should highlight here that the experiment has been conducted with 5 levels of tracking and mapping. For most of the robotics navigation applications, the requirements are much lower than those experimented in this section as the maximum level depth can be set to 2 which brings the mapping time further down and has very strong impact on the tracker execution time as it has been shown in the previous chapter. Moreover, upon loop closure, the loop closure detection time was about 30ms while the graph optimization time was about 10 ms as the number of optimization steps was set relatively low. The redrawing time was the biggest source of system latency and required one second to complete.

6.2.2 Two Floors Laboratory Structure

The next experiment considers the case of large environments. A handheld kinect2 sensor travels a structure which consists of two large and one small room, two small kitchen spaces, two corridors, stairs which spread accross two floors. The environment presents three loops through each of the main three rooms and a number of source of noise and inaccuracies. Most notably the floor in the corridor was made of a material with high reflectivity which strongly corrupts the point cloud acquired with the kinect2 sensor. Moreover, large uniform walls with little texture spread all along the stair area which marks the transition between the two floors. The overall walk accounts for about a 100m in linear displacement and 150 rad of angular displacement mostly due to body motions during the experiment. The overall reconstructed environment is shown in figure : 6.2.2.

As it can be shown in the figure the overall reconstruction presents little visual global error, the structure alignments over the floors does not present major disturbances even though the environment presented serious mapping challenges and multiple sources of severe noise. Note that the trajectory crosses multiple times narrow doors where the field of view shrinks to a small window only with little salient texture only. These areas of transition accounted for most of the visual errors observed. Small and fine details in the map were inserted with high accuracy and loops closed correctly. Both the photometric tracker and geometric tracker were used in cascade for the whole experiment. Note that either approach's result is discarded when the score falls under a precomputed threshold. Throughout the entire reconstruction process tracking was not lost, ie both approach never failed simultaneously. The geometric tracker failure indicates strong corruption in the map which never happened through the experiment even in areas with high geometrical frequencies whereas the geometric traker failure happened twice : once during the transition through a narrow door and multiple times in the transition area marked with large white structure with scarse texture. In these cases the geometric tracker result alone was conserved and sufficed to carry on the experiment without any major decrease in the accuray. Note that using either of the approaches alone in such complex environment result in either large global localization errors with non negligible translational error along the geometrically ambiguous corridors (when using the geometric tracker alone) or early global failure due to tracking loss (when using the photometric tracker alone). This is shown in figure 6.2.2. Note that using both the geometric and photometric tracker solves the tracking loss issue but without further reasoning on each of the tracker's performance also results in important global error.

The scores for each tracker has been recorded troughout the experiment and plotted in figure 6.2.2. Note that only frames which preceed a map write has been conserved in the plot and denote the lowest score before new keyframe insertion. These scores take two factors into consideration : the number of data assocations and the noisiness of points in the point cloud processed. The point clouds recorded on the first floor were taken from closer range than those acquired in the second floor. Moreover, rooms and the corridor in the first floor presented high textured areas whereas the stairs presented scarse texture and the second floor corridor had minimal texture only. This is translated by higher scores on the first part of the plot, low ones in the middle part which correspond to the stairs area then average scores in the last part of the plots which presented less texture and longer range data. Note that the photometric tracker score slumped multiple times especially in the middle part which corresponds the the stairs area. The point clouds corresponding to the the point of failure are further provided in figure 6.2.2 and provide concrete examples of challenging environments to map. Note how the geometric tracker performs uniformly accross the experiment with now major drop that would indicate serious mapping failure.

6.3 Conclusion

In this chapter we describe the overall architecture of our system and how we solve the full SLAM problem. The system can scale to larger workspaces and conserves a good runtime profile when putting all the previously described components all together. This is due to good allocation of system resources and concurrent execution between the macro threads of our system. Experiments have shown how the system can keep with tight speed update requirements even when run on low end hardware using a high number of depth levels. Concrete experiments have shown that the system is robust enough to run through large and challenging environments and with limited global error.

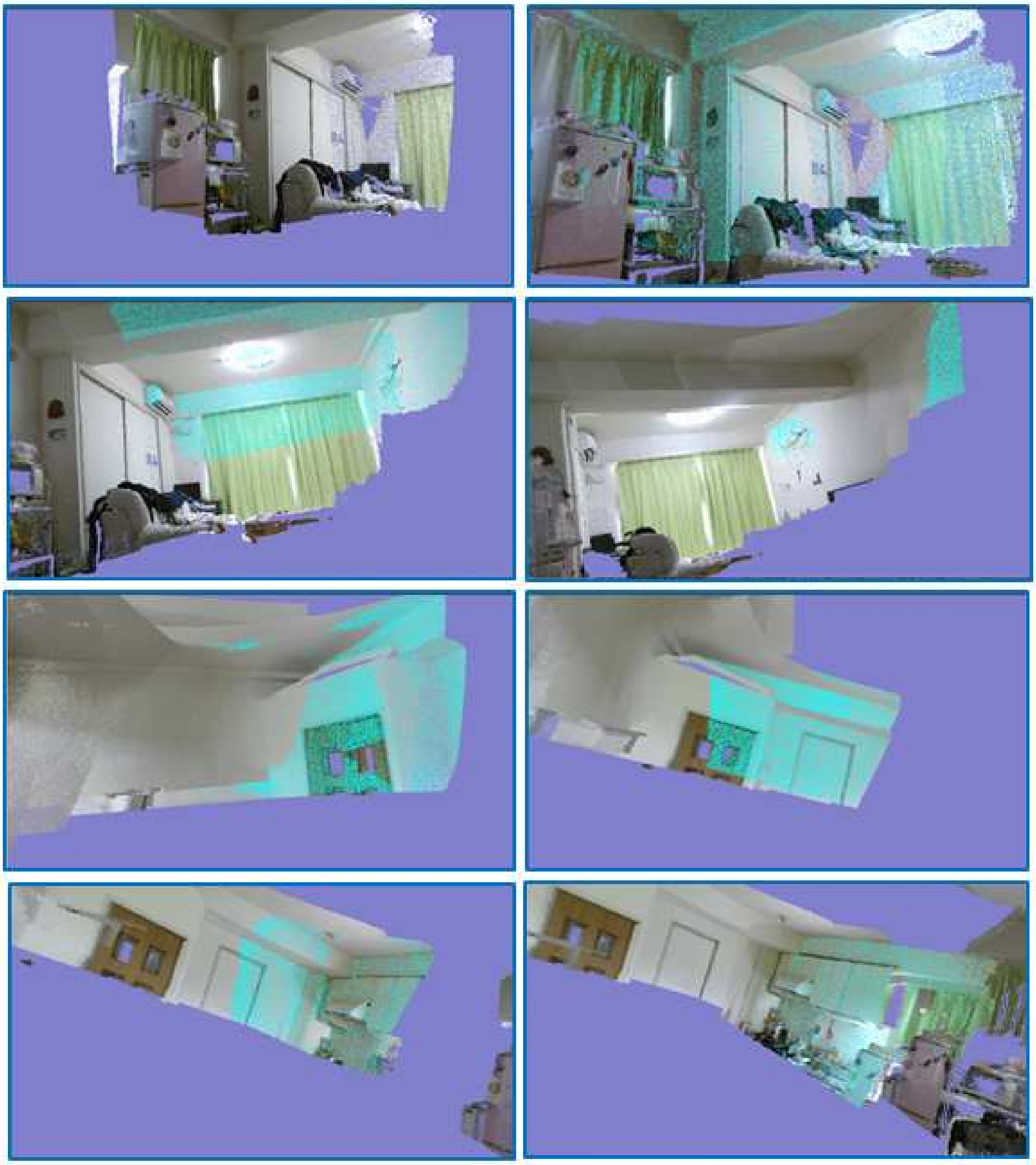
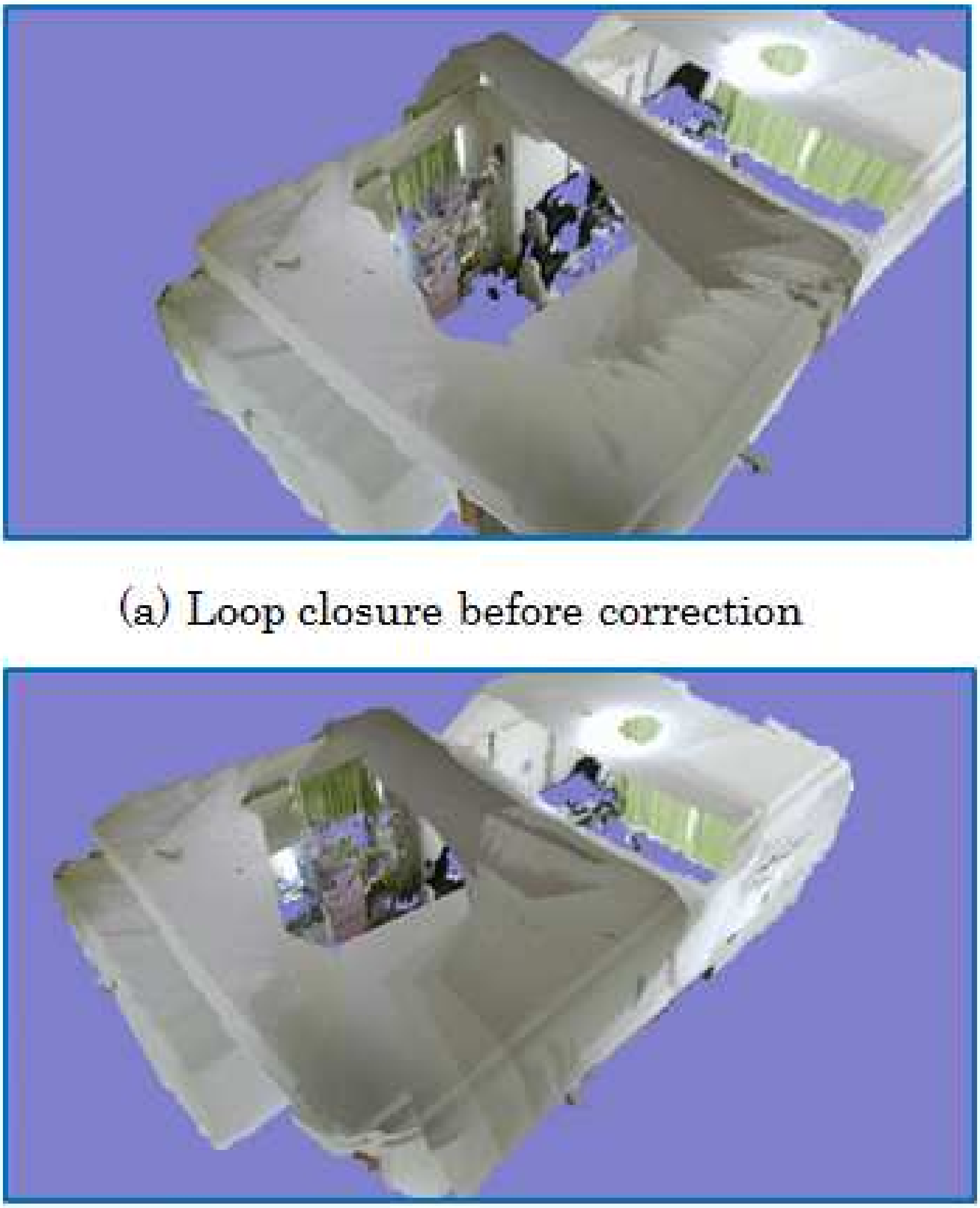


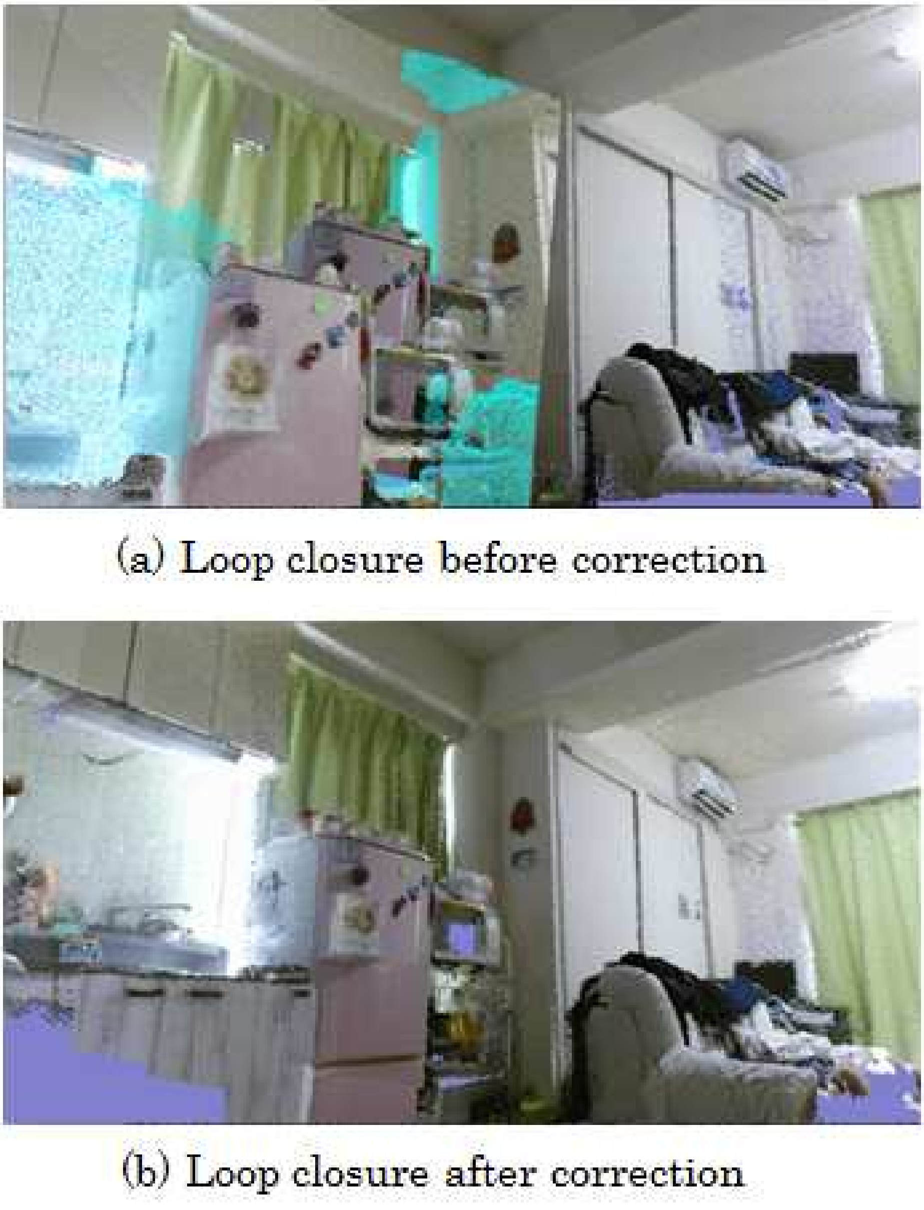
図 6.7: Step by step reconstruction of a room sized environment.



(a) Loop closure before correction

(b) Loop closure after correction

図 6.8: Upper view of the full room reconstruction. Serious artifacts appear upon loop closure in (a). These are corrected in (b)



(a) Loop closure before correction

(b) Loop closure after correction

図 6.9: Zooming on the loop closure point. Serious artifacts appear upon loop closure in (a). These are corrected in (b)

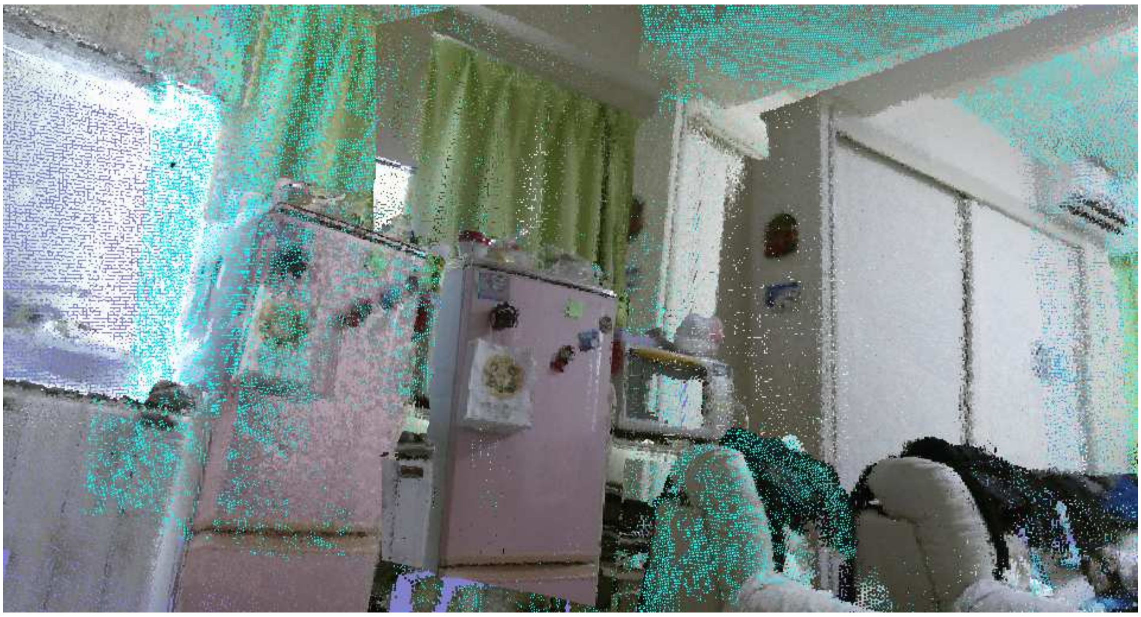


図 6.10: Loop closure error using the geometric tracker alone.

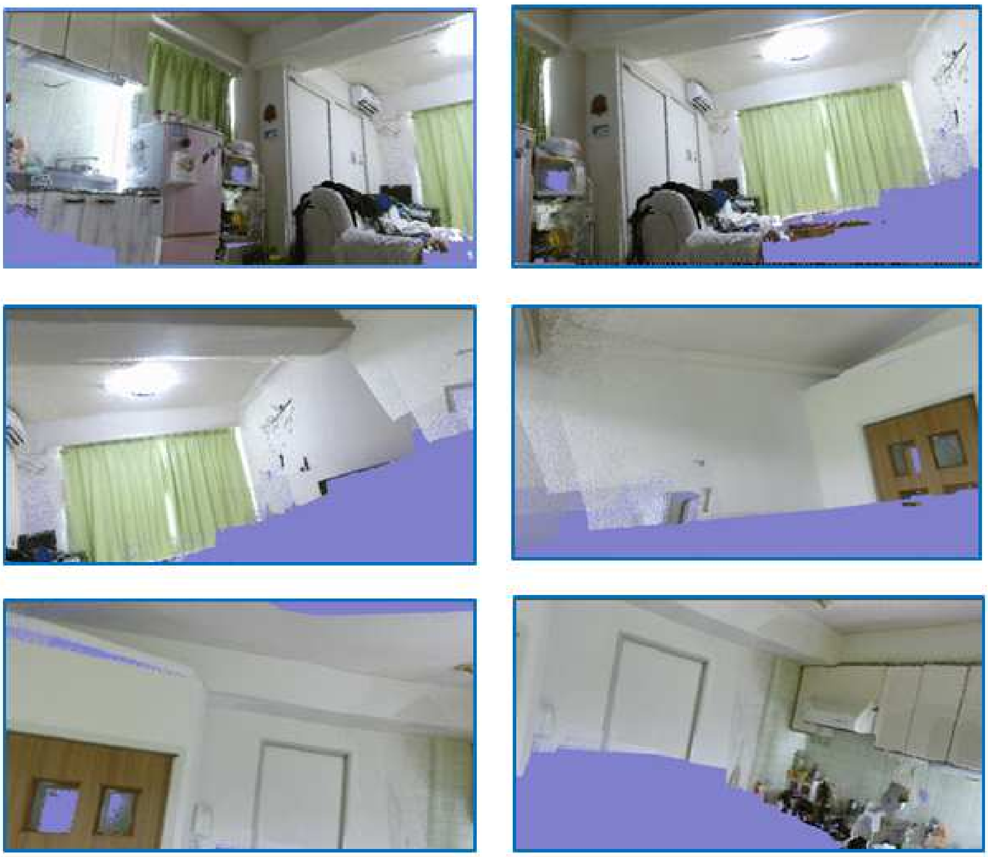


図 6.11: Step by step reconstruction of a room sized environment.

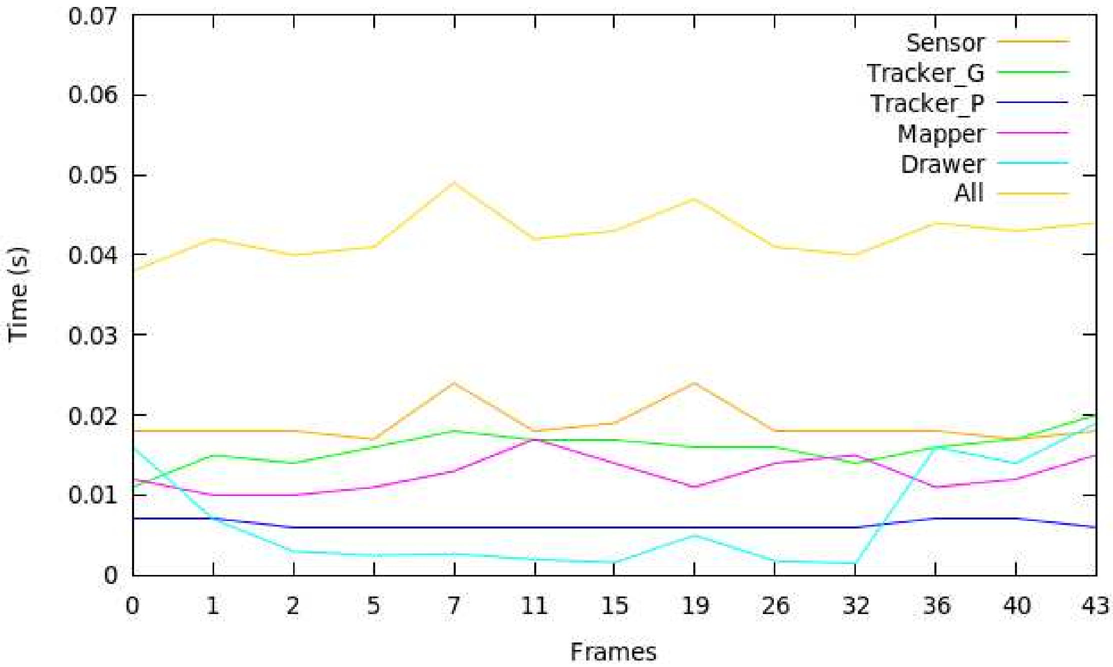


図 6.12: Runtime profile of the system.

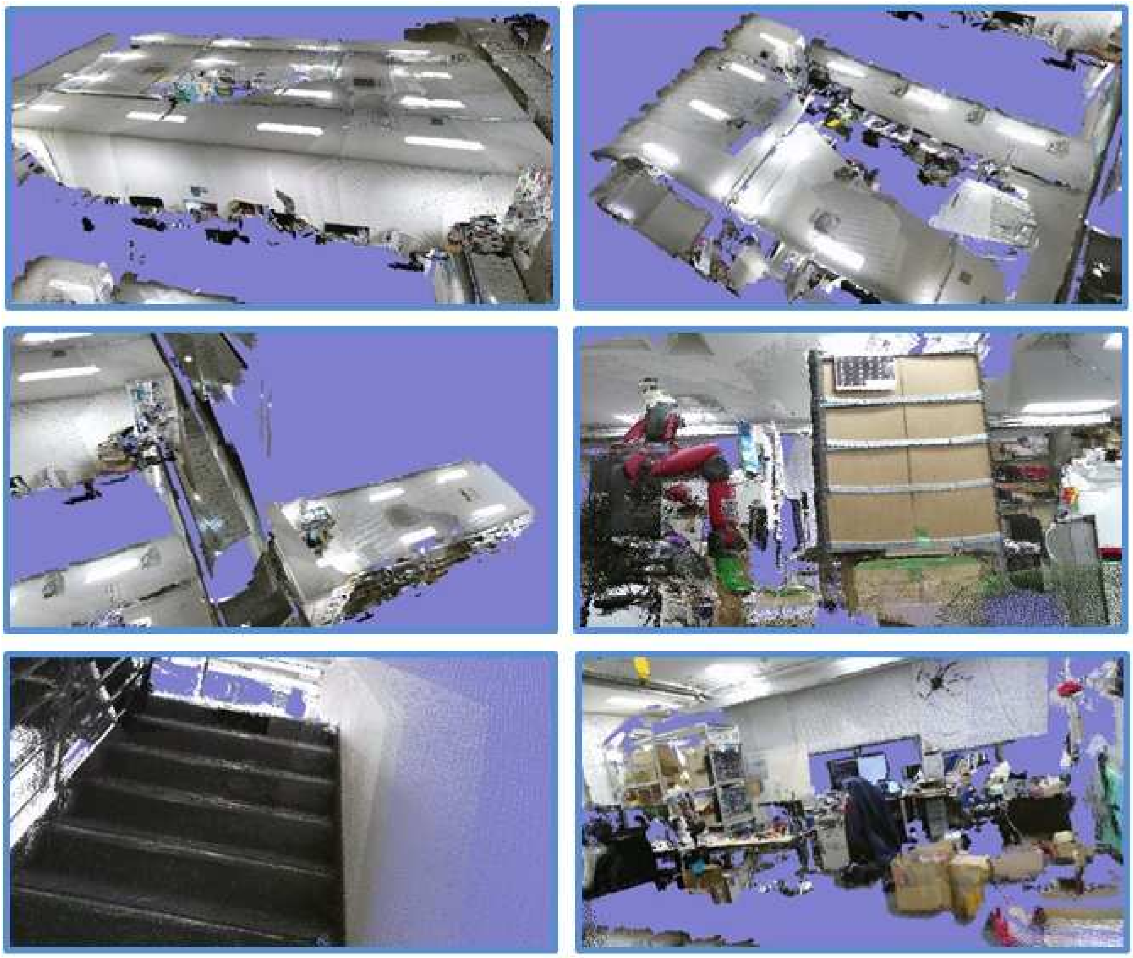


図 6.13: Reconstructed lab environment.

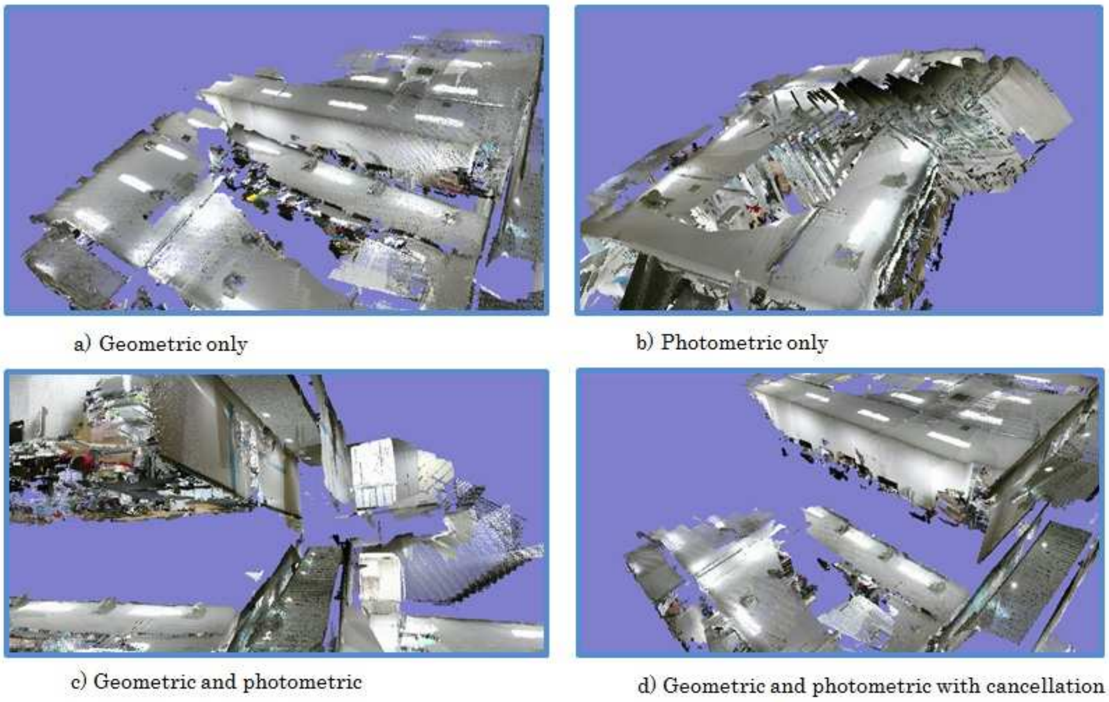

a) Geometric only
b) Photometric only
c) Geometric and photometric
d) Geometric and photometric with cancellation

図 6.14: Using the geometric tracker alone (on the left) or the photometric approach alone (on the right).

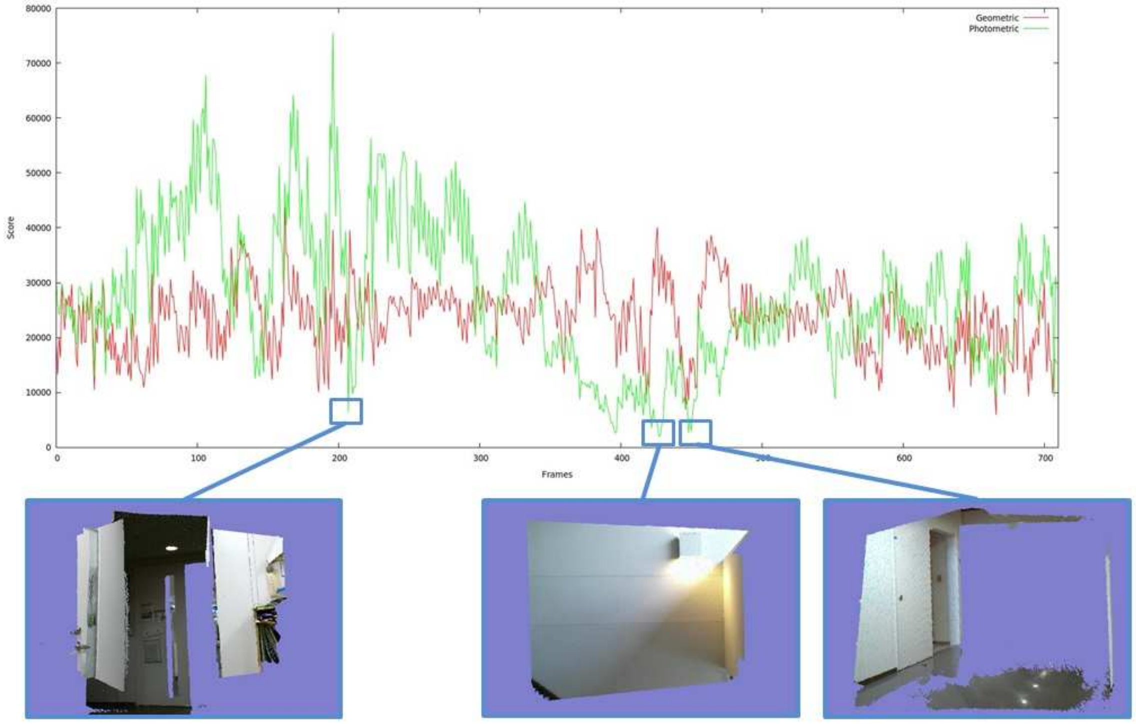


図 6.15: Geometric and photometric trackers scores.

第7章

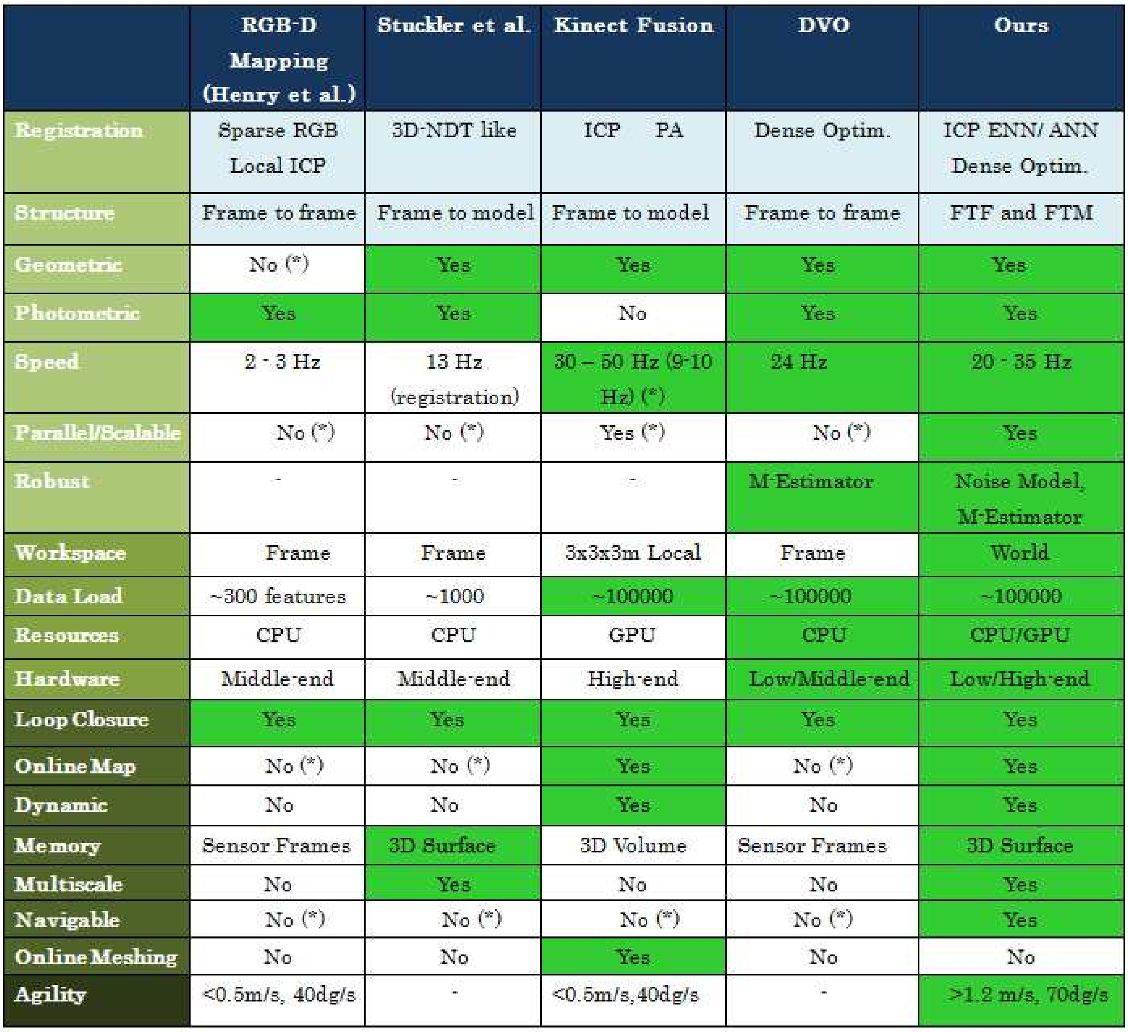
Conclusion

The present research has presented a solution for the full 3D SLAM problem. This solution can be adapted for real-time robotic navigation in unknown and challenging environments or in order to quickly create highly detailed dense volumetric maps. Such map representations can later be used for visually appealing rendering or as a base for robotic localization. Our contribution to the field is threefold :

- We proposed a map representation with associated stepping and traversal iterators. Our map formulation allowed us to derive fast insertion, freeing, raycasting and neighbor search algorithms. The enhanced speed we obtain is crucial to be able to build highly detailed maps online and in real-time. The memory compression is also such that large workspaces and maps can be handled. Finally, the map is essentially multiscale. The multiscale property is used by all algorithms for speed-ups but also as different points have different noise amplitude, mapping proceeds by inserting each point at the correct scale hence avoiding corruptions of more precise voxels with less precise data.

- A real-time agile tracker which builds on the association of a direct optimization based dense photometric tracker and a model based geometric tracker. The geometric tracker builds on our map iterators to extract at high speed the exact nearest neighbors in a 3D neighborhood around candidates points and run subsequent ICP optimization. This tracker shows large basin of attraction to the minimum cost solution with a marked convexity and hence converges in few iterations only. The photometric tracker complements the geometric tracker's behavior and adds more stability and robustness against environments with poor geometric features. A sensor model associates each point at the input stage with a proper variance derived from a normal distribution approximation. The point noisiness is taken into account during the tracking stage to yield more noise resilient estimates.

- An architecture which solves the full 3D SLAM problem at high speed. The architecture blends tracking, sensing, map insertions, map freeing, drawing, loop detection and optimization in a concurrent way and such that at each moment distinct threads request different computational resources on the CPU or the GPU side. The architecture as it has been designed allows to solve the full 3D SLAM problem and render highly detailed 3D maps in real-time.



	RGB-D Mapping (Henry et al.)	Stuckler et al.	Kinect Fusion	DVO	Ours
Registration	Sparse RGB Local ICP	3D-NDT like	ICP PA	Dense Optim.	ICP ENN/ ANN Dense Optim.
Structure	Frame to frame	Frame to model	Frame to model	Frame to frame	FTF and FTM
Geometric	No (*)	Yes	Yes	Yes	Yes
Photometric	Yes	Yes	No	Yes	Yes
Speed	2 - 3 Hz	13 Hz (registration)	30 – 50 Hz (9-10 Hz) (*)	24 Hz	20 - 35 Hz
Parallel/Scalable	No (*)	No (*)	Yes (*)	No (*)	Yes
Robust	-	-	-	M-Estimator	Noise Model, M-Estimator
Workspace	Frame	Frame	3x3x3m Local	Frame	World
Data Load	~300 features	~1000	~100000	~100000	~100000
Resources	CPU	CPU	GPU	CPU	CPU/GPU
Hardware	Middle-end	Middle-end	High-end	Low/Middle-end	Low/High-end
Loop Closure	Yes	Yes	Yes	Yes	Yes
Online Map	No (*)	No (*)	Yes	No (*)	Yes
Dynamic	No	No	Yes	No	Yes
Memory	Sensor Frames	3D Surface	3D Volume	Sensor Frames	3D Surface
Multiscale	No	Yes	No	No	Yes
Navigable	No (*)	No (*)	No (*)	No (*)	Yes
Online Meshing	No	No	Yes	No	No
Agility	<0.5m/s, 40dg/s	-	<0.5m/s,40dg/s	-	>1.2 m/s, 70dg/s

図 7.1: Comparison of our approach with other state-of-art approaches.

Figure 7 shows a comparison chart of our approach with some of the other state-of-art approaches. Each of the approaches compared with has different properties, use different tracking strategy, mapping representation or hardware support. In terms of speed, our approaches reaches equivalent performance with the GPU based

KinectFusion[13] without requiring a high end GPU to run. Conversely, it offers superior flexibility as the multiscale character allows to tradeoff quality for speed on the fly. Note that compared with KinectFusion, we have neither workspace limitations nor tight memory constraints. We also support revisiting as a natural property and very fast point look-up routines at any level of resolution. Stuckler's approach [14][99][100] uses a multiscale data structure but that comes at the expense of lower runtime speed, accuracy and agility. Henry et al. [92][93] also use ICP as a geometric registration support and feature based tracking but show significantly slower update speed mainly due to their costly ICP implementation. Moreover, the online map reconstruction routine accounts for a significant part of the computational burden. DVO by Kerl et al. [103][104] is one of the most recent and promising approaches for dense implementation on mobile robot. DVO essentially updates a graph and does not propose a routine to build multiscale maps online. In this regard it proposes less features than our approach for robot navigation and shows less agility.

Throughout this work experiments were conducted with high precision and quality requirements whereas for most robotics scenarios and tasks lower degree of resolution is enough to get through most of the missions assigned. And advantage of our approach is that it can be adapted on the fly with the hardware capabilities or the system precision needs and hence yield increased speed, agility and lower memory footprint.

We see our work as a strong building block behind complete autonomous mobile agents. The online and real-time availability of dense 3D volumetric and multi-scale maps with fast access properties allows to run all subsequent planning, recognition or semantic discrimination tasks seamlessly based on the data streamed from the 3D SLAM process. This work has, by proposing all necessary algorithms and describing a proper architecture, essentially proven that highly detailed maps and agile tracking can be achieved in dynamic environments, on low end hardware with high load sensors, in an online fashion, exhibit high memory compression ratios and still perform with high enough speed.

The present work can be extended in many aspects and also beyond the 3D SLAM realm. As part of an autonomous robotic navigation pipeline, planning algorithms, object and attribute segmentation, semantic discovery can be further appended. Dynamic objects detection, discrimination and representation can also be a promising subject of research. Our work can handle a certain degree of variability and environment hazards as outliers rejection and weighting has been implemented in each if our tracking approaches in order to cope with these. The remaining dynamic points are treated as noise, inserted at first then subject to subsequent freeing. In highly dynamic environments however, their impact can be acute and a routine which takes into consideration the nature of obstacles and alleviates their effect inside the tracking and mapping components seems a promising complement for the present research and a key for narrowing the bridge between mobile intelligent robots and the rest of our societies. In addition, scaling up to very large environments is an important subject to tackle. Even if our approach provides with high memory compression and loop closing capabilities and as such guarantees a natural extension of our approach to large environments, very large exploration scenario can break the boundaries of the underlying system in use. Moreover, ever growing loops and associated drift ask for a place recognition routine which can explore a higher number of candidate without loosing the real-time benefits of our method. For very large environments a promising direction is to augment our work with an additional level of hierarchy. For instance, based on semantic or localization criteria, the system can upload or offload corresponding submaps. Submaps can be handled in a hierarchically superior semantical graph whereas submaps are grown and explored with the process described in the present work. Finally, we have essentially considered the case of a single agent with no interaction with other agents. Collaborative mapping and mutual tracking has been an active subject of research. Extending the present work with such capabilities can also be an interesting path to explore.

参考文献

[1] F.M. Mirzaei and S.I. Roumeliotis. A kalman filter-based algorithm for imu-camera calibration: Observability analysis and performance evaluation. *IEEE Transactions on Robotics*, 24(5):1143 –1156, Oct. 2008.

[2] A.M. Sabatini. Quaternion-based extended kalman filter for determining orientation by inertial and magnetic sensing. *IEEE Transactions on Biomedical Engineering*, 53(7):1346 –1356, July 2006.

[3] Angelo Maria Sabatini. Kalman-filter-based orientation determination using inertial/magnetic sensors: Observability analysis and performance evaluation. *Sensors*, 11(10):9182–9206, 2011.

[4] D. Gebre-Egziabher, G.H. Elkaim, J.D. Powell, and B.W. Parkinson. A gyro-free quaternion-based attitude determination system suitable for implementation using low cost sensors. In *IEEE Position Location and Navigation Symposium*, pages 185 –192, 2000.

[5] S. O. Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. *internal report*, pages 187–194, April 2012.

[6] Davide Scaramuzza, Michael Achtelik, Lefteris Doitsidis, Friedrich Fraundorfer, Elias B. Kosmatopoulos, Agostino Martinelli, Markus W. Achtelik, Margarita Chli, Savvas A. Chatzichristofis, Laurent Kneip, Daniel Gurdan, Lionel Heng, Gim Hee Lee, Simon Lynen, Marc Pollefeys, Alessandro Renzaglia, Roland Siegwart, Jan Carsten Stumpf, Petri Tanskanen, Chiara Troiani, Stephan Weiss, and Lorenz Meier. Vision-controlled micro flying robots: From system design to autonomous navigation and mapping in gps-denied environments. *IEEE Robot. Automat. Mag.*, 21(3):26–40, 2014.

[7] M Achtelik, M Achtelik, Y Brunet, M Chli, S Chatzichristofis, J Decotignie, K Doth, F Fraundorfer, L Kneip, D Gurdan, L Heng, E Kosmatopoulos, L Doitsidis, G Lee, S Lynen, A Martinelli, L Meier, M Pollefeys, D Piguet, A Renzaglia, D Scaramuzza, R Siegwart, J Stumpf, P Tanskanen, C Troiani,

and S Weiss. sfly:swarm of micro flying robots. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[8] Markus Achtelik, Michael Achtelik, Stephan Weiss, and Roland Siegwart. Onboard IMU and monocular vision based control for mavs in unknown in- and outdoor environments. In *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China, 9-13 May 2011*, pages 3056–3063, 2011.

[9] S. Weiss, M. Achtelik, S. Lynen, M. Achtelik, L. Kneip, M. Chli, and R. Siegwart. Monocular Vision for Long-term Micro Aerial Vehicle State Estimation: A Compendium. *Journal of Field Robotics*, 30(5):803–831, 2013.

[10] S Weiss, M Achtelik, S Lynen, M Chli, and R Siegwart. Real-time onboard visual-inertial state estimation and self-calibration of mavs in unknown environments. In *Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[11] L Kneip, M Chli, and R Siegwart. Robust real-time visual odometry with a single camera and an imu. In *Proc. of The British Machine Vision Conference (BMVC)*, Dundee, Scotland, August 2011.

[12] A. Martinelli. Vision and imu data fusion: Closed-form solutions for attitude, speed, absolute scale, and bias determination. *IEEE Transactions on Robotics*, 28(1):44–60, Feb 2012.

[13] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Proceedings of the 2011 10th IEEE International Symposium on Mixed and Augmented Reality*, ISMAR '11, pages 127–136, Washington, DC, USA, 2011. IEEE Computer Society.

[14] J. St 端 ckler and S. Behnke. Robust real-time registration of rgb-d images using multi-resolution surfel representations. In *7th German Conference on Robotics (ROBOTIK)*, 2012.

[15] Tommi Tykkälä, Hannu Hartikainen, Andrew I. Comport, and Joni-Kristian Kämäräinen. RGB-D tracking and reconstruction for TV broadcasts. In *VISAPP 2013 - Proceedings of the International Conference on Computer Vision Theory and Applications, Volume 2, Barcelona, Spain, 21-24 February, 2013.*, pages 247–252, 2013.

[16] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers. Direct camera pose tracking and mapping with signed distance functions. In *Demo Track of the RGB-D Workshop on Advanced Reasoning with Depth Cameras at the Robotics: Science and Systems Conference (RSS)*, June 2013.

[17] Abraham Bachrach, Samuel Prentice, Ruijie He, Peter Henry, Albert S Huang, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. Estimation, planning, and mapping for autonomous flight using an rgb-d camera in gps-denied environments. *Int. J. Rob. Res.*, 31(11):1320–1343, September 2012.

[18] Shaojie Shen, Nathan Michael, and Vijay Kumar. Autonomous indoor 3d exploration with a micro-aerial vehicle. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 9–15. IEEE, 2012.

[19] S. Thrun. A probabilistic on-line mapping algorithm for teams of mobile robots. *The International journal of Robotics Research*, 20(5):335–363, 2001.

[20] Youssef Ktiri, Tomoaki Yoshikai, , and Masayuki Inaba. Multi-robot exploration framework using robot vision and laser range data. In *IEEE/SICE International Symposium on System Integration SII*, 2011.

[21] Shaojie Shen, N. Michael, and V. Kumar. Autonomous multi-floor indoor navigation with a computationally constrained mav. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 20 –25, may 2011.

[22] A. Bachrach, A. de Winter, Ruijie He, G. Hemann, S. Prentice, and N. Roy. Range - robust autonomous navigation in gps-denied environments. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1096 –1097, may 2010.

[23] S. Grzonka, G. Grisetti, and W. Burgard. Towards a navigation system for autonomous indoor flying. In *IEEE International Conference on Robotics and Automation*, pages 2878 –2883, may 2009.

[24] M. Angermann and P. Robertson. Footslam: Pedestrian simultaneous localization and mapping without exteroceptive sensors;hitchhiking on human perception and cognition. *Proceedings of the IEEE*, 100(Special Centennial Issue):1840–1848, May 2012.

[25] Maria Garcia Puyol, Patrick Robertson, and Oliver Heirich. Complexity-reduced footslam for indoor pedestrian navigation using a geographic tree-based data structure. *Journal of Location Based Services*, 7(3):182–208, September 2013.

[26] Y. Bar-Shalom. Update with out-of-sequence measurements in tracking: exact solution. *Aerospace and Electronic Systems, IEEE Transactions on*, 38(3):769 – 777, jul 2002.

[27] Duncan Smith and Sameer Singh. Approaches to multisensor data fusion in target tracking: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 18:1696–1710, 2006.

[28] Y. Bar-Shalom, M. Mallick, Huimin Chen, and R. Washburn. One-step solution for the general out-of-sequence-measurement problem in tracking. In *IEEE Aerospace Conference Proceedings*, volume 4, pages 4–1551 – 4–1559 vol.4, 2002.

[29] M. Mallick, S. Coraluppi, and C. Carthel. Advances in asynchronous and decentralized estimation. In *IEEE Aerospace Conference Proceedings*, volume 4, pages 4/1873 –4/1888 vol.4, 2001.

[30] Xiaojing Shen, Yunmin Zhu, Enbin Song, and Yingting Luo. Optimal centralized update with multiple local out-of-sequence measurements. *IEEE Transactions on Signal Processing*, 57(4):1551–1562, 2009.

[31] Shuo Zhang and Y. Bar-Shalom. Optimal removal of out-of-sequence measurements from tracks using the if-equivalent measurement. In *49th IEEE Conference on Decision and Control (CDC)*, pages 1312 –1317, dec. 2010.

[32] Andrew Howard, Maja J Mataric, and Gaurav S. Sukhhatme. Putting the i in team: an ego-centric approach to cooperative localization. *International Conference on Robotics and Automation*, 3(1):34–46, 2006.

[33] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun. Collaborative multi-robot exploration. In *IEEE International Conference on Robotics and Automation Proceedings*, volume 1, pages 476 –481 vol.1, 2000.

[34] W. Burgard, M. Moors, C. Stachniss, and F.E. Schneider. Coordinated multi-robot exploration. *IEEE Transactions on Robotics*, 21(3):376 – 386, june 2005.

[35] A. Howard. Multi-robot simultaneous localization and mapping using particle filters. *The International Journal of Robtics Research*, 25(12):1243–1256, 2006.

[36] Regis Vincent, Dieter Fox, Jonathan Ko, Kurt Konolige, Benson Limketkai, Benoit Morisset, Charles Ortiz, Dirk Schulz, and Benjamin Stewart. Distributed multirobot exploration, mapping, and task allocation. *Annals of Mathematics and Artificial Intelligence*, 52(2-4):229–255, April 2008.

[37] Been Kim, Michael Kaess, Luke Fletcher, John Leonard, Abraham Bachrach, Nicholas Roy, and Seth Teller. Multiple relative pose graphs for robust cooperative mapping. In *IEEE International Conference on Robotics and Automation*, 2010.

[38] N. Michael, S. Shen, K. Mohta, V. Kumar, K. Nagatani, Y. Okada, S. Kiribayashi, K. Otake, K. Yoshida, K. Ohno, E. Takeuchi, and S. Tadokoro. Collaborative mapping of an earthquake-damaged building via ground and aerial robots. In *Journal of Field Robotics*, 2012.

[39] H. P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9:61–74, 1988.

[40] Kai M. Wurm, Armin Hornung, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems. In *Proc. of the ICRA 2010 workshop*, 2010.

[41] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 303–312, New York, NY, USA, 1996. ACM.

[42] Andrew J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, pages 1403–, Washington, DC, USA, 2003. IEEE Computer Society.

[43] J. Shi and C. Tomasi. Good features to track. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 593–600, Jun 1994.

[44] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, pages 430–443, 2006.

[45] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008.

[46] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[47] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Proceedings of the 2011 International Conference on Computer Vision*, pages 2548–2555, 2011.

[48] A. Alahi, R. Ortiz, and P. Vandergheynst. FREAK: Fast Retina Keypoint. In *Computer Vision and Pattern Recognition*, pages 510–517, June 2012.

[49] V. Lepetit M. Ozuyal, M. Calonder and P. Fua. Fast keypoint recognition using random ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010.

[50] A. Eliazard and R. Parr. Dp-slam:fast robust simultaneous localization and mapping without predetermined landmarks. *Proceeding of the Intrnational Conference on Artificial Intelligence*, 2003.

[51] A. Eliazard and R. Parr. Dp-slam 2.0. *IEEE International Conference on Robotics and Automation*, 2004.

[52] W. Burgard G. Grisetti, C. Stachniss. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. *Robotics and Automation*, 2005.

[53] W. Burgard G. Grisetti, C. Stachniss. Improved techniques for grid mapping with rao-blackwellized particle filters. *In Robotics IEEE Transactions on Robotics*, 3(1):34–46, 2007.

[54] Tim Bailey and Hugh Durrant-Whyte. Simultaneous Localisation and Mapping (SLAM): Part II State of the Art. *Robotics & Automation Magazine, IEEE*, 13(3):108–117, September 2006.

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

[56] Mark Cummins and Paul Newman. Probabilistic appearance based navigation and loop closing. In *Proc. IEEE International Conference on Robotics and Automation(ICRA'07)*, Rome, April 2007.

[57] Mark Cummins and Paul Newman. Fab-map: Probabilistic localization and mapping in the space of appearance. *Int. J. Rob. Res.*, 27(6):647–665, June 2008.

[58] D. Galvez-Lopez and J.D. Tardos. Real-time loop detection with bags of binary words. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 51–58, Sept 2011.

[59] Frank Dellaert and Michael Kaess. Square root sam: Simultaneous localization and mapping via square root information smoothing. *International Journal of Robotics Reasearch*, 25:2006, 2006.

[60] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Trans. on Robotics (TRO)*, 24(6):1365–1378, December 2008.

[61] Edwin Olson, John J. Leonard, and Seth J. Teller. Fast iterative alignment of pose graphs with poor initial estimates. In *ICRA'06*, pages 2262–2269, 2006.

[62] C. Estrada, J. Neira, and J.D. Tardos. Hierarchical slam: Real-time accurate mapping of large environments. *IEEE Transactions on Robotics*, 21(4):588 – 596, aug. 2005.

[63] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Non-linear constraint network optimization for efficient map learning. *IEEE Transactions on Intelligent Transportation Systems*, 2009.

[64] Giorgio Grisetti, Rainer Kmmerle, Cyrill Stachniss, Udo Frese, and Christoph Hertzberg. Hierarchical optimization on manifolds for online 2d and 3d mapping. In *Proc. IEEE International Conference on Robotics and Automation*, 2010.

[65] Kurt Konolige, Giorgio Grisetti, Rainer Kmmerle, Wolfram Burgard, Benson Limketkai, and Rgis Vincent. Efficient sparse pose adjustment for 2d mapping. In *IROS*, pages 22–29. IEEE, 2010.

[66] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A General Framework for Graph Optimization. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.

[67] Giorgio Grisetti, Rainer Kmmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intell. Transport. Syst. Mag.*, 2(4):31–43, 2010.

[68] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

[69] Hokuyo. http://www.hokuyo-aut.jp/.

[70] Sick. http://www.sick.com.

[71] Velodyne Lidar. http://velodynelidar.com.

[72] Davide Scaramuzza Christian Forster, Matia Pizzoli. Fast semi-direct monocular visual odometry. In *IEEE International Conference on Robotics and Automation*, 2014.

[73] Andrew J. Davison, Ian D. Reid, Nicholas D. Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(6):1052–1067, June 2007.

[74] Kenji Kaneko, Fumio Kanehiro, Shuuji Kajita, Hirohisa Hirukawa, Toshikazu Kawasaki, Masaru Hirata, Kazuhiko Akachi, and Takakatsu Isozumi. Humanoid robot hrp-2. In *IEEE Int. Conf. Rob. Aut*, pages 1083–1090, 2004.

[75] Javier Civera, Andrew J. Davison, and J. M. Mart 鎯 nez Montiel. Unified inverse depth parametrization for monocular slam. In *Proceedings of Robotics: Science and Systems*, 2006.

[76] Javier Civera, Andrew J. Davison, and J. M. M Montiel. Inverse depth parametrization for monocular slam. In *IEEE transactions on robotics*, 2007.

[77] Ethan Eade and Tom Drummond. Scalable monocular slam. In *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 1*, CVPR '06, pages 469–476, Washington, DC, USA, 2006. IEEE Computer Society.

[78] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that probably converges. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI)*, Acapulco, Mexico, 2003. IJCAI.

[79] Willlow Garage. www.willowgarage.com.

[80] Ethan Eade and Tom Drummond. Monocular slam as a graph of coalesced observations. In *Proc. 11th IEEE International Conference on Computer Vision*, Rio de Janeiro, Brazil, October 2007.

[81] Georg Klein and David Murray. Improving the agility of keyframe-based SLAM. In *Proc. 10th European Conference on Computer Vision (ECCV'08)*, pages 802–815, Marseille, October 2008.

[82] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry: Part i the first 30 years and fundamentals. *IEEE Robotics and Automation Magazine*, 18(4), 2011.

[83] Friedrich Fraundorfer and Davide Scaramuzza. Visual odometry: Part ii: Matching, robustness, optimization, and applications. *Robotics & Automation Magazine, IEEE*, 19(2):78–90, 2012.

[84] Richard A. Newcombe, S.J. Lovegrove, and A.J. Davison. Dtam: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2320–2327, Nov 2011.

[85] T. Schöps, J. Engel, and D. Cremers. Semi-dense visual odometry for AR on a smartphone. In *ismar*, September 2014.

[86] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *eccv*, September 2014.

[87] Andreas Nuchter, Kai Lingemann, Joachim Hertzberg, and Hartmut Surmann. 6d slam 3d mapping outdoor environments: Research articles. *J. Field Robot.*, 24(8-9):699–722, August 2007.

[88] T. Liu, M. Carlberg, G. Chen, J. Chen, J. Kua, and A. Zakhor. Indoor localization and visualization using a human-operated backpack system. In *International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–10, Sept 2010.

[89] George Chen, John Kua, Stephen Shum, Nikhil Naikal, Matthew Carlberg, and Avideh Zakhor. *Indoor Localization Algorithms for a Human-Operated Backpack System.*

[90] J. Sturm, E. Bylow, F. Kahl, and D. Cremers. CopyMe3D: Scanning and printing persons in 3D. In *German Conference on Pattern Recognition (GCPR)*, September 2013.

[91] Kai Berger, Stephan Meister, Rahul Nair, and Daniel Kondermann. A state of the art report on kinect sensor setups in computer vision. In *Time-of-Flight and Depth Imaging. Sensors, Algorithms, and Applications*, 2013.

[92] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgbd mapping: Using depth cameras for dense 3d modeling of indoor environments. In *In RGB-D: Advanced Reasoning with Depth Cameras Workshop in conjunction with RSS*, 2010.

[93] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *Int. J. Rob. Res.*, 31(5):647–663, April 2012.

[94] David Nistr and Henrik Stewnius. Scalable recognition with a vocabulary tree. In *CVPR*, pages 2161–2168, 2006.

[95] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 335–342, 2000.

[96] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the rgb-d slam system. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1691–1696, May 2012.

[97] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard. 3-d mapping with an rgb-d camera. *Robotics, IEEE Transactions on*, 30(1):177–187, Feb 2014.

[98] Albert S. Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. Visual Odometry and Mapping for Autonomous Flight Using an RGB-D Camera. In *Int. Symposium on Robotics Research (ISRR)*, Flagstaff, Arizona, USA, August 2011.

[99] J. St端ckler and S. Behnke. Integrating depth and color cues for dense multi-resolution scene mapping using rgb-d cameras. In *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems (MFI)*, 2012.

[100] Jorg Stuckler and Sven Behnke. Multi-resolution surfel maps for efficient dense 3d modeling and tracking. In *Journal of Visual Communication and Image Representation*, 2013.

[101] Tommi Tykkala, Andrew I. Comport, and Joni-Kristian Kamarainen. Photorealistic 3d mapping of indoors by RGB-D scanning process. In *2013 IEEE/RSJ*

International Conference on Intelligent Robots and Systems, Tokyo, Japan, November 3-7, 2013, pages 1050–1055, 2013.

[102] F. Steinbrucker, J. Sturm, and D. Cremers. Real-time visual odometry from dense rgb-d images. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 719–722, Nov 2011.

[103] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for rgb-d cameras. In *icra*, May 2013.

[104] C. Kerl, J. Sturm, and D. Cremers. Dense visual slam for rgb-d cameras. In *Proc. of the Int. Conf. on Intelligent Robot Systems (IROS)*, 2013.

[105] F. Steinbruecker, C. Kerl, J. Sturm, and D. Cremers. Large-scale multi-resolution surface reconstruction from rgb-d sequences. In *iccv*, Sydney, Australia, 2013.

[106] 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 *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 559–568, New York, NY, USA, 2011. ACM.

[107] Ming Zeng, Fukai Zhao, Jiaxiang Zheng, and Xinguo Liu. Octree-based fusion for realtime 3d reconstruction. *Graph. Models*, 75(3):126–136, May 2013.

[108] T. Whelan, M. Kaess, M.F. Fallon, H. Johannsson, J.J. Leonard, and J.B. McDonald. Kintinuous: Spatially extended KinectFusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Sydney, Australia, Jul 2012.

[109] T. Whelan, M. Kaess, J.J. Leonard, and J.B. McDonald. Deformation-based loop closure for large scale dense RGB-D SLAM. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, IROS*, Tokyo, Japan, November 2013.

[110] T. Whelan, M. Kaess, H. Johannsson, M.F. Fallon, J.J. Leonard, and J.B. McDonald. Real-time large scale dense RGB-D SLAM with volumetric fusion. *Intl. J. of Robotics Research, IJRR*, 2014.

[111] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers. Real-time camera tracking and 3d reconstruction using signed distance functions. In *Robotics: Science and Systems Conference (RSS)*, June 2013.

[112] H. Strasdat, J.M.M. Montiel, and A.J. Davison. Real-time monocular slam: Why filter? In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2657–2664, May 2010.

[113] C. Forster, M. Pizzoli, and D. Scaramuzza. Air-ground localization and map augmentation using monocular dense reconstruction. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 3971–3978, Nov 2013.

[114] Matthias Faessler, Flavio Fontana, Christian Forster, Elias Mueggler, Matia Pizzoli, and Davide Scaramuzza. Autonomous, vision-based flight and live dense 3d mapping with a quadrotor micro aerial vehicle. *Journal of Field Robotics*, 2015.

[115] Kourosh Khoshelham and Er Oude Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. In *Sensors 2012, 12, 14371454. 2013*, page 8238.

[116] Youssef Ktiri. Multiple humanoid robots based cooperative system for simultaneous localization, mapping and target search in unknown environments. In *Masters Thesis*, 2012.

[117] Gerda Kamberova and Ruzena Bajcsy. Sensor errors and the uncertainties in stereo reconstruction. In *Empirical Evaluation Techniques in Computer Vision*, pages 96–116. IEEE Computer Society Press, 1998.

[118] A. Jaakkola, S. Kaasalainen, H. Niittym 辰 ki, and A. Akuj 辰 rvi. Intensity calibration and imaging with swissranger sr-3000 range camera.

[119] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *Int. J. Comput. Vision*, 13(2):119–152, October 1994.

[120] D. Holz and S. Behnke. Sancta simplicitas - on the efficiency and achievable results of slam using icp-based incremental registration. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1380–1387, May 2010.

[121] E.B. Olson. Real-time correlative scan matching. In *IEEE International Conference on Robotics and Automation*, pages 4387 –4393, may 2009.

[122] Dirk Hähnel and Wolfram Burgard. Probabilistic matching for 3D scan registration. In *In.: Proc. of the VDI - Conference Robotik 2002 (Robotik*, 2002.

[123] S. Kohlbrecher, J. Meyer, O. von Stryk, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, Kyoto, Japan, November 1-5 2011.

[124] Marius Muja and David G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *In VISAPP International Conference on Computer Vision Theory and Applications*, pages 331–340, 2009.

[125] Fran 巽 ois Pomerleau, Francis Colas, Roland Siegwart, and St 迦 phane Magnenat. Comparing icp variants on real-world data sets. *Autonomous Robots*, 34(3):133–148, 2013.

[126] Siegwart R. Elseberg J., Magnenat S. and N 端 chter A. Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration. *Software Engineering for Robotics*, 2012.

[127] J. Engel, J. Sturm, and D. Cremers. Semi-dense visual odometry for a monocular camera. In *iccv*, Sydney, Australia, December 2013.

[128] Vincent Lepetit and Pascal Fua. Monocular model-based 3d tracking of rigid objects: A survey. In *Foundations and Trends in Computer Graphics and Vision*, pages 1–89, 2005.

以上

1p～ 139p 完

博士論文 (要約)

平成27年12月15日提出

知能機械情報学専攻
48127512 カチリ ユセフ