Graduation thesis submitted in partial fulfilment of the requirements for the degree of engineering sciences: Electromechanical Engineering - Robotics & Mechanical Construction

Object Detection for Authoring XR Factory Workplaces

Comparative Study on Point Cloud Registration Techniques

Menthy Denayer

Master thesis submitted under the supervision of
Prof. Dr. Ir. Bram Vanderborght & Ir. Joris De Winter

The co-supervision of
Prof. Dr. Ir. Tom Verstraten

Academic year
2022-2023

In order to be awarded the Master's programme in
Electromechanical Engineering

Acknowledgments

The mystery of life isn't a problem to solve,
but a reality to experience.

- The Reverend Mother
from *Dune* by Frank Herbert

Writing this thesis has been a personal journey, a challenging task, but also a learning experience that I luckily did not have to undertake on my own. I am very grateful for the continued advice, new ideas and trust from my supervisor, Joris De Winter. His comments, always to the point and honest, and directions have been indispensable for the success of this thesis. I cannot thank him enough for working his way through my endless emails, which I am sure must have been a challenge on its own.

I also want to express my gratitude towards Professor Vanderborght and Professor Verstraten for proof-reading the report and organizing the midterm presentations, as a chance to receive feedback.

Of course, I could not have achieved any of this, without my family and friends, whose boundless love and care, as platitudinous as it may sound, has been the fuel to fire my spark.

Menthy Denayer, 2022-2023, VUB

Abstract

Background Extended reality (XR) applications have many uses, ranging from training in workplaces, to inspection and virtual collaboration. One of the major challenges in authoring XR environments, is the time intensive process of creating the 3D assets. An alternative approach is to use 3D scanning methods, where the environment or objects are captured using depth cameras. The obtained point cloud then needs to be classified and aligned with a template model, to deduce its pose in space. This alignment problem is called Point Cloud Registration (PCR). Many different algorithms exist to solve this problem. A distinction is made between methods based on machine learning and global methods, which do not use a learning approach.

Aim and objective The goal of this thesis is to compare different registration methods. This in order to determine a framework of choosing which algorithm should be used in a specific application. The capturing of the point clouds is also considered, to derive some guidelines for achieving best results. Finally, training of the learning-based methods is also investigated. More specifically, whether training using available CAD models for creating a synthetic datasets, yields good performances in terms of the studied metrics.

Materials and Methods The literature is studied to obtain a list of state-of-the-art registration methods. The studied algorithms are ICP, RANSAC, FGR, PointNetLK, RPMNet and ROPNet. Performances are measured using typical metrics found in the literature, such as relative and mean errors, recall, R^2 and timings. The experiments are performed using the Realsense Intel D435i camera. The objects are positioned at a know pose with respect to the camera, for obtaining a ground truth solution. Pre-processing is applied to clean up the point clouds. All codes are run on a personal device or on a GPU provided by Google Colab.

Results During the experiments several parameters such as the voxel size, training models and input data are varied. From these tests it is concluded that different registration methods can be used depending on the application. GO-ICP yields overall the best accuracies, while requiring more time. RANSAC and FGR are very fast, but less precise and with more variability in the results. The learning-based methods perform the best on average when trained on normal data, created from the CAD models. PointNetLK and RPMNet achieve results similar to GO-ICP after applying refinement. ROPNet leads to the highest errors. For PointNetLK and RPMNet, training with available CAD models leads to performances comparable to RANSAC and GO-ICP after refinement. For capturing the point clouds, important aspects to consider are the camera used, pre-processing to remove outlier or irrelevant points and the size of the transformation. Best results are achieved when the point cloud is cleaned up from any background information, while maintaining as much points of the actual object. In the end, a framework is presented to choose a registration method, based on the application. It considers typical questions such as the speed requirements, accuracies and available resources.

Discussion and conclusion The experiments are carried out for medium sized objects, based on the Cranfield and ModelNet40 datasets. The captured point cloud is created by fixing the camera and creating a depth image from a single point of view. Further improvements are possible by creating a larger point cloud, combining different point of views, from different cameras. The accuracy of the ground truth estimation and the pre-processing of the point clouds could also be improved further. Finally, other registration methods with more complex training datasets can be considered to compare against the found results.

Keywords: Point Cloud Registration, Extended Reality, Intel RealSense, Comparison, Framework

Contents

Acronyms

PCR	**P**oint **C**loud **R**egistration
XR	e**X**tended **R**eality
VR	**V**irtual **R**eality
AR	**A**ugmented **R**eality
ICP	**I**terative **C**losest **P**oint
RANSAC	**Ran**dom **Sa**pmle **C**onsensus
FGR	**F**ast **G**lobal **R**egistration
PRNet	**P**artial **R**egistration **Net**work
RPM-Net	**R**obust **P**oint **M**atching **Net**work
ROPNet	**R**epresentative **O**verlapping **P**oints **Net**work
PIH	**P**eg **I**n **H**ole (assembly)
MRAE	**M**ean **R**elative **A**ngular **E**rror
MRTE	**M**ean **R**elative **T**ranslational **E**rror
RMSE	**R**oot **M**ean **S**quare **E**rror
MAE	**M**ean **A**bolsute **E**rror
MSE Threshold	**M**ean **S**quared **E**rror Threshold

Chapter 1

Introduction

1.1 Motivation

Extended Reality (XR) environments have many applications such as training in manufacturing, education or entertainment. An extended reality environment is the combination of virtual reality (VR) and augmented reality (AR). VR refers to a completely digital environment, often experienced by the user using a headset. In AR the real world is extended using virtual information. Extended reality is a space in which the real and virtual world coexist.

The creation, also called authoring, of such an application consists of two main steps, which are visualised in Figure 1.1. The first one entails the modelling of the virtual space, which is the focus in this paper. The programming of the interactions between the user and the virtual objects is the second step. There are again two main ways to model the virtual space. The objects can all be modelled using 3D software such as Blender, Maya or Studio Max. However, this is a time consuming and labour intensive task. Hence the need for an alternative, scanning-based method. Here, the environment is scanned using a 3D camera to obtain a virtual copy of the real world. It remains then to locate and align the objects of interest with their CAD models, to find their position in the scene. This alignment problem is solved using Point Cloud Registration (PCR).

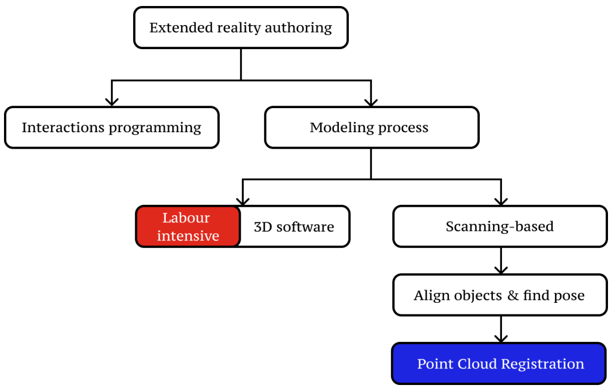


Figure 1.1: Schematic overview of the steps required in XR authoring. The focus in this paper lies on Point Cloud Registration after applying the scanning-based technique.

The problem of Point Cloud Registration consists in finding a rigid transformation between two point clouds. A point cloud is a representation of an object by points in space. The idea is shown in Figure 1.2, where the template (blue) point cloud is obtained from an existing CAD model of the object. The source (red) point cloud is the one obtained from the scanning of the environment. The pose of the source is determined with respect to the template using registration. When the pose of the camera itself is known in a coordinate system $(xyz)_G$, then the pose of the source is known too.

Point Cloud Registration (PCR) is an important problem in computer vision. When the two point clouds contain an equal number of points, obtained by applying a rigid transformation, an accurate solution

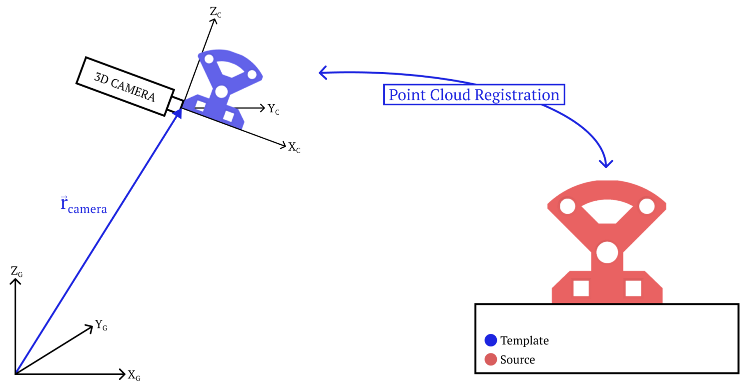


Figure 1.2: Concept of Point Cloud Registration.

often exists. However, when noise is introduced or the point clouds only share a limited number of points, the performance drops. Most algorithms solving the registration problem are tested on synthetic data. Even though noise can be added artificially, this data does not always represent real data, obtained by 3D cameras in an accurate way. In this thesis, different registration methods are compared when applied to data captured by 3D cameras. A visualisation of these steps is shown in Figure 1.3.

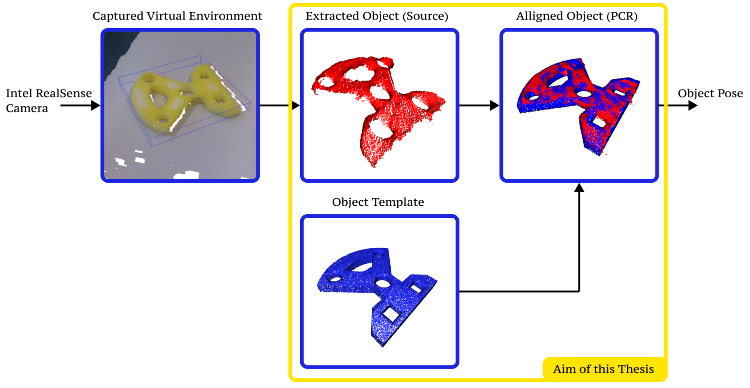


Figure 1.3: Schematic overview of the goal of this thesis.

1.2 Thesis Goals

1.2.1 Scope

In this first stage, the objects are captured separately by a 3D camera. The problem of detection is not considered here, as it is outside of the scope of this thesis. Detection refers to the process of assigning a label to each point or point cloud. However, YOLO by Redmon, J. et al. [35], which uses 2D images, could be used for this step. Alternatively, the 3D point cloud itself can also be used for this, with PointNet [2]. Thus, two assumptions are made:

1. The point cloud is already segmented from the environment, however the cutout may not be very precise.
2. The points are already assigned a label corresponding to the object they represent.

The objects are 3D printed and thus not reflecting. A CAD model is available for all the components, which serves as the template point cloud.

Working with real-world data introduces challenges such as data noise, partiality and outliers. Due to noise, exact correspondences will not exist anymore. Partiality means that one or both point clouds contain only a part of the total object, resulting in only a few points which overlap. Finally, outliers introduce points that are not a part of the object but are part of the point cloud due to the method of acquiring the data.

The environment is considered to be indoors, well-lit and relatively structured. The focus lies on manufacturing environments, which meet these conditions.

1.2.2 Research Questions

The main goal of this thesis is to compare different available Point Cloud Registration techniques, given a set of template CAD models and depth camera scans, considering different parameters. The performance of the result is measured using a set of metrics. These are based on the difference between the ground truth solution and the one obtained using the algorithms. A more detailed discussion can be found in Chapter 3 and Appendix B.

The problem is split into three sub-research questions. First, there is the choice of the Point Cloud Registration technique. Different methods exist, where some have to be trained and others not. *Which of the studied Point Cloud Registration techniques yields the most accurate results and best performance for XR applications?* The performance is again measured using the metrics explained in Chapter 3. As a first hypothesis, it is assumed the non-training based methods will lead to a more robust result, while the training based methods might have more problems recognizing features in real scans.

Secondly, for learning-based methods, training is an important step. The available template models can be modified to account for noise and partiality during training. *Is it sufficient to only use synthetic data, based on the available CAD models of the objects, to train the methods?* In other words, does the final registration result yield accurate results, when the method is trained using synthetic data? These results depend greatly on the way in which the data is generated. Different cases, such as data with noise, partiality and environmental clutter are considered.

Lastly, the captured images have to fit certain constraints in order to fulfil the desired quality for registration. *How accurate should the depth image of the object be to achieve satisfying results?* For this question, different situations are considered such as different sized objects, varying poses and the presence of environmental clutter.

These questions are not unrelated; a certain registration method may require different training methods or may impose other constraints on the data. An overview is given in Figure 1.4.

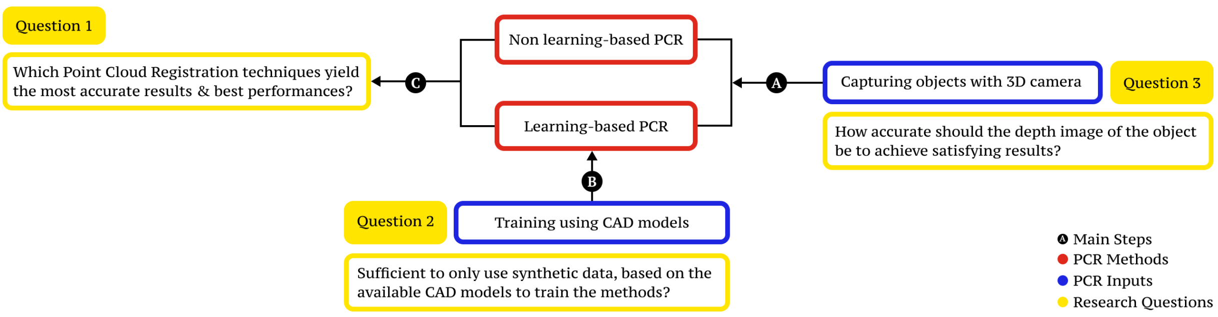

Question 1
Which Point Cloud Registration techniques yield the most accurate results & best performances?
C
Non learning-based PCR
Learning-based PCR
A
Capturing objects with 3D camera
Question 3
How accurate should the depth image of the object be to achieve satisfying results?
B
Question 2
Training using CAD models
Sufficient to only use synthetic data, based on the available CAD models to train the methods?
Main Steps
PCR Methods
PCR Inputs
Research Questions

Figure 1.4: Schematic overview of the three main research questions. Arrows indicate the relationship between the different aspects of the thesis, with A, B & C corresponding to the three main steps to answer the research questions. Blue boxes indicate inputs for the models, represented by the red boxes. Yellow boxes contain the respective research question.

1.3 Thesis Structure

The structure of this thesis is divided into three main parts. First, the literature is discussed together with the selection of the registration techniques in Chapter 2. In this part an overview is given of the current (2022-2023) state-of-the-art about authoring of XR environments. Afterwards, the main applications of Point Cloud Registration are discussed in various domains. Lastly, a selection is made of registration techniques based on their pros and cons. This part leads to the definition of the methods, shown in the red boxes of Figure 1.4.

In the next part, the metrics that are considered to asses the performances are explained in Chapter 3. The experimental setup, the 3D camera and ground truth estimation are discussed in Chapter 4. These chapters are all used in preparation of the experiments and are represented by step A and the related blue box in Figure 1.4.

The final part shows the obtained results. The outcomes of training the learning-based methods are shown in Chapter 5. This is step B in Figure 1.4. Lastly, the experiments are carried out. A parameter study is given in Chapter 6 and the experimental conclusions are drawn in Chapter 7, to finish step C in Figure 1.4.

A discussion on future work is presented in Chapter 8 and conclusions are summarised in Chapter 9.

In Appendix A-E, some additional background information and findings are given for the interested reader, which is used as a reference throughout the paper.

All codes used during the project are presented on the project Github page[1].

[1]In case the Github hyperlink malfunctions, a backup of all the additional files is made available here.

Chapter 2

Literature Study

This chapter summarizes the results of the literature study. This thesis is made in the context of XR applications. Section 2.1 gives an overview of XR applications and authoring methods. After, the applications of Point Cloud Registration in general are discussed in Section 2.2. Finally, the actual algorithms that are used during the experiments are shown in Section 2.3.

2.1 Authoring XR Environments

In this first section, the current state-of-the-art related to the authoring of eXtended Reality (XR) environments is discussed. Applications are mentioned in Section 2.1.1. Finally, some techniques for creating XR environments are given in Section 2.1.2 to formulate conclusions on future work in Section 2.1.3.

2.1.1 Applications

Applications for XR environments are vast. In this section, a few uses are discussed, which introduce new ways of authoring the environment. Although VR is often related to recreational purposes, many educational applications also exist. Lee, Y. et al. [26] discusses the use of XR for sharing a virtual environment, allowing users to experience the surroundings of another user, even when they are not themselves present at the location. In a study, Nilsson, S. et al. [32] also mentions the positive results and acceptance of using XR environments for real work settings. By researching methods of increasing the performance and speed of authoring XR environments, more researchers, students or workers could benefit from its many uses. Some popular applications are discussed in the following subsections.

Training

XR environments can help with the vocational training of personnel in manufacturing or other industries. Doolani, S. et al. [12] mention the benefits of using XR, such as consistency in the training process and safety when the real environments provide possible threats. It is even stated that training in XR environments results in a longer recall after training and a lower mental workload. The paper mentions the usefulness of the different techniques (AR, VR, MR) for the different training phases. Mixed Reality (MR or XR) appears to be an interesting application in almost all phases, ranging from the first introductory phase to the operational and end phases.

A specific case is worked out by Bhattacharya, B. et al. [5], where AR is used to assist with complex assemblies. Using AR authoring, the assembly can be first recorded by a professional and used afterwards for new trainees. During the training, textual information and 3D digital objects are used to guide the user towards the desired end product. The information is displayed on a screen before the user, instead of using a head-mounted display. From their studies, AR appeared to be the most effective system for this application, instead of VR or a manual guide.

Outside of manufacturing, XR environments can be used in other domains, such as safety, military and medical training as well as education [12]. The main idea behind these concepts is often the need for highly efficient, cost-effective and optimized working conditions. XR could provide a flexible and effective tool for educating new workers with the needed skills. On the other hand, these tools also provide a safe alternative to training in real scenarios. For example, for medical training, the students could exercise as often as needed, without additional costs or health risks for the patients. Alternatively, novice workers can be trained without any risk in dangerous environments, where mistakes could be harmful to themselves or the equipment. Lastly, it is also noted that VR environments can even be used in language learning. Users indicated the overall enjoyment of learning with these tools, but retention of the subjects also increased.

An important comment is made by Doolani, S. et al. [12] about the real-world application of XR environments. It is noted that tools which require head-mounted displays are often harder to roll out and adopt than tools which only rely on mobile devices. Methods depending strongly on head-mounted displays are more often used in research contexts.

The success of using XR environments for manufacturing training is reviewed using many metrics such as performance, time taken to complete assignments, user feedback and more. The performance of these methods also greatly depends on the hardware and software used inside the devices and tools.

Building Information Modelling (BIM)

Building Information Modelling (BIM) refers to a representation of the physical and functional properties of a building [45]. BIM can aid with design consistency, cost estimation, visualisation and more. This can help with the recycling and maintenance of long-lifetime buildings.

As mentioned by Alizadehsalehi, S. [1] construction progress is often measured using pictures, visual inspections, technical drawings and reports. To increase the efficiency of these inspections, XR environments can be used to visualize progress in an interactive manner.

Prouzeau, A. et al. [33] discusses another use, where digital twins are used to aid with maintaining buildings. Digital twins are copies of physical assets. A digital twin can be made of a room, where other information such as the location of ventilation, and electricity... can be added. Several use cases are discussed in the paper. The idea is that maintenance personnel can enter a room, scan a QR code and see relevant information using an AR display. In this way, it would be quicker to find, for example, the reason for a high temperature in the room or a fault in a pipe. This inspection would even be possible from a remote location, as the user can experience the digital environment from anywhere.

Context aware applications

A final set of applications for eXtended Reality consists of building virtual environments in which users and their avatars can interact with each other and their surroundings. Lee, Y. et al. [26] focuses on XR authoring for building a collaborative, virtual environment where workers can interact remotely. The paper mentions the possibility to deliver virtual information in an intuitive manner, as well as the benefits of spatial awareness when looking around a virtual environment. In their experiments, Lee, Y. et al. [26] created a web browser application, which can be used together with a tablet to share the virtual environment.

Finally, Wang, T. et al. [47] developed CAPturAR, which is a tool to record daily activities using a head-mounted device. Once the information is captured, users can create their own AR applications. These can be used to remind them of taking medication, exercise and more. Different use cases are discussed such as healthy life operation and augmenting non-smart objects. Even though user feedback is positive, the device used for capturing the environment and activity is still bulky, limiting the movement of the user.

2.1.2 Authoring Techniques

In this section, a non-exhaustive list of different authoring techniques and toolboxes is presented. As mentioned by Hendricks, Z. et al. [20], there are two steps in XR authoring: the modelling process and the programming of the interactions between the user and the virtual environment. Here, the focus lies on the modelling of the environment and the creation of virtual assets. However, the programming process is often difficult and a major challenge in authoring methods. Toolboxes designed for novice users often lack the possibility of using XR for their specific purpose, while other tools require experienced programmers. Some existing tools for programming interactions include RhoVer [3], CoRgi [11], Avango [44] and DIVE [14].

Authoring tools are defined by Bhattacharya, b. et al. [5] as tools combining all the relevant information, such as sensor readings, objects and other graphical info, to create AR, more general XR, data. The typical process of creating these environments starts with the creation of virtual objects. This is achieved by modelling the objects using 3D software (Maya, Studio Max, Blender) or by capturing data using laser-scanning methods [1]. The latter is achieved using 2D or 3D near-depth sensors or head-mounted devices. Afterwards, these components are placed inside the environment. The placement is either done manually or automatically, possibly with respect to a marker. A maker is a symbol or picture used for fast recognition and registration [5].

Bhattacharya, b. et al. [5] use depth sensors in order to capture visual information. Their technique called AREDA is used for the authoring of assemblies. The assembly is first constructed by an experienced worker and simultaneously captured by the Microsoft Kinect camera. Using an algorithm called RANSAC and markers, a plane can be fitted on which the objects are present. The data consists of a point cloud which is post-processed to achieve the desired results. The skin of the hand is removed from the picture and by comparing different frames, single assembly components are extracted. By loading 3D models of the same components into a library, point cloud registration is performed to find the pose of the objects. In this case, Principal Component Analysis (PCA) combined with Iterative Closest Point (ICP) is used for the registration process. The latter is selected because of its robustness and simplicity. More complicated algorithms might be more accurate but would require more computing time. An overview of Point Cloud Registration techniques can be found in Appendix A.

Similarly, Alizadehsalehi, Z. [1] discusses using laser-scanning methods to capture point cloud data from a construction scene. Even though this might require the aid of experienced workers and is more expensive, the results are more accurate than image-based methods. The point cloud can be post-processed afterwards and converted to a mesh or BIM. The data can also be aligned with a template model to visualise the as-built state of the construction. Volk, R. et al. [45] mention the different post-processing possibilities of using point clouds. Object detection can be performed to extract objects of interest, noise and clutter can be removed and image alignment can be performed. Wang, T. et al. [47] use the depth information together with Yolo v3 [35] for object detection. Once the object is detected in the 2D image, the result is projected back onto the depth information to extract the object.

Lee, Y. et al. [26] use both 3D-created models and depth camera information for authoring the digital environment. Objects Of Interest (OOI) are created as a model since they will often move inside the environment. The surroundings however remain mostly static, such that depth cameras can be used to capture the environment, to limit the workload. The OOI is then also masked or removed from the background point cloud, containing the Ambient Objects (AO).

Prouzeau, A. et al. [33] use digital twins of the environment in order for the user to be able to easily place the virtual assets inside of the room. Inside the digital twin, a marker is placed where the information should be visualised. Afterwards, a marker is placed at the same location inside the real world. In this way, digital twins are used to easily author the environment and add digital assets, without needing to be present inside the real environment. Aside from markers, the paper also proposes machine-learning techniques for matching real objects and their virtual copies.

Tools such as Unity3D, which is a game engine, are used for working with 3D environments, while, for example, Vuforia, an AR library, is used for the registration process. Other methods exist like Powerspace or AMIRE, which are linking systems used to link the virtual content with the real world using markers.

AR Core is an authoring tool for programming situated visualisations[1] manually.

Based on the idea of "What You eXperience Is What You Get", iaTar lets the user immediately experience the created AR environment [33].

Farrago [51] is an IOS application which allows the creation and viewing of 3D assets directly in the camera view. It also uses markers for tracking. Objects are created using a so-called object brush, where a mesh is attached to the markers, allowing the users to create complex geometries based on basic templates.

In order to train, for example, the YOLO v3 method, with labelled training images, a technique proposed by Laielli, M. et al. [24] is used. The idea is to use AR to extract objects from their surroundings when captured by a depth camera. The user indicates the position of the object by creating a 3D volume

[1]A situated visualisation refers to a tight link between the virtual information and real physical environment [33].

around it. For obtaining the pose of the 3D object volumes, SLAM is used. SLAM is another AR tool for mobile devices, which builds a map of the environment and locates the devices inside of it. For this, it uses the gyroscope and accelerometer sensors inside of the mobile device. Once the 3D object volume is found and correctly aligned, it is projected in 2D for finding an accurate bounding box around the object.

2.1.3 Conclusions

Extended reality applications are extensive. Use cases range from education to vocational training and entertainment. User feedback in real working environments is overall positive, although tools relying on heavy head-mounted displays are difficult to roll out commercially.

For the authoring of the environments, many tools exist. Environments are created using 3D models (CAD) or using laser-scanning technologies. The latter allows different post-processing steps to extract and visualise the information of interest. For this reason, the scanning-based approach is further investigated in this thesis. The Intel Realsense camera is used to capture a point cloud of the studied objects.

3D scanning is a promising tool for the creation of virtual environments, but requires experienced personnel and is more expensive. Maintenance of the digital environments and the quality of 3D representations are other challenges still present in this domain. The problem of object alignment is looked into in this paper.

2.2 Point Cloud Registration Applications

Point Cloud Registration (PCR) is the problem of aligning two point clouds with each other. Typically, this means finding the rigid transformation that transforms the source point cloud into the template point cloud. An example is shown in Figure 2.1. A more complete discussion on Point Cloud Registration is given in Appendix A.

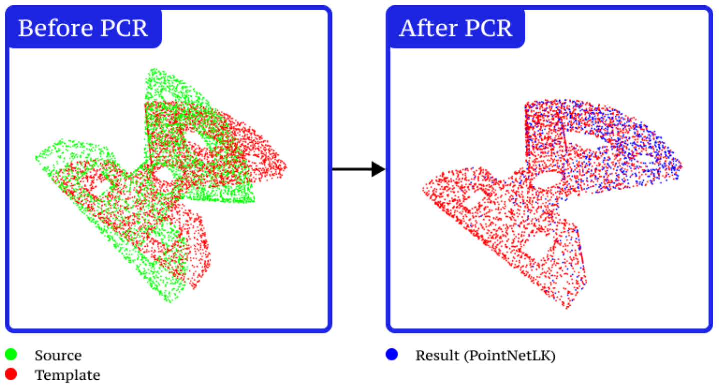


Figure 2.1: The basic idea behind Point Cloud Registration: a source point cloud (green) is aligned with a template point cloud (red). The result is shown on the right (blue) after applying PointNetLK.

Point Cloud Registration has many uses and applications. In this section, some relevant cases are discussed. The problem of registration and its several solutions is of use in robotics applications, computer vision, construction, autonomous driving and more. As discussed in Appendix A, much research exists on different techniques to solve the registration problem. However, the obtained methods are usually tested on synthetic data. This is data on which a rigid transformation is applied inside the computer. The benefit of this is the knowledge of the ground truth solution, to accurately assess the method's performance. It is the aim of this thesis to rather test the methods on real-world data, captured by a 3D camera. The examples discussed below often deal with similar problems, where noise and partiality[2] introduce additional difficulties.

[2]Partiality refers to the fact that point clouds, captured by a depth camera, are incomplete. Only one angle from the object is captured, not the complete cloud.

2.2.1 Point Cloud Registration In Robotics

Jerbić, B. et al. [22] discusses the idea of using a robot arm combined with Point Cloud Registration to achieve a full scan of an unknown object. The robot arm gripper is placed before a stereo camera and captured into a point cloud. Afterwards, the gripper rotates over a set angle and a new point cloud is registered. ICP is then performed to find the rigid transformation, which can be compared to the real rotation enacted by the robot itself. If the results are sufficiently close, the clouds are concatenated. After a full 360° revolution, a complete training template is obtained for the object. This template can, in turn, be used for object recognition or registration in the scene.

The idea of using a robot to align the objects, in order to know the ground truth, allows for the registration process to be evaluated. However, as also explained by Jerbić, B. et al. [22], the object point cloud must be separated from the gripper holding it.

Ciocarlie, M. et al. [8] propose a framework for household object grasping by a robot. The robot is equipped with stereo cameras and laser range sensors to capture a point cloud of the environment. This point cloud is first segmented to distinguish the objects from the surroundings. For this, a plane is first fitted, since household objects are often placed on a flat surface. Afterwards, the closest points based on the Euclidean distance, to the plane are detected, representing the objects. These are then compared to a library of object models. Using an ICP variant, alignment is achieved. If the match between the segmented point cloud and the model exceeds a certain threshold, the object is detected. This information is then used for collision detection and movement planning.

2.2.2 Point Cloud Registration In Construction

Similarly to the discussion in Section 2.1.1, Qian, W. et al. [46] discuss the use of Point Cloud Registration in construction. The idea is to capture point clouds using 3D cameras from different positions and combine them into a mapping of the construction site. This allows for quicker inspection, monitoring and comparison of "as-designed" and "as-built" conditions.

Liu, S. et al. [29] proposes another method of environment recreation using PCR. Similarly to the paper by Jerbić, B. et al. [22], multiple pictures are taken using a depth camera and recombined using ICP. However, the ICP algorithm is modified to take into account the depth measurement errors of the RealSense Intel camera. These errors scale quadratically with the depth measurements. For this reason, points are weighted depending on their error when minimizing the loss function.

A final example of 3D mapping is given by Mahmood, B. et al. [30]. In this paper, RANSAC, another PCR technique is used to reconstruct indoor scenes. The results are aimed to be used in interactive AR devices. RANSAC is used because of the many similar features in indoor scenes, such as doors and windows. The algorithm is able to reject false correspondences leading to an accurate reconstruction.

Even though this thesis is not aimed at creating a mapping of the environment, similar techniques and steps are still relevant. The capturing of the data using the right camera and the cleaning of the obtained point cloud to remove noise and outliers are two crucial steps when working in real-world conditions.

2.2.3 Point Cloud Registration In Real-Time

Donghoon, L. et al. [25] discusses the problem of Point Cloud Registration in the case of unknown objects captured with a 3D camera. As already mentioned before, most methods are tested using synthetic data, for which CAD models exist. These can be altered slightly by adding Gaussian noise to mimic distortions introduced by the camera. However, these synthetic data sets often do not properly model effects such as self-occlusions, partiality and noise.

In this paper, Point Cloud Registration is applied online, on unseen objects, without prior knowledge of any CAD models, in real-time. Opposite to many other papers, this means the input of the system consists of two partial point clouds, overlapping only partially. For this reason, training is done using the Linemod dataset [6], which consists of real-world data and their camera pose to estimate the ground truth.

Lastly, the paper mentions the need for learning-based methods when working in real-time applications. ICP and its variants are often slower due to their iterative nature. However, they can be used for further offline refinement.

2.2.4 Conclusion

Point Cloud Registration has many applications in robotics, construction and more. Point clouds are often captured using depth cameras such as Microsoft Kinect or Intel RealSense. After some pre-processing, such as filtering or segmentation, the relevant objects are extracted and PCR is applied. In many cases, ICP is used for its robustness and simplicity. In this thesis, other techniques are compared to verify whether they can provide more accurate results when dealing with data captured by depth cameras.

2.3 Point Cloud Registration Techniques

Point Cloud Registration is a well-researched problem with many different solutions available in the literature. Overviews[3], containing some of the most prominent techniques, have been made by Li, L. et al. [27], Gu, X. et al. [17] and Zhang, Z. et al. [56].

2.3.1 Selection Criteria

To make a selection of these methods, criteria are used based on the performance of the techniques and their availability and flexibility. These criteria are also used in literature to describe the properties of the algorithms. This allows to summarize the different methods as in Tables A.2-A.3 (Appendix A). Since this project concerns the alignment of a source, captured by a 3D camera, noise and partiality have to be taken into account. Performance criteria considered are:

- Robustness to noise
- Robustness to partiality
- Limited computation time
- Generalizability to different objects or training capabilities

Noise or measurement errors are unavoidable when working with real data. Especially in working environments, containing multiple objects and non-optimal lighting conditions, noise is an issue. For this reason, the selected techniques have to be able to take this into account.

When capturing the environment, objects might only be visible from a certain angle. This means the registration methods have to be able to work with partial information. As mentioned before, working environments are often cluttered with numerous objects and making a precise, complete scan for all of them is time-consuming. Hence the reason for selecting methods that are able to work with the partial scans. Computation time is another important consideration, especially when applying registration for real-time applications.

Lastly, it is important for the methods to yield accurate results for a wide variety of objects. Learning-based methods can be trained with the available template CAD models, in order to identify relevant features for the used object set. Non-learning-based methods are typically more generalizable, although this may come at the cost of lower accuracy or longer computation.

Since it offers a lot of packages and libraries for data processing, Python is mainly used for coding. The programs are run on a Windows laptop with an NVIDIA GeForce RTX 3070 GPU and AMD Ryzen 7 processor.

2.3.2 Selected Methods

Based on the mentioned criteria in Section 2.3.1 and the summary made in Appendix A and Tables A.2-A.3, the following methods are chosen. Three categories are made. Firstly, the global registration methods, which are not learning-based. Secondly, the learning-based methods, which require training. Lastly, the ICP-based methods are used for refinement, after applying a global or learning-based registration technique. More details about the methods are presented in Appendix A. In this section, each method is briefly discussed, more specifically its major benefits and pitfalls.

[3]An overview is given as a GitHub repository by XuyangBai [52] with an extensive list of methods and feature detectors.

Global Registration Methods

Global registration methods do not require any training process, which makes them often easy to implement and use. The following methods are some popular examples. RANSAC and FGR are implemented from the Open3D Python library [60], GO-ICP is taken from the authors code [53].

- **RANSAC** [13] is a popular registration method for scene reconstruction. It shows good performance for partial data and is robust to noise and initialization. No training is required, however, RANSAC requires a preliminary step for feature extraction. Aside from this, there are also different parameters to initialize and tweak.
- **Fast Global Registration (FGR)** [59] has the major benefit of being very quick. It is also used for partial registration and does not require any training. However, some papers [54, 63] mention the sensitivity of FGR to noise.
- **Globally Optimal ICP (GO-ICP)** [53] is robust to noise and improves on the basic ICP method by finding a globally optimal solution. Its major downside is in the speed of the method.

Learning-based Methods

Learning-based methods use machine learning and therefore require training. The training process itself can be quite computationally expensive and time-consuming since it is important for the obtained model to not underfit or overfit the data. The quality of the data is thus also crucial. The more the data resembles the real world data, the better the results. On the other hand the more training might be required for the model to fit. Learning-based methods are often more robust and fast (after training). All the methods described below are implemented from the learning3D Python library [37], except for ROPNet, which is taken from the author's provided code [63, 62].

- **PointNetLK** [2] uses PointNet to extract features from the point clouds. It is noise robust and also works on partial data. Performance however drops when applied to unseen categories of data and for large transformations. Even though the latter is a major disadvantage, in this case, PointNetLK is trained on the available CAD models of the objects. This means the problem of unseen data categories is not relevant here.
- **Partial Registration Network (PRNet)** [49] is specifically designed for partial registration. It yields accurate results even when the point clouds only partially overlap. However, it suffers from the same issue as PointNetLK regarding larger transformations.
- **Robust Point Matching Network (RPM-Net)** [54] is more robust to initialization. This means it also works for larger transformations. RPM-Net is robust to noise but slower than other methods such as ICP and DCP.
- **Representative Overlapping Points Network (ROPNet)** [63] is another method, specifically created to solve the partial overlap problem. It is robust to noise and has good generalizability.

ICP-based methods (refinement)

Iterative Closest Point (ICP) [21] and its variants are very popular and basic registration methods, which are often used. The main problem with ICP is its lower performance for larger transformations. For this reason, ICP is often used as a refinement step after applying a learning-based or global registration method. ICP works by establishing correspondences and evaluating a metric. The metric can be based on point-to-point distances, point-to-plane or plane-to-plane. These might yield different results depending on the case.

Chapter 3

Metrics

In order to compare the registration methods, selected in Section 2.3, metrics are required. This chapter contains a summary of the major metrics used during the experiments. A mathematical definition for each criteria is given in Appendix B. Here, the meaning behind each metric is further discussed and their implementation.

3.1 Metrics Selection

The metrics used for evaluating the performance of the registration methods can be classified in different groups. The first group of metrics evaluates the rotational and translational errors in degrees and a measure of length. These metrics are the easiest to interpret and are directly linked to the rigid motion estimation. The rotation and translation errors are computed in different ways. A first approach is to use the relative errors. The relative rotational error is computed using Equation B.1. It is a measure of the remaining rotational angle after first transforming the object using the ground truth, and afterwards the inverse (or transposed) of the predicted rotation matrix. When the registration is perfect, $R_{pre}^T = R_{gt}$, and the relative rotation error becomes zero.

Another set of rigid motion metrics are the root-mean-square errors. These errors compute the difference between the expected result and the obtained result. The difference is squared, averaged and finally the root is taken. The formulas are given by Equations B.4.

Lastly, the absolute errors are also computed. These represent the absolute difference between the ground truth and Euler angles or translation vector and the one obtained from the registration process.

Secondly, there is a group of metrics evaluating the accuracy of the prediction in a more general way. The Recall and Coefficient of determination (R^2) tools make up this group. They are also linked to determining whether or not a result is considered successful or not. The Recall coefficient is given by Equation B.5 and corresponds to the percentage of points for which the mean-squared-error (computed with the complete transformation matrix) is lower than a given threshold. For this, the point cloud transformed using the ground truth and estimated transformations are compared. An example of these two point clouds is given in Figure 3.1.

The Coefficient of determination R^2 is computed using Equation B.6. It is a measure of the fit of the predicted result with respect to the ground truth. Ideally, the two point clouds perfectly overlap, resulting in a value of one for R^2. Notice that it is also possible for R^2 to become negative in this context. This can be seen from Equation B.6, when the sum of squared distances (SSD) becomes larger than the total sum of squares (TSS). This means that the the squared distances between the prediction and the ground truth are larger than the square distances between the point and the mean of the point cloud. A typical example of a negative R^2 value is shown in Figure 3.2.

The registration process can also fail. This means the result does not match the template in a satisfactory way, resulting in large errors. It is not fair to estimate the performance of the methods by also considering these cases when computing the mean errors. Instead, the number of failure cases are saved, as well as the scan it relates to. This results in a final metric, which is called the *number of failure cases* and is expressed as a percentage of the total number of considered scans. This metric essentially gives an

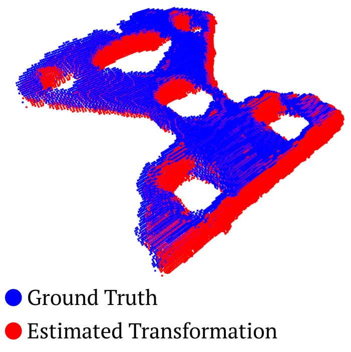


Figure 3.1: Example of the *Base-Top_Plate* object, where the source, transformed with the ground truth (●) and estimated transformation (●) are compared, in order to compute the recall metric.

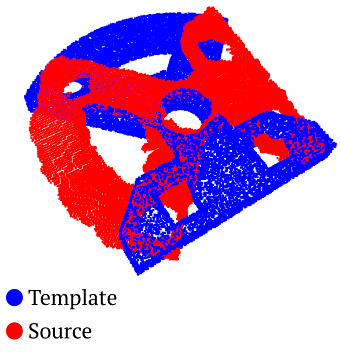


Figure 3.2: Example of a solution with a negative R^2 value. The estimated transformation (●) clearly does not overlap the template (●). Result shown is for the Base-Top_Plate object, after applying PointNetLK.

idea of the required quality for the scanned point cloud. When the number of failure cases is low, the method is robust to different angles and positions of the object. A large number of failure cases indicates that the method only works for certain scans, while it fails for other scans, containing for example less information.

The implementation of the errors is realised in *compute_errors.py* and the *errors.py* class, in the code available on Github. When multiple symmetric solutions are possible, the code computes the different possibilities and selects the one yielding the highest recall value. Finally, the R^2 value is verified together with the Mean Relative Angular Error (MRAE). The registration result is only accepted when the R^2 value is positive and the MRAE is below 120°. The last condition is arbitrarily chosen. However, when the rotation error reaches as high values as 120°, it is clear the result is no longer satisfactory.

A final metric that is used to asses the performance of the algorithms is the computation time. The time is measured from the point of starting the registration process - thus not including the loading of the data - up to the point where the transformation matrix is obtained. In order to achieve a fair comparison, all the registration methods are run multiple times, on the same computer, in the same conditions[1].

3.2 Conclusion

This chapter contains a summary of the metrics that are used during the experiments. There are four types of metrics considered. The first group consists of metrics evaluating the quality of the rigid motion estimation in terms of rotational and translational errors. The second group determines the quality of the point cloud alignment and whether or not the result is considered satisfactory. The third group evaluates the number of failure cases and the scans they relate to. Finally, the computation time makes up the final group of metrics. A summary is also shown in Table 3.1.

[1]During the computation of the registration, no other programs are run in the background. However, it is difficult to completely remove the effects of background tasks. For this reason, the algorithms are run multiple times. The results should be interpreted considering the used specifications and with the possibility of small variations.

Table 3.1: Overview of the different metrics used during the experiments.

Metric	**Group**	**Meaning**	**Unit**
Mean Relative Error (MRE)	Group 1	Relative translational and rotational errors	$^\circ/mm$
Root Mean Square Error (RMSE)	Group 1	RMS of errors between ground truth and estimation	$^\circ/mm$
Mean Absolute Error (MAE)	Group 1	Absolute difference between ground truth and estimation	$^\circ/mm$
Recall	Group 2	Percentage of points for which MSE is smaller than given threshold	%
Coefficient of Determination (R^2)	Group 2	Fit of the predicted point cloud with template	/
Number of Failure Cases	Group 3	Number of results for which $R^2 < 0$ and $MRAE > 120^\circ$	/
Computation Time	Group 4	Time for the registration process	s

Chapter 4

Experimental Setup

To finalize the preparation of the experiments, a setup is created to capture the point clouds. First, the datasets and objects used during the experiments are mentioned in Section 4.1. After, the experimental conditions are given in Section 4.2. Finally, a summary of the different steps can be found in Section 4.3.

4.1 Datasets

Before performing Point Cloud Registration, a template point cloud must first be created. This data can be acquired from typical datasets such as ModelNet40 or Linemod [25]. The dataset can be generated on a computer, which is called synthetic data. On the other hand, the data can also be captured using a 3D camera, resulting in real-world data. The latter often contains noise or represents only a certain part of an object.

In this paper, two main datasets are considered. The Cranfield benchmark dataset [9] is used as a benchmark to evaluate robotic assembly operations. The ModelNet40 dataset [58] is typically considered to compare the different Point Cloud Registration methods with each other.

4.1.1 Cranfield benchmark

The Cranfield benchmark dataset contains six unique objects, visualised in Figure 4.1. CAD models of these objects are available to generate the required templates and training data. For the experiments, the objects are 3D printed, as shown in Figure 4.1. They are thus non-reflecting, as reflections on shiny surfaces are not considered in this paper.

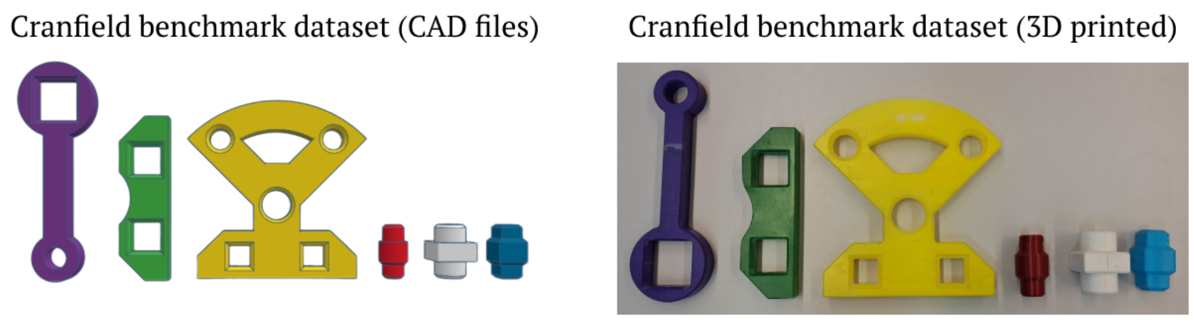

Cranfield benchmark dataset (CAD files)
Cranfield benchmark dataset (3D printed)

Figure 4.1: Left: 3D CAD models of the Cranfield benchmark objects. Right: 3D printed Cranfield benchmark objects.

This dataset is chosen for several reasons. Firstly, the objects are used to compare robotic Peg-In-Hole (PIH) assembly operations. This operation requires a high amount of accuracy to be performed correctly. For this reason, it is interesting to consider the accuracy of Point Cloud Registration for these types of objects. Secondly, the objects have relatively simple shapes, without too much details. They are geometric and often times symmetric. Similar objects are expected to be found inside a manufacturing environment.

4.1.2 ModelNet40

The ModelNet40 dataset, created by Zhirong, W. et al. [58], consists of 12,311 objects from 40 different categories. It is often used to compare different registration methods with each other. The CAD models are each verified to be correctly classified and cleaned from any irrelevant information, such as floors. The categories include a wide variety of objects, like cars, airplanes, vases... Some examples are shown in Figure 4.2.

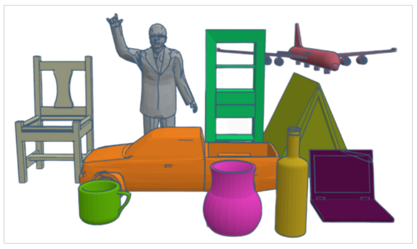


Figure 4.2: A selection of ModelNet40 objects from different categories (chair, person, bookshelf, airplane, tent, car, cup, vase, bottle and laptop).

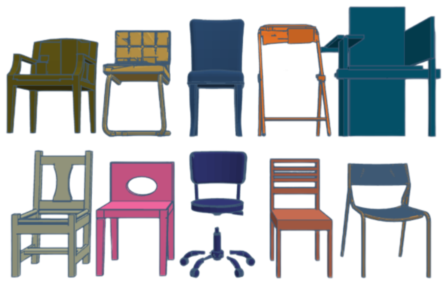


Figure 4.3: Collection of 10 different ModelNet40 objects all contained under the "chair" category.

The differences between these two datasets are summarized in Table 4.1. The six objects from the Cranfield benchmark dataset are all unique, while those of the ModelNet40 dataset are part of a category. Inside this category, different variations of the same object exist, as shown in Figure 4.3. This, together with the difference in size, has an effect on the construction of the training dataset for the learning-based registration methods, as is explained in Chapter 5. The symmetry of the objects is another important consideration to be taken into account when interpreting the results. Symmetric objects introduce additional solutions to the registration problem, as is shown in Figure 4.4. This means that the found transformation matrix may still be a correct solution to the problem, even though it does not exactly correspond to the ground truth transformation, as there are several ground truth solutions.

Table 4.1: Main differences between the Cranfield benchmark and ModelNet40 datasets.

Property	Cranfield benchmark	ModelNet40
CAD models	6	12,311
Categories	6	40
Symmetry	All objects symmetric	Symmetric & non-symmetric objects
Application	PIH assembly	Point Cloud Registration

Since pretrained models are available for the ModelNet40 dataset and because it is generally used as a benchmark for testing registration methods, three objects are also 3D printed from this dataset. These are chosen for their print-ability and relatively simple geometries. The objects are shown in Figure 4.5.

4.2 Experimental Setup

Aside from synthetic data, generated on a computer, the point clouds can also be captured using a 3D camera. Registration is then applied to align the obtained real-world scan with the CAD model of the same object. In order to verify the result, the ground truth transformation is compared to the one obtained after registration. This section handles the major considerations of these steps.

4.2.1 Environment

The experiments are all conducted inside a well-lit, indoors environment (Figure 4.6). The objects used are 3D printed and matte. They are placed upon a light coloured table to a create contrast with the background.

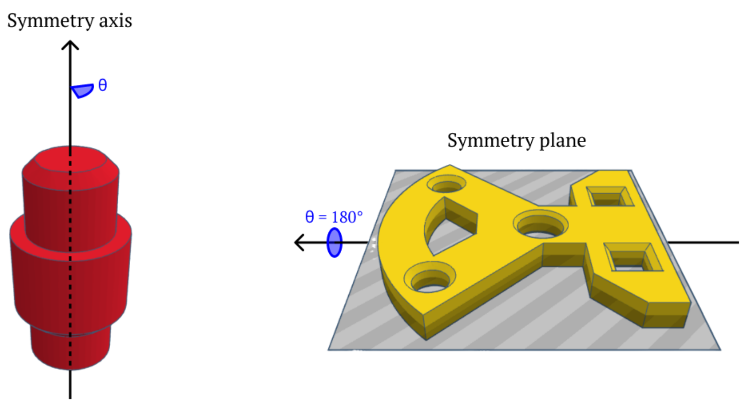


Figure 4.4: Left: The *Round-Peg* object from the Cranfield benchmark dataset contains a symmetry axis, indicated by the arrow. Every rotation (blue) around this axis, does not change the pose of the object in a meaningful way. In this case, there are an infinite amount of ground truth solutions. Right: The *Base_Top-Plate* object has a symmetry plane. Mirroring the object around this plane results again in the same pose.

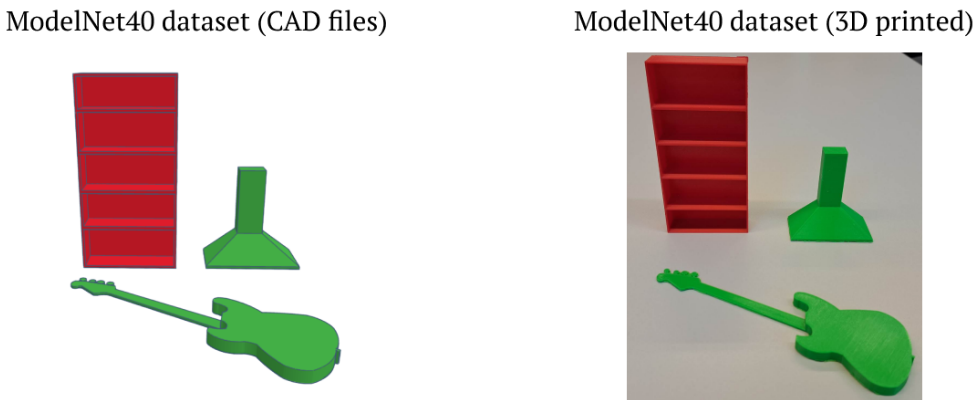

ModelNet40 dataset (CAD files)
ModelNet40 dataset (3D printed)

Figure 4.5: Three objects from the ModelNet40 dataset, 3D printed.

4.2.2 Intel RealSense Camera

Working Principle

In order to capture the objects in the real world, a depth camera is used. More specifically, the Intel RealSense D4xx series camera captures the objects and saves them into a point cloud or *.ply* file. These cameras are able to determine the depth information using stereo-vision. The working principle is explained by Tadic, V. et al. [41]. The RealSense cameras are fitted with an infrared projector and a left- and right side camera. The two side cameras capture the scene separately. After, the processor correlates the same points between the two images in order to compute the depth information. The idea illustrated in Figure 4.7.

Different types of RealSense cameras are available on the market with varying accuracies, field-of-views, resolutions and depth capabilities [10]. During the experiments, the D455 and D435i variants are mainly used. The D455 camera provides a more accurate depth map, however, its minimal depth range is limited to 50cm. The D435i camera is able to capture depth already at 30cm, which allows for a more accurate scan for the smaller objects. Since the objects used during the experiments are not very large - the largest object fits inside a 20x18x2cm box - the D435i camera is used to capture the scans. This camera also has an Inertial Measurement Unit (IMU) incorporated, which is useful for robotics applications. Other cameras, such as the D415 and D405 cameras are also available on the market. The D415 camera achieves a high accuracy [41], while the D405 version has a minimum depth range of only 7cm [10]. Tests are also performed with the Realsense Lidar L515 camera. However, the captured point clouds are of lower quality, hence why this camera is not further discussed.

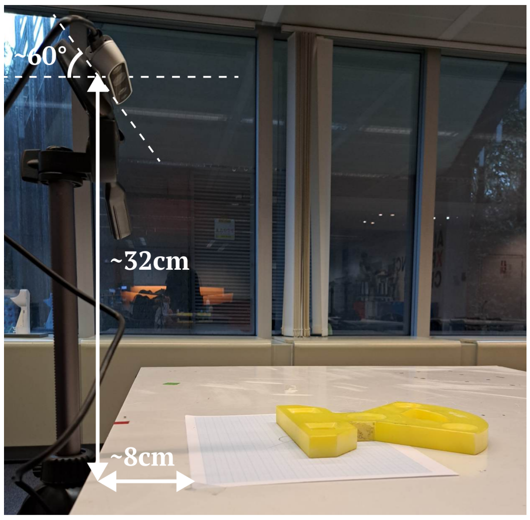


Figure 4.6: Example of the setup used during the experiments. The camera height and angle are variable to obtain accurate scans of the objects. The paper is used to determine the ground truth (Section 4.2.3).

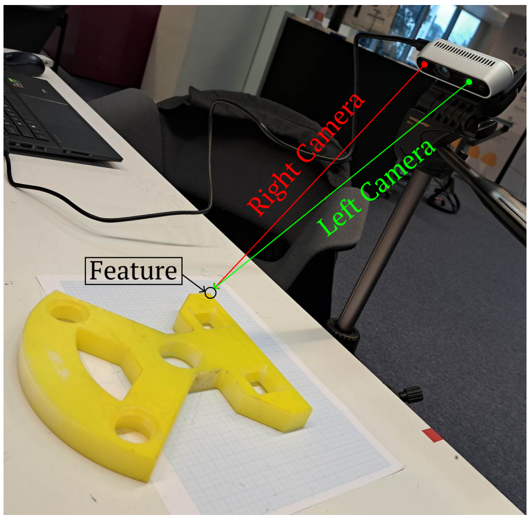


Figure 4.7: Concept of stereo-vision. The left and right camera identify the same feature, allowing the processor to estimate the depth.

Filtering

The point clouds are captured by the camera and saved using the RealSense viewer software (Figure 4.8). The program allows to perform some initial processing in order to increase the accuracy of the scans. The settings used for the camera are saved inside a *.json* file and added to the Github page.

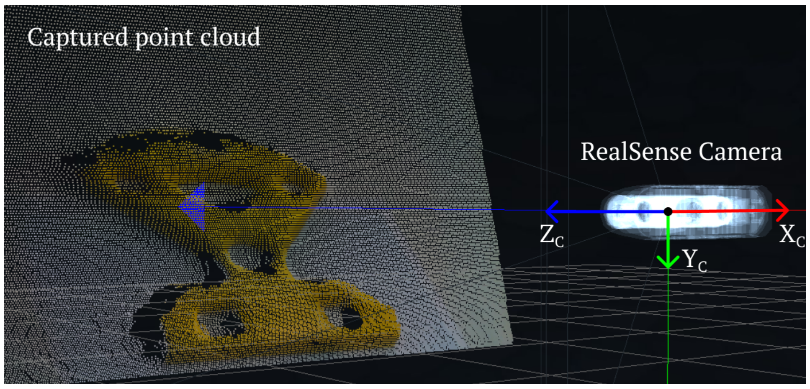


Figure 4.8: Example of the RealSense viewer software.

Before applying registration to the scanned point cloud, the objects first need to be extracted from the environment. As mentioned before, segmentation and detection of the objects is not within the scope of this thesis. However, the Open3D toolbox in Python still provides some basic functions to remove the background information. The process is shown in Figure 4.9, coded in *pcl_post_processing.py* and consists of the following steps.

1. The floor or table is first detected using the *segment_plane* function. This allows to split up the point cloud into the objects, floor (and clutter). The function also allows to estimate the normal vector required to estimate the ground truth (Section 4.2.3).
2. The remaining point cloud is segmented into the different objects using the *cluster_dbscan* function. The user should then manually choose the index of the desired object to retain.
3. Outliers are removed from the extracted object with the *remove_statistical_outlier* function. This is done to make sure only the relevant points are included for the registration process.
4. The oriented bounding box is estimated using the *get_oriented_bounding_box* function. Even though

the object is already obtained in Step 3, it is interesting to be able to change the size of the oriented bounding box to assess the effect of including environmental clutter (floors, noise...).

5. The oriented bounding box allows to extract all the points inside using the created *inside_rect* function. The bounding box is scaled as desired, to study the effects mentioned before.

6. The normal information of the point cloud is estimated using the *estimate_normals* function.

7. The final point cloud is saved inside a *.txt* file to be used in the remaining process.

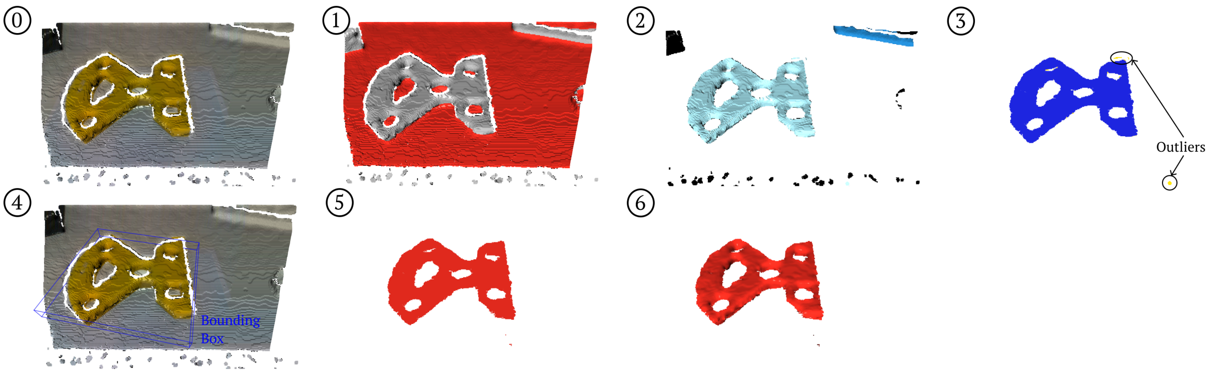


Figure 4.9: Different steps of the pre-processing procedure. 0. The initial scan, obtained with the RealSense D435i camera. 1. Plane (red) detection and segmentation from the objects (grey). 2. Segmentation of the remaining objects into the object of interest (light-blue) and clutter (black). 3. Removal of the outliers (yellow). 4. Estimation of the oriented bounding box overlaid with the original scan. 5. Extraction of the points inside the estimated bounding box. 6. Estimation of the normals.

4.2.3 Ground Truth Determination

In order to be able to assess the performance of the results, using the metrics described in Chapter 3, a ground truth transformation is required. Determining the ground truth is not a trivial task, as the used RealSense cameras do not allow to determine the absolute pose of the camera with respect to the positioned object. To solve this problem, the camera is placed such that it is parallel to a reference marker (piece of paper) on the table, as indicated in Figure 4.6. The object is placed on the reference marker and oriented as desired. The ground truth transforms the template point cloud, attached to the camera center, into the source point cloud, placed on the table. This transformation is now characterized by two rotations and a translation. The complete process is visualised in Figure 4.10. The following steps are performed:

1. The template is rotated over an angle θ_X around the X-axis. This operation aligns the Y-axes of both objects with each other. The angle θ_X corresponds to the angle between the Y-axis of the template coordinate frame (Y_T) and the normal to the plane, on which the objects sits (Y_S). It is computed using the scalar product.

$$\vec{Y_T}\vec{Y_S} = |\vec{Y_S}|cos(\theta_X) = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} Y_{SX} \\ Y_{SY} \\ Y_{SZ} \end{pmatrix} = Y_{SY} \tag{4.1}$$

$$\Rightarrow \theta_X = cos^{-1}\left(\frac{Y_{SY}}{|\vec{Y_S}|}\right) \tag{4.2}$$

2. The template is rotated around the new Y-axis (Y_T') over an angle θ_Y. This angle depends on the orientation of the object on the table and is chosen as desired. When the object lies parallel to the template, the angle is zero.

3. The template is also shifted by a translation vector. A first guess is obtained by using the mean center of the source point cloud on the table. Afterwards, some corrections are added to make the two clouds align. This step can also be performed as a first step as in Figure 4.10, where it is placed first for clarity.

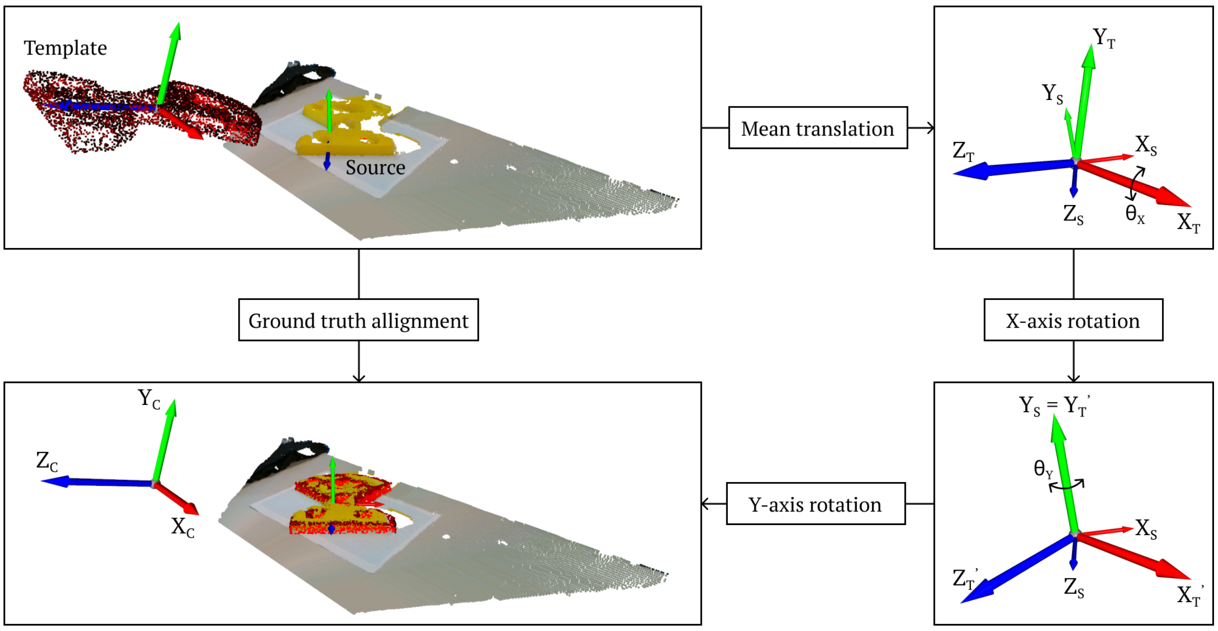


Figure 4.10: Overview of the steps required to approximate the ground truth transformation. The red point cloud represents the template, initially positioned in the camera center. The yellow point cloud is the source, captured on the table. The coordinate systems $X_T Y_T Z_T$ and $X_S Y_S Z_S$ are attached to the template and source respectively.

The final ground truth estimation is only an approximation of the real transformation. The captured scans are already subject to measurement noise. This results in some deformations of the scanned object, which means it will not align perfectly with the original template. Thus, the transformation is corrected in the end, until the desired alignment is achieved. It is likely that the obtained transformation from the registration methods differs with a couple of degrees and millimeters from the computed ground truth result. This is to be taken into account when analyzing the results in Chapter 6. The described process is automated inside the function *input_writer.py* which can be found on the Github page.

4.3 Procedure

The flow of operation for the experiments is visualised in Figure 4.11. The obtained *.hdf5* file, containing the source and template point clouds and the ground truth transformation is used in the subsequent steps. Before applying the registration, the other, symmetric ground truth transformations are also added to the same file. For objects, such as *Round-Peg*, with an infinite amount of solutions, only a couple of transformations are added. Adding the symmetric solutions is performed through the *add_symmetric.py* function.

After applying the initial processing, registration is performed with the methods selected in Section 2.3. The estimated transformations are saved in a *.hdf5* file. The results are used to compute the metrics, explained in Chapter 3, visualise the results or perform a refinement step. The refinement method, ICP, also returns a file containing the found transformation matrices. These can again be used to compute the errors or visualise the results.

4.4 Conclusion

In this chapter, the process of the performed experiments is discussed, as well as the datasets used. The objects are part of the ModelNet40 and Cranfield benchmark datasets. The latter is typically used to asses PIH assambly operations, while the ModelNet40 objects form a benchmark for registration.

During the experiments, the RealSense D435i camera is used. The scanned point clouds are first filtered and pre-processed to extract the object of interest. The ground truth is estimated using the normal vector

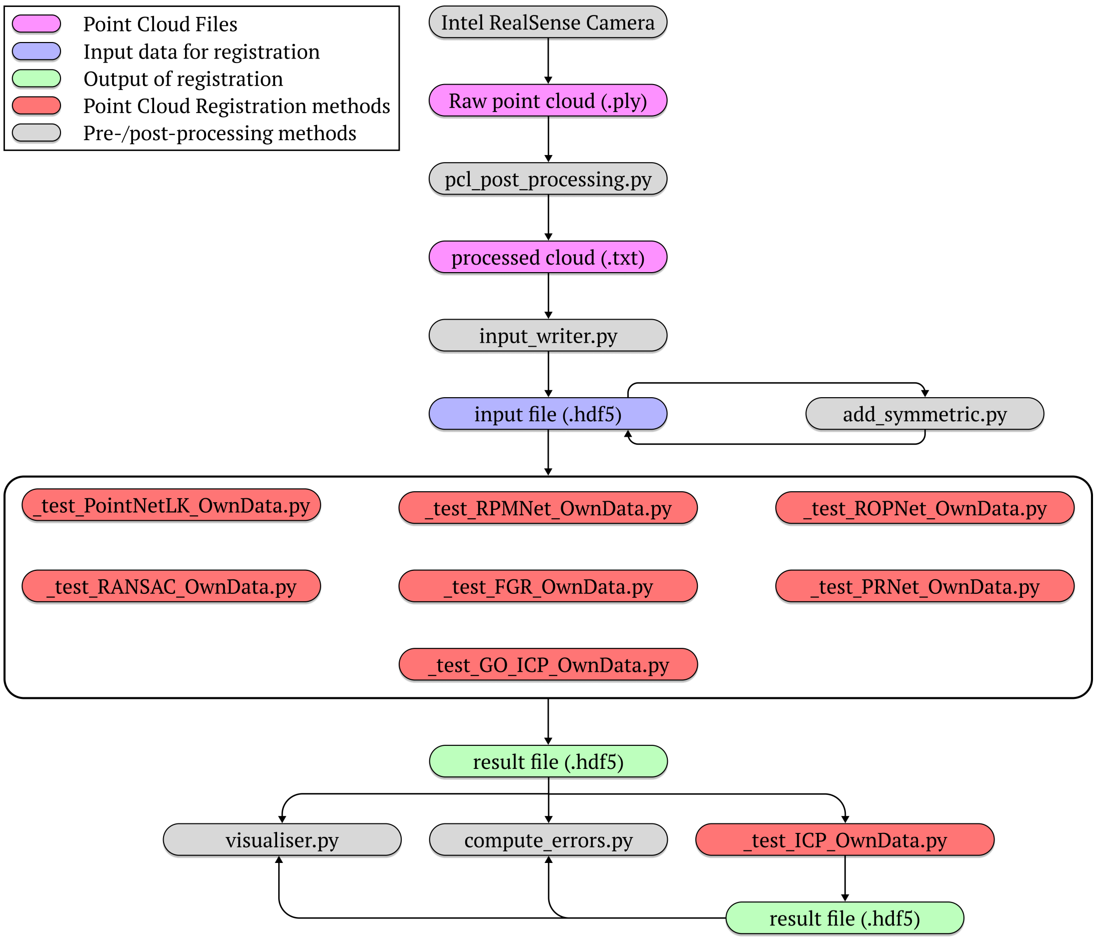


Figure 4.11: Flowchart of the different processing steps during the experiments. The gray and red rectangles represent a Python file. The magenta, blue and green rectangles represent the file types taken as inputs, or produced as outputs of the different Python files.

estimation and known orientation of the object on the table. After, a *.hdf5* file is created to perform the chosen registration method. The results can then be visualised, refined and characterized.

Chapter 5

Training

The learning-based registration methods, PointNetLK, RPMNet, PRNet and ROPNet have to be trained. Section 5.1 discusses the different considered training datasets. The main results of the training process are given in Section 5.2. This chapter contains the conclusions of the training process that are required to understand the results discussed in Chapter 6. A more detailed discussion can be found in Appendix D.

5.1 Training Files Generation

For the creation of the training files, the idea is to use to available CAD models of the considered objects. These models are sampled into a point cloud with a desired number of points. The obtained point cloud is then randomly transformed with a translation vector and rotation matrix. Different variations of data can be created, following the basic guidelines proposed in the literature [54, 2, 49, 63, 48, 38].

As mentioned in Section 4.1 (Table 4.1), the Cranfield benchmark dataset, used during the experiments, only contains six objects. Thus, each object is transformed multiple times to create a sufficiently large dataset. The number of variations per object is set to 820 for the training set and 205 for the testing set. This is chosen in correspondence to the number of objects in each category of the ModelNet40 dataset.

In general, three basic training sets are considered, an example is shown in Figure 5.1.

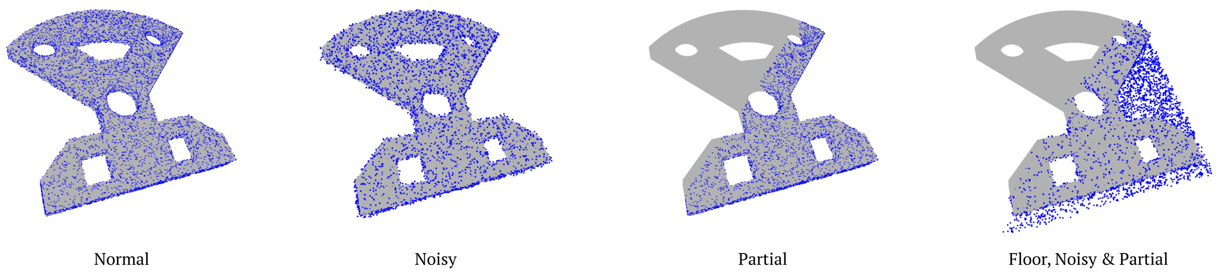


Figure 5.1: The algorithms are trained using different datasets. They can be simply sampled from the CAD models (Normal), or noise can be added (Noisy). Points can also be removed (Partial). Finally, a floor can be added to mimic, for example, the presence of a table (Floor, Noisy & Partial).

- **Normal Data** containing the point clouds sampled from the original CAD models, transformed according to the above mentioned considerations.
- **Noisy Data** consists of the same data with noise added to the source (transformed) point cloud.
- **Partial Data** where 70% of the points are removed from the source (transformed) point cloud. The points are sampled as the ones closest to a random chosen point in space.

Additional, more complex, datasets can also be considered, such as:

- **Noisy** & **Partial Data** created as a combination of noisy and partial data to better mimic the real-world scans.

- **Floor Data** where a floor (plane) is added to the mesh in order to simulate a table or floor on which the object is positioned.
- **Floor, Noisy** & **Partial Data** which is a combination of the before mentioned cases.

The convergence of the latter training sets is not always guaranteed, as more complexity is added to the dataset, as is further discussed in Section 5.2 and Appendix D.

Since the datasets are extensive in size and the training takes multiple hours (sometimes even more than twelve hours), Google Colab is used [16]. It provides the user with a CPU or GPU runtime that can be accessed online, either for free or payed (Colab Pro(+)).

The generation of the training files is coded in *create_train_files.py*. In Section 5.2, the results of training the different learning-based methods are discussed. A more detailed discussion can be found in Appendix D. Each method is first trained and afterwards validated with a new (synthetic) dataset, consisting of 100 new transformations.

5.2 Training Results

Training the learning-based methods requires significant time and computational efforts. On top of this, the convergence of the methods depends on the training parameters and the considered dataset. In the end, the training is validated for PointNetLK on normal and noisy Cranfiel data. For RPMNet and ROPNet, the training is validated on normal, noisy, partial and floor Cranfield data. The training of PRNet did not converge and requires more power than the other three methods. This method is thus no longer considered in the further experiments. The detailed results of the validation are reported in Appendix D.

The datasets have to be generated carefully, to not cause under- or overfitting. It is clear that the training data used is not a perfect representation of the real world data. However, a compromise has to be made between the complexity of the training set, the available resources and the required accuracy. In this thesis, the goal is to verify whether simple models can already lead to satisfying results comparable to, or better than, those of the global registration methods. For this reason, more complex training is left for future work.

5.3 Conclusion

Training is done on various training sets where the original objects are randomly transformed and additional features such as noise or partiality are added.

Training the methods can be difficult due to the significant time and computation power required. Some methods, such as PointNetLK have more trouble converging on the training set, while RPMNet and ROPNet can be trained on more datasets. PRNet does not stabilize during training and is therefore no longer considered.

Chapter 6

Experimental Results

Using the setup mentioned in Chapter 4 and the training models from Chapter 5, the obtained point clouds are run through the registration methods selected in Section 2.3. The results are used to compute an array of metrics, given in Chapter 3. Firstly, in Section 6.1, the main variables considered during the experiments are discussed. In Section 6.2, the results are summarised for the global registration methods, including FGR, RANSAC and GO-ICP. After, the results for the learning-based methods, PointNetLK, ROPNet and RPMNet are given in Section 6.3.

The goal of this chapter is to obtain insight into the effect of the different parameters on the registration results. The specific evolutions are discussed case by case. An overview of the main conclusions and an answer to the research questions posed in Chapter 1, can directly be found in Chapter 7. In Chapter 7, a flowchart is also presented, based on the main learning outcomes, to decide on a registration method. All results and data from the experiments are also available on the Github page.

6.1 Variables

Each considered registration method comes with its own set of parameters, that can be tuned to achieve a better or worse result. Other parameters are related to the data itself. The parameters that are considered and varied, are summarised in Table 6.1.

Table 6.1: Parameters considered during the experiments with their respective categories and range of considered (or feasible) values.

Parameter	**Category**	**Values**
Voxel size	Data Processing	0.001-0.1
Bounding Box	Data Processing	1.0-1.8
Zero-mean	Data Processing	True/False
MSE Threshold	GO-ICP	0.00001-0.1
Trim Factor	GO-ICP	0.0001-0.1
Refinement	PCR Methods	True/False
Training Model	Learning Methods	Normal, Noisy, Partial, Combination

6.1.1 Voxel Size

The voxel size (VS) is typically used to downsample the data, resulting in less points and an averaged version of the scan. It works by selecting a cube with a size defined by the voxel size and taking the mean of all the points inside this cube. The resulting points define the new point cloud that can be used for registration. An example is shown in Figure 6.1.

6.1.2 Bounding Box

As already mentioned in Chapter 4, the scans are preprocessed in such a way to allow the user to choose a bounding box around the object. The bigger the bounding box, the more information from

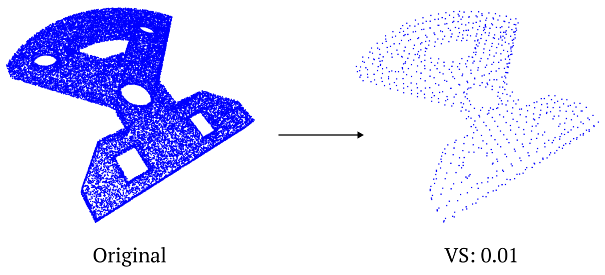


Figure 6.1: Example of applying a voxel size equal to 0.01 to the *Base-Top_Plate* object.

the environment (such as the table) is included in the point cloud. The bounding box is expressed as a number that is multiplied with the corners of the basic extracted box. A factor of 1.5 means the size of the box is increased by a factor of 50%. An example of the effect of changing the bounding box is shown in Figure 6.2.

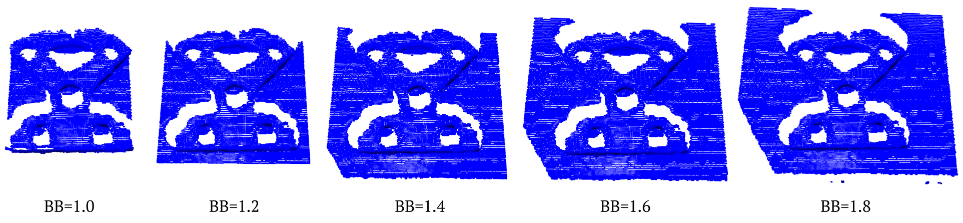


Figure 6.2: Evolution of increasing the bounding box for the *Base-Top_Plate* object. The bounding box is increased relatively, starting from the initial case for a unit box ($BB = 1.0$). The unit box is the smallest considered bounding box. Afterwards it is increased by 20% for a bouning box of 1.2.

6.1.3 Zero-mean

The registration process aims to determine a transformation between the captured source point cloud and the template. However, it is also possible to already move the captured point cloud to the origin, by removing its mean vector. This results in a smaller transformation to be found by the registration method. However, it is hypothesized that this may lead to some algorithms getting stuck inside local minima solutions.

6.1.4 MSE Threshold & Trim Factor

The Mean-Square-Error (MSE) Threshold and Trim Factor are two parameters used in the GO-ICP method. The first variable defines the error threshold which needs to be achieved in order for the algorithm to converge. The second parameter (trim factor) removes a percentage of the point cloud to counteract outliers. Notice that the Trim Factor does not work in the same way as the voxel size defined before.

6.1.5 ICP-Refinement

To improve performance of the registration methods, ICP-refinement is applied. The results are compared when the refinement is either applied (True) or not (False). Important to note is that the ICP algorithm itself is also a registration method. This means that the previously discussed parameters, such as the voxel size, zero-mean and bounding box, can also be varied here.

6.1.6 Training Models

Finally, for the learning-based methods, the training model is also varied according to the ones obtained in Chapter 5.

6.2 Global Registration Methods

In this section the results for the global registration methods are compared and discussed. The section is divided into subsections, reporting the results related to the before-mentioned parameters (Section 6.1). The algorithms are run on the created scans mentioned in Chapter 4 and the metrics from Chapter 3 are computed. In order to assess whether the algorithms yield the same results every time and to obtain a good measure of the registration time, they are applied multiple times on the same scans. For GO-ICP, the algorithm is applied two times on each scan, due to long computation times and the method always yielding the same results. For RANSAC, FGR and ICP, the algorithms are applied 20 times on each scan.

6.2.1 Zero Mean

The results show that adapting the point cloud, centering its mean on the origin, usually has a small influence on the metrics. Depending on the object, the errors are either very similar or a bit higher. The complete list of metrics can be found in Appendix E, Table E.1. The results are taken as an average over the different objects. Table E.1 shows that GO-ICP performs better when the point cloud is not centered on the origin. For RANSAC and FGR, centering yields better results. For the remaining experiments, the best cases are used. Thus a non-zero mean for GO-ICP and a zero-mean for RANSAC and FGR.

6.2.2 MSE Threshold & Trim Factor

The MSE threshold and Trim Factor are specific for GO-ICP. In Figure 6.3, the registration time for GO-ICP is plotted as a function of the MSE threshold.

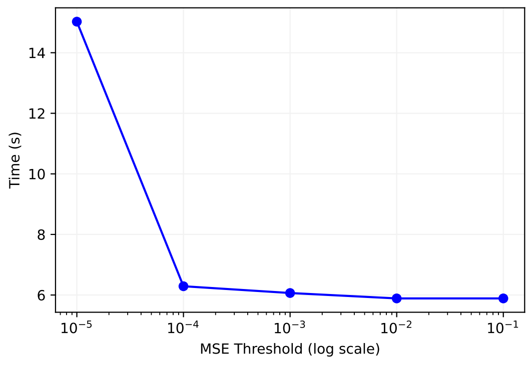


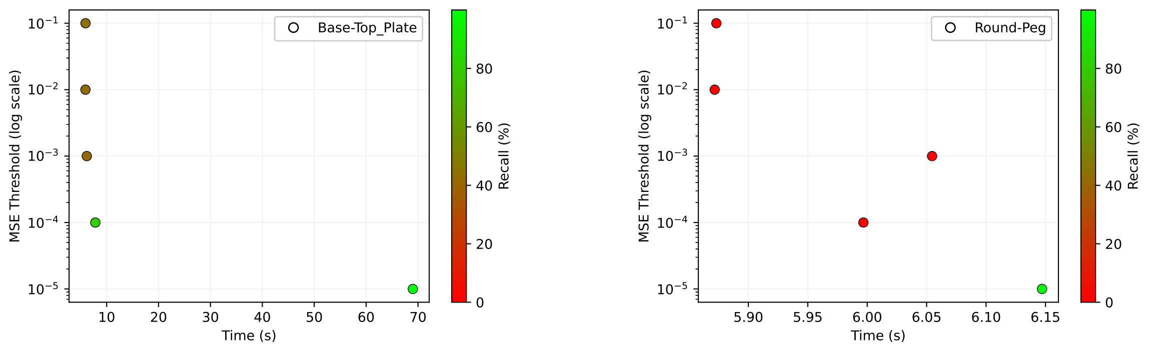
Figure 6.3: Evolution of the registration time for GO-ICP, taken as the mean over all objects, as a function of the MSE threshold. The lower the threshold, the higher the number of iterations and thus computation time.

From Figure 6.3 it can be seen that the computation time decreases with an increasing threshold. This is as expected, as increasing the threshold means that the algorithm needs to perform less iterations before converging to a result. In the end, the timing seems to converge to a value of around 6s. This is already relatively long, especially for real-time applications.

Increasing the threshold comes at the cost of a lower performance for the algorithm. In Figure 6.4 the threshold is plotted as a function of the computation time. Each point is colored corresponding to the recall metric. As the threshold is increased, the recall metric lowers. For some objects such as the *Round-Peg* (Figure 6.4b), this happens quite drastically. The recall metric reduces from 100% for a threshold of 0.00001 to a recall of 0% for a threshold of 0.0001. Reducing the threshold immediately leads to the object failing for all scan cases considered.

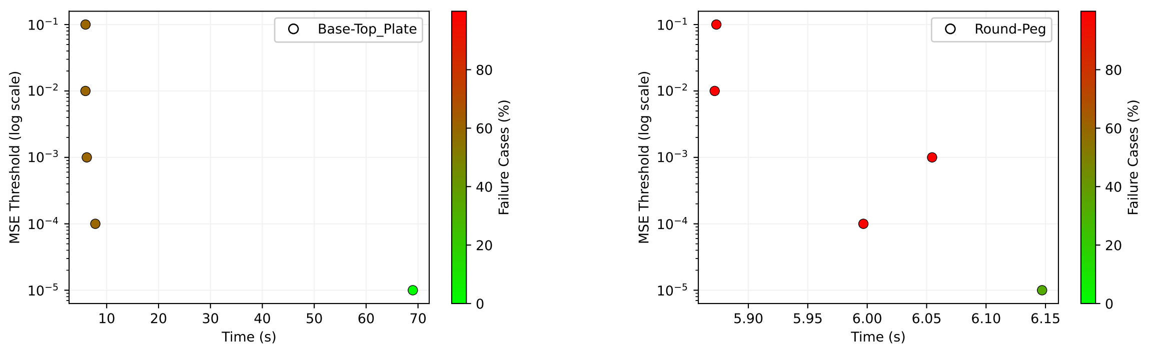
The failure cases are also indicated in Figure 6.5. The colors now represent the percentage of failure cases over the scans instead of the recall metric.

Figures 6.4-6.5 make it clear that GO-ICP yields the best results when used at a low threshold, due to the strict convergence criteria. As indicated in Figure 6.5, the number of failure cases falls to zero in most cases, which means GO-ICP always leads to a satisfactory result. The algorithm does not fall into



(a) Base-Top Plate (b) Round-Peg

Figure 6.4: Recall for GO-ICP, two different objects, as a function of the time and voxel size. The higher the recall, the better the the registration results fits unto the template. The different objects each yield the highest values of recall for a low MSE threshold. Increasing the threshold leads to a sudden decrease of the recall. This means the result does no longer match the template in an accurate way.



(a) Base-Top Plate (b) Round-Peg

Figure 6.5: Failure cases for GO-ICP, for two objects, as a function of the time and voxel size. As the threshold is lowered, more scans lead to unsatisfactory results ($R^2 < 0$, $MRAE > 120°$), increasing the number of failure cases. Similar to the recall metric, the best performance is obtained for the lowest MSE threshold. However, this comes at the cost of the highest computation times.

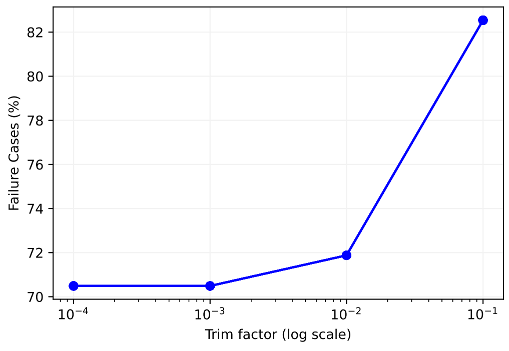
local minima solutions and is able to find the global solution, in contrary to ICP for example. However, this comes at the clear cost of a longer computation time.

Aside from the MSE threshold, GO-ICP also uses a trim factor to remove outliers. From Figure 6.6, it can be concluded that increasing the trim factor serves no useful purposes. This only leads to an increase in the number of failure cases and even a little increase in the required registration time. As the scans used during the experiments are already cleaned up from outliers, it is not surprising for the trim factor to not yield any added benefits.

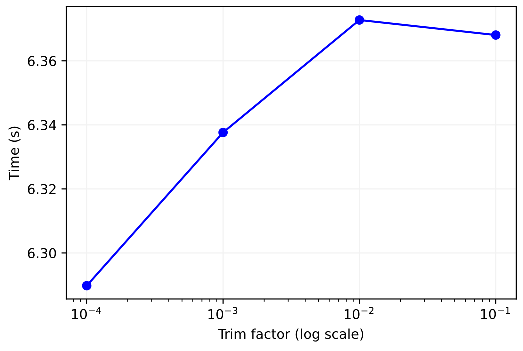
6.2.3 Voxel Size

The voxel size is adapted for RANSAC, FGR, ICP and GO-ICP. Firstly, the effect on the computation time is considered in Figure 6.7. It is again important to note that the time measurements are largely dependent on the device specifications and to some extent on the other programs that are run in the background.

From Figure 6.7, it is clear that the increase in voxel size reduces the required timing for RANSAC and FGR. This is not surprising, as there are less points to consider during the registration process, thus lowering the number of operations and time. Between RANSAC and FGR, it seems FGR is usually the fastest method. For very small voxel sizes, both methods converge to each other.



(a)



(b)

Figure 6.6: Effect of the trim factor for GO-ICP on the mean number of failure cases (a) and registration time (b). Increasing the trim factor leads to a higher number of scans yielding unsatisfying results and thus more failure cases (a). The computation time (b) increases slightly, due to the additional computations of removing the outlier points.

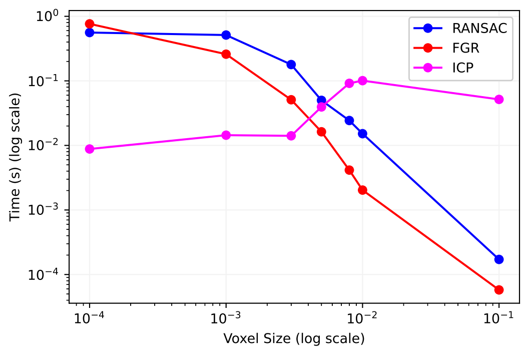


Figure 6.7: Comparison of the average computation time over different objects for ICP, RANSAC and FGR as a function of the voxel size. For both RANSAC and FGR, the computation time decreases with an increasing voxel size. ICP shows an opposite trend as the algorithm is unable to find a solution for low voxel sizes and only finds a solution for higher voxel sizes.

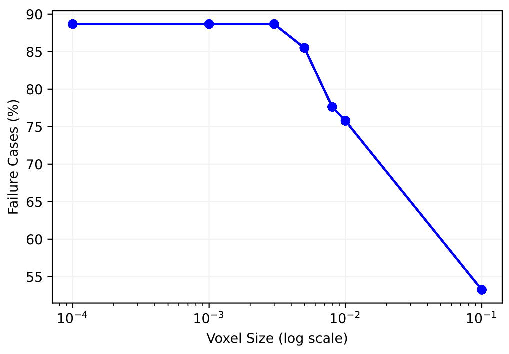
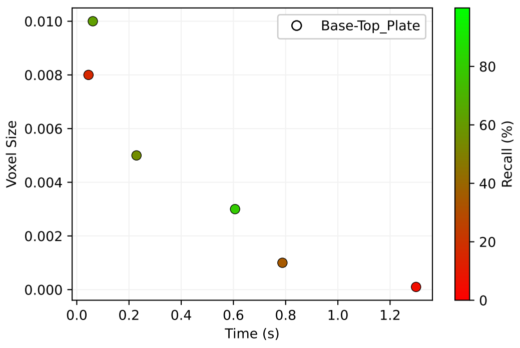


Figure 6.8: Evolution of the mean failure cases over all objects for ICP, as a function of the voxel size. For higher voxel sizes, ICP is able to find a better result for more scans, reducing the number of failure cases.

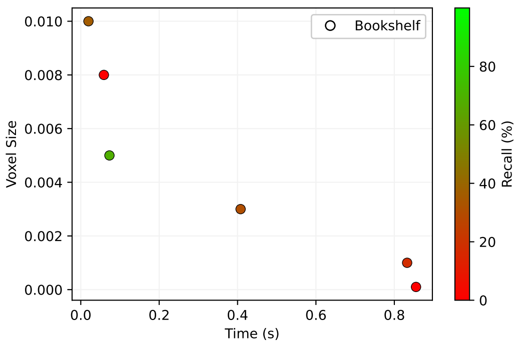
The standard ICP method sees an increase in the computation time for an increasing voxel size. This is counter-intuitive, since there are less points to match and thus a decrease in timing would be expected. However, looking at the number of failure cases (Figure 6.8), it is clear that for a large range of voxel sizes, ICP does not yield satisfactory results. In these cases, the algorithm does not really do anything, as the result is usually coinciding with the identity matrix. This means the method runs a lower number of iterations, resulting in a lower computation time. For larger voxel sizes the method begins to return better transformation results, which require more computation time and more iterations. The threshold for convergence also changes with the voxel size. It increases as the voxel size increases. Thus for a higher voxel size, the threshold is less strict and ICP is able to find a converged result. It is then also easier for ICP to identify pairs of points and escape local minima. Hence the increase in performance. In the remainder of the experiments, ICP is used solely as a refinement technique.

All registration times in Figure 6.7 appear to be below $1s$, which means these algorithms are relatively quick, especially in comparison to GO-ICP (Section 6.2.2). It is also clear that changing the voxel size can lead to drastic changes in the computation time. Selecting a larger voxel size would be preferential, especially for real-time applications.

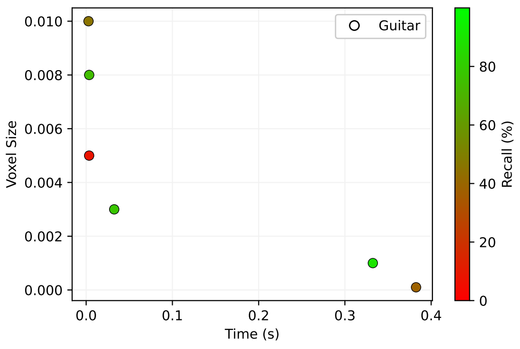
It is of course important to verify how the accuracy of the results changes by adapting a larger voxel size. To this end, the voxel size and registration times are plotted for each object, together with the achieved recall value. Figure 6.9 shows the results for RANSAC for four objects.



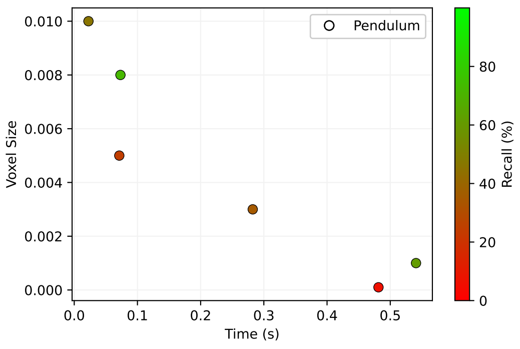
(a) Base-Top Plate



(b) Bookshelf



(c) Guitar



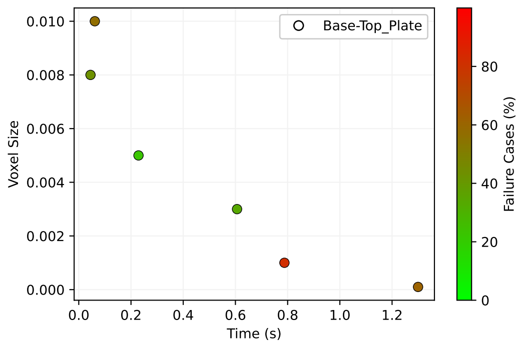
(d) Pendulum

Figure 6.9: Recall for RANSAC, for different objects, as a function of the time and voxel size. The higher the recall (green), the better the registration results fit unto the templates. The different objects each yield the highest values of recall for a certain voxel size. An optimal voxel size can be identified for each object, which might differ between the different objects. For example, the optimal voxel size in terms of recall, is 0.003 for the *Base-Top_Plate* (a), but 0.008 for the *Pendulum* (d).

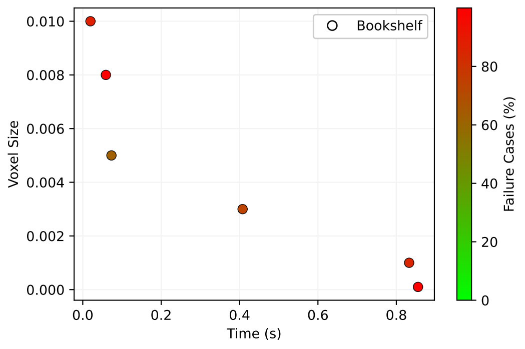
Figure 6.9 shows the same trend of registration time and voxel size for each object. Larger objects, such as the *Base-Top_Plate* (Figure 6.9a), require longer registration times, as they contain a larger number of points. Smaller objects, like the *Guitar* object (Figure 6.9c), require less registration time. Looking at the recall values, most objects have one (sometimes more), optimal voxel size(s), at least in terms of the recall percentage. The *Base-Top_Plate* object (Figure 6.9a), for example, has an optimal voxel size of around 0.003. This value, is not the same for each object, as the objects are considerably different in shape and size. For the same voxel size, smaller objects will have less points than larger objects. However, the shape of the object and the chosen algorithm also play an important role in selecting the voxel size. The optimal voxel size is also dependent on the metric used. Figure 6.10 shows the same plot as in Figure 6.9, however, now the number of failure cases is used to color the different points.

The optimal voxel size in Figure 6.10 has now changed for some objects. For example, for the *Base-Top_Plate* object (Figure 6.10a), the optimal is now around 0.005. To illustrate this more clearly, the optimal voxel sizes in terms of the recall metric and the number of failure cases are shown in Figure 6.11.

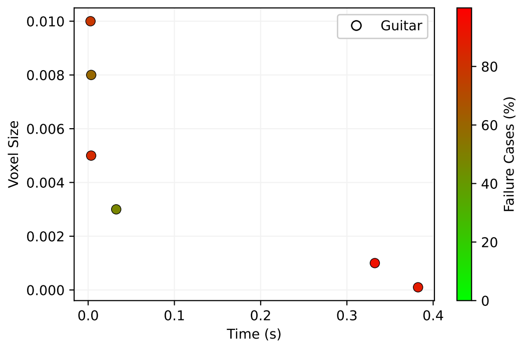
For some objects, the optimal voxel sizes in Figure 6.11 are not too different between the two metrics. For others, such as the *Range-Hood* object, the difference is more noticeable. Another interesting comment is the fact that the optimal voxel sizes, for FGR, are less critical for the smaller objects. The same number of failure cases are found for different voxel sizes, for the same object, hence the presence of multiple markers. When looking at the optimal values in terms of the number of failure cases, the difference in smaller (right) and larger (left) objects is also more clear. However, the conclusion is different between



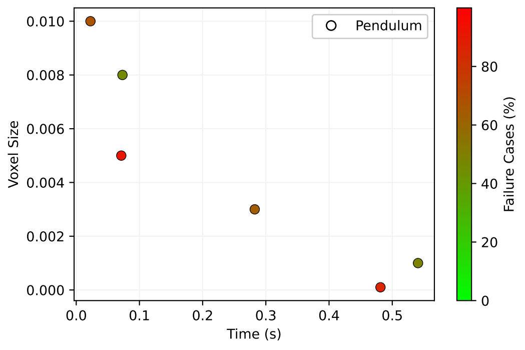
(a) Base-Top Plate



(b) Bookshelf



(c) Guitar



(d) Pendulum

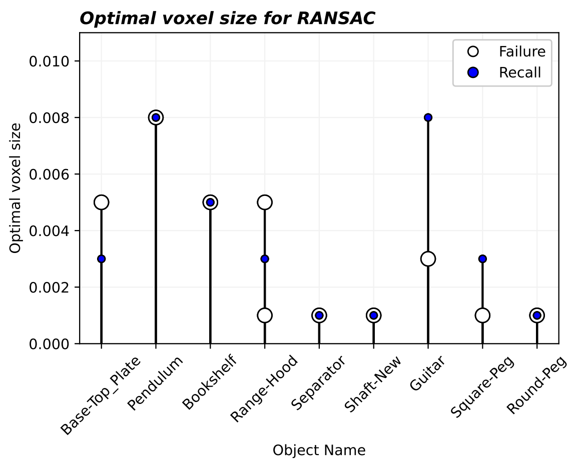
Figure 6.10: Failure cases, for RANSAC, for different objects, as a function of the time and voxel size. The lower the number of failure cases (green), the more scans yield a result for which $R^2 > 0$ and $MRAE < 120°$. An optimal voxel size can be identified for each object, which might differ between the different objects. For example, the optimal voxel size in terms of the number of failure cases, is 0.005 for the *Base-Top_Plate* (a), but 0.008 for the *Pendulum* (d).

RANSAC and FGR. For RANSAC, bigger objects seem to result in less failure cases for somewhat bigger voxel sizes. While for FGR, the opposite seems to be true.

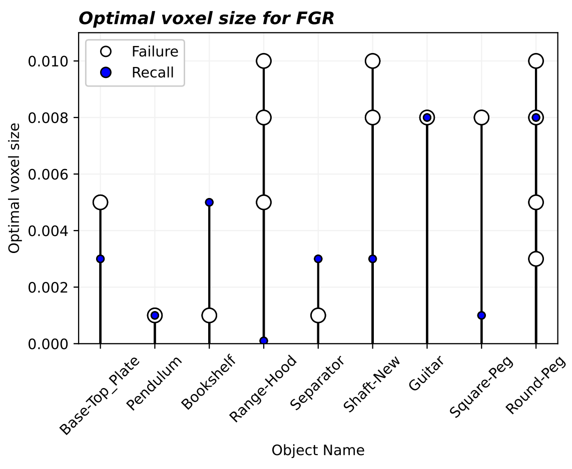
A relevant question would be to wonder which metric should be chosen to select the correct voxel size. The answer depends on the requirements. When the goal is to use the algorithms independently, to quickly get an accurate result, it would be better to consider the recall metric. However, as mentioned before in Chapter 3, the recall result is computed over the number of successful registration cases only. Figures 6.10 and 6.11 have shown that a high value for the recall metric, does not necessarily correspond to a low number of failure cases. This only means that, for the successful registration attempts, the result coincides well with the template. When the goal is to make sure that the registration process succeeds over a wider array of point cloud scans, it is better to consider the number of failure cases as the metric to select the voxel size. Even though the results might not be the most accurate initially, refinement can be applied to improve the recall metric in these cases. Refinement, as is discussed in Section 6.2.5, improves the metrics and can sometimes lead to an even lower number of failure cases.

The results for GO-ICP are discussed separately due to the different working of the algorithm. GO-ICP uses the MSE threshold to determine when the result has converged. However, when changing the voxel size, it is possible that the method cannot converge anymore. This is because the increase in voxel size reduces the density of the point cloud. As a result, the distances between the template points and the source (scanned point cloud) points might become larger than the threshold, even in the case of a perfect alignment. For this reason, the algorithm never converges. Thus, it is important, when changing the voxel size, to also adapt the MSE threshold accordingly. Figure 6.12 compares the results for recall, when the voxel size is adapted (Figure 6.12a) and when the whole, original point cloud is used (Figure 6.12b).

Figure 6.12 shows that adapting the voxel size can lead to an improvement in results, when using a

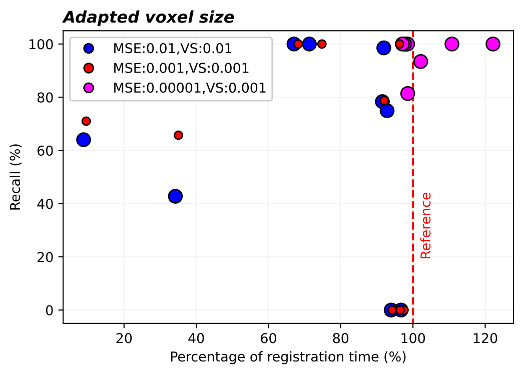


(a) RANSAC

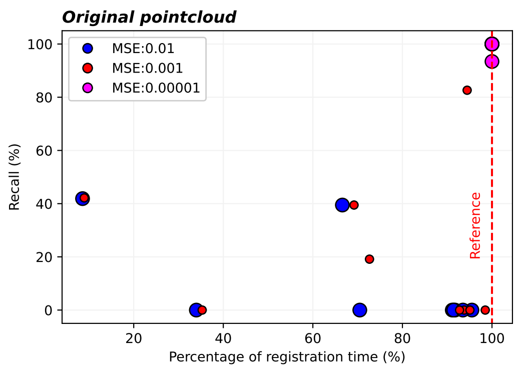


(b) FGR

Figure 6.11: Optimal voxel sizes in terms of recall (●) and number of failures (○) for all considered objects. The objects are ordered from biggest to smallest from left to right. The optimal voxel size does not always align between the two metrics. Some objects (*Range-Hood*, *Round-Peg*) have multiple voxel sizes, yielding the same recall results or number of failure cases. The size of the object correlates to some degree to the optimal voxel size. This is most clear when considering the number of failure cases (○) for RANSAC. The optimal voxel sizes are bigger for the bigger objects and smaller for the smaller objects.



(a) Voxel size adapted



(b) Original point cloud

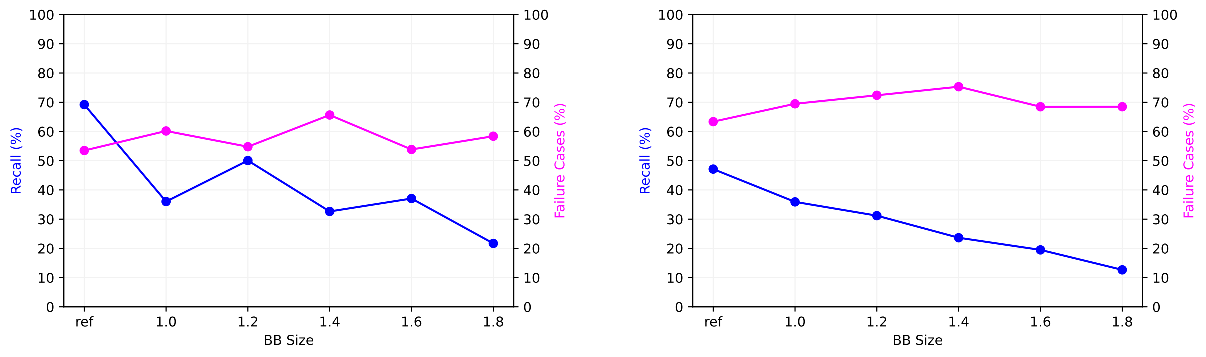
Figure 6.12: By adapting the voxel size together with the MSE threshold, the GO-ICP algorithm achieves better results in terms of recall. The recall for a MSE threshold of 0.01 (●) only achieves values of around 40% for the original point cloud (b). When the voxel size is adapted to 0.01, the recall is increased to higher values (a), except for the smaller objects, which always fail, resulting in a zero recall value. Additionally, the computation time for lower values of the threshold (●, ●), is always lower than the reference for a threshold of 0.00001 (····), without adapting the voxel size (●, b).

larger MSE threshold. As shown in Figure 6.4, increasing the MSE threshold, degrades the recall metric significantly. This is also shown in Figure 6.12b, where the original point clouds are used. The timing is expressed as a percentage of the time required to do registration for the lowest MSE threshold of 0.00001. For this value, the registration time is on average the longest (Figure 6.3). On Figure 6.12a, the voxel size is adapted, for each MSE threshold, while allowing the algorithm to still converge. For the thresholds 0.01 (●) and 0.001 (●), the recall values have improved to higher values, for almost all objects, except the smallest ones (*Square-Peg*, *Round-Peg*). The additional benefit of using a lower threshold, is the reduced registration time, compared to the reference of 0.00001 (····).

6.2.4 Bounding Box

Changing the bounding box means introducing more points to the source point cloud. For GO-ICP, this leads to very long registration times, without convergence. The presence of a high number of points, that are not appearing inside of the template point cloud, lead to the algorithm not converging anymore. Thus, for GO-ICP it is recommended to make sure the scanned point cloud is post processed to contain as little outlier points as possible, either using a method as in Chapter 4 or by using the trim factor.

For RANSAC and FGR, the tests are redone for five different bounding boxes, increasing each time with 20% of the original box. The scans considered are from the *Base-Top_Plate*, *Pendulum*, *Separator* and *Range-Hood* objects. The algorithms are run for one voxel size per object, chosen based on the results shown in Section 6.2.3, where a good performance in terms of failure cases and recall is obtained. Figure 6.13 shows the evolution of recall and the number of failure cases, for the different bounding boxes and the reference case, where the point cloud is cleaned up from any outlier points.



(a) RANSAC (b) FGR

Figure 6.13: Effect of changing the bounding box on the recall (-●-) and the number of failure cases (-●-). The effect is most clear on the recall metric, which decreases as the bounding box increases. The evolution is more constant for FGR (b), while RANSAC (a) is able to reach higher values of recall for a bounding box of 1.8. However, for both methods the recall drops significantly from the reference value.

Both RANSAC and FGR see a drop in performance, in terms of the recall metric, when the bounding box is increased, as shown in Figure 6.13. For FGR, the evolution is more steady, while being less constant for RANSAC. The number of failure cases is less affected by the change in bounding box. This is because the R^2 value, used for determining whether the scan is considered to be failed or not, is not always a good indicator for the performance when dealing with a lot of outlier points. A more in depth discussion on the R^2 metric and the relative errors is given in Appendix E. Additionally, when looking at the computation time, it also increases on average, when increasing the bounding box, as shown in Figure 6.14.

6.2.5 Refinement

Finally, ICP refinement is applied, for three different voxel sizes, on the results obtained for RANSAC and FGR. Figures 6.15-6.16, show the positive effects of applying ICP refinement, leading to precise results of a couple of degrees and millimeters in the best cases for almost all objects. For most scans, the optimal voxel size when applying refinement is 0.1. The refinement step has less danger of falling into local minima and usually requires only a small correction. Thus, it is sufficient to use a limited set of points or a large voxel size.

From Figures 6.15-6.16, RANSAC seems to outperform FGR most of the time. The smaller objects, specifically the *Round-Peg* show higher rotational errors. However, as mentioned before, this object has an infinite number of ground truth solutions. For this reason, the rotational error is not a good metric for this object. It can be noted that, even considering only four possible solutions, the errors are still limited to 25° in the worst case.

Finally, the refinement can also reduce the number of failure cases. This means, more scans will lead to a satisfactory result. However, one must be careful to interpret this result. It is possible that the found

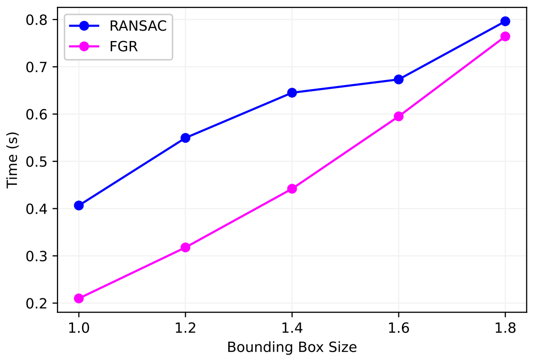
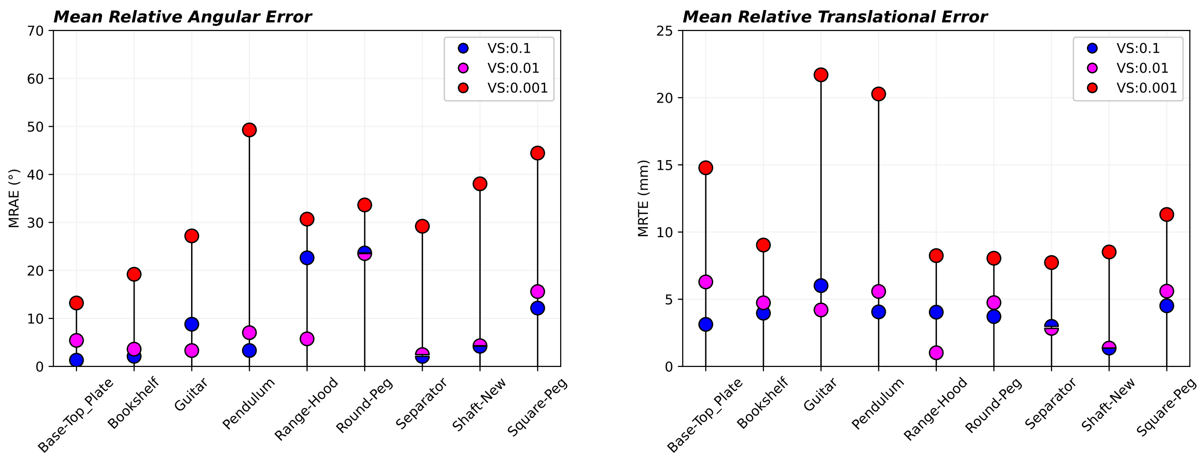


Figure 6.14: Evolution of the registration time for RANSAC (-●-) and FGR (-●-), taken as the mean over all objects, as a function of the bounding box size. As expected, the mean registration time increases over time, as more points are present in the point cloud to process. FGR still yields the lowest timings, compared to RANSAC.



(a) Mean Relative Angular Error (MRAE) (b) Mean Relative Translational Error (MRTE)

Figure 6.15: Applying ICP refinement, after RANSAC, improves both the rotational (a) and translational (b) errors for all objects. For most objects, the highest voxel size (●) yields the lowest error values. The *Guitar* and *Range-Hood* objects reach slightly better errors for a voxel size of 0.01 (●). For some voxel sizes, the objects fail for all scans (*Shaft-New*, *Round-Peg*), hence the lower number of datapoints.

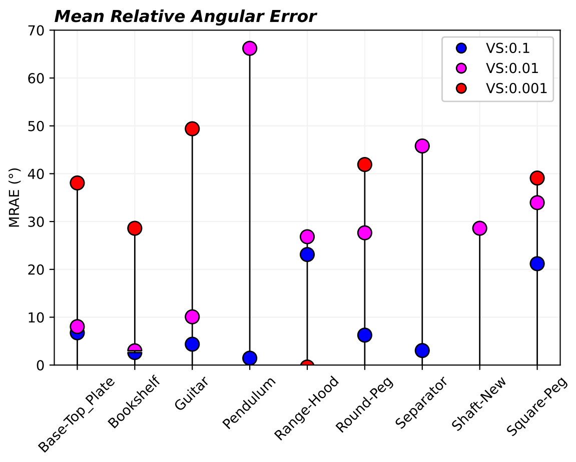
transformation is completely the result of ICP itself. Thus, in this case, ICP has performed more than simply a refinement step.

For GO-ICP, the effect of refinement depends on the used MSE threshold, as is shown in Figure 6.17, where the relative gain is plotted of applying refinement for each threshold. As the MSE threshold increases, the results of GO-ICP worsen, as already shown in Section 6.2.2. Thus, the additional gain of applying refinement increases, as the errors reduce by a factor of around 90% for high thresholds. For a low threshold, the results of GO-ICP are already very precise, up to a couple of degrees and millimeters. Here, the gain of applying refinement is almost zero.

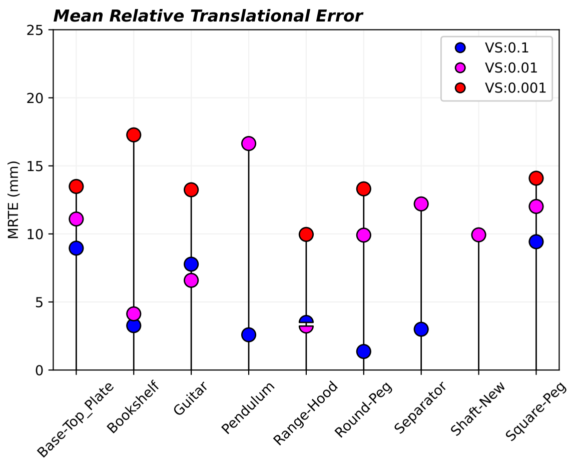
6.2.6 Failure Cases

From the experiments, it is found that different scans may lead to a failure for some parameter values, while not for others. However, for the different objects, there are typically some scans that more often lead to a positive result and others which almost always lead to a failure. By studying these scans, some conclusions can be drawn.

Essentially, three elements are important when applying registration. Firstly, the quality of the scan



(a) Mean Relative Angular Error (MRTE)



(b) Mean Relative Translational Error (MRTE)

Figure 6.16: Applying ICP refinement, after FGR, improves both the rotational (a) and translational (b) errors for all objects. For most objects, the highest voxel size (●) yields the lowest error values. The *Range-Hood* object reaches better errors for a voxel size of 0.001 (●). For some voxel sizes, the objects fail for all scans (*Shaft-New*, *Round-Peg*), hence the lower number of datapoints.

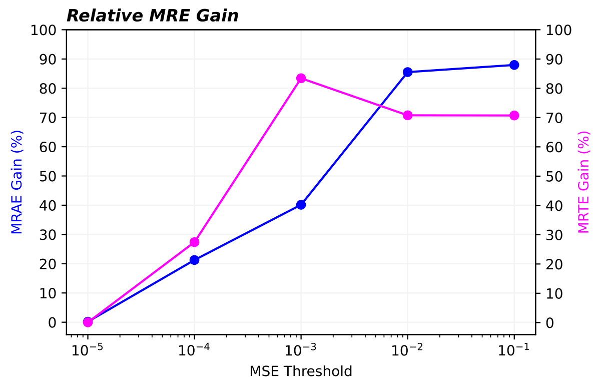


Figure 6.17: Relative reduction of the mean relative angular (-●-) and translational (-●-) errors. The gain of applying refinement is limited for small values of the MSE threshold. However, for larger MSE threshold, where the results of GO-ICP decline, the gain of applying refinement increases significantly.

plays a role in the obtained result. This includes the presence of outliers, environmental clutter and the accuracy of the shape of the object. A clear example of this are the *Shaft-New*, *Guitar* or *Bookshelf* objects, also shown in Figure 6.18. For these object, the scans are of a lower quality than other objects, such as the *Base-Top_Plate*. This is related mainly to their size and the accuracy of the used camera. The *Bookshelf* object, for example, has small parts with a thickness lower than $1mm$. These are no longer detected by the camera. The better the quality of the scan, the better the features are identified and thus also the better the result.

Aside from the quality of the scan, the initial pose of the template and source also play a role. When the template and the source are already somewhat aligned before the registration, the chances for the method to fall into a local minimum are lower. An example for the *Base-Top_Plate* is shown in Figure 6.19. Related to this, the size of the transformation to be estimated is also important. When the method has to find a difficult transformation, over multiple axes, with large angles, the results are worse or there might not be a successful result at all. When the transformation consists of only a smaller rotation around a single axis, the methods are usually able to identify the solution better.

Finally, which scans fail and which succeed, also depends on the parameters, bounding box and method used. Guidelines for creating the point cloud scans are further given in Chapter 7.

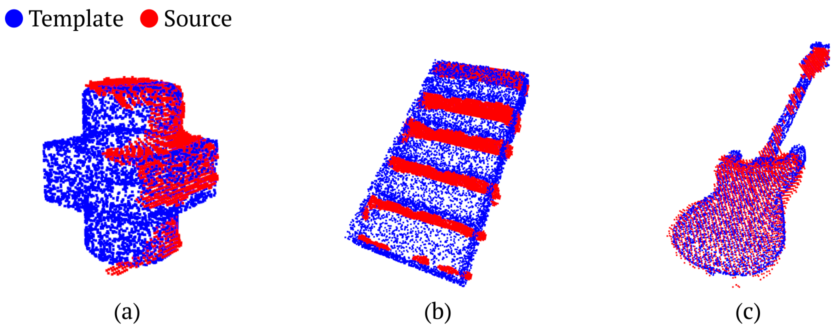


Figure 6.18: Example of lower quality scans, usually leading to failures, for (a) *Shaft-New*, (b) *Bookshelf* and (c) *Guitar*.

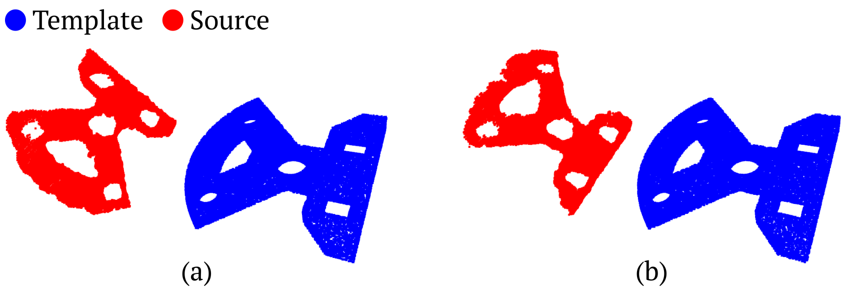


Figure 6.19: Example of different initial alignments for different scans of the *Base-Top_Plate* object. The initial pose of (a) leads to failure more often than (b), since the latter is already better aligned with the template initially.

6.2.7 Conclusions

Different parameters play an important role in the registration result for the global registration methods. For GO-ICP, the MSE threshold is the key variable. The lower the threshold, the better the results, however, also the longer the registration takes. The trim factor does not result in major changes. However, it can be used to remove outliers, since GO-ICP does not converge for larger bounding boxes. An adapted voxel size can be used to improve the results for lower thresholds. Refinement only leads to an added gain for higher thresholds.

RANSAC and FGR are mainly influenced by the voxel size. Depending on the metric, an optimal voxel size can be defined. However, there is no clear trend to the dependency of the voxel size. Adapting the bounding box usually leads to worse results. ICP refinement has a positive influence on the results, especially for smaller voxel sizes.

6.3 Learning-based Methods

The learning-based methods are studied in a similar way to the global methods in Section 6.2. Again, the algorithms are applied multiple times on the same scans. Since the results are always the same, the number of iterations is limited to five, to get an accurate reading of the timing results.

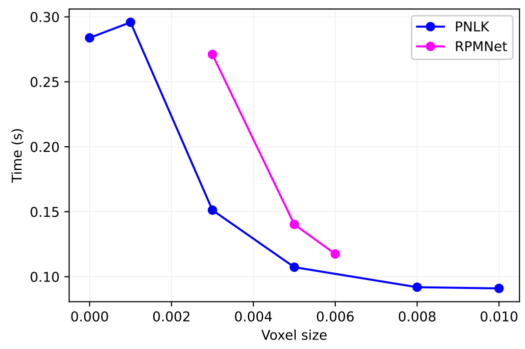
6.3.1 Zero Mean

Similarly to the global methods, the zero mean has a limited effect on the results. The difference is illustrated in Table E.1 for RPMNet and ROPNet. PointNetLK centers the point cloud inside the algorithm, hence why it is not included inside of the table. The best results are further used. This means a non-zero mean for RPMNet and ROPNet. Due to ROPNet failing for most objects, the results in Table E.1 are only shown for a limited selection of objects that do not fail.

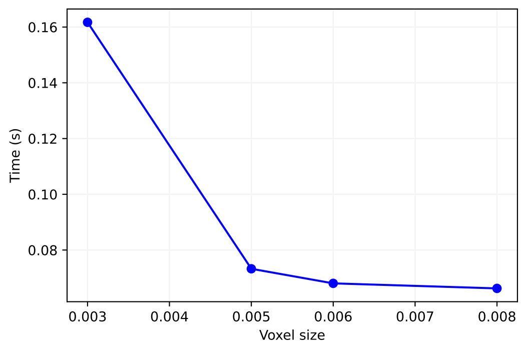
6.3.2 Voxel Size

As was the case for RANSAC and FGR, the registration time decreases for PointNetLK, RPMNet and ROPNet, with an increasing voxel size, for a similar reason; a reduction of the number of points. This is shown in Figure 6.20. PointNetLK and RPMNet also show a significant preparation time of 11s for the data, when the results are run on a personal computer[1]. This is not the case when the registration is run on Google Colab.

The datapoints for RPMNet and ROPNet are also more limited than those of PointNetLK. Both RPMNet and ROPNet require more intensive GPU power. The small voxel sizes can therefore only be run on Google Colab. To keep the time measurements consistent, the curves start at a higher voxel size. Both algorithms also require a minimal number of datapoints to work. When the voxel size is too large, the smaller objects will have too little points, and thus the registration cannot be performed. Thus the datapoints are limited to a voxel size of 0.006 for RPMNet. ROPNet shows some similar problems, however, a 0.003 voxel size is the only one where all the objects work. To still illustrate the effect of the voxel size on the timing, only a limited number of objects is considered in Figure 6.20b.



(a) PointNetLK, RPMNet



(b) ROPNet

Figure 6.20: Average registration time, as a function of the voxel size, for PointNetLK and RPMNet (a) and ROPNet (b). Increasing the voxel size, decreases the registration time for all methods. RPMNet and ROPNet require a minimal number of points in order to work. For this reason, the smaller objects can no longer be processed for larger voxel sizes. Bigger objects, require more memory to be processed at small voxel sizes. The timings for ROPNet are limited to only four, big objects, due to before mentioned limits.

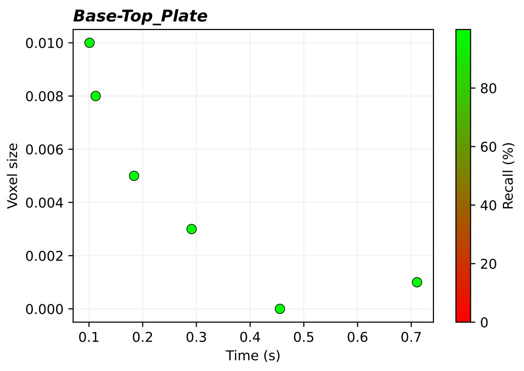
The effect of the voxel size on the metrics is shown in Figures 6.21-6.22, for PointNetLK. The effect is limited for most objects, except the *Guitar* object. This could be explained by the fact that the training model used for these results is the one obtained by training PointNetLK on the normal Cranfield dataset (Chapter 5). This dataset does not include the *Guitar* object. When looking at the failure cases in Figure 6.22, the same conclusions can be drawn. Important to note is the fact that PointNetLK, as well as RPMNet and ROPNet, always yield the same results for a specific scan. This is not the case for RANSAC and FGR. Thus, the scan either always fails or succeeds.

For RPMNet, the evolution of the recall metric and the number of failure cases is given in Figure 6.23. Here, the voxel size has a more noticable effect on the results. Similarly to RANSAC and FGR, an optimal voxel size could be determined. However, the evolution is again not very defined. Notice also the link between the number of failure cases and the recall. As more scans succeed, the average is taken over more cases, leading to a change in the recall metric.

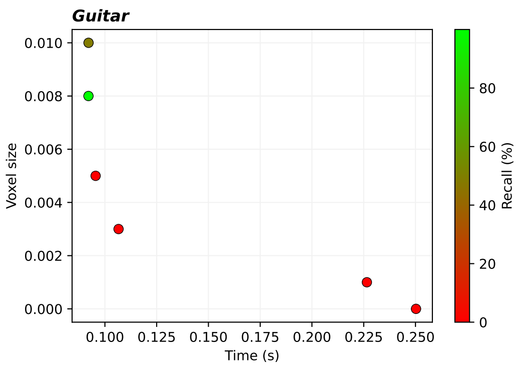
6.3.3 Training Models

The learning-based methods can be trained using different datasets, as discussed in Chapter 5. For PointNetLK, the method is trained on normal and noisy Cranfield data. The available pretrained ModelNet40 data is also used during the tests. RPMNet is trained on normal, noisy, partial and floor Cranfield data.

[1]The personal computer used has the following specifications: Windows laptop, NVIDIA GeForce RTX 3070 GPU and AMD Ryzen 7 processor

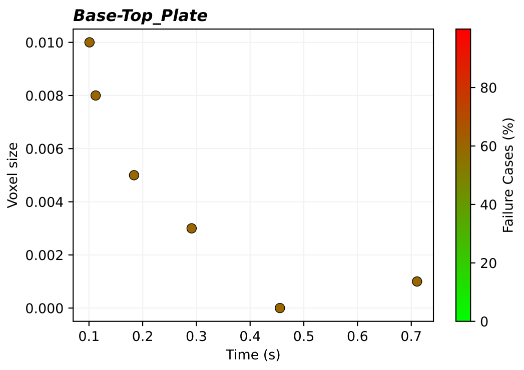


(a) *Base-Top_Plate*

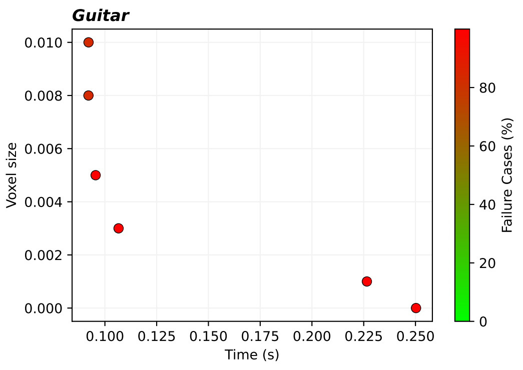


(b) *Guitar*

Figure 6.21: Effect of the voxel size on the registration time and recall for PointNetLK, for different objects, using the normal Cranfield data trained model. The effect of the voxel size on the recall metric is less pronounced compared to the global registration methods. Except for the *Guitar* object, the voxel size has a limited effect. However, a higher voxel size results in a high recall for most objects. This is interesting, as the registration time is also the lowest for these values.



(a) *Base-Top_Plate*



(b) *Guitar*

Figure 6.22: Effect of the voxel size on the registration time and failure cases for PointNetLK, for different objects, using the normal data trained model. Again, the number of failure cases does not change significantly by adapting the voxel size. The *Guitar* object fails with almost every scan, except for larger voxel sizes. The optimal voxel size, in the sense of recall or the number of failure cases, is mostly the same, as the effect of the voxel size is limited.

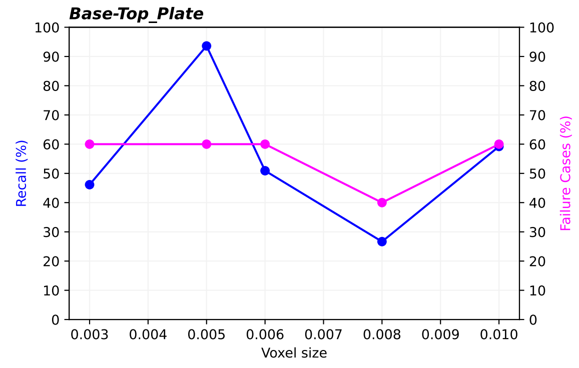
Here, pretrained models for normal, noisy and partial ModelNet40 data are also available. Figure 6.24 shows the effect of the voxel size on the registration time for the different training models.

Figure 6.24 shows that the change of training model does not impact the registration time in a meaningful way. The small differences are to be contributed to background tasks, running on the computer, and not to the registration result itself.

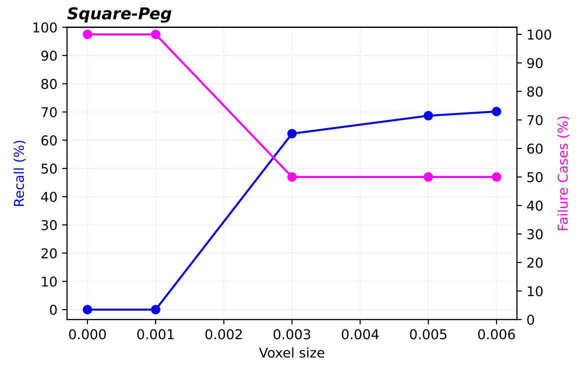
The effect of the models, trained on the Cranfield data, on the recall metric and the number of failure cases, is shown for PointNetLK in Figure 6.25. For the recall metric, the differences between the training models are not very large, usually between $5 - 10\%$. For the number of failure cases however, the normal Cranfield data model outperforms the noisy Cranfield data model.

The results of the Cranfield data trained models can be compared to those of the ModelNet40 trained models, which do not contain the Cranfield data [2], but they do contain the *Guitar*, *Bookshelf* and *Range-Hood* objects. Figure 6.26 shows that the normal Cranfield data model (●) performs overall the best when considering the recall metric. For some objects however the model trained on ModelNet40 data (●) performs better, especially when considering the number of failure cases. This is in line with expectations

[2] *Base-Top_Plate*, *Pendulum*, *Separator*, *Round-Peg*, *Square-Peg* and *Shaft-New*

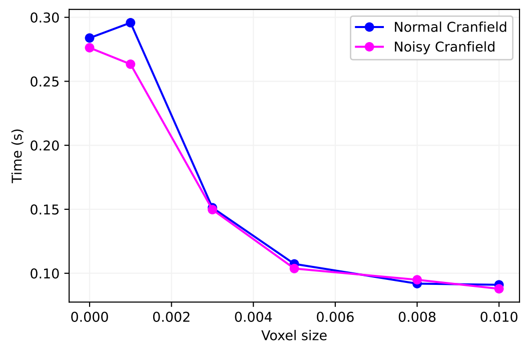


(a) *Base-Top_Plate*

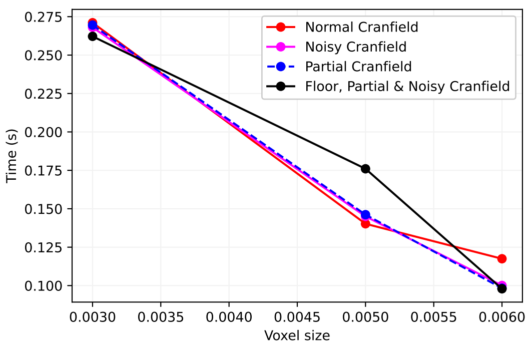


(b) *Square-Peg*

Figure 6.23: Recall (●) and Failure cases (●) for RPMNet, using the normal Cranfield data trained model, as a function of the voxel size, for different objects. The voxel size plays a more important role compared to PointNetLK. Especially the recall metric varies significantly. Notice the relation between both metrics. As more or less scans fail, the recall metric adapts accordingly. A voxel size of around 0.005 yields good results for both recall and the number of failure cases for most of these objects.

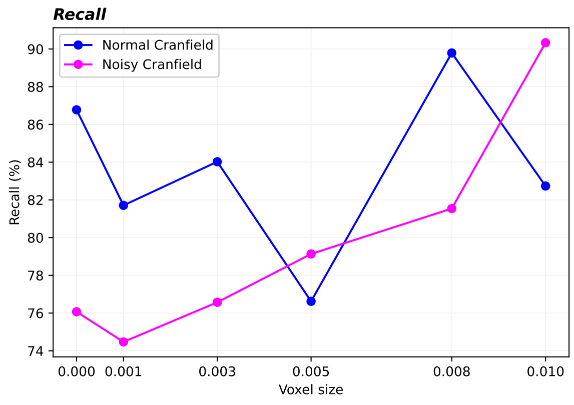


(a) PointNetLK

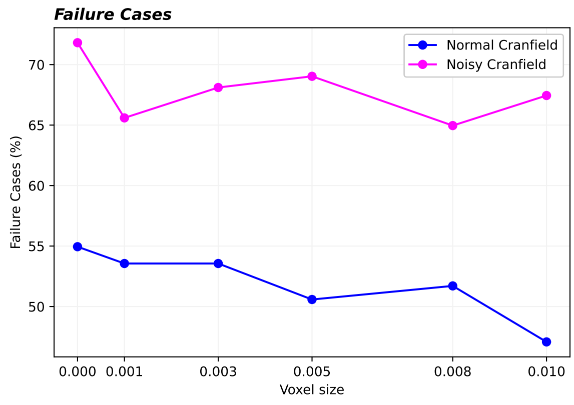


(b) RPMNet

Figure 6.24: Average registration time, as a function of the voxel size, for PointNetLK (a) and RPMNet (b). Changing the trained model does not influence the registration times. Differences could be related to random background tasks and not the actual registration process.



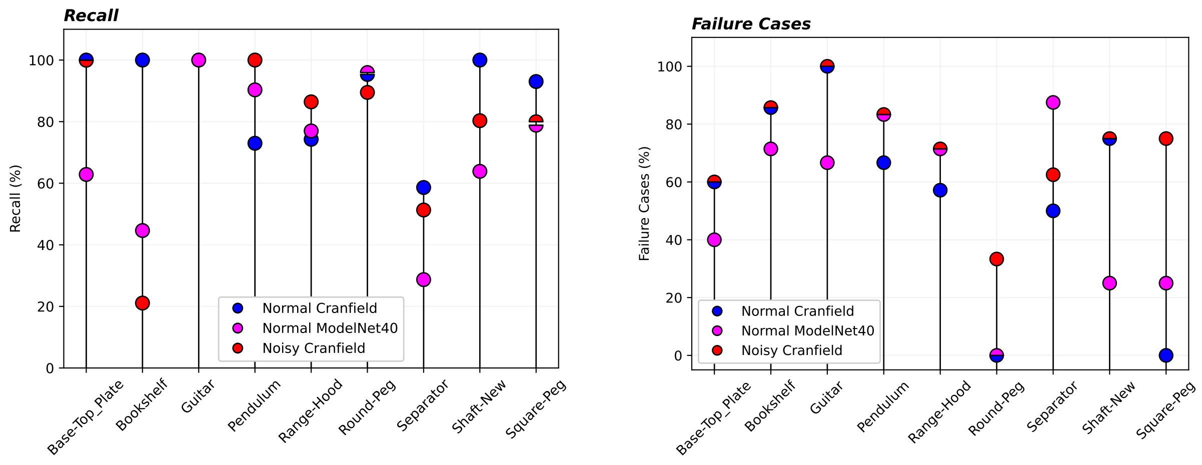
(a) Recall



(b) Failure Cases

Figure 6.25: Recall (a) and Failure cases (b) for PointNetLK, trained on normal (●) and noisy (●) Cranfield data. The noisy Cranfield data model performs worse for almost all voxel sizes when considering the recall (a) and is outperformed by the normal Cranfield data model over all voxel sizes for the number of failure cases (b).

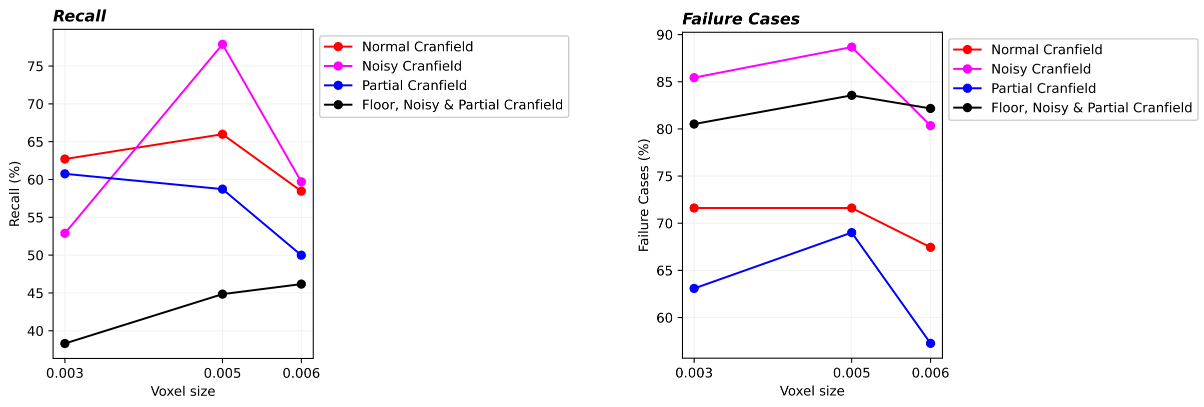
for the *Guitar* object for example. However, since the ModelNet40 data contains many more objects, compared to the limited six objects of the Cranfield data set, the model is able to generalize even to objects, on which it was not trained. The Cranfield trained models (●, ●) are trained on a more limited dataset and thus have more trouble generalizing to other objects, or even certain scans.



(a) Recall (b) Failure Cases

Figure 6.26: Recall (a) and Failure cases (b) for PointNetLK, trained on normal (●) and noisy (●) Cranfield data, and normal ModelNet40 (●) data. For the recall metric (a), the results are best when using the normal Cranfiel data trained model for most objects. The *Guitar* object is an exception to this. This object is however not included in the dataset for the normal Cranfield data trained model. It is included in the pretrained model, for which the recall is indeed higher. In terms of the number of failure cases, the pretrained data generalizes better than the normal Cranfield data, yielding a low number of failure cases even for the *Base-Top_Plate*, *Shaft-New* and *Round-Peg* objects, for which it is not trained.

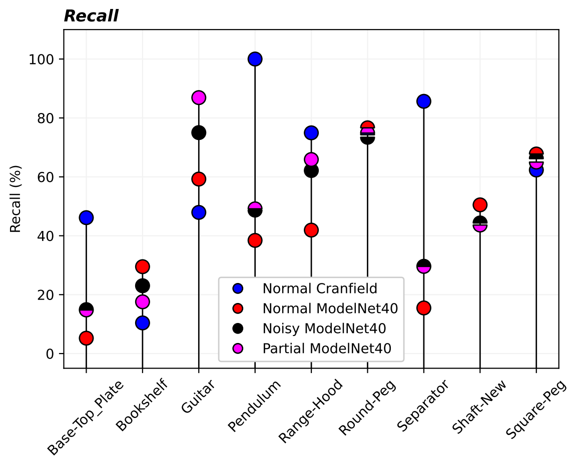
For RPMNet, the different models are compared in Figure 6.27. Here, the normal and noisy Cranfield data models (-●-, -●-) perform the best for both metrics compared to the partial and floor Cranfield data models (-●-, -●-). Notice also the decrease overall in recall (Figure 6.27a) and the increase in the number of failure cases (Figure 6.27b), compared to PointNetLK (Figure 6.25).



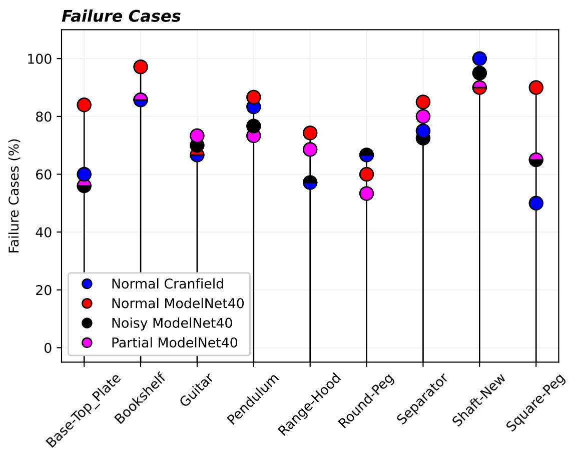
(a) Recall (b) Failure Cases

Figure 6.27: Recall (a) and Failure cases (b) for RPMNet, trained on normal (-●-), noisy (-●-), partial (-●-) and floor (-●-) Cranfield data. The results are best when using the normal (-●-) or partial (-●-) models. Depending on the metric, the normal or partial model yields a better result. However, both the recall (a) and number of failure cases (b) are not too different between the two.

Figure 6.28 compares the results to the the models trained on normal, noisy and partial ModelNet40 data. Again, the normal Cranfield data model (●) achieves, on average, a good result for most objects. The *Guitar* and *Bookshelf* objects obtain a better recall for the ModelNet40 trained models, as they are part of this dataset.



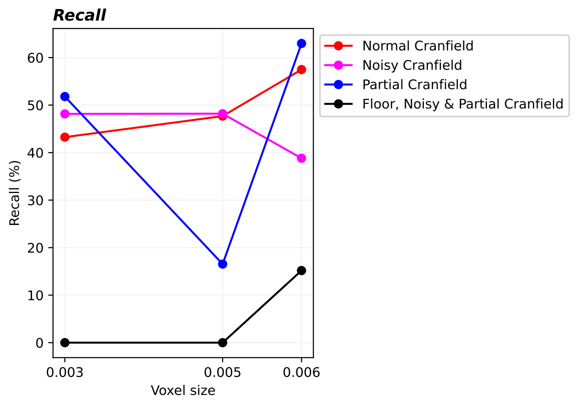
(a) Recall



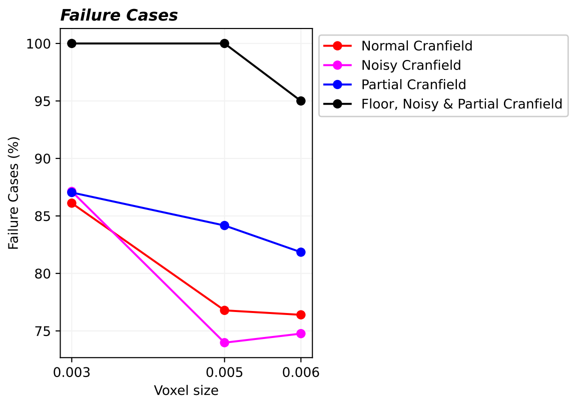
(b) Failure Cases

Figure 6.28: Recall (a) and Failure cases (b) for RPMNet, trained on normal Cranfield data (●), pretrained normal (●), pretrained noisy (●) and pretrained partial (●) ModelNet40 data. The results are best when using the normal ModelNet40 (●) or partial ModelNet40 (●) models. The normal Cranfield model (●) yields the most consistent results over all objects. Some objects, such as the *Bookshelf* or *Guitar* objects achieve a higher recall for the pretrained models, as they are not included in the normal Cranfield training data.

Finally, the same tests are repeated for ROPNet, as shown in Figure 6.29. The normal Cranfield data model (-●-) yet again performs, on average, good for both metrics, together with the noisy Cranfield data model (-●-). Again, the recall metric has dropped compared to RPMNet (Figure 6.27) and the number of failure cases has increased.



(a) Recall



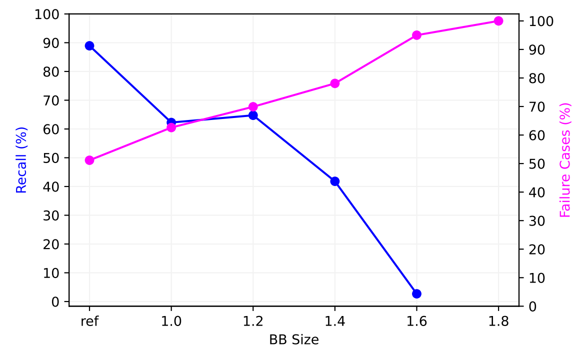
(b) Failure Cases

Figure 6.29: Recall (a) and Failure cases (b) for ROPNet, trained on normal (-●-), noisy (-●-), partial (-●-) and floor (-●-) Cranfield data. The results are shown as the average of for four objects, the *Base-Top_Plate*, *Pendulum*, *Separator* and *Bookshelf*. The normal (-●-) or noisy (-●-) models yield the best performance. On average, a voxel size of 0.006 yields the best results. However, ROPNet is outperformed by both PointNetLK and RPMNet.

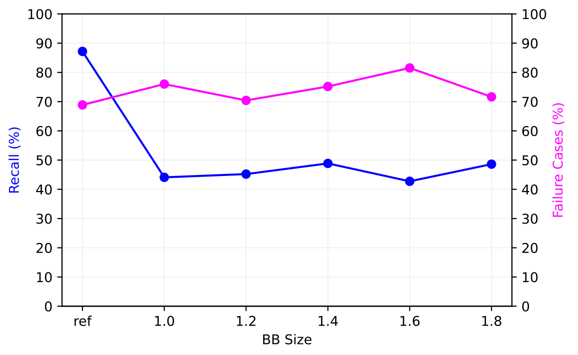
In conclusion, the specific training models, which consider for example partial or noisy data, do not necessarily lead to better results. The opposite is often true. The goal of these models is to better approximate real world data. However, simply adding noise or removing points clearly is insufficient for obtaining better results. Instead, the trained models probably overfit more on the data, leading to a worse generalization and thus worse results compared to the normal data trained models. In order for the models to generalize well to other objects, not included in the dataset, it is also important to have a wide variety of objects, as is the case for the ModelNet40 trained training models.

6.3.4 Bounding Box

The bounding box is adapted for PointNetLK and RPMNet, since these two methods lead to the best results and ROPNet only works for a limited voxel size. The results are shown for PointNetLK and RPMNet in Figure 6.30 for the recall and number of failure cases and Figures 6.31-6.32 for the relative errors.



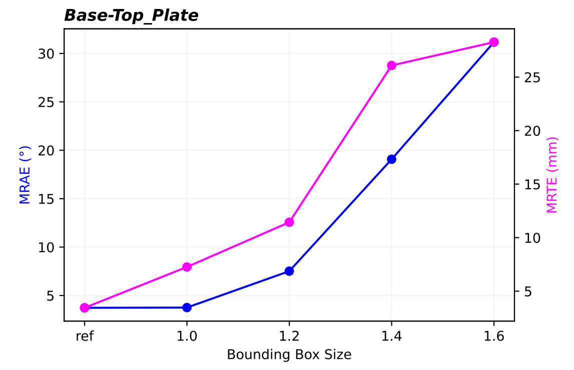
(a) PointNetLK



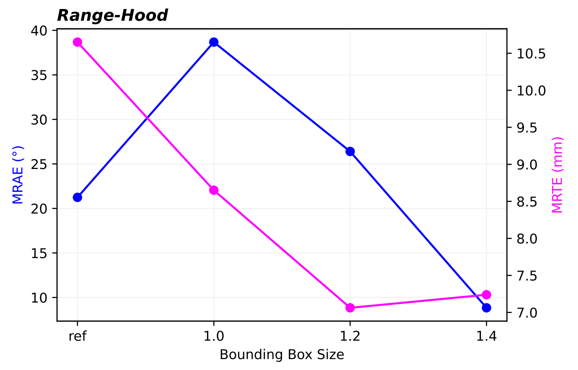
(b) RPMNet

Figure 6.30: Effect of changing the bounding box on the recall (-●-) and the number of failure cases (-●-), for PointNetLK (a) and RPMNet (b). PointNetLK (a) shows a clear decrease in the recall and increase in the number of failure cases, when the bounding box is increased. RPMNet (b) has no drastic change in performance when the bounding box is further decreased.

The effect is most clear for PointNetLK in Figure 6.30a. Both the recall and number of failure cases worsen when the bounding box is increased. For most objects, a similar effect can be seen for the relative errors in Figure 6.31. The *Range-Hood* object on the other hand, outperforms the reference case for a bounding box of 1.2 and 1.4. Some objects, like the *Pendulum* even lead to all scans failing for one value (1.2) and not for a bigger value of 1.4.



(a) *Base-Top_Plate*

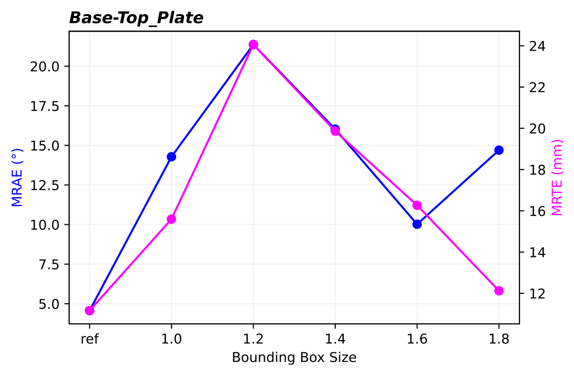


(b) *Range-Hood*

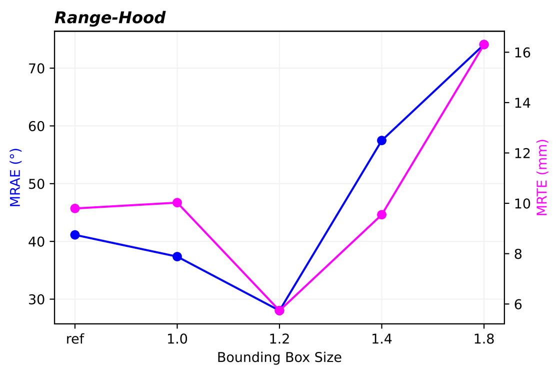
Figure 6.31: Mean relative angular (-●-) and translational (-●-) errors for PointNetLK, as a function of the different bounding boxes. For the *Base-Top_Plate* (a) object, the errors show a clear evolution. The other objects fail at bounding boxes higher than 1.4 (b) or 1.2 and show a less clear evolution. The *Range-Hood* (b) object benefits again from the additional points, leading to lower errors, however also a higher number of failure cases.

RPMNet shows a small decrease for the recall in Figure 6.30b, indicating that the method is more resistant to including more points, especially when they do not significantly change the features present in the scan. The errors in Figure 6.32 also indicate a less clear evolution, with some objects such as the *Separator* and *Range-Hood* outperforming the reference case for a bigger bounding box.

Although it is difficult to generalize these results, it is clear that the different methods, lead to different conclusions and considerations. However, as with the global registration methods, the best results are obtained when the most relevant information is present inside the point cloud. For the *Base-Top_Plate* or *Pendulum* objects this is usually the cleaned up, reference point cloud. For the *Range-Hood* object, this is around a bounding box of 1.2 or 1.4 with respect to the reference. It is also important to note that, changing the bounding box, may lead to different scans failing or working. For example, Figure 6.31a contains the results of the same scans always working, while for the *Pendulum* object, the scans change depending on the bounding box. Finally, as a recommendation, all methods should ideally be used with a cleaned-up point cloud. When this is not possible, some methods, such as PointNetLK or FGR will have a sharper decrease in performance compared to RANSAC and RPMNet. This however does not mean that RANSAC and RPMNet should be used with point clouds containing a lot of background information.



(a) *Base-Top_Plate*

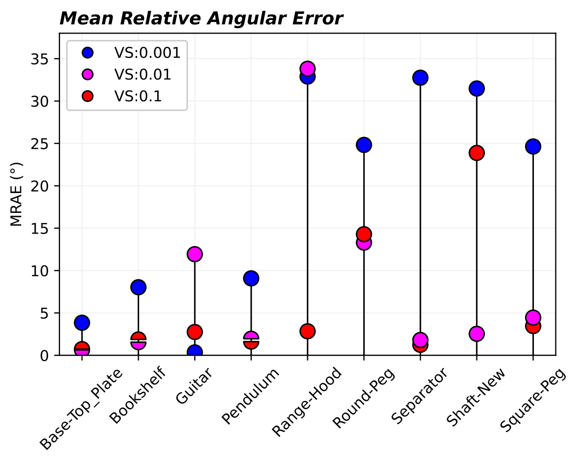


(b) *Range-Hood*

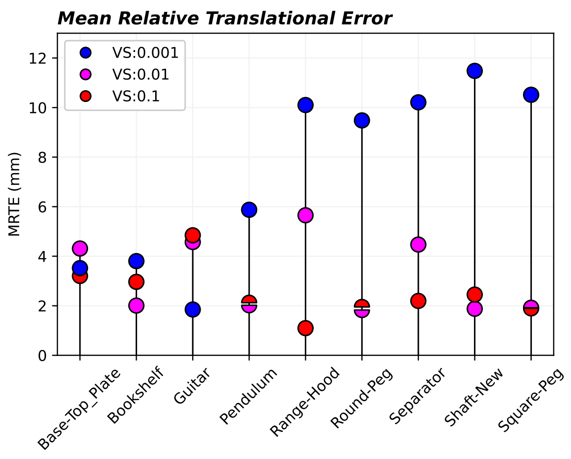
Figure 6.32: Mean relative angular (●) and translational (●) errors for RPMNet, as a function of the different bounding boxes. For larger bounding boxes, the errors mostly show an upwards trend. Again, the presence of additional points can lead to a better identification of features, resulting in lower errors. Missing points indicate a failure for this bounding box, over all scans.

6.3.5 Refinement

ICP refinement improves the rotational and translational errors, as shown in Figure 6.33 for PointNetLK and Figure 6.34 for RPMNet. Again, the higher voxel size refinement (●) performs the best overall for both the translational and rotational errors. This is also the case for the global registration methods.

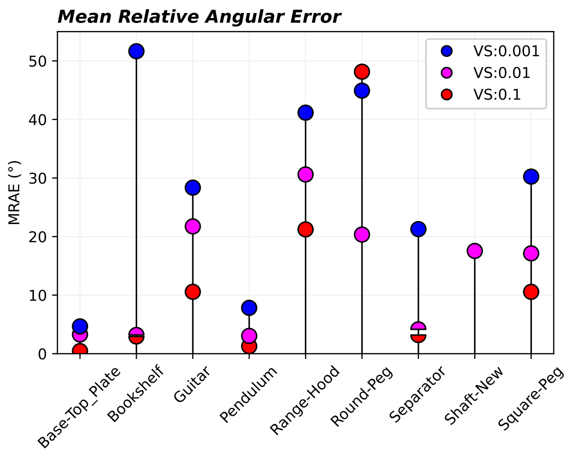


(a) MRAE

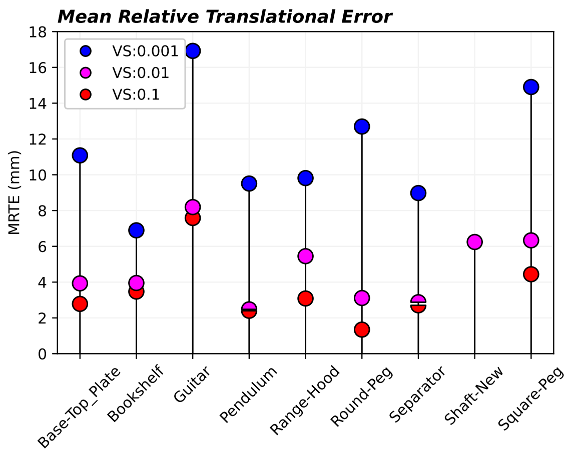


(b) MRTE

Figure 6.33: Mean relative angular (a) and translational (b) errors, for PointNetLK, all objects, after applying ICP refinement, for different voxel sizes. As for the global registration methods, the 0.1 voxel size (●) yields the best performance overall. For some objects (*Shaft-New*, *Bookshelf*), a voxel size of 0.01 (●) results in even lower errors.



(a) MRAE



(b) MRTE

Figure 6.34: Mean relative angular (a) and translational (b) errors, for RPMNet, all objects, after applying ICP refinement, for different voxel sizes. As for the global registration methods, the 0.1 voxel size (●) yields the best performance overall. For some objects (*Shaft-New*, *Round-Peg*), a voxel size of 0.01 (●) results in even lower errors.

6.3.6 Failure Cases

The scans leading to failure and success are, for most objects, similar to the global registration methods. To reiterate, the quality of the scan is an important factor. However, the size of the transformation and the initial pose of the template also play a role.

6.3.7 Conclusion

PointNetLK is relatively insensitive to a changing voxel size, and yields on average the best results for the normal Cranfield data training model. The bounding box leads to a steep decrease in performance, similarly to the global registration methods. Refinement can help to improve the registration results overall.

RPMNet leads to different results, depending on the selected voxel size. However, a too large voxel size can cause the algorithm to not work anymore, due to too little points. A low voxel size requires a larger computational effort, especially for the larger objects. The normal and partial Cranfield trained models perform the best in terms of recall and number of failure cases. Again, ICP refinement leads to better translational and rotational errors when a larger voxel size is selected.

ROPNet performs the worst between the registration methods, with an overall lower recall and a higher number of failure cases. In the same way as RPMNet, ROPNet does not work anymore for too high voxel sizes and requires more GPU power. The normal Cranfield data trained model shows the best results overall.

Chapter 7

Experiments Conclusions

In this chapter, the main conclusions from Chapter 6 are first summarised in Section 7.1. An answer to each of the research questions is formulated in Section 7.2 and some guidelines for selecting the best registration method, considering certain specifications are also given in Section 7.3.

7.1 Parameters

In this section the main effects of each parameter, introduced in Chapter 6 are given. More details into the reasons behind these evolutions can be found in Chapter 6. This section is only meant to give a brief overview of the main conclusions.

7.1.1 Zero Mean

Before applying registration, the point clouds can be centered on the origin to reduce the size of the transformation to be estimated. It was found that this operation has little effect on the metrics. The best case was chosen for each method; GO-ICP, RPMNet and ROPNet are not centered (non-zero mean), while RANSAC and FGR are centered (zero mean). PointNetLK centers the point clouds automatically.

7.1.2 MSE Threshold & Trim factor

The MSE threshold and trim factor are parameters used only for GO-ICP. The MSE threshold determines the convergence of the method, while the trim factor is used to remove outliers. From the experiments, it was concluded that the lowest thresholds yield the best results, but require the highest computation times (several seconds up to a minute for larger objects). Increasing the thresholds lowers the performance drastically. The trim factor has little effect on the results.

7.1.3 Voxel Size

The voxel size can be seen as a simple filtering technique, where the point cloud is reduced by taking the average of all points inside a voxel with a given size, the voxel size. This parameter affects both the learning-based and the global registration methods.

The larger the voxel size, the lower the number of points and thus the lower the registration time. This holds true for all methods, except for ICP. This is however a local registration method, to be used only for smaller transformations. For too large transformations, the algorithm performs little iterations leading to lower registration times.

For RANSAC, FGR, RPMNet and ROPNet, the voxel size also affects the registration results. For these four methods, an optimal voxel size could be determined by considering the maximal recall, the minimal number of failure cases or any other metric. However, this optimal value might differ between these different metrics and is highly dependent on the object shape, size, the considered method and the considered metric. For these methods, no clear evolution could be seen when changing the voxel size. One important comment is related to ROPNet and RPMNet. These are learning-based methods and require a minimal number of points in order to work. Thus, after increasing the voxel size too much, too little points will be left, and the methods will not work anymore.

PointNetLK appeared to be more resistant against a changing voxel size, with little effect on the recall metric and the number of failure cases for most objects. However, best performances were found for higher voxel sizes (0.01).

GO-ICP can also be combined with the voxel size parameter. However, it is important to combine the voxel size with the MSE threshold, as these two cannot be changed independently from one another. Increasing the voxel size, reduces the spacing between the points in the point cloud leading to convergence issues when the threshold is not adapted. The voxel size can prove especially useful in cases of higher MSE thresholds, to improve upon the results without considering the voxel size. This is when the original point cloud is used, without filtering.

7.1.4 Bounding Box

Another studied parameter is the bounding box. Here, the scanned point cloud is considered for different situations, where each time, more points from the environment are considered as the bounding box increases. This is again shown in Figure 7.1.

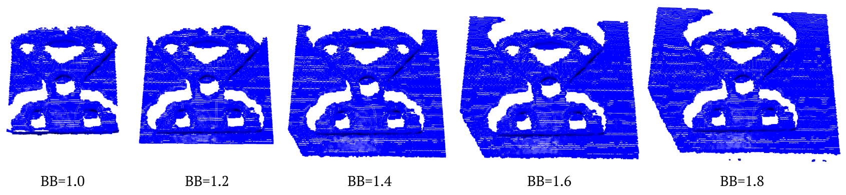


Figure 7.1: Example of increasing the bounding box for the *Base-Top_Plate*, each time with 20% from the reference case of $BB = 1.0$.

It was found that increasing the bounding box yields overall a worse result for most methods. Exceptions occur for objects, such as the *Range-Hood* object, where the additional points may lead to better registration results. A general guideline would be too clean up the point cloud, while maintaining the most relevant information, to achieve the best results. The main difference between the considered methods lies in the sensitivity to the bounding box. PointNetLK and FGR led to a steeper decrease in performance as the bounding box increased, while RPMNet and RANSAC still lose in performance, but at a lesser rate. GO-ICP does not converge anymore when too much points, that are not present in the template, are included. ROPNet results in a low performance overall and only works for a limited number of voxel sizes and was thus not considered during these tests.

7.1.5 Training Models

For the learning-based registration methods, the different trained models are compared to each other. In terms of timing, the models do not have any significant effects. The other metrics, such as recall and the number of failure cases can be influenced. For PointNetLK, the normal Cranfield data trained model performed the best overall. For RPMNet, the same model yields on average a good performance. Depending on the considered metric, the partial or noisy Cranfield data trained model can outperform the normal Cranfield data trained model. ROPNet leads to the highest errors, but similar conclusions to RPMNet still apply.

For PointNetLK and RPMNet, some models trained on the ModelNet40 dataset were also compared. These models have the advantage of being trained on a much larger dataset, allowing them to generalize better, at the expense of less accurate results. Except for objects such as the *Guitar* or *Bookshelf*, which are only a part of the ModelNet40 trained models, the normal Cranfield data trained model, usually performs the best on a wide range of tested objects.

7.1.6 Refinement

All methods achieved better results when a refinement step, using ICP, was applied on the results. Overall, a voxel size of $0.01 - 0.1$ led to the best result for all objects when applying refinement. For GO-ICP, the effect of refinement depends on the MSE threshold. For very low thresholds, the results are already converged and the gain of applying refinement is close to zero. For larger thresholds, this gain increases, as the metrics can still be improved.

7.1.7 Failure Cases

Finally, from the tests it was seen that there are in general three reasons for a failed registration result. The first one is related to the quality of the captured point cloud. If the scan does not represent the object or its features in an accurate way, there is a higher chance of failure. Secondly, the initial pose and size of the transformation also play a role. The registration is more likely to fail when the point clouds are positioned close to a local minimum in the original pose. A larger transformation also increases the difficulty of finding an accurate registration result. Finally, the parameters discussed before also play a vital role. Depending on the voxel size, bounding box and used method, some scans might lead to failure, while others do not. It is thus important to be mindful when selecting these variables, considering the results mentioned in this section.

7.2 Research Questions

In this section, an answer to the research questions posed in Chapter 1 is formulated. The conclusions are based on the setup of the experiments (Chapter 4), the training of the learning-based methods (Chapter 5) and the parametric study (Chapter 6). For obtaining a general overview, the tables mentioned in the subsequent sections consist of the averages taken over the best results for each object. This means that the test case from Chapter 6, resulting in the highest number of best metrics, for each method is considered. Chapter 6 presents a series of test cases that examine the impact of varying parameters, including voxel size, mean squared error (MSE) threshold, trim factor, and training models, on overall system performance through different experiments. The best metrics are the lowest errors for the group 1 (rotational and translational errors) and group 3 (number of failure cases) metrics (Table 3.1) and the highest metrics for group 2 (recall and R^2). The complete set of data, as well as all of the individual results from the methods are available on the Github page.

First of all, the training and the trained models are discussed in Section 7.2.1, to answer the following question: *Is it sufficient to only use the available CAD models for training the learning-based methods?*.

Secondly, the required quality of the point cloud scans is discussed in Section 7.2.2. *How accurate should the depth image of the object be to achieve satisfying results?* This is the main question answered.

Finally, a comparison of all methods is given in Section 7.2.3, to answer the question: *Which of the studied registration methods yields the best performance?*

7.2.1 Is it sufficient to only use the available CAD models for training the learning-based methods?

As explained in Chapter 5, the CAD models of the objects are used to generate different types of training datasets. The Cranfield objects are converted to a point cloud and randomly transformed. Noise, partiality or floor information can also be added. Some examples are shown in Figure 7.2.

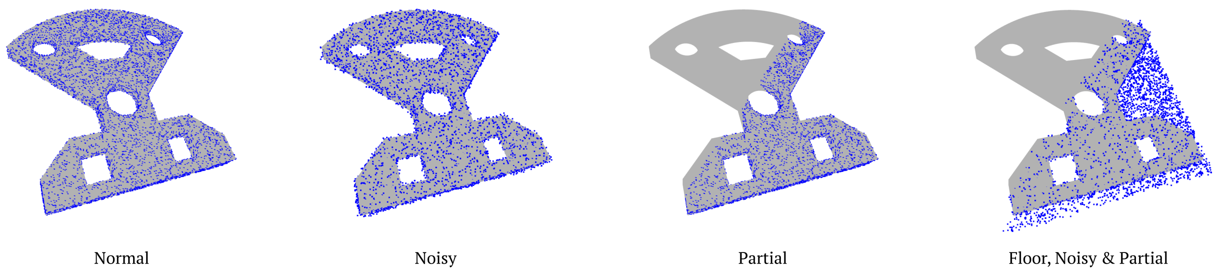


Figure 7.2: The algorithms are trained using different datasets. They can be simply sampled from the CAD models (Normal), or noise can be added (Noisy). Points can also be removed (Partial). Finally, a floor can be added to mimic, for example, the presence of a table (Floor, Noisy & Partial).

The results for the different training models, for each method, taken as the average over the best cases in terms of the voxel size, are summarised in Table 7.1.

Table 7.1: Overview of all metrics, for the learning-based registration methods, for each training model. The results are taken as the average over all objects. For each object, the best results are selected, in terms of the voxel size. Recall is taken with a 0.01 threshold. Best errors are shown in **bold** per method. For ROPNet, the floor Cranfield data model is not mentioned as only one scan, from one object does not fail.

Model	MRAE [°] ↓	MRTE [mm] ↓	RMSE [°] ↓	RMSE [mm] ↓	MAE [°] ↓	MAE [mm] ↓	Recall [%] ↑	R^2 [/] ↑	Failure [%] ↓
PointNetLK									
Normal	**14.14**	**7.06**	**0.12**	**4.07**	13.00	**0.02**	**95.20**	**0.87**	**51.71**
Noisy	16.17	9.62	0.13	5.55	**9.20**	0.03	90.95	0.81	65.60
RPMNet									
Normal	21.68	11.40	**0.18**	6.58	13.77	0.05	**76.53**	**0.63**	66.92
Noisy	28.60	**9.50**	0.23	**5.48**	12.83	**0.03**	60.96	0.61	77.83
Partial	27.30	13.19	0.22	7.61	**9.87**	0.06	59.35	0.63	**60.97**
Floor	**18.12**	13.54	0.32	7.82	38.90	0.07	43.67	0.60	77.08
ROPNet									
Normal	**22.73**	11.14	**0.18**	6.43	**10.54**	0.05	**71.11**	**0.74**	79.06
Noisy	24.25	14.44	0.20	8.34	12.85	0.07	60.30	0.63	76.55
Partial	31.50	**10.32**	0.25	**5.96**	40.67	**0.04**	68.92	0.60	**75.26**

Table 7.1 shows that PointNetLK outperforms both RPMNet and ROPNet in all metrics. For PointNetLK, the normal Cranfield data trained model achieves the best results overall. However, except for the number of failure cases, the metrics are not very different. For RPMNet and ROPNet, the best metrics are more scattered over the different models. For RPMNet, the normal and noisy Cranfield data trained models perform the best overall. For ROPNet, the normal and partial Cranfield trained models yield the best errors. Important to note is that the results are not all too different. When training the methods with the available CAD models, it is thus sufficient to use a normal Cranfield data trained model. Adding noise or partiality may improve the results for some metrics, but the training process also becomes more challenging, as already mentioned in Chapter 5.

Of course, it is possible to create more complex and extensive datasets, to mimick the real world scans in a better way. However, as discussed in Chapter 5, a balance must be found between the stability of training, the available resources, time and the complexity of the training dataset. Adding simple noise or partiality may lead to some improvements, but may also lead to overfitting of the data, leading to more failure cases and less generalizability as also shown in Table 7.1, when looking at the number of failure cases.

The results presented in Table 7.1 can still be refined using ICP refinement, which is further discussed in Section 7.2.3.

7.2.2 How accurate should the depth image of the object be to achieve satisfying results?

Assigning a quantitative value to the accuracy of the depth image is difficult. There are many parameters that play a role as explained in Section 7.1. Whether the results are considered to be satisfactory can also depend on the specific application. For this reason, this section aims to give some qualitative considerations, to take into account when creating point clouds. The quality of the registration result depends on the captured point cloud, the transformation size an the initial pose. In general, based on the experiments, some general, qualitative guidelines are the following.

- Clean up the point clouds from any outliers. These refer to a single point, or a small cloud of points that are not related to the object. Simple denoising operations can be performed with the Open3D toolbox in Python, as mentioned in Chapter 4.

- Limit the point cloud to the relevant information, excluding environment points such as the floor. This is more critical for some methods, such as PointNetLK and GO-ICP. Removing planes can be achieved using the Open3D toolbox (Chapter 4).
- Limit the number of points removed related to the object. Maximize the information related to the object, while minimizing the environment effects.
- Keep in mind the limitations of the used camera when creating a point cloud. Smaller objects, require the camera to be closer and more precise. Most Intel cameras have a minimal depth distance, before which they cannot capture a point cloud. Pick the right camera for the application, depending on the object size.
- If possible, capture the object from a position that requires a limited transformation. The smaller the transformation to be found, the better the results are in general. An alternative would be to use a random initialization of the algorithm.
- If the point cloud is generated from a single pose, capture the object from a view that includes as much of the object as possible, as opposed to a small part or section. Be mindful of specific features, such as holes or edges.
- Consider lightning conditions and reflections on the objects. Symmetry of the objects also plays a role in determining the performance or when training the learning-based methods.
- Tweak the camera settings to obtain a sharp and clear point cloud, reducing waviness and rounding. The Intel Realsense cameras already provide some pre-processing options, as discussed in Chapter 4.

These guidelines are only meant to focus the reader on some specific points to be taken into account when creating a point cloud. There are other improvements or future paths that can be explored, which are discussed in Chapter 8.

7.2.3 Which of the studied registration methods yields the best performance?

Finally, all the registration methods can be compared to each other, in terms of their best performances, averaged over all objects. The results are shown in Table 7.2 for the global registration method and Table 7.3 for the learning-based methods. In Tables 7.2-7.3, both the metrics (ϵ) discussed in Chapter 3 as well as the standard deviation (σ) between the objects and scans are reported. This standard deviation is not a measure of the repeatability of the experiments. Instead, it signifies the variations in metrics between the objects and the individual scans of the same object. High values indicate that some specifc scans may lead to a much better performance than others. Low values indicate that most objects yield similar errors. These standard deviations are computed as follows:

$$\sigma = \sqrt{\frac{\sum_{obj} Var(obj)(N_{obj}-1)}{\sum_{obj}(N_{obj}-1)}} \tag{7.1}$$

$$Var(obj) = \frac{1}{N_{obj}-1}\sum_{i=1}^{N_{obj}}(x_i - \bar{x})^2 \tag{7.2}$$

$$N_{obj} = N_s N_{it} - N_f, \tag{7.3}$$

where the sum is taken over the variance of all the objects ($Var(obj)$), weighted by the number of samples. The objects variance is computed with the typical variance formula, where N_{obj} is the total number of experiments per object, or the number of scans N_s multiplied with the number of iterations per scan N_{it} for which the experiments is repeated. From this value, the number of failure cases N_f is subtracted, since the errors are not computed here. x represents one of the nine considered metrics from Tables 7.2-7.3 and $\bar{x}$ its mean value over the experiments. For the number of failure cases, Equation 7.2 is used directly, with the sum taken over the different objects instead.

As mentioned in Chapter 4.2.3, the ground truth definition is not perfect, as the scanned point cloud is usually a bit deformed compared to the template. This means that errors of a couple of degrees and millimeters are to be expected. Figure 7.3 shows an example of the alignment result, for some of the best cases, used in determining the results of Tables 7.2-7.3.

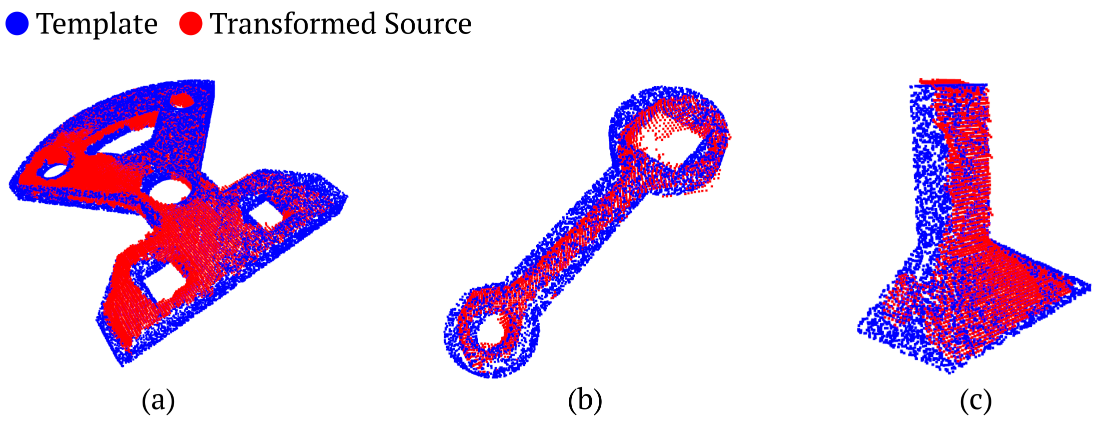


Figure 7.3: Best results for (a) *Base-Top_Plate*, RANSAC, VS: 0.003, $MRAE = 2.88°$, $MRTE = 3.27mm$, (b) *Pendulum*, PointNetLK, VS: 0.008, Model: Noisy Cranfield, $MRAE = 3.35°$, $MRTE = 7.04mm$, (c) *Range-Hood*, GO-ICP, MSE: 0.0001, Trim: 0.0001, $MRAE = 4.51°$, $MRTE = 0.45mm$

First of all, GO-ICP yields overall the best performances over all objects, compared to the other registration methods. With only around 5° rotational errors and around $2mm$ translational errors. The number of failure cases is limited and usually only appear for smaller objects or lower quality scans. However, as shown in Chapter 6, the registration times can run up to $5s$ or even a minute for the biggest objects, when the lowest thresholds are selected. ICP refinement does not influence the results significantly, as already mentioned before in Section 7.1. The variations between objects and scans is the most limited for GO-ICP, indicating a good result over a wide range of scans (or poses) and object types. These results are in line with the literature [53], discussed in Section 2.3, where the speed of GO-ICP is also indicated as its major disadvantage.

For the global registration methods, RANSAC follows the performance of GO-ICP, yielding similar errors of around 7° and $3mm$ after applying ICP refinement. However, RANSAC on itself leads to higher errors of around 18° and $5mm$, with often high variances, especially for the mean absolute errors (MAE). This is not surprising, as RANSAC uses a random initialization, meaning that the same scans or input point clouds may lead to very different results. This variance can be beneficial to overcome the problem of local minima, but introduces less robustness. The refinement also reduces the standard deviations. As mentioned in Section 2.3, RANSAC is able to overcome initialization problems and partiality. However, parameters need to be chosen correctly to achieve high performances. This is in correspondence to the literature [13].

FGR leads to similar conclusions as RANSAC, although FGR performs a bit worse with errors around 10° and $5mm$ after refinement. The number of failure cases is also higher, indicating that FGR works for a more limited range of object types and initial poses. This is in line with Section 2.3, where the sensitivity of FGR to noise is mentioned, while also being the quickest method. This is again similar to the performance mentioned in the literature [59].

Between the learning-based methods, PointNetLK achieves overall the best results, except for the mean absolute error (MAE) and the number of failure cases, which is a bit lower for a refined RPMNet. After applying refinement, PointNetLK achieves errors around 6° and $2mm$, which is close to the results obtained for GO-ICP. However, the number of failure cases, the variations between objects and the recall are all worse than GO-ICP. Similar to RANSAC and FGR, the results are worse without refinement. Here, PointNetLK reaches errors of around 13° and $5mm$. Since PointNetLK yields the same results for a single scan, the deviations are a bit lower than FGR and RANSAC. However, this comes at the cost of sometimes falling into local minima and thus a higher number of failure cases compared to RANSAC. PointNetLK requires the least strong GPU between the learning-based methods. The registration time itself is limited to around $0.1s$, however, during testing on a personal device, the dataloading could take up to $11s$. This effect was significantly reduced when working with the GPU from Google Colab to a time similar to the registration time. The data loading is however not related to the actual registration method and could be improved. As discussed in Section 2.3 and literature [2], the generalizability of PointNetLK to unseen categories of data is limited, shown by the reduced performance for objects such as the *Guitar*. Additionally, as discussed in Section 7.1, large transformations can prove challenging.

Table 7.2: Overview of all metrics, for the global registration methods. The results are taken as the average over all objects. For each object, the best results are selected, in terms of the voxel size, MSE threshold and trim factor, meaning the test case with the highest amount of best errors. Recall is taken with a 0.01 threshold. For each method the errors (ϵ) and standard deviations (σ) are given, as well as the refinement case (ICP). Best errors are shown in **bold**.

Data [ϵ/σ]	MRAE [°] ↓	MRTE [mm] ↓	RMSE [°] ↓	RMSE [mm] ↓	MAE [°] ↓	MAE [mm] ↓	Recall [%] ↑	R^2 [/] ↑	Failure [%] ↓
Global Registration									
GO-ICP									
ϵ	**4.85**	2.73	**0.04**	1.58	4.00	**0.00**	**100**	**0.98**	**28.64**
σ	3.08	1.61	0.02	0.93	4.13	0.01	0.00	0.04	30.08
GO-ICP + ICP									
ϵ	4.85	**2.73**	0.04	**1.58**	**3.99**	0.00	100	0.98	28.64
σ	3.10	1.58	0.03	0.91	4.14	0.01	0.00	0.04	30.08
RANSAC									
ϵ	18.62	6.23	0.16	3.60	16.43	0.02	85.94	0.88	50.82
σ	12.64	4.20	0.09	2.42	31.31	0.03	21.71	0.13	26.24
RANSAC + ICP									
ϵ	5.85	2.83	0.05	1.63	9.70	0.01	97.34	0.93	43.58
σ	7.20	2.51	0.06	1.45	29.05	0.02	9.52	0.16	24.63
FGR									
ϵ	27.41	11.67	0.29	6.74	19.08	0.06	61.18	0.58	75.57
σ	38.52	4.56	0.13	2.63	14.14	0.06	27.50	0.20	17.52
FGR + ICP									
ϵ	12.59	4.95	0.10	2.86	7.81	0.01	91.26	0.81	47.26
σ	12.26	3.40	0.10	1.96	7.00	0.02	13.62	0.20	11.46

However, performances for the translational and rotational errors were able to almost match those of GO-ICP after applying refinement.

RPMNet achieves errors of around 6° and $3mm$ after ICP refinement. The number of failure cases is a bit lower compared to PointNetLK, RANSAC and FGR, but the variations are again higher. Although these deviations are more limited than for RANSAC, FGR and even PointNetLK (without considering refinement). RPMNet has a slightly higher registration time compared to PointNetLK and the global methods. It also suffers from the same effect as PointNetLK when loading the data. As indicated in Section 2.3, RPMNet is more robust to initialization problems, which is reflected in the lower number of failure cases, compared to PointNetLK and FGR, especially after refinement. The method is also a bit slower, as also mentioned in the literature study [54].

Finally, ROPNet reaches errors of around 20° and $3mm$ with a higher number of failure cases of 70% after refinement. This method is thus outperformed by the other registration methods. One slight advantage of using ROPNet is the limited time required to load the data and perform the registration (around $0.05s$), compared to PointNetLK and RPMNet. However, this difference quickly diminishes when using a stronger GPU. ROPNet has the additional disadvantage of requiring more GPU power and only working for a more limited voxel size range. Compared to the global methods, ROPNet also needs to be trained. ROPNet yields a lower performance than expected from the literature study [63]. This could be related to the dataset not being sufficiently representative of the real world data scans.

To summarise, a qualitative comparison is made in Table 7.4. Different applications, might require different methods. When speed is crucial, method such as RANSAC and FGR can be considered. When

Table 7.3: Overview of all metrics, for the learning-based registration methods. The results are taken as the average over all objects. For each object, the best results are selected, in terms of the voxel size and training model, meaning the test case with the highest amount of best errors. Recall is taken with a 0.01 threshold. For each method the errors (ϵ) and standard deviations (σ) are given, as well as the refinement case (ICP). Best errors are shown in **bold**.

Data $[\epsilon/\sigma]$	MRAE [°] ↓	MRTE [mm] ↓	RMSE [°] ↓	RMSE [mm] ↓	MAE [°] ↓	MAE [mm] ↓	Recall [%] ↑	R^2 [/] ↑	Failure [%] ↓
Learning-based Registration									
PointNetLK									
ϵ	14.04	6.36	0.11	3.67	12.55	0.02	95.18	0.86	57.92
σ	13.86	1.09	0.11	0.63	34.82	0.01	12.44	0.16	34.05
PointNetLK + ICP									
ϵ	**4.12**	**2.36**	**0.03**	**1.36**	7.80	**0.00**	**99.57**	**0.97**	58.65
σ	8.55	1.31	0.07	0.76	36.18	0.00	2.95	0.15	28.24
RPMNet									
ϵ	25.78	10.21	0.21	5.90	11.08	0.04	71.48	0.62	59.31
σ	7.77	1.09	0.06	0.63	2.75	0.01	14.83	0.14	25.20
RPMNet + ICP									
ϵ	7.59	3.65	0.06	2.11	**4.52**	0.01	95.94	0.89	**45.52**
σ	8.05	2.80	0.06	1.62	6.61	0.01	9.31	0.22	30.34
ROPNet									
ϵ	25.88	10.99	0.21	6.35	30.84	0.05	67.38	0.72	74.70
σ	10.22	2.19	0.08	1.27	9.32	0.02	6.50	0.08	16.90
ROPNet + ICP									
ϵ	7.90	4.89	0.06	2.82	31.91	0.01	94.33	0.95	70.60
σ	12.31	3.30	0.09	1.91	8.76	0.02	6.29	0.12	17.01

higher accuracies are required it is possible to use GO-ICP or PointNetLK with refinement. If the precision of the method is not as crucial, for example to get a first approximation of the result, methods such as FGR, RPMNet or even ROPNet can be used.

Table 7.4: Comparison of the tested registration methods. + indicates a very good performance, m a medium performance and − a bad performance. GO-ICP leads to very precise, low variance results, but takes the longest computation time and requires cleaned-up pointclouds. PointNetLK is much faster, but has a lower precision, more variance and a higher requirement on the GPU. RANSAC and FGR are the quickest and most performing, however they lead again to low precision and high variance. Except for GO-ICP, most methods still work with a non-cleaned-up point cloud, however usually at a worse performance.

Method	**Precision**	**Variance**	**Time**	**Pre-processing**	**GPU Requirements**
GO-ICP	+	+	−	−	m
RANSAC	m	−	+	m	+
FGR	−	−	+	m	+
PointNetLK	m	m	m	−	m
RPMNet	m	m	m	m	−
ROPNet	−	m	+	m	−

7.3 Guidelines

To conclude this chapter, a framework is presented in the form of a flowchart to choose a registration method, depending on the requirements and specifications. The aim of these flowcharts is to help other researches who want to use a registration method in their own application or research. These flowcharts, presented in Figures 7.4-7.6, contain a number of questions, based on the carried out experiments. In this section, a rundown is given of how to read these charts.

Firstly, the availability of computation power is a major consideration when choosing between learning-based and global registration methods. As mentioned in Chapters 5-6, learning-based methods require a significant amount of GPU power during the training process. Thus, if no strong GPU is available, these methods are no longer a viable option. When sufficient power is available, both approaches are still open.

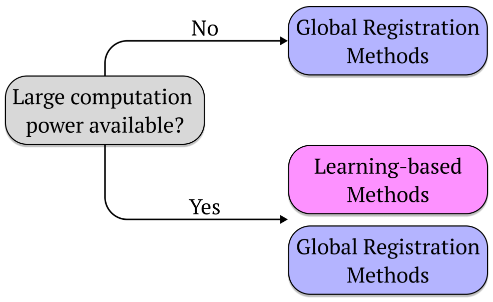

Large computation power available?
No
Global Registration Methods
Yes
Learning-based Methods
Global Registration Methods

Figure 7.4: The availability or large computation power plays an important role in choosing the right registration method. If sufficient power is available, both options remain open. Otherwise, the global registration methods are to be used.

For the global registration methods, RANSAC, FGR and GO-ICP, the flowchart of Figure 7.5 is followed. A first important question is related to the pre-processing of the data. This question refers to the possibility of cleaning up the point cloud to only contain relevant points and thus no floor points for example. As mentioned in Section 7.1, a too large bounding box reduces the performance significantly for most methods. However, some methods are more resistant to this than others. GO-ICP does no longer converge when there are too many irrelevant points and thus no longer becomes an option when pre-processing is not possible in the application.

When pre-processing, to remove the environmental information, is possible, all three methods are still open. The next question is related to the computation time. When speed is critical in the application, GO-ICP again should no longer be considered, since it leads to registration times of several seconds up to a minute. If speed is not critical, the precision and variance can be considered. GO-ICP yields the lowest errors and variances as shown in Table 7.2. If precision is not critical, for example when an approximate solution is sufficient or when speed is more important, RANSAC or FGR should be considered.

The choice between RANSAC and FGR, which is slightly faster, lies mainly in the accuracy but also in the number of failure cases (Table 7.2). Or, in other words, the number of poses of an object that lead to a successful registration result. Alternatively, when looking for speed or when pre-processing is not available, the alternative path of the learning-based methods can also be looked into. This is of course only possible when sufficient computation power is available.

For GO-ICP, the threshold, trim factor and voxel size are still to be determined. For achieving the best results, the threshold should be chosen as low as possible. When the process needs to be sped up, the threshold can be increased. Adapting the voxel size at the same time can improve the results here too. For RANSAC and FGR, mainly the voxel size has to be chosen.

All methods can be refined using ICP to achieve better performance. As mentioned in Section 7.1, for GO-ICP, the refinement only leads to an added gain for higher thresholds.

Figure 7.6 shows a similar flowchart for the learning-based methods. Again, a first question to be asked is related to the availability of pre-processing. When the point cloud cannot be cleaned up, PointNetLK is not an ideal choice as shown in Section 7.1. RPMNet or ROPNet are better choices in this case.

When pre-processing is possible, the GPU strength should be considered. PointNetLK requires much less GPU power compared to RPMNet and ROPNet. Thus, when a strong GPU is not available for executing

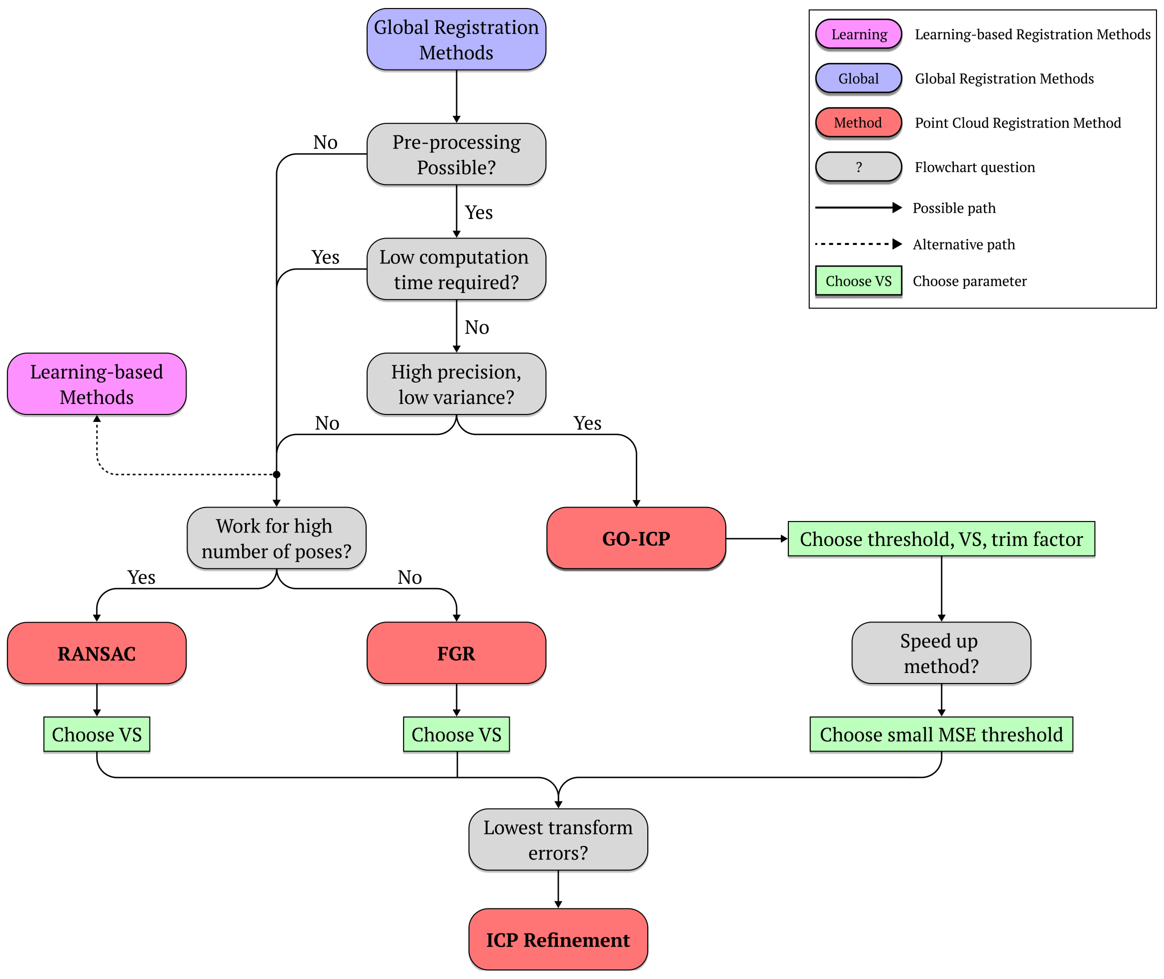

Global Registration Methods
Pre-processing Possible?
No
Yes
Low computation time required?
Yes
No
High precision, low variance?
No
Yes
Learning-based Methods
Work for high number of poses?
Yes
No
GO-ICP
Choose threshold, VS, trim factor
RANSAC
FGR
Speed up method?
Choose VS
Choose VS
Choose small MSE threshold
Lowest transform errors?
ICP Refinement
Learning
Learning-based Registration Methods
Global
Global Registration Methods
Method
Point Cloud Registration Method
?
Flowchart question
Possible path
Alternative path
Choose VS
Choose parameter

Figure 7.5: Flowchart showing the different questions to be asked when choosing one of the considered global registration methods, RANSAC, FGR and GO-ICP. A legend is shown in the top right corner. The gray boxes indicate the question to be asked and full arrows the path to follow. The red boxes show the registration methods. The green boxes indicate the parameters to tune. Dotted lines indicate alternative paths, where the learning-based methods could be considered instead.

the registration, PointNetLK is the better choice. Important to note is that this question differs from the one introduced in Figure 7.4. The training of the learning-based methods is an intensive process, that requires significant GPU power. However, the training process is normally done only once, before using the methods in the application. For PointNetLK, the training process benefits from using stronger GPU's, to save time and consider bigger datasets, however the actual registration process does not need as high specifications. For RPMNet and ROPNet, the actual registration process can also be intensive in terms of required memory and power, especially for lower voxel sizes.

A next question is related to the complexity of the dataset when training. As discussed in Chapter 5, training PointNetLK is more difficult than RPMNet or ROPNet. The latter two methods could be trained with noisy, partial or even floor Cranfield data, while PointNetLK is limited to noisy Cranfield data. When an even more complex dataset is considered, RPMNet and ROPNet are more viable options to achieve training convergence. The main difference between RPMNet and ROPNet lies in their performance. When a low number of failure cases is required, with higher accuracies, RPMnet should be chosen. ROPNet has a limited advantage when timing is critical over RPMNet.

When the complexity of the training dataset is limited, all methods are still open. In this case, the next question is related to generalizability. PointNetLK achieves good results over almost all objects, and is thus the method of choice when this is an important factor. If not, the number of failure cases is again considered. ROPNet leads in general to a high number of failure cases, meaning it succeeds only for a limited number of poses of the same object. RPMNet and PointNetLK achieve a lower number of failure cases. Between these two, PointNetLK leads to the highest precisions, as also shown in Table 7.3.

For the three methods, the voxel size should still be chosen. For PointNetLK, the voxel size has a limited effect as mentioned in Section 7.1. For RPMNet and ROPNet, as is the case for RANSAC and FGR, the voxel size has a more significant effect. However, when dealing with a large number of objects, an average, general value can be chosen.

Finally, as with the global registration methods, ICP refinement improves the errors, as is also shown in Table 7.3.

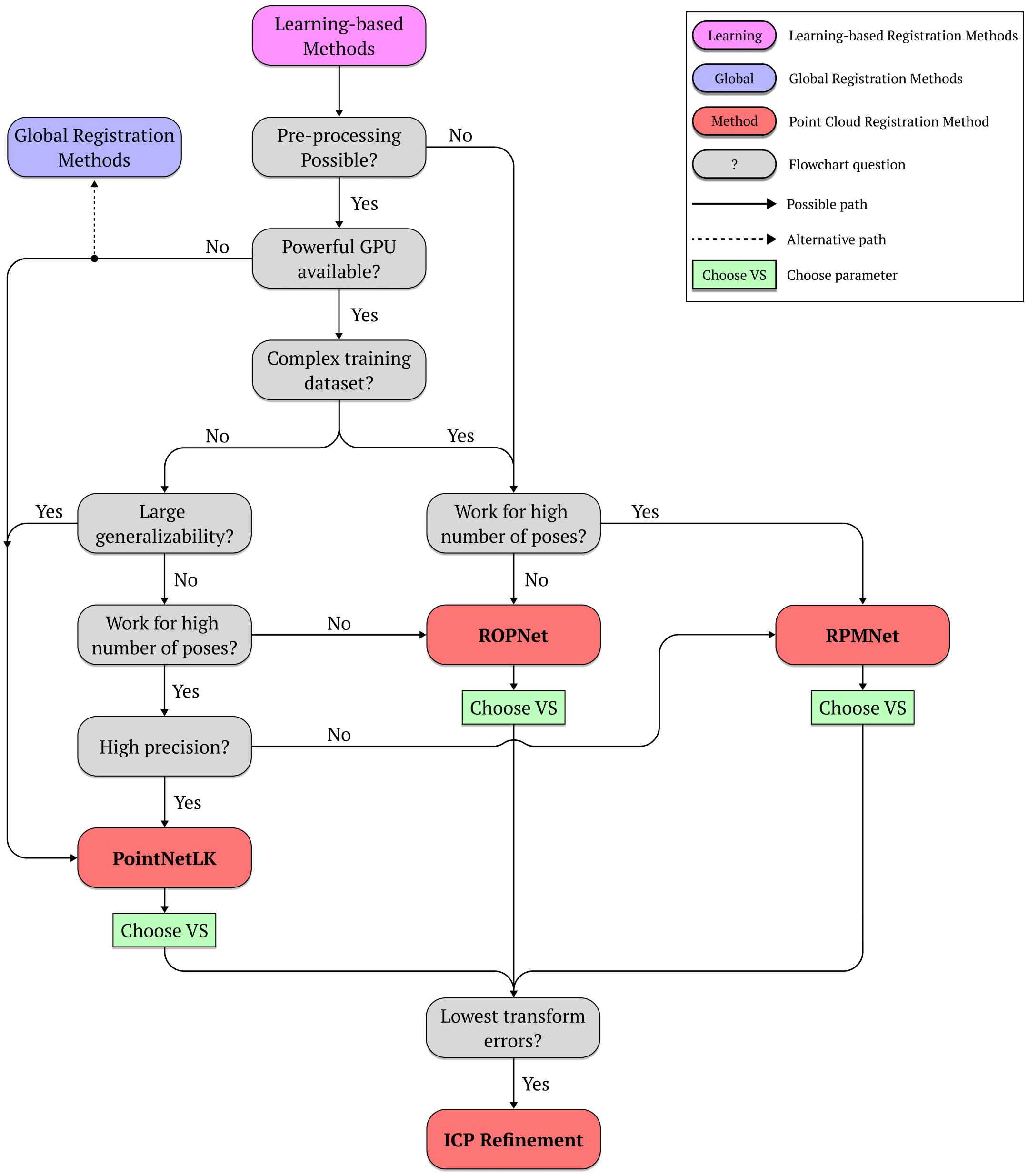


Figure 7.6: Flowchart showing the different questions to be asked when choosing one of the considered learning-based registration methods, PointNetLK, RPMNet or ROPNet. A legend is shown in the top right corner. The gray boxes indicate the question to be asked and full arrows the path to follow. The red boxes show the registration methods. The green boxes indicate the parameters to tune. Dotted lines indicate alternative paths, where the global methods could be considered instead.

Chapter 8

Future Work

In this final chapter, some possible improvements and paths for future work are discussed. The chapter is divided into different sections, related to the different steps of the thesis, which could be improved. Sections 8.1-8.2 are focused on enhancements of the preparation steps, before applying the registration methods. Section 8.3 deals with other registration techniques that can be tested. Finally, Section 8.4 discusses the training models for the learning-based methods. A schematic overview is given in Figure 8.1.

8.1 Point Cloud Capturing

During the experiments, the Intel Realsense D435i camera is used to capture the point clouds. Other cameras, such as the D405 camera could be considered, since it provides a better accuracy for smaller objects. In this report, the used objects are part of typical benchmarks and datasets of computer vision and assembly operations. These objects are typically medium sized. The same experiments can be redone for bigger objects or complete scenes and much smaller objects, such as bolts or screws. Finally, the scanned parts are all created using 3D printing. The influence of different surfaces could also be investigated in future work.

The camera is placed at a fixed position and the object is placed at a known pose. To gather a more complete point cloud, different cameras could be used to capture different angles of the same object [43]. In this way, partiality can be reduced. The difficulty here lies in the synchronization of the different cameras and the additional costs. Alternatively, the point cloud can be increased by moving the camera around the object to achieve a similar result. This is then becomes a SLAM (Simultaneous Localization And Mapping) problem. An example of a possible implementation, using ICP, is given by Newcombe, R. A. et al. [31, 28]. Another example of 3D reconstruction is shown by Sungjoon, C. [40].

8.2 Pre-processing

Another step that can be improved is the point cloud pre-processing. In this thesis, the point clouds are cleaned up by removing the background information, such as the floor and by applying some basic denoising algorithms based on the removal of statistical outliers. More advanced processing techniques consist of filtering the point cloud based on a template. This is called guided image filtering [19, 57]. Other filtering approaches are summarised by Han, X. et al. [18]. An alternative approach to filtering is the reconstruction of the point clouds, to also reduce partiality. TopNet [42] or surface reconstruction [4] could be used as part of point cloud completion. This also serves as an alternative to using multiple cameras or moving the camera around the object, as discussed in Section 8.1.

Another area of improvement is related to the ground truth estimation. During the experiments, the objects are oriented to a known pose and using an estimation of the normal vectors of the table, are aligned with their templates. However, some corrections need to be made to ensure a good alignment. Since there is no $1:1$ correspondence between the points and due to some deformations of the captured point cloud, it is difficult to define an exact ground truth definition. Another approach could be to use a marker based system, to determine the pose of the camera in the scene. Alternatively, other registration methods can be used as a reference during the experiments. However, since the goal of this thesis is to compare the different methods, this approach was not considered here.

Finally, the problem of object detection can also be considered. Combined with registration, this would allow for the system to automatically select the correct template model. The detection can be performed on the 2D captured image, using Yolo [35]. An alternative would be to apply the detection on the 3D point cloud directly [61].

8.3 Point Cloud Registration Techniques

In this thesis, six methods were mainly tested, namely RANSAC, FGR, GO-ICP, PointNetLK, RPMNet and ROPNet. Other algorithms exist, both in the space of global registration methods, as well as learning-based methods. One example, also applicable to real-world data, is DeepPRO, by Donghoo, L. [25]. Many other methods exists, a more extensive list of other methods can be found in Appendix A or the Github page created by XuyangBai.

8.4 Training Datasets

Finally, as mentioned before, a more complex training dataset can be considered, to better capture the real world scans. Instead of simply removing points or adding partiality, more complex operations can be looked into. An example is given by Jiwan, K. [23]. Alternatively, the algorithms can be trained on real captured data instead of synthetically created datasets. However, this would require very large datasets to be created, for each set of objects, for which the method needs to work.

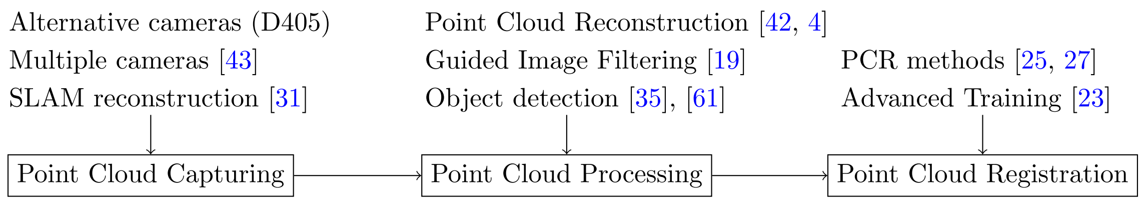


Figure 8.1: Overview of the different areas of future work possibilities, based on the main steps of this thesis.

Chapter 9

Conclusion

In this thesis, different Point Cloud Registration methods were compared in the context of extended reality environments, to speed up the authoring process of the virtual assets. Three main research questions are answered: (1) *Which algorithm, for achieving object alignment, works best, in terms of some defined metrics, specifically in an industrial setting?*, (2) *For the learning-based methods, is it sufficient to train the algorithms using the available CAD models?* and (3) *How accurate does the captured depth image need to be to achieve satisfying results?*.

To answer these questions, the literature was studied to determine a set of registration methods, that could be applied to real-world point cloud scans. The list of algorithms includes RANSAC, FGR, GO-ICP, PointNetLK, RPMNet and ROPNet. Metrics were also taken from the literature to study the performance of these methods. Relative and absolute errors, recall, number of failure cases and required computation times are the main ones used. The point clouds are created using the Intel Realsense D435i camera. This device is able to create accurate scans, for medium sized, 3D printed objects in a well lit environment. The point clouds are processed to remove outliers and deduce the ground truth transformation.

The learning-based methods, PointNetLK, RPMNet, PRNet and ROPNet, were trained on different datasets. The training data was generated based on the Cranfield and ModelNet40 benchmarks. Different cases were considered, where noise, partiality (removed points) or even floor information (added points to mimic a table or floor) was added to the point clouds. PRNet proved to be unstable during training. RPMNet and ROPNet could easily be trained on most cases, while PointNetLK converged less easily.

Experiments were setup, where parameters, such as the voxel size, bounding box, MSE threshold and training models were varied. Varying these variables allowed to determine the achievable performance for each method and to formulate an answer to the main research question (1). In the end, GO-ICP was found to produce the smallest errors, while requiring the most computation time and being more sensitive to outlier points. RANSAC, PointNetLK and RPMNet achieved similar results, reaching a performance close to GO-ICP after applying ICP refinement, but for a shorter computation time. FGR and ROPNet proved to be the quickest methods, yielding also the largest errors. Thus the method to use depends on the application and requirements.

The learning-based methods achieved overall a performance similar to the global methods, when trained on the available CAD models. The answer to research question (2) is thus that the learning-based methods can be trained on the synthetic dataset, yielding already satisfying results. However, these methods also require a larger setup effort due to the training process. For capturing the point clouds, different guidelines were presented as an answer to research question (3). To achieve best performances, information from the object should be maximized, while environmental clutter should be minimized. Ideally, the point cloud is cleaned up from any background information such as floors or tables, since this decreases the performance significantly. In the end, a framework was presented, which allows the reader to choose a registration method, based on required specifications.

Finally, future work is possible on all steps of the thesis. Different or multiple scanning cameras could be considered. Processing and training can be more advanced with, for example, point cloud reconstruction or guided image filtering. Other registration methods, such as DeepPRO can also be considered and compared to the results presented in this thesis.

Bibliography

[1] Sepehr Alizadehsalehi. “BIM/Digital Twin-Based Construction Progress Monitoring through Reality Capture to Extended Reality (DRX)”. In: (), p. 212.

[2] Yasuhiro Aoki et al. “PointNetLK: Robust & Efficient Point Cloud Registration Using PointNet”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Long Beach, CA, USA: IEEE, June 2019, pp. 7156–7165. ISBN: 978-1-72813-293-8. DOI: 10.1109/CVPR.2019.00733. (Visited on 09/05/2022).

[3] Shaun Bangay et al. “Building the Second Generation of Parallel/Distributed Virtual Reality Systems”. In: *Parallel Computing* 23.7 (July 1997), pp. 991–1000. ISSN: 01678191. DOI: 10.1016/S0167-8191(97)00040-9. (Visited on 10/27/2022).

[4] Matthew Berger et al. “A Survey of Surface Reconstruction from Point Clouds”. In: *Computer Graphics Forum* 36.1 (Jan. 2017), pp. 301–329. ISSN: 0167-7055, 1467-8659. DOI: 10.1111/cgf.12802. (Visited on 04/30/2023).

[5] Bhaskar Bhattacharya and Eliot H. Winer. “Augmented Reality via Expert Demonstration Authoring (AREDA)”. In: *Computers in Industry* 105 (Feb. 2019), pp. 61–79. ISSN: 01663615. DOI: 10.1016/j.compind.2018.04.021. (Visited on 10/19/2022).

[6] Eric Brachmann. *6D Object Pose Estimation Using 3D Object Coordinates [Data]*. 2020. DOI: 10.11588/DATA/V4MUMX. (Visited on 10/27/2022).

[7] Christopher Choy, Wei Dong, and Vladlen Koltun. “Deep Global Registration”. In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Seattle, WA, USA: IEEE, June 2020, pp. 2511–2520. ISBN: 978-1-72817-168-5. DOI: 10.1109/CVPR42600.2020.00259. (Visited on 02/24/2023).

[8] Matei Ciocarlie et al. “Towards Reliable Grasping and Manipulation in Household Environments”. In: *Experimental Robotics*. Ed. by Oussama Khatib, Vijay Kumar, and Gaurav Sukhatme. Vol. 79. Berlin, Heidelberg: Springer Berlin Heidelberg, 2014, pp. 241–252. ISBN: 978-3-642-28571-4 978-3-642-28572-1. DOI: 10.1007/978-3-642-28572-1_17. (Visited on 10/03/2022).

[9] K. Collins, A. J. Palmer, and K. Rathmill. “The Development of a European Benchmark for the Comparison of Assembly Robot Programming Systems”. In: *Robot Technology and Applications*. Ed. by Keith Rathmill et al. Berlin, Heidelberg: Springer, 1985, pp. 187–199. ISBN: 978-3-662-02440-9. DOI: 10.1007/978-3-662-02440-9_18.

[10] *Compare Intel RealSense Depth Cameras (Tech Specs and Review)*. https://www.intelrealsense.com/compare-depth-cameras/. (Visited on 02/19/2023).

[11] *Computer Science*. https://www.ru.ac.za/computerscience/. Aug. 2011. (Visited on 10/27/2022).

[12] Sanika Doolani et al. “A Review of Extended Reality (XR) Technologies for Manufacturing Training”. In: *Technologies* 8.4 (Dec. 2020), p. 77. ISSN: 2227-7080. DOI: 10.3390/technologies8040077. (Visited on 10/21/2022).

[13] Martin A. Fischler and Robert C. Bolles. “Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Carthography.Pdf”. In: (June 1981), p. 15.

[14] Emmanuel Frécon and Mårten Stenius. “DIVE: A Scaleable Network Architecture for Distributed Virtual Environments”.

In: *Distributed Systems Engineering* 5.3 (Sept. 1998), pp. 91–100. ISSN: 0967-1846, 1361-6390. DOI: 10.1088/0967-1846/5/3/002. (Visited on 10/27/2022).

[15] *Global Registration — Open3D 0.15.1 Documentation.* http://www.open3d.org/docs/release/tutorial/pipelines/global_registration.html. (Visited on 10/27/2022).

[16] *Google Colab.* https://research.google.com/colaboratory/faq.html. (Visited on 02/28/2023).

[17] Xuejing Gu, Xu Wang, and Yucheng Guo. "A Review of Research on Point Cloud Registration Methods". In: *Materials Science and Engineering* (2019), p. 10.

[18] Xian-Feng Han et al. "A Review of Algorithms for Filtering the 3D Point Cloud". In: *Signal Processing: Image Communication* 57 (Sept. 2017), pp. 103–112. ISSN: 09235965. DOI: 10.1016/j.image.2017.05.009. (Visited on 04/30/2023).

[19] Kaiming He, Jian Sun, and Xiaoou Tang. "Guided Image Filtering". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.6 (June 2013), pp. 1397–1409. ISSN: 0162-8828, 2160-9292. DOI: 10.1109/TPAMI.2012.213. (Visited on 04/30/2023).

[20] Zayd Hendricks, Gary Marsden, and Edwin Blake. "A Meta-Authoring Tool for Specifying Interactions in Virtual Reality Environments". In: (), p. 10.

[21] "Iterative Closest Point". In: *Wikipedia* (June 2022). (Visited on 09/05/2022).

[22] Bojan Jerbić et al. "Robot Assisted 3D Point Cloud Object Registration". In: *Procedia Engineering* 100 (2015), pp. 847–852. ISSN: 18777058. DOI: 10.1016/j.proeng.2015.01.440. (Visited on 10/03/2022).

[23] Jiwan Kim et al. "Accurate Depth Image Generation via Overfit Training of Point Cloud Registration Using Local Frame Sets". In: *Computer Vision and Image Understanding* 226 (Jan. 2023), p. 103588. ISSN: 10773142. DOI: 10.1016/j.cviu.2022.103588. (Visited on 04/30/2023).

[24] Michael Laielli et al. "LabelAR: A Spatial Guidance Interface for Fast Computer Vision Image Collection". In: *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology.* New Orleans LA USA: ACM, Oct. 2019, pp. 987–998. ISBN: 978-1-4503-6816-2. DOI: 10.1145/3332165.3347927. (Visited on 10/22/2022).

[25] Donghoon Lee et al. "DeepPRO: Deep Partial Point Cloud Registration of Objects". In: *2021 IEEE/CVF International Conference on Computer Vision (ICCV).* Montreal, QC, Canada: IEEE, Oct. 2021, pp. 5663–5672. ISBN: 978-1-66542-812-5. DOI: 10.1109/ICCV48922.2021.00563. (Visited on 10/03/2022).

[26] Yongjae Lee, Byounghyun Yoo, and Soo-Hong Lee. "Sharing Ambient Objects Using Real-time Point Cloud Streaming in Web-based XR Remote Collaboration". In: *The 26th International Conference on 3D Web Technology.* Pisa Italy: ACM, Nov. 2021, pp. 1–9. ISBN: 978-1-4503-9095-8. DOI: 10.1145/3485444.3487642. (Visited on 10/21/2022).

[27] Leihui Li, Riwei Wang, and Xuping Zhang. "A Tutorial Review on Point Cloud Registrations: Principle, Classification, Comparison, and Technology Challenges". In: *Mathematical Problems in Engineering* 2021 (July 2021). Ed. by Paolo Spagnolo, pp. 1–32. ISSN: 1563-5147, 1024-123X. DOI: 10.1155/2021/9953910. (Visited on 09/06/2022).

[28] *Librealsense/Wrappers/Opencv/Kinfu at Master · IntelRealSense/Librealsense.* https://github.com/IntelRealSense/librealsense. (Visited on 04/29/2023).

[29] Shuntao Liu et al. "A Depth-Based Weighted Point Cloud Registration for Indoor Scene". In: *Sensors* 18.11 (Oct. 2018), p. 3608. ISSN: 1424-8220. DOI: 10.3390/s18113608. (Visited on 10/21/2022).

[30] Bilawal Mahmood and SangUk Han. "3D Registration of Indoor Point Clouds for Augmented Reality". In: (), p. 8.

[31] Richard A Newcombe et al. "KinectFusion: Real-Time Dense Surface Mapping and Tracking". In: ().

[32] Susanna Nilsson and Björn Johansson. "Acceptance of Augmented Reality Instructions in a Real Work Setting". In: *CHI '08 Extended Abstracts on Human Factors in Computing Systems.* Florence Italy: ACM, Apr. 2008, pp. 2025–2032. ISBN: 978-1-60558-012-8. DOI: 10.1145/1358628.1358633. (Visited on 10/22/2022).

[33] Arnaud Prouzeau et al. "Corsican Twin: Authoring In Situ Augmented Reality Visualisations in Virtual Reality". In: (2020), p. 9.

[34] #0K Srinivasan Ramachandran. *NOTE:* Feb. 2023. (Visited on 02/17/2023).

[35] Joseph Redmon et al. *You Only Look Once: Unified, Real-Time Object Detection.* May 2016. arXiv: 1506.02640 [cs]. (Visited on 11/04/2022).

[36] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. "Fast Point Feature Histograms (FPFH) for 3D Registration". In: *2009 IEEE International Conference on Robotics and Automation.* Kobe: IEEE, May 2009, pp. 3212–3217. ISBN: 978-1-4244-2788-8. DOI: 10.1109/ROBOT.2009.5152473. (Visited on 10/27/2022).

[37] vinit sarode. *Learning3D: A Modern Library for Deep Learning on 3D Point Clouds Data.* May 2020. (Visited on 11/04/2022).

[38] Vinit Sarode et al. *PCRNet: Point Cloud Registration Network Using PointNet Encoding.* Nov. 2019. arXiv: 1908.07906 [cs]. (Visited on 09/23/2022).

[39] Aleksandr Segal, Dirk Hähnel, and Sebastian Thrun. "Generalized-ICP". In: *Proc. of Robotics: Science and Systems.* June 2009. DOI: 10.15607/RSS.2009.V.021.

[40] Sungjoon Choi, Qian-Yi Zhou, and Vladlen Koltun. "Robust Reconstruction of Indoor Scenes". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* Boston, MA, USA: IEEE, June 2015, pp. 5556–5565. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7299195. (Visited on 10/27/2022).

[41] Vladimir Tadic et al. "Application of Intel RealSense Cameras for Depth Image Generation in Robotics". In: 18 (2019).

[42] Lyne P. Tchapmi et al. "TopNet: Structural Point Cloud Decoder". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).* June 2019, pp. 383–392. DOI: 10.1109/CVPR.2019.00047.

[43] T. Teo. "VIDEO-BASED POINT CLOUD GENERATION USING MULTIPLE ACTION CAMERAS". In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XL-4/W5 (May 2015), pp. 55–60. ISSN: 2194-9034. DOI: 10.5194/isprsarchives-XL-4-W5-55-2015. (Visited on 09/12/2022).

[44] Dipl-inform Tramberend et al. "Avocado: A Distributed Virtual Environment Framework". In: (May 2003).

[45] Rebekka Volk, Julian Stengel, and Frank Schultmann. "Building Information Modeling (BIM) for Existing Buildings — Literature Review and Future Needs". In: *Automation in Construction* 38 (Mar. 2014), pp. 109–127. ISSN: 09265805. DOI: 10.1016/j.autcon.2013.10.023. (Visited on 10/22/2022).

[46] Qian Wang, Yi Tan, and Zhongya Mei. "Computational Methods of Acquisition and Processing of 3D Point Cloud Data for Construction Applications". In: *Archives of Computational Methods in Engineering* 27.2 (Apr. 2020), pp. 479–499. ISSN: 1134-3060, 1886-1784. DOI: 10.1007/s11831-019-09320-4. (Visited on 09/12/2022).

[47] Tianyi Wang et al. "CAPturAR: An Augmented Reality Tool for Authoring Human-Involved Context-Aware Applications". In: *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology.* Virtual Event USA: ACM, Oct. 2020, pp. 328–341. ISBN: 978-1-4503-7514-6. DOI: 10.1145/3379337.3415815. (Visited on 10/19/2022).

[48] Yue Wang and Justin M. Solomon. *Deep Closest Point: Learning Representations for Point Cloud Registration.* May 2019. arXiv: 1905.03304 [cs]. (Visited on 09/08/2022).

[49] Yue Wang and Justin M. Solomon. *PRNet: Self-Supervised Learning for Partial-to-Partial Registration.* Oct. 2019. arXiv: 1910.12240 [cs, stat]. (Visited on 09/08/2022).

[50] Wanshun Wong. *What Is Gumbel-Softmax?* https://towardsdatascience.com/what-is-gumbel-softmax-7f6d9cdcb90e. May 2020. (Visited on 09/23/2022).

[51] Mike Wozniewski and Paul Warne. "Towards in Situ Authoring of Augmented Reality Content". In: (2011), p. 4. (Visited on 10/26/2022).

[52] XuyangBai. *Awesome-Point-Cloud-Registration.* Oct. 2022. (Visited on 10/23/2022).

[53] Jiaolong Yang et al. “Go-ICP: A Globally Optimal Solution to 3D ICP Point-Set Registration”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 38.11 (Nov. 2016), pp. 2241–2254. ISSN: 0162-8828, 2160-9292. DOI: 10.1109/TPAMI.2015.2513405. arXiv: 1605.03344 [cs]. (Visited on 09/21/2022).

[54] Zi Jian Yew and Gim Hee Lee. *RPM-Net: Robust Point Matching Using Learned Features.* Mar. 2020. DOI: 10.1109/CVPR42600.2020.01184. arXiv: 2003.13479 [cs]. (Visited on 09/08/2022).

[55] Wentao Yuan et al. *DeepGMR: Learning Latent Gaussian Mixture Models for Registration.* Aug. 2020. arXiv: 2008.09088 [cs]. (Visited on 09/08/2022).

[56] Zhiyuan Zhang, Yuchao Dai, and Jiadai Sun. “Deep Learning Based Point Cloud Registration: An Overview”. In: *Virtual Reality & Intelligent Hardware* 2.3 (June 2020), pp. 222–246. ISSN: 20965796. DOI: 10.1016/j.vrih.2020.05.002. (Visited on 09/06/2022).

[57] Jerry Zhao. *Point Cloud Denoise.* Mar. 2023. (Visited on 04/30/2023).

[58] Zhirong Wu et al. “3D ShapeNets: A Deep Representation for Volumetric Shapes”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).* Boston, MA, USA: IEEE, June 2015, pp. 1912–1920. ISBN: 978-1-4673-6964-0. DOI: 10.1109/CVPR.2015.7298801. (Visited on 02/18/2023).

[59] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. “Fast Global Registration”. In: *Computer Vision – ECCV 2016.* Ed. by Bastian Leibe et al. Vol. 9906. Cham: Springer International Publishing, 2016, pp. 766–782. ISBN: 978-3-319-46474-9 978-3-319-46475-6. DOI: 10.1007/978-3-319-46475-6_47. (Visited on 10/27/2022).

[60] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. “Open3D: A Modern Library for 3D Data Processing”. In: (), p. 6.

[61] Yin Zhou and Oncel Tuzel. “VoxelNet: End-to-End Learning for Point Cloud Based 3D Object Detection”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition.* Salt Lake City, UT, USA: IEEE, June 2018, pp. 4490–4499. ISBN: 978-1-5386-6420-9. DOI: 10.1109/CVPR.2018.00472. (Visited on 04/30/2023).

[62] Lifa Zhu et al. *Deep Models with Fusion Strategies for MVP Point Cloud Registration.* Oct. 2021. arXiv: 2110.09129 [cs]. (Visited on 11/04/2022).

[63] Lifa Zhu et al. *Point Cloud Registration Using Representative Overlapping Points.* July 2021. arXiv: 2107.02583 [cs]. (Visited on 10/10/2022).

Appendix A

Point Cloud Registration Techniques

A.1 Introduction

In this chapter, the main Point Cloud Registration techniques are discussed. The methods are divided into ICP-based, Learning-based and Probabilistic methods[1]. Other techniques, belonging to different categories are discussed at the end. An overview is given in Table A.1.

First, the problem of Point Cloud Registration is illustrated. After, the basic idea behind each technique is explained. Their advantages and disadvantages as well as their main application domains are also discussed. A summary is given in Tables A.2-A.3.

A.2 Problem statement

Point Cloud Registration is the problem of aligning two point clouds with each other by means of a (non-)rigid transformation. As is explained by Wang, Y. et al. [49] in mathematical terms, if X and Y represent two point clouds, which means they contain a set of points in the euclidean space $\mathbb{R}^3$. Usually, the original point cloud Y is called the template, while the altered point cloud X is called the source. Assuming both clouds are linked with a rigid transformation, thus with a rotation matrix $\mathbf{R}$ and a translation vector $\vec{t}$. The problem of Point Cloud Registration consists of finding the rigid transformation $[\mathbf{R}, \vec{t}]$ minimizing

$$E = \frac{1}{N}\sum_{i}^{N} \|\mathbf{R}\vec{x_i} + \vec{t} - \vec{y}_{m(x_i)}\|^2. \tag{A.1}$$

To simplify the notation, it is assumed both X and Y contain the same number of data points N. However, in practice, this is often not the case, as the data captured from a 3D camera may have a different number of points and at different locations. The transformation is also assumed to be rigid, meaning X is created from Y by applying only a rotation $\mathbf{R}$ and translation $\vec{t}$ to its points. In the case of a non-rigid transformation, shearing and scaling are also applied [27].

In Eq. A.1 $m(x_i)$ represents a mapping function, which links each point of point cloud X to its corresponding point in cloud Y. This is further explained in A.3. Different methods, such as DCP and ICP, define this function in other ways.

A.3 ICP-based methods

Iterative Closest Point (ICP) by Besl & McKay [2] is considered to be a simple but groundbreaking Point Cloud Registration method. It is an iterative and local method. There are two main steps in the ICP algorithm, which are repeated until the error metric reaches a sufficiently low value [17]:

[1]Some techniques belong to more than one class of methods. For such cases, the class making up the bulk of the technique is chosen.

1. Establish correspondences between point clouds based on geometric information.
2. Solve a least-squares problem in order to find the transformation parameters.

These steps can be further divided into different phases [27] [56], including:

1. **(Key) Point Selection** Reduce the number of data points by selecting a local descriptor based on distance or in a uniform or random manner.
2. **Matching** Estimating correspondences using the euclidean distance or other relevant information (i.e. intensity, visual information).
3. **Point Rejection** Identifying which correspondences are invalid and removing them. This is done based on the euclidean distance, a percentage threshold, RANSAC-based or using other metrics.
4. **Error metric establishment & minimization** Different metrics (point-to-point, point-to-plane) are used together with several minimization algorithms such as the Levenberg-Marquardt algorithm.

As mentioned in A.2, the mapping function $m(x_i)$ takes on a different form depending on the method. For ICP, $m(x_i)$ is defined as

$$m(x_i, Y) = \underset{j}{\operatorname{argmin}} \|\mathbf{R}\vec{x_i} + \vec{t} - \vec{y_j}\|. \tag{A.2}$$

Thus, $m(x_i, Y)$ tries to find the closest point in cloud Y to each point $x_i \in X$. As mentioned by Wang, Y et al. [49] this leads to a "chicken-and-the-egg" problem. To solve Eq. A.1, the correspondences must first be known. Vice-versa, to find the correspondences, the rigid transformation $[\mathbf{R}, \vec{t}]$ must be known. For this reason, ICP tries to iteratively find a mapping $m^k(x_i, Y)$ and the resulting transformation $[\mathbf{R}, \vec{t}]^k$ at each step k.

ICP is a simple and efficient method, that yields the best results with small transformations [55]. On the other hand, the method is sensitive to outliers, noise and initialization. This can cause the solution to converge to a local minimum. The problem scales quadratically and integration of deep learning is also non-trivial [2]. For this reason, ICP is applied after using a global method, which can be learning- or probabilistic-based, as a final refinement step [27].

In order to solve the problems of ICP, many variants exist such as GO-ICP and Generalized-ICP. Other methods such as DCP are based on the same ideas, but use a learning approach instead.

A.3.1 GO-ICP

Globally Optimal ICP (GO-ICP) builds further onto the ICP algorithm but uses a Branch-and-Bound (BnB) scheme to avoid falling into local minima [27]. It is therefore considered a global method, while ICP is a local method. The BnB scheme searches the entire rigid transformation space to find a globally optimal solution for $[\mathbf{R}, \vec{t}]$. This can be compared to searching along the branches of a tree structure to find an increasingly more accurate solution. One of the difficulties is defining the boundaries in which the solution is expected to be found [53]. Bnb is used to solve non-convex problems, such as ICP or NP-hard problems. For GO-ICP, it guides the solution from ICP along the local minima until it finds the global optimum [53]. An in-depth explanation of the GO-ICP algorithm is given by Jialong, Y. et al. [53].

The main advantages of GO-ICP are the robustness to noise and outliers, which is also confirmed by the experiments of Wang, Y. et al. [48] applied on noisy ModelNet40 data. Because of the global estimation, GO-ICP does not fall into local-minima [27]. However, because of the global approach, GO-ICP has a much longer computation time. This makes it unusable for real-time applications. GO-ICP is more fitted for situations in which an accurate solution is requested or when a good initialization is not available [53].

A.3.2 Generalized-ICP

Generalized-ICP is another ICP variant which uses a different approach for the minimization step of Eq. A.1. Generalized-ICP assumes the points of X and Y to be subjected to a Gaussian distribution. This leads to the introduction of covariance matrices in Eq. A.1. This method is thus considered to be a probabilistic approach [39]. Furthermore, it also includes the point-to-plane variant of ICP. This means

Eq. A.1 is changed to

$$E = \frac{1}{N} \sum_{i}^{N} \| \vec{n_i} \cdot (\mathbf{R}\vec{x_i} + \vec{t} - \vec{y}_{m(x_i)}) \|^2 \tag{A.3}$$

to include the information of the surface normal $\vec{n_i}$ at the point $\vec{y_i}$ [39].

The advantages of Generalized-ICP include better resistance to incorrect correspondences and a more accurate result, while still maintaining the simplicity and speed of ICP [39].

A.4 Learning-based methods

Another class of Point Cloud Registration methods is the class of Learning-based methods. These techniques involve Machine-learning and training a network, using a loss function, to recognize features and eventually align the point clouds.

These methods can be divided into two classes: correspondence-free and correspondence-based methods. Correspondence-free methods use only a global description of the point clouds to align them. Important examples include PointNetLK and PCRNet, which are both based on PointNet. Correspondence-based methods use a framework inspired by traditional methods such as ICP and use local information of the point clouds. They try to find explicit correspondences between the two point clouds [56].

In general, Learning-based methods are more robust and faster. They can also be used for other means such as segmentation or classification. Since real-world data often includes noise and outliers, Learning-based methods provide a good solution there. The main problem of these methods is to identify invariant features that can be trained [27].

A.4.1 PointNet

PointNet is not an actual Point Cloud Registration technique. Instead, it is a deep learning approach to generate descriptors for each point in the point cloud or a global descriptor for the entire cloud. The network used is called PoitnNet-Cla [56]. However, the results of PointNet are used to apply Point Cloud Registration based on the generated descriptors. The network generates a resulting vector of a size irrespective of the number of samples of the point cloud [2]. PointNet itself is also used for other tasks such as classification or segmentation [2].

PointNet++ is another approach to extracting local features from the point cloud. It is also based on PointNet [56].

The two techniques mentioned below are correspondence-free methods. These methods are very straightforward with their performance depending greatly on the quality of the extracted descriptors. Zhang, Z. et al. [56] note that ”the performance is ideal when the point clouds are the same, except for the pose. However, when there is a large difference between the two input point clouds, the performance depends greatly on the feature extractor network.”

PointNetLK

PointNetLK is a Point Cloud Registration method based on PointNet. The idea behind this technique is to use PointNet to achieve a fixed dimensional vector describing the point clouds. A schematic overview is given below.

$$\begin{tikzcd}
\boxed{\forall \vec{x_i}\ (i = 1, ...N) \in X \subset \mathbb{R}^3} \arrow[r, "\text{MLP}"] \arrow[rr, bend right, "\phi : \mathbb{R}^{3xN} \to \mathbb{R}^K"'] & \boxed{\psi(\vec{x_i}) \in \mathbb{R}^K} \arrow[r, "\text{pooling function}"] & \boxed{\vec{G} = \phi(X) \in \mathbb{R}^K}
\end{tikzcd}$$

In the network, PointNet applies a Multi-layer Perceptron (MLP) to each point of X to create a K-dimensional vector for every point. Afterwards, a pooling function combines this information into a single global vector $\vec{G}$. The pooling function is either a function extracting the maximum or the average.

According to Aoki, Y. et al. [2], an average pooling function yields better results when working with noisy data. The goal is then to find the rigid body transformation T such that

$$\phi(Y) = \phi(T \cdot X) \tag{A.4}$$

or in other words, such that the global descriptors of the original and transformed point cloud match. For this, an interactive algorithm is used similar to the Lucas-Kanade algorithm. The rigid body transformation is then assumed to be

$$T = exp(\sum_i \xi_i A_i). \tag{A.5}$$

A_i are called the generators of the exponential map and ξ_i $(i = 1...6)$ (twist parameters) are found by training the network [2].

According to Aoki, Y. et al. [2], PointNetLK achieves very high performance, and robustness against noise and can even achieve good results on untrained classes of data[2]. Furthermore, it can also be used to align 2.5D data, which is often available in real-time applications. Lastly, it is also insensitive to the initial configuration [27]. However, since PointNetLK is an iterative, local technique, its performance drops for large transformations [55].

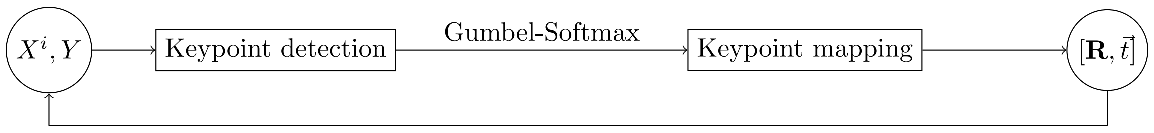
PCRNet

Point Cloud Registration Network (PCRNet) is another learning-based, correspondence-free method, which uses PointNet in a similar way as PointNetLK. However, when the global descriptor is found for both point clouds, they are concatenated and afterwards a data-driven technique is used to align the features instead of the Lucas-Kanade algorithm in PointNetLK [56]. PCRNet can be used as a single-shot method, meaning it does not need to iterate. This makes it especially interesting for high-speed applications. However, an iterative version of PCRNet also exists, which uses the result of the last iteration as an input for the network [38].

The experiments of Sarode, V. et al. [38] show that PCRNet is more robust to noise than PointNetLK and achieves convergence in only a few iterations. It can achieve similar accuracy to GO-ICP but is much faster. PointNetLK does show better generalizability.

PRNet

Partial Registration Network (PRNet) focuses on Point Cloud Registration for point clouds that only have partial overlap. This means they only share a part of their data. The goal is to fit the two point clouds together to achieve a complete picture of the object [49]. PRNet is an iterative, local method [55]. It is correspondence-free as it does not try to match every point of both clouds, but instead uses keypoints. The basic idea behind PRNet is explained by the graph below.



Iterate with: $\vec{x}^i = \mathbf{R^{i-1}}\vec{x}^{i-1} + \vec{t}^{i-1}$

As explained in an article by Wong, W. [50], Gumbel-Softmax is a way of making the function *argmax* differentiable. This means the mapping function m introduced in Section A.2, becomes differentiable, which is not the case for ICP. As a result, the correspondences in ICP are "sharp". However, when working with neural networks, back-propagation of the error is desirable. For this reason, the mapping function is based on Gumbel-Softmax, so point correspondences become sharper as the problem converges for a more accurate result. Wang, Y. et al. [49] gives a complete explanation.

As with other correspondence-free methods, the performance of PRNet drops for big transformations [55]. The method can also be slow [49].

[2]It must be noted however that the performance of learning-based methods is much lower when applied on unseen categories of data [55].

A.4.2 DCP

Deep Closest Point (DCP) is another global (correspondence-based) Point Cloud Registration technique which is based on ICP [49]. However, it takes a deep learning approach. DCP is a one-shot algorithm, that differs from ICP by its choice of the mapping function m (Section A.2) [49]. The method can be summarized in three steps, represented in the scheme below [48].

$$\begin{tikzcd}
X, Y \arrow[r, "\text{PointNet/DGCNN}"] & \boxed{\text{Common embedding}} \arrow[r, "\text{Pointer generation}"] & \boxed{\text{Matching}} \arrow[r, "\text{SVD}"] & {[\mathbf{R}, \vec{t}]}
\end{tikzcd}$$

The initial point clouds X and Y are first mapped onto a common high-dimensional space. This can be done using either PointNet or a graph-based neural network called DGCNN. From the experiments of Wang. Y. et al. [48], DGCNN provided the best results, as it also incorporates local geometry information. The resulting features are then used in a mapping function m, which is now defined as

$$m(x_i, Y) = softmax(\phi_Y \phi_{x_i}^T). \tag{A.6}$$

In Eq. A.6, ϕ_Y represents the embedding of Y and ϕ_{x_i} the i^{th} row of the embedding matrix ϕ_x. The equation for m yields a probabilistic vector for each point $x_i \in X$ in relation to Y [48].

DCP results in a reliable solution, even after a single pass. However, it can be improved by using either an iterative version or by applying ICP to the obtained result. It does not fall into local minima, as is the case for ICP and is also faster [48]. DCP is also insensitive to the initial configuration, achieving the same accuracy even for larger transformations [55].

The main disadvantages of DCP include its sensitivity to noise [55]. The performance discussed by Wang, Y. et al. [48] is mainly based on experiments with synthetic data. It is not ideal for partial-to-partial registration [49]. Since DCP is a learning-based method, it also performs less accurately on unseen data [55]. Lastly, the embedding dimension must also be chosen in advance [48].

A.4.3 RPM-Net

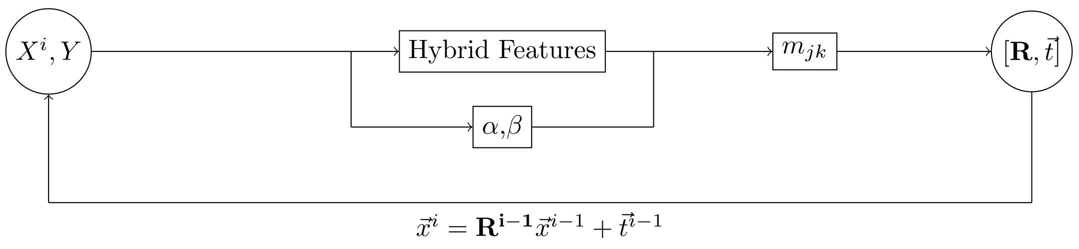
Robust Point Matching Network (RPM-Net) is a learning-based Point Cloud Registration approach. It is a combination of the iterative method RPM and deep learning. RPM itself is similar to ICP, but it uses a permutation matrix M to assign correspondences. As a result, Eq. A.1 turns into

$$\arg\min_{\mathbf{M},\mathbf{R},\vec{t}} \sum_{j=1}^{J} \sum_{k=1}^{K} m_{jk}(\|\mathbf{R}\vec{x_j} + \vec{t} - \vec{y_k}\|_2^2 - \alpha), \tag{A.7}$$

where the matrix M is initialized by

$$m_{jk} \leftarrow e^{-\beta(\|\mathbf{R}\vec{x_j} + \vec{t} - \vec{y_k}\|_2^2 - \alpha)}. \tag{A.8}$$

The higher the parameter β, the closer m_{jk} comes to either zero or one, such that $m_{jk} \in \{0, 1\}$. RPM-Net expands on RPM by using hybrid features instead of the spatial distances in Eq. A.7 and Eq. A.8. This allows the method to use additional information besides only the spatial information. Moreover, RPM-Net also updates the values for α and β iteratively. The algorithm is visualised below [54].


X^i, Y
Hybrid Features
m_{jk}
$[\mathbf{R}, \vec{t}]$
α,β
$\vec{x}^i = \mathbf{R}^{\mathbf{i-1}}\vec{x}^{i-1} + \vec{t}^{i-1}$

From the input clouds, the hybrid features are estimated as well as the parameters α and β. From this information, the permutation matrix M is computed, resulting in the rigid body transformation after applying SVD. The transformation is used to update the point cloud X, such that the method can work iteratively [54].

RPM-Net is robust to initialization and works well even when not all points have correspondence [54]. It is also robust to noise [27]. However, RPM-net is slower than ICP and one-shot DCP [54].

A.4.4 ROPNet

Representative Overlapping Points Network (ROPNet) by Lifa, Z. et al. [63] is a learning-based registration method, specifically designed for solving the partial-to-partial registration problem. While other techniques such as RPM-Net and PRNet solve this problem in an indirect way, ROPNet explicitly considers the fact that not all points in the template cloud have a corresponding point in the source cloud. By establishing an overlap score and finding which points are related, the remaining non-overlapping points can be removed when calculating the final estimation for the rigid transformation. In this way, the problem of partial-to-partial registration is shifted to partial-to-complete registration. The main structure of the algorithm is shown below.

$$\begin{tikzcd}
X, Y \arrow[r, "\text{CG}"] & \boxed{\text{Initial alignment \& Overlap}} \arrow[r, "\text{TFMR}"] & \boxed{\text{Representative Points}} \arrow[r, "\text{SVD}"] & {[\mathbf{R}, \vec{t}]}
\end{tikzcd}$$

The two initial point clouds X, Y are the input of the context-guided (CG) module, which finds the first set of overlapping points. After a first alignment, the Transformer-based feature matching removal (TFMR) module, removes the wrong correspondences and finally, using SVD, the transformation is computed [63].

Lifa, Z. et al. [63] report the very high performance of ROPNet, outperforming DeepGMR, RPMNet and RPNet on unseen categories and shapes of data as well as in its robustness to noise.

A.5 Probabilistic methods

A final set of techniques use statistics and probability computations to solve the Point Cloud Registration problem. The idea is to consider point clouds as density functions and optimize a statistical discrepancy [27]. The main advantages of using probabilistic methods include a better resilience to uncertainty in the data [48] and improved accuracy when dealing with partiality [49].

A.5.1 DeepGMR

Deep Gaussian Mixture Registration (DeepGMR) is a global probabilistic Point Cloud Registration method. The idea is to fit the point clouds to a Gaussian Mixture Model (GMM), which is a weighted sum of Gaussian distributions in the 3D space. The registration process then tries to minimize the divergence between the probabilistic distributions [55].

DeepGMR is robust to noise, generalizable and invariant to the initial pose. The solution is found in a single step, thus making DeepGMR a real-time technique [55].

A.6 Global Registration Methods

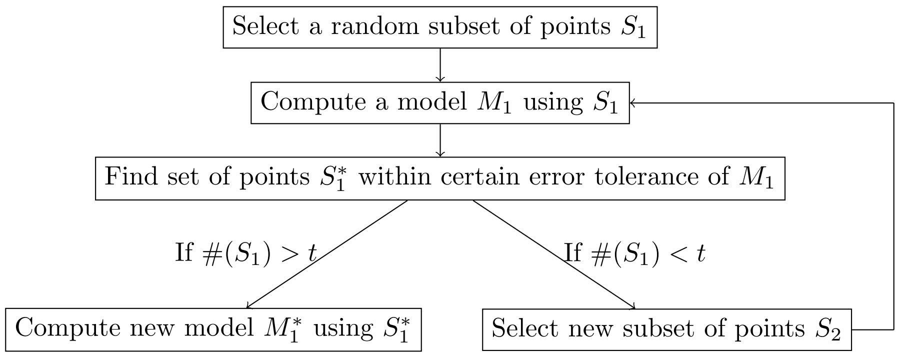
The following two methods, RANSAC and FGR, are considered global registration methods [15]. This refers to the fact that they do not require any initialization. However, the obtained results are more coarse. Further refinement is possible by applying a local method to the obtained results such as ICP, which yields more accurate results for small transformations.

Both methods discussed in this section are not learning-based. Instead, features are first extracted from the point cloud using Fast Point Feature Histograms (FPFH) [36]. The resulting features are then used as input for the algorithms. Rusu, R. et al. [36] explain the details of the FPFH algorithm. Essentially, for every point p inside of the point cloud, the k-nearest neighbours are selected and the geometrical relations between the point p and its neighbours are computed. These properties range from the 3D coordinates and normal estimations to curvatures. This results in pose-invariant local features representing the model properties at the point p.

A.6.1 RANSAC

RANdom SAmple Consensus (RANSAC) was first introduced by Fischler, M. et al. [13] in 1981. It is a paradigm for fitting a model to experimental data. This is another way of looking at the Point Cloud Registration problem, where the source contains the experimental data and the template is the model. The idea behind RANSAC is different from the typical least-squares algorithm. The latter tries to fit a model to all data points but has no method of dealing with gross errors[3]. Heuristics are used to remove these outliers by iteratively removing the most erroneous points until certain criteria are met.

The concept of RANSAC is the following:


Select a random subset of points S_1
Compute a model M_1 using S_1
Find set of points S_1^* within certain error tolerance of M_1
If $\#(S_1) > t$
If $\#(S_1) < t$
Compute new model M_1^* using S_1^*
Select new subset of points S_2

As is clear from the diagram, RANSAC tries to grow a set of points, which fit the model instead of trying to fit the model immediately to all data points. There are however some undefined parameters, for which different selection methods exist, such as the number of subsets (S_1) to try, the error tolerance and the threshold t.

RANSAC is specifically designed to deal with erroneous feature detectors. For this reason, it is often used when working with partial point clouds of environments which need to be combined to create a virtual map.

A.6.2 FGR

Fast Global Registration (FGR) is another global registration algorithm by Zhou, Q. et al. [59]. It does not involve model fitting or local refinement as it combines the coarse and fine registration steps into one. FGR also takes the features from the FPFH algorithm as input and does not recompute the correspondences during execution, allowing for a fast registration process. Correct and false correspondences are distinguished using a robust estimator $\rho(\dot{})$, which is called the scaled Geman-McClure estimator.

The main benefits of FGR are its robustness to noise and partiality, according to Zhou, Q. et al. [59]. The method is also fast compared to other global registration techniques and has a lower computational cost. It does not require any initialization but matches the accuracy of well-initialized local methods such as ICP. Several other papers [54, 63] do however mention the sensitivity of FGR to noise.

[3]This is the term used for outliers by Fischler, M. et al. [13]

Table A.1: Overview of the different Point Cloud Registration methods based on various criteria. Unless otherwise indicated, the information is taken from [27].

Classification of Point Cloud Registration techniques		
Types	Explanation	Examples
1. According to the number of inputs		
- pairwise registration	2 point clouds	
- multi-view registration	$>$ 2 point clouds	
2. Accuracy performance		
- coarse registration	- random initial position - rough transformation	RANSAC, FGR [15]
- fine registration	- points closer - more accurate results	ICP [17]
3. Object deformation		
- rigid registration	Object non-deformed (rotation, translation)	
- non-rigid registration	Object deformed (rotation, translation, shear, scaling)	CPD, FlowNet3D, PLFlowNet [56]
4. Iterative		
- Iterative	Multiple iterations	ICP, DCP
- Direct	1-step solution	PRNet
5. ICP-based methods	based on the ICP-method	ICP, GO-ICP, Generalized-ICP
6. Feature-based methods		
- global descriptors	encoding geometric information of whole point cloud	
- local descriptors	using local information	PFH, SHOT
- hybrid-based descriptors	local + global	
7. Learning-based methods	using machine learning	PointNet, PointNet++, DGR, DCP, PRNet, RPM-Net
- Correspondence-free method [17]	searching global features [17]	PointNetLK, PCRNet [17]
- Correspondence-based methods [17]	extract the per-point or per-patch embeddings of clouds to generate a mapping [17]	
8. Probabilistic methods	using statistical approach	Deep-GMR [55]

Table A.2: Overview of the main Point Cloud Registration methods with their pros and cons; summary of Appendix A (1/2).

Pros and cons of the main Point Cloud Registration methods (1/2)			
Name	Advantages	Disadvantages	Applications
ICP	- simple - efficient	- noise, outliers sensitive - initialization sensitive - non-trivial deep learning integration	- small transformations - refinement
GO-ICP	- Robust to noise, outliers - finds global optimum	- computationally slow	- initialization unavailable - accurate solution requested
Generalized-ICP	- robust to incorrect correspondences - simple, efficient - accurate		similar to ICP
PointNetLK	- robust - accurate - generalizable	- training required	- noisy data - 2.5D data
PCRNet	- robust to noise - fast	- less generalizable - training required	- high-speed applications - noisy data
PRNet	- best results for partial-to-partial registration	- slow - low accuracy for large transformations	- partial-to-partial registration
DCP	- one-shot (fast) - finds global optimum - invariant to the initial pose	- training required - choice of embedding dimension - noise sensitive - less generalizable	- larger transformations - high-speed applications
RPM-Net	- robust to noise - robust to initialization - works well even when not all points correspond	- slower than ICP	- large transformations
DeepGMR	- robust to noise - invariant to the initial pose - fast - generalizable	- training required	real-time applications
ROPNet	- robust to noise - generalizable - fast	- training required	- partial-to-partial registration - noisy data

Table A.3: Overview of the main Point Cloud Registration methods with their pros and cons; summary of Appendix A (2/2).

Pros and cons of the main Point Cloud Registration methods (2/2)			
Name	Advantages	Disadvantages	Applications
RANSAC	- robust to initialization - no training required - robust to erroneous features	- selection of parameters - feature detector required	- large transformations - partial-to-partial registration (environments)
FGR	- fast - no training required	- feature detector required - noise sensitive	- partial-to-partial registration (environments) - high-speed applications

Appendix B

Metrics

In order to evaluate and compare the different Point Cloud Registration methods, various metrics are used. This section gives an overview of the most common metrics and a selection for the experiments.

First, in Section B.1, metrics evaluating the accuracy of the rigid motion are discussed. Section B.2 explains the Feature Extraction Metrics, used for evaluating the results of the trained feature extractor. Finally, Section B.3 mentions some miscellaneous methods for evaluating the performance of the techniques.

B.1 Rigid Motion Estimation

The output of the Point Cloud Registration problem is the rigid transformation $[\mathbf{R}, \vec{t}]$ showing the relation between the source X and template Y point clouds. Metrics computed using the rotation matrix and translation vector are therefore chosen to assess the performance of the results.

B.1.1 Mean Relative Error (MRE)

The Mean Relative Error (MRE) is obtained by first calculating the difference between the ground truth and the obtained result and then taking the mean. It can be defined for the rotation as well as the translation, as is explained by Zhang, Z. et al. [56].

Mean Relative Angular Error (MRAE)

The Mean Relative Angular Error (MRAE, $^\circ$) is defined as

$$MRAE = mean\left(cos^{-1}\left(\frac{trace(\mathbf{R}_{pre}^T\mathbf{R}_{gt}) - 1}{2}\right)\right) \tag{B.1}$$

where pre denotes the prediction result and gt is the ground truth.

Mean Relative Translation Error (MRTE)

The Mean Relative Translation Error (MRTE, m) is defined as

$$MRTE = mean\left(\|\vec{t}_{pre} - \vec{t}_{gt}\|_2\right). \tag{B.2}$$

B.1.2 (Root) Mean Squared Error ((R)MSE)

The (Root) Mean Squared Error ((R)MSE) is a more detailed metric, computed from the ground truth and predicted values. They are, for the translation (MSTE, m^2) and rotation (MSRE, $^{\circ 2}$), respectively:

$$MSTE = mean\left(\vec{t}_{pre} - \vec{t}_{gt}\right)^2 \tag{B.3}$$

$$MSRE = mean\left(\mathbf{R}_{pre}^T\mathbf{R}_{gt} - \mathbf{I}\right)^2. \tag{B.4}$$

Taking the root out of these errors results in the Root Mean Squared Error.

B.1.3 Mean Absolute Error (MAE)

The Mean Absolute Error (MAE) is the mean of the absolute error between the ground truth and predicted values. For the rotation, the Euler Angles can be used. The error is then expressed in degrees.

B.1.4 Recall (R)

Recall (R) is another metric introduced by Wentao, Y. [55], based on the RMSE. The idea is to identify the number of pairs for which the RMSE is smaller than a certain threshold. Since the real pairs are often unknown due to noise in the data, the ground truth transformation $\mathbf{T}_{gt}$ is used instead.

$$RMSE = \sqrt{mean(\|\mathbf{T}_{gt}(\vec{x}_i) - \mathbf{T}_{pre}(\vec{x}_i)\|_2^2} < Threshold\ \tau \tag{B.5}$$

While the before-introduced metrics should approximate zero as close as possible, recall should converge to one.

B.1.5 Area Under Curve (AUC)

A final metric for evaluating the rigid motion solution is the Area Under Curve (AUC) metric by Vinit, S. [38]. The success ratio is defined as the ratio of experiments that are considered a success over the total number of experiments. The threshold for defining whether or not an experiment is a success is called the success criteria, which can for example be an allowed rotation error. AUC is then defined as the area under the curve obtained by plotting the success ratio as a function of the success criteria. AUC is a measure of registration success. The higher the value of AUC, the better.

B.2 Feature Extraction Metrics

Another set of metrics discussed by Zhiyuan, Z. [48], aims to analyse the accuracy of the found correspondences. These metrics are of more importance to evaluate the found features. The aim of this report is more on the Rigid Motion estimation. Nevertheless, they are mentioned for completeness.

Feature-match Recall

Feature-match Recall expresses the percentage of pairs found between point clouds X and Y, which can predict the pose with high confidence.

Registration Recall

Registration Recall aims to express the number of overlapping fragments with the ground-truth pose that can be correctly identified.

B.3 Other metrics

Computation speed

Some Point Cloud Registration Methods, such as Generalized-ICP, achieve high accuracy but are computationally slow. In real-time applications, speed is a crucial parameter. Hence the use of computation speed to compare the different techniques.

Coefficient of determination (R^2)

The Coefficient of determination (R^2) is a statistical tool indicating the accuracy of a prediction. It is most easily understood by looking at the problem of fitting a linear curve to some data points as shown in Figure B.1. The closer the value of R^2 gets to one, the better the fitting. R^2 is defined as

$$R^2 = 1 - \frac{SSD}{TSS} \tag{B.6}$$

where SSD indicates the Sum of Squared Distances (the blue lines in Figure B.1),

$$SSD = \sum_i |f_{pre}(x_i) - y_i|^2, \tag{B.7}$$

with f_{pre} the fitted curve and y_i the data points. TSS is defined as

$$TSS = \sum_i |y_i - \bar{y}|^2, \tag{B.8}$$

with $\bar{y}$ the mean of the data points y_i.
The Coefficient of determination is also used by Yue, W. [49] as a metric for Point Cloud Registration. A definition is not given but based on Eq. B.6 - B.8, the following formulas can be used:

$$SSD = \sum_i \|\mathbf{T}_{gt}(\vec{x}_i) - \mathbf{T}_{pre}(\vec{x}_i)\|_2^2, \tag{B.9}$$

where $\vec{x}_i \in X$, $\mathbf{T}_{gt}$ is the ground truth and $\mathbf{T}_{pre}$ the predicted rigid transformation.

$$TSS = \sum_i \|\vec{x}_i - \vec{\bar{x}}\|_2^2 \tag{B.10}$$

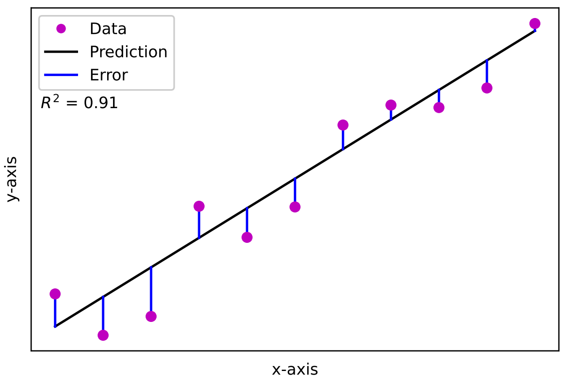


Figure B.1: Example of the Coefficient of determination metric R^2.

(Modified) Chamfer distance

The Chamfer distance is another metric, which computes the minimal distance between points of two point clouds. It is mentioned and modified by Zi Jian, Y. [54]. The reason why this metric is an interesting one compared to the metrics in Section B.1, is because it does not penalize alternative solutions in the case of symmetry. Furthermore, since RPM-Net [54] is especially effective in the case of partial point clouds, the metric is modified to

$$\tilde{CD} = \frac{1}{|X|} \sum_{\vec{x} \in X} \min_{\vec{y} \in Y_{clean}} \|\vec{x} - \vec{y}\|_2^2 + \frac{1}{|Y|} \sum_{\vec{y} \in Y} \min_{\vec{x} \in X_{clean}} \|\vec{x} - \vec{y}\|_2^2, \tag{B.11}$$

where X, Y are the partial point clouds and X_{clean}, Y_{clean} are the complete point clouds.

Appendix C

Code

In this chapter the layout of the code, that is used during the experiments, is discussed. The files are all written in the Python programming language. The goal of this chapter is to give an overview of the structure of the code and the available functionalities. Specific information about the individual functions is given in the related chapters or as comments inside the Python files. The full package of code used, can be found at the project Github page.

C.1 General Layout

The package is subdivided into different modules. The functions inside these modules are called upon from inside the *main.py* file. The following modules are available:

- **_test** Contains a set of Python files to perform Point Cloud Registration with the desired method. Each of these functions has the same type of input and output. The input consists of an *.hdf5* file containing the template point cloud, source point cloud and the ground truth transformation. The output is saved inside a folder, bearing the same name as the used method. This folder contains a *.hdf5* file with the found transformation, as well as the original point clouds and ground truth solutions.

- **_train** Contains a set of Python files to train the learning-based registration methods. The training is further discussed in Chapter 5. The functions require a training and testing dataset and return a trained model. This model is saved inside the **toolboxes** folder, under the respective registration method.

- **datasets** Holds the different datasets of CAD models that are used. In this case, it contains the CAD models used in the experiments and the ModelNet40 dataset. These are discussed more in depth in Chapter 4.

- **h5_files** This folder contains a set of functions to pre- and post-process the different file types into the required *.hdf5* files. The obtained scans from the experiments and a series of various produced *.hdf5* files is also included. These steps are further discussed in Chapter 4.

- **misc** Is a collection of miscellaneous functions with different utilities. This includes computing the metrics used for comparing the results and visualising the obtained transformations.

- **toolboxes** In this folder, the different toolboxes are added. These toolboxes mainly contain the required files to make the different Point Cloud Registration methods work. The most prominent packages here are the learning3d toolbox [37], ROPNet [63] and GO-ICP [53, 34].

The goal of this report is not too explain the details of every Python file. The algorithms themselves are only implemented from the authors source code. The remaining files are used to efficiently perform the experiments. The results of the registration are also added on the Github page, together with the created codes. Further documentation is available inside the files in the form of comments. The *main.py* files control the different tools available and should provide a good starting point for applying the code.

C.2 Toolboxes

Several modules and packages are used to perform the registration methods, visualise the point clouds and process the data. The main ones are described in Table C.1.

Table C.1: Summary of the libraries and modules used in the package.

Module	**Description**	**Application**
PyTorch	Machine-learning module for Python	Training, testing of learning-based methods and tensor computations.
Numpy	Module for array operations	Point cloud data transformations, error computations
Open3D [60]	Open-source library for handling 3D data	Point cloud visualisations and transformations, ICP, FGR, RANSAC
Learning3D [37]	Library of common registration methods[1]	PointNetLK, RPMNet, PRNet
ROPNet [63]	Library for ROPNet registration	ROPNet
GO-ICP [53]	Library for GO-ICP registration	GO-ICP
CUDA	Module for accessing GPU capabilities	Accessing GPU

[1]The methods are implemented using the source code of the original methods.

Appendix D

Training

This chapter discusses the training of the learning-based methods more in depth. The method of generating the training files is further explained in Section D.1. After, the results of the training process and its validation are considered for PointNetLK, RPMNet, PRNet and ROPNet in Section D.2. The chapter concludes with a short discussion on the feasibility of the learning based methods.

D.1 Training Files Generation

As mentioned in Chapter 5, different variations of data can be created, using some basic guidelines as proposed in the literature [54, 2, 49, 63, 48, 38]:

- The objects are sampled with 1024 (or 2048) points.
- The point clouds are shifted, such that their means are placed in the origin.
- The point clouds are rescaled to fit inside a unit sphere.
- Translations are limited to be within a $[-0.5, 0.5]$ range.
- Rotation are created about any random axis, limited to a $[0°, 45°]$ range.
- Noise is added according to a normal distribution with zero mean and a standard deviation of 0.01, $\mathcal{N}(0, 0.01)$. The noise is clipped to be within a $[-0.05, 0.05]$ range.
- Partiality is achieved by removing a percentage of points in a certain direction.

In Section D.2, the results of training the different learning-based methods are discussed. Special cases and findings for each method are mentioned in respective subsections (D.2.1-D.2.4). Each method is first trained and afterwards validated with a new (synthetic) dataset, consisting of 100 new transformations.

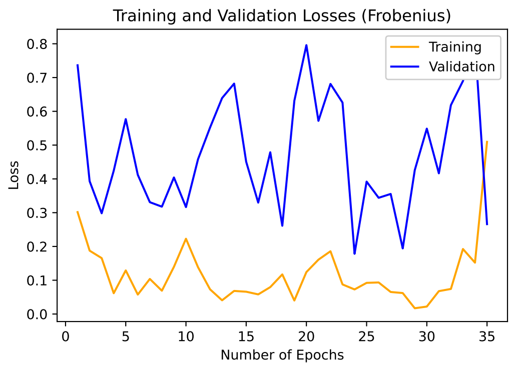
D.2 Training Results

The training process depends on some important parameters, including the training data itself, the batch-size and the number of epochs. During training, the method applies registration on the data and compares the result to the ground truth estimation. From this, a loss (error) is computed, which is used to adapt the model parameters. The basic idea behind this, is to guide the model towards the lowest loss, by adapting a gradient-descent approach. Instead of adapting the loss after each individual case, it is also possible to compute the loss over multiple situations, before adapting the parameters. The benefit of this, is to allow the algorithm to generalize better and to converge more quickly to the optimal solution. However, this comes at the risk of overfitting to the data and a higher demand on the memory. The amount of cases combined before adapting the parameters is called the batch-size. A batch-size of one corresponds to the case where the parameters are updated after each solution. A higher value of the batch-size, means that more objects are combined before updating the parameters.

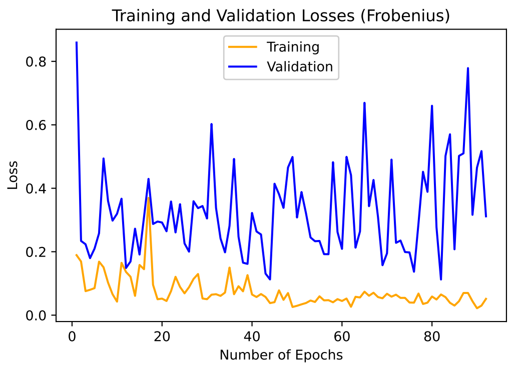
In order to allow the algorithm to learn the features of the objects and to prevent overfitting, the training data is run multiple times trough the method in a random order. The total amount of times the algorithm moves through the complete training data, is called the number of epochs. As the number of epochs increases, meaning the algorithm has seen the data multiple times and been able to capture the important features, the loss should decrease. Once the loss has converged to a low value, the training is stopped. Ideally, the number of epochs should be sufficiently large, such that the loss reaches its lowest value.

During training, the algorithm is not only applied on the training set. The method is also used on another dataset, called the testing dataset. This dataset contains data, constructed in exactly the same fashion, however each iteration differs from the training set. The method is then verified on this testing set, which is not used for training, in order to asses the generalization properties. When the testing loss achieves a minimal value, the parameters are saved. In the end, the model used, is the one that achieves the lowest loss during the training cycle on the testing data.

Care must be taken to initiate the training process correctly. A too low batch-size may lead to the loss decreasing very slowly or not at all, leading to a model that is underfitted. This typically leads to a loss for training and testing that remains high, as in Figure D.1a. On the other hand, the algorithm should be able to generalize. This means it should be able to still yield satisfactory results when applied to new data, on which it is not trained. If the batch-size is chosen too large, the algorithm may overfit the data. As a result, it will only achieve high performance for data on which it is trained, but not on never-before seen data. In this case, the training loss decreases to a low value, while the testing loss remains high. This is illustrated in Figure D.1b. The process of training is summarised in the pseudo-code Listing D.1.



(a) Underfitting, both training and testing losses do not converge, batch-size is 4



(b) Overfitting, only the training loss converges, batch-size is 7

Figure D.1: Example of the evolution of the training and validation loss as a function of the epochs, for PointNetLK on noisy data (0.01 standard deviation) . In this case, the too low batch-size (a) leads to underfitting of the data, as both the validation and training loss do not converge to a lower value. A too high batch-size (b) leads to overfitting, as the training loss converges but the validation loss does not.

Listing D.1: Training process basics

```
for epoch in range(number_epochs):

    #TRAIN DATA
    for batch in range(number_batches):
      ...
      update(parameters)
      iteration ++

    (end for)

    #TEST DATA
    ...

(end for)
```

The results of training the learning-based methods are summarised in Table D.1.

As can be seen from Table D.1, the errors are all low, with values of R^2 close to one, rotational errors (Relative, RMSE, MAE) at most 3° and small absolute translation errors (MAE). It is important to note that the translation errors are expressed in millimeters. However, as mentioned in Section D.1, the training objects are re-scaled to fit inside a unit radius sphere. This means, the objects do not have the same scale as in the real world. Instead, they are much bigger, due to the scaling operation. A comparison for the *Base-Top_Plate* object is shown in Figure D.2.

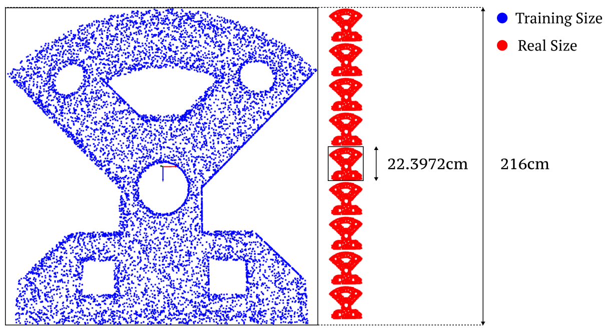


Figure D.2: Difference in the sizes of the training and captured point clouds. The blue point cloud is scaled to fit inside a unit sphere (with a radius of $1m$ considering the convention used). The red point cloud is scaled to the size of the captured data in cm.

The recall values are sometimes (RPMNet, partial, floor data) lower, for the same reason. For these cases, the translational errors are of the order of $2cm$ on average. Since the recall values are taken with a threshold of $1cm$, this explains the lower percentages.

Finally, during training, only one ground truth solution is considered. Most objects contain some symmetry planes as mentioned in Section 4.1. This results in some situations where the obtained transformation could be considered satisfactory, even though it differs from the ground truth solution. However, considering the results in Table D.1, there does not seem to be a big impact on the results. Instead, the symmetry of the objects results in bigger challenges during the training process itself. It is possible for the method to not converge due to this problem, such as is the case for PointNetLK (Section D.2.1-D.2.4). The results mentioned in Table D.1 only include instances of converged training sessions.

The subsections below detail the findings for each method more in depth.

Table D.1: Validation results of training PointNetLK, RPMNet, ROPNet and PRNet on different datasets. (1): Normal data, (2): Noisy data, (3): Partial data, (4): Floor, Noisy & Partial data. The training parameters are also indicated, the number of iterations indicate the training-test dataset distribution (820 – 205), the data used for training (A: all, L: limited), the noise standard deviation σ_n (0.01 or / when no noise is added), the percentage of points removed in case of partiality (50%, 70% or / when no points are deleted) and whether a floor is added to the objects (T: True, F: False).

Dataset	Iterations [/]	Data [A/L]	σ_n [/]	Partial [%]	Floor [T/F]	MRAE [°]	MRTE [mm]	RMSE [°]	RMSE [mm]	MAE [°]	MAE [mm]	Recall (0.01) [%]	R^2 [/]
						PointNetLK							
1	820-205	A	/	/	F	2.13	0.78	0.07	0.05	0.37	0.00	91.71	1.00
2	820-205	L	0.01	/	F	0.31	2.94	0.00	2.88	0.13	0.01	97.82	1.00
						RPMNet							
1	820-205	A	/	/	F	2.60	3.00	0.04	2.23	0.82	0.00	70.76	1.00
2	820-205	A	0.01	/	F	2.90	3.16	0.05	2.14	0.67	0.00	72.82	1.00
3	820-205	A	/	50%	F	2.71	16.42	0.04	10.52	0.85	0.11	49.78	1.00
4	820-205	A	0.01	70%	T	2.40	25.76	0.02	17.94	0.98	0.32	30.40	1.00
						ROPNet							
1	820-205	A	/	/	F	0.02	0.00	0.00	0.00	0.00	0.00	100.00	1.00
2	820-205	A	/	/	F	0.24	1.76	0.00	1.22	0.11	0.00	99.91	1.00
3	820-205	A	/	70%	F	0.99	8.83	0.02	11.95	0.49	0.14	79.67	1.00
4	820-205	A	0.01	70%	T	1.12	11.51	0.01	7.33	0.48	0.05	69.58	1.00
						PRNet							
1	820-205	L	/	/	F	1.70	24.61	0.03	19.07	0.45	0.36	32.01	1.00

D.2.1 Training PointNetLK

PointNetLK is the most sensitive to the presence of symmetric objects. The method is first trained on normal data, without the addition of noise or partiality. The evolution of the loss is shown in Figure D.3.

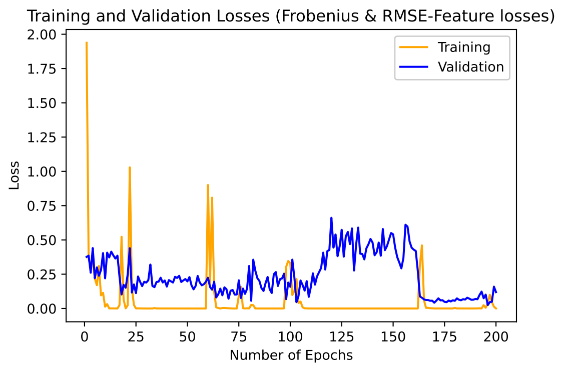


Figure D.3: Training and testing loss evolution as a function of the number of epochs for PointNetLK, trained with a batch-size of 32, Colab-GPU, on all the data, 820-205 training-testing distribution, errors validated

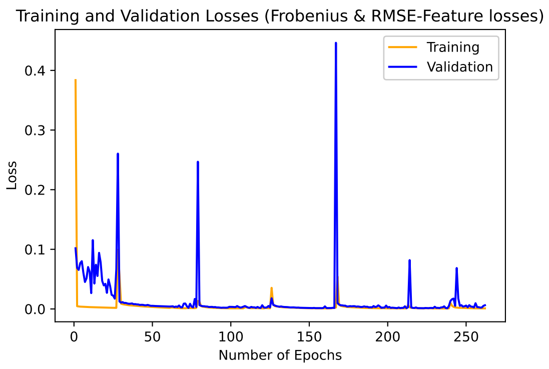


Figure D.4: Training and testing loss evolution as a function of the number of epochs for PointNetLK, trained with a batch-size of 20, Colab-GPU, no *Round-Peg*, 820-205 training-testing distribution, symmetric function on average, errors validated

The loss is indeed decreasing when the amount of epochs increases as one would expect. However, the losses show some peaks, where their values suddenly increase. One explanation for this is the use of symmetric objects. Since the algorithm is trained by comparing the estimated transformation and the ground truth, it is possible for the solution to lead to a high loss, even though it matches one of the (many) symmetric solutions. This effect is also illustrated in Figure D.5 where the results of training are shown for epoch 62. This indeed lines up with the higher values of Figure D.3 around this epoch.

When introducing noise to the training data, additional complexity is added. Due to the noise, there no longer exists a 1 : 1 correspondence between points in both point clouds. Different batch-sizes are tried to achieve a convergence of the loss, however, in the end the *Round-Peg* object was removed from the training set. This object has an infinite amount of symmetric solutions. Combined with the additional noise, the training did not converge. When the object is removed, the loss evolves as in Figure D.4.

The loss now converges to a low value. Further investigation of the peaks around epochs 79, 167 and 214 shows some failures of registration for the *Base-Top_Plate* and *Shaft-New* objects here, as shown in Figure D.6. Lastly, the algorithm is trained by choosing the symmetric function parameter as average instead of max. The symmetric function refers to one of the parameters for PointNetLK. As mentioned by Aoki, Y. et al. [2], choosing the symmetric function as average improves the results for noisy data.

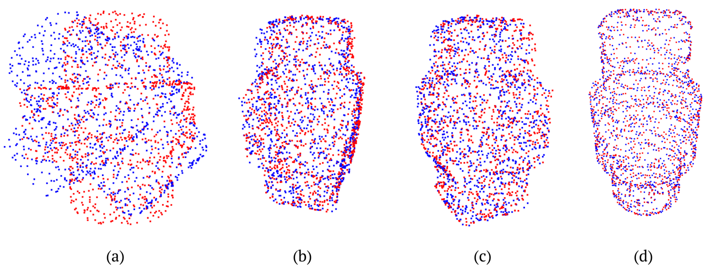


Figure D.5: Examples of failure cases during the training process of PointNetLK on normal data, after epoch 62 in Figure D.3. The main failures are observed for the *Shaft-New* (a) and object, or due to symmetric solutions for the *Round-Peg* (d) and *Square-Peg* (b,c) objects.

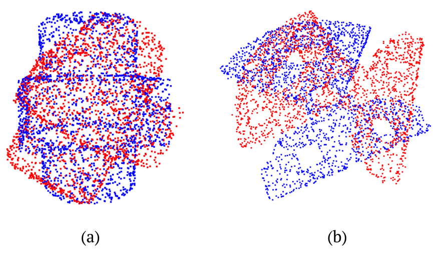


Figure D.6: Examples of failure cases during the training process of PointNetLK on noisy data, for epochs 167 and 214 in Figure D.4. The main failures are observed for the *Shaft-New* (a) and *Base-Top_plate* (b) objects.

Other datasets including partiality or floor information are tried, but lead to the algorithm not converging for the considered objects.

D.2.2 Training RPMNet

RPMNet is trained on similar datasets with normal and noisy data. However, RPMNet seems to be more robust to the presence of symmetric objects. Thus, the method is also trained on noisy data with all objects present, but also on partial data and even data containing floor information. An important comment is the different loss definition used in Figures D.7-D.8. The implementation of the RMSE-Feature loss in the available code does not work in the case of a different number of points between the two point clouds. For this reason, it is removed from the loss computation in some training sessions. The batch-size is also chosen smaller than for PointNetLK, since better results are obtained.

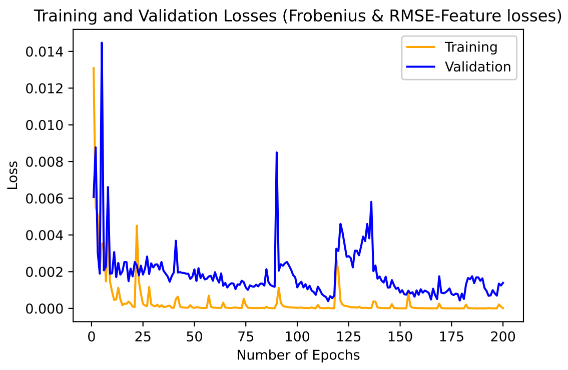


Figure D.7: Training and testing loss evolution as a function of the number of epochs for RPMNet, trained with a batch-size of 8, PC-GPU, on all the data, 820-205 training-testing distribution, errors validated

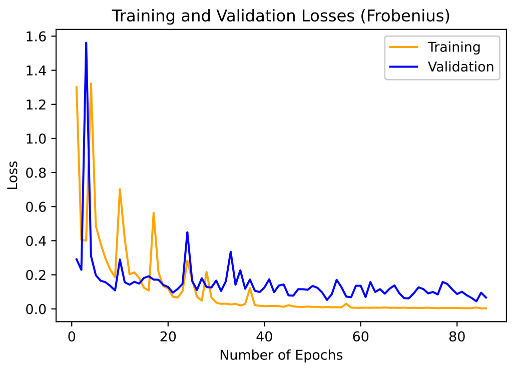


Figure D.8: Training and testing loss evolution as a function of the number of epochs for RPMNet, trained with a batch-size of 8, Colab-GPU, on all the data, 820-205 training-testing distribution, errors validated

D.2.3 Training ROPNet

Similarly to RPMNet, ROPNet is also trained on normal data, noisy data and floor data. From the results in Table D.1, ROPNet achieves a better performance over all metrics than both PointNetLK and RPMNet in almost all cases.

D.2.4 Training PRNet

PRNet is the final learning-based method considered. However, after running different training cycles, it is concluded PRNet is too demanding to be trained successfully. One attempt is made by training the algorithm with only a single object. The results are shown in Table D.1

From these results it is clear PRNet is already outperformed by the other learning-based methods on the synthetic dataset. This is also observed by Choy, C. et al. [7], who mention issues with high-variance training losses and crashes during training. Considering also the amount of time required to train the algorithm, it is not considered during the further investigation of the methods. The issue of memory and time is further described in Section D.3.

D.3 Learning-based Methods Feasibility

A major consideration when using learning-based approaches is the availability of sufficient computation power. A good example of this is the PRNet algorithm. This method specifically requires, at least in the tests performed for this thesis, the most computing power and memory. A small comparison is given in D.2.

Table D.2 gives an overview of the required computation time for different possible GPU's. From this table it is clear that training the algorithm on a GPU with less than 15GB of memory becomes infeasible. Google Colab, while providing a more powerful GPU, limits the usage using computation units (CU). In

Table D.2: Overview of the training process for PRNet on different runtimes. The total computation time is computed for $N_E = 200$. The bottom half of the table shows the available CU, CH and maximal epoch that can be computed, given the 100 available computation units, for the different runtimes. The number of computation units is not limited when training on a personal device or the basic Google Colab plan. This is indicated in the table as /. Abbreviations: T_E - Time per epoch, N_E - Total number of epochs, HR - High-ram usage, GPU$_P$ - Premium GPU, CU - Computation Units, CH - Computation Hours, BS - Batch Size

Variable	**Laptop**	**Google Colab**	**Google Colab Pro**		
Runtime Type	GPU 8GB	GPU 12GB	GPU 14.76 GB	GPU + HR	GPU$_P$ + HR
T_E [min]	252.5	16.58	18.25	17.87	4.13
T_E X N_E [h]	841.7	55.27	60.83	59.57	13.77
CU/h	/	/	1.96	2.05	13.08
Max. CU	/	/	100	100	100
Max. CH	/	/	51.02	48.78	7.645
Max. Epoch	/	/	167.7	163.8	111.1
Max. BS	4	8	8	8	22

total, 100 units are available. These decrease while using the GPU at a rate depending on the runtime type. For PRNet, it is thus only possible to reach epoch 111 in a time of $8h$. In Sections D.2.1-D.2.2, reaching an epoch of 111 is not always required to achieve convergence of the loss. However, as also noted by Choy, C. et al. [7], the training of PRNet is not very stable and can lead to crashed and high-variance losses.

The training in general for the different methods can take a significant amount of time and requires a lot of computation power. Besides this, the dataset must be correctly constructed. Some datasets are too specific and will not lead to convergence of the method. The more difficult the training set becomes, the more difficult it is to train the method on this dataset and the more danger of overfitting if not sufficient data is provided. Training on a simpler dataset requires less effort and time, but may lead to a worse accuracy in the results. This is schematically illustrated in Figure D.9.

The different types of training models are proposed based on the literature. However it is clear that the datasets, constructed as in Figure 5.1, are still not a perfect description of the real world scans. More advanced modelling is possible, although requiring more pre-processing and a longer training time. Additionally, other loss functions could be considered to take into account the symmetry of the objects.

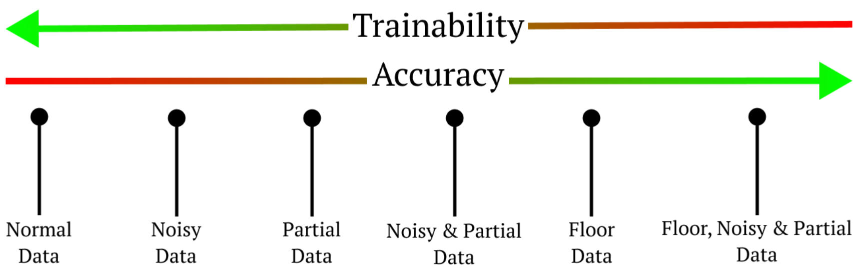


Figure D.9: Schematic representation of the compromise between accuracy of the training set in representing the real world scans and the ability the train the method. The different datasets are added according to their complexity.

Appendix E

Experimental Results

In this chapter, some additional results from the experiments are shown. The main parameter study can be found in Chapter 6. Conclusions are summarised in Chapter 7.

E.1 Zero Mean

Table E.1 illustrates the effect of centering the point cloud for GO-ICP, RANSAC, FGR, RPMNet and ROPNet. As discussed in Chapter 6, the best results are considered for the other experiments.

E.2 Global Registration Methods

E.2.1 Bounding Box

The R^2 metric is not the best metric to use when there are many outlier points. As a reminder, the result of registration is considered unsatisfactory when the coefficient of determination R^2 is negative and the rotational error (MRAE) is larger than 120°. The R^2 value is computed by considering the sum of squared distances (SSD) between the result and the ground truth and the total sum of squares (TSS) between the result and its mean. Since there are more points, that are spread out, the distance to the mean (TSS) is, on average, larger than the distance between the result and the ground truth (SSD). Thus, the ratio of SSD and TSS is typically small, leading to a higher value of R^2, since

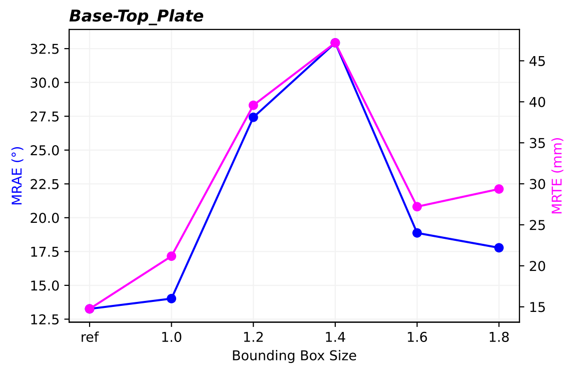
$$R^2 = 1 - \frac{SSD}{TSS}. \tag{E.1}$$

For this reason, the amount of failure cases is not necessarily representative for the performance of the results. The recall value determines the amount of points, for which the mean-squared error (MSE) is lower than a given threshold.

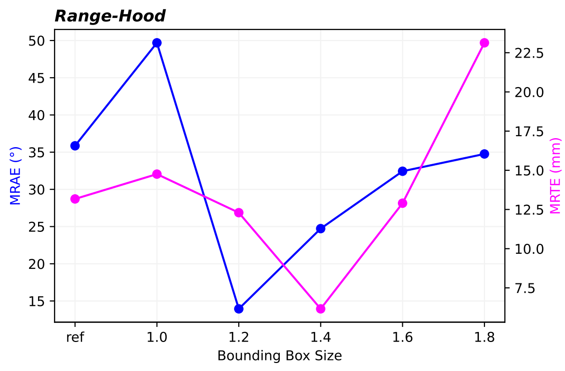
The Mean Relative Error is only based on the found transformation matrix and is thus a better fit to asses the performance in this case.

Table E.1: Comparison of the results of registration for GO-ICP, RANSAC, FGR, RPMNet (N: normal data) and ROPNet (N: normal data), when the pointcloud is centered on the origin (Zero Mean: True) or not (Zero Mean: False). The difference is generally limited. For the remained of the experiments, the best results are used. This means a zero mean for RANSAC, FGR and not for GO-ICP, ROPNet and RPMNet. ROPNet† fails for almost every object to provide a non-failure solution. The results in this table are only taken for the *Pendulum*,*Round-Peg* and *Square-Peg* objects for a non-zero mean and only the first two for a zero-mean. Abbreviations: VS - Voxel Size, MSE_thresh - Mean Squared Error Threshold, Trim - Trim Factor.

Model [/]	MSE_{thresh} [/]	Trim [%]	VS [m]	Zero Mean [T/F]	MRAE [°]	MRTE [mm]	RMSE [°]	RMSE [mm]	MAE [°]	MAE [mm]	Recall (0.01) [%]	R^2 [/]	Failure [%]
GO-ICP													
/	0.00001	0.001	/	F	5.93	3.04	0.05	1.76	4.69	0.00	99.27	0.97	24.40
/	0.00001	0.001	/	T	6.81	4.23	0.06	2.44	5.35	0.01	95.58	0.87	29.76
RANSAC													
/	/	/	0.005	F	30.06	12.64	0.24	7.30	24.60	0.07	59.83	0.65	71.36
/	/	/	0.005	T	31.82	12.17	0.26	7.03	17.47	0.06	57.78	0.64	71.36
FGR													
/	/	/	0.005	F	34.27	13.22	0.36	7.63	20.66	0.07	50.42	0.36	81.14
/	/	/	0.005	T	32.82	13.47	0.34	7.78	21.04	0.07	47.62	0.37	80.92
RPMNet													
N	/	/	0.003	F	30.17	11.27	0.24	6.51	6.66	0.05	62.71	0.56	71.61
N	/	/	0.003	T	31.90	11.27	0.26	6.51	8.81	0.05	62.50	0.57	71.61
ROPNet†													
N	/	/	0.003	F	45.41	23.25	0.36	13.42	37.09	0.24	57.67	0.35	86.11
N	/	/	0.003	T	-35.37	25.74	0.52	14.86	67.45	0.29	46.53	0.55	90.74

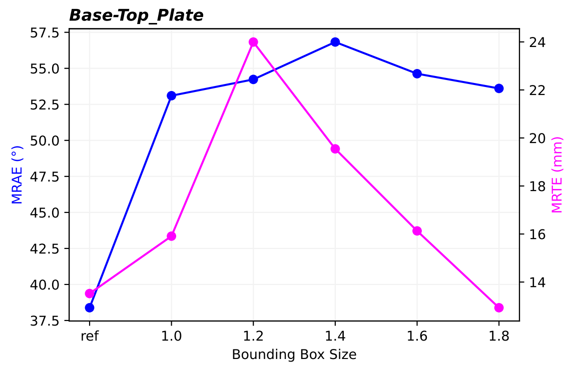


(a) *Base-Top_Plate*

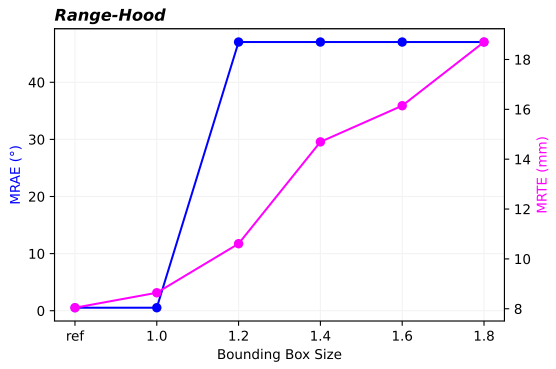


(b) *Range-Hood*

Figure E.1: Effect of the bounding box size on the Mean Relative Error (MRE) for RANSAC, for four objects. The first data point is taken from the previous experiments, at the same voxel size, for the cleaned up point cloud. The errors do not show a clear trend for most scans. This can be related to the fact that, some scans taken from a certain pose, may benefit from the additional points, as this can help to better identify features. However, near the end ($BB = 1.8$), the error seems to increase for almost all objects. The angular (-●-) and translational (-●-) errors show overall a similar evolution.



(a) *Base-Top_Plate*



(b) *Range-Hood*

Figure E.2: Effect of the bounding box size on the Mean Relative Errors (MRE) for FGR, for four objects. The first data point is taken from the previous experiments, at the same voxel size, for the cleaned up point cloud. The errors now show a more clear evolution for *Pendulum* and *Range-Hood*. Again, this is most likely due to the specific scans, which benefit from the additional points. Except for the *Separator*, The angular (-●-) and translational (-●-) errors show a similar evolution.

As can be seen from Figures E.1-E.2, the errors show again a more clear trend for FGR, where they are generally increasing. For RANSAC, this trend is more difficult to identify. The reason for this could lie in the scans that are used during the experiments. Since, for each object, different poses of the objects are considered, some scans might benefit from the additional points added through the larger bounding box. In some cases, most notably, the *Range-Hood* object (Figure E.1), the error even drops below the value obtained for the cleaned up point cloud. However, for this specific object, the bigger bounding box could indeed prove useful. Due to the cleanup process, some information is lost around the bottom of the object, which is recovered by increasing the bounding box, as shown in Figure E.3.

For a very large bounding box, which is increased by 80%, most errors show again an upwards trends. For FGR, the errors seem to increase over almost all objects, except the *Separator*. Again, the increased bounding box can yield some additional information, used to find features in the point cloud, leading to lower errors. It should be noted too that, for this specific object, the error only changes by a couple of degrees with a changing bounding box size.

Another important comment is that the errors are only computed over the scans, which are not considered to be failed. Again, this means scans for which $R^2 > 0$ and the $MRAE < 120°$. Looking at the

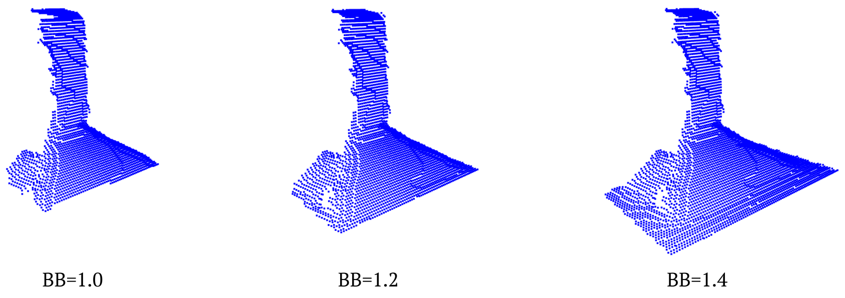


Figure E.3: By increasing the bounding box for the *Range-Hood* object, additional information is recovered, that is used to better identify features.

distribution of the successful and failed scans, it is clear that this varies a lot between bounding boxes. For some values of the bounding box, certain scans may perform better, while not for another value. This is yet again related to the fact that some scans can benefit from the additional points, while others do not.