



Technical University of Munich

TUM Department of Civil, Geo and Environmental Engineering

Chair of Computational Modeling and Simulation

Image-Based Localization in 3D Point Clouds

Master Thesis

for the Master of Science Program Computational Mechanics

Author: Ahmethan Başak

Student ID: [REDACTED]

Supervisors: Florian Noichl, M.Sc.

Yuandong Pan, M.Sc.

Date of Issue: 14. December 2020

Date of Submission: 14. June 2021

Abstract

Localization is a highly anticipated and active research topic. It divides into various areas concerning different interests and needs. Application of localization starts with basic questions like “where am I” and “how can I acquire the location information.” In terms of hardware usage in localization, laser scanners and cameras are examples of collecting environment information in different kinds of data. In this thesis, image-based localization is conducted with the images that are retrieved by cameras.

Cameras are tools for computer-vision-based approaches. According to the needs of the user, image data is processed during operation, or mapping of an area is done with all collected images. These images are the starting point of image-based localization. The area that localization takes place is reconstructed by collected images from this area because of the relationship between localization accuracy and 3D reconstruction of the scene. It is significant for localization. Reconstruction in this study is done by COLMAP, which is open-source software that has a pipeline structure to create a sparse and dense model from collected images. The reconstruction process is executed by using theories such as Scale-Invariant Feature Transform (SIFT), Structure-from-Motion (SfM), and Multi-View Stereo (MVS).

Reconstruction of the scene is used to find the location of a particular image with its camera poses inside an aligned large laser-scanned point cloud. To achieve this, the reconstruction steps are observed in a different set of parameters to reach a dense reconstructed model, and alignment quality is tried to be improved.

Zusammenfassung

Lokalisierung ist ein hochaktuelles und aktives Forschungsthema. Es unterteilt sich in verschiedene Bereiche, die unterschiedliche Interessen und Bedürfnisse betreffen. Die Anwendung der Lokalisierung beginnt mit grundlegenden Fragen wie "wo bin ich" und "wie kann ich die Standortinformationen erfassen." In Bezug auf die Verwendung von Hardware in der Lokalisierung sind Laserscanner und Kameras Beispiele für das Sammeln von Umgebungsinformationen in verschiedenen Arten von Daten. In dieser Arbeit wird die bildbasierte Lokalisierung mit den Bildern durchgeführt, die von Kameras abgerufen werden.

Kameras sind Werkzeuge für computer-vision-basierte Ansätze. Je nach Bedarf des Anwenders werden die Bilddaten während des Betriebs verarbeitet, oder es wird eine Kartierung eines Bereichs mit allen gesammelten Bildern durchgeführt. Diese Bilder sind der Ausgangspunkt der bildbasierten Lokalisierung. Der Bereich, in dem die Lokalisierung stattfindet, wird durch gesammelte Bilder aus diesem Bereich rekonstruiert, da ein Zusammenhang zwischen Lokalisierungsgenauigkeit und 3D-Rekonstruktion der Szene besteht. Sie ist für die Lokalisierung von Bedeutung. Die Rekonstruktion in dieser Studie erfolgt mit COLMAP, einer Open-Source-Software, die eine Pipeline-Struktur zur Erstellung eines spärlichen und dichten Modells aus gesammelten Bildern hat. Der Rekonstruktionsprozess wird unter Verwendung von Theorien wie Scale-Invariant Feature Transform (SIFT), Structure-from-Motion (SfM) und Multi-View Stereo (MVS) durchgeführt.

Die Rekonstruktion der Szene wird verwendet, um die Position eines bestimmten Bildes mit seinen Kamerapositionen innerhalb einer ausgerichteten großen lasergescannten Punktwolke zu finden. Um dies zu erreichen, werden die Rekonstruktions-schritte in einem unterschiedlichen Satz von Parametern betrachtet, um ein dichtes rekonstruiertes Modell zu erreichen, und es wird versucht, die Ausrichtungsqualität zu verbessern.

List of Contents

List of Figures	VII
List of Tables	XI
List of Abbreviations	XII
1 Introduction and Motivation	13
1.1 Image-based Localization	13
1.2 Aims and Objectives	14
1.3 The layout of the thesis.....	15
2 Theoretical Background	16
2.1 Structure-from-Motion	16
2.1.1 Incremental Structure-from-Motion	17
2.2 Scale-Invariant Feature Transform	19
2.2.1 Scale Space Extrema Detection	20
2.2.2 Keypoint Localization	24
2.2.3 Orientation Assignment.....	25
2.2.4 Keypoint Descriptor.....	26
2.3 Multi-View Stereo.....	28
3 Methodology	31
3.1 Toolkit	31
3.1.1 Docker	31
3.1.2 COLMAP.....	34
3.1.3 Connection with Server.....	35
3.1.4 CloudCompare.....	37
3.2 COLMAP Reconstruction.....	38
3.3 Initialize the reconstruction on the server.....	39
3.4 Adding New Images to Reconstructed Model.....	41
3.5 Alignment of Point Clouds.....	44
3.6 Finding Camera Poses	50

4	Experiments and Results	57
4.1	Alignment Quality.....	59
4.2	Camera Position Finding for Varied Reconstructions.....	66
5	Conclusion and Future Work	70
	References	73
	Appendix	77

List of Figures

Figure 2.1.1: COLMAP's incremental Structure-from-Motion pipeline (Schoenberger, 2020a)	17
Figure 2.2: Multi-Scale representation of a signal (Lindeberg, 1996)	20
Figure 2.3: Blurred Images (Lowe, 2004)	21
Figure 2.4: Gaussian Blur Operator (Lowe, 2004)	21
Figure 2.5: Blurring in several octaves (Tyagi, 2019)	21
Figure 2.6: The Scale-Space representation of the 2D image (Lindeberg, 1996)	22
Figure 2.7: Computing Difference of Gaussian (Lowe, 2004)	22
Figure 2.8: Difference of Gaussian of each scale (Lowe, 2004)	23
Figure 2.9: Comparison of $D(x, y, \sigma)$ value between neighbors (Lowe, 2004)	23
Figure 2.10: Location of extremum (Lowe, 2004)	24
Figure 2.11: Calculations for extrema elimination (Feature Detection - SIFT, 2019) (Lowe, 2004)	25
Figure 2.12: Keypoint with detected scale (Utkarsh Sinha, 2016)	25
Figure 2.13: Keypoint Histogram (Utkarsh Sinha, 2016).....	26
Figure 2.14: Keypoint Descriptor 16x16 Histogram (Tyagi, 2019)	27
Figure 2.15: Histogram with sub-blocks (Tyagi, 2019).....	27
Figure 2.16: Multi-View Reconstruction pipeline (Slobodan Ilic, 2011)	29
Figure 2.17: Window-based Multi-View Stereo algorithm (Agarwal et al., 2011)	29
Figure 3.1: Docker architecture (Docker, 2018).....	31
Figure 3.2: Docker build command.....	32
Figure 3.3: Docker Container Run Commands For COLMAP's Reconstruction	32
Figure 3.4: Percentage of usage for each GPU in the server	33
Figure 3.5: Docker volume system (Docker, 2020).....	33
Figure 3.6: “docker ps -a” command.....	34
Figure 3.7: Create an SSH keypair (Gite Vivek, 2021)	35
Figure 3.8: PuTTY Key Generator	36

Figure 3.9: FileZilla interface	37
Figure 3.10: CloudCompare Logo and GUI (Daniel Girardeau-Montaut, 2011).....	37
Figure 3.11: COLMAP Pipeline with Mapper function.....	38
Figure 3.12: COLMAP’s Feature Extractor Commands	40
Figure 3.13: Usage of Docker cp command	40
Figure 3.14: Workspace folder.....	41
Figure 3.15: File system for the mapper	41
Figure 3.16: Docker Container Run Commands For COLMAP’s Mapper Function ..	42
Figure 3.17: DB Browser for SQLite	43
Figure 3.18: COLMAP Visualization of Sparse model and Camera Locations	43
Figure 3.19: COLMAP Visualization of Generated Point Cloud	44
Figure 3.20: Laser-scanned point cloud without any cutting	44
Figure 3.21: MeshLab Toolbar.....	45
Figure 3.22: Left: Before Deleting Points Right: After Deleting Points	45
Figure 3.23: Align Tool Window of MeshLab	46
Figure 3.24: Point-Based Glueing in MeshLab	46
Figure 3.25: MeshLab Error Text.....	47
Figure 3.26: Unsuccessful Alignment of Point Cloud in MeshLab	47
Figure 3.27: CloudCompare Toolbox.....	47
Figure 3.28: Left: Picking Points From “Align” Point Cloud Right: Picking Point From “Reference” Point Cloud	48
Figure 3.29: Align Tool Windows of CloudCompare	48
Figure 3.30: Generated Transformation Matrix in CloudCompare	49
Figure 3.31: Aligned Point Clouds in CloudCompare	49
Figure 3.32: Colored Point Clouds in CloudCompare	50
Figure 3.33: Point Cloud Visualized with Python Open3D library	50
Figure 3.34: Exported Text File Format of the model	51
Figure 3.35: COLMAP GUI Features for importing and exporting models	51
Figure 3.36: Camera Poses inside laser-scanned point cloud.....	52
Figure 3.37: Filtering Point outside Camera View.....	53

Figure 3.38: Application of Pythagorean theorems	54
Figure 3.39: Point Viewpoint Check.....	54
Figure 3.40: Camera Locations and their captured points	55
Figure 3.41: An image file and its corresponding viewpoint with captured points	56
Figure 4.1: Project Initialization File.....	57
Figure 4.2: Patch Match Configuration	58
Figure 4.3: Mapper and first reconstruction comparison.....	59
Figure 4.4: Alignment with 4 points.....	60
Figure 4.5: Cloud-to-Cloud feature	60
Figure 4.6: Density comparison between point clouds	61
Figure 4.7: Choose Role.....	61
Figure 4.8: Left: General settings for Distance Computation Right: Local Model Type Selection	62
Figure 4.9: Left: Cloud to Cloud distance for Reconstruction 1 Right: Cloud to Cloud distance visualization	62
Figure 4.10: Left: Cloud to Cloud distance for Reconstruction 2 Right: Cloud to Cloud distance visualization	63
Figure 4.11: Left: Cloud to Cloud distance for Reconstruction 3 Right: Cloud to Cloud distance visualization	63
Figure 4.12: Left: Cloud to Cloud distance for Reconstruction 4 Right: Cloud to Cloud distance visualization	63
Figure 4.13: Left: Cloud to Cloud distance for Reconstruction 5 Right: Cloud to Cloud distance visualization	64
Figure 4.14: Left: Cloud to Cloud distance for Reconstruction 6 Right: Cloud to Cloud distance visualization	64
Figure 4.15: Left: Cloud to Cloud distance for Reconstruction 7 Right: Cloud to Cloud distance visualization	64
Figure 4.16: image455, image456 and image459	66
Figure 4.17: image460, image461 and image463	66
Figure 4.18: Captured Points for image455 and image461 for Reconstruction 4.	67
Figure 4.19: Reconstruction 6 with camera poses	67
Figure 4.20: Reconstruction 4 with camera poses	68

Figure 4.21: Camera Viewpoint of image463 from Reconstruction 6 (Left) and 4 (Right)	68
Figure 4.22: Captured points of Camera Viewpoint Reconstruction 6 in CloudCompare	68
Figure 4.23: Captured points of Camera Viewpoint Reconstruction 4 in CloudCompare	69
Figure 4.24: CloudCompare Visualization of captured points of image455 and image461	69
Figure 5.1: CloudCompare ICP Setting window (CloudCompare, 2015b)	70
Figure 5.2: Classic ICP algorithm flow chart (Liu et al., 2018)	71
Figure 5.3: Improved ICP algorithm flow chart (Liu et al., 2018)	71
Figure 5.4: CMS laser-scanned point cloud	72
Figure 5.5: Office 3218 manually found	72

List of Tables

Table 1: Reconstructed Model Analyze after the mapper	59
Table 2: Reconstruction C2C Comparison with respect to a threshold value	65
Table 3: CloudCompare data storing structure for C2C distance	65
Table 4: Reconstruction C2C Comparison with respect to first five class	66
Table 5: Captured Points by cameras for different reconstructions	67

List of Abbreviations

2D	Two-dimensional
3D	Three-dimensional
4PC	Four Point Congruent
C2C	Cloud-to-Cloud
CMS	Chair of Computational Modelling and Simulation
CV	Computer Vision
DoG	Difference of Gaussians
DSP	Domain Size Pooling
GUI	Graphical User Interface
ICP	Iterative Closest Point
MVS	Multi-View Stereo
OS	Operating System
PnP	Perspective-n-Point
SfM	Structure-from-Motion
SIFT	Scale-Invariant Feature Transform

1 Introduction and Motivation

1.1 Image-based Localization

Image-based localization is an essential topic of computer vision. Localization applications can be varied according to a different set of conditions. Some of the cases occur in indoor or outdoor environments. For example, augmented reality, navigation, localization of pedestrians and robots, and visualization of photo collections are the topics that image-based localization is needed and can improve the application process. For photo collections, image-based localization uses them for large-scale reconstructions that are based on the initial pose estimate of the localization approach. Reconstruction of the selected environment starts with estimating the orientation of the camera, and the scale of the reconstruction is rising with advanced research such as Structure from Motion (SfM).

3D models are used for image-based localization. The usage of the 3D model is described in this paper (Li et al., 2010). 3D points and their descriptors are named as points, 2D image features, and their descriptors as features. Matching is handled as direct and indirect 2D to 3D matching. Direct matching focuses on searching for the nearest neighbors of a specified feature descriptor in a given space which consists of 3D point descriptors. As a result of this search, a 3D point corresponding to a 2D feature can be found. Indirect matching follows a more generalized approach than searching individual matches. Points and their descriptors are represented by an intermediate construct. In this method, the proximity does not preserve in descriptor space, unlike the direct way. The local 2D features in the query image are processed to establish correspondence between other features. This correspondence search is crucial for image-based localization, and it is accepted as a start point of image-based localization. The feature descriptors like Scale-Invariant Feature Transform (SIFT) are one of the common approaches used for the reconstructed 3D points in the model. These SIFT descriptors formulate the correspondence search as a descriptor matching problem. Classical direct matching approaches, such as approximative tree-based search, can be considered because of their performance in providing good matching results from the image to descriptors from the model. This good performance comes with a drawback in terms of computational effort and time. In large and dense descriptor

collections, the search for matching makes the process expensive to handle. Thus, recent approaches consider using the indirect matching method to process massive databases (Sattler et al., 2011).

1.2 Aims and Objectives

This thesis focuses on localizing a particular image and the camera pose, which is related to this image inside a given laser-scanned point cloud of an area. Image-based localization is applied to get information from an unknown area and locate positions. Images that are taken from a video as a series of screenshots are collected from the area to reconstruct the scene. Then, the reconstructed point cloud is aligned with a laser-scanned point cloud. The laser-scanned point cloud is referred to as ground truth, and benchmarks are done with respect to it.

The first part of the localization, reconstruction of the area, is done by an open-source software COLMAP perform the reconstruction process. Series of images are used to reconstruct a sparse and dense model. The 3D point cloud is created separately after the reconstruction process. Densification of the point cloud is a need for localization accuracy. Therefore, we have the point cloud, which is based on the dense model instead of a sparse model. Additional functions of COLMAP are used to add new images into the reconstruction. Image registration is applied, and a new image is successfully registered into the COLMAP database file. Then, the mapper function is implemented to match the feature points of added images into the previous database with a camera position inside the point cloud.

The reconstructed dense 3D point cloud is the requirement to move on to the next phase of the thesis. In the second part, the main objective of the thesis is finding the camera pose of a new image inside the laser-scanned point cloud. Comparison between the laser-scanned point cloud and the COLMAP generated point cloud is started with the alignment process. CloudCompare software performs the alignment between point clouds. Therefore, the quality comparison and the error between point clouds are available with the help of CloudCompare. Then, a python script is used to filter the added images and their related information from the database of reconstruction that includes new images. As a result of this, the camera locations of the new images are reachable and visualized.

Afterward, several trials are conducted with various parameters, and the influence of the parameters is discussed with their needs of computation power and time.

Moreover, the success rate of adding new images into produced reconstruction is analyzed, and the location of the new camera poses that are generated from added images is compared inside a given laser-scan point cloud. A further approach is addressed for better alignment between the 3D reconstructed point cloud and laser-scanned point cloud.

1.3 The layout of the thesis

Chapter 1 identifies the usage and current state of the art of image-based localization and presents the objective of this thesis with a brief explanation of further processes.

Chapter 2 introduces the theoretical background behind the COLMAP software. The pipeline of SfM and MVS are explained with used terms like feature detection and image matching of SIFT.

Chapter 3 includes the explanation of the tools and software that are used in this study and then presents the workflow of reconstruction, describing the alignment process between point clouds and finding the location of the camera pose and the image inside the laser-scanned point cloud with an explained script.

Chapter 4 presents some specific test cases with different assumptions and parameters. Experiment setups are explained, and their corresponding results are discussed.

Chapter 5 discusses the performance of the alignment process and success in finding camera poses of added images inside laser-scanned point clouds with different reconstructions of the scene.

2 Theoretical Background

2.1 Structure-from-Motion

Structure from motion (SfM) is originated from two key fields, photogrammetry and computer vision. Photogrammetry is an old topic that is used for measuring and processing lengths and angles in photos for mapping purposes (Visser, 1982). One of the early researches in vision consists of the recovery from the stereo, which is proposed by Marr and Poggio (2016). In that study, the correspondence between images is established by an iterative cooperative algorithm.

The current state-of-the-art SfM consists of several sections to reconstruct 3D structure from a series of images taken from different viewpoints inside an area. These images sequentially are added into the final reconstruction.

SfM is applied by using three different strategies. These are incremental strategies, hierarchical strategies, and global approaches. Global approaches of SfM reconstructed all images at once instead of processing them sequentially. During the reconstruction, global camera positions are estimated based on different approaches such as pairwise rotations and vanishing points, rotation averaging in the RANSAC framework, discrete-continuous optimization, and lie-algebraic averaging. After the estimation step is completed, SfM solves a linear problem to evaluate 3D structure and camera translations. One of the drawbacks of the global approach of SfM occurs in case the pairwise geometries are inaccurate or the number of pairwise geometries is not enough. This method is not able to average pairwise motions properly. Other SfM strategy is named as hierarchical strategies which aim to avoid fully sequential reconstruction of incremental SfM like global approaches. Hierarchical strategies propose a difference apart from the global approach. It does not use global estimations. There are various studies for reconstruction that are based on hierarchical strategies. Havlena (2009) uses visual words to find candidate image triplets for reconstruction. Later these reconstructions are merged into a more extensive reconstruction. Gherardi (2010) proposes implementing a balanced tree on images to use clustering of match-graph. These clusters are merged hierarchically to build a larger reconstruction (Shah et al., 2015).

SfM aims to get unknown parameters by only using 2D point coordinate measurements over several views or frames. These measurements are the locations of the 2D features in the images which depend on three parameters. The first parameter is the coordinates of the feature points in 3D space, the second reason is the relative 3D motion between the camera and the scene, and the third parameter is the internal geometry of the camera (Jebara et al., 1999).

In this thesis, incremental strategy is used for SfM. The algorithm and the related software are written by Schönberger. The name of the software is known as COLMAP, which is an open-source tool for structure-from-motion and multi-view modeling.

2.1.1 Incremental Structure-from-Motion

Incremental SfM differs from other strategies with the sequential processing of each image in collections. Good seed image pairs are evaluated to reconstructs the cameras and points. Reconstruction progress sequentially images by image by adding well-connected images, estimating camera parameters, and triangulating feature matches. (Brown & Lowe, 2005; Snavely et al., 2006) Global bundle adjustment, which is a refinement of camera poses and 3D point positions, is used to avoid drift accumulation. Bundle adjustment is repeated during every step in incremental SfM, and this causes the complexity of the incremental SfM is resulted as $O(n^4)$ due to repeated BA. (Shah et al., 2015) Various methods are proposed to improve the efficiency of the BA. Among these methods, two topics come into prominence. These are fast approximations of the sparse bundle adjustment and exploiting many-core architectures to parallelize it. (Agarwal et al., 2011; Agarwal et al., 2010; Wu et al., 2011) As an example of parallelization, leveraging highly parallel GPU architecture for an optimized pipeline is offered by Wu. (2013)

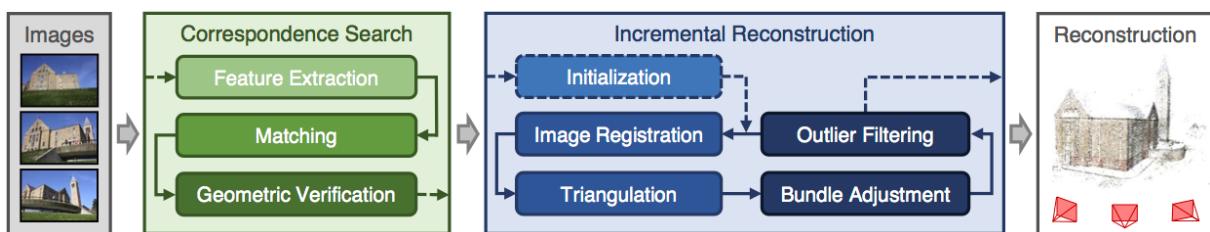


Figure 2.1.1: COLMAP's incremental Structure-from-Motion pipeline (Schoenberger, 2020a)

Incremental strategy is chosen for this thesis application since it is the leading strategy for unordered photo collection. COLMAP introduces a new approach of SfM to improve the resulted reconstruction in terms of accuracy, robustness, scalability, and

completeness (Schönberger & Frahm, 2016). COLMAP pipeline of SfM that is shown in Figure 2.1.1 is divided into two sections for the process of incremental SfM. These are correspondence search and incremental reconstruction. Correspondence search is related to find the relationship between images in terms of feature point matching.

Feature extraction is defined as feature point detection in images. These points are points that are likely to be detected in corresponding images. SIFT is used as the primary tool to extract these features. The appearance descriptor represents detected sets of local features at the location for each image. One condition is set for the features to be detected by SfM. The features can be recognizable in multiple images if they are invariant under geometric and radiometric changes. After feature extraction is completed successfully, feature points need to be matched to know the correspondence between points in each pair of images. SfM detects the same scene part by using the appearance description of the images, which is a representation of features in images. Every image pair is tested for scene overlap, which means if a pair is overlapped, there should be a feature correspondence between them. Feature correspondence is evaluated by using a similarity metric that compares the appearance of the features. Finding the most similar feature in an image for every feature in the image proves that correspondence.

The third stage of the COLMAP pipeline is geometric verification which is responsible for verifying the potentially overlapping image pairs. The reason why this verification step is needed is that matching is based entirely on appearance. So there is no guarantee that corresponding features actually map to the same scene point. Thus, verification of matches involves estimating the transformation that maps feature points between images. This estimation is managed by projective geometry, which depends on the spatial configuration of an image pair and changes with respect to different mapping that describe the geometric relation.

After geometric verification is confirmed, reconstruction of the 3D structure proceeds with the second part of the pipeline. In this part, COLMAP's incremental reconstruction takes place. Initial pair selection is significant for reconstruction. The reason for the significance, bad chosen initial pair makes COLMAP hard to progress towards the end result because the number of overlapping camera views will not be enough to reach robust and accurate reconstruction. This lack of redundancy affects the performance of the process. To avoid this negative effect, the model should be initialized with a

particularly selected two-view reconstruction (Beder & Steffen, 2006). During the reconstruction, the model is growing by registering new images. The registration step is done by solving the Perspective-n-Point (PnP) problem. The problem is defined as using feature correspondences to triangulate points in registered images. PnP follows different routes according to the calibration of the camera. Estimation of the pose of a calibrated camera is executed with a given set of 3D points in the world and their equivalent 2D projections in the images. RANSAC (Frahm et al., 2006) and minimal pose solver are used for this estimation. For uncalibrated cameras, the intrinsic parameters are used with various minimal solvers and sampling-based approaches. A novel robust following best image selection method for accurate pose estimation is proposed by COLMAP. Registration is finalized with triangulation. Triangulation is applied to calculate the 3D point corresponding to each pair of matched points. Triangulation is an important step in SfM. It increases the stability of the existing model through redundancy and enables registration of new images by providing additional 2D-3D correspondences (Triggs et al., 2000). Although image registration and triangulation are separate procedures, their results are highly correlated, and the uncertainties of the camera pose may propagate to triangulated points. Therefore, further refinement is needed. In the last step of the incremental reconstruction, an iterative bundle adjustment is used to provide a refinement to the model. The joint non-linear refinement of camera parameters and point parameters minimizes the reprojection error and potentially down-weight outliers by using a loss function which is called function π_i , that projects scene points to image space.

This proposed SfM pipeline of COLMAP introduces a method to identify and highly overlapping images for efficient bundle adjustment of dense collections and establish complete and precise models with improved robustness and accuracy.

2.2 Scale-Invariant Feature Transform

Scale-Invariant Feature Transform (SIFT) is a widely used feature-based method for object recognition and was first presented by Lowe in 1999. It uses a class of local image features that are invariant to image scaling, translation, and rotation and partially invariant to changes in lighting and projection (Forsman, 2011).

There are four steps of generating the set of image features. They are listed as Scale-space extrema detection, keypoint localization, orientation assignment, and keypoint descriptor (Lowe, 2004).

2.2.1 Scale Space Extrema Detection

Scale-space extrema detection is a filtering approach to identify the locations and scales of features from different views of the same object. This identification is made by using the scale-space function (*SIFT Image Features*, 2015). This section includes the definition of scale space and usage of the scale-space function to find keypoints in the images.

Real-world objects are only recognizable and exist as meaningful entities if they are in a certain range of scale. This recognition difference can be described with an example. In this example, a branch of a tree can be identified easily with all details from a few centimeters to a few meters. If the distance between the objects and viewer increases or decreases too much, objects should change with respect to the scale. At the nanometer level, observing the molecules inside the leaves of the tree becomes meaningful. Also, looking at a forest with a lot of trees is logical for the kilometer level. (Lindeberg, 1996) A lot of familiar examples of this multi-scale nature of objects can be observed. The scale of observation affects the appearance of the objects in the real world and shows the notion of the scale. Scale-space is a way of replicating this conception on digital images. The need for this replication comes from the question of how to analyze and derive information from real-world measurement data.

Multi-scale representation is visualized in Figure 2.2. The original signal with different levels of scale is represented by an ordered set of derived signals. The source of the signal is the image that its features are unknown. “t” represents the levels of the scale (Lindeberg, 1996).

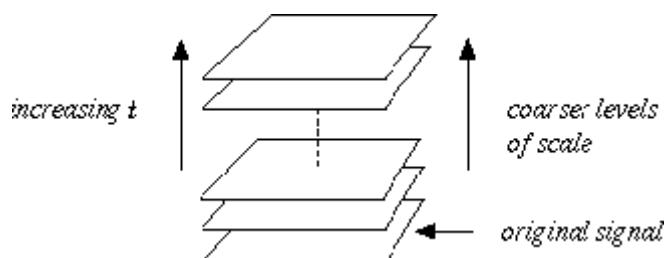


Figure 2.2: Multi-Scale representation of a signal (Lindeberg, 1996)

The main focus of the scale-space representation is producing a set of derived signals which includes the suppressed form of the fine-scale information. A function is used to represent this set as the scale-space of an image. This function, $L(x,y,\sigma)$, is generated from the convolution of a Gaussian kernel at different scales from the input images.

This convolution and its derivatives create smoothing transformations (*Feature Detection - SIFT*, 2019).

Two terms take into place to describe this process. These are octaves and gaussian blurring. Octaves are the separated form of the scale-space, and the number of octaves and scale are related to the size of the original image. Gaussian blurring is the convolution of the gaussian operator and the image in mathematical representation. The blurred image is defined with a function in Figure 2.3 where $*$ is the convolution parameter, $G(x, y, \sigma)$ is a variable-scale Gaussian, and $I(x, y)$ is the input image (Lindeberg, 1993).

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y),$$

Figure 2.3: Blurred Images (Lowe, 2004)

“ σ ” is the scale operator in Figure 2.4. It is the same parameter with “ t ” in Figure 2. The amount of blur in the image depends on the value of $G(x, y, \sigma)$.

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}.$$

Figure 2.4: Gaussian Blur Operator (Lowe, 2004)

In Figure 2.5, several octaves of the original image are generated. The size of each octave is half of the previous octave. Inside an octave, each pixel in the images gradually blurred by Gaussian Blurring.

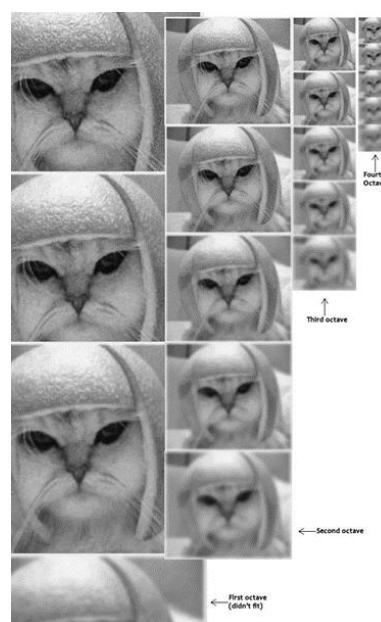


Figure 2.5: Blurring in several octaves (Tyagi, 2019)

Another example of different levels in the scale-space representation of a two-dimensional image is shown in Figure 2.6. Scale levels are defined as t or $\sigma = 0, 2, 8, 32, 128$ and 512 . Grey-level blobs indicate local minima at each scale.

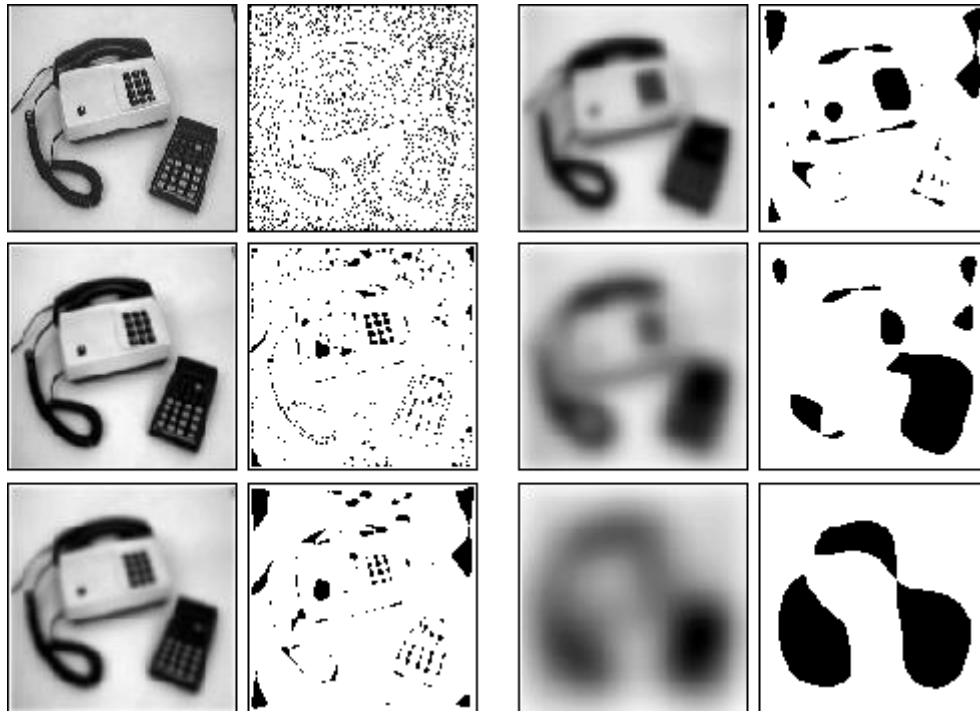


Figure 2.6: The Scale-Space representation of the 2D image (Lindeberg, 1996)

In finer scales, each noise and texture are detected separately. When the scale becomes coarser with applied smoothing, separated sections are unified and merge into one unit. Finally, bigger single blobs are observed. This example shows the hierarchical shape decomposition by observing the effect of varying scale parameters in the scale-space representation (Lindeberg, 1996).

Difference of Gaussians (DoG)

Blurred images that are produced by the Gaussian kernel are used to generate another set of images to detect locations of the stable keypoint locations in the scale-space. The DoG is one of the various methods that can locate scale-space extrema. DoG takes the difference of Gaussian blurring of two images with different scale values. One of the images is scaled k times as $k\sigma$, and the scaled image differentiate from the other image that has a scale value of σ , as it is seen in Figure 2.7.

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma).$$

Figure 2.7: Computing Difference of Gaussian (Lowe, 2004)

In the Gaussian Pyramid, this process is repeated for different octaves of the image.. In Figure 2.8, the result of the process that is done for different octaves of the image in the Gaussian Pyramid is represented (Lowe, 2004).

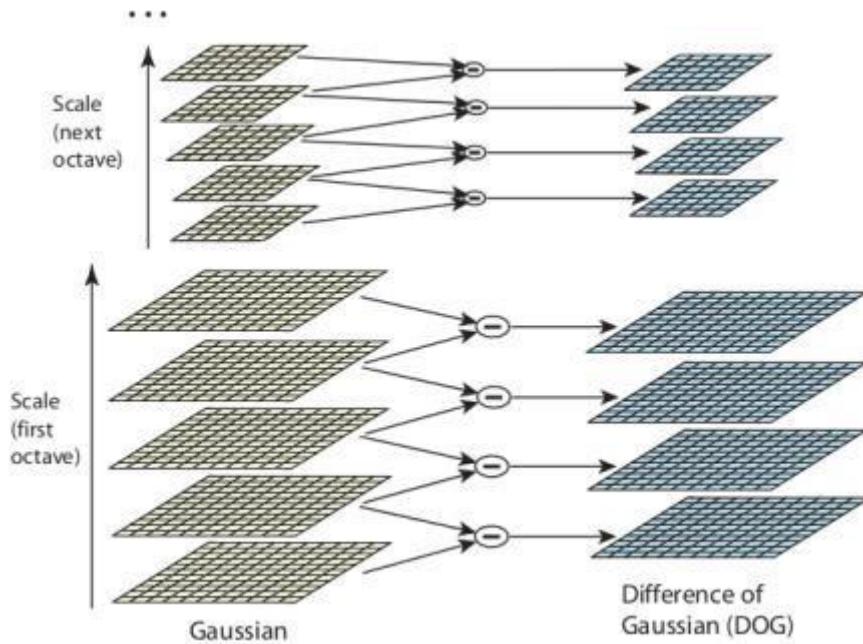


Figure 2.8: Difference of Gaussian of each scale (Lowe, 2004)

Finding Keypoints

Until this part, scale space is generated and used to calculate the Difference of the Gaussians. Those DoGs are used to compute Laplacian of Gaussian approximations that are scale-invariant.

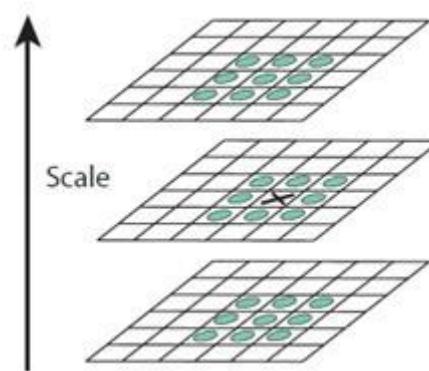


Figure 2.9: Comparison of $D(x, y, \sigma)$ value between neighbors (Lowe, 2004)

Search for the local extrema is done by several comparisons between different scales and inside scales in Figure 2.9. Comparison inside the scale starts with comparing the 8 neighbors of a selected pixel, $D(x, y, \sigma)$ value of each pixel is compared with the neighbors to detect the local maxima and minima at the same scale. Then, the upper

and lower scales take into comparison. 9 pixels in upper and lower neighbors are compared with the selected pixel of up and down one scale. Thus, 26 checks are made for each pixel in an image. If the $D(x, y, \sigma)$ value is the minimum or maximum of all these compared values of pixels, this selected pixel will become local extrema. If it is a local extremum, it is also a potential keypoint or the best representation of the keypoint in that scale.

2.2.2 Keypoint Localization

In the previous step, plenty of keypoints are generated. Therefore, they should be filtered out according to some criteria. The keypoints which have low contrast or are poorly localized on edge are eliminated from the list of keypoints. The reason for the elimination is these keypoints are not useful as features, and they should be removed. After that, a similar approach to the Harris Corner Detector is applied to remove edge features, and low contrast features are removed with respect to their intensity value.

To make the decision about elimination in terms of contrast, Laplacian is calculated for each keypoint in the list, and a value z which represents the location of extremum, is found in Figure 2.10. Additionally, Taylor series expansion of scale space is used to get a more accurate location of extrema (*SIFT Image Features*, 2015).

$$\hat{\mathbf{x}} = -\frac{\partial^2 D}{\partial \mathbf{x}^2}^{-1} \frac{\partial D}{\partial \mathbf{x}}.$$

Figure 2.10: Location of extremum (Lowe, 2004)

If the function x value is below a threshold value, extrema are eliminated because of low contrast. For the elimination decision with respect to the location of extrema, a condition is searched. This exclusion is required since the difference of Gaussians has a higher response for edges. In this condition, a large principle curvature across the edge but a small curvature in the perpendicular direction in the difference of Gaussian function. If the difference value is below the ratio of largest to the smallest eigenvector, the extrema are eliminated. Principal curvature is computed from the 2×2 Hessian matrix (H) at the location and scale of the extrema (*Feature Detection - SIFT*, 2019).

In Figure 2.11, filtering for keypoints is done for two criteria. Flats mean checking the contrast and value is compared to a threshold which is taken as 0.03 in this example. Edge checking for this case is applied by computing Hessian.

- Reject flats:

- $|D(\hat{x})| < 0.03$

- Reject edges:

$$\mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Let α be the eigenvalue with larger magnitude and β the smaller.

$$\text{Tr}(\mathbf{H}) = D_{xx} + D_{yy} = \alpha + \beta,$$

$$\text{Det}(\mathbf{H}) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta.$$

Let $r = \alpha/\beta$.
So $\alpha = r\beta$

$$\frac{\text{Tr}(\mathbf{H})^2}{\text{Det}(\mathbf{H})} = \frac{(\alpha + \beta)^2}{\alpha\beta} = \frac{(r\beta + \beta)^2}{r\beta^2} = \frac{(r+1)^2}{r},$$

- $r < 10$

$(r+1)^2/r$ is at a min when the 2 eigenvalues are equal.

Figure 2.11: Calculations for extrema elimination (Feature Detection - SIFT, 2019) (Lowe, 2004)

2.2.3 Orientation Assignment

The previous process verifies the keypoints in terms of stability, and the scale that detected keypoint is located is known as is seen in Figure 2.12. The blurred image represents the scale of detection for the yellow keypoint. Therefore, scale invariance exists.

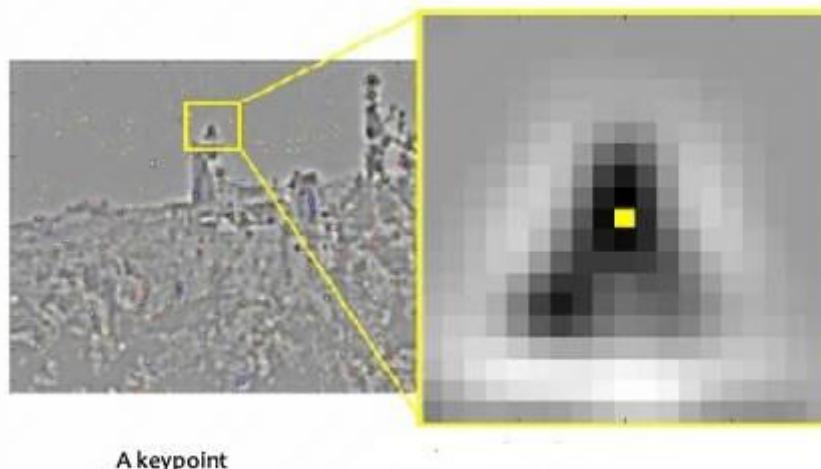


Figure 2.12: Keypoint with detected scale (Utkarsh Sinha, 2016)

In that step, the goal is assigning a consistent orientation to the keypoints based on local image properties. This assignment is needed because keypoint descriptors are described relative to this orientation, and invariance to rotation is achieved.

To get orientation, the keypoint scale is used to select the Gaussian smoothed image L , then gradient magnitude, m , and orientation, θ , are computed. Computed gradient orientations of sample points form a histogram, and the highest peak in the histogram is located. This peak and any other local peak within 80% of the height of this peak is

used to create a keypoint with that orientation. Furthermore, some points might be assigned multiple orientations, and a parabola can be used to reach an interpolation of peak value from 3 closest histogram values to the peak are interpolated. This is done for all pixels around the keypoint; the histogram will have a peak at some point.

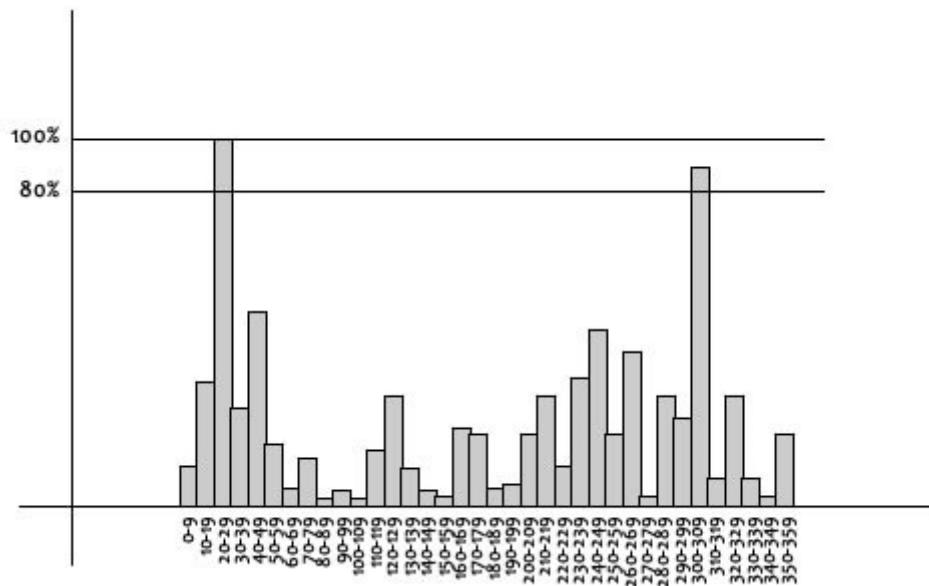


Figure 2.13: Keypoint Histogram (Utkarsh Sinha, 2016)

To compute the orientation, the highest peak in the histogram is picked, and each peak above 80% of this peak is also taken into account. It is shown with a line for 80% of the peak in Figure 2.13. Keypoints are created with the same location and scale, although their directions are different. The stability of matching increased with the contribution of this situation.

2.2.4 Keypoint Descriptor

Each keypoint now has a location, scale, and orientation. The local gradient data is used to generate keypoint descriptors for each keypoint in the local image region that is as distinct and invariant as feasible to changes in viewpoint and illumination. To do this, the gradient information is rotated to line up with the orientation of the keypoint and then weighted by a Gaussian with a variance of $1.5 * \text{keypoint scale}$. Visualization of keypoint descriptors is done by a set of histograms over a window centered on the keypoint.

In Figure 2.14, a 16x16 window around the keypoint is created and is divided into 16 sub-blocks of 4x4 size.

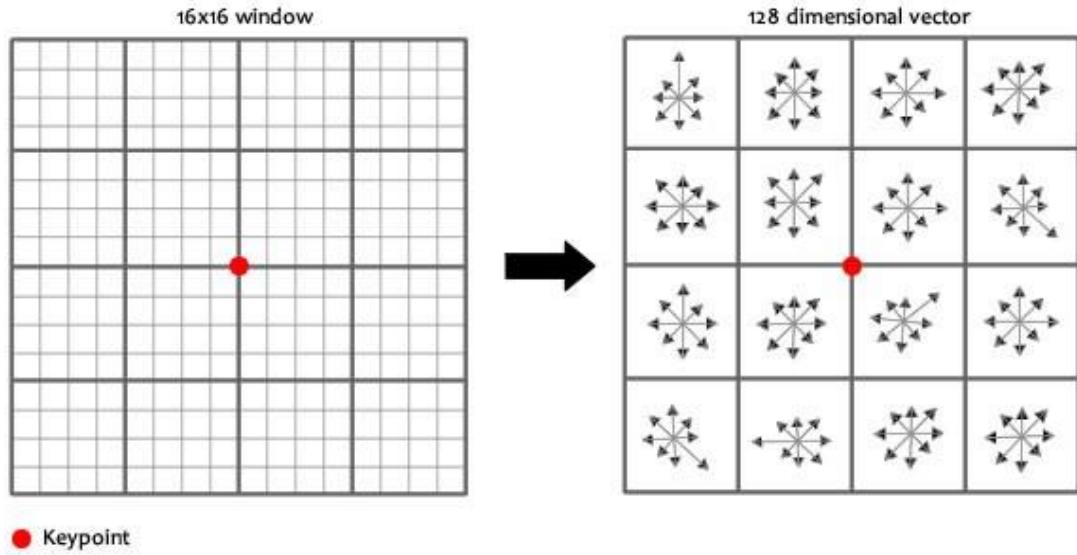


Figure 2.14: Keypoint Descriptor 16x16 Histogram (Tyagi, 2019)

Each sub-block consists of 8 bin orientation bins in Figure 2.15.

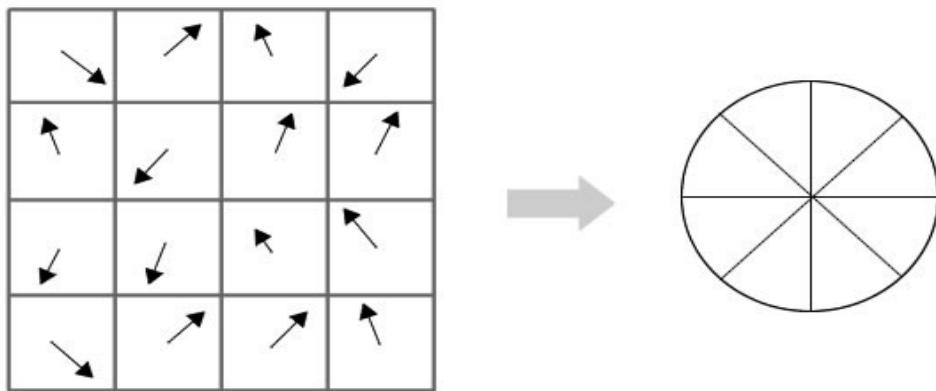


Figure 2.15: Histogram with sub-blocks (Tyagi, 2019)

In total, 4x4 descriptors over 16x16 sample array were used in practice. 4 direction represents the main compass directions, and other 4 represents the mid-points of these directions. As a result of 4x4x8 directions, it generates 128 bin values. It is represented as a feature vector of 128 elements to form a keypoint descriptor. This feature vector introduces a few complications and dependences. These are related to rotation and illumination. The feature vector uses gradient orientations which means a rotation of the image causes changing of all gradient operations. Another dependence is related to illumination; it is connected to the threshold that is used to eliminate keypoints.

Keypoint Matching

The closest neighbors of keypoints in two images are identified and matched. However, due to noise or other factors, the second closest match may be quite close to the first. The closest distance to second closest distance ratio is used in this scenario.

SIFT takes place in the correspondence search stage of the COLMAP. During this stage, image matching and recognition have proceeded, and distinctive image features, which are named SIFT features, are extracted from a set of reference images and stored in a database. Different images cause a challenge related to find a correlation between those extracted features of images and match them. To overcome this challenge, a new image is matched by individually comparing each feature of the new image to the prior database and identifying candidate matching features based on the Euclidean distance of their feature vectors. Lowe discusses fast nearest-neighbor algorithms that can perform this computation rapidly against large databases. (Lowe, 2004)

When memory and computational power are limited, SIFT must be modified to meet the requirements of the problem. Compression is applied to use memory efficiently. Compressed SIFT descriptors ensure that the system contains more visited maps to avoid the constant loading and unloading of maps. In addition, the time complexity of SIFT limits the use of the algorithm for online purposes. After SIFT descriptors are extracted, VLAD can be applied for further steps (Wei et al., 2015).

2.3 Multi-View Stereo

Sparse models are generated as outcome products of SfM, which is introduced in the previous section. Alongside the sparse model, camera poses and 3D points are recovered. Sparse models are not the best in case of precision of geometric primitives is needed, and densification method is required (Forsman, 2011). Because the reconstructed 3D model after SfM only contains the distinctive image features which match other images nicely.

Multi-View Stereo (MVS) is used to produce robust, accurate, and efficient dense models. MVS takes the registered images and produces dense and accurate models. These images are collected from studio conditions, internet, video, and unstructured image collections. Images are usually collected for image-based rendering,

classification, and localization applications. According to the needs of the application method, the implementation method can also change.

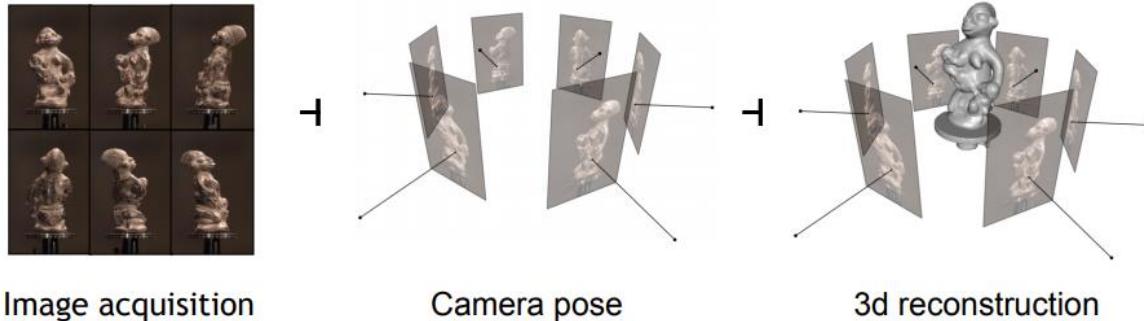


Figure 2.16: Multi-View Reconstruction pipeline (Slobodan Ilic, 2011)

MVS reconstruction pipeline in Figure 2.16 briefly consists of three steps. These are image acquisition, camera pose extraction, and generating 3D reconstruction. Image acquisition is completed with collected images, and the required camera poses are already known since the SfM process is completed and camera parameters for each image are computed along with the sparse model. Then 3D geometry of the scene is reconstructed by using the set of images and corresponding camera parameters. Additionally, the materials of the scene can be reconstructed.

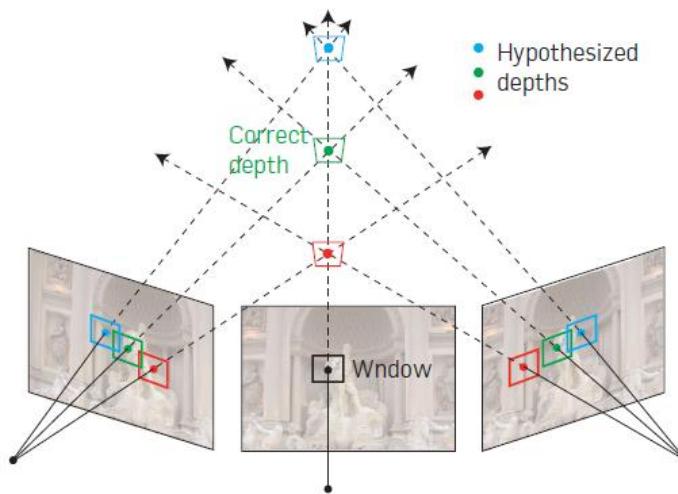


Figure 2.17: Window-based Multi-View Stereo algorithm (Agarwal et al., 2011)

In Figure 2.17, a set of images are used to find the correct depth. An infinite number of depth line vectors are created. All of them start from the viewing point. At each depth value, the window is projected into the other images. In this way, consistency

among textures at these image projections is computed. The criterion that shows the correct depth is at the true depth; the consistency score is maximum (Agarwal et al., 2011). This brief example is extended with different methods, such as dense pixelwise correspondence search.

Dense pixelwise correspondence search is the base of the stereo methods, and this search is related to illumination and viewing geometry conditions. These conditions define the setting of the problem that occurs in controlled or uncontrolled environments. MVS grasps multiple views to overcome the inherent occlusion problems of two-view approaches (Schönberger et al., 2016). The method used for MVS in COLMAP concurrently accounts for a diversity of photometric and geometric priorities, also improves the robustness and accuracy of Zheng's depth determination (Zheng et al., 2014).

COLMAP uses the output of the SfM in MVS. Thus, the workload of the MVS is reduced, and because of the calibration, which is done in SfM, results are accurate. COLMAP computes depth and normal information for every pixel inside an image. (Schoenberger, 2020a) The dense point cloud of the scene is produced by fusion of the depth and normal maps of multiple images in 3D. Pixelwise view selection with MVS is performed for depth and normal estimation and fusion.

3 Methodology

3.1 Toolkit

3.1.1 Docker

Docker is a service that uses operating system virtualization to deliver prepared software packages called containers. These containers are processes that are isolated from all other processes on the host machine and include their own operating system kernel, software, and libraries.

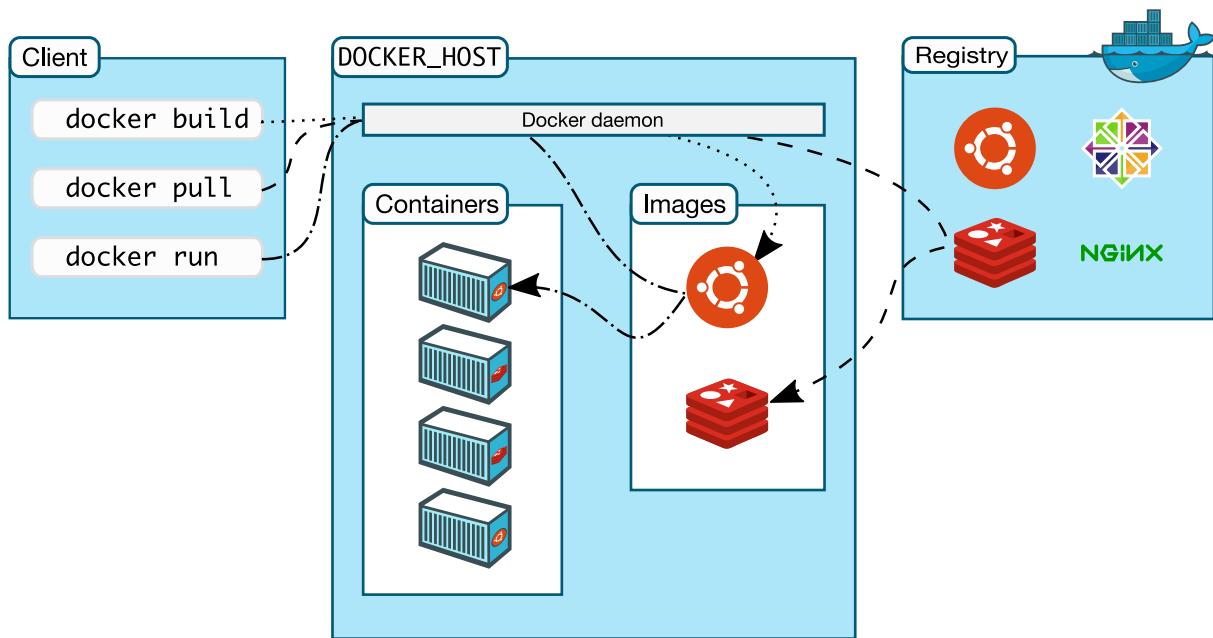


Figure 3.1: Docker architecture (Docker, 2018)

Client-server architecture is used in Docker, as seen in Figure 3.1. The client communicates with the Docker daemon, which has the responsibility of building, running, and distributing the Docker containers. When Docker pull and docker run commands are used, Daemon communicates with the registry. Docker registries are the storage for the Docker images. While running a container, the container uses a custom image. Images are built according to Dockerfile. This script includes instructions that define layers in the images. Additionally, GPU usage and volume mounting are done by definitions inside Dockerfile.

Dockerfile, which is used for COLMAP reconstruction, consists of a list of dependent images and requires software packages. Ripfreeworld COLMAP image is pulled from

Docker Hub, and a script is added to insert bash commands into the running container. The link for the image is “https://hub.docker.com/r/ripfreeworld/colmap_cuda10.2”. This script is used to start and configure reconstruction in COLMAP. A detailed description of this script is in section 3.3. Docker-build command uses the Dockerfile inside the specified directory.

```
docker build -t ahmethan/colmap:v0.5 .
```

Figure 3.2: Docker build command

“-t” flag is used to reference the version of the built image in Figure 3.2. Therefore, different versions of similar images can be built with the help of this flag. “.” means that the required Dockerfile location is the same as the terminal path. After the build process of the image is completed, the image is running by a container with the docker run command. To check an image is built correctly, the “docker images” command is used to list all of the images. In case of a need to remove one of the images, “docker rmi image_id” is used.

```
docker run \
-d \
--user $(id -u):$(id -g) \
-- userns=host \
-- net=host \
-- ipc=host \
-- name simplepinhole_hp_v5 \
-- gpus device=1 \
-e DISPLAY=$DISPLAY \
-- mount type=bind,source=/home/abasak/colmaptrials/office4/images,target=/tmp/images \
-- mount type=bind,source=/home/abasak/colmaptrials/office4/data,target=/tmp/data \
-v /tmp/.X11-unix:/tmp/.X11-unix:ro \
-v ${HOME}/.Xauthority:/home/$(whoami)/.Xauthority:ro \
-v /etc/passwd:/etc/passwd:ro \
-v /etc/group:/etc/group:ro \
ahmethan/colmap:v0.5
```

Figure 3.3: Docker Container Run Commands For COLMAP’s Reconstruction

In Figure 3.3, there are flags that are added below the docker run command. “-d” makes the container run in the detached mode, which allows the container to run in the background. Thus, when the root process inside the container is completed, containers exit. “-name” gives identification to the container. Moreover, graphic cards can be assigned to containers in case of multiple devices like our case. Pointy has two graphic cards, and they are available for multiple users. According to the graphic memory usage of these cards, the user must choose the one with free memory. “nvidia-smi” command is used to list the current situation of the system in Figure 3.4, and the user can choose the suitable card for a certain container.

```
abasak@TUBVCMS-Pointy: $ nvidia-smi
Sun May 30 04:08:47 2021
+-----+
| NVIDIA-SMI 460.73.01    Driver Version: 460.73.01    CUDA Version: 11.2 |
| Persistence-M| Bus-Id     Disp.A  Volatile Uncorr. ECC | | |
| GPU  Name      Perf Pwr:Usage/Cap| Memory-Usage | GPU-Util Compute M. |
| Fan  Temp  Perf  P2 / 260W |      5MiB / 48601MiB |      0%     Default |
|          |             |                  |                MIG M. |
+-----+
| 0  Quadro RTX 8000     Off  00000000:01:00.0 Off | Off | | |
| 34% 59C   P8   32W / 260W |      5MiB / 48601MiB |      0%     Default |
|          |             |                  |                N/A |
+-----+
| 1  Quadro RTX 8000     Off  00000000:4C:00.0 Off | Off | | |
| 55% 75C   P2   175W / 260W |      931MiB / 48598MiB |      99%    Default |
|          |             |                  |                N/A |
+-----+
+-----+
| Processes:
| GPU  GI CI     PID  Type  Process name          GPU Memory |
| ID   ID          |           Usage
|-----|
| 0    N/A N/A    2403   G  /usr/lib/xorg/Xorg        4MiB |
| 1    N/A N/A    2403   G  /usr/lib/xorg/Xorg        20MiB |
| 1    N/A N/A   321755  C  colmap                   907MiB |
+-----+
```

Figure 3.4: Percentage of usage for each GPU in the server

Docker offers several methods to store generated data by containers. These methods are volumes, bind mounts, and tmpfs mount, and the working structures of methods are shown in Figure 3.5.

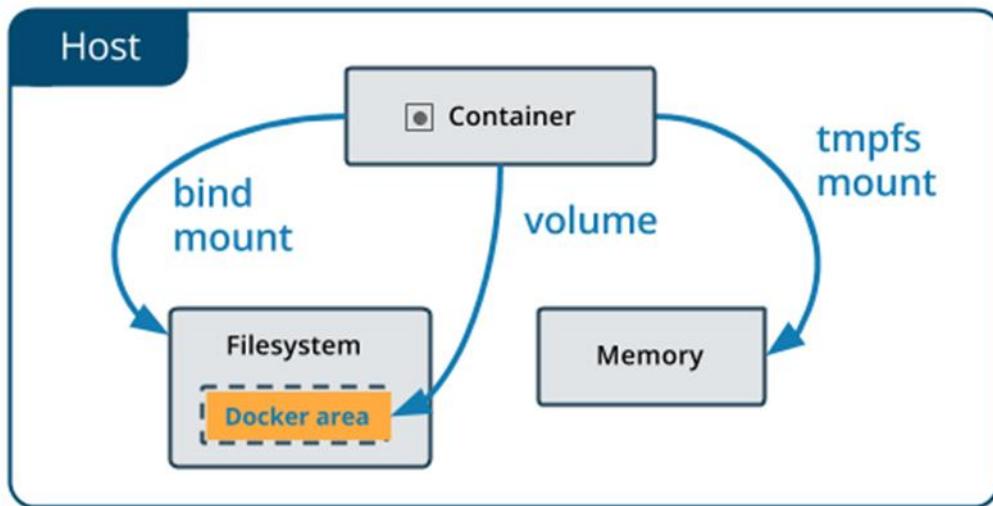


Figure 3.5: Docker volume system (Docker, 2020)

Volumes are significant to persist data in Docker and have an advantage over bind mounts in some aspect. Both methods need a mounting directory in the host machine and Docker container. As a difference between the two methods, volumes are maintained by Docker and are isolated from the core functionality of the host machine. On the other hand, bind mountings have a dependence on the directory structure and the operating system of the host machine. Although bind mounts are basic when it is compared to volumes, and they are performant. “—mount” or “-v” flags are used to assign which method is chosen. “—mount” is preferred because of recommendations in

docker hub which claims that the syntax is more clear. The source part indicates the directory path inside the host machine. The target indicates the directory inside the created container. In this thesis, bind mountings are used to transfer series of images into the container, and after reconstruction is completed, dense model and sparse model are reachable by using these volumes.

Running containers can be listed by using “docker ps” commands. Also, by adding -a flag, it can show existed containers as listed in Figure 3.6.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS
4e5ba4e948ba	docker_helios	"/usr/local/bin/heli..."	7 hours ago	Exited (137) 4 hours ago
d6df0a9b8fd9	docker_helios	"/usr/local/bin/heli..."	2 days ago	Exited (137) 2 days ago
5036272128fa	ydpan/workshop_ev	"/bin/bash"	5 days ago	Up 5 days
bc48c3ed24ff	ahmethan/colmap_mapper:v1.0	.. ./entrypoint.sh"	7 days ago	Exited (0) 7 days ago
a21e936fea29	ahmethan/colmap_mapper:v1.0	.. ./entrypoint.sh"	7 days ago	Exited (0) 7 days ago
48d7cf0b8d56	ahmethan/colmap:v6.0	.. ./entrypoint.sh"	8 days ago	Exited (0) 7 days ago
6c4a5b5992f3	ahmethan/colmap:v6.0	.. ./entrypoint.sh"	8 days ago	Exited (0) 7 days ago
46b38360a0f7	ahmethan/colmap_mapper:v1.6	.. ./entrypoint.sh"	10 days ago	Exited (0) 10 days ago
13a02a7f5266	ahmethan/colmap_mapper:v1.5	.. ./entrypoint.sh"	10 days ago	Exited (0) 10 days ago

Figure 3.6: “docker ps -a” command

“docker stop container_id” is used to stop a running process during the runtime. Existed containers can be removed by using “docker rm container_id”.

Additionally, after the container is left to running, the process can be observed by using “docker logs -f container_id”. This command gives the opportunity to the user to debug the error and follow every step of the reconstruction.

3.1.2 COLMAP

COLMAP is a general-purpose SfM and MVS pipeline with a graphical and command-line interface. A wide range of features comes with the software, which is focused on the reconstruction of ordered and unordered image collections.

Current SfM algorithms fail to produce fully satisfactory results in terms of completeness and robustness. As a result of this, the system cannot register large fraction images, or broken models are observed because of misregistration or drift. The reasons that cause these outcomes occur during the SfM steps. In the correspondence search part, it should not produce an incomplete scene graph. Incompleteness affects the connectivity for complete models and the required redundancy for reliable estimation. In the reconstruction part, the error can exist in the image registration or triangulation step, and it may be related to missing or inaccurate scene structure. Therefore, images cannot be registered properly (Schönberger & Frahm, 2016).

A new algorithm which is presented by Schönberger, comes with a solution to these known problems. Firstly, COLMAP introduces a geometric verification strategy that is used to increase the robustness of the initialization and triangulation parts. Secondly, a new best view selection is applied to maximize the robustness and accuracy of the incremental reconstruction process. Thirdly, a robust triangulation method is used to produce a more complete scene structure than the current algorithm with a reduced amount of computational effort. Fourth, an iterative Bundle Adjustment, re-triangulation, and outlier filtering strategy improve completeness and accuracy by mitigating drift effects. Finally, a more efficient BA parameterization for dense photo collections through redundant view mining is applied. All of these changes make the new algorithm surpass the current state-of-the-art systems Bundler and VisualSfM.

COLMAP has two kinds of interfaces which are GUI and command-line interface. GUI is used to show points and camera positions of reconstructed models. The command-line interface can be controlled on the terminal and has specific flags to change parameters in each step of reconstruction.

3.1.3 Connection with Server

Reconstruction of series of images in COLMAP needs a certain amount of computation power. The Chair of Computational Modelling and Simulation has a server that is dedicated to the high-performance required computation. The technical specifications of Pointy are capable of solving these computations. This server is named Pointy, and it has two Quadro RTX 8000 GPUs with 48 GB graphic memory and an AMD Ryzen Threadripper 3900X 64-Core processor. Connection to this server is made by SSH File Transfer Protocol (SFTP). For authentication purposes, the user must have credentials that are needed for connection. These are a public key and a private key. Keys are created in the terminal by this line on Linux in Figure 3.7.

```
$ ssh-keygen -t rsa -b 4096 -f ~/.ssh/vps-cloud.web-server.key -C  
"My web-server key"
```

Figure 3.7: Create an SSH keypair (Gite Vivek, 2021)

“-t rsa” specifies the type of key to create. “dsa”, “ecdsa”, “ed25519”, “rsa1” (for protocol version 1 of rsa) and “rsa” (for protocol version 2) are the possible types that can be defined. “-b” flag species the number of bits in the key to creating. In this example,

4096 bits version is preferred. The default value for the RSA key is 2048 bits. “-f” flag specifies the filename of the key file, and “-c” flag is used to set a new comment.

The public key is sent to the admin of the server, and the private key is used to make the connection to the server. The SFTP server configuration includes the host key. When the client requests the host key, the server must send the host key to the client (*SFTP Server Overview - IBM Documentation, 2021*).

Admin of server gives a username besides only taking the public key, and the client uses this username and private key to transfer files. Firstly, the private key must be transformed into a format by using Putty. Then, the user is able to make the transfer by using an SFTP connection.

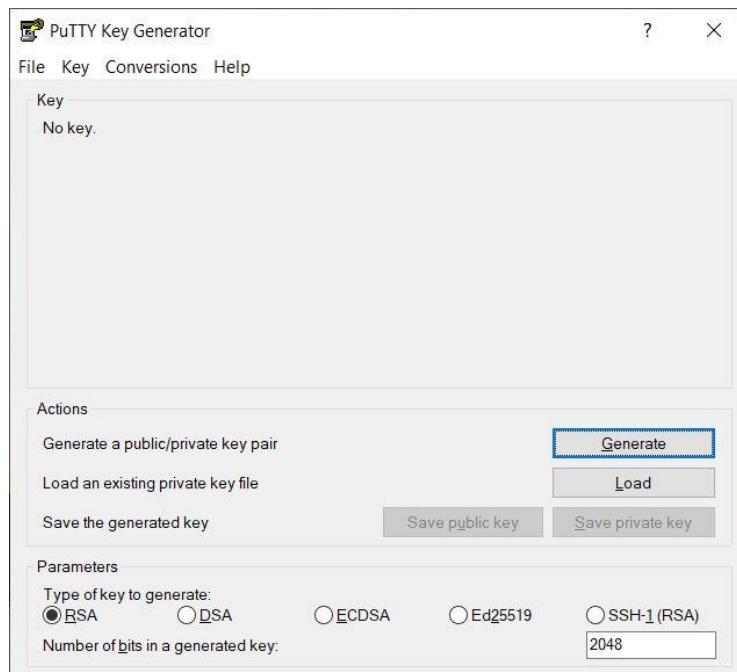


Figure 3.8: PuTTY Key Generator

PuTTY key generator in Figure 3.8 is used to convert the private ssh key into the “.ppk” type. Since FileZilla needs this type of private key to connect the server. File transfer between Pointy and the local machine is maintained by FileZilla. FileZilla is an open-source, cross-platform FTP application. It performs the file transfer using FTP and encrypted FTP such as SFTP like the connection of Pointy (*FileZilla - Wikipedia, 2021*).

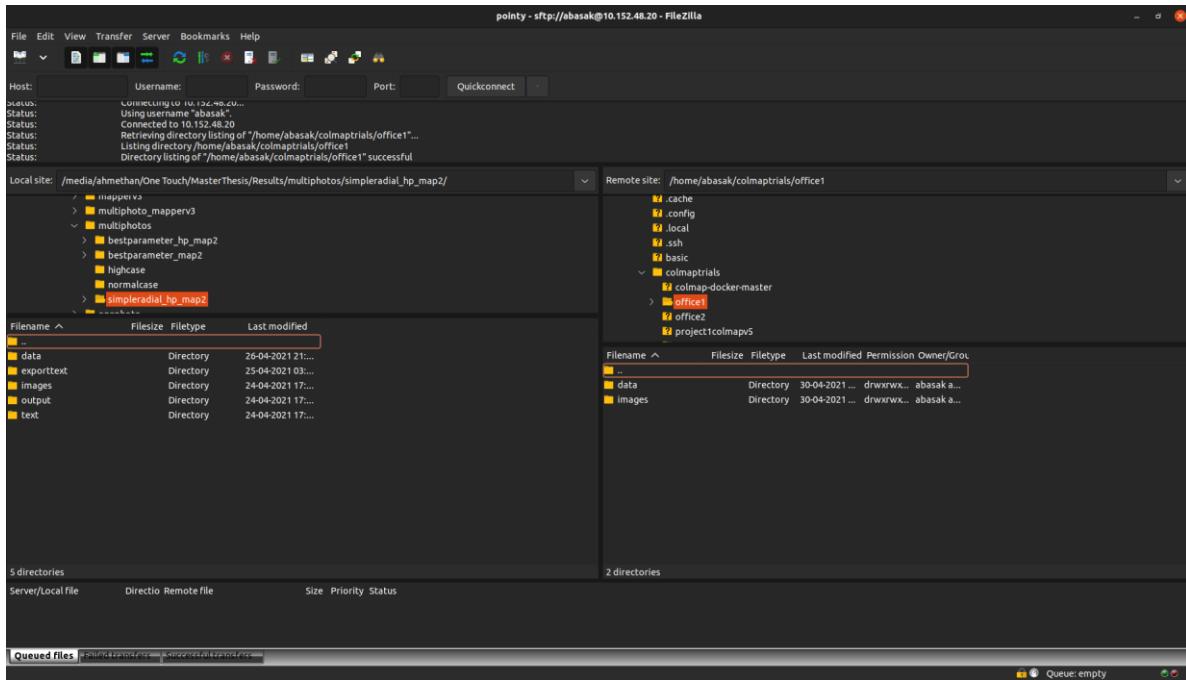


Figure 3.9: FileZilla interface

In Figure 3.9, the FileZilla interface is seen with two tabs. The left tab is the explorer for the file system of the local machine, and the right tab is for the server's file system. Files can be easily transferred via Drag and Drop, and if there are lots of files for the transfer, it creates a queue for files. The transfer between the server and the local machine can be paused or continued. Also, new folders and files can be created and moved to different locations on the server. These features are used to create the folder that is needed for reconstruction.

3.1.4 CloudCompare

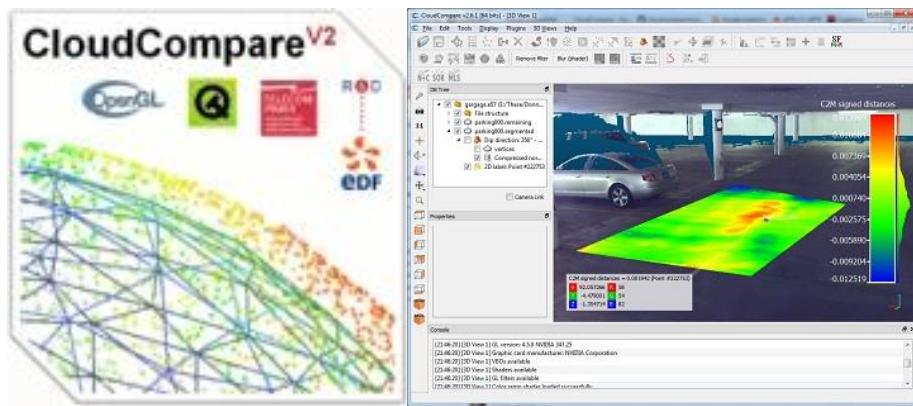


Figure 3.10: CloudCompare Logo and GUI (Daniel Girardeau-Montaut, 2011)

CloudCompare is a 3D point cloud (and triangular mesh) editing and processing software. It was created with the intention of comparing two dense 3D point clouds (such as those obtained with a laser scanner) or a point cloud and a triangular mesh. It makes

use of a dedicated octree structure for this purpose. After that, it was expanded to include more general point cloud processing software, including many advanced algorithms (registration, resampling, color/normal/scalar fields handling, statistics computation, sensor management, interactive or automatic segmentation, display enhancement, etc.) (CloudCompare, 2016).

3.2 COLMAP Reconstruction

The reconstruction process is split into two parts. COLMAP starts the sparse reconstruction by loading all extracted data from the database into memory and seeding it with an initial image pair. The scene is then gradually expanded by adding fresh images and triangulating additional points. Following the reconstruction of a sparse representation of the scene and camera poses from the input image, the reconstruction process continued with dense reconstruction. MVS now has the ability to recover denser scene geometry. COLMAP features an integrated dense reconstruction pipeline that generates depth and normal maps for all registered images before fusing them into a dense point cloud with normal information (Schoenberger, 2020a). Inside the dense reconstruction process, the first process is undistorting the images. Secondly, the depth and the normal maps are computed by using stereo. Thirdly, these maps are fused to a point cloud.

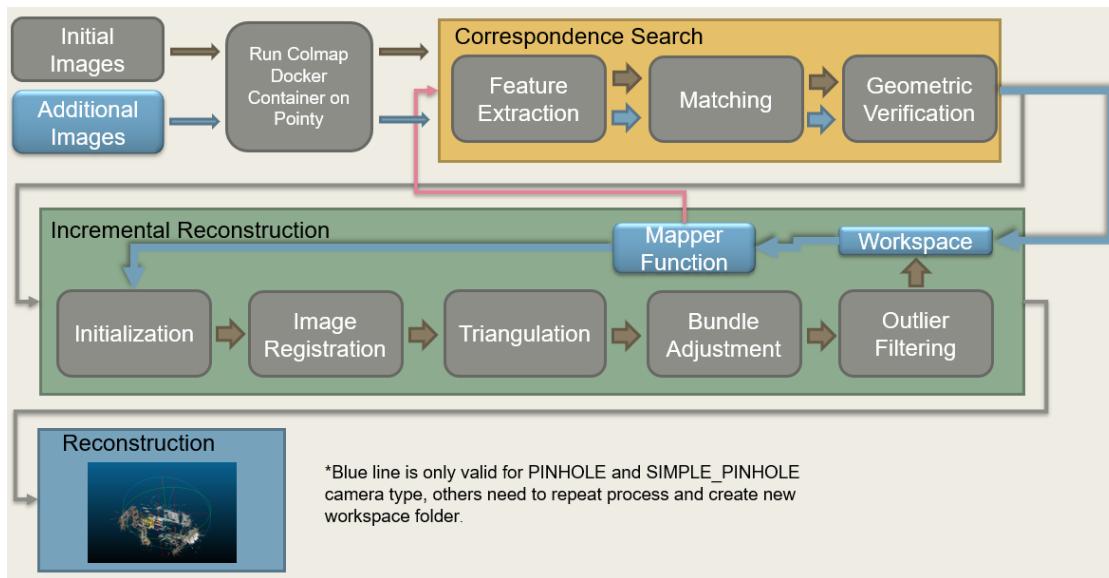


Figure 3.11: COLMAP Pipeline with Mapper function

The image registration and mapper function are related to the reconstruction step. The difference between these processes is triangulation. For image registration, firstly, new features of the added images are extracted. Then it matches them with the existing

images in the database. Thus, all of the new images are registered into the model. In the case of more accurate result is needed, the mapper function is preferred, as seen in Figure 3.11. It enables COLMAP to restart or continue the reconstruction process to find the scene, and the camera poses of the added images instead of just registering the added images into the existed database.

By using the GUI of COLMAP, the user can start reconstruction and view the results of reconstruction. In this thesis, the command-line interface is preferred because of several reasons. Firstly, Pointy is a server and does not offer Linux's GUI to operate. Secondly, the command-line interface works with docker containers and gives the user freedom to change the parameters of each step of reconstruction. A script called "Entrypoint.sh" is used along with the building of the docker container, which includes COLMAP. This script includes paths of images, database, and initialization files inside the container.

3.3 Initialize the reconstruction on the server

Images are taken from a video as screenshots. Frame per second value decides the number of images. A high FPS value makes the number of images higher. These images are transferred into the Pointy. Reconstruction in COLMAP can be done automatically, or every step of the reconstruction sets manually. These steps require a project.ini file that includes the parameter values regarding the process of the reconstruction in that specific step.

In Figure 3.12, available commands of COLMAP's feature extractor are listed by using the -h, -help flag. COLMAP has two options in terms of setting configuration. One of them is using the command line with listed flags and changing every parameter individually. Another option is preparing an initialization file with assigned values and changing parameters inside this file. Additionally, paths related to project files and the database is given inside the file. These paths are the file locations inside containers.

The camera model is also specified in this file. --ImageReader from the defined models in COLMAP documentation. Pinhole and radial camera models are chosen during trials. Pinhole cameras are basic because it uses one and two focal length parameters even with a setting of undistorted images. Therefore, the radial camera model is preferred to cover more possibility of having images that are taken with different cameras. This model is quite useful with cases with unknown camera intrinsics and images which have different camera calibration.

```
$ colmap feature_extractor -h
Options can either be specified via command-line or by defining
them in a .ini project file passed to `--project_path`.

-h [ --help ]
--project_path arg
--database_path arg
--image_path arg
--image_list_path arg
--ImageReader.camera_model arg (=SIMPLE_RADIAL)
--ImageReader.single_camera arg (=0)
--ImageReader.camera_params arg
--ImageReader.default_focal_length_factor arg (=1.2)
--SiftExtraction.num_threads arg (=1)
--SiftExtraction.use_gpu arg (=1)
--SiftExtraction.gpu_index arg (=1)
--SiftExtraction.max_image_size arg (=3200)
--SiftExtraction.max_num_features arg (=8192)
--SiftExtraction.first_octave arg (=1)
--SiftExtraction.num_octaves arg (=4)
--SiftExtraction.octave_resolution arg (=3)
--SiftExtraction.peak_threshold arg (=0.00666666666666671)
--SiftExtraction.edge_threshold arg (>0)
--SiftExtraction.estimate_affine_shape arg (=0)
--SiftExtraction.max_num_orientations arg (=2)
--SiftExtraction.upright arg (=0)
--SiftExtraction.domain_size_pooling arg (=0)
--SiftExtraction.dsp_min_scale arg (=0.1666666666666666)
--SiftExtraction.dsp_max_scale arg (=3)
--SiftExtraction.dsp_num_scales arg (=10)
```

Figure 3.12: COLMAP’s Feature Extractor Commands

Parameters that are related to SIFT theory are defined in the feature extractor. Domain size pooling (DSP) is a development on SIFT that improves matching and feature descriptors.

After the project initialization file is prepared with decided parameters, a container that is capable of running COLMAP with the needed reconstruction steps is built. Dockerfile in Appendix A.1 is used to build the container. Inside this docker file, Entrypoint.sh file is copied from the directory, and the commands inside the file start the reconstruction sequentially. For the reconstruction, the file in Appendix A.2 is used. Every step of the reconstruction can be followed by mentioned docker command (“docker logs -f”) to view the terminal inside the container.

Docker cp command is used to copy the stored data inside the container to the host machine. The workspace folder includes dense and sparse models. Inside the dense model folder, point cloud .ply file, depth maps, normal maps, and consistency graphs are created after the dense stereo stage, which includes patch match stereo and stereo fusion. In Figure 3.13, the section between cp and “:” represents the container ID. The remaining part is the path of files inside the container, which will be copied into the server’s hard disk separately.

```
docker cp 9a7f35cb54dd:/tmp/workspace .
```

Figure 3.13: Usage of Docker cp command

This workspace folder in Figure 3.14 includes dense and sparse models, depth and consistency maps, and reconstructed point clouds.

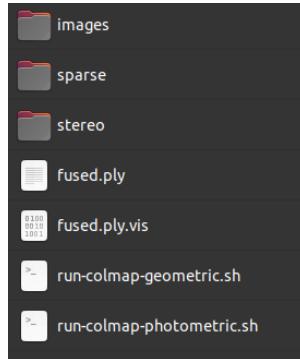


Figure 3.14: Workspace folder

3.4 Adding New Images to Reconstructed Model

COLMAP includes an option to make user adds new images to existing reconstruction. This option follows a route that is similar to first registration. Firstly, features in new images are extracted and registered into the database, which is created during the first reconstruction. New images are listed in a text file per line with image names. Then, they are registered to the model. COLMAP offers two methods to add new images. The first method follows a path that takes new images and extracts the features from the images. Finally, it registers them into the database. If the accuracy of the reconstruction is a significant criterion for the registration, the second method, the mapper function, is preferred since it is a more accurate alternative to only registering images to the model. With the mapper function, reconstruction is restarted or continued, and it offers an accurate image registration with triangulation. To execute the mapper function inside a container, folders should be filled with files according to the structure in Figure 3.15. The images folder includes both the images from the first reconstruction and newly added images. The data folder includes a database.db file and project initialization file of the previous reconstruction. The text folder includes a text file of new image names. Previously reconstructed dense and sparse models are stored in the workspace folder.

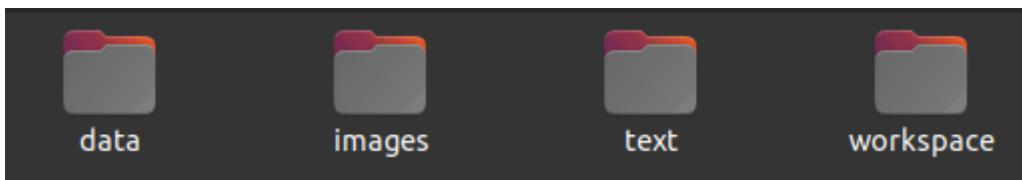


Figure 3.15: File system for the mapper

A container for the mapper is prepared apart from the reconstruction because the mapper function is called by using “colmap mapper”. The new Entrypoint.sh file in Appendix A.3 is written and used inside docker build operation to create the container that is adding new images into existed reconstruction.

In Figure 3.16, the container is prepared to run with volume mountings according to the described file structure. –d flag starts the container in detached mode, the –name flag gives a container name, and the –gpus flag is used to select a GPU. The last line represents the image that runs in the container.

```
docker run \
-d \
--user $(id -u):$(id -g) \
--userns=host \
--net=host \
--ipc=host \
--name simplepinhole_hp_mmap4 \
--gpus device=1 \
-e DISPLAY=$DISPLAY \
--mount type=bind,source=/home/abasaki/colmaptrials/office6/images,target=/tmp/images \
--mount type=bind,source=/home/abasaki/colmaptrials/office6/data,target=/tmp/data \
--mount type=bind,source=/home/abasaki/colmaptrials/office6/workspace,target=/tmp/workspace \
--mount type=bind,source=/home/abasaki/colmaptrials/office6/text,target=/tmp/text \
-v /tmp/.X11-unix/:/tmp/.X11-unix:ro \
-v ${HOME}/.Xauthority:/home/${whoami}/.Xauthority:ro \
-v /etc/passwd:/etc/passwd:ro \
-v /etc/group:/etc/group:ro \
ahmethan/colmap_mapper:v0.4
```

Figure 3.16: Docker Container Run Commands For COLMAP’s Mapper Function

The results of the mapper can be checked by opening the reconstructed point cloud on COLMAP’s GUI in my local computer. Also, new images and camera data should be registered into the database of the reconstruction. Added images are named as image455.jpeg to image463.jpeg. In Figure 3.17, matching features and cameras are listed in multiple tabs of DB Browser for SQLite, which is a software used to open SQLite database files on Linux.

Database Structure			
	Type	Schema	
Tables (7)			
cameras			
camera_id	INTEGER	"camera_id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, model INTEGER NOT NULL, width INTEGER NOT NULL, height INTEGER NOT NULL, prior_focal_length REAL, prior_width REAL, prior_height REAL, params BLOB	
model	INTEGER	"model" INTEGER NOT NULL	
width	INTEGER	"width" INTEGER NOT NULL	
height	INTEGER	"height" INTEGER NOT NULL	
params	BLOB	"params" BLOB	
descriptors			
image_id	INTEGER	CREATE TABLE descriptors (image_id INTEGER PRIMARY KEY NOT NULL, rows INTEGER NOT NULL, cols INTEGER NOT NULL, data BLOB)	
rows	INTEGER	"rows" INTEGER NOT NULL	
cols	INTEGER	"cols" INTEGER NOT NULL	
data	BLOB	"data" BLOB	
images			
image_id	INTEGER	CREATE TABLE images (image_id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, name TEXT NOT NULL UNIQUE, camera_id INTEGER NOT NULL, prior_name REAL)	
name	TEXT	"name" TEXT NOT NULL UNIQUE	
camera_id	INTEGER	"camera_id" INTEGER NOT NULL	
prior_qw	REAL	"prior_qw" REAL	
prior_qx	REAL	"prior_qx" REAL	
prior_qy	REAL	"prior_qy" REAL	
prior_qz	REAL	"prior_qz" REAL	
prior_tx	REAL	"prior_tx" REAL	
prior_ty	REAL	"prior_ty" REAL	
prior_tz	REAL	"prior_tz" REAL	
keypoints			
image_id	INTEGER	CREATE TABLE keypoints (image_id INTEGER PRIMARY KEY NOT NULL, rows INTEGER NOT NULL, cols INTEGER NOT NULL, data BLOB)	
rows	INTEGER	"rows" INTEGER NOT NULL	
cols	INTEGER	"cols" INTEGER NOT NULL	
data	BLOB	"data" BLOB	
matches			
pair_id	INTEGER	CREATE TABLE matches (pair_id INTEGER PRIMARY KEY NOT NULL, rows INTEGER NOT NULL, cols INTEGER NOT NULL, data BLOB)	
rows	INTEGER	"rows" INTEGER NOT NULL	
cols	INTEGER	"cols" INTEGER NOT NULL	
data	BLOB	"data" BLOB	
solar_sequence			
two_view_geometries			
Indices (1)			
index_name		CREATE UNIQUE INDEX index_name ON images(name)	
Views (0)			
Triggers (0)			

Figure 3.17: DB Browser for SQLite

The reconstruction is visualized by opening the model in COLMAP GUI after every step is completed in Figures 3.18 and 3.19.



Figure 3.18: COLMAP Visualization of Sparse model and Camera Locations



Figure 3.19: COLMAP Visualization of Generated Point Cloud

3.5 Alignment of Point Clouds

Point cloud of the CMS and videos of office room 3218 are given by supervisors for usage in this study in Figure 3.20. This point cloud is generated by using laser scanners. It includes a large number of points with regularly scatter behavior than COLMAP's reconstructed point cloud.

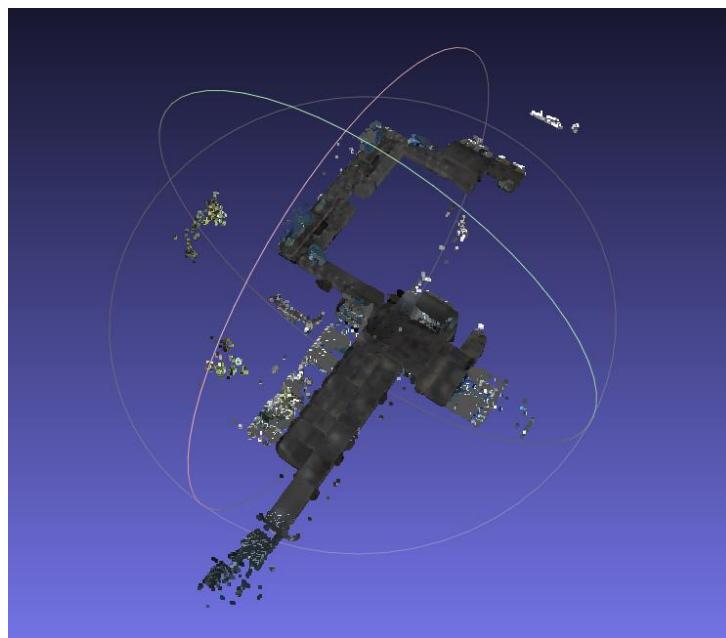


Figure 3.20: Laser-scanned point cloud without any cutting

Because of the number of points, it is hard to visualize in terms of hardware requirements. MeshLab, which is another software dedicated to processing and editing

triangular meshes and point clouds, used to visualize and edit this point cloud. Firstly, sampling is applied to laser-scanned point cloud to reduce the computational effort of the local computer to visualize the point cloud. Therefore, the local computer is able to make rotation and move along the point cloud. Secondly, the location of the office room 3218 is found. Then point cloud is cut by MeshLab tools. Points that are unrelated to the room are picked by using the Select Vertices tool. After this operation, these points are deleted by Delete the current set of selected vertices tool. In Figure 3.21, the blue square indicates the Select Vertices tool, and the green square indicates Delete the current set of selected vertices.



Figure 3.21: MeshLab Toolbar

Unrelated points are the points located outside of the room and the ceiling of the room. These points are deleted by explained tools.

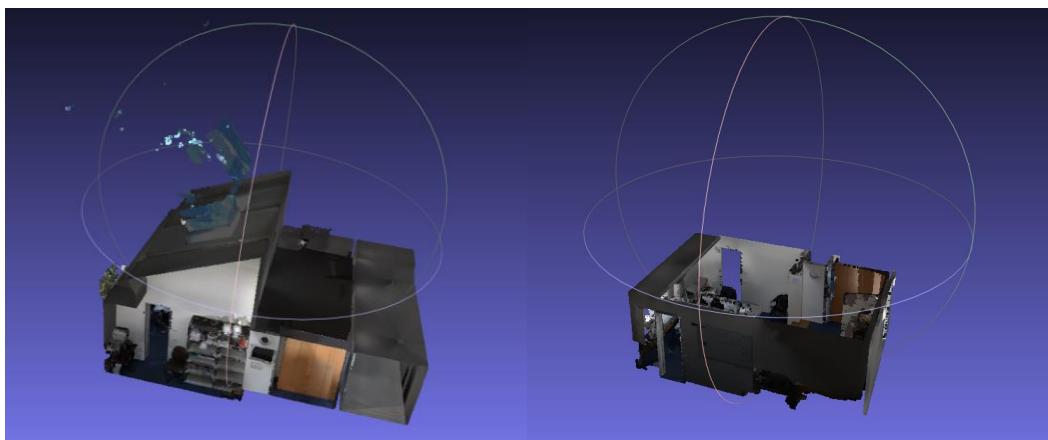


Figure 3.22: Left: Before Deleting Points Right: After Deleting Points

This edited point cloud needs to align with the COLMAP's reconstructed point cloud, as seen in Figure 3.22.

Firstly, MeshLab's align tool, which is shown in Figure 3.23, is used, and the following steps are applied. The user needs to choose one point cloud as a reference by using the Manual Rough Glueing option from the same window.

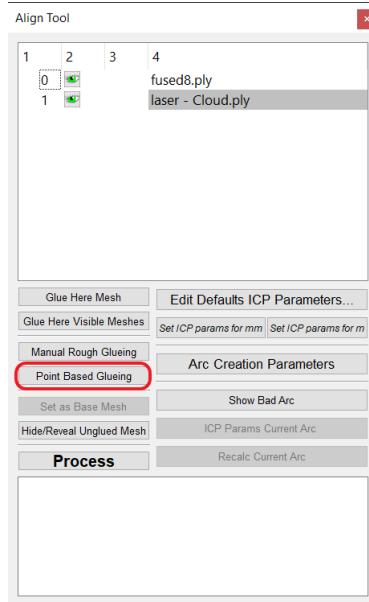


Figure 3.23: Align Tool Window of MeshLab

Point-Based Glueing needs at least 4 points in each point cloud to complete alignment. In Figure 3.24, there are three additional options at the bottom of the window. Allow scaling option is used to make the models be the same size.

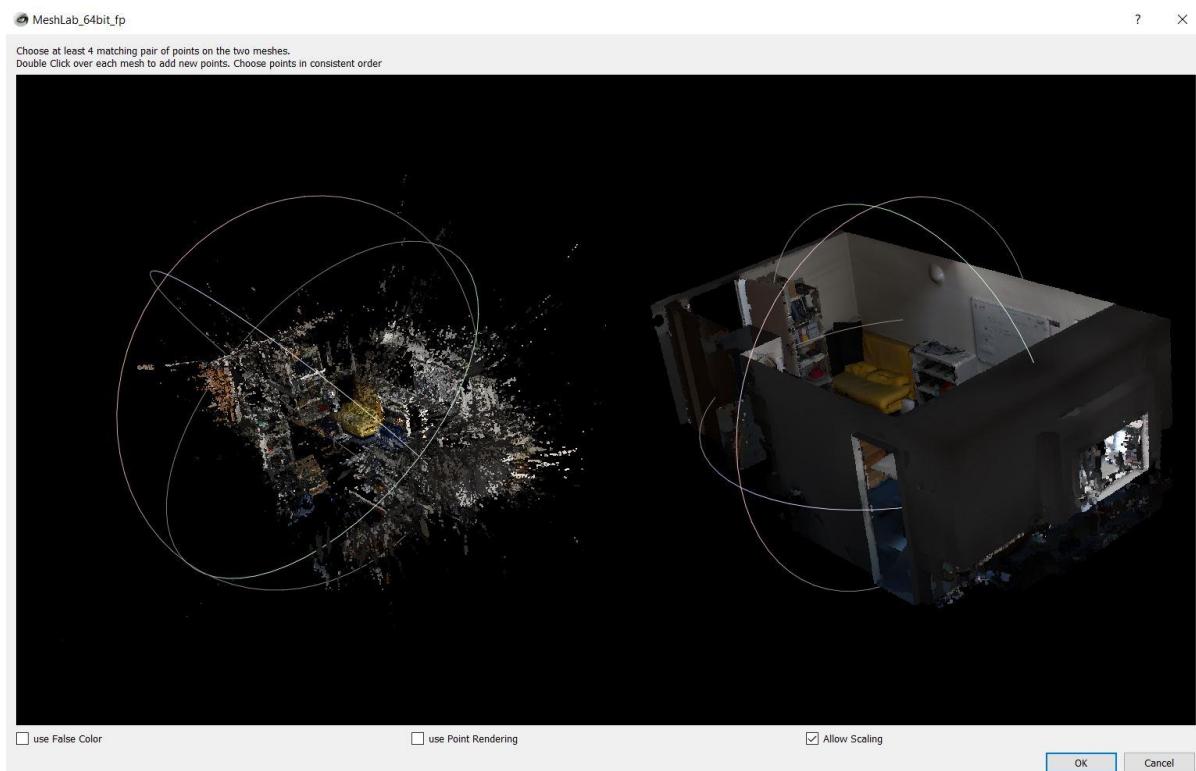


Figure 3.24: Point-Based Glueing in MeshLab

After points are selected from both point clouds, the user presses the Process button in Figure 3.23, and the alignment starts with computing overlaps between point clouds. The problem with MeshLab occurs in this step. The result of the process keeps giving errors with an explanation that says there is no successful arc among candidate alignment arcs in Figure 3.25.

```
Starting Processing of 2 glued meshes out
of 2 meshes
Computing Overlaps 2 glued meshes...
Arc with good overlap      1 (on      1)
    0 preserved 0 Recalc
( 1/ 1) 0 -> 1 Failed Alignment of one
arc Too much scale

Failure. No successful arc among
candidate Alignment arcs. Nothing Done.
```

Figure 3.25: MeshLab Error Text

The resultant point cloud is shown in Figure 3.26. Alignment is not completed properly, and MeshLab is not successful in matching pair of points.

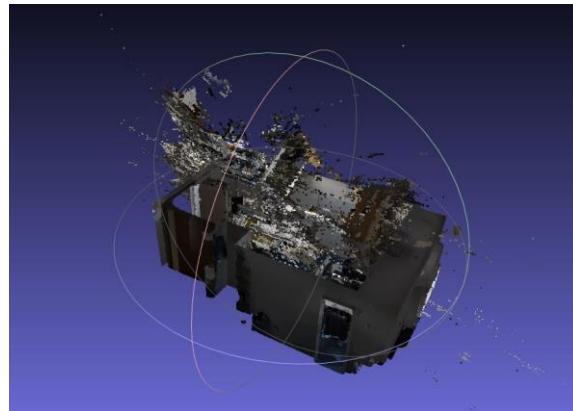


Figure 3.26: Unsuccessful Alignment of Point Cloud in MeshLab

This problem is solved by switching to another software called CloudCompare. Similar to MeshLab, alignment of the point clouds are made by picking points from two-point clouds. For alignment in CloudCompare, the red squared tool is used in Figure 3.27. This tool needs at least picked equivalent point pairs for alignment



Figure 3.27: CloudCompare Toolbox

These point clouds are referred to as align and reference. In Figure 3.28, points that are chosen from the “reference” are labeled as RX, and points that are chosen from the “align” are labeled as AX during the process.

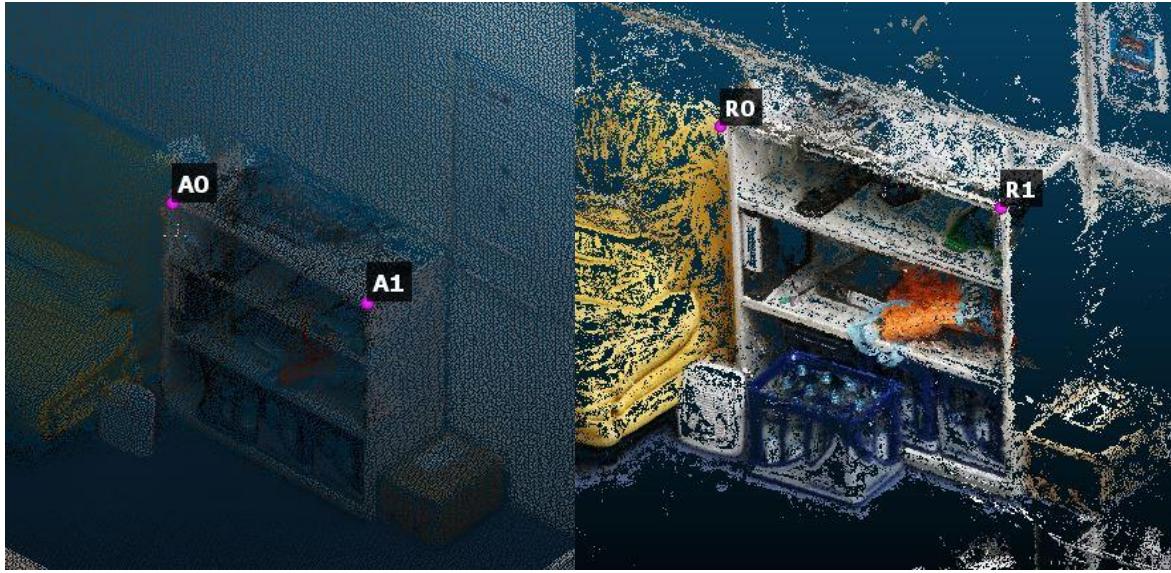


Figure 3.28: Left: Picking Points From “Align” Point Cloud Right: Picking Point From “Reference” Point Cloud

Coordinates of points with their labels are listed in Figure 3.29. Additionally, the scale of the point clouds may need an adjustment. Thus, adjust scale option of the Cloud-Compare is used.

	X	Y	Z	Error	
A0	-29.485130	-39.876102	-0.597922	0.224315	<input checked="" type="checkbox"/>
A1	-30.435888	-39.879490	-0.609162	0.173755	<input checked="" type="checkbox"/>
A2	-28.131905	-38.532368	0.351612	0.333373	<input checked="" type="checkbox"/>
A3	-28.133575	-39.285152	0.365073	0.332075	<input checked="" type="checkbox"/>

	X	Y	Z	Error	
R0	-1.188252	0.983044	4.061876	0.224315	<input checked="" type="checkbox"/>
R1	0.607494	1.017157	2.903193	0.173755	<input checked="" type="checkbox"/>
R2	-4.644846	-1.740544	2.614900	0.333373	<input checked="" type="checkbox"/>
R3	-3.982315	-1.021197	4.578679	0.332075	<input checked="" type="checkbox"/>

show ‘align’ entities

show ‘reference’ entities

adjust scale
Rotation Tx Ty Tz

auto update zoom

Figure 3.29: Align Tool Windows of CloudCompare

After alignment is completed, CloudCompare computes a transformation matrix that includes an integrated scaling parameter that is defined between the aligned point cloud and the reference point cloud, as is seen in Figure 3.30.

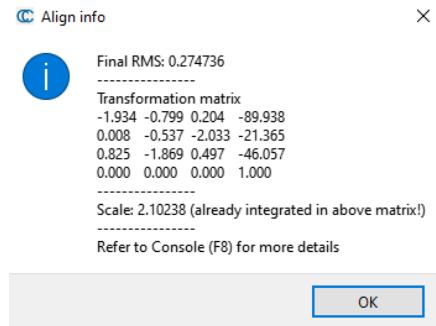


Figure 3.30: Generated Transformation Matrix in CloudCompare

At the end of the alignment process, aligned point clouds are visualized in Figure 3.31. There are some objects that are not scanned in laser. Apart from these, matches between the remaining points are visible.

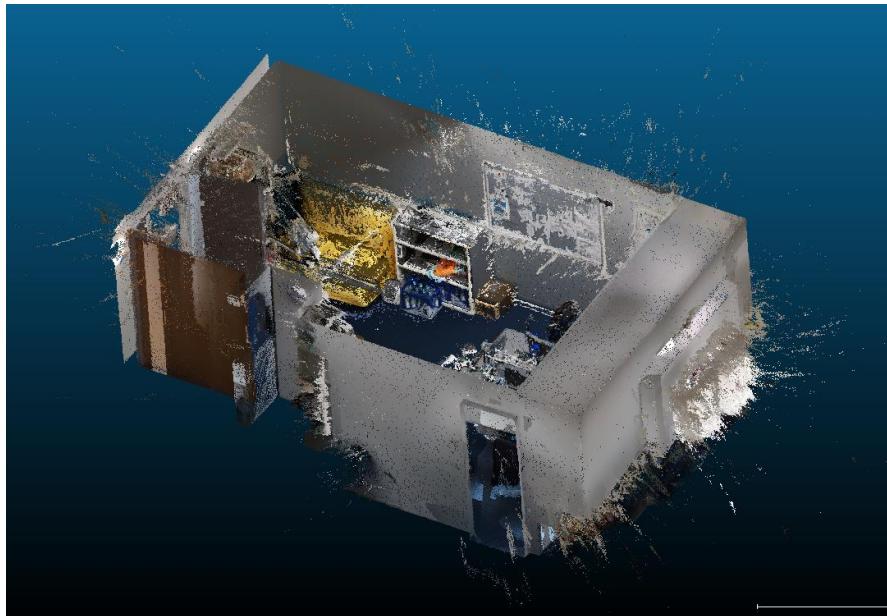


Figure 3.31: Aligned Point Clouds in CloudCompare

To understand which regions of the room have better-represented reconstruction, they are colored with different colors in Figure 3.32. The red color is used for the laser-scanned point cloud, and the green color is used for the reconstructed point cloud.

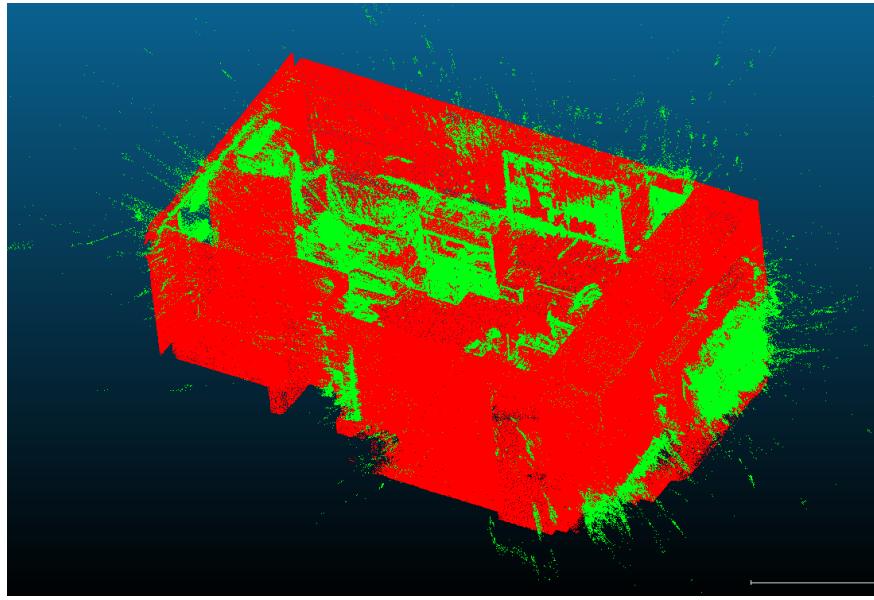


Figure 3.32: Colored Point Clouds in CloudCompare

3.6 Finding Camera Poses

Another way of visualizing the reconstructed point cloud is using the visualize_model.py python script that comes with COLMAP installation. This script opens the exported text file format of the COLMAP and visualizes reconstructed points and camera poses in Figure 3.33. The exported text file format includes cameras.txt that is filled with a list of the cameras and their data, points3D.txt is filled with coordinates of points and related image information, and images.txt is filled with a list of images used in the reconstruction. The id of the camera that the image is taken and 3D points generated from the related images are also in the images text file.

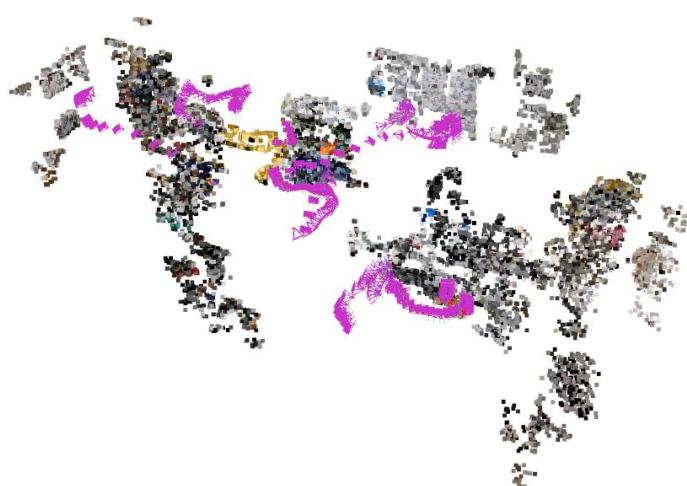


Figure 3.33: Point Cloud Visualized with Python Open3D library

For further processes, this script needs a modification to visualize aligned laser-scanned point cloud and camera poses of reconstruction in the same space. Also, the laser-scanned point cloud is converted into an exported text file format. To make this conversion, COLMAP GUI's import model option is used in Figure 3.35. After point cloud is imported as a model, GUI's export the model as a text file option is used, and the resultant files are created, as is seen in Figure 3.34.

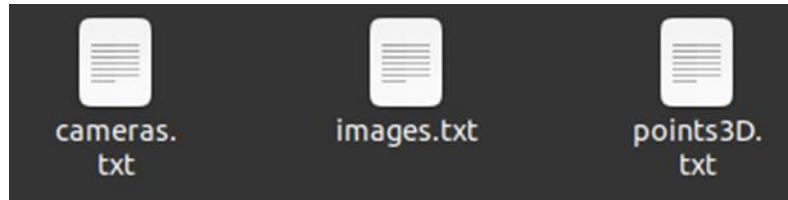


Figure 3.34: Exported Text File Format of the model

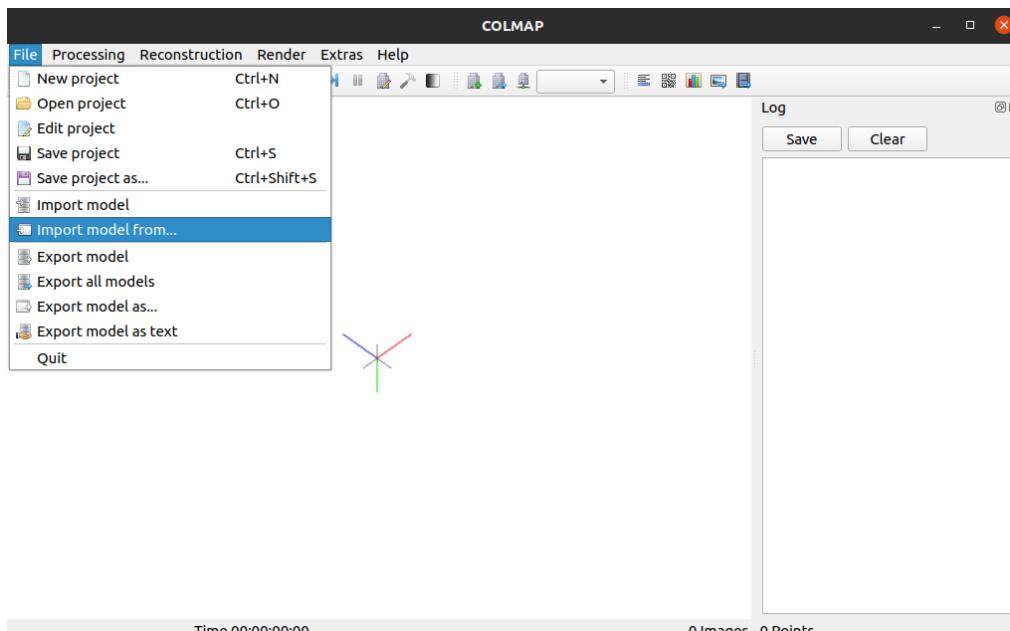


Figure 3.35: COLMAP GUI Features for importing and exporting models

The laser-scanned point cloud and the reconstructed point cloud are aligned in the previous step. After alignment, the coordinate system of both point clouds is assumed as same. Therefore, they can be visualized in the same space, and they will show the correct viewpoint of cameras. In the first place, `visualize_model.py`, which is written by COLMAP developer, needs some extension of functions to visualize laser-point cloud. The default version of this python script is written to visualize the reconstructed point clouds with all the camera positions. The first added features are made to visualize camera position with pose direction inside the laser-scanned point cloud instead of the reconstructed point cloud. This modified script is firstly used for the result of the mapper function. The aim is the finding the new images inside the reconstruction because

COLMAP dense reconstruction does not add every added image into the final reconstruction with respect to parameters that are used in Patch Match Sampling. To find out the number of added images inside the resultant reconstruction. IDs of the images are compared with the text list that mapper function is used. Found camera poses are visualized in Figure 3.36.

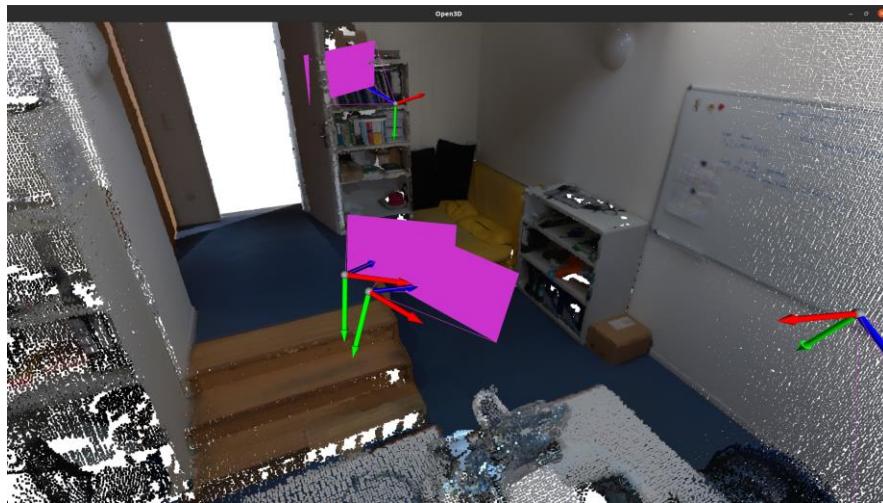


Figure 3.36: Camera Poses inside laser-scanned point cloud

The next task is finding the captured points which stay in front of the camera by using camera information. The camera information includes camera intrinsic, rotation matrix, translation, and the position of the camera inside the reconstructed point cloud. This data related to reconstruction is stored inside the exported text files. “images.txt” file stores a quaternion (QW, QX, QY, QZ) and a translation vector (TX, TY, TZ) to generate the pose of an image by defining the relation between the projection world to camera coordinate system for every image. “camera.txt” file stores camera model type, width, and height of each camera line by line. “points3D.txt” file contains coordinates and RGB values of all points in the point cloud (Schoenberger, 2020b).

To get all of the captured points by the camera, a geometric relationship is needed to filter out the points. Camera position is defined with 5 points in the space. In Figure 3.37, these points are indicated as C0,1,2,3,4. The first point, C0, represents the location of the camera sensor, and the remaining points create the rectangle viewpoint in front of the camera and show the direction. Every point in the point cloud is filtered iteratively to find out the point is inside or outside of the camera, and according to the number of the found images in the reconstruction, this process is repeated.

Four lines are drawn as starting from the location of the sensor and passing by the corners of the frame. These lines are named Line 1, Line 2, Line 3, and Line 4. Line 1

is between [C0,1]. Line 2 is between [C0,2]. Line 3 is between [C0,3]. Line 4 is between [C0,4]. They are extended as continuous lines without a limitation in the space. For further steps, they will be treated as vectors with a starting point from C0. Therefore, there are an infinite number of frames can be drawn in front of the camera. The only restriction is that the corners of the frame should be on the defined lines. Vector projection is used to find a specific frame that is on the same plane as the trial point in the filtering operation. Firstly, two vectors are created. The first one starts from the location of the camera sensor (C0) to the first defined frame's center (C1). The second one starts from the same location as the first one (C0) to the trial point in the space (P1). Then, the second vector (C0, P1) is projected on the first vector (C0, C1). Thus, the distance between the new frame and the camera sensor is known since the magnitude of the projection vector is equal to the distance. Also, the coordinates of the new frame's corners which are points (1'), (2'), (3'), (4'), and width and height value of the new frame, are known by scaling the norms of the vectors. Scaling is done between the norms of the vectors (C0,1) and (C0,1') to find the coordinate of the new frame's corner. These norms are acquired by applying two Pythagorean theorems, as is seen in Figure 3.38.

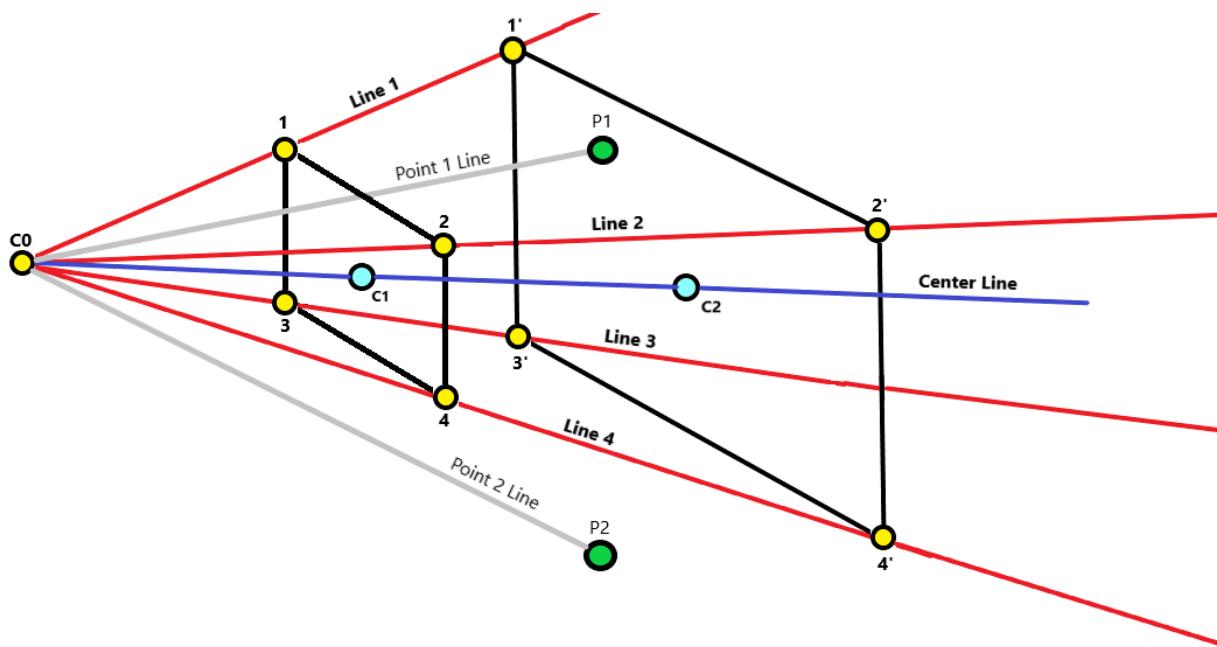


Figure 3.37: Filtering Point outside Camera View

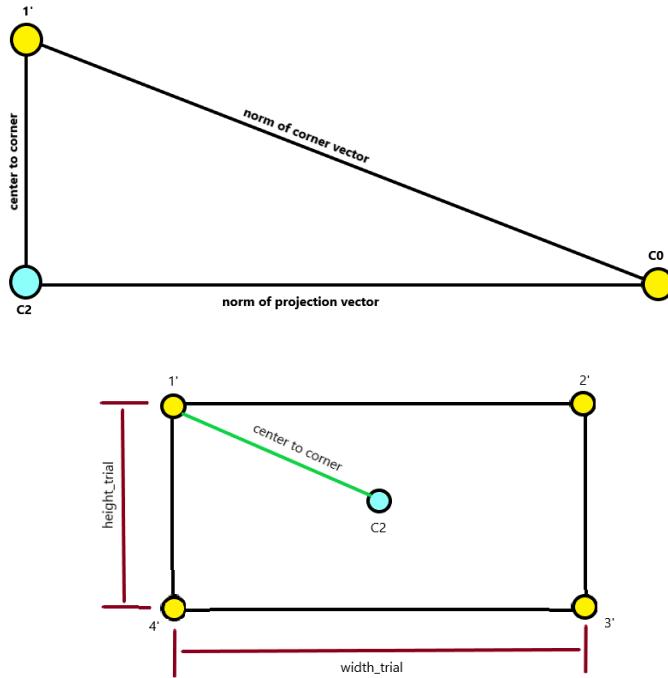


Figure 3.38: Application of Pythagorean theorems

The last check of the points is simply deciding if a point is inside or outside of the rectangle, which is the new frame in that case. Inside, the case of this check means that the trial point is captured by the specified camera. In Figure 3.39, points are checked whether they are in the camera's viewpoint or not. From the general perspective, we know that P1 is inside the viewpoint and P2 is not inside the viewpoint.

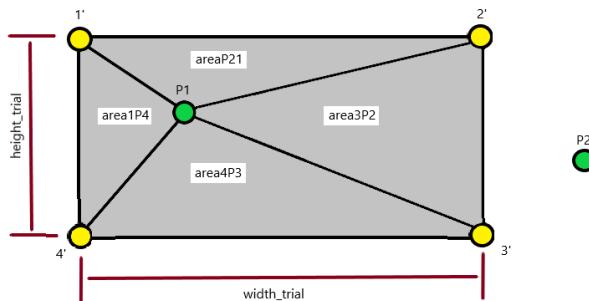


Figure 3.39: Point Viewpoint Check

For the first case, P1 divides the rectangle into 4 small triangles. The rectangle's area is known since the new frame's width, and height is known. Also, coordinates of all corners and coordinates of P1 are known. Therefore, the rectangle's area and the areas of triangles can be calculated. The criteria that decide that points are inside or outside is the summation of 4 small areas that cannot be larger than the area of the

rectangle. For P1, this criterion is not violated, so it is inside the viewpoint, but for the second case, P2 divides the rectangle's area into 4 small triangles, and the total area of triangles is larger than the rectangle's area, so the criteria are violated. As a result of this, P2 is outside of the camera's viewpoint. This check is continued sequentially for every point in the laser-scanned point cloud.

All of these steps are computed inside a python script which is in Appendix B.1. In that script, the argument parser is set to take the paths of the exported file version of the laser-scanned point cloud with (--laserscan_model), exported file version of the reconstructed point cloud with (--input_model), and the name list of the newly added images (--exporttext_model). Two collections are created as a global variable to store the data of the cameras of found images and the points that are inside the viewpoint of the camera. Additionally, one more check is needed, and it uses the "angle_between" function. The requirement for this check is coming from the case that the selected point in the iteration is on the rear side of the camera. In that case, points are filtered from both sides of the camera. To prevent this problem, the angle between the projection vector and a vector that is defined between (C0, C1) is calculated. The angle must be 0 to add the selected point into the collection of points.

In Figure 3.40, the cameras of found images are displayed with the points inside their viewpoint. This script has a commented section that allows the user to select one image and shows only its camera's viewpoint.

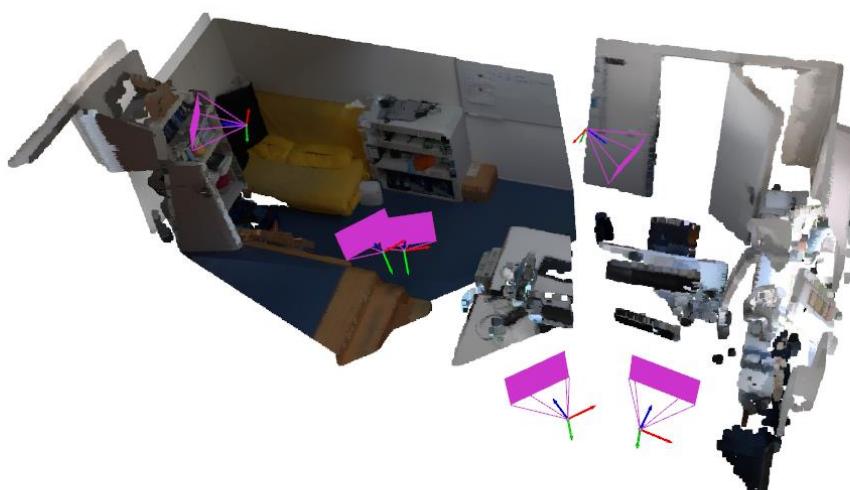


Figure 3.40: Camera Locations and their captured points

This script has a commented section that allows the user to select one image and shows only its camera's viewpoint. In Figure 3.41, the script only displays the points that are inside the viewpoint of the camera that has the same position as the camera that captures the following image file.



Figure 3.41: An image file and its corresponding viewpoint with captured points

4 Experiments and Results

The process of finding the camera pose of new images is explained in the last part of the Methodology section. In the experiment section, various reconstructions are created, and camera poses in each reconstruction will be found.

The parameters of reconstruction steps are changed to have reconstruction with different variations. COLMAP includes configuration files for each reconstruction step. The first step of the reconstruction starts with setting a file directory with an initialization script. This script which is seen in Figure 4.1, includes parameters for feature extraction, preferred camera type, and paths.

```

1 log_to_stderr=false
2 log_level=2
3 database_path=/tmp/data/database.db
4 image_path=/tmp/images
5 [ImageReader]
6 single_camera=false
7 single_camera_per_folder=true
8 existing_camera_id=-1
9 default_focal_length_factor=1.2
10 camera_model=SIMPLE_RADIAL
11 camera_params=
12 [SiftExtraction]
13 use_gpu=true
14 estimate_affine_shape=false
15 upright=false
16 domain_size_pooling=true
17 num_threads=-1
18 max_image_size=8000
19 max_num_features=24576
20 first_octave=-1
21 num_octaves=6
22 octave_resolution=5
23 max_num_orientations=2
24 dsp_num_scales=10
25 peak_threshold=0.006666666666666671
26 edge_threshold=10
27 dsp_min_scale=0.16666666666666666666
28 dsp_max_scale=4
29 gpu_index=-1
30

```

Figure 4.1: Project Initialization File

In this file, domain size pooling is set to 1 to have a better version of SIFT which means feature extraction will be more accurate, but it has a negative side in which DSP needs more computational power than the default condition. Also, max_image_size, max_num_features, and num_octaves parameters are changed to have a better reconstruction. The maximum number of features and number of octaves are increased to search further range for more features.

Patch Match is also an important step for the reconstruction and has a direct influence on the point cloud quality. The parameter of patch match is configured in the script that is used with Dockerfile. Entryfile.sh files in Appendix A.2 and A.3 have parameters to modify the patch match step. In Figure 4.2, the related section of the file is given.

```
colmap patch_match_stereo \
--workspace_path $OUTPUT_PATH/dense \
--workspace_format COLMAP \
--PatchMatchStereo.geom_consistency true \
--PatchMatchStereo.filter 1 \
--PatchMatchStereo.write_consistency_graph false \
--PatchMatchStereo.max_image_size -1 \
--PatchMatchStereo.window_radius 6 \
--PatchMatchStereo.window_step 1 \
--PatchMatchStereo.num_samples 15 \
--PatchMatchStereo.num_iterations 5 \
--PatchMatchStereo.filter_min_num_consistent 2 \
--PatchMatchStereo.depth_min -1 \
--PatchMatchStereo.depth_max -1 \
--PatchMatchStereo.sigma_spatial -1 \
--PatchMatchStereo.sigma_color 0.2000000298023224 \
--PatchMatchStereo.ncc_sigma 0.60000002384185791 \
--PatchMatchStereo.min_triangulation_angle 1 \
--PatchMatchStereo.incident_angle_sigma 0.89999997615814209 \
--PatchMatchStereo.geom_consistency_regularizer 0.30000001192092896 \
--PatchMatchStereo.geom_consistency_max_cost 3 \
--PatchMatchStereo.filter_min_ncc 0.10000000149011612 \
--PatchMatchStereo.filter_min_triangulation_angle 3 \
--PatchMatchStereo.filter_geom_consistency_max_cost 1 \
--PatchMatchStereo.cache_size 200 \
--PatchMatchStereo.gpu_index -1
```

Figure 4.2: Patch Match Configuration

To get a better result from dense reconstruction, some specific parameters are variously tested in the patch match section. According to COLMAP documentation, a large patch window radius (PatchMatchStereo.window_radius) can improve the quality of results. Additionally, the filtering threshold can be reduced but with a loss from the photometric consistency. (PatchMatchStereo.filter_min_ncc). Window radius is setting the size of the patch, which captures the surrounding pixel and the number of pixels around a selected pixel. COLMAP increases the appearance of the little features with respect to the increasing value of window radius. Moreover, a bigger window radius affects the reconstruction in terms of computation time since it needs more pixels for patch matching. Reconstruction 1 to 7 is computed with the corresponding window radius values of 2,4,6,8,10,12,14. For steps, defined parameters are used as variables in trials. Produced reconstruction is processed for finding camera poses, and their results are compared.

The related data about the model is collected by using the model analyzer extension of the COLMAP after reconstruction is finished, as seen in Table 1. In Figure 4.3, the number of images that are used in the dense model is compared for the first reconstruction and the second reconstruction after the mapper.

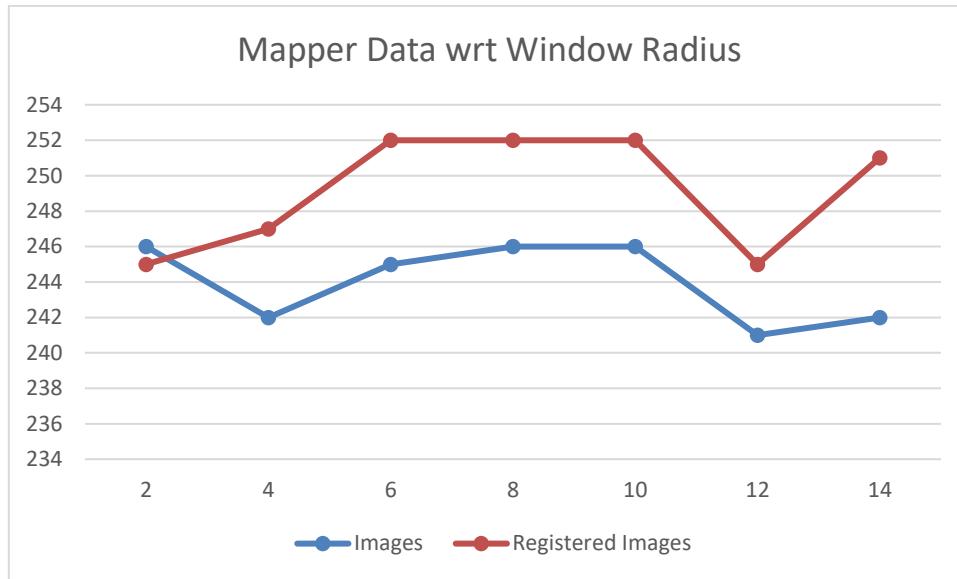


Figure 4.3: Mapper and first reconstruction comparison

According to results in Table 1, for window radius values 6,8,10, the mapper performs stable, and the number of the image in the dense models are the same, and points inside reconstructed point clouds are closer.

Window Radius	2	4	6	8	10	12	14
Cameras	6	6	7	7	7	6	7
Images	245	247	252	252	252	245	251
Registered images	245	247	252	252	252	245	251
Points	34457	34599	34804	34776	34898	34342	34923
Observations	180377	180812	181560	181505	181893	179813	181775
Mean track length	5,234843	5,225931	5,216642	5,219260	5,212133	5,235950	5,205022
Mean observation per image	736,232653	732,032389	720,476190	720,257937	721,797619	733,930612	724,203187
Mean reprojection error (px)	1,066989	1,068538	1,070000	1,073391	1,070763	1,070799	1,068729
Number of fused points	263740	933025	1618745	2147781	2624225	2982279	3408930

Table 1: Reconstructed Model Analyze after the mapper

4.1 Alignment Quality

Alignment between point clouds is significant because the localization accuracy is related to the distance to the laser-scanned point cloud. In this study, it is taken as ground

truth, and benchmarks are done based on it. For the second set of experiments, alignment affects the results because the coordinate system is assumed as the same after the alignment process. Therefore, the distance between point clouds is also a description of the accuracy of the second part.

Laser-scanned point cloud and COLMAP reconstructed point cloud is aligned using CloudCompare. For comparison purposes, reconstructions with different window radius are produced. The alignment between the resultant point cloud and a laser-scanned point cloud is done by picking the same 4 points for every trial in Figure 4.4.

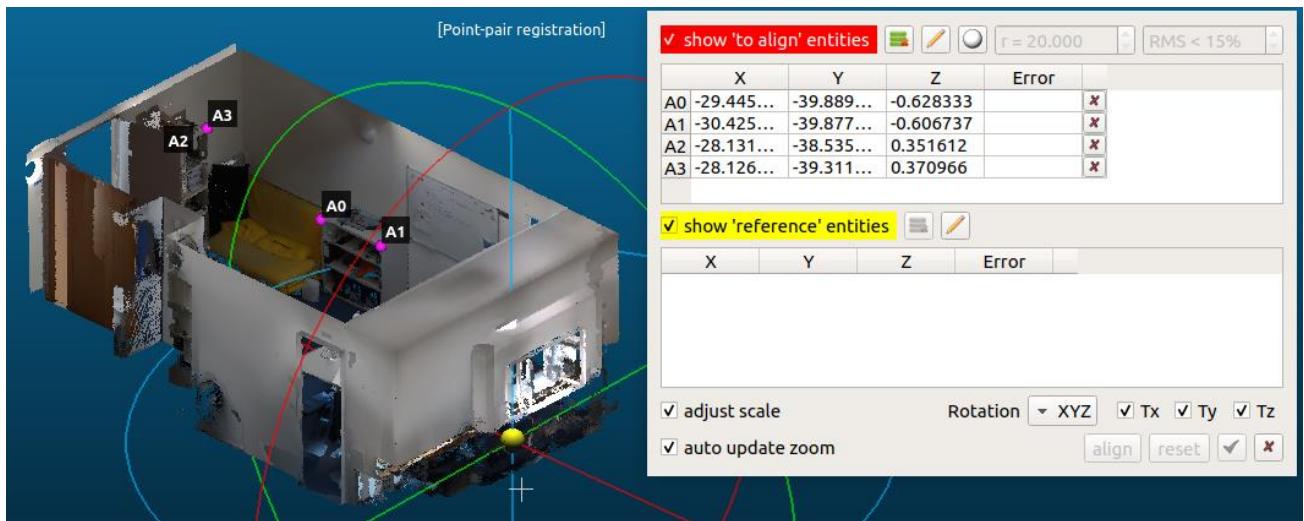


Figure 4.4: Alignment with 4 points

After alignment for every reconstruction is completed and their transformation matrices applied to the laser-scanned point cloud. By using CloudCompare's Cloud-to-Cloud Distance feature is used, which is located in the toolbar, as is shown in Figure 4.5.



Figure 4.5: Cloud-to-Cloud feature

According to the documentation of CloudCompare, point clouds are labeled as compared and reference. The distances are calculated between the points of compared point cloud and reference point cloud. All computations are done relatively to reference point cloud's points. To choose a role among point clouds, there are some conditions which are reference point cloud should be the one with widest extents and higher density.

The density of point clouds is calculated by CloudCompare and compared. Reconstruction 4 with window radius 8 is taken and compared with the laser-scanned point cloud in Figure 4.6.

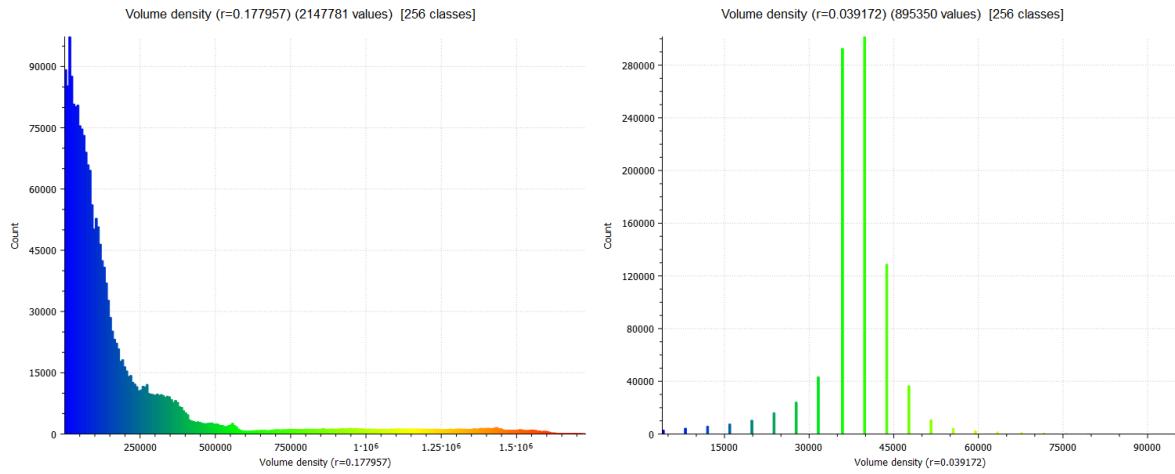


Figure 4.6: Density comparison between point clouds

As a result of this comparison, reconstructed point clouds are taken as a reference for all comparisons, and it is chosen in Figure 4.7.

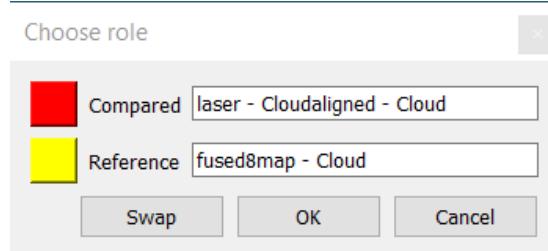


Figure 4.7: Choose Role

After roles of point clouds are defined, a distance computation window appears with several options. Octree level is the level of subdivision of octrees that distance computation is performed. It is decided automatically by the software. Max distance is used as a threshold value to filter out further points from the computation.

Local modeling strategy is preferred for distance computation because of its success in coping with sampling-related issues. These issues might be a globally too small density or too high local variations of the density of the reference cloud. The reference cloud is chosen from reconstructed clouds, and these reconstructions consist of well and badly constructed sections. Therefore, local modeling is needed, and a quadratic model, among other types of local models, is selected since it is recommended to use as follows in Figure 4.8 (CloudCompare, 2015a).

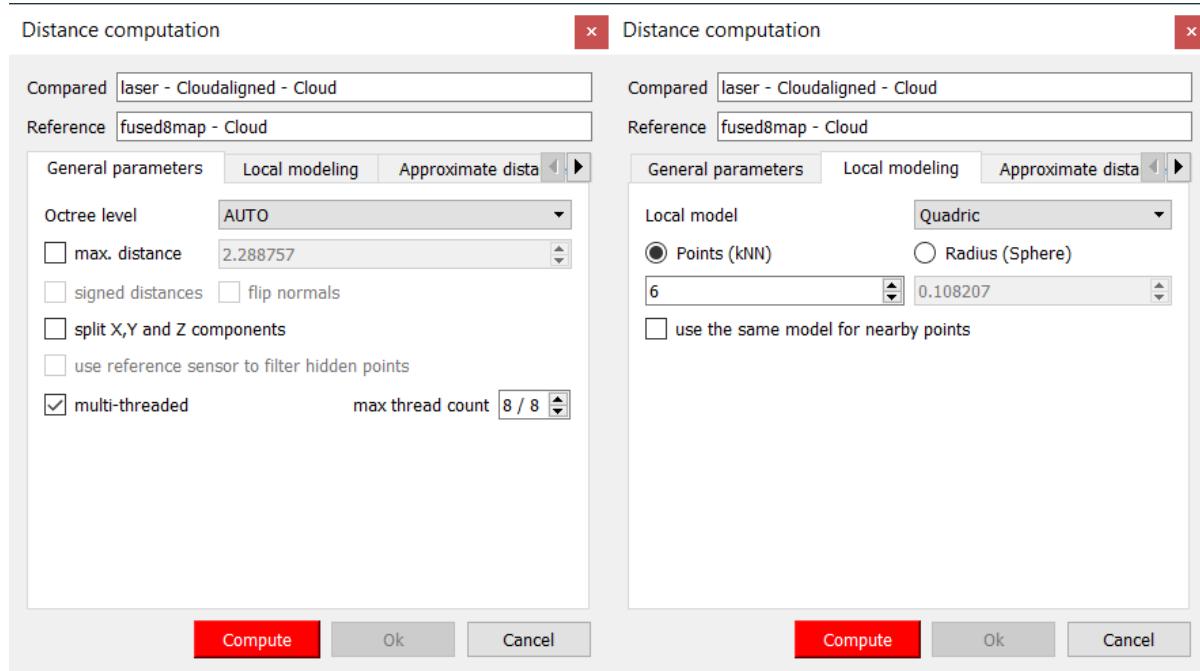


Figure 4.8: Left: General settings for Distance Computation Right: Local Model Type Selection

Cloud to cloud distance for every reconstruction is computed, and a histogram with the information of C2C absolute distances is generated. Additionally, compared point cloud is colored according to the distance of points of the reference point cloud. The blue color is used to closest distances between point clouds, and it is followed by green, yellow, and red with respect to distance. The results of C2C distance for reconstructions are shown in Figures 4.9, 4.10, 4.11, 4.12, 4.13, 4.14, and 4.15 as a histogram and 3D visualization with colored values.

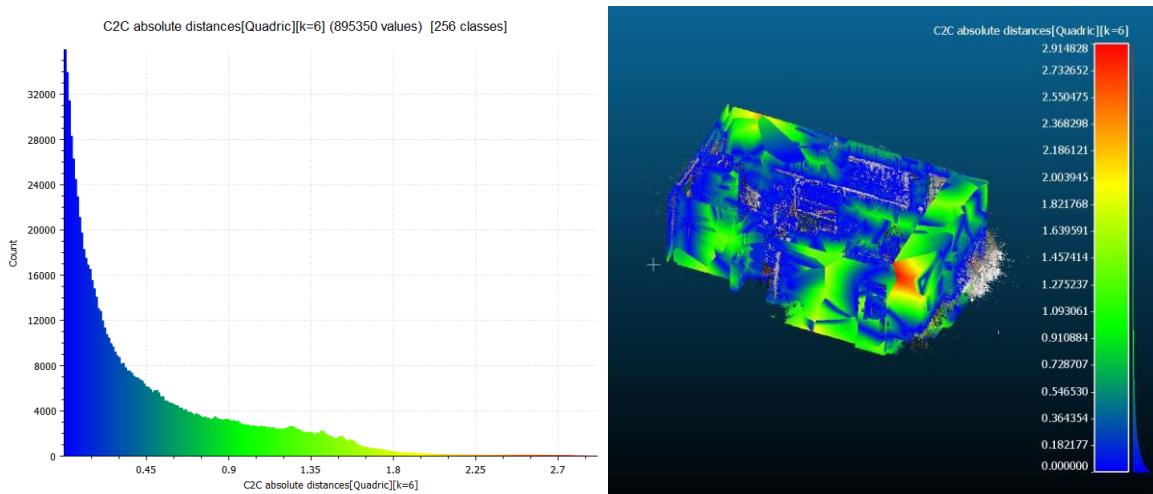


Figure 4.9: Left: Cloud to Cloud distance for Reconstruction 1 Right: Cloud to Cloud distance visualization

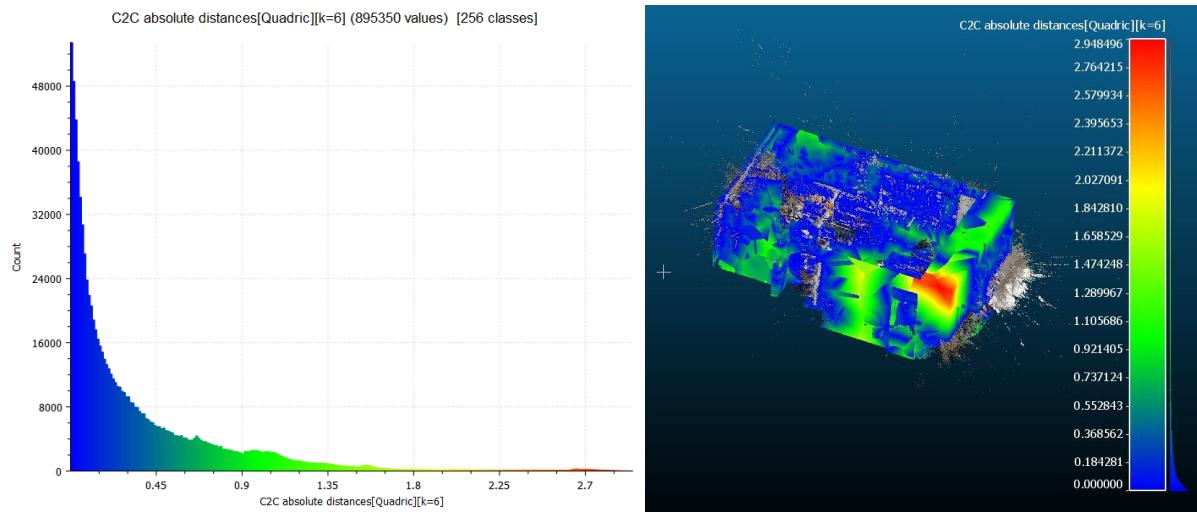


Figure 4.10: Left: Cloud to Cloud distance for Reconstruction 2 Right: Cloud to Cloud distance visualization

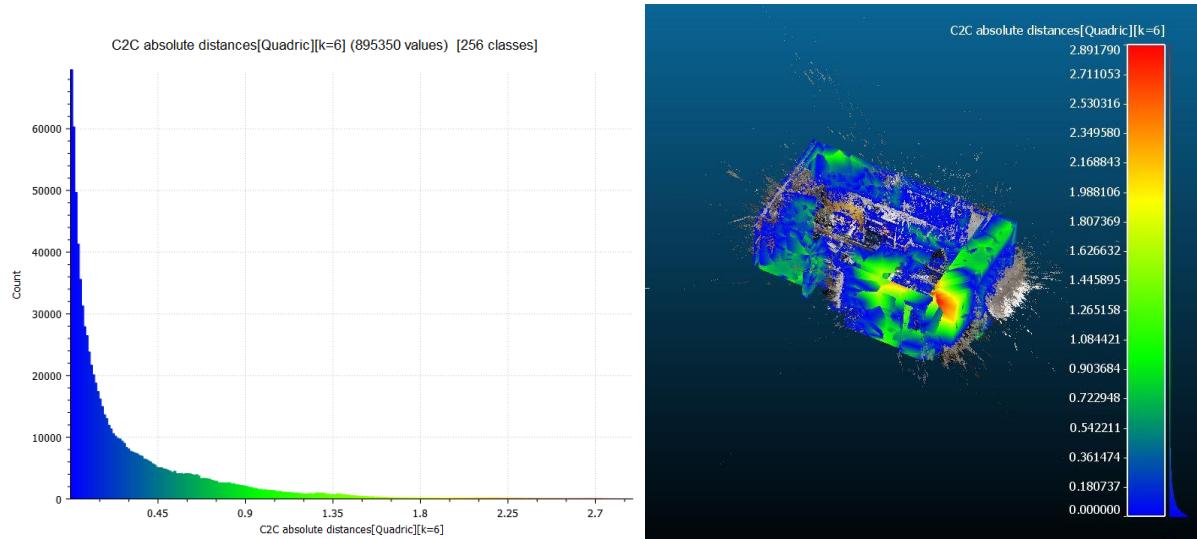


Figure 4.11: Left: Cloud to Cloud distance for Reconstruction 3 Right: Cloud to Cloud distance visualization

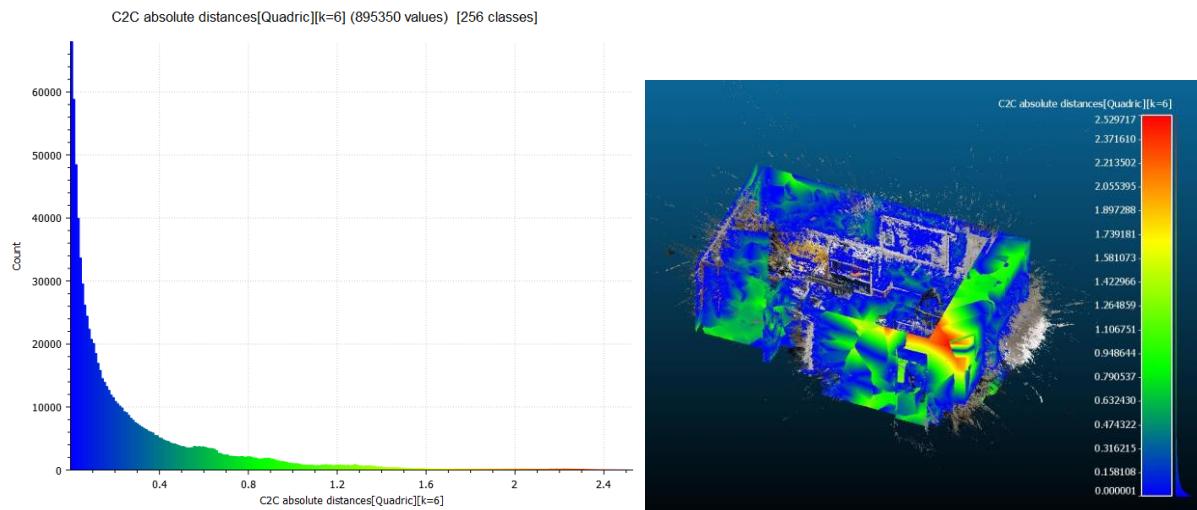


Figure 4.12: Left: Cloud to Cloud distance for Reconstruction 4 Right: Cloud to Cloud distance visualization

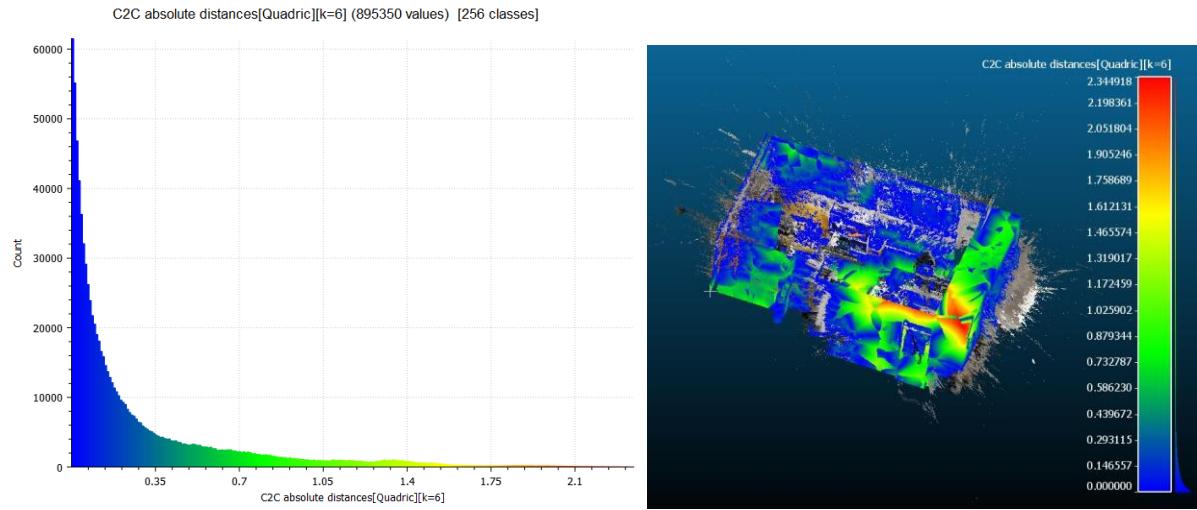


Figure 4.13: Left: Cloud to Cloud distance for Reconstruction 5 Right: Cloud to Cloud distance visualization

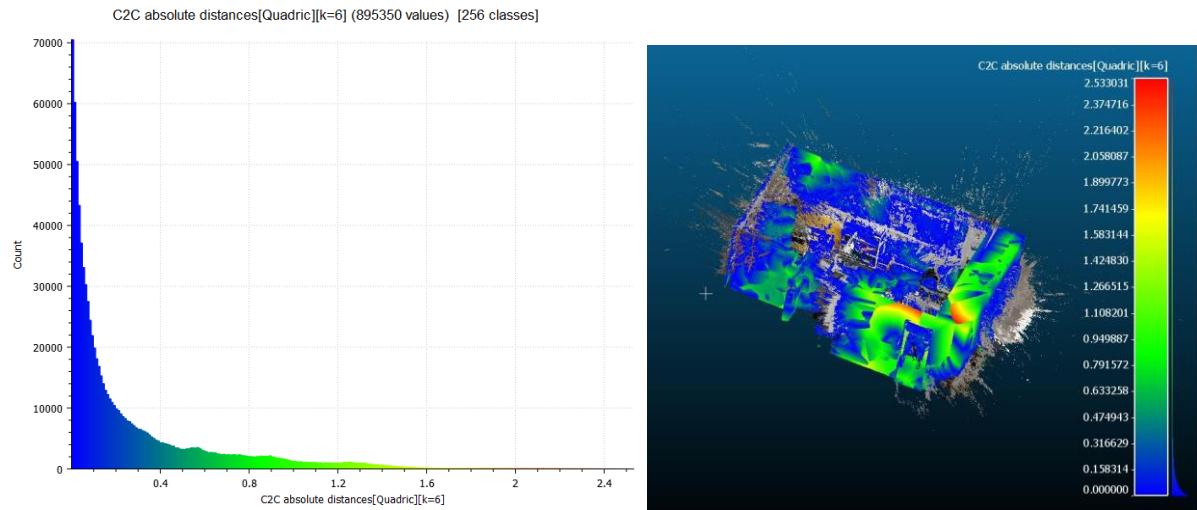


Figure 4.14: Left: Cloud to Cloud distance for Reconstruction 6 Right: Cloud to Cloud distance visualization

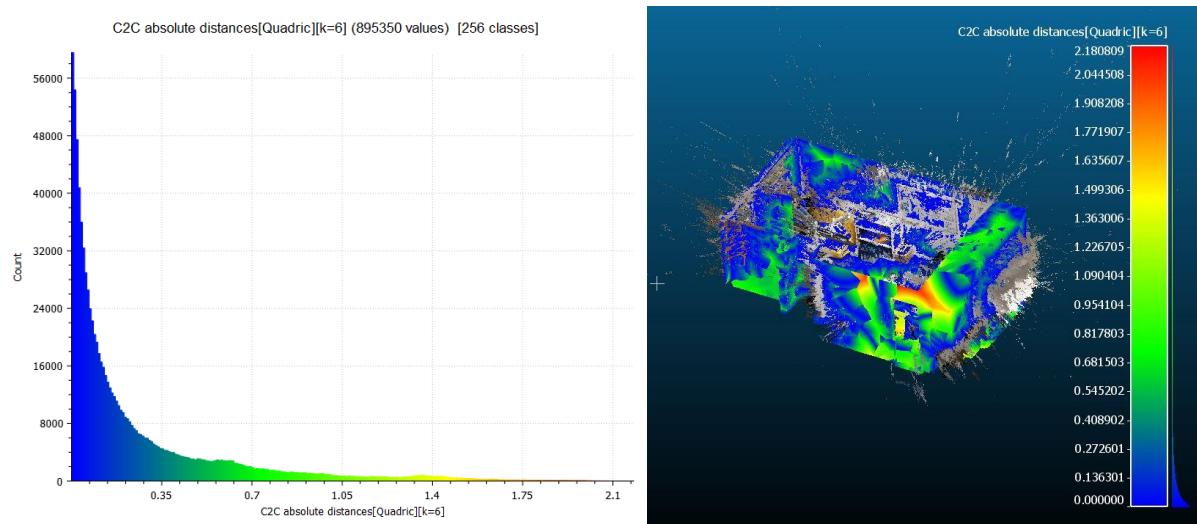


Figure 4.15: Left: Cloud to Cloud distance for Reconstruction 7 Right: Cloud to Cloud distance visualization

CloudCompare is capable of exporting cloud to cloud distance data as an excel file. By using these files, the quality of the alignment will be compared. The distance between

point pairs is investigated according to a threshold value. Therefore, the percentage of the number of points that their C2C distances that are bigger than the threshold value are compared for different reconstructions. The threshold value is defined as 0.35 for comparisons.

Reconstruction	Number of Points	Total Number of Points	Percentage
Reconstruction 1	506303	895350	56,55
Reconstruction 2	592930	895350	66,22
Reconstruction 3	630524	895350	70,42
Reconstruction 4	653866	895350	73,03
Reconstruction 5	665150	895350	74,29
Reconstruction 6	660542	895350	73,77
Reconstruction 7	682425	895350	76,22

Table 2: Reconstruction C2C Comparison with respect to a threshold value

This comparison in Table 2 shows that the best alignment is Reconstruction 7, and it is followed by Reconstruction 5 and 6.

CloudCompare computes C2C with classes. The excel exported file stores the classes and C2C distances, as seen in Table 3. Each class consists of a range of values. The third column and fourth column indicate the minimum and maximum values of the class. The second column represents the number of points inside the class. In total, 256 classes are created.

Class	Value	Class start	Class end
1	53431	0.000000119791	0.011517683158
2	48590	0.011517683158	0.023035246525
3	43800	0.023035246525	0.034552809892
4	38566	0.034552809892	0.046070373259
5	34138	0.046070373259	0.057587936626

Table 3: CloudCompare data storing structure for C2C distance

Another comparison is made for the first 5 classes of C2C distances. The percentage of the number of the points in the first 5 classes is compared for all of the reconstruction.

Reconstruction	Number of Points	Total Number of Points	Percentage
Reconstruction 1	155882	895350	17,41
Reconstruction 2	218525	895350	24,41
Reconstruction 3	256474	895350	28,65
Reconstruction 4	248913	895350	27,80
Reconstruction 5	240929	895350	26,91
Reconstruction 6	261662	895350	29,22
Reconstruction 7	238121	895350	26,60

Table 4: Reconstruction C2C Comparison with respect to first five class

Presented results in Table 4 demonstrate that the best alignment is Reconstruction 6, and it is along with Reconstruction 3 and 4.

4.2 Camera Position Finding for Varied Reconstructions

In this set of experiments, camera information is used to capture the points in front of the camera viewpoint. Different reconstructions which are produced with varied parameters are taken into the position finding process, and results are shown. The total number of the captured points by each camera is calculated for all of the images that are used in the process listed in Figure 4.16 and Figure 4.17. The total number of points in the laser-scanned point cloud is 895350. This number of points belongs to the version of the point cloud after it is edited to extract the location of office room 3218.



Figure 4.16: image455, image456 and image457



Figure 4.17: image460, image461 and image463

The captured points and the camera's viewpoint of image455 and image461 are visualized for Reconstruction 4 in Figure 4.18. These images are selected since the amount of captured points has a large difference because of the wider viewpoint of the image461.

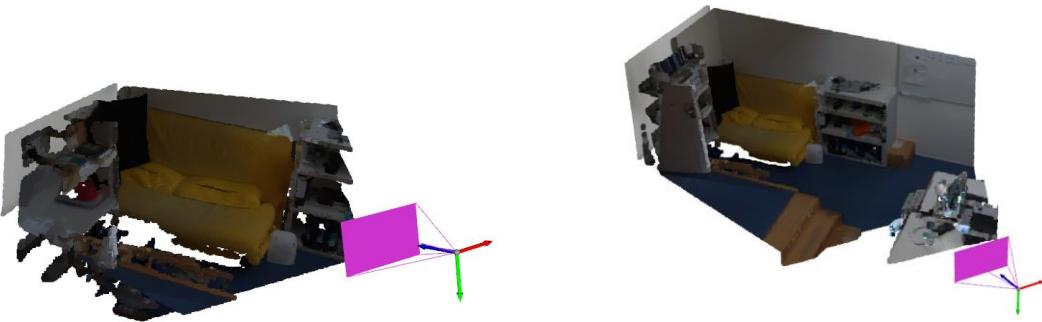


Figure 4.18: Captured Points for image455 and image461 for Reconstruction 4.

Reconstruction 6 is not able to catch the location of the camera that captures image459.jpeg in Figure 4.19. On the other hand, the location of the camera that captures the same image is available for other reconstructions (3,4,5,7), as is seen in Figure 4.20 and Table 5.

Reconstruction	Image455	Image456	Image459	Image460	Image461	Image463
Reconstruction 1	60611	46913	not available	51055	135560	79820
Reconstruction 2	77940	47996	not available	2159	259377	61149
Reconstruction 3	34881	16281	16835	22561	258690	43998
Reconstruction 4	75522	13238	13708	31593	256435	28704
Reconstruction 5	50251	3175	23546	50799	200037	69810
Reconstruction 6	75720	56573	not available	51301	260230	62899
Reconstruction 7	58782	56573	23561	48949	148346	72465

Table 5: Captured Points by cameras for different reconstructions

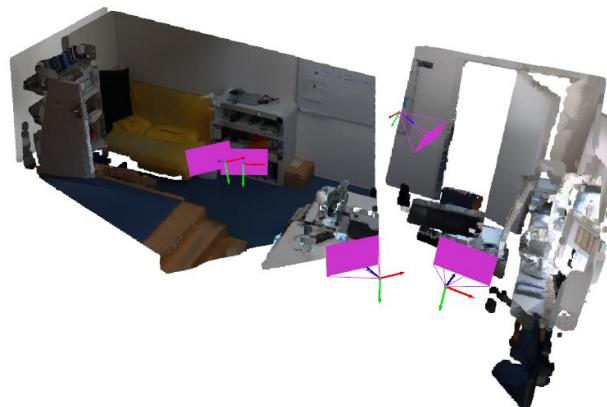


Figure 4.19: Reconstruction 6 with camera poses

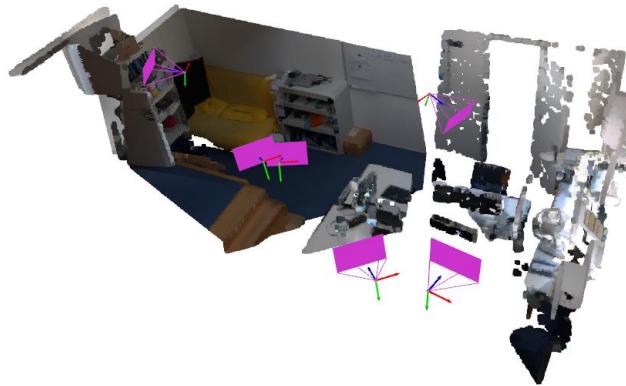


Figure 4.20: Reconstruction 4 with camera poses

Although Reconstruction 6 cannot add all of the images into reconstruction. The results of image463.jpeg consist of significant differences. As it is seen in Figure 4.21, the point cloud on the left side includes blanks, and the right one is completed.

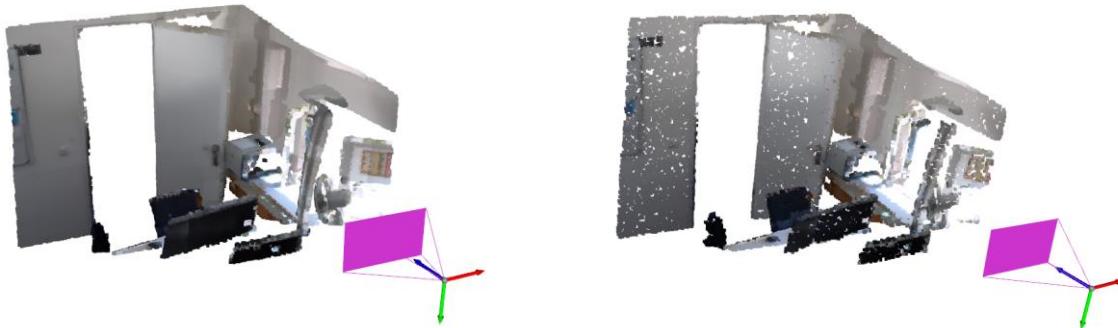


Figure 4.21: Camera Viewpoint of image463 from Reconstruction 6 (Left) and 4 (Right)

In the python script, the cameras, images, and points3D text files are created with the results of all filtering out processes, and they can be visualized by using CloudCompare. The differences can also be visualized in this software, as seen in Figures 4.22 and 4.23.

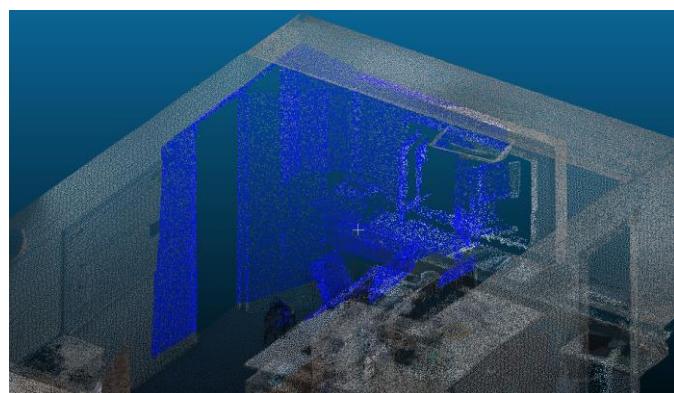


Figure 4.22: Captured points of Camera Viewpoint Reconstruction 6 in CloudCompare

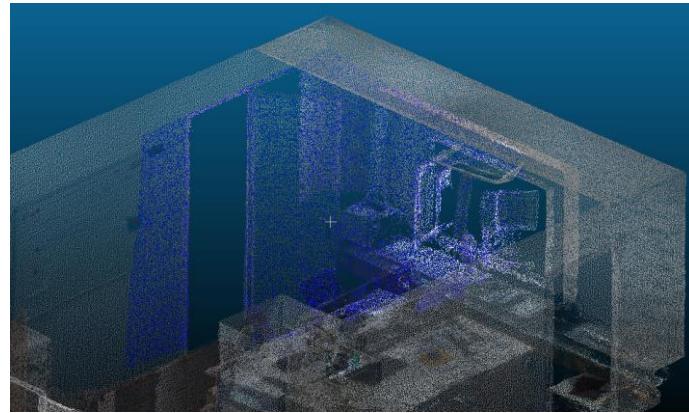


Figure 4.23: Captured points of Camera Viewpoint Reconstruction 4 in CloudCompare

According to Table 5, there are 62899 points inside the camera's viewpoint in Reconstruction 6. For Reconstruction 4, this value drops to 28704, and with visualization of point clouds in Figures 4.22 and 4.23, the difference is clearly visible.

A similar approach can be made for image455 and image461 to see additional captured points because of the wider viewpoint. In Figure 4.24, the red color shows the points from image461, and the green color shows the points from image455.

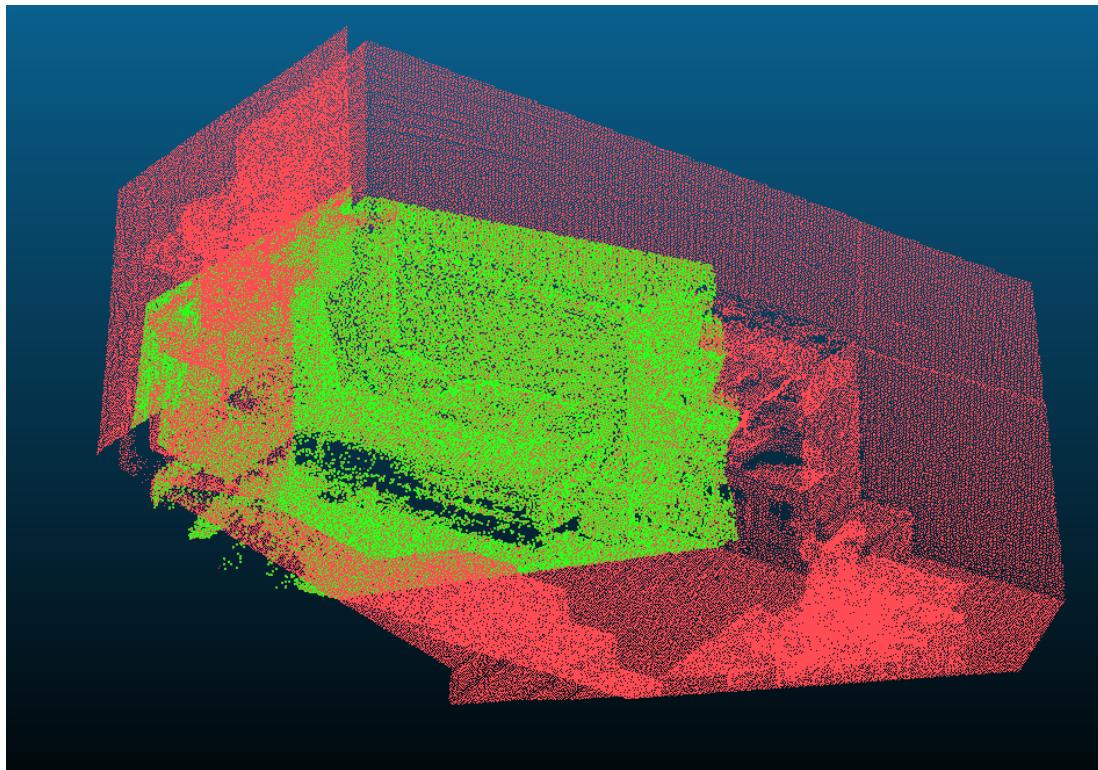


Figure 4.24: CloudCompare Visualization of captured points of image455 and image461

5 Conclusion and Future Work

Alignment of the reconstructed point cloud and a laser-scanned point cloud is done manually by picking points inside both point clouds. However, the software which is used to visualize point clouds also offers an algorithm which is called Iterative Closest Point (ICP). ICP is used to minimize the differences between point clouds. Both MeshLab and CloudCompare include ICP implementation. In Figure 5.1, CloudCompare's ICP parameter settings window is seen.

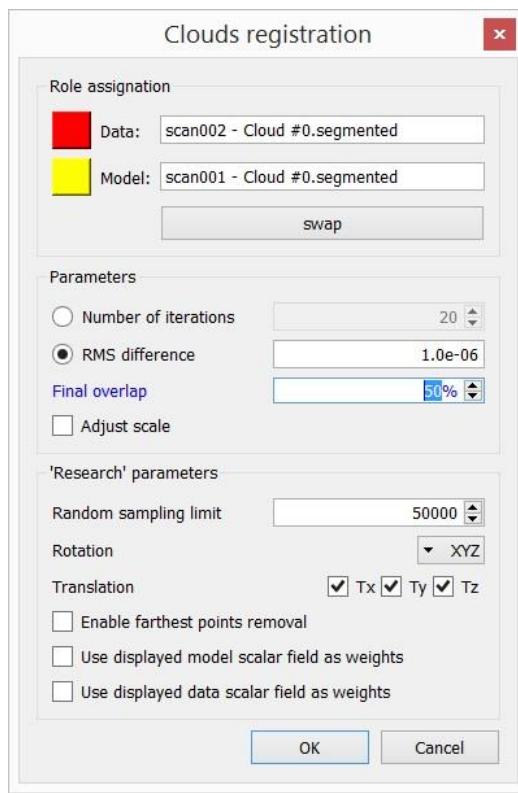


Figure 5.1: CloudCompare ICP Setting window (CloudCompare, 2015b)

For CloudCompare, the ICP registration will not work effectively when two-point clouds have big variances on large scales. As a result, this strategy is extremely effective for precisely aligning clouds, and it is the only way to achieve a good outcome in similar situations. CloudCompare will display the resulting RMS when the user has picked at least 3 or more pairs of points in both point clouds, and the user can preview the result with the 'align' button. Each pair's error contribution is listed next to each point in the table. As a result, the worst case can be deleted or re-picked. Users can also add additional points to both sets at any time to provide more limitations and improve the accuracy of the output.

The initial part of the ICP starts similar to manual point picking. One point cloud is fixed into its coordinates, and the other point cloud becomes the source for the algorithm. The user matches the closest points in both point clouds. After this part, the difference of ICP becomes clearer. The algorithm iteratively estimates the combination of rotation and translation and apply a minimization technique to align the point cloud. Until it reaches the limit of iteration or its error value reaches the defined RMS value, iteration continues (ICP-Wikipedia, 2021).

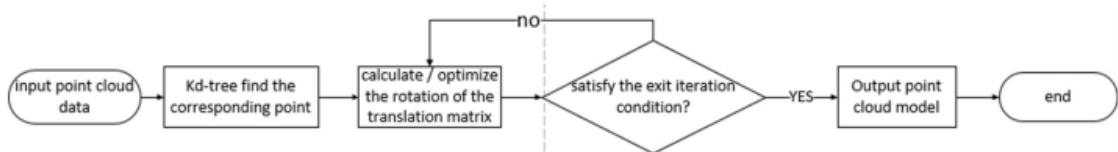


Figure 5.2: Classic ICP algorithm flow chart (Liu et al., 2018)

In Figure 5.2, ICP's pipeline is seen with its termination point. An improved ICP algorithm in this paper (Liu et al., 2018) is offered. The new algorithm uses the four-point Congruent (4PC) algorithm instead of kd-tree to increase the search efficiency of the corresponding point. The main idea of the 4PC algorithm is to find four corresponding points in the plane of reference and source point cloud and make the algorithm faster than the classical algorithm. The flow chart of the improved ICP algorithm in Figure 5.3.

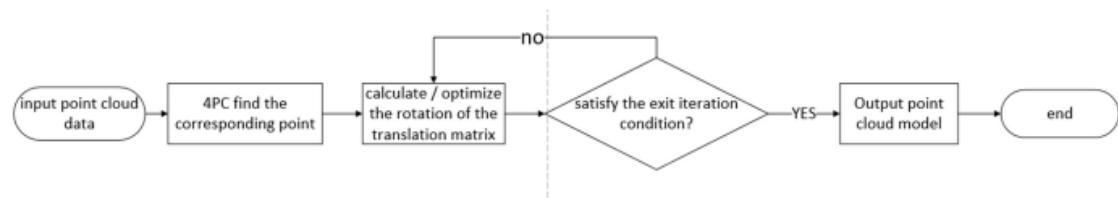


Figure 5.3: Improved ICP algorithm flow chart (Liu et al., 2018)

ICP can extend the limits of the project and increase the localization accuracy because the current state of the alignment process is error-prone. Even the ICP algorithm needs user decisions for first matching. But, in some cases picking points from the point clouds are not easy. In Figure 5.4, the laser-scanned point cloud consists of lots of points, and the details are under the layers of the points. For example, we need to find the room and edit point cloud to make the processing part faster. The office room is found in Figure 5.5.

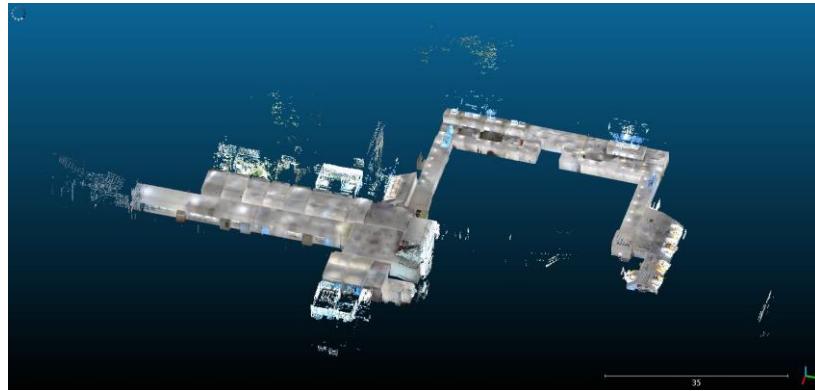


Figure 5.4: CMS laser-scanned point cloud

After this step, the point cloud needs some cuts because the COLMAP reconstruction focuses on the bookshelves and the yellow sofa in the room. From this perspective, it is impossible to recognize any of the furniture.

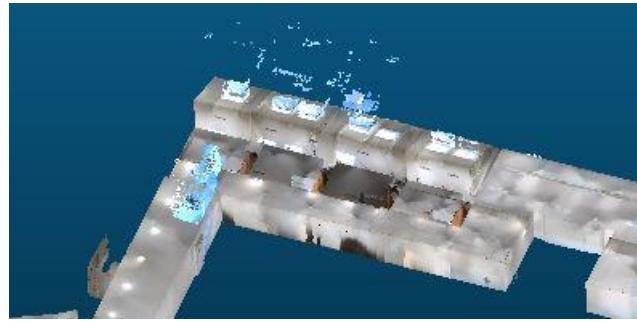


Figure 5.5: Office 3218 manually found

Therefore, there should be no need to edit point clouds and manual point picking in the perfect scenario. Moreover, this condition would remove the human influence on the processes. The reason why this scenario is hard to achieve is that a huge point cloud needs computational power and efficient algorithms, and there are different kinds of point clouds in terms of density and distribution in space, so an algorithm should handle various point clouds.

Another improvement to increase the localization accuracy can be taking some measurements in the scanned area manually to be sure that the laser-scanned point cloud is a proper reference for the computation.

In terms of computational time, improvements might be made in the camera pose finding section. The script that is used checks every point in the point cloud to decide these points are inside the camera's viewpoint or they are outside. This process takes a long time proportionally with increasing cameras because points are sequentially tested for the criterion. The binary search might be implemented to reduce computation time for this step.

References

- Agarwal, S., Furukawa, Y., Snavely, N., Simon, I., Curless, B., Seitz, S. M., & Szeliski, R. (2011). Building Rome in a Day. *Commun. ACM*, 54(10), 105–112. Available at <https://doi.org/10.1145/2001269.2001293>
- Agarwal, S., Snavely, N., Seitz, S. M., & Szeliski, R. (2010). Bundle Adjustment in the Large. In K. Daniilidis, P. Maragos, & N. Paragios (Eds.), *Computer Vision -- ECCV 2010* (pp. 29–42). Springer Berlin Heidelberg.
- Beder, C., & Steffen, R. (2006). Determining an Initial Image Pair for Fixing the Scale of a 3D Reconstruction from an Image Sequence. In K. Franke, K.-R. Müller, B. Nickolay, & R. Schäfer (Eds.), *Pattern Recognition* (pp. 657–666). Springer Berlin Heidelberg.
- Brown, M., & Lowe, D. G. (2005). Unsupervised 3D object recognition and reconstruction in unordered datasets. *Fifth International Conference on 3-D Digital Imaging and Modeling (3DIM'05)*, 56–63. Available at <https://doi.org/10.1109/3DIM.2005.81>
- CloudCompare. (2015a). *Cloud-to-Cloud Distance* - CloudCompareWiki. Available at http://www.cloudcompare.org/doc/wiki/index.php?title=Cloud-to-Cloud_Distance
- CloudCompare. (2015b). *ICP* - CloudCompareWiki. Available at <http://www.cloudcompare.org/doc/wiki/index.php?title=ICP>
- CloudCompare. (2016). *Introduction* - CloudCompareWiki. Available at [https://www.cloudcompare.org/doc/wiki/index.php?title=Introduction](http://www.cloudcompare.org/doc/wiki/index.php?title=Introduction)
- Daniel Girardeau-Montaut. (2011). CloudCompare - Open Source project. In *OpenSource Project*. Available at <https://www.danielgm.net/cc/>
- Docker. (2018). Docker overview | Docker Documentation. In *Docker.Com* (p. 1). Available at <https://docs.docker.com/get-started/overview/>
- Docker. (2020). *Use volumes* | Docker Documentation. Available at <https://docs.docker.com/storage/volumes/>
- Feature Detection - SIFT. (2019). Available at

- <https://github.com/deepanshut041/feature-detection/tree/master/sift>
- FileZilla - Wikipedia.* (2021). Available at <https://en.wikipedia.org/wiki/FileZilla>
- Forsman, M. (2011). *Point cloud densification.* Available at <https://people.cs.umu.se/tfy98mfn/exjobb/report.pdf>
- Frahm, J., Pollefeys, M., Science, C., Carolina, N., & Hill, C. (2006). *RANSAC for (Quasi-) Degenerate data (QDEGSAC) (submitted to CVPR 2006).*
- Gherardi, R., Farenzena, M., & Fusiello, A. (2010). Improving the efficiency of hierarchical structure-and-motion. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1594–1600. Available at <https://doi.org/10.1109/CVPR.2010.5539782>
- Gite Vivek. (2021). *How to set up ssh keys on Linux-unix.* Available at <https://www.cyberciti.biz/faq/how-to-set-up-ssh-keys-on-linux-unix/>
- Havlena, M., Torii, A., Knopp, J., & Pajdla, T. (2009). Randomized structure from motion based on atomic 3D models from camera triplets. *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2009*, 2009 IEEE, 2874–2881. Available at <https://doi.org/10.1109/CVPRW.2009.5206677>
- ICP-Wikipedia. (2021). *Iterative closest point - Wikipedia.* Available at https://en.wikipedia.org/wiki/Iterative_closest_point
- Jebara, T., Azarbapejani, A., & Pentland, A. (1999). 3D Structure from 2D Motion. *IEEE Signal Processing Magazine, May*.
- Li, Y., Snavely, N., & Huttenlocher, D. P. (2010). Location Recognition Using Prioritized Matching Algorithm. *Eccv*, 978–993.
- Lindeberg, T. (1993). *Scale-Space Theory in Computer Vision.*
- Lindeberg, T. (1996). Scale-space: A framework for handling image structures at multiple scales. *Computing*, 1–12.
- Liu, J., Shang, X., Yang, S., Shen, Z., Liu, X., Xiong, G., & Nyberg, T. R. (2018). Research on Optimization of Point Cloud Registration ICP Algorithm. In S. Satoh (Ed.), *Image and Video Technology* (pp. 81–90). Springer International Publishing.
- Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2), 91–110. Available at <https://doi.org/10.1023/B:VISI.0000029661.91165.89>

<https://doi.org/10.1023/B:VISI.0000029664.99615.94>

Marr, A. D., & Poggio, T. (2016). *Cooperative Computation of Stereo Disparity* Published by: American Association for the Advancement of Science Stable Available at URL: <http://www.jstor.org/stable/1742217> Linked references are available on JSTOR for this article: Cooperative Computation of Stereo D. 194(4262), 283–287.

Sattler, T., Leibe, B., & Kobbelt, L. (2011). Fast image-based localization using direct 2D-to-3D matching. *Proceedings of the IEEE International Conference on Computer Vision*, 667–674. Available at <https://doi.org/10.1109/ICCV.2011.6126302>

Schoenberger, J. L. (2020a). *COLMAP Documentation*. Available at <https://colmap.github.io/tutorial.html>

Schoenberger, J. L. (2020b). *Output Format — COLMAP*. Available at <https://colmap.github.io/format.html>

Schönberger, J. L., & Frahm, J. M. (2016). Structure-from-Motion Revisited. *Conference on Computer Vision and Pattern Recognition*, June, pp.4104-4113.

Schönberger, J. L., Zheng, E., Frahm, J. M., & Pollefeys, M. (2016). Pixelwise view selection for unstructured multi-view stereo. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9907 LNCS, 501–518. Available at https://doi.org/10.1007/978-3-319-46487-9_31

SFTP server overview - IBM Documentation. (2021). Available at <https://www.ibm.com/docs/en/b2badv-communication/1.0.0?topic=concepts-sftp-server-overview>

Shah, R., Deshpande, A., & Narayanan, P. J. (2015). *Multistage SFM: A Coarse-to-Fine Approach for 3D Reconstruction*. iii. Available at <http://arxiv.org/abs/1512.06235>

SIFT Image Features (pp. 2–3). (2015). Available at https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/AV0405/MURRAY/SIFT.html

Slobodan Ilic. (2011). *Multi-View 3D-Reconstruction*. Available at

- <http://campar.in.tum.de/twiki/pub/Chair/TeachingWs11Cv2/Multi-View3D-Reconstruction.pdf>
- Snavely, N., Seitz, S. M., & Szeliski, R. (2006). Photo Tourism: Exploring Photo Collections in 3D. *ACM Trans. Graph.*, 25(3), 835–846. Available at <https://doi.org/10.1145/1141911.1141964>
- Triggs, B., McLauchlan, P. F., Hartley, R. I., & Fitzgibbon, A. W. (2000). Bundle adjustment – a modern synthesis. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 1883, 298–372. Available at https://doi.org/10.1007/3-540-44480-7_21
- Tyagi, D. (2019). *Introduction to SIFT(Scale Invariant Feature Transform) | Medium*. Available at <https://medium.com/data-breach/introduction-to-sift-scale-invariant-feature-transform-65d7f3a72d40>
- Utkarsh Sinha. (2016). *SIFT: Theory and Practice: Keypoint orientations - AI Shack* (p. 1). Available at <http://aishack.in/tutorials/sift-scale-invariant-feature-transform-keypoint-orientation/>
- Visser, J. (1982). Manual of photogrammetry. *Photogrammetria*, 38(3), 116–117. Available at [https://doi.org/10.1016/0031-8663\(82\)90012-6](https://doi.org/10.1016/0031-8663(82)90012-6)
- Wei, B., Guan, T., Duan, L., Yu, J., & Mao, T. (2015). Wide area localization and tracking on camera phones for mobile augmented reality systems. *Multimedia Systems*, 21(4), 381–399. Available at <https://doi.org/10.1007/s00530-014-0364-2>
- Wu, C. (2013). Towards Linear-Time Incremental Structure from Motion. *2013 International Conference on 3D Vision - 3DV 2013*, 127–134. Available at <https://doi.org/10.1109/3DV.2013.25>
- Wu, C., Agarwal, S., Curless, B., & Seitz, S. M. (2011). Multicore bundle adjustment. *CVPR 2011*, 3057–3064. Available at <https://doi.org/10.1109/CVPR.2011.5995552>
- Zheng, E., Dunn, E., Jojic, V., & Frahm, J. M. (2014). PatchMatch based joint view selection and depthmap estimation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1510–1517. Available at <https://doi.org/10.1109/CVPR.2014.196>

Appendix

Appendix A.1: Dockerfile for COLMAP

```
FROM ripfreeworld/colmap_cuda10.2:2020Aug26

COPY entrypoint.sh entrypoint.sh

CMD [ "./entrypoint.sh" ]
```

Appendix A.2: Entrypoint.sh for COLMAP reconstruction

```
#!/bin/bash
set -eE

DATASET_PATH=/tmp/data
IMAGE_PATH=/tmp/images

colmap database_creator --database_path /tmp/data/database.db

echo "DATABASE CREATED"

echo "OPERATION START"

mkdir -p /tmp/workspace && colmap feature_extractor \
--project_path $DATASET_PATH/project.ini \
--database_path $DATASET_PATH/database.db \
--image_path $IMAGE_PATH

OUTPUT_PATH=/tmp/workspace

echo "END OF FEATURE EXTRACTOR"

colmap exhaustive_matcher \
--database_path $DATASET_PATH/database.db

echo "END OF EXHAUSTIVE MATCHER"

mkdir $OUTPUT_PATH/sparse

colmap mapper \
--database_path $DATASET_PATH/database.db \
--image_path $IMAGE_PATH \
--output_path $OUTPUT_PATH/sparse

echo "END OF MAPPER"

mkdir $OUTPUT_PATH/sparse/triangulated

colmap point_triangulator \
--database_path $DATASET_PATH/database.db \
--image_path $IMAGE_PATH \
--input_path $OUTPUT_PATH/sparse/0 \
--output_path $OUTPUT_PATH/sparse/triangulated

mkdir $OUTPUT_PATH/dense

colmap image_undistorter \
```

```
--image_path $IMAGE_PATH \
--input_path $OUTPUT_PATH/sparse/triangulated \
--output_path $OUTPUT_PATH/dense \
--output_type COLMAP \
--max_image_size 2000

echo "END OF IMAGE UNDISTORTER"

colmap patch_match_stereo \
--workspace_path $OUTPUT_PATH/dense \
--workspace_format COLMAP \
--PatchMatchStereo.geom_consistency true \
--PatchMatchStereo.filter 1 \
--PatchMatchStereo.write_consistency_graph false \
--PatchMatchStereo.max_image_size -1 \
--PatchMatchStereo.window_radius 8 \
--PatchMatchStereo.window_step 1 \
--PatchMatchStereo.num_samples 15 \
--PatchMatchStereo.num_iterations 5 \
--PatchMatchStereo.filter_min_num_consistent 2 \
--PatchMatchStereo.depth_min -1 \
--PatchMatchStereo.depth_max -1 \
--PatchMatchStereo.sigma_spatial -1 \
--PatchMatchStereo.sigma_color 0.2000000298023224 \
--PatchMatchStereo.ncc_sigma 0.60000002384185791 \
--PatchMatchStereo.min_triangulation_angle 1 \
--PatchMatchStereo.incident_angle_sigma 0.89999997615814209 \
--PatchMatchStereo.geom_consistency_regularizer 0.30000001192092896 \
--PatchMatchStereo.geom_consistency_max_cost 3 \
--PatchMatchStereo.filter_min_ncc 0.1000000149011612 \
--PatchMatchStereo.filter_min_triangulation_angle 3 \
--PatchMatchStereo.filter_geom_consistency_max_cost 1 \
--PatchMatchStereo.cache_size 200 \
--PatchMatchStereo.gpu_index -1

echo "END OF PATCH MATCH STEREO"

colmap stereo_fusion \
--workspace_path $OUTPUT_PATH/dense \
--workspace_format COLMAP \
--input_type geometric \
--output_path $OUTPUT_PATH/dense/fused.ply

echo "END OF STEREO FUSION"

echo "COLMAP POINT CLOUD IS GENERATED"

mkdir $OUTPUT_PATH/exporttext

colmap model_converter \
--input_path $OUTPUT_PATH/dense/sparse \
--output_path $OUTPUT_PATH/exporttext \
--output_type TXT

echo "RESULTS ARE EXPORTED AS TXT FILES"

colmap model_analyzer \
--path $OUTPUT_PATH/sparse/triangulated

echo "DENSE MODEL ANALYZE"

colmap model_analyzer \
--path $OUTPUT_PATH/dense/sparse
```

Appendix A.3: Entrypoint.sh for COLMAP mapper

```
#!/bin/bash
set -eE

DATASET_PATH=/tmp/data
IMAGE_PATH=/tmp/images

echo "OPERATION START"

mkdir -p /tmp/output && colmap feature_extractor \
    --database_path $DATASET_PATH/database.db \
    --image_path $IMAGE_PATH \
    --image_list_path /tmp/text/images.txt

OUTPUT_PATH=/tmp/output
mkdir $OUTPUT_PATH/sparse

colmap exhaustive_matcher \
    --database_path $DATASET_PATH/database.db

echo "END OF EXHAUSTIVE MATCHER"

echo "MAPPER START"

colmap mapper \
    --database_path $DATASET_PATH/database.db \
    --image_path $IMAGE_PATH \
    --output_path $OUTPUT_PATH/sparse

echo "MAPPER END"

echo "IMAGE UNDISTORTER START"

mkdir $OUTPUT_PATH/dense

colmap image_undistorter \
    --image_path $IMAGE_PATH \
    --input_path $OUTPUT_PATH/sparse/0 \
    --output_path $OUTPUT_PATH/dense \
    --output_type COLMAP \
    --max_image_size 2000

echo "END OF IMAGE UNDISTORTER"

colmap patch_match_stereo \
    --workspace_path $OUTPUT_PATH/dense \
    --workspace_format COLMAP \
    --PatchMatchStereo.geom_consistency true \
    --PatchMatchStereo.filter 1 \
    --PatchMatchStereo.write_consistency_graph false \
    --PatchMatchStereo.max_image_size -1 \
    --PatchMatchStereo.window_radius 4 \
    --PatchMatchStereo.window_step 1 \
    --PatchMatchStereo.num_samples 15 \
    --PatchMatchStereo.num_iterations 5 \
    --PatchMatchStereo.filter_min_num_consistent 2 \
    --PatchMatchStereo.depth_min -1 \
    --PatchMatchStereo.depth_max -1 \
    --PatchMatchStereo.sigma_spatial -1 \
    --PatchMatchStereo.sigma_color 0.20000000298023224 \
    --PatchMatchStereo.ncc_sigma 0.60000002384185791 \
    --PatchMatchStereo.min_triangulation_angle 1 \
```

```
--PatchMatchStereo.incident_angle_sigma 0.89999997615814209 \
--PatchMatchStereo.geom_consistency_regularizer 0.30000001192092896 \
--PatchMatchStereo.geom_consistency_max_cost 3 \
--PatchMatchStereo.filter_min_ncc 0.1000000149011612 \
--PatchMatchStereo.filter_min_triangulation_angle 3 \
--PatchMatchStereo.filter_geom_consistency_max_cost 1 \
--PatchMatchStereo.cache_size 200 \
--PatchMatchStereo.gpu_index -1

echo "END OF PATCH MATCH STEREO"

colmap stereo_fusion \
--workspace_path $OUTPUT_PATH/dense \
--workspace_format COLMAP \
--input_type geometric \
--output_path $OUTPUT_PATH/dense/fused.ply

echo "END OF STEREO FUSION"

echo "COLMAP POINT CLOUD IS GENERATED"

mkdir $OUTPUT_PATH/exporttext

colmap model_converter \
--input_path $OUTPUT_PATH/dense/sparse \
--output_path $OUTPUT_PATH/exporttext \
--output_type TXT

echo "MAPPER RESULTS ARE EXPORTED AS TXT FILES"

colmap model_analyzer \
--path $OUTPUT_PATH/sparse/0

echo "DENSE MODEL ANALYZE"

colmap model_analyzer \
--path $OUTPUT_PATH/dense/sparse
```

Appendix B.1: Added sections of visualize_model.py inside COLMAP (Schoenberger, 2020a)

```

camerainfo = collections.namedtuple(
    "camerainfo", ["camera_id", "image_id", "lines", "points"])

Point3Dinfo = collections.namedtuple(
    "Point3D", ["id", "xyz", "rgb", "error", "image_ids", "point2D_idxs"])

# These two dictionary are created for filtering cameras and points with their data
cameradict = {}
pointdict = {}

# Dictionary is created to store camera data that added after the mapper

def cam_register(image_id, camera_id, cam_lines, cam_points):
    cameradict[image_id] = camerainfo(camera_id=camera_id, image_id=image_id,
                                       lines=cam_lines, points=cam_points)

def point_register(point3D_id, xyz, rgb, error, image_ids, point2D_idxs):
    pointdict[point3D_id] = Point3Dinfo(id=point3D_id, xyz=xyz, rgb=rgb,
                                         error=error, image_ids=image_ids,
                                         point2D_idxs=point2D_idxs)

# Added lines to add_cameras function
def add_cameras(self, scale=1):
    .
    .
    .
    .
    .
    cam_points = np.asarray(cam_model[2].points)
    cam_lines = np.asarray(cam_model[2].lines)
    cam_register(img.id, img.camera_id, cam_lines, cam_points)
    .
    .
    .

def parse_args():
    parser = argparse.ArgumentParser(description="Visualize COLMAP binary and text
models")
    parser.add_argument("--input_model", required=True, help="path to input model
folder")
    parser.add_argument("--laserscan_model", required=True, help="path to laser
scan model folder")
    parser.add_argument("--exporttext_model", required=True, help="path to exported
.txt folder")
    parser.add_argument("--input_format", choices=[".bin", ".txt"],
                        help="input model format", default="")
    args = parser.parse_args()
    return args

# r is the point with a distance from the line
# p and q are the given points on the line
def t(p, q, r):
    x = p-q
    return np.dot(r-q, x)/np.dot(x, x)

def d(p, q, r):
    return np.linalg.norm(t(p, q, r)*(p-q)+q-r)

def projectionvector(u, v):
    # Projection of Vector u on Vector v
    # finding norm of the vector v
    v_norm = np.sqrt(sum(v**2))

```

```

# finding dot product using np.dot()
return (np.dot(u, v)/v_norm**2)*v

def trianglearea(p1,p2,p3):

    v1 = p1 - p2
    v2 = p1 - p3
    v3 = p2 - p3

    normv1 = np.sqrt(sum(v1**2))
    normv2 = np.sqrt(sum(v2**2))
    normv3 = np.sqrt(sum(v3**2))

    # Calculate the semi-perimeter
    s = (normv1 + normv2 + normv3) / 2

    # Calculate the area
    area = (s*(s-normv1)*(s-normv2)*(s-normv3)) ** 0.5
    return area

def unit_vector(vector):
    # Returns the unit vector of the vector.
    return vector / np.linalg.norm(vector)

def angle_between(v1, v2):
    # Returns the angle in radians between vectors 'v1' and 'v2'::

    # Angle_between((1, 0, 0), (0, 1, 0))
    # 1.5707963267948966
    # Angle_between((1, 0, 0), (1, 0, 0))
    # 0.0
    # Angle_between((1, 0, 0), (-1, 0, 0))
    # .141592653589793

    v1_u = unit_vector(v1)
    v2_u = unit_vector(v2)
    return np.arccos(np.clip(np.dot(v1_u, v2_u), -1.0, 1.0))

def main():

    #args = parse_args()
    #model_path = args.input_model
    #laserscan_path = args.laserscan_model
    #imagelist_path = args.exporttext_model
    model_path = "/home/ahmethan/Desktop/pointcloud/winr/6/process"
    laserscan_path = "/home/ahmethan/Desktop/pointcloud/winr/6/process/laseralign8"
    points3D_path = laserscan_path + "/points3D.txt"
    modelpc3_path = model_path + "/points3D.txt"
    imagelist_path = "/home/ahmethan/Desktop/pointcloud/winr/6/process/text/images.txt"

    with open(imagelist_path) as f:
        templist = f.readlines()

    image_names = []
    for temp in templist:
        index = temp.find(".")
        if index != -1:
            subtemp = temp[index-3:index]
            image_names.append(int(subtemp))

    points3dlaser = read_points3D_text(points3D_path)
    print("Total number of points in the laser-scanned point cloud is " +
str(len(points3dlaser)) + ".")"

    # Read COLMAP model
    model = Model()

```

```

#model.read_model(model_path, ext=args.input_format)
model.read_model(model_path, ext=".txt")

images = model.images

found_images = []
del_list = []

# Search additional images inside exported model
for img in images:
    prev_len = len(found_images)
    for x in image_names:
        if x == img:
            found_images.append(x)
            print("Camera pose of image" + str(x) + ".jpeg is found.")
    if prev_len == len(found_images):
        del_list.append(img)

# Uncomment this section to choose only one image for further process

num_img = int(input("Enter the image number to find captured points: "))
for img_id in found_images:
    if img_id != num_img:
        del_list.append(img_id)
found_images.clear()
found_images.append(num_img)

# Delete unrelated camera poses from images dictionary
for delete in del_list:
    images.pop(delete)

model.images = images
points3dlaser = read_points3D_text(points3D_path)

model.create_window()
model.add_cameras(scale=1)

for imageid in found_images:

    points = cameradict[imageid].points
    lines = cameradict[imageid].lines
    widthcalc = np.sum((points[1]-points[2])**2, axis=0)
    width = np.sqrt(widthcalc)
    heightcalc = np.sum((points[2]-points[4])**2, axis=0)
    height = np.sqrt(heightcalc)

    framecenter = (((points[1]+points[3])/2)+((points[2]+points[4])/2))/2
    framevec = framecenter-points[0]
    framevec_norm = np.sqrt(sum(framevec**2))
    point_num = 0

    for id in points3dlaser:

        testpoint = points3dlaser[id].xyz

        pointvec = testpoint-points[0]
        provec = projectionvector(pointvec, framevec)
        provec_norm = np.sqrt(sum(provec**2))

        width_trial = (width*provec_norm)/framevec_norm
        height_trial = (height*provec_norm)/framevec_norm

        center_to_corner = np.sqrt(((width_trial)/2)**2+((height_trial)/2)**2)
        norm_provec = np.sqrt(sum(provec**2))
        norm_cornervector = np.sqrt((center_to_corner**2)+(norm_provec**2))
        line1vec = points[1]-points[0]
        line2vec = points[2]-points[0]
        line3vec = points[3]-points[0]

```

```

line4vec = points[4]-points[0]

normline1 = np.sqrt(sum(line1vec**2))
normline2 = np.sqrt(sum(line2vec**2))
normline3 = np.sqrt(sum(line3vec**2))
normline4 = np.sqrt(sum(line4vec**2))

corner1vec = (norm_cornervec/normline1)*line1vec
corner2vec = (norm_cornervec/normline2)*line2vec
corner3vec = (norm_cornervec/normline3)*line3vec
corner4vec = (norm_cornervec/normline4)*line4vec

corner1point = corner1vec + points[0]
corner2point = corner2vec + points[0]
corner3point = corner3vec + points[0]
corner4point = corner4vec + points[0]

rectangle_area = width_trial * height_trial

area1P4 = trianglearea(corner1point,testpoint,corner4point)
area4P3 = trianglearea(corner4point,testpoint,corner3point)
area3P2 = trianglearea(corner3point,testpoint,corner2point)
areaP21 = trianglearea(testpoint,corner2point,corner1point)

totalarea = area1P4 + area4P3 + area3P2 + areaP21

if totalarea < rectangle_area:
    if angle_between(provec,framevec) == 0.0:
        point_regs-
ter(points3dlaser[id].id,points3dlaser[id].xyz,points3dlaser[id].rgb,points3dlaser[id].error,
      points3dlaser[id].im-
age_ids,points3dlaser[id].point2D_idxs)
        point_num += 1

    print("Camera pose of image" + str(imageid) + ".jpeg captures " +
str(point_num) + " points from laser-scanned point cloud.")

write_points3D_text(pointdict, modelpc3_path)
#write_points3D_text(points3dlaser, modelpc3_path)
#model.read_model(model_path, ext=args.input_format)
model.read_model(model_path, ext=".txt")
model.add_points()
model.show()

result_path = "/home/ahmethan/Desktop/pointcloud/winr/6/result"
result_image_path = result_path + "/images.txt"
result_points_path = result_path + "/points3D.txt"
result_camera_path = result_path + "/cameras.txt"
cameras = model.cameras

write_cameras_text(cameras,result_camera_path)
write_images_text(images,result_image_path)
write_points3D_text(pointdict, result_points_path)

if __name__ == "__main__":
    main()

```

Declaration of Originality

With this statement I declare that I have independently completed this Master Thesis. The thoughts taken directly or indirectly from external sources are properly marked as such. This thesis was not previously submitted to another academic institution and has also not yet been published.

First Name Family Name

[REDACTED]

[REDACTED]

[REDACTED]

[REDACTED]