RGB-D Handheld Mapping and Modeling

Alexandru - Eugen Ichim

École Polytechnique Fédérale de Lausanne

School of Computer and Communications
Master of Science

Supervisors:

Prof. Dr. Mark Pauly
Dr. Radu Rusu

Lausanne, EPFL, 2013

The voyage of discovery is not in seeking new landscapes but in having new eyes.
– Marcel Proust

To my parents...

Contents

List of Figures

List of Tables

List of terms

Below is a list of terms and acronyms that will be used extensively throughout this document.

- **Point cloud** - a set of vertices in 3D space, defined by their X, Y, Z coordinates; additional properties such as normal direction, curvature, colors, or per-point feature descriptors can be attached to each point.
- **Point cloud alignment** - the process of bringing together two point sets P and Q such that the error (under various error metrics) between corresponding point pairs (p, q) is minimized.
- **Point cloud registration** - see *point cloud alignment*.
- **Pixel** - also called *pel* or *picture element*, is the smallest element in a sensor array or display, an addresable sample of the continous function captured by the sensor or displayed by the screen.
- **Dexel** - instead of associating gray or color values to a pixel, certain applications need to include depth information in a raster, thus introducing the concept of a *depth pixel*.
- **Moving Least Squares (MLS)** - an approach for reconstructing continuous functions from unorganized samples [Lev98], used in Computer Vision for inferring continuous surfaces from discrete scanned vertices.
- **Random Sample Consensus (RANSAC)** - a non-deterministic iterative algorithm for estimating the parameters of a mathematical model from a set of samples containing outliers [Pul99].
- **Iterative Closest Point (ICP)** - an algorithm for registering two point clouds, first introduced by [BM92], which has been heavily optimized throughout the years [RL01].
- **Pose graph** - a graph representation of the track of a robot in space. The nodes are the poses of the robot at different instances in time, connected together by edges which represent noisy measurements of relative transformations between the poses. The graph is translated into an error function that is minimized during a process called graph optimization.

- **Simultaneous Localization and Mapping (SLAM)** - this problem requires a robot that is placed at an unknown location in an unknown environment to be able to build a map, to localize itself in the map and update it in the process.
- **Mean squared error (MSE)** - estimator to compute the error between two sets of corresponding values, which corresponds to the expected value of the squared error loss:

$$MSE(A,B) = \frac{1}{n}\sum_{i=1}^{n}(a_i - b_i)^2 \tag{1}$$

- **Root mean squared error (RMSE)** - estimator similar to *MSE*, only that it has the same unit of measurement as the initial quantity. It is equivalent to the standard deviation for unbiased estimators:

$$RMSE(A,B) = \sqrt{MSE(A,B)} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(a_i - b_i)^2} \tag{2}$$

1 Introduction

In the early days of robotics, perception was consistently confounded with artificial intelligence. People thought that a robot is *smart* if it can correctly perceive the world around it and adapt its actions to the way the environment is structured and evolves. In order to enable mobility, robots need a good indication of how the world around them looks like, and, depending on the task to be performed, the complexity and coverage of the state of the world varies from simple one-dimensional signals to full-fledged 3D models with information such as color, surface orientation and semantic information of what the seen objects are, areas where the robot can access safely etc.

As such, one of the most important and sought for applications in computer vision is Simultaneous Localization and Mapping (SLAM). The two concepts are close to a chicken-and-egg situation, as the mobile operator equipped with a camera in an unknown environment has to map the scene, but can not do so before localizing itself, and good localization needs a map in the first place. Mapping is needed especially in robotics to aid in tasks ranging from 2D navigation to object modeling and planning for complex manipulation tasks. The problem of 2D SLAM has been considered solved for some time, and there have been multiple systems published with code available as open source [QCG*09]. Adding another degree of freedom does not make the naive adoption of the successful techniques from 2D a viable solution.

Outside the robotics field, mapping can be transposed to a mean of digital communication, allowing people to capture, analyze and share models of their environments, be it for scientific, engineering or entertainment purposes. Downscaling the problem leads to object modeling, a concept that opens a new space of possibilities with applications in computer-generated animations, gaming and 3D printing.

Until recently, researchers and engineers focused their attention mostly on industrial applications of 3D perception, as the hardware necessary for good sensing was not financially viable for the mass market. Stereo cameras and structure from motion were the most accessible options to provide end-users with 3D capturing and processing capabilities. Both options have disadvantages that proved fatal for making such systems popular (high computation

Figure 1.1: Typical jumps in the exposure settings of RGB-D cameras - the color of the wall changes from white to pitch black in just a few frames.

costs, cumbersome hardware, relatively low robustness etc.). Microsoft introduced the Kinect as a controller-free gaming interface, but due to its low price and both 3D and 2D capabilities it became very popular amongst the computer vision and robotics communities. This launched the idea that 3D can be accessible and caused an acceleration in the development of robust, user-friendly computer vision applications.

Being the first of its breed, the Microsoft Kinect [Pri] comes with a pack of limitations: the quality of the data leaves much to be desired compared to its more expensive counterparts, featuring various artifacts that are hard to deal with. Furthermore, data can not be captured in certain conditions (the camera is based on stereo with an near infrared projector and an IR sensor, which make it very sensitive to external IR sources such as the sun, reflective or dark surfaces). Also, the hardware is still relatively bulky. All the issues are alleviated by the very low price point and the promise of considerable improvements in the next iterations of the product.

1.1 Depth-only Registration

The system that is going to be presented in this thesis uses only the depth images produced by the sensor. The main reason for using only the depth information of the RGB-D sensors is that the RGB information captured with cheap depth and color cameras varies a lot under changing lighting conditions. As there is no direct control on the exposure of the camera (it defaults to auto-exposure), the device tends to make sudden jumps between exposure levels (see Figure 1.1). This makes it nearly impossible for RGB features to be stable and repeatable and not cause the loss of the camera track. It becomes especially problematic when scanning entire rooms, as it is inevitable that the camera will be pointed to a strong source of light (i.e., window, light bulb). In addition, we aim at developing a system that is independent from the lighting conditions and repeatable at any time of the day, under natural or artificial light.

Color information is not reliable under heavy noise (in low-light conditions, the ISO settings

of the camera are increased, causing a lot of static noise) and under motion blur. Furthermore, RGB-D cameras do not work outdoors because of the presence of strong infrared lighting from the sun. An option is to scan after sunset or in shady areas, straining the color sensor a lot or not capturing any color at all. This means that one can only rely on depth for such cases. The problem of localization can involve having a map and attempting to localize in it at different times of the day, under very different lighting situations - the geometry will be constant, but the colors will differ heavily.

One can argue that color texture is ubiquitous in household and office settings, but those are very repetitive (i.e., carpets with repeating patterns, wooden furniture etc.). This property causes a lot of false positives when trying to match frames with possibly unknown initial poses. On the contrary, geometry is a lot less repetitive in these conditions, leading to more confident matches of 3D features.

The last reason for focusing only on depth data for this thesis is that in the last couple of years there have been a number of scientific publications presenting RGB-D mapping, but none of them became the definitive solution to the simultaneous localization and mapping in 3D. We find that their approaches combining RGB and depth in rather incoherent ways, as the systems seem to be mere engineering attempts at combining known approaches from 2D computer vision and 3D registration, without looking closely into the particularities of this new generation of sensors. As such, we will thoroughly explore the possibilities and limitations of the Z-information generated by cheap 3D cameras.

1.2 Similar Systems

The ubiquity of cheap RGB-D cameras attracted the attention of computer vision researchers in the last years, resulting in numerous publications tackling the mapping problem in various ways. These systems are split into two categories:

- *incremental* - the camera is tracked in a progressive fashion, accumulating errors from frame to frame, and never adjusting past pose estimations.
- *the ones that perform loop closure* - the difference from the above category is that previous estimations are adjusted as new information about the scene is retrieved by the camera.

Microsoft Research propose a system called Kinect Fusion [IKH*11], which falls into the first group. The authors do incremental registration using only depth data with the incoming frame against a depth map obtained from a rendering under the same camera parameters of a truncated signed distance function. This global data structure is updated with the new point cloud and its estimated pose, and the process is repeated for the next scan. The pipeline was implemented only for GPUs, making use of highly parallel operations such as bilateral filtering

Figure 1.2: Augmented reality with Kinect Fusion.

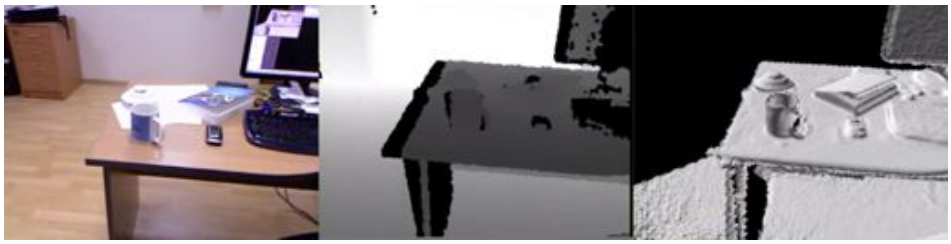


Figure 1.3: From left to right: the input RGB image, the input depth image, the reconstructed mesh with Kinect Fusion as seen from the current camera pose. Notice the fact that the mesh is not complete on the left side, as the volume of the scanned scene is directly proportional to the amount of memory on the graphics card. This results in being limited to desktop-size volumes with currently available hardware.

and depth map renderings. This allows the algorithm to run at 30 Hz on capable hardware, opening up possibilities for real-time applications such as augmented reality (Figure 1.2). However, the system has fundamental flaws that make it unreliable for mapping of indoor environments larger than a desk. First of all, the camera tracking is done in an incremental fashion, making it impossible for the system to fix past misalignments. This means that if the camera track is lost, the scanning process has to be completely reset and started from scratch. Furthermore, drift will be accumulated, and no loop closure techniques are employed, making it difficult to robustly scan complete objects or entire rooms. Another issues is that the TSDF requires considerable memory and is stored on the GPU. This causes a trade-off between between the volume to be scanned and the level of detail of the reconstruction. Positioning the starting pose of the camera track inside the volume is a tricky choice. The system simply stops adding new areas to the TSDF if the limit of the volume is reached (Figure 1.3). However, the TSDF allows for quick mesh extraction because the structure is ready to be used in a marching cubes algorithm at any stage of the reconstruction.

Kintinuous [WJK*12] overcomes some of the limitations of Kinect Fusion. The first improvement is the implementation of the truncated signed distance function as a cyclical buffer, allowing for larger scenes to be scanned. Once the limit of the volume is reached, the TSDF is moved, making space for the new areas of the scene. The discarded data is moved to the much larger main memory of the computer and eventually dumped to disk. In addition to the depth-only ICP used for camera tracking, the authors experiment with various combinations

of visual odometry and ICP. In addition, color information is integrated in the TSDF, and the output meshes have per-vertex coloring. Although this system still does not do explicit loop closure, the quantitative results are encouraging.

The framework that is common in some publications [HKH*10], [SEE*12], [PKDB10] is that of using sparse RGB features in combination with dense cloud registration forming constraints that are optimized in a pose graph, making up the second category of RGB-D mapping systems. [HKH*10] use both the SIFT feature correspondences and the dense XYZ subsampled point to plane correspondences in a joint Levenberg-Marquardt optimization. The optimizer is initialized by a RANSAC step with the color features or by using a constant velocity assumption in case the previous method fails. [SEE*12] compute the transformation between two frames with RANSAC on SURF correspondences, followed by a refinement step using the dense point to plane correspondences. Both approaches compute the normals by principal component analysis in a local spherical neighborhood for each point. [HKH*10] stop the minimizer after a fixed number of iterations or when the error does not change significantly from one iteration to the next. While [SEE*12] do additional pairwise registration between the current frame and past frames uniformly sampled from the timeline, [HKH*10] define the concept of keyframe as a frame that does not have sufficient feature matches with the previously labelled keyframe. This new keyframe is then registered against all the other keyframes detected so far. Both methodologies use graph optimizers to relax the errors (TORO [GSB] and G2O [KGS*11]). The results presented by the authors are very promising and we build our work on the ideas suggested by this research.

All the systems presented so far perform fully automatic registration with some amount of feedback to the user in the process. Du et al. [DHR*11] claim that naturally all RGB-D mapping systems will fail due to far from perfect robustness or due to mishandling by the operator (for example, by moving the camera too fast thus causing blur or large frame to frame displacements, getting too close or too far from the object of interest, pointing the camera at surfaces that are not registrable due to lack of geometry or texture etc.). In order to alleviate the difficulties that a system faces under hard conditions, the authors suggest a set of ways in which the user is asked to intervene in the pipeline during acquisition, and not only at the post-processing stage as most systems do. The interactive mapping approach will alert the user if the incremental registration failed, stopping the processing and asking the user to physically return to the area of the scene where the last successfully registered frames were collected. The user receives information about the current state of the scanned model as a top-down view of the environment with the unknown areas highlighted. To simplify the work of the graph optimizer, loop closures are done under the supervision of the operator. Instead of automatically deciding if a loop is to be closed, the system presents the user with the most plausible pairs of frames that produce loop closures in the current map.

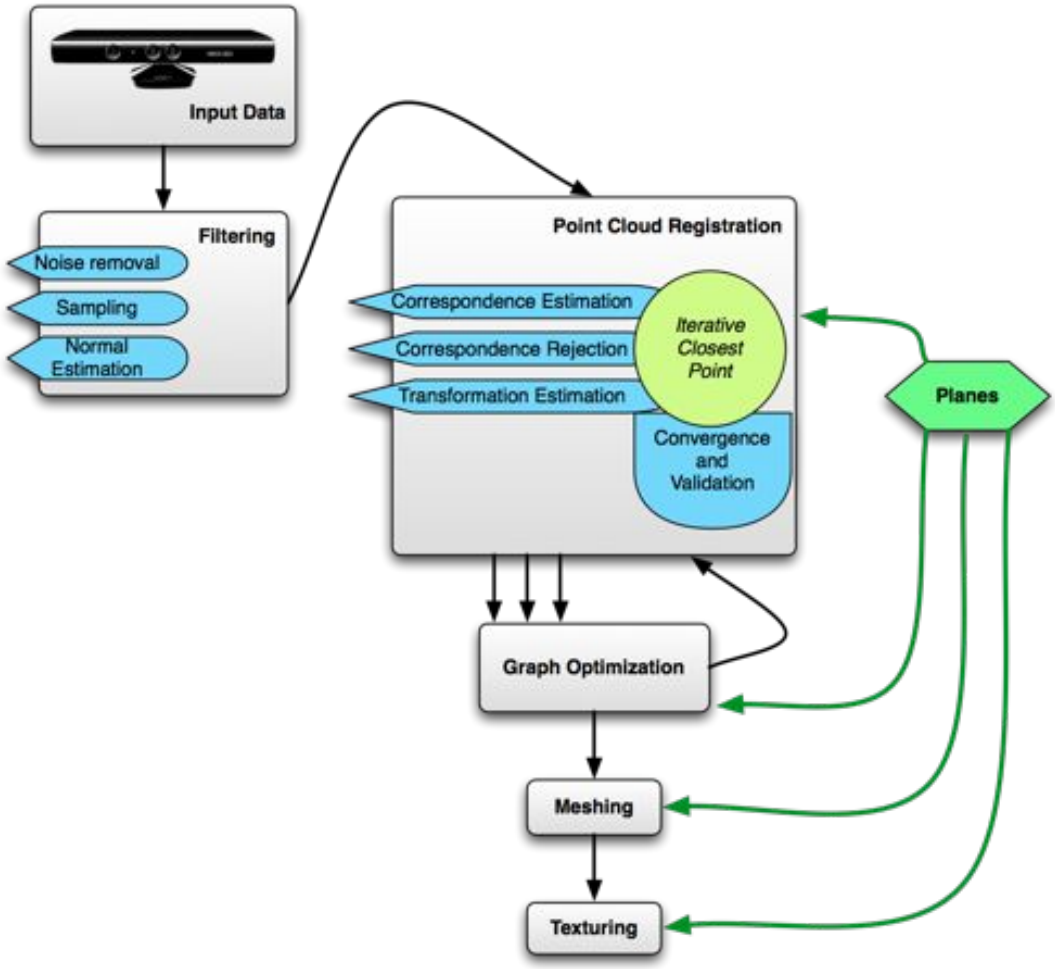


Figure 1.4: The proposed pipeline overview.

1.3 Pipeline Overview and Thesis Organization

An overview of the pipeline that is proposed is shown in Figure 1.4. Throughout this thesis, each component of the system is presented in detail and benchmarked against other state-of-the-art options, motivating the choices we made at each step. As mentioned in a previous section, we limit ourselves to using only the depth of the Kinect. Section 2.2 shows ways in which the data can be cleaned, enhanced and subsampled for maximum efficiency. A large portion of our discussions is centered on the pairwise point cloud registration, explaining all the steps and possible enhancements, covered in Chapter 2. Graph optimization techniques are presented in Chapter 3. Surface reconstruction techniques and different ways of adding color to the meshes for more realistic results are discussed in Chapter 4. Chapter 5 presents how planar features can be used in conjunction with the proposed system in order to enhance the output at various stages of the pipeline. Chapter 6 shows results, comparisons with other systems and proposals for future work.

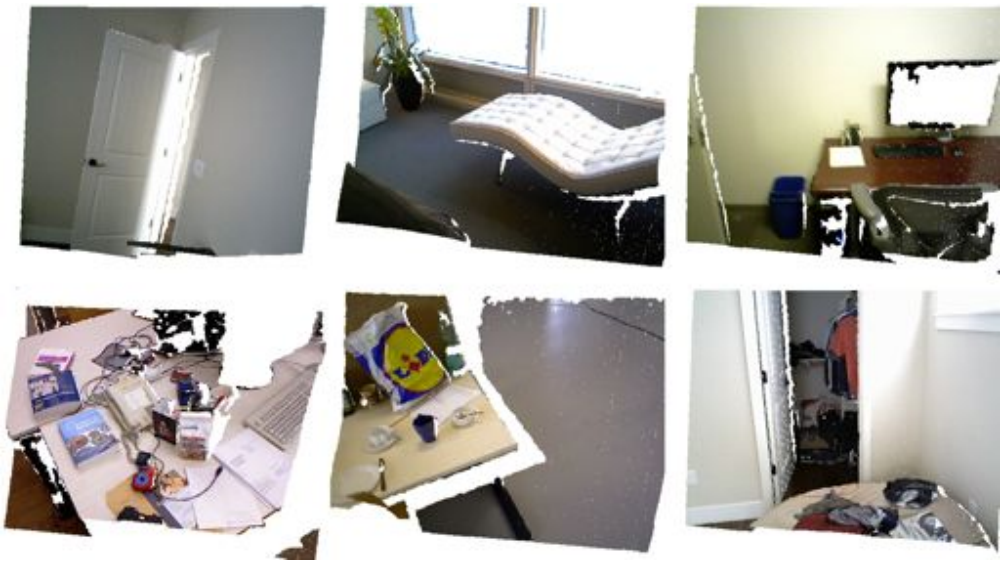


Figure 1.5: Some sample datasets that are going to be used in the thesis.

1.4 Datasets

The work that is presented in the following pages concentrates on RGB-D sensors developed by PrimeSense [Pri]. These are commercialized as the Microsoft Kinect (both the *for XBox* and *for Windows* versions) [kin] and the Asus Xtion Pro [asu]. These devices were advertised as gaming controllers and quickly became attractive to researchers because of their accessible prices and good price to quality ratio. Another point of interest is their ubiquity in households (Microsoft claims 18 million units (Kinect) sold as per January 2012), so developing new software guarantees a large market.

The data is captured by moving the camera unconstrained in static indoor environments. We scanned offices and rooms, as well as individual objects, cluttered desktops etc. (see Figure 1.5 for a few samples). An important source of datasets apart from the ones collected by ourselves is the collection provided by the Computer Vision Group at Technische Universität München [SEE*12]. The sequences they offer for free for research purposes on their website present very specific motions (only rotation, only translation, combined), recording patterns (without any loop closures, with multiple small loop closures, with few large loop closures), various scenes (with or without any color texture, rich or poor in geometrical details, large open spaces, tight cluttered environments etc.). Apart from the advantage of being able to use the same data as other perception researchers across the world, these datasets come with ground truth poses. These are grabbed from a high-accuracy motion-capture system able to provide absolute transformations with a claimed average error of 1 cm and 0.5 degrees [SEE*12]. The pose information is not synchronized with the Kinect frames, which also come in a different format than the one used in our application.

In order to benchmark some steps in our pipeline, we needed scans without the inherent

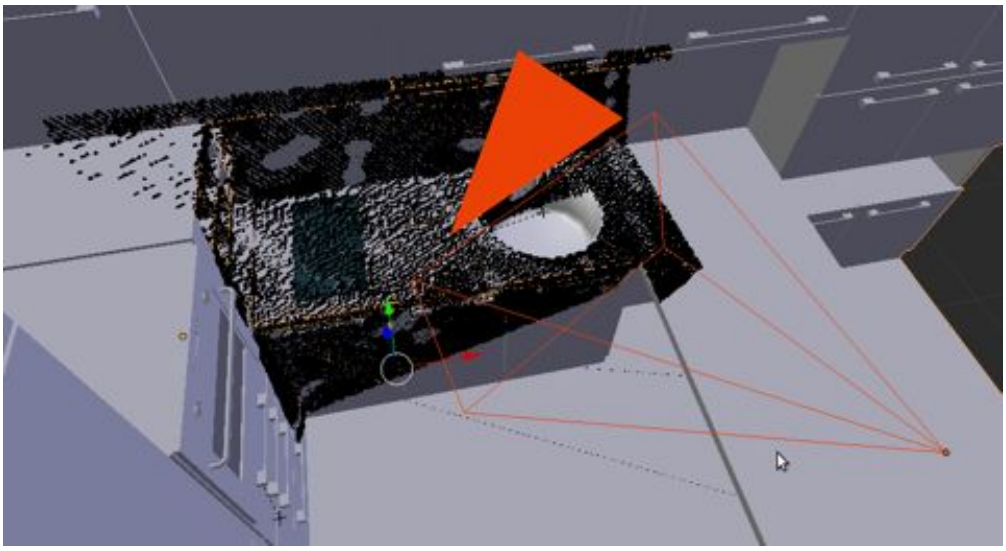


Figure 1.6: The Blensor software interface for generating simulated RGB-D scans.

noise of the sensor. Some researchers solve this by scanning objects for which they have CAD models and then fit these models in the scans. We opted for a simpler, but less realistic method - that of simulating the scans. For this purpose, we employed Blensor [GKUP11], which is built on top of the popular Blender modeling and animation software and can generate scans that simulate the ones of various commercial sensors in artificial scenes (see Figure 1.6).

1.5 Open Source Software

The implementation of the pipeline presented in this thesis would not have been possible without open source software. We extensively used libraries such as: Boost [boo], Visualization Toolkit (VTK) [SML], Eigen [GJ*10], FLANN [ML09], Point Cloud Library (PCL) [RC11]. This is because the concepts we work with are so extensive, that implementing everything from scratch would be highly unfeasible. As Linus Torvalds pointed out: "given enough eyeballs, all bugs are shallow", successful open source software is based on large communities of active developers with diverse backgrounds. This highly increases the probability of the implementations being and correct and well optimized, which is difficult with a one-person effort.

The Point Cloud Library is the main building block for the work done in this thesis and most of the concepts and novelties presented in the following chapters are contributed back to the library. That is why we dedicate a section to explain what the library has to offer.

PCL is a large scale, open source project for 2D/3D image and point cloud processing. The code is divided into multiple modules (see Figure 1.7) that cover subjects such as filtering, registration etc., each one containing BSD-licensed implementations of state-of-the-art computer vision algorithms. The library also comes with extensive visualization tools and support

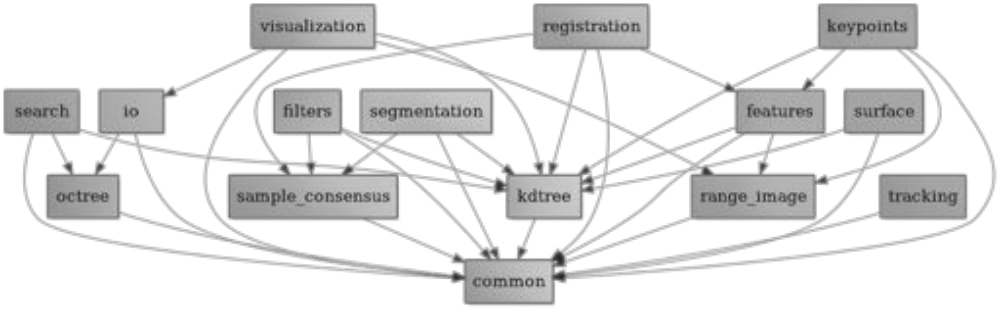


Figure 1.7: The dependency graph of the Point Cloud Library modules.

for popular sensors such as the Microsoft Kinect or the Velodyne laser scanners.

The code is contributed by more than 500 developers from across the world, ranging from hobbyists and artists to engineers and researchers. The popularity of the library is confirmed by the over 100 million hits on the domains, out of which more than half a million are unique visitors. It is hosted and maintained by its own non-profit foundation, Open Perception, which is currently catering for the commercial and academic success of the project.

The sub-modules cover the following topics:

pcl_io includes classes and methods for loading and saving ppint clouds and polygonal meshses in various popular file formats (e.g., .pcd, .vtk, .obj, .ply). It also contains wrappers on top of drivers for sensing devices such as the PrimeSense RGB-D cameras.

pcl_search provides a framework for fast search algorithms in both organized and unorganized point clouds. In the case of organized, it uses the grid structure to speed up the search. The searches in general point clouds are performed using kd-trees, aided by the FLANN library [ML09]. All the classes allow for *k nearest neighbor* and *radius search* methods.

pcl_octree hierarchical tree structure for storing and processing point sets.

pcl_filters covers outlier and noise removal methods, sampling strategies based on point distribution, normal distribution or covariances etc.

pcl_features data structures and algorithms for extracting 2D color features, 3D geometrical features, and combinations of both from images and point clouds. These include but are not limited to: BRISK, PFH, FPFH, PPF etc.

pcl_keypoints implementations of interest point detection in point clouds; i.e., points that are stable, distinctive and repeateable across views of the same scene/object. These can be used in combination with feature descriptors in order to efficiently match point clouds.

pcl_segmentation contains algorithms for segmenting a point cloud into distinct clusters based on some criteria. This can be useful for separating differnts parts of a cloud in the semantic sense, or just for ease of further processing.

pcl_sample_consensus the module is based on sample consensus methods such as RANSAC and other of its derivatives, and contains models for isolating structures such as spheres, cylinders, planes, or arbitrary shapes in unorganized point clouds.

pcl_surface methods for reconstructing surfaces from their sampled representation. This includes meshing approaches, local polynomial fitting (for re-sampling) etc.

pcl_registration contains components (correspondence estimation and filtering, transformation estimation etc.) for registering point sets together in iterative (ICP) or randomized frameworks (RANSAC).

pcl_visualization built to allow for quick prototyping and visualizing algorithmic results. It has support for viewing point cloud an mesh data in various ways, along with additional propertires such as normal information, feature descriptors and so on.

pcl_range_image contains classes for representing and working with range images.

1.6 Contributions

The contributions brought by the work presented in this thesis include:

- thorough exploration and analysis of how state-of-the-art point cloud registration techniques, most of which where originally introduced in uncolored laser scanning scenarios, can be adapted to the particularities of the depth maps of novel RGB-D cameras: data organized in a pixel/dexel grid with pinhole camera projective properties, but with large amounts of noise in various forms.
- a complete system for RGB-D SLAM using advanced graph optimization technique
- methods for assigning colors to arbitrary meshes that result from our mapping system.
- local geometric feature descriptors, in particular planar shapes, as an efficient and robust mean of enhancing the pipeline:
  - point cloud registration with and without initial aligment
  - landmarks in the scene for pinning the graph optimizer
  - using indoor environment heuristics for improving the quality of the final model by adding plane-to-plane constraints.

2 Point Cloud Registration

2.1 Introduction and Related Work

In this chapter, the methodology for registering a pair of point clouds is presented, as well as the reasoning behind the design decisions that led to the final computational graph of the system. Rusinkiewicz summarizes the optimizations that were proposed over the years in his 2001 publication [RL01]. An analysis is performed on the available options at each step of the process: *selection of points*, *matching*, *pair weighting*, *pair rejection*, *error metrics*, *minimization of the error metric*. The authors use a standard pipeline as baseline for their comparisons and just plug in the various methods in each of the steps, comparing the convergence rate for each situation. Starting from the baseline, [Pul99]:

- random sampling of both input clouds
- closest point correspondence, filtering out pairs with normals more than 45° apart
- constant weighting of point pairs
- point-to-plane error metric solved with the classic *select-match-minimize* iteration

they reach the conclusion that the best combination in terms of speed would be:

- projection-based point pairing
- point-to-plane error metric with *select-match-minimize* iterations
- all the other stages did not seem to considerably influence the convergence rates, so the best in terms of speed are:
  - random sampling
  - constant pair weighting

– distance threshold for rejecting pairs

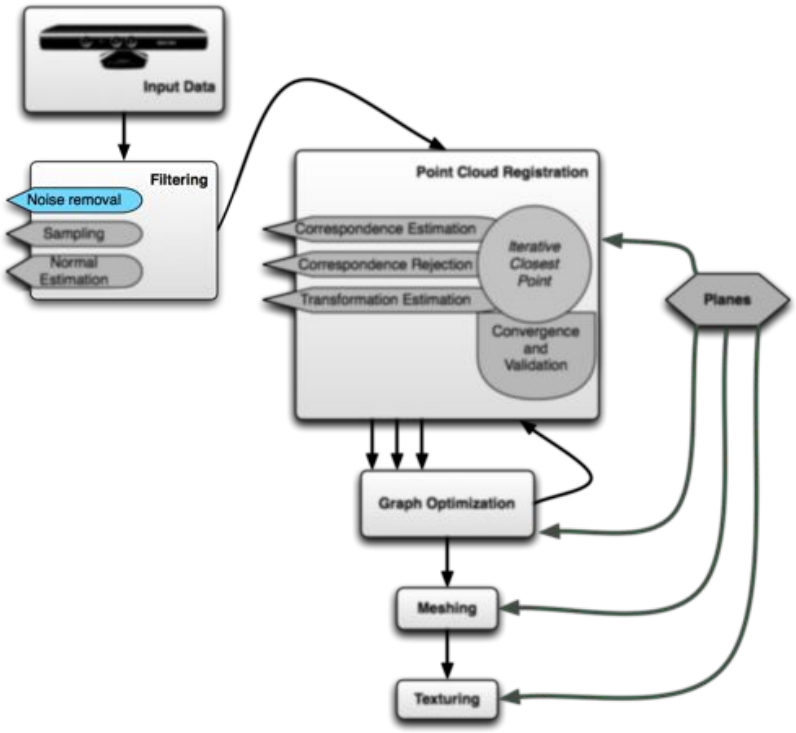
By starting from their conclusions, in the following sections we look into additional methods of improving the registration pipeline. First, in Section 2.2, possible pre-processing steps will be presented. Next, correspondence estimation and rejection is detailed in Section 2.3. Finally, transformation estimation methods including ways of weighting point pairs are evaluated in Section 2.4.

Numerous researchers [DRT*04], [PMC*11] conclude that no fixed parameters are good for an ICP pipeline, and that they have to be adaptive or tuned for a specific application, taking into consideration the expected motion (handheld camera, mounted on a robot, vibrations), sensor characteristics (frequency, noise, field of view), the environment particularities (planarity, distance of objects from the sensor etc.).

2.2 Point Cloud Pre-processing

In this section, noise filtering for Kinect data will be presented, then state-of-the-art sampling procedures, and efficient ways of computing per-point normals for organized point clouds.

2.2.1 Filtering



Before proceeding into using the Kinect data, a filtering steps needs to be done in order to

(a) (b) (c)

Figure 2.1: Various types of noise RGB-D cameras produce: (a) noise on a flat surface; (b) quantization effects; (c) missing pixels.

alleviate the various types of noise that this type of sensor produces (see Figure 2.1). There are multiple ways in which noise could be efficiently eliminated from general point cloud data [Rus], but none of which take into account the difficult quantization effects seen in this new type of data. Nguyen et al. [NIL12] devise noise models for both the lateral and the axial noise of the Kinect, as in Equations 2.1 and 2.2, where p_x is the pixel size and f_x is the focal length of the depth camera.

$$\begin{aligned} \sigma_L(\theta)[px] &= 0.8 + 0.035 * \frac{\theta}{\pi/2-\theta} \\ \sigma_L(\theta)[m] &= \sigma_L(\theta)[px] * z * p_x/f_x \end{aligned} \tag{2.1}$$

$$\sigma_z(z,\theta) = 0.0012 + 0.0019 * (z-0.4)^2, \text{ when } 10^\circ \leq \theta \leq 60^\circ \tag{2.2}$$

The authors claim that the coefficients in the equations were found manually. As such, the lateral noise is linear with depth, and the depth noise is quadratic with respect to depth, as concluded by multiple other authors in similar publications. But this model, like other proposed models, still does not take into consideration the quantization effects.

Next, we will look into three noise removal methods: *median filter*, *bilateral filter*, *moving least squares fitting*.

The median filter [Tuk77] is one of the simplest and wide-spread image processing filters. It is known to perform well with "shot"/impulse noise (some individual pixels having extreme values), it does not reduce contrast across steps in the function (as compared to filters based on averaging), and it is robust to outliers. Furthermore, it is simple to implement and efficient, as it requires a single pass over the image. It consists of a moving window of fixed size that replaces the pixel in the center with the median inside the window.

The bilateral filter is a non-linear filter that smooths a signal while preserving strong edges, and it has been successfully used in computer vision and computer graphics. Equation 2.3 shows how the center pixel of a moving window of fixed size is updated. p is the current pixel, q is a pixel in the kernel $\mathcal{N}$ around p, and I_p is the intensity of the pixel. Functions f and g measure

Gaussian-weighted geometric distances, and photometric similarity, respectively. Intuitively, the bilateral filter tends to smooth more when neighboring pixels are similar (uniform areas), and smooth less when there are big jumps (avoid smoothing edges). This idea can be extended to filtering a signal based on another synchronized signal. In our situation, we can filter (and even up-sample) the depth map of a Kinect scan, by processing the contrast in the color image. Mathematically, Equation 2.4 is similar, but the smoothing operators work in the color image domain: S is the depth image, I is the RGB image and f and g are Gaussian functions centered at 0 and with standard deviations σ_{color} and σ_{depth}.

$$\tilde{I}_p = \frac{\sum_{s \in \mathcal{N}} f(p-s) * g(I_p - I_s) * I_s}{\sum_{p \in \mathcal{N}} f(p-s) * g(I_p - I_s)} \tag{2.3}$$

$$\tilde{S}_p = \frac{\sum_{q \in \mathcal{N}} S_q * f(||p-q||) * g(||I_p - I_q||)}{\sum_{q \in \mathcal{N}} f(||p-q||) * g(||I_p - I_q||)} \tag{2.4}$$

Paris et al [PD09] introduce a novel way of approximating the bilateral filter in order to make fast implementations possible. This is done by interpreting the image as a function plotted in 3D, where the whole filter becomes a convolution in 3D. The authors prove that this approach yields very similar results to the original bilateral filter, but with much faster runtimes. In the following discussion, all the bilateral filtering results will use the PCL implementation of this algorithm.

The last noise removal method we used is based on fitting local maps using the moving least squares technique [ABCo*03]. For each point p of the cloud, a local plane is estimated for the neighborhood by principal component analysis (PCA) and a 2D function $g(x,y)$ (usually of degree 2 or 3) is fitted, as in Equation 2.5, where q is the origin of the reference frame (i.e., the origin of the local plane), n is the normal of the plane, and p_i is a point in the neighborhood of q. The initial point is then projected to this function and pushed into the result cloud.

$$\sum_{i=1}^{N} [g(x_i, y_i) - n \circ (p_i - q)]^2 * \theta(||p_i - q||) \tag{2.5}$$

In order to smooth the high noise of Kinect data, large neighborhood radii need to be used, especially for points farther away from the camera. This is computationally intensive, so various improvements can be done. Instead of using every point in the input cloud, one could use each k-th point and make sure that all the surfaces are taken into account by sampling

Cloud	RMSE original Simulated Noise [cm]	RMSE Bilateral Filter [cm]	RMSE MLS [cm] Filter [cm]	RMSE Median Filter [cm]
1	18.219	17.465	17.672	17.457
2	19.730	19.092	19.123	19.147
3	17.066	16.782	16.665	16.436
4	17.259	17.173	16.756	16.475
5	11.542	11.533	11.197	10.938

Table 2.1: RMS errors of simulated Kinect data using Blensor, after being filtered with the three proposed approaches, as compared to the ground truth noise-free data.

uniformly or within a voxel grid. Furthermore, this method allows for changing the density of points in the input data by sampling the local plane and projecting the new points to the local function. More details on different sampling methodologies and their effects can be found here [Ich12]. An advantage of this approach is that it works on general, unorganized point clouds, so moving least squares (MLS) smoothing can be applied as a post-processing step on the global point cloud (concatenation of registered point clouds) in order to smooth out any remaining noise or registration imperfections. In addition, this method can be made more robust by computing the local function in an iterative fashion on the same patch of points and using an M-estimator to eliminate outliers in each iteration.

Figure 2.2 displays the results using the three described filtering methods, with the best parameters we found. It is clear that the median filter (second row) and the MLS filter (third row) have difficulties with the quantization effects presented by the Kinect sensor at large depths, by clustering slices together and exaggerating the distances between them. This will most likely cause issues when pairing points between frames. The bilateral filter (fourth row) performs best at all depths, handling the sharp edge of the box well (first column), and alleviating the quantization effects at high depths, by spreading the points on the surface. In terms of running times, MLS is the slowest with an average run time in the seconds range for each 640x480 cloud, followed by the median filter at 300 ms, and the fast implementation of the bilateral filter at 80 ms per cloud. In conclusion, the filtering of choice for our application is the latter.

We performed a benchmark for the proposed algorithms using a simulated Kinect sensor in the Blensor software. We generated five scans of a kitchen setting with and without noise, applied the three filters on each noisy cloud and computed the RMSE improvement compared to the input. The results are shown in Table 2.1.

2.2.2 Sampling

As introduced by [RL01], there are multiple simple ways of sampling point clouds in order to improve the performance of registration. The authors look into the options of:

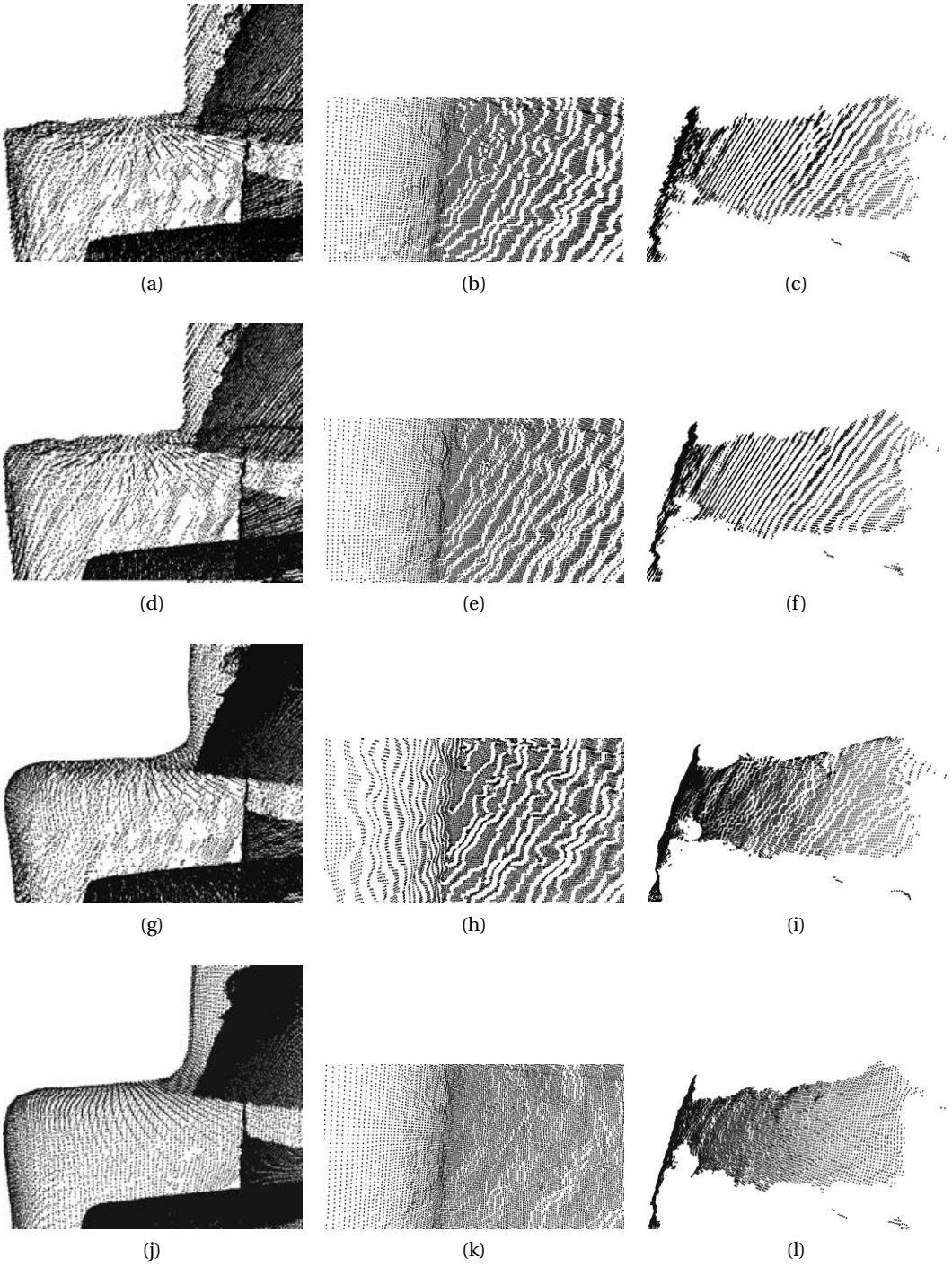
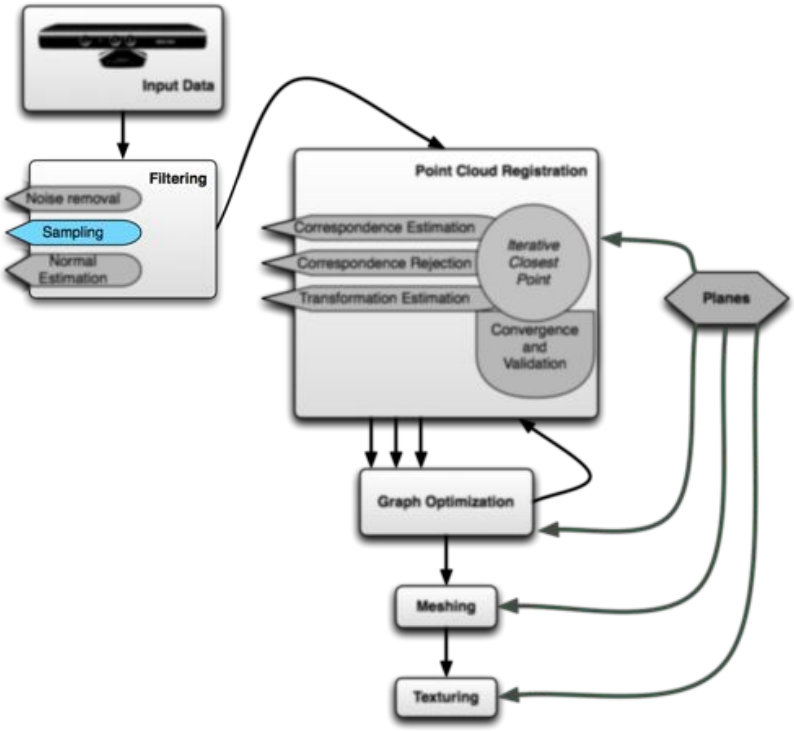


Figure 2.2: Close-up on results of different types of filtering: (a, b, c) unfiltered input cloud; (d, e, f) median filter; (g, h, i) MLS filter; (j, k, l) bilateral filter



- using all the points
- uniform sampling
- random sampling
- normal space sampling
- sampling points on color edges

Rusinkiewicz quantifies the performance of these sampling methods by plugging them into a baseline ICP pipeline and checking how fast two clouds converge, using artificially-generated datasets. Their conclusion is that *normal space sampling* performed the best. We add one more method to the list, based on [GR03], called *covariance sampling*. In addition to evaluating the convergence rate on a few RGB-D scan pairs that represent common situations in our reconstruction application, we perform an analysis of how fast the pose estimation drifts away from the ground truth in an incremental registration scenario.

Using all the points is a special case of *uniform sampling* (equivalent to using a sampling rate of 1). As concluded by [RL01], *uniform* and *random sampling* perform almost identically in their tests. The only theoretical advantage of *random sampling* is that it forces the correspondence estimation method to choose different pairs at every iterations, thus reducing the probability of choosing the same possibly incorrect correspondences that might lead to slower convergence or convergence into a local minima.

Normal space sampling consists of placing the normals into m^3 bins, where m is the number of bins per dimension, and then randomly selecting an equal number of points from each bin. Intuitively it suggests that the sampled cloud contains points that cover the half-sphere of orientations uniformly. In [RL01], the authors concluded it performed better than the afore mentioned methods.

Covariance sampling is based on the fact that a point cloud can be sampled better if we select points such that the transformation is locked more easily. The target is to have a sampled cloud with all eigenvalues of the covariance matrix being equal. Starting from the function that is to be minimized by the registration process:

$$E = \sum_{i=1}^{k} [(Rp_i + t - q_i) \cdot n_i]^2 \tag{2.6}$$

By linearizing the rotation matrix, the transformation becomes a 6-vector $[r^T \ t^T]$, and the error becomes:

$$E = \sum_{i=1}^{k} [(p_i - q_i) \cdot n_i + r \cdot (p_i \times n_i) + t \cdot n_i]^2 \tag{2.7}$$

Each point exerts two types of forces on its pair: a translational force along the normal n of each point p and a rotation torque around the axis $(p \times n)$ (i.e., orthogonal to the normal), shown by the last two terms of the equation above. Thus, the error of a single pair changes as follows, if one of the points moves by $[\Delta r^T \ \Delta t^T]$:

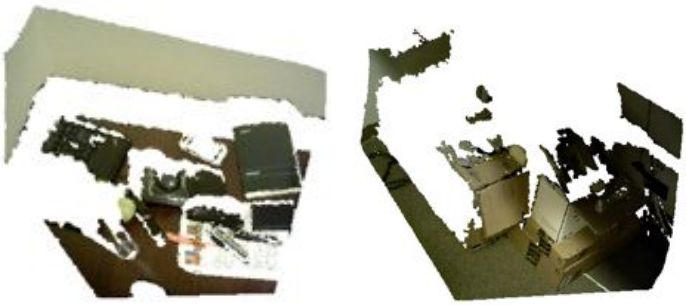
$$\Delta d_i = [\Delta r^T \ \Delta t^T] \begin{bmatrix} p_i \times n_i \\ n_i \end{bmatrix} \tag{2.8}$$

which leads to the covariance matrix for the cloud:

$$C = FF^T = \begin{bmatrix} p_1 \times n_1 & \dots & p_k \times n_k \\ n_1 & \dots & n_k \end{bmatrix} \begin{bmatrix} (p_1 \times n_1)^T & n_1^T \\ \dots & \dots \\ (p_k \times n_k)^T & n_k^T \end{bmatrix} \tag{2.9}$$

This covariance matrix helps us identify unconstrained transformations (i.e., clouds that can "slide" on each other - the sliding direction is defined by the eigenvector corresponding to the

(a) single wall (b) two walls (c) room corner

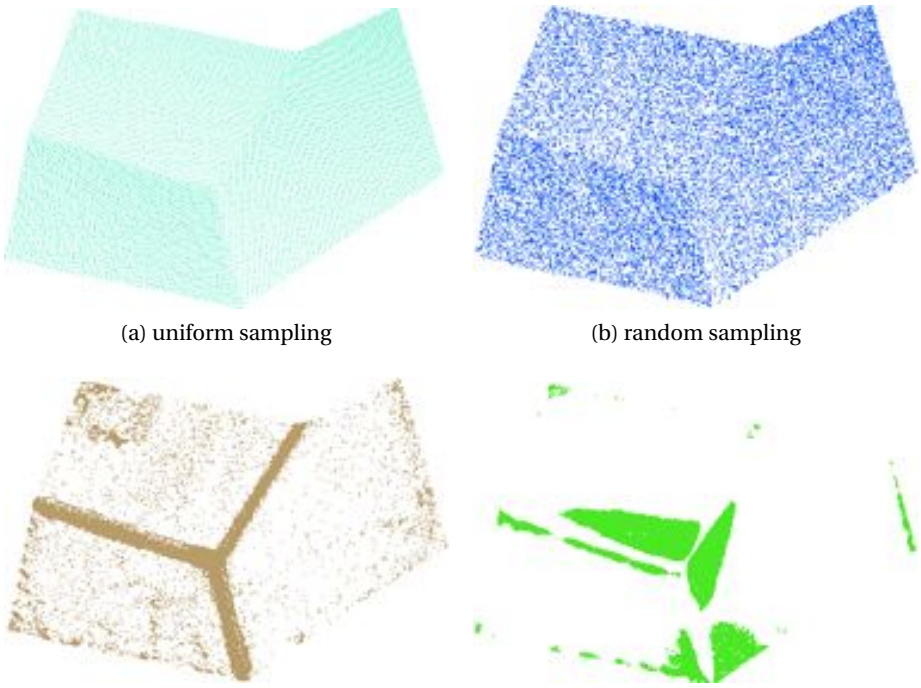


(d) desk clutter (e) boxes

Figure 2.3: Sample scans to show the correlation between the registration stability and the condition number of the covariance matrix, see Table 2.2.

smallest eigenvalue). For example, if the matrix is not full rank, that means that the solution transform is not unique. In order to quantify this, the authors suggest to use the condition number of the matrix, which is the ratio of the largest eigenvalue to the smallest eigenvalue. Figures 2.3, 2.4 and Table 2.2 show some Kinect scans along with their condition numbers under certain sampling techniques. The closer this measurement is to 1.0, the more stable the cloud is. As such, Gelfand et al. suggest an algorithm [GR03] to sample a point cloud so that to minimize the condition number of the resulting set of points, and prove better ICP convergence. Our experiments encouraged a combination between normal space sampling and covariance sampling to be the best solution.

State-of-the-art SLAM systems employing RGB-D cameras make use of uniform and random sampling, without dealing with more complex sampling schemes. [HKH*10] downsample the clouds to $\frac{1}{4}$ - $\frac{1}{10}$ of their initial size (75000 - 30000 points) for geometric alignment. In order to find the ideal parameters for an ICP pipeline for registering RGB-D clouds, [PMC*11] consider that complex subsampling techniques are too slow for an online ICP procedure, suggesting that uniformly sampling the cloud to a resolution of 160x120 ($\frac{1}{16}$ of the initial size), and then randomly sampling to about 3700 points ($\frac{1}{100}$) is a good trade-off between precision and performance. The authors also note that the factor that influences the running time of the



(a) uniform sampling

(b) random sampling

(c) normal space sampling

(d) covariance sampling

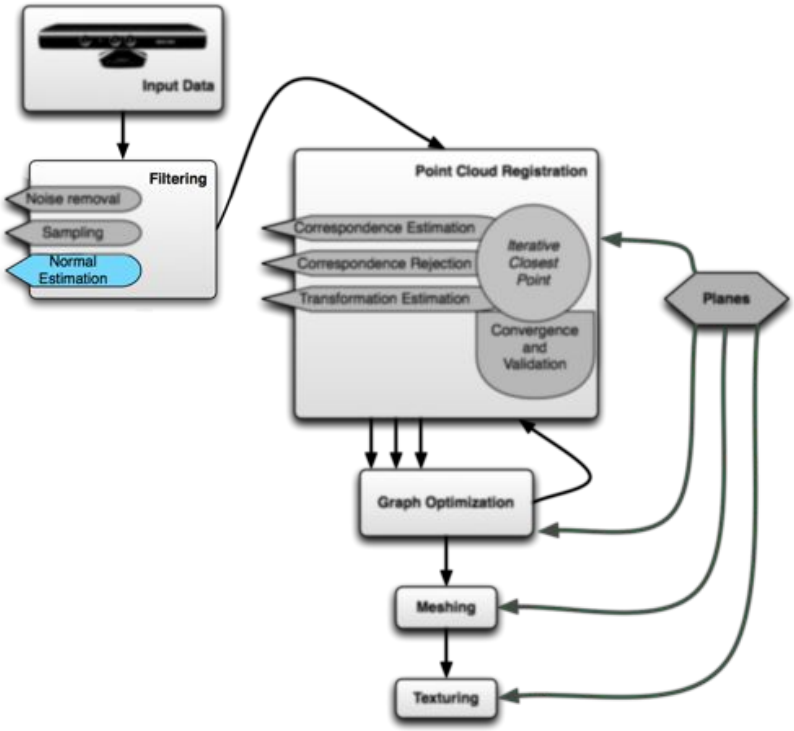
Figure 2.4: The room corner point cloud sampled with various techniques.

Scan	Condition number				
	input	uniform	random	normal	covariance
single wall	2946.26	2910.05	2959.42	498.65	635.89
two walls	862.45	1099.66	824.77	98.47	101.21
room corner	10.27	10.27	10.31	7.7	6.97
desk clutter	12.83	13.10	12.91	6.89	21.36
boxes	4.58	4.59	4.73	3.32	9.33

Table 2.2: Condition numbers for the covariance matrices of the clouds in Figure 2.3 sampled differently.

pipeline most significantly is the number of points left after sampling the clouds., so tuning this parameter is essential.

2.2.3 Normal Estimation



A lot of computer vision tasks (e.g., segmentation, object recognition, reconstruction) make use of normal data in addition to the 3D coordinates of the points. 3D sensors scan surfaces by providing discrete sample points, usually with added noise. In this sampling process, information about the orientation and curvature of the surface is lost. Normal estimation has the purpose of restoring these values for each sample point by looking at its local neighborhood.

Klasing et al. publish [KAWB09] a thorough comparison of normal estimation methods, shedding light on the mathematical and implementation details of each of the numerous normal estimation methods developed by researchers in robotics, computer vision and computer graphics. They split normal estimation methods into two categories: *averaging methods* ([Gou71], [JLW05], [Max99], [TW98]), and *optimization-based methods*. Approaches that average triangles in the neighborhood of the query point p in order to compute its normal fall into the first category. The individual triangle normals are computed by cross product of the sides, and then they are averaged using various schemes: *angle-weighted*, *area-weighted*, *centroid-weighted*, and *gravitational-weighted*. The performance of these methods leaves to be desired as compared to the optimization-based approaches, named like this because they

solve the following optimization problem:

$$\min_{n_i} J(p_i, Q_i, n_i) \tag{2.10}$$

where J is a cost functional that penalizes certain criteria, p_i is the query point, and n_i is the normal to be computed. The most popular ways of solving this are:

PlaneSVD fit a local plane $P_i = n_{i_x} x + n_{i_y} y + n_{i_z} z + d$ to the points of the local neighborhood Q_i, and minimize the error by using singular value decomposition.

PlanePCA minimize the variance of the points along a local coordinate system, where the axis of lowest variance is the normal, i.e., the normal is the eigenvector of the XYZ covariance matrix of the points in Q_i corresponding to the smallest eigenvalue.

Based on the evaluation done by the authors, *PlanePCA* is superior to all the other methods in terms of speed an quality of the results under different neighborhood sizes and amounts of noise of the input data. PCL implements this as its main normal estimation method, and we will be using it in our evaluation.

Recently, Holzer et al. [HRD*12] propose a new approach for normal estimation from organized clouds using the concept of integral images. The advantage of this data structure is that it requires a linear pre-processing time and allows to compute the average value within a rectangular area of the image in constant time. The integral image I_O will be of the same size as the input image O (Equation 2.11), and can be computed efficiently in a single pass through the image as in Equation 2.12. Equation 2.13 proves that computing the average value in a rectangular region of inner radius r can be done in constant time.

$$I_O(m,n) = \sum_{i=0}^{m} \sum_{j=0}^{n} O(i,j) \tag{2.11}$$

$$I_O(m,n) = I_O(m-1,n) + I_O(m,n-1) - I_O(m-1,n-1) + O(m,n) \tag{2.12}$$

$$V(I_O,m,n,r) = \frac{1}{4r^2}[I_O(m+r,n+r) - I_O(m-r,n+r) - I_O(m+r,n-r) + I_O(m-r,n-r)] \tag{2.13}$$

In the following, we present only the *Smoothed Depth Change* method suggested by Holzer

et al., as their experiments showed it performed best under noisy data, as is the case with the Kinect sensor. The first step is to smooth the data. As mentioned before in this section, the noise levels along the z-axis of PrimeSense cameras vary quadratically with the depth measurements. That is why smoothing will be done with a variable window size, proportional to the square of the depth, creating the *Smoothing area map* $B(m,n) = \alpha D(m,n)^2$, where α is a scaling factor that controls the smoothing area size. Smoothing this way can cause averaging of points situated on different surfaces, so the *Depth change indication map* is introduced to detect high contrast areas in the depth map (i.e., depth jumps). The contrast threshold is adaptive, using the same observation about the depth noise as before:

$$C(m,n) = \begin{cases} 1 \begin{cases} \text{if } ||\delta D_x(m,n)|| \geq \beta D(m,n)^2 \\ \text{or } ||\delta D_y(m,n)|| \geq \beta D(m,n)^2 \end{cases} \\ 0 \text{ otherwise} \end{cases} \tag{2.14}$$

The *final smoothing area map* is computed as:

$$R(m,n) = \min(B(m,n), \frac{T(m,n)}{\sqrt{2}}) \tag{2.15}$$

where T is the distance transform map corresponding to C. The normal for each point is then calculated as the cross-product of the vectors formed by the upper and lower neighbors, and the left and right neighbors, respectively: $\vec{n_p} = \vec{v_{p,h}} \times \vec{v_{p,v}}$. The two vectors are computed using the input point coordinates P_x, P_y, P_z, the integral image for the depth I_z and the adaptive window radius $r = R(m,n)$:

$$\begin{aligned} \vec{v_{p,h,x}} &= \frac{P_x(m+r,n) - P_x(m-r,n)}{2} \\ \vec{v_{p,h,y}} &= \frac{P_y(m+r,n) - P_y(m-r,n)}{2} \\ \vec{v_{p,h,z}} &= \frac{V(I_z,m+1,n,r-1) - V(I_z,m-1,n,r-1)}{2} \\ \vec{v_{p,v,x}} &= \frac{P_x(m,n+4) - P_x(m,n+r)}{2} \\ \vec{v_{p,v,y}} &= \frac{P_y(m,n+r) - P_y(m,n-r)}{2} \\ \vec{v_{p,v,z}} &= \frac{V(I_z,m,n+1,r-1) - V(I_z,m,n-1,r-1)}{2} \end{aligned} \tag{2.16}$$

In comparison with the normal estimation based on PCA, the integral image approach yields better results with less processing time, as concluded by the authors in their experiments. A downside is the fact that the integral images normal estimation will not be able to compute the normals near the border of the depth map, as represented in Figure 2.5. This is not problematic for our implementation, as we are eliminating rows and columns around the image borders anyway, due to the large distortions they suffer (see the previous subsections for more details).

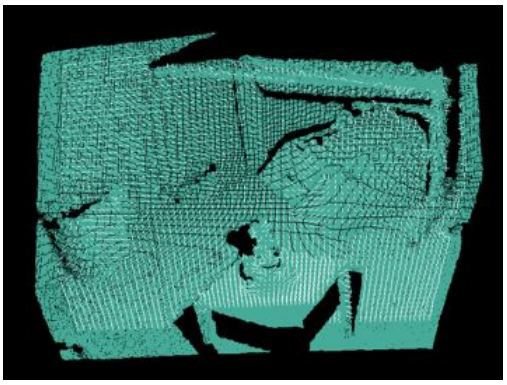


Figure 2.5: The integral images approach cannot compute the normals near the border of the depth map.

We ran tests with both the PCA-based normal estimation and the ones based on integral images. The parameters we found best for indoor scanning situations are a radius of 5 cm for the PCA normal estimation, and a depth change factor of 1 cm with a smoothing size of 50 for the integral images normal estimation. The run times for a 640x480 Kinect cloud are in the seconds range for the first approach and on average 50 ms for the second. In terms of the quality of the normals, the integral images normal estimation performs better at different depths (Figure 2.6), and they handle high curvature areas equally well. As such, we use the integral images normal estimation in our pipeline.

2.3 Correspondence Estimation and Rejection

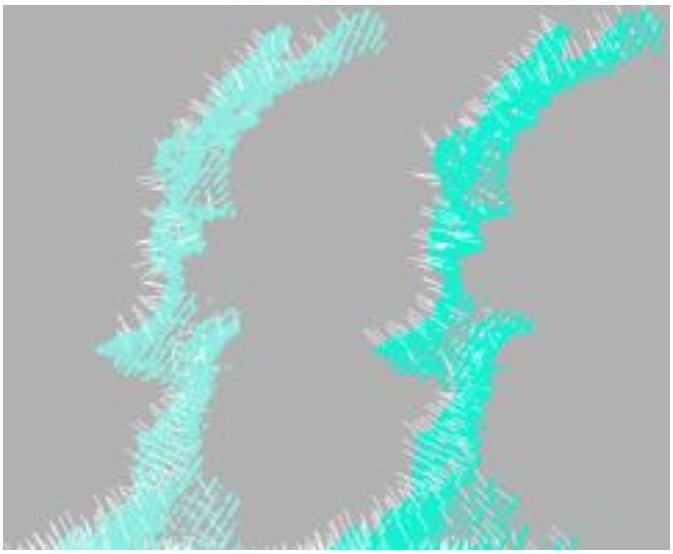
2.3.1 Correspondence Estimation

Correspondence estimation is the process of pairing points from clouds after filtering and subsampling. We propose two ways of doing this: *projection-based* and *search-based*.

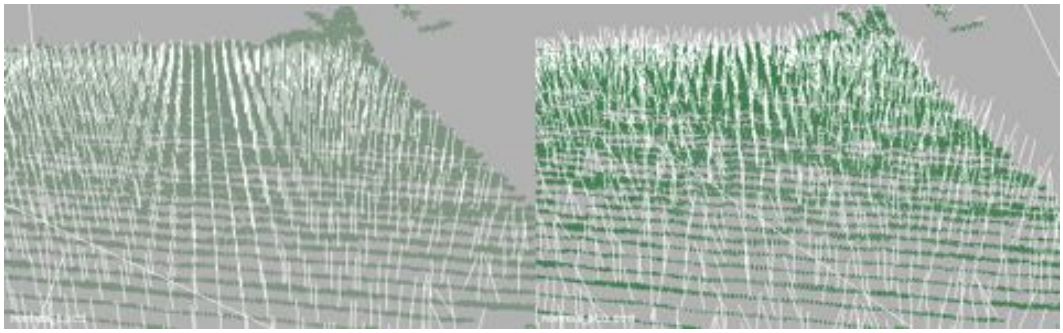
PrimeSense RGB-D cameras output a depth map and a registered RGB image, the colored point cloud being a product of the two. This means that each point in the cloud corresponds to a uv pixel/dexel, allowing us to do projections from points in world coordinates to the camera plane by using the intrinsic and extrinsic camera parameters:

$$\begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = \begin{bmatrix} d \cdot u \\ d \cdot v \\ d \end{bmatrix} \tag{2.17}$$

where the first matrix is the projection matrix of the source camera, the second matrix is the transformation of the source camera, d is the depth of the projected point P in the target

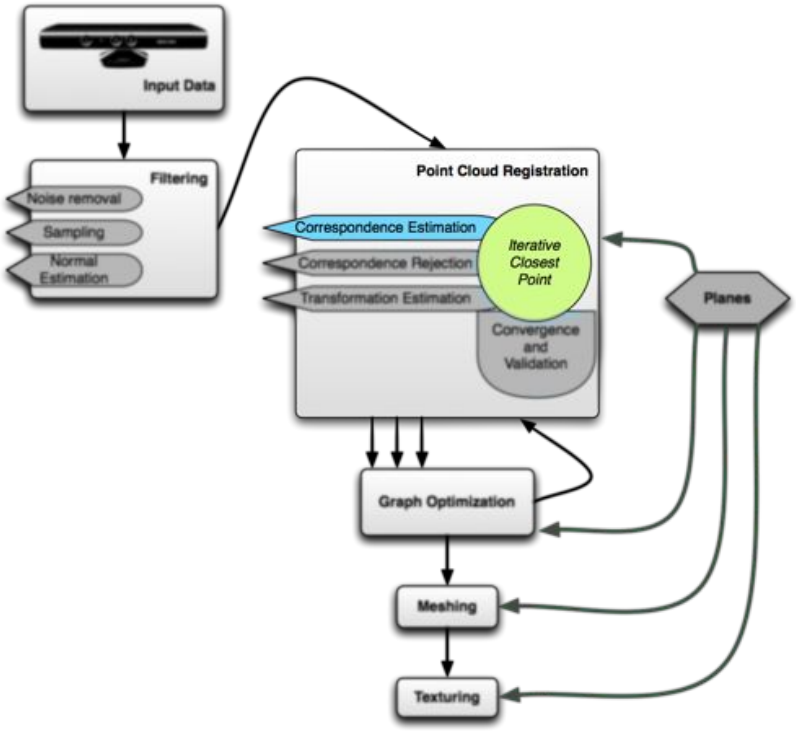


(a) Near, 1 meter depth



(b) Far, 3-4 meters depth

Figure 2.6: Comparison of the quality of the normals at different depths: [left] integral images normal estimation, [right] PCA-based normal estimation



camera, and u, v are the coordinates of the P in the target camera plane.

This approach is very fast, but imprecise, especially when considering point clouds with large depth discontinuities. That is why this method is recommended to be used only after the two point clouds have been brought close together, making it good for aligning consecutive point clouds in a stream recorded at high frame rate.

Search-based methods are precise, always returning the closest point to the query point. A naive way of searching for the nearest neighbor is by doing an exhaustive search through all the target points for the the closet one to each source point. As this was proven to be prohibitively expensive for applications using millions of points, various data structures for rapid searches have been proposed, such as octrees and kd-trees. These data structures have logarithmic search times (as opposed to the linear naive variant), but take longer to initialize, usually $O(N \log N)$ or $O(N^2)$ in the worst-case. In our implementation, we are using FLANN [ML09], an open-source library for fast approximate nearest neighbor searches. It has been empirically proven to be one of the fastest solution for 3D nearest neighbor searches, as Elseberg et al. conclude [EMSN12] after comparing multiple open-source implementations of nearest neighbor search strategies.

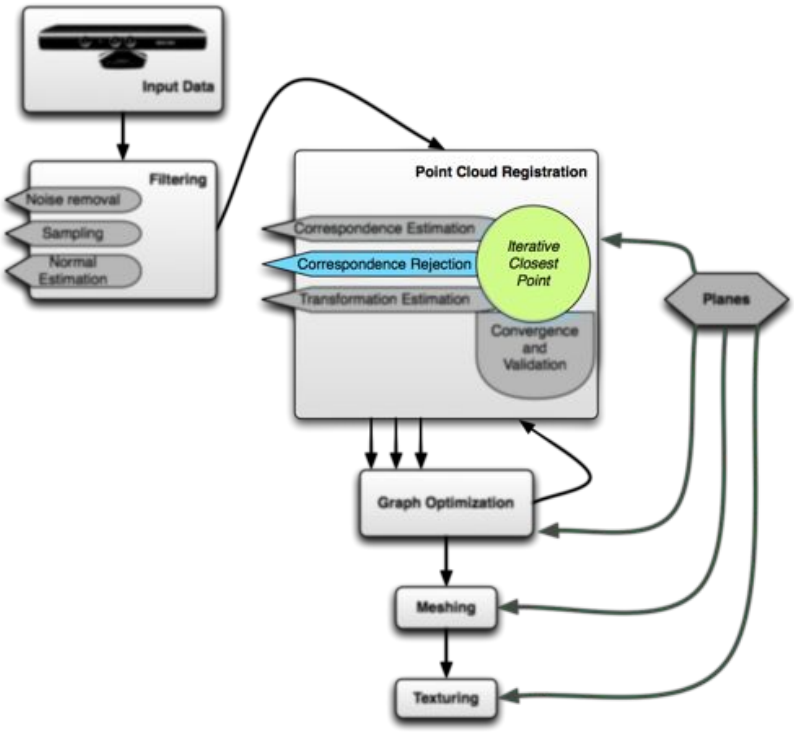
We perform a small timing test (results in Table 2.3). Considering the fact that we will generally be using a small number of points for correspondences (around 3000-10000), the gain in performance for projection-based correspondence estimation is not worth the imprecisions that this method exposes, so kd-trees will be used throughout our implementation, as [HKH*10]

Cloud size	Naive search [ms]	Kd-tree		Projection-based [ms]
		init [ms]	search [ms]	
307 200	35045	106	660	3
76800	2188	23	130	1
19200	139	5	25	1
4800	8	1	6	1

Table 2.3: Running times for different nearest neighbor search algorithms

also do.

2.3.2 Correspondence Rejection



Correspondence rejection is the step of filtering incorrect correspondences that have been collected during the estimation process. This is an important step, as having "clean" correspondences can ease the transformation estimation algorithm into finding the global minimum. Furthermore, this step can take advantage of extra information about the input point clouds, such as normal information or statistics about the correspondences. We investigate the following methods:

pcl::CorrespondenceRejectorDistance This method filters out point pairs that have an Euclidean distance between them larger than a given threshold, suggested by [RL01] (see

Figure 2.7a)

pcl::CorrespondenceRejectorMedianDistance Unlike the previous rejector, this one does not use a fixed threshold, but computes the threshold as being the median distance between the input set of correspondences. This approach considers the distribution of the distances between the points and it adapts, becoming smaller as the two point clouds come closer together in the ICP iterations. And unlike an adaptive threshold based on the mean, using the median reduces the influence of outliers.

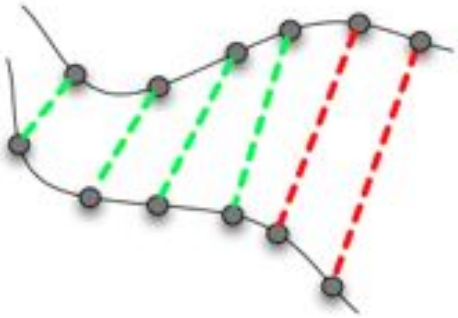
pcl::CorrespondenceRejectorOneToOne Usually, each sampled point in the source cloud gets a correspondence in the target cloud, so it might be the case that the same point in the target cloud gets multiple corresponding source points due to different sampling rates (see Figure 2.7c). This method keeps a single such pair $(p_{src_{min}}, p_{tgt})$: the one with the minimum distance out of all the pairs (p_{src_i}, p_{tgt}), $i = 1 : k$. This rejection method is not suited for registering surfaces with different point densities: suppose a source surface has a higher density than a target surface that is close to it; in a point-to-plane situation, having a target point paired with multiple source points will not affect the registration, and thus these pairs should not be thrown away, as they positively influence the convergence of the algorithm.

pcl::CorrespondenceRejectorSampleConsensus Uses *RANSAC* to estimate a transformation for the given set of correspondences and eliminates the outlier correspondences based on the Euclidean distance between the points after the best transformation is applied to the source point. This method is very effective in keeping the ICP algorithm from converging into a local minima, as it always produces slightly different correspondences and is good at filtering outliers. In addition, it provides a good initial estimate for ICP.

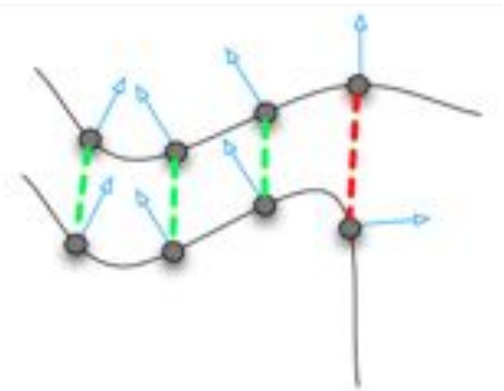
pcl::CorrespondenceRejectorSampleConsensus2D Similar to the previous approach, only that it rejects pairs based on their distance in the image plane after the source point is transformed and projected into the target camera plane.

pcl::CorrespondenceRejectorSurfaceNormal Introduces normal information about the pairs, and rejects those pairs that have inconsistent normals, i.e., the angle between their normals is larger than a given threshold. It can reject erroneous pairs that seem correct judged by the distance between the points, such as the case depicted in Figure 2.7b.

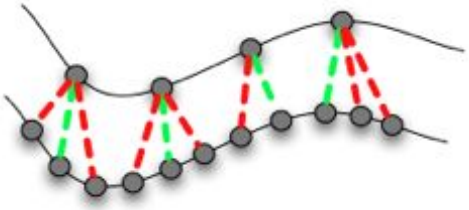
pcl::CorrespondenceRejectorBoundaryPoints When the two point clouds represent surfaces that have partial overlap, allowing for correspondences containing surface boundary points can introduce errors (Figure 2.7d). In order to detect boundary points, we can use the organized nature of the input point clouds and eliminate the correspondences that contain points on depth discontinuities by moving a window across the depth map and checking if there are enough points in the window on the same surface with the center point (within a certain depth range from the center point).



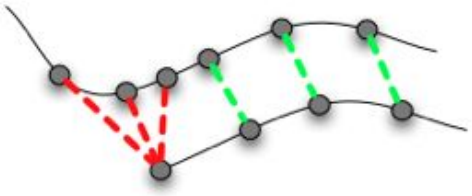
(a) Correspondence rejection based on the Euclidean distance between the points



(b) Correspondence rejection based on normal compatibility

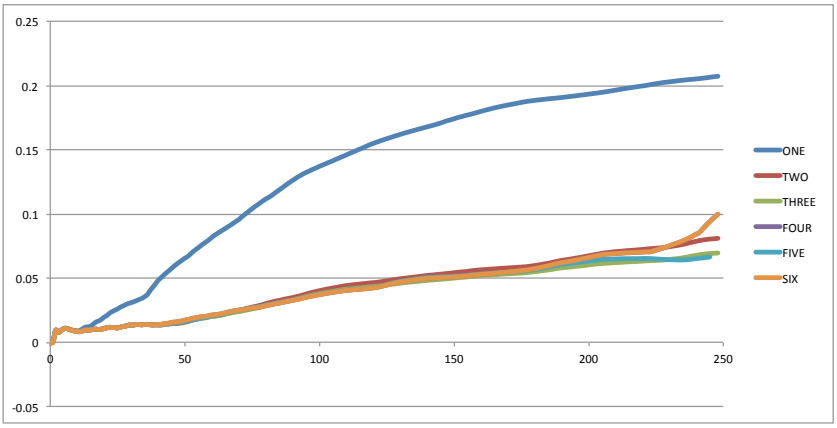


(c) Correspondence rejection of pairs with duplicate target matches

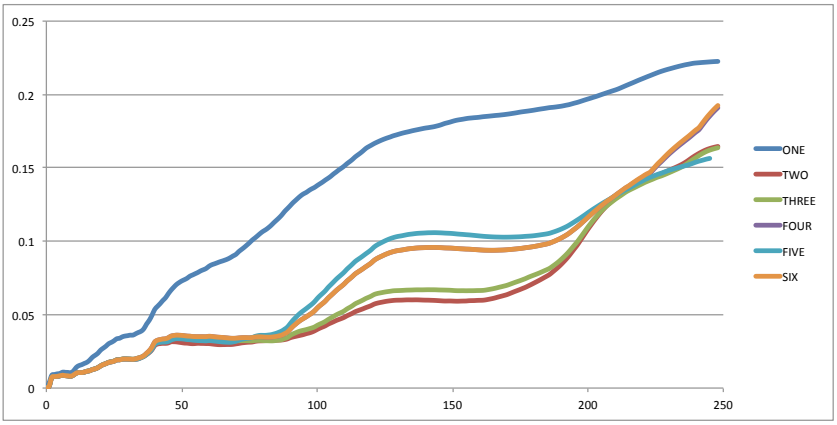


(d) Correspondence rejection of pairs that contain boundary points

Figure 2.7: Correspondence rejection methods



(a) cloud number vs angle RMSE in radians



(b) cloud number vs translation RMSE in meters

Figure 2.8: The drift accumulated by the six different correspondence rejection pipelines in an incremental registration situation (no loop closure).

Diebel et al. [DRT*04] apply a combination of ways for rejecting point pairs in their active stereo point clouds, emphasizing the importance of rejecting points on mesh boundaries. In addition, they reject pairs based on their normal compatibility and do a statistical analysis on an adaptive threshold for rejecting pairs based on the Euclidean distance between the points. The outcome is that a threshold equal to three times the median helps reach better convergence with ICP.

In order to test these methods, we use the same approach as done for the sampling approaches in Section 2.2.2, which is to run individual or different combinations of rejectors in a baseline ICP pipeline and inspect the drift of the pose estimation in an incremental registration setting on a sequence of 250 point clouds. Figure 2.8 shows the drift the six different pipelines accumulate in terms of angle and translation (as compared to the ground truth).

We evaluate the convergence rate and the robustness of the transformation estimation under various correspondence filtering pipelines. The optimization algorithm used for computing

the transformation that minimizes the point-to-plane error metric was Levenberg-Marquardt, under the following combinations of correspondence filtering methods:

ONE closest-point correspondence estimation with no filtering.

TWO closest-point correspondence estimation, with **pcl::CorrespondenceRejectorMedianDistance** (threshold equal to twice the median distance) filtering.

THREE closest-point correspondence estimation, with **pcl::CorrespondenceRejectorMedianDistance**, and **pcl::CorrespondenceRejectorSurfaceNormal** (threshold of 30°) filtering.

FOUR closest-point correspondence estimation, with **pcl::CorrespondenceRejectorMedianDistance**, **pcl::CorrespondenceRejectorSurfaceNormal**, and **pcl::CorrespondenceRejectorBoundaryPoints** filtering.

FIVE closest-point correspondence estimation, with **pcl::CorrespondenceRejectorMedianDistance**, **pcl::CorrespondenceRejectorSurfaceNormal**, **pcl::CorrespondenceRejectorBoundaryPoints**, and **pcl::CorrespondenceRejectorOneToOne** filtering.

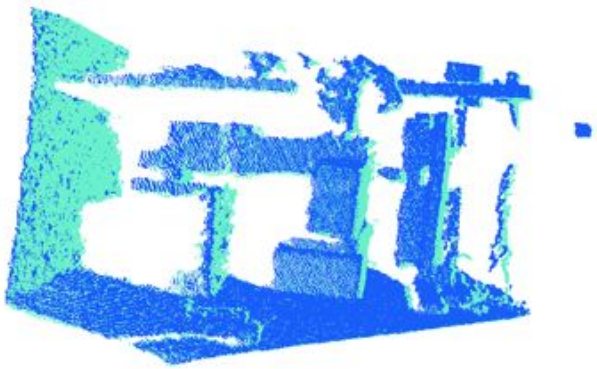
SIX closest-point correspondence estimation, with **pcl::CorrespondenceRejectorMedianDistance**, **pcl::CorrespondenceRejectorSurfaceNormal**, **pcl::CorrespondenceRejectorBoundaryPoints**, and **pcl::CorrespondenceRejectorSampleConsensus** (with a maximum of 1000 iterations and inlier threshold of 5 cm) filtering.

We will use a single source scan and pair it against 3 target scans that are at different distances from the initial scan (see Figure 2.9):

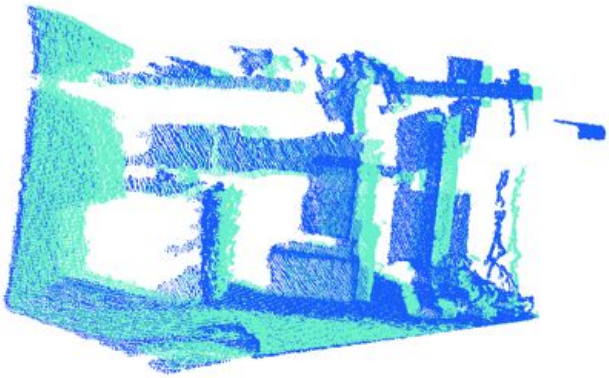
- rotation of 1.3° and translation of 1 cm.
- rotation of 4.3° and translation of 4.5 cm.
- rotation of 8.3° and translation of 23 cm.

The conclusion of the evaluation based on pair registration convergence is that in the case of general indoor scenes scanned with a Kinect, the correspondence filtering methods do not have a strong influence on the result. They all converge with similar rates, and noisy correspondences do not influence the robust LM-based optimizer a lot. Another exception is the non-filtered pipeline (**ONE**)

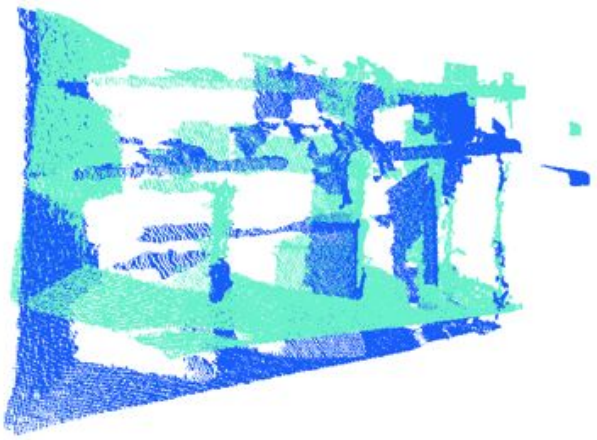
The difference between the various pipelines is the number of actual pairs that the optimizer has to solve for. Having more pairs will cause the solver to take more time to compute the solution, as presented in Table 2.4. In conclusion, pipeline **SIX** provided the best results both in terms of time and convergence rate, and the usefulness of the RANSAC-based correspondence rejection will be analyzed again in the next section.



(a) clouds 1 and 2



(b) clouds 1 and 3

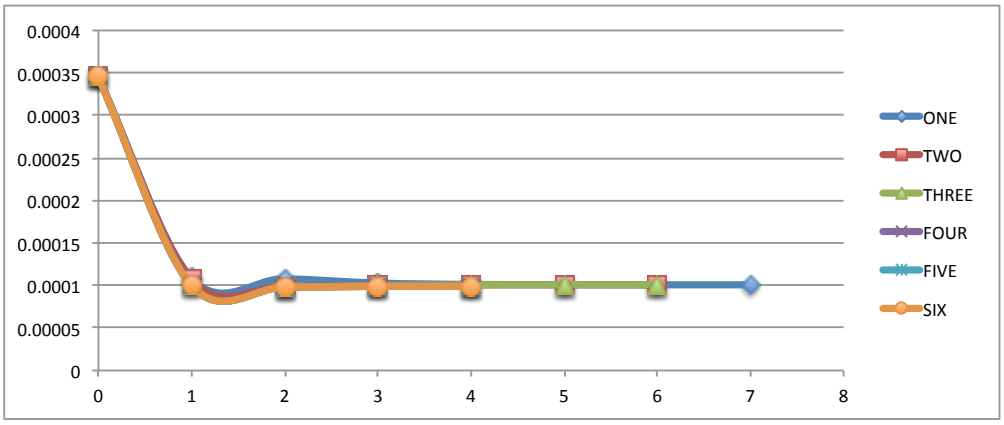


(c) clouds 1 and 4

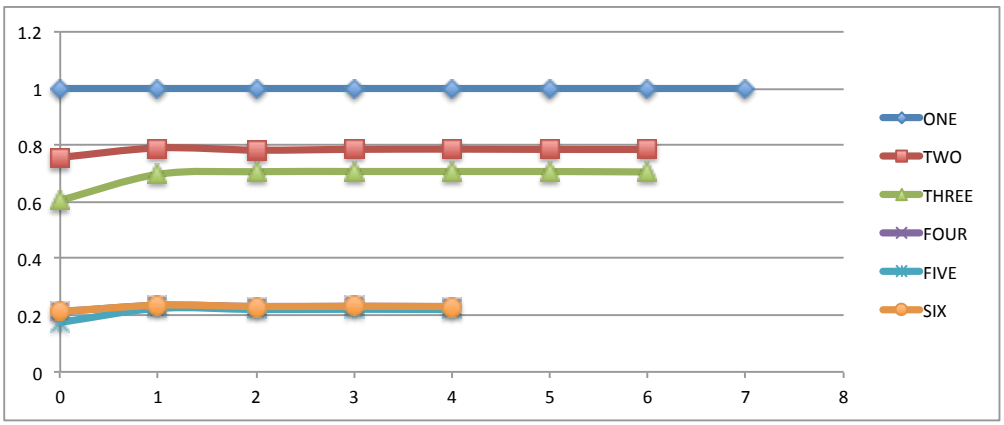
Figure 2.9: The three pair clouds that will be used in the correspondence filtering and transformation estimation methods evaluation.

Pipeline	Correspondence filtering time [ms]	Optimizer time [ms]
ONE	14.1, 15, 27.4	7.2, 7.4, 12.3
TWO	14.9, 15.5, 21.5	6.1, 5.5, 5.7
THREE	14.4, 15.2, 22.5	5.1, 4.7, 2
FOUR	15, 15, 23.8	1.6, 1.4, 0.6
FIVE	15, 15, 24	1.4, 1.6, 0.56
SIX	19, 18.4, 26	1.6, 1.36, 0.5

Table 2.4: Timing for the six correspondence filtering pipelines and the optimization.

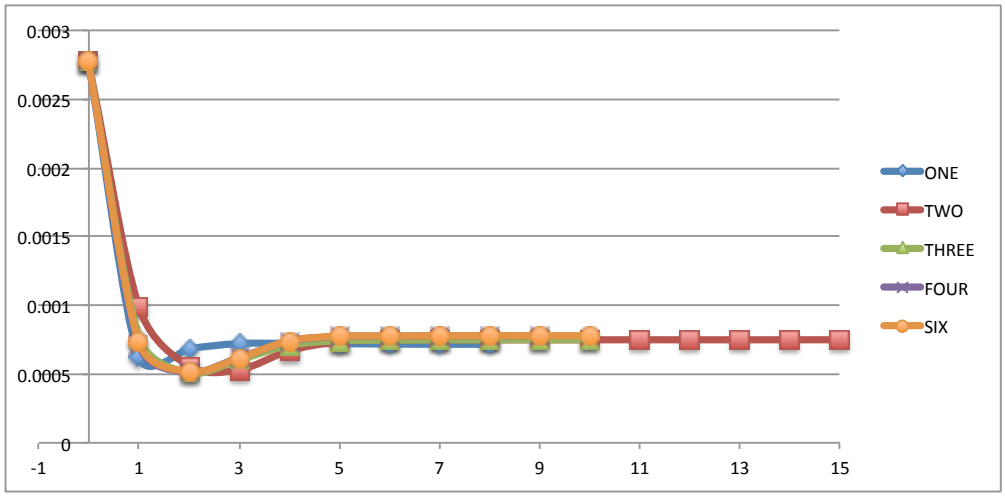


(a) iteration vs MSE

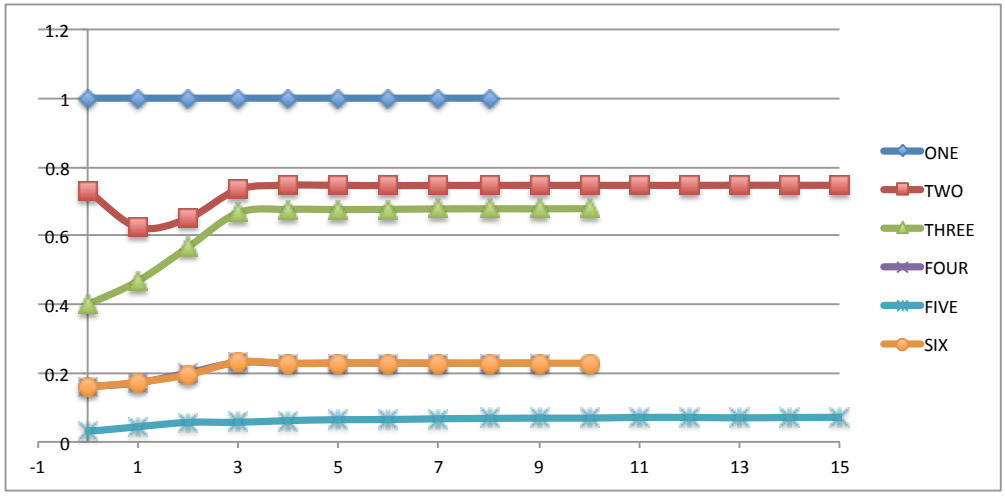


(b) iteration vs percentage of filtered correspondences

Figure 2.10: The progress of the different correspondence estimation and rejection pipelines as they converge in registering the first and second test clouds.

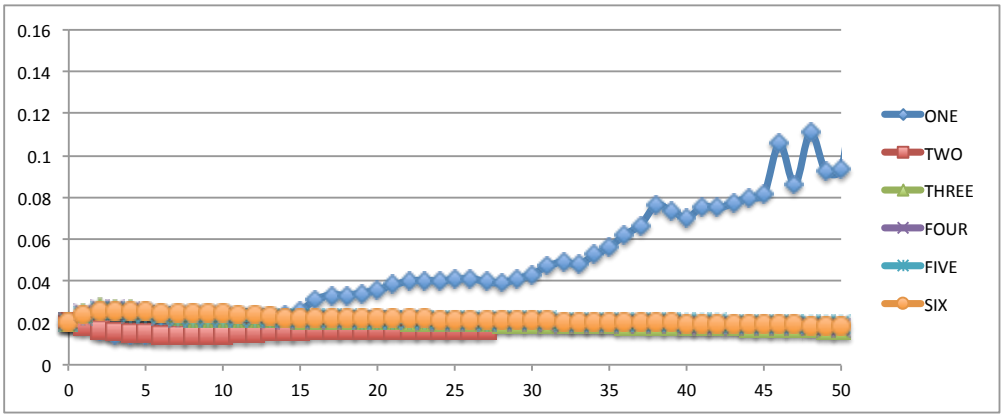


(a) iteration vs MSE

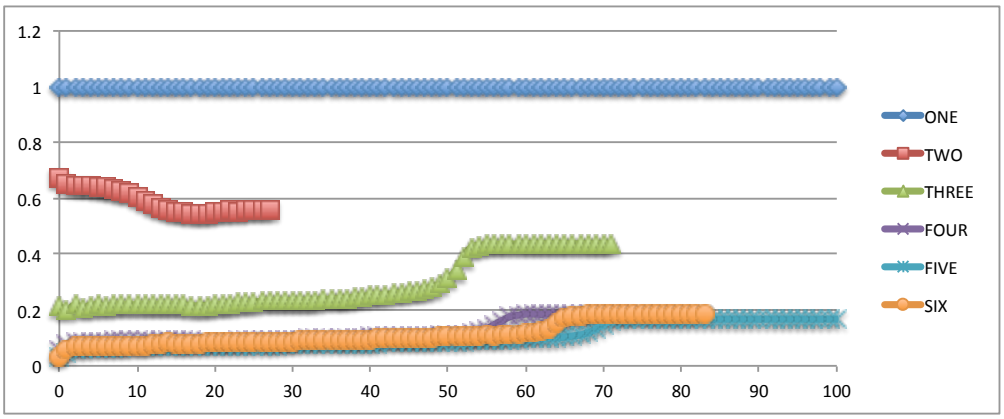


(b) iteration vs percentage of filtered correspondences

Figure 2.11: The progress of the different correspondence estimation and rejection pipelines as they converge in registering the first and third test clouds.



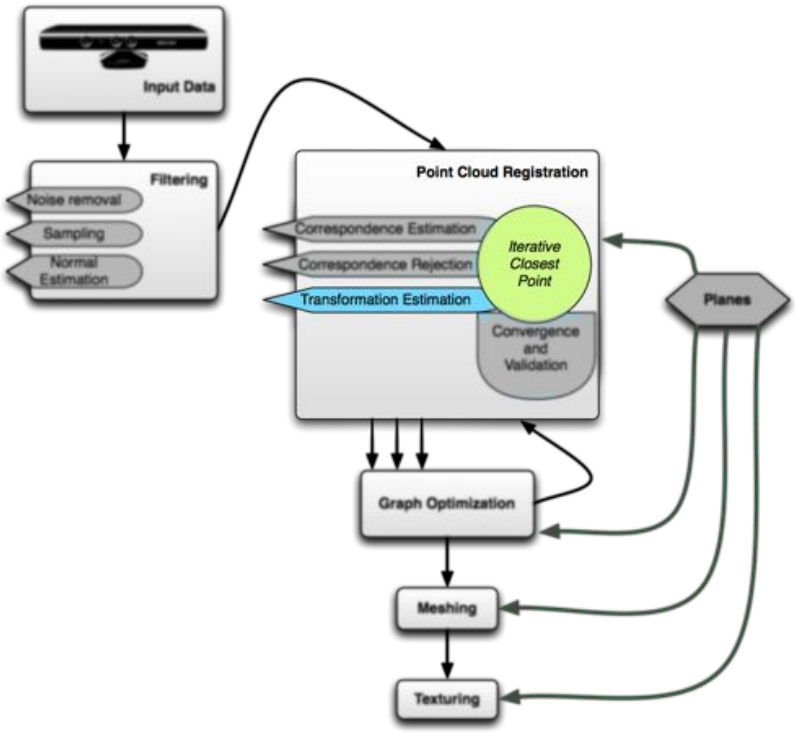
(a) iteration vs MSE



(b) iteration vs percentage of filtered correspondences

Figure 2.12: The progress of the different correspondence estimation and rejection pipelines as they converge in registering the first and fourth test clouds.

2.4 Transformation Estimation and Pair Weighting



The transformation estimation step is done after the corresponding points from the source and target cloud have been filtered. Over the years, there have been numerous mathematical approaches for solving for the transformation that minimizes the error of the point pairs.

There are two main error metrics to be minimized that have been considered in literature: *point-to-point* (Equation 2.18) and *point-to-plane* (Equation 2.19), where (p_i, q_i) are N pair correspondences from the source cloud to the target cloud.

The point-to-point metric was first introduced by Arun [AHB87] in 1987; researchers proposed various ways of minimizing this error metric [Hor87], [HHN88], [WSV91], followed by the introduction of the Iterative Closest Point Algorithm in 1992 [BM92]. Eggert et al. [ELF97] evaluated each of these methods in terms of numerical stability and accuracy reaching the conclusion that they are close performers.

In 1991, Chen [CM92] introduces the point-to-plane metric and proves to be more stable and converge faster than the previous approaches. This metric does not have a closed-form solution like the point-to-point, so the solutions are split into non-linear solvers (such as Levenberg-Marquadt, as proposed by [DF01]), or by linearizing it [Low04] (by assuming small

rotations, i.e. $sin\theta \sim \theta$ and $cos\theta \sim 1$).

$$\sum_{i=1}^{N} ||R\ p_i + t - q_i||^2 \tag{2.18}$$

$$\sum_{i=1}^{N} ((R\ p_i + t) - q_i) \cdot n_{q_i})^2 \tag{2.19}$$

The weighting of the point pairs can be seen as a soft correspondence rejection, adjusting the influence of that pair in the minimization process, as opposed to the hard rejection of completely ignoring the pair. The weighting can be a function of the point-to-point or point-to-plane distance between the points, a function of the angle between the normals of the corresponding points, or a function of the noise model of the sensor that has been used. We will employ the latter in our implementation, inspired by the Kinect noise model derived in [NIL12] (Equation 2.20). All of the papers we have researched that mention the weighting of point pairs have marked this as a beneficial step for the robustness and convergence rate of the transformation estimation algorithm [RL01], [NIL12], [HKH*10], [DRT*04].

$$w(p_i, q_i) = 0.0012 + 0.0019 * (max(p_{i_z}, q_{i_z}) - 0.4)^2 \tag{2.20}$$

We perform an evaluation on the convergence rate and the robustness to noisy correspondences of these methods. We will use the same three pairs of frames with increasing relative distances. In order to compute correspondences, we propose three schemes:

noisy correspondences - each source point is in correspondence with its closest target point if the target point is within a fixed Euclidean distance.

clean correspondences - same as above, with a filtering step based on normal compatibility.

clean correspondences + randomization - same as above with a correspondence rejection step based on RANSAC. This is to perturb the convergence enough so that the optimizer does not fall intro local minima. It is expected that by using this rejection method, the ICP will converge slower (i.e., in more iterations), but it is less likely to fall into local minima.

The approaches we will look into, along with their corresponding PCL classes are:

pcl::registration::TransformationEstimationPointToPlaneLLS - based on [Low04], linearizes

the point-to-plane error and reduces it to $A^t\ A\ v = A^t b$, with v being the 6-vector solution.

pcl::registration::TransformationEstimationPointToPlaneLLSWeighted - same as above, but the point pairs have non-constant weights that depend on the squared distance from the sensor.

pcl::registration::TransformationEstimationSVD - singular value decomposition approach for solving for the transformation that minimizes the point-to-plane error metric.

pcl::registration::TransformationEstimationPointToPlane - based on [DF01], uses the Levenberg Marquardt non-linear optimizer to minimize the point-to-plane error.

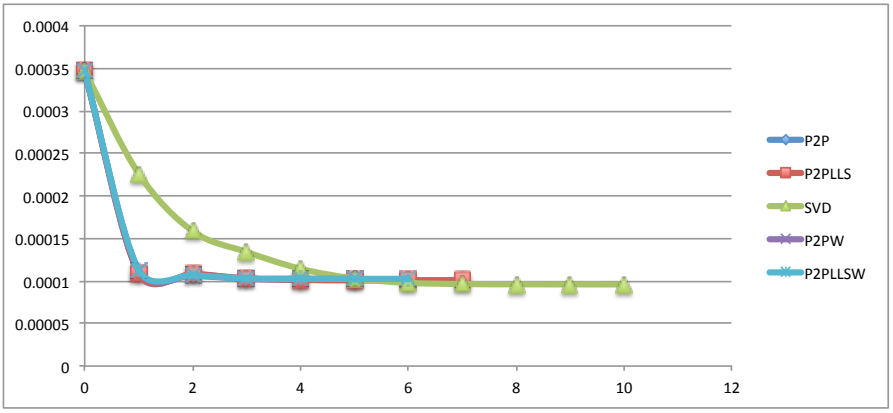
pcl::registration::TransformationEstimationPointToPlaneWeighted - same as above, but the point pairs have non-constant weights, that depend on the squared distance from the sensor.

The results are shown in Figures 2.13, 2.14, 2.15, as plots of the number of iterations against the MSE between the source and the target clouds for all the transformation estimation methods (both weighted and unweighted), using the three proposed correspondence estimation and rejection setups. The SVD method performs the worst, taking more iterations to converge for the first pair of clouds. For the second pair, it does not converge to the global minima with the *hard* and *medium* difficulty correspondences, but manages to with the *easy* clean ones (proving the importance of randomization in the transformation estimation procedure). For the third pair of clouds, it never manages to converge, no matter what correspondences we use. P2PLLS works rather well on all the situations, except for the distant pair at the end, where it converges to the wrong local minima without the randomization. The Levenberg-Marquardt-based point-to-plane transformation estimation (P2P and P2PW) is the best performer, managing to converge to the correct solution in all the situations, and with less iterations than the other methods.

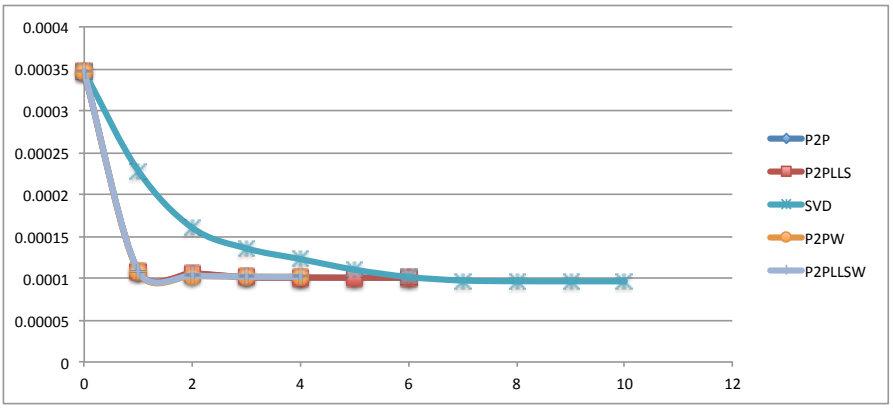
Unfortunately, the last series of tests did not prove the usefulness of weighting the point pairs using the Kinect noise model, because the weighted versions of P2P and P2PLLS perform very closely to their unweighted counterparts. As such, we will opt for a scene with a wider variance of depths, ranging from 0.8 to about 5 meters (see Figure 2.17). Theoretically, the noisy points far away from the camera should prove difficult to the transformation estimation methods that consider uniform weighting. This is confirmed by the results we got from our runs in Figure 2.16, as both the unweighted versions did not converge to the global minima, and the weighted ones did.

2.5 Stopping Criteria and Transformation Validation

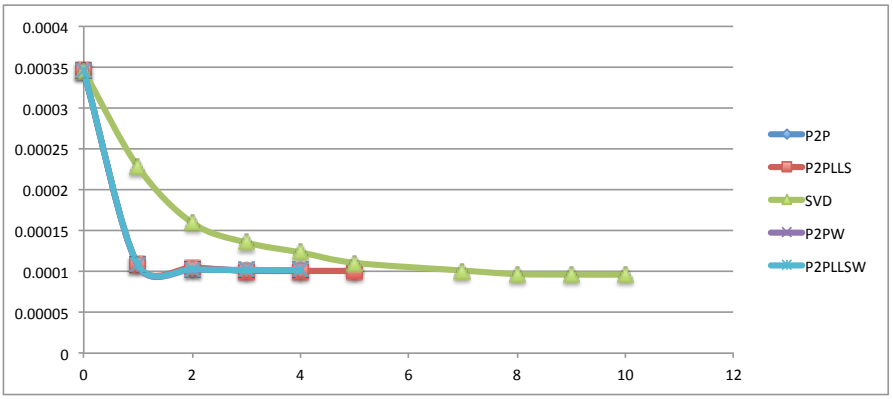
The registration process is an iterative algorithm, so we need a set of criteria to stop it by, and to determine whether it was stopped due to the fact that it converged or it is irrevocably



(a) difficult correspondences

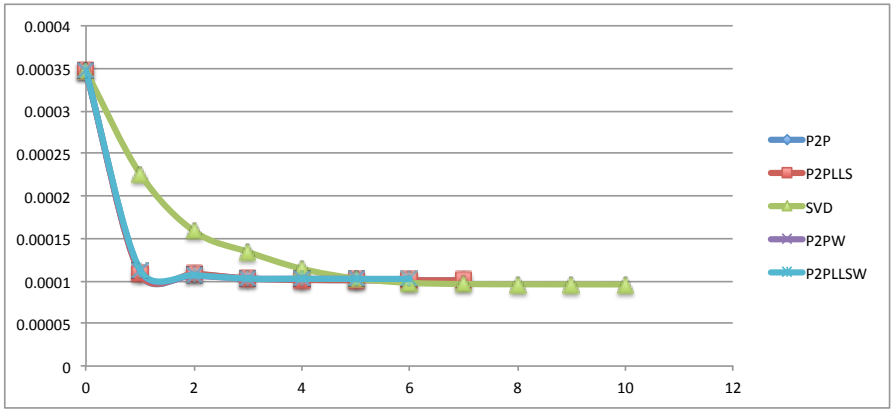


(b) medium difficulty correspondences

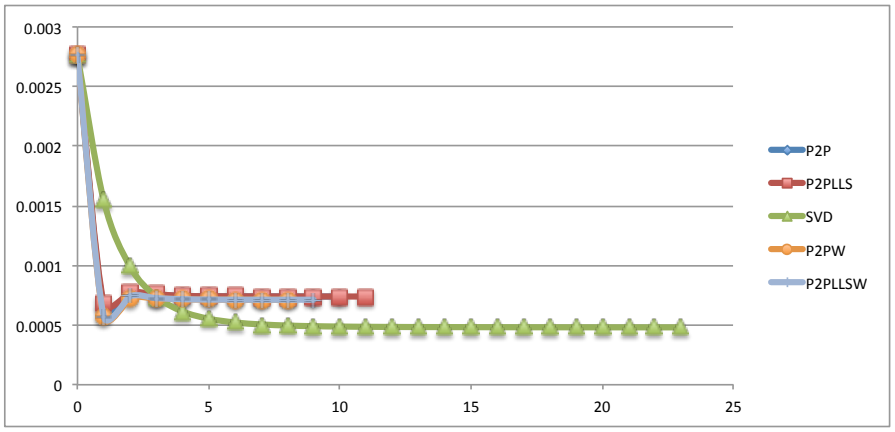


(c) medium + RANSAC

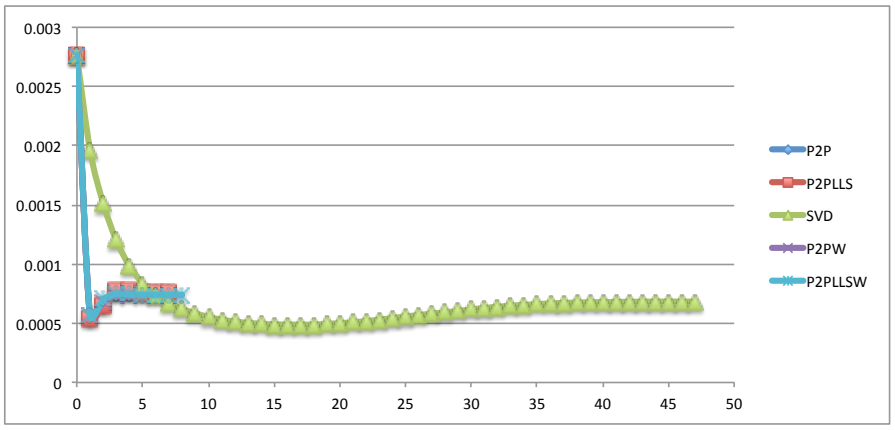
Figure 2.13: The progress of the different transformation estimation methods, plotted as the iteration number against the MSE measured in meters, for the first tested pair of clouds.



(a) difficult correspondences

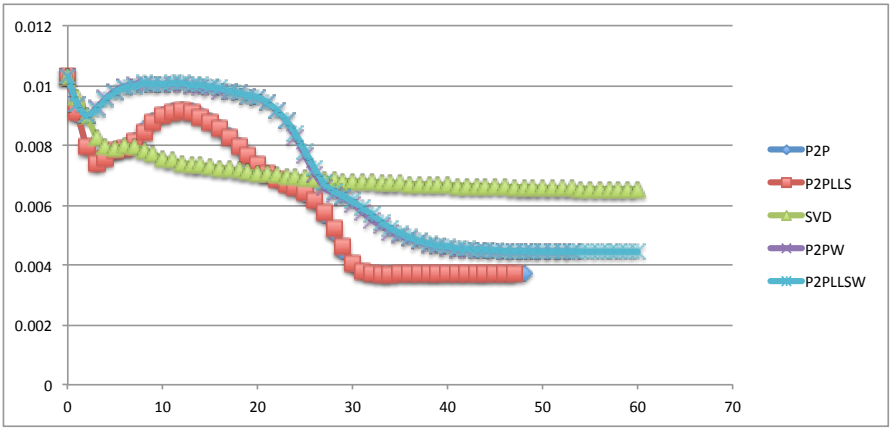


(b) medium difficulty correspondences

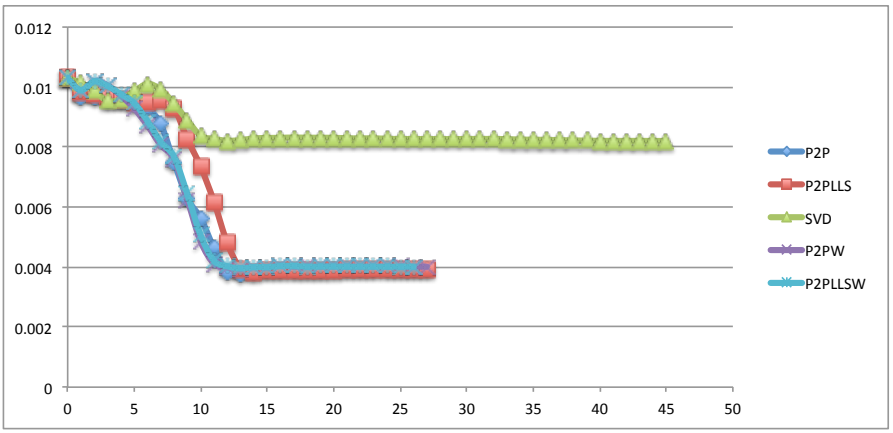


(c) medium + RANSAC

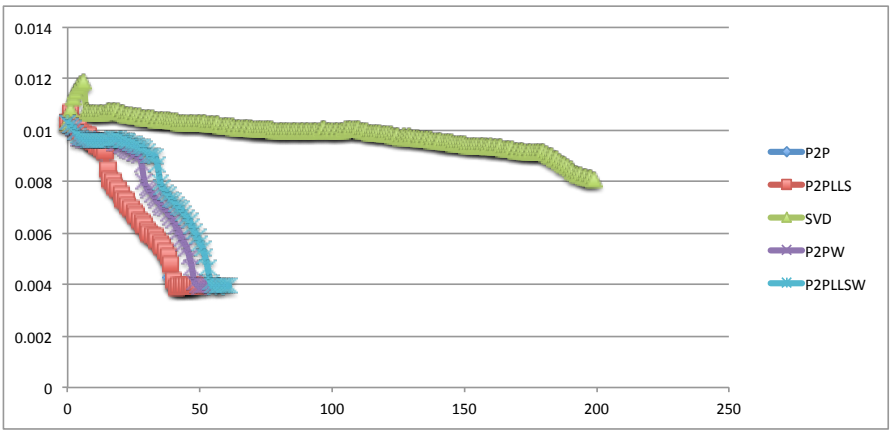
Figure 2.14: The progress of the different transformation estimation methods, plotted as the iteration number against the MSE measured in meters, for the second tested pair of clouds.



(a) difficult correspondences



(b) medium difficulty correspondences



(c) medium + RANSAC

Figure 2.15: The progress of the different transformation estimation methods, plotted as the iteration number against the MSE measured in meters, for the third tested pair of clouds.

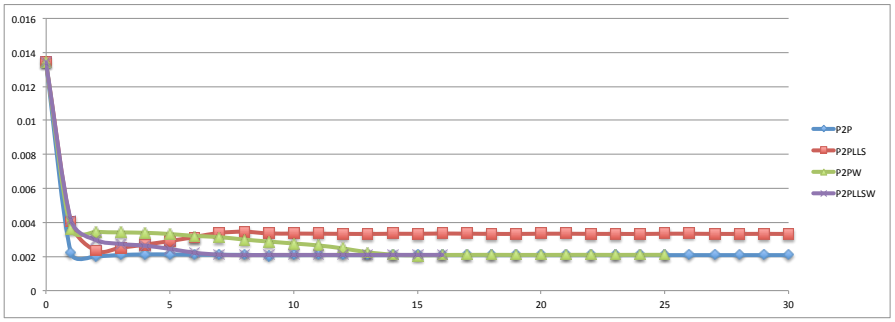


Figure 2.16: Plot showing the iteration number against the MSE for the Levenberg-Marquardt point-to-plane transformation estimation (P2P), and the linear least squares point-to-plane optimizer (P2PLLS), as compared to their weighted versions (P2PW and P2PLLSW). Please note that the unweighted variants did not converge in the global minimum, leading to a wrong transformation. Both weighted versions gave the correct results.

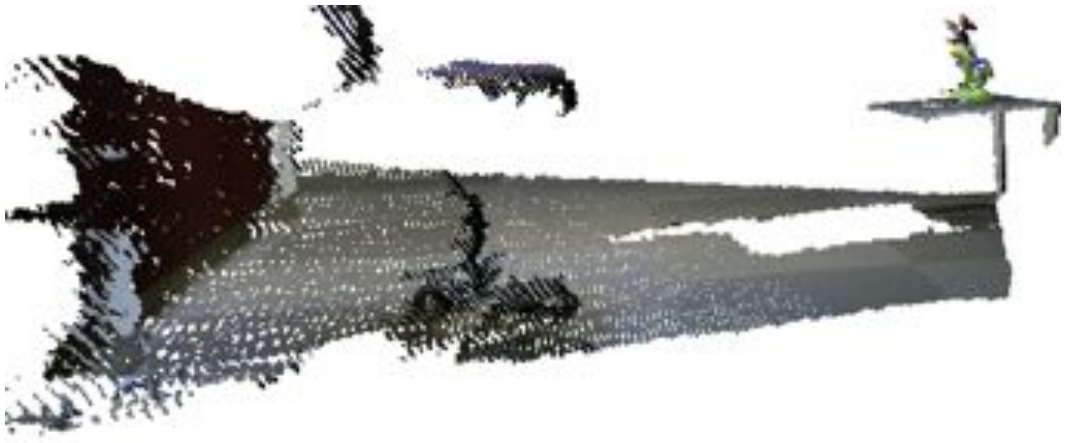


Figure 2.17: Example scene with a wider range of depths, that will be used to test the improvements brought by the weighting of the point pairs in the transformation estimation procedure.

diverging, and any number of supplemental iterations will turn it towards a correct solution. The actual methods used are implemented in **pcl::registration::DefaultConvergeCriteria**:

maximum number of iterations Exceeding the number of iterations means that the optimizer diverged in most of the cases. This threshold has to be tuned depending on the complexity of the registration problem (expect that registering a pair of scans that are far away from each other will require more iterations, than two scans that have a good initial alignment). (Figure 2.18a)

absolute transformation threshold Stops the ICP when the currently estimated transformation (rotation and translation) is over a certain value. This is an early termination criteria for optimizations that are diverging. The intuition behind it is that the two clouds to be aligned are expected to be within a certain range of distances from each other, and so transformations that are outside that range need to be rejected (e.g., two consecutive handheld-Kinect scans recorded at 30Hz can not be more than 10 cm and 20 degrees apart from each other). (Figure 2.18b)

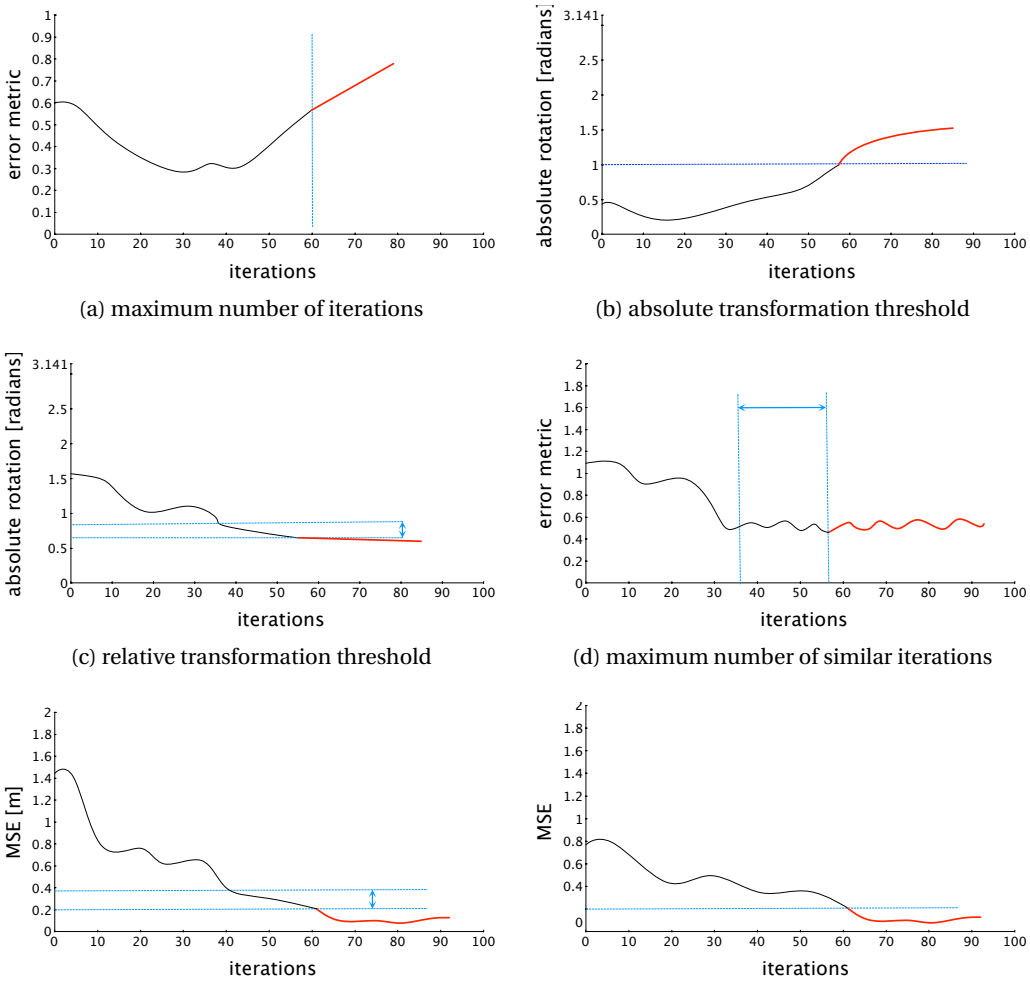
relative transformation threshold Specifies what is the minimum transformation step from one iteration to the next that is considered small enough for the optimizer to have converged. In other words, it is a balance between precision and time performance (important for tuning a system for real-time applications), as the threshold determines what transformation increment is considered to be good enough for the application. Also, it is possible that the minimizer never converges (i.e., null incremental transformation from one iteration to another), but produces small oscillations around the local solution indefinitely; this criterion is designed to stop the ICP when such a situation is reached. (Figure 2.18c)

maximum number of similar iterations The previous stopping criteria has the downside that a minimizer might temporarily seem to have converged, but it is actually oscillating in a local minima and has a chance of escaping it and converging into the global minima (or another local minima). For this reason, we allow the optimizer to consistently spend a number of iterations around a minima before considering it converged. (Figure 2.18d)

relative MSE This criterion is similar to the relative transformation threshold, using the mean square error metric instead of the rotation/translation increment. (Figure 2.18e)

absolute MSE Another absolute threshold that stops the registration when the error between the two aligned clouds is below a certain value. Sets the precision of the alignment in the detriment of performance. (Figure 2.18f)

Due to the high risk of converging into local minima when registering two point clouds, another series of checks is performed even if the ICP procedure confirms a successful convergence:



(a) maximum number of iterations

(b) absolute transformation threshold

(c) relative transformation threshold

(d) maximum number of similar iterations

(e) relative MSE threshold

(f) absolute MSE threshold

Figure 2.18: Graphical explanation of each of the ICP stopping criteria that we employed in our application.

percentage of valid dexels The clouds we are registering should not have a lot of NaN depth values (pixels in the depth map for which the sensor could not triangulate the depth). As such, we check if the ratio between the number of valid dexels and the total number of dexels in the image is above a certain threshold (around 60% proved to be a good value). Such a test can be performed before the ICP routine is ran on the source and target clouds individually.

percentage of inliers If the previous check is passed, the registration overlapping region is measured. In order to do that, the source cloud is transformed with the resulting ICP transform and projected into the target image. We go through every dexel of the target image and check the depth difference between it and the projected source point, counting the number of inliers (within an adaptive threshold based on depth of 5 cm for a z-value of 1 m). The ratio of inliers to valid dexels should be around 80%. This operation is similar to raytracing and checking if the empty space of the source point cloud corresponds to the one of the target cloud. See Figure 2.19 for a visual explanation.

motion limit A simple check is done to make sure that the relative transformation between the two frames is not exaggerated, which suggests convergence into an incorrect minimum. It uses rotation and translation thresholds, which need to be tuned based on the expected motion of the camera. In the case of our handheld Kinect scenario, we reject rotations larger than 30 degrees between consecutive frames and translations greater than 30 cm.

stability of the overlapping regions If all the previous tests have passed, a check on the stability of the overalpping area is performed to make sure that the clouds do not slide against each other, inspired by [GR03]. The 6D covariance matrix of the inliers is computed and the condition number of this matrix is verified to be under a certain value. We ran experiments on a number of datasets and concluded that a value of about 100 is good for stable frame to frame registration. Similarly, one could verify the trace of the information matrix to be larger than a given threshold, as it is loosely correlated to the condition number.

2.6 Conclusion

In this chapter we have presented various approaches for the steps necessary to register two point clouds. We started with noise reduction, extracted normal information and then subsampled the point sets in order to speed up the subsequent computations without hurting the alignment quality. Various techniques for estimating and filtering correspondences were analyzed and benchmarked, then fed into an optimizer to obtain the transformation from the source to the target frame. We then looked into ways of determining how many times we need to run the above sequence in an iterative framework.

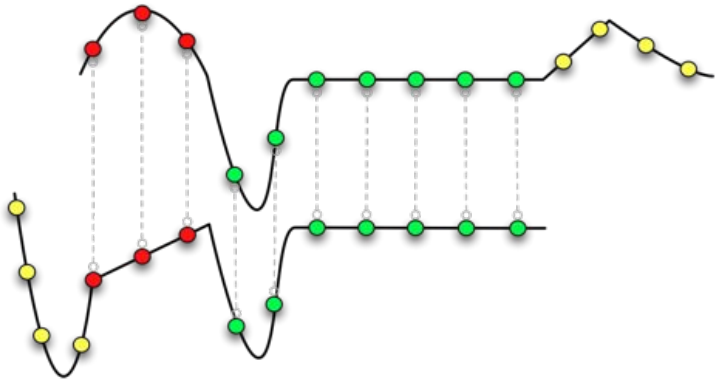
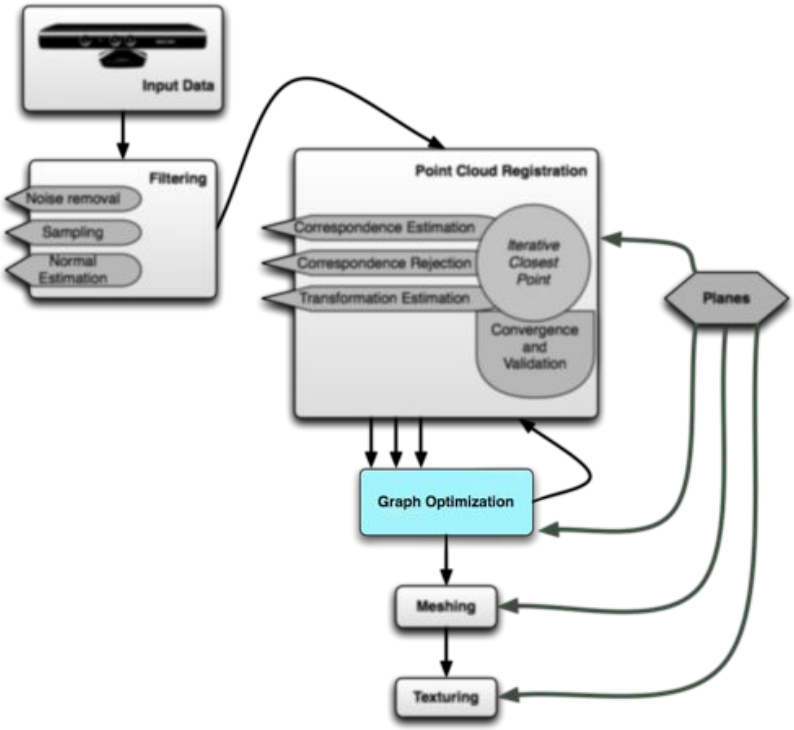


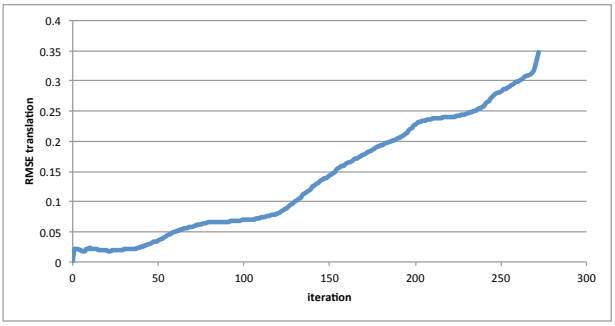
Figure 2.19: Transformation validation by looking at the ratio of inliers in the overlapping regions. If two surfaces match locally, we need to make sure that their empty space also matches and that there is no additional depth information in the overlap region of any of the clouds that does not have correspondences in the other cloud.

3 Graph optimization



3.1 Introduction and Related Work

In the previous chapter, we discussed about ways of efficiently and robustly registering a pair of point clouds, along with analyses on how each design decision in the pipeline influences the results. We saw that employing incremental registration will lead to quick error accumulation, creating unacceptably erroneous maps only after a couple of hundred frames (less than 10 seconds of scanning at 30Hz) - see Figure 3.1. Any error in the estimation of the transformation between a pair of two consecutive clouds will be passed on to all the subsequent clouds (Figure



(a) translational error accumulation

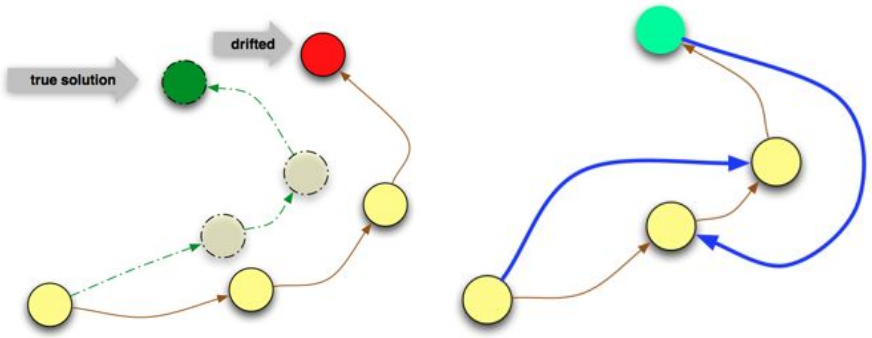
(b) incremental registration map

(c) ground truth map

Figure 3.1: Incremental registration produces considerable drift. The graph on the left shows the evolution of the translational error of incremental registration as compared to the ground truth; the last two images show the resulting map and the ground truth map.

3.2a). The 3D camera we are using outputs data at 30Hz, meaning that in a handheld scenario the scans will have a good overlap with their neighbors on the timeline. This immediately leads to the idea of doing additional registrations between non-consecutive scans in order to correct for the precision of the absolute localization of each scan (Figure 3.2b). These additional edges are meant to pull the track from drifting away, but the problem becomes overdetermined.

We are now facing an over-constrained problem which we shall solve in the least-squares sense. This problem is often visually represented as a graph, where the nodes are the variables that needs to be solved for and the edges represent the constraints (see Figure 3.3). In the case we have described in the previous paragraph, we will be using a pose graph, which aims at solving Equation 3.1, where x_i are the 6D poses of the cameras, z_{ij} are the measurements between two camera poses x_i and x_j, as 6D transformations. To avoid problems caused by singularities (e.g, Gimbal lock) that the non-Euclidean space of 3D rotations has, the optimizer uses an over-parametrized representation for rotations (e.g., rotation matrices or quaternions)



(a) incremental registration drift in graph (b) with additional edges to correct for the drift

Figure 3.2: The effect of accumulated drift in a pose graph, and correcting it by adding additional links between overlapping poses.

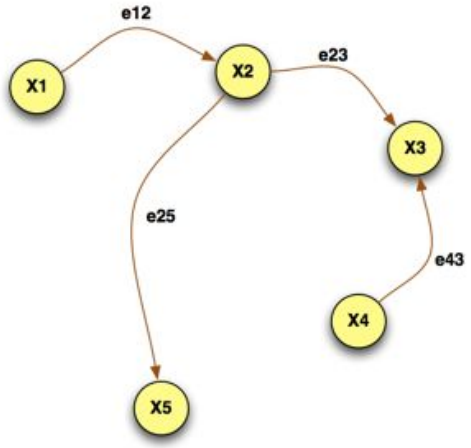


Figure 3.3: Graph representation of the example objective function in Equation 3.2.

instead of Euler angles.

$$\begin{aligned} x^* &= argmin_x[F(x)] \\ F(x) &= \sum e(x_i, x_j, z_{ij})^T \Omega_{ij} e(x_i, x_j, z_{ij}) \\ e(x_i, x_j, z_{ij}) &= T(z_{ij})T(x_i)T(x_j)^{-1} \end{aligned} \tag{3.1}$$

$$\begin{aligned} F(x) &= e_{12}^T \, \Omega_{12} \, e_{12} \\ &+ e_{23}^T \, \Omega_{23} \, e_{23} \\ &+ e_{25}^T \, \Omega_{25} \, e_{25} \\ &+ e_{43}^T \, \Omega_{43} \, e_{43} \end{aligned} \tag{3.2}$$

In order to solve this least-squares problem, the error function is linearized by approximating it to its first order Taylor expansion:

$$e_{ij}(\breve{x}_i + \Delta x_i, \breve{x}_j + \Delta x_j) = e_{ij}(\breve{x} + \Delta x) \simeq e_{ij} + J_{ij}\ \Delta x \tag{3.3}$$

where J_{ij} is the Jacobian of $e_{ij}(x)$. Equation 3.1 now becomes:

$$\begin{aligned} F_{ij}(\breve{x} + \Delta x) &= e_{ij}(\breve{x} + \Delta x)^T\ \Omega_{ij}\ e_{ij}(\breve{x} + \Delta x) \\ &\simeq (e_{ij} + J_{ij}\ \Delta x)^T\ \Omega_{ij}\ (e_{ij} + J_{ij}\ \Delta x) \\ &= e_{ij}^T\ \Omega_{ij}\ e_{ij} + 2e_{ij}^T\ \Omega_{ij}\ J_{ij}\ \Delta x + \Delta x^T\ J_{ij}^T\ \Omega_{ij}\ J_{ij}\ \Delta x \\ &= c_{ij} + 2b_{ij}\ \Delta x + \Delta x^T\ H_{ij}\ \Delta x \end{aligned} \tag{3.4}$$

Plugging this back into the main equation:

$$\begin{aligned} F(\breve{x} + \Delta x) &= \sum_{i,j} F_{ij}(\breve{x} + \Delta x) \\ &\simeq \sum_{i,j} c_{ij} + 2b_{ij}\ \Delta x + \Delta x^T\ H_{ij}\ \Delta x \\ &= c + 2b^T\ \Delta x + \Delta x^T\ H\ \Delta x \\ c &= \sum c_{ij} \\ b &= \sum b_{ij} \\ H &= \sum H_{ij} \end{aligned} \tag{3.5}$$

The whole system then reduces to:

$$H\Delta x^* = -b \tag{3.6}$$

Which is solved by incrementing the initial guess with Δx^*:

$$x^* = \breve{x} + \Delta x^* \tag{3.7}$$

This can be done using the Gauss-Newton algorithm which will iteratively apply the last three equations. Another option is the Levenberg-Marquardt algorithm which adds a dynamically-controlled damping factor λ to control the Gauss-Newton convergence, Equation 3.6 is trans-

formed into:

$$(H + \lambda I)\, \Delta x^* = -b \tag{3.8}$$

This challenge or other similar ones have been encountered in SLAM research, resulting in a variety of approaches and available code for tackling them. The OpenSLAM community [SFG] is gathering libraries, such as: HOG-Man [GKS*], TORO [GSGB07], iSAM [KRD08], G2O [KGS*11]. Due to their popularity and active development, we have decided to use the last two for our application wrapped under a common API. Published RGB-D mapping systems use various optimizers without motivating their choices: [TRC12] use GTSAM, [HKH*10] use TORO, [SEE*12] use G2O, and [PKDB10] build their own optimization framework.

3.2 Pose Graphs for Kinect Mapping

3.2.1 Incremental Construction of the Pose Graph

One of the targets of our system is to construct an architecture that is fast enough for real-time applications. Ideally, to obtain good maps, one would need as many edges in the pose graph as possible (maximum of $N\ (N-1)$ edges, equivalent to having registered all the N frames against all the other frames in the sequence). Such an approach would make the computations intractable for longer sequences of clouds. A careful selection of the frames to be registered and the contents of the graph to be optimized needs to be done. Each frame is registered with its predecessor and the new edge is added to the graph if the registration was successful (i.e., converged). In order to reduce drift, as explained in the previous section, additional links between camera nodes have to be added.

As a result, the concept of *keyframes* is introduced; an ordinary frame is considered a keyframe if it is far away from the other keyframes in terms of angle, translation or time (meaning that the minimum distance in any one of the angular, translational, or temporal spaces is larger than a threshold). This guarantees a uniform distribution of frames in the scene, such that most of the surfaces are covered, but the graph is not cluttered to ensure good optimization performance. Another setup used in [HKH*10] is to consider a frame as being a keyframe when the RGB features from the previous keyframe do not have sufficient correspondences in the new frame. We considered our approach to be semantically similar, but with a much lower computation cost as compared to checking for correspondences in feature descriptor space or computing the overlap between two point clouds. Oppositely, [SEE*12] do not use keyframes per se, but match each frame with the most recent 3 frames and 17 previous frames obtained by uniformly subsampling in time space.

Due to its step-by-step build-up, the construction approach of the pose graph allows for apply-

ing more efficient incremental graph optimization algorithms such as iSAM [KRD08],[KJR*11], as opposed to batch optimizers (such as G2O [KGS*11] or SAM [DK06]) that consider the previous optimization results as a mere initial guess for the new whole-graph optimization. iSAM uses QR factorization of the sparse smoothing information matrix, and recalculates only the values that actually change, being proven to be faster than its batch processing counterparts.

Algorithm 1 summarizes the steps of our incremental pose graph construction.

Algorithm 1 Pose graph construction algorithm

while ($cloud_{current} \leftarrow$ nextCloud ()) ! = NULL **do**
  matrix $covariance$, $transform_{pair}$;
  if !(register ($cloud_{current}$, $cloud_{previous}$, $transform_{pair}$, $covariance$) **then**
    skip cloud & continue to the next cloud;
  end if
  $graph$.addEdge ($cloud_{current}$, $cloud_{previous}$, $transform_{pair}$, $covariance$);
  $transforms[cloud_{current}] \leftarrow transforms[cloud_{current}] * transform$;
  if isKeyframe ($cloud_{current}$) **then**
    $keyframe$.add ($cloud_{current}$);
    for each $keyframe_i$ **do**
      $cloud_{current,transformed} \leftarrow transforms[keyframe_i].inverse() * transforms[cloud_{current}] * cloud_{current}$;
      if register ($cloud_{current,transformed}$, $keyframe_i$, $transform_{pair}$, $covariance$) **then**
        $transform \leftarrow transform_{pair} * transforms[keyframe_i].inverse() * transform[cloud_{current}]$;
        $graph$.addEdge ($cloud_{current}$, $keyframe_i$, $transform$, $covariance$);
      end if
    end for
    $graph$.optimize ();
    $transforms$.updateTransforms ($graph$);
  end if
end while
output $transforms$;

3.2.2 Loop Closures

The concept of loop closures is ubiquitous in SLAM. It refers to robustly determining when the current view of the scene exposes parts that have been seen before in the sequence and allows for a re-alignment between selected frames such that all the errors that have been accumulated are relaxed. Systems such as Kinect Fusion [IKH*11] can be implemented very efficiently because they use a global structure for storing all the previous registration results. This structure called truncated signed distance function (TSDF) is built incrementally by including the newly registered frames in an irreversible fashion; no links are kept between the original frame and the TSDF, and no optimization on the previous camera poses is ever done,

as this would involve the reconstruction of the TSDF from scratch. As a result, the output from Kinect Fusion is often very smooth due to the accumulation of errors in the TSDF, or the output is completely incorrect when the user attempts to close a larger loop: the running TSDF will look very different compared to the incoming frame, and an incorrect registration will happen.

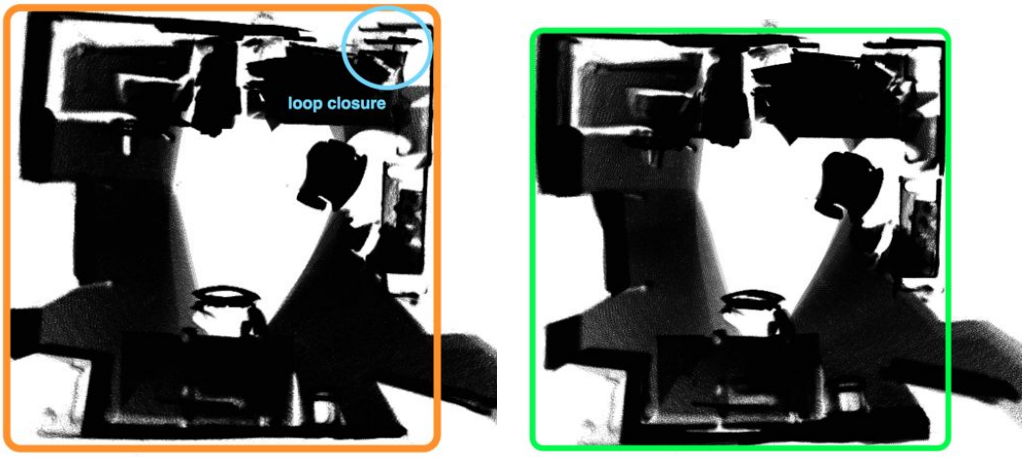
The RGB-D Mapping system [HKH*10] employ SIFT features extracted from the RGB images for an initial alignment between a new keyframe with all the other keyframes using RANSAC based on the SIFT feature matches - the advantage of this approach is that the initial poses of the two keyframes are not taken into account, so any amount of drift can be corrected; on the other hand, there can be false matches that can cause far away keyframes to be incorrectly matched and lead to breaking the graph. May et al. [MFD*09] are one of the first to publish about a SLAM system that uses only depth information from an early time-of-flight camera (SwissRanger SR-3000). They emphasize the importance of closing loops for error relaxation within a graph optimization framework. For this purpose, the authors used *GraphSLAM* [TM05], a seminal graph optimization library for localization and mapping applications.

In conclusion, there is no way of determining if two random frames represent the same view of the scene or not without trying to register them, without relying on local features. The only way of closing loops in the graph is by proximity, i.e., we try to match only the frames that are close together and have a good overlap, otherwise they are ignored (see Figure 3.4). Thus, it becomes essential not to lose track, emphasizing the need to do a lot of local adjustments between keyframes, guaranteeing that when we return to a previously seen part of the scene after a few thousand frames, we are not too off so that a loop is easily identified and closed. Our handheld scanning system does not target large scale mapping. In Chapter 5 we discuss how planar features can be used to aid in the loop closure process, and in Section 5.5.1 we propose a way of robustly registering two clouds with an unknown initial alignment by using planes.

A test is performed to prove that the techniques presented in this chapter are effective. A sequence of clouds is aligned with the naive incremental registration and with graph optimization and the translational errors are plotted. Figure 3.5 confirms the expectations of the graph optimization approach having a much smaller error gradient.

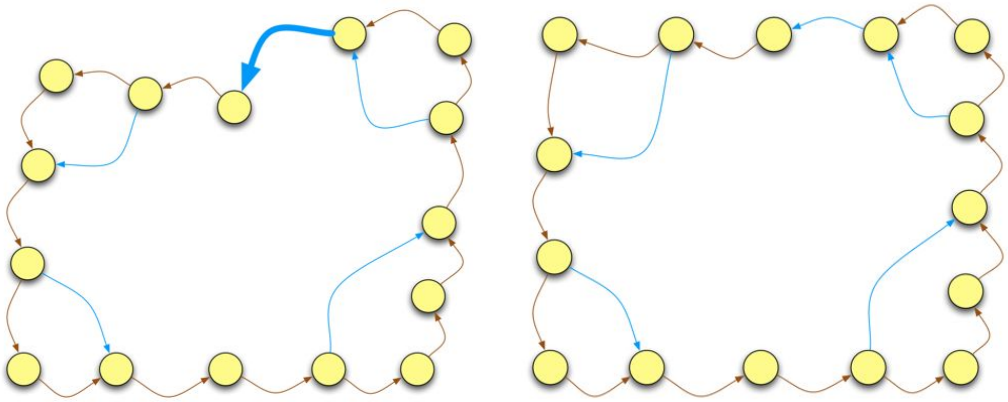
3.2.3 Edge Weights

As any optimization, different weights can be assigned to the constraints in order to ease the minimizer to the global minimum. For point-to-plane ICP we discussed about weighting the point pairs and its effectiveness, based on various heuristics such as Euclidean distance, angular difference of the corresponding normals, sensor noise model etc. In the case of pose graphs, this weight must reflect the importance of that certain edge in the graph. In their most general form, these weights are actually the Ω matrices mentioned in the basic pose graph equation to be solved (Equation 3.1). Ω is the information matrix (inverse of the covariance



(a) map before loop closure

(b) map after loop closure



(c) graph before loop closure

(d) graph after loop closure

Figure 3.4: The effect of loop closure on the graph optimizer and the resulting improved map.

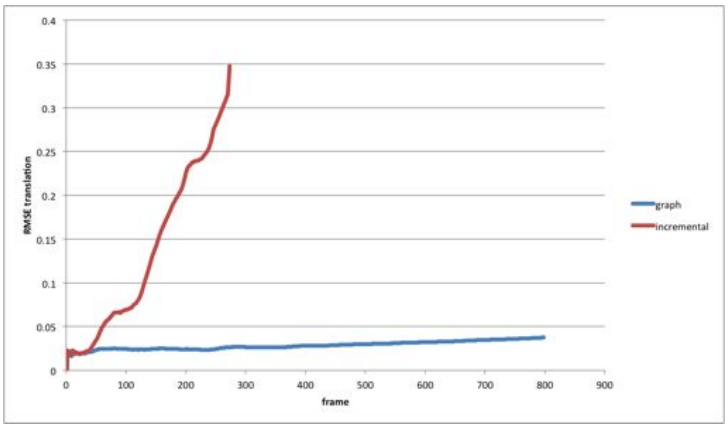


Figure 3.5: Error accumulation of incremental registration as compared to graph-based optimization with loop closures.

matrix).

We experimented with several options:

uniform weighting All the edges in the pose graph have the same weight; alternatively, one could increase the weight of the edges between keyframes in order to pull lower weighted incremental registration edges to a better local minimum. This weight is materialized as the isotropic information matrix: $\Omega = w\ I, w \in \mathscr{R}$.

isotropic weighting based on the pairwise registration There are several heuristics on how this can be approached:

- overlap between the two frames as a value between 0 and 1.
- condition number of the 3D (uses only the (x, y, z) coordinates of the points) or 6D ((x, y, z) and normal information) covariance matrices of the overlapping regions of the two frames.

true covariance matrix of ICP It can be computed in multiple ways:

- based on perturbating the pose with small Δ values in each direction, and recomputing the RMSE between the two clouds. In theory, $2^6 - 1 = 63$ perturbations are needed. This becomes too computationally expensive, so only six one-dimension perturbations are considered. The jacobian is computed as:

$$\left[\frac{\delta RMSE}{\delta x}, \frac{\delta RMSE}{\delta y}, \frac{\delta RMSE}{\delta z}, \frac{\delta RMSE}{\delta \alpha}, \frac{\delta RMSE}{\delta \beta}, \frac{\delta RMSE}{\delta \gamma}\right] \tag{3.9}$$

  and the covariance matrix becomes: $C = J\ J^T$. [DRT*04] use a similar method.
- covariance as described by Gelfand et al. [GR03]. It is not the actual covariance matrix of the ICP algorithm, but the covariance matrix representing the stability of the match between the two clouds (check Section 2.2.2 for the mathematical explanation).
- Censi et al. [Cen07] propose an approximation of the ICP covariance in closed-form.

From our test runs, the method that performed better is the point cloud stability covariance matrix, and is used in our final pipeline.

3.3 Optimizations

[SEE*12] propose a couple of methods of helping the graph optimizer converge and improve the results. Incorrect edges in the graph can cause significant changes to the topology of the map, which might produce additional wrong edges due to false loop closures, ending up in

a completely distorted model. A remedy for this is to remove 'bad' edges after every graph optimization. These edges are the ones that hold an error larger than a threshold, i.e., the difference between the edge measurement and the optimized variable nodes that the edges is connected too is high.

Another issue appears when the camera track is lost, as the current frame cannot be matched to its predecessors. Endres et al. argue that a sensible solution would be to fragment the graph by creating a new subgraph starting with the new frame and later on reconnect it if that area of the scene is scanned again. For evaluation purposes, they do not take this approach but add a highly uncertain edge between the frame that can not be matched and the previous one, with the relative transformation obtained from a constant velocity motion model. From our experiments, we decided that artificially keeping the graph connected is not a good solution for our system, as we do not make use of local features that can be matched without initial alignment, risking to create false edges in the graph. The conservative solution is to split the graph and output multiple models that can later be registered together separately (see Figure 3.6).

3.4 Global ICP

On top of the incrementally-built pose graph approach presented in the previous section, we opt for a subsequent processing step in order to improve the quality of the registration. This approach also uses a graph optimization technique, but with a slightly different architecture:

nodes Cameras poses are nodes in the graphs, variables represented as 6D transformations (same as for pose graphs).

edges Instead of having at most one edge between two camera nodes marking the registration result of the two frames (relative 6D transformation along with its information matrix), there will be multiple edges between two nodes, each edge materializing the point-to-plane constraint between one point from the source frame and one point and its normal from the target frame.

All in all, the optimization reduces to finding the camera poses for which the expression in Equation 3.10 is minimized. The variables are the rotations and translations (R_i, t_i) for each camera, solved by taking into consideration all the possible point pairs between all the cloud pairs. The weights $w_{i,j,k,l}$ control which pairs of points are taken into account and which not by setting the value to zero or to a value different than zero. The point pair selection and non-zero weighting is done similarly to the one for pairwise registration presented in the previous chapter, only that there is no pair-relative transformation estimation, but a single

(a) (b)

(c)

Figure 3.6: (a) and (b) show the resulting connected components of the map when the graph is fragmented due to unregisterable frames in the middle of the sequence such as the one in (c).

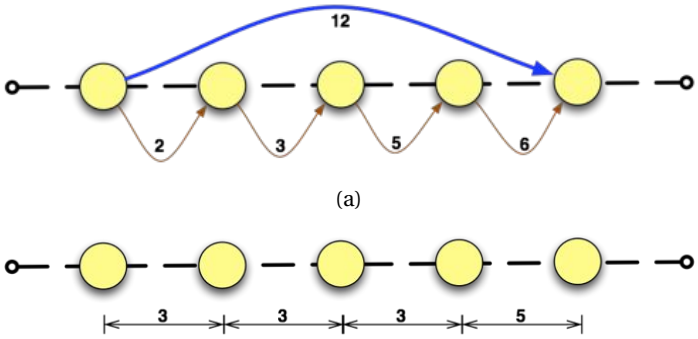


Figure 3.7: 1-dimensional loop closure example: (a) the pairwise measurements between the clouds - the incremental registration drifted: length $2+3+5+6=16$ vs 12; (b) the absolute positions of the nodes after the optimization - the length $3+3+3+5=14$ is a consensus between the incremental registration and the keyframe-based loop closure.

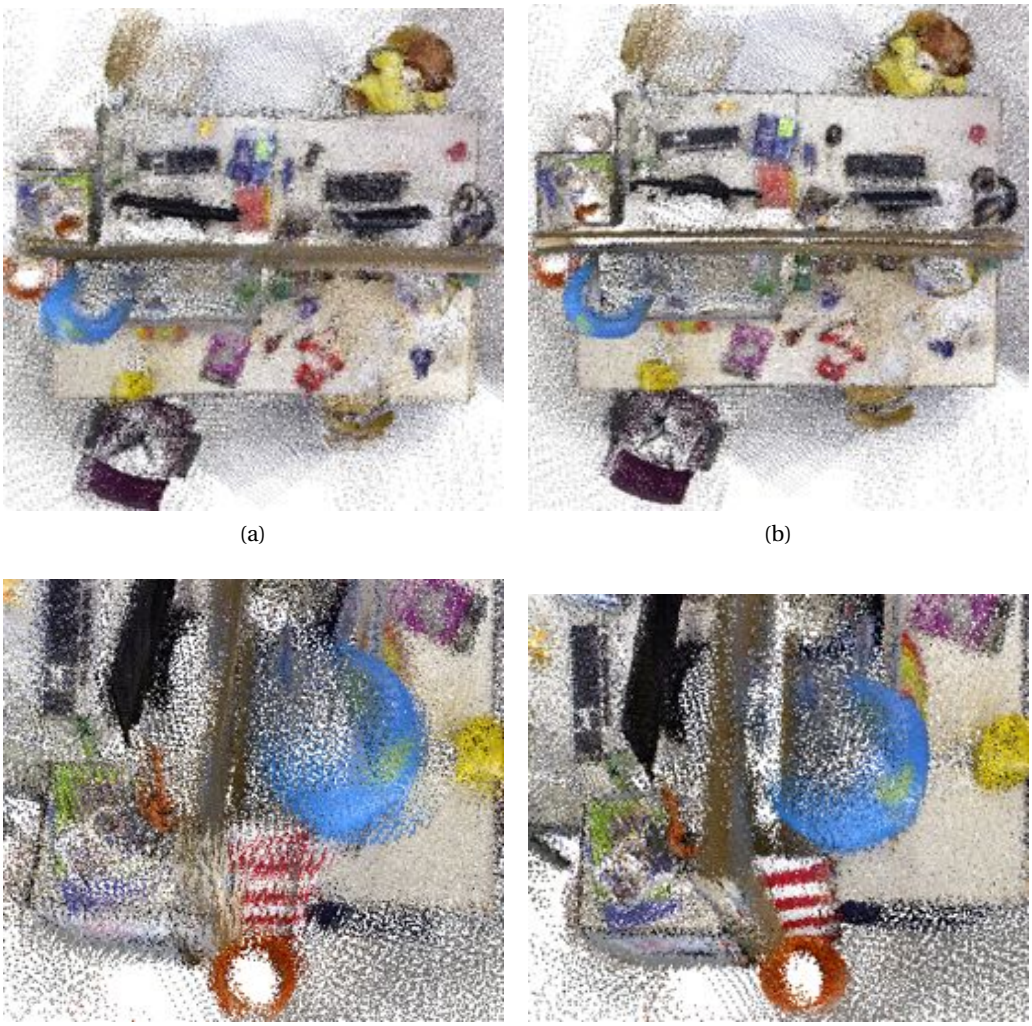
optimization done for all the cloud poses at once.

$$argmin_{R_i,t_i}(\sum_{clouds\ i}\sum_{clouds\ j}\sum_{points\ k}\sum_{points\ l}[[(R_i\ p_{i,k}+t_i)-(R_j\ p_{j,l}+t_j)]\cdot(R_j\ n_{j,l})\ w_{i,j,k,l}]^2) \tag{3.10}$$

The pose graph method explained in the previous section has the downside that the clouds are well-aligned on a high level (i.e., the map is topologically correct), but there are noticeable registration errors when looking at the model up-close. This is because the pose graph is sparse in order to keep the computation burden low. Figure 3.7 explains this issue for the 1D case: whenever a loop is closed (i.e., two keyframes are registered together successfully), the error accumulated by the incremental registration of the frames in-between the keyframes is relaxed along the graph based on the information matrices. Globally, the graph will go into a better minima, but locally there is no additional process being performed between the frames whose poses have been changed by the loop closure. This is the reasoning behind the global ICP approach, where we do a joint optimization of all the poses at once. An example of this situation on a real dataset is shown in Figure 3.8.

The main requirement of global ICP is that the point clouds already have good poses, as this approach will never converge to a topologically different map (or will require a large number of iterations to do so). Global ICP will only refine the poses such that the final map will get closer to the correct solution. Due to its high computation complexity, we do not use all the camera frames for this operation, but a smaller subset, 100-200 frames lead to good results.

Table 3.1 shows the improvement of the registration of a few datasets after this step.



(a) (b)

(c) (d)

Figure 3.8: The model before and after global ICP: (a) the map is topologically correct, but looking closer, (c) there are a lot of misalignments. (b) Global ICP does not change the topology of the map, but (d) local alignments are much better.

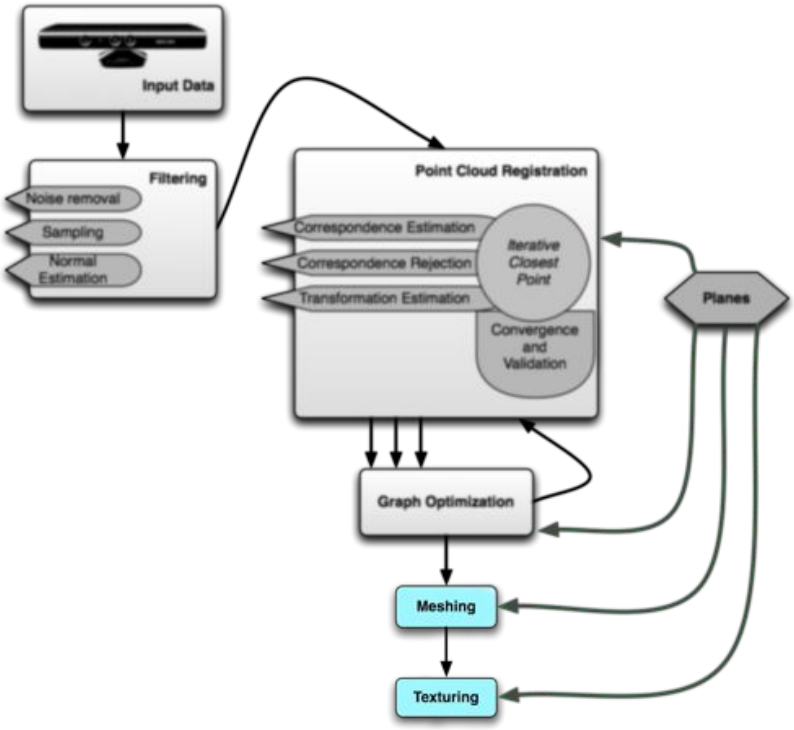
Dataset	Pose graph		Global ICP	
	Angular RMSE [degrees]	Translation RMSE [m]	Angular RMSE [degrees]	Translation RMSE [m]
freiburg1_360	12.93	0.337	9.76	0.201
freiburg1_desk	3.08	0.071	2.77	0.052
freiburg1_desk2	4.25	0.141	3.46	0.161
freiburg1_rpy	3.41	0.081	3.86	0.123
freiburg1_xyz	1.62	0.033	1.52	0.032
freiburg3_long_ office_household	4.53	0.098	4.25	0.068

Table 3.1: RMS errors when before and after global ICP.

3.5 Conclusion

After studying how two RGB-D frames can be registered together in the previous chapter, we looked into how to robustly align long sequences of point clouds in order to create coherent models. As such we opted for graph optimizers to relax the errors accumulated in the over-constrained system formed by incremental registration and the additional links between keyframes. Finally, we saw how effective it is to switch from a sparse pose graph to a graph that optimizes for the point correspondences.

4 Surface Reconstruction and Texture Mapping



4.1 Introduction and Related Work

The next step after having computed the poses for the cameras is to generate a mesh. There are numerous approaches for this in literature, and they can be split into two main categories: computational geometry and implicit functions. The Alpha Shapes [EM94] and Ball Pivoting [BMR*99] algorithms fall into the first category, but are considered impractical for point clouds of varying point densities, which are common for indoor reconstruction scenarios. In this same category is the GP3 algorithm [GK02], which is not suitable for our kind of data because it

will use all of the vertices in the input cloud in order to triangulate the mesh, creating artifacts in noisy areas or on surfaces that have been poorly registered.

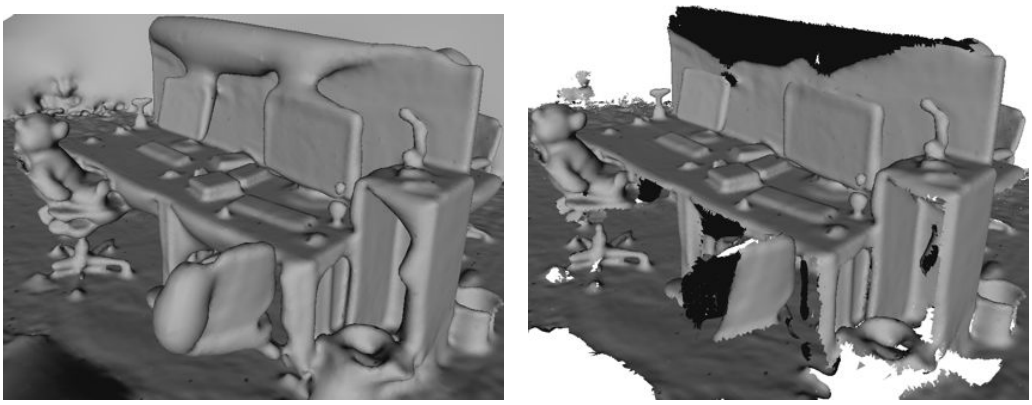
All the methods presented so far consider the input cloud as a general point set where all the points have equal confidence/importance in the final reconstruction. Weise et al. [WWLvG09] propose an incremental approach where the mesh is updated with each new incoming frame using a surfel-based technique that uses operations such as addition, removal and update, based on confidence and visibility heuristics. The authors target GPU implementations for their algorithm, and our CPU port proved to be too slow.

A similar approach is proposed in the KinectFusion system [IKH*11]: a truncated signed distance function (TSDF) is updated with every new incoming frame, and a marching cubes algorithm is finally applied on the TSDF in order to produce the output mesh. The downside is that the algorithm uses GPU-specific operations, making the CPU port highly inefficient. Furthermore, in order to cover large volumes (such as the ones expected in indoor scene reconstruction) in great detail (0.5 cm resolution), the TSDF will need more memory than what current hardware has to offer.

Because the surface reconstruction procedure should be in itself a noise reduction method, the algorithms based on implicit functions are better suited. The Poisson [KBH06] and Smooth Signed Distance [Tau12] algorithms approximate the mesh from points with normal information without the need for spatial partitioning heuristics and blending by using a hierarchy of locally supported basis functions. This reduces the problem to a sparse linear system which can be solved efficiently. Both methods proved to offer similar results, so we decided opted for the Poisson approach, as the implementation has a more convenient API at this time. This choice is confirmed by similar SLAM systems [PKDB10].

Due to the fact that implicit function methods tend to *hallucinate* (i.e., create surfaces that were not present in the input data), an additional post-processing step is carried out in order to clean up the mesh. For each triangle of the mesh, the points within its circumscribing sphere are counted. A triangle is considered valid if the local point density is higher than a threshold. Figure 4.1 shows an example result.

After a mesh representation of the registered point cloud is created, the mesh needs to be textured. In order to do so, we need to use the color images provided by the Kinect. Pitzer et al. [PKDB10] present a robotics system for autonomous floor plan reconstruction, describing in detail the way they do texture mapping on the resulting meshes. It is a 5-step process: *surface segmentation*, *unfolding*, *mesh re-parametrization*, *color reconstruction*, and *color blending*. The mesh is segmented into planar regions by region growing from random seeds and a merge step at the end to avoid over-segmentation. PCA is used to fit a plane through the vertices of each region and a local UV map is created for each such planar region. This UV map is just the projection of each vertex to the underlying plane. This guarantees the fact that there will be minimal distortions and no artifacts, but a downside is that the final model will contain numerous texture files. Next, the mesh is re-parametrized by associating each texture pixel to



(a) (b)

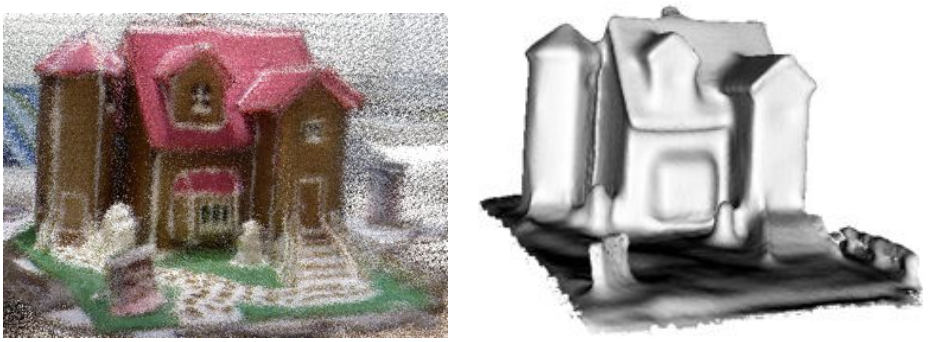
Figure 4.1: Meshing a point cloud with the Poisson surface reconstruction algorithm (a) without clipping onto the input cloud; (b) with clipping.

a 3D vertex in mesh space. This eases up the color reconstruction step where each of these vertices will be projected into the camera view and the color copied directly to the texture (this projection is done only after an occlusion check along the camera ray is done). The last step is done in order to account for the discontinuities that will be present in the texture due to the changing camera exposure settings during data acquisition. The authors formulate the multi-view blending problem as a Poisson problem and solve it for each of the texture images in their model.

In their publication [WJK*12], Whelan et al. discuss how to reduce artifacts when coloring the mesh. They noticed that the surface coloring is often inaccurate around edges, being caused by an imprecise calibration between the depth and color cameras, noise in the depth measurements or light diffraction around objects. The approach taken is to determine if a point is on a boundary by using a 7x7 mask, and by weighting in each color by the cosine between the normal at the surface and the incoming camera view direction.

For the purpose of demonstrating the algorithms in this section, we shall consider the gingerbread house dataset shown in Figure 4.2a. This was obtained from an Asus Xtion Pro camera and registered using our pipeline. The mesh in Figure 4.2b is obtained with the Poisson surface reconstruction algorithm [KBH06].

This chapter starts with the straightforward method of assigning a color property on each vertex of the mesh (not using a texture) in Section 4.2, after which we shall dive into a discussion on uv-maps in Section 4.3, and methods of transferring the color information from the Kinect data into the mesh in Section 4.4.



(a) (b)

Figure 4.2: (a) The Gingerbread house dataset, the colored registered point cloud; (b) The uncolored mesh computed by the Poisson surface reconstruction algorithm.

4.2 Vertex Coloring

Before going into proper texture mapping for our meshes, we shall first look into simply coloring the vertices of the mesh. This approach avoids the need for cumbersome uv-mapping techniques, the need to store and handle uv-coordinates and texture files. For example, the Kintinuous [WJK*12] uses the KinectFusion truncated signed distance function to accumulate color information for each vertex. For their 3d puppetry system [HGCA12], the authors describe a similar vertex coloring approach as the one we employed.

The way the renderer handles such meshes is to compute the color of each pixel by interpolating the colors of the vertices of the corresponding triangle - similar to the way normals and per-pixel brightness is computed in Phong shading. This leads to the immediate issue that the quality of the mesh coloring will depend on the number of faces in the mesh. One of the reasons for which texturing was introduced in the first place was to allow simulation of highly detailed objects with a reasonably low number of polygons. In the case of vertex coloring, not having a dense mesh leads to very blurry colors. Having sufficient polygons will solve this issue, but will put an unnecessarily heavy burden on the renderer. Furthermore, it is not efficient to represent planar surfaces (which are common in our indoor application domain) with a lot of faces.

The algorithm for coloring a mesh using this approach is fairly simple: each mesh vertex is projected into each RGB frame by using the camera intrinsics and the poses computed with our registration pipeline with an additional occlusion check. The color of each vertex is computed as a weighted average of the color of the pixels the vertex projects to. There are multiple weighting heuristics, explained in Section 4.4.2. Figure 4.3 shows vertex coloring on a high polygon mesh and its low-poly version.

(a) (b)

Figure 4.3: Vertex coloring on: (a) a mesh with a large number of faces; (b) its low-poly version

4.3 UV Maps

A UV map is a function that projects a 2D image texture to a 3D model, assigning 2D image coordinates to the 3D vertices of a mesh.

The basic method for computing the UV mapping is by trivial per-triangle parametrization (implementation provided in MeshLab [CCR08]). This method works by taking each triangle of the mesh and assigning it a patch in the shape of an orthogonal isosceles triangle in the texture. One can alleviate distortions by sorting the mesh triangles by their area and binning them into several groups, where each group is assigned a row with a height proportional to the area in the texture image. Figures 4.4a and 4.4b show a texture with its superimposed mesh triangles. The problem with this approach is that the neighborhood connectivity of the triangles in the mesh are is not taken into account when distributing the triangles in the texture image. Because the texture image is discretized into pixels with integer coordinates and the UV values are floating points, interpolations are done when rendering, so color bleeding from neighboring pixels in the texture image will occur. If these pixels do not correspond to triangles that are close in the mesh, then artifacts will be produced.

In order to overcome this issue, we looked into what professional 3D modeling software has to offer. Autodesk Maya 2013 [Aut13] offers four ways of computing the UV mapping for a mesh with some user interaction: planar mapping, cylindrical mapping, spherical mapping and the so-called automatic mapping. The first three are simplistic, and are recommended to be used only with non-complex meshes. The latter method tries to find the best UV placement by projecting the mesh into 4 orthogonal planes that can be positioned by the user. This results in a texture with more coherent and semantically meaningful triangle distribution, as shown in Figure 4.4c, and the mesh does not suffer of color bleeding between triangles as observed before (Figure 4.5).

(a) (b) (c)

Figure 4.4: (a) Trivial per-triangle UV map; (b) Space optimizing trivial per-triangle UV map; (c) Automatic mapping offered by Autodesk Maya

(a) (b)

Figure 4.5: (a) The Gingerbread house mesh where color bleeding between triangles can be seen due to the trivial per-triangle UV mapping; (b) Issue alleviated by the automatic UV mapping.

4.4 Color Transfer

After having a satisfactory UV map for the mesh, we proceed to coloring the texture. There are multiple ways in which this can be done. A very straightforward way is to transfer the color of the registered point cloud back to the mesh by connecting the corresponding 3D vertex of each texture pixel to its closest neighbor in the registered point cloud, as presented in Algorithm 2.

The variant explained above does not take into account the projective nature of the Kinect images, as it treats the registered point cloud as a general unorganized cloud. The view angle on the surface is not considered, which can lead to artifacts such as color discontinuities on the same triangle.

For this purpose, we suggest a second solution (Algorithm 3), which projects each triangle onto each camera image and does a per-triangle weighted average. Results can be seen in Figure 4.5. Notice in the algorithm that we are checking for occlusions before deciding whether to use the pixel in the camera image or not for its corresponding texture triangle. This can be done in multiple ways, the complete solution being ray tracing. Ray tracing is very expensive, so we will make use of the fact that we know the 3D coordinates of each of our camera pixels: we check if the 3D position in the mesh that we are assigning the camera pixel to is within a certain distance from the 3D position of the camera pixel. This will provide good results as long as we adapt the threshold to the scene we are processing: a threshold that is too large will cause close surfaces to share colors, and a threshold that is too small will not account for the deformations that the surface undertook when the point cloud was meshed. Furthermore, in our implementation we are rejecting all the camera pixels that have no depth reading. The data is dense enough and there are a lot of views of the same surface during a frame sequence, so discarding pixels will not leave uncolored areas.

Algorithm 2 The color transfer algorithm using the registered point cloud

for each mesh triangle **do**
    Compute homography from (u, v) texture coordinates to the barycentric coordinates of the triangle
end for

Set up kd-tree for the registered point cloud

for each texture pixel **do**
    Compute the triangle barycentric coordinates for the pixel center
    Compute the 3D coordinates using the barycentric coordinates
    Find nearest neighbor point p in point cloud
    Color pixel with the color of p
end for

Algorithm 3 The color transfer algorithm using camera projections

Generate the list of texture pixels corresponding to each mesh triangle
Dilate each pixel region by 1 pixel in each direction (to avoid floating point approximations)
for each camera image **do**
  $view\_direction \leftarrow T(camera) * Vector(0,0,1)$ ▷ T - camera transformation matrix
  for each mesh triangle (a, b, c) **do**
    $normal \leftarrow \frac{(a-c)\times(b-c)}{\|(a-c)\times(b-c)\|}$
    $cos\_view\_normal \leftarrow view\_direction \cdot normal$
    if $cos\_view\_normal \leq threshold$ **then**
      process next triangle
    end if

    $p\_[a,b,c] \leftarrow P(camera) * T(camera) * [a,b,c]$ ▷ P camera projection matrix
    Compute H - 2x3 affine homography matrix from the texture pixels to the camera image pixels

    for each pixel in the texture image corresponding to the current triangle **do**
      $proj\_[u,v] \leftarrow H * [u,v]$
      if $occlusion\_check(camera(proj\_[u,v]) \neq pass)$ **then**
        continue to next pixel
      end if
      $pixel\_camera\_color \leftarrow bilinear\_interpolation(proj\_u, proj\_v)$
      $texture(u,v) \leftarrow \frac{pixel\_camera\_color * cos\_view\_normal + texture(u,v) * weight(u,v)}{weight(u,v) + cos\_view\_normal}$
      $weight(u,v) \leftarrow weight(u,v) + cos\_view\_normal$
    end for

  end for
end for

(a) (b) (c)

Figure 4.6: Perspectively correct texture mapping: (a) flat; (b) texture mapping using affine transformation; (c) texture mapping using perspective transformation.

(a) (b)

Figure 4.7: (a) Affine texturing and (b) Perspectively correct on the Gingerbread house dataset.

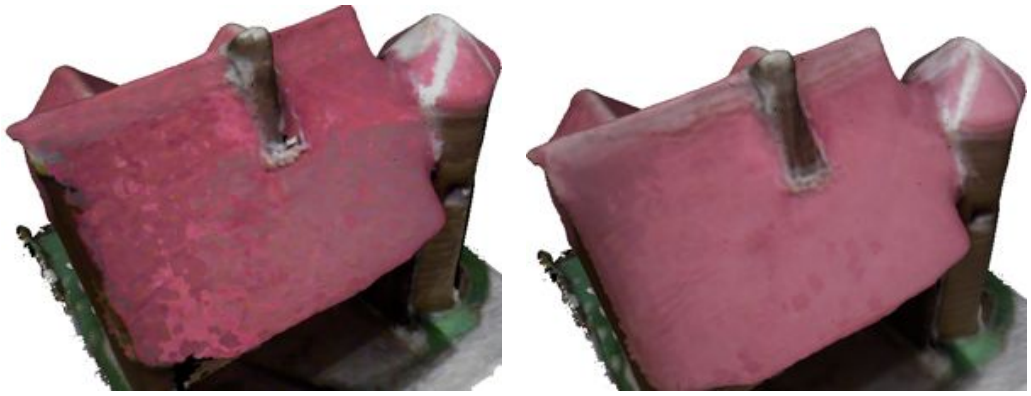
4.4.1 Perspectively Correct Texture Mapping

The solution we proposed computed a homography between the triangle in the texture plane and its corresponding triangle in each image plane, resulting in an affine transformation. This does not take into account the distortion of the texture introduced by not doing a perspective projection (See Figure 4.6 for a visualization of the effects of the two methods).

In order to correct for the foreshortening effects in the texture, the 3D coordinates for each pixel in the texture map are computed. This is done by interpolating between the mesh coordinates of each vertex of the face which the 2D pixel corresponds to. Then, for each camera image, instead of computing the UV location in an affine way, we project the 3D points. This dramatically improves the quality of the texture reconstruction, as exemplified on the Gingerbread house in Figure 4.7.

4.4.2 Heuristics for Weighting the Per-texel Contributions

The main advantage of averaging the colors for each texel is that there will be no sharp changes between neighboring faces in the mesh and the variance of the light during acquisition will be



(a) (b)

Figure 4.8: (a) Texture mapping without averaging - taking a single camera image sample for each texel - presents a lot of artifacts; (b) Texture mapping with weighted averaging by the triangle area in the camera image.

corrected for. The possible imprecisions in the registration process and erroneous meshing can cause the projected camera image patches to be misaligned on the mesh texture. The averaging approach fixes this, but causes blur.

Another source of blur is when a triangle from a frame where it does not have enough visibility (i.e., at a steep angle or far away from the object) is projected to the texture. In order to minimize this blur, we propose three weighting strategies:

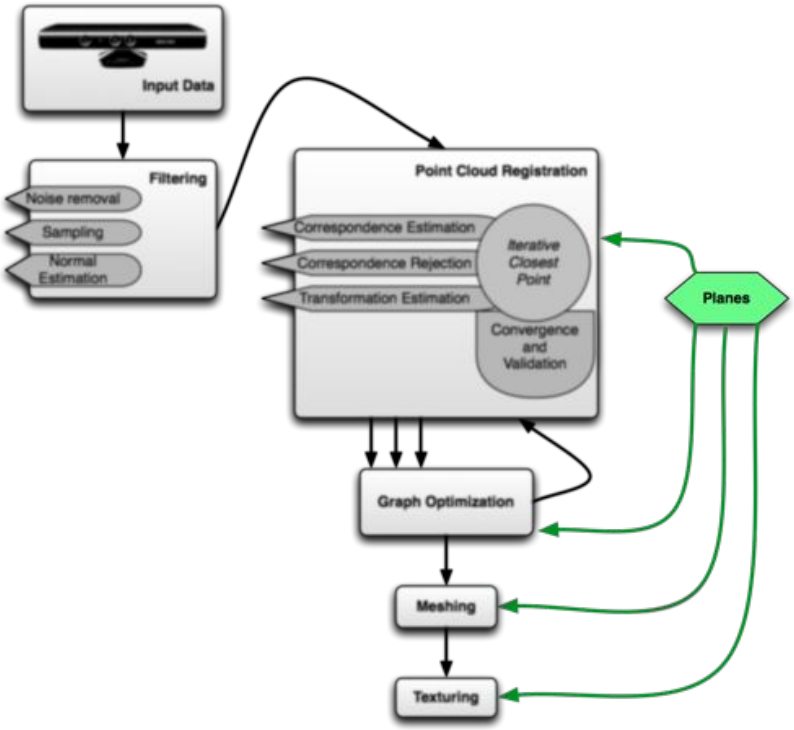
- weighting based on a function of the angle between the surface normal and the viewing direction of the camera (as in Algorithm 3)
- weighting using a function of the distance of the 3D point on the mesh triangle to the camera plane.
- a linear combination of the previous two.
- weighting based on a function of the pixel area of the triangle in the camera image. This has proven to be the best performer, as it guarantees that the larger contributions come from the triangles with the better visibility.

By not averaging, one can increase the sharpness of the texture by taking a single image source for each triangle. Figure 4.8 visualizes the result when taking the image where the mesh triangle projection had the largest area. As expected, there are discontinuities between neighboring triangles in terms of features and image brightness. Another similar approach we tried and presented similar issues was to take the median color for each texel from the total set of contributions corresponding to it. A rather complex post-processing step is needed to fix these artifacts, and this is out of the scope of this work.

4.5 Conclusion

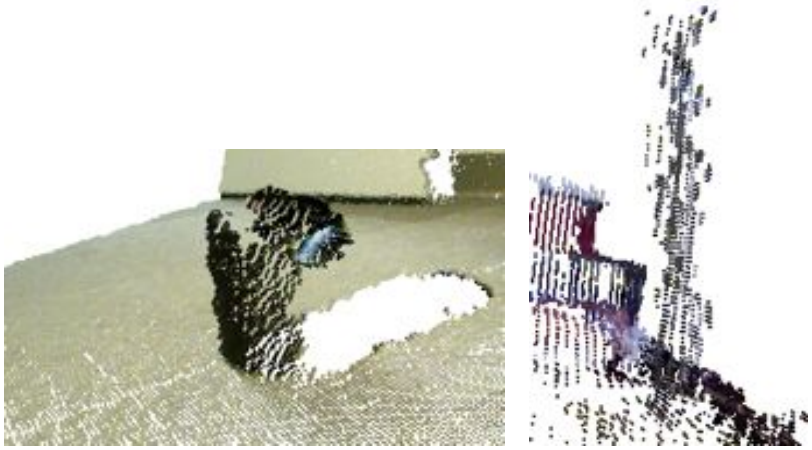
The point cloud obtained after registering sequences of RGB-D frames is not suitable for efficient visualization and usage in applications such as computer games or CAD. The standard data representation for such scenarios is that of meshes. As such, we looked into how we can convert large, discrete point clouds into continuous surfaces. Next, we tackled the problem of coloring these meshes, discussing various issues faced in the process.

5 Geometric Features, Planes



5.1 Introduction

Sparse local features have been proven to be useful for registering frames with no initial alignment, thus aiding in detecting loop closures in large maps. In most of the literature, these features are usually extracted from color images. In his work [RBB09], [RBMB08], Rusu proposes a number of feature descriptors based only on the geometry of the point clouds, which cumulate local neighborhood information about points and their normals in compact histograms. Other such descriptors include SHOT [TSDS10], NARF [SRKB11] and spin images



(a) (b)

Figure 5.1: High level of noise in the Kinect data, making it impossible to reliably fit cylinders in the scans.

[JH98]. All of these features were introduced in the context of laser scanning, where the data is a lot more dense and precise as compared to the Kinect cameras we are dealing with. As such, the requirements for our choice of geometric features are: speed, robustness to high levels of noise and invariance to the distance from the sensor (i.e., local point density). We have experimented with the PFH and FPFH descriptors available in PCL and concluded that they offer good performance only around the sweet spot of the sensor, between 0.8 and 1.5 meters. This is not satisfactory for our purposes, as we are expecting that most of the data collected for indoor mapping will be at a depth of around 3 meters.

Having eliminated this possibility, we turn towards the idea of semantic mapping introduced by Rusu [Rus]. The author detects geometric primitives (cylinders, spheres, planes etc.) in laser scans in order to add semantic meaning to the scanned environment: cups, doors, handles, shelves etc. Due to the high noise of the RGB-D sensors, tackling cylinders and spheres is difficult (see Figure 5.1). Rusu encourages the idea that indoor environments are mostly planar by doing an analysis on the distribution of his feature histograms in typical household scans. As such, this chapter will cover possible applications of planar features for the purpose of indoor mapping and modeling.

The sensors we are using only work indoors, so we can safely assume that in indoor mapping scenarios planes will be present in relatively large numbers. Object modeling scenarios also benefit from the usage of planes, as the objects to be modeled will sit on some kind of plane most of the times (e.g., desk, table, floor). Planes are easy to characterize: infinite planes are defined by 4 floating point values (the normal and the distance along the normal). In order to describe the shape of a planar surface, one can store the inlier indices from each of the point clouds that contribute to the plane. This option requires a lot of memory, so it is simplified by

reducing the descriptor to only the contour points, or the planar polygonal mesh for rapid visualization. Planes are relatively straightforward to detect in point clouds, and easy to match between consecutive frames.

There are multiple authors that proposed using planes extracted from point clouds in various ways. Nüchter et al. [NH08] mark planes as representing walls, floors, doors or ceilings in their laser scanned point cloud maps. Rusu et al. [RMB*09] use planar surfaces to add semantics to the point cloud maps, such as defining various fixtures (e.g., shelves, drawers) in indoor environments. Planar features are used for aligning individual point clouds in [PBV*10], and then globally optimizing a pose graph. Trevor et al. [TRC12] propose the usage of planes as landmarks in a graph optimization framework. For their results they used both 3D and 2D planes, captured by an RGB-D Asus Xtion Pro camera and a Hokuyo UTM-30 2D laser scanner which offers more precision and significantly larger range, and aided by the odometry of a Segway robot base. We will use some of their ideas in our system, but in a less constrained handheld camera context.

Planes become useful when used for aligning frames in various transformation estimation setups, can be used as landmarks for aiding graph optimizers to converge to the global minimum, or for constraining the ICP algorithm. Each of the sub-applications will be discussed in the sections of this chapter, after the methods for extracting, and manipulating clouds are presented.

5.2 Plane Extraction

We propose two ways of extracting planes from point clouds: an approach based on region growing, and one using RANSAC. Both come with advantages and disadvantages.

The first one uses the organized nature of the Kinect point clouds, as it generates random seed points in the image and then using the grid information, it grows regions from each seed point along the direction of the smallest gradient in terms of normal orientation, curvature and distance between points. An efficient implementation of this approach reaches realtime frame rates (30 Hz) with raw Kinect images (640x480), but the quality degrades a lot if lower resolution images are used. Furthermore, this method can output both the set of inlying points and the contour of the polygonal region. The main downside of this approach is that if a surface has a higher gradient of any kind, the algorithm will stop growing the respective region, making the parameter tuning very difficult, especially when considering the high amounts of noise present in the data. Figure 5.2 showcases two situations that are difficult to isolate.

The second method is based on RANSAC, and treats the frame as an unorganized point cloud. The sequence of operations is presented in Algorithm 4. The first part is the classic RANSAC approach for fitting a plane to a point set. Note that the number of iterations K can be determined using probabilities (Equation 5.1). After the inliers have been computed, an Euclidean clustering step is applied in order to take into consideration only the largest

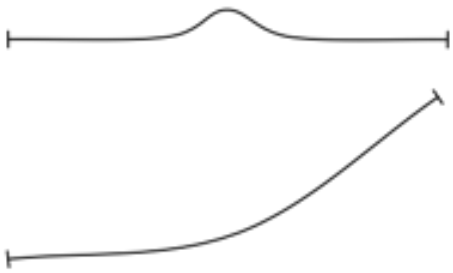


Figure 5.2: In the 2D examples above, the upper segment should be considered a single planar surface (and not two), and the lower segment should be eliminated as being non-planar. By using the region growing plane extraction approach it is hard to satisfy both cases, as the algorithm never computes per-cluster plane parameters during the region growing phase. It is only done once at the end. This means that the curvature gradient parameter should be large enough to allow the growing to go over the bump in the top segment, but small enough to not allow the lower segment to be grown at all.

component of the inliers. This is done because the RANSAC algorithm has no notion of neighboring points in the input cloud, and it considers disparate regions as being a single surface if they fit the same plane equation (e.g., desk tops at the same height in an office). We start from a set of seed points and grow and merge regions based on the distances between the 3D points (a normal deviation threshold can also be added). Figure 5.3 shows the effects of this step. Next, the plane parameters of this cluster are refined. This is done in an iterative fashion by computing the normal using principal component analysis (PCA), and then eliminating the points that have a projection distance to the new model larger than a multiple of the standard deviation. At the end of the process we are left with a set of planar inliers with less noise. Using them as such would be inefficient in terms of computation and storage performance, so we aim for a polygonal representation. To do so, we first compute the alpha shapes of the 2D projection of the inliers on the plane using the QHull library [BDH96]. This step will output a dense contour for each plane (thousands of points), so it needs to be simplified to reduce the contour to tens of points (See Figure 5.4 for the differences). For visualization purposes, the contours can be triangulated. The sequence of operations presented above extracts the largest plane in the point cloud. In order to extract the next largest plane, the inlier points of the previous plane are taken out of the point set and the whole procedure is repeated on the remaining points.

K = $\frac{\log(1-p)}{\log(1-w^n)}$, where:
n – the minimum number of points necessary to compute the model (3 for planes)
p – probability that we select only inliers when choosing n random points from the set.
w – the probability of choosing an inlier when choosing one random from the entire set.
= $\frac{number\ of\ inliers}{point\ set\ size}$

(a) (b)

Figure 5.3: Clustering the planar inliers helps concentrate only on a single surface at a time. (a) Before clustering, raw output from the RANSAC plane fitting; (b) The largest cluster only.

(5.1)

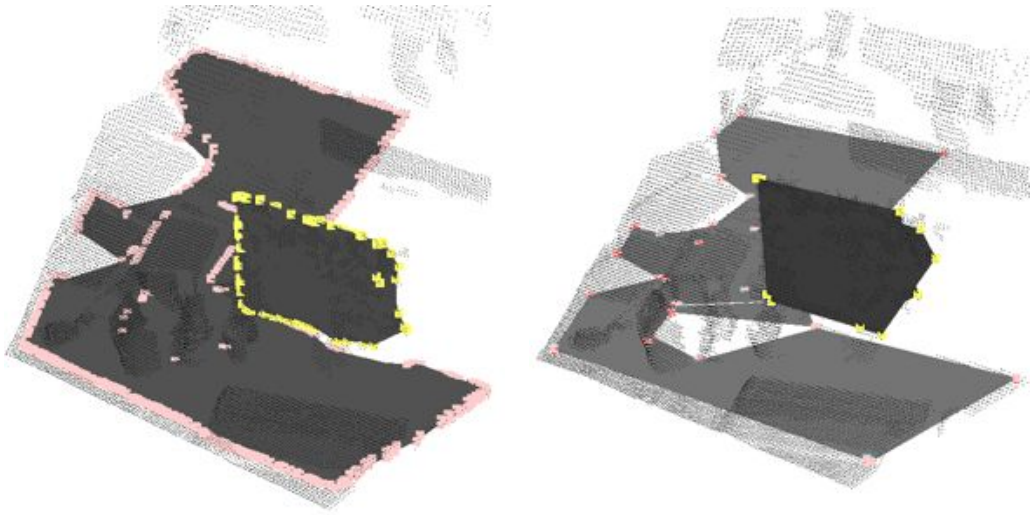
$$\begin{array}{rcl} & & \text{Given 3 points } v_1,\, v_2,\, v_3\text{:} \\ n & = & \frac{(v_1-v_2)\times(v_3-v_1)}{||(v_1-v_2)\times(v_3-v_1)||} \\ d & = & -n \circ v_1 \end{array} \tag{5.2}$$

Figure 5.5 shows an example where the RANSAC plane extraction pipeline performs better as compared to the region growing method. RANSAC produces contours that cover the entire planar surfaces and are more consistent across frames. The downside is that it is slower, but we went for precision in our system, so this is what we will use.

5.3 Plane Tracking

There are two ways in which plane tracking is employed in the pipeline: by projecting inliers from one frame to the other and by computing the actual intersection area of the candidate planes.

The first approach can be used when the relative transformation between the two frames is small. For each plane in the source frame, we count how many of its inliers hit a plane in the target frame by looking at the same uv-coordinates. Two planes are paired if the source plane hits the target plane the most and vice-versa (maximums in both directions), and the average number of hits from one to the other is above a certain threshold. This method is computationally cheap given that we work on downsampled frames, and it is well suited for



(a) (b)

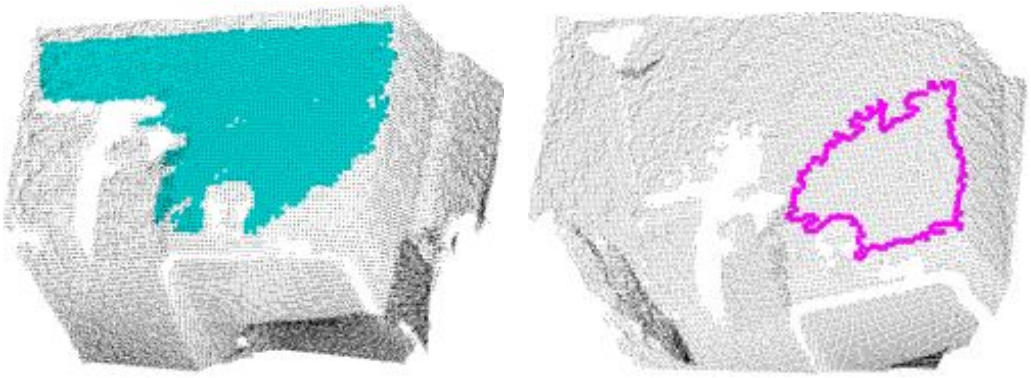
Figure 5.4: Planar polygon contour (a) alpha shapes; (b) alpha shapes and contour simplification, leading to a much compact representation of the same polygonal surface.

Algorithm 4 Plane extraction algorithm using RANSAC

repeat
  Choose 3 points at random from the point set
  Compute the plane parameters using Equation 5.2
  for each other point v in the cloud **do**
    if $v \circ n + d \leq thresh$ **then**
      $inliers \leftarrow inliers + 1$
    end if
  end for
  if $inliers > max_{inliers}$ **then**
    $max_{inliers} \leftarrow inliers$
    $params_{best} \leftarrow (n, d)$
  end if
until $k < K$

Euclidean clustering to take the largest connected component.
for m iterations **do**
  Compute the plane parameters for the remaining inlier set using PCA
  Compute the standard deviation σ for the projection distance from each point to the plane
  Eliminate the points with distances larger than $\alpha * \sigma$
end for

Compute the convcave hull of the inliers.
Simplify the contour.



(a) RANSAC with contour post processing (b) region growing

Figure 5.5: The two plane extraction approaches we proposed.

tracking planes between consecutive frames, given that the camera motion was not very fast and the dataset was recorded at full frame rate (30 Hz).

The second method is to be used when computing the overlap between planes in world coordinates. It begins by checking if the equations of the two candidate planes are within thresholds in terms of normal deviation and displacement along the normal. If this test is passed, then the contour of the candidate source plane is projected onto the target plane. Another check is done on the root mean square of the projection distances, and some false positives are eliminated. Next, the intersection of the two 2D polygons is computed using the Clipper library [Joh], which is an open-source implementation of [Vat92]. The decision behind using this library against a more popular framework such as Boost Polygon [boo] is that it allows for polygons with self-intersections and it was proven faster in various performance benchmarks. The pair of polygons is considered to be in correspondence if the area of the intersection is large enough.

Our plane tracking pipeline works by extracting the planes in each incoming frame, finding the correspondences against the previous frame, then checking for other correspondences against all the planes collected so far in the world frame. All the corresponding planes are updated as follows: the source plane contour is projected to the target plane, and the union of the two 2D polygons is computed, then un-projected back to 3D, forming the contour of the new polygon.

5.4 Planes as Landmarks in Graph Optimization

The first paper to propose augmenting the extended Kalman filter (EKF) state vector with landmark positions was [SC86], but SLAM implementations today evolved to using graph optimization techniques. Folkesson uses a graph representation to optimize for both the landmarks and robot trajectory in [FC04].

Along those lines, Trevor et al. [TRC12] extend the GTSAM framework [DK06] to allow for planes to be used as landmarks. After extracting each plane and refining its parameters, a constraint is added to the graph between the camera node and the plane landmark. This constraint models the plane parameters in the current frame. The graph is then optimized for both the poses of the cameras and the parameters for each plane in the world frame. The set of equations 5.3 show the error function that is used for the edge and the Jacobians to aid the convergence of the optimizer.

$$
\begin{aligned}
e &= \begin{bmatrix} R^T * \vec{n} \\ \vec{n} \circ t + d \end{bmatrix} - \begin{bmatrix} \vec{n}_m \\ d_m \end{bmatrix} \\
\frac{\delta h}{\delta X_r} &= \begin{bmatrix} 0 & -n_x & n_y & 0 & 0 & 0 \\ n_x & 0 & -n_z & 0 & 0 & 0 \\ -n_y & n_z & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & n_x & n_y & n_z \end{bmatrix} \\
\frac{\delta h}{\delta n_{map}} &= \begin{bmatrix} [R] & \vec{0} \\ \vec{X_r^T} & 1 \end{bmatrix}
\end{aligned}
\tag{5.3}
$$

By using the plane tracking and merging techniques presented in the previous sections with the graph constraints allows for effectively closing loops without the need of local feature descriptors. Figure 5.6 depicts the results of our system with and without plane features. This dataset was especially difficult due to the lack of geometric features in the scans. Planes aided the graph optimizer not to diverge, and it ended up in a reasonable local minima, allowing the map to be further improved using global ICP.

On top of the constraints between camera poses and plane parameters, we propose to add to constraints between the planes themselves. This idea is based on the fact that most indoor environments contain planar surfaces that are either parallel to each other or orthogonal to each other: walls are orthogonal to floors and ceilings, walls either meet under 90 degrees at a corner or are opposite walls, in which case they are parallel, tables and shelves are parallel to the ground etc. As such, we add a refinement step in our system at the end of the pipeline where all the angles between the normals of all pairs of planes are calculated and the ones that are close to being orthogonal or parallel are enforced to snap using an edge in the graph, with the error function and Jacobians in Equation 5.4. In our experiments these additional constraints did help, the effects being most visible in top-down views of rooms, as in Figure

(a) (b)

Figure 5.6: The proposed reconstruction pipeline (a) based only on point cloud registration; (b) with additional camera-plane constraints in the graph optimization.

5.7. The architecture of the graph becomes as shown in Figure 5.8.

$$\begin{aligned} e &= \vec{n_{src}}^T \vec{n_{tgt}} - \alpha \\ \frac{\delta e}{\delta P_{src}} &= \begin{bmatrix} n_{tgt_x} & n_{tgt_y} & n_{tgt_z} & 0 \end{bmatrix} \\ \frac{\delta e}{\delta P_{tgt}} &= \begin{bmatrix} n_{src_x} & n_{src_y} & n_{tgt_z} & 0 \end{bmatrix} \end{aligned} \tag{5.4}$$

Another advantage of this approach is that the global planes are optimized independently from the camera poses, meaning that we have a consistent plane map at the end. This map is useful as a low-cost representation of the environment or in robotics-oriented tasks such as collision detection, identifying stable locations where objects can be placed safely (e.g., tables, desks). Figure 5.9 shows the differences between the point cloud map, the plane inliers and the globally-optimized polygonal planes planes.

5.5 Frame to Frame Alignment using Planes

In the following section, three methods for aligning individual frames by using planar features will be introduced.

(a) (b)

Figure 5.7: Top-down view of a room (a) without using the additional orthogonality and parralelism constraints. (b) after imposing them and doing an additional graph optimization step - notice.

5.5.1 RANSAC and Rendered Depth Maps

We propose a novel approach for aligning two frames by using planes in a RANSAC framework. Due to the nature of the algorithm, no initial alignment is required, allowing for registering non-neighboring frames easily. The steps are shown in Algorithm 5. The algorithm expects a set of source and target planes, which are extracted and processed using the methods presented in the previous sections. The first pair of planes is selected at random from the source planes set and target planes set, respectively. The next two pairs are selected such that the angles between planes 0-1, 1-2, and 0-2 are similar for both the source and target planes, respectively. This is inspired by the checks usually done in point-based RANSAC where the distances between the points in each frame are imposed to be similar.

The transformation is computed in two stages, and needs a minimum of three plane corre-

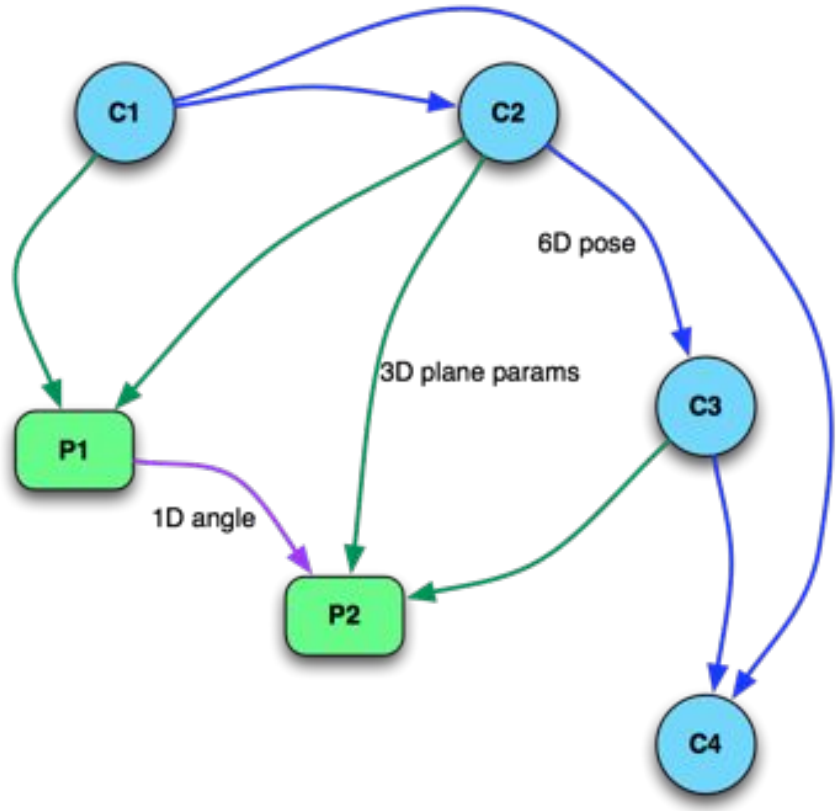


Figure 5.8: Complete graph architecture for mapping using plane features. The circles represent the camera nodes (6D world coordinates) and the green squares are plane nodes (3D world coordinates). There are three types of edges: camera-to-camera (relative 6D pose), camera-to-plane (3D plane parameters in the camera frame), and plane-to-plane (relative 1D angle between the normals of the planes.

(a) (b) (c)

Figure 5.9: The proposed reconstruction pipeline (a) based only on point cloud registration; (b) with additional camera-plane constraints in the graph optimization.

spondences. First, the rotation is estimated using singular value decomposition:

$$
\begin{aligned}
N_{src} &= \begin{bmatrix} n_{src_{1x}} & n_{src_{2x}} & \dots & n_{src_{mx}} \\ n_{src_{1y}} & n_{src_{2y}} & \dots & n_{src_{my}} \\ & & & \\ n_{src_{1z}} & n_{src_{2z}} & \dots & n_{src_{mz}} \end{bmatrix} \\
N_{tgt} &= \begin{bmatrix} n_{tgt_{1x}} & n_{tgt_{2x}} & \dots & n_{tgt_{mx}} \\ n_{tgt_{1y}} & n_{tgt_{2y}} & \dots & n_{tgt_{my}} \\ n_{tgt_{1z}} & n_{tgt_{2z}} & \dots & n_{tgt_{mz}} \end{bmatrix} \\
H &= N_{src}\, N_{tgt}^T \\
H &= U \Sigma V^* \\
\Rightarrow R &= V\, U^T
\end{aligned}
\tag{5.5}
$$

The transformation for plane parameters is computed from the point transformation as follows:

$$
\begin{aligned}
T &= \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \\
T_{plane} &= T^{-T} = \begin{bmatrix} R & 0 \\ -t^T R & 1 \end{bmatrix}
\end{aligned}
\tag{5.6}
$$

The translation is then isolated:

$$
\begin{aligned}
\begin{bmatrix} R & 0 \\ -t^T R & 1 \end{bmatrix} \begin{bmatrix} n_{src} \\ d_{src} \end{bmatrix} &= \begin{bmatrix} n_{tgt} \\ d_{tgt} \end{bmatrix} \\
-t^T R\, n_{src} + d_{src} &= d_{tgt} \\
(R\, n_{src})^T\, t &= d_{src} - d_{tgt}
\end{aligned}
\tag{5.7}
$$

and then computed using LU decomposition for $A\, x = b$, with:

$$
\begin{aligned}
A &= (R\, N_{src})^T \\
b &= \begin{bmatrix} d_{tgt_1} - d_{src_1} \\ d_{tgt_1} - d_{src_1} \\ \dots \\ d_{tgt_m} - d_{src_m} \end{bmatrix}
\end{aligned}
\tag{5.8}
$$

After the transformation is computed from the three plane correspondences, the transformed triangulation of the source planes is rendered to obtain a low resolution depth map (40x30

Figure 5.10: Example of the low resolution renderings used by the plane RANSAC algorithm for computing the number of inliers for a candidate source transformation.

offered a good balance between precision and performance, see Figure 5.10). This depth map is then compared dexel by dexel with the target depth map rendering and the number of inliers is counted. This will represent the score of the current transformation that will be used in the RANSAC iterations.

Figure 5.11 shows two example runs of the algorithm. This approach was found to fail when two frames have multiple valid relative transformations that cannot be differentiated by using only planes, as shown in Figure 5.12, and we will address these special cases in future work. Performance-wise, the current implementation needs considerable time to align two frames (in the seconds range). We believe that this approach is promising, as it can be heavily optimized or used as a building block for a more complex algorithm.

5.5.2 Joint Point and Plane Optimization

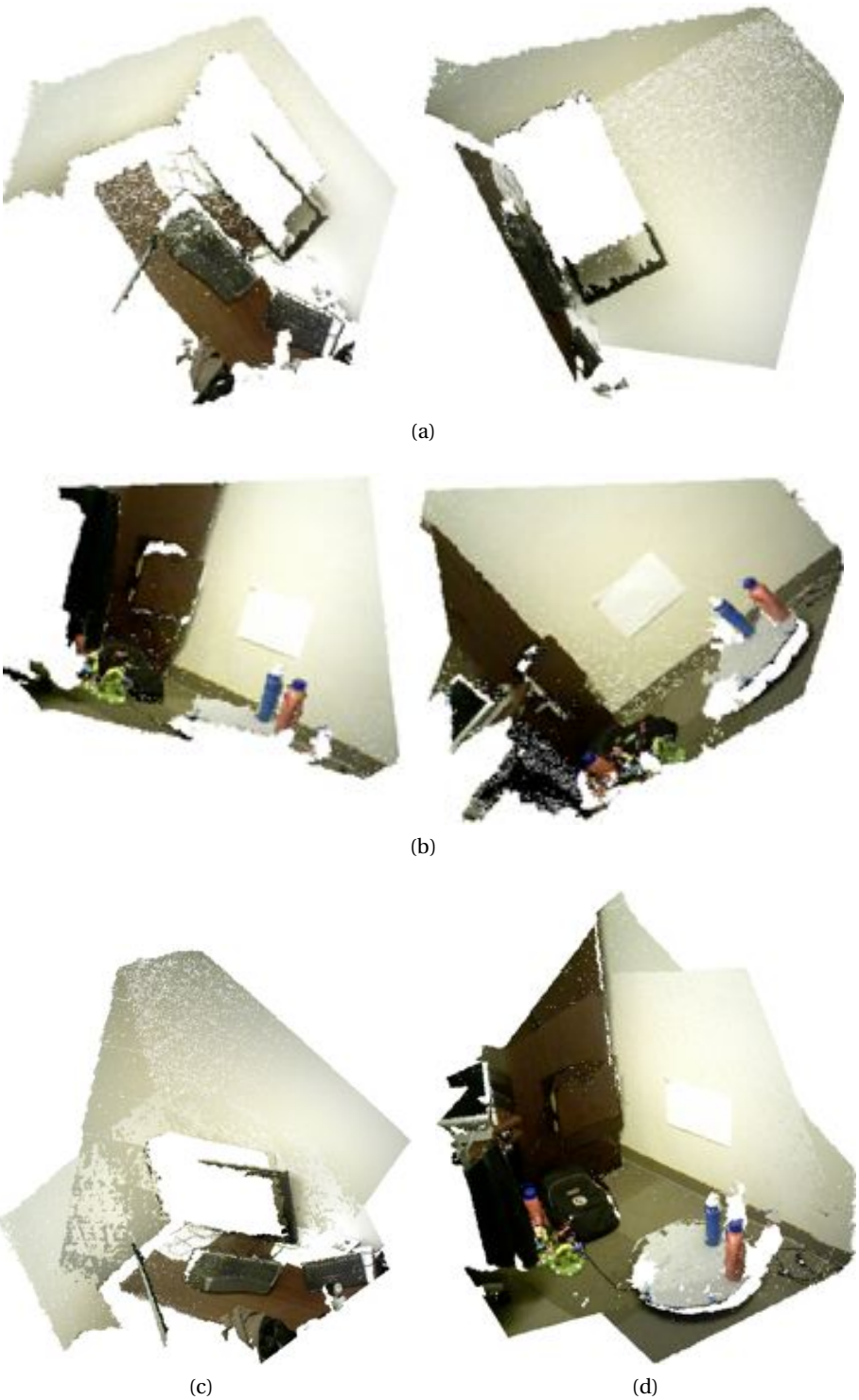
Planes could also be used to influence the convergence of the ICP algorithm. This is inspired by mapping systems that use sparse RGB features in a joint optimization with point-to-point or point-to-plane dense correspondences. Point correspondences can be determined using any tuned sampling, correspondence estimation and rejection pipeline. Plane correspondences are found by either of the two methods explained in Section 5.3, but the second one is more suitable as the overlapping area can be used as a weighting factor in the error function. The transformation is solved for by using the Levenberg Marquardt approach on the following error function, where N is the number of point correspondences and M the number of plane correspondences. α adjusts the importance of the point matches against the plane matches and β is a parameter that weights the distance component of the planes.

$$E = \alpha \sum_{i=0}^{N} [(p_i - q_i) \circ n_i]^2 + (1-\alpha) \sum_{i=0}^{M} w_i [(1.0 - n_{src_i} \circ n_{tgt_i}) + \beta (d_{src_i} - d_{tgt_j})]^2 \tag{5.9}$$

In order to aid the convergence speed, one can start the algorithm with a larger plane weighting

Algorithm 5 Frame registration using planes in a RANSAC framework and rendered depth maps

repeat
  Choose one plane P_{src_1} at random from the source plane set
  Choose one plane P_{tgt_1} at random from the target plane set
  Choose plane pair (P_{src_2}, P_{tgt_2})
  $\alpha_1 \leftarrow acos[normal(P_{src_1}) \circ normal(P_{src_2})]$
  $\alpha_2 \leftarrow acos[normal(P_{tgt_1}) \circ normal(P_{tgt_2})]$
  if $||\alpha_1 - \alpha_2|| > \Delta\alpha$ **then**
    continue
  end if
  Choose plane pair (P_{src_3}, P_{tgt_3})
  $\alpha_3 \leftarrow acos[normal(P_{src_1}) \circ normal(P_{src_3})]$
  $\alpha_4 \leftarrow acos[normal(P_{tgt_1}) \circ normal(P_{tgt_3})]$
  if $||\alpha_3 - \alpha_4|| > \Delta\alpha$ **then**
    continue
  end if
  $\alpha_5 \leftarrow acos[normal(P_{src_2}) \circ normal(P_{src_3})]$
  $\alpha_6 \leftarrow acos[normal(P_{tgt_2}) \circ normal(P_{tgt_3})]$
  if $||\alpha_5 - \alpha_6|| > \Delta\alpha$ **then**
    continue
  end if
  Compute the transformation T using the three plane pairs
  $T_{plane} \leftarrow T^{-T}$
  Render the source planes transformed with T_{plane}.
  $inliers \leftarrow$ the number of inliers between the rendered source and target depth maps
  if $inliers > max_{inliers}$ **then**
    $max_{inliers} \leftarrow inliers$
    $T_{best} \leftarrow T$
  end if
until $k < K$



(a)

(b)

(c) (d)

Figure 5.11: Two examples of frames with bad initial alignments (a), (b), and the results when aligned with our proposed plane RANSAC algorithm: (c), (d).

Figure 5.12: Example where the planes RANSAC method fails, as there are three ways in which the two frames representing one corner of a room can be aligned, and they cannot be discriminated by using just planes.

Pairs	Number of iterations	
	points only	points and planes
1	6	6
2	19	11
3	30	200 (did not converge

Table 5.1: Joint point and plane optimization benchmark.

in order to roughly align the frames and then decrease it for more precise registration by using only the points. Table 5.1 shows the improvements of this method on the three sample point cloud pairs used in our previous benchmarks.

5.5.3 Plane-Constrained ICP

In order to reduce the number of degrees of freedom that the transformation estimator has to deal with, the relative transformation between two frames can be locked down from six to three degrees of freedom by using plane features. The best plane correspondence (the pair of planes whose overlap is largest) is found between the plane sets of the two frames. Next, the transformation for which the normal of this plane is aligned to the z-axis and the origin is at the centroid of the contour of the planar region is computed for each of the two frames. The plane inliers are removed from each frame as they will not contribute to the error function, reducing the number of correspondences and errors that have to be computed, especially for typical indoor environment sequences, as the camera is pointed towards walls or the floor most of the time. By bringing both frames to this common coordinate system, the only unknowns that need to be estimated are the translation in the XY-plane and the rotation

around the Z-axis. This is done using an LM optimization with a point-to-plane error metric between the 3D point pairs that only solves for the remaining three relative pose components.

5.6 Conclusion

In this chapter we have looked into approaches for describing point clouds with local features. From literature and our experiments we reached the conclusion that 3D features that were proposed in the context of laser scanning are not applicable to current RGB-D cameras due to the noisy nature of the data. As a consequence, we investigated the usage of planar polygons as feature descriptors for various tasks as aligning frame pairs or as landmarks in graph optimization frameworks.

6 Results, Benchmarks and Future Work

6.1 TUM Datasets Benchmarks

As mentioned in the introductory Section 1.4, Sturm et al. [SEE*12] provide researchers with a large collection of datasets acquired with a Kinect in different situations, along with ground truth pose information and a suite of tools for standardizing the error computations. [SB12], [EHE*12] benchmark their RGB-D mapping systems against the same sequences, allowing for easy comparison of the performance of our work. Because of the fact that the authors of each of the aforementioned publications use different subsets of the collection for the evaluation and also different error representations, we will compare our results with theirs in separate tables: 6.1 and 6.2.

Some of the datasets provided have non-static scenes that contain people moving around etc. The pipeline we propose proves robust to small motions such as the *sitting** datasets, but fails when larger motions are introduced such as people walking. See Table 6.3 for more details.

As expected, our system did have problems with some datasets due to the fact that there are frames in the sequences that do not contain enough geometric information to easily register them without the use of colors. For this reason, we ran these datasets with the pipeline that uses planar features and the results improved, as shown in Figure 6.3.

On top of the registration, our system can also generated textured meshes. Figure 6.4 shows three examples from the TUM collection.

6.2 Our own datasets

Below are screenshots of meshes obtained from registering our own collection of datasets. The sequences ranged from 1000 to 3000 frames.

Dataset	RMS ATE [m]	
	ours	other [SB12]
freiburg1_360	0.113	0.069
freiburg1_desk2	0.038	0.049
freiburg1_desk	0.038	0.043
freiburg1_plant	0.055	0.026
freiburg1_room	-	0.069
freiburg1_rpy	0.031	0.027
freiburg1_teddy	0.109	0.039
freiburg1_xyz	0.016	0.013
freiburg2_desk	-	0.052
freiburg2_rpy	0.026	0.024
freiburg2_xyz	0.018	0.020
Average	0.0493	0.0319

Table 6.1: The RMS ATE of our system against [SB12].

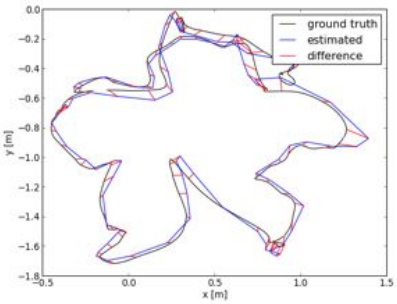
Dataset	RMSE translation [m]		RMSE rotation [°]	
	our	RGB-D Mapping	our	RGB-D Mapping [EHE*12]
freiburg1_360	0.102	0.103	5.85	3.41
freiburg1_desk2	0.157	0.102	4.26	3.81
freiburg1_desk	0.056	0.049	3.44	2.43
freiburg1_floor	-	0.055	-	2.35
freiburg1_plant	0.084	0.142	6.74	6.34
freiburg1_room	-	0.219	-	9.04
freiburg1_rpy	0.114	0.042	3.78	2.50
freiburg1_teddy	0.155	0.138	7.19	4.75
freiburg1_xyz	0.033	0.021	3.00	0.90
Average	0.1001	0.0852	4.894	3.448

Table 6.2: The RMS RPE of our system against [EHE*12].

Dataset	RMSE ATE [m]	Observations
freiburg3_sitting_halfsphere	0.035	Two persons discussing at a desk, with fair head and hand motions
freiburg3_sitting_rpy	0.090	Same as above, different camera motion
freiburg3_sitting_xyz	0.042	Same as above, different camera motion
freiburg3_sitting_static	0.009	Same as above, camera is almost static
freiburg3_walking_static	1.336	People walking in front of the camera with large occlusions, slight camera movement
freiburg3_walking_xyz	0.952	Same as above, camera moves
freiburg2_desk_with_person	0.483	Person walks in the scene half-way during the recording

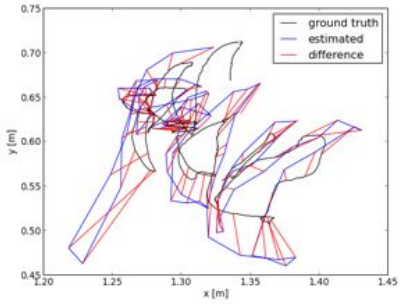
Table 6.3: The behavior of our system when faced with non-static scenes.

(a) freiburg1_plant map



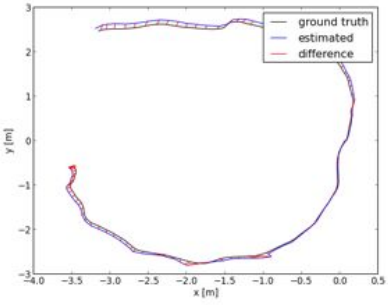
(b) freiburg1_plane track

(c) freiburg1_rpy map



(d) freiburg1_rpy track

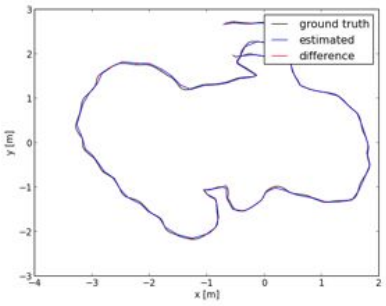
(e) freiburg3_large_cabinet map



(f) freiburg3_large_cabinet track

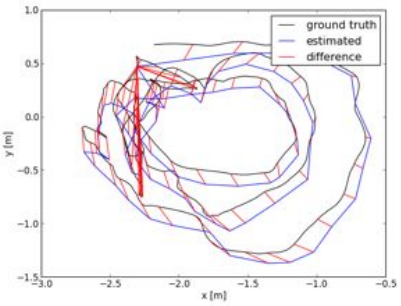
Figure 6.1: Screenshots of the registered point cloud maps along with their tracks plotted against ground truth on some of the RGB-D TUM datasets.

(a) freiburg3_long_office_household map



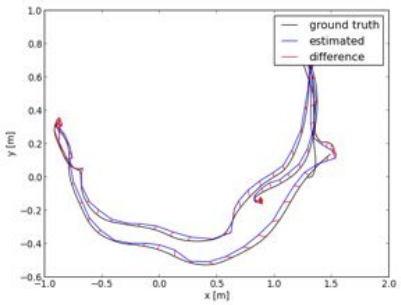
(b) freiburg3_long_office_household track

(c) freiburg3_teddy map



(d) freiburg3_teddy track

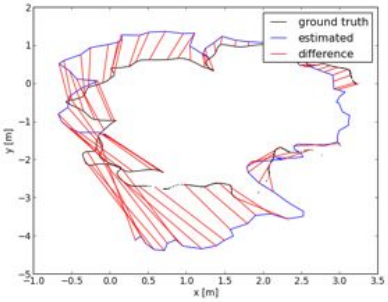
(e) freiburg1_desk map



(f) freiburg1_desk track

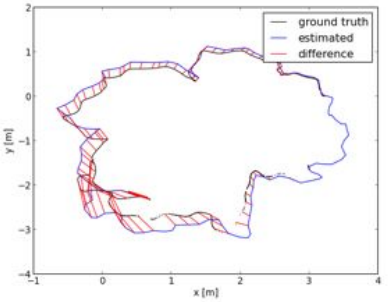
Figure 6.2: Screenshots of the registered point cloud maps along with their tracks plotted against ground truth on some of the RGB-D TUM datasets.

(a) freiburg2_desk map



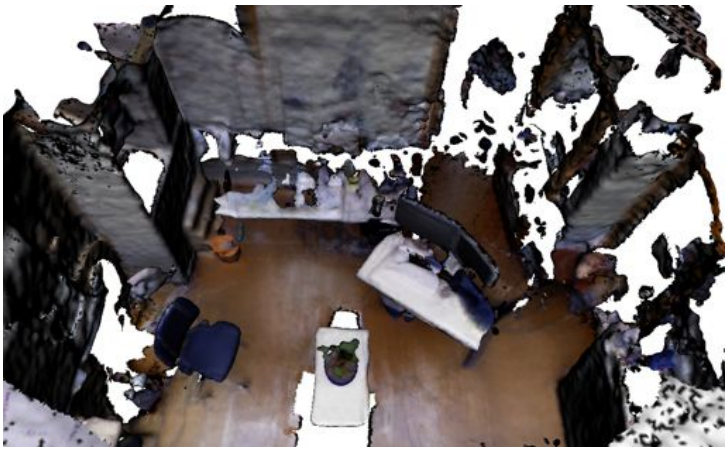
(b) freiburg2_desk track

(c) freiburg2_desk map with planes



(d) freiburg2_desk track with planes

Figure 6.3: The improvements using planar features in the pipeline



(a) freiburg1_plant

(b) freiburg3_long_office_household

(c) freiburg3_teddy

Figure 6.4: Meshes of TUM datasets.

(a) car, view 1

(b) car, view 2

(c) PR2 robot, view 1

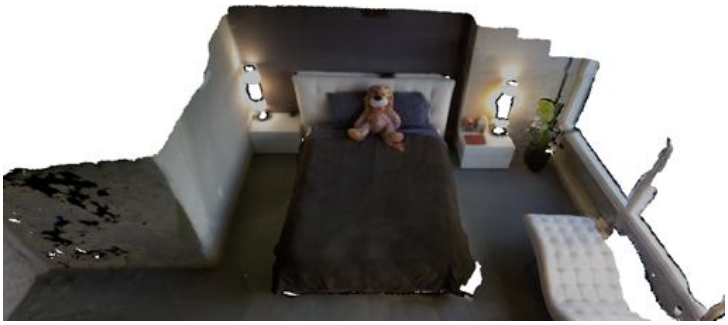
(d) PR2 robot, view 2

(e) person, view 1

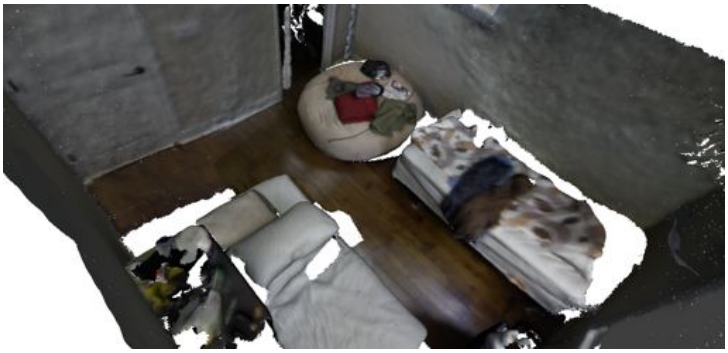
(f) person, view 2

Figure 6.5: Mesh results of our own datasets.

(a) living room, view 1



(b) living room, view 2



(c) room 1

(d) room 2

Figure 6.6: Mesh results of our own datasets.

(a) gingerbread house

(b) fruits

(c) plush toy

Figure 6.7: Mesh results of our own datasets.

6.3 Future Work

As we have seen, state-of-the-art RGB-D mapping software gets great benefit from using local feature descriptors from color images (under controlled lighting conditions). We concluded that existing geometric local features are too sensitive to the noisy nature of current commercial RGB-D cameras. We are planning to work on developing 3D features that are adapted to this new type of sensing data.

In a mapping system, the option of offering feedback to the user during the acquisition time is valuable, informing the operator about possible locations of the scene where data is missing or where misalignments occurred. Building on this, one can also think of allowing the user to make manual adjustments of the model during acquisition, in case the registration fails.

During this work, we have looked into using multiple cameras for mapping the same scene synchronously. The proposed architecture currently allows for that, as one can do incremental frame to frame registration independently for each camera and then do loop closures and error relaxation on a common graph representation of the scene.

An issue we noticed during evaluation is the lack of robustness under the presence of large moving objects in the scene. This cannot be solved by simply considering the points corresponding to the objects as inliers, but needs a semantic approach. We will look into this in the future.

6.4 Conclusions

In this thesis we have investigated methodologies for enabling simultaneous localization and mapping with handheld consumer-level RGB-D cameras. Benchmarks have been done on ways of registering pairs of point clouds, and we have explained the advantages of choosing an approach for each step of the pipeline. Next, we looked into how the maps can be represented as graphs and improved by loop closure and graph optimization techniques. After a precise point cloud of the scene is obtained, surface reconstruction is employed in order to create a mesh that allows for more efficient visualization. Going forwards, we inspected ways of transferring the color information from the camera frames to the model. After these steps are done, we obtain the final colored mesh which can immediately be used in various applications such as CAD or computer games. However, in order to improve robustness, we made use of assumptions about typical indoor scenes. These are focused on the heuristic that indoor scenes are mostly planar, and have properties similar to a Manhattan World (planes are parallel or orthogonal to each other). As such, we introduced novel ways in which planar polygons are a medium for registering frames and for improving the robustness in the graph optimizer.

Bibliography

[ABCo*03] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Computing and rendering point set surfaces. *IEEE Transactions on Visualization and Computer Graphics 9* (2003), 3–15.

[AHB87] ARUN K. S., HUANG T. S., BLOSTEIN S. D.: Least-squares fitting of two 3-d point sets. *IEEE Trans. Pattern Anal. Mach. Intell. 9*, 5 (May 1987), 698–700.

[asu] Asus Xtion PRO. http://www.asus.com/Multimedia/Xtion_PRO/.

[Aut13] AUTODESK: Maya. http://usa.autodesk.com/maya/, 2013.

[BDH96] BARBER C. B., DOBKIN D. P., HUHDANPAA H.: The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw. 22*, 4 (Dec. 1996), 469–483.

[BM92] BESL P., MCKAY N.: A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14*, 2 (1992), 239–256.

[BMR*99] BERNARDINI F., MITTLEMAN J., RUSHMEIER H., SILVA C., TAUBIN G., MEMBER S.: The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics 5* (1999), 349–359.

[boo] Boost polygon library. http://www.boost.org/doc/libs/1_53_0/libs/polygon/doc/index.htm.

[CCR08] CIGNONI P., CORSINI M., RANZUGLIA G.: Meshlab: an open-source 3d mesh processing system. *ERCIM News*, 73 (April 2008), 45–46.

[Cen07] CENSI A.: An accurate closed-form estimate of ICP's covariance. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)* (Rome, Italy, April 2007), pp. 3167–3172.

[CM92] CHEN Y., MEDIONI G.: Object modelling by registration of multiple range images. *Image Vision Comput. 10*, 3 (Apr. 1992), 145–155.

[DF01] DEPARTMENT A. F., FITZGIBBON A. W.: Robust registration of 2d and 3d point sets. In *In British Machine Vision Conference* (2001), pp. 411–420.

[DHR*11] DU H., HENRY P., REN X., CHENG M., GOLDMAN D. B., SEITZ S. M., FOX D.: Interactive 3d modeling of indoor environments with a consumer depth camera. In *Proceedings of the 13th international conference on Ubiquitous computing* (New York, NY, USA, 2011), UbiComp '11, ACM, pp. 75–84.

[DK06] DELLAERT F., KAESS M.: Square root sam: Simultaneous location and mapping via square root information smoothing. *International lJournal of Robotics Research (IJRR) 25*, 12 (2006), 1181. Special issue on RSS 2006.

[DRT*04] DIEBEL J., REUTERSWARD K., THRUN S., DAVIS J., GUPTA R.: Simultaneous localization and mapping with active stereo vision. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on* (sept.-2 oct. 2004), vol. 4, pp. 3436 – 3443 vol.4.

[EHE*12] ENDRES F., HESS J., ENGELHARD N., STURM J., CREMERS D., BURGARD W.: An evaluation of the RGB-D SLAM system. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)* (St. Paul, MA, USA, May 2012).

[ELF97] EGGERT D. W., LORUSSO A., FISHER R. B.: Estimating 3-d rigid body transformations: a comparison of four major algorithms. *Mach. Vision Appl. 9*, 5-6 (Mar. 1997), 272–290.

[EM94] EDELSBRUNNER H., MÜCKE E. P.: Three-dimensional alpha shapes. *ACM Trans. Graph. 13*, 1 (Jan. 1994), 43–72.

[EMSN12] ELSEBERG J., MAGNENAT S., SIEGWART R., NÜCHTER A.: Comparison on nearest-neigbour-search strategies and implementations for efficient shape registration. *Journal of Software Engineering for Robotics (JOSER) 3*, 1 (2012), 2–12.

[FC04] FOLKESSON J., CHRISTENSEN H. I.: Graphical slam - a self-correcting map. In *ICRA* (2004), pp. 383–390.

[GJ*10] GUENNEBAUD G., JACOB B., ET AL.: Eigen v3. http://eigen.tuxfamily.org, 2010.

[GK02] GOPI M., KRISHNAN S.: A fast and efficient projection-based approach for surface reconstruction. In *Proceedings of the 15th Brazilian Symposium on Computer Graphics and Image Processing* (Washington, DC, USA, 2002), SIBGRAPI '02, IEEE Computer Society, pp. 179–186.

[GKS*] GRISETTI G., KÜMMERLE R., STACHNISS C., FRESE U., HERTZBERG C.: Hierarchical optimization on manifolds for online 2d and 3d mapping.

[GKUP11] GSCHWANDTNER M., KWITT R., UHL A., PREE W.: BlenSor: Blender Sensor Simulation Toolbox Advances in Visual Computing. vol. 6939 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2011, ch. 20, pp. 199–208.

[Gou71] GOURAUD H.: Continuous shading of curved surfaces. *IEEE Trans. Comput. 20*, 6 (June 1971), 623–629.

[GR03] GELFAND N., RUSINKIEWICZ S.: Geometrically stable sampling for the icp algorithm. In *Proc. International Conference on 3D Digital Imaging and Modeling* (2003), pp. 260–267.

[GSB] GRISETTI G., STACHNISS C., BURGARD W.: Non-linear constraint network optimization for efficient map learning. *IEEE Transactions on Intelligent Transportation Systems*, 2009.

[GSGB07] GRISETTI G., STACHNISS C., GRZONKA S., BURGARD W.: A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *In Proc. of Robotics: Science and Systems (RSS* (2007).

[HGCA12] HELD R., GUPTA A., CURLESS B., AGRAWALA M.: 3d puppetry: a kinect-based interface for 3d animation. In *Proceedings of the 25th annual ACM symposium on User interface software and technology* (New York, NY, USA, 2012), UIST '12, ACM, pp. 423–434.

[HHN88] HORN B. K. P., HILDEN H., NEGAHDARIPOUR S.: Closed-form solution of absolute orientation using orthonormal matrices. *JOURNAL OF THE OPTICAL SOCIETY AMERICA 5*, 7 (1988), 1127–1135.

[HKH*10] HENRY P., KRAININ M., HERBST E., REN X., FOX D.: 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).

[Hor87] HORN B. K. P.: Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A 4*, 4 (1987), 629–642.

[HRD*12] HOLZER S., RUSU R. B., DIXON M., GEDIKLI S., NAVAB N.: Real-Time Surface Normal Estimation from Organized Point Cloud Data Using Integral Images. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Vila Moura, Algarve, Portugal, October 2012).

[Ich12] ICHIM A.-E.: Toyota code sprint final report. http://svn.pointclouds.org/tocsweb/source/aichim/files/tocs_final_ichim.pdf, 2012.

[IKH*11] IZADI S., KIM D., HILLIGES O., MOLYNEAUX D., NEWCOMBE R., KOHLI P., SHOTTON J., HODGES S., FREEMAN D., DAVISON A., FITZGIBBON A.: 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* (New York, NY, USA, 2011), UIST '11, ACM, pp. 559–568.

[JH98] JOHNSON A. E., HEBERT M.: Surface matching for object recognition in complex 3-d scenes. *Image and Vision Computing 16* (1998), 635–651.

[JLW05] JIN S., LEWIS R. R., WEST D.: A comparison of algorithms for vertex normal computation. *The Visual Computer 21*, 1-2 (2005), 71–82.

[Joh] JOHNSON A.: Clipper library. https://sourceforge.net/projects/polyclipping/.

[KAWB09] KLASING K., ALTHOFF D., WOLLHERR D., BUSS M.: Comparison of surface normal estimation methods for range sensing applications. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation* (Piscataway, NJ, USA, 2009), ICRA'09, IEEE Press, pp. 1977–1982.

[KBH06] KAZHDAN M., BOLITHO M., HOPPE H.: Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2006), SGP '06, Eurographics Association, pp. 61–70.

[KGS*11] KÜMMERLE R., GRISETTI G., STRASDAT H., KONOLIGE K., BURGARD W.: g2o: A general framework for graph optimization. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)* (Shanghai, China, May 2011).

[kin] Kinect for Windows. http://www.microsoft.com/en-us/kinectforwindows/.

[KJR*11] KAESS M., JOHANNSSON H., ROBERTS R., ILA V., LEONARD J., DELLAERT F.: iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering. In *IEEE Intl. Conf. on Robotics and Automation, ICRA* (Shanghai, China, May 2011), pp. 3281–3288.

[KRD08] KAESS M., RANGANATHAN A., DELLAERT F.: iSAM: Incremental smoothing and mapping. *IEEE Trans. on Robotics (TRO) 24*, 6 (Dec. 2008), 1365–1378.

[Lev98] LEVIN D.: The approximation power of moving least-squares. *Math. Comput. 67*, 224 (Oct. 1998), 1517–1531.

[Low04] LOW K.-L.: Linear least-squares optimization for point-to-plane icp surface registration. In *Technical Report TR04-004, Department of Computer Science, University of North Carolina at Chapel Hill* (2004).

[Max99] MAX N.: Weights for computing vertex normals from facet normals. *journal of graphics, gpu, and game tools 4*, 2 (1999), 1–6.

[MFD*09] MAY S., FUCHS S., DROESCHEL D., HOLZ D., NÜCHTER A.: Robust 3d-mapping with time-of-flight cameras. In *Proceedings of the 2009 IEEE/RSJ international conference on Intelligent robots and systems* (Piscataway, NJ, USA, 2009), IROS'09, IEEE Press, pp. 1673–1678.

[ML09] MUJA M., LOWE D. G.: Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)* (2009), INSTICC Press, pp. 331–340.

[NH08] NÜCHTER A., HERTZBERG J.: Towards semantic maps for mobile robots. *Robot. Auton. Syst. 56*, 11 (Nov. 2008), 915–926.

[NIL12] NGUYEN C. V., IZADI S., LOVELL D.: Modeling kinect sensor noise for improved 3d reconstruction and tracking. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on* (oct. 2012), pp. 524 –530.

[PBV*10] PATHAK K., BIRK A., VASKEVICIUS N., PFINGSTHORN M., SCHWERTFEGER S., POPPINGA J.: Online three-dimensional slam by registration of large planar surface segments and closed-form pose-graph relaxation. *J. Field Robot. 27*, 1 (Jan. 2010), 52–84.

[PD09] PARIS S., DURAND F.: A fast approximation of the bilateral filter using a signal processing approach. *Int. J. Comput. Vision 81*, 1 (Jan. 2009), 24–52.

[PKDB10] PITZER B., KAMMEL S., DUHADWAY C., BECKER J.: Automatic reconstruction of textured 3d models. In *IEEE International Conference on Robotics and Automation, ICRA 2010, Anchorage, Alaska, USA, 3-7 May 2010* (2010), IEEE, pp. 3486–3493.

[PMC*11] POMERLEAU F., MAGNENAT S., COLAS F., LIU M., SIEGWART R.: Tracking a depth camera: Parameter exploration for fast icp. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2011).

[Pri] PRIMESENSE: Primesense 3d sensors. http://www.primesense.com/wp-content/uploads/2013/02/PrimeSense_3DsensorsWeb.pdf.

[Pul99] PULLI K.: Multiview registration for large data sets. In *3-D Digital Imaging and Modeling, 1999. Proceedings. Second International Conference on* (1999), pp. 160 –168.

[QCG*09] QUIGLEY M., CONLEY K., GERKEY B. P., FAUST J., FOOTE T., LEIBS J., WHEELER R., NG A. Y.: Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software* (2009).

[RBB09] RUSU R. B., BLODOW N., BEETZ M.: Fast point feature histograms (fpfh) for 3d registration. In *The IEEE International Conference on Robotics and Automation (ICRA)* (Kobe, Japan, 05/2009 2009).

[RBMB08] RUSU R. B., BLODOW N., MARTON Z. C., BEETZ M.: Aligning Point Cloud Views using Persistent Feature Histograms. In *Proceedings of the 21st IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Nice, France, September 22-26 2008).

[RC11] RUSU R. B., COUSINS S.: 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)* (Shanghai, China, May 9-13 2011).

[RL01] RUSINKIEWICZ S., LEVOY M.: Efficient variants of the ICP algorithm. In *Third International Conference on 3D Digital Imaging and Modeling (3DIM)* (June 2001).

[RMB*09] RUSU R., MARTON Z., BLODOW N., HOLZBACH A., BEETZ M.: Model-based and learned semantic object labeling in 3d point cloud maps of kitchen environments. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on* (oct. 2009), pp. 3601 –3608.

[Rus] RUSU R. B.: *Semantic 3D Object Maps for Everyday Manipulation in Human Living Environments*. PhD thesis, Computer Science department, Technische Universität München, Germany.

[SB12] STÜCKLER J., BEHNKE S.: Integrating depth and color cues for dense multi-resolution scene mapping using rgb-d cameras. In *Proceedings of the IEEE International Conference on Multisensor Fusion and Information Integration (MFI)* (2012).

[SC86] SMITH R. C., CHEESEMAN P.: On the representation and estimation of spatial uncertainly. *Int. J. Rob. Res. 5*, 4 (Dec. 1986), 56–68.

[SEE*12] STURM J., ENGELHARD N., ENDRES F., BURGARD W., CREMERS D.: A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)* (Oct. 2012).

[SFG] STACHNISS C., FRESE U., GRISETTI G.: Openslam. https://openslam.informatik.uni-freiburg.de.

[SML] SCHROEDER W., MARTIN K., LORENSEN B.: *The Visualization Toolkit, Third Edition*. Kitware Inc.

[SRKB11] STEDER B., RUSU R. B., KONOLIGE K., BURGARD W.: Point feature extraction on 3d range scans taking into account object boundaries. In *International Conference on Robotics and Automation* (2011 2011).

[Tau12] TAUBIN G.: Smooth signed distance surface reconstruction and applications. In *CIARP* (2012), Álvarez L., Mejail M., Gómez L., Jacobo J. C., (Eds.), vol. 7441 of *Lecture Notes in Computer Science*, Springer, pp. 38–45.

[TM05] THRUN S., MONTEMERLO M.: The GraphSLAM algorithm with applications to large-scale mapping of urban structures. *International Journal on Robotics Research 25*, 5/6 (2005), 403–430.

[TRC12] TREVOR A. J. B., ROGERS III J. G., CHRISTENSEN H. I.: Planar surface slam with 3d and 2d sensors. In *Intl. Conf. On Robotics and Automation* (St. Paul, MN, May 2012), IEEE.

[TSDS10] TOMBARI F., SALTI S., DI STEFANO L.: Unique signatures of histograms for local surface description. In *Proceedings of the 11th European conference on computer vision conference on Computer vision: Part III* (Berlin, Heidelberg, 2010), ECCV'10, Springer-Verlag, pp. 356–369.

[Tuk77] TUKEY J. W.: *Exploratory Data Analysis*. Addison-Wesley, 1977.

[TW98] THÜRMER G., WÜTHRICH C. A.: Computing vertex normals from polygonal facets. *journal of graphics, gpu, and game tools 3*, 1 (1998), 43–46.

[Vat92] VATTI B. R.: A generic solution to polygon clipping. *Commun. ACM 35*, 7 (July 1992), 56–63.

[WJK*12] WHELAN T., JOHANNSSON H., KAESS M., LEONARD J., MCDONALD J.: *Robust Tracking for Real-Time Dense RGB-D Mapping with Kintinuous*. Tech. Rep. MIT-CSAIL-TR-2012-031, Computer Science and Artificial Intelligence Laboratory, MIT, Sep 2012.

[WSV91] WALKER M. W., SHAO L., VOLZ R. A.: Estimating 3-d location parameters using dual number quaternions. *CVGIP: Image Underst. 54*, 3 (Oct. 1991), 358–367.

[WWLvG09] WEISE T., WISMER T., LEIBE B., VAN GOOL L.: In-hand scanning with online loop closure. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on* (27 2009-oct. 4 2009), pp. 1630 –1637.