Multi-Camera Registration for VR:
A flexible, feature-based approach

Version of December 14, 2018

Qinzhuan QIAN

Multi-Camera Registration for VR:
A flexible, feature-based approach

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Qinzhuan QIAN
born in Shanxi, China

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

CWI
Science Park 123
Amsterdam, the Netherlands
www.cwi.nl

Cover picture: A “random” maze generated in postscript.

Multi-Camera Registration for VR: A flexible, feature-based approach

Author: Qinzhuan QIAN
Student id: 4755154
Email: Q.Qian@student.tudelft.nl

Abstract

Real-time point cloud capturing and multiple depth camera 3D reconstruction are vital elements that bring real-time representations into a virtual world and provide an immersive experience. To make this possible, camera calibration plays an essential role in providing important camera spatial information for 3D scene reconstruction. However, there are still many drawbacks left to improve on camera spatial parameters calculation in most existing systems: such as the procedure is complicated and relies too much on additional calibration markers, or specific data type provided by a depth camera that cannot easily be generalized to other depth sensors.

To improve on this, we propose a markerless, feature-based pipeline for multiple camera calibration. This pipeline contains four main stages. It adopts feature descriptor extracting and matching to solve the issue of requiring additional markers, and the point cloud registration accuracy is improved by using point cloud segmentation and part selection.

The experiment results obtained in this research show that this pipeline can calibrate four cameras with a single object (such as a chair, lamp) without the need for additional calibration markers. The camera parameters calculated using this pipeline is more accurate and requires less processing time than originally. This pipeline provides the potential for further human point cloud capturing and camera re-calibration in real-time 3D reconstruction.

Thesis Committee:

Chair: Prof. Dr. A. Hanjalic, Faculty EEMCS, TU Delft
University supervisor: Dr. P. S. Cesar Garcia, Faculty EEMCS, TU Delft
External supervisor: Dr. A. A. M. Kuijk, Distributed and Interactive Systems, CWI
Committee Member: Dr. Klaus Hildebrandt, Faculty EEMCS, TU Delft

Contents

List of Figures

Chapter 1

Introduction

Virtual reality is a technology that uses computer simulation tools to generate a virtual world in a three-dimensional space. It provides users with sensory simulations such as vision and touch, making users feel as if they are immersed in a virtual environment and can observe 3D space in real time.

The concept of virtual reality was proposed back in 1932[49]. In the following twenty years, researchers designed VR devices and products to realize this concept. The early VR products were mainly used in the military field. With the development of hardware and software technologies, an increasing number of VR products have been released in the consumer market in recent years, and VR technology has also been applied in various fields.

One particular type of application scenarios is social VR. Many platforms in the market currently provide social media functions, such as VRchat, Facebook spaces, and Second Life. VRchat creates a virtual world in which users can interact with others by using fictional avatars. Facebook's social application (Facebook spaces) provides a platform for sharing and communication. Users can quickly set up customized avatars and interact with other users through avatars' expressions, actions, and sounds. Second Life created a virtual reality world where users can socialize and participate in individual or group activities; the user can also create and trade property and services in this world. Fig.1.1 shows an image of Second Life.

The user in the above mentioned social VR products is represented as a cartoon image or a virtual avatar, and apparently there is a big difference between a cartoon avatar and a real person. The objective of social VR applications is to build a virtual social platform to simulate and replace face-to-face communication. However, using a 3D virtual avatar representation loses real expression, eye contact and demeanor details of the user, whereas real-time reconstruction of the human body, this highly realistic 3D model can reproduce these important social signals of the user in real-time. The combination of real-time 3D reconstruction of the human body and the surrounding environment in a virtual world is a challenge in VR/AR applications.

In VR and AR pipelines, there are three key components: data acquisition, optimization transmission, and real-time scene reconstruction and interaction. Data acquisition and trans-

Figure 1.1: Image of Second Life Game. The 3D avatar shown in the image is wearing a VR/AR device for interaction and game in the virtual world.
source: https://www.roadtovr.com/second-life-oculus-rift-beta-test-linden-labs/

mission ensure data quality and optimal transmission speed. 3D rendering reconstructs scenes and reproduces the movement of characters in real-time. An efficient rendering algorithm is used to apply colors and lighting to improve realism.

Microsoft released a depth camera Kinect for gaming and entertainment in 2012. Subsequently, with the development of depth sensor technology, depth sensors became relevant for civilian consumers and researchers. Compared to depth scanners and professional depth sensors, the latest generation depth cameras are smaller, more convenient and cheaper.

As affordable depth cameras entered the market, real-time 3D reconstruction has also been widely developed. Real-time 3D reconstruction technology aims to reconstruct the user's body and the surrounding environment in three-dimensions. The movements and expressions of the user can be reconstructed reliably, so that this reconstructed human 3D model can be used to represent the user in the virtual world.

The KinectFusion[76] system published in 2012 performs real-time 3D reconstruction through camera pose estimation and data fusion. This real-time 3D reconstruction system mainly uses the ICP algorithm[84] and TSDF representation[25] method. However, KinectFusion can only work with a single depth camera, the reconstruction results have occlusion problems, and it cannot be used in large-scale scenes. Four years later, the so named Fusion4D system[77] proposed how to use multiple depth cameras for real-time 3D reconstruction, which extended 3D reconstruction from a single camera to multi-camera configurations in order to achieve a 360-degree 3D reconstruction. With the development of deep learning technology, researchers had been trying to make the machine have the ability to identify and understand the surrounding environment and achieve semantic 3D reconstruction. The SemanticFusion system[69] published in 2017 is a semantic real-time reconstruction system based on the SLAM system[122] and CNN network[50]. This system can more intelligently understand and reconstruct the surrounding environment and thus be used in various fields.

An essential part of multi-camera real-time 3D reconstruction technology is camera pose estimation and calibration. This requires combining the contributions of the individual cameras. Currently, there are two types of camera calibration methods. One is based on specific calibration objects, such as a checkerboard or dotted pattern. These calibration methods can be implemented by using MATLAB toolkit[13] or OpenCV[14] library. The operation of the MATLAB toolkit is complex and requires manual corner detection, but it provides higher accuracy calibration results than the OpenCV method. Another calibration method removes the use of additional calibration objects and can calibrate cameras using unmarked objects. For example, The calibration system proposed by Kevin et al.[56] is a human skeleton-based calibration system that calibrates multiple-camera based on the skeleton extracted from a depth camera. However, this calibration method relies on the skeleton data provided by the software tools that come with the camera. Unfortunately not all cameras on the market provide human skeleton data. Therefore, this calibration method cannot be generalized for all depth cameras.

The project I have been collaborating with (VRTogether) is a social VR platform that builds a virtual world by reconstructing scenes and users in real-time. Unlike other social VR products that are currently available on the market, it reconstructs human models to replace virtual cartoons and provides a platform for meetings and entertainment. The overview of the platform is shown in Fig.1.2, the whole system is divided into a CLIENT and a SERVER side. The point cloud data acquired from the depth camera is first processed on the SERVER side. On the SERVER side, the point cloud data fragments of multiple-camera are merged in real-time, and then the obtained reconstructed point cloud that forms a complete representation of the user is sent to the encoding module for data processing. This system provides encoding channels to accommodate different data quality and transfers this data to the CLIENT side. On the CLIENT side, the system selects the required data quality levels, then the decoding module decodes the corresponding data, and finally displays the point cloud data on the appropriate display device.

This project focuses on the multiple-camera pose estimation and calibration needed on the SERVER side to be able to do the 3D reconstruction by merging the point cloud fragments from the individual cameras. This process is known as camera registration. We design a multi-camera calibration system, which includes coarse registration, pre-processing, fine registration and point cloud update steps. Fig.1.3 shows an overview of our system. At first, we use MATLAB toolkit[13] and the ICP (Iterative Closest Point)[126] algorithm in our pipeline. However, the experimental results show that this calibration method has limitations: the captured point cloud from each depth camera is incomplete because a depth camera can only cover a part of the object and may have occlusion (e.g., an arm covering part of the body). The incomplete part of different cameras may cause the ICP algorithm to generate registration errors. Second, the system still requires a checkerboard as a calibration object. Although this process is a one-time process when cameras are at a fixed position,

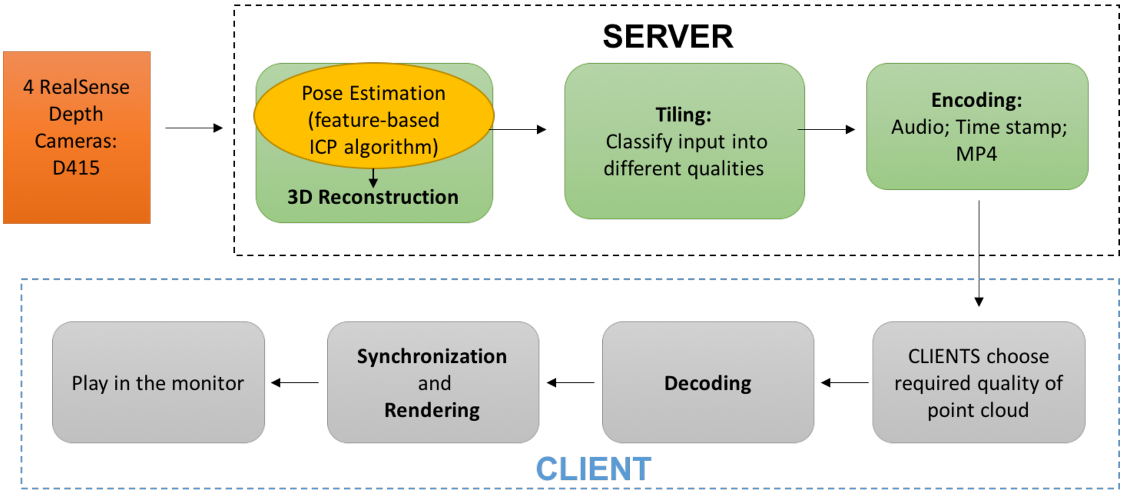

SERVER
4 RealSense Depth Cameras: D415
Pose Estimation (feature-based ICP algorithm)
3D Reconstruction
Tiling: Classify input into different qualities
Encoding: Audio; Time stamp; MP4
CLIENTS choose required quality of point cloud
Decoding
Synchronization and Rendering
Play in the monitor
CLIENT

Figure 1.2: An overview of the Social VR platform system. The red module is the step of capturing 3D data by the depth camera. The green modules are on the SERVER side, the 3D reconstruction and data encoding of the point cloud is performed. The gray modules are steps that run on the CLIENT side for decoding, rendering and point cloud visualization. This thesis focuses on multi-camera registration (the yellow module) that is needed to be able to perform the 3D reconstruction process on the SERVER side. Note that camera registration is not part of the continuous real-time streaming of camera data. It includes coarse and fine registration steps. The coarse registration is a one time event when cameras are static. The other modules in this figure are continuous processes. A more detailed overview of multi-camera registration system is shown in Fig.1.3.

this step needs to be re-run when any of the cameras in the system moves.

Therefore, we propose a feature-based multi-camera calibration system to implement a flexible calibration system without the use of additional calibration objects. The idea of this system is first to use a segmentation and selection method to select a part of a point cloud that is covered by the cameras that need to be calibrated. This reduces the errors caused by missing portions of the point clouds. We segment the point cloud by using the segmentation method provided by PCL(Point Cloud Library)[81], but as we will see in section 4.4 the segmentation results are not ideal and are not suited to be used in our system. So we use PointNet deep learning network to segment the point cloud, and then compare the point clouds captured by different cameras based on the volume of these segments (see section 3.3.4). In this way, the most suitable part that the individual cameras have in common is selected for registration. Secondly, we use the feature extraction, feature matching and alignment method in the PCL to perform the coarse registration step. This method is used to replace the MATLAB toolkit calibration step.

We use four calibration objects (chair, lamp, mug, and earphone) to test the pipeline. The experimental results show that our pipeline can shorten the processing time and improve the accuracy of point cloud registration. We show that the initial transformation matrix of

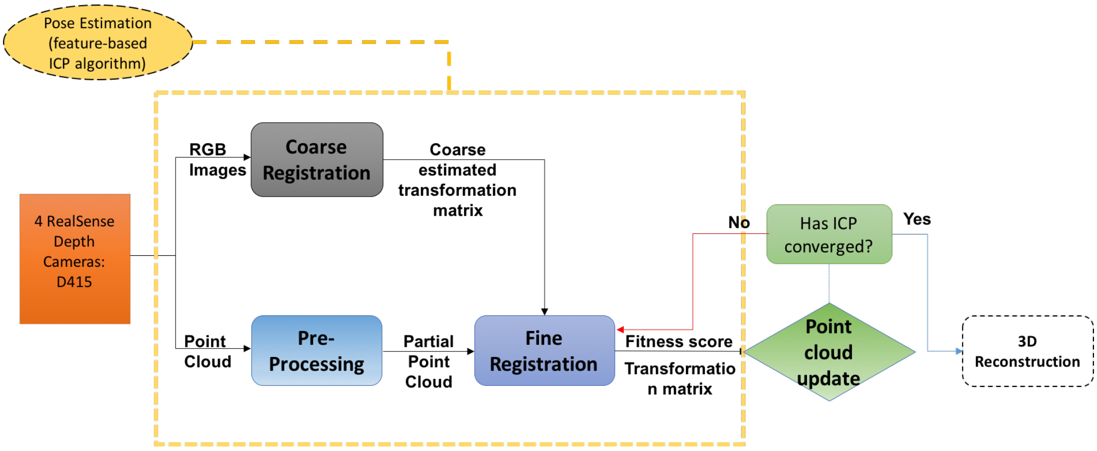


Figure 1.3: Overview of the multiple depth camera re-calibration system. This thesis implements and validates the modules in the yellow dashed box, including coarse registration, pre-processing and fine registration steps. The coarse registration is a one time procedure when the camera position is fixed. The remaining steps are continuous operations. The initial transformation matrix obtained by coarse registration and the pre-processed point clouds are used as inputs for the fine registration step, the camera position and pose is estimated by using ICP algorithm. In addition, we provide a solution for automatic re-calibration the system, which is shown in green modules. This system performs a re-calibrate process when the registration does not reach the matching accuracy and is lower than a predefined threshold. Note that this re-calibration process does not run for every captured frame, but runs at a much lower frequency.

feature extraction and alignment is comparable to the MATLAB method in performance, but our method is more convenient and time-saving in practice.

1.1 Objectives and Research Questions

There are two types of camera registration systems; one requires very specific calibration objects such as a calibration checkerboard[36][80], printed pattern[57] or labelled boxes[3]. These systems are time-consuming and have complex operating procedures that involves human intervention (positioning of calibration objects). Recently we can find so called markerless calibration systems that are less strict on the calibration object. An example is [56] that uses features of the human skeleton. However, this system requires a depth camera that provides human skeleton data based on capturing a human body. A tool for providing skeleton data is currently not available for all depth cameras on the market. Our research aims to design a multi-camera system that does not need specific marked calibration objects and that does not rely on proprietary tools of specific depth cameras.

The research questions investigated in this thesis are:

Can we improve the robustness and performance of the ICP algorithm in the fine registration step?

This research question deals with the poor robustness of the ICP[11] algorithm which occurs in practice. Due to the fact that the point cloud data captured by one depth camera does not cover the complete object, as it can capture one side of the object only, the algorithm can fall into a local optimum and produce wrong results. We need a solution to overcome this inherent shortcoming of the ICP algorithm. Improving the robustness of the ICP algorithm is a must, but improving the performance comes next as this reduces setup time and may even enable run-time realignment.

Can we replace specific calibration objects in the coarse registration step by more generic objects and achieve a more user-friendly calibration procedure?

This research question deals with how to replace the commonly used complex coarse registration methods. There exist two popular types of coarse camera calibration methods: one is based on the MATLAB toolkit with high accuracy and complicated procedures; another one is based on the OpenCV library. Both approaches require specific checkerboards or marked objects. The alternative studied in this thesis is to use a feature-based coarse registration method to replace the MATLAB toolkit camera calibration system, which can avoid the need to make use of specific calibration objects and achieve more user-friendly multi-camera registration in terms of procedures and processing time.

How to achieve an automatic multi-camera self-calibration system by using the captured point cloud data?

Automatic camera calibration that ideally offers dynamic calibration of multiple cameras without manual intervention would be very attractive for a real-time 3D reconstruction system. Based on point cloud data, we studied a generic feature-based multi-camera registration system that can be used with all depth cameras on the market. This research question deals with how to automatically re-calibrate the camera based on the ICP algorithm and the point cloud data used in this project.

Thesis Outline:

This thesis is structured as follows. We first discuss the background and related work of technologies and equipment used in the system in Chapter 2, and then in Chapter 3 we present the contributions of this thesis, illustrate key modules in details and introduce the database used for system validation. In Chapter 4 we demonstrate the experimental procedures (theories), goals, settings and results. We analyze the results from a quantitative and, qualitative perspective and compare this system with the state-of-the-art in Chapter 5. Finally, we summarize our work and discuss the future development work in Chapter 6.

Chapter 2

Background

This chapter introduces the background and related work of our system. First, we discuss current VR and AR scenarios and introduce the potential development of social VR applications. Then, we introduce the technology behind depth cameras and compare commercial products on the market. In section 2.3, we present several 3D reconstruction technologies based on RGB-D cameras. By analyzing these 3D reconstruction systems, we favour the ICP algorithm for our system. For the ICP algorithm, an initial transformation matrix is required to calculate the registration of the point cloud. Therefore, in section 2.4, we study the method of camera coarse calibration that provides such an initial transformation. Finally, in order to improve the ICP algorithm, we propose a feature-based approach to increase the registration accuracy. We examined the use of a deep learning network to divide the calibration object into different parts. Therefore, in section 2.6 we introduce related work in 3D data deep learning networks.

2.1 AR/VR Scenarios and Point Clouds

The concept of virtual reality has been proposed in the past. For example, it was mentioned in the novel "Brave New World"[48] published in 1932. This novel describes a centralized high-tech world, which indicates a head-mounted device that can provide viewers with a series of sensory experiences such as images, smells, and sounds.

Researchers in the 1950s showcased prototypes of VR products. However, most of the VR products were used in the military and scientific research fields and were not widely developed. Until VPL Research[119] launched VR products into the market in 1991, more and more companies invested in and developed in the field of VR and AR. According to the forecast by Statista[110], the worldwide market size for VR and AR industries will have a significant increase in the coming four years. Fig.2.1 shows the forecast result by Statista. Nowadays, VR and AR products have been used in a wide range of scenarios include entertainment, commercial applications, and domestic service.

Entertainment

Games are one of the most attractive ways to learn new things. Currently, games are also

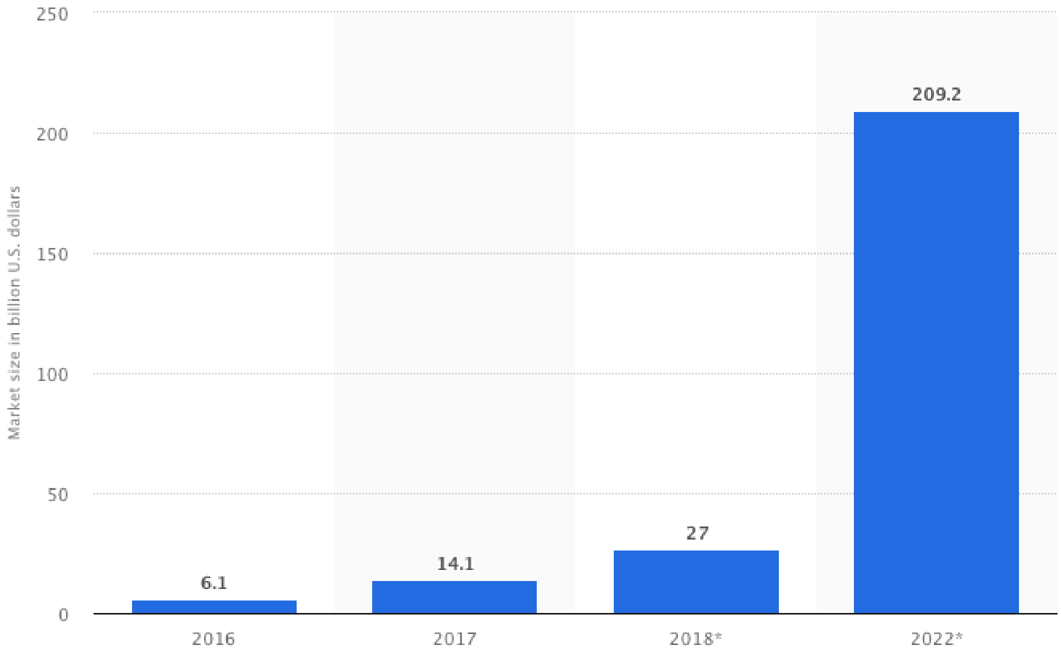


Figure 2.1: VR and AR worldwide market size from 2016 to 2022 (estimated)
source:
https://www.statista.com/statistics/591181/global-augmented-virtual-reality-market-size/

the primary use case for VR and AR industries. VR games have attracted a large number of users. Current VR game includes genres ACG (anime, comics, and games), first-person shooter (FPS) and action role-playing video games (ARPG). These three types of games have a high visual impact on the screen and a high level of player immersion. Fig.2.2 shows an image of a VR FPS game.

For social networking, the application of VR technology provides users the ability to communicate and share information in a virtual world. Development of social VR will enhance the user's immersion and sociality. For users, social VR provides more fun than regular video calls and allows users to better focus on communication and sharing in the virtual world. For example, Facebook space shown in Fig.2.3 is a VR product designed for memory sharing and communication.

Commercial applications

VR can as well be employed for museums. Previously, visitors were able to navigate through prefabricated virtual scenes and exhibits in space using mouse and keyboard operations. With the development of VR technology, participants can use 3D glasses and other wearable devices to improve human-computer interaction and the experience of being immersed[41].

By using VR, telling stories to customers by describing objects and narrating can enhance

Figure 2.2: VR FPS game: Arktika.1
source: https://uploadvr.com/arktika-preview-tell-show/

the interactivity and disseminate knowledge in a more interesting way[82]. Currently, many virtual museums have been established worldwide, such as the British museum[73], the Olympic museum[74].

VR technology has also been applied to the military field. These applications include soldier simulation training[99] and multi-military joint virtual simulation exercises. The virtual simulation exercise adopts the seamless interactive environment simulation training[4] of virtual entities and distributed virtual battlefields. The VR technology simulation training also includes distributed simulation training, integrating games into military training and incorporating holographic imaging technology into virtual reality to realize future combat systems with LVC(Live Virtual Constructive)[78].

In the medical field, VR technology can be used for virtual surgery training, assisted teaching[32], remote collaboration[71] and rehabilitation treatment. Among them, the virtual surgery training is the mainstream application. It combines VR with a 3D visualization system to represent various organs, tissues, and other information in an interactive virtual environment, which provides medical personnel the ability to learn and evaluate independently. At the same time, the low-cost VR system has advantages in novice doctor training and enhanced surgical techniques. The virtual surgery training can help doctors to customize a reasonable surgical plan, reduce surgical damage and improve the success rate of surgery.

Domestic service

With the increase of people's demands, tourism has become an indispensable part of life. Nowadays, applications for VR tourism has also been developed. At first, virtual tourism combined VR with geographic information and panoramic technology to generate a panoramic

Figure 2.3: Social VR product: Facebook Space
source: https://newsroom.fb.com/news/2017/04/facebook-spaces/

tourist model and achieve virtual roaming. Virtual tourism system based on Web 3D has been developed so that users can select any route and simulate arbitrary 3D historical or existing landscape without leaving the room. Users can also view and browse the road from any angle[1]. These functional applications have effectively improved the user's sense of presence and reduced travel costs.

Point Clouds

In VR systems, point clouds have recently been introduced as an object representation that has a high potential for enhancing VR and even bringing in new types of VR applications. A point cloud is a dataset of points in a 3D coordinate system. The point cloud contains three-dimensional coordinates X, Y, Z, and per point parameters like the color and optionally normals.

Point clouds can be obtained by 3D scanners such as light detection scanners and ranging devices (LiDAR). These devices identify many points on the object surface and output the data in a specific format. The point cloud data has proven to be a relevant type and has been applied in the following 3D computer vision field.

Object recognition: In the autonomous car area, pedestrians, cars, bicycles, and road auxiliary facilities (such as street lights, pedestrian crossing.) in the scene are detected based on laser scanner data.

Shape detection and classification: Point cloud technology has a wide range of application in reverse engineering. After constructing many geometric models, how to manage and retrieve them effectively is a difficult problem. The point cloud (mesh) model requires to be characterized and classified. The model is retrieved based on the feature information of the model.

Semantic classification: After acquiring the scene point cloud, we need to use the information and understand the content of the point cloud scene. It is necessary to classify the point cloud and perform labeling for each point cloud. It can be divided into point-based and segmentation-based classification methods.

2.2 Depth Sense Cameras

With the development of computer vision, augmented reality and robots, it became common practice to use depth cameras to capture the depth information of the environment and then perform object recognition and environmental modeling. Compared to traditional 2D cameras, depth cameras add one-dimensional depth (distance) information to describe the real world better. There are three popular 3D machine vision technologies used in the depth camera industry: structured light, TOF (Time of Flight) and stereo vision.

Structured light

The structured light method is also known as an active 3D measurement. Here active means that this method needs to project structured light onto the measured object actively. The parameters of the measured object will be calculated based on the deformation of the structured light.

In structured light technology, the position and the depth information of the object are calculated by refraction of the laser. The first step is to emit a specific speckle pattern or a laser infrared dot matrix pattern as shown in Fig.2.4. When the measured object reflects the patterns, these reflected patterns are captured by the camera. Then, the size of the speckle or dot pattern will be calculated by comparing with the projected speckle or dot pattern. The distance between the measured object and the depth camera is calculated based on the comparison of the original pattern size and the reflected pattern size.

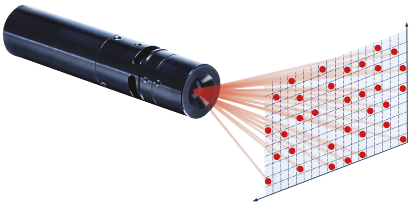


Figure 2.4: Laser infrared dot matrix pattern
source:https://www.vision-systems.com/articles/2014/05/osela-releases-random-pattern-laser-dot-matrix.html

Currently, many laser radar and 3D scanning technologies in the industry use structured light technology. For example, the depth cameras Kinect 1.0 (PrimeSense), Intel RealSense

SR300 and F200 series, and the iPhoneX make use of this technique. This method calculates the object position by the displacement of the refracted light. As a result, this technique cannot estimate the depth information with high accuracy and has limitations on the recognized distance. Moreover, this technique is easily interfered by ambient light and is not suitable under strong light examples.

TOF(Time of Flight)

Time of flight system is a LIDAR (light radar) system that emits light pulses from the emitter to the object. The receiver can determine the distance to the object by calculating the duration time since the emitter sends the light pulse until the light pulse returns to the receiver. The TOF system can simultaneously acquire the entire scene and determine the 3D image. Then the 3D image can be created using measured object coordinates.

The TOF camera includes a laser generator and a photosensitive unit which consists of a photosensitive laser or an avalanche diode. Fig.2.5 shows the setup of TOF cameras. After the laser generator emits a laser, the laser is reflected by the obstacle. The photosensitive unit in the camera senses the reflection and calculates the time required for the return trip of the laser. The distance of the obstacle from the camera is obtained by multiplying the time of flight by the speed of light.

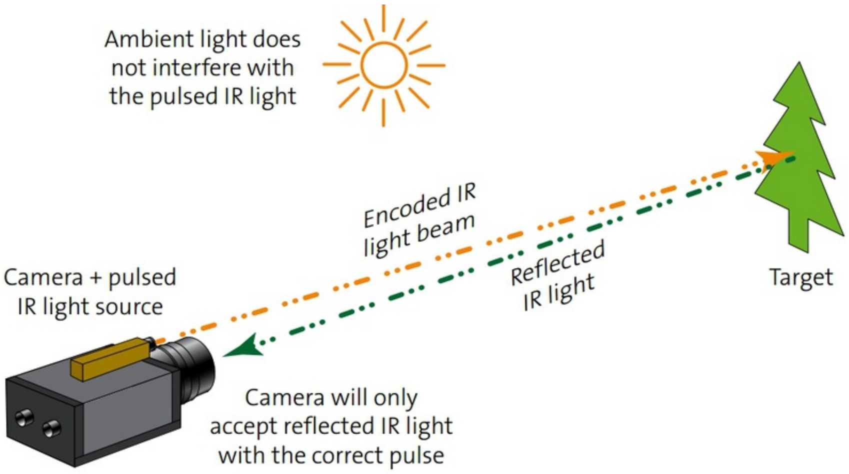

Ambient light does not interfere with the pulsed IR light
Encoded IR light beam
Reflected IR light
Target
Camera + pulsed IR light source
Camera will only accept reflected IR light with the correct pulse

Figure 2.5: TOF scanner measurement
https://www.stemmer-imaging.com/en/knowledge-base/cameras-3d-time-of-flight-cameras/

The advantage of TOF is that the response speed is fast and the depth information is relatively precise. At the same time, this technique is not easily interfered by ambient light. Therefore, it can be easily used in all kind of use cases and application. The commercial products of this type of the camera will be introduced in section.2.2.1.

Stereo vision camera

The traditional stereo vision cameras do not project any light. Similar to the human eyes, this type of camera observes the environment through two calibrated 2D cameras. Then, feature matching based on the image content is performed to calculate the depth information. Since there is no prior knowledge, this method only relies on feature matching based on the captured images. Therefore, the stereo vision camera is suited for light environments and rich feature images.

Since the image is produced based on ambient light only, this technique is suited for outdoor environments. When this type of camera is used in an indoor environment, an external light source may be required to collect the images. In this case, reflection needs to be minimal.

For areas where image features are not distinct, such as a flat ground or endless deserts, it is difficult to match features using stereo vision technique.

The biggest problem with pure stereo vision is that the whole process, including feature points searching and feature matching, requires sophisticated algorithms and calculations, and the final result may not be stable enough to be applied in real-world applications.

In real-world applications, the response time, performance under different light environments, and a range of recognized distance are important metrics to evaluate the different depth technologies. Table 2.1 shows a comparison of these three depth camera technologies.

Table 2.1: Comparison of depth camera technologies

	Structured Light	TOF (Time of Flight)	Stereo Vision
Response time	Slow	Fast	Medium
Indoor environment (low light) performance	Good	Good	Weak
Studio condition performance	Weak	Medium	Good
Outdoor environment performance	Weak	Medium	Good
Resolution	Medium	Low	High
Depth range	0.1m-6m (Limited by facula pattern)	0.1m-10m (Limited by light intensity)	Medium
Algorithm complexity	Medium	Low	High
Hardware cost	Medium	High	Medium
Disadvantages	Easy to be affected under strong light environment	The depth image has lower resolution, not suited for high precision scenarios	High algorithm complexity; Not applicable in dimly light environment; Target requires good feature changes

2.2.1 Commercial Products and Companies

The invention of the RGB-D cameras has rapidly increased the area of 3D reconstruction. The Kinect sensor, designed by the Microsoft team, appeared early in the market. Since then, many systems and applications have been deployed using the Kinect sensor. Until now, the Kinect2.0 sensor is one of the most popular RGB-D sensors for the field of Virtual and Augmented Reality. The Kinect2.0 sensor can capture all sorts of information such as the human skeleton and 3D joints. However, it has some limitations as it cannot be used outdoor and the depth range is shorter than RealSense cameras. Also it is no longer available in the market.

The RealSense depth camera D415 [90], developed by Intel, can be used in outdoor and indoor environments and output depth maps with higher frame rates of up to 90 frames per second.

The Xtion 3D [7] the camera, invented by ASUS, also provides good image resolution and has a convenient development environment. Table 4.1 shows a comparison of commercial depth camera products.

Table 2.2: Comparison of different commercial products

		Kinect 1.0	Kinect 2.0	RealSense SR300	RealSense D415	Xtion2
RGB Image	Resolution	640*480	1920*1080	1920*1080	1920*1080	1920*1080
	Frame Per Second	30fps	30fps	30fps	30fps	30fps
Depth Image	Resolution	320*240	512*424	640*480	1280*720	640*480
	Frame Per Second	30fps	30fps	60fps	90fps	30fps
Capture Distance		0.8m-4.0m	0.5m-4.5m	0.2m-1.5m	0.16m-10m	0.8m-3.5m
Working environment		Indoor	Indoor	Indoor	Outdoor and Indoor	Indoor
Technology		Structured Light	TOF	Structured Light	Active IR Stereo	TOF
Company		Microsoft	Microsoft	Intel	Intel	ASUS
Price		$149.99	$199.99	$109	$149	$269.99
In production		No	No	Yes	Yes	Yes

From the table we can conclude that compared with other cameras, the RealSense D415 can provide RGB and depth images with high-quality resolution, it has the longest capture distance and can work in the outdoor environment, which is a cost-effective depth camera in the market. In this project, we use RealSense D415 for data acquisition and pipeline testing.

2.3 3D Reconstruction Technologies Based on RGB-D Cameras

3D reconstruction techniques can be divided into offline and real-time 3D reconstruction systems. The offline 3D reconstruction systems focus on reconstruction accuracy rather than on instantaneity. Offline 3D reconstruction techniques such as Structure from Motion (SfM)[115] and Multi-view Stereo (SVM)[35] use image sequences as input to reconstruct a 3D model of the scene through camera tracking, feature extraction, and bundle adjustment. VR/AR applications by nature are 3D reconstructions of dynamic scenes and human characters. Therefore, one of the fundamental requirements in VR/AR applications is real-time 3D reconstruction.

Simultaneous Localisation and Mapping (SLAM) algorithms focus on real-time 3D reconstruction. Prior to the advent of consumer-oriented depth cameras, 3D reconstruction techniques typically used RGB cameras, such as monocular RGB cameras in the MonoSLAM[26] system. These systems use RGB images as input and reconstruct 3D models using computer graphics and computer vision techniques. With the limitation of the input data, such systems usually only produce sparse reconstruction results.

With the development of depth camera technology described in the section2.2 and the emergence of consumer-oriented depth cameras, depth camera-based 3D scanning technology and reconstruction technology have been rapidly developed. In this section, we will introduce the fundamental techniques based on the stages of the real-time 3D reconstruction system.

The typical real-time 3D reconstruction pipeline contains three phases: sensor pose estimation, data fusion and surface prediction. A common choice for the 3D reconstruction system is to reconstruct and fuse the points in the input frame to the current 3D model and reconstruct the 3D scene in real time through continuous reconstruction updates.

2.3.1 Pose Estimation

The pose estimation is used to estimate the 6 Degrees of Freedom (6-DoF) information between the input frame and the current frame or model.

ICP (Iterative closest point): The ICP algorithm[11] is a point cloud registration algorithm. This algorithm is an optimal registration method based on the least squares method[109]. The ICP algorithm iteratively selects corresponding point pairs and calculates the optimal rigid transformation until the convergence accuracy requirement is met. The ICP algorithm calculates the positional relationship of two sets of point clouds, which represent the different positions of the camera. This algorithm is based on reducing the distance between two sets of point clouds instead of feature extraction to achieve data alignment. This pose estimation method is used by several approaches[76][77][122][30][69][121]. Among them, the KinectFusion[76] system directly uses the icp algorithm to calculate the position information of the camera.

In KinectFusion pipeline, the first stage is surface measurement. This surface measurement is a pre-processing stage, the purpose is to generate the dense vertex map and a normal map of the object surface. The raw depth data captured from the depth camera consists of the pixel coordinates and the corresponding depth information. The parameters of the camera are calculated by camera calibration; the back-project method is used to obtain the 3D points in the camera coordinate, then the normal of each point is calculated according to the adjacent pixels on the depth map. In this process, the KinectFusion system uses the multi-scale method, each depth map is scaled in three layers, and the resolution of each layer is half of the next layer.

Then, in the next pose estimation step, the system uses the point-plane[22] ICP algorithm.

There are two sets of point clouds used to calculate the position of the camera (sensor). One of them is the point cloud with the normal vector calculated from the previous step, and the other is a point cloud projected by the ray casting[93] algorithm according to the pose of the previous frame. The two sets of point clouds are registered and calculated by the ICP algorithm. The rotation and translation matrix calculated from the ICP algorithm reflects the position of the camera[87].

Although the ICP algorithm iteratively calculates the pose of the camera according to the distance between the corresponding points, this system cannot realize non-rigid 3D scene reconstruction in real-time. The subsequent methods propose solutions for optimizing the pose estimation stage based on the ICP algorithm.

Dense Non-rigid Warp Field: The warp field is proposed to solve the problem of non-rigid 3D scene reconstruction by DynamicFusion[77] system. Before the DynamicFusion system, the traditional SLAM system can only reconstruct the non-rigid scenes in two ways. One way is to limit the reconstruction scene to a static view and requires a prepared template for real-time reconstruction. The other way is to reconstruct the scene offline, which requires three to four orders of magnitude more time than real-time reconstruction. In order to overcome these limitations, the DynamicFusion system was presented to reconstruct the non-rigid scene based on depth cameras in real-time.

The idea of the DynamicFusion system is to convert the dynamically changing scene (the object to be reconstructed) captured from each frame into a canonical space, and create a static object surface (canonical) model in this canonical space. Then the corresponding volumetric warp field of each frame will fuse the canonical model into a live frame. The key idea of DynamicFusion system is the warp field which is used to represent the dynamic scene motion. The 6D transformation warp function is defined as follow:

$$W(x_c) = T_{lw}SE3(DQB(x_c)) \tag{2.1}$$

In Equ.2.5, the T_{lw} represents the movement of camera. In this warp field, the amount of computation to sample every point in the canonical space is too large, which makes it almost impossible to reconstruct the surface in real-time. Therefore, the DynamicFusion defines the W by using the DQB (Dual-quaternion blending)[54] interpolation algorithm. The $SE3$ represents the transformation between quaternion and transformation matrix.

In DynamicFusion system, the pose of the camera is calculated by the parameter estimation of the warp field, the estimation of warp field parameters is defined by minimizing the energy function shown in Equ.2.6:

$$Data(W_t, V, D_t) and \lambda Reg(W_t, \varepsilon) \tag{2.2}$$

This energy function contains two terms: the ICP cost term $Data(W_t, V, D_t)$ and the regularisation term $\lambda Reg(W_t, \varepsilon)$. The V is the current reconstructed surface, the D_t represents the depth map, the ε is an edge set.

The dense non-rigid ICP data cost term is used to estimate the data-association between model and the current live frame, which is quantified by a robust regression method - the Tukey penalty function[15]. However, The data processed in this term is incomplete which contains only the currently visible points. Therefore, the second term is used to constrain the current invisible points. Those data that are presently occluded may appear as newly observed data in the subsequent frames. The $\lambda Reg(W_t, \varepsilon)$ term adds an edge constraint between the deformed nodes and then adds the cost to the energy function with the Huber penalty[47]. Also, the DynamicFusion system used the hierarchical deformation tree[112] to increase stability and reduce the computational cost.

Killing Vector Field: Based on approximately Killing vector field[108] which minimizes the killing condition. The KillingFusion[107] system propose a regularizer term to approximate deformation field and determine rigid camera motion. Besides, this system adopts SDF-2-SDF[106] registration energy to replace the ICP algorithm for camera pose estimation. Compared with the ICP algorithm, the voxel grid based SDF-2-SDF method reduces the error of large deformation.

Embedded Deformation Model: Robert Sumner proposed the embedded deformation model[112] in 2007. This method builds a deformed mesh from the model, and each node on the grid is responsible for controlling the deformation. However, this model could not reach real-time reconstruction due to the limitation of processing complexity. This was changed with the publication of the above-mentioned DynamicFusion system which proposed energy function and optimized the algorithm by using the Gauss-Newton method[60]. This was combined with implementation on a GPU to accelerate the 3D reconstruction to real-time processing.

The KinectFusion and DynamicFusion only perform real-time 3D reconstruction with a single depth camera. Single camera systems cannot completely recover the entire 3D information of the reconstructed object as they can cover one side of the object only. Also, they cannot adequately handle the topology change. The Fusion4D[30] pipeline solves these problems and performs the real-time 3D reconstruction by using multiple (eight) cameras. In the energy function proposed by Fusion4D, in addition to the two energy terms (data term, regularization term) used in the ED deformation model, an energy term of correspondence of points and an energy term of a visual hull are added.

- *Visual Hull Term:* The visual hull[59] term mainly limits the range of the energy function solution to the view frustum of the eight cameras, which effectively solves the problem that the space is too large when the occlusion problem occurs.

- *Correspondence Term:* The correspondence term is improved based on the method proposed in the paper[120]. The Fusion4D system uses a decision tree machine learning method to find the corresponding point-to-point relationship between two adjacent RGB images.

2.3.2 Data Fusion

The pose estimation calculates the spatial relationship between the input frame and the current model. Using this pose relationship, the input data and the model are fused to update the reconstructed scene. In this phase, the key component is the choice of 3D space representation. In this section, we introduce several representative models and optimization methods.

TSDF (Truncated Signed Distance Function): The truncated signed distance function[25] uses a voxel grid to represent the 3D space. In this representation, each grid in the voxel stores the distance from the grid to the object model surface. The positive and negative value is used to indicate the occluded side and the visible side of the surface. The zero crossings point is the point on the surface.

In the KinectFusion system, the TSDF model represents the reconstructed surface. In data fusion phase, the KinectFusion system fuses the point cloud of the current frame into the voxel (TSDF) model according to the pose calculated in the pose estimation step. The data fusion focuses on the representation and update of the TSDF volume. In addition to the TSDF value, each spatial location also stores a weight value w. The reconstructed surface is updated by calculating the weighted sum of the current frame contributions.

In DynamicFusion system, the TSDF representation is used to fuse the live frame depth map into the canonical space. The differences between these two systems are: first, the value of projective TSDF (PSDF in the paper) in KinectFusion is calculated in the camera space. In DynamicFusion, the value of TSDF is calculated in the world space (canonical space). Second, in the DynamicFusion system, in the process of updating the weights for each PSDF, a linear relationship is added to solve the problem of object motion and deformation.

Surfel model: Another common choice of 3D space representation is Surfel model (point representation model), which is shown in Fig.2.6:

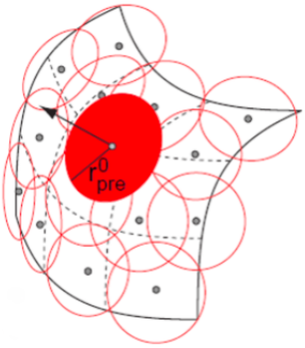


Figure 2.6: The Surfel representation model
source: from the Internet searchinghttps://pan.baidu.com/

For each point in the Surfel model, is stored: position information of the point (x, y, z); the radius of the surface patch r; the normal vector n; the color information (R, G, B) and the point capturing time t. When the system performs the fusion of points, the update methods of position information, the normal vector, and the color information are similar to the way of weighting fusion of KinectFusion. The radius of the surface patch is calculated by the distance between the surface of the scene and the center of the camera. The larger distance represents the larger radius of the surface patch.

Compared to the TSDF model, the Surfel model does not calculate and store the topology information. Therefore, the memory efficiency of Surfel model is higher[122].

Deformation Graph: The deformation graph is used to optimize the map reconstruction and update.

The deformation graph consists of nodes, which are uniformly sampled from the reconstructed Surfel model. The structure of the deformation graph is shown in Fig.2.7. The red nodes shown in the figure indicate the extracted points, the black nodes indicate other reconstructed points. These deformation nodes establish connections according to time relationship, which means that these nodes search for the nearest point according to the time relationship. The deformation graph is updated after the newly captured point cloud is merged into the global model (Surfel model).

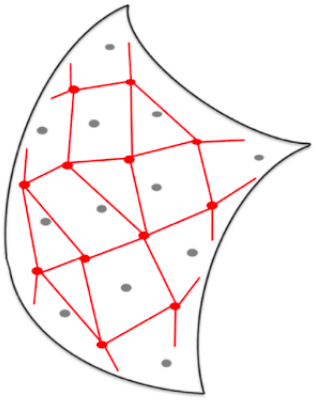


Figure 2.7: The structure of Deformation Graph
source: from the Internet searchinghttps://pan.baidu.com/

Local Loop Closure and Global Loop Closure: Using fundamental 3D space representations, point cloud registration, and energy function to fuse frames causes ghosting and occlusion problems. For example, in the case when hands are separated and then crossed if the depth camera moves following the *loop* route, the reconstruction result will overlap the two images at the same position and cause the ghosting (double images) effect.

Therefore, the system[122] uses the local loop closure and global loop closure algorithms to solve the above problems. The local loop closure algorithm is used to divide the reconstructed points into *ACTIVE* and *INACTIVE* according to time. *ACTIVE* represents that the point is reconstructed in the most recent time. *INACTIVE* is the previously reconstructed point. Then, according to the spatial relationship calculated by using the ICP algorithm and these two types of points, two sets of point cloud can be derived and registered. If these two sets of point cloud can be registered, the camera is moving in a loopback, and a constraint is established on the point to align the *ACTIVE* point to the *INACTIVE* point.

However, the local loop closure algorithm is not suited for all situations. When the camera moves in a large distance or degree, the error will accumulate. When there is a loopback movement of the camera, the corresponding points of *ACTIVE* and *INACTIVE* cannot overlap, the global loop closure (the randomized Ferns [39]) algorithm is required to set constraints and match point cloud.

Error Correction Mechanism: An important contribution of Fusion4D system[30] to data fusion phase is the fusion error correction mechanism. When the model reconstructed from the previous frame does not match the current frame well, the current frame replaces the previously unmatched portion as the $keyframe$. Instead of the original reference model, the subsequent data fusion and point cloud registration will be based on this $keyframe$.

CNN-based Fusion: The combination of the SLAM system and the neural network tries to make the machine better ”understand” and reconstruct the surrounding environment. The SemanticFusion[69] system applies the CNN network into SLAM system, the process of this pipeline is: the CNN receives the 2D images and returns a classification probability distribution for each pixel, then the Bayesian update scheme will track the classification probability distribution for each surface and then update these probabilities based on the CNN prediction by using the data correlation provided by the SLAM system. Finally, the Conditional Random Field (CRF) regularisation[58] framework is used to improve the semantic prediction.

The SemanticFusion system adopts the ElasticFusion[122] as the real-time SLAM system. The reason is that the ElasticFusion based on the deformation graph which does not need to destroy the geometry correlation or the probability distribution, which is suited for the semantic label fusion in the SemanticFusion pipeline.

The CNN model applied in this system based on the Caffe framework[50]. The system uses the Deconvolutional Semantic Segmentation Network[79] architecture and is additionally equipped with the max pooling and deconvolutional training to output a dense pixel-level semantic probability map.

Compared to the Surfel model in the ElasticFusion pipeline, the Surfel model used in SemanticFusion pipeline contains an additional discrete probability distribution. In the Bayesian update scheme stage, these probabilities are updated based on the CNN predictions from the CNN Architecture stage.

2.3.3 Surface Prediction

The KinectFusion system uses the ray casting algorithm to derive the point cloud from the current frame view of the model based on the current frame camera pose, and then calculates its normal vector for registering the next input frame to obtain the posture of the next frame.

For systems that use the deformation graph to optimize the data fusion process, in the surface prediction stage, new deformation nodes are inserted, and the regularisation graph is updated to show the deformation result. Due to the change of each frame, the deformation nodes set is continuously updated. The new uncovered deformation nodes in the reconstructed surface and it is *k* nearest nodes need to be added into the nodes set.

2.3.4 System Overview

The real-time 3D reconstruction pipelines introduced in this section contain three stages: pose estimation and tracking, data fusion and surface prediction. The ICP algorithm is used for camera position tracking and estimation in all mentioned systems. In addition to the ICP algorithm, the DynamicFusion system defines the warp field, the KillingFusion system defines the killing vector field, and the ElasticFusion system optimized the estimation by using the deformation graph. Except for the KinectFusion and Kintinuous systems, all of the rest systems use the energy function to constrain the deformation.

In the data fusion stage, most systems use the TSDF representation for data fusion. The ElasticFusion uses the Surfel model and adds local and global loop closure algorithms to solve the ghosting problem. The Fusion4D system optimized the fusion stage based on the ED model and the error correction mechanism. Table 2.3 shows the comparison of mentioned real-time 3D reconstruction systems.

Method	**Input**		**Pose Estimation**						**Data Fusion**					**Surface Prediction**		**Year**
	Multi-View	Single-View	ICP	Warp-Field	Killing-Field	Energy Function	Deformation Graph	ED Model	TSDF	Surfel Model	CNN	Loop Closure	Error Correction	Inactive Model	Projection	
KinectFusion[76]	✓	-	✓	-	-	-	-	-	✓	-	-	-	-	-	✓	2011
Kintinuous[121]	✓	-	✓	-	-	-	-	-	✓	-	-	-	-	-	-	2012
DynamicFusion[77]	✓	-	✓	✓	-	✓	✓	-	✓	-	-	-	-	-	-	2015
ElasticFusion[122]	✓	-	✓	-	-	✓	✓	-	-	✓	-	✓	-	✓	-	2016
Fusion4D[30]	-	✓	✓	-	-	✓	-	✓	✓	-	-	-	✓	-	-	2016
SemanticFusion[69]	✓	-	✓	-	-	✓	✓	-	-	✓	✓	✓	-	✓	-	2016
KillingFusion[107]	✓	-	-	-	✓	✓	-	-	✓	-	-	-	-	-	-	2017

Table 2.3: Comparison and overview of state-of-the-art 3D real-time reconstruction systems.

In this project, we focus on the pose estimation phase. The output of this phase is the transformation matrix, which is used to represent the spatial relationship between different cameras. The key algorithm used in this stage is the ICP algorithm. In the next section, we will introduce the theory of the ICP algorithm in detail.

2.3.5 The ICP Algorithm

The ICP algorithm consists of two parts: corresponding points search and pose estimation. The output of the ICP algorithm is an optimized transformation matrix. The typical ICP algorithm includes the following steps:

- *Closest point set search:* Search the closest point set Q according to the coordinates of the target point set P. The distance comparison between the target point set and the nearest set is a continuous process until the closest point set Q is found.

- *Calculate the center of gravity of two sets and generate a new point set:* Calculate the barycentric coordinates of P and Q, and generate a new point set by calculating the difference between each point and the gravity coordinates.

- *Calculate the covariance matrix:* Calculation of the covariance matrix, the largest eigenvalue of the covariance matrix and its largest eigenvector. These values will be used to calculate the rotation matrix in the next step.

- *Calculate the rotation matrix:* When the residual sum of squares is the smallest, the largest eigenvector calculated in the previous step is equivalent to the rotated quaternion. The quaternion is converted into a rotation matrix R.

- *Calculate the translation matrix:* Given the determined rotation matrix R, the translation matrix t is the gravity difference of the two point sets. The translation matrix can be determined by the center of gravity points and the rotation matrix.

- *Calculate the iteration judgment value:* The calculated transformation matrix transforms the point set P into a new point set P. Then, the iterative judgment value is calculated by using the Equation(2.3).

  $$I = \frac{1}{n}\sum_{n}^{i=1} |q_i - Rp_i - T|^2 \tag{2.3}$$

  The I is an iteration judgment value, n is the total number of point sets, q_i is the coordinate vector of the point set Q, p_i is the coordinate vector of the target point set.

- *Judge the end of iteration:* In this step, we need to set a threshold to determine when to terminate the iteration process. When the iterative determination value is less than the threshold, the iteration process ends. If not, it continues with the previous procedure.

There are also many ICP variants[84], the point-to-point and point-to-plane are the most popular of all these variants. Also, the combination of the modified K-D tree [126] and ICP algorithms provide good performance. In this project, we use the traditional ICP algorithm for registration and propose a feature-based improvement method.

2.4 Camera Calibration

The ICP algorithm relies on the original transformation matrix of point clouds. The camera calibration process provides a coarse estimation of the relationship between cameras. There are two popular approaches for camera calibration, one is based on the MATLAB calibration toolkit[13] and the other one using the OpenCV library[14].

The report by Suriansky and Cmarada[114] analyzed different camera calibration methods and compared the performance of the methods based on OpenCV and MATLAB. By evaluating the re-projection error and processing time, the experiment results indicate that the MATLAB calibration toolkit is more accurate, while the OpenCV-based calibration method is much faster and easier to use. The main disadvantage of the MATLAB camera calibration toolkit is that it requires manual operation steps, which is a time-consuming process. Until now, some MATLAB toolkit can automatically calculate the boarders of the calibration pattern. However, this automatic process leads to poorer accuracy. To obtain a more accurate calibration result, we need to manually determine the borders of the calibration pattern for each calibration image.

In this project, we adopt the MATLAB calibration toolkit[13] to calculate the transformation matrix because the results of the coarse registration provide a rough estimate of the transformation matrix and require high calibration accuracy. Once the position of the camera is fixed, the coarse registration step will be a one-time process. This MATLAB toolkit mainly follows Zhang's calibration algorithm[127] published in 1999.

In the coarse registration process, the calibration pattern is used to *calculate the intrinsic and extrinsic parameters*. The toolkit used in this project adopts corner detection and uses the checkerboard pattern as shown in Fig.2.8.

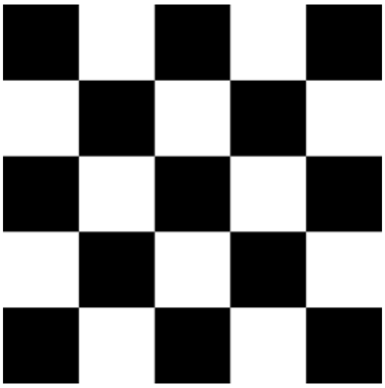


Figure 2.8: Checkerboard pattern
source:https://zh.wikipedia.org/wiki/File:Checkerboard_pattern.svg

The coarse registration process includes the following steps: single camera calibration, stereo system calibration, and transformation matrix calculation.

Single Camera Calibration

The single-camera calibration step calculates the intrinsic and extrinsic parameters of the camera. The optical measurement has four coordinates: pixel coordinate, image coordinate, camera coordinate, and world coordinate. The calibration process requires the transformation of the point P in world coordinates into pixel coordinates. The correlation of these coordinates is shown in Fig.2.9.

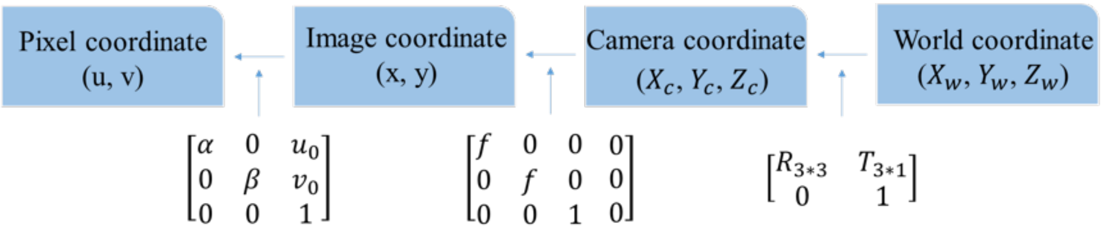

Pixel coordinate (u, v)
Image coordinate (x, y)
Camera coordinate (X_c, Y_c, Z_c)
World coordinate (X_w, Y_w, Z_w)
$\begin{bmatrix} \alpha & 0 & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix}$
$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
$\begin{bmatrix} R_{3*3} & T_{3*1} \\ 0 & 1 \end{bmatrix}$

Figure 2.9: Correlations among the pixel coordinate, the image coordinate, the camera coordinate and the world coordinate

Each point $P = (X_w, Y_w, Z_w)$ in the scene will be transformed into a pixel point $p = (u, v)$ in the 2D image, Fig.2.9 shows the following three transformations:

1. Transform the point P from the world coordinates to the camera coordinates by a rigid transformation, which uses the relative pose between the cameras. The R matrix and t matrix are the *extrinsic parameters*
2. Transform from the camera coordinate to the image coordinate $p = (x, y)$
3. Transform the point p from image coordinate to pixel coordinate $p = (u, v)$

The transformation process can be concluded into the multiplication of the matrix:

$$\begin{aligned} \mathrm{s}\begin{pmatrix} u \\ y \\ 1 \end{pmatrix} &= \begin{bmatrix} \alpha & 0 & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \\ &= \begin{bmatrix} f_x & 0 & u_o & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \end{aligned} \tag{2.4}$$

The α and β are the number of pixels per unit distance on the image. And $f_x = \alpha f, f_y = \beta f$ transforms the camera focal length f to pixel metrics in the x and y directions. The parameter γ represents the distortion of two axes in the pixel coordinate. Then the *intrinsic parameter* of a camera is K:

$$K = \begin{bmatrix} f_x & \gamma & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

The calibration process aims to calculate the *intrinsic and extrinsic parameters.* Zhang's algorithm includes the following steps:

Homography between the model plane and its image: The mapping between two planes. In Zhang's calibration algorithm, the *checkerboard* (shown in Fig.2.8) used for calibration is a plane Π in a 3D scene, and the image in the image plane is another plane π. The coordinates of the corresponding points in these two planes can be used to calculate the homography matrix H. The coordinates of the corner points in the checkerboard are labelled by ourselves; the corner extraction algorithm can obtain the corners in the image. Based on Equation(2.4), the correlation between the pixel point p and checkerboard point P is:

$$p = K[R|t], H = K[R|t] \tag{2.6}$$

Equation(2.8) indicates that the calculation of intrinsic and extrinsic parameters can be deducted from homography matrix H. The checkerboard used here is a plane. Therefore, we build the world coordinate into a plane: $Z = 0$. Then, Equation(2.4) turns into:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} r_1 & r_2 & r_3 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{2.7}$$

$H = \lambda K \begin{bmatrix} r_1 & r_2 & t \end{bmatrix}$ is the homography matrix.

Calculate the intrinsic parameter matrix K by using constraint conditions: The rotation matrix R can be represented as:

$$\begin{aligned} \mathrm{r}_1 &= \lambda K^{-1} h_1; \\ \mathrm{r}_2 &= \lambda K^{-1} h_2; \\ t &= \lambda K^{-1} h_3; \end{aligned} \tag{2.8}$$

The rotation matrix R is an orthogonal matrix, which means that:

$$\begin{aligned} \mathrm{r}_1^T r_2 &= 0; \\ \|r_1\| = \|r_2\| &= 1; \end{aligned} \tag{2.9}$$

With this constraint equation, the intrinsic matrix parameters can be calculated.

Calculate the extrinsic matrix and optimization: Using the knowledge of the intrinsic matrix, the extrinsic matrix can be easily calculated. The Zhang's algorithm provides several ways to optimize the calculation: maximum likelihood estimation and eliminate radial distortion. The MATLAB toolkit used in this project also adopts algorithms to optimize the calibration results.

Stereo system calibration

In our project, we use four RealSense cameras to capture the point cloud of an object. The first step is to calibrate all cameras separately. Then we need to calculate the spatial correlation between all neighbouring camera pairs. The stereo system calibration uses the intrinsic and extrinsic matrix parameter obtained from a single camera calibration step to calibrate the stereo system and determine the relative positions of the two cameras.

2.5 Point Cloud Feature Extraction and Segmentation Based on PCL

In our project, we use the functions in PCL to process point cloud data. In this section, we introduce the main modules of PCL and point cloud segmentation and feature extraction algorithms based on PCL.

2.5.1 PCL

The Point Cloud Library (PCL) is an open source project library developed since 2011[89]. It is a cross-platform C++ programming library for point cloud processing and 3D geometry processing, which contains point cloud-related algorithms involve point cloud acquisition, filtering, segmentation, registration, recognition and feature extraction. PCL supports multiple operating system platforms including Windows, Linux, Android, Mac OS, and some embedded real-time systems.

Structure and Modules of PCL

For 3D point cloud processing, PCL is a modular C++ template library for 3D point cloud processing. It is based on the following third-party libraries: Boost, Eigen, FLANN, VTK, CUDA, OpenNI, Qhull. PCL utilizes high-performance computing technologies such as OpenMP[12], GPU and CUDA to improve program real-time performance through parallel computing. All modules and algorithms in PCL use Boost shared pointers to transfer data and to avoid the need to copy existing data in the system. The main modules and functions of PCL include:

libpcl filters: implement filters such as sampling, removing outliers and feature extraction.

libpcl features: implement feature extraction such as surface normals, curvatures, boundary point estimation, moment invariants, principal curvatures, point feature histogram and rotation-invariant feature transform (RIFT).

libpcl I/O: implement data input and output processing, such as reading and writing of point cloud data files (PCD, PLY).

libpcl segmentation: implement clustering extraction, model fitting and polygonal prism extraction.

libpcl surface: implement surface reconstruction algorithms such as mesh reconstruction, convex hull reconstruction, and moving least squares smoothing.

libpcl registration: implement point cloud registration methods such as Iterative Closest Point (ICP).

libpcl key points: implements extraction of different critical points in pre-processing step.

libpcl range: depth images processing generated by different point cloud datasets.

2.5.2 FPFH Descriptor

Fast Point Feature Histograms (FPFH) is a simplified calculation method feature descriptor based on Point Feature Histograms (PFH), which is used to describe the local geometric features of the 3D point cloud. FPH was proposed by R. B. Rusu in 2008[96][95]. In this section, we will first introduce the theory of PFH and then illustrate the FPFH descriptor.

Point Feature Histogram (PFH)

For a particular point in the point cloud, the surface normal and estimated curvature are fundamental geometric feature descriptors. Point Feature Histogram is a point cloud geometry feature descriptor which combines information from 3D axis data and surface normals so that more information in the point cloud can be captured.

PFH is one kind of quantification of the spatial difference of the central point. This histogram is used to describe the geometric information of the neighborhood by using mathematical statistics methods. FPH aims to encode the geometric features of the neighborhood mean curvature of a point into a multidimensional histogram, such high-dimensional data provides an informative feature representation.

PFH is based on the relationship between points and their k neighborhoods and their normals, which means that it will capture the variation of the sampling surface by considering all the interactions between the normals to describe its geometric features.

Fig.2.10 shows the PFH calculation influence area of a center point (P_q), where (P_q) (the red point) is the center point of the sphere with radius r in 3D space. The center point and all its k neighborhood points inside the sphere of radius r are connected to form a network, and the final PFH geometric feature will be calculated by computing the relationship between all the pairs of points in the neighborhood. Therefore, the calculation complexity of PFH for each point is $O(k^2)$.

In order to calculate the relative difference between two points, given two points p_1 and p_2 and the respective normals n_1 and n_2, a local coordinate system is defined at one of the points and three unit vector u, v, and w follow the rule:

$$u = n_1; v = u \times \frac{p_2 - p_1}{\|p_2 - p_1\|_2}; w = u \times v \tag{2.10}$$

Where $d = \|p_2 - p_1\|$ represents the Euclidean distance between two points p_1 and p_2. Using the above u, v, w coordinate system, the difference between n_1 and n_2 can be represented by three angles $(\alpha, \varphi, \theta)$:

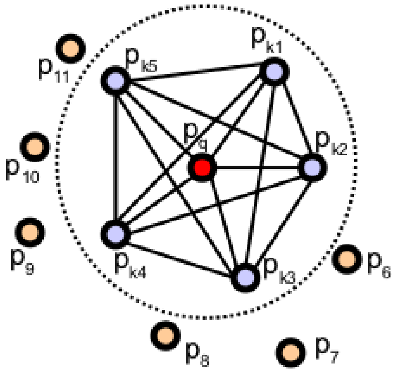


Figure 2.10: The FPH calculation area of center point (P_q)
source: http://pointclouds.org/documentation/tutorials/pfh_estimation.php

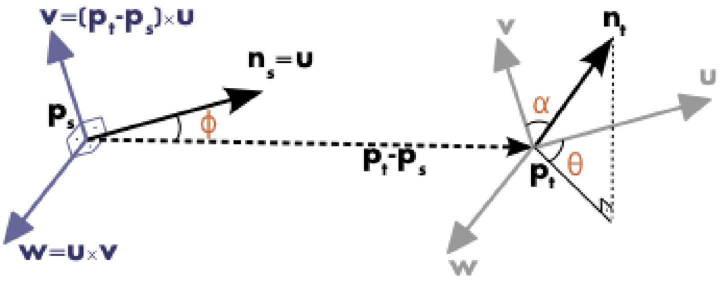


Figure 2.11: The spatial coordinates of points p_1 and p_2.
source: http://pointclouds.org/documentation/tutorials/pfh_estimation.php

$$\alpha = v \cdot n_2; \varphi = u \cdot \frac{p_1 - p_2}{\|p_2 - p_1\|_2}; \theta = arctan(w \cdot n_2, u \cdot n_2) \tag{2.11}$$

The spatial coordinate of these two points is shown in Fig.2.11. The quadruplet $(\alpha, \varphi, \theta, d)$ is then calculated for all pairs of points in the k neighborhood. This kind of expression uses three parameters to cover 12 parameters in the original information of two points (the position and the normal of each point need three parameters to express spatial information). Then, the quadruplet is placed into sub-intervals of the histogram to form the PFH feature representation.

Fast Point Feature Histogram (FPFH)

Based on the theory of PFH, given the point cloud P containing n points, the complexity of calculating the PFH feature of all points in the point cloud P is $O(nk^2)$, where k is the number of neighbors of a point p_i in P. Therefore, the efficiency of calculating PFH features is low, and such algorithm complexity cannot be realized in real-time applications. FPFH can preserve most of the characteristics and approximates results of PFH and reduce the computational complexity to $O(n_k)$.

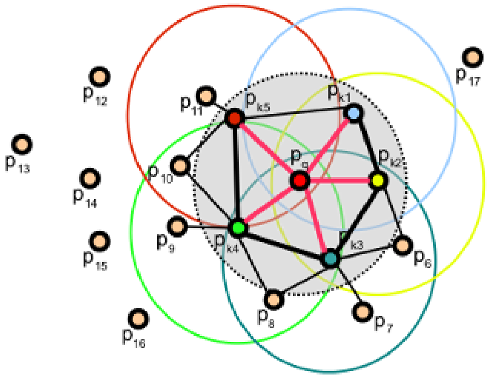


Figure 2.12: The k neighborhood influence range graph centered on the point p_q
source: http://pointclouds.org/documentation/tutorials/fpfh_estimation.php

The input of FPFH is a point cloud with normal information, and the output is a histogram that reflects the neighborhood-related features around each point. The difference between FPFH and PFH is that FPFH takes some simplification and optimization methods in the following way:

FPFH simplifies and optimizes the calculations in two steps to speed up the calculation:

- For each point p_q, calculate the quadruplet $(\alpha, \varphi, \theta, d)$ between point and each of its neighborhoods similar to the method used in PFH, and obtain a Simplified Point Feature Histogram (SPFH).
- Re-determine the k neighborhood of each point and use the neighboring SPFH values to weigh the calculated histogram of the point p_q. The calculation of FPFH follows:

$$FPFH(p_q) = SPFH(p_q) + \frac{1}{k} \sum_{i=1}^{k} \frac{1}{w_k} \cdot SPFH(p_k) \tag{2.12}$$

  Where the weight w_k depends on the distance between the center point p_q and a neighboring point p_k to measure the weight of the point pair (p_q, p_k).

Fig.2.12 shows the k neighborhood influence range graph centered on the point p_q. As shown in the figure, given a point p_q, the algorithm first calculates the SPFH of the point p_q and its neighborhood (connected by the line marked in red) and then perform this step on all points in the point cloud. Comparing with PFH, FPFH lacks the connection and influence between neighborhood points. Therefore, it is necessary to weight the SPFH of all its neighborhood points p_k and combine it with SPFH of p_q to update the final FPFH of the point p_q.

The calculation of SPFH is based on the point pairs formed by the center point and its neighbors, while the PFH also requires point pairs between the neighborhoods, so the computational complexity of the FPFH is significantly reduced to $(O(n_k))$, making it applicable in real-time applications.

2.5.3 RANSAC Segmentation

Point cloud segmentation subdivides point clouds according to features such as space, geometry, and texture, which produces point cloud segments with similar characteristics.

Random Sample Consensus (RANSAC) is an algorithm for calculating valid mathematical data by iteratively calculating the mathematical model parameters according to a set of sample data. RANSAC was proposed by Fischler and Bolles[34] in 1981.

The RANSAC algorithm has an underlying assumption: the sample contains correct data (*INLIERS*, valid data that fit the model) and abnormal data (*OUTLIERS*, invalid data that do not match the model). If valid data is the majority (more than 50%), the parameters of the model can be determined by the least squares method.

The input of the RANSAC algorithm is a set of observations and a parametric model that can be interpreted or adapted to the observed data. The RANSAC is iteratively selecting a random set of data in the dataset. The selected subset is assumed to be *INLIERS* points, and it verifies the model by the following steps:

- There is a model that is adapted to the assumed *INLIERS* points, and all unknown parameters can be calculated from the assumed *INLIERS* points;
- Test all other data using the model obtained in the last step. If a point fits the estimated model, it is considered to be a *INLIERS* point;
- If there are enough points to be classified as *INLIERS* points, then the estimated model is reasonable;
- Re-estimate the model with all hypothetical *INLIERS* points;
- Evaluate the model by estimating the error of *INLIERS* points and model.

In point cloud segmentation, the RANSAC first need to determine a segmentation model, such as a plane, cone, cylinder. Then, by determining the valid *INLIERS* points in the model to find the point cloud that fits the model. Finally, the desired shape will be segmented.

2.6 3D Data Deep Learning Network

Deep learning had developed rapidly in recent years and has attracted widespread attention in research and industry areas. However, deep learning is not a *new* research subject. It has gone through a long history of development.

In 1943, the psychologist McCulloch and the mathematical logician Pitts proposed the McCulloch-Pitts (MP) model[70]. The MP model is a neural network-based mathematical model that simulates the structure and working principle of *neurons*. Then, the Canadian psychologist Donald Herb proposed the Hebb Rule[42] based on unsupervised learning. The Hebb rule mimics the process of human recognition behavior to establish a ”network model.” This network model performs a large number of training processes on the training set and extracts the statistical features of the training set, and then classifies samples according to the similarity degree. These samples which have high similarity are grouped into one class. The Hebb learning rule is consistent with the condition response mechanism, which lays a foundation for future neural networking learning algorithms. In the late 1950s, based on the study of the MP model and the Hebb learning rule, Rosenblatt proposed a perceptron-based learning algorithm[92] similar to the human learning process. In 1958, the neural network ”perceptron”[92] consisting of two layers of neurons was proposed. The perception is a linear model that can classify the input training set data and automatically update the weights in the training set. The proposed perceptron had attracted a large number of scientists' interest in artificial neural network research. However, in 1969 Marvin Minsky and Simon Piper proved that the single-layer perceptrons could not solve linear *indivisible* problems. Due to this flaw and the failure to promote the perceptron to the multi-layer neural network, the research on neural networks has been stagnant for nearly 20 years.

In 1982, the physicist John Hopfield popularized the Hopfield neural network[46]. The Hopfield neural network is one kind of cyclic neural network combining the memory system and the binary system. The Hopfield network can simulate human memory. Depending on the choice of activation function, there are two types of functions: continuous and discrete functions which are used to optimize computation and associative memory. However, due to the defect that it was easy to fall into the local minimum, this algorithm had not been applied widely. Until 1986, Geoffrey Hinton proposed the back-propagation (BP) algorithm[94] which is a backpropagation algorithm for the multi-layer perceptron. Based on the forward propagation of traditional neural networks, the BP algorithm increases the backpropagation process. The backpropagation process continually adjusts the weights and thresholds between neurons until the error in the output is reduced to within the allowable range, or until a pre-determined amount of training is reached. The BP algorithm solves the problem of nonlinear classification. When the size of the BP network increases, it will lead to a vanishing gradient problem, which limits the development of the BP algorithm. In the training iteration, the update of the weight of the neural network is proportional to the derivative of the error function, but in some cases, the gradient will disappear, the vanishing gradient will result in the weight not being updated, and the neural network cannot continue training. Besides, in the middle 1990s, other shallow machine learning algorithms represented by the support vector machine (SVM) [24] were proposed. The shallow machine learning algorithm obtained good results on classification and regression problems.

In 2006, Geoffrey Hinton and Salakhutdinov published an article[44] which gave a solution to the problem of vanishing gradient. The method first trains model layer-by-layer by using unsupervised learning methods and then uses supervised backpropagation. In 2012, the team led by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton proposed the AlexNet[2], AlexNet used the ReLU (Rectified Linear Units) activation function [75] to fundamentally solve the gradient disappearance problem, and it uses the GPU to improve the speed of the model significantly. In the ImageNet Image Recognition Competition, the AlexNet achieved the best result and won the championship. Since then, with the advancement of deep learning and the improvement of data processing capabilities, the deep learning related algorithms achieved good results in many fields such as for healthcare, arts, and self-driving cars. For example, the invention of AlphaGo and AlphaGo Zero proved the good performance of deep learning technology.

Nowadays, Deep learning has been applied in various fields. For example, in computer vision deep learning applications include face recognition[113], object detection[91][38] and object tracking[65].

2.6.1 Deep Learning With 3D Dataset

Deep learning algorithms have been applied more in Euclidean structured data such as 1D acoustic wave signal, 2D picture, and video. Deep learning has been applied in various industries and fields. However, there are still many problems to be overcome. One of the most critical issues is how to detect and identify objects in 3D space.

The recognition and segmentation of 3D models can be applied in many fields, for example, virtual reality, robot control, and autonomous. Taking virtual reality as an example, the depth sensor captures the depth and color information of surrounding scenes, based on this information, the algorithm infers the relevant semantics, predicts and divides the scene. Here we need to create a set of non-Euclidean data that is similar to spatial coordinates as the input. The output is the detection or segmentation of objects.

In general, we can use the following three structures to represent 3D data: voxel grid, multi-view, and point cloud. The multi-view structure can be translated to use 2D Convolutional Neural Network (CNN) resolution on multiple images. For 3D voxels, 2D CNN can be made to work for 3D voxels by defining a 3D convolution kernel. Therefore, in the early stage of 3D data deep learning research, a lot of 3D voxel grid and multi-view neural networks were proposed. With the development of data processing, deep learning based on point clouds has also been proposed in recent years. In this section, we will introduce representative deep learning neural networks based on different 3D representations.

2.6.2 Voxel-based CNN (ShapeNets)

The core idea of voxel-based technology is to convert the input 3D shape data representation (such as build the depth map based on the distance from the visual observation point to the surface of the scene) into a standard 3D data representation (volumetric representation), and proposes a network framework for convolution of regular, fixed-sized 3D cubes (voxel grids) for model identification segmentation. There are currently many deep networks based on voxel 3D data such as 3D-CNN[67], VAE[16], VoxNet[68] and ShapeNet[124]. In this section, we will introduce the ShapeNet in details.

The ShapeNets solves the problem of using 3D data for object detection and recovering the entire 3D shapes from 2.5D maps. The 3D ShapeNets is proposed based on the CDBN (Convolutional Deep Belief Network)[62] network:

- Learn the 3D shape distributions of different types and different poses and getting hierarchical expressions.

- Support joint object recognition, which can restore 3D maps from 2.5D depth maps, active object recognition through view planning.

Two difficulties in 3D shape data recognition are category recognition and reconstruction of 3D shape from 2.5D shapes (shape completion). The limitations of previous approaches to solving these issues are:

- Shapes of various objects with large changes: In order to build deformable component models, most work[20][116][52] use assembly-based methods, which means that most previous research is part-based, for example, divide a table into table face and legs. This method restricts to a small difference within the class.
- Surface reconstruction of imperfect scanning input: Most of the previous work[103][8] was based on smooth interpolation or extrapolation, which can only solve image loss. Moreover, the results of previous methods are limited by the image quality.
- 3D data: Input for object recognition, in general is 2D data, not 3D.
- Complex real-world object 3D shapes: Use the Shape Boltzmann Machine[31] to generate objects can effectively capture the object intra-class variation. This paper is inspired by ShapeBM to generate models and learn the expression of 3D shapes.
- The object cannot be recognized in a single view, or the object cannot be reconstructed: This method adopts the Next-Best-View[102] to solve this problem.

ShapeNets includes two variants: The *3D ShapeNets* is used to represent geometric 3D shape through the probability distribution of the binary variables on the 3D voxel grid. *2.5D recognition and reconstruction* are used to recognize and reconstruct objects.

3D ShapeNets

The Fig.2.13 shows the structure of the 3D ShapeNets model. The first three layers are convolutional RBM, the fourth layer is a standard fully connected RBM with 1200 nodes, and the inputs to the top layer are labelled variables and Bernoulli characteristic variables. The model training process includes the layer-by-layer pre-training process and the full network fine-tuning process. The pre-training uses the standard Contrastive Divergence[43] to train the lower four-layer RBM. The wake-sleep algorithm[45] is used in the fine-tuning process.

2.5D Recognition and Reconstruction

This process has two parts: View-based sampling and Next-Best-View prediction.

View-based sampling: The model was trained on complete 3D shapes, it can identify objects in a single view 2.5D depth map. In this process, the 2.5D depth map will first be represented as volumes. The sampling process uses Gibbs sampling to approximate the posterior distribution. The result map to the most frequently sampled class.

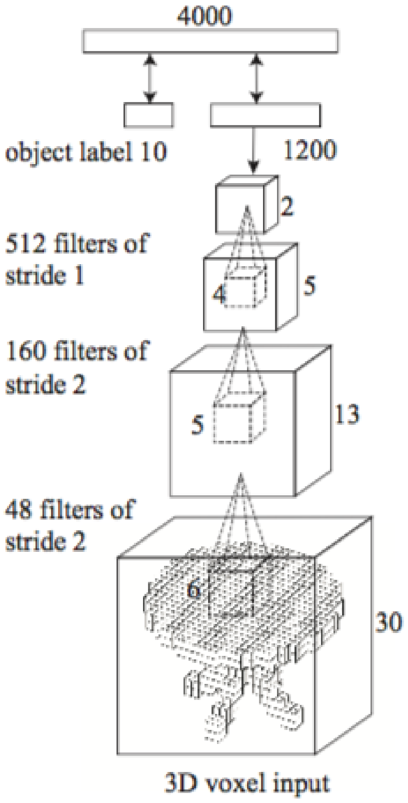


Figure 2.13: Architecture of 3D ShapeNets model
source:3D ShapeNets: A Deep Representation for Volumetric Shapes[124]

Next-Best-View prediction: This method uses the volumetric representation as the input data of ShapeNets for object reconstruction. When there are three kinds of reconstruction results, the reconstruction is *uncertain*. Then it needs to obtain the next-view data to determine the final classification result.

There are four steps to obtain the next view: first, find all different next-view candidates; next, determine the possible shapes of next-view candidates in the original light; then calculate the uncertainty of the corresponding shape of all candidates; finally, the next-best-view is the one with the least uncertainty.

After deciding the best view, the camera will capture the other surface of the object from the best view. The prediction and reconstruction deduce a new observation.

Voxel-based neural networks can effectively detect and segment 3D data, but using voxel as representation can result in massive memory usage during the calculation process and the long learning time.

2.6.3 Multi-View CNN

Different from the voxel-based method, the core idea of the multi-view neural network is to extract surface features with multiple 2D images from different angles. Segmentation and classification of 3D objects can be achieved by processing corresponding 2D images. In this section, we will introduce a representative Multi-view CNN[111] proposed in 2015.

The innovation of this method is that it uses 2D images from different viewpoints of the 3D object as training data. The model is trained with a classic and mature two-dimensional image convolution network.

The model trained by this method has much better recognition and classification results on 3D objects than the model directly trained with 3D data. Compared with direct 3D data training, significant advantages are the maturity and speed of 2D image convolution networks. The use of 3D images to process 3D data is equivalent to the ”dimensionality reduction” of 3D training data, which can also result in a better performance than 3D data.

In this method, the way to use 2D images is to generate a descriptor for each view; these descriptors will be used for recognition based on a voting or ranking mechanism. The method has the following steps:

- *Multi-View Representation:* In this step, this system adopts the Phong reflection model[83] to generate the rendered polygon meshes.
- *Recognition with Multi-View Representations:* Each 3D shape produces 12 or 80 images to recognize an object, multiple 2D images need to be integrated to describe the 3D shape features.
- *Image descriptors:* This method considers two kinds of 2D image descriptors: one image descriptor is based on Fisher Vectors[100] and multi-scale SIFT, another descriptor is based on CNN activation features[29].
- *Classification:* Multi-view CNN adopts linear SVMs to classify 3D shapes.
- *Retrieval:* This step defines a measure of distance and similarity.

For most 3D shapes, the multiple descriptors mentioned before has better performance than other 3D shape feature descriptor. However, in many cases this algorithm is inefficient. Efficient and better performance requires the integration and fusion of descriptors.

Finding the average of the feature descriptors of a 3D shapes or joining these feature descriptors together can lead to bad results. Therefore, this paper proposed the Multi-view CNN (MVCNN) as shown in Fig.2.14 to fuse feature descriptors.

In this neural network, each view image of the same 3D shape passes through the first layer of CNN1 convolution network, all these images will be aggregated in the view-pooling layer, and the output will be then sent into CNN2 convolution network.

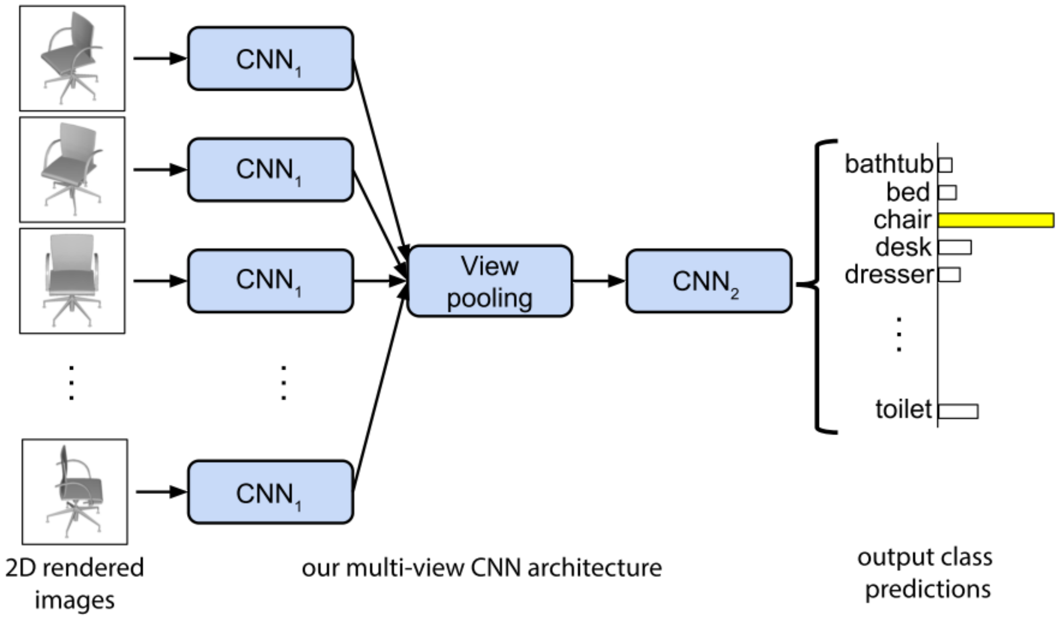


Figure 2.14: Multi-view CNN architecture
source: Multi-view Convolutional Neural Networks for 3D Shape Recognition[111]

The disadvantage of MVCNN is that because of self-occlusion of the object, some surface information is not available when the picture is taken, and the angle is chosen manually. These factors will influence the recognition accuracy.

2.6.4 Point-based CNN (PointNet++)

The CNN based on volumetry and multi-view have one thing in common. They both require to re-cut the original 3D data or map 3D data to 2D space to get a highly regular format as network input. The most significant difference between the other two types of CNN and point-based CNN is that the point-based CNN can directly process the original 3D data. In this section, we will introduce the Point-based CNN PointNet[85].

PointNet is based on three types of work: point cloud features, deep learning on 3D data and deep learning on unordered sets.

Point Cloud Features: The characteristics of point clouds are usually divided according to the specific case. For example, based on the invariants to certain transformation, the statistical properties of the points can be divided into intrinsic[9][17] and extrinsic[21][51] properties. Alternatively, features can also be classified into local features and global features.

3D Data Deep Learning: In addition to the aforementioned Volume CNN and the Multi-view CNN, there are other types of deep learning on 3D data: Spectral CNN[18][66] and Feature-based CNN[33][40]. The drawback of spectral CNN is that it cannot be used for

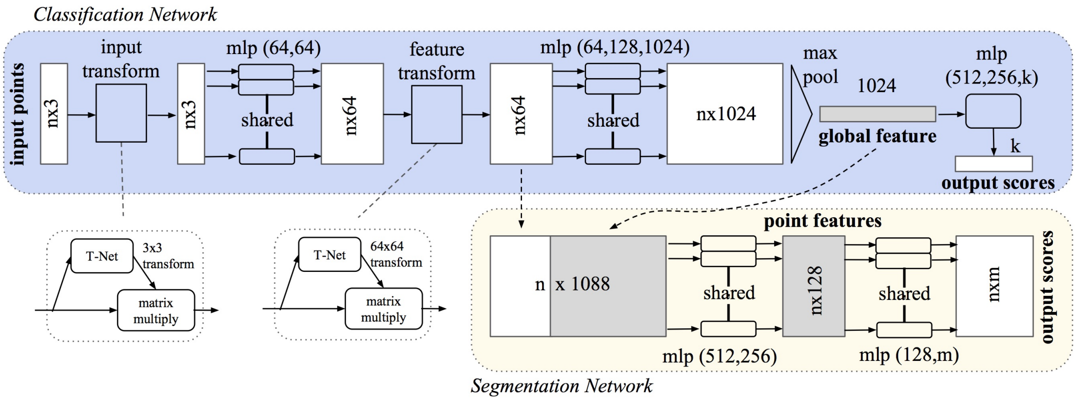


Figure 2.15: PointNet Architecture: The ”mlp” represents multi-layer perceptron, Batchnorm is used for all layers with ReLU.
source: PoineNet: Deep learning on point sets for 3D classification and segmentation[85]

3D data with non-isometric shapes. The feature-based neural network converts 3D data into a vector and uses the network to classify shapes.

Unordered Sets: A point cloud is a set of vectors. Neural networks require regular format data such as images and volumes as network input. Recent work from team Oriol[118] solves the unordered problem by using a ”read-process-write” network. However, their work ignores the geometry of sets.

Given the features of point cloud data, there are two challenges in the point-based deep learning network:

- Unordered: Unlike regular format data such as images or volumetric grids, point clouds consist of a series of points without a specific order.
- Transformation: The coordinate of point cloud will be changed when it undergoes a certain rigid change (rotation or translation) in space. The neural network needs to classify the shape of point cloud regardless of the point cloud coordinates, which means that the point set should be invariant to certain transformations.

The architecture of PointNet is shown in Fig.2.15. There are mainly three modules in this network:

Symmetry Function for Unordered Input: In order to solve the unordered property of point clouds, there are three existing approaches:

1. The input is sorted into a canonical order. However, there does not exist a stable permutation in high-dimensional space. The instability of order requires the map to maintain spatial similarity in the case of reduced dimensions, so the ordering stability is a necessity for this project. Therefore, this scheme cannot be adopted here.

2. Use a Recurrent Neural Network (RNN) and consider the input as a sequence. By using randomly sorted data to train RNN, the RNN will be invariant to input data. However, article[118] shows that the ordering of input data in RNN cannot be completely ignored. Therefore, RNN can be used for small-sized input data, but it is difficult to be expanded to thousands of points. Point clouds usually consist of thousands of points and therefore cannot use this scheme.

3. Use symmetric function. This method adopts the max pooling to solve the unordered problem of the point cloud. After the network performs a certain degree of feature extraction for each point, max pooling can extract the global features of the point cloud.

Local and Global Information Aggregation: The output from the previous module forms a global feature of the point cloud. To achieve point cloud segmentation (Segmentation Network in Fig.2.15), local features and global features information need to be combined. After calculating the global features in the previous step, the new feature per point will be extracted based on the combination of global features and point features.

Joint Alignment Network: This module is used to achieve the geometric transformation invariance of point cloud semantic label. One solution is to align all inputs into canonical space before extracting the feature. The PointNet uses a mini network (T-net in Fig.2.15) to predict the affine transformation matrix. The T-net consists of feature extraction, max pooling and a fully connected layer that is similar to the big network.

PointNet can be applied to 3D object classification, 3D object part segmentation, and semantic segmentation. Experimental results show that PointNet has better performance and accuracy in 3D databases such as ModelNet40 than state of the art. However, in PointNet, a significant problem is the local feature problems. In the original PointNet, for part segmentation and scene semantic sparsing, when we need to get the score of each point, the solution is to directly combine the point feature with the global feature and ignore the influence of local features. For example, in a scene point cloud classification task, if we first identify the local features such as the chair is the chair, the table is the table, it will be more accurate when finally classifying each point. Therefore, the proposed optimization PointNet++[86] adds the local feature by using the layered structure and leads to a better classification result.

The main contributions of PointNet++ are to add local feature extraction on the basis of PointNet and to add density adaptation in the network structure to make the segmentation more accurate.

The Pointnet++ designed a hierarchical feature learning framework to solve the problem of local features extraction. The framework consists of three layers: the sampling layer, grouping layer and PointNet layer introduced before. This framework is capable of extracting local features at different scales and obtaining deep features through a multi-layer network structure. The sampling layer is responsible for sampling the data. The function of the grouping layer is to find all the local points of each point to facilitate subsequent extraction of features for each part. The feature extraction layer uses PointNet to perform feature extraction on each part given by the combination layer to obtain local features.

In order to solve the problem that the distribution of point cloud data in space is irregular and uneven, the PointNet++ uses the Multi-scale grouping (MSG)-based[6] (Multi-resolution grouping) MRG method to reduce the amount of computation. The local feature extracted by MRG in a certain layer is composed of two vectors connected in series. By proposing the hierarchical feature learning framework and feature extraction of point cloud data, PointNet++ solves the problem of lack of local features and obtains more accurate point cloud segmentation results.

Chapter 3

Contributions and Methodology

This chapter describes the challenges faced in the project and the contributions and methodology of this thesis. The methods mentioned in this thesis had been implemented on the Ubuntu system by using C++ and Python.

3.1 Challenges

The goal of the project I have been collaborating with (VRTogether) is to implement the SERVER-side real-time 3D reconstructed based on depth cameras and encoding of this point cloud by a coding system before data transmission. The user can see the reconstructed scenes and characters on the CLIENT side in real-time. As one part of the 3D reconstruction of the social VR project, our system is required to calibrate multiple cameras and output the calibrated results (transformation matrix) into the next 3D reconstruction stage.

Most of the current camera calibration systems use complex calibration objects. For example, the traditional camera calibration methods[53][36] rely on a calibration checkerboard and complicated corner detection operation, which are time-consuming and require manual interventions. In real-time 3D reconstruction, the cameras need to be calibrated in real-time. Therefore, this type of calibration method does not meet the requirements of instantaneity.

In 2015, Marek, Jacek et al.[57] proposed a calibration system which contains two steps: rough estimation and refinement. The idea of this system is to first perform initial calibration by using the marker shown in Fig.3.1 and then use the camera pose refinement to refine the estimate of the transformation matrix. This system allows users to calibrate Kinect v2 sensors using any calibration object at any physical place but still relies on the use of additional markers, and the registration of point clouds overlap.

OpenPTrack system[72] is a multi-camera calibration and human tracking system. This system aims to create a calibration system for depth camera networks. In this system, they used the checkerboard to calibrate cameras in two steps: intrinsic calibration and extrinsic calibration. The intrinsic calibration step calibrates each camera separately; the extrinsic calibration step relies on ROS communication, which runs in real-time and involves all sensors. This system allows networking camera calibration system to calibrate up to 10

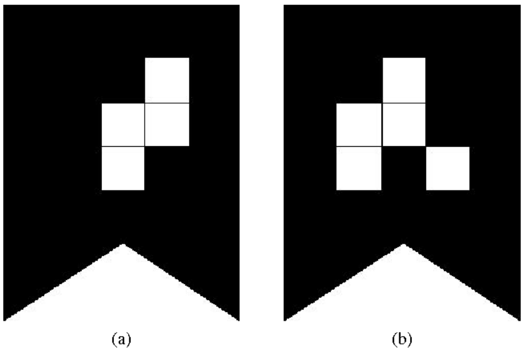


Figure 3.1: The Marker used in calibration system proposed from [57].

cameras, but it still relies on specific calibration objects and is limited by the networking capacities.

The human body tracking system published in 2017[80] proposed a real-time skeleton fusion and tracking system for Kinect sensors. This system first performs camera calibration by using a checkerboard pattern and corner detection. Then it tracks human movements based on the human skeleton as provided by the Kinect's software development tools. In this system, they fixed the calibration board on the ground to avoid it interfering with the rest of steps. However, this system did not achieve marker-free calibration and cannot be generalized to other depth sensors.

The calibration system from CERTH[19][3] uses labelled IKEA boxes with unique QR markers shown in Fig.3.2 as the calibration objects. This calibration process requires a specific large object, which cannot be easily replaced with other calibration objects and has strict requirements for the calibration. In addition, the system needs to be used in specific lighting conditions such as in a studio.

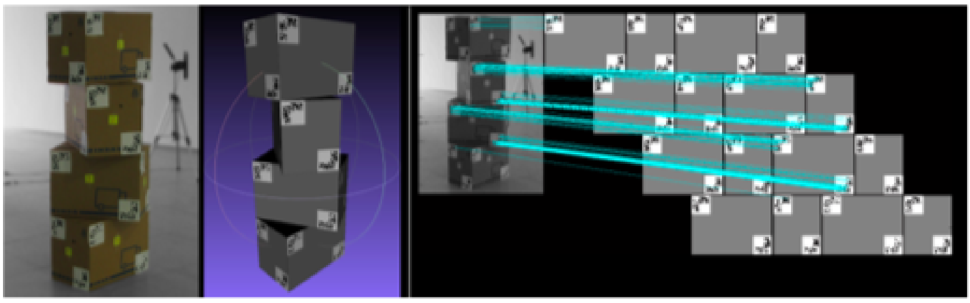


Figure 3.2: Calibration objects used for live 3D human reconstruction and motion capturing: 4 standardized IKEA boxes along with 32 QR markers
source: An integrated platform for live 3D human reconstruction and motion capturing[3]

The paper[56] published in 2018 presented a calibration system for multiple Kinect cameras. This skeleton-based approach does not require any complex calibration object and uses only a person standing in the scenes for calibration. The method uses the 3D joints extracted from the Kinect skeleton[105] for camera calibration. Also, this automatic calibration system re-calibrate cameras when calibration errors become higher than a threshold. The contribution of this system is that it does not rely on any additional calibration objects such as a checkerboard or dot pattern plane. The limitation of this system is that it cannot be used for depth sensors that do not provide the features of the human skeleton.

In this thesis, the goal is to reduce requirements on the calibration objects; we aim to design a more flexible calibration system, in which the camera can be calibrated without specific calibration objects. In this project, we have the following challenges:

- The skeleton-based calibration system mentioned in [56] gets rid of the limitations of complex calibration objects. The Kinect cameras used in this system provided the human skeleton data. However, the depth camera RealSense D415 used in our project does not provide similar specific calibration data, so we need to design a camera calibration system based on the point cloud data that is flexible on requirements on the calibration object.
- The ICP algorithm easily falls into a local optimum. As the point cloud data captured by different cameras are incomplete and do not have a part in common, the algorithm will have errors in the registration process. In this case, the ICP algorithm ignores the point cloud space features and the 3D structure and matches the non-corresponding parts of the two sets of point clouds to achieve local optimum. How to optimize the system to make the calculated transformation matrix more accurate is a challenge.

3.2 Contributions

Our system has the following contributions:

1. Design a *markerless, flexible calibration system.* This system calibrates multiple cameras based on a diversity of objects (currently the objects are chair and lamp, the type of object can be extended according to the items in the training dataset), which makes the calibration process more flexible and much easier.
2. Propose a *feature-based point cloud registration method.* We adopt point cloud segmentation and selection in the data pre-processing process to divide the object into different parts. We use a selected partial point cloud of an object segment that shows up in both cameras instead of the original point cloud for registration. This helps to overcome the shortcoming of the ICP algorithm that it can easily fall into a local optimum.

3.3 Methodology and Architecture

In this project, we need to create a more flexible calibration system that can be used with almost any type of objects. We first design the basic flow of multi-camera calibration and test the pipeline with different objects, and then propose and validate the feature-based optimization method based on the shortcomings and errors in the previous test results. Moreover, we provide a suggestion for real-time camera automatic re-calibration.

Our pipeline has four key modules: coarse registration, point cloud pre-processing, fine registration and point cloud update. The architecture of our system is shown in Fig.3.3. In this section, we will introduce these key modules in more details. The overview of pipeline in details is shown in Fig.3.4.

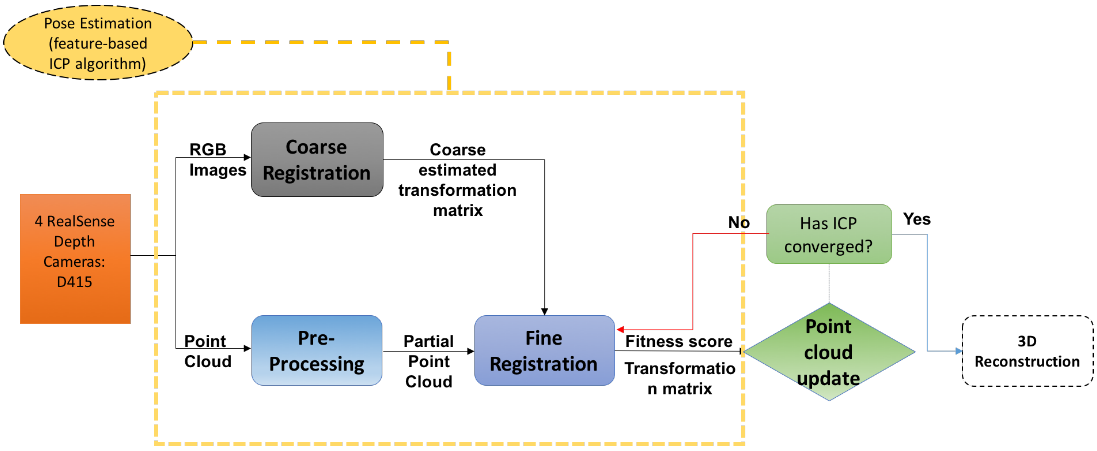


Figure 3.3: The system architecture. The grey module (coarse registration) is a one-time process; the blue modules are continuously processing procedures. In this thesis we validate the modules within the yellow dotted box and propose a suggestion (green modules) for achieving an automatic re-calibration.

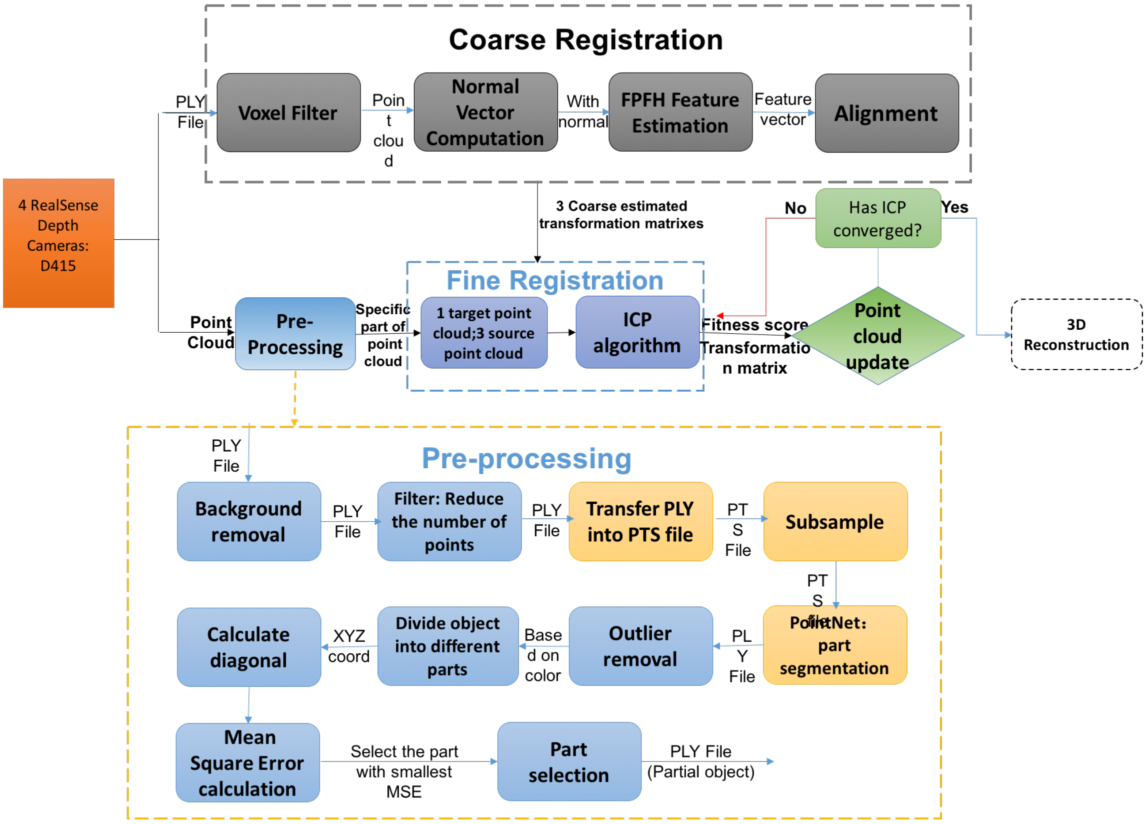


Figure 3.4: The system detailed modules. The module in grey (coarse registration) is a one time procedure; the modules in blue are continuously process. In detail, the modules in yellow (in Pre-processing step) are written in python, the other modules in pre-processing step are written in C++.

3.3.1 Coarse Registration:

In order to obtain an appropriate calibration result, we need to first calculate a coarse estimated transformation matrix. In our project, we placed four cameras in different positions to cover all angles of the object, which means that the translation and rotation varies a lot among cameras. When the positional distance between cameras is large, the ICP algorithm requires the initial matrix to provide an initial transformation so that two point clouds are transferred to the same coordinate system as much as possible to avoid the algorithm falling into a local optimum. In the coarse registration step, we first adopt the MATLAB calibration toolkit[13] to roughly estimate the camera position. Then, we use the FPFH feature based point cloud alignment to calculate the initial transformation matrix. This feature-based method is introduced in Section.3.3.4. Fig.3.5 shows the flow of MATLAB toolkit coarse registration method.

The coarse registration by using MATLAB toolkit consists of three steps, the first step is sin-

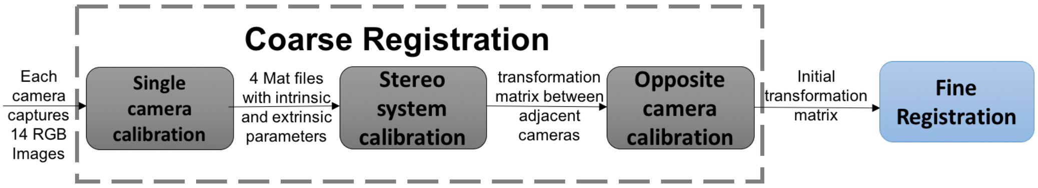

Coarse Registration
Each camera captures 14 RGB Images
Single camera calibration
4 Mat files with intrinsic and extrinsic parameters
Stereo system calibration
transformation matrix between adjacent cameras
Opposite camera calibration
Initial transformation matrix
Fine Registration

Figure 3.5: Flow of coarse registration based on MATLAB toolkit.

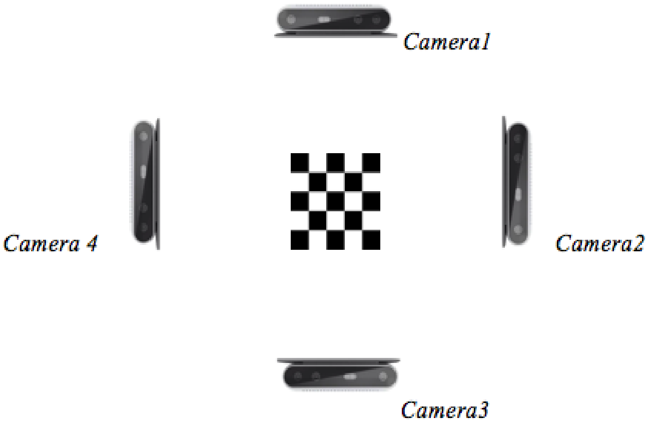


Figure 3.6: Position of four cameras

gle camera calibration, in this step we use the calibration toolkit function *calib* to calculate the camera parameters separately. For each camera, we capture 14 color images of a planar checkerboard[13]. For effectiveness of the stereo camera calibration and the consistency of the scene, each pair of cameras (adjacent cameras) must simultaneously take pictures of the calibration checkerboard. An essential step in the single camera calibration is the corner extraction. MATLAB provides two different corner extraction methods: automatic extraction and manual extraction. Compared to automatic extraction, manual extraction is complicated and time-consuming, but with a high accuracy[114]. The coarse registration step in this project is a one-time operation when the camera position is fixed, and the result of the ICP algorithm relies on the accuracy of the initial transformation matrix, so in this step, we manually extract the point corners. The output of the single camera calibration is the intrinsic and extrinsic parameters of each camera.

The second step is stereo camera calibration, this step is used to estimate the spatial relationship between adjacent cameras, and the input of this step is the parameters of the cameras. The transformation matrix that coarsely matches adjacent cameras is calculated by the stereo camera calibration function *stereo* provided by the MATLAB toolbox.

In this project, we need to calculate the spatial relationship between all four cameras, so starting with camera1 we can calibrate camera2 and camera4, but then we still have to calculate the transformation matrix between camera1 and its opposite camera3. The position of cameras is shown in Fig.3.6.

The MATLAB toolkit only works for cameras that can capture the same side of the checkerboard. Camera1 and camera3 stand in opposite directions, which means that they cannot capture the same side of the planar checkerboard. After the stereo calibration, we obtain the transformation matrix between camera1 and camera2 (or camera 4) and the transformation matrix between camera3 and camera2 (or camera4). The positional relationship between two cameras is:

$$\begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} = \begin{bmatrix} R_{ij} & t_{ij} \\ 0 & 1 \end{bmatrix} \begin{pmatrix} x_i \\ y_i \\ z_i \\ 1 \end{pmatrix} \tag{3.1}$$

The left side of equation represents the target camera, the R_{ij} is the rotation matrix between the target camera and source camera, the t_{ij} is the translation matrix between the target and source camera. Therefore, the correlation between camera1 and camera3 is:

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{pmatrix} = \begin{bmatrix} R_{12} & t_{12} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_{23} & t_{23} \\ 0 & 1 \end{bmatrix} \begin{pmatrix} x_3 \\ y_3 \\ z_3 \\ 1 \end{pmatrix} \tag{3.2}$$

Given the correlation between camera1 and camera2 (or camera4) and the relationship between camera3 and camera2 (or camera4), we can calculate the transformation matrix of camera1 and camera3 based on equation3.2. In this way, all cameras' transformation matrixes can be calculated to serve as input for the ICP program.

3.3.2 Point Cloud Pre-processing

The point cloud captured from the depth camera first needs to be preprocessed, the processed point cloud is the input to the ICP program. The pre-processing stage includes the following operations:

Background removal: In order to get a clean point cloud of the object, we need to remove the background of the captured data. This step determines the closest distance to the point cloud by calculating the closest point to the depth camera and based on this and on a chosen depth range we can set a threshold. Points farther away than this threshold are considered to be background and are removed.

Filter: In this project we use the outlier removal filter function in PCL[81]. An outlier is determined by the product of the standard variance of k neighbors and a $threshold$. First, calculate the standard deviation dev of the distance between a point and the surrounding 50 points, and then calculate the distance s from the point to the neighboring points, if $d > dev * threshold$, the point is an outlier point and will be removed.

3.3.3 Fine Registration and Point Cloud Update

The fine registration stage serves to implement the point cloud fusion by using the ICP algorithm. After capturing two point clouds (source and target) from two different angles, we can transform the source point cloud based on the initial rotation and translation matrix calculated by the coarse registration as described in Section 3.3.1. Then we use the ICP algorithm to fuse the transformed source point cloud with the target point cloud. When the target point cloud and source point cloud successfully converge, we have a refined transformation that is needed to fuse the point clouds of the individual cameras.

For the final dynamic reconstruction, we provide a multi-camera auto re-calibration solution for real-time 3D reconstruction. The point cloud update can be achieved by judging the convergence score and setting the threshold. The degree of convergence reflects the accuracy of the registration. In the experiment, the $fitnessscore$ is used to calculate the matching degree of the two point clouds. The smaller the fitness score, the better the alignment of the two point clouds. If the current frame can converge to the previous frame, which means that the fitness score is less than the threshold, then we should update the current frame to the output; if not, we need to return to the fine registration step and re-calculate the transformation matrix.

3.3.4 Feature-based Optimization

Pre-processing

The ICP algorithm can easily fall into a local optimum. In the point cloud registration, the ICP algorithm does not take the 3D shape, and spatial characteristics of the point cloud into consideration; it only uses the distance to iteratively align point clouds. In the course of our experiment, we used a chair with armrests as a calibration object. If the point clouds of two different cameras capture different armrests respectively, for example, the point cloud in camera1 has only the left armrest, and the point cloud in camera2 captures the right armrest only. In this case, the ICP algorithm will fall into a local optimum and try to match these two non-corresponding armrests, which produces a completely unuseful result (as will be shown in the next chapter).

Therefore, we conclude that using the whole incomplete point cloud of the object as produced by each camera will affect the accuracy of the ICP algorithm. Thus, to make up for this behaviour of the ICP algorithm, we need to select a specific part of the object that adjacent cameras share as the registration part.

In this stage, the first step is to subdivide the object point cloud of each camera into different parts; the next step is to choose a relatively complete part that all the cameras cover. To test with a variety of calibration objects, we use multiple methods to segment the point cloud and then select the shared part through volume estimation.

PCL based point cloud segmentation: PCL library[**?**] provides various functions for point cloud segmentation. In order to segment the point cloud and input the divided part into the ICP program, we used RANSAC segmentation[34] and region growing segmentation[88] in different models like a chair, lamp and mug.

The RANSAC segmentation method separates specific geometrical objects (such as a plane, cylinder, and cone) from the point cloud. However, the RANSAC segmentation algorithm does not perform well enough. For example, for the lamp dataset, the RANSAC segmentation based on a plane extracts a transverse plane of the lamp. This segmentation method cannot divide the point cloud according to the components of the object, which means that the lamp cannot correctly be divided into a lampshade, a bracket, and a lamp holder. Moreover, the RANSAC segmentation method only retains the valid (*INLIERS*) part which contains points within the specific model, and the valid part is not the most suitable part for ICP in some cases. Therefore, the RANSAC method as will be demonstrated is of limited use in our project.

The region growing algorithm divides the object point cloud into different clusters according to curvature and normal, the objective is to divide the point which satisfies the smoothing constraint and its curvature smaller than the threshold into the same cluster. However, the point cloud captured from the camera is not smooth and contains a lot of noise, which affects the correctness of the algorithm, so the segmentation result is not good and cannot be used for subsequent ICP registration step.

Deep learning based Segmentation: We need to come up with a segmentation method that can divide objects into different parts based on various characteristics (components) of objects. Also, the segmentation result is required to preserve all parts of the object. Then we can choose the appropriate part (the relatively complete and common part of different cameras) for the ICP registration step.

Based on our literature research, we found that the PointNet++ neural network directly processes point cloud data and has a good performance in part segmentation. So we use the PointNet++ to segment the point cloud.

Part segmentation: The purpose of this step is to divide the point cloud into different parts. First, we need to select some objects from the dataset as the calibration object. The reason is that in our system, some of the objects in the dataset are not suited to be used as a calibration object, such as an airplane. Second, because the training set data is different from the point cloud we captured from the depth camera, We need to filter and downsample the point cloud, and change the data format. Finally, we need to output the predicted results and annotate the different parts of the point cloud with specific colors.

Part Selection: The test results of the PointNet++ network label the different parts of the object with different colors. We calculate the cube diagonal values of different parts to represent the size of the volume. For the same part (same color) of the four cameras, the coverage and completeness of the same part among the four cameras are measured by calculating the Mean Square Error. The part with the smallest *MSE* value indicates that the volume difference of this part is the smallest, which means that this part of point cloud acquired by the four cameras is the most ”similar” part in volume. Therefore, the part with the smallest Mean Square Error is considered to be the most suitable part for point cloud registration.

Coarse registration

In our system, the coarse registration step uses the MATLAB toolkit to calculate the initial transformation matrix. Although the results of MATLAB toolkit have good accuracy, the operation process is complicated and time-consuming, and we need a checkerboard as the calibration object for MATLAB toolkit calibration. If manual positioning of a checkerboard is not required, our system will be more flexible and convenient.

The point cloud library provides a variety of feature descriptors and feature extraction methods. The feature descriptors extracted by these feature extraction methods can then be matched by using the RANSAC algorithm, and the transformation matrix can be calculated. In our system, we use FPFH feature extraction and matching functions to replace MATLAB toolkit to calculate the initial transformation matrix.

3.3.5 Dataset

In the PointNet++ model training process, we use a subset[85] of the 3D ShapeNetCore[104] dataset. The 3D ShapeNetCore dataset contains clean 3D models and segmented sections marked by different colors. The dataset covers ”55 common object categories with approximately 51,300 unique 3D models”[104].

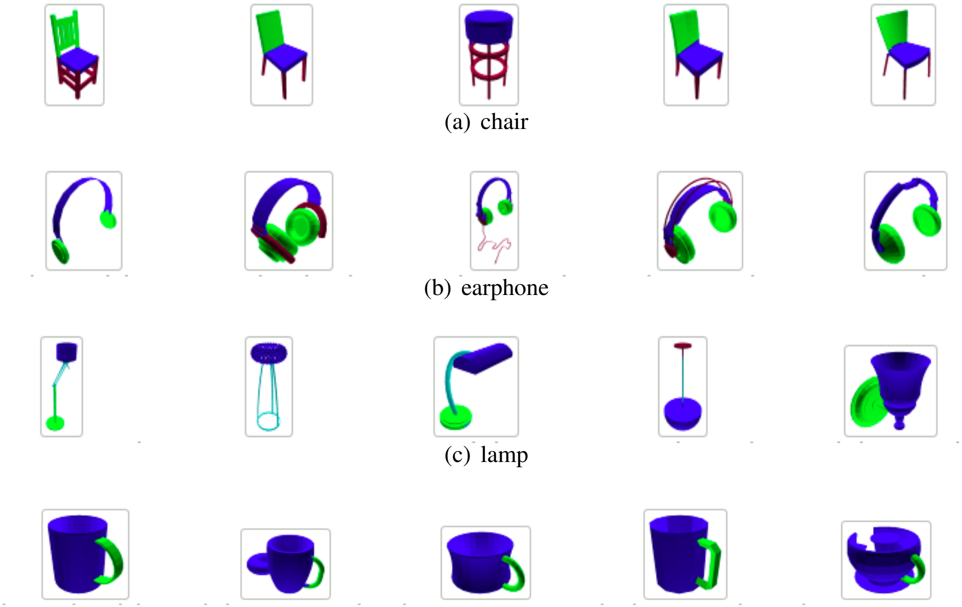
The subset used in our project is released based on the framework proposed by the paper[125]. The framework used to generate labelled 3D model is an active learning method for semantic part annotation, which is used to correctly label object parts with minimal manual operations. This framework includes two main interfaces: an annotation interface and a verification interface. The annotation interface provides painting brushes for the user to label the object into several parts, then the rest of the shapes will be labelled by propagating the manual label. The verification interface shows the annotation results and inquires users to verify the correct propagation and improves the propagation technique.

The subset used as the training set in our project includes 16 categories: airplane, bag, cap, car, chair, earphone, guitar, knife, lamp, laptop, motorbike, mug, pistol, rocket, skateboard, and table. Table 3.1 shows the description of the category in this subset. These objects are subdivided into different parts based on their components; each part is labelled with a specific color. Each data in this dataset contains two types of files: a *.seg* file and a *.pts* file. The *.seg* file is the label value for each point. The *.pts* file is transferred from 3D point cloud files and saves the 3D information of the model.

Table 3.1: Description of the 3D model category in the training set

	Number of Samples	Parts per Sample	Description of the Parts
Airplane	2690	4	Body, Engine, Tail, Wing
Bag	76	2	Body, Handle
Cap	55	2	Panels, Peak
Car	1824	4	Hood, Roof, Wheel, Body
Chair	3746	4	Arm, Back, Leg, Seat
Earphone	69	3	Earphone, Headband
Guitar	787	3	Body, Head, Neck
Knife	392	2	Blade, Handle
Lamp	1546	4	Bracket, Lamp holder, Lampshade
Laptop	445	2	Keyboard, Screen
Motorbike	202	6	Gas-tank, Handle, Light, Seat, Wheel, Body
Mug	184	2	Handle, Body
Pistol	275	3	Barrel, Handle, Trigger-and-guard
Rocket	66	3	Body, Fin, Nose
Skateboard	152	3	Deck, Wheel
Table	5266	3	Leg, Top

Among these 16 categories, we selected four types of objects that are commonly used in daily life and suitable to be used as calibration objects for deep learning network training and part segmentation. Fig.3.7 shows some of the training data used in our experiment. Each color represents one part, the chair contains four parts, earphone and lamp have three parts, and mug data consists of two parts.



(a) chair

(b) earphone

(c) lamp

(d) mug

Figure 3.7: Partial 3D model images of 3D ShapeNetCore dataset. source: [85]

Chapter 4

Experiments and Results

This project uses C++ and python as programming languages, the RGB-D cameras used here are Intel RealSense D415. The results and performance are evaluated by processing time, fitness score, and number of iterations. The *processing time* refers to the running time of the ICP algorithm. In the feature-based optimization method, this time also includes the running time of the point cloud segmentation. The *fitness score* is used to measure the matching degree of the alignment results, the smaller value represents the better matching result. The *number of iterations* is used to compare the performance of different methods under the same number of iterations condition. The detailed explanation of measurement metrics is introduced in Section 5.2. During the experiments, we validate the pipeline with different angles, different numbers of cameras and different objects.

4.1 Pre-experiments

Before we implement the whole pipeline, we first test the ICP algorithm by using the PCL[81] library. These pre-experiments include two experiments. In this section, we will illustrate the experiment settings and demonstrate the results.

Experiment 1.0

Goals: In this thesis, we start an experiment with a single camera to test the ICP algorithm and different types of objects, such as a bunny rabbit and toys. The goal is to implement the ICP algorithm and test its performance.

Settings: In our first experiment we test with a bunny rabbit model. The bunny rabbit is a 3D model data created by Stanford University, the PLY file is downloaded from the Internet. We use two identical PLY files as input, one is set as the *target*, and the other one is set as the *source*. The transformation matrix between two point clouds is rotation degree with $\pi/8$ and a 0.4 meters translation on the z-axis.

Results: In the first experiment with a bunny rabbit 3D model, the registration process achieves good performance after 18 iterations. The original bunny rabbit (white) and the transformed one (green) is shown in Fig.4.1. Fig.4.2 shows the aligned point cloud.

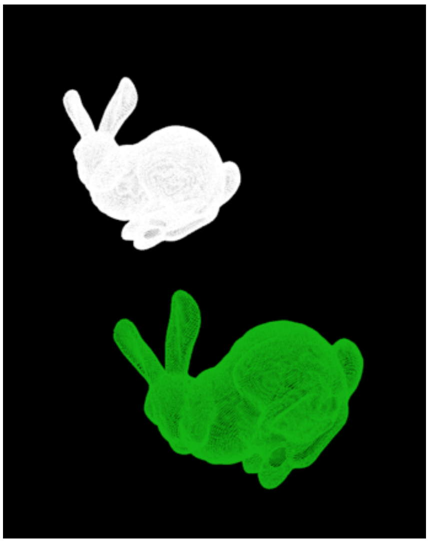


Figure 4.1: Original and transformed bunny

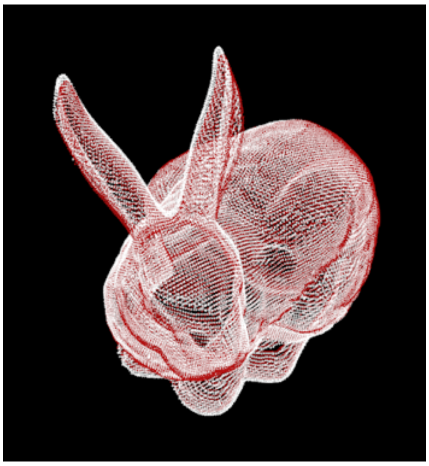


Figure 4.2: Aligned bunny

In this experiment, we try to match the red point cloud with the original point cloud. In the iteration process, these two point clouds are getting closer and closer. After 24773ms, the fitness score achieves 1.033e-6. The smaller the score, the higher the similarity between these two point clouds.

Experiment 2.0

Goals: The bunny model is a clean and complete computer-generated model and in real practice, we cannot capture such point cloud using depth cameras. In order to test the ICP algorithm with real data captured from RealSense D415 camera, in the next experiment, we used one RealSense camera to capture the point cloud of static objects instead of using computer-generated models.

Settings: In this experiment, we only translate the object 0.4 meters along the x-axis. In this case, we skip the coarse registration step, directly pre-process the point cloud and then perform fine registration step. Unlike the first experiment, we need to capture the point cloud from the RealSense camera and transfer it to the ICP process. The pre-processing in this experiment includes background removal. In order to get a clear, recognizable point cloud of the object, we need to remove the irrelevant background.

The background removal process is based on the distance from the object to the camera. We capture the point cloud from the nearest point to a certain range. In this case, we calculate the closest distance between the camera and the object, and then we add a distance of 0.2 meters to cover the entire object and remove the uncorrelated background that is beyond that distance. Then we export the depth and color information of the captured point cloud into the ICP program. The next fine registration step is the same as in the first experiment.

In this second experiment, we capture the point cloud from a single RealSense camera. The original *source* point cloud and the *target* point cloud are shown in Fig.4.3 and Fig.4.4. In

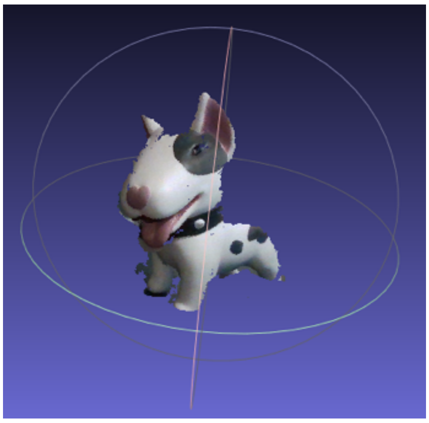


Figure 4.3: Original source dog point cloud

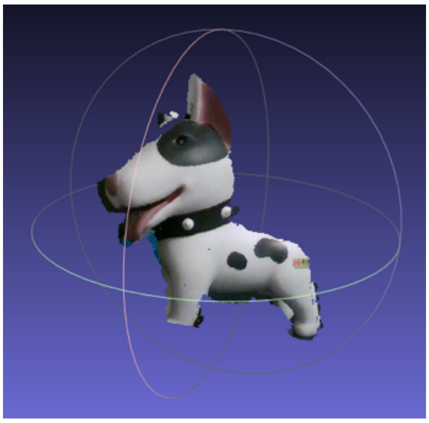


Figure 4.4: Original target dog point cloud

this experiment, we move the object from the original position to the target position by 0.4 meters.

We use the original point cloud and the target point cloud as the input, and then use the ICP algorithm to align these two point clouds. In the ICP program, the input original and target point clouds are shown in Fig.4.5. The white one is the *source* point cloud, and the green one is the *target* point cloud.

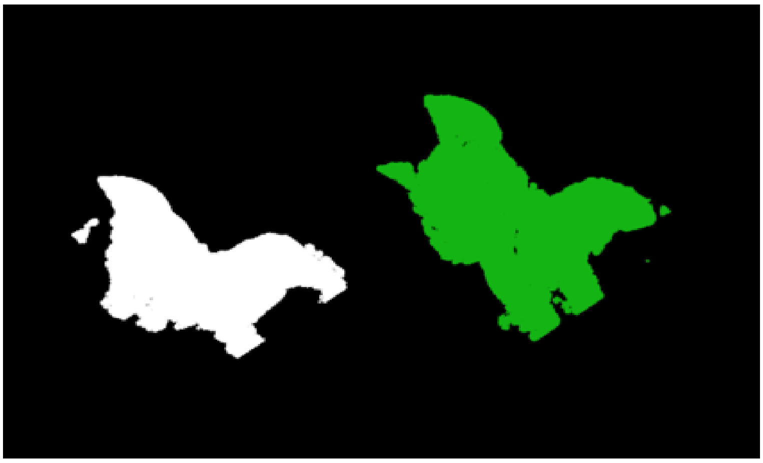


Figure 4.5: Original source and target toy dog point cloud

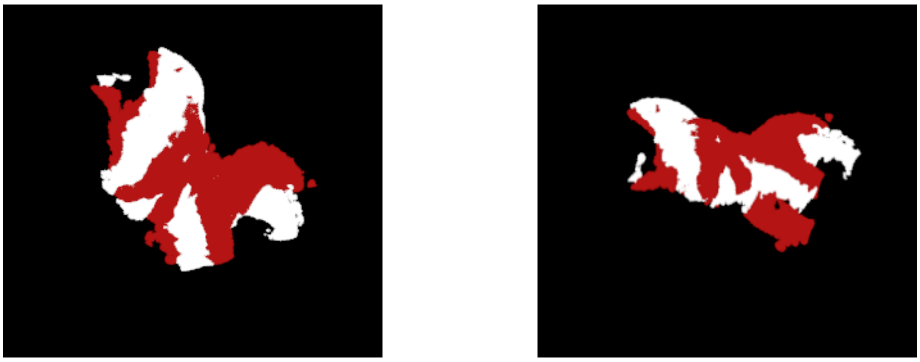


Figure 4.6: Aligned point cloud of the toy dog

Results: After 20 iterations, some parts of toy dog such as dog ear and legs have been matched. The fitness score is 1.657e-4 and the processing time is 164938ms. The aligned point cloud is shown in Fig.4.6.

4.2 Validate the Pipeline (Two Cameras)

In the following experiments, we set point cloud captured by camera1 as the *target*, and the rest of the point clouds are *source*. The objective of point cloud registration is to match the point cloud acquired by the *source* camera to the *target* point cloud.

Goals: In section 4.1 we implement and validate the ICP algorithm. In order to test the pipeline shown in Fig.3.3, we use two cameras with a 90-degree angle for pipeline testing. The target object is a static item - chair.

Settings: In this experiment, we put two cameras at different places as shown in Fig.4.7. The experimental process is as follows:

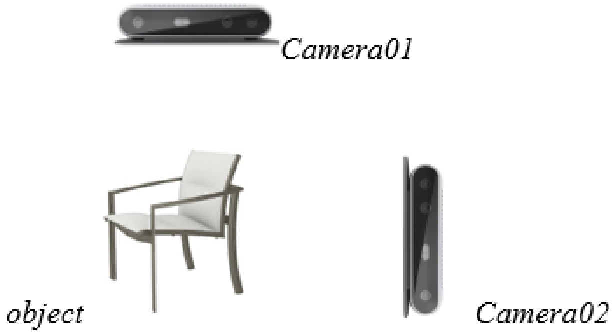


Figure 4.7: Positions of two depth cameras

1) *Coarse registration:* In this step, we use a calibration checkerboard and the MATLAB toolkit to calculate the transformation matrix. In this process, we take 14 RGB images

of the calibration checkerboard shown in Fig.4.8 as the input to the MATLAB calibration toolkit. The position of the calibrated board is different in 14 images, and the two cameras need to take pictures at the same time. Based on the RGB images captured from cameras and manual corner detection process, we first calculate the intrinsic and extrinsic parameters of each camera, then use the stereo calibration to compute the transformation matrix.

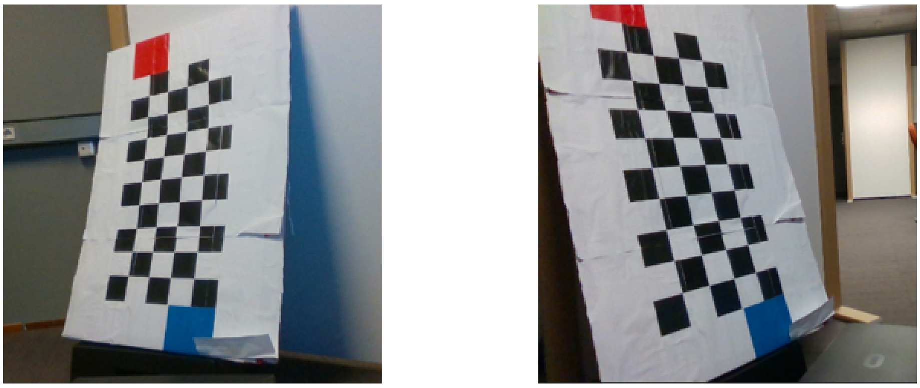


Figure 4.8: Photos of calibration from camera1 (left) and camera2 (right)

2) *Pre-processing:* Besides background removal described in section 4.1, in this experiment, we add a filter (see in Section 4.4.1) to reduce the number of points. The filter only discards certain points according to the filter parameters. This process can reduce the computational burden and save processing time. The captured point cloud is shown in Fig.4.9.

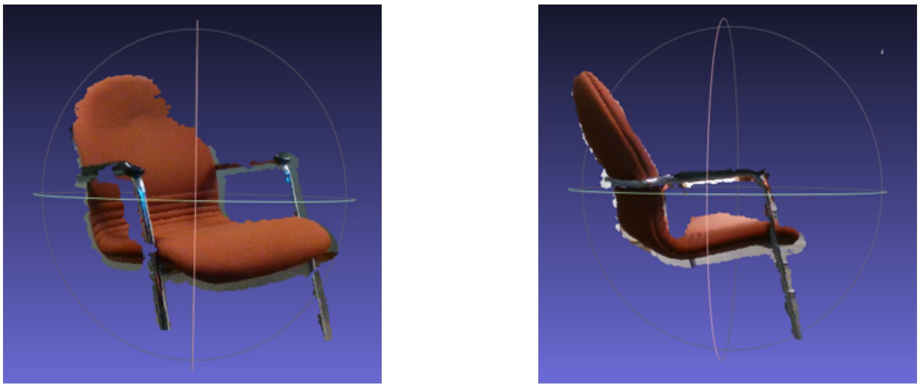


Figure 4.9: Point clouds of a chair captured from camera1 (left) and camera2 (right)

3) *Fine registration:* Different from the previous experiment, we need to use the transformation matrix calculated in the coarse registration step to roughly estimate the spatial relationship of cameras.

i) Create a temporary point cloud *point_temp* to store the transformed point cloud. In this step, we apply the transformation matrix to the *source* point cloud.

ii) Align the temporary point cloud *point_temp* with the *target* point cloud by using the ICP algorithm.

iii) Calculate the processing time, fitness score and iteration times.

iv) Display the *source* point cloud, *target* point cloud and aligned result on the screen.

Results: In this experiment, the process requires 80 iterations to achieve the convergence. In this case, we also need more numbers of iterations than before. In order to save processing time, we adopt a filter to reduce the number of points. For this reason, the entire processing time is 30657ms. The fitness score is 7.231e-4. Fig.4.10 shows the original *source* point cloud and the *target* point cloud. Specifically, the white one is the target point cloud from camera1, the blue one is source point cloud from camera2, and the green one is transformed from source point cloud based on the transformation matrix which is obtained by the coarse registration step. Fig.4.11 and Fig.4.12 show the aligned point cloud when the number of iterations are 1 and 80.

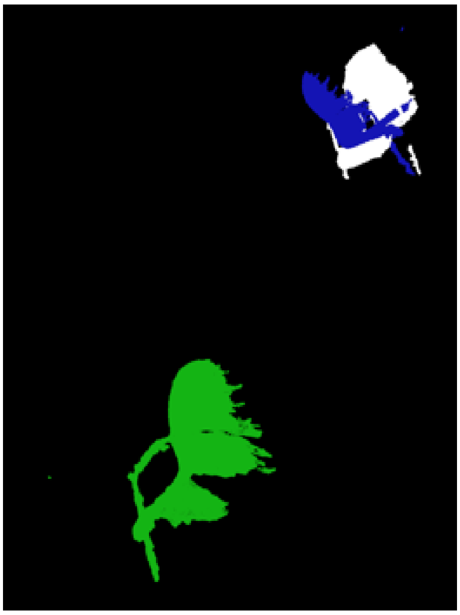


Figure 4.10: Source, target and transformed point cloud of a chair

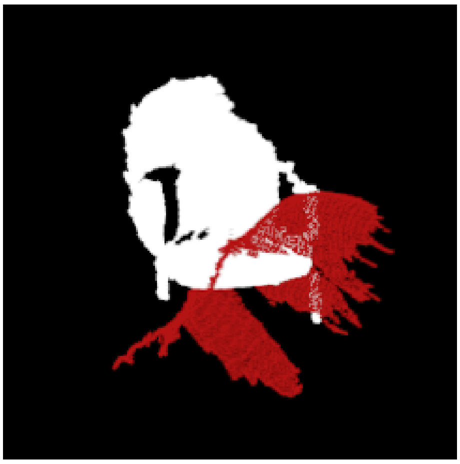


Figure 4.11: Aligned point cloud after 1 iteration

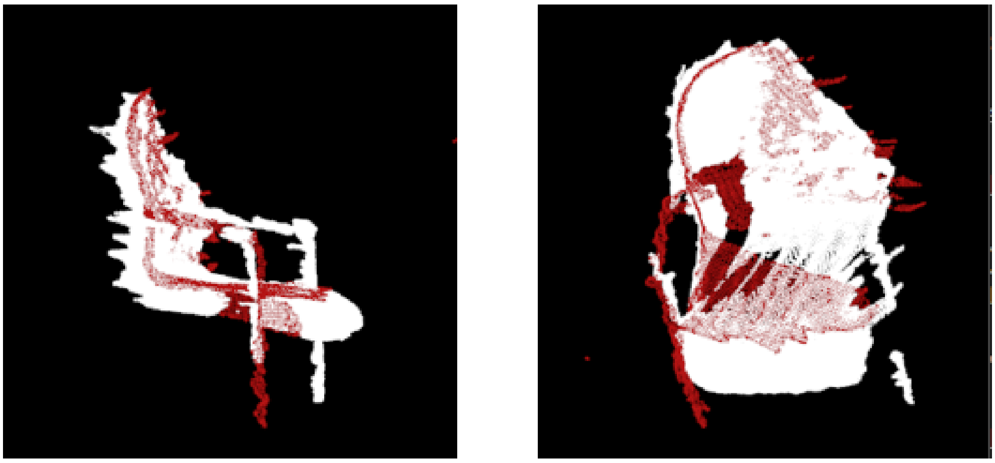


Figure 4.12: Aligned point cloud after 80 iterations

4.3 Validate the Pipeline (Four Cameras)

Goals: The point clouds in section 4.2 only cover parts of the object, the next step is a test with four RealSense cameras that cover the entire 360 degrees.

Extending on the experiment with two RealSense cameras, we use four RealSense cameras to cover the entire 360 degrees of the objects. The object is a chair.

Settings: In this experiment, we put four cameras in different positions as shown in Fig.4.13.

The alignment processes of camera4 and camera2 are the same as Section 4.2 because they both use camera1 as the reference. The main difference between this experiment and the experiments described in Section 4.2 is that we need to calculate the transformation matrix of camera3.

In the coarse registration step, we can only get the relationship of cameras located on the

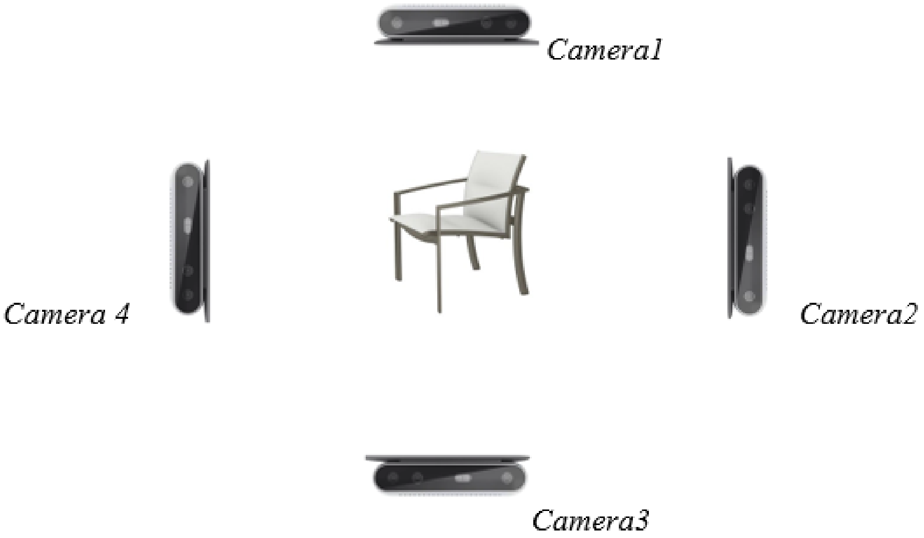


Figure 4.13: Positions of four cameras

same side of the calibration checkerboard. As can be seen from Fig.4.13, the camera1 and camera3 are located on the opposite side of the object, which means we cannot obtain the transformation matrix between camera1 and camera3 in the coarse registration step.

In this experiment, we obtain the positional relationship of camera1 and camera2(or camera4), also the positional relationship of camera2(or camera4) and camera3. Based on the method mentioned in section 3.3.1 we calculate the transformation matrix of camera1 and camera3.

Results: In this experiment, the calibration object is a chair. In order to get a clear view of the registration result, we will separately demonstrate the registration result of three *source* cameras. Fig.4.14 shows the captured original point cloud from four cameras.

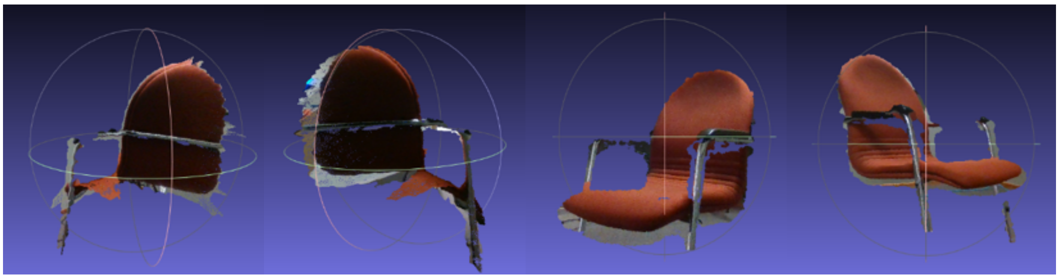


Figure 4.14: Captured original point cloud from (left to right) camera1, camera2, camera3 and camera4

The spatial relationship of camera1 and camera2 is computed in the coarse registration step.

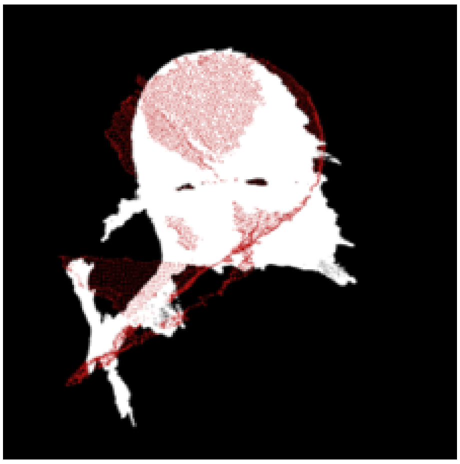


Figure 4.15: Aligned point cloud after 40 iterations. The white point cloud is the *target* captured by camera1, the red one is the *source* captured by camera2.

From Fig.4.14 we observe that the captured point clouds of the individual cameras do not cover the entire object. The point cloud from camera1 covers the right armrest of the chair while the point cloud from camera2 covers the left armrest of the chair, which leads to an error in the fine registration step. The result of registration of camera2 is shown in Fig.4.15.

The result shows that after the registration step, the left armrest in the *source* point cloud (camera2) is getting closer to the right armrest in the *target* point cloud (camera1). This means that the classic ICP algorithm is confused. It demonstrates that in some situations, when the *source* point cloud and the *target* point cloud only cover different parts of the object feature, the algorithm does not produce useful results.

The same happens with camera2; we also get the spatial relationship of camera4 and camera1 in the coarse registration step. In this case, even if the target point cloud from camera1 and the source point cloud from camera4 both cover the right armrest, the output still has an error. The reason is that the two point clouds do not cover the complete armrest. After the ICP process, the *source* point cloud aligns the edge of the chair seat with armrest in the *target* point cloud. The alignment result of the camera4 is shown in Fig.4.16.

The processing time is 34972ms. After 40 iterations, the fitness score of these two point clouds is 5.687e-3.

Unlike camera2 and camera4, we calculate the transformation matrix of camera3 according to formula 3.2. The result is shown in Fig.4.17. The cause of camera3 error is the same as camera4. The processing time is 37980ms. After 40 iteration times, the fitness score is 3.397e-3.

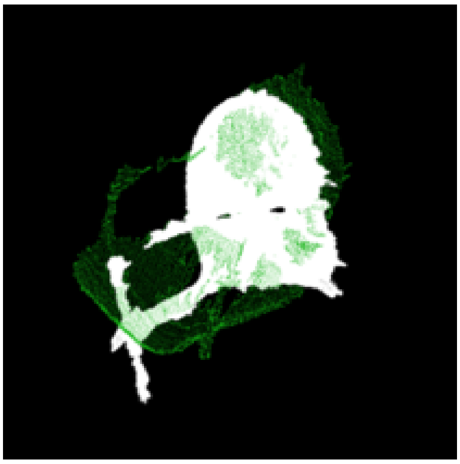


Figure 4.16: Aligned point cloud after 40 iterations. The white point cloud is the *target* captured by camera1, the green one is the *source* captured by camera3.

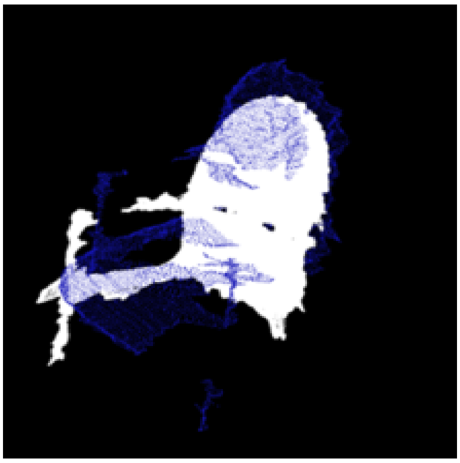


Figure 4.17: Aligned point cloud after 40 iterations. The white point cloud is the *target* captured by camera1, the blue one is the *source* captured by camera4.

Optimization

The reason for the error in this experiment is that the chair point cloud is incomplete and the point cloud angles captured by different cameras are different. In order to improve the registration accuracy, we remove parts of the point clouds and test with only one specific part of the point cloud. By comparing the four point clouds in Fig.4.14, we found that the point cloud in the back of the chair is the most complete part. We manually remove the rest part of the chair and export the chair back into PLY file for testing.

Fig.4.18 shows the alignment result of original point clouds (the left column) with aligned point clouds (the right column) covering only the back part of the chair. Each row represents the point clouds captured from camera2 (the red ones), camera3 (the green ones) and

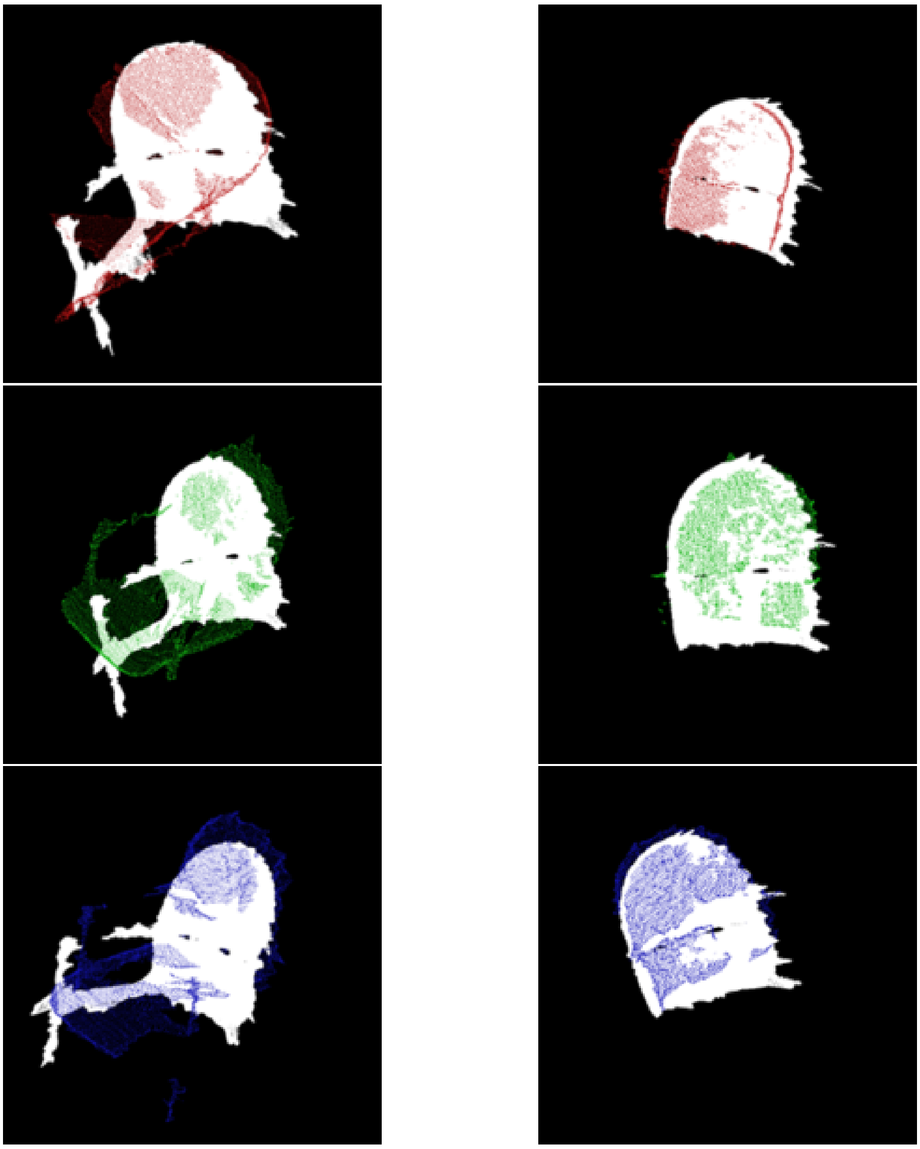


Figure 4.18: The comparison between original alignment point cloud and edited alignment point cloud

camera4 (the blue ones). As can be seen from the above pictures, when the point cloud only contains one common part of the object, the registration result is better. Table 4.1 shows an evaluation comparison of these two cases.

Analysis: Based on Table 4.1 we conclude that: if we only keep the back part of the chair, the registration results are better than the results of original point clouds. Meanwhile, the processing time is shorter, and the fitness score is smaller, which means that the manually edited point cloud aligns better than the original ones.

Table 4.1: Registration evaluation and comparison of original point cloud and edited point cloud - Chair Point Cloud

	Point Cloud	Processing Time	Number of iteration	Fitness Score
Source Camera2	Original	24782ms	40	1.112e-3
	Chair back only	10524ms	40	1.075e-4
Source Camera3	Original	46031ms	40	3.397e-3
	Chair back only	18192ms	40	1.791e-4
Source Camera4	Original	39096ms	40	5.687e-3
	Chair back only	15226ms	40	4.581e-4

4.4 PCL-Based Segmentation with Pipeline

In this section, we will illustrate the point cloud segmentation based on the point cloud library and show the result of point cloud registration based on the segmented point cloud.

The purpose of point cloud segmentation is to divide the target point cloud according to its different components (parts) and be able to select corresponding parts, leading to better results as demonstrated by the experiment in Section 4.3. In the segmentation algorithm provided by PCL, SAC segmentation can extract different geometrical forms (such as the plane, cylinder) of the point cloud. Therefore, in different point clouds, the target part can be segmented based on features of the object. For example, the back of the chair is the target segmentation part. We can use the SAC segmentation and plane model to segment the target part. Besides, we also use region growing method to segment the point cloud.

4.4.1 SAC Segmentation

Goals: Based on the experimental results from the section4.3, we concluded that using a partial target point cloud instead of the entire point cloud for registration can achieve better performance with smaller fitness score and shorter processing time. However, in the previous experiment, we manually selected the appropriate part (chair back) as the input of the ICP algorithm for point cloud registration. To make it practical, it is necessary to have an autonomous process for segmentation of the point cloud.

Plane Model-Based SAC segmentation

Settings: In this experiment, we use functions provided by the PCL to automatically achieve point cloud segmentation automatically. The first segmentation method we used is the SAC segmentation algorithm. The flow of SAC plane segmentation is shown in Fig.4.19.

Filter: In this project, we use the voxel filter to downsample the point cloud. The function $pcl::VoxelGrid < pcl::PointXYZ >$ first constructs a 3D voxel grid, then it uses the gravity

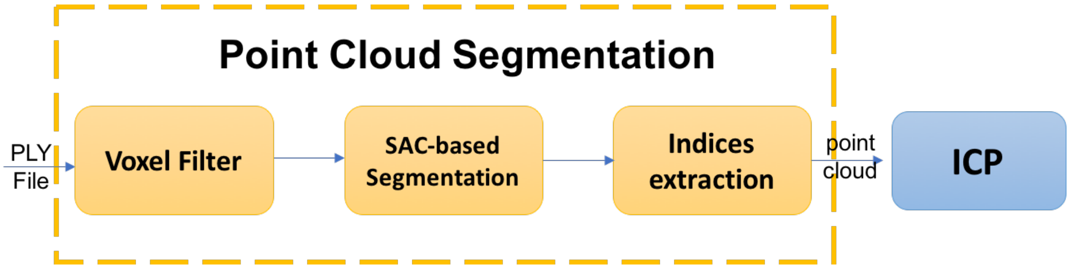


Figure 4.19: The flow of SAC segmentation

of all points in the voxel to represent these points. All points within the same voxel grid will be shown as one gravity point. In this way, the number of points in point cloud is reduced.

SAC Segmentation: The input of SAC plane segmentation is a set of points and model (plane) parameters. The segmentation including the following steps:

- The SAC segmentation algorithm divides the data into $INLIERS$ (valid) and $OUTLIERS$ (invalid) and iteratively chooses a random subset as the $INLIERS$ to calculate the parameters of the model.
- Use the calculated parameters to test other data and select $INLIERS$ points.
- Repeat the above steps and calculate different models and parameters. Compare the number of $INLIERS$ points between different models and set the model with the most $INLIERS$ points as the $best$ model.
- Finally, return the index of $INLIERS$ in the $best$ model to the next module.

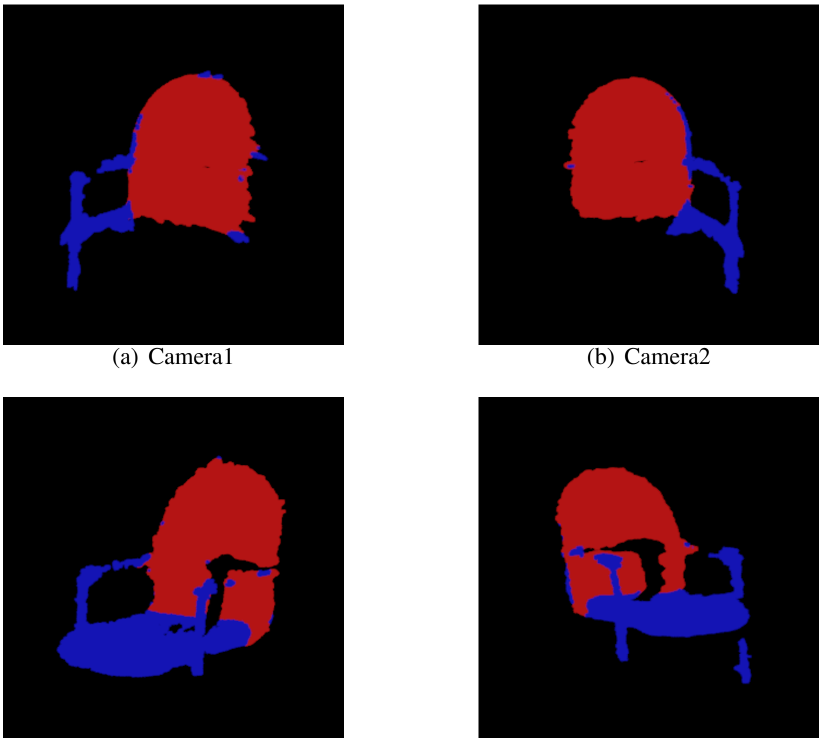
The function $pcl :: SACSegmentation < pcl :: PointXYZ >$ is used to process the above steps. The function $pcl :: SACSegmentation.segModelType$ is used to set the segmented model. In this experiment we select the $SACMODEL\_PLANE$ to extract a plane in the point cloud. The four coefficients of the plane model are in the Hessian Normal form[123][37]: $(normal\_x, normal\_y, normal\_z, d)$.

Indices extraction: Then, we use the function $pcl :: ExtractIndices < pcl :: PointXYZ >$ to extract the $INLIERS$ points in the point cloud obtained from SAC segmentation and output the point cloud with valid points in the plane.

Results: Fig.4.20 shows the chair segmentation result by using the SAC segmentation function.

Then, we extract the red part and input the segmented part into the ICP algorithm to calculate the transformation matrix. Fig.4.21 shows the registration results of the segmented point clouds.

Analysis: Table 4.2 shows the evaluation and comparison of ICP registration with different input point clouds. From the table, we can conclude that for the chair point cloud, the



(a) Camera1

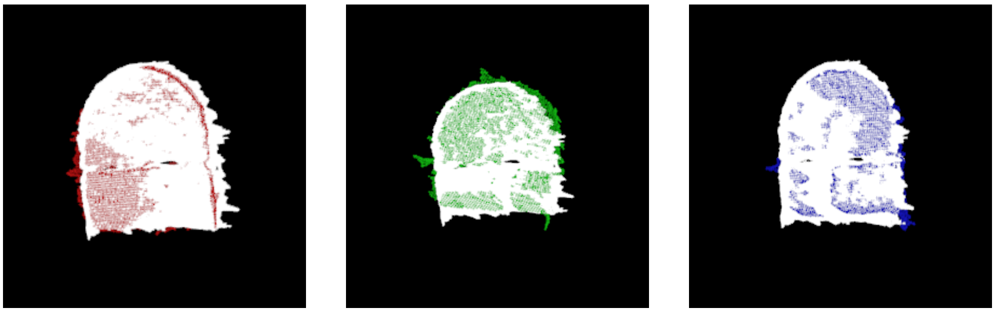
(b) Camera2

(c) Camera3

(d) Camera4

Figure 4.20: The plane model-based SAC segmentation results of chair. The red part is the segmented plane, the blue part is the original point cloud.

plane model-based SAC segmentation method effectively segments the point cloud and reduces the fitness score compared to the original point cloud. It improves the registration accuracy and shortens the running time. Compared with the manual segmentation method, the automatic segmentation has a shorter processing time and achieves comparable accurate registration results.



(a) Camera2 (b) Camera3 (c) Camera4

Figure 4.21: ICP alignment results. The white point cloud is the target point cloud from camera1, the red, green and blue ones represent the aligned point cloud from camera2, camera3 and camera4.

Table 4.2: Registration evaluation and comparison of original point cloud, manual and automatic edited point cloud - Chair Point Cloud

	Point Cloud	Processing Time	Number of iteration	Fitness Score
Source Camera2	Original	24782ms	40	1.112e-3
	Automatic	11764ms	40	7.325e-5
	Manual	10524ms	40	1.075e-4
Source Camera3	Original	46031ms	40	3.397e-3
	Automatic	17501ms	40	2.386e-4
	Manual	18192ms	40	1.791e-4
Source Camera4	Original	39096ms	40	5.687e-3
	Automatic	11195ms	40	1.802e-4
	Manual	15226ms	40	4.518e-4

Validate with Other Objects

In the following experiments, we validate the pipeline with various calibration objects. Fig.4.22 shows the original lamp point clouds and Fig.4.41 shows the segmentation results of the lamp by using the plane model-based SAC segmentation method.

From the segmentation results we can conclude that the plane model-based SAC segmentation method divides a transverse surface according to the lamp bracket. Due to the uncertainty of the segmented plane angle, the plane segmented from different point clouds cannot represent the same part. Therefore, the use of plane model-based SAC segmentation is not

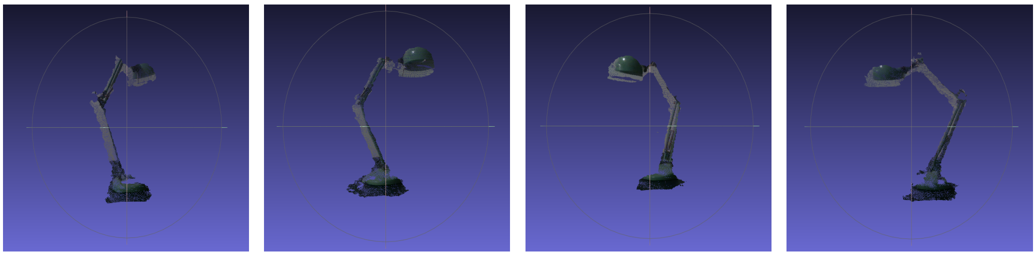
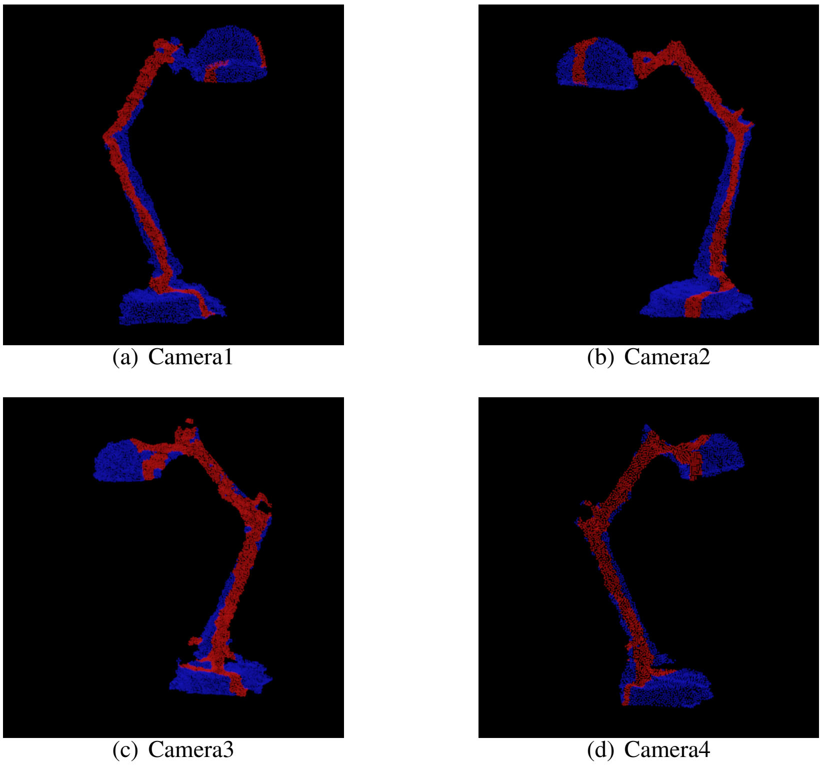


Figure 4.22: Captured lamp point cloud from (left to right) camera1, camera2, camera3 and camera4



(a) Camera1

(b) Camera2

(c) Camera3

(d) Camera4

Figure 4.23: The plane model-based SAC segmentation results of the lamp. The red part is the segmented plane, and the blue part is the original point cloud.

valid for the lamp object.

Fig.4.24 shows the original mug point clouds from four cameras. Fig.4.25 shows the segmentation results of mug point cloud by using the plane model-based SAC segmentation method. The mug point cloud contains two components: mug body and mug handle. Due to the fact that the mug body is a rotation invariant uniform cylinder, the mug handle is more useful in the registration process. Therefore, it is helpful if we can segment the handle part and perform point cloud registration based on the handle. However, in this experiment, the segmentation results of camera1 and camera2 only contain a portion of the mug body. The segmentation results of camera3 and camera4 divide a portion of mug body and mug handle as the segmented plane. For point clouds from camera1 and camera2, the algorithm treats a portion of the mug as a plane and splits it out. For point clouds from camera3 and camera4, the mug handle is connected with the mug body. Therefore, the segmented plane including mug handle and mug body contains the maximum number of *INLIERS* points. Parts of the segmented mug are represented in different point clouds. Therefore, the plane model-based SAC segmentation is ineffective for the mug point cloud.

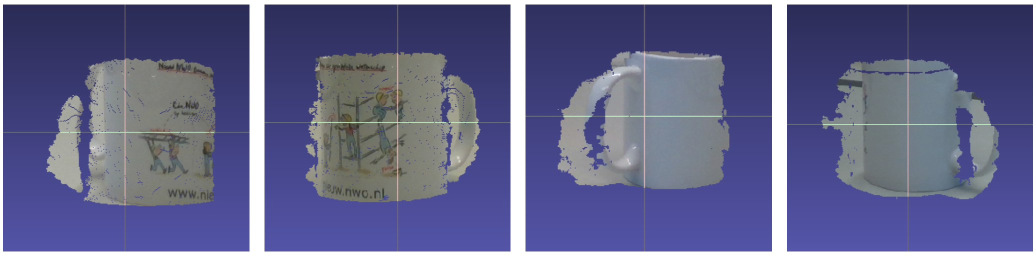
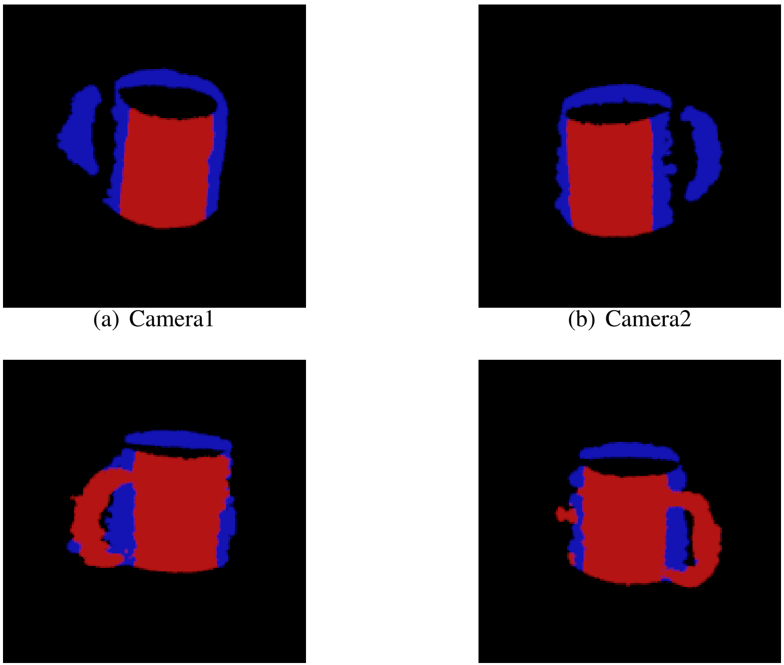


Figure 4.24: Captured mug point cloud from (left to right) camera1, camera2, camera3 and camera4



(a) Camera1

(b) Camera2

(c) Camera3

(d) Camera4

Figure 4.25: The plane model-based SAC segmentation result of the mug point cloud. The red part is the segmented plane, and the blue part is the original point cloud.

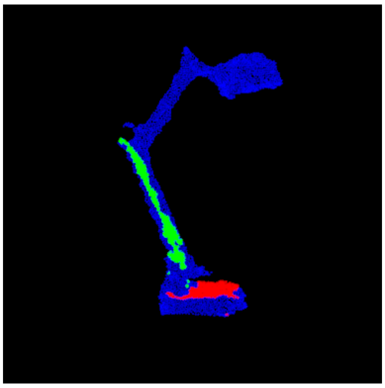


Figure 4.26: Lamp segmentation result from cylinder model-based SAC segmentation.

Cylinder Model-Based SAC segmentation

The plan model-based SAC segmentation did not provide valid segmentation results for lamp and mug point clouds. A cylinder form can be recognized in both the lamp point cloud and the mug point cloud. Therefore, we use the cylinder model-based SAC segmentation method to divide the lamp and the mug point clouds.

For cylinder segmentation, we use the $SACMODEL\_CYLINDER$ as the SAC segmentation model. The cylinder model has seven coefficients: $(point\_on\_axis.x,$ $point\_on\_axis.y, point\_on\_axis.z, axis\_direction.x, axis\_direction.y, axis\_direction.z,$ $radius)$.

Lamp: Fig.4.26 shows the segmentation result of the lamp by using the cylinder model-based SAC segmentation method. The blue part is the original point cloud, the red part is the segmented plane point cloud, and the green part is the segmented cylinder point cloud. The method incorrectly treats the lamp racket as a cylinder. This segmentation leads to non-coincident segments in different point clouds. Therefore, the segmented point cloud cannot be used as the input to the ICP algorithm.

Mug: Fig.4.27 shows the segmentation results of the mug by using the cylinder model-based SAC segmentation method. In figures below, the blue part is the original point cloud, and the green part is the rest points. We saved the green part by using the member function $pcl :: ExtractIndices.setNegative(True)$. The cylinder segmentation will segment the points within the cylinder and output $INLIERS$ points, the true value in the above function will extract indices in the point cloud except for the points within the cylinder.

4.4.2 Region Growing:

Settings: Fig.4.28 shows the flow of region growing segmentation by using the Point Cloud Library. The filter used here is the VoxelFilter introduced in the Section 4.4.1. For normal estimation, PCL provides class $pcl :: NormalEstimation$ and uses the least square estimation[109] to calculate the surface normal.

The region growing segmentation contains the following steps:

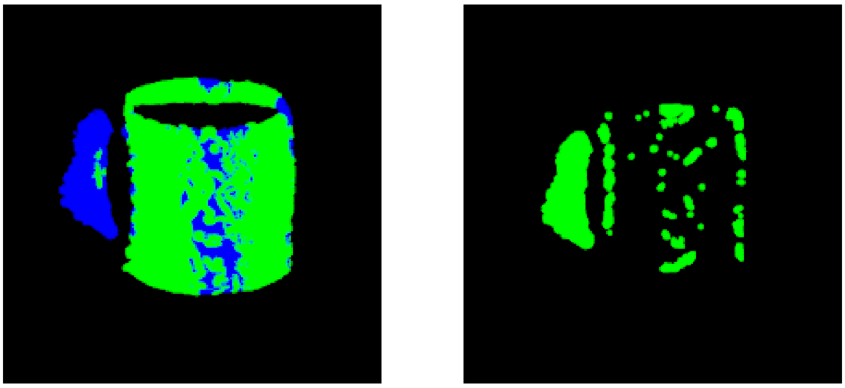


Figure 4.27: The cylinder model-based SAC segmentation results of mug showing a fragmented result.

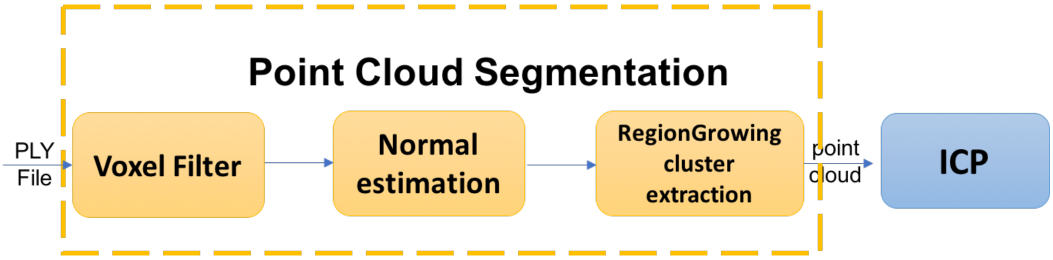


Figure 4.28: Flow of region growing segmentation method in PCL

- Sort points based on the normal and curvatures;
- Choose the points with lowest curvatures as *seed*;
- For seed and its neighborhood points, compare the angle between normal of *seed* and its neighbor. If the angle difference is less than the smoothness threshold, save this point into the next step;
- For angles less than the threshold, add the current neighborhood points to the cluster.

The class $pcl :: RegionGrowing$ is used to cluster the point cloud based on normals and curvatures. The implementation of region growing requires input of some initial parameters, such as *cluster size*, *neighbor_number* (number of neighbors to use), *smoothness threshold* and *curvature threshold*.

After all clusters have been segmented, we build vectors for each cluster by using the function $std :: Vector$, then we use the function $pcl :: RegionGrowing :: extract$ for cluster extraction.

Fig.4.29 shows the region growing segmentation results by using the following parameters:

- $RegionGrowing.setMinClusterSize(500)$;

- $RegionGrowing.setNumberOfNeighbours(300);$
- $RegionGrowing.setCurvatureThreshold(1.0);$

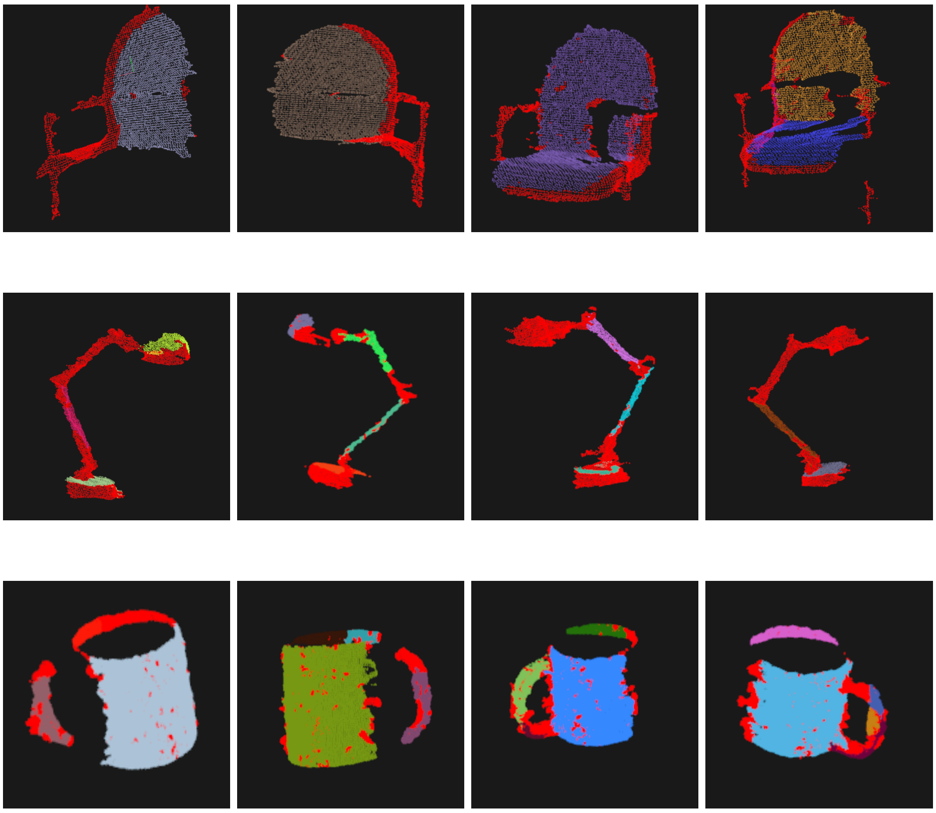


Figure 4.29: Region growing segmentation results of a chair, lamp and mug. Each row from left to right is point clouds captured from camera1, camera2, camera3 and camera4. Different color indicates different clusters.

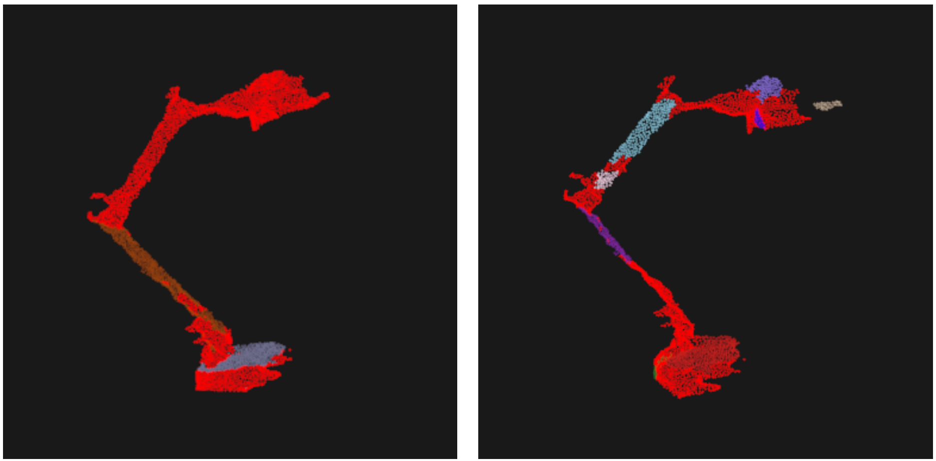
The results in Fig.4.29 are all obtained by running the region growing segmentation under the same parameters. However, for the same object, the number of clusters varies in different point clouds. Table.4.3 shows the number of clusters of each point cloud.

Table 4.3: Number of clusters of object point clouds

	camera1	camera2	camera3	camera4
Chair	1	1	1	3
Lamp	3	4	3	2
Mug	3	4	4	5

From Table 4.3 we learn that for the same object, region growing method causes splitting in a different number of clusters. In our project, we need to find the common parts of different point clouds based on these clusters, the different number of clusters means that this segmentation method cannot segment object based on its components (parts), which means that the common cover part of the point clouds from different cameras is hard to determine. Therefore, this method is also ineffective in our project. Another limitation of region growing segmentation is that it relies on multiple input parameters. Fig.4.30 shows the segmentation results when use different parameters. Fig.4.30(a) shows the segmentation result of the lamp by using parameters described before, Fig.4.30(b) shows the segmentation result of the lamp by using the following parameters:

- $RegionGrowing.setMinClusterSize(50);$
- $RegionGrowing.setNumberOfNeighbours(300);$
- $RegionGrowing.setCurvatureThreshold(1.0);$



(a) parameters_1

(b) parameters_2

Figure 4.30: Region growing segmentation results of lamp by using different initial parameters. Fig(a) has 2 clusters, Fig(b) has 8 clusters. The red part is the original point cloud.

Segmentation with settings as in (a) results in 2 clusters and segmentation with settings as in (b) leads to 8 clusters. From Fig.4.30, we can conclude that in this case, the (a) result is more suitable because too many clusters make it is unable to find a cluster that can completely contain a part of the lamp. In case (b), we set smaller *MinClusterSize*, which means that clusters with fewer points are still considered to be a *cluster*, leading to an increase in the number of clusters. Also, the smaller $NumberOfNeighbours$ leads to search within a shorter distance, which also affects the number of clusters and their sizes. The $Threshold$ also influences the number of clusters, because for a larger $Threshold$, points on a distance will be treated as similar points and be segmented into the same cluster. In Chapter 5 we analyze the influence of initial parameters on the segmentation results in more details.

In theory, the region growing segmentation algorithm is based on normal and curvature, while the point cloud data obtained from the depth camera is noisy on depth, the curvature changes and does not reflect the actual object surface and will confuse the algorithm.

In conclusion, region growing is not very useful; the first reason is that the region growing segmentation produces different numbers of clusters for point cloud of the same object, which is not conducive to the subsequent comparison and selection of point cloud clusters. Secondly, the region growing algorithm relies on the input parameters. In practice, different input parameters are required for diverse objects. If we want to achieve an ideal result, manual operation is required, which does not meet the requirements of a real-time system.

4.5 PointNet++ with Pipeline

In our next experiment, we validate the pipeline with a deep learning network. We test the pipeline with different dataset. In this section, we will illustrate the experiment settings, goals and presents the experimental results.

4.5.1 Validate with Chair Dataset

Goals: In the experiment of point cloud segmentation by using the Point Cloud Library in Section.4.4, only the plane model-based SAC segmentation results of chair ware found to be suitable to be used in our system. The point cloud segmentation using PCL has the following limitations: first, the point cloud segmentation method using PCL is difficult to generalize to other objects. In our experiment, of the three objects only fits the chair point cloud. Second, objects with different shapes in the same category have different segmentation results. It is uncertain that this method is valid for all objects in a category. Finally, the result of plane model-based SAC segmentation is only based on the plane surface, not on the components (parts) of the object. Therefore, it is not possible to select and compare the common parts of the point cloud captured by different cameras.

Settings: In order to compare with the previous experimental results, in this experiment we use the same set of point cloud data as in the previous experiments. The PointNet++ part segmentation is done in a pre-processing step. This experiment is conducted in the following settings:

System: Ubuntu 16.04
GPU: NVIDIA GeForce GTX 1050 (notebook)
GPU driver: 396
Deep learning framework: Tensorflow with Cuda Toolkit 9.0 and cuDNN SDK 7.0
Python: version 2.7

In this experiment, we first selected and set the training set from ShapeNet[104] for model training, which contains 4765 3D chairs with four labelled parts (back, seat, arm and leg). Then, we test the trained model with the rest data from the same dataset.

The next step is to test the real data from the RealSense camera with the trained model. We transfer the PLY file into PTS file and set the predicted parts with different colors: back (12, 242, 12), arm (242, 12, 242), seat (12, 12, 242) and leg (242, 12, 12), and then export the predicted chair into PLY file and send it to a C++ program.

The volume calculation and part selection are done in C++. In order to compare the similarity of multiple point clouds, we first calculated the cube diagonal of point cloud by using their XYZ coordinates. The diagonal can represent the volume value of the point cloud. The similar (selected) parts should meet the following conditions:

- The diagonal value of same part (color) from four cameras should have the minimum Mean Square Error.
- The number of points in the selected parts point cloud should be higher than 200.

The part with the smallest MSE value represents that parts acquired by different cameras are the most similar parts in volume, and the limit of the number of points excludes the parts with too few points. In practice, parts with smaller than 200 points are not enough to form a complete part and cannot be used in point cloud registration. With these conditions, we then select the most suitable part as the input point cloud data for the ICP registration step.

Results: Fig.4.31 shows the segmentation results by using PointNet++: the chair is segmented into four parts: arm, back, seat and leg. Each part is labelled with different colors. Based on the color information, we input this segmentation result into the part selection process.

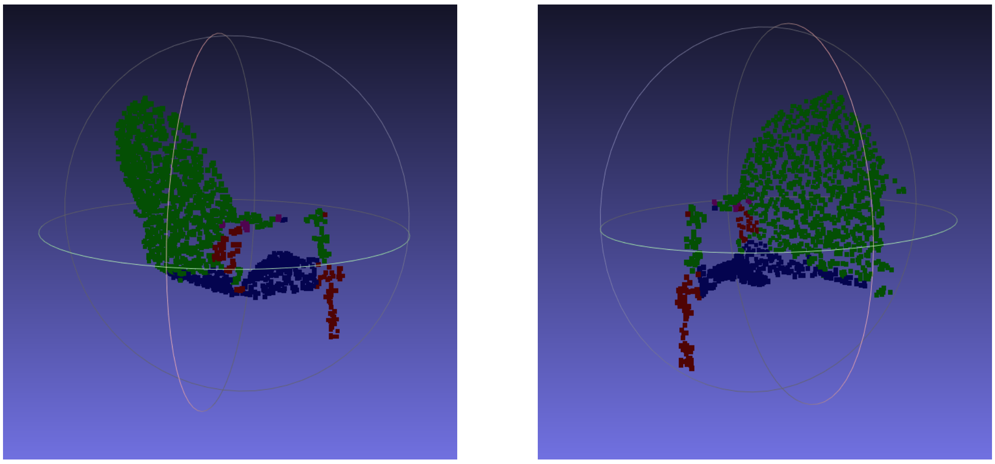


Figure 4.31: Output point cloud from PointNet++ network

In part selection step, we first use *SegObject* to segment point cloud based on the color information, in this step, we get 16 point clouds from 4 cameras. The function *Dcalculator* then calculates the cube diagonal of these point clouds. For each set of point clouds with the same color, we use *calMSE* to compute the MSE value for each part, and the function *PartChoice* will finally choose the part with the smallest MSE value. In this experiment, the program automatically selects the chair back as the best part for point cloud registration.

The selected part will be processed in the ICP algorithm, Fig.4.32 shows the fusion visualization results after point cloud registration. Due to the small number of points in the point cloud, it is a little bit difficult to visually judge the degree of coincidence. In this case, we mainly analyze the result based on the statistical metrics.

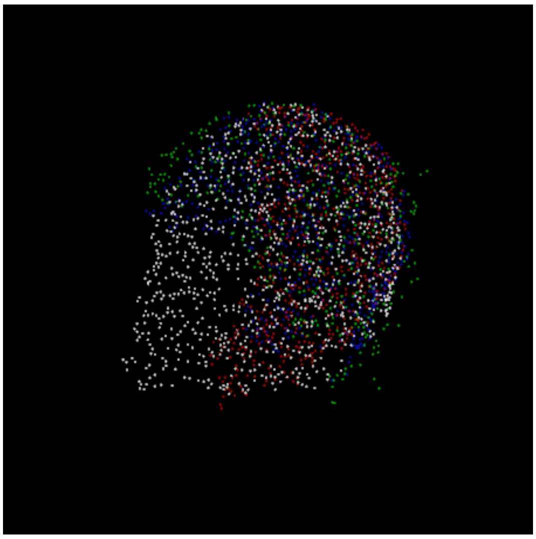


Figure 4.32: Aligned point cloud (from four cameras) after 40 iterations.
The red point cloud comes from camera2, the green point cloud from camera3 and blue point cloud from camera4

Analysis: The goal of this experiment is to replace the manual selection of the appropriate part into the automatic selection. Therefore, we analyze and compare the experiment results with results in section 4.3.

Table 4.4 shows the metrics comparison in all conditions. From the table we can conclude that: deep learning requires the shortest processing time due to the small number of points in the point cloud. In 3D real-time reconstruction, this feature will save processing time and help to reduce the time delays. For fitness score, the deep learning method performs better than the original point cloud and is less than the manual selection process. The reason is that in the manual selection process, we select the most appropriate part based on our human experience and judgment, and we manually select a complete and similar part from all captured point clouds. However, in the deep learning process, the predicted segment result is not as ”perfect” as the manual selection, so the convergence result is less than manual selection. However, this method still improves the performance of the classical ICP algorithm and avoids the drawback of falling into a local optimum. Compared with the SAC segmentation method, the deep learning method shortens the processing time but achieves worse registration accuracy. Due to the fact that the deep learning method contains 4765 items in the training data, the deep learning method can be extended to a variety type of chairs. Also, the output of the deep learning method divides the chair into different parts according to the components, which is more suitable for the comparison and selection of the appropriate parts used for ICP registration.

Table 4.4: Evaluation and comparison of registration based on the original point cloud, manual selection, SAC segmentation and deep learning (DL) processed point cloud - Chair Point Cloud

	Point Cloud	Processing Time	Number of iteration	Fitness Score
Source Camera2	Original	24782ms	40	1.112e-3
	SAC-based	11764ms	40	7.325e-5
	DL-based	6939ms	40	1.875e-4
	Manual	10524ms	40	1.075e-4
Source Camera3	Original	46031ms	40	3.397e-3
	SAC-based	17501ms	40	2.386e-4
	DL-based	4654ms	40	1.507e-4
	Manual	18192ms	40	1.791e-4
Source Camera4	Original	39096ms	40	5.687e-3
	SAC-based	11195ms	40	1.802e-4
	DL-based	4885ms	40	1.167e-3
	Manual	15226ms	40	4.518e-4

4.5.2 Validate with other Datasets

In order to verify the effectiveness of the system, we use different calibration objects datasets to train the deep learning network and segment the point cloud data. In this section we will demonstrate and analyze the experimental results of different calibration objects.

Lamp

We validate the pipeline with the Lamp dataset in the ShapeNet. In this experiment, we input original point clouds from four cameras to our pipeline. Fig.4.33 shows the registration results.

Then, we train the PointNet++ with lamp dataset and segment the original point clouds by using the output trained model. Fig.4.34 shows the result of segmentation. The lamp is divided into three parts: the lampshade part is blue, the lamp bracket is pink and the lamp holder is labelled with green. The part selection program chose the lamp bracket as the most suitable part for point registration and after 40 iterations. Fig.4.35 shows the aligned point cloud of the segmented lamp.

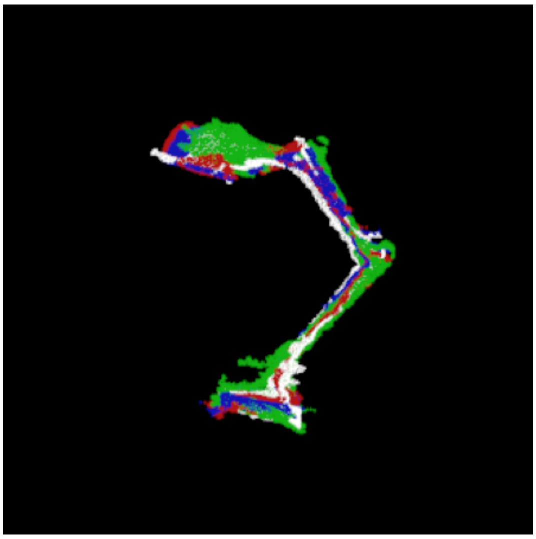


Figure 4.33: Aligned lamp point cloud of four depth cameras after 40 iterations.

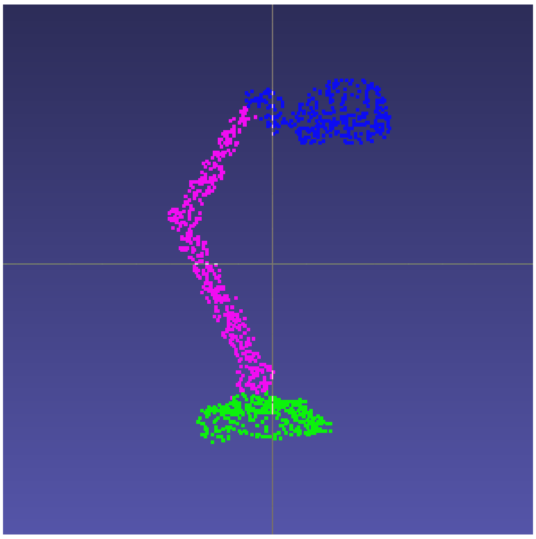


Figure 4.34: The PointNet++ segmented point cloud of the lamp

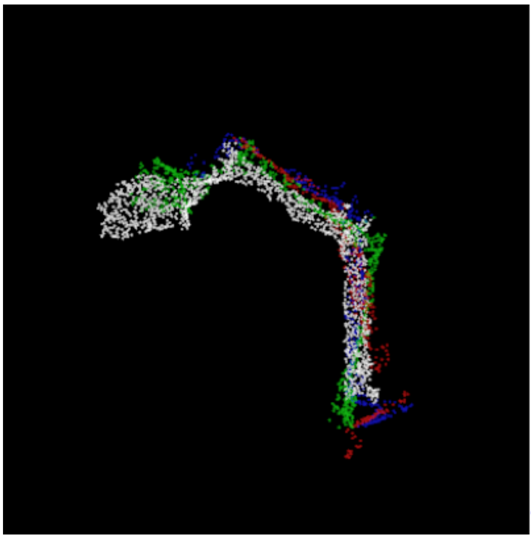


Figure 4.35: Aligned segmented lamp point cloud after 40 iterations

Table 4.5 shows the statistical comparison between the original point cloud and the segmented point cloud. Due to the incomplete input point cloud data, the point cloud segmentation resulting from the deep learning network is not good enough. For example, the selected part, in this case, is the lamp bracket, and the white point cloud in Fig.4.35 contains both the lamp bracket and the lampshade.

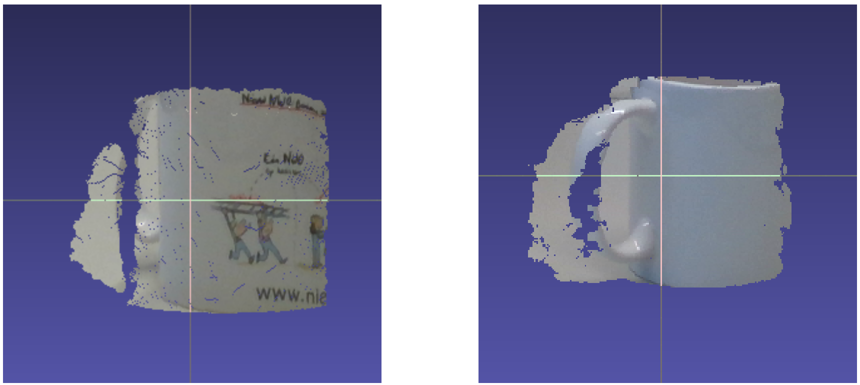
Table 4.5: Registration evaluation and comparison of registration of the lamp point clouds

	Point Cloud	Fitness Score	Number of Iterations	Processing time
Original	source camera2	2.509e-4	40	14653ms
	source camera3	2.403e-4		
	source camera4	2.500e-4		
Automatic	source camera2	6.342e-4	40	4781ms
	source camera3	3.602e-4		
	source camera4	2.800e-4		

Also, the original point clouds have obtained the ideal registration result, and there is no interference part in the lamp object that makes the ICP algorithm fall into a local optimum. We conclude that the feature-based point cloud registration pipeline does not improve the accuracy much, but it shortens the processing time.

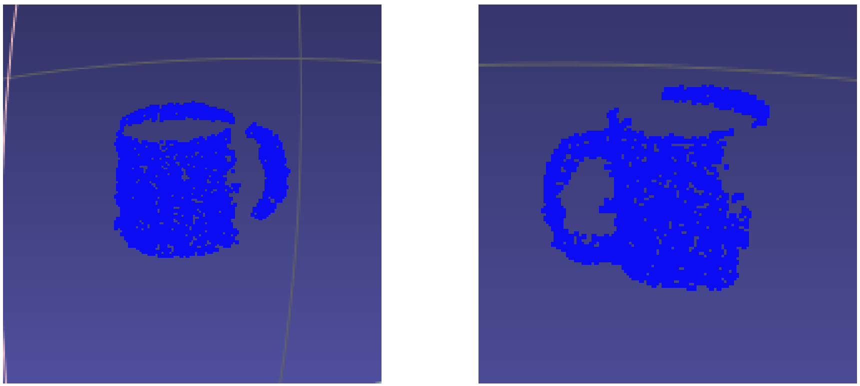
Mug

We also validate the PointNet++ with mug dataset. The original point cloud is shown in Fig.4.36. For the mug, the training point cloud only contains two parts: mug body and mug handle. Fig.4.37 shows the deep learning segmentation results of a mug.



(a) Camera1 (b) Camera3

Figure 4.36: Original mug point clouds.



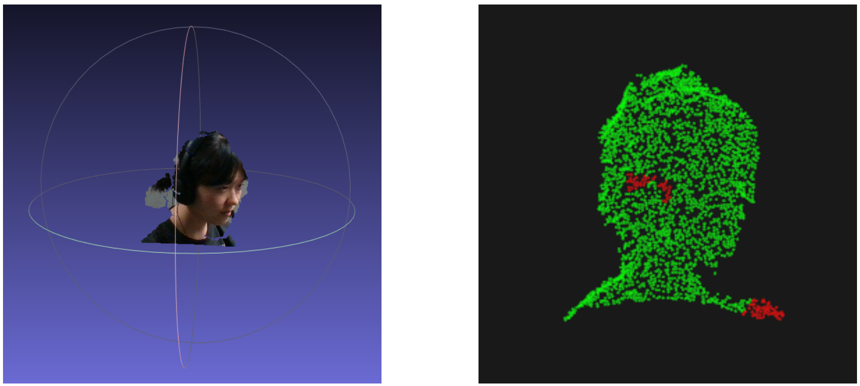
(a) Camera1 (b) Camera3

Figure 4.37: The PointNet++ segmented mug point cloud.

For figure (a) from camera1, the mug body and mug handle are separated due to occlusion, for figure (b) from camera2, the shape of the mug is destroyed due to the inconsistent point cloud of the mug body. Due to the incomplete and occlusion problems, the deep learning network was unable to segment the mug captured from the depth camera correctly.

Earphone

In this experiment, we also validate the pipeline with earphones as a calibration object. We capture point clouds of a person wearing an earphone and train the PointNet++ network using the dataset of headphones. The original and segmented point clouds are shown in Fig.4.38. The segmentation network cannot correctly segment the point cloud, so this pipeline cannot divide the point cloud of human wearing earphones.



(a) Original Point Cloud

(b) Segmented Point Cloud

Figure 4.38: The original and segmented point clouds of a person wearing a earphone

From Fig.4.38 we can conclude that the earphone part cannot be easily identified in the original point cloud. Due to the unrecognizable shape of the point cloud of the human head and earphone, the PointNet++ network is unable to segment this kind of point clouds correctly.

4.6 Feature-Based Coarse Registration with Pipeline

Goals: The typical camera calibration method requires additional calibration objects such as checkerboard[128] or printed pattern[57]. The steps it involves are cumbersome and time-consuming. In our pipeline, the coarse registration is a one-time operation, but if the camera position changes, the initial transformation matrix needs to be recalculated. PCL provides a series of feature extraction and alignment classes, which can be used to calculate the initial transformation matrix. In this experiment, we test the result of FPFH feature extraction and alignment functions and compare it with results of MATLAB toolkit calibration.

Settings: Fig.4.39 shows the work flow of FPFH feature extraction and alignment. Fast Point Feature Histogram (FPFH) is extended from the Point Feature Histogram, which provides simplification and optimization for accelerating the calculation process as described in 2.5.2. The input of FPFH is the point cloud with standard XYZ information. The output is the histogram which can reflect the features of the points. We first use the FPFH to extract features, calculate the histogram and build feature vectors for alignment, then we use the Sample Consensus Initial Alignment to align features and calculate spatial correlation.

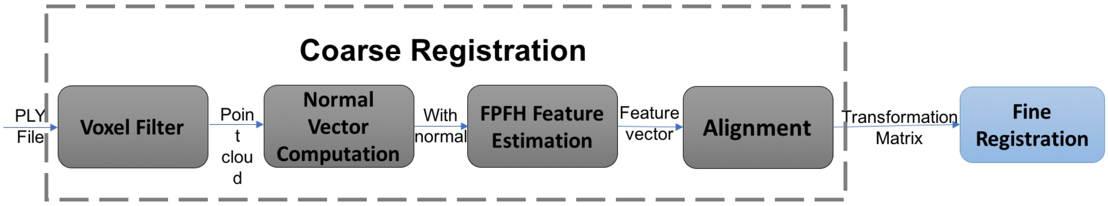


Figure 4.39: Flow of FPFH feature estimation and alignment

Voxel Filter: The filter used here is the voxel filter, which is introduced in Section.4.4.1. The class used here is $pcl :: VoxelGrid < pcl :: PointXYZ >$.

Normal Vector Computation: The normal information is used here to measure the histogram component values. The region growing segmentation also computes normal vectors for point sorting, which is introduced in Section.4.4.2. The class used here is $pcl :: NormalEstimation$.

FPFH Feature Estimation: The feature histogram is calculated separately and then combined into a feature vector. The FPFH feature estimation includes the following steps:

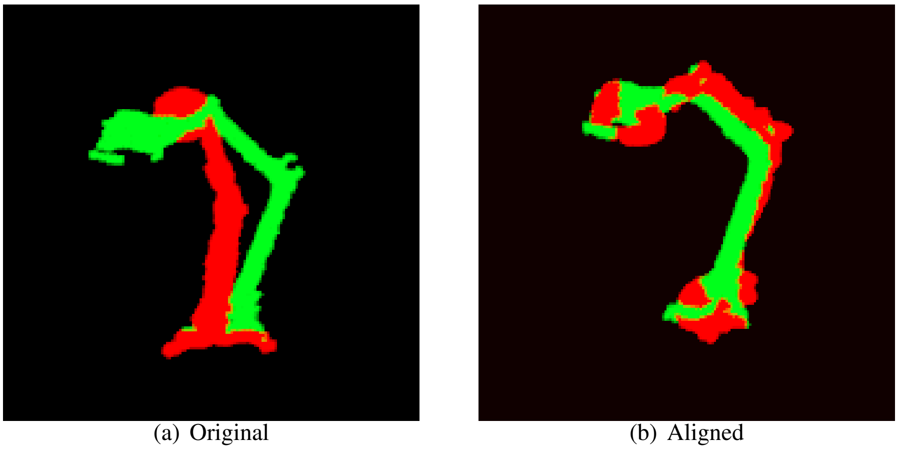
- Obtain the neighborhood elements of point p;
- Calculate the three angular values $(\alpha, \varphi, \theta)$ of p and its neighbor k. These angular values are calculated based on the difference of point normal;
- Output the statistical values to a SPFH histogram;
- Obtain the nearest neighborhood element of point p;
- Use the SPFH calculated from the third step and weigh it to get the final FPFH.

The above steps are fulfilled by using functions in class $pcl :: FPFHEstimation < pcl :: PointXYZ, pcl :: Normal, pcl :: FPFHSignature33 >$. The feature vector is represented by $FPFHSignature33$ point type.

Alignment: The alignment is based on the Sample Consensus Initial Alignment (SAC-IA) algorithm[97]. This algorithm contains the following steps:

- Select sample points from a point cloud within a distance d;
- Select a list of points from another point cloud which is similar to this sample points' histogram based on the FPFH features;
- Use Huber Penalty[61] metric to measure the transformation matrix between sample points and its corresponding points. The transformation with the lowest metric value is the output transformation matrix.

Results: Fig.4.40 shows the results of a lamp based registration by using the FPFH feature extraction and alignment. The left one shows the original point clouds. The right one shows the aligned point clouds, which demonstrates that the alignment method matches the *target* point cloud and the *source* point cloud.



(a) Original

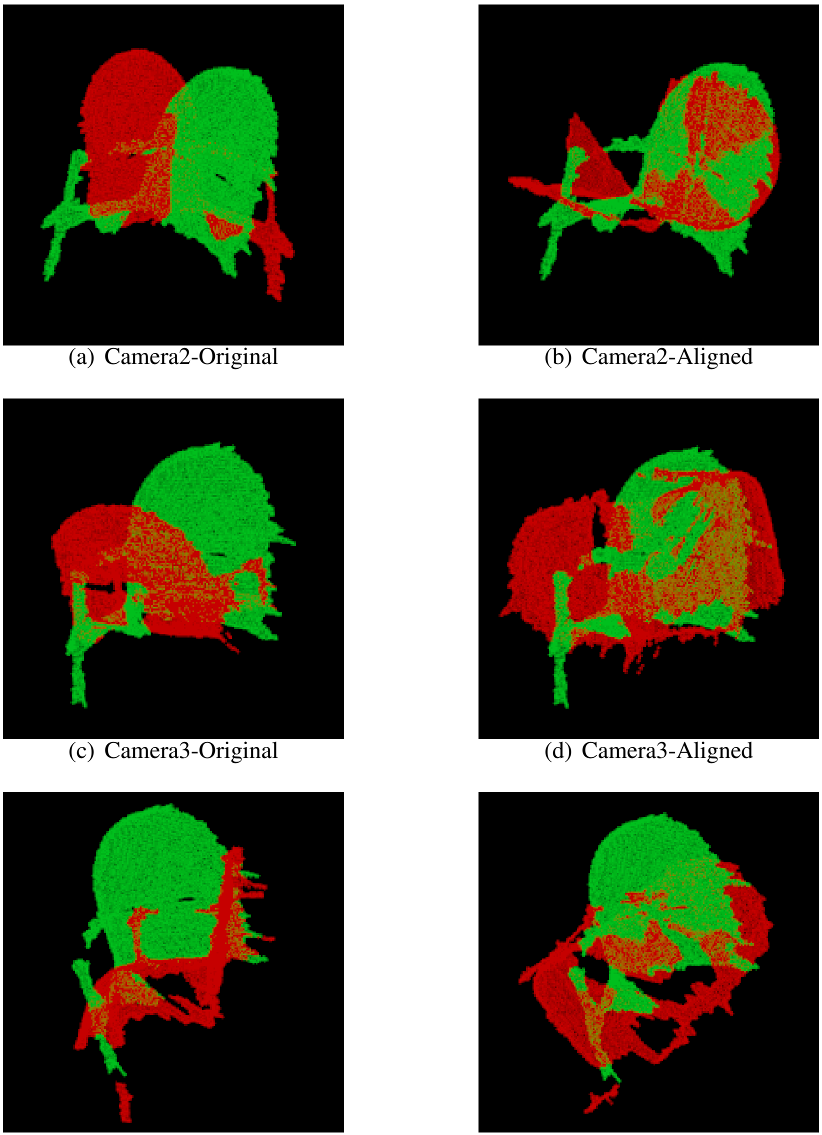
(b) Aligned

Figure 4.40: The alignment result of the lamp by using FPFH feature extraction and SAC-IA alignment.

Fig.4.41 shows the registration results of a chair based registration by using the FPFH feature extraction and SAC-IA alignment. The left column represents the original spatial relationship of the *target* point cloud and *source* point cloud. The right column is the alignment results. From Fig.4.41, we can conclude that this alignment method can only provide a coarse registration result.

Analysis: We compare the performance of feature-based alignment with MATLAB calibration toolkit method by evaluating the ICP registration performance. We use the initially calculated transformation as the input of the ICP algorithm. For the same point cloud data, we let the ICP program iterate for 40 times and compare the performance of processing time and fitness score. In this experiment, we tested two datasets; one is the original chair point cloud, another one is the deep learning network segmented point cloud. Table.4.6 shows the statistical comparison of ICP algorithm performance when using a different initial transformation matrixes. The *c* represents the source camera that captures the point clouds.

From Table 4.6 we can conclude that in general these two methods have comparable performance. The running time and fitness score of these two methods are similar. Regarding to performance, these two methods behave similarly. For practical applications, MATLAB toolkit calibration is complex and relies on handling a calibration checkerboard. This process requires the manual acquisition of RGB images and manual corner detection. The FPFH&SAC-IA method only needs to input the PLY files and does not require any additional calibration objects. Therefore, the FPFH&SAC-IA method is more suitable for use in our pipeline.



(a) Camera2-Original

(b) Camera2-Aligned

(c) Camera3-Original

(d) Camera3-Aligned

(e) Camera4-Original

(f) Camera4-Aligned

Figure 4.41: The alignment results of the chair by using FPFH feature extraction and SAC-IA alignment: The left column is the original point clouds, the right column is the aligned point clouds.

Table 4.6: Evaluation and comparison of the ICP algorithm by using different initial transformation matrixes on original and segmented Chair point cloud

	Transformation Matrix	Processing Time	Number of Iterations	Fitness Score
Original(c2)	MATLAB	24782ms	40	1.112e-3
	FPFH& SAC-IA	13294ms	40	1.159e-3
Original(c3)	MATLAB	46031ms	40	3.397e-3
	FPFH& SAC-IA	25094ms	40	3.332e-3
Original(c4)	MATLAB	39096ms	40	5.687e-3
	FPFH& SAC-IA	20959ms	40	3.811e-3
Segmented(c2)	MATLAB	6939ms	40	1.875e-4
	FPFH& SAC-IA	7733ms	40	4.411e-4
Segmented(c3)	MATLAB	4654ms	40	1.507e-4
	FPFH& SAC-IA	4663ms	40	1.409e-4
Segmented(c4)	MATLAB	4885ms	40	1.167e-3
	FPFH& SAC-IA	4861ms	40	5.764e-4

Chapter 5

Analysis

In Chapter 4, we introduced the goals and settings of the experiments conducted in this thesis and showed the experimental results. In this chapter, we will analyze the experimental results and pipeline performance through qualitative analysis, quantitative analysis and, comparison with the state-of-the-art.

In this thesis, we present a feature-based multi-camera calibration pipeline. The system segments the relatively complete common part of the captured point clouds by point cloud segmentation and part selection and uses the segmented part to perform point cloud registration, thereby making up for the defect that the ICP registration algorithm ignores the point cloud 3D shape and falls into the local optimum error. In this chapter, we conduct a qualitative analysis of the results of the segmentation and registration of each test object. For point cloud registration, we use three metrics for quantitative analysis. Finally, we compare and analyze the pipeline in this paper with the state-of-the-art.

5.1 Qualitative Analysis

In the qualitative analysis, we evaluate the performance of the point cloud segmentation and registration visualization results based on subjective experience.

In the feature-based multi-camera calibration and point cloud registration pipeline, we used different point cloud segmentation methods to divide the point cloud. The results of point cloud segmentation have an impact on the subsequent feature-based point cloud alignment results.

The point cloud segmentation methods used in this paper include manual segmentation, SAC segmentation, region growing and PointNet++. In the quantitative analysis of the point cloud segmentation results, the performance of the segmentation results is analyzed by the completeness and accuracy. The ***completeness*** is used to measure whether the segmentation result is complete and contains as many $target$ points as possible. The ***accuracy*** is used to measure whether the segmentation result contains incorrect segmentation parts, segmentation results with points that are incorrectly divided will result in low accuracy.

In this project, we used a chair, lamp, mug and cap as a calibration object to validate the pipeline. The point clouds captured from multiple depth cameras are in different camera coordinate systems. The purpose of point cloud registration is to make the point cloud captured in the *source* camera align with the *target* point cloud through rotation and translation, so that the point clouds in different coordinate systems can be transformed into the same coordinate system. The transformation results provide camera pose estimation and tracking information for the next 3D reconstruction step. The qualitative analysis of point cloud registration is based on the alignment visual results, the correctness of point cloud coincidence is judged according to the 3D shape and structure of aligned point clouds.

In this section, for each object, we first qualitatively analyze the results of different segmentation methods to evaluate the segmentation performance. Then, the qualitative analysis of original point cloud registration and feature-based point cloud registration results is performed.

5.1.1 Calibration Object - Chair

In the project, we use a chair as the calibration object. Chairs are common in daily life and can be conveniently used as a calibration object, and the size of the chair is also suitable for calibrating multiple cameras. The next aim of our project is to calibrate and track the camera according to the human point cloud. The position of the camera needs to be suitable for capturing the complete human 3D point cloud data. The oversized or too small calibration object (such as a mug) will limit the camera position. In this section, we qualitatively analyze the results of chair point cloud segmentation and registration.

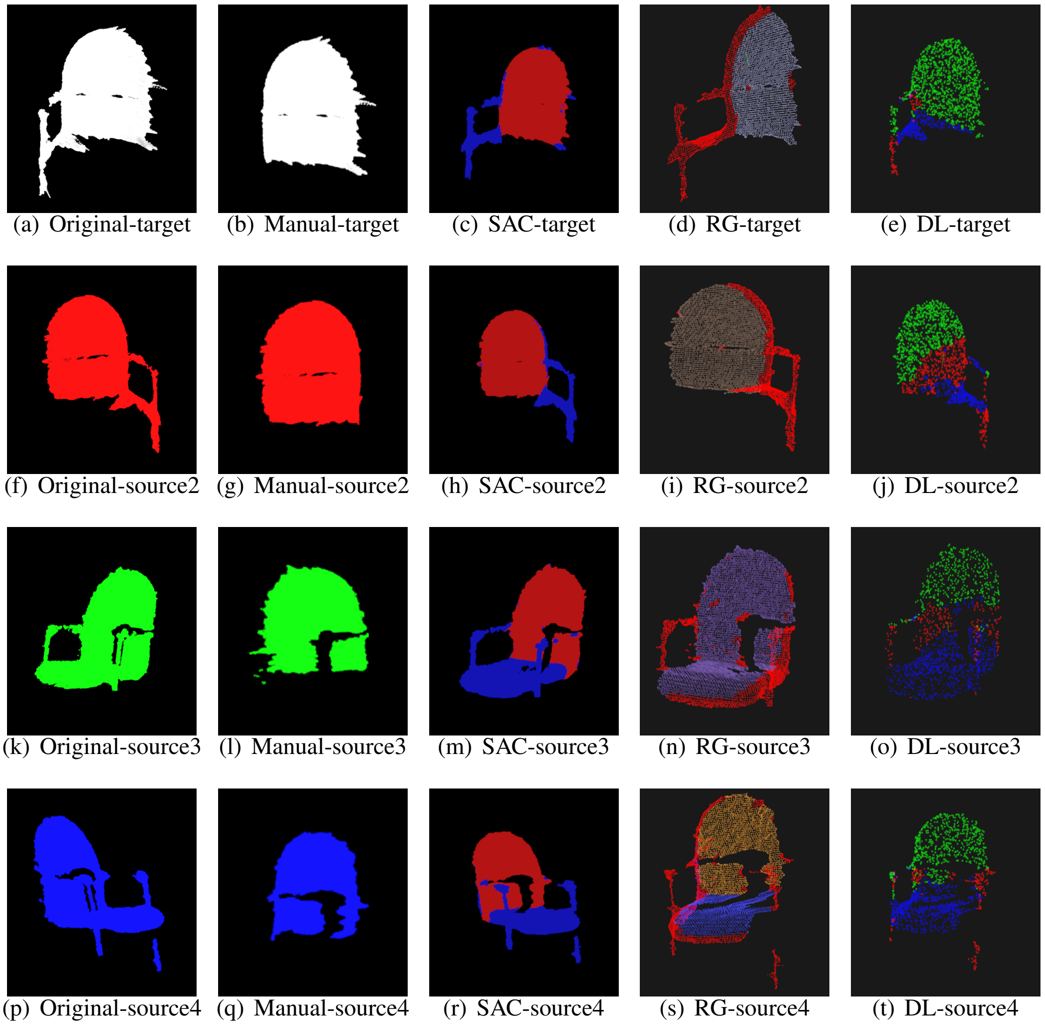
Point Cloud Segmentation - Chair

Fig.5.1 shows the segmentation results by using different segmentation methods including SAC-based plane segmentation, region growing segmentation and PointNet part segmentation.

The red part in the plane model-based SAC segmentation results is the segmented plane. The different color in region growing segmentation point cloud represents different clusters (parts). For deep learning segmentation results, the green part is the chair back, the blue part represents the chair seat, the purple part is the armrest, and the red part represents the chair legs.

The manual segmentation is achieved by using MeshLab[23] to remove points outside the chair back. This operation is based on the subjective experience and judgment of the experimenter; the manual segmentation result is treated as the *best* result of segmentation in qualitative analysis.

In order to realize automatic point cloud segmentation, we segmented the chair using the functions provided by the Point Cloud Library (PCL). The 3D geometry structure of *target* chair back part is a plane, the SAC plane segmentation function provides a model for segmenting a plane in a given point cloud. Therefore, we used the plane model-based SAC segmentation to subdivide the chair point cloud. This algorithm extracts a set of point clouds that satisfies the plane model constraints and contains the largest number of points as the segmented plane. In the experiment, we obtain segmentation results shown in Fig.5.2



(a) Original-target (b) Manual-target (c) SAC-target (d) RG-target (e) DL-target

(f) Original-source2 (g) Manual-source2 (h) SAC-source2 (i) RG-source2 (j) DL-source2

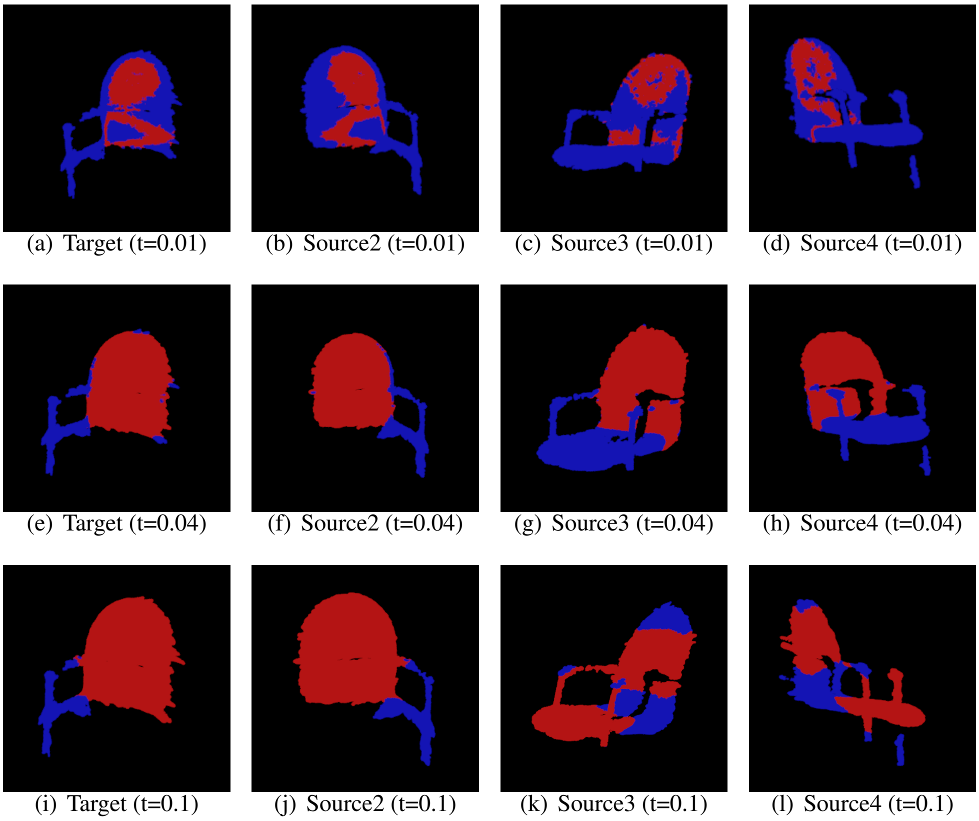
(k) Original-source3 (l) Manual-source3 (m) SAC-source3 (n) RG-source3 (o) DL-source3

(p) Original-source4 (q) Manual-source4 (r) SAC-source4 (s) RG-source4 (t) DL-source4

Figure 5.1: Chair Point Cloud Segmentation Results. From the top to the bottom are point clouds from *target* camera1, *source* camera2, *source* camera3 and *source* camera4. From left to right of each line are the original point cloud, manual segmentation, SAC segmentation, region growth and PointNet++ segmentation results.

by inputting different $DistanceThreshold$ parameters.

The SAC segmentation method iteratively calculates a set of *INLIERS* (valid) points from the input point cloud (observation points) as the segmented part, which satisfies the parameter constraints of the specified model (in this case the model is a plane). In the experiment, we set different thresholds t by using function $setDistanceThreshold()$, the rest of the parameters are the same. Furthermore, $DistanceThreshold$ represents the threshold of the $distance$ between the observation point and the model, and $distance$ represents the value of the dot product between the observation point vector and the plane normal vector. When the



(a) Target (t=0.01) (b) Source2 (t=0.01) (c) Source3 (t=0.01) (d) Source4 (t=0.01)

(e) Target (t=0.04) (f) Source2 (t=0.04) (g) Source3 (t=0.04) (h) Source4 (t=0.04)

(i) Target (t=0.1) (j) Source2 (t=0.1) (k) Source3 (t=0.1) (l) Source4 (t=0.1)

Figure 5.2: Segmentation results of chair when t = 0.01, 0.04 and 0.1. The parameter t represents the *DistanceThreshold*. Target and Source represents different capture cameras. The blue point cloud is the original input, and the red part is the segmented point cloud.

distance is greater than this threshold, the observation point is not within the target model. As can be seen from Fig.5.2, when the threshold is too small ($t = 0.01$), the plane segmentation result contains only parts of the chair back, and the segmentation results are incomplete. When the threshold is too large ($t = 0.1$), the incorrect points with larger *distance* from the model is judged as being part of the back of the chair, which cause wrong segmentation results in (k) and (l). When $t = 0.04$, the algorithm achieves the best segmentation result, the 3D shape of the chair back is complete, and the segmented part does not contain non-backrest points.

However, the SAC segmentation method has limitations. The SAC segmentation can only make a subdivision based on a specified model, such as the plane used in this experiment, and cannot segment the chair according to its component structure. In our pipeline, we manually select the back of the chair by first determining which part of the chair is the most complete and common part of different cameras, then we divide the chosen part out and enter it into the registration step. In order to automation the above operation, the auto-segmentation also requires the following steps: first, the point cloud is segmented according to the component composition of the object, and then the most suitable portion is compared and selected based on the volume (or other criteria) of the segmented parts. The SAC segmentation method cannot divide the point cloud into different parts, which does not fully meet the requirements of automatic point cloud segmentation and part selection.

In order to automatically segment the point cloud into different parts, we use the normal-based region growing algorithm for point cloud segmentation. In the experiment, we set the same input parameters for point clouds captured by different cameras. From the segmentation results in the fourth column of Fig.5.1, we can conclude that under the same initial parameters and experimental conditions, different input point clouds obtained different numbers of segmentation clusters. For the *target* point cloud and *source*2, the algorithm completely separates the chair back. For *source*3, the algorithm assigns the chair back and chair seat into the same cluster. For *source*4, the algorithm segmented the point cloud into two clusters. The region growing segmentation results of the chair have high completeness and low accuracy. This method completely segments the seat back in all cases, but for *source*3, due to the depth information and capture angles of the point cloud, the method incorrectly divides the seat into the back cluster, which leads to the low accuracy.

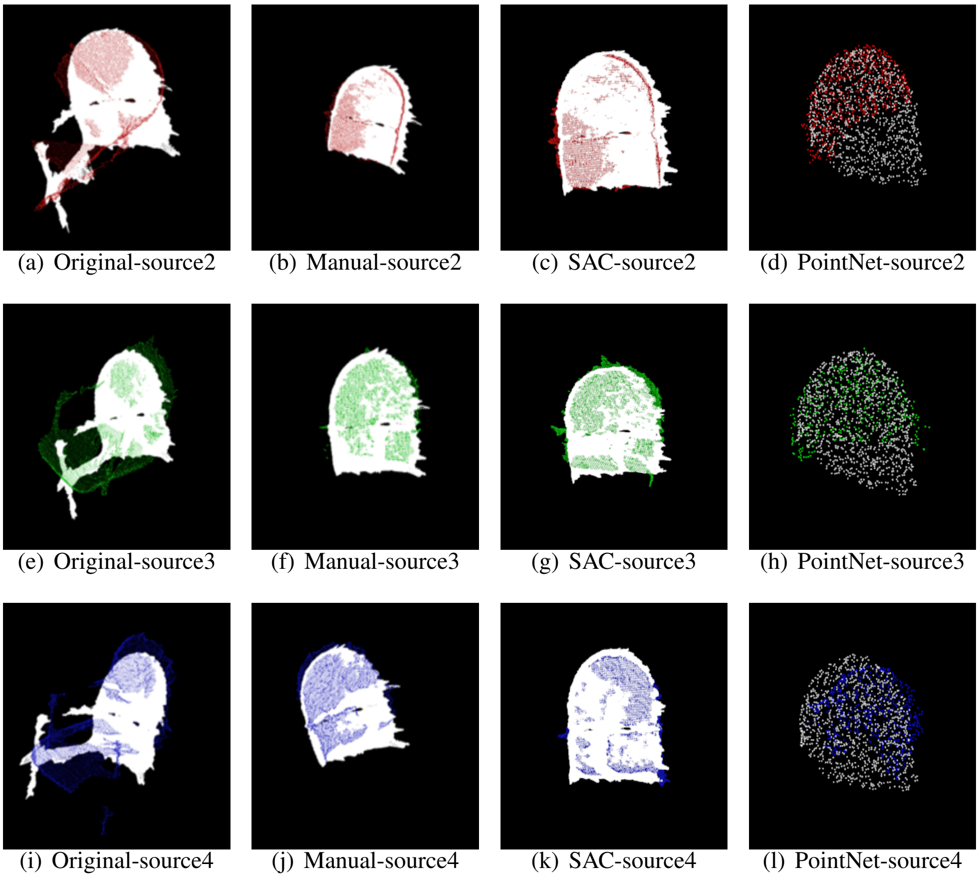
In conclusion, the segmentation results of the region growing algorithm are not accurate, and different numbers of clusters are obtained under the same initial conditions (see Section4.4.2). In our pipeline, the next step of segmentation is the part selection. Since segmentation leads to different numbers of clusters for various point clouds, these clusters have no features or labels that can be compared with each other. For example, in this experiment, based on the segmentation results we cannot judge which one of these two clusters of *source*4 belong to the chair back. If there is no human intervention, we also can not confirm whether the cluster in *target*, *source*2 and *source*3 is the chair back. Therefore, the region growing algorithm is not fit for our pipeline.

In the next step, we use PointNet++ to segment the chair in a more intelligent way. The experimental results show that the PointNet++ can divide the chair according to its components: back, seat, armrest, and leg. In the experiment, we assign different parts to different color values as labels and select a specific part based on the color values; then we select the most appropriate part for the point cloud registration by comparing the volume of the selected part. The two requirements for automatic segmentation and part selection are: segmenting point clouds by component composition; automatically selecting specific parts and comparing the volume of the point cloud. The PointNet++ segmentation method can meet these two requirements. Therefore, this method is suitable for our pipeline. From the experimental segmentation results, we can conclude that the PointNet++ can roughly divide the point cloud, but there are still large errors in the results. The point cloud segmentation in different parts is not complete and there are many incorrect classification points, so the completeness and accuracy of PointNet++ segmentation result are low.

Point Cloud Registration - Chair

After the point cloud segmentation step, we use the segmentation results as the input of point cloud registration. Next, we will qualitatively analyze the alignment results of different point clouds.

Fig.5.3 shows the point cloud registration results comparison of original point cloud, manual segmentation, SAC segmentation and PointNet++ segmentation point cloud.


(a) Original-source2 (b) Manual-source2 (c) SAC-source2 (d) PointNet-source2

(e) Original-source3 (f) Manual-source3 (g) SAC-source3 (h) PointNet-source3

(i) Original-source4 (j) Manual-source4 (k) SAC-source4 (l) PointNet-source4

Figure 5.3: Chair Point Cloud Registration Results. The white point clouds represent the *target* ones, the red point clouds represent the point clouds captured from *source* camera2, the green points represents *source* camera3 and the blue ones represent *source* camera4. From left to right of each row shows the registration results by using the original point cloud, manual segmentation, SAC segmentation and PointNet++ segmentation point clouds.

From Fig.5.3 we can conclude that for the original point cloud, the alignment results have significant errors. The point clouds in each set of registration results are not correctly matched. The cause of this error is due to the incompleteness of the captured point cloud. Taking the point cloud from *source* camera3 (the second row) as an example, the seat edge of the *source*3 point cloud is trying to coincide with the armrest of the *target* point cloud, so that the ICP algorithm falls into this local optimum and results in large errors in overall registration results. The alignment result errors of the other two sets are also due to interference from the armrest or the edge of the seat.

For this problem, we propose the following solution: select a part of the point cloud which is complete and be covered by both *target* and *source* cameras as the point cloud to be aligned, thereby removing the errors caused by the incomplete point cloud. In this experiment, the armrest that causes the error and the incomplete parts (the seat of the chair and chair legs) are removed, only the back part of the chair is retained as a $feature$ for point cloud registration. From the experimental visualization results, it can be observed that each *source* point cloud is correctly aligned to the *target* point cloud.

According to the 3D shape and contour of the chair registration in Fig.5.3, We can conclude that the accuracy of point cloud registration using the SAC segmentation is much better when compared to the original point cloud alignment. The segmented chair back point cloud obtained by using the PointNet++ is selected as the most suitable part for registration based on the calculation and comparison of the point cloud volume. However, the chair back part obtained by the deep learning network is more incomplete than the SAC segmentation results, from the figure we can hardly recognize the 3D shape and coincidence degree of the back point cloud. Therefore, qualitative analysis can not be used to determine the accuracy of point cloud registration, the quantitative analysis is required to evaluate the point cloud registration accuracy.

In summary, the feature-based point cloud registration has a better performance in qualitative analysis than the original point cloud registration. After removing the armrest and the seat edge, the ICP algorithm avoids the errors that fall into local optimum and ignore globally 3D information.

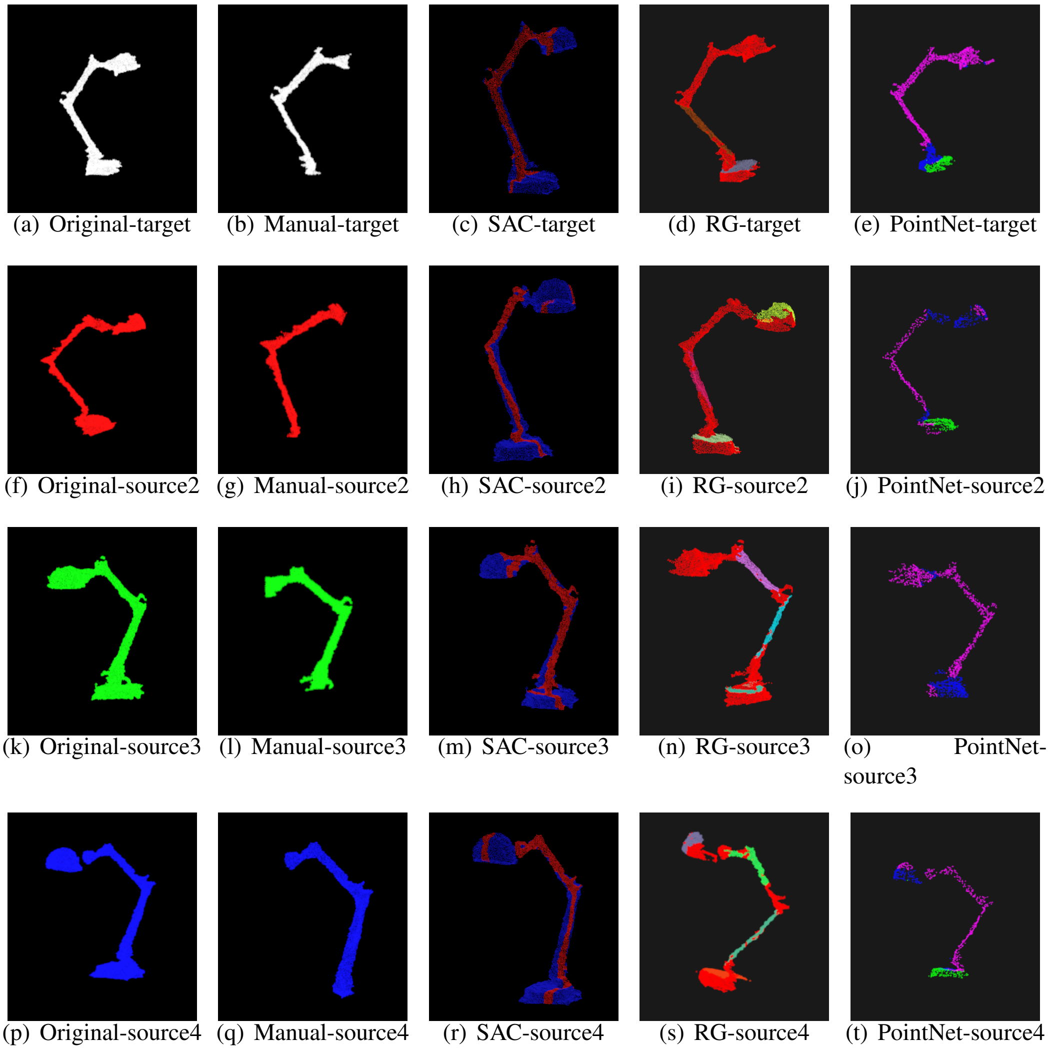
5.1.2 Calibration Object - Lamp

In the project, we use the lamp as the calibration object. Although the size of the lamp is appropriate, it needs to be placed on a table or other object to reach a suitable height. In this section, we will analyze the segmentation and registration results when using a lamp as a calibration object.

Point Cloud Segmentation - Lamp

Fig.5.4 shows the segmentation results of the lamp by using different segmentation methods. The lamp tested in the experiment consisted of three parts: a lampshade, a bracket, and a lamp holder. The lampshade is a hemisphere, the bracket is a combination of several cuboid sticks, and the lamp holder is a cylinder. Because the lampshade and the lamp holder are objects of uniform shape, the shape characteristics at different angles are the same, and the determined position information and features cannot be provided in the point cloud registration process. Therefore, these parts are not suitable for use in the ICP registration program. The 3D shape of the bracket in the lamp vary at different angles and these can be used to characterize the spatial orientation. Therefore, in the manual segmentation step, we made a subdivision based on the bracket as the ideal feature and use the bracket part for point cloud registration.

Similar to the chair, in the SAC segmentation of the lamp, we test the effect of different $DistanceThreshold$ parameter values on the segmentation results. From the test results shown in Fig.5.5, we can see that the larger the threshold, the more points are included in


(a) Original-target (b) Manual-target (c) SAC-target (d) RG-target (e) PointNet-target

(f) Original-source2 (g) Manual-source2 (h) SAC-source2 (i) RG-source2 (j) PointNet-source2

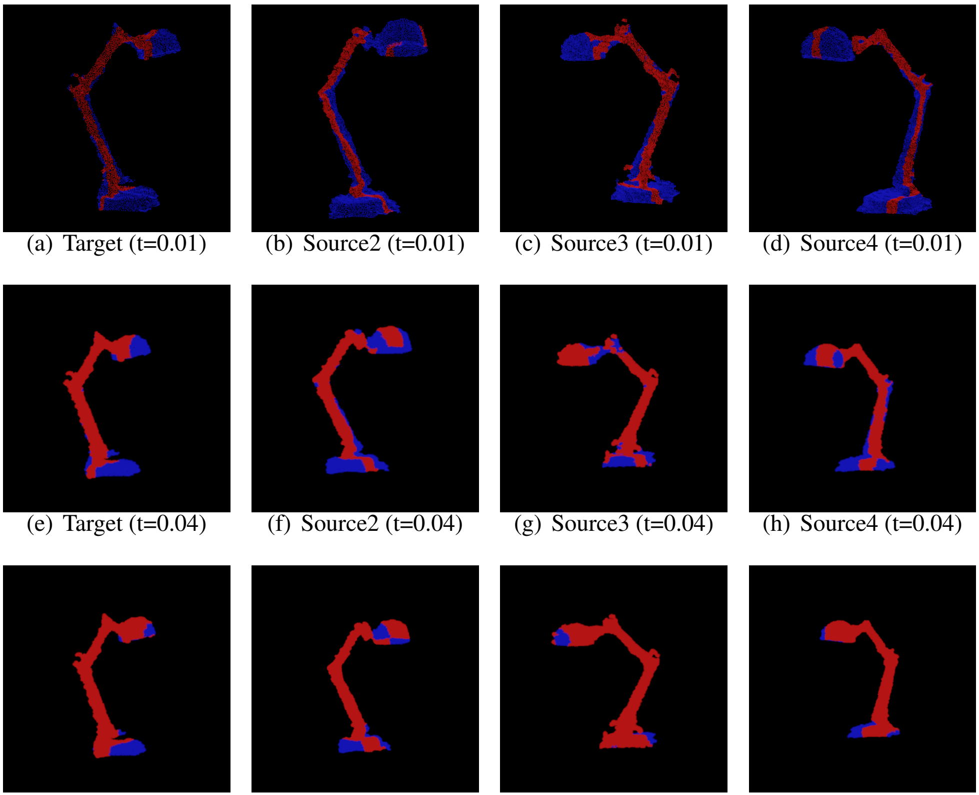
(k) Original-source3 (l) Manual-source3 (m) SAC-source3 (n) RG-source3 (o) PointNet-source3

(p) Original-source4 (q) Manual-source4 (r) SAC-source4 (s) RG-source4 (t) PointNet-source4

Figure 5.4: Lamp Point Cloud Segmentation Results. The point clouds in each column from left to right are original, manual segmentation, plane model-based SAC segmentation, region growing, and PointNet++ segmentation results.

the segmentation result. The reason is that a large threshold causes a point with a large *distance* from the model to be judged as *INLIERS* and is included in the valid point set so that the red point cloud is large in volume and the number of points included is increased.

When $t = 0.01$, the SAC plane segmentation method extracts the cross-sectional plane of the bracket from the lamp. In the point cloud segmentation results of four different cameras, the cross-sectional plane does not include the complete bracket, but only part of the bracket, which represents different sections and angles from the four point clouds. When the threshold increases, the SAC method did not correctly separate the bracket from the lamp, and the red point cloud contains many non-bracket parts. Therefore, the SAC segmentation results



(a) Target (t=0.01) (b) Source2 (t=0.01) (c) Source3 (t=0.01) (d) Source4 (t=0.01)

(e) Target (t=0.04) (f) Source2 (t=0.04) (g) Source3 (t=0.04) (h) Source4 (t=0.04)

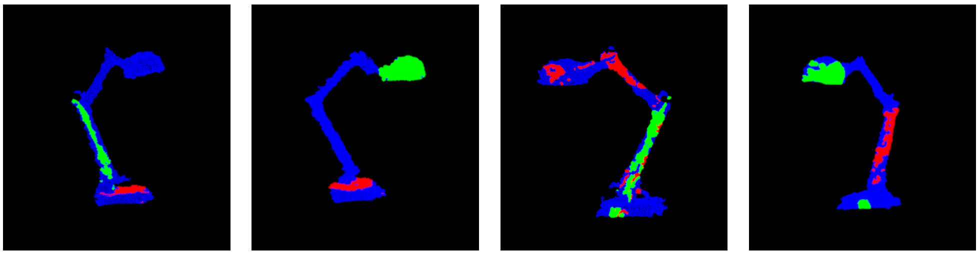
(i) Target (t=0.1) (j) Source2 (t=0.1) (k) Source3 (t=0.1) (l) Source4 (t=0.1)

Figure 5.5: Segmentation results of lamp when t = 0.01, 0.04 and 0.1. Parameter t represents the *DistanceThreshold*. Target and Source represent different capture cameras. The blue point cloud is the original input, and the red part is the segmented point cloud.

cannot be used in the point cloud alignment step.

The SAC segmentation algorithm provides different segmentation models. The lamp consists of a cuboid, cylinder, and hemisphere. Based on its components, we use the cylinder segmentation model to subdivide the lamp point cloud. Fig.5.6 shows the segmentation results of the lamp by using a cylinder model-based SAC segmentation. The blue part in the figure is the original point cloud, the red part is the segmented plane, and the green part is the segmented cylinder. The results show that for the same object lamp, this algorithm divides the lampshade as the cylinder in *source*2 and *source*4 point clouds and divides the part of the bracket as the cylinder in *target* and *source*3 point clouds. Therefore, the segmentation results of the lamp are inconsistent, and thus this model cannot correctly segment the lamp point cloud.

The fourth column in Fig.5.5 shows the region growing segmentation results of the lamp. These figures show that the region growing segmentation method did not successfully segment the bracket part, and different input point clouds will be segmented into different numbers of clusters. Therefore, this method cannot provide effective results for subsequent part



(a) Target (b) Source2 (c) Source3 (d) Source4

Figure 5.6: SAC-based Cylinder Model Segmentation results of the Lamp

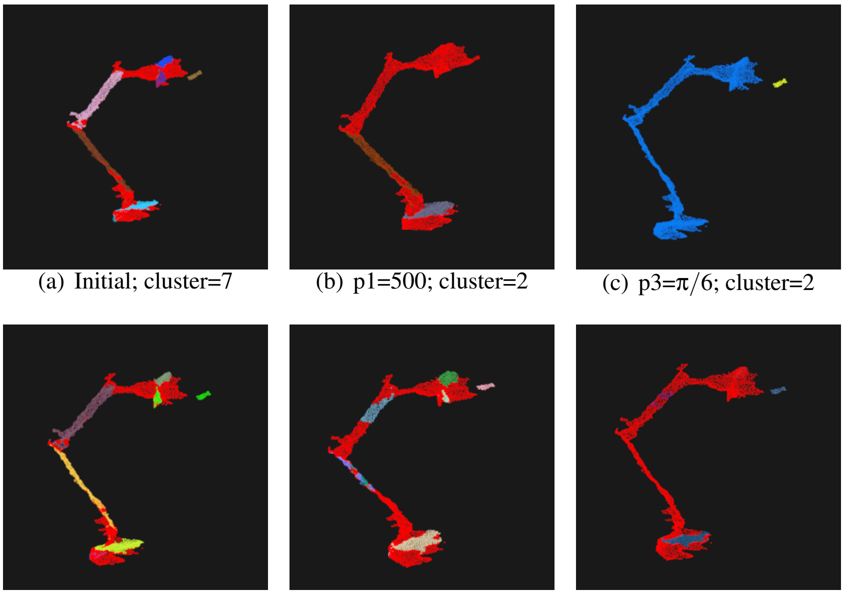
comparison and selection. Moreover, the region growing approach is dependent on the input parameters. Based on the normal and curvature of the point cloud surface, region growing segmentation method combines the points satisfying the smooth constraint (see below) into one cluster and divides the input point cloud into different parts. There are several parameters in this method that affect the segmentation result. Fig.5.7 shows the effect of different parameters on the segmentation results. The parameters tested in the experiment include: *MinClusterSize*(p1), *CurvatureThreshold*(p2) and *SmoothnessThreshold*(p3). The results in (a) are tested under the following parameters:

- $MinClusterSize(p1) = 50$;
- $CurvatureThreshold(p2) = 1.0$;
- $SmoothnessThreshold(p3) = \pi/60$;

In the rest of the images, we change the value of one parameter, the other two parameters remain the same. The caption of each image represents the value of the changed parameter.

MinClusterSize is the minimum size of each cluster. This term limits the minimum number of points included in each cluster. Case (b) increases the value of the *MinClusterSize* compared to (a), which means that the number of points contained in each cluster increases, and the clusters with less than 500 points in case (a) are discarded in (b). Therefore, the number of clusters of the case (b) is reduced as compared with (a).

CurvatureThreshold sets the curvature difference threshold of the points within the same cluster. If the curvature difference between the two points is less than the threshold, these two points belong to the same cluster. From cases (d) to (f), the threshold of curvature gradually decreases, which means that only points with smaller curvature differences can be included in the same cluster. Compared with case (d), the number of points included in each cluster in case (e) is decreasing because the conditions of curvature are more strict, and points with bigger curvature differences are discarded. The number of clusters in case (f) is reduced because the curvature threshold is decreased, and some points having big curvature differences do not satisfy the curvature condition to form a cluster.



(a) Initial; cluster=7

(b) p1=500; cluster=2

(c) p3=$\pi/6$; cluster=2

(d) p2=0.1; cluster=7

(e) p2=0.01; cluster=7

(f) p2=0.005; cluster=4

Figure 5.7: Region growing segmentation results of lamp by using different values of parameters. The parameter p1 represents *MinClusterSize*, p2 represents *CurvatureThreshold*, p3 represents *SmoothnessThreshold*, cluster represents the number of clusters in point cloud.

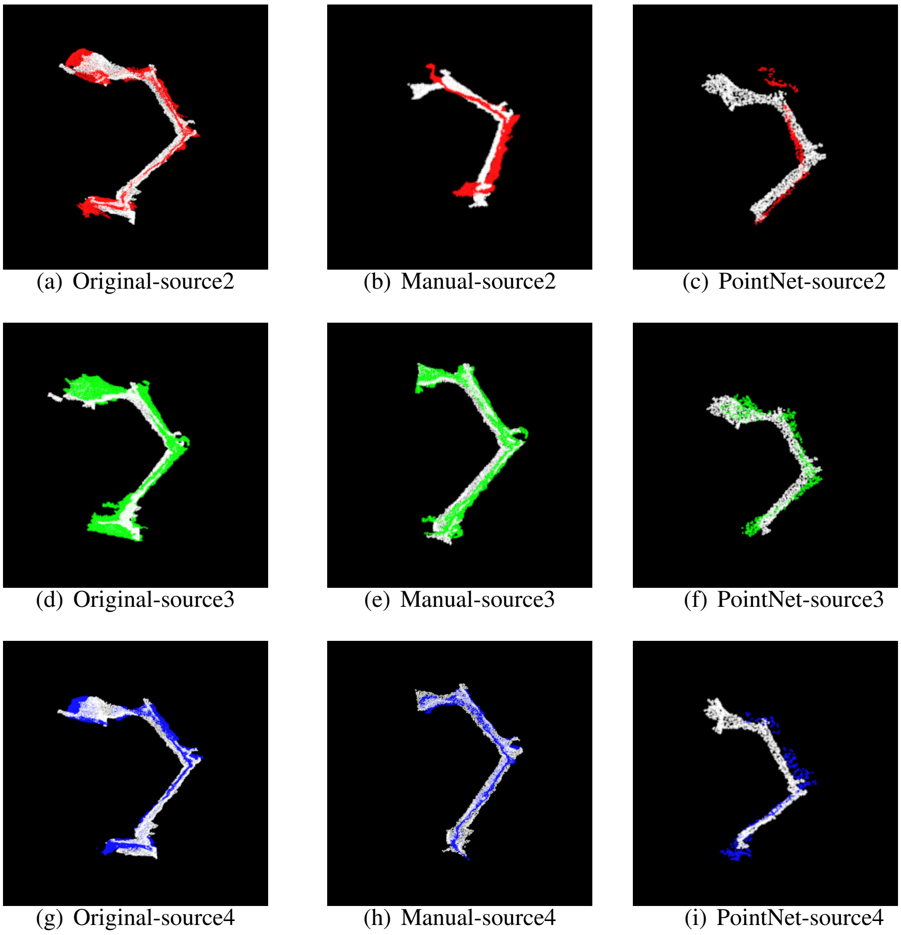
SmoothnessThreshold sets the maximum value of the normal angle difference between two points within the same cluster. Compared with case (a), the *SmoothnessThreshold* in case (c) becomes larger, which means that some points in case (a) are segmented into different clusters because the normal angles differ significantly. In case (c), with the larger threshold, the cluster expands and adds new points. Therefore, the number of clusters in case (c) is reduced, the number of points within the same cluster is increased, and the difference in the normal angle between the points within the same cluster becomes larger.

Based on the above analysis, we can conclude that the region growing method relies on initial parameters. In practice, different point clouds and objects require various parameters to achieve ideal segmentation, which is a complicated and time-consuming operation process. Therefore, this method cannot be used in our project.

The PointNet++ method divide the lamp point cloud into three parts: the blue part is the lampshade, the pink part is the lamp bracket, and the green part is the lamp holder. For each point cloud shown in the figure, the completeness and accuracy of the segmentation results compared to manual segmentation are not good. Each part is labelled with a color value, which is useful in the part selection and ICP registration step.

Point Cloud Registration - Lamp

Fig.5.8 shows the point cloud registration results of the original point cloud, manual segmentation and PointNet++ segmentation point cloud. From the figure we can conclude that for the lamp point cloud, the original point cloud has a good alignment result, the point clouds in the original three sets of registration results can be ideally matched. Unlike the chair object, due to the incomplete handle point clouds, the registration result of a chair has significant errors. The captured lamp point cloud does not contain symmetrical parts like the armrest and the edge of a chair seat and the lamp point clouds are relatively complete. Therefore, we cannot compare matching results based on qualitative analysis, the quantitative analysis is needed for evaluation.



(a) Original-source2 (b) Manual-source2 (c) PointNet-source2

(d) Original-source3 (e) Manual-source3 (f) PointNet-source3

(g) Original-source4 (h) Manual-source4 (i) PointNet-source4

Figure 5.8: Lamp Point Cloud registration Results. The white point clouds represent the *target* ones, the red point clouds represent the point clouds captured from *source* camera2, the green represents *source* camera3 and the blue ones represent *source* camera4. From left to right of each row shows the registration results by using the original point cloud, manual segmentation and PointNet++ segmentation point clouds.

5.1.3 Calibration Object - Mug

In this project, we also validate the pipeline with a mug as a calibration object. In general, a mug consists of two parts: mug body and mug handle. The handle is an irregular shape and the body is a uniform rotation invariant cylinder. In practice, depending on the placement of the mug, the composition of the captured point cloud can be divided into two cases. The first is that all cameras can capture the mug body and the mug handle, in which case the point cloud can be aligned based on the handle, Another case is that there is one camera that can only capture the mug body, and due to its rotation invariance the uniform cylinder cannot provide spatial information. Therefore, in the point cloud registration process, the mug handle can provide valuable spatial information and 3D space features. In the manual segmentation, we divide the mug handle as the target segmented part. Fig.5.9 shows the segmentation results of the mug by using various segmentation methods.

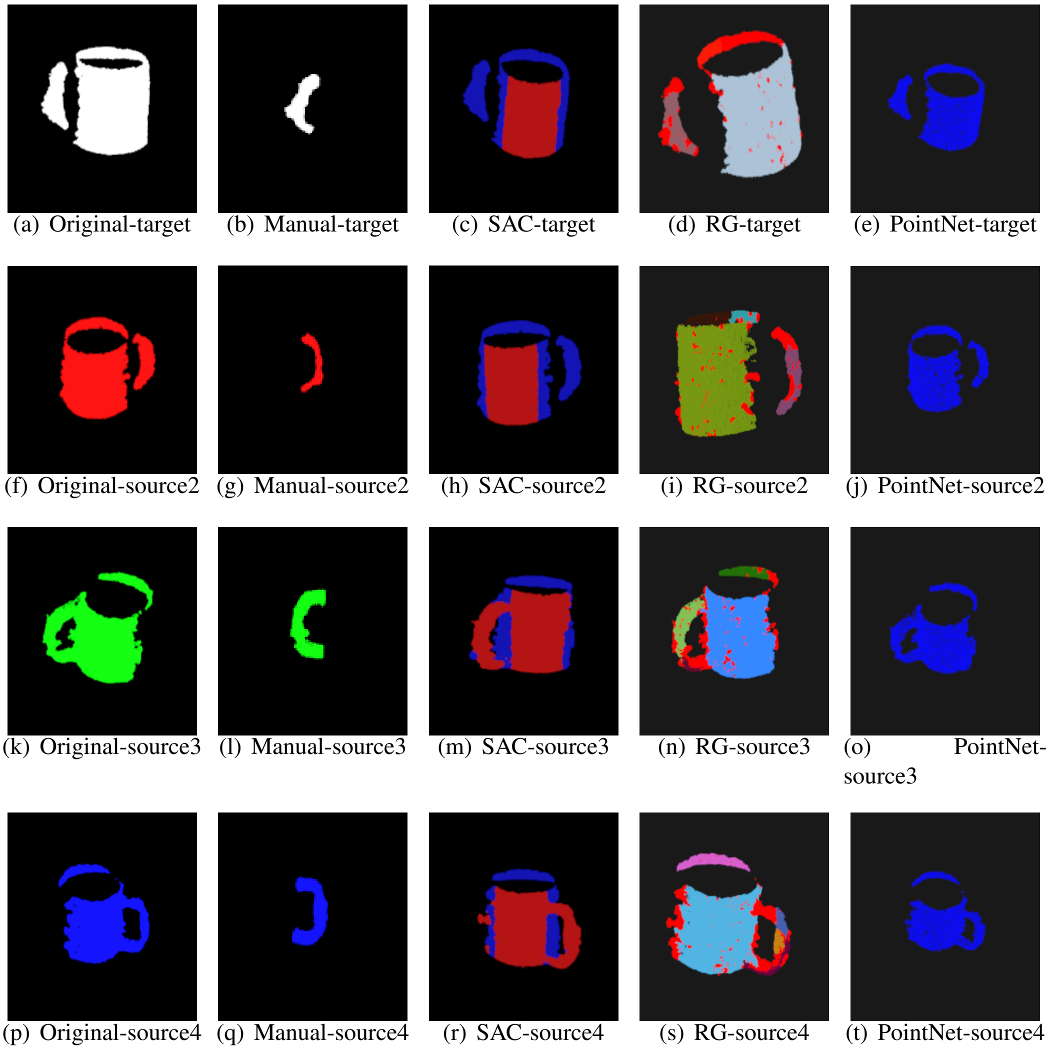
From the SAC segmentation results, we can see that the algorithm recognizes a plane (red part) in the mug body. The red part is a part of mug body or combination of mug body and handle, so the segmentation results are invalid.

According to the 3D shape of the mug, we use the cylinder model-based SAC segmentation method to segment the mug point cloud. Fig.5.10 shows the segmentation results under different $DistanceThreshold$. The blue part in figures is the segmented cylinder, and the green point cloud is the rest. $DistanceThreshold$ is the angular distance between the point normal and the dir vector, dir is the point projection on the cylinder axis. The larger threshold value represents that points with a large $distance$ are also assigned to be part of a cylinder. Compared with case (e), more points in the case (m) are assigned to a cylinder. However, for case (o) and (p), because the threshold is set too large, the algorithm incorrectly divides the point cloud of the handle portion into the cylinder model. Although the algorithm effectively splits the mug handle in some cases, the algorithm is not valid for all inputs due to occlusion and noise problems.

In addition to the $DistanceThreshold$, we also test the effect of different cylindrical radius $RadiusLimits$ on the segmentation results. $RadiusLimits$ sets the radius of the cylinder model. When the radius is reduced like in case (c) and (f), the radius of the segmented cylinder is too small to cover the body of the mug, so the algorithm fails. Therefore, if we want to get a valid segmentation result, the radius should not be too small.

The region growing segmentation method divides most of the mug handle into the same cluster in $target$, $source2$ and $source3$ point clouds; but for $source4$, the handle of the mug contains 3 clusters, so this algorithm is ineffective for the mug object. Moreover, different point clouds are subdivided into a different number of parts, and these parts are not marked. We cannot find the part representing the handle from these point clouds, so we conclude that this algorithm cannot be applied in our pipeline.

The PointNet++ method did not subdivide the mug into two parts, the output point cloud contains only one cluster. For $target$ and $source3$ the handle is separated from the mug body, the mug body point clouds in $source2$ and $source4$ are discontinuous and the complete mug shape is not included in all captured point clouds, we conclude that the algorithm fails for


(a) Original-target (b) Manual-target (c) SAC-target (d) RG-target (e) PointNet-target

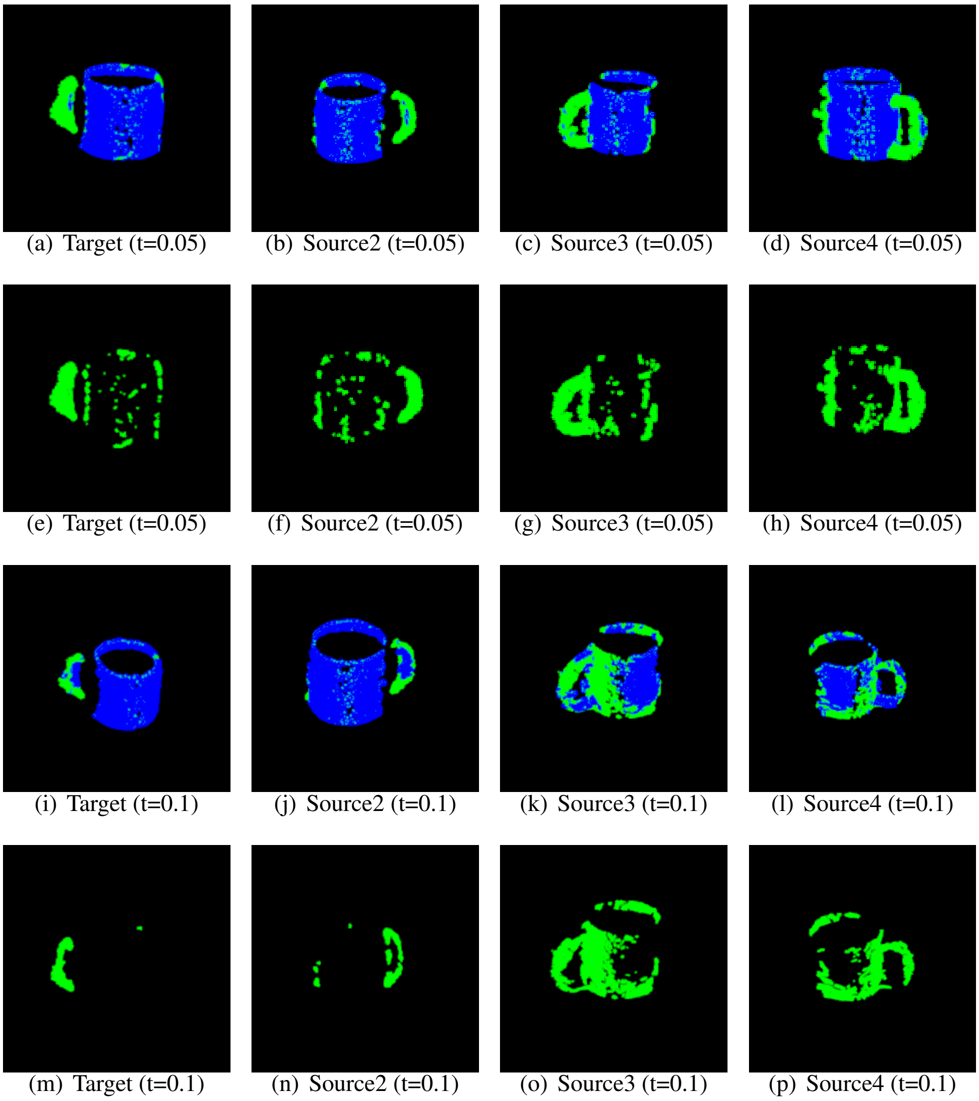
(f) Original-source2 (g) Manual-source2 (h) SAC-source2 (i) RG-source2 (j) PointNet-source2

(k) Original-source3 (l) Manual-source3 (m) SAC-source3 (n) RG-source3 (o) PointNet-source3

(p) Original-source4 (q) Manual-source4 (r) SAC-source4 (s) RG-source4 (t) PointNet-source4

Figure 5.9: Mug Point Cloud Segmentation Results. The point clouds in each column from left to right are original, manual segmentation, plane model-based SAC segmentation, region growing, and PointNet++ segmentation results.

the mug object.



(a) Target (t=0.05) (b) Source2 (t=0.05) (c) Source3 (t=0.05) (d) Source4 (t=0.05)

(e) Target (t=0.05) (f) Source2 (t=0.05) (g) Source3 (t=0.05) (h) Source4 (t=0.05)

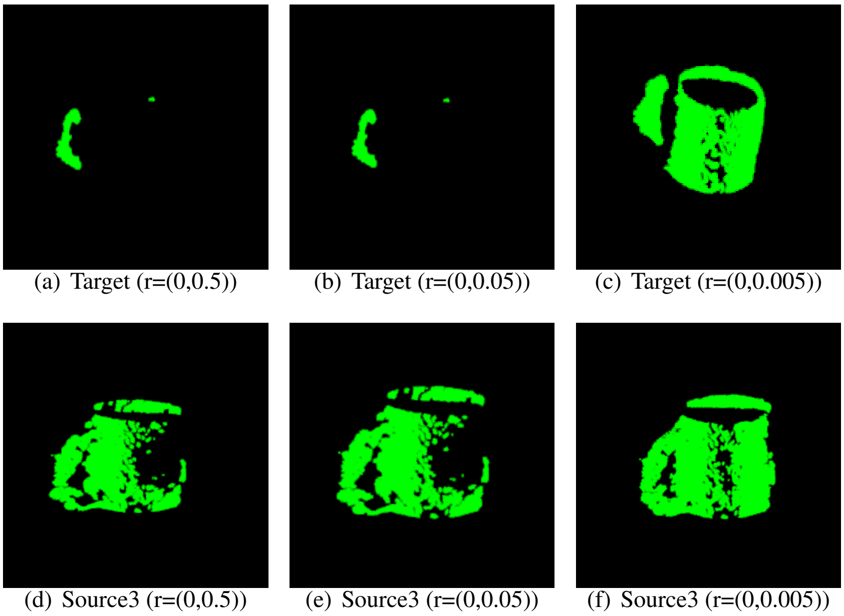
(i) Target (t=0.1) (j) Source2 (t=0.1) (k) Source3 (t=0.1) (l) Source4 (t=0.1)

(m) Target (t=0.1) (n) Source2 (t=0.1) (o) Source3 (t=0.1) (p) Source4 (t=0.1)

Figure 5.10: SAC segmentation results of mug when t = 0.05 and 0.1. The parameter t represents the $DistanceThreshold$. Target and Source represent different capture cameras. The blue point cloud is the segmented cylinder points, and the green part contains other points.

5.1.4 Calibration Object - Earphone

The point cloud data we will capture and reconstruct is human body data. In this experiment, we test with an object in the dataset and combine it with the human body for point cloud registration. Fig.5.12 shows a point cloud with a person wearing headphone. From this angle, only one side of the earphone can be observed, and the headphone in the figure is not easy to recognize, the PointNet++ network cannot divide the earphone. Therefore, our



(a) Target (r=(0,0.5)) (b) Target (r=(0,0.05)) (c) Target (r=(0,0.005))

(d) Source3 (r=(0,0.5)) (e) Source3 (r=(0,0.05)) (f) Source3 (r=(0,0.005))

Figure 5.11: SAC cylinder model segmentation results of a mug by using different values of *RadiusLimits* parameter. Target and Source represents different capture cameras.

approach fails in this situation.

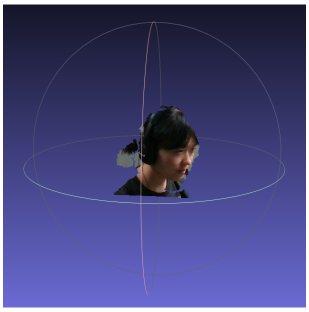
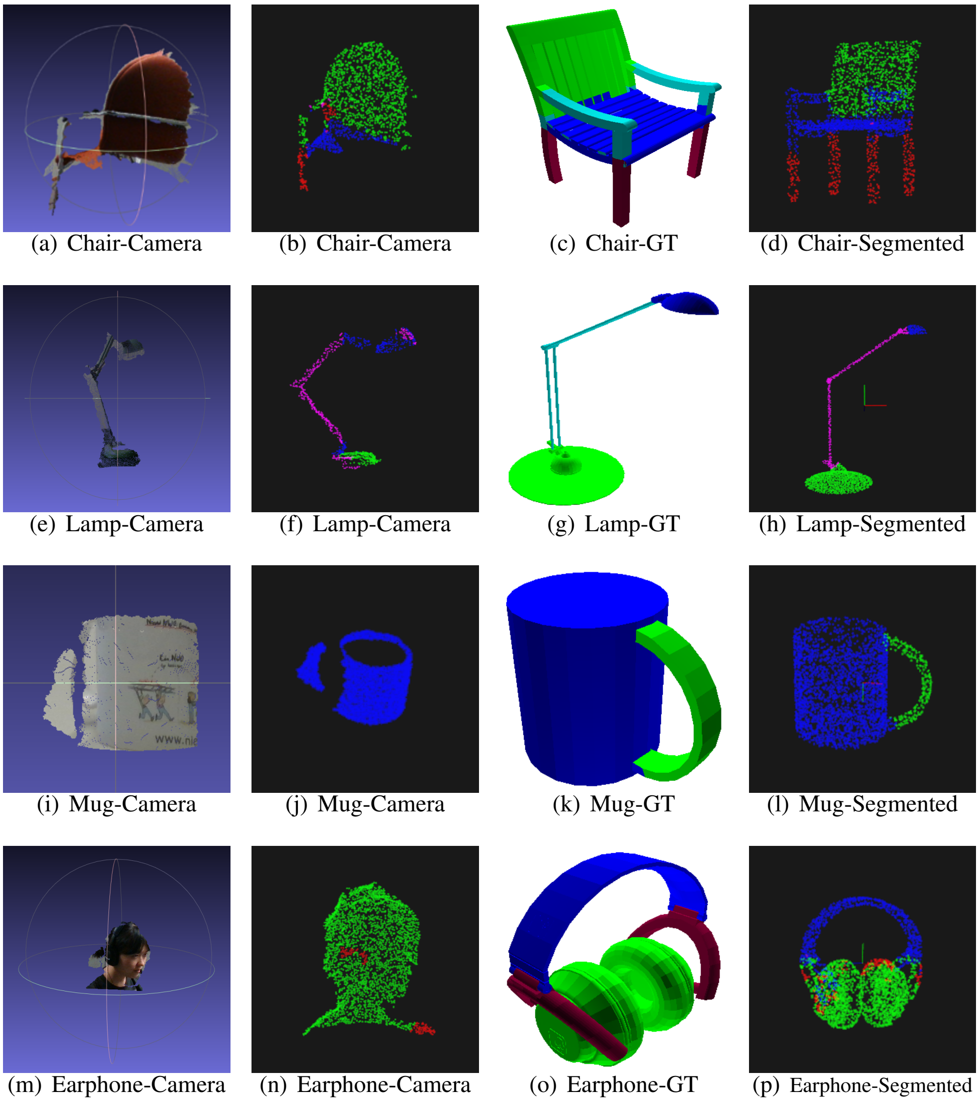


Figure 5.12: The original point cloud of a person wearing a earphone

5.1.5 Analysis of PointNet++ segmentation results

From the previous results, we can conclude that the PointNet++ network effectively subdivides the chair and the lamp. It reaches relatively good results but the accuracy and completeness of the segmentation result are not sufficiently good in some cases. For instance, for the human body wearing an earphone and the mug object, the PointNet++ network cannot successfully segment the object.

In the experiments, we use the complete, clean and labelled 3D point cloud models in ShapeNetCore[104] to train the model, and use the incomplete and noise-containing point cloud captured by the camera as the testing data. The different data distribution and insufficient training dataset are the reasons why the PointNet++ is not effective enough. Fig.5.13 shows the segmentation results by using point clouds in ShapeNeteCore and point clouds captured by the camera.


(a) Chair-Camera (b) Chair-Camera (c) Chair-GT (d) Chair-Segmented

(e) Lamp-Camera (f) Lamp-Camera (g) Lamp-GT (h) Lamp-Segmented

(i) Mug-Camera (j) Mug-Camera (k) Mug-GT (l) Mug-Segmented

(m) Earphone-Camera (n) Earphone-Camera (o) Earphone-GT (p) Earphone-Segmented

Figure 5.13: PointNet++ segmentation results by using different testing data. The first column is the original point cloud captured by the camera, and the second column is the segmentation result of the camera captured point cloud. The third column is the ground truth (GT) of the complete 3D model, and the fourth column is the segmentation result of the 3D model.

The segmentation results shown in Fig.5.13 indicate that PointNet++ fails for some objects captured by the RealSense cameras. For the test data in ShapeNetCore, PointNet++ effectively subdivides all point clouds. Therefore, in our project, the limitations of PointNet++ are due to the different data distribution of the training and the testing sets and the insufficient diversity of the training sets.

- **Data distribution:** The data distribution of the example used in the training and the sample used in the testing is inconsistent, which leads to the problem of domain adaptation[29][63]. Due to the limitation of training data, We train the model in a specific source domain, but we need to deploy our model to one or several different target domains. The segmentation model which used complete clean point cloud as training data cannot be ideally adapted to segment the incomplete, noisy point clouds.
- **Insufficient training data:** Since the captured point cloud is affected by noise and incompleteness, the surface of the point cloud is not smooth, but the training point clouds are clean 3D models. Therefore, the training data used in this project is insufficient. This limitation can be enhanced by adding noise and sampling (leading to incompleteness) in the training point clouds through data augmentation[27].

5.2 Quantitative Analysis

In the quantitative analysis, we used three metrics to evaluate the results of point cloud registration, including processing time, number of iterations and fitness score.

Real-time performance is an essential feature in dynamic 3D reconstruction system. The multi-camera registration pipeline in this paper provides camera position information for real-time 3D reconstruction, and the processing time is an important metric. The camera calibration provides important registration information for the accuracy of subsequent 3D reconstructions. Therefore, accuracy is an important indicator of pipeline performance. The ICP algorithm iteratively calculates and matches the point cloud, and the number of iterations affects the running time and accuracy.

- **Processing time**: This value records the duration of the point cloud registration. For the alignment of the original point clouds, this value includes the running time of the ICP algorithm for a certain number of iterations. For point cloud registration based on SAC segmentation, the processing time includes SAC point cloud segmentation and ICP algorithm run-time. For the deep learning based point cloud registration method, the processing time includes point cloud segmentation, part selection and ICP algorithm running time.

- **Fitness score**: This value determines the point cloud registration accuracy by calculating the sum of the squared Euclidean distance[5] of the *target* point cloud and the aligned *source* point cloud. A smaller fitness value represents a higher registration accuracy. This fitness score is obtained by using the *getFitnessScore* function in the Point Cloud Library.

- **Number of iterations**: This value is the number of iterations of the ICP algorithm. The ICP algorithm iteratively aligns point clouds until convergence. In order to compare the performance of different methods in our experiments, we set the same number of iterations in our experiments to be able to compare the processing time and fitness score of different methods.

5.2.1 Calibration Object - Chair

Table5.1 shows the quantitative evaluation and comparison of the original chair point cloud, manually segmentation, SAC segmentation and PointNet++ segmentation point cloud registration.

Fig.5.14 shows the polyline chart of processing time and fitness score. From the table and figures, we can observe: the similar to the conclusion of the qualitative analysis, the result of manual segmentation and the selection of the chair back for point cloud registration shows the highest accuracy. Compared with other methods, manual segmentation achieves a smaller fitness score with the same number of iterations, which means that manual segmentation can achieve the most accurate registration results. This result proves that the feature-based point cloud registration can effectively improve the accuracy of the alignment

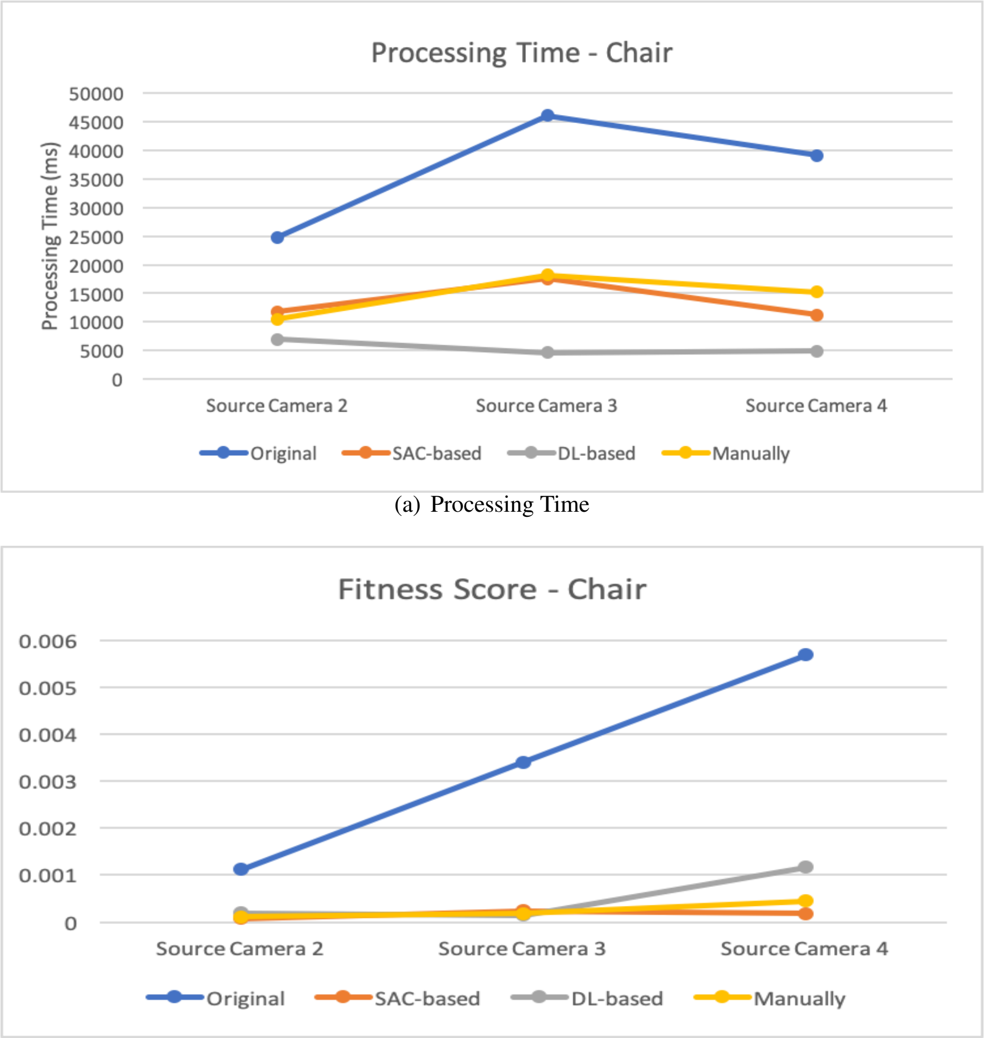
Table 5.1: Quantitative evaluation and comparison of different point cloud registration results - Chair Point Cloud (DL represents deep learning network PointNet++)

	Point Cloud	Processing Time	Number of iterations	Fitness Score
Source Camera2	Original	24782ms	40	1.112e-3
	SAC-based	11764ms	40	7.325e-5
	DL-based	6939ms	40	1.875e-4
	Manually	10524ms	40	1.075e-4
Source Camera3	Original	46031ms	40	3.397e-3
	SAC-based	17501ms	40	2.386e-4
	DL-based	4654ms	40	1.507e-4
	Manually	18192ms	40	1.791e-4
Source Camera4	Original	39096ms	40	5.687e-3
	SAC-based	11195ms	40	1.802e-4
	DL-based	4885ms	40	1.167e-3
	Manually	15226ms	40	4.518e-4

results. However, the manual segmentation method requires a long running time and human intervention, which cannot be applied in a real-time camera pose estimation and 3D reconstruction system.

For the automatic point cloud segmentation and registration pipelines, having the same number of iterations, the point cloud part segmented by the SAC segmentation method has better performance in accuracy than PointNet++, In summary, the automatic segmentation method has higher accuracy than the original point cloud, and it requires less processing time having the same number of iterations of the ICP algorithm. It demonstrates that the above methods can effectively improve the accuracy of point cloud registration. Among the four mentioned methods, the deep learning method has the shortest running time. Compared to the original point cloud, the PointNet++ segmented point cloud reduces the processing time by four-fifths during the point cloud registration process.

For the same number of iterations, the processing time depends on the number of points in the point cloud. Point cloud segmentation and filtering can reduce the number of points. In the experiment, the feature-based point cloud registration method reduces the number of points by doing the alignment based on the chair's back only. In all the methods mentioned earlier, we use filters to reduce the number of points. Among them, the PointNet++ method performs down sampling twice which results in the lowest number of points for alignment,



(a) Processing Time

(b) Fitness Score

Figure 5.14: Polyline Chart of Processing Time and Fitness Score - Chair Point Clouds

so the processing time is the shortest.

The fitness score is determined by the Euclidean distance difference between the aligned point clouds. Compared with the original point cloud, the feature-based point cloud registration method improves the accuracy, because the point cloud segmentation process removes the part of the chair that makes the ICP algorithm ignore the global 3D shape and fall into the local optimum.

In conclusion, feature-based point cloud registration and camera calibration improve the accuracy of alignment and shorten the processing time compared to the original process. In the experiment by using the chair's back to calibrate the cameras, the deep learning pipeline requires the shortest running time and gets a better accuracy than the original pipeline.

However, deep learning algorithms are limited in producing the *best* matching object components. It is not able to completely and correctly segment the point cloud for the incomplete captured point cloud. The data distribution and insufficient training data discussed in Section 5.1.5 is the reason why the accuracy of the deep learning algorithm is lower than the *ideal* situation.

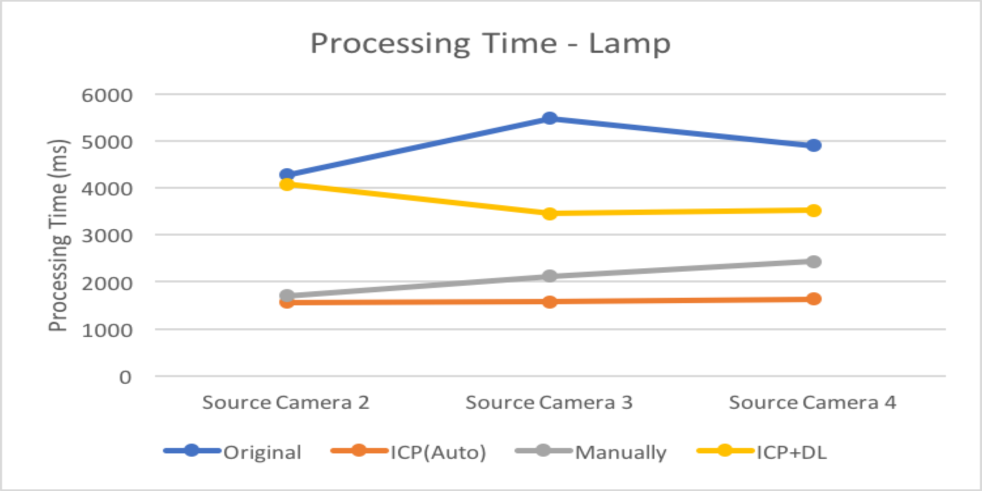
5.2.2 Calibration Object - Lamp

Table 5.2 shows the quantitative evaluation of the original lamp point cloud, manual segmentation point cloud and PointNet++ segmentation point cloud registration results.

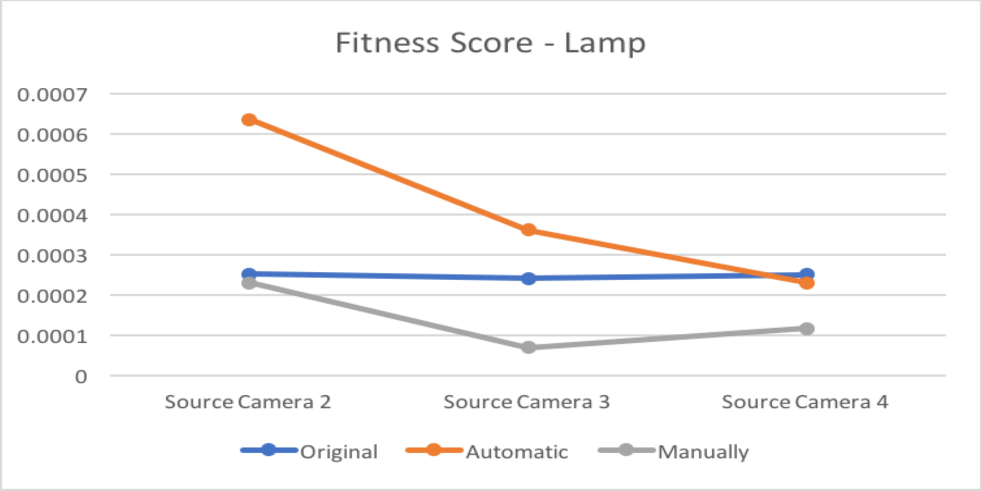
Table 5.2: Quantitative evaluation and comparison of different point cloud registration results - Lamp Point Cloud(DL represents deep learning network PointNet++)

	Point Cloud	Processing Time	Number of iteration	Fitness Score
Source Camera2	Original	4272ms	40	2.509e-4
	Automatic	1567ms	40	6.342e-4
	Manually	1703ms	40	2.288e-4
Source Camera3	Original	5484ms	40	2.403e-4
	Automatic	1577ms	40	3.602e-4
	Manually	2120ms	40	6.859e-5
Source Camera4	Original	4897ms	40	2.500e-4
	Automatic	1637ms	40	2.800e-4
	Manually	2430ms	40	1.165e-4

Fig.5.15 shows the processing time and fitness score of point cloud registration for different input point clouds. For the lamp, the original point clouds based method has good accuracy, and the manual feature-based method improves the registration accuracy somewhat further. All metrics are compared based on the same number of iterations. The processing time shown is the total time, so the ICP(Auto) time includes part selection and ICP running time, and the ICP&DL includes the PointNet++ segmentation running time and ICP algorithm running time. If we only count the processing time of the ICP algorithm, the required time for the automatically segmented point cloud is the shortest of all methods. If we add the running time of the PointNet++ segmentation process, the time for automatic segmentation and registration are still shorter than the time needed for the original point cloud registration. Compared with the original point cloud, the automatic segmentation and registration method shortens the processing time (see Section 5.2.1), but the automatic segmentation and registration method did not improve the performance in accuracy. In conclusion, feature-



(a) Processing Time



(b) Fitness Score

Figure 5.15: Polyline Chart of Processing Time and Fitness Score - Lamp Point Clouds

based registration improves the registration accuracy and shortens the processing time. In our experiment, the automatic segmentation method did not obtain the *ideal* segmentation results can be obtained by manual segmentation, which influences the accuracy of point cloud registration.

5.2.3 Feature-based Coarse Registration

From the above analysis, we can conclude that feature-based point cloud segmentation and registration method can effectively improve the accuracy and shorten the processing time. However, the coarse registration step in our pipeline still requires a checkerboard as a calibration object and a complicated calibration process to get the initial transformation matrix. Although the camera position is fixed in the system, once the camera position changes, this step needs to be re-run. Therefore, we use the FPFH and SAC-IA algorithm instead of the MATLAB toolkit for coarse registration.

The input of the MATLAB toolkit is a set of RGB images of a pair of cameras that have a view from different angles to the same calibration object. The transformation matrix is calculated by Zhang's calibration algorithm[127]. The FPFH method directly processes point cloud data (PLY file) to calculate the matrix. Then the calculated transformation matrix is used as the initial matrix in the ICP algorithm for point cloud registration, the quality of this affects the performance of the ICP algorithm.

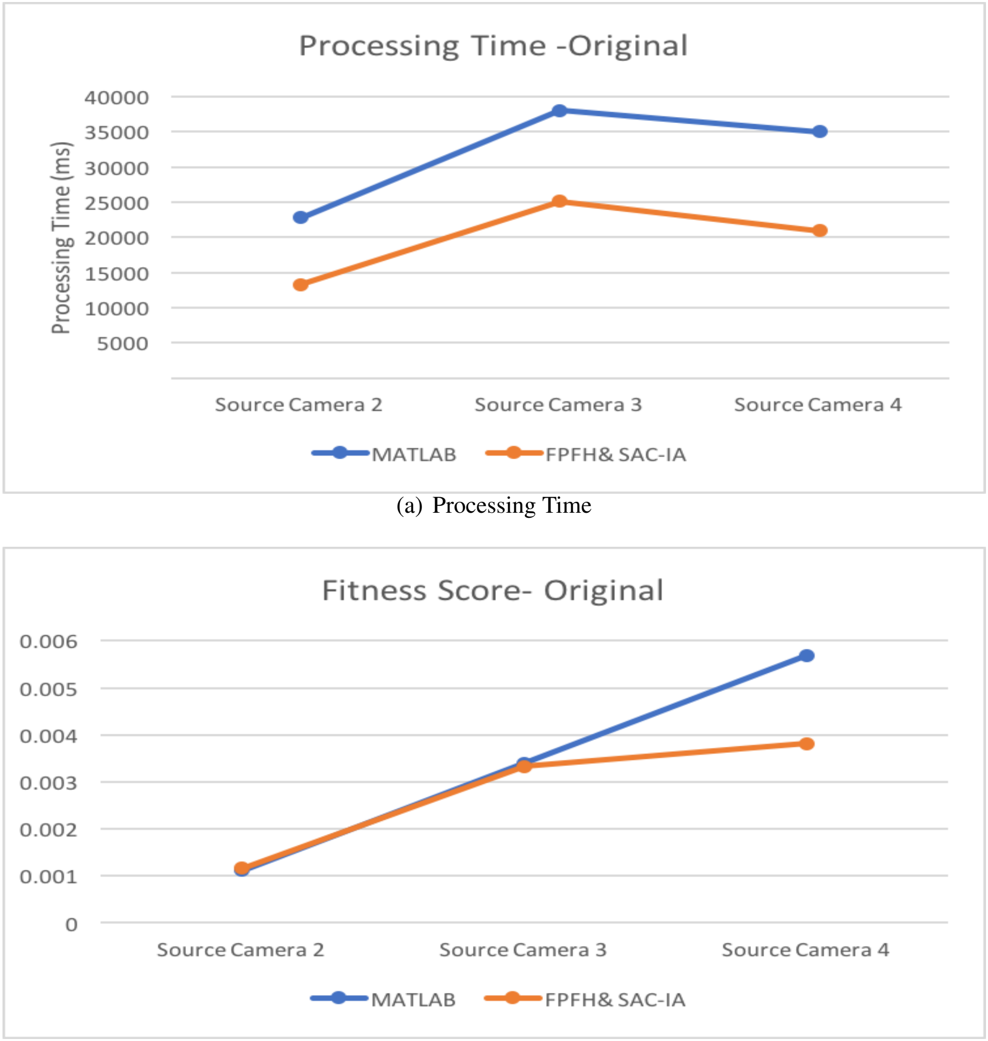
In order to compare the performance of these two coarse registration methods, we input the calculated transformation matrix into the ICP program and compare the processing time and fitness score by running the same number of iterations. Thereforethe algorithm with shorter processing time and smaller fitness score performs better. We input the matrix into the original point cloud and feature-based point cloud registration program to compare the performance of the two types of coarse registration. Table5.3 shows the quantitative evaluation of the coarse registration methods. Fig.5.16 and Fig.5.17 shows the difference between the MATLAB and FPFH&SAC-IA methods when using original point clouds and segmented point clouds.

According to the table and polyline charts, we can observe that for both the original point cloud and the segmented point cloud, the two coarse registration methods have comparable performance. In the comparison based on the original point cloud, it can be seen that compared with the MATLAB toolkit method, the transformation matrix obtained by FPFH&SAC-IA method results in shorter processing time and slightly higher accuracy. In the comparison of the segmented point cloud, the two methods have a similar performance. Therefore, we can conclude that based on the impact of the initial transformation matrix on the performance of the ICP algorithm, the two methods have comparable effects and performance in the coarse registration procedure.

Of these two methods, the MATLAB toolkit requires a calibration object and complicated operations, the process involves manual manipulation such as taking RGB pictures, changing the angle of the checkerboard, selecting corners in pictures and entering the commands in MATLAB. These operations take around an hour in the MATLAB coarse registration process. In contrast with this, the FPFH&SAC-IA method only needs a PLY file and the calculation takes about 150s, which only takes twenty fourth of the time required by the MATLAB toolkit. As the resulting matrixes are of comparable quality, we conclude to make use of the FPFH&SAC-IA method in the coarse registration step.

Table 5.3: Evaluation and comparison of ICP algorithm by using different inital transformation matrixes on original and PointNet++ segmented Chair point cloud

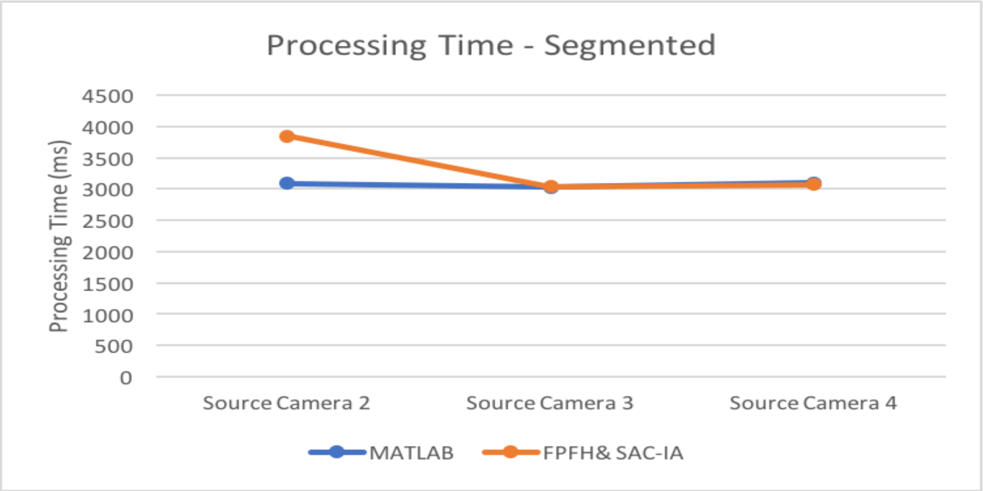
	Transformation Matrix	Processing Time	Number of Iterations	Fitness Score
Original(c2)	MATLAB	22805ms	40	1.112e-3
	FPFH& SAC-IA	13294ms	40	1.159e-3
Original(c3)	MATLAB	37980ms	40	3.397e-3
	FPFH& SAC-IA	25094ms	40	3.332e-3
Original(c4)	MATLAB	34972ms	40	5.687e-3
	FPFH& SAC-IA	20959ms	40	3.811e-3
Segmented(c2)	MATLAB	3090ms	40	1.874e-4
	FPFH& SAC-IA	3844ms	40	4.411e-4
Segmented(c3)	MATLAB	3032ms	40	1.508e-4
	FPFH& SAC-IA	3041ms	40	1.409e-4
Segmented(c4)	MATLAB	3098ms	40	1.167e-3
	FPFH& SAC-IA	3074ms	40	5.764e-4



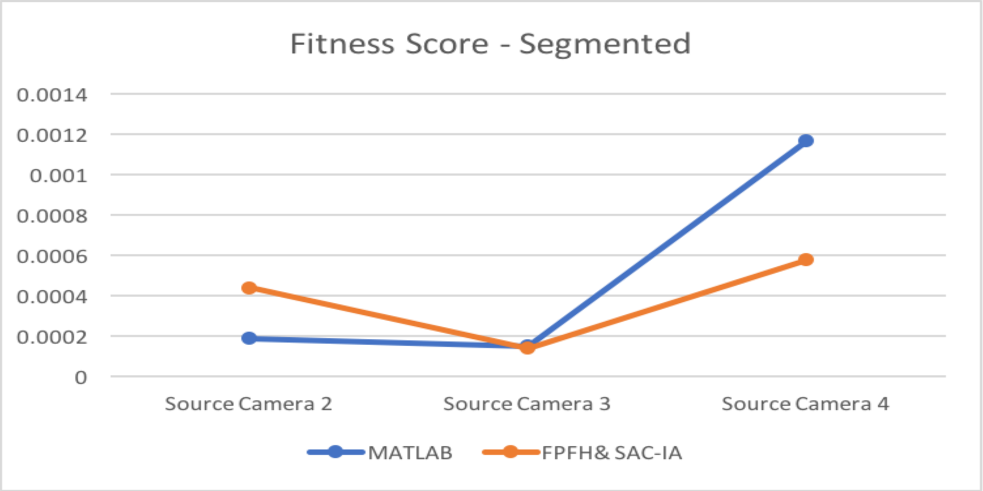
(a) Processing Time

(b) Fitness Score

Figure 5.16: Polyline Chart of Processing Time and Fitness Score - Original Point Clouds



(a) Processing Time



(b) Fitness Score

Figure 5.17: Polyline Chart of Processing Time and Fitness Score - Segmented Point Clouds

5.3 Comparison with State of the Art

In this study, we present a markerless, flexible, feature-based multiple depth cameras calibration pipeline. Next, we will compare the advantages and disadvantages of this pipeline with other state-of-the-art systems from several perspectives.

The calibration of depth cameras can be divided into two types. One type of calibration system[57][72][80][3] relies on additional calibration object (specific marker, checkerboard or boxes with QR code) to calculate camera parameters. These systems require human intervention (such as artificial placement of the calibration object) and other constraints (position of the calibration object, lighting) during the calibration process. Another type of calibration system[56] does not require any additional calibration, which is dynamically calibrated by using human's skeleton and joints information. But such a system requires a specific type of camera to provide human skeleton data. Due to the limitation of the types of data provided by the cameras on the market, this kind of system is not easily generalized. Compared with state-of-the-art, this thesis proposes a flexible multi-camera calibration pipeline. In this section, we compare the flexibility of this system with other systems from the following aspects:

- **Additional Marker:** The need for additional calibration markers is one of the limitations of the camera calibration system. The current methods relying on specific calibration object are complicated and time-consuming. Eliminating the limitations of additional calibration object will simplify the calibration process and shorten the processing time.

- **Operation Complexity:** The complexity of the process is influenced by several factors: additional calibration markers; special requirements for the placement of the calibration object; special requirements for the calibration environment; accessibility of the calibration objects and human intervention (parameters manipulation and marker placement).

- **Supported Sensor:** The supported camera is affected by two factors: the limitations of the 3D data type captured by the camera, and the type of data used by the calibration system. If the calibration system can be extended to various types of cameras, the system can get rid of the limitations of a particular sensor.

Table 5.4: Comparison with State of the Art depth camera calibration systems

	LiveScan3D [57]	OpenPTrack [72]	Human Body Tracking[80]	Live 3D Motion Capturing[3]	Skeleton-based[56]	Thesis Pipeline
Core Technologies	Orthogonal Procrustes [101],ICP	ROS[28], bundle adjustment [10]	Corner detection,skeleton fusion	SIFT[64], Procrustes analysis[55]	Least-squares [117]	Feature-based,ICP
Extra Marker Required	Yes	Yes	Yes	Yes	No	No
Skeletal Tracking	No	No	Yes	No	Yes	No
Calibration Object	Marker[98]	Checker-board	Checker-board	Labelled IKEA boxes	Human Skeleton	Chair,Lamp
Operation Complexity	High	High	High	High	Low	Medium
Human-based Calibration	No	Yes	Yes	Yes	Yes	No

Supported Sensor	Kinect v2	Kinect;
		SR4500;
		Stereo
			Kinect	Kinect	Kinect	Depth sensors with point cloud capturing

Year	2015	2016	2017	2017	2018	2018

Additional Marker: As can be seen from Table 5.4, most calibration systems require additional calibration objects such as markers, checkerboards, and labelled boxes. The human skeleton-based calibration system[56] can get rid of the use of additional calibration object, but it only works for specific depth sensors - Kinect. Currently, our system can use chairs and lamps as calibration objects, and no additional markers are required, the diversity of calibration objects can be extended based on the deep learning training dataset.

Operation Complexity: Compared with many systems, the operation of our pipeline is more simple. For example, systems[57][72][80][3] require additional markers for calibration, which may put limits on the placement of objects, the operation process and the fixed camera position. In the OpenPTrack system, after all cameras have captured the calibration object, the calibration object needs to be manually removed in the next step. In the human body tracking system[80], the calibration object needs to be fixed on the ground, and the process needs to ensure that the calibration object is complete and not occluded. Therefore, there is a limit of human movement in the working space. In the live 3D human tracking system[3], the labelled IKEA boxes require to be placed in a specific order to be able to calibrate the system calibrates the cameras based on a QR code on boxes. The pipeline proposed in this thesis does not require an additional calibration marker. The cameras can be calibrated using different calibration objects without artificially changing parameters. By using a feature-based approach, our pipeline improves the accuracy of the point cloud registration and reduces the running time.

Supported Sensor: Most of the systems in the table, except OpenPTrack[72], only support the Kinect series depth cameras. Such systems rely on the software tools that go with the sensor and cannot be easily used for other types of depth cameras, as the production of the Kinect has stopped the future of these methods may be limited. Our pipeline can be used in any type of multiple camera systems, and the only requirement is that the depth camera needs to be able to capture the point cloud data, and most of the depth cameras on the market fulfill this requirement. Therefore, this pipeline can be generalized into more types of cameras, and it is more flexible in practical applications.

However, our pipeline is not the most convenient system. Compared with the skeleton-based system[56], our pipeline cannot be directly calibrated based on a human body. For our method, a static calibration object is needed in the scene. The reason is that we use a deep learning network for feature-based point cloud segmentation and registration. As at this moment there is no dataset of labelled human point cloud, we were not able to handle a human-based point cloud as the calibration object to perform the multi-camera calibration.

Chapter 6

Conclusions

6.1 Summary

In this thesis we studied on multiple depth cameras calibration and pose estimation for real-time 3D reconstruction. Most of the existing calibration systems[57][80][3] require complex calibration objects such as labelled boxes and checkerboards, and the process of camera calibration is complicated and time-consuming. Some systems also require manual placement of calibration objects and have limitations on camera position and human movement in the scene. The skeleton-based calibration system[56] provided an alternative solution to complex calibration objects. Inspired by this idea, we propose a feature-based, markerless and flexible point cloud registration and pose estimation system.

Our pipeline as shown in Fig.6.1 consists of the following stages: pre-processing, coarse registration, fine registration and point cloud update. According to our research on RGB-D camera-based 3D reconstruction system in recent years, we choose the ICP algorithm for point cloud registration in the fine registration step. The advantage of the ICP algorithm in achieving continuous and re-calibration system is that we can redo the calibration process by setting a threshold to achieve an automatic re-calibration process. Besides, the ICP algorithm requires an initial transformation matrix. In order to provide a more accurate coarse transformation matrix, we use the MATLAB toolkit to calibrate four cameras in the coarse registration step.

The research questions we deal with in this thesis are: how to deal with the error of the ICP algorithm caused by the incomplete parts of the captured point clouds; how to get rid of the limitations of extra calibration objects and specific camera types to achieve a user-friendly calibration system; and how to achieve an automatic re-calibration system.

In the experiment of using the chair as the calibration object, we notice that the camera could not capture the complete chair point cloud data due to the limitation of the range and the scope of cameras, and the incomplete point cloud from different cameras will make the ICP algorithm fall into a local optimum and lead to errors, which means that the ICP algorithm tries to match two inconsistent parts and causes errors for the whole object.

In order to solve this problem, we use the point cloud segmentation algorithm to divide the

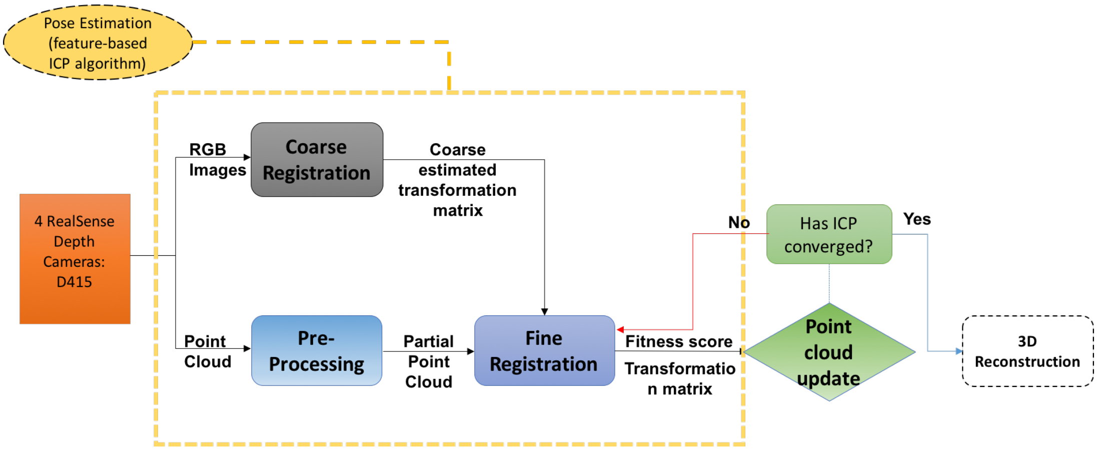


Figure 6.1: The system architecture. The grey module (coarse registration) is a one time process; the blue modules are continuously processing procedures. In this thesis we validate the modules within the yellow dotted box and propose a suggestion (green modules) for achieving an automatic re-calibration system.

point cloud into different parts, and perform ICP registration by using selected appropriate parts which are commonly covered by different cameras. This method reduces registration errors caused by the incomplete part of the point cloud. We first use the SAC segmentation (based on plane and cylinder model) and region growing algorithm in the PCL library to segment the point cloud. For the three calibration objects we test chairs, lamps and mugs, only the plane model-based SAC segmentation is shown to be effective for the chair. However, this calibration method only selects the largest plane in the chair as the segmentation output, we cannot judge whether the selected part is the common and suitable part of all cameras. Therefore, this method does not meet the requirement to be used in our system.

In the next step, we use a deep learning network PoineNet++ to train the model and perform point cloud segmentation. Compared with the previously tested methods, the deep learning network can segment the object according to the components (parts) of the object. We can determine and select the appropriate part for ICP registration by volume comparison of the segmentation result. The deep learning network method is effective for chairs and lamps, it improves the registration accuracy and shortens the processing time.

Experimental results show that the feature-based pipeline can shorten the registration time and improve the registration accuracy. For example, for a chair object (target point cloud captured by camera1, source point cloud captured by camera2), the original point cloud data is affected by an incomplete chair armrest, and the registration result is inaccurate. The entire registration process requires 24782ms for 40 iteration times, and the fitness value is 1.112e-3. The SAC segmentation method requires 11764ms to reach 7.325e-5 fitness score. After using deep learning network method, the whole process needed 6939ms for 40 iterations, and the fitness score is 1.875e-4. In summary, compared with the original point cloud, the use of PointNet++ segmented point cloud effectively shortens the running time and improves the registration accuracy.

For the second problem, We propose a feature-based calibration system that performs point cloud registration by using point cloud data and the ICP algorithm. Point cloud data can be converted from the 3D coordinates captured by the depth camera and is currently available from most depth cameras on the market (Kinect series, Xtion, RealSense series). Also, we replace the MATLAB calibration method with the feature extraction and matching functions provided by the Point Cloud Library.

The MATLAB toolkit used in the coarse registration step in our system is complicated, which requires a calibration checkerboard and involves manual operations. Although this procedure is a one-time procedure, any camera movement in multiple cameras systems force this step to be re-executed. Therefore, we use the feature extraction and alignment method to replace the MATLAB toolbox. This feature-based coarse registration method has comparable performance with the MATLAB toolkit procedure, and it only requires a PLY file as input, which simplifies the procedure and shortens the processing time into twenty-fourth of the other method.

In this thesis, we propose a solution for flexible multi-camera calibration and improve the robustness and performance of the ICP algorithm. For the third research question mentioned in Chapter 1, we make a suggestion to use the convergence and fitness score of ICP algorithm as a threshold to automatic control the re-calibration process, in this way to achieve dynamic multi-camera registration in the future.

6.2 Future Work

Although our pipeline improved the point cloud registration results, our pipeline still has limitations. In this section we make suggestions for future work and further development.

Indistinguishable data: In this thesis, we try to combine a human body with the static object for multi-camera calibration. For example, calibration based on a person wearing an earphone as shown in Fig.6.2. For these point clouds, because the earphone object cannot be recognized by the algorithm, this method is unable to segmentation and registration of this kind of situations. For this type of data, we need to first be able to recognize the static object and separate it from the original point cloud. How to distinguish and recognize the static object is a challenge. One possible solution is that we can divide the object by using color-based or texture-based recognition methods. Based on the recognition result, we can then apply the feature-based registration on a static object.

Human-based calibration: The ultimate goal of our project is to calibrate cameras and reconstruct 3D scenes with human body point cloud data. Due to the limitations of the training dataset, there is currently no point cloud dataset for human-based segmentation. For this pipeline, the appropriate human point cloud data should classify the human data according to the biological composition and assign parts with different colors. For example, the human body can be segmented into a head, upper limb, trunk, lower limb and foot. In the human tracking and motion capturing process, the point cloud can be divided according to the above classification, and the parts such as the upper limb and lower limb can provide spatial position information for matching. After segmentation of the human point cloud, we

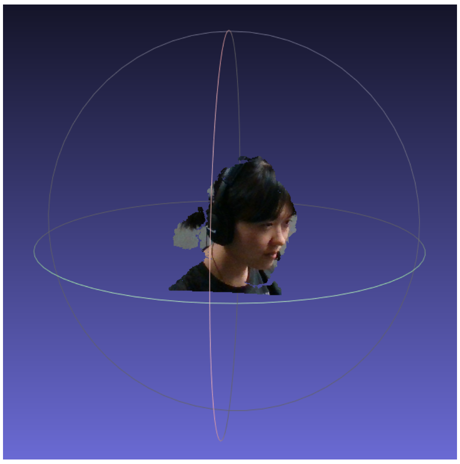
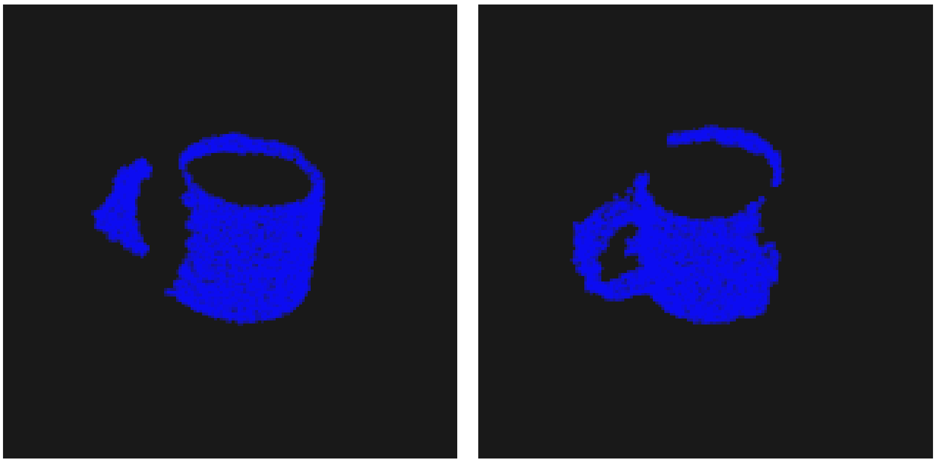


Figure 6.2: The point cloud of a person wearing an earphone

can then do ICP registration based on the chosen parts and align point clouds.

Occlusion and Incomplete point cloud: In this project, we did not find a suitable solution for the mug point cloud. None of the segmentation methods used in our experiments can correctly subdivide the mug into two parts and leave a clean, complete mug handle as the segmentation result. Fig.6.3 shows the mug segmentation results of deep learning network of *source*1 and *source*3. The mug has a uniform body and a handle. For point cloud registration, the handle part is more useful for determining the spatial location. However, the captured point cloud is incomplete due to occlusion problems, so the point cloud cannot be correctly segmented. For figure (a), the handle is separated from mug body, for figure (b), the mug body is divided into two parts due to the discontinuity. How to solve the problem of occlusion and point cloud discontinuity and how to improve the deep learning segmentation accuracy of incomplete input point clouds are still big challenges. One suggestion is to enhance the training data[27] by adding noise and sampling to solve the problem of data distribution.



(a) Camera1

(b) Camera3

Figure 6.3: The PointNet++ segmented mug point cloud.

Common part selection optimization: In the segmented point cloud selection phase, we select the most suitable common part for registration stage based on the volume of the point cloud. This method cannot be applied in all situations and thus is limited in its use. The point cloud data is sparse and unordered, due to the limited amount of geometric information provided by the point clouds, the 3D spatial similarity measurement of different point cloud data is still a challenge.

Bibliography

[1] TAN Yunlan JIA Jinyuan PENG Shuoet al. Survey on some key technologies of virtual tourism system based on Web3D. 2014.

[2] Geoffrey E. Hinton Alex Krizhevsky, Ilya Sutskever. ImageNet Classification with Deep convolutional Neural Networks. Accessed Aug 2018. http://vision.stanford.edu/teaching/cs231b_spring1415/slides/alexnet_tugce_kyunghee.pdf, 2015.

[3] Dimitrios S Alexiadis, Anargyros Chatzitofis, Nikolaos Zioulis, Olga Zoidi, Georgios Louizis, Dimitrios Zarpalas, and Petros Daras. An integrated platform for live 3d human reconstruction and motion capturing. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(4):798–813, 2017.

[4] Xing An, Gang Li, Linwei Xu, and Ying Shi. A survey on application of virtual reality technology in u. s. military simulation training. *Dianguang yu Kongzhi(Electronics, Optics & Control)*, 18(10):42–46, 2011.

[5] Howard Anton. *Elementary Linear Algebra, Binder Ready Version*. John Wiley & Sons, 2013.

[6] Pablo Arbeláez, Jordi Pont-Tuset, Jonathan T Barron, Ferran Marques, and Jitendra Malik. Multiscale combinatorial grouping. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 328–335, 2014.

[7] ASUS. Xtion-2 Depth Camera Introduction. Accessed Aug 2018. https://www.asus.com/3D-Sensor/Xtion-2/overview/, 2017.

[8] Marco Attene. A lightweight approach to repairing digitized polygon meshes. *The visual computer*, 26(11):1393–1406, 2010.

[9] Mathieu Aubry, Ulrich Schlickewei, and Daniel Cremers. The wave kernel signature: A quantum mechanical approach to shape analysis. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 1626–1633. IEEE, 2011.

[10] Filippo Basso, Riccardo Levorato, and Emanuele Menegatti. Online calibration for networks of cameras and depth sensors. In *OMNIVIS: The 12th Workshop on Non-classical Cameras, Camera Networks and Omnidirectional Vision-2014 IEEE International Conference on Robotics and Automation (ICRA 2014)*, 2014.

[11] Paul J Besl and Neil D McKay. Method for registration of 3-d shapes. In *Sensor Fusion IV: Control Paradigms and Data Structures*, volume 1611, pages 586–607. International Society for Optics and Photonics, 1992.

[12] OpenMP Architecture Review Board. OpenMP: Open Multi-Processing. Accessed Aug 2018. `https://www.openmp.org`, 2013.

[13] Jean-Yves Bouguet. Camera Calibration Toolbox for Matlab. Accessed Aug 2018. `http://www.vision.caltech.edu/bouguetj/calib_doc/`, 2015.

[14] Gary Bradski and Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc.", 2008.

[15] Patrick Breheny. Robust regression.

[16] Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. Generative and discriminative voxel modeling with convolutional neural networks. *arXiv preprint arXiv:1608.04236*, 2016.

[17] Michael M Bronstein and Iasonas Kokkinos. Scale-invariant heat kernel signatures for non-rigid shape recognition. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1704–1711. IEEE, 2010.

[18] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013.

[19] CERTH. camera calibration by CERTH. Accessed Aug 2018. `http://pathway2health.eu/centre-for-research-and-technology-hellas-certh/`, 2018.

[20] Siddhartha Chaudhuri, Evangelos Kalogerakis, Leonidas Guibas, and Vladlen Koltun. Probabilistic reasoning for assembly-based 3d modeling. In *ACM Transactions on Graphics (TOG)*, volume 30, page 35. ACM, 2011.

[21] Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. On visual similarity based 3d model retrieval. In *Computer graphics forum*, volume 22, pages 223–232. Wiley Online Library, 2003.

[22] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3):145–155, 1992.

[23] Paolo Cignoni, Marco Callieri, Massimiliano Corsini, Matteo Dellepiane, Fabio Ganovelli, and Guido Ranzuglia. Meshlab: an open-source mesh processing tool. In *Eurographics Italian chapter conference*, volume 2008, pages 129–136, 2008.

[24] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[25] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312. ACM, 1996.

[26] Andrew J Davison. Real-time simultaneous localisation and mapping with a single camera. In *null*, page 1403. IEEE, 2003.

[27] Li Deng, Dong Yu, et al. Deep learning: methods and applications. *Foundations and Trends® in Signal Processing*, 7(3–4):197–387, 2014.

[28] Jorge Francisco Madrigal Díaz and Jean-Bernard Hayet. Color and motion-based particle filter target tracking in a network of overlapping cameras with multi-threading and gpgpu. *Acta Universitaria*, 23(1):9–16, 2013.

[29] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655, 2014.

[30] Mingsong Dou, Sameh Khamis, Yury Degtyarev, Philip Davidson, Sean Ryan Fanello, Adarsh Kowdle, Sergio Orts Escolano, Christoph Rhemann, David Kim, Jonathan Taylor, et al. Fusion4d: Real-time performance capture of challenging scenes. *ACM Transactions on Graphics (TOG)*, 35(4):114, 2016.

[31] SM Ali Eslami, Nicolas Heess, Christopher KI Williams, and John Winn. The shape boltzmann machine: a strong model of object shape. *International Journal of Computer Vision*, 107(2):155–176, 2014.

[32] Jannat Falah, Soheeb Khan, Tasneem Alfalah, Salsabeel FM Alfalah, Warren Chan, David K Harrison, and Vassilis Charissis. Virtual reality medical training system for anatomy education. In *Science and Information Conference (SAI), 2014*, pages 752–758. IEEE, 2014.

[33] Yi Fang, Jin Xie, Guoxian Dai, Meng Wang, Fan Zhu, Tiantian Xu, and Edward Wong. 3d deep shape descriptor. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2319–2328, 2015.

[34] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[35] Andrew W Fitzgibbon and Andrew Zisserman. Automatic camera recovery for closed or open image sequences. In *European conference on computer vision*, pages 311–326. Springer, 1998.

[36] Andreas Geiger, Frank Moosmann, Ömer Car, and Bernhard Schuster. Automatic camera and range sensor calibration using a single shot. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3936–3943. IEEE, 2012.

[37] Walter Gellert, M Hellwich, H Kästner, and H Küstner. *The VNR concise encyclopedia of mathematics*. Springer Science & Business Media, 2012.

[38] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[39] Ben Glocker, Jamie Shotton, Antonio Criminisi, and Shahram Izadi. Real-time rgb-d camera relocalization via randomized ferns for keyframe encoding. *IEEE transactions on visualization and computer graphics*, 21(5):571–583, 2015.

[40] Kan Guo, Dongqing Zou, and Xiaowu Chen. 3d mesh labeling via deep convolutional neural networks. *ACM Transactions on Graphics (TOG)*, 35(1):3, 2015.

[41] Masaki Hayashi, Steven Bachelder, Masayuki Nakajima, and Akihiko Iguchi. A new virtual museum equipped with automatic video content generator. In *Cyberworlds (CW), 2014 International Conference on*, pages 377–383. IEEE, 2014.

[42] D. O. Hebb. *The Organizatino of Behavior*. John Wiley Sons, Inc., 1949.

[43] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

[44] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[45] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[46] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8): 2554–2558, 1982.

[47] Peter J Huber et al. Robust estimation of a location parameter. *The annals of mathematical statistics*, 35(1):73–101, 1964.

[48] Aldous Huxley. *Brave New World*. 1932.

[49] Aldous Huxley. *Brave new world*. Ernst Klett Sprachen, 2008.

[50] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[51] Andrew E Johnson and Martial Hebert. Using spin images for efficient object recognition in cluttered 3d scenes. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (5):433–449, 1999.

[52] Evangelos Kalogerakis, Siddhartha Chaudhuri, Daphne Koller, and Vladlen Koltun. A probabilistic model for component-based shape synthesis. *ACM Transactions on Graphics (TOG)*, 31(4):55, 2012.

[53] Branko Karan. Calibration of kinect-type rgb-d sensors for robotic applications. *FME Transactions*, 43(1):47–54, 2015.

[54] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Skinning with dual quaternions. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 39–46. ACM, 2007.

[55] David G Kendall. A survey of the statistical theory of shape. *Statistical Science*, pages 87–99, 1989.

[56] Suraj Raghuraman Kevin Desai, Balakrishnan Prabhakaran. Skeleton-based continuous extrinsic calibration of multiple rgb-d kinect cameras. 2018.

[57] Marek Kowalski, Jacek Naruniec, and Michal Daniluk. Live scan3d: A fast and inexpensive 3d data acquisition system for multiple kinect v2 sensors. In *3D Vision (3DV), 2015 International Conference on*, pages 318–325. IEEE, 2015.

[58] Philipp Krähenbühl and Vladlen Koltun. Efficient inference in fully connected crfs with gaussian edge potentials. In *Advances in neural information processing systems*, pages 109–117, 2011.

[59] Kiriakos N Kutulakos and Steven M Seitz. A theory of shape by space carving. *International journal of computer vision*, 38(3):199–218, 2000.

[60] Wen Huey Lai, Sie Long Kek, and Kim Gaik Tay. Solving nonlinear least squares problem using gauss-newton method.

[61] Denis Le Bihan, Robert Turner, Chrit TW Moonen, and James Pekar. Imaging of diffusion and microcirculation with gradient sensitization: design, strategy, and significance. *Journal of Magnetic Resonance Imaging*, 1(1):7–28, 1991.

[62] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pages 609–616. ACM, 2009.

[63] Mingsheng Long, Yue Cao, Jianmin Wang, and Michael I Jordan. Learning transferable features with deep adaptation networks. *arXiv preprint arXiv:1502.02791*, 2015.

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

[65] Chao Ma, Jia-Bin Huang, Xiaokang Yang, and Ming-Hsuan Yang. Hierarchical convolutional features for visual tracking. In *Proceedings of the IEEE international conference on computer vision*, pages 3074–3082, 2015.

[66] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45, 2015.

[67] Daniel Maturana and Sebastian Scherer. 3d convolutional neural networks for landing zone detection from lidar. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3471–3478. IEEE, 2015.

[68] Daniel Maturana and Sebastian Scherer. Voxnet: A 3d convolutional neural network for real-time object recognition. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 922–928. IEEE, 2015.

[69] John McCormac, Ankur Handa, Andrew Davison, and Stefan Leutenegger. Semanticfusion: Dense 3d semantic mapping with convolutional neural networks. *arXiv preprint arXiv:1609.05130*, 2016.

[70] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[71] Shaohua Mi, Zengguang Hou, and Fan Yang. An 3d interactive virtual reality software toolkit for minimally invasive vascular surgery. In *Mechatronics and Automation (ICMA), 2014 IEEE International Conference on*, pages 588–593. IEEE, 2014.

[72] Matteo Munaro, Filippo Basso, and Emanuele Menegatti. Openptrack: Open source multi-camera calibration and people tracking for rgb-d camera networks. *Robotics and Autonomous Systems*, 75:525–538, 2016.

[73] British Museum. British Museum virtual exhibits. Accessed Aug 2018. `http://www.britishmuseum.org`, 2018.

[74] Olympic Museum. Olympic museum virtual tours. Accessed Aug 2018. `https://www.olympic.org/museum/visit/seminars-and-corporate-events`, 2018.

[75] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[76] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[77] Richard A Newcombe, Dieter Fox, and Steven M Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 343–352, 2015.

[78] Brandon J Newendorp, Christian Noon, Joe Holub, Eliot H Winer, Stephen Gilbert, and Julio de la Cruz. Configuring virtual reality displays in a mixed-reality environment for lvc training. In *ASME 2011 World Conference on Innovative Virtual Reality*, pages 423–430. American Society of Mechanical Engineers, 2011.

[79] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1520–1528, 2015.

[80] Juan C Núñez, Raúl Cabido, Antonio S Montemayor, and Juan J Pantrigo. Real-time human body tracking based on data fusion from multiple rgb-d sensors. *Multimedia Tools and Applications*, 76(3):4249–4271, 2017.

[81] PCL. Point Cloud Library. Accessed Aug 2018. `http://pointclouds.org/`, 2018.

[82] Panagiotis Petridis, Ian Dunwell, Fotis Liarokapis, George Constantinou, Sylvester Arnab, Sara de Freitas, and Maurice Hendrix. The herbert virtual museum. *Journal of Electrical and Computer Engineering*, 2013:16, 2013.

[83] Bui Tuong Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.

[84] François Pomerleau, Francis Colas, Roland Siegwart, et al. A review of point cloud registration algorithms for mobile robotics. *Foundations and Trends® in Robotics*, 4(1):1–104, 2015.

[85] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, 1(2):4, 2017.

[86] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In *Advances in Neural Information Processing Systems*, pages 5099–5108, 2017.

[87] Qinzhuan QIAN. State of the art in real-time 3d reconstruction based on rgb-d cameras. 2018.

[88] Tahir Rabbani, Frank Van Den Heuvel, and George Vosselmann. Segmentation of point clouds using smoothness constraint. *International archives of photogrammetry, remote sensing and spatial information sciences*, 36(5):248–253, 2006.

[89] B. Rusu. Radu. PointClouds.org: A new home for Point Cloud Library (PCL). Accessed Aug 2018. `http://www.willowgarage.com/blog/2011/03/27/point-cloud-library-pcl-moved-pointcloudsorg`, 28 March 2011.

[90] Intel RealSense. The Intel RealSense Depth Camera D415 Product Brief. Accessed Aug 2018. `https://simplecore.intel.com/realsensehub/wp-content/uploads/sites/63/D415_Series_ProductBrief_010718.pdf`, 2017.

[91] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[92] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[93] Scott D Roth. Ray casting for modeling solids. *Computer graphics and image processing*, 18(2):109–144, 1982.

[94] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[95] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. Aligning point cloud views using persistent feature histograms. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 3384–3391. IEEE, 2008.

[96] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Learning informative point classes for the acquisition of object model maps. In *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*, pages 643–650. IEEE, 2008.

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

[98] Tomasz Rybus, T Barciński, J Lisowski, J Nicolau-Kukliński, K Seweryn, M Ciesielska, K Grassmann, J Grygorczuk, M Karczewski, M Kowalski, et al. New planar air-bearing microgravity simulator for verification of space robotics numerical simulations and control algorithms. In *12th ESA Symposium on Advanced Space Technologies in Robotics and Automation, Noordwijk, The Netherlands*, 2013.

[99] Amela Sadagic. Next generation of physical training environments: Bringing in sensor systems and virtual reality technologies. In *International Conference on Augmented Cognition*, pages 717–726. Springer, 2013.

[100] Jorge Sánchez, Florent Perronnin, Thomas Mensink, and Jakob Verbeek. Image classification with the fisher vector: Theory and practice. *International journal of computer vision*, 105(3):222–245, 2013.

[101] Peter H Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966.

[102] William R Scott, Gerhard Roth, and Jean-François Rivest. View planning for automated three-dimensional object reconstruction and inspection. *ACM Computing Surveys (CSUR)*, 35(1):64–96, 2003.

[103] Shy Shalom, Ariel Shamir, Hao Zhang, and Daniel Cohen-Or. *Cone carving for surface reconstruction*, volume 29. ACM, 2010.

[104] ShapeNet. 3D ShapeNet dataset. Accessed Aug 2018. `https://www.shapenet.org/model-querier`, 2018.

[105] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-time human pose recognition in parts from single depth images. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 1297–1304. Ieee, 2011.

[106] Miroslava Slavcheva, Wadim Kehl, Nassir Navab, and Slobodan Ilic. Sdf-2-sdf: highly accurate 3d object reconstruction. In *European Conference on Computer Vision*, pages 680–696. Springer, 2016.

[107] Miroslava Slavcheva, Maximilian Baust, Daniel Cremers, and Slobodan Ilic. Killingfusion: Non-rigid 3d reconstruction without correspondences. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 3, page 7, 2017.

[108] Justin Solomon, Mirela Ben-Chen, Adrian Butscher, and Leonidas Guibas. As-killing-as-possible vector fields for planar deformation. In *Computer Graphics Forum*, volume 30, pages 1543–1552. Wiley Online Library, 2011.

[109] Harold W Sorenson. Least-squares estimation: from gauss to kalman. *IEEE spectrum*, 7(7):63–68, 1970.

[110] Statista. Forecast augmented (AR) and virtual reality (VR) market size worldwide from 2016 to 2022 (in billion U.S. dollars). Accessed Aug 2018. `https://www.statista.com/statistics/591181/global-augmented-virtual-reality-market-size/`, 2018.

[111] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. Multi-view convolutional neural networks for 3d shape recognition. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015.

[112] Robert W Sumner, Johannes Schmid, and Mark Pauly. Embedded deformation for shape manipulation. In *ACM Transactions on Graphics (TOG)*, volume 26, page 80. ACM, 2007.

[113] Yi Sun, Ding Liang, Xiaogang Wang, and Xiaoou Tang. Deepid3: Face recognition with very deep neural networks. *arXiv preprint arXiv:1502.00873*, 2015.

[114] J Suriansky and M Cmarada. Analysis of methods for camera calibration in 3d scanning systems. *Annals & Proceedings of DAAAM International.–2012*, 2012.

[115] Richard Szeliski. Structure from motion. In *Computer Vision*, pages 303–334. Springer, 2011.

[116] P. Shilane P. Min W. Kiefer A. Tal S. Rusinkiewicz T. Funkhouser, M. Kazhdan and D. Dobkin. Modeling by example. 2004.

[117] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (4):376–380, 1991.

[118] Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

[119] VPL. VPL Research. Accessed Aug 2018. `https://www.vrs.org.uk/virtual-reality-profiles/vpl-research.html`, 1984.

[120] Shenlong Wang, Sean Ryan Fanello, Christoph Rhemann, Shahram Izadi, and Pushmeet Kohli. The global patch collider. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 127–135, 2016.

[121] Thomas Whelan, Michael Kaess, Maurice Fallon, Hordur Johannsson, John Leonard, and John McDonald. Kintinuous: Spatially extended kinectfusion. 2012.

[122] Thomas Whelan, Renato F Salas-Moreno, Ben Glocker, Andrew J Davison, and Stefan Leutenegger. Elasticfusion: Real-time dense slam and light source estimation. *The International Journal of Robotics Research*, 35(14):1697–1716, 2016.

[123] Math World. Hessian Normal Form. Accessed Aug 2018. `http://mathworld.wolfram.com/HessianNormalForm.html`, 2018.

[124] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015.

[125] Li Yi, Vladimir G. Kim, Duygu Ceylan, I-Chao Shen, Mengyan Yan, Hao Su, Cewu Lu, Qixing Huang, Alla Sheffer, and Leonidas Guibas. A scalable active framework for region annotation in 3d shape collections. *SIGGRAPH Asia*, 2016.

[126] Zhengyou Zhang. Iterative point matching for registration of free-form curves and surfaces. *International journal of computer vision*, 13(2):119–152, 1994.

[127] Zhengyou Zhang. Flexible camera calibration by viewing a plane from unknown orientations. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 666–673. Ieee, 1999.

[128] Zhengyou Zhang. A flexible new technique for camera calibration. *IEEE Transactions on pattern analysis and machine intelligence*, 22, 2000.