2016•2017
FACULTEIT INDUSTRIËLE INGENIEURSWETENSCHAPPEN
master in de industriële wetenschappen: energie

Masterproef

3D Indoor scanning & map building

Promotor :
dr. ir. Johan BAETEN

Promotor :
dr. HAI HOANG

Sander Vanspauwen

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: energie

Gezamenlijke opleiding Universiteit Hasselt en KU Leuven

2016•2017

Faculteit Industriële ingenieurswetenschappen

master in de industriële wetenschappen: energie

Masterproef

3D Indoor scanning & map building

Promotor :
dr. ir. Johan BAETEN

Promotor :
dr. HAI HOANG

Sander Vanspauwen

Scriptie ingediend tot het behalen van de graad van master in de industriële wetenschappen: energie

universiteit hasselt | KU LEUVEN

Preface

My project discusses 3D mapping by making use of a RGB-D camera. The project took from 16th February 2017 till the 23th June 2017. I did this project at the university Hanoi university of science and technology (HUST) in Hanoi, Vietnam.

With this project, I finish my Master's degree in energy industrial engineering technology: automation at the university UHasselt / KU Leuven at Diepenbeek Belgium.

First, I would like to thank my two supervisors at the university in HUST Dr. Ir. Hoang. Hai and Dr. Ir. Thanh Hung and prof. dr. Ir. J. Baeten as my supervisor from the UHasselt / KU Leuven for supporting me during this project and guiding me in the right direction.

I would also like to thank Prof. Nguyen Trong Doanh Zwang for allowing me to do my masters project at his department, the mechatronic department at HUST.

The program involves computer programming and modern technology, these are also my personal main interests. During this project, I had the opportunity to widen my experience in image and point cloud processing and widen my knowledge in C++. Given that I had no experience in image and point cloud processing and C++. I spent time in gathering information before I could start programming an algorithm. Since I had less than four months to complete this project I am satisfied with the results. But I know I would do better when more time was given. Besides the project took place in two university's communication was very important during this project.

As an exchange student, I had the opportune the widen my cultural experience with different cultures in Vietnam. I was also able to travel around the country and discover the beautiful nature of Vietnam. During this trip, I had the chance to improve my English and learning to live on my own. I also met great people during this exchange program and I am grateful I took this opportunity. This experience would never be possible without the opportunity I had from IR. G. Raymaekers and IR. W. Claes and the ASEM-DUO Programme in Seoul, KOREA.

June 2017

Sander Vanspauwen

Table of contents

Preface 3
List of Figures 9
List of Tables 11
Abstract 13
Abstract in Dutch 15
1 Introduction 17
1.1 Context 17
1.2 Problem statement 17
1.3 Goals / requirements 18
1.4 Method 19
1.5 Structure of the book 20
2 Point cloud registration 21
2.1 Finding the transformation matrix 22
2.2 ICP 22
2.3 Registration pipeline 24
3 Point cloud 25
3.1 Organized Point clouds 25
3.2 Unorganized point clouds 25
3.3 Transform a point cloud 26
3.4 KD-Tree Storing a point cloud 28
3.4.1 Adding a new point to the tree 30
3.4.2 Removing a point from the tree 31
3.4.3 Comparing with organized point clouds 31
4 Filters 33
4.1 Pass Through Filter 33
4.2 Voxel grid filter 34
4.3 Statistical outlier removal filter 34
4.4 Radius outlier removal filter 35
4.5 Bilateral filter 36
4.6 Conditional removal filter 36
5 Key points detection & subsets 37

5.1 NARF Points (Normal aligned radial feature) 37
5.2 Uniform sampling 38
6 Point cloud features descriptor 39
6.1 Surface normal in a point cloud 40
6.2 Defining a point cloud feature (PFH) 41
7 Registration techniques 45
7.1 Matching (key) points 45
7.2 Rejection correspondences 46
7.2.1 Correspondences rejection based on distance 46
7.2.2 Correspondences rejection based on duplicate matches 46
7.2.3 Correspondences rejection based on surface normal 47
7.2.4 RANSAC-based rejection 48
7.3 Transformation estimation 49
7.3.1 Point to point 49
7.3.2 Point to plane 49
7.4 Transformation criteria 50
7.4.1 Maximum number of iterations 50
7.4.2 Absolute transformation threshold 50
7.4.3 Relative transformation threshold 50
7.4.4 Maximum number of similar transformations 50
7.4.5 Relative mean square error 50
7.4.6 Absolute mean square error 50
7.4.7 Transformation validation 50
8 Practical implementation 51
8.1 PCL 51
8.2 RGB-D camera 51
8.3 CUDA 53
9 Practical Pipeline 55
9.1 Init camera 55
9.2 Get cloud 55
9.3 Analyze cloud 56
9.4 Registration 56

9.5 Check registration 57
9.6 Inverse transformation 57
9.7 Transform target cloud 57
9.8 Update camera pose 57
9.9 Source = target 58
9.10 Add to map 58
9.10.1 Voxel grid filter 58
9.10.2 KD-Tree adding 58
10 Results and speed 61
10.1 Dataset 1 63
10.2 Dataset 2 64
10.3 Dataset 3 64
11 Conclusion 65
11.1 Cycles time 65
11.2 Accuracy 66
11.3 Shift register 67
11.4 Decrease degrees of freedom 68
11.5 External sensors 68
12 References 69
13 Attachments 71
Attachment A: A* algorithm 72
A.1: Pseudo code 73
A.2: Example 74
Attachment B: Hardware 75
Attachment C: Dataset 1 76
Attachment D: Dataset 2 80
Attachment E: Dataset 3 84
E.1: Test 1 85
E.2: Test 2 87
E.3: Test 3: 89
E.4: Test 4 91
E.5: Test 5 93

E.6: Test 6 95
E.7: Test 7 97
Attachment F: Dataset 4 99
Attachment G: Workshop test 101

List of Figures

Figure 1:1 HUST logo (Hanoi University of Science and Technology, 1985) 17
Figure 1:2 3D map example (Fleming, 2011) 18
Figure 2:1 Point cloud registration example (Pascal Willy Theiler, 2012) 21
Figure 2:2 Standard registration pipeline 24
Figure 3:1 organized point cloud 25
Figure 3:2 Transforming a point cloud (Rusu, 2011) 27
Figure 3:3 Kd-tree graphical interpretation (K-d tree, 2017) 29
Figure 3:4 Kd-tree graphical interpretation (K-d tree, 2017) 29
Figure 3:5 Kd-tree adding a new point (K-d tree, 2017) 30
Figure 3:6 Kd-tree adding a new point (K-d tree, 2017) 31
Figure 4:1 point cloud (Libpointmatcher, 2006) 34
Figure 4:2 voxel filtered point cloud (Libpointmatcher, 2006) 34
Figure 4:3 6.1.3 Statistical outlier removal filter (Rusu, 2011) 35
Figure 4:4 6.1.4 Radius outlier removal filter (Rusu, 2011) 35
Figure 4:5 Bilateral filter (Farzana, 2011) 36
Figure 5:1 Key points example (Bashan, 2015) 37
Figure 6:1 Feature matching (Rusu, 2011) 39
Figure 6:2 Plane normal (Rusu, 2011) 40
Figure 6:3 Vector normal (Rusu, 2011) 40
Figure 6:4 PFH K-neighbours (Rusu, 2011) 41
Figure 6:5 PFH UVW coordinate frame (Rusu, 2011) 42
Figure 6:6 FPFH (Rusu, 2011) 43
Figure 7:1 Matching corresponds example (Rusu, 2011) 45
Figure 7:2 Correspondences rejection based on distance (Dirk Holz, 2015) 46
Figure 7:3 Correspondences rejection based on duplicate matches (Dirk Holz, 2015) 47
Figure 7:4 Correspondences rejection based on surface normal (Dirk Holz, 2015) 47
Figure 7:5 RANSAC-based rejection (Joe Ahuja, n.d.) 48
Figure 8:1 RGB-D camera sensors Construct the point cloud (Kinect for Windows Sensor Components and Specifications, n.d.) 51
Figure 8:2 Image plane 52
Figure 8:3 Image plane triangulation 52
Figure 9:1 Practical registration pipeline 55
Figure 9:2 Analyse cloud flowchart 56
Figure 9:3 Registration flowchart 57
Figure 9:4 Update camera pose flowchart 57
Figure 9:5 Map building voxelgrid filter flowchart 58
Figure 10:1 RGB image Kitchen scene 61
Figure 10:2 Point cloud kitchen scene 62
Figure 10:3 Color point cloud kitchen scene 62
Figure 10:4 Object kitchen scene 64
Figure 11:1 Multithreading flowchart 65
Figure 11:2 Feature analysing 66
Figure 11:3 Improved accuracy algorithm 1 66
Figure 11:4 Improved accuracy algorithm 2 67
Figure 11:5 Fixed Y position scan (Feifer, 2012) 68
Figure 13:1 Node map example 72
Figure 13:2 A* example (Lague, 2014) 74
Figure 13:3 Analyze functions 78
Figure 13:4 Other functions 78

Figure 13:5 ICP **79**
Figure 13:6 Correspondences **79**
Figure 13:7 Analyze functions **82**
Figure 13:8 Other functions **82**
Figure 13:9 ICP **83**
Figure 13:10 Correspondences **83**
Figure 13:11 ICP **84**
Figure 13:12 Front view **85**
Figure 13:13 Top view **85**
Figure 13:14 Object view **86**
Figure 13:15 Front view **87**
Figure 13:16 Top view **87**
Figure 13:17 Object view **88**
Figure 13:18 Front view **89**
Figure 13:19 Top view **89**
Figure 13:20 Object view **90**
Figure 13:21 Front view **91**
Figure 13:22 Top view **91**
Figure 13:23 Object view **92**
Figure 13:24 Front view **93**
Figure 13:25 Top view **93**
Figure 13:26 Object view **94**
Figure 13:27 Front view **95**
Figure 13:28 Top view **95**
Figure 13:29 Object view **96**
Figure 13:30 Front view **97**
Figure 13:31 Top view **97**
Figure 13:32 Object view **98**
Figure 13:33 Front view **99**
Figure 13:34 Top view **99**
Figure 13:35 Object view **100**
Figure 13:36 Workshop RGB **101**
Figure 13:37 workshop point cloud **101**
Figure 13:38 workshop point cloud **102**
Figure 13:39 workshop point cloud top view **102**

List of Tables

Table 1 Kd-tree example first split **28**
Table 2 Kd-tree example result **29**
Table 3 Functions overview **76**
Table 4 Functions overview **80**
Table 5 ICP **84**

Abstract

This research project aims to develop a 3D indoor scanning and map building application (algorithm) at the Hanoi University of Science and Technology (HUST). 3D technology is often applied in: transport, navigation, robots, 3D object scanning and more. This project focuses on gathering more information from the environment with a real-time cycle time for indoor environments. The goal is to create a 3D map on a computer, with all the obstacles that are in the environment.

A major part of this master's thesis was the programming of an algorithm that makes use of PCL (Point cloud library) software and a RGD-D hardware. The algorithm creates a point cloud with the depth image received from the RGB-D camera. This point cloud correlates to everything that is in the camera's field of view. Every point cloud is taken from a specific view-point in the real world. The algorithm combines these point clouds of the scene to one point cloud (registration).

The algorithm can combine two point clouds in ±450 ms and needs ±700 ms to update the 3D map. The scene is represented well but still contains small errors, especially relating to the alignment of the small objects in the scene.

Abstract in Dutch

Dit onderzoek richt zich op het ontwerpen van een 3D indoor scanning en map building applicatie (algoritme) ontwikkeld aan de Hanoi University of Science and Technology (HUST). 3D technology is toegepast in verscheidende onderdelen zoals: transport, navigatie, 3D object scanners en meer. Het doel van het project is het scannen van indoor omgevingen met een real time cyclus tijd. De scan wordt geconverteerd naar een 3D map op de computer, met alle obstakels die in de omgeving aanwezig zijn.

Een belangrijk onderdeel van dit project was het ontwerpen van een algoritme dat gebruikt maakt van PCL (Point cloub library) software en een RGB-D camera. Het algoritme genereert een puntenwolk met de diepte afbeeldingen van de RGB-D-camera. Deze puntenwolken bevatten alle obstakels die in het gezichtsveld van de camera aanwezig zijn. Elke puntenwolk is genomen op een andere locatie en bezit dus een specifiek deel van de scene. De puntenwolken worden aan elkaar gehecht tot één puntenwolk die de hele scene toont.

Het algoritme is instaat om twee puntenwolken aan elkaar te hechten in 450 ms en 700 voor een update van de 3D map. De scene is in grote lijnen uitgelijnd. Het algoritme bezit nog een kleine fout wie zich vooral manifesteert bij de uitlijning van de kleinere objecten in de scene.

1 Introduction

1.1 Context

The master project takes place at the Hanoi University of Science and Technology (HUST) in Hanoi, Vietnam. HUST was established in 1956 as the first multidisciplinary technical university in Vietnam. HUST is the also an industrial training center for building and developing processes. The mission of HUST is to develop human resources, and to provide high quality workforce training and, to support the country in scientific research and technological innovation. HUST aims to becoming one of the leading research universities in techniques and technology (Hanoi University of Science and Technology, 1985).

Figure 1:1 HUST logo (Hanoi University of Science and Technology, 1985)

The future view for this projects lies with automated self-driving robots. When an object has to go from point A (objects starting position) to point B (objects target position), there are obstacles in the objects path which the object doesn't know. By making use of PCL (Point cloud library) and a RGD-D scanner a 3D map is built with all the objects that are in the environment. After the mapping process, the object will try to find position A and B in the map. When this information is known, the object generates the most optimal path from point A to point B, by using the A* path planning algorithm. The generated path avoids all the obstacles. During the path execution, the 3D map is continuously updated and the objects path is adjusted if needed. More info about the A* algorithm can be find in: A* algorithm.

1.2 Problem statement

The research discusses a solution to recreate the environment of an automated system on a computer in 3D. It is likely machines will work more and more closer together with humans than ever before (e.g. a human and a robot that are working together next to a conveyor belt). Systems even take over human tasks such as driving a car. To make sure all these processes are safe and reliable. The systems must be aware of their surrounding environment. The more complex and responsible

these systems become the more important it will be for these machines to capture the environment in a high resolution and with real-time updates. Figure 1:2 shows an example of a 3D map.

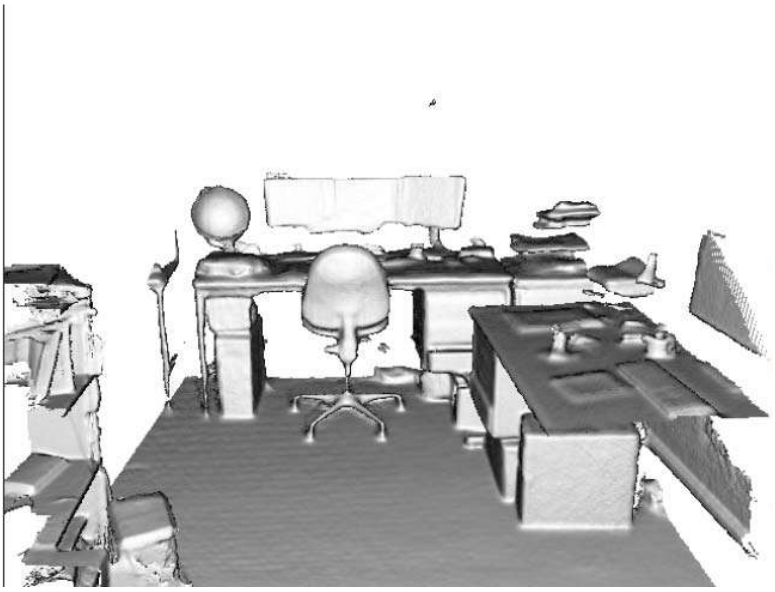


Figure 1:2 3D map example (Fleming, 2011)

Therefore, 3D mapping is a big research topic the last years. Every detail and dimension of the scene must be represented in the map. The color of the scene can also be captured and added to the 3D map. On the basis of 3D scanning CAD models of objects can be reconstructed quickly and more easily when using other methods. Since it is impossible to capture one scene from different viewpoints in one frame, shot or image, different frames must be taken from the scene and later combined to one part. Instead of taking 2D images, 3D images are captured with a special camera.

These 3D images are called point clouds. After the point clouds are taken, all the point clouds are combined to one point cloud. This process is called registration and is the main topic during this research. After the registration is done the aligned point clouds are represented in a 3D viewer for the user. All the processes will be executed in a certain time (< 100 ms) for a real-time application.

1.3 Goals / requirements

The main goal of the project is to create a computer algorithm that is capable of capturing point clouds from a camera. Next the algorithm needs to analyze the data of these clouds. When the analyzing is completed, the point clouds are aligned to one point cloud and shown in a 3D viewer.

Certain hardware is used to capture the environment. This hardware needs to be capable of capturing 3D data from the scene that is visible in the field of view of the camera. Capturing color data is not a must in this project, but requirements are that the hardware needs to be: low cost, user friendly and suitable for indoor environments.

The algorithm needs to work as efficiently as possible and as fast as possible to achieve real-time application speed. Apart from the speed, also the accuracy of the analyzing and registration will also play a big role in selecting methods and program flows. Speed and accuracy are almost in conflict with each other in every aspect. Hence the right ratio between speed and accuracy needs to be determined during the developing.

The efficiency of the algorithm will also be determined based on its user friendliness. First, the program may not crash while it is executing. Secondly, program needs to be Platform friendly, which means the algorithm should be executable across different devices. Finally, the code should be coded organized orderly with commenting so different programmers can understand the code and fast maintenance on the code can be performed. These conditions make sure the project is easily continued when it is paused for a certain time.

1.4 Method

To meet the conditions of the hardware a RGB-D scanner that is compatible with OpenNI is used. The market provides allot of different low cost RGB-D scanners. One of the most popular is the Microsoft Kinect or one of the Prime Sense devices. RGB-D cameras are already available for 99$. The camera that is used during this project is the Asus Xtion Pro Live. Both cameras are compatible with OpenNI. To make sure the algorithm can run on different devices the algorithm is coded in C++. The compiler that is used is Microsoft Visual Studio 2015.

To know what the algorithm main functions must be, some research must be done before starting coding. PCL and OpenCV provides basic information about image and point cloud processing. Both organizations are open source and contain image and point cloud algorithms that can be used in C++.

When the learning material is studied, a first step is to get familiar with the open source libraries. To code the first proto type, examples and papers about point cloud registration will be used. The algorithm will be tested in different environments. As a final step, it is examined on how it can be improved. When the core of the project is finished. The studied lecture and methods are documented and presented.

1.5 Structure of the book

The book starts with an introduction about the principle of point cloud registration and describes in short the two main types of point cloud registration. The next chapter describes in detail what a point cloud is and how it is created. Followed by chapters that explain different filter techniques and feature extractions used in 3D image processing. This is followed by the used methods for point cloud registration algorithms. The last chapters contain the practical implementation and setup that is used. The final chapters discuss the conclusions associated with the current unsolved problems and their possible solution. The main information sources of the book are: (Rusu, 2011) and (Dirk Holz, 2015).

2 Point cloud registration

Point cloud registration is another word for aligning two point clouds to one point cloud. When there are two point clouds A and B that represent the same environment / object. The main difference between the two clouds is that they are taken at a different location (viewpoint) in the real world. This means that the camera was at two different positons X and Y. between the two positions X and Y a transformation can be computed which means that X = T(Y) where T is the transformation matrix.

If the transformation matrix is found the two point clouds are combined to one big point cloud by transforming one of the two point clouds. After the transformation, the object / environment is visible from different aspects. By repeating this process, a point world that is a combination of all the point clouds can be created. The registration algorithm tries to find this transformation by making use of the data given by the two point clouds.

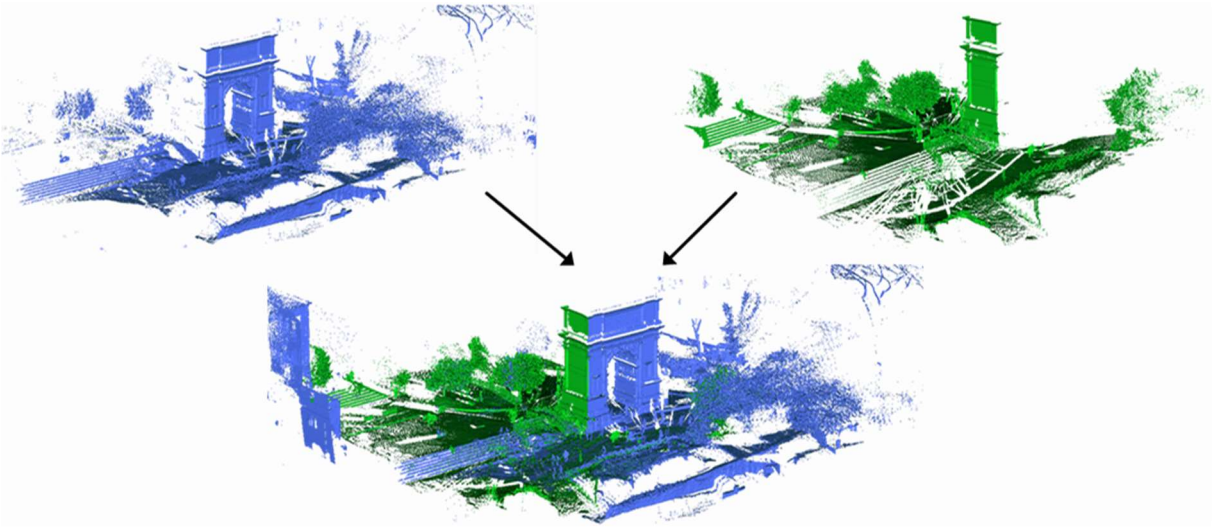


Figure 2:1 Point cloud registration example (Pascal Willy Theiler, 2012)

There are two main types of point cloud registration. Feature based registration and iterative registration algorithms. Feature based registration will use 3D point descriptors to describe points in the cloud. Next, the common features in cloud A and B are searched and a transformation matrix is calculated. Iterative registration algorithms don’t use any feature descriptors. They will make use of finding a transformation to align the closed point pairs (least Euclidian distance) as good as possible in an iterative way.

2.1 Finding the transformation matrix

When the two point clouds are perfectly aligned with each other. The distance between all the possible pair-points of cloud A and B is the smallest. The transformation matrix makes sure that the two point clouds are aligned to each other as best as possible. To find the transformation matrix an iterative method is applied that calculates the local minimum Euclidean distance of each possible pair-point of cloud B and T(A). T(A) is the transformed cloud A to align with cloud B. In formulas, we can note it in the following way:

$$A = cloud\ A$$
$$B = cloud\ B$$
$$T(A) = the\ transformed\ cloud\ A\ to\ align\ with\ cloud\ B$$
$$T(A, \theta) = T(A)\ where\ \theta\ stands\ for\ an\ optimisation\ parameter$$
$$based\ on\ the\ number\ of\ points\ dimensions\ or\ an\ estimated\ transformation$$

As told the smallest Euclidean distance is searched between each possible pair-point of B and T(A):

$$dist(T(A), B) = \sum_{\substack{i=1 \\ m \in T(A)}}^{n} \sum_{\substack{q=1 \\ s \in B}}^{n} (m_i - s_q)^2$$

Dist: the function that will calculate the sum of the Euclidian distance of each pair-point in cloud T(A) and B.

The minimum distance that is required is always be set by the user since reaching zero is impossible there will always be an error. This error is caused to the noisy data and outlier points. To make the registration more robust: Key points, sub sampling, filters, correspondence estimation and correspondence rejection is used during calculating the transformation matrix. These methods are discussed later. There are 2 main types of transformations rigid transformations and non-rigid transformations. A rigid transformation consists of a rotation and a translation while a non-rigid registration consists of a non-linear transformation. Since the most famous point set registration method ICP (iterative closed point) uses a rigid transformation. This will mainly be discussed.

2.2 ICP

Every registration algorithm is computably expensive. This is caused by the high number of points in a cloud and the high number of possible outcomes. The ICP algorithm calculates the best possible transformation matrix to fit the point clouds A

and B. This by using an iterative method by always adjusting the transformation matrix. The transformation matrix starts with a default value and is then adjusted each loop to find the least square transformation matrix. The ICP algorithm has many variants based on the basics of the ICP algorithm. The standard ICP algorithm works as follows:

$\theta := \theta_O$
$While\ not\ registred$
$X := \theta$
$for\ m_i \in T(A, \theta)$
$s_j := Closed\ point\ in\ B\ to\ m_i$
$X := X + \langle s_j | m_i \rangle$
$\theta := Least\ square\ (X)$
$return$

While the algorithm is working, it tries to align all the correspondences as best as possible in an iterative way. The algorithm keeps looping until a converged criterion is met. To receive a good result, it is required that the two point clouds contains a decent overlap. If the overlap is not large enough the alignment is wrong. The speed of the algorithm depends on the number of points that are used and the number of iterations needed to align the two point clouds (François Pomerleau, 2015 May).

2.3 Registration pipeline

Figure 2:2 shows an example how a full registration pipeline looks like. There are many ways for setting up a registration pipeline and this process usually depends on the: scene, quality, sensor type and speed that is required. In the next chapters, every part of the flowchart will be discussed in detail.

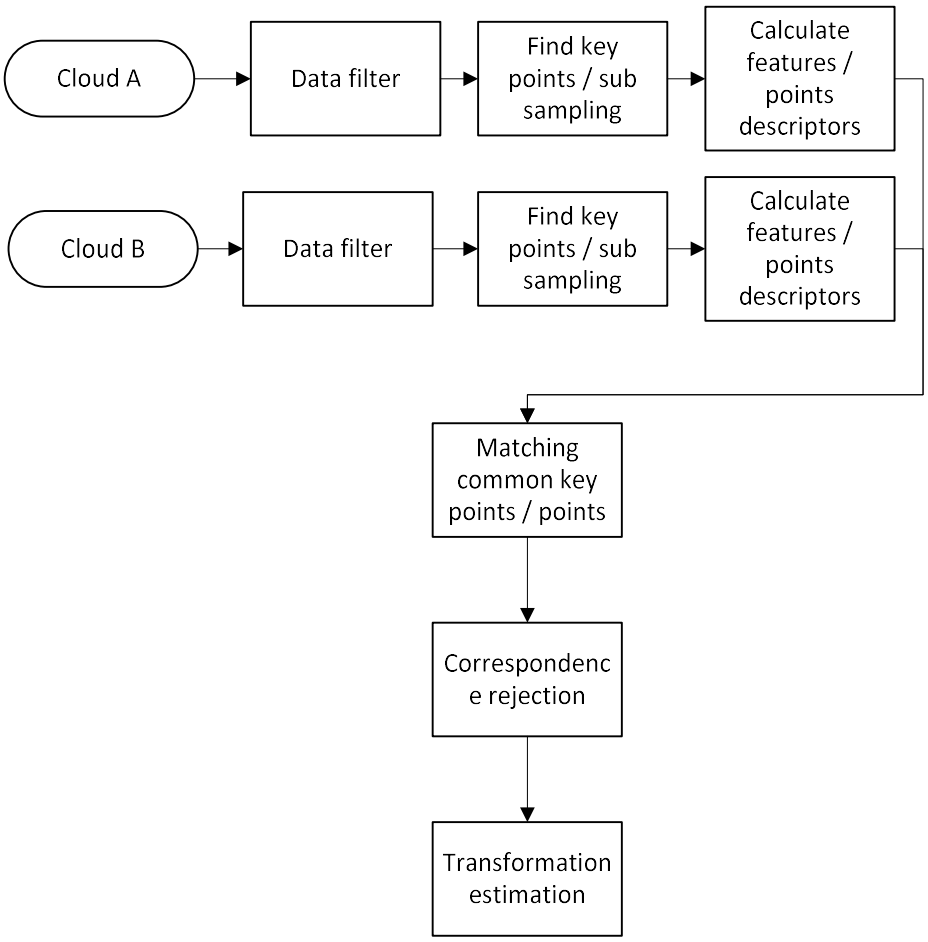


Figure 2:2 Standard registration pipeline

3 Point cloud

A point cloud is a list of data points in a coordinate system. In the project the point cloud consists of 3D data points X, Y and Z coordinates. All these points represent the environment that is registered by the 3D scanner. Sometimes each point can contain more information than just its location in the cloud such as: RGB data, normal vector values and more. Point clouds usually contains a high amount of points for example, a raw RGB-D image with a resolution of 640 by 480 contains 307 200 points. Point clouds can be subdivided into two main types.

3.1 Organized Point clouds

In this type of cloud the data points are stored in structure way. The cloud resembles an organized image (matrix) as showed in Figure 3:1. The data is subdivided in rows and columns and this causes much faster calculations in the cloud. For example, the nearest neighbor of a point is find by just shift one column and / or row in the wanted direction. This type of clouds can be received from stereo camera's or time of flight sensors. RGB-D images / depth images will deliver an organized point cloud.

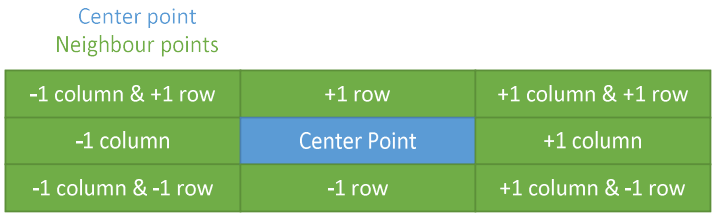


Figure 3:1 organized point cloud

3.2 Unorganized point clouds

The points of these clouds are structured in a random way. All the points are stored in one list. To find the nearest neighbor(s) it needs to iterate through the whole list. This extends the computation time since point clouds can contain many points. It is recommended that an unorganized point cloud always is converted to an organized point cloud. Since the nearest neighbor(s) method is executed more than once.

3.3 Transform a point cloud

A common request with point clouds is to move the point cloud in a digital space / world. Figure 3:2 shows a representation of a point cloud transformation. A transformation can be acquired by a homogenous transformation applied to each point in the point cloud. This homogenous transformation is split in a rotation and a translation. The rotation will rotate the point cloud over the X, Y and Z axis while the translation shifts the point cloud over the X, Y and Z directions. This homogenous translation is presented by a four by four matrix.

3D rotation + translation: $\begin{Bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \\ 0 & 0 & 0 & 1 \end{Bmatrix}$

Rotation around X, Y and Z (also known as the XYZ Euler transformation):

$$\begin{Bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{Bmatrix} * \begin{Bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{Bmatrix} * \begin{Bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{Bmatrix}$$

=

$$\begin{Bmatrix} \cos\alpha * \cos\beta & -\sin\alpha * \cos\beta & \sin\beta \\ \cos\alpha * \sin\beta * \sin\theta + \sin\alpha * \cos\theta & \cos\alpha * \cos\theta - \sin\alpha * \sin\beta * \sin\theta & -\cos\beta * \sin\theta \\ \sin\alpha * \sin\theta - \cos\alpha * \sin\beta * \cos\theta & \cos\alpha * \sin\theta + \sin\alpha * \sin\beta * \cos\theta & \cos\beta * \cos\theta \end{Bmatrix} =$$

$$\begin{Bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{Bmatrix}$$

This 4 by 4 transformation matrix will be applied to each point in the cloud to generate the same cloud in another position and orientation in the digital world.

$$\begin{Bmatrix} X_{tn} \\ Y_{tn} \\ Z_{tn} \\ 0 \end{Bmatrix} = \begin{Bmatrix} r_{11} & r_{12} & r_{13} & T_x \\ r_{21} & r_{22} & r_{23} & T_y \\ r_{31} & r_{32} & r_{33} & T_z \\ 0 & 0 & 0 & 1 \end{Bmatrix} * \begin{Bmatrix} X_n \\ Y_n \\ Z_n \\ 0 \end{Bmatrix}$$

With t, the transformed point

With n, the point number in the cloud

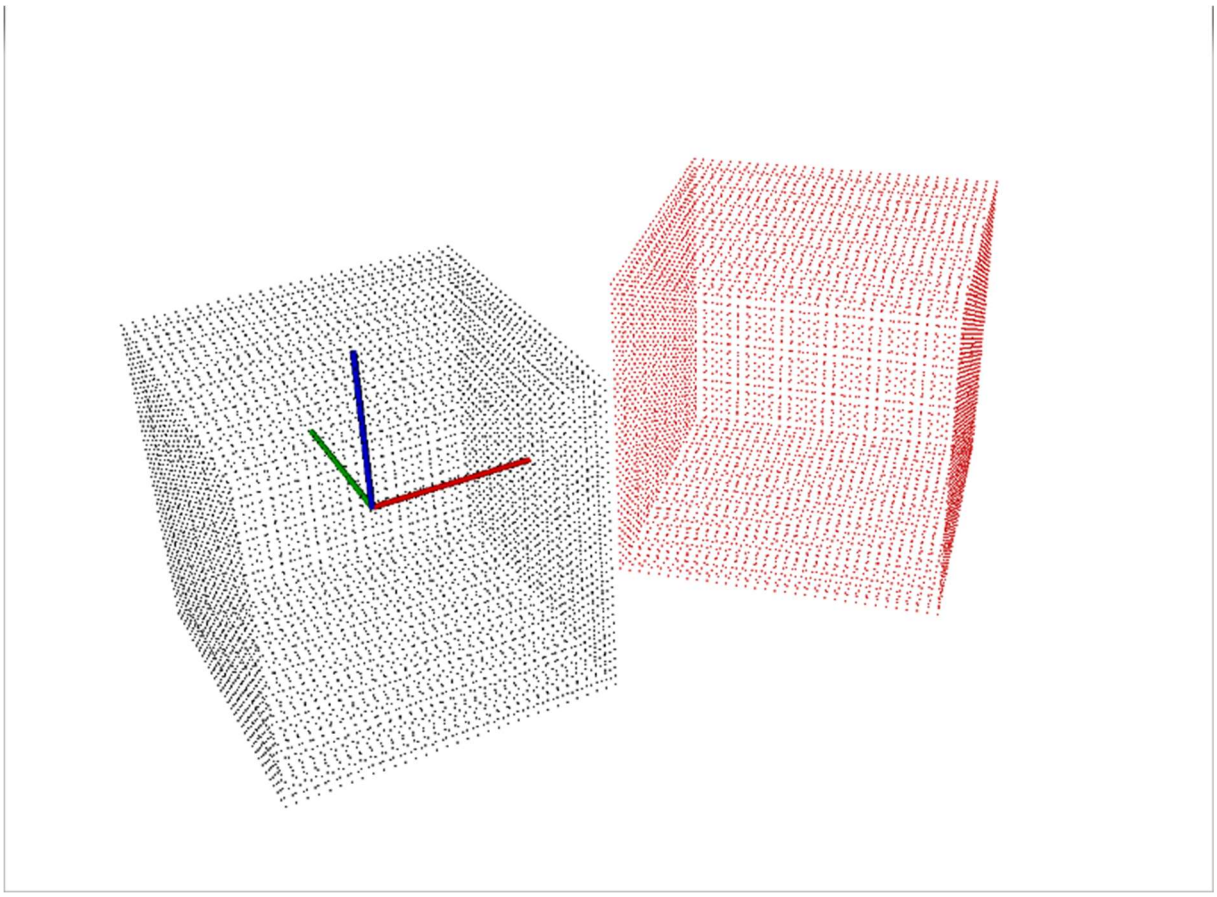


Figure 3:2 Transforming a point cloud (Rusu, 2011)

3.4 KD-Tree Storing a point cloud

As already mentioned a point cloud can contain many points. It is recommended to store the points in an intelligent way. The Kd-tree structure is a very efficient to store the point clouds and find the neighbor(s) of a point in the point cloud. The Kd-tree is a k dimensional tree that organizes the points. Point clouds mostly contains 3D data so the kd-tree that is used is 3 dimensional. The following example shows how a 2D kd-tree works for simplicity:

Given a set of random points:

(2,3), (5,4), (9,6), (4,7), (8,1), (7,2).

The first step is to find the median of the points of a chosen axis. In this case, the X-axis is chosen. The median of the X values is 7 so the starting coordinate is (7,2)

The next step is to split the data based on the axis where the median was calculated on. In this case, we will split the points where the values are higher and lower then 7.

Table 1 Kd-tree example first split

(7,2)	
X < 7	X > 7
(2,3)	(9,6)
(5,4)	(8,1)
(4,7)	

Next, the next dimension is chosen (y-axis) and the median is determined again on the two given columns from the previous split.

Medians: 4 and 6

coordinates: (5,4) and (9,6)

Again, we will split the coordinates where y < 4 and y > 4 and where y < 6 and y > 6

Table 2 Kd-tree example result

<table>
<tr><td colspan="4">(7,2)</td></tr>
<tr><td colspan="2">X < 7</td><td colspan="2">X > 7</td></tr>
<tr><td colspan="2">(2,3)</td><td colspan="2">(9,6)</td></tr>
<tr><td colspan="2">(5,4)</td><td colspan="2">(8,1)</td></tr>
<tr><td colspan="2">(4,7)</td><td colspan="2"></td></tr>
<tr><td colspan="4"></td></tr>
<tr><td colspan="2">(5,4)</td><td colspan="2">(9,6)</td></tr>
<tr><td>Y < 4</td><td>Y > 4</td><td>Y < 6</td><td>Y > 6</td></tr>
<tr><td>(2,3)</td><td>(4,7)</td><td>(8,1)</td><td>none</td></tr>
</table>

All the points are now sorted in a kd-tree. The following images shows a graphical interpretation of the kd-tree of the given points.

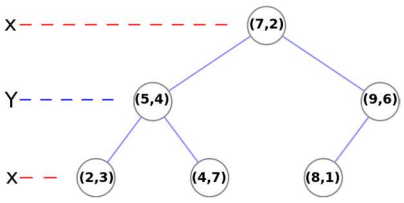


Figure 3:3 Kd-tree graphical interpretation (K-d tree, 2017)

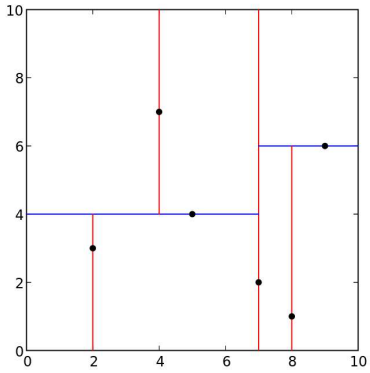


Figure 3:4 Kd-tree graphical interpretation (K-d tree, 2017)

The pseudo code of a kd-tree can be noted as follows:

```
function kdtree (list of points pointList, int depth)
{
    // Select axis based on depth so that axis cycles through
all valid values
    var int axis := depth mod k;

    // Sort point list and choose median as pivot element
    select median by axis from pointList;

    // Create node and construct subtree
    node.location := median;
    node.leftChild := kdtree(points in pointList before
median, depth+1);
    node.rightChild := kdtree(points in pointList after
median, depth+1);
    return node;
}
```

3.4.1 Adding a new point to the tree

This topic is explained by an example. Adding points to the tree is similar as building the tree. For example, the coordinate (3,2) is added to the tree represented in Figure 3:5

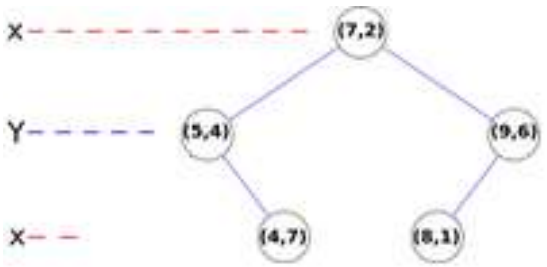


Figure 3:5 Kd-tree adding a new point (K-d tree, 2017)

When adding a new point to the tree the function always starts at the top. Based on which axis the data is split. In this case the x-axis, the function determines if the input coordinate (3,2) has an x value lower or higher than the current x value of the starting node. If lower, then we go left if higher then we go right. In this case, we will go right (3<7) to point (5,4). Based on the y values of the points (5,4) and (3,2) there will be a new leave created with point (3,2). The result is displayed in Figure 3:6:

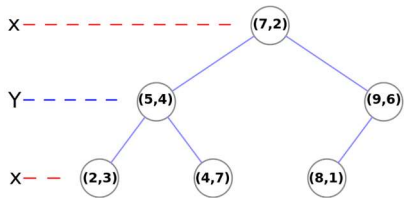


Figure 3:6 Kd-tree adding a new point (K-d tree, 2017)

Adding too many new points to the tree can make the tree unstable. When the tree is unstable the search techniques in the tree perform slower and the tree is not optimal anymore. Refreshing the tree (or a part of the tree) can be a solution for this problem or using another kd-tree structure that has an adaptive function like: divided k-d tree, pseudo k-d tree, k-d B-tree, hB-tree and Bkd-tree.

3.4.2 Removing a point from the tree

When removing a point from the tree while still retaining the tree structure. The rest of the tree that is under the deleted point is recreated again. Another solution is to find a point that can replace the deleted point.

3.4.3 Comparing with organized point clouds

KD-trees are still not organized point clouds. The structure of an organized point cloud (integral image) is always faster than a KD-tree. In general, it is recommended to use organized point clouds whenever possible.

4 Filters

Due every point cloud contains: noise, outliers, false data or data that is not interested, filtering is applied to the point cloud process. Filtering makes the point cloud processing more stable and faster. Manny types of filters are available for point cloud processing.

4.1 Pass Through Filter

This filter removes points from a point cloud that are out of the specified XYZ dimensions. For example, we only want the points that are in a specified region of the cloud. A simple filter example would look like this:

```
foreach(Point point in pointcloud.Points)
{
    bool remove = false;
    if(point.x > upper_x || point.x < lower_x)
    {
        remove = true;
    }

    if(point.y > upper_y || point.y < lower_y)
    {
        remove = true;
    }

    if(point.z > upper_z || point.z < lower_z)
    {
        remove = true;
    }

    if(remove)
    {
        pointcloud.removePoint(point);
    }
}
```

4.2 Voxel grid filter

A voxel grid filter down samples the point cloud. The number of points in the point cloud is reduced while still representing the input cloud. This down sampling will make calculations like: normal estimation and feature detection must faster. One voxel can be represented as a 3D rectangle with given dimensions. Next, the spatial average is taken from all the points that lays in one voxel. All these points are now represented by one voxel (bigger point). This is done for each point in the point cloud. The resolution from the point cloud is decreased but calculations are much faster which is always wanted with real time applications. Figure 4:1 shows a point cloud before voxel grid filtering and Figure 4:2 shows the same point cloud after the filtering.

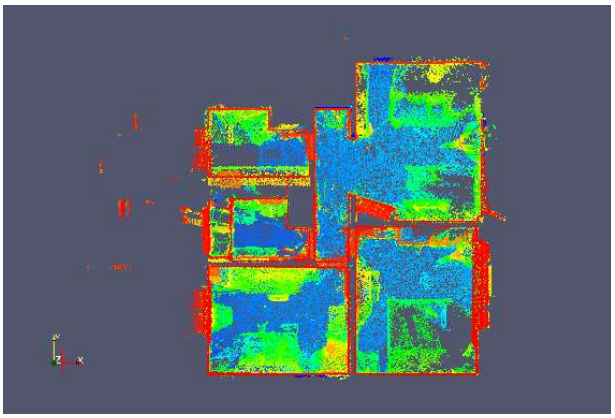


Figure 4:1 point cloud (Libpointmatcher, 2006)

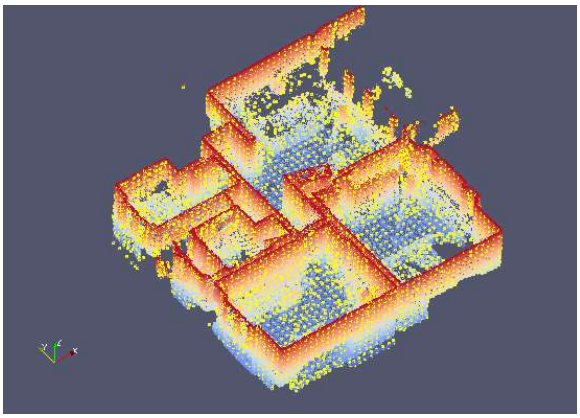


Figure 4:2 voxel filtered point cloud (Libpointmatcher, 2006)

4.3 Statistical outlier removal filter

Every point cloud contains errors. Some errors are outliers these outliers can create big errors on point cloud registration and normal estimation. These outliers can be removed by statistical outlier analyzes on every point in comparison with the k-neighborhood points. The k-neighborhood points who doesn't recommend the specified criteria are removed from the point cloud. For each point in the point cloud the mean distance from all the k-neighborhood points is calculated. Assumed that the result is a Gaussian distribution with a mean and a standard deviation. Where the mean distance of the points is outside of the interval, these points are removed as showed in Figure 4:3

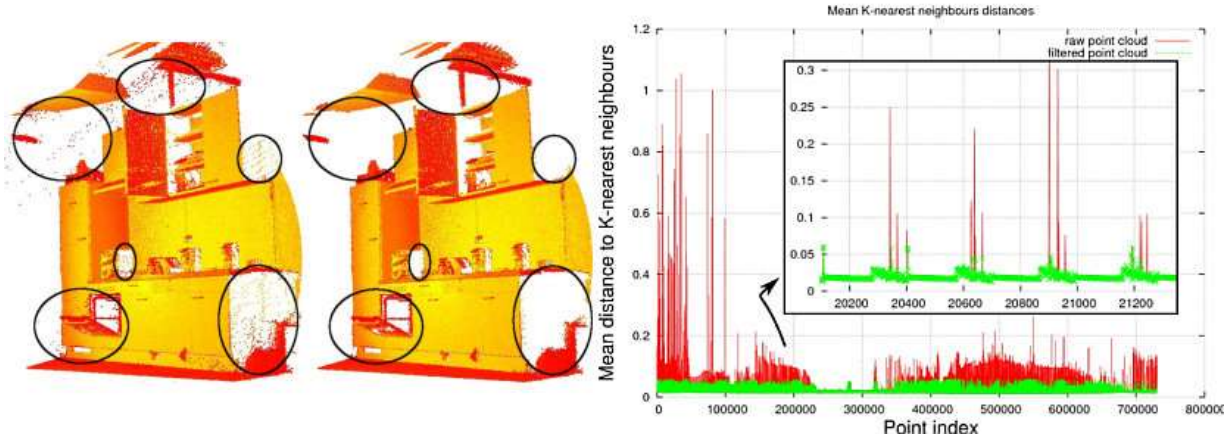


Figure 4:3 6.1.3 Statistical outlier removal filter (Rusu, 2011)

4.4 Radius outlier removal filter

This filter will create a 3D sphere around each point in the point cloud. The next step is to find all the points that are in the 3D sphere. If the number of points that are in the 3D sphere is too low, then this point will be removed. Since we can see it as a lonely point in space that probably is: an outlier, noise or measurement error. Figure 4:4 shows a simple implementation of this filter.

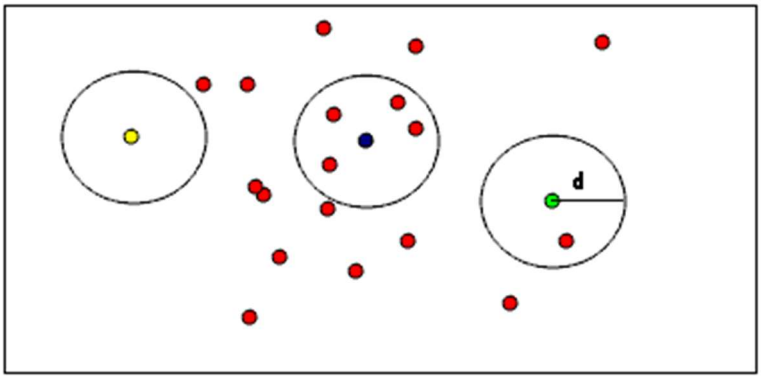


Figure 4:4 6.1.4 Radius outlier removal filter (Rusu, 2011)

4.5 Bilateral filter

Bilateral filters are very useful to remove the noise while keeping the edges sharp also called smoothening / blurring. The filter works with a weighted distribution. Each weight of the pixel is determined by the intensity value of that pixel and its neighbors. Assumed that the values are disturbed by a Gaussian distribution. When this filter is applied to an image, a cartoonish effect is the output. In point clouds this weight is calculated by the depth values (Z distance) of every pixel. The bilateral filter is demonstrated on a 2D image in Figure 4:4Figure 4:5

Figure 4:5 Bilateral filter (Farzana, 2011)

4.6 Conditional removal filter

This filter can vary in many ways. The user gives some conditions that every point needs to require. If the point doesn't meet the requirements, then it is removed from the point cloud.

5 Key points detection & subsets

Due the high number of points in the clouds, processing can become quite computationally expensive. The point clouds needs to be down sampled. A way for doing this is finding the key points in the cloud. And do further calculations with these key points instead of using all the points in the cloud. Another way to reduce the number of points in the cloud. Is to take every n-th point in the cloud. Because point clouds mostly contain redundant data, this will have no influence on the quality of the process. Key points are interesting points in a (3D range) image. These points will mostly represent an object. Finding these points can be used for object recognition in a point cloud or registration of point clouds and many other applications There are still many 2D key point detectors used in 3D clouds such as Harris and Stiff. The 2D key point detectors are still reliable in finding the key points. A requirement is that one must have a RGB image of the scene. 3D key points detectors work with comparing the 3D surface. 2D key point detectors use the RGB or grey color values. A hybrid detector makes use of both.

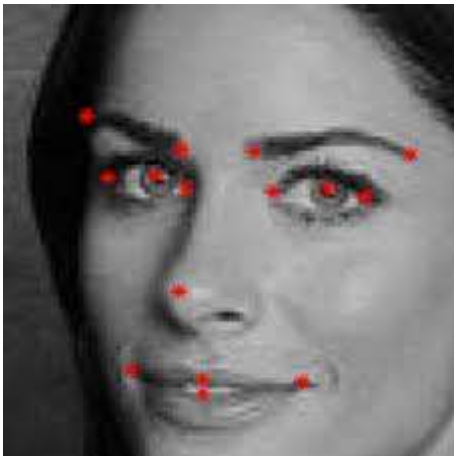


Figure 5:1 Key points example (Bashan, 2015)

5.1 NARF Points (Normal aligned radial feature)

NARF key points are extracted from depth images. A NARF point or interesting point has two common features. The points around a NARF point represent a stable surface and where there are big changes in the vicinity. Secondly, because we will see the objects from different viewpoints. We will focus on the outer borders of the objects, since these are unique in different viewpoints. A stable interest point must contain a significant change of the surface in the local neighborhood to be detected from different viewpoints. To find these stable interest points we must be able to find the borders of objects in the point cloud. This can be done by:

- Looking for great changes in distance in the 3D image between the neighborhood and an interesting point;
- Determine a score about how much the neighborhood surface changes around a point;
- Look for how stable the surface is of a point in the point cloud and how much the vicinity changes in the point cloud;
- Perform a smoothing on all the interesting points;
- Perform a non-maximum suppression to find the final interest points;

Describing the features of a point makes it possible to compare different points with each other and perform actions based on the results of the comparison. The goal of a NARF feature detector is that it captures the key points in a depth image (Steder, Bogdan, Konolige, & Wolfram, 2010).

5.2 Uniform sampling

Uniform sampling is a way to down sample a cloud e.g. by taking every n-th point. The user can tell the filter an increment where every point must be taken. An advantage from this method instead of key points is that you keep information over the whole scene, instead of just from the key points. Another advantage is, when there are not so many objects in the scene. There will likely be too less key points to do calculations on the cloud, such as registration. This problem is already solved with uniform sampling. Also, uniform sampling is easier to implement and faster.

6 Point cloud features descriptor

Each point cloud mostly contains features (key points). These features can represent objects in the real world or parts of these objects. These features can be used to detect objects in a point cloud or to combine two point clouds that where taken at a slightly different position. The two point clouds from Figure 6:1 have common features. Finding these features is one of the main things in combing the point clouds. Extra information can be used to detect the features such as the RGB values of each point, intensity and the surface normal of each point. To distinguish the different features and there surfaces the use of the surrounding neighbors of each feature point will be used. Good feature estimations are immune for:

1. Rigid transformations: 3D rotation + translation;
2. Noise;
3. Varying sample density: sometimes the same feature will contain out of less samples if it is taken from a different viewpoint;

The goal of a feature descriptor is that it captures the key points and occupied space around a point. In this way, objects and free space can described. Secondly, the feature descriptor need to be robust against noise. The last requirement is to attach a local coordinate frame to the point so that it can be used to compare with other features taking from different viewpoints.

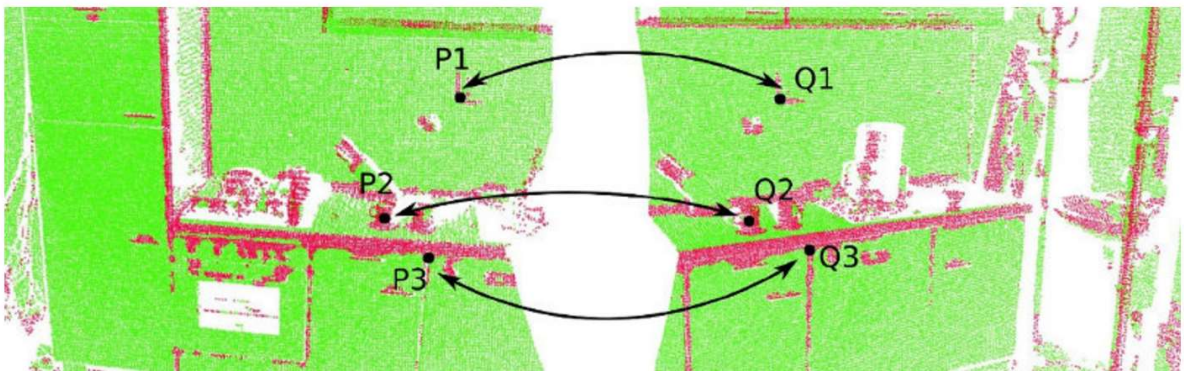


Figure 6:1 Feature matching (Rusu, 2011)

6.1 Surface normal in a point cloud

In a three dimensional space a normal on a 3D surface is a vector that is perpendicular to a point on the surface. The normal can represent a line, vector, surface or plane (Figure 6:2 & Figure 6:3). These will all be perpendicular to the point in the 3D surface. The normal are common used in 3D computer graphics to calculate the light and shadows based on the position and orientation of the camera that looks at the surface. The normal estimation is a very simple way to describe a feature of a point. The disadvantage of these 'features' is that there will be many duplicates. Although a normal of a point is also used to describe a 3D point in a higher dimensional space.

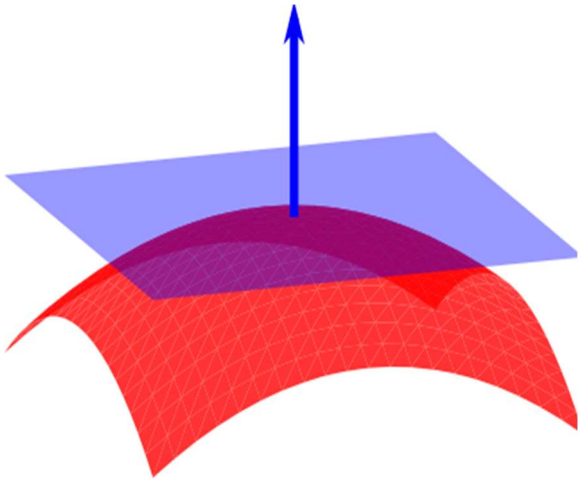


Figure 6:2 Plane normal (Rusu, 2011)

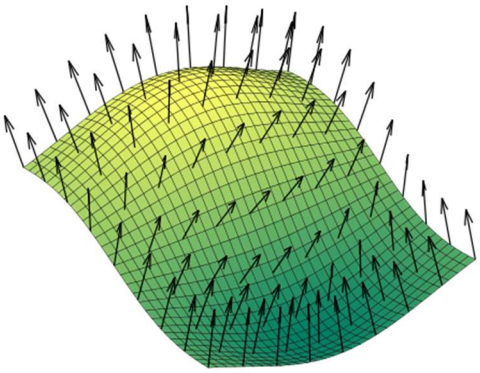


Figure 6:3 Vector normal (Rusu, 2011)

6.2 Defining a point cloud feature (PFH)

To describe a point cloud feature, the surface normal and curvature around a point in the cloud is used. In other words, we will try to calculate the geometrical properties (mean curvature) around a point using its k-neighborhood. These geometrical properties are stored in a multi-dimensional histogram (point feature histogram PFH). This histogram describes the geometrical feature around a point and is invariant for 6D movement. The neighbors of the point (k-neighbors) are used to calculate the 3D curvature. A common consequence is that scenes contain many points with very similar feature values. Filter techniques will occur that common feature values will be used. The PFH is calculated by using the estimated normal of the k-neighborhood points. Hence the quality of the feature point estimation dependents on the estimation of the normal values of the points in the cloud.

To calculate the geometrical features. A point in the cloud is chosen. Then a 3D sphere is drawn around the point with a given radius R. All the k-neighborhoods points that are in the 3D sphere are used to calculate the geometrical feature around the point. The final PFH is computed by the relationships of all the pair-points in the 3D sphere and thus this is computed with a complexity of: $O(n*K^2)$ with K the number of neighbors in the sphere. The Figure 6:4 below shows a representation of the 3D sphere with the k-neighborhood points.

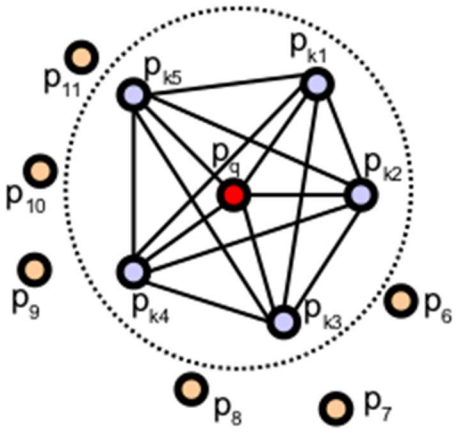


Figure 6:4 PFH K-neighbours (Rusu, 2011)

The used data for each pair-point is a total of 12 values. The XYZ coordinates of P_Q and P_K (6 values) and the normal values of P_Q and P_K (6 values). By using a set of angular values the 12 values are stored in a quadruplet (4 values). By using the angular values, an UVW coordinate frame is attached to each point in the 3D sphere. In the following Figure 6:5, P_s represents P_Q and P_t represents P_k.

$$\mathsf{u} = \boldsymbol{n}_s$$

$$\mathsf{v} = \mathsf{u} \times \frac{(\boldsymbol{p}_t - \boldsymbol{p}_s)}{\|\boldsymbol{p}_t - \boldsymbol{p}_s\|_2}$$

$$\mathsf{w} = \mathsf{u} \times \mathsf{v}$$

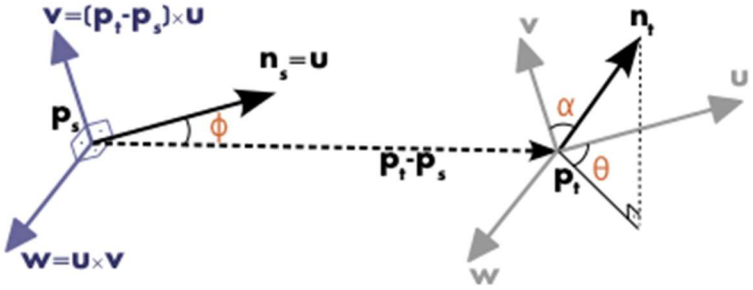


Figure 6:5 PFH UVW coordinate frame (Rusu, 2011)

The angles between the two points can be calculated in the following way:

$$\alpha = \mathsf{v} \cdot \boldsymbol{n}_t$$

$$\phi = \mathsf{u} \cdot \frac{(\boldsymbol{p}_t - \boldsymbol{p}_s)}{d}$$

$$\theta = \arctan(\mathsf{w} \cdot \boldsymbol{n}_t, \mathsf{u} \cdot \boldsymbol{n}_t)$$

(Rusu, 2011)

The fourth value is the Euclidian distance between the 2 points:

$$d = \|\boldsymbol{p}_t - \boldsymbol{p}_s\|_2$$

(Rusu, 2011)

The Euclidian distance is often not used especially when 2.5D images are used. 2.5 D images are images where the distance between the pixels increases when the distance between the viewpoint and object increases. When the Euclidian distance is used, the quadruplet can be noted as: $(\alpha, \phi, \theta, d)$. The final PFH represents all the quadruplets sets of the pair-points in the 3D sphere. When the d is not used, the quadruplet is transformed to a triplet as: (α, ϕ, θ) when triplets are used we will speak from a SPFH (simplified point feature histogram).

6.2.1.1 FPFH

FPFH stands for fast point feature histogram. This method is used when real time point cloud feature implementation is needed. Due the faster calculation method some estimations are made and thus this method is less accurate. The method tries to reduce complexity from $O(n*K^2)$ to $O(n*K)$. As described in the previous section a SPFH is used to describe each point. When the geometric feature around a point is calculated the k-neighbors will be determined around the point P_Q. Next, the SPFH values of the neighbor points are used to determine the geometric feature around the point P_Q. These geometric values are called FPFH. (fast point feature histogram). The following formula shows how the FPFH values of a point is calculated:

$$FPFH(\boldsymbol{p}_q) = SPFH(\boldsymbol{p}_q) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_k}\cdot SPFH(\boldsymbol{p}_k)$$

(Rusu, 2011)

Here is W_k a weighted value, it is the Euclidian distance between P_K and P_Q. Points that are further away from P_Q have less influence on the FPFH values of P_Q. To optimize the quality of the FPFH values, the neighbor of the neighbors are used to determine the final value. First all the neighbors of P_Q are calculated these neighbors are P_K. The next step is to find all the neighbors of the P_K points and calculate the SPFH values of the P_K points. Next the SPFH values of the P_K points are used to calculate the FPFH value around P_Q. Note that points in Figure 6:6 are used twice when finding the neighbors of the neighbors. These points are noted with thicker lines.

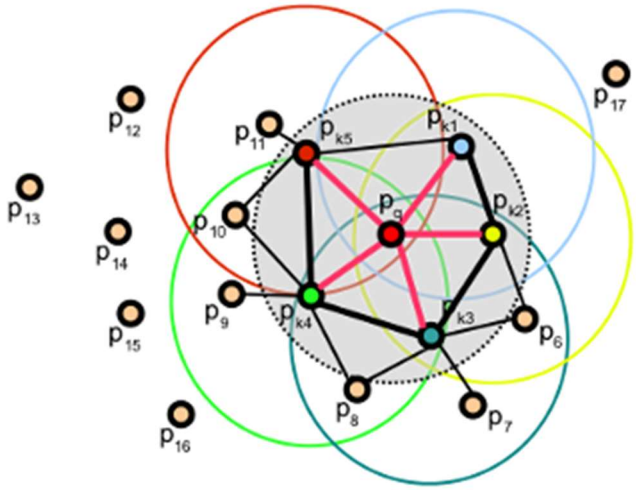


Figure 6:6 FPFH (Rusu, 2011)

6.2.1.2 Difference between PFH and FPFH

- All the neighbors of P_Q are not fully used;
- The FPFH method include points max 2*R away from $P_{Q;}$
- Some P_K points will be used twice;
- Faster and easier to compute;

7 Registration techniques

Registration algorithms consists out of different steps. This chapter explains the basic steps in a registration algorithm. There are three main steps in a registration algorithm: matching, rejection and transformation estimation. These steps happen after the point cloud analyzing. In the analyzing step data from the point cloud is sampled to use in the registration algorithm. The information is collected from the paper (Dirk Holz, 2015).

7.1 Matching (key) points

In the matching process the corresponds between the two point clouds is estimated. Given two point clouds A and B. P_A represent a point in cloud A, the goal is to find the approximated corresponding point of P_a in cloud B which is represented as P_b. The matching process is determined on which type of registration is applied. If it is a feature based registration the matching algorithm will search for the points that has the same feature descriptor values. When an Iterative registration is applied. The closed neighbor (P_b) in a 3D space of point P_a in the target cloud B will be searched. Instead of iterating through all the points in the target cloud a KD-tree or organized point cloud will be applied to find the closed neighbor of the point P_a faster. To make to matching process more robust. The user can define a minimum Euclidian distance that the point pairs must have. If the Euclidian distance is bigger the point pairs is filtered out. Figure 7:1 shows an example of correspondences matching.

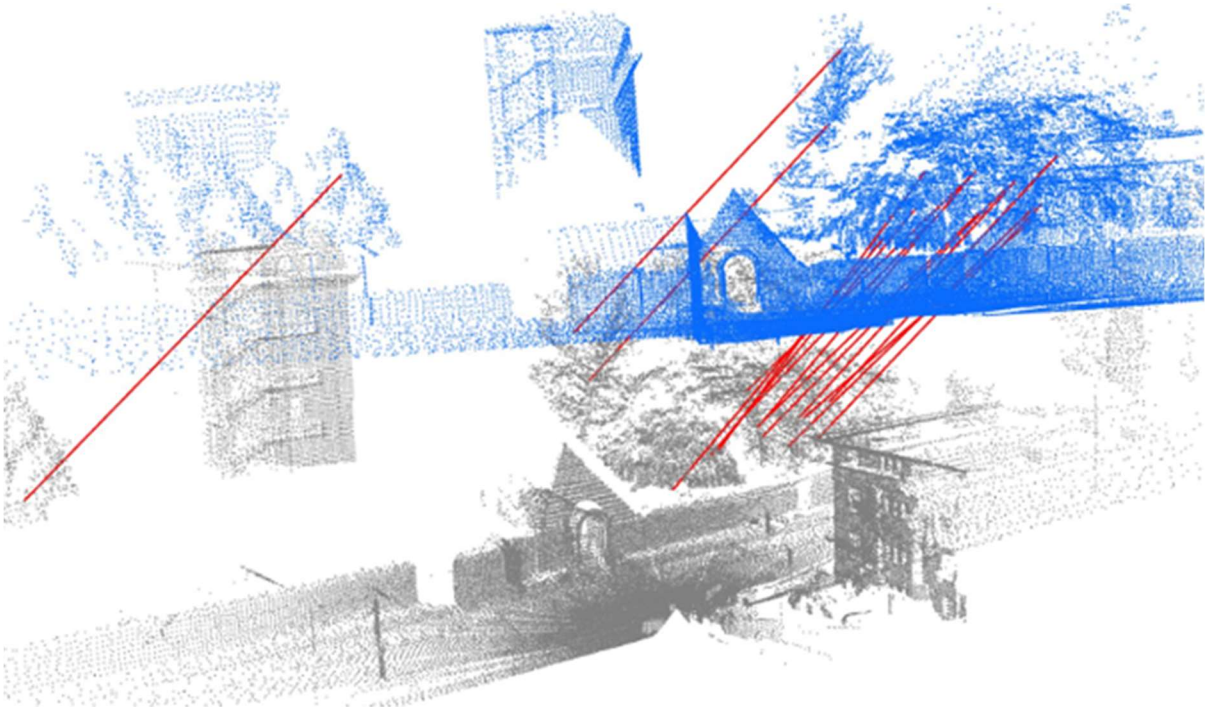


Figure 7:1 Matching corresponds example (Rusu, 2011)

7.2 Rejection correspondences

Matching (key) points is a quick approximation. False correspondences have a negative effect on the registration outcome. Filtering out these false correspondences is done in this step. Multiple type of rejecters can be applied in series. But one must keep in mind that each step that is added in a registration process increases cycles time. If a real-time application is desired sometimes we must offer speed for quality. Next different rejecters will be discussed.

7.2.1 Correspondences rejection based on distance

When rejection based on distance is applied, the Euclidian distance between each point pair is calculated. When the Euclidian distance is larger than the threshold value, the point pair is removed from the correspondences list (Figure 7:2). There are two ways the distance can be configured. One way is to choose a fixed threshold value. A second way is to compute the median Euclidian distance of all the point pairs. The median value will be the threshold value to filter out the point pairs. The advantage of this method is that the median value will become smaller the next time the correspondences are filtered. Since this method will be used in a loop. The median rejection also doesn't require any parameters.

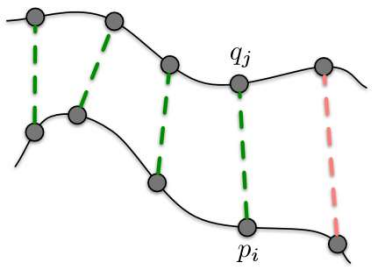


Figure 7:2 Correspondences rejection based on distance (Dirk Holz, 2015)

7.2.2 Correspondences rejection based on duplicate matches

While each point in the source cloud will be matched with a point in the target cloud, it is possible that a point in the source cloud will be assigned to multiple points in the target cloud. This can happen when multiple points are almost described in the same way. The best pair-point will be the pair-point with the smallest Euclidian distance. (Figure 7:2).

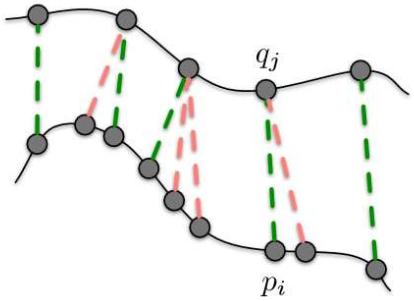


Figure 7:3 Correspondences rejection based on duplicate matches (Dirk Holz, 2015)

7.2.3 Correspondences rejection based on surface normal

As already described in the previous sections the surface normal of each point can be computed. Another way and very effective to filter out the false pair-points is to compare the surface normal of the two points. If the surface normal pair is not aligned in the same direction (Figure 7:4), the pair-point is removed from the list.

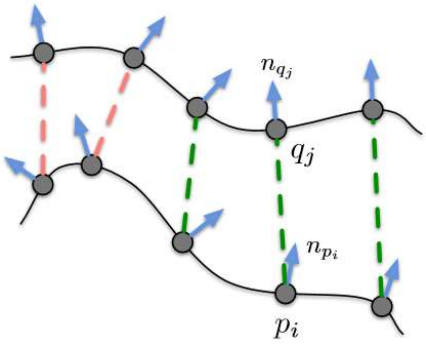


Figure 7:4 Correspondences rejection based on surface normal (Dirk Holz, 2015)

7.2.4 RANSAC-based rejection

RANSAC stands for random sample consensus, this method is used to detect the outlier pair-points in the correspondences list. This is done by picking two random pair point in the list and draw a line between these two points. Next, all the pair points that are in the threshold value around the drawn line are accepted. These points are called the inliers, all the pair-points that are outside this threshold value are called outliers. This calculation is executed several times and the line that contains most inliers is accepted. As demonstrated in Figure 7:5 the blue line is the line that contain most inliers (showed in blue) the outliers are showed in red.

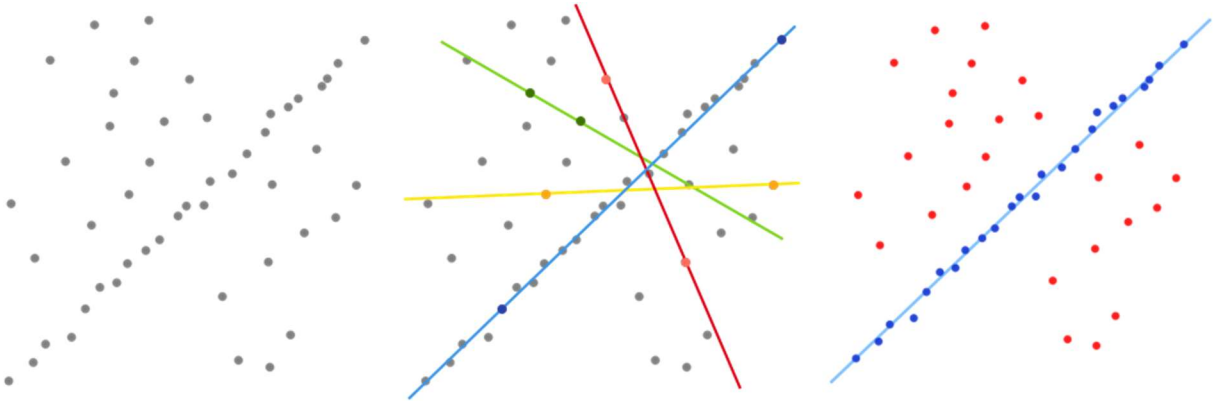


Figure 7:5 RANSAC-based rejection (Joe Ahuja, n.d.)

7.3 Transformation estimation

After the point clouds are analyzed, the correspondences are searched and rejected, a transformation estimation will be calculated. This transformation T consist of a rotation R and a translation t. there are two main types of transformation estimations. Point to point and point to plane.

7.3.1 Point to point

With a point to point estimation the transformation with the least distance between all the point pairs is estimated:

$$E_{point\ to\ point} = \sum_{k=1}^{N} w_k * \|T * p_k - q_k\|^2$$

$E_{point\ to\ point} = the\ total\ error\ of\ all\ the\ points$
$N = the\ total\ number\ of\ point\ pairs$
$K = the\ index\ in\ the\ point\ pair$
$w_k = a\ weight\ that\ can\ be\ given\ to\ all\ the\ point\ pairs.$
$This\ can\ be\ calculated\ by\ a\ gaussian\ distribution\ of\ the\ distances\ of\ all\ the\ point\ pairs$
$T = the\ transformation\ matrix$
$p_k - q_k$
$= a\ point\ pair\ where\ p_k\ is\ a\ point\ from\ cloud\ P\ and\ q_k\ is\ a\ point\ from\ cloud\ Q$

The point to point estimation is used in the famous ICP algorithm and can only be used when the point clouds are already roughly aligned with each other

7.3.2 Point to plane

Point to plane has be showed to be more robust and faster than point to point estimations (Dirk Holz, 2015).

$$E_{point\ to\ plane} = \sum_{k=1}^{N} w_k * ((T * p_k - q_k) * n_{qk})^2$$

$E_{point\ to\ point} = the\ total\ error\ of\ all\ the\ points$
$N = the\ total\ number\ of\ point\ pairs$
$K = the\ index\ in\ the\ point\ pair$
$w_k = a\ weight\ that\ can\ be\ given\ to\ all\ the\ point\ pairs.$
$This\ can\ be\ calculated\ by\ a\ gaussian\ distribution\ of\ the\ distances\ of\ all\ the\ point\ pairs$
$T = the\ transformation\ matrix$
$p_k - q_k$
$= a\ point\ pair\ where\ p_k\ is\ a\ point\ from\ cloud\ P\ and\ q_k\ is\ a\ point\ from\ cloud\ Q$

$n_{qk} = the\ normal\ of\ point\ q_k$

Point to plane estimation uses the distances between the transformed point p and the plane of point q. while point to plane estimation are non-linear. Linearization is done by sin(θ) =θ and cos(θ) = 1.

7.4 Transformation criteria

The transformation is calculated in an iterative way after the estimated transformation is computed. If there are no criterions configured this iteration can be go forever. When a criterion is met, the iterative loop is exited. The following criterions can be configured with the most algorithms.

7.4.1 Maximum number of iterations

Simply set the maximum number of iterations that is allowed for computing the algorithm. The number of iterations that is needed mostly depends on how far the two point clouds are away from each other.

7.4.2 Absolute transformation threshold

Stops the loop when the computed transformation in the loop is too far away from the estimated transformation. This because the two point clouds should be in each range. If the computed transformation is greater than this range the loop is exited.

7.4.3 Relative transformation threshold

This is the difference between a transformation calculated in the loop and the next one. If the difference between these transformations is small. The algorithm has converted and the loop is exited

7.4.4 Maximum number of similar transformations

For allowing to reach the algorithm a global minimum, several similar transformations are allowed. If this criterion would not exist, the relative transformation threshold would end the algorithm too fast. With this criterion, smaller minimums can be reached.

7.4.5 Relative mean square error

This one computes the mean square error between the translation / rotation increment between two transformations.

7.4.6 Absolute mean square error

Stops the transformation when the transformed source cloud is in a given range of the target cloud

7.4.7 Transformation validation

A transformation validation will check if the transformation is succeeded. This can be done by checking the distance between the point pairs after the transformation and give it a score. Keep in mind that this transformation check only can be computed on the overlapping section of the two point clouds.

8 Practical implementation

The practical implementation makes uses of the open source library PCL. This open source library is available for C++ and can work on any platform. A C++ compiler will be used to compile the code. A RGB-D camera is used to scan the environment. For fast processing a platform with CUDA is recommended. The used hardware is listed in Hardware.

8.1 PCL

The Point Cloud Library (or PCL) is a large scale, open project [1] for 2D/3D image and point cloud processing. The PCL framework contains numerous state-of-the art algorithms including filtering, feature estimation, surface reconstruction, registration, model fitting and segmentation. These algorithms can be used, for example, to filter outliers from noisy data, stitch 3D point clouds together, segment relevant parts of a scene, extract key points and compute descriptors to recognize objects in the world based on their geometric appearance, and create surfaces from point clouds and visualize them. (Rusu, 2011)

8.2 RGB-D camera

A RGB-D camera is a camera that consists out of an RGB camera and an infrared sensor (D= depth). With the RGB camera classical RGB images are captured. By making use of the infrared sensor an infrared image and depth image is captured. A depth image is an image where every pixel tells the distance between the corresponding objects captured in the scene and the viewpoint. By combining the data of the RGB image and the depth image a 3D colour point cloud can be constructed by using triangulation. A disadvantage of this type of camera is that transparent glass objects will not be detected by the infrared sensor. Figure 8:1 shows an example of a RGB-D-camera.

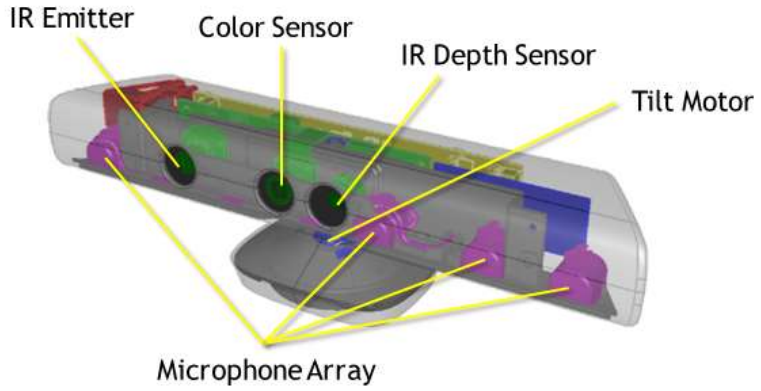


Figure 8:1 RGB-D camera sensors Construct the point cloud (Kinect for Windows Sensor Components and Specifications, n.d.)

Point cloud construction

The point cloud is constructed by applying triangulation between: the focal length, the values of the depth image and the image plane coordinates.

$$P_w = \begin{Bmatrix} x_w \\ y_w \\ z_w \end{Bmatrix} = world\ coordinates\ of\ point\ P$$

$$P_i = \begin{Bmatrix} x_i \\ y_i \\ z_i \end{Bmatrix} = image\ plane\ coordinates\ of\ point\ P$$

$$With\ "z_i"\ the\ distance\ value\ of\ the\ depthsteam$$

$$F = focal\ lenght$$

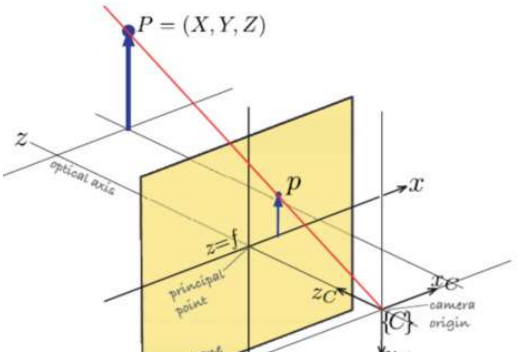
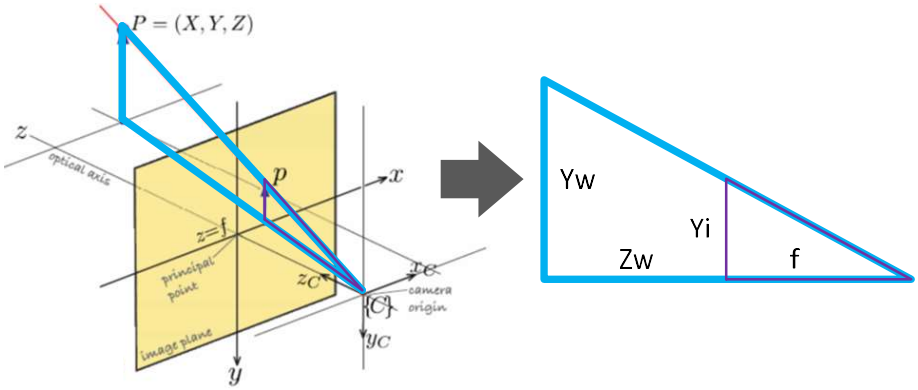


Figure 8:2 Image plane

Figure 8:2 shows a point in the real world and an image plane. The goal is to get the position of the point in the real world oriented from the image plane. The following pictures shows the triangulation that can be applied to get the X and Y coordinates.



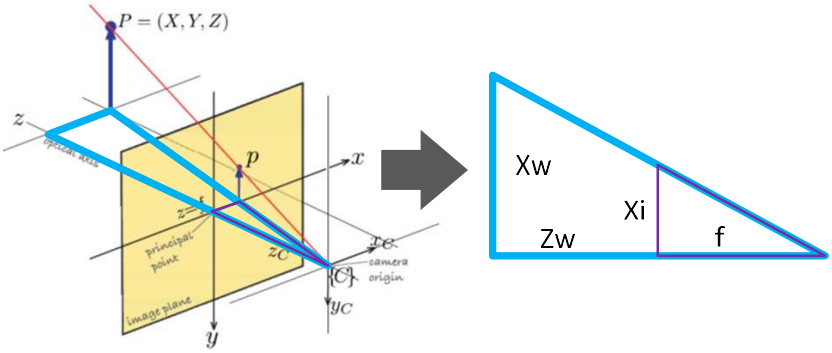


Figure 8:3 Image plane triangulation

Applying the triangulation from Figure 8:3:

$$Y_w = \frac{Y_I * Z_w}{f}$$

$$X_w = \frac{X_I * Z_w}{f}$$

(Collins, 2007)

With these two formulas, all the world coordinates oriented from the camera viewpoint can be calculated. The next step is to add the RGB values to the world coordinates. The image planes from the depth image and RGB image are aligned with each other after calibration and have the same resolution. In this way, it is easy to know which RGB pixel value belongs to a certain depth pixel. After combining all the data, the point cloud is created. PCL has a function to creates these point clouds and delivers them as a structured point cloud.

8.3 CUDA

With CUDA applied to the algorithm it will result in faster cycles times (4 times and more). Which is wanted in a real-time application. The current hardware did not support CUDA, therefore this isn't tested and no information is available to compare.

> CUDA® is a parallel computing platform and programming model invented by NVIDIA. It enables dramatic increases in computing performance by harnessing the power of the graphics processing unit (GPU). With millions of CUDA-enabled GPUs sold to date, software developers, scientists and researchers are finding broad-ranging uses for GPU computing with CUDA. (CUDA, 2017)

9 Practical Pipeline

Figure 9:1 shows the structure of the algorithm. Each part of the algorithm will be discussed in detail in the next paragraphs.

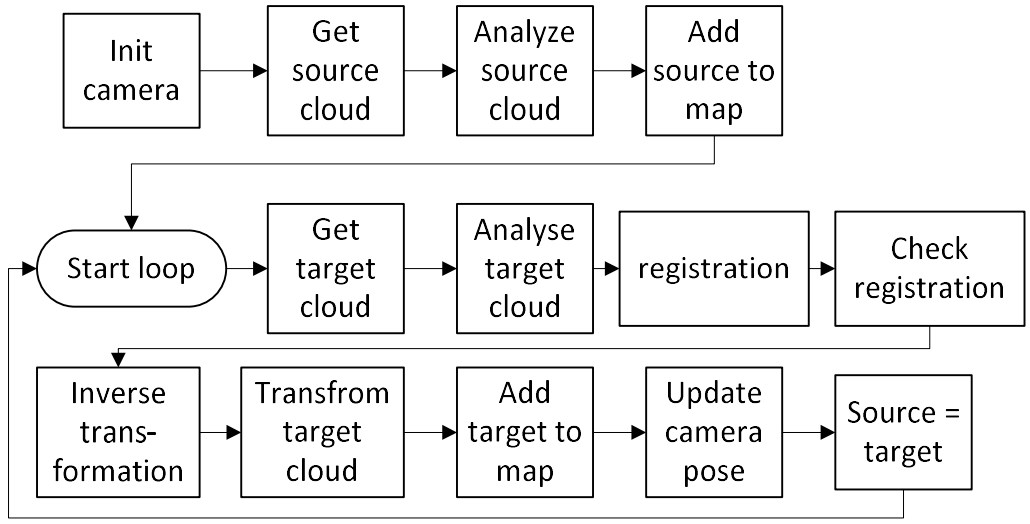


Figure 9:1 Practical registration pipeline

9.1 Init camera

The init camera is the first step that happens when executing the program. This is the initialization step for the camera. The program will try to connect with the RGB-D camera by making use of OpenNI. OpenNI is an open source SDK that allows the user to connect with 3D camera's. The next step is to initialize the 6D position of the camera in the digital point world. Since also to camera position will be kept in the 3D digital world. The initial camera position is zero in all values.

9.2 Get cloud

The get cloud step tells the algorithm which type of point cloud is requested. A colorless point cloud is used. Since no color is requested only the depth image is needed from the camera. Hence no calibration between the RGB image and depth image is needed. During the project, there were problems in receiving a RGB-point cloud in terms of speed. By asking for colorless point cloud half of the data is used. One RGB color point contains six data values (XYZ, RGB) while one colorless point only contains three data points (XYZ). The resolution that is used is 640*480 this will result in 307 200 points. Reducing half the data is an advantage with 307 200 points. By setting the receiving cloud to a colorless the max fps of the camera could be used (30 fps) for receiving data. When a RGB cloud was requested this resulted in receiving the clouds in no stable fps due too slow hardware, less than one fps.

9.3 Analyze cloud

When the colorless point cloud is received, the data will be analyzed. in this step: filtering, normal estimation and uniform sampling will be applied. For filtering a Bilateral filter is used. PCL provides a fast-bilateral filter that only works with organized point clouds. But since an organized point cloud is received from the camera this is no problem. The filtering is done to make the normal estimation more robust. PCL also provides a faster normal estimation function for organized point clouds. This function is called 'integral normal estimation' which is also used in the algorithm. After calculating the normal on the point cloud down sampling will be applied via uniform sampling. After the down sampling a point cloud with ± 3000 point normals is received. A point normal is a 3D point which contains 6 data types the location in space XYZ and the normal vector (XYZ). There has been chosen for uniform sampling because the application must be fast. During the project a test was made with detecting features (narf, harris and stiff) and calculating the feature description (FPFH). This tended to be more than ten times slower. Feature extraction will give more accurate results but is slower. Faster hardware is needed to do more testing with the feature extraction. Figure 9:2 shows the used methods for the cloud analyzing.

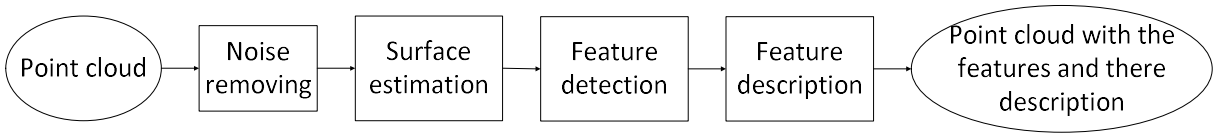


Figure 9:2 Analyse cloud flowchart

9.4 Registration

After analyzing the target cloud, the most important step of the algorithm is executed. In this step a transformation between the source and target cloud is estimated. The estimated transformation is a transformation to align the source cloud with the target cloud. The registration uses the down sampled clouds to make these calculations. Using a smaller number of points will reduce the cycles time. The flowchart is demonstrated in Figure 9:3. First the matching correspondences will be determined. After the correspondences are matched two rejection functions will take place. The first will reject all the correspondences based on the median distance. The resulting point pairs are filtered based on the surface normal angles. The transformation is estimated with filtered correspondences. At the end of the loop the converged criteria will check if one of the criteria's is met. if true then the algorithm is finished. If not the source cloud will be transformed with the current estimated transformation and the loop is repeated until a criterion is met.

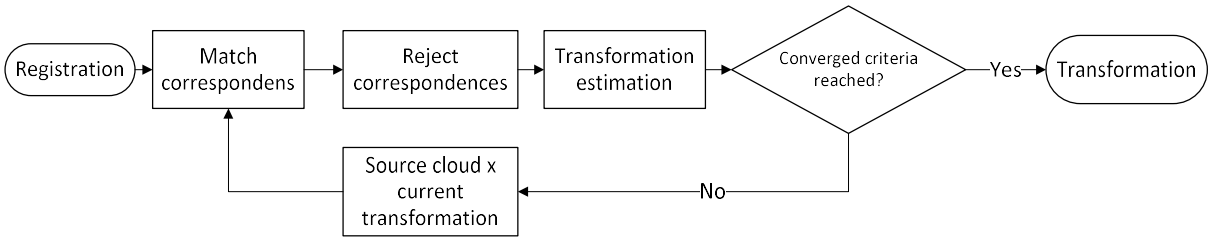


Figure 9:3 Registration flowchart

9.5 Check registration

After registration, a check will take place to determine a transformation score. The score is calculated based on the Euclidian distance of all the pair points after transformation. PCL has a function called 'TransformationValidationEuclidean'. But the result of this function is not trustworthy due unknow reason for this moment. Even if two identic point clouds get judged the score is still too low.

9.6 Inverse transformation

Like already told the registration will align the source cloud to the target cloud. The map building starts with the initial cloud, the source cloud. Hence all the clouds need to be aligned to the source cloud. To do this the final transformation will be inversed. With the inversed transformation, the target cloud can be aligned to the source cloud.

9.7 Transform target cloud

In 'transform target cloud', two clouds will be transformed. The initial target cloud with 307 200 points. This cloud will be later added to the 3D map (final cloud). The second cloud that needs to be transformed is target down sampled cloud with the normal vectors. This cloud is used as the source cloud for the next registration.

9.8 Update camera pose

The program keeps track of the camera's position during the registration. Since the two point clouds have a certain overlap the camera will move in space. By multiplying the previous position of the camera with the final transformation, the current position of the camera is obtained (Figure 9:4).

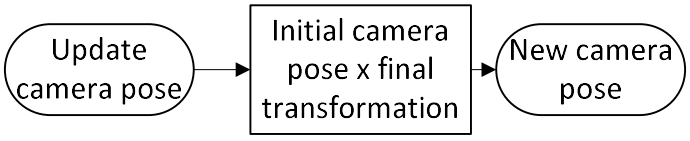


Figure 9:4 Update camera pose flowchart

9.9 Source = target

Like the working of a shift register, at the end of the loop the data of the down sampled target cloud will be copied to the down sampled source point cloud. When the next target cloud will be loaded, it will always be orient to the previous one by making use of the shift register. By using the shift register only one cloud needs to be analyzed every loop instead of two clouds. Which results in faster calculation speeds.

9.10 Add to map

9.10.1 Voxel grid filter

This process is showed in Figure 9:5 and works as followed. When a point cloud is analyzed and transformed to its relative position oriented from the previous cloud. It will be added to the final cloud which contains all the point clouds aligned with each other that are captured by the camera. Since there will always be an overlap between two point clouds. The redundant points must be removed from the final cloud to save data and creating a better visualization. A voxel grid filter is applied to the final cloud. The disadvantage of this method is that when more number of points are used, the filter will work longer. This problem can be solved by making the sure the final point cloud doesn't get too large. When the final cloud contains ten point clouds the final cloud will be saved on the operating system. Next the final cloud will be emptied and the process will be repeated. Another disadvantage is that certain data is lost when the voxel grid is applied. When the scene is registered all the save files can be analyzed in a point cloud viewer such as MeshLab.

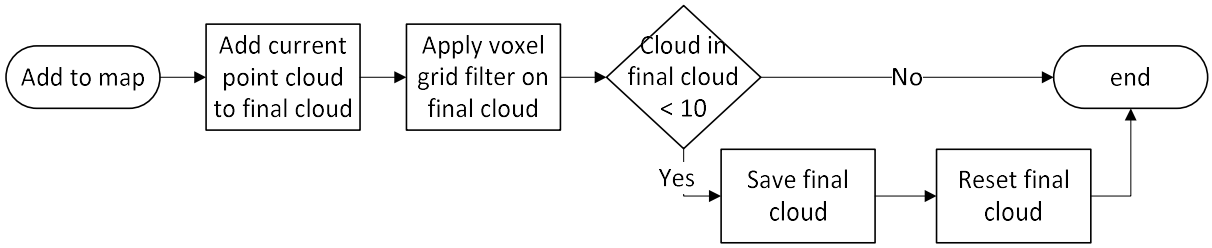


Figure 9:5 Map building voxelgrid filter flowchart

9.10.2 KD-Tree adding

Another method to build the map is by making use of a pass-through filter and a kd-tree. This method starts with calculating the minimum and maximum XYZ values of the transformed target cloud. After these values are calculated a pass-through filter is applied to the final cloud and the filtered data is stored in a temporary cloud. Next, a kd-tree is built from the temporary cloud. When the tree is build, a for loop will iterate through all the points in the transformed target cloud. In each iteration, a radius search in the kd-tree will be calculated on the iterated point. If no points are founded

in this radius search the point is added to the final cloud else the point is ignored and the next iteration will happen.

Pseudo code:

```
//get min and max point in transfomred target cloud
point3DMin =  transfTgtCloud.minPoint();
point3DMax =  transfTgtCloud.maxPoint();

//apply passthroug filter
tempcloud = passthroug(point3DMin,point3DMax,finalCloud);

//create KD tree from temp cloud
KdTree tree = new KdTree(tempcloud);

//iterate through all the points in the transformed target cloud
//do radius search in the tree on each iteration
int newPoints =0;
for(int i; i< transfTgtCloud.size(); i++){
   int points = kdTree.RadiusSearch(radius, transfTgtCloud[i])

   if(points = 0){
      finalCloud.addPoint(transfTgtCloud[i]);
      newPoints++;
   }
}

//estimate the overlap
double EstOverlap = newPoints/ transfTgtCloud.size() * 100;
```

This method provides a better visualization and no data is lost. A disadvantage is that this method is slower. The pass-through filter causes the kd-tree is built quicker, because less points are used to build the kd-tree. The overlap of the two point clouds can be estimated by the formula showed in the pseudo code. The calculation for the overlap highly depends on the accuracy of the transformation. If the transformation is computed wrong the estimated overlap is also computed wrong.

10 Results and speed

If the overlap between the two point clouds is large enough a successful registration is obtained. When the objects are close to the camera, the camera can move more through space instead of when the objects are further away. This Is due to that there always needs to be a certain overlap between the two pictures. The speed of the program depends on three main factors:

- System hardware
- Number of max iterations
- Number of points that are used during the registration algorithm

The test scene represents an apartment kitchen and the RGB picture is showed in Figure 10:1. The scene is registered in 10 different point clouds. Each point cloud is taken from a different viewpoint the single color point cloud is represented in Figure 10:2. For better understanding of the registration process each eight point clouds in a row gets a unique color (Figure 10:3). Different settings where used to test the algorithm and differ in:

- Max number of iterations
- Points used during registration algorithm

Figure 10:1 RGB image Kitchen scene

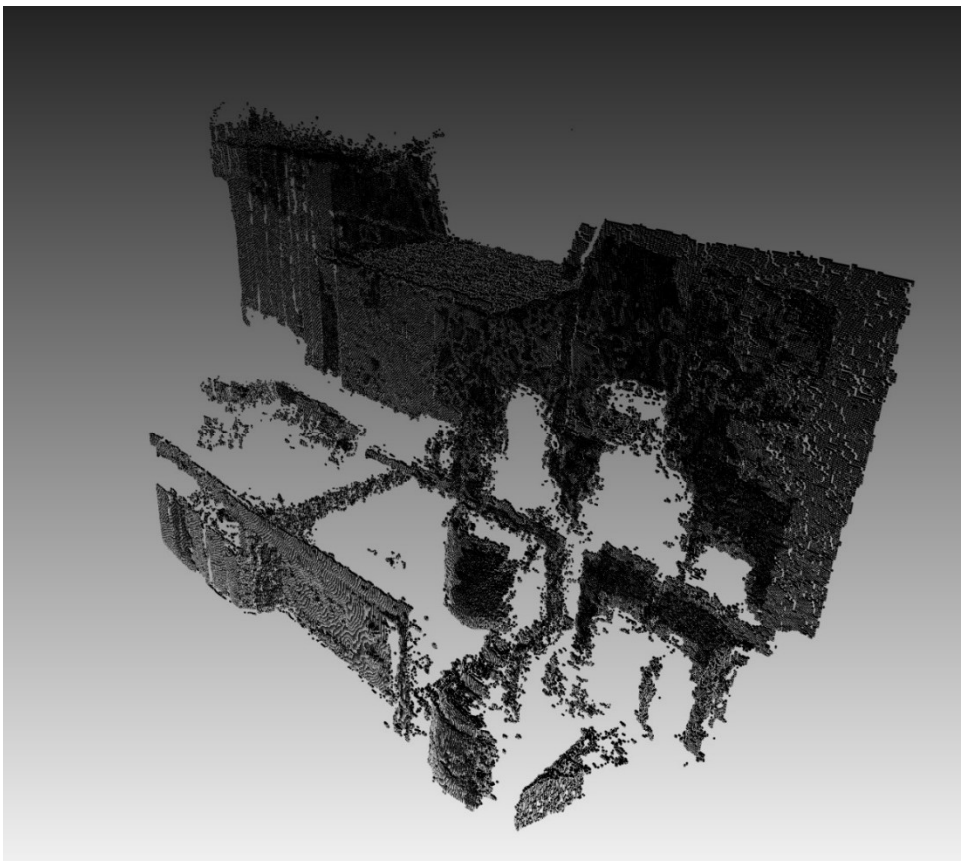


Figure 10:2 Point cloud kitchen scene

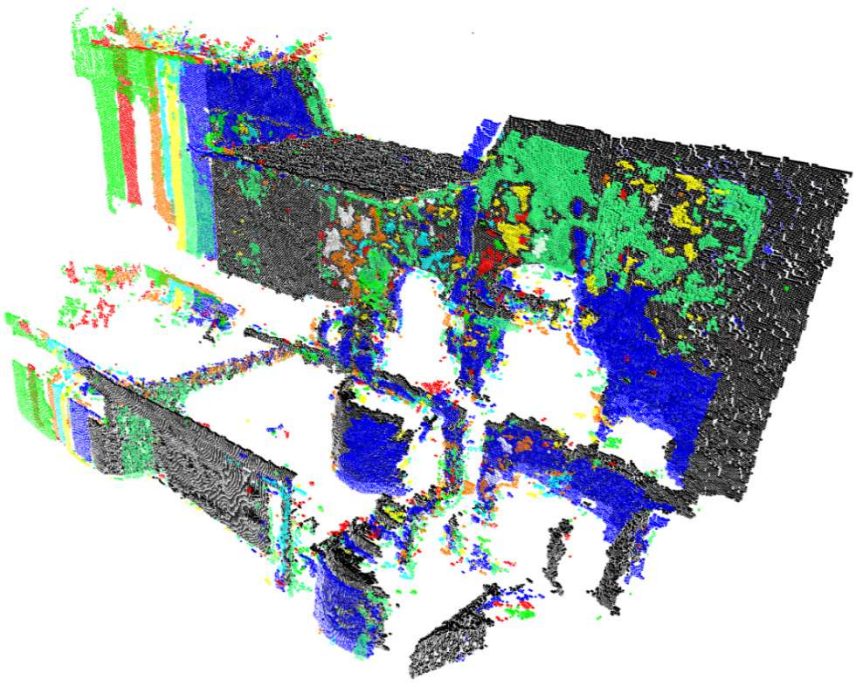


Figure 10:3 Color point cloud kitchen scene

10.1 Dataset 1

The number of the maximum iterations is set to 300. The radius search for down sampling the cloud is 50 mm. The radius has an influence on the number of points that are used during the registration. If the radius is smaller more points are used. The data can be found in Dataset 1

Conclusion

The functions to analyses the point cloud stay relative consistent over all the registrations (Figure 13:3). Figure 13:4 shows that these functions take more than 50% of the total cycles time. Keep in mind that the function “Calculate overlap” will not be used for the real practical implementation. This function is only used to analyse the data. The “get cloud” function is slow because it reads a point cloud file stored on the computer and not directly from the camera. This is done to use the same clouds for each test. When the file is directly received from the camera, the average speed to get the cloud is 5 ms. The map building time showed here is with the KD-tree map building time without a pass-through filter. Implementing the pass-through filter brings the map building time to 700 ms. This one is slower than the voxel grid filter but gives a better visual representation of the data and no data is lost. Following conclusions can be made for Figure 13:5:

- The number of registrations and time does not depend on the overlap of the clouds
- The registration time has the same progression as the number of iterations. Hence there is concluded that the time of registration mainly depends on the number of iterations

The number of points sampled from the point clouds is very consistent over each registration. Sometimes the algorithm finds more correspondences then points that are sampled from the point cloud. This is caused by duplicate correspondences. After rejection around 60% of the points are kept (Figure 13:6).

10.2 Dataset 2

The number of the maximum iterations is set to 50. The radius search for down sampling the cloud is 50 mm. The radius has an influence on the number of points that are used during the registration. If the radius is smaller more points are used. The data can be found in Dataset 2

Conclusion

The same conclusions from dataset 1 can be made. One big difference between the two data sets is the ICP time (registration time). In dataset 2 the average time from Table 4 is more than the half of the time in the first dataset. This is due to the smaller number of max iterations. The disadvantage of this setting is that the registration will happen less accurate. This can be seen in the alignment of the point clouds. Dataset 3 shows the different alignments with different settings.

10.3 Dataset 3

Dataset 3 compares the average time and quality from 10 registrations where the number of max iterations and the number of points used for registration are different. The larger the radius search the less points are used during the registration. Another aspect that is researched in this dataset is the quality of the registration. This is done by comparing the final point clouds in a visual way. This by checking the alignment of the walls and one object out of the scene. The object that is focused is in Figure 10:4. All the data and illustrations can be found in Dataset 3.

Figure 10:4 Object kitchen scene

11 Conclusion

The cycle time mainly depends on the (maximum) number of iterations and the number of points that is used in the algorithm. Where the number of iterations have more impact than number the points that is used. By making use of the top view and front view, the walls are well aligned in every test. Either details in the scene and small objects represent different in every outcome. By looking at the different registrations of the test object, there is concluded that no registration is perfectly aligned with zero error. Looking at the images, test 2 gives the best results for this scene. By comparing test 4 with test 2. There is concluded that with uniform sampling a certain number of points must be available to receive a fine registration. Test 4 and test 7 shows that at least 100 iterations are needed to reduce the error to a relative minimum. The shift register requires that every point cloud must be aligned perfectly. If cloud X is badly aligned with cloud Y all the following clouds are also aligned wrong oriented from cloud Y even if they are perfectly aligned with cloud X. Working with feature detection and the related feature describers could solve this problem. Workshop test shows a registration of a workshop in the school HUST. This test shows that the alignment error has more influence when more point clouds are used. During this test the algorithm crashed sometimes because it couldn't find any correspondences. The point clouds showed are the best results that are obtained with the algorithm. The settings of the algorithm where the same as in dataset 1. The workshop test consists of 165 point clouds and took a total of ± 200 seconds for registration and map building.

11.1 Cycles time

The best results that were obtained are a cycle times of: ±450 ms for the registration and ± 700 ms for the map building time. The total cycles time is still far from the real-time speed 100 ms or less that is required. Upgrading the hardware with a faster GPU and making use of CUDA will have a positive impact on the cycles time. Another solution is using a multi-threading program. The multi-threading program will split up the current algorithm in four different programs or threads that will work parallel with each other. The following flowchart (Figure 11:1) shows a structural way for multi-threading.

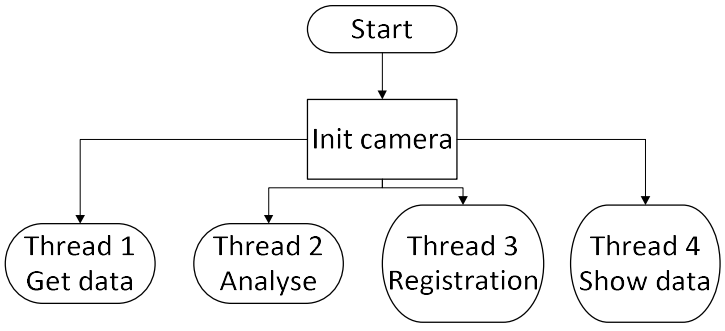


Figure 11:1 Multithreading flowchart

The multi-threading approach has as advantage that every function doesn't need to wait for the previous function. The data needed for every function will always be ready in time. Since the cycle time for thread 1 < thread 2 and so on.

11.2 Accuracy

The current accuracy for the registrations is too low for analyzing small objects in the scene. The accuracy can be improved by making use of features. But as already mentioned the feature detection is too slow on the current hardware. The use of features only makes a change in the analyzing of the point clouds. The features descriptions provide a better set of correspondences that can be used for the registration. Figure 11:2 shows a flowchart how the analyzing would look for feature based registration. Using feature descriptions can also help in improving the accuracy of registration multiple point clouds. More information can be find in Shift register.

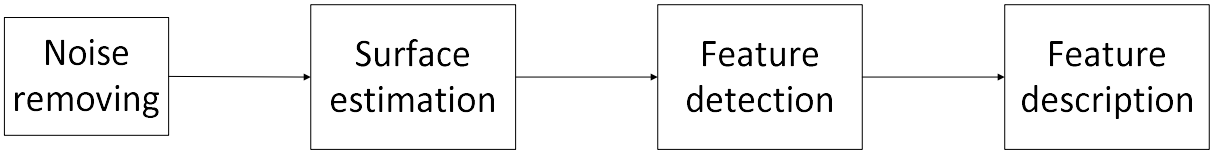


Figure 11:2 Feature analysing

A test with two build in registration algorithms in PCL ("IterativeClosestPoint" and "IterativeClosestPointWithNormals) has been tested to remove the error. The algorithms (Figure 11:3 & Figure 11:4) where tested in two different pipelines and all possible point clouds, the down sampled clouds and not down sampled clouds both with and without point normal, were used.

Algorithm 1:

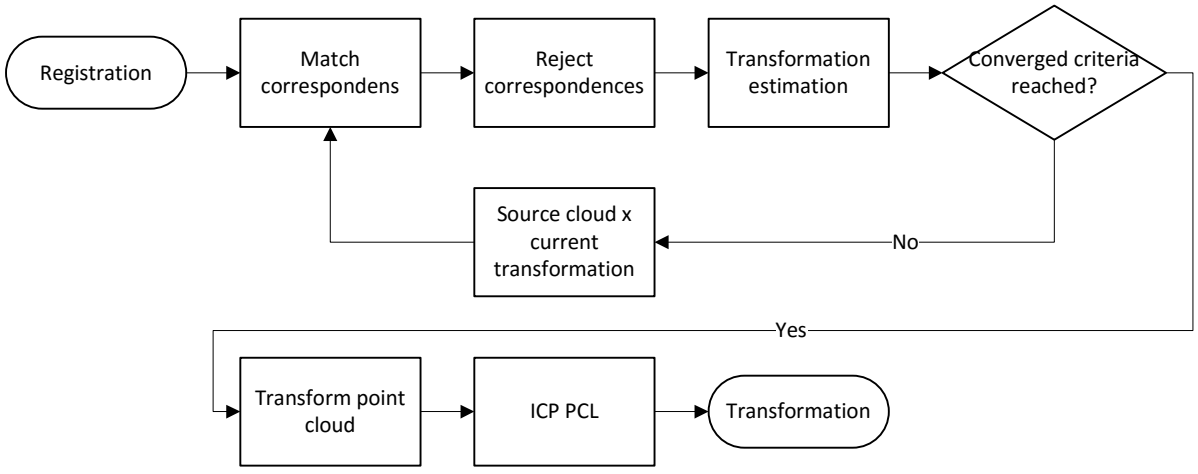


Figure 11:3 Improved accuracy algorithm 1

Algorithm 2:

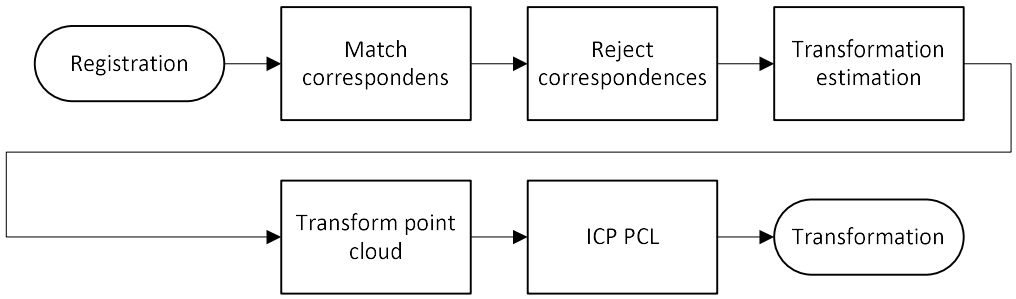


Figure 11:4 Improved accuracy algorithm 2

Regardless no success was booked by making use of PCL's ICP algorithm. The registration turned out to be even less accurate. The outcome was very similar for all possible combinations. In the algorithm 2 is showed by making use of the down sampled point normal clouds and "IterativeClosestPointWithNormals". shows the outcome of this test.

11.3 Shift register

The core function that's making possible multiple point clouds can be registered is a shift register. The advantage of this sift register is that it is fast and easy to implement. A disadvantage is that zero errors are allowed during the registration. As already mentioned, if cloud X is badly aligned with cloud Y all the following clouds will also be registered wrongly, oriented from cloud Y, even if they are perfectly aligned with cloud X. A registration error can occur when the overlap between the two point clouds is too small or in other words when viewpoint 1 is too far away from viewpoint 2 or the difference in orientation between the two viewpoints is too large. A false registration will occur. During the tests, it showed that each time when a false transformation was estimated. The estimated camera position was very far in space from the previous camera position. Hence by using the estimated camera position a bad registration can detected. By calculating the translation and rotation between the two camera positons a safety mechanism based on too much camera movement can be incorporated. When absolute values of the rotation and translation are greater than the threshold value. The cloud will not be added to the final cloud. When the rotation value and translation value are in range, the cloud will be added to the final cloud. This feature has been tested and resulted in a positive outcome. One disadvantage of this feature is to get the camera position back close enough to the previous position. Which is hard for a human but easy for a robot.

Another implementation for aligning multiple point clouds is using features. During the registration, a list of all features and their position in the real world will be kept. With each new registration, the algorithm will search through this list and find

common features. If no features are found the camera moved too much and the algorithm will be “paused” until common features are found. The power of the feature list is that not only the last point cloud will be used to make the alignment but all the features that are within the range of the current viewpoint. A pass-through filter and a kd-tree can speed up the search through the feature list. Due too slow hardware, this second approach couldn’t be tested. Although the core code for this extension is provided.

11.4 Decrease degrees of freedom

Decreasing the degrees of freedom of the camera will result in better accuracy and speed. For example, to fix the camera’s Y position is an option that is used for room scanners this is showed in Figure 11:5. By fixing the camera’s Y position the estimation of the transformation will know that there will be no movement in Y. Due this fix the number of unknows is decreased to four. Since no rotation on the z-axis is also performed with this fix.

Figure 11:5 Fixed Y position scan (Feifer, 2012)

11.5 External sensors

External sensors can be used to keep track of the camera’s position. The current IMU’s (Integrated motion unit) provide reliable measurements. The data of the IMU can be combined with the data of the estimated transformation by the registration algorithm. The next step is to implement a probabilistic method that estimate the current position of the camera.

12 References

A search algorithm*. (2017, May 12). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/A*_search_algorithm

Bashan, Y. (2015). *http://webee.technion.ac.il*. Retrieved from http://webee.technion.ac.il: http://webee.technion.ac.il/cgm/Computer-Graphics-Multimedia/Undergraduate-Projects/2014/FacialKeypointsDetection/ProjectWeb/

Collins, R. (2007). *Camera projection*. (Penn State) Retrieved April 2017, from http://www.cse.psu.edu/~rtc12/CSE486/lecture12.pdf

CUDA. (2017, May). Retrieved from NVIDIA: http://www.nvidia.com/object/cuda_home_new.html

Dirk Holz, A.-E. I. (2015). *IEEE Robotics and Automation Magazine.* Retrieved March, April, May 2017, from http://lgg.epfl.ch/~ichim/registration_tutorial_ram_2015/data/registration_paper_tutorial_ram_2015.pdf

Farzana, E. (2011, September 30). *Image Denoising : Adaptive Bilateral Filter*. Retrieved from Esmat Farzana: https://esmat-farzana.page4.me/65.html

Feifer, J. (2012, August 8). *Fast Company*. Retrieved from Fast Company: https://www.fastcompany.com/3000044/how-create-quick-affordable-3d-renderings-rooms-if-it-was-doable

Fleming, N. (2011, September 30). *MIT Technology Review*. Retrieved from MIT Technology Review: https://www.technologyreview.com/s/425596/high-fidelity-3-d-images-created-using-kinectfusion/

François Pomerleau, F. C. (2015 May). *A Review of Point Cloud Registration Algorithms for Mobile Robotics.* NOW.

Hanoi University of Science and Technology. (1985). (Hanoi University of Science and Technology) Retrieved May 2017, from https://sep.hust.edu.vn/home

Joe Ahuja, I. C.-J. (n.d.). *Eye-Tracking Robotic Arm*. (University of Leicester - Bioengineering Group) Retrieved from Eye-Tracking Robotic Arm: https://robotarm.jamesreuss.co.uk/#/step-56

K-d tree. (2017, February 20). (Wikipedia) Retrieved March 2017, from https://en.wikipedia.org/wiki/K-d_tree

Kinect for Windows Sensor Components and Specifications. (n.d.). Retrieved from Microsoft Developer Network: https://msdn.microsoft.com/en-us/library/jj131033.aspx

Lague, S. (2014, December 16). *A* Pathfinding (E01: algorithm explanation)*. Retrieved from Youtube: https://www.youtube.com/watch?v=-L-WgKMFuhE

Libpointmatcher. (2006). (ETHZ ASL) Retrieved March, April, Mai 2017, from https://libpointmatcher.readthedocs.io/en/latest/

Pascal Willy Theiler, K. S. (2012). *ETH Zürich*. Retrieved from Automatic registration of partially overlapping terrestrial laser scanner point clouds: https://www.ethz.ch/content/specialinterest/baug/institute-igp/photogrammetry-and-remote-sensing/en/research/completed_projects/automatic_registration_of_point_clouds.html

Patel, A. (2017, April 18). *Amit's A* Pages*. Retrieved from Red blob Games: http://theory.stanford.edu/~amitp/GameProgramming/index.html

Rusu, R. B. (2011, March). *PCL (Point Cloud Library)*. (Open Source) Retrieved March, April, Mai 2017, from http://pointclouds.org/

Steder, B., Bogdan, R., Konolige, K., & Wolfram, B. (2010). *NARF: 3D Range Image Features for Object Recognition.* Retrieved March 2017, from https://pdfs.semanticscholar.org/e070/1662a370622a1cdbb7c6a83bbede3d0e6c23.pdf

13 Attachments

Attachment A: A* algorithm

The A* algorithm is a path planning algorithm that is an extension of Esdger Dijkstra's algorithm in 1995. The algorithm finds the most optimal path between two points if it exists. To use the algorithm a node map (grid map) must be available of the world. A node map is a map where the world is divided in nodes as shown in Figure 13:1 Node map example (Patel, 2017).

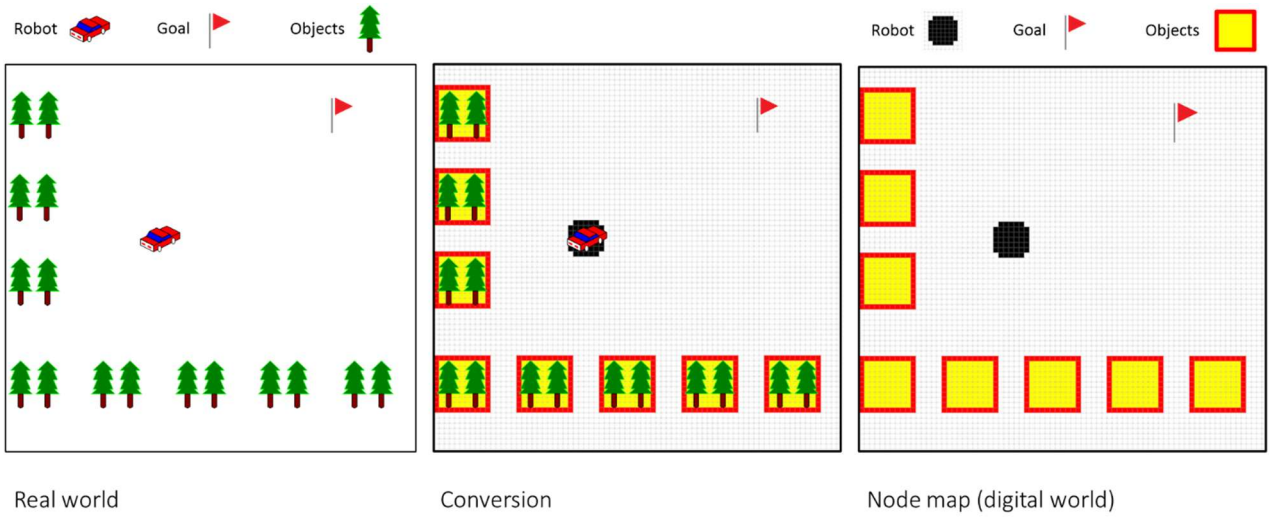


Figure 13:1 Node map example

When the node map is built. Each node receives three values:

- G-cost: the cost from the current node to the start node (robot start position);
- H-cost: the cost from the current node to the end node (goal);
- F-cost: G cost + H cost;

With these three values the most optimal path is calculated. To determine which nodes the robot will travel, the algorithm chooses the node with the lowest F-cost. If there are multiple nodes with the lowest F-cost, the algorithm chooses the node with the lowest G cost. If there are multiple nodes with lowest F-cost and G-cost, it chooses the first one it finds in the list. The algorithm keeps track of all the nodes with two lists:

- Closed list: Nodes that are evaluated
- Open list: the surrounding nodes (neighbor nodes) of the nodes in the closed list;

The distance between each node equals 10. This means that the costs for moving vertical and horizontal is 10. The cost to move diagonal is 14 ($\sqrt{10^2 + 10^2}$). To test this algorithm a simulation is programmed in C# WPF. This simulation contains a second feature where a grid map is build where the A* algorithm will navigate in. The grid map and the A* algorithm contains a function where both can be updated during the path execution. This means that objects are added to the map during the path

execution also the path is adjusted if needed. The simulation delivered positive results and works well.

A.1: Pseudo code

```
OPEN //Create the list for the set of nodes to be evaluated
CLOSED // Create the list for the set of nodes already evaluated
add the start node to OPEN
loop
    current = node in OPEN with the lowest f_cost
    remove current from OPEN
    add current to CLOSED
    if current is the target node //path has been found
        return
    foreach neighbor of the current node
        if neighbor is not traversable or neighbor is in
CLOSED
            skip to the next neighbor
        if new path to neighbor is shorter OR neighbor is
not in OPEN
            set f_cost of neighbor
            set parent of neighbor to current
            if neighbor is not in OPEN
                add neighbor to OPEN
```

A.2: Example

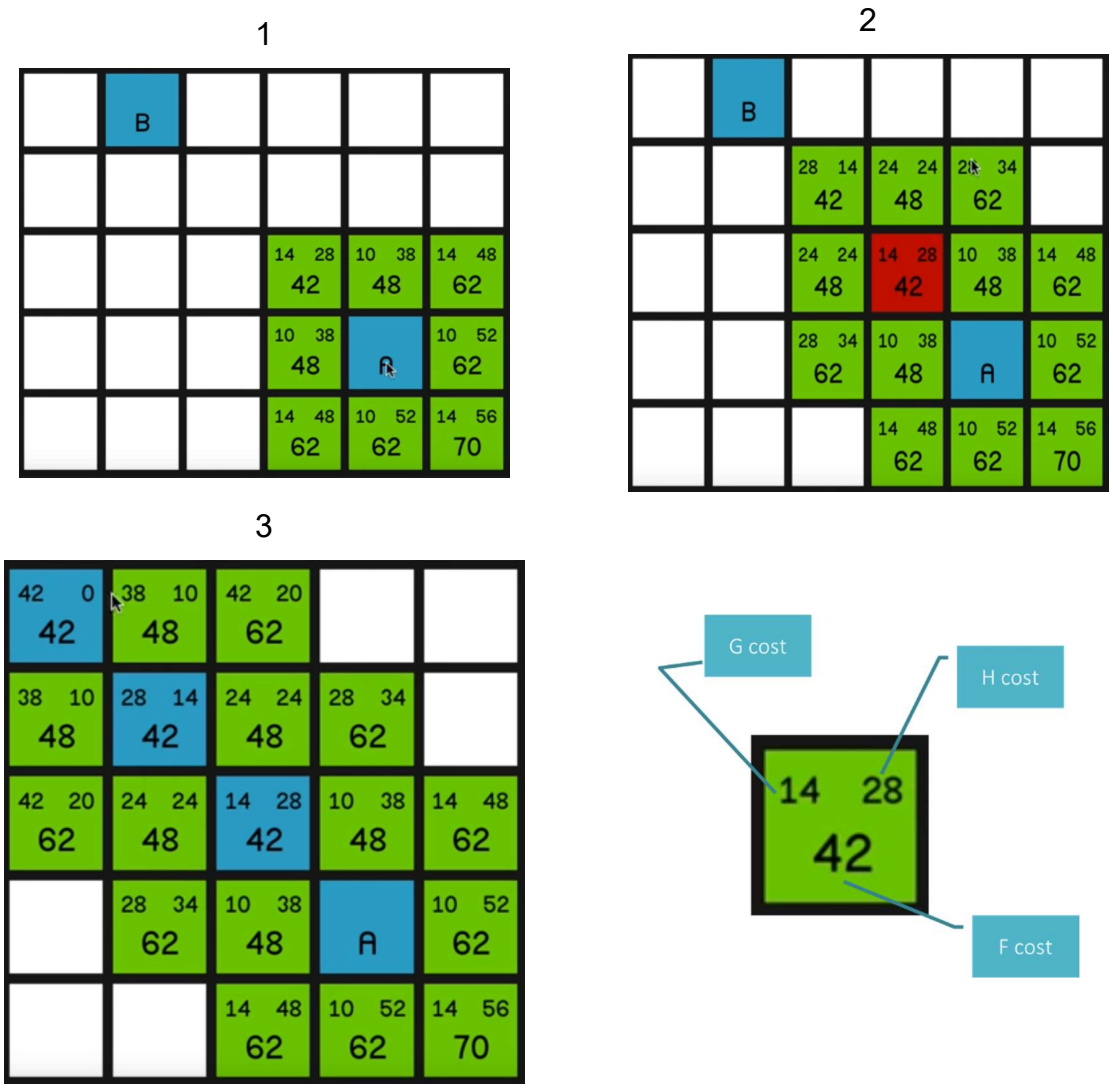


Figure 13:2 A* example (Lague, 2014)

Attachment B:Hardware

- Platform
  - Windows 10 pro;
- Computer
  - Windows surface pro;
    - 64 bit;
    - Intel core I7-665O U CPU @ 2.20 Ghz up to 3.4 GHz, 4 MB cache
    - i7: Intel Iris 540 graphics;
    - 16 GB DDR3 Ram memory;
    - SSD memory;
- RGB-D camera
  - ASUS Xtion PRO LIVE;
    - Field of view;
      - 58° Horizontal;
      - 45° Vertical;
      - 70° Diagonal;
    - RGB camera;
    - Infrared sensor (depth image);
      - VGA (640x480) : 30fps;
      - QVGA (320x240): 60fps;
      - Detection range: between 80cm and 3,5 m;
  - 2 microphones;
- PCL 1.8;

Attachment C:Dataset 1

Table 3 Functions overview

	Registrations									
Function	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Read cloud [ms]	1730	1729	1677	1661	1651	1681	1558	1555	1548	1544
Starting registration extraction program	0	0	0	0	0	0	0	0	0	0
Convert xyzRGBa cloud to xyz cloud [ms]	2	2	3	2	2	2	3	3	2	2
Fast bilateral filter [ms]	88	62	69	80	73	72	63	67	65	67
Integral normal estimation [ms]	43	40	42	48	49	46	46	40	43	41
Create point Normal cloud [ms]	5	7	7	6	6	7	6	5	7	6
Number of points after down sampling	1034	1049	1080	1087	1095	1052	1083	1145	1153	1193
Normal space sampling [ms]	16	15	17	15	18	15	16	14	14	15
Number of 'all correspondences'	1037	1034	1049	1080	1087	1095	1052	1083	1145	1153
Number of 'good correspondences'	695	702	586	662	679	661	663	667	667	706
Registration iterations	300	74	300	300	300	174	24	19	300	300
Registration ended with	1	2	1	1	1	4	2	2	1	1
ICP [ms]	1257	284	1089	1204	1166	679	93	74	1086	1294
Transformation score	0	0	0	0	0	0	0	0	0	0

Update map [ms]	2197	1825	1285	1554	1747	1655	1489	1510	1580	1525
estimated Overlap [%]	99.52%	99.33%	67.74%	79.11%	92.61%	90.20%	93.38%	95.27%	94.79%	85.17%
Calculate overlap [ms]	1991	1783	1219	1443	1576	1474	1469	1616	1507	1391
Analyse & ICP [ms]	1410	410	1226	1354	1314	821	226	202	1217	1424
Total calculation time [ms]	7327	5746	5407	6012	6288	5631	4742	4883	5851	5884
Average analyse & ICP [ms]	960.2549									
Average total calculation time [ms]	5777.0893									

Legend
Time Analyze & ICP functions [ms]
Time other functions [ms]
Total time [ms]
Other information

Transformation ended with	
0	CONVERGENCE_CRITERIA_NOT_CONVERGED
1	CONVERGENCE_CRITERIA_ITERATIONS
2	CONVERGENCE_CRITERIA_TRANSFORM
3	CONVERGENCE_CRITERIA_ABS_MSE
4	CONVERGENCE_CRITERIA_REL_MSE
5	CONVERGENCE_CRITERIA_NO_CORRESPONDENCES

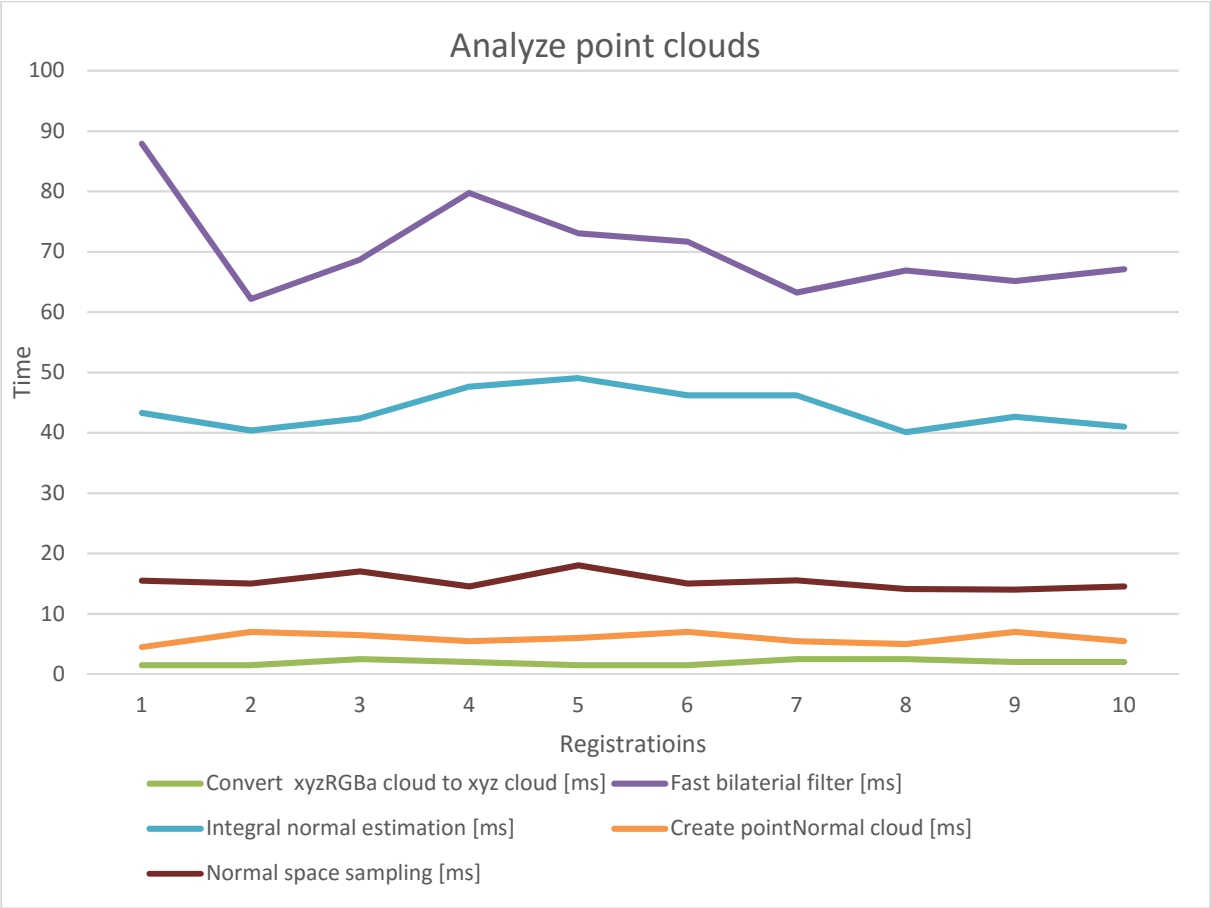


Figure 13:3 Analyze functions

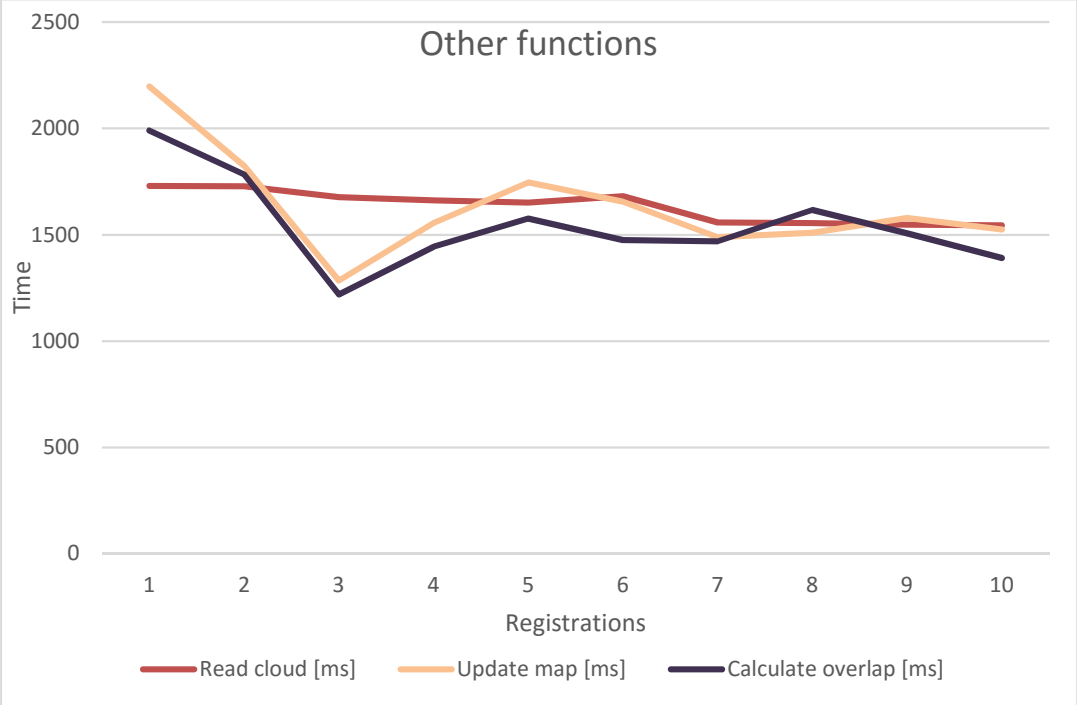


Figure 13:4 Other functions

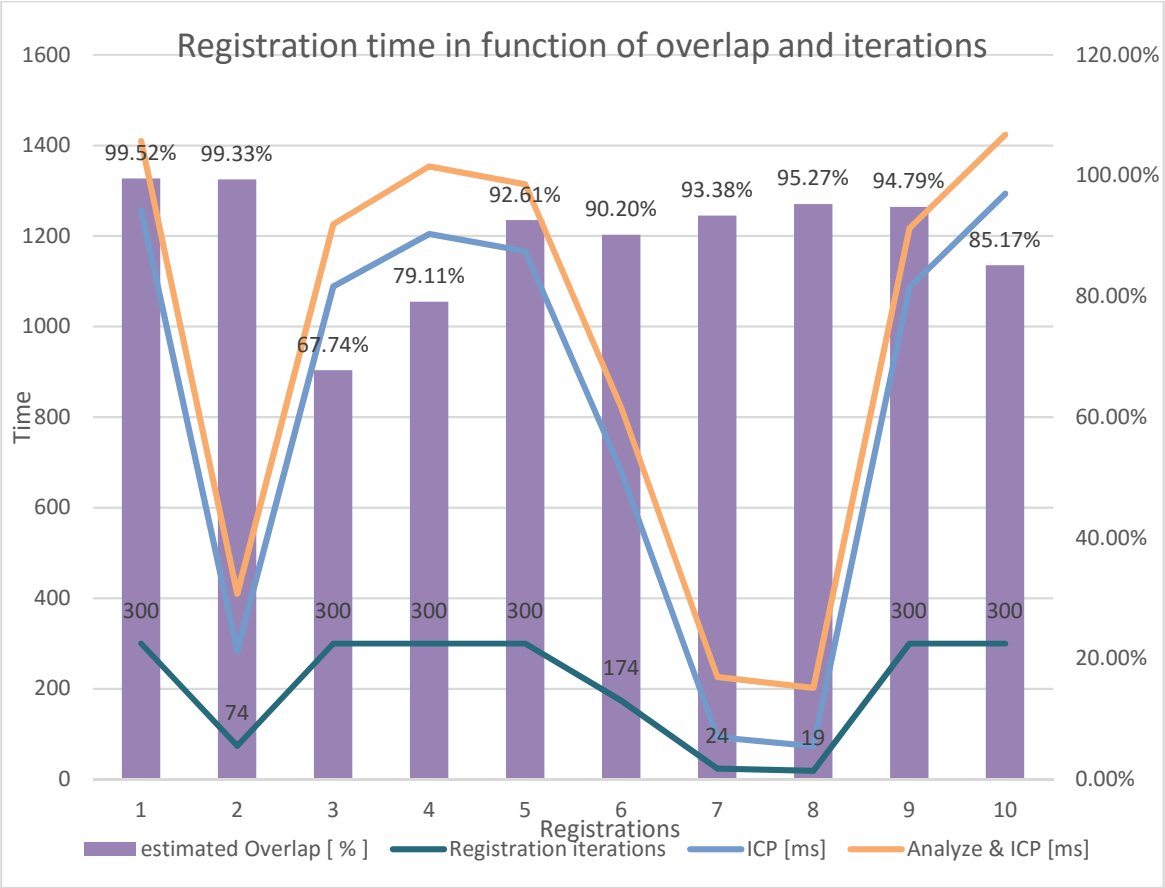


Figure 13:5 ICP

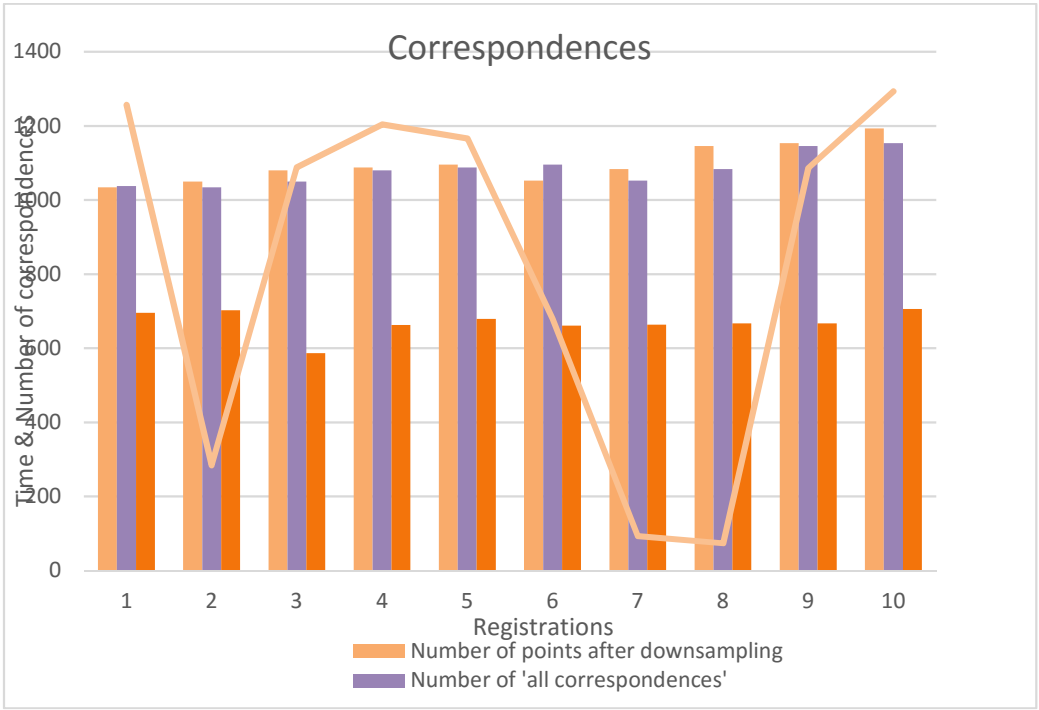


Figure 13:6 Correspondences

Attachment D:Dataset 2

Table 4 Functions overview

	Registrations									
Function	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Read cloud [ms]	2387	2268	1900	1675	1681	1689	1621	1556	1543	1502
Starting registration extraction program	0	0	0	0	0	0	0	0	0	0
Convert xyzRGBa cloud to xyz cloud [ms]	0	4	2	3	2	0	2	2	2	17
Fast bilateral filter [ms]	139	73	71	88	69	85	82	72	79	70
Integral normal estimation [ms]	65	40	46	50	44	45	45	27	42	39
Create point Normal cloud [ms]	8	7	7	0	10	6	0	6	6	0
Number of points after down sampling	1034	1049	1080	1087	1095	1052	1083	1145	1153	1193
Normal space sampling [ms]	20	9	16	28	18	16	28	6	15	8
Number of 'all correspondences'	1037	1034	1049	1080	1087	1095	1052	1083	1145	1153
Number of 'good correspondences'	692	700	587	654	677	666	663	663	558	693
Registration iterations	50	9	50	50	40	50	17	13	50	50
Registration ended with	1	2	1	1	2	1	2	2	1	1

ICP [ms]	267	50	237	269	172	228	80	85	233	240
Transformation score	0	0	0	0	0	0	0	0	0	0
Update map [ms]	2310	1805	1274	2462	2077	1713	1608	1734	1711	1604
estimated Overlap [%]	99.52%	99.33 %	61.76 %	79.22 %	93.07 %	90.38 %	93.38 %	95.24 %	70.24 %	84.14 %
Calculate overlap [ms]	2310	1733	1111	1558	1641	1401	1489	1576	1255	1310
Analyse & ICP [ms]	499	183	379	438	314	379	237	197	375	373
Total calculation time [ms]	7507	5988	4664	6132	5714	5181	4956	5062	4885	4790
Average analyse & ICP [ms]	337.33									
Average total calculation time [ms]	5487.91									

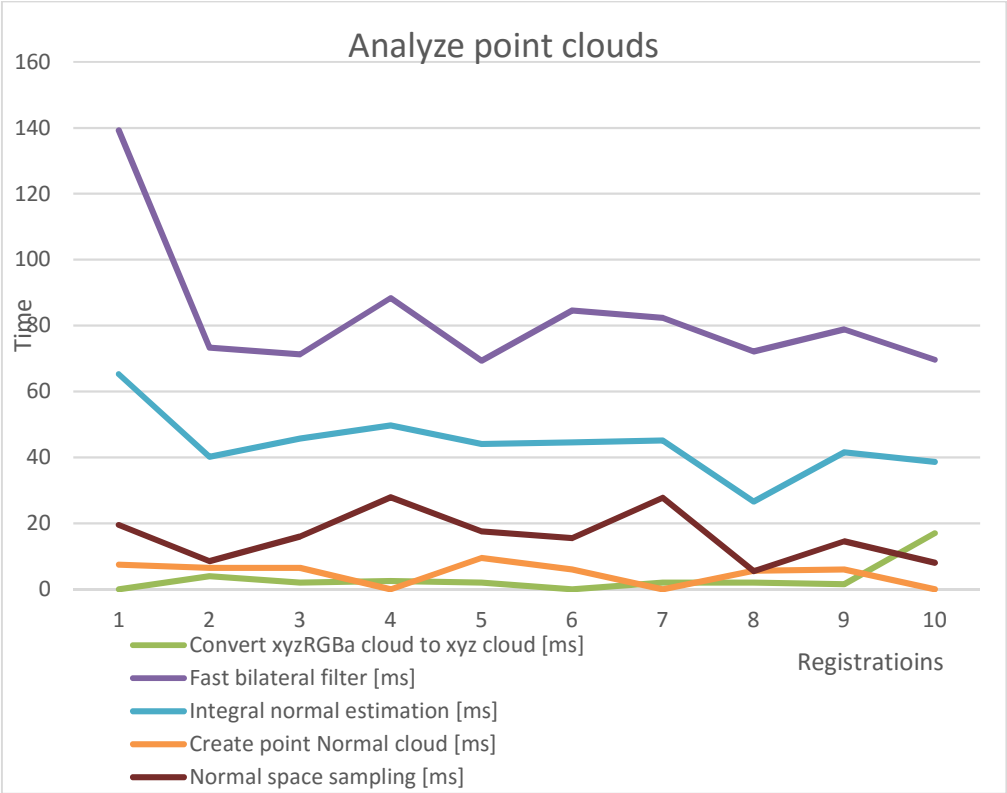


Figure 13:7 Analyze functions

Figure 13:8 Other functions

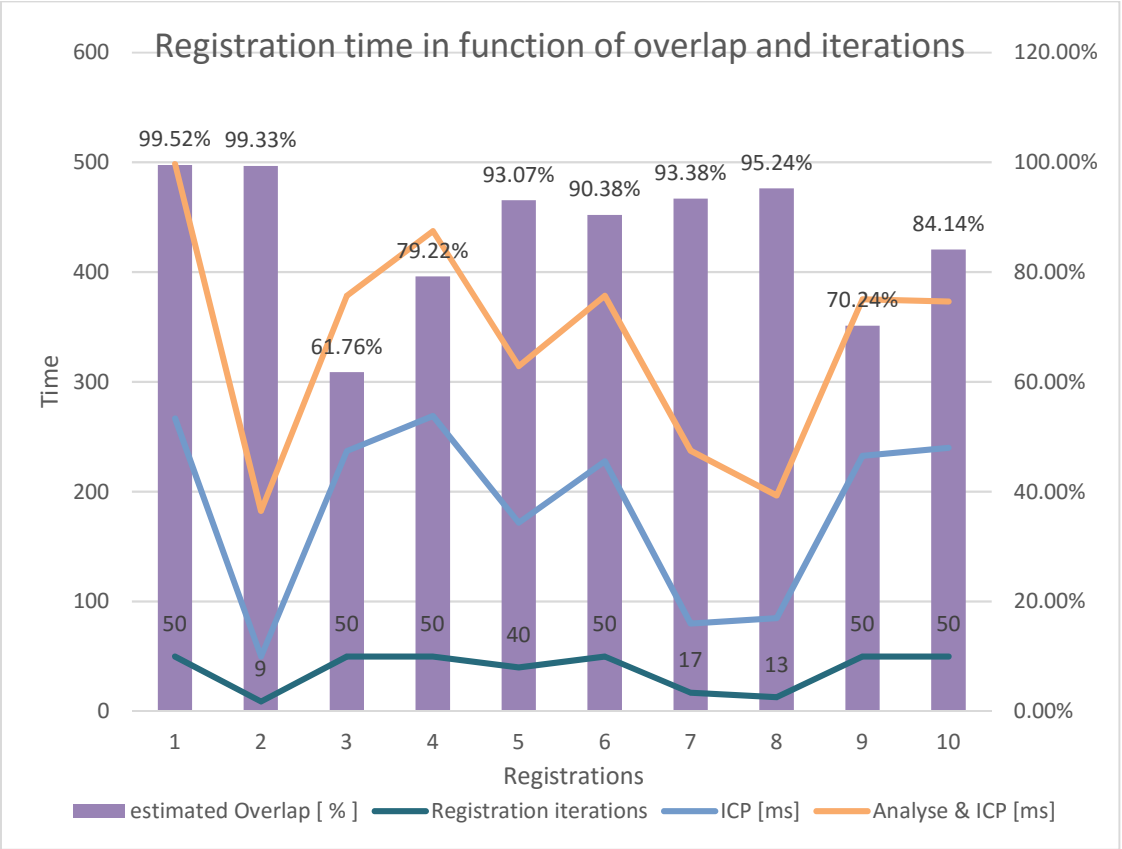


Figure 13:9 ICP

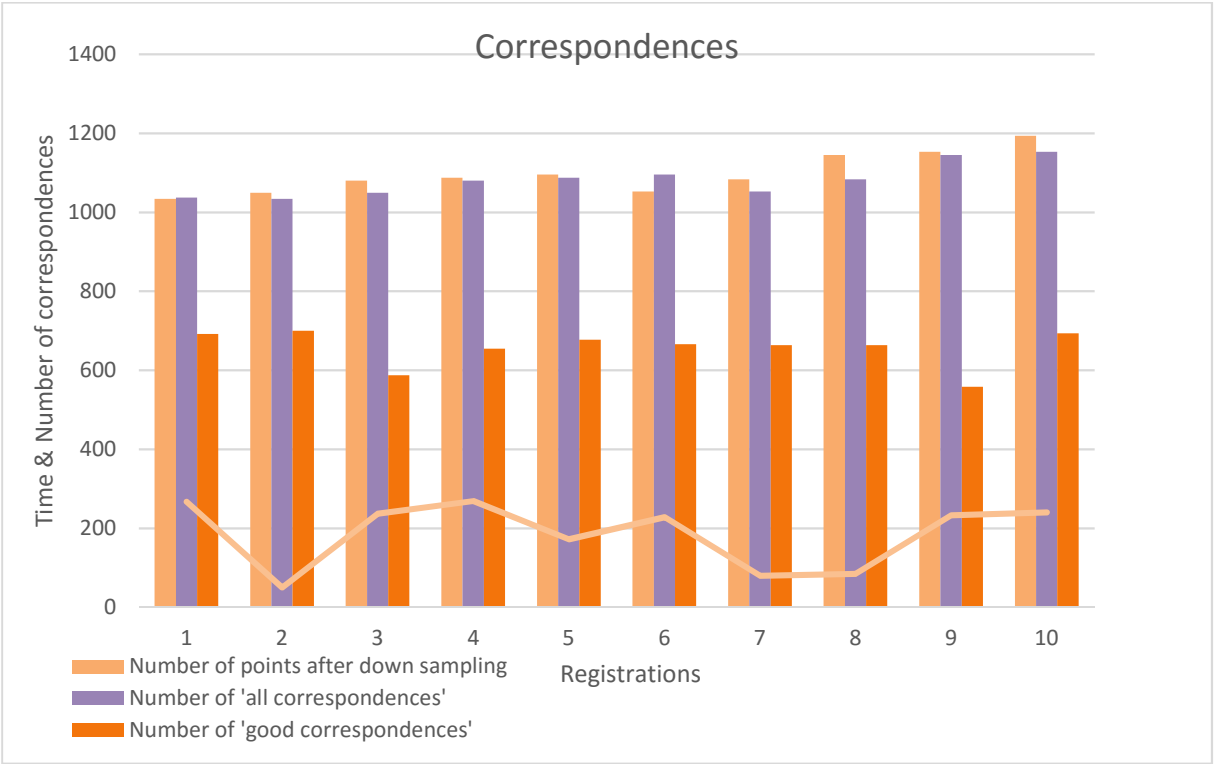


Figure 13:10 Correspondences

Attachment E:Dataset 3

Table 5 ICP

test	Max iterations	Radius search	Average analyze & ICP [ms]
1	300	0.05	960.2549
2	100	0.05	422.5422
3	50	0.05	337.3303
4	2000	0.1	2146.1284
5	100	0.1	253.7892
6	100	0.15	194.641
7	50	0.15	186.2641

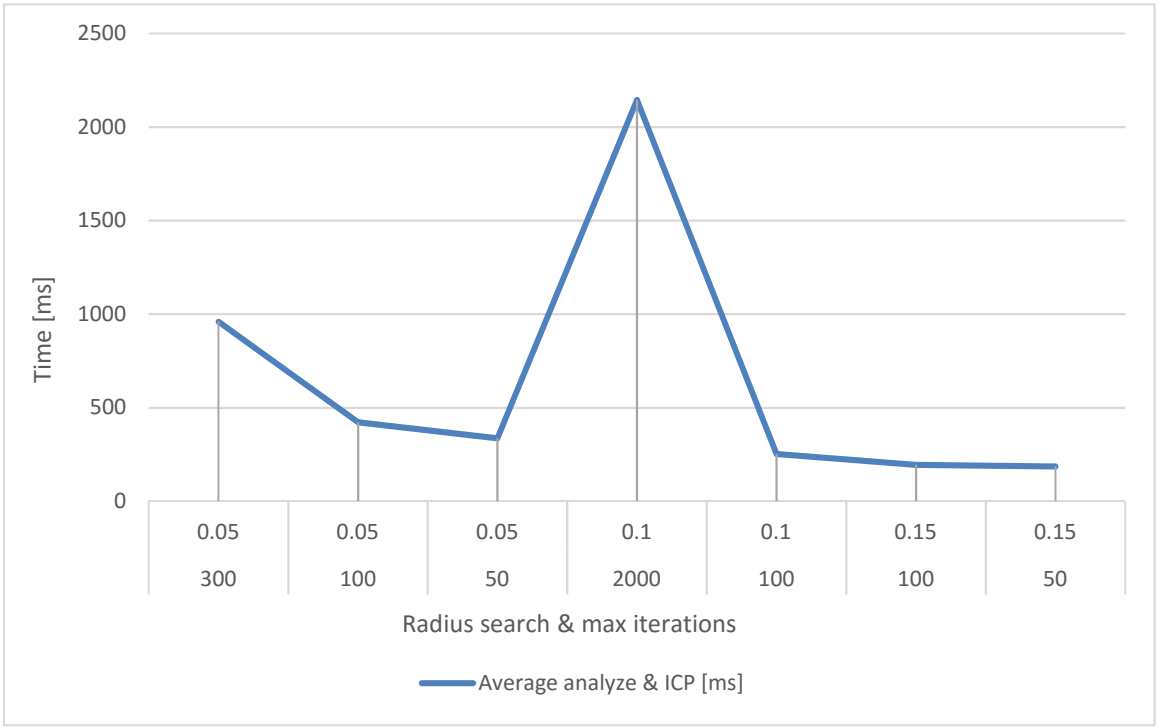


Figure 13:11 ICP

E.1: Test 1

Front view:

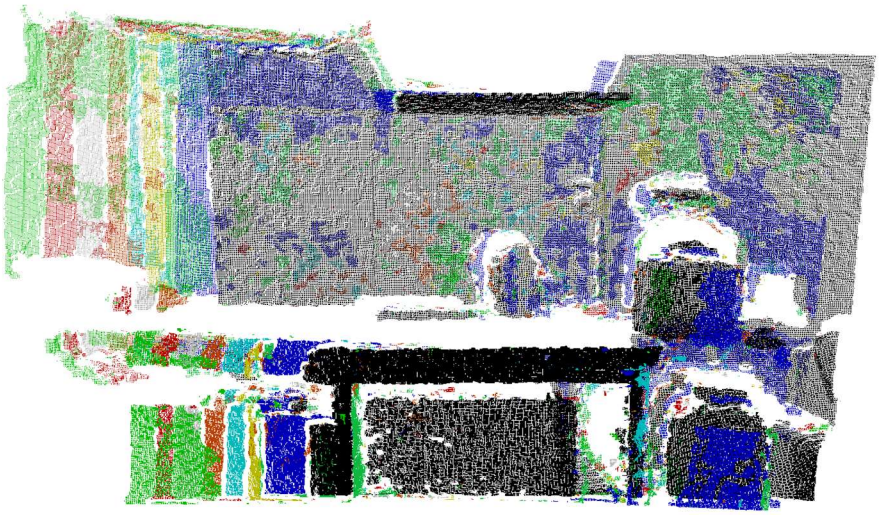


Figure 13:12 Front view

Top view:

Figure 13:13 Top view

Object in scene:

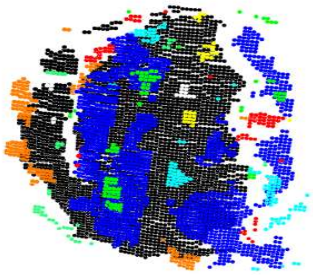


Figure 13:14 Object view

E.2: Test 2

Front view:

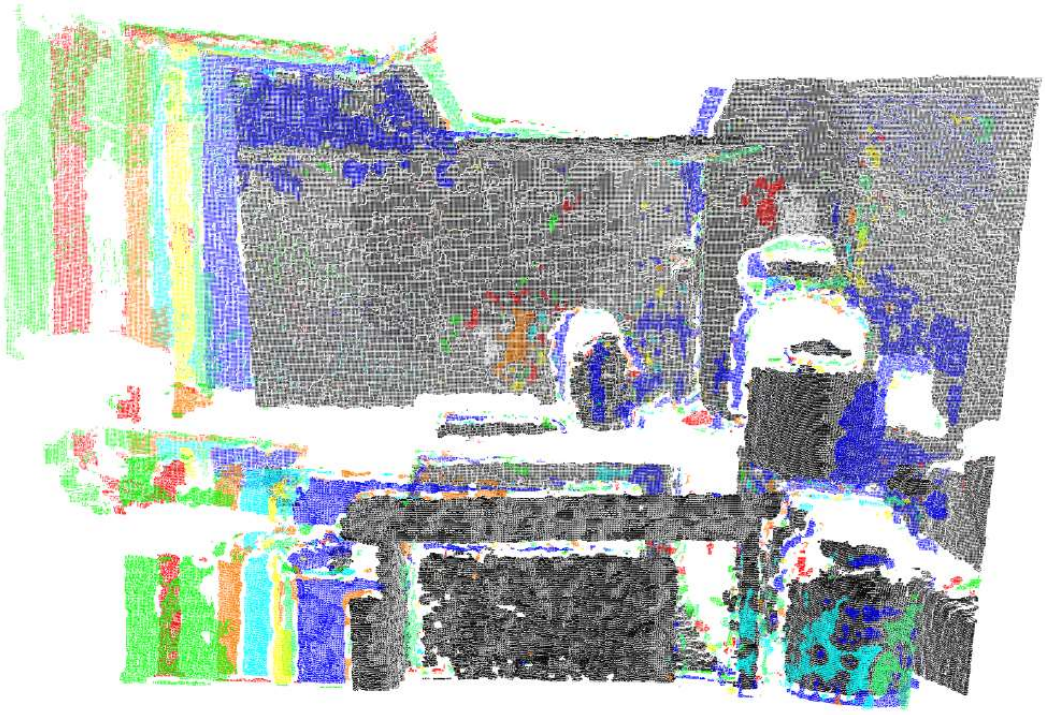


Figure 13:15 Front view

Top view:

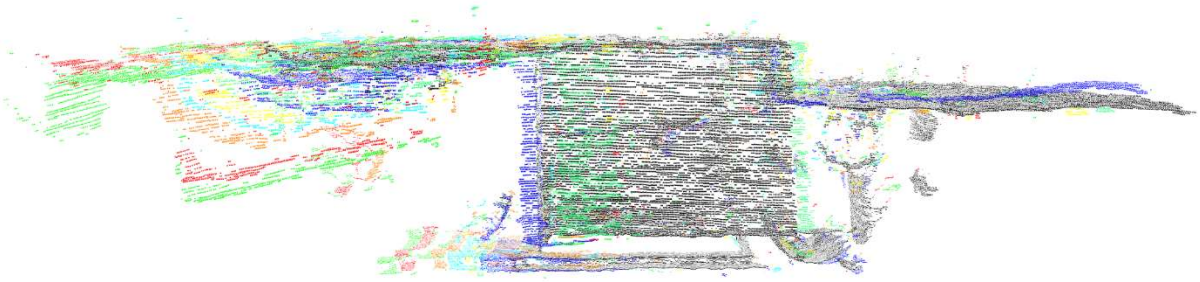


Figure 13:16 Top view

Object in scene:

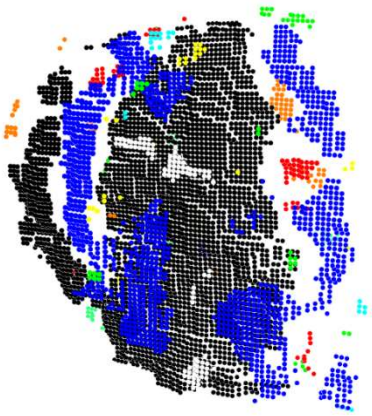


Figure 13:17 Object view

E.3: Test 3:

Front view:

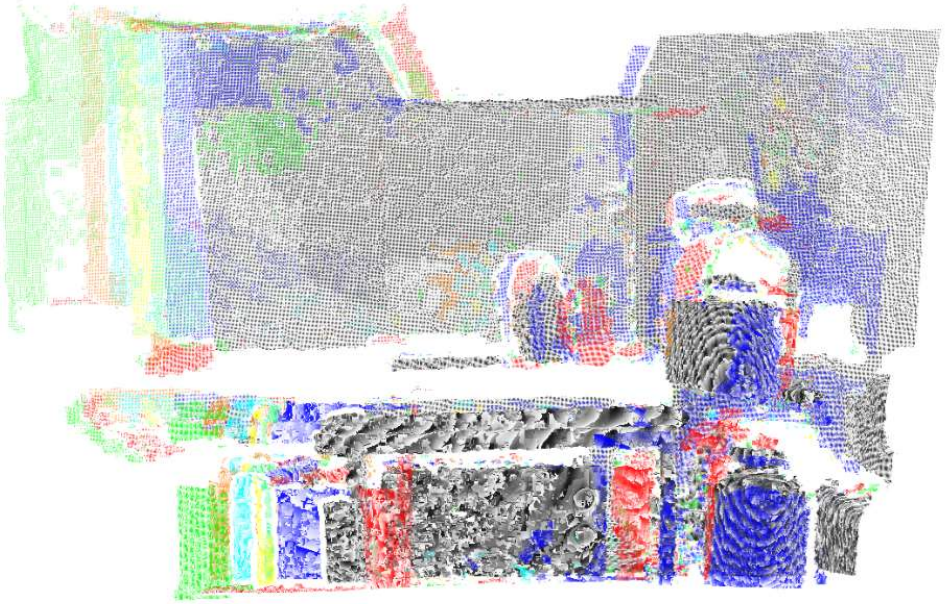


Figure 13:18 Front view

Top view:

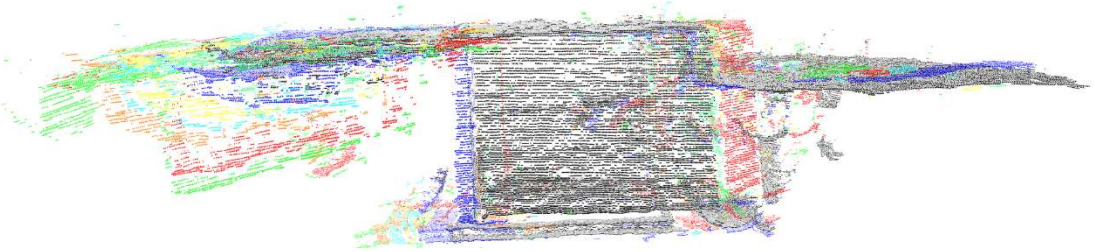


Figure 13:19 Top view

Object in scene:

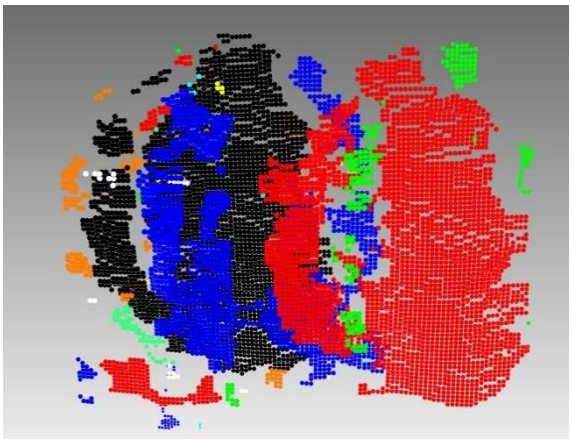


Figure 13:20 Object view

E.4: Test 4

Front view:

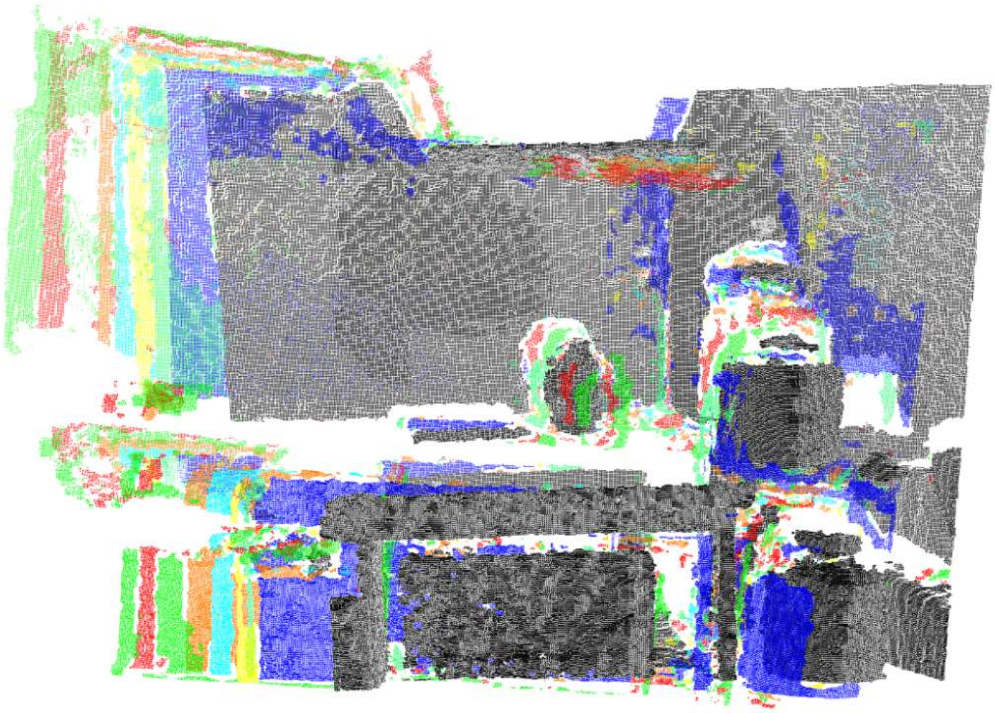


Figure 13:21 Front view

Top view:

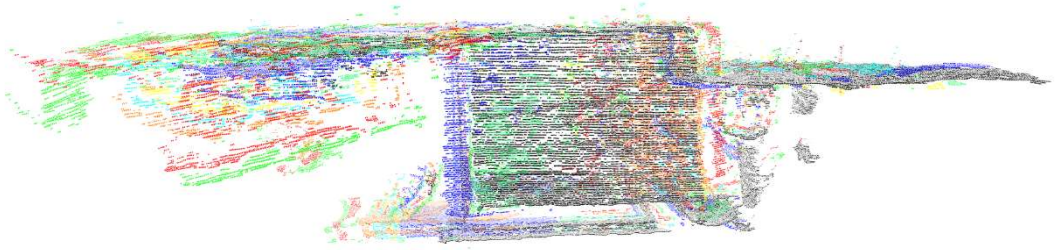


Figure 13:22 Top view

Object in scene:

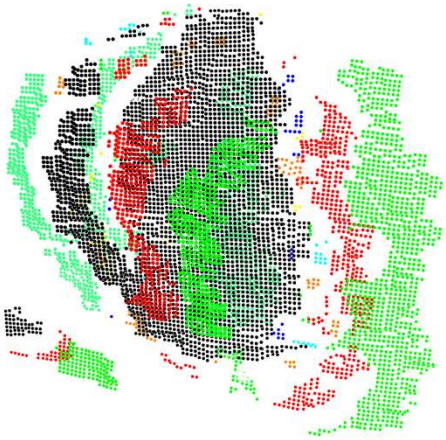


Figure 13:23 Object view

E.5: Test 5

Front view:

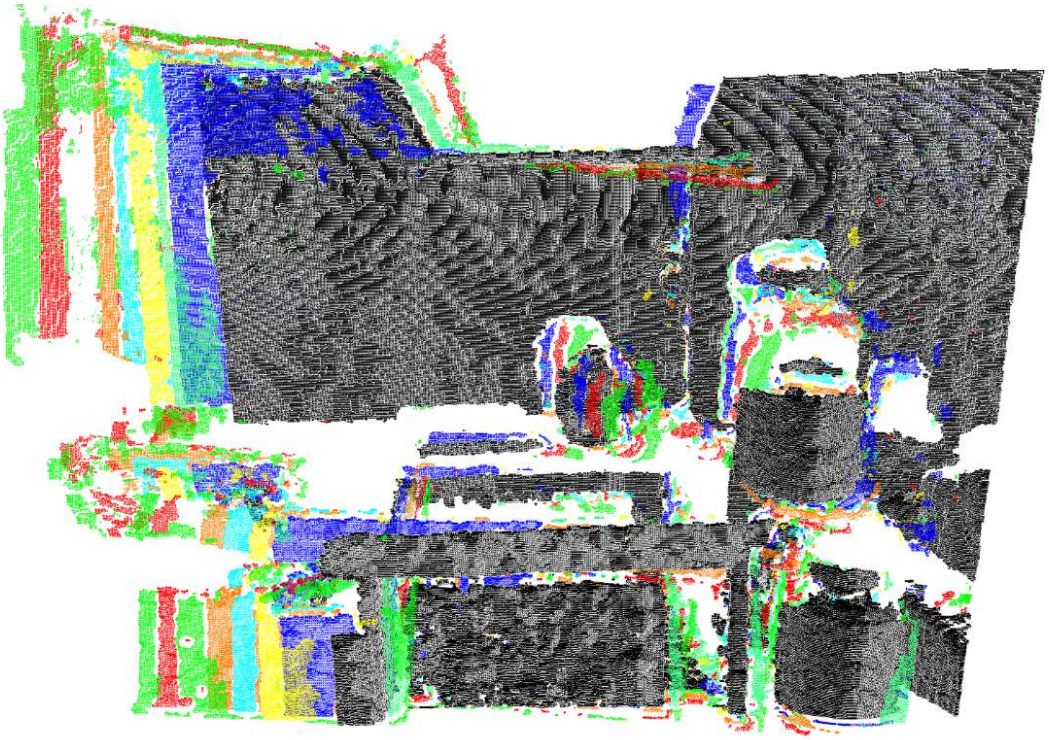


Figure 13:24 Front view

Top view:

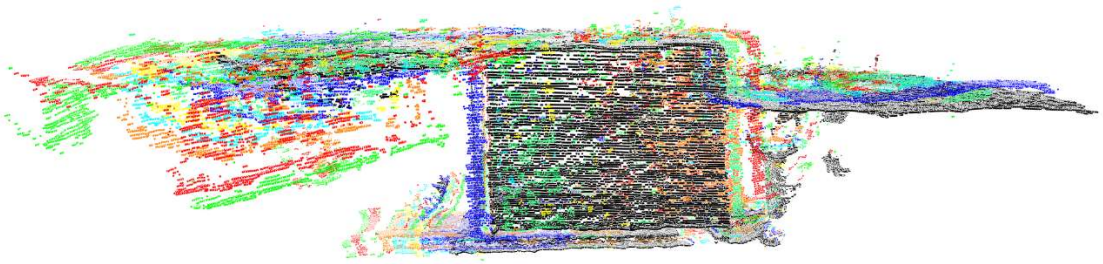


Figure 13:25 Top view

Object in scene:

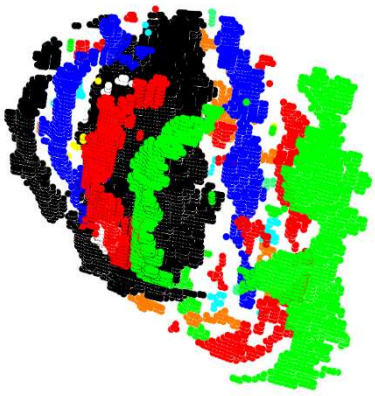


Figure 13:26 Object view

E.6: Test 6

Front view:

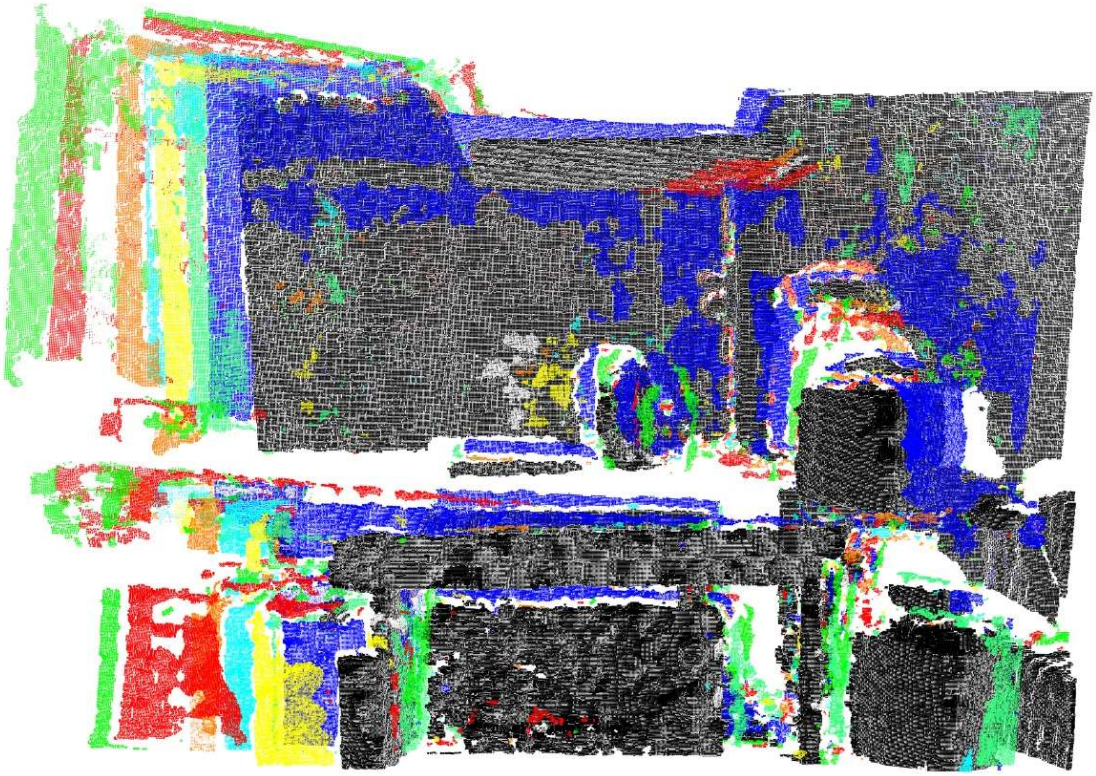


Figure 13:27 Front view

Top view:

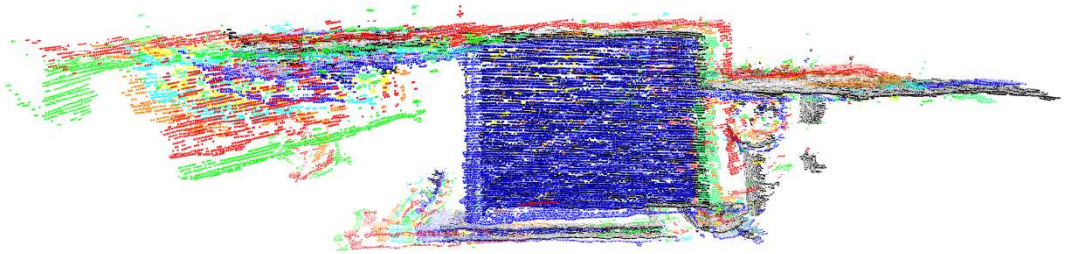


Figure 13:28 Top view

Object in scene:

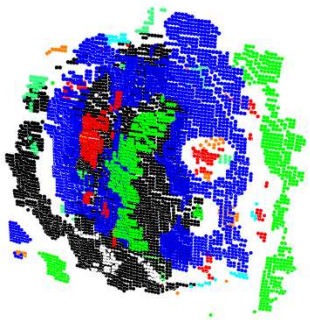


Figure 13:29 Object view

E.7: Test 7

Front view:

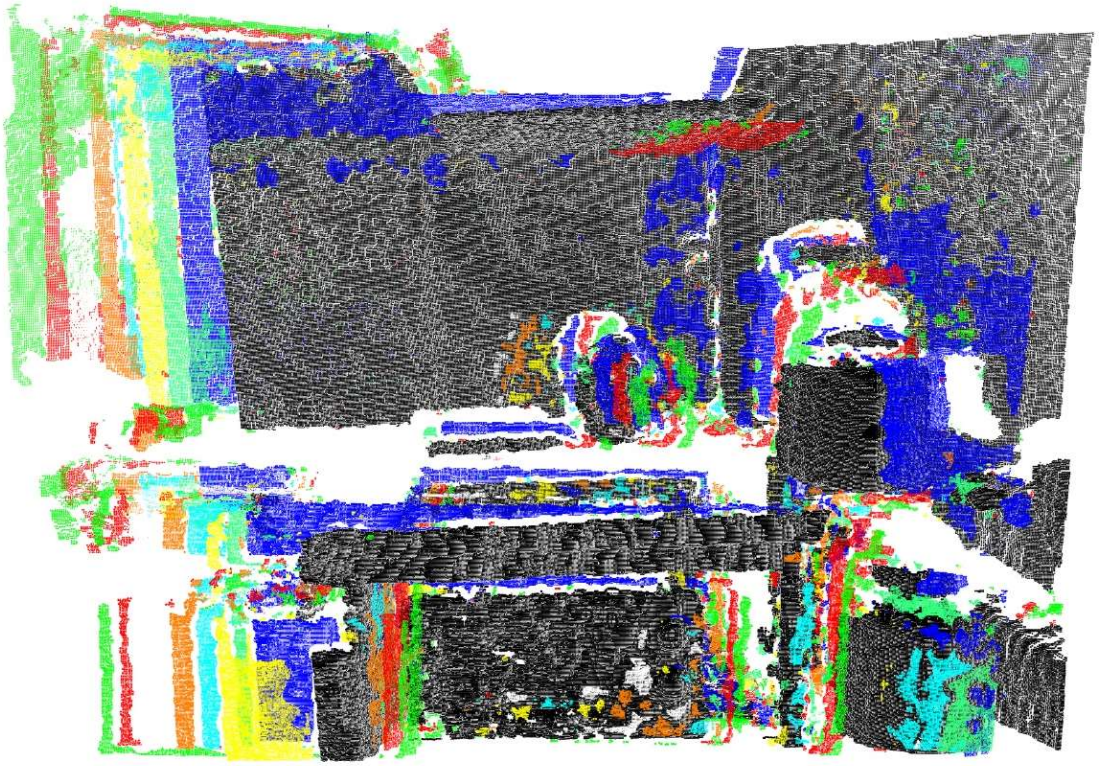


Figure 13:30 Front view

Top view:

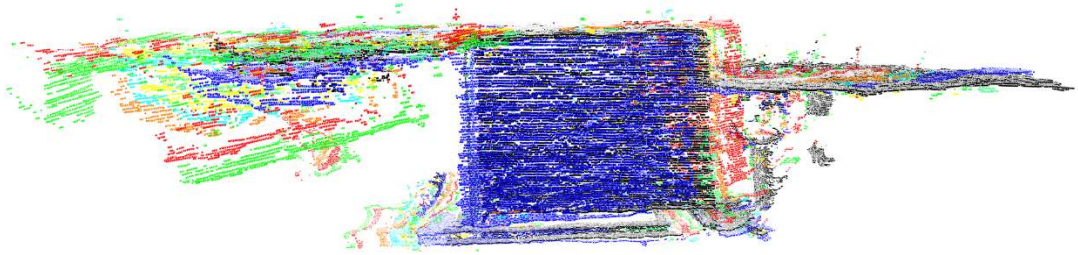


Figure 13:31 Top view

Object in scene:

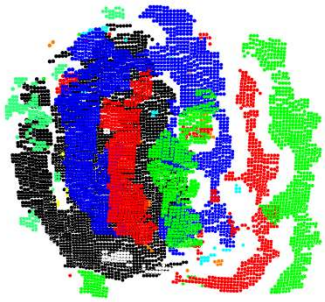


Figure 13:32 Object view

Attachment F:Dataset 4

Front view:

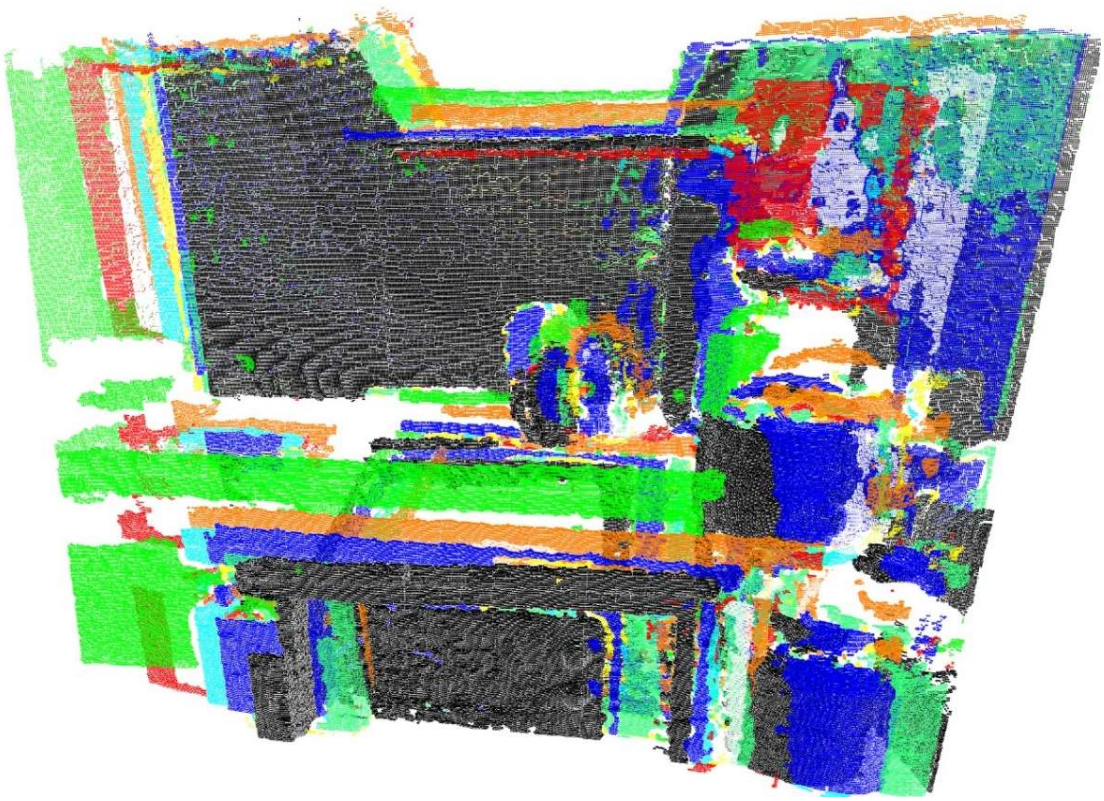


Figure 13:33 Front view

Top view:

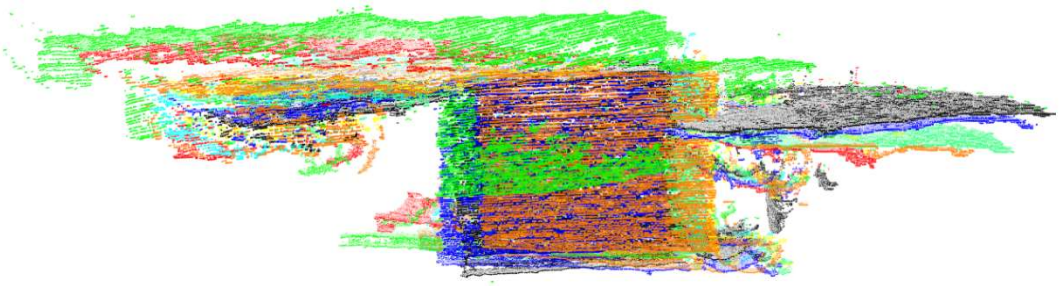


Figure 13:34 Top view

Object view:

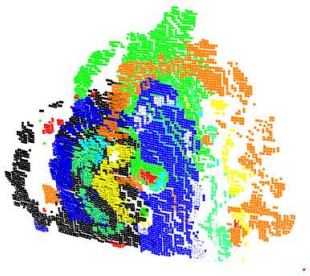


Figure 13:35 Object view

Attachment G:Workshop test

Figure 13:36 Workshop RGB

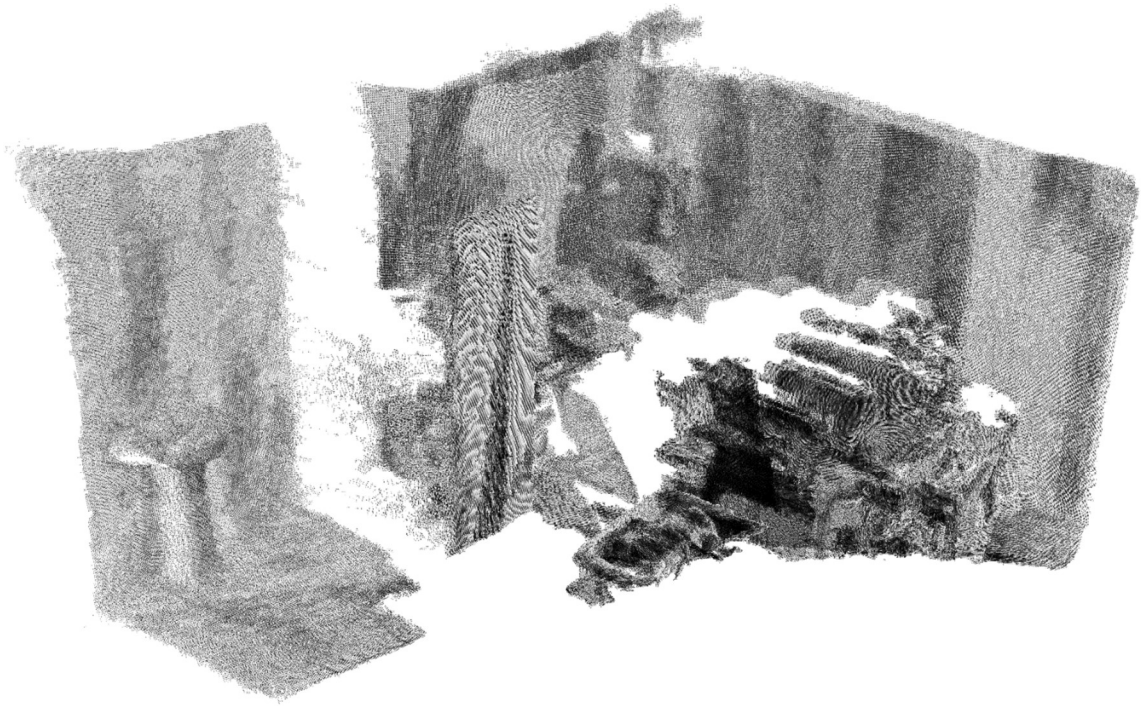


Figure 13:37 workshop point cloud

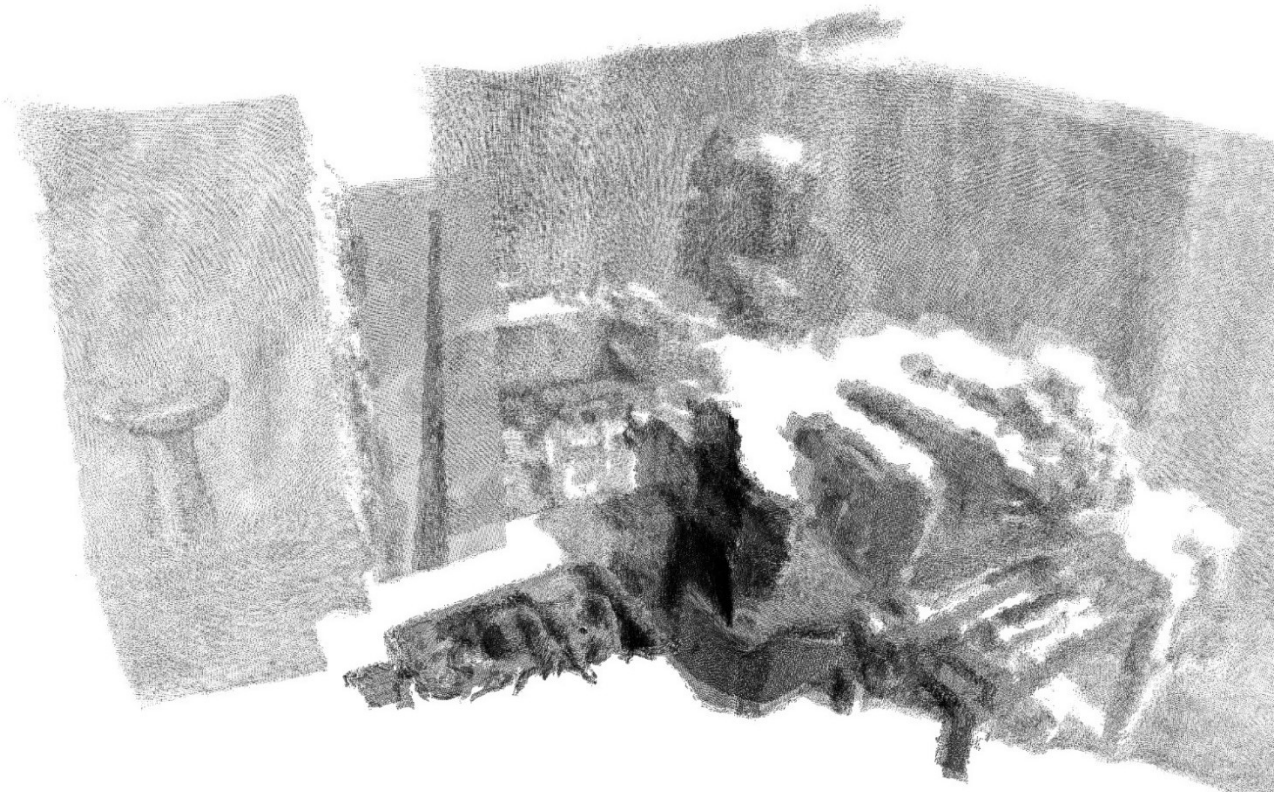


Figure 13:38 workshop point cloud

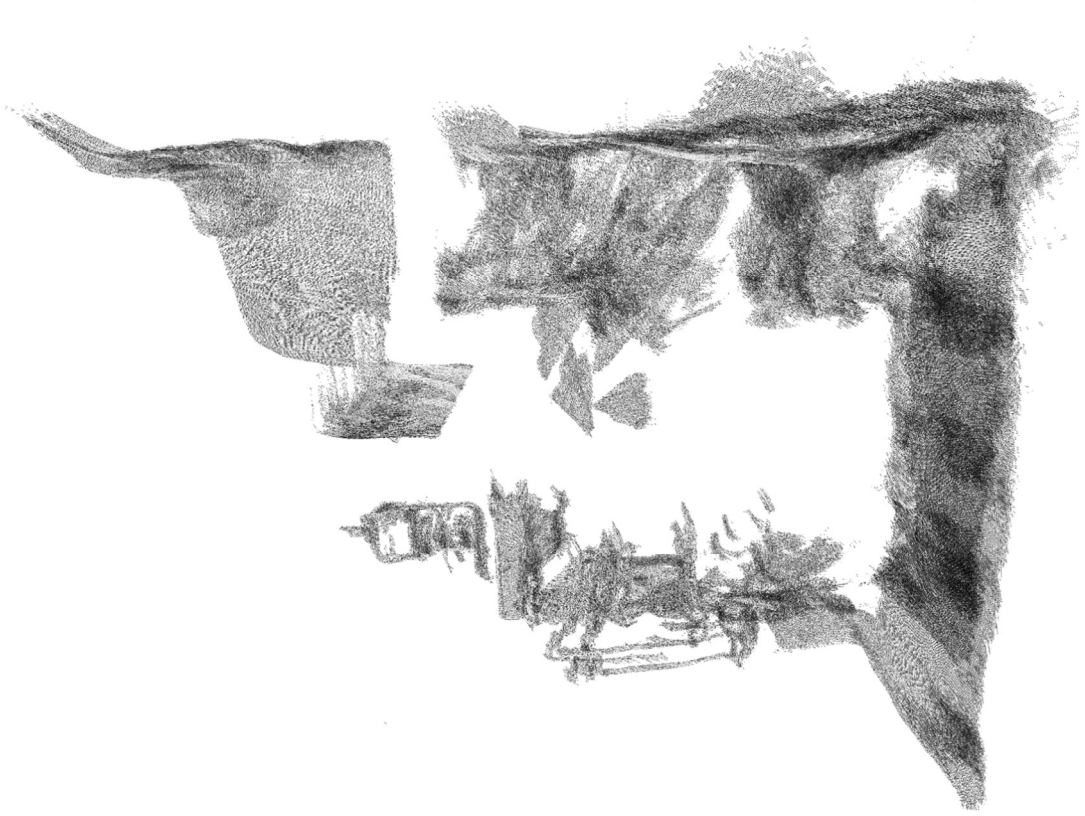


Figure 13:39 workshop point cloud top view

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
3D Indoor scanning & map building

Richting: **master in de industriële wetenschappen: energie-automatisering**
Jaar: **2017**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Vanspauwen, Sander

Datum: **8/06/2017**