End of Bachelor Degree Project

Industrial Technology Engineering

Reconstruction of scenes using a hand-held range imaging camera

MEMORY

Author: Javier Gallostra Acín

Director: Federico Thomas Arroyo

Rapporteur: Manuel Moreno Eguilaz

Call: September 2017

Escola Tècnica Superior

d'Enginyeria Industrial de Barcelona

Reconstruction of scenes using a hand-held range imaging camera

Javier Gallostra Acín

September 2017

This project has been developed at

Institut de Robòtica i Informàtica Industrial (IRI, CSIC-UPC)

Abstract

The current final degree project covers the development of a C++ program to reconstruct 3D scenes from a stream of incoming RGBD images from the Kinect for Windows v2 sensor using the PCL library and the libfreenect2 open source Kinect drivers. A background analysis is performed to analyze state of the art registration algorithms and similar registration applications which are already distributed.

All the computing steps are explained and detailed, from the data acquisition through the coarse and fine alignment steps to the final surface reconstruction algorithm. Several algorithms are discussed for each step, giving reasons for the final algorithm choice and the parameters set.

Finally, a brief discussion on the results and problems encountered during the development of the project serves as the basis of a proposal of improvements for future work on the subject. The project budget and the environmental impact are analyzed before the final conclusion chapter.

Contents

ETSEIB

List of Figures

List of Tables

1 Glossary

- CPU : *Central Processing Unit*
- FOV : *Field Of View*
- FPFH : *Fast Point Feature Histogram*
- GPU : *Graphics Processor Unit*
- HMI : *Human Machine Interface*
- ICP : *Iterative Closest Point*
- IR : *Infrared*
- ISS : *Intrinsic Shape Signatures*
- LED : *Light Emitting Diode*
- LIDAR : *Laser Imaging Detection And Ranging*
- LM: *Levenberg-Marquardt*
- MLS : *Moving Least Squares*
- NARF : *Normal Aligned Radial Feature*
- PC : *Personal Computer*
- PCL : *Point Cloud Library*
- PFH : *Point Feature Histogram*
- RAM : *Random Access Memory*
- RANSAC : *Random Sample Consensus*
- RGB : *Red Green and Blue*
- RGBD : *Red Green Blue and Depth*
- SIFT : *Scale Invariant Feature Transform*
- SL : *Structured Light*

- SSD : *Solid State Drive*
- SVD : *Singular Value Decomposition*
- TOF : *Time Of Flight*
- USB : *Universal Serial Bus*
- VRAM : *Video Random Access Memory*

2 Introduction

The field of robotics is one of the fastest growing fields in scientific research. Every day we hear of innovations and ground breaking discoveries which bring robotics closer to a wider public and promise to make life easier to everyone, for example in the fields of health and services. This rapid development of the field has also generated a debate on the ethics of robotics, which has grouped worldwide known scientists in an effort to define the moral of robotics. Meanwhile, researchers from around the world devote their lives and efforts to improve the abilities of robots and develop new strategies to enable them perform more difficult and challenging tasks.

One of this research areas is the world of sensing. A robot communicates with the real world with sensors and actuators, and the information gathered by the sensors is then processed to obtain a desired result. Sensing ranges from obtaining black and white images of the surroundings of the robot to measuring electromagnetic waves or atmospheric parameters. In this wide range of ways to get world data we find the world of 3D sensing and RGBD cameras.

In the past years, the development of affordable RGBD cameras has enabled a growth in the research on 3D sensing which was limited by the cost of expensive lasers and 3D equipment. In 2010 Microsoft presented Kinect, a RGBD camera for their XBOX gaming system which became a break-through in affordable 3D sensing equipment. Since then, a vast amount of research has been done to let robots know how the real world is.

In this context we find the launch of the second version of the Kinect: Kinect v2. With a new technology it promises more accurate measures with less processing needed. This project aimed at developing a program that gets a stream of data from the Kinect v2 and aligns it to obtain a final 3D model of a real world scene.

This is done by using several C++ open source libraries. For the interaction with Kinect v2 and image acquisition, libfreenect2[1] is used. As for data processing we use the well known Point Cloud Library[2], an open source standalone project for 2D/3D image and point cloud processing.

[1]See https://github.com/OpenKinect/libfreenect2 for more details.

[2]All the information about Point Cloud Library (manuals, examples, documentation) can be found at http://pointclouds.org/

2.1 Objectives

The main objective of the project is to write a C++ program which reconstructs a 3D scene from a stream of Kinect v2 RGBD images. This general objective can be subdivided into:

- Carry out a background analysis and a state of the art study to choose the point cloud processing strategy.
- Learn the C++ programming language, its programming paradigms and structure, in order to code the program and integrate the required libraries.
- Learn to use the libfreenect2 library and the PCL environment for acquiring Kinect data and processing the resulting point clouds.
- Develop a well structured program that fulfills the main objective of the project.
- Document the program to make it easy to understand and use in future applications.
- Analyze the results and propose future work and improvements on the finished program.

2.2 Scope

The project will be qualified as successful if it is able to reconstruct accurately a real world 3D scene from a stream of Kinect v2 RGBD image data. This reconstruction does not need to occur in real time[3] for the project to be satisfactory.

[3]We consider real time reconstruction when the process of aligning two consecutive RGBD images can be done before the next image from the stream is received.

3 Analysis of background

3.1 Range Imaging Techniques

The process of acquiring 3-D representations of the real world is not a trivial issue. Many researchers and scientists have devoted years of study to develop a technique for the acquisition of precise 3-D data resulting in a vast amount of different approaches and solutions. These solutions try to provide efficient methods specifically suited to their field of application: industry, biomedicine, robotics, computer vision, tracking and mapping, HMI...

In the present work only the techniques known as "Range Imaging"[4] will be considered. A Range Image is a 2-D image whose pixel values represent real world depth measurements relative to a known point. Several range images can be aligned applying the appropriate transformations in order to obtain a final 3-D representation of an object or a scene. The process of acquisition of range images aims to provide a fast, precise, and computationally cost-effective stream of data for its further analysis and processing. Thus, a technique must be chosen among the wide variety of possibilities trying to meet the requirements of the present application.

In order to reconstruct a scene with a hand-held device its volume and weight play a relevant role in the decision process. It is, however, the cost of the hardware, the quality of the data and the frame rate what most frequently determine the option to choose. A brief overview of the most commonly used techniques with their advantages and drawbacks will provide the key to success in selecting the adequate device.

3.1.1 Stereo Vision

The Stereo Vision approach to the problem consists of two cameras aligned with each other but separated which take photos from different viewpoints. The 2-D images can be matched by complex algorithms that find correspondences between them and finally calculate the distance of the points to the baseline that links the cameras.

It is a passive solution because it does not need any artificial light source and does not require moving parts. Furthermore, the overall cost of the hardware is very low. However, the computational time required to align the images and obtain the depth data is high and the results are not robust, due to the presence of shades and occlusion in the scenes which can lead to failure in solving the correspondence problem.

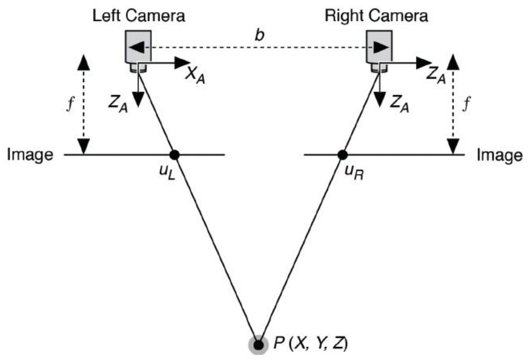


Figure 3.1: Simplified stereovision system. Source: [19]

3.1.2 Structured Light

Another commonly used method is to project a known pattern onto the surface to scan and capture an image of the projected pattern. By carefully analyzing the distortion in the pattern and knowing the relative position between the projector and the camera, the depth of the projected area can be obtained. The pattern is projected using IR light and captured by IR sensors.

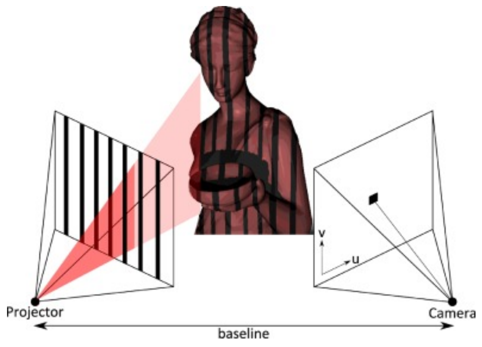


Figure 3.2: Principle of structured light based systems. Source: [24]

This is a far more precise method than Stereo Vision, providing a higher acquisition rate and a good performance even under various ambient light conditions. It still requires a computation process to analyze the projected pattern in order to calculate the depth image, but its implementation is reasonably affordable so it has become a widely used technique for range imaging.

3.1.3 Time of Flight

The TOF (Time Of Flight) technique relies on the precise knowledge of the constant value of the speed of light. By measuring the time it takes for a beam of light to travel from a light source to a receiver and knowing their relative position, the distance to the reflecting surface can be easily calculated.

There are various types of light sources for TOF applications, the most common ones being IR LED arrays and laser beams. Laser based applications such as LIDAR are more expensive and involve moving parts due to the fact that only one depth point can be acquired at a time, whereas the IR arrays are capable of capturing the entire FOV at once, providing depth images at a higher frame rate. As stated by Maged Aboali et al. [4], this significant difference leads to TOF mainly referring to IR applications and not Laser ones.

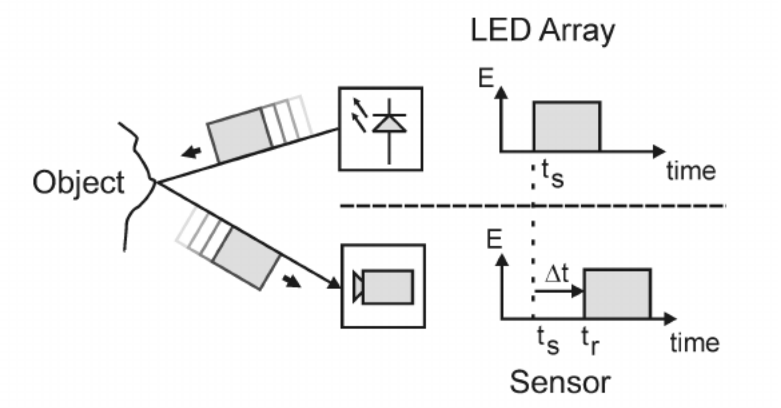


Figure 3.3: Principle operation mode of a time-of-flight camera. Source: [29]

Due to its direct measuring of depth, the computational costs of TOF are very low thus enabling a high frame rate. Since it also uses IR light it provides good results both at bright and low lighting conditions but it reaches a low range compared to the Structured Light and Stereo Vision options. TOF devices have received great attention in the past decades due to the advances in electronics and have become one of the best alternatives for capturing range images. Table 3.1 shows a brief comparison between the reviewed techniques.

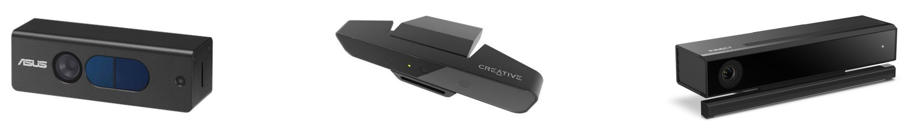
Characteristic	**Stereo vision**	**Structured light**	**TOF**
Correspondence problem	Yes	Yes	No
Extrinsic calibration	Yes	Yes	No
Auto illumination	No	Yes	Yes
Performance on untextured surfaces	Bad	Good	Good
Image resolution	High	High	Medium

Table 3.1: Comparison between stereo vision, structured light and time-of-flight. Source: [4]

3.2 Range Imaging Devices

Image acquisition has to provide good quality data in order to solve the problem of registration. It must be considered that even the best algorithm will strive to produce acceptable results if fed with noisy and highly distorted images, so the quality of the acquisition equipment plays a vital role in the registration process. However, better equipment is often expensive and difficult to use and the recent development of affordable and open source devices has led the scientific community to search for a midway solution.

This is where devices such as the Microsoft Kinect [2], the Asus Xtion [3] or the Intel RealSense [1] provide the necessary tools for developing both affordable and reliable solutions.



(a) ASUS Xtion 2 (b) Intel RealSense SR300 (c) Kinect for Windows v2

Figure 3.4: Consumer range imaging devices. Sources: [5], [15], [11]

3.2.1 ASUS Xtion

The Asus Xtion 2 is a compact device (11 x 3.5 x 3.5 cm) with a 5MP RGB camera and a 640x480 depth sensor which provides images at a rate of 30 Hz. It can sense objects from 0.8 to 3.5 m from the camera, and its FOV is 74 by 52 degrees. It costs 269.99 $ and uses the structured light technique.

3.2.2 Intel RealSense

Intel has developed several versions of depth sensors based on its RealSense technology. The Intel RealSense SR300 has a depth resolution of 640x480 at a nominal frame rate of 60 Hz and a FOV of 71.5 by 55 degrees. It also provides a 1920x1080 RGB image stream and costs 135.99 $. The RealSense technology uses the coded or structured light technique for depth image acquisition, providing a depth range from 0.3 to 2 meters.

3.2.3 Kinect for Windows

The Kinect 2 is a range imaging camera which operates with the TOF principle, providing a 525x424 depth image and a 1920x1080 RGB image at a frame rate of 30 Hz. It has a FOV of 70.6 by 60 degrees and a depth range of 0.5 - 4 m[1] making it adequate for capturing indoor scenes and small objects. It costs 99.99 $ and the development kit for PC costs another 39.99 $.

In the present work all the images have been acquired with a Kinect 2 sensor. A comparison between the presented range imaging devices is shown in Table 3.2.

[1]According to the Kinect manufacturers, the physical depth range of the Kinect goes from 0.5 to 8 m, but measurement precision drops significantly for points further than 4 m from the camera.

Characteristic	ASUS Xtion 2	Intel RealSense SR300	Kinect for Windows v2
RGB image	2560x1920	1920x1080	1920x1080
Depth image	640x480	640x480	515x424
Frame rate	30 Hz	60 Hz	30 Hz
Imaging technique	SL	SL	TOF
Price	269.99 $	135.99 $	139.99 $

Table 3.2: Comparison between range imaging cameras. Own source.

3.3 3D Registration Algorithms

The problem of consistently aligning two different images to produce a final combined image emerged before the development of 3D imaging systems. Over the years, a broad range of techniques were developed for aligning 2D images which in turn led to algorithms for 3D image registration. Several authors have worked on surveys of the different methods developed over time, such as L.G. Brown [10], B. Zitová and J. Flusser [32] or B. Bellekens et al. [8] among others.

When it comes to RGB-D image registration, two main approaches follow. The first one uses RGB data to align the images as if they were in plain 2D and then performs a fine alignment of the 3D data based on the first transformation. The second approach uses only 3D information for aligning the images, as is the case of the present work. The reason for choosing the second approach was not robustness nor efficiency but rather the simplicity of the desired solution. We wanted to keep the registration algorithms within the tools provided by the PCL library and the usage of RGB image registration algorithms implied the need of specific third party libraries.

The process of registration can be simplified by considering only the problem of aligning one image to another. Once this problem is solved, the next acquired image can be aligned to the previous one and so on in a process called pairwise registration. This simplification can be done when working with a continuous stream of data, and the registration can be performed between each acquisition (real time registration) or after the acquisition process has finished.

In both cases the first decision to make is which pipeline will be used for the pairwise registration. Based on its program work flow, almost every 3D registration application can be gathered under one of the following categories: Iterative Closest Point or Feature based applications.

3.3.1 Iterative Closest Point

The ICP algorithm was first proposed by Besl and McKay [9] for registration of 3-D shapes. It has been since then one of the most used registration algorithms, with many different variants of the original algorithm being developed over the years until today. The idea of the algorithm is to iterate over two steps: the *matching step* to determine correspondences between the two point sets, and the *alignment step*, to compute the global distance error (usually the sum of the squared differences between corresponding pairs) and calculate a transformation to minimize the error. These steps are carried out iteratively until certain convergence criteria is met.

The point sets are called *target* and *source*, the result of the algorithm being the transformation applied to the source in order to register it with the target. The basic structure of ICP is shown in Figure 3.5.

```
input  : Point sets $target$ and $source$, $ConvergenceCriteria$
output: $Transformation$ of $source$ into $target$
while ConvergenceCriteria not met do
    foreach point in source do find closest point of target;
    compute error;
    find transformation to minimize error;
    apply transformation and compute transformedError;
    if $|$ transformedError $-$ error $| < threshold$ then
        end algorithm;
    else
        iterate again;
    end
end
```

Figure 3.5: ICP Algorithm basic structure. Own source.

While all the variants of ICP present this basic structure each of them has its advantages and drawbacks. Ones differ in the way of computing the error metric, others in the transformation estimation procedure, and most of them try to find an efficient and robust method for finding corresponding point pairs. This step is crucial for the outcome of the iterative process and has thus become a broad field of study for many scientists.

The most common methods for pairing are **point-to-point** and **point-to-plane**. The first one finds the closest point by minimizing the euclidean distance of a source point and the target point set, whereas the second method uses normal information of the point sets to minimize the perpendicular distance from the source point to the tangent plane of a target point. Moreover, after the corresponding pairs have been found it is a common practice to reject bad correspondences by means of a certain criteria (maximum distance, normal compatibility, duplicate matches, etc.) in order to better estimate the transformation.

Every iteration of the algorithm is computationally expensive so another goal is to optimize the nearest-neighbor search procedure to speed up the process. Therefore, many ICP applications use kd-trees or octrees for rapid searches and most of them also perform a subsampling of the clouds prior to the matching step. Subsampling must be done carefully so as to preserve all the relevant points of the point clouds. However, the main drawback of ICP is not its speed but the fact that the algorithm can get caught in local minima if the clouds are not initially roughly aligned or they only partially overlap. This leads in many cases to a previous step for a rough alignment of the clouds before they are passed to ICP.

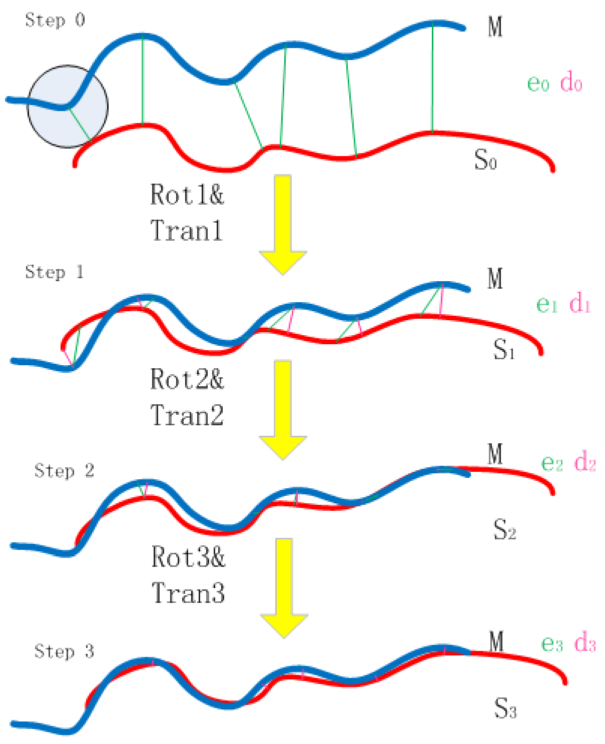


Figure 3.6: Illustration of an ICP process. Source: [30]

3.3.2 Feature Based Algorithms

Although ICP is a robust and well-known registration algorithm, the fact that it may converge to local minima does not make it suitable for every application. For example, in the case of a fast moving sensor consecutive images are only partially overlapped and need to be roughly aligned before they can be processed by ICP. This is where Feature based algorithms play a vital role in the registration process.

The main difference between ICP and Feature based algorithms[2] is the usage of Descriptors for the matching step. This makes the alignment of partially overlapped clouds a feasible task although the resulting transformation is not as precise as the ICP transformation. This is why the process is often called *Coarse Alignment* and is usually followed by a fine alignment via ICP registration.

The steps of Feature based algorithms are the following:

1. **Keypoint Detection**: detect interest points which are distinctive, repeatable and robust
2. **Feature Description**: describe the interest points using 3D information of their surroundings

[2]Also known as Descriptor Matching

3. **Feature matching and rejection**: match interest points of both clouds based on their descriptors and reject bad correspondences

4. **Transformation estimation**: use keypoint matches to estimate a transformation between the clouds

Each of these steps is relevant to the outcome of the whole process but the Keypoint Detection and Feature Description stages are the most discussed topics of Feature algorithms. Many variants of Keypoint Detectors and Feature Descriptors have been developed over the years and PCL library includes a wide range of implementations. The ones considered in this work are presented in the following points.

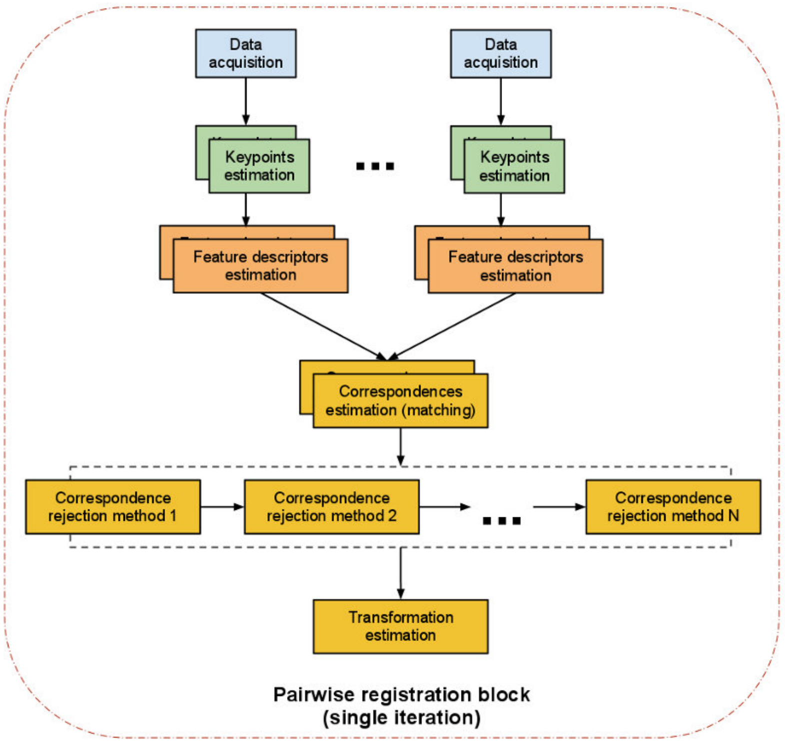

Data acquisition
Data acquisition
Keypoints estimation
Keypoints estimation
Feature descriptors estimation
Feature descriptors estimation
Correspondences estimation (matching)
Correspondence rejection method 1
Correspondence rejection method 2
Correspondence rejection method N
Transformation estimation
Pairwise registration block (single iteration)

Figure 3.7: Pairwise registration pipeline using Feature detection and matching. Source: [13]

3.3.2.1 Keypoint Detection

Descriptor Matching differs from ICP in that only a selected subset of points is used for the alignment process. This reduces the number of calculations needed and also ensures that the correspondences are not ambiguous. These points must be therefore carefully extracted to be easily and robustly paired between clouds. To do that they must be:

- Distinctive: the area surrounding a keypoint should have a unique shape or appearance.

- Repeatable: the keypoint should be detected even if the cloud is rotated, scaled or if the image is taken from a different viewpoint.

A distinctive keypoint will have a unique descriptor that can easily be matched with its corresponding pair in another cloud. As an example, corners are distinctive keypoints whereas points of a plane are similar to each other and are prone to generating bad correspondences in the matching step.

Repeatibility is a condition *sine qua non* for Feature based algorithms because the objective of the resulting transformation is to align points that are the same in both clouds.

The keypoint detectors considered in the present work are SIFT3D, ISS and NARF, which are both included in the PCL library. An example of the keypoints detected by each method is shown in Figure 3.8.

- **SIFT3D**: SIFT3D is a 3D implementation of the SIFT (Scale Invariant Feature Transform) algorithm presented by D.G. Lowe [16]. It was designed as a novel method for detecting and describing scale-invariant keypoints in 2D images. It must be noted that scale is an important factor in 2D images, where two shots taken from different distances to an object produce very different images. In 3D clouds, however the difference between a close and a far shot of the same scene is the density of the cloud.

  SIFT extracts keypoints by computing the Difference of Gaussian of the image intensity through different scales. A slightly modified version of SIFT adapted to 3D clouds was introduced in PCL and has been tested in this work due to its short computational time[6].

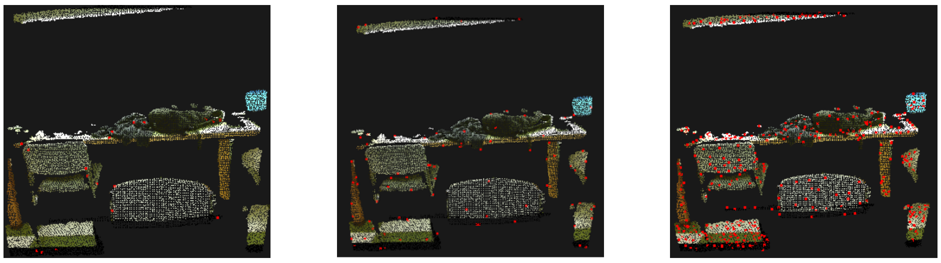
- **ISS**: Intrinsic Shape Signatures was presented in 2009 by Y. Zhong [31] as a novel method to characterize a region of a point cloud, and was ported to PCL by G. Ballin during the Google Summer of Code '12.

  ISS is based on the eigenvalues λ_1, λ_2, λ_3 of the covariance matrix of the support region around a certain point. It computes two ratios: λ_2/λ_1 and λ_3/λ_2 which must be smaller than a user defined threshold to pass to the next stage. Then the saliency of the points is computed and after a Non Maxima Suppression (NMS) stage the final keypoints are obtained.

- **NARF**: Normally Aligned Radial Features is an algorithm developed by B. Steder et al. [27] in 2010. It uses range images and not point clouds as the source for keypoint detection, aiming at detecting borders and keypoints where a surface is stable but the neighboring points present substantial change.

  NARF requires the computation of a range image from a point cloud in order to detect the keypoints. It is a fast algorithm which performs best in non-noisy clouds.

ISS yielded the best results in terms of invariance and repeatability in the comparative evaluation by S. Filipe and L.A. Alexandre[14]. Although the tests performed by G.Ballin [6] show that ISS is slower than SIFT3D and NARF, computing time was reduced significantly by using 4 threads instead of 1 for keypoint detection. Furthermore, the initial alignment provided with ISS keypoints proved to be the most robust of the three.



(a) NARF Keypoints (b) SIFT3D Keypoints (c) ISS Keypoints

Figure 3.8: Comparison of keypoints detected by three different methods. Own source.

3.3.2.2 Feature Description

A feature descriptor is a compact representation of the surrounding region of a point aimed at detecting similarities between surfaces. Among the various types of descriptors included in PCL, the present work uses the Fast Point Features Histogram (FPFH) descriptor. It was developed by R.B. Rusu et. al [21] for its usage in 3D point cloud applications.

FPFH is an extension of PFH (Point Feature Histogram) to make the computation of features faster. PFH creates a histogram for each desired point p_q that can be compared to establish correspondences. To compute the histogram, the algorithm follows these steps:

1. Define a set of points P which are neighbors to the desired point p_q.
2. For each point pair in the neighborhood of p_q, define a point source p_s and point target p_t.
3. With the information of the normals n_s and n_t, compute three values which express the mean curvature at point p_s.
4. Quantizise the values and increment the correspondent bin of the histogram of p_q.
5. Repeat until all combinations of point pairs in point set P have been considered.

The quantizing step makes b subdivisions of the three values, so b^3 possible combinations of value ranges exist and the final histogram has b^3 bins. The default subdivision size of PFH is 5 and the correspondent histogram is stored as a vector of 125 values which represents surface variations around point p_q.

FPFH has proven to be very useful with 2.5D images (range images) and thus has been used in our work, although point clouds have to be filtered to remove significant noise for FPFH to produce robust results.

3.3.2.3 Feature Matching

The next step of Feature based algorithms is to establish correspondences between the keypoints of two clouds using their Feature information. This is done by searching the nearest point to each source cloud point in the feature-space of the target cloud. A naive approach would search then estimate a transformation to bring all the corresponding points together. However, it is common to have points with similar features which do not correspond to each other in the 3D space. A correspondence rejection step is therefore needed prior to the transformation estimation.

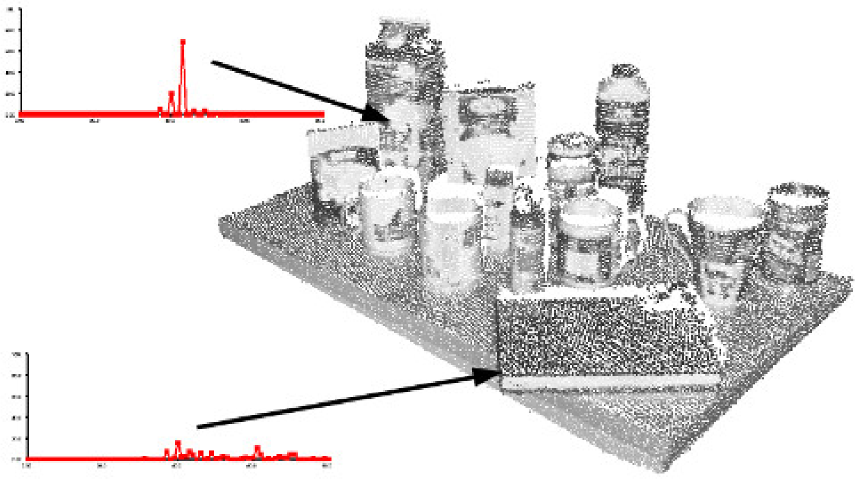


Figure 3.9: Example of Point Feature Histograms of two different points. Source: [12]

The most common algorithm for rejecting false correspondences is the Random Sample Consensus or RANSAC. RANSAC is a random iterative method for detecting outliers in a mathematical model, and it is used in registration applications to detect which correspondences will lead to an incorrect transformation. To do so, it selects a random subsample of correspondences and rapidly estimates a transformation using the subsampled points. The transformation is then evaluated with the rest of correspondences, keeping the ones which lie under a certain threshold. If the number of correct correspondences is higher than a minimum amount, it is considered to be a good transformation. This process is iteratively repeated until a certain level of inliers is found.

```
input  : Point sets target and source, correspondences
output: Inlier correspondences and best transformation
1  while not maximumIterations do
2      select a random subset of correspondences;
3      compute best transformation according to subset correspondences;
4      apply transformation to evaluate the rest of correspondences;
5      define the accuracy of transformation by the number of correspondence inliers;
6      if inlierProportion > threshold then
7          end algorithm;
8      else
9          iterate again;
10     end
11 end
```

Figure 3.10: RANSAC outlier rejection algorithm. Own source.

3.3.2.4 Transformation Estimation

The final step of the coarse alignment performed by Feature based algorithms is to estimate a transformation between the source and target clouds using the inlier correspondences from the rejection step. In some cases, such as the PCL RANSAC outlier rejection algorithm, this rejection step provides a transformation based on the best random subset of correspondences used to detect inliers. However, after all the inliers have been identified, it is recommended to recalculate the transformation using all the inliers and a more accurate estimation method.

PCL provides two different methods for estimating a transformation:

- **SVD Transformation Estimation**: this method uses the singular value decomposition of the covariance matrix of the data to estimate the best possible transformation of one cloud into another in a single step.
- **LM Transformation Estimation**: this is a method based on the Levenberg-Marquardt least squares algorithm. It is an iterative process which requires an initial guess which can handle non-linear least squares problems.

The LM algorithm is used by some ICP variants included in PCL, especially in those whose cost function changes at each iteration. In our case, it is better to use the SVD method as it computes a transformation in a single step.

When a transformation has been calculated, the pairwise registration process has finished and an initial coarse alignment of both clouds has been found. This coarse alignment serves as the initial guess for the fine ICP alignment which should now converge to a global minimum instead of a local one.

3.3.3 Algorithm Weaknesses and Limitations

Both ICP and Feature based algorithm have their limitations. We have discussed that ICP can converge to local minima due to the initial relative position of both clouds. In the case of point-to-plane ICP, the result of the iterative process depends highly on the precision with which the point normals are calculated.

When it comes to Feature based algorithms, there are more steps where a small error can lead to a failure in the process. For example, detecting few keypoints makes it harder for RANSAC to find an appropriate transformation, and if the matching step produces a high amount of false correspondences or the rejection algorithm identifies many false correspondences as inliers the transformation estimation will not provide a successful transformation.

It is therefore necessary to perform a series of tests with real data to adjust the parameters and conditions of every algorithm of the process. It must be taken into account that there is a trade-off between precision and speed, and that the result of the process does not only depend on algorithm parameters but also on the quality of the data. The solution presented in this work has been adjusted to provide a result in a computation time less than 10 seconds. For real time applications many variations and optimization techniques exist but they have not been considered in our case.

3.4 Point Cloud Filtering

Due to the relevance of the data quality in terms of robustness for the registration algorithms, it is necessary to refine the raw data provided by the range imaging sensor. Sometimes a downsampling is also carried out in order to reduce the number of points and speed up the process. This must be done carefully to not lose any relevant information of the cloud.

PCL provides a wide range of filters for this purpose. A group of them are focused on smoothing surfaces or removing noise from the data:

- *Fast Bilateral Filter*: it is a nose reduction algorithm that preserves edges of the cloud and produces a smooth result. It is derived from the filter designed for 2D images, which replaces the intensity of each pixel with a weighted value based on a Gaussian distribution. An implementation of the Bilateral Filter is also included in the libraries used for Kinect v2 image acquisition.
- *Radius Outlier Removal*: given a spherical radius r and a minimum number of neighbors $minNeighbors$, it removes the points of the cloud which have less than $minNeighbors$ points in a sphere of radius r centered in the point. If the cloud comes from a range image, the computation of the neighbors is fast because the points can be transformed to a 2.5D space in a pixel-like structure.
- *Statistical Outlier Removal*: the principle of operation of the Statistical Outlier Removal method is the same as the Radius Outlier Removal. In this case, a point is removed if its average distance to its k nearest neighbors is statistically larger than the average distance of the rest of cloud points. The usual threshold for removing points lies between one or two standard deviations from the mean distance, and is slower than the Radius Outlier Removal filter.
- *Moving Least Squares*: it is a surface reconstruction method for smoothing and resampling noisy data. It fits a polynomial surface to the sampled points and then moves the points to a new position in the fitted surface. It is a relatively slow method and thus is not suitable for real time applications. In this work it is used as a final smoothing method for reconstructing the surface of the final registered clouds.
- *PassThrough Filter*: the PassThrough Filter does not qualify as a noise reduction algorithm but as a conditional filter. It removes the points which do not meet a condition defined by the user such as a maximum z value. In our case it is used to crop the depth data by a z value to avoid the farther points due to their planar distortion.

As for downsampling, PCL presents two methods which are based on dividing the 3D space in a grid and downsampling the points within each grid:

- *Sampling Surface Normal*: it divides the space into grids until each grid contains a maximum on N points. Then it computes a normal using all the points in the grid, randomly samples a ratio of points from the grid and assigns them the same normal vector.
- *Voxel Grid Filter*: the Voxel Grid Filter is the most used filter of PCL for downsampling clouds. It divides the space into equal voxels and replaces all the points in a voxel with

their centroid. It is a more stable method than Sampling Surface Normal and it produces a cloud with constant point density. An optimization of this filter called Approximate Voxel Grid Filter can be used for speeding up the computation but it reduces the quality of the result. An example of the effect of a Voxel Grid Filter is shown in Figure 3.11.

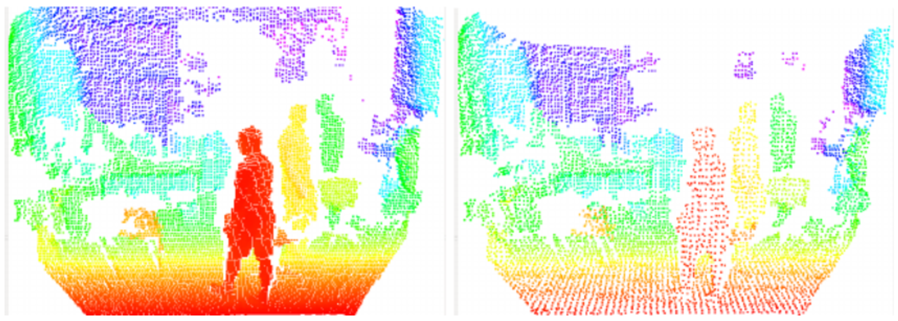


Figure 3.11: Example of the effect of the VoxelGrid Filter on Kinect data. Source: [18]

4 Development of the Program

4.1 Development Environment

4.1.1 Hardware

The hardware used in the development of the project consists in one Kinectv2 sensor and an Acer Aspire E15 laptop computer. The Kinectv2 specifications have already been presented in the Analysis of Background. As for the Acer specifications, they are the following:

- *CPU*: Intel Core i5-7200U 2.5 GHz[1]
- *RAM*: 8 Gb DDR4 RAM
- *Graphics Card*: NVIDIA GeForce 940 MX with 2 Gb dedicated VRAM
- *Storage*: 256 Gb SSD

Another required specification for the computer was to have at least one USB 3.0 port because due to the high data transmission rate of the Kinectv2, it does not work with USB 2.0 connectivity ports. No additional hardware was required for the development of the project.

4.1.2 Software

Three main software components were used for the programming of the registration program: the C++ programming language, libfreenect2 open source library and the Point Cloud Library in its 1.7 development version. Everything was programmed in an Ubuntu 16.02.3 LTS distribution using Sublime 3 for text editing and Cmake for compilation. The code of the project is distributed as free software and available at GitHub.

[1]Up to 3.1 GHz with Turbo Boost

4.2 Program Structure

In order to organize the code in an efficient way and avoid unnecessary repetitions, it is structured following the Object Oriented Programming paradigm. In this way the program is subdivided into several classes which interact with each other in an organic way. From this code subdivision follows a file subdivision: the program is divided into files, each of them storing the code of a respective class. In order to include all the different C++ files in the main program, we use C++ header files.

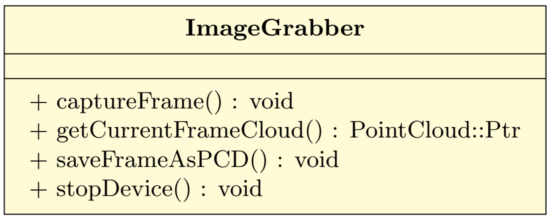
The program classes are the following:

- **ImageGrabber**: located in the file *acquisition.cpp*, it provides methods for capturing new frames, saving them in the .pcd format and stopping the communication with the Kinectv2 device. When an ImageGrabber object is instantiated it automatically searches for any Kinectv2 connected to the computer and starts to communicate with it.
- **CoudFilters**: located in the file *filterCloud.cpp*, it provides several cloud filtering methods: Radius Outlier Removal, PassThrough and Voxel Grid filters. It also has a method for applying all three filters to a cloud, one after another.
- **CoarsePairwiseAligner**: located in the file *coarse_align.cpp*, it includes the Feature based algorithm for the coarse alignment of one cloud into another. This class makes use of the KeypointDetector class for the calculation of point cloud keypoints. It has to main methods:
  - *align*: takes a *source* and a *target* cloud as an input and returns the transformation of *target* into *source*
  - *alignToLast*: takes a *source* cloud as an input and uses the last inputted cloud as the *target* to compute the transformation.

  Using these two methods we can perform a coarse pairwise alignment of a stream of incoming data, aligning the new image to the last processed one. This class also provides methods for getting useful data calculated within the process such as cloud normals, keypoints or features, in case this data is required by other algorithms or for visualization purposes.
- **KeypointDetector**: the KeypointDetector class provides several keypoint detector algorithms which can be used by the CoarsePairwiseAligner to compute point cloud keypoints. Currently there are three different keypoint types: SIFT3D, NARF and ISS3D keypoints, with ISS3D as the default type. It is located in the *kp_detectors.cpp* file.
- **FineAligner**: this class implements the PCL ICP point-to-plane algorithm to perform a fine alignment of two point clouds. Its main method is called *align*, which takes a *source* cloud, a *target* cloud and an *initial guess* transformation matrix as an input and returns the final transformation outputted by the ICP algorithm. It also provides a method for setting the maximum number of iterations of ICP and is located in the *fine_align.cpp* file.
- **SurfaceReconstructor**: located in the *surface_reconstruction.cpp* file, this class provides one method with two different options for the reconstruction of surfaces from a point cloud: the Greedy Projection Triangulation and the Poisson algorithm. Prior to passing the cloud to any of the reconstructors, it applies a Moving Least Squares filter to smoothen the surfaces of the cloud and then downsamples them with a Voxel Grid filter.

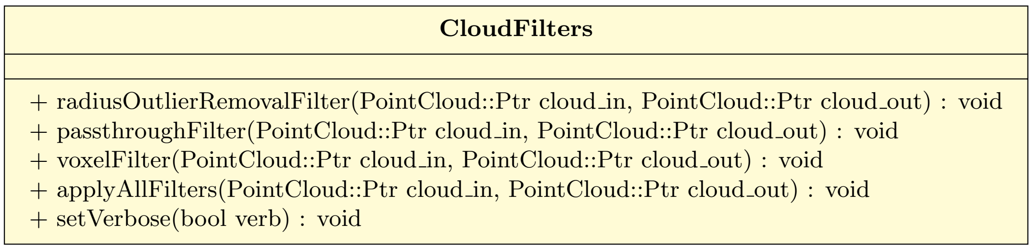
- **Viewer**: this is the last class of the program and it is designed for visualizing the results using the PCLVisualizer class. It is located in the *visualization.cpp* file and provides several methods for updating the displayed point clouds as the registration process gets new input data. This class also handles the keyboard input for the interaction with the user.

A detailed representation of the public methods of each class can be seen in the following figures.


ImageGrabber

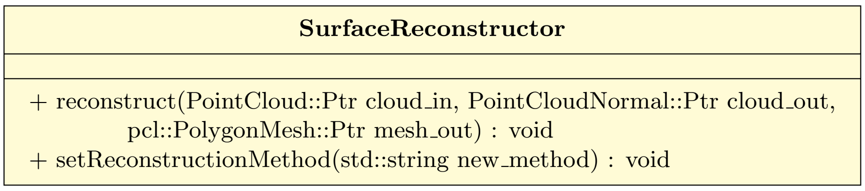
+ captureFrame() : void
+ getCurrentFrameCloud() : PointCloud::Ptr
+ saveFrameAsPCD() : void
+ stopDevice() : void

Figure 4.1: ImageGrabber class


CloudFilters

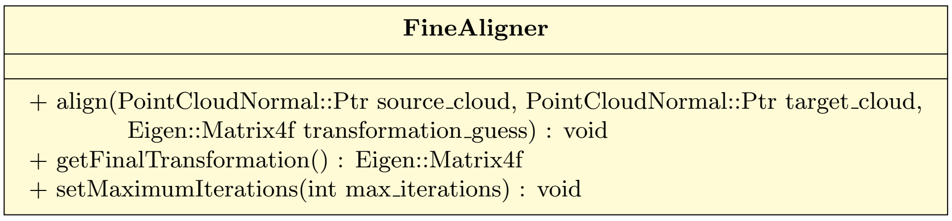
+ radiusOutlierRemovalFilter(PointCloud::Ptr cloud_in, PointCloud::Ptr cloud_out) : void
+ passthroughFilter(PointCloud::Ptr cloud_in, PointCloud::Ptr cloud_out) : void
+ voxelFilter(PointCloud::Ptr cloud_in, PointCloud::Ptr cloud_out) : void
+ applyAllFilters(PointCloud::Ptr cloud_in, PointCloud::Ptr cloud_out) : void
+ setVerbose(bool verb) : void

Figure 4.2: CloudFilters class


SurfaceReconstructor

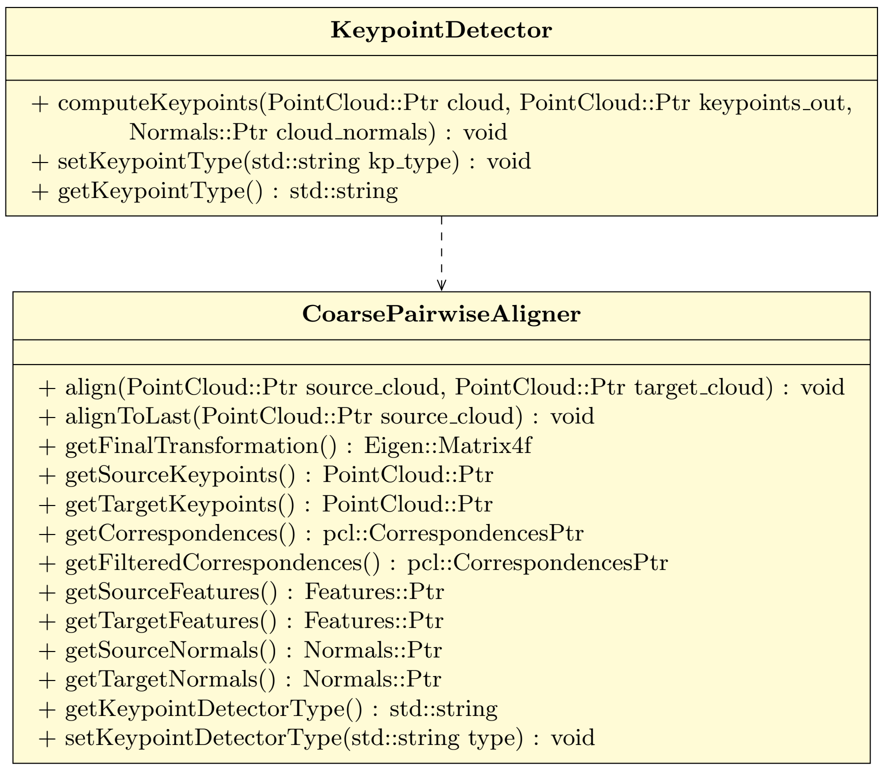
+ reconstruct(PointCloud::Ptr cloud_in, PointCloudNormal::Ptr cloud_out, pcl::PolygonMesh::Ptr mesh_out) : void
+ setReconstructionMethod(std::string new_method) : void

Figure 4.3: SurfaceReconstructor class


FineAligner

+ align(PointCloudNormal::Ptr source_cloud, PointCloudNormal::Ptr target_cloud, Eigen::Matrix4f transformation_guess) : void
+ getFinalTransformation() : Eigen::Matrix4f
+ setMaximumIterations(int max_iterations) : void

Figure 4.4: FineAligner class



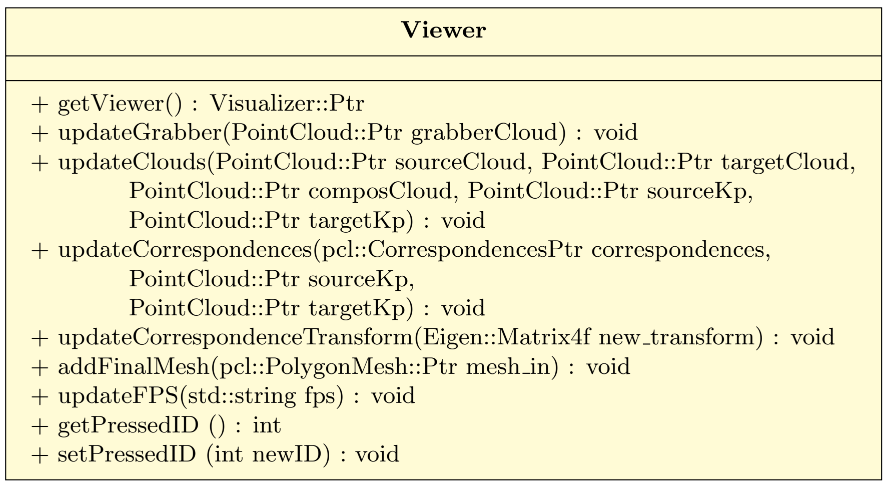
```
KeypointDetector

+ computeKeypoints(PointCloud::Ptr cloud, PointCloud::Ptr keypoints_out,
            Normals::Ptr cloud_normals) : void
+ setKeypointType(std::string kp_type) : void
+ getKeypointType() : std::string
```

```
CoarsePairwiseAligner

+ align(PointCloud::Ptr source_cloud, PointCloud::Ptr target_cloud) : void
+ alignToLast(PointCloud::Ptr source_cloud) : void
+ getFinalTransformation() : Eigen::Matrix4f
+ getSourceKeypoints() : PointCloud::Ptr
+ getTargetKeypoints() : PointCloud::Ptr
+ getCorrespondences() : pcl::CorrespondencesPtr
+ getFilteredCorrespondences() : pcl::CorrespondencesPtr
+ getSourceFeatures() : Features::Ptr
+ getTargetFeatures() : Features::Ptr
+ getSourceNormals() : Normals::Ptr
+ getTargetNormals() : Normals::Ptr
+ getKeypointDetectorType() : std::string
+ setKeypointDetectorType(std::string type) : void
```

Figure 4.5: CoarsePairwiseAligner and KeypointDetector classes



```
Viewer

+ getViewer() : Visualizer::Ptr
+ updateGrabber(PointCloud::Ptr grabberCloud) : void
+ updateClouds(PointCloud::Ptr sourceCloud, PointCloud::Ptr targetCloud,
            PointCloud::Ptr composCloud, PointCloud::Ptr sourceKp,
            PointCloud::Ptr targetKp) : void
+ updateCorrespondences(pcl::CorrespondencesPtr correspondences,
            PointCloud::Ptr sourceKp,
            PointCloud::Ptr targetKp) : void
+ updateCorrespondenceTransform(Eigen::Matrix4f new_transform) : void
+ addFinalMesh(pcl::PolygonMesh::Ptr mesh_in) : void
+ updateFPS(std::string fps) : void
+ getPressedID () : int
+ setPressedID (int newID) : void
```

Figure 4.6: Viewer class

4.3 Image Acquisition

The first step of the program is to acquire RGBD images from the Kinectv2 and store them in a format which can be later processed using the PCL toolbox. To do this the libfreenect2 open source Kinectv2 drivers were used.

The libfreenect2 library provides the necessary tools to acquire RGB, Depth and IR data from the Kinectv2. It also includes useful functions for registering RGB data onto the Depth image and vice-versa, in order to obtain a RGBD image. It must be noted that the resolution of RGB and Depth images differ from each other: RGB has 1920x1080 pixels while Depth has 512x424 pixels.

If one chooses to register RGB onto Depth data, he result will be a 512x424 depth image with colour. However, if the real FOV of the RGB camera does not fully overlap the Depth camera FOV some pixels will not have color information. The same principle applies for registering Depth onto RGB data: the result will be a 1920x1080 color image with depth, but in this case adjacent color pixels might have the same depth value or even no depth data if the Depth FOV does not fully overlap the RGB FOV.

Roland Smeenk developed a web tool called *Kinect FOV explorer*[2] which illustrates this principle. In Figure 4.7 it can be seen that the real RGB FOV is wider than the Depth one whereas the Depth FOV is higher than the RGB one. This results in RGB to Depth registered images having two black bands in the upper and lower image sections and in Depth to RGB registered imaged having no depth data in the left and right image borders, as shown in Figure 4.8.

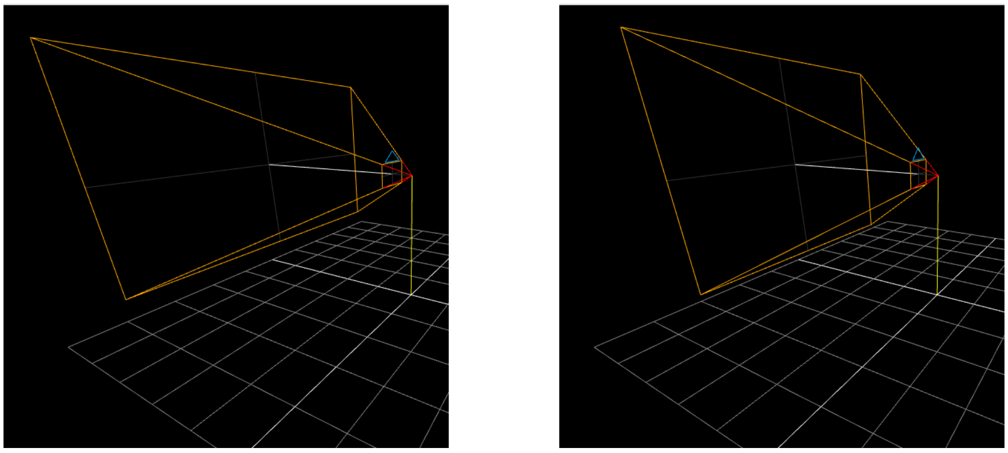

(a) RGB FOV (b) Depth FOV

Figure 4.7: Kinectv2 fields of view compared. Source: own elaboration from [25]

In this project the RGB to Depth registration has been used for two main reasons. The first one is that registering Depth to RGB means that some RGB pixels will be assigned the depth value of its nearest pixel of the depth image[3]. This approximation can create fake depth pixels

[2]Available here
[3]This happens due to the difference in point density between RGB and Depth images

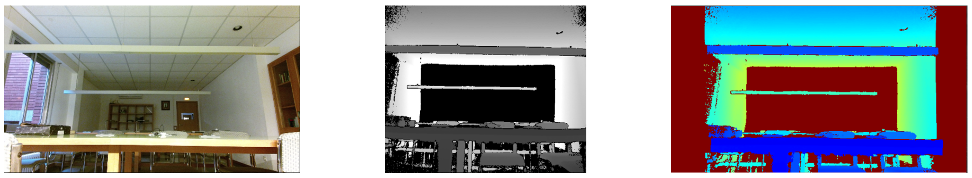

(a) RGB capture (b) Depth capture (c) Depth mapped onto RGB

Figure 4.8: Example output of mapping Depth onto RGB. Own source.

that may difficult the registration process. The second one is that even if no fake pixels are created, the 1920x1080 RGBD image has no additional depth information than the 512x424 RGBD image but it does have 9.5 times more depth points. Having such an amount of points will greatly increase the computational time required to process them. This is the second reason why the RGB to Depth registration was used in this project.

4.3.1 From 2.5 D to 3 D

An example output of the RGB to Depth registration can be seen in Figure 4.11a. However, the data is currently in a so called 2.5 D format: the program has an image made of pixels with their respective i, j pixel indices and two information fields: a float representing the color of the pixel in a RGB format and a depth value in meters[4]. This gives us the real world Z value of each pixel, but for a point cloud representation the X and Y real world coordinates need to be obtained.

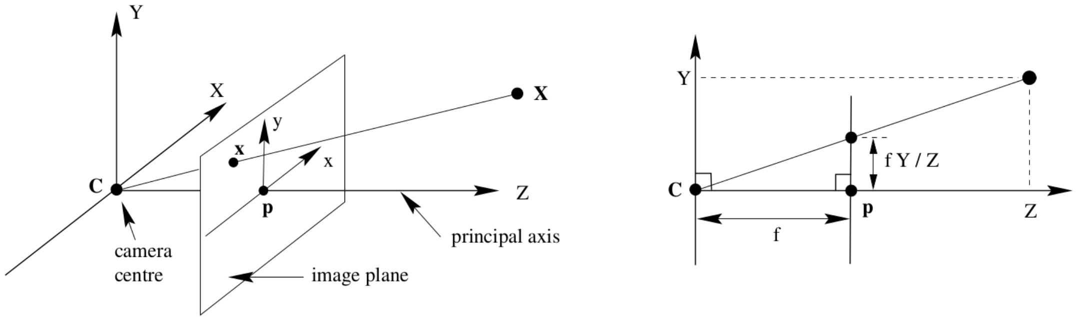


Figure 4.9: Illustration of the pinhole camera model. Source: [20]

Thankfully there is an easy way of computing these values. Due to the fact of Kinectv2 being a camera and not a rotating scanner, the data captured by it follows the pinhole camera model. Figure 4.9 illustrates this principle, where it can be seen that given the focal length of the camera f in pixels, the real world Z distance of the point to the camera and the indices i, j of a given pixel p_{ij}, the real world X, Y coordinates of p_{ij} can be easily calculated as:

$$j = f \times \frac{Y}{Z} \implies Y = j \times \frac{Z}{f} \quad and \quad X = i \times \frac{Z}{f} \tag{4.1}$$

[4]The depth data collected from the Kinectv2 using libfreenect2 is stored as a float and the units used are meters

Libfreenect2 provides an implementation of this calculation which uses default Kinectv2 focal distances. After the computation of each pixel's X and Y values, the last step is to store the data in the PCL format. The program iterates through all the pixels of the registered RGBD frame, computes the real world coordinates of each pixel and stores them in a PCL PointCloud object[5] which can be processed by the PCL toolbox.

One last annotation that must be made to the acquisition process is that we only store the central 400x400 pixels from the RGBD image. This is because the borders of the depth image present significant noise, with values that clearly do not correspond to real depth values. To avoid storing false measurements, only the center part of the image is stored. All this process is done by the function `captureFrame`. An example of a resulting cloud is shown in Figure 4.11a.

```
void ImageGrabber::captureFrame ()
{
   // Release previous frames
   listener->release(frames);
   // Wait for frames
   listener->waitForNewFrame(frames);
   // Get frames
   rgb_frame = frames[libfreenect2::Frame::Color];
   ir_frame = frames[libfreenect2::Frame::Ir];
   depth_frame = frames[libfreenect2::Frame::Depth];
   // Register rgb to depth
   registration->apply(rgb_frame, depth_frame, undistorted_frame, registered_frame);
   // Fill pcl cloud
   frame_cloud->width = image_width;
   frame_cloud->height = image_height;
   frame_cloud->is_dense = false;
   frame_cloud->points.resize (frame_cloud->width * frame_cloud->height);
   float X, Y, Z, RGB;
   int counter = 0;
   // For each pixel inside the desired range
   for (int xi = width_gap; xi < (registered_frame->width - width_gap); xi++)
   {
      for (int yi = height_gap; yi < (registered_frame->height - height_gap); yi++)
      {
         // Get real world coordinates
         registration->getPointXYZRGB(undistorted_frame,registered_frame,yi,xi,X,Y,Z,RGB);
         frame_cloud->points[counter].x = -X; // mirror image
         frame_cloud->points[counter].y = Y;
         frame_cloud->points[counter].z = Z;
         frame_cloud->points[counter].rgb = RGB;
         counter++;
      }
   }
}
```

Figure 4.10: ImageGrabber's captureFrame function. Own source.

[5]In this process, the real X values relative to the origin of the camera are inverted due to the original Kinect image being mirrored

4.4 Preprocessing

A closer look to the raw image provided by libfreenect2 shows that it presents significant planar distortion in the farther points and some noise in those points closer to the camera, thus requiring point cloud filtering prior to any in-depth data analysis. The planar distortion could be corrected by performing an accurate calibration of the camera. However, even though there is significant distortion in the farther points, if the interest points lay in the middle range of the camera no extrinsic calibration is needed. This is the case of the present work, and it motivates the first filter used in the preprocessing pipeline prior to the registration algorithms[6].

4.4.1 Cropping by Z values

In the analysis of background we have presented the specifications of the Kinectv2, one of which is the depth range of the camera: from 0.5 to 4 meters. To discard the points with planar distortion, we use a PassThrough filter and erase all the points of the cloud with a Z value of more than 2 meters. This is a very fast process which gives us a new point cloud with which theoretically do not present distortion, although there are still some chunks of noisy points which have to be removed.

4.4.2 Subsampling

The second preprocessing step aims at reducing computation time while keeping the point cloud as close as possible to the original. In order to reduce the calculations needed in the following processes, we carry out a subsampling with a Voxel Grid Filter. The only parameter to set is the leaf size i.e. the length of the side of the cubes used for dividing the space into a 3D grid. In this case we use 0.01 meters as a leaf size. The Voxel Grid filter also helps to unify the point cloud density.

4.4.3 Noise Removal

The final step aims at removing the noisy data from the point cloud. For that purpose we have chosen to use the Radius Outlier Removal Filter. The principle of the filter has been discussed in the second chapter, stating that the parameters that define the rejection threshold are: the spherical radius which defines the neighbors of a point and the minimum number of neighbors a point needs in order to be considered not noisy. If we estimate that the surface of interest will stand 1.5 meters away from the camera, given pixel density of the Kinectv2 (7 pixels per degree) and an approximate focal length of 370 pixels, this 7 pixels will be within:

$$7 \times \frac{1.5}{370} \approx 0.03\, meters \tag{4.2}$$

from the considered point. Given that this means 7 points just in one direction, if we consider a whole sphere of radius 0.03 it is fairly easy for a good point to have at least 20 neighbors. These

[6]There is a previous step to the first filter which removes all the points of the cloud which have NaN (Not a Number) values

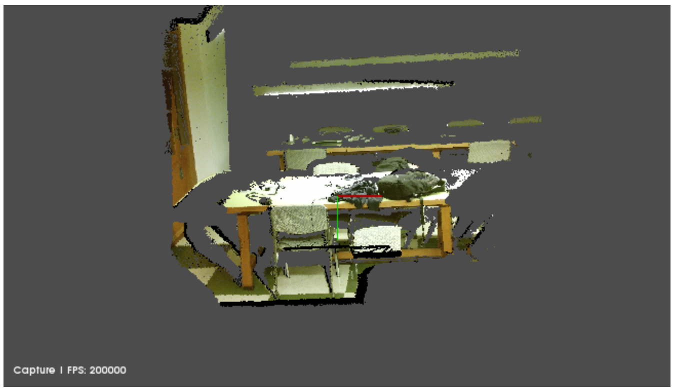
are the parameters used for the Radius Outlier Removal Filter which have proved to produce non-noisy clouds from Kinectv2 data.

	Voxel Grid Filter	**Radius Outlier Removal**	**All filters**
Points removed	64.82%	6.01%	70.83%

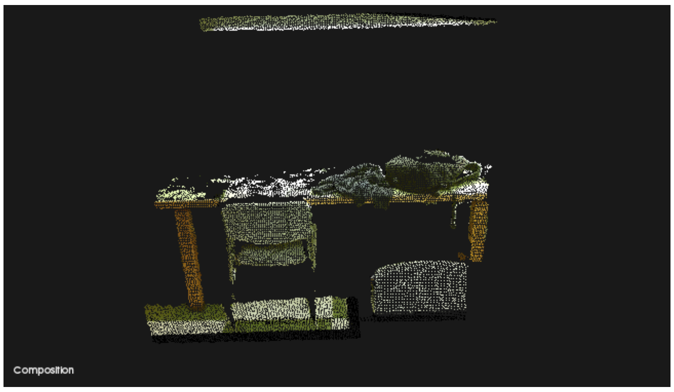
Table 4.1: Percentage of points removed by the filters*. Own source.
*Relative to the points which pass the PassThrough filter

4.4.4 Results

A series of tests took a mean time of 0.1 seconds for applying the three filters to real point clouds. More results can be seen in Table 4.1. After the preprocessing step, we obtain a cloud with low planar distortion, small amount of noise and with an almost constant point density, as shown in Figure 4.11 This cloud can then be passed onto the alignment algorithm.



(a) Non-filtered cloud



(b) Filtered cloud

Figure 4.11: Point cloud before and after preprocessing. Own source.

4.5 Coarse Alignment

Once the depth data is filtered it can be processed by the coarse alignment algorithm. The steps of this algorithm were discussed in the analysis of background: keypoint detection, feature description, correspondences estimation and rejection and finally compute an estimation of the best transformation. The methods selected for carrying out these steps are presented in the following.

4.5.1 ISS Keypoint Detection

The ISS 3D Keypoint Detector has proven to be the most robust keypoint detector method when working with our Kinect data. Even though it is not the fastest detector (see Figure 4.12 - the time required for computing keypoints increases linearly with the size of the input cloud for all three methods), it yields enough keypoints for the correspondence estimation step.

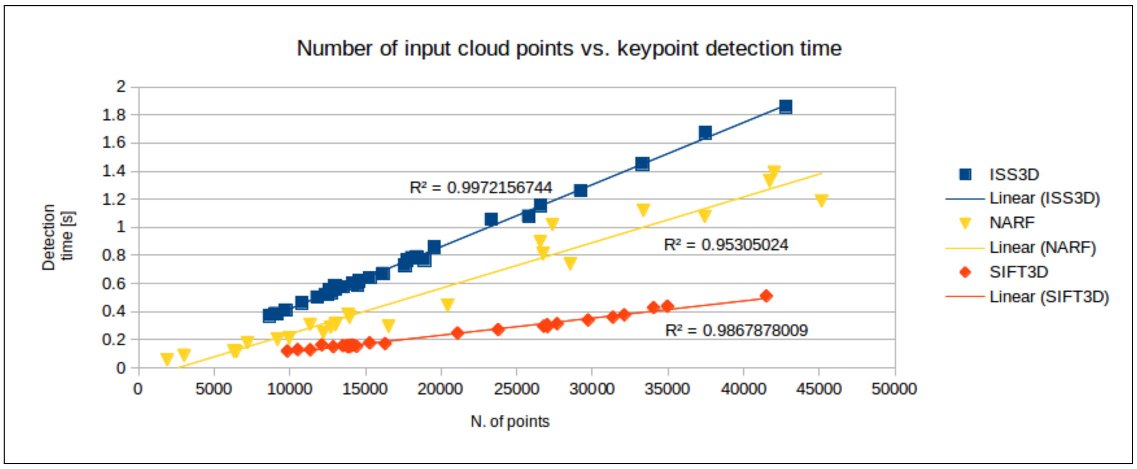


Figure 4.12: Computing times of the different keypoint detection methods. Own source.

Both SIFT3D and NARF are faster but detect a significantly smaller amount of keypoints which are too few for establishing correspondences. A comparative test shows that with similar threshold parameters[7] and with the same input point cloud, ISS3D is the best option for detecting keypoints which lead to a good correspondence estimation.

	ISS3D	**SIFT3D**	**NARF**
% of keypoints detected (relative to ISS3D results)	100.00%	17.82%	4.89%
% of true correspondences (relative to ISS3D results)	100.00%	16.34%	9.01%
true to total detected correspondences ratio	19.18%	16.58%	31.46%[8]

Table 4.2: Comparative test of ISS3D, SIFT3D and NARF detection algorithms. Own source.

[7]The parameters used were the default parameters suggested by the authors of each detector algorithm

[8]NARF results are misleading because in many cases the number of detected correspondences was almost the minimum number of correspondences which RANSAC samples for estimating a transformation, thus giving a higher ratio of true to total correspondences

As seen in in Table 4.2, SIFT3D detects 82.18% less keypoints than ISS3D, with NARF detecting 95.11% less. The correspondences found with these keypoints are then evaluated to reject false ones. Even though this true-to-total correspondences ratio is similar between SIFT3D and ISS3D, SIFT3D keypoints provide 83.66% less true correspondences than ISS3D (NARF results cannot be compared because they are close to the random sample size of RANSAC rejection algorithm). This leads to 83.66% less information for estimating a good initial transformation between the two point clouds. We have thus decided to use the ISS3D keypoint detector in our project.

However, the fact that ISS3D yields the most keypoints does not mean that it passes all the cloud points to the next alignment steps: its goal is to select a number of reliable candidate keypoints in order to speed up the process. Even though there is no ”input cloud size to number of keypoints detected” ratio, the tests performed with our data show that by applying the ISS3D detector the detected keypoints are on average 2.1% of the input cloud points[9]. Thus the computation time of the next steps is greatly reduced because they only need to process 2.1% of the original cloud points while the results obtained are still accurate.

The `ISSKeypoint3D` class implemented by G. Ballin[7] can be run in two different modes: with and without performing a boundary estimation. The border estimation step detects boundary points which are discarded while computing the final keypoints. This has a time penalty but we consider that due to the keypoint detection step not being the bottleneck of the process, the border estimation is worth applying.

The algorithm parameters recommended by the author are the following:

- `gamma_21`: 0.975 - upper threshold for λ_2/λ_1
- `gamma_32`: 0.975 - upper threshold for λ_3/λ_2
- `min_neighbors`: 5 - minimum neighbors required for the NMS algorithm
- `salient_radius`: 4·`model_resolution`[10]- spherical radius used to compute the scatter matrix
- `non_max_radius`: 4·`model_resolution` - spherical radius used in the NMS algorithm
- `normal_radius`: 4·`model_resolution` - spherical radius used for estimating the cloud normals
- `border_radius`: 1·`model_resolution` - spherical radius used for border detection

4.5.2 Feature Matching and RANSAC Rejection

The keypoints detected by the ISS algorithm are passed to the next step, where a FPFH descriptor is computed for each keypoint using the `FPFHEstimationOMP` PCL class. This class follows the Open Multi-Processing standard which enables parallel processing for a rapid computation.

[9]Values ranged from 1.5% to 2.5%

[10]The resolution of the point cloud, calculated as the average nearest neighbor distance of the whole cloud

Another class which uses this standard is `NormalEstimationOMP`, used for calculating the normals of the point cloud which are required by the FPFH estimator. The parameters used for the normal estimation are `NumberOfThreads` = 8 and `RadiusSearch` = 0.03 meters, whereas for the FPFH estimation they are set to 4 and 0.03 respectively.

The keypoints and their features can be then passed to a `CorrespondenceEstimation` object which carries out the matching step. It uses the feature descriptors to compare the clouds and establish feature correspondences. This can be performed in two different ways:

- *Source to target correspondence estimation*: for each keypoint in cloud *source*, establish its correspondences of cloud *target* in the feature space. One source keypoint can have zero, one or several target keypoints as correspondence candidates.
- *Reciprocal correspondence estimation*: for each keypoint in cloud *source*, establish its correspondences of cloud *target* in the feature space and vice versa. Only the reciprocal correspondences are kept so one source keypoint can only have either one or no target keypoint as a correspondence candidate.

This project uses the reciprocal estimation approach and then passes the keypoints and the estimated correspondences to the RANSAC rejection algorithm in order to detect and remove false correspondence pairs. PCL's `CorrespondenceRejectorSampleConsensus` class provides the necessary methods for correspondence rejection. The algorithm parameters defined are:

- `InlierThreshold`: the maximum distance between two corresponding points. A pair of points which is farther than this threshold will not be considered as inliers. It is set to 0.2 meters because the fine aligner algorithm can still handle clouds misaligned by 0.2 meters.
- `MaximumIterations`: if the algorithm does not find the minimum number of inliers it will continue sampling random subsets of keypoints until it either considers all the possible combinations or it reaches a predefined number of maximum iterations. In this project the maximum iterations is set to 1000.

This parameter values yield the results discussed in the previous section: an average of 80.82% of the initial correspondences are rejected by the RANSAC algorithm. The last step is to compute the best possible transformation using the true correspondences.

4.5.3 SVD Transformation Estimation

In order to compute the final coarse transformation, `TransformationEstimationSVD` class is used. It first computes the optimal rotation R by computing the SVD of the covariance matrix of both clouds and then finds the translation t using the centroids $\bar{q}$, $\bar{p}$ of target and source clouds respectively, according to the equation

$$t = \bar{q} - R\bar{p} \tag{4.3}$$

A detailed description of the steps to compute the best-fitting rigid transformation that aligns two sets of corresponding points can be found in this note by O. Sorkine-Hornung and M. Rabinovich [26].

4.5.4 Runtime

We performed several tests to analyze the runtime of each coarse alignment step. Figure 4.13 shows that the most time consuming step is the keypoint detection, which depends linearly on the number of input points.

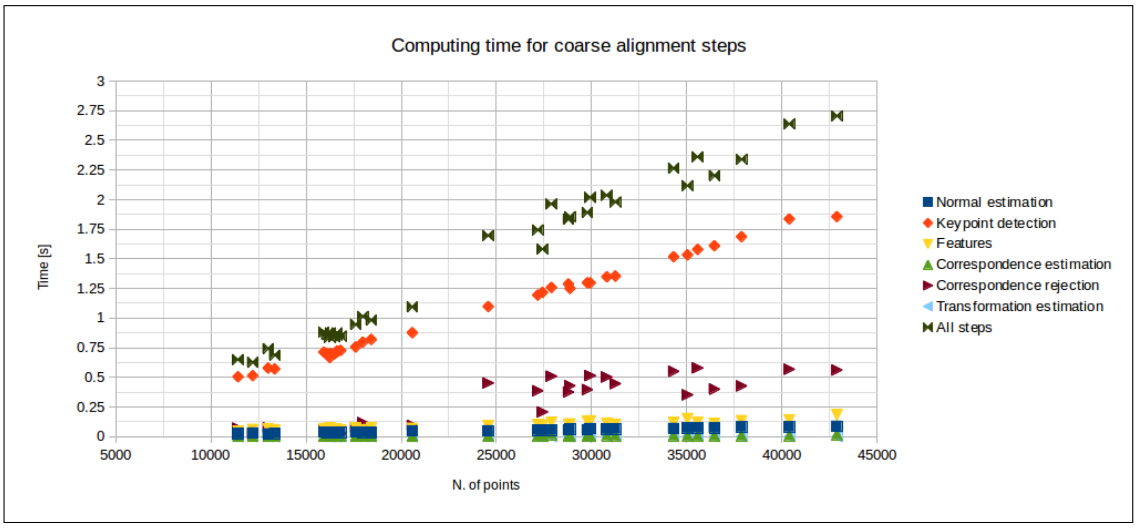


Figure 4.13: Runtime of the coarse alignment steps. Own source.

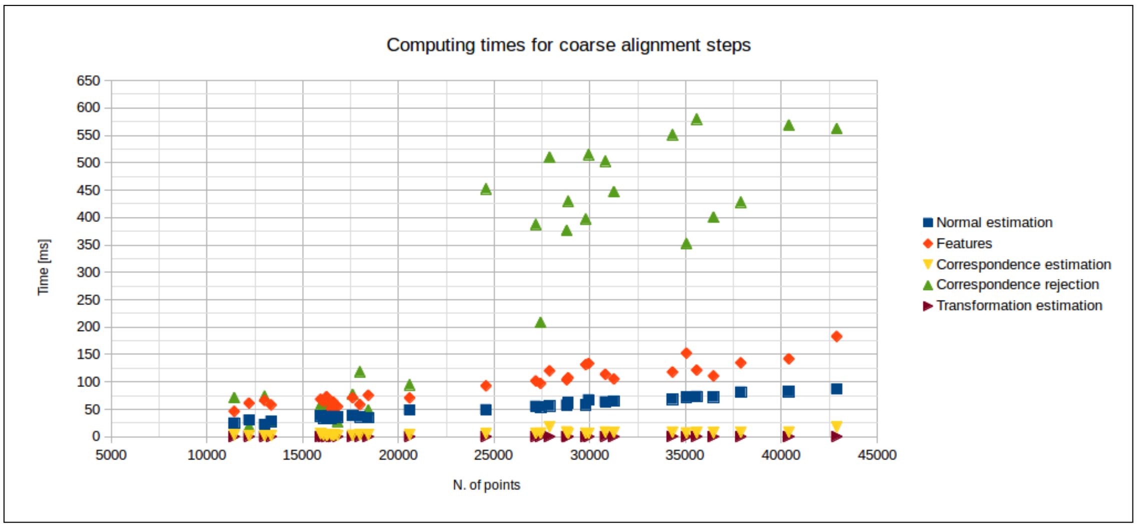


Figure 4.14: Closer look at RANSAC runtime. Own source.

The second step which takes more time is the RANSAC rejection algorithm, although it does not show any correlation with the point number. It seems, however, that there is a significant increase in the computation time for clouds larger than 20000 points. A closer look in Figure 4.14 shows an increase of almost 400 ms when surpassing the 20000 cloud point size.

All in all the total runtime of the coarse alignment step ranges from 0.75 to 2.75 seconds, with keypoint detection being the slowest operation of all the steps.

4.5.5 Results

There is not a fast-forward way to compute the accuracy of a coarse alignment algorithm if the ground truth coordinates of the camera are not known. Such a validation can be performed by manually aligning the clouds and then comparing the algorithm results with the manual transformation. Instead of carrying out this tedious process, a visual validation was performed. The following figures show two typical results when aligning two consecutive point clouds.

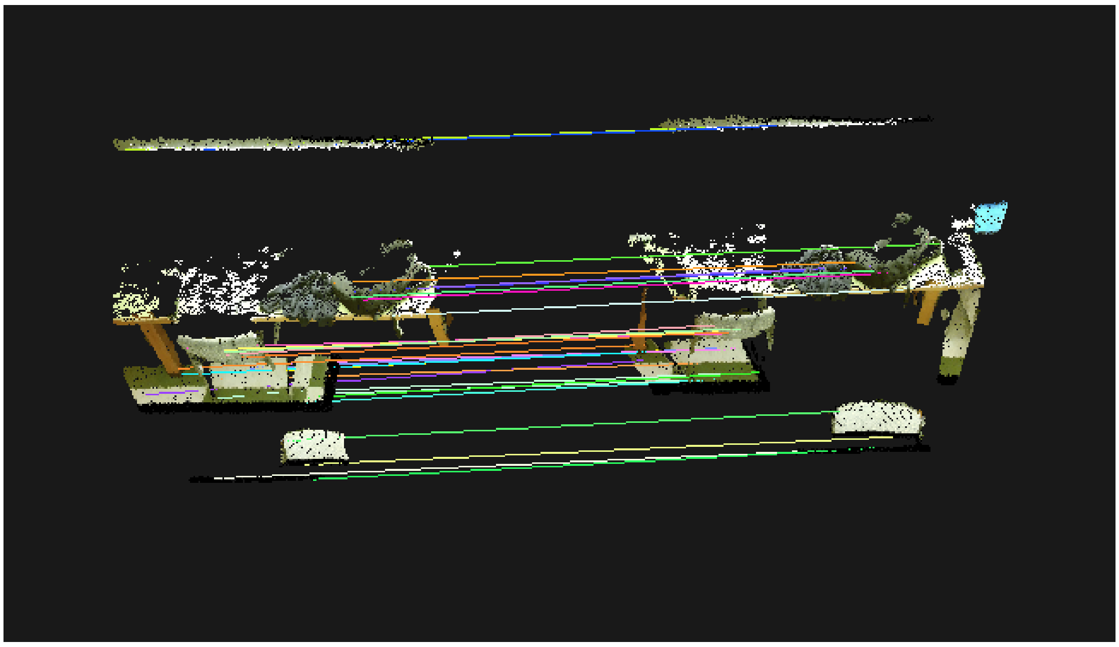


Figure 4.15: Example 1 - correpondences after the RANSAC rejection algorithm. Own source.

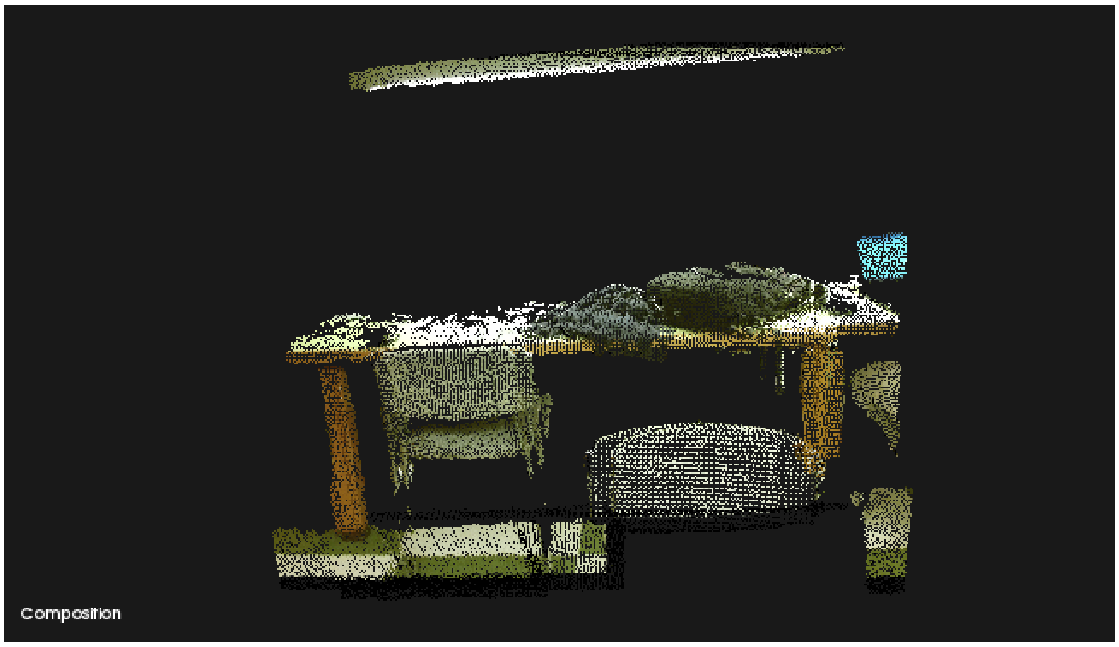


Figure 4.16: Example 1 - final transformation of the coarse alignment process. Own source.

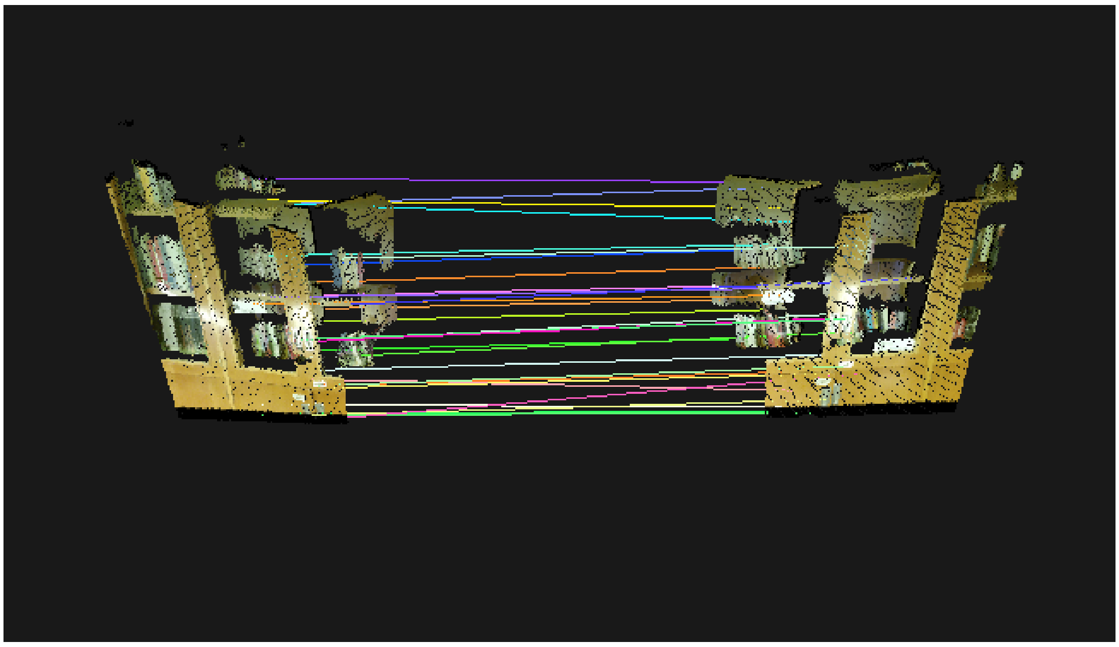


Figure 4.17: Example 2 - correpondences after the RANSAC rejection algorithm. Own source.

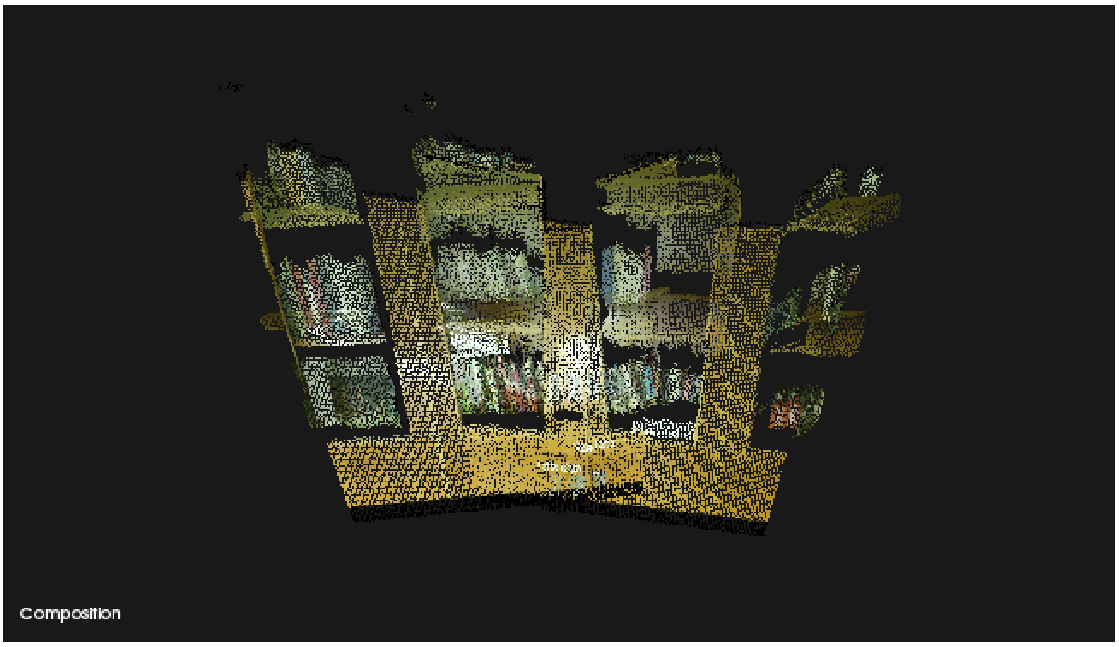


Figure 4.18: Example 2 - final transformation of the coarse alignment process. Own source.

These results show that the coarse alignment final transformation provides an accurate initial guess for the ICP algorithm to converge to the global minimum.

4.6 ICP Fine Alignment

The final step for aligning the clouds is to perform a fine alignment based on the transformation estimated in the coarse alignment step. For this purpose we use PCL's implementation of the point-to-plane ICP algorithm: `IterativeClosestPointWithNormals`. The parameters used for this project are the following:

- `MaxCorrespondenceDistance`: 0.1 meters - threshold distance for rejecting point pairs which are too far from each other
- `MaximumIterations`: 30
- `TransformationEpsilon`: $1e^{-8}$ - if the translation squared difference between two consecutive transformations is lower than this threshold, the algorithm is considered to have converged

With the coarse alignment transformation as an initial guess, ICP has proved to converge in an average of 10 iterations. It is the most time-consuming step of the whole aligning process, ranging from 400 ms to 4 seconds to compute a final fine alignment. The computation time is highly dependent on the quality of the coarse alignment, and even though it sometimes takes up to 4 seconds, average values lie around 1 second. This wide time range is due to the variable quality of the initial alignment which may require more iterations to converge. Moreover, there is a correlation between the number of inputted points and the runtime of one algorithm iteration, as shown in Figure 4.19. It can also be seen that the variability of the runtime for a single iteration increases with the input point number.

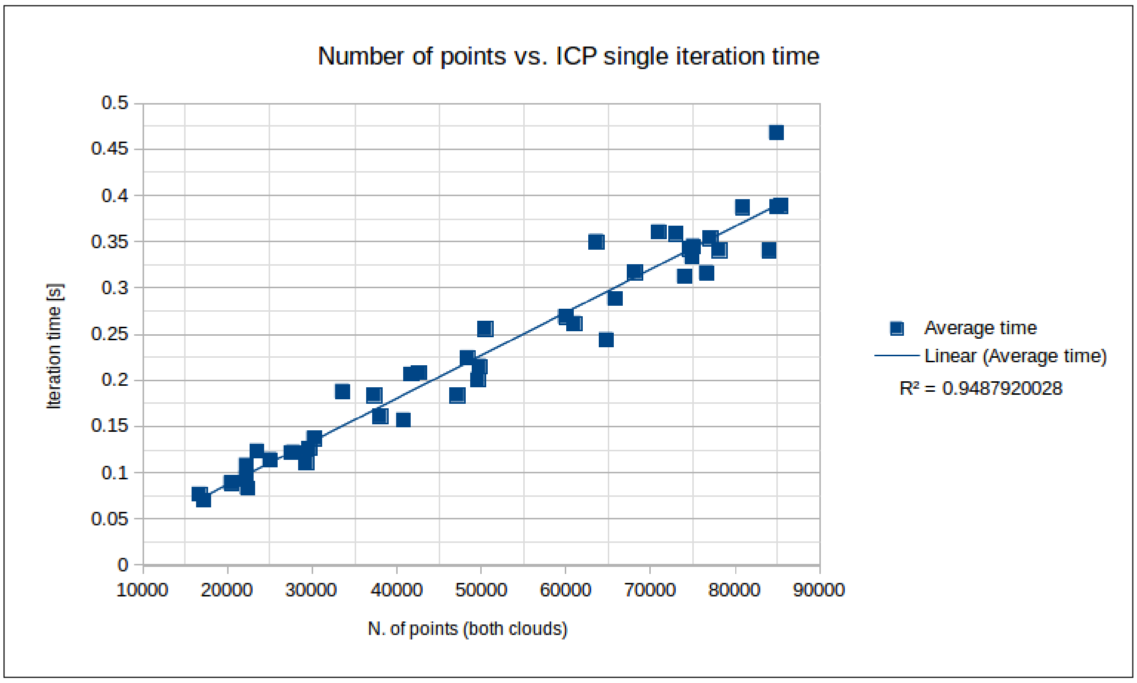


Figure 4.19: ICP single iteration times. Own source.

4.6.1 Results

An example of the results obtained with the point to plane ICP approach can be seen in Figures 4.20 - 4.23. There are some cases where the algorithm still converges to a local minimum, for example when both clouds have large planar areas and no corners or distinct shapes. In this cases the algorithm sometimes makes the two clouds ”slide” along their planar surfaces until the convergence criteria is met. Another case of failure occurs when the initial transformation guess does not properly align the clouds.

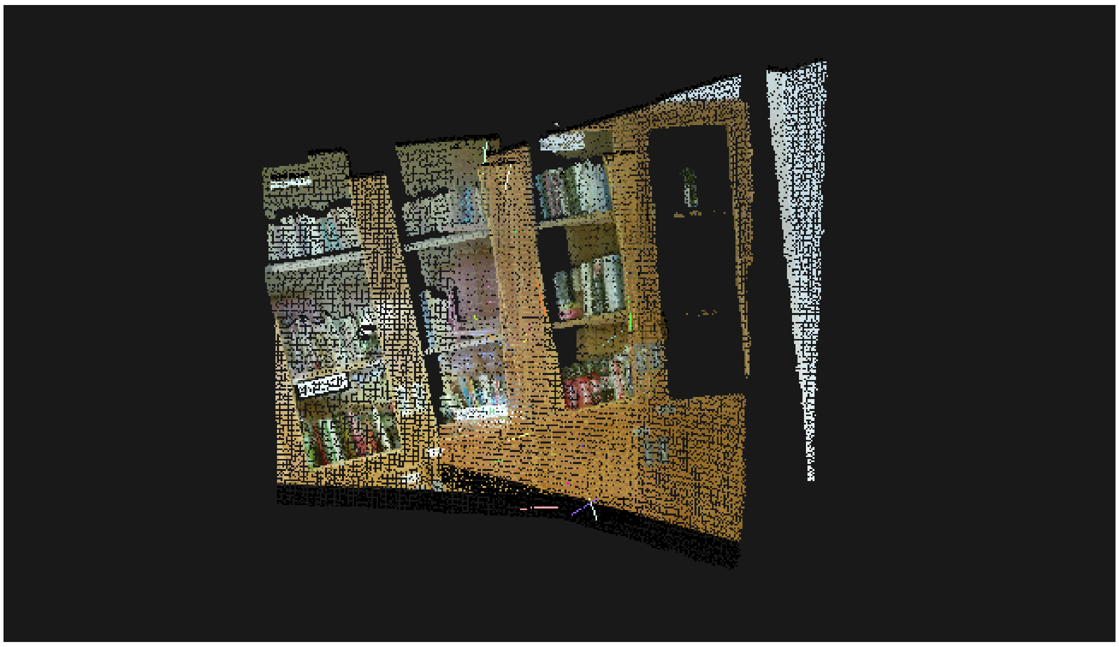


Figure 4.20: Example 1 - initial guess. Own source.

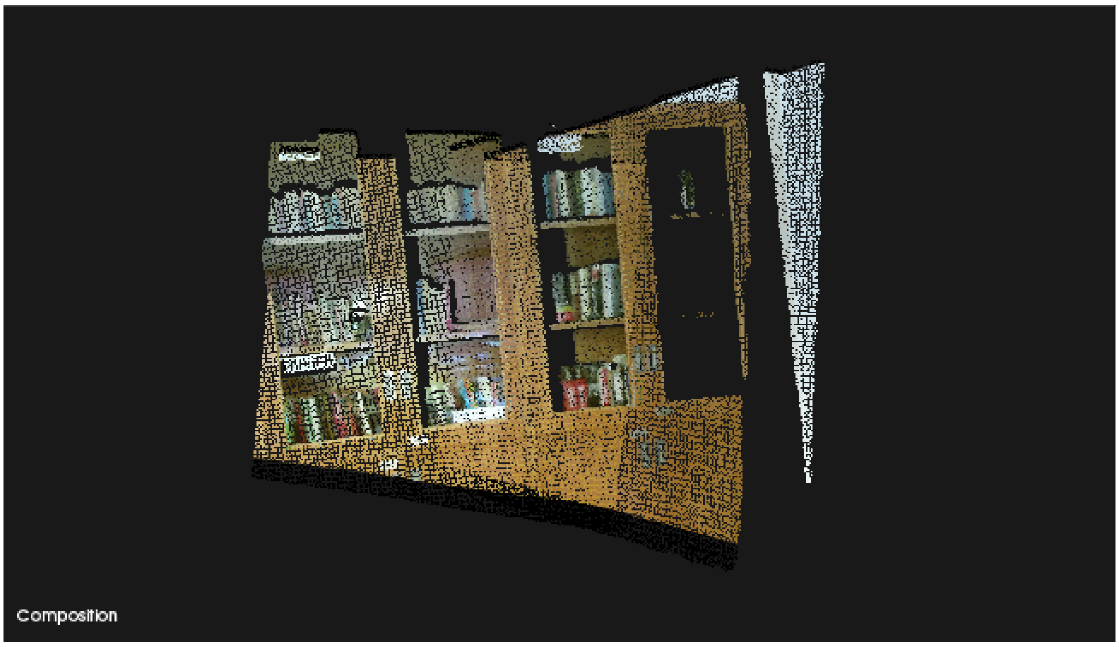


Figure 4.21: Example 1 - after ICP. Own source.

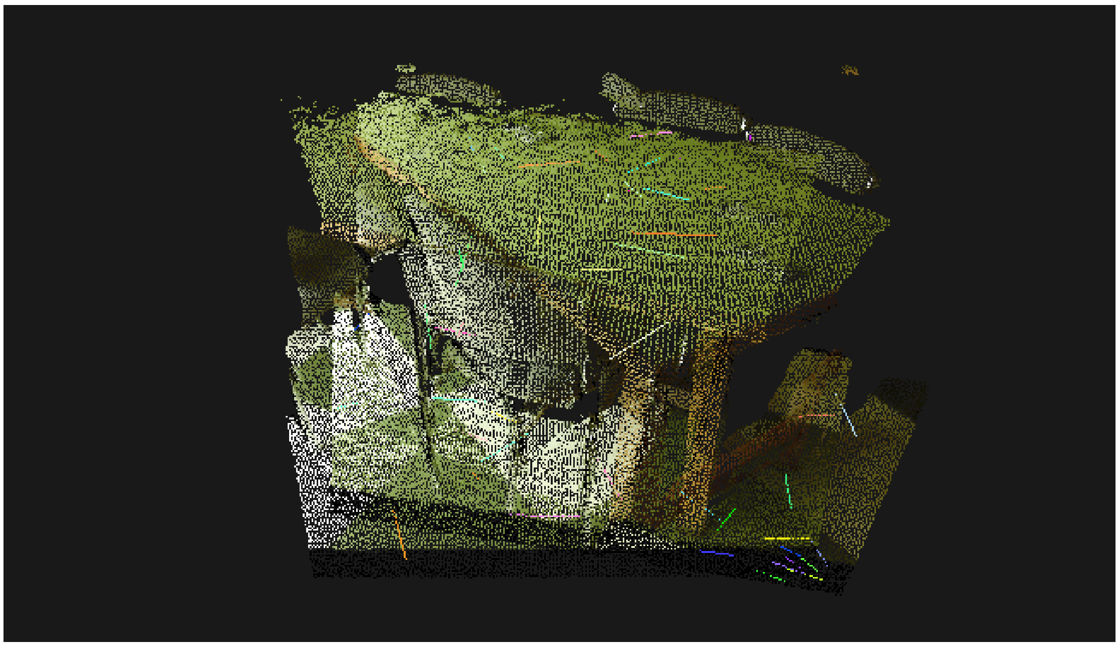


Figure 4.22: Example 2 - initial guess. Own source.

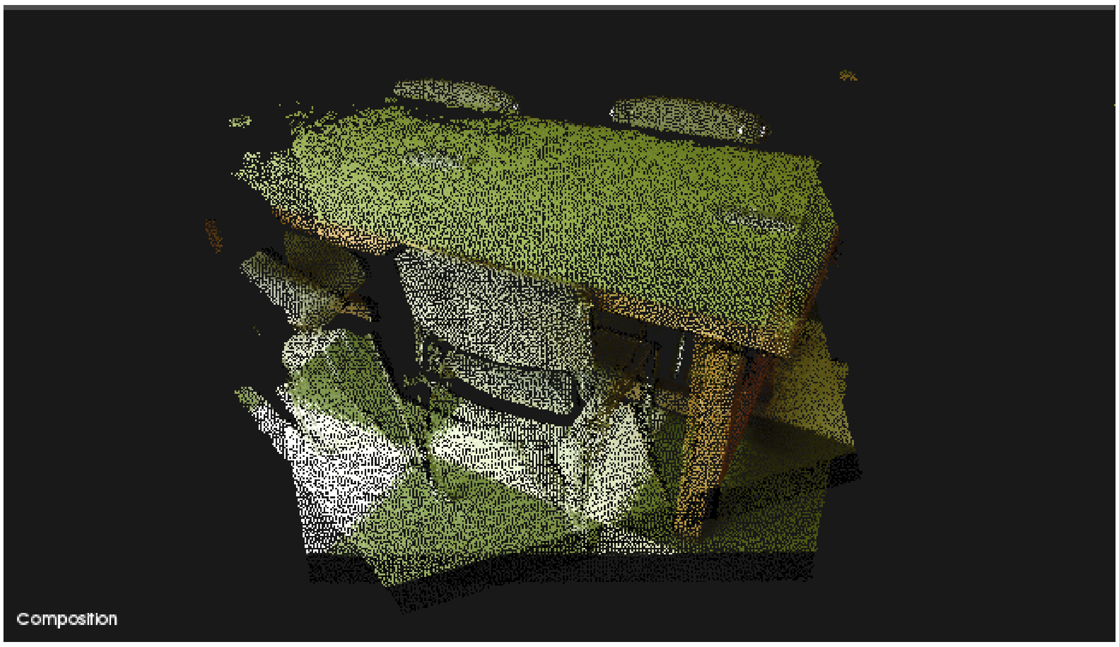


Figure 4.23: Example 2 - after ICP. Own source.

4.7 Surface Reconstruction

At this point the 3D reconstruction is finished. However, there is an additional step which can be easily added to the pipeline and provides a beautiful visualization result as well as serving as a base for further computations: surface reconstruction.

Once the consecutive clouds have been pairwise aligned, we use the Moving Least Squares algorithm to smoothen the surfaces and adjust the points to polynomial surface models of grade 2. Then another Voxel Grid Filter is applied in order unify the point density and finally the 3D scene is passed onto the surface reconstruction algorithm.

In this project we use the `GreedyProjectionTriangulation` which is an algorithm focused on speed[17] rather than accuracy - suitable for the visualization purpose of this project. It searches for neighbors around a point and if certain conditions are met it creates a triangle and assigns a color for the surface. There are many parameters which can be modified to produce different results. In this project we have used the ones recommended by the authors of PCL's implementation, setting the maximum edge length of a triangle to 0.1 meters.

MLS and Voxel Grid filtering are fast algorithms but the Greedy Triangulation is slow[11], so it is only performed when all the clouds are aligned i. e. there are no more frames in the input stream. An example of the reconstruction of a single point cloud is shown in Figure 4.24.

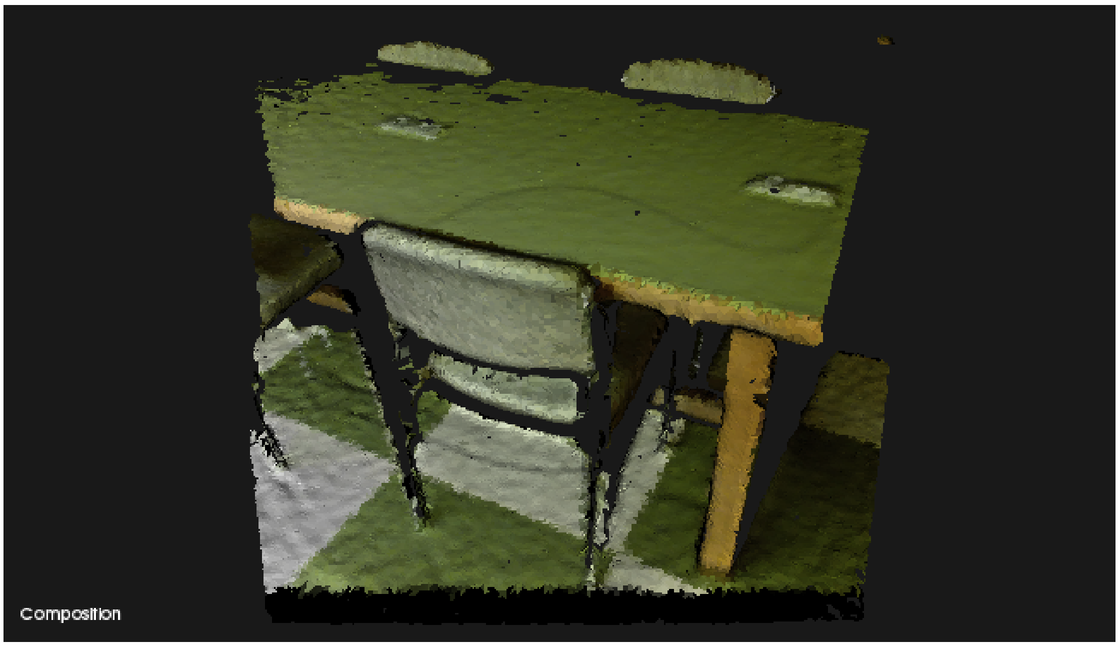


Figure 4.24: Example output of the surface reconstruction of a single cloud. Own source.

[11]Greedy Triangulation is a fast surface reconstruction method but slow when compared to the rest of algorithms of the application

5 Results

All the previous processing steps are managed by the `main` program, which offers different options (loading files from disk, saving captured images, switching between different keypoint detectors, visualizing the ICP iterations, etc.). The main loop captures images from the Kinectv2 (if the load from disk option is not activated) until the Enter key is pressed. It then captures the current image from the Kinectv2 or from a file directory, applies the preprocessing filters and if it is not the first capture it performs a coarse and a fine alignment with the previous image. This process can be repeated indefinitely until the BackSpace key is pressed. At that point the program applies the surface reconstruction algorithm to the resulting cloud of all the registrations.

During this process, the program displays 4 viewports: the Kinectv2 current view, the keypoints of the last two frames, the correspondences of the last two frames and the registration result of all the previous clouds (see Figure 5.1.)

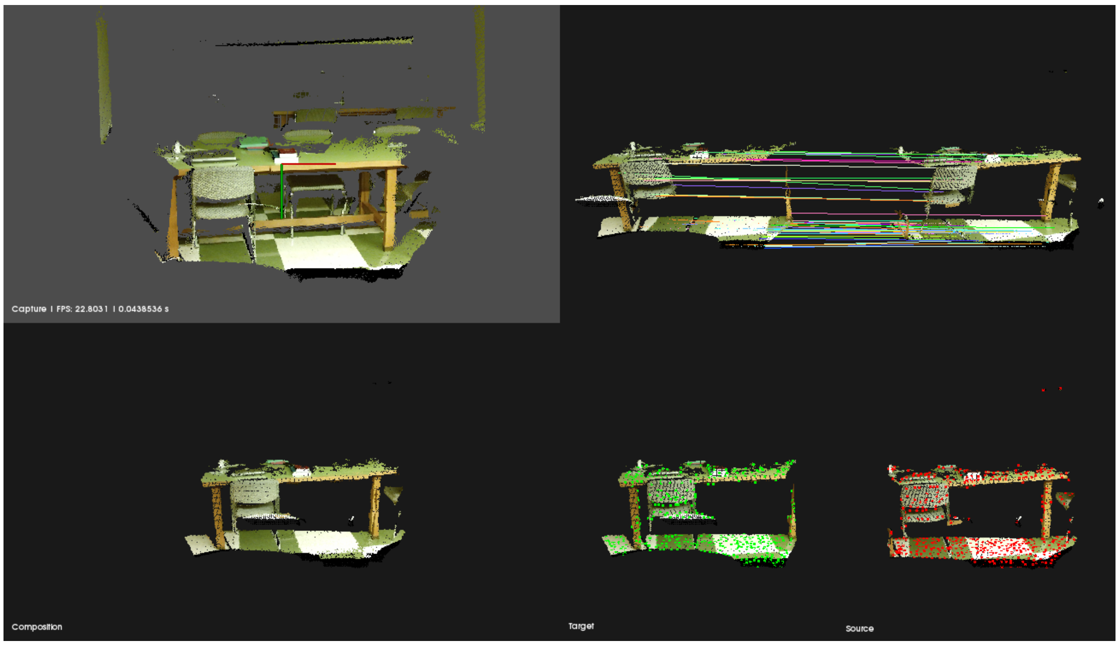


Figure 5.1: Different viewports displayed by the program. Own source.

We have tested the final program in different environments obtaining the following results.

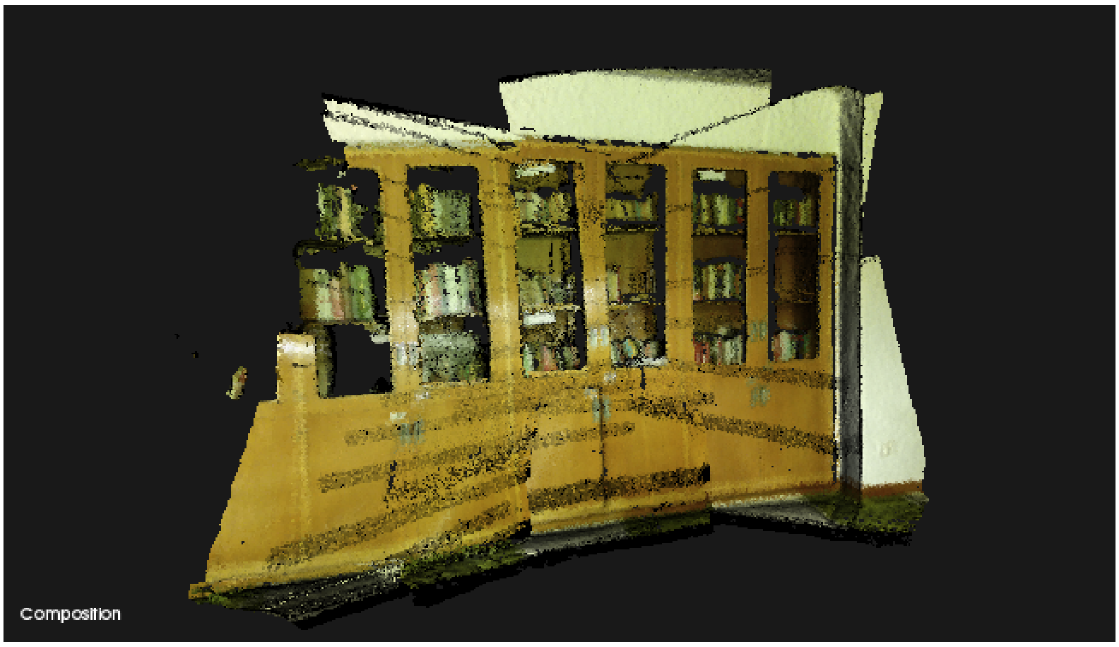


Figure 5.2: Example final result 1: *Library_1* set, 33 images aquired using continuous mode - note the effect of carrying small errors. Own source.

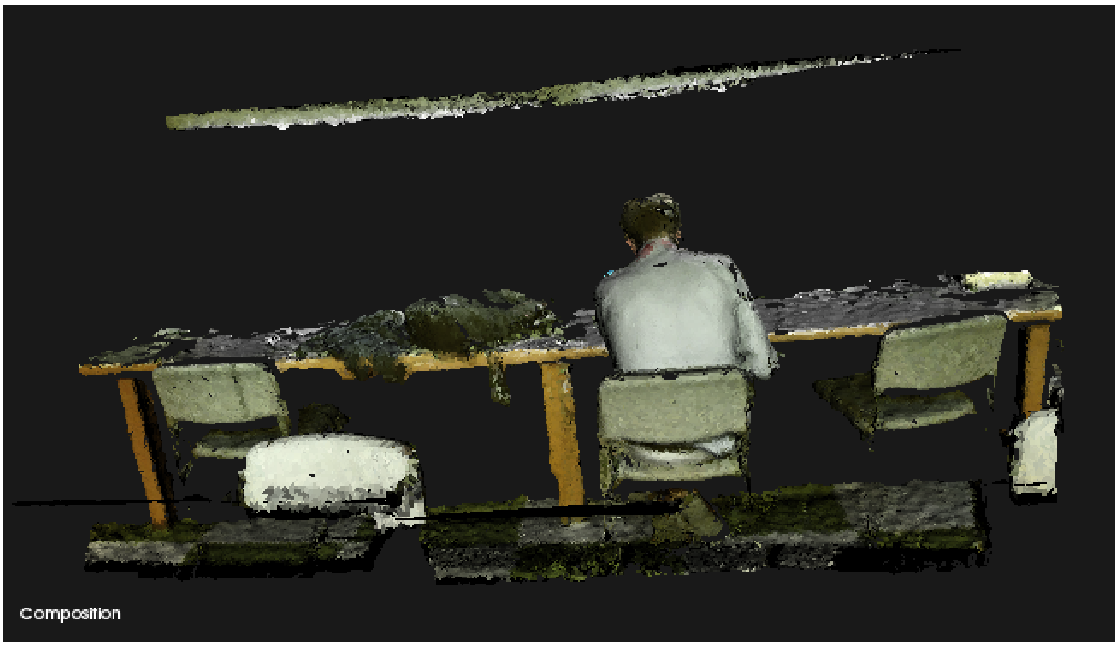


Figure 5.3: Example final result 2: *Long_table* set, 6 images acquired moving the sensor along the x axis. Own source.

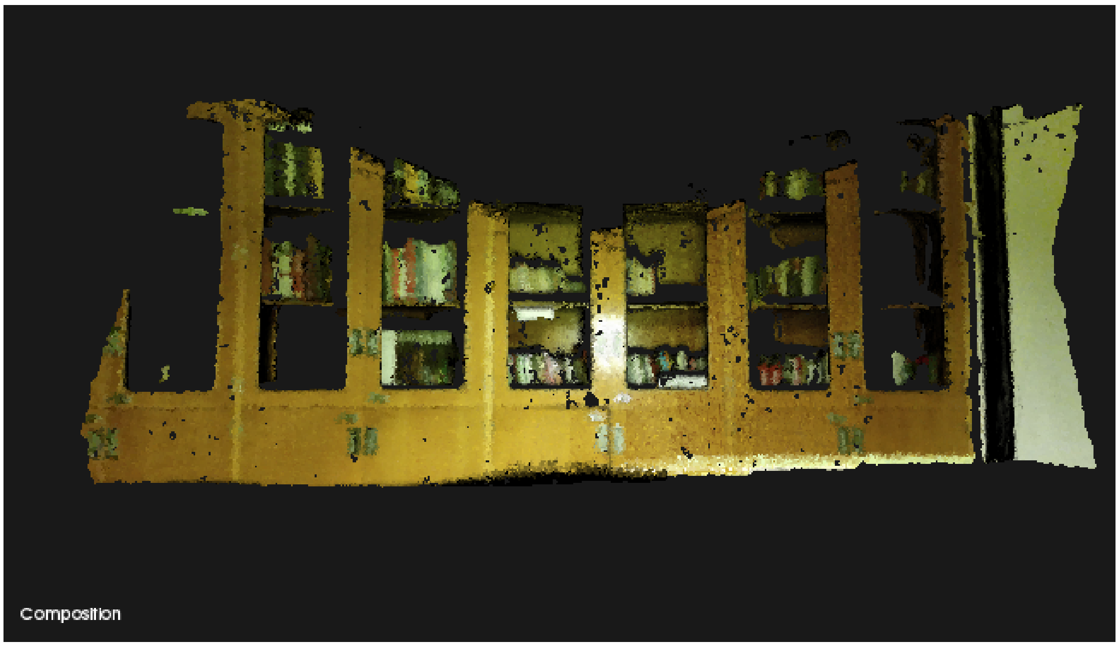


Figure 5.4: Example final result 3: *Library_2* set, 9 images obtained by rotating the Kinectv2 around the z axis. Own source.

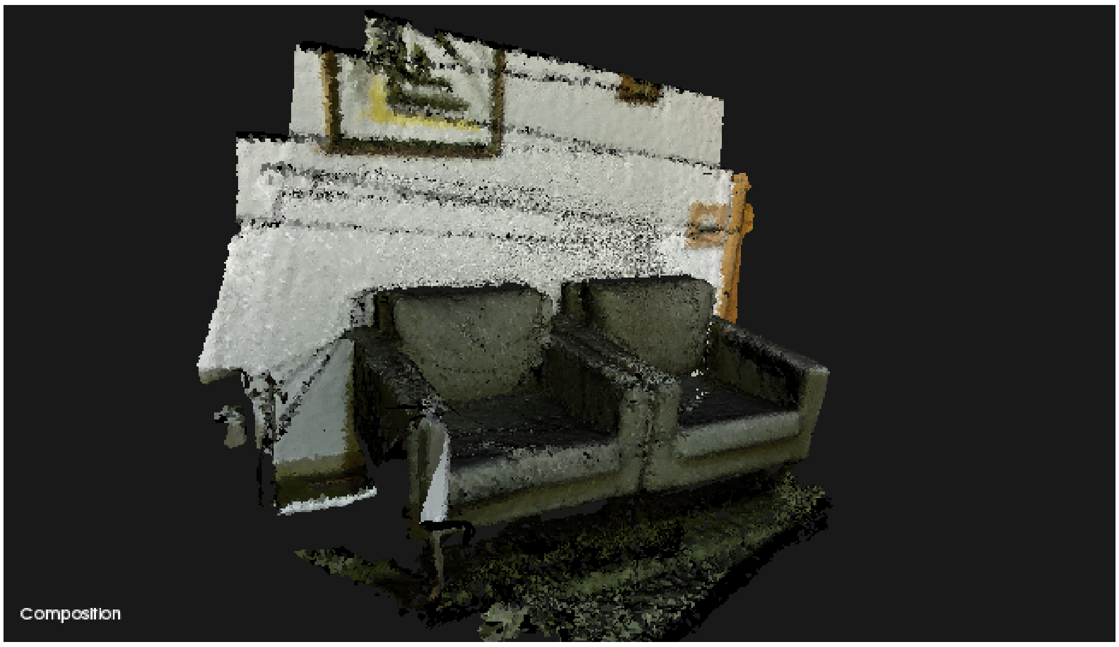
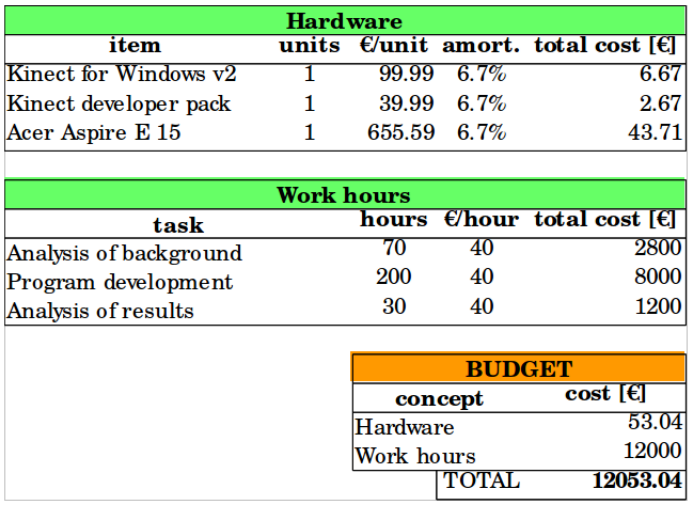


Figure 5.5: Example final result 4: *Sofas* set, 9 images - note the noise of the reflecting ground material. Own source.

6 Project Budget

The project budget has been estimated diving the costs into hardware and work hours. The cost per hour of work has been set to 40, while the hardware cost only includes its amortization because both the Kinectv2 and the laptop PC were acquired before the beginning of the project for their use in several applications. We have considered a life cycle of 5 years for both products and 4 months as the time they have been used for the project, so only 6.7% of their initial cost will be included in the costs of the project. As for the software used, all of it is distributed as free software: Ubuntu, LaTeX, LibreOffice, CMake, Sublime 3, libfreenect2 and PCL. Table 6.1 shows the calculation of the project budget.



Hardware				
item	**units**	**€/unit**	**amort.**	**total cost [€]**
Kinect for Windows v2	1	99.99	6.7%	6.67
Kinect developer pack	1	39.99	6.7%	2.67
Acer Aspire E 15	1	655.59	6.7%	43.71

Work hours			
task	**hours**	**€/hour**	**total cost [€]**
Analysis of background	70	40	2800
Program development	200	40	8000
Analysis of results	30	40	1200

BUDGET	
concept	**cost [€]**
Hardware	53.04
Work hours	12000
TOTAL	**12053.04**

Table 6.1: Calculation of the project budget

The final budget is 12053 euros, 99.56% of which is spent in work hours and the rest 0.44% on hardware.

7 Environmental Impact

The environmental impact of this project can be analyzed from two different viewpoints: the effects derived from using the Kinectv2 and the impact that the project might have in future applications.

Regarding the first point, the Kinect for Windows v2 has been designed for human use and its IR emitters are as harmful as a conventional IR LED can be. The light they emit does not harm the environment nor the people in front of it. In this sense we can assure that this project does not have a harmful impact on the environment.

Nonetheless, the Kinectv2 will eventually fail, break or become unusable. In order to reduce the environmental impact, at the end of its life cycle it must be recycled properly. For a proper recycling of the Kinectv2 sensor it must be taken to an electronics waste management center, where the contaminant materials of the sensor will be properly taken care of while the recyclable materials such as plastics and metals will be treated for their reuse.

As for the impact of future applications which make use of this project, we must analyze the range of fields were the project can be applied. For example, if an autonomous car implements the 3D reconstruction program described in this memory, it might help the car system find the shortest possible route to its destiny and save energy for the transport. This example shows how the future applications of this project can have a beneficial impact on the environment.

8 Conclusions

In the context of 3D scene reconstruction and point cloud registration many solutions have already been proposed, even final real time applications are available such as Kinect Fusion[1], a real time 3D object scanning and model creation tool developed by Microsoft. The scientific community has also produced a wide range of applications for this purpose and even PCL has its own tutorial on image registration.

However, it is not easy to find a full implementation of the whole registration process. All the necessary steps are scattered as separate pieces of code which are often aimed at different devices and produce results specifically suited to certain applications. Moreover, a great deal of these closed solutions make use of a long list of libraries for each registration step. This project provides an easy implementation of a complete registration application: from the connection to the Kinectv2 device to the final point cloud surface reconstruction, using only the open source libraries PCL and libfreenect2. It can serve as a basic tool for testing new algorithms, as a functional starting point for more complex registration applications, or as a 3D scene reconstructor which provides 3D models of scenes for their further analysis.

The problems that arouse during the project were mainly related to coding. Even though many researchers use PCL in their projects and the library was released almost 7 years ago, most of its modules are poorly documented. This is due to an assumption that the users of PCL have previous knowledge of the algorithms it offers. However, for a user that begins to study the 3D vision field it can be very tedious to find specific documentation about the algorithms. Many hours were spent trying to fix coding problems which could have been solved much faster if the library had a broader documentation.

Apart from the coding issues, there are still some problems which have not been solved. For example, in scenes where the predominant figures are large planar areas, the algorithm is prone to failing at aligning two images if there is few overlap or a large rotation between them. This type of frames also take a much longer time to align. These problems could be addressed by using the RGB data to help align the clouds.

Despite the issues described above, final results show that the main objective of the project has been fulfilled: we have written a C++ program that reconstructs 3D scenes from a stream from Kinect v2 RGBD images. Furthermore, the program is well structured and documented both in code comments and in the present memory for its future use in other applications. There is one objective which has yet to be addressed: suggest improvements as future work.

[1]More information on Kinect Fusion available here.

8.1 Future Work

The two main areas which could be improved are the robustness and speed of the program. These improvements could be addressed in the following points:

- The keypoint detection time cloud be reduced by using RGB detection algorithms instead of 3D based detectors. This is a common approach used in real time applications, but currently there is not an implementation of such algorithms in the PCL library. In order to keep the program simple, the possibility of porting RGB keypoint detection algorithms into the PCL code trunk should be considered.
- The overall speed of the coarse alignment step could be enhanced by using RGB data for feature description and correspondence matching and rejection. The final coarse transformation estimation would then be computed applying the found correspondences to the 3D point clouds.
- The runtime of the ICP iterations could be improved by replacing the current approach for a GPU implementation of the algorithm.
- To improve the robustness of the algorithm, an in-depth study of the best keypoint detectors and feature descriptors for Kinectv2 data could be performed. Moreover, if the overall speed of the process is improved, more complex and accurate algorithms could be used in the coarse alignment pipeline.
- The surface reconstruction step has been included in the current project only as a way of providing a better visual representation of the registration result. There are many algorithms which cloud be considered for improving this process, such as Marching Cubes or Alpha Shapes.
- The PassThrough Filter limits the range of the Kinectv2 to 2 meters due to the planar distortion of the farther points. An extrinsic calibration of the camera could be used to correct the distortion and enhance the range to 4 meters.
- For minimizing the carrying error due to the pairwise registration approach, a loop closure detection algorithm could be implemented.

Bibliography

[1] Intel® RealSense™ Technology. https://www.intel.com/content/www/us/en/architecture-and-technology/realsense-overview.html. Accessed: 05-06-2017.

[2] Kinect hardware. https://developer.microsoft.com/en-us/windows/kinect/hardware. Accessed: 05-06-2017.

[3] Xtion 2. https://www.asus.com/3D-Sensor/Xtion-2/. Accessed: 05-06-2017.

[4] Maged Aboali, Nurulfajar Abd Manap, Darsono Abd Majid, and Zulkalnain Mohd Yusof. Review on Three-Dimensional (3-D) Acquisition and Range Imaging Techniques. *International Journal of Applied Engineering Research*, 12(10):2409–2421, 2017.

[5] ASUS. Asus Xtion 2. https://www.asus.com/media/global/products/W0aYrhbhOBInh4gD/P_setting_fff_1_90_end_500.png. Accessed: 05-06-2017. [Image].

[6] Gioia Ballin. Detectors evaluation: repeatability and time performances, 2012. Available at http://www.pointclouds.org/blog/gsoc12/gballin/tests.php.

[7] Gioia Ballin. How to use the ISS 3D keypoint detector, 2012. Available at http://www.pointclouds.org/blog/gsoc12/gballin/iss.php.

[8] Ben Bellekens, Vincent Spruyt, Rafael Berkvens, and Maarten Weyn. A Survey of Rigid 3D Pointcloud Registration Algorithms. *The Fourth International Conference on Ambient Computing, Applications, Services and Technologies*, 2014.

[9] Paul J. Besl and Neil D. McKay. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–254, February 1992.

[10] Lisa Brown Gottesfel. A Survey of Image Registration Techniques. *ACM Computing Surveys*, 24:326–376, January 1992.

[11] Microsoft Corporation. Microsoft Kinect. https://news.microsoft.com/wp-content/uploads/2014/04/KinectforWindowsv2_03_Web.png. Accessed: 05-06-2017. [Image].

[12] PCL Documentation. Point Feature Histograms (PFH) descriptors. http://pointclouds.org/documentation/tutorials/pfh_estimation.php. Accessed: 10-07-2017. [Image].

[13] PCL Documentation. The PCL Registration API. http://pointclouds.org/documentation/tutorials/registration_api.php. Accessed: 20-06-2017. [Image].

[14] Silvio Filipe and Luís A. Alexandre. A comparative evaluation of 3D keypoint detectors in a RGB-D object dataset. In *Computer Vision Theory and Applications (VISAPP), 2014 International Conference on*, volume 1, pages 476–483. IEEE, 2014.

[15] Intel. Intel RealSense Developer Kit (SR300). `https://click.intel.com/intelrealsense-developer-kit-featuring-sr300.html`. Accessed: 05-06-2017. [Image].

[16] David G. Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157, 1999.

[17] Zoltan Csaba Marton, Radu Bogdan Rusu, and Michael Beetz. On Fast Surface Reconstruction Methods for Large and Noisy Datasets. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Kobe, Japan, May 2009.

[18] Matteo Munaro. The effect of the Voxel Grid filter on Kinect 3D data. `https://www.researchgate.net/profile/Matteo_Munaro/publication/286624443/figure/fig3/AS:306520454647810@1450091579067/Fig-4-The-effect-of-the-voxel-grid-filter-on-Kinect-3D-data.png`, 2013. Accessed: 15-07-2017. [Image].

[19] Dinesh Nair. Simplified stereovision system. `http://www.aerodefensetech.com/component/content/article/14925?start=1`, 2012. Accessed: 02-06-2017. [Image].

[20] Carlo Nicolini. A C++ code to compute OpenGL 4x4 GL_MODELVIEW_MATRIX from 2D-3D points homography. `https://braintrekking.wordpress.com/2013/06/02/a-c-code-to-compute-opengl-4x4-gl_modelview_matrix-from-2d-3d-points-homography/`, 2013. Accessed: 23-07-2017. [Image].

[21] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (FPFH) for 3D registration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3212–3217. IEEE, 2009.

[22] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *Robotics and automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[23] Hamed Sarbolandi, Damien Lefloch, and Andreas Kolb. Kinect Range Sensing: Structured-Light versus Time-of-Flight Kinect. *Computer Vision and Image Understanding*, 139:1–20, 2015.

[24] Hamed Sarbolandi, Damien Lefloch, and Andreas Kolb. Principle of structured light based systems. Reproduced as Figure 1 in [23], 2015. [Image].

[25] Roland Smeenk. Kinect V1 and Kinect V2 fields of view compared. `http://smeenk.com/kinect-field-of-view-comparison/`. Accessed: 22-06-2017.

[26] Olga Sorkine-Hornung and Michael Rabinovich. Least-Squares Rigid Motion using SVD. Departament of Computer Science, ETH Zurich, January 2016.

[27] Bastian Steder, Radu Bogdan Rusu, Kurt Konolige, and Wolfram Burgard. NARF: 3D range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, volume 44, 2010.

[28] Agnes Anna Swadzba. Estimation of Camera Motion from Depth Image Sequences. Master's thesis, University Erlangen-Nuremberg, 2006.

[29] Agnes Anna Swadzba. Principle operation mode of a time-of-flight camera. Reproduced as Figure 2.1 in [28], 2006. [Image].

[30] Taylor Wang. Iterative Closest Point algorithm-point cloud/mesh registration. `https://taylorwang.wordpress.com/2012/04/06/iterative-closest-point-algorithm-point-cloudmesh-registration/`, 2012. Accessed: 17-06-2017. [Image].

[31] Yu Zhong. Intrinsic shape signatures: A shape descriptor for 3D object recognition. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 689–696. IEEE, 2009.

[32] Barbara Zitová and Jan Flusser. Image registration methods: a survey. *Image and Vision Computing*, 21(11):977–1000, June 2003.