Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Denisz Mihajlov

Autonome 3D-Objektmodellierung durch aktive Szenenexploration mit einem UR5 Greifarm

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Denisz Mihajlov

Autonome 3D-Objektmodellierung durch aktive Szenenexploration mit einem UR5 Greifarm

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Pareigis
Zweitgutachter: Prof. Dr. Tiedemann

Eingereicht am: 30. Juni 2021

Denisz Mihajlov

Thema der Arbeit

Autonome 3D-Objektmodellierung durch aktive Szenenexploration mit einem UR5 Greifarm

Stichworte

3D-Objektmodellierung, Punktwolke, autonom, Szenenexploration, Registrierung

Kurzzusammenfassung

Der Prozess der 3D-Modellierung ist oft zeitaufwendig. Das zu modellierende Objekt muss von allen Seiten untersucht werden, um eine 3D-Kopie dieses Objekts zu erstellen. Diese Bachelorarbeit beschäftigt sich mit einem Prozess der 3D-Objektmodellierung der kleinen Objekten, der autonome Verläufe hat. Dabei wird das Objekt mit einer Tiefenkamera, die auf einem Greifarm UR5 befestigt ist, untersucht. Während der Untersuchung werden die Punktwolken von der Objektoberfläche mit der Kamera aufgenommen und gespeichert. Diese Arbeit beschäftigt sich zuerst mit der Lösung des Registrierungsproblems und danach mit der Umwandlung der registrierten Punktwolken zu einem festen 3D-Objekt. In dieser Bachelorarbeit werden unterschiedliche Methoden zur Lösung dieser Probleme gezeigt. Die Punktwolkenregistierung erfolgt mit verschiedenen Varianten des ICP Algorithmus. Die 3D-Objekte aus diesen Punktwolken werden mithilfe der Poisson Surface Rekonstruktion und Alpha Shape Algorithmus gemacht. Die vorgestellten Methoden werden zusammen mit mathematischen Grundlagen dieser Methoden ausführlich erklärt.

Denisz Mihajlov

Title of the paper

Autonom 3D-Object Modelling throw Scene exploration with Roboticarm UR5

Keywords

3D object modeling, ICP, Pointcloud

Abstract

The 3D modeling process is often time-consuming. The object, that needs to be modeled, must be examined from all sides in order to create a 3D copy of this object. This bachelor thesis deals with a process of 3D modeling of small objects, this process is autonomous. The object is examined with a depth camera that is attached to a UR5 gripper arm. During the examination, the pointclouds from the object surface will be recorded with the camera and saved. This thesis deals first with the solution of the registration problem and then with the conversion

of registered pointclouds to a solid 3D object. This bachelor thesis shows different methods of solving these problems. The pointcloud registration is done with different variants of the ICP algorithm. The 3D objects from the pointclouds were made using the Poisson Surface Reconstruction and the Alpha Shape algorithm. The presented methods are explained in detail together with their mathematical basics.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

In der 3D-Computergraphik ist 3D-Modellierung der Prozess der Entwicklung einer mathematischen koordinatenbasierten Darstellung einer beliebigen Oberfläche eines Objekts (unbelebt oder lebend) in drei Dimensionen über eine spezielle Software [17]. Sie wird in vielen Bereichen eingesetzt, zum Beispiel in der Medizin[18],[19], Archäologie [20], [21] oder Computergraphik. In der Computergraphik können mithilfe von 3D-Modellierung 3D-Karten von Gebäuden oder großen Räumen erstellt werden. Zusätzlich ist es möglich bestimmte Objekte im Raum durch Szenenexploration zu modellieren. Um die Modellierung von solchen Objekten geht es in dieser Bachelorarbeit.

Diese Bachelorarbeit wurde im Rahmen des Projekts Testfeld Intelligente Quartiersmobilität (TIQ) geschrieben. TIQ oder *Testfeld Intelligente Quartiersmobilität* [22] ist ein Projekt der HAW Hamburg, welches sich mit der Entwicklung von autonomen Robotersystemen beschäftigt. Diese Systeme sollen verschiedene Aufgaben autonom erfüllen. Sie sollen den Menschen helfen komplexe und zeitaufwendige Aufgaben zu erledigen. Der Begriff der „Quartiersmobilität“ bezieht sich auf die Bewegung von Menschen oder Gütern in einem Radius von weniger als drei Kilometern. Im Rahmen dieser Arbeit wird die Roboterplattform *Husky* [3] verwendet. Als selbstfahrendes Vehikel soll sich der *Husky* autonom in einer dynamischen Umgebung zurechtfinden. Er soll lernen, sich autonom im Freien zu bewegen, SLAM Karten von der Umgebung erstellen oder die 3D-Modellierung von Objekten durchführen. Im TIQ Projekt werden auch Multisensorsysteme (MSS) entwickelt. Diese Systeme sollen die Integration von autonomen Robotersystemen in die Quartiermobilität erleichtern.

Die Systeme, die autonome 3D-Objektmodellierung durch aktive Szeneexploration darstellen, wurden schon früher gebaut und präsentiert [23] [24]. Alle diese Systeme haben unterschiedliche Systemarchitekturen und benutzten verschiedene Algorithmen für die 3D-Modellerstellung. Im Rahmen dieser Arbeit wird ein weiteres System präsentiert, das 3D-Modellierung durchführen kann. Das System nutzt einen Roboterarm und eine Tiefenkamera um Daten wie Punktwolken und Bilder von Objekten zu sammeln. Diese Daten werden benötigt, um die 3D-Modelle von Objekten erfolgreich zu konstruieren. Die Arbeit zeigt die Methoden, wie auch für kleine Objekte im Raum 3D-Modelle erstellt werden können. Bevor der Modellierungsprozess

gestartet wird, wird versucht das Objekt mithilfe des Machine Learning Objekterkennungsmodells zu erkennen. Nur für nicht erkannte Objekte wird das 3D-Modell erstellt. Der gesamte Vorgang von der Erkennung des Objekts bis zur vollständigen Modellierung dieses Objekts wird automatisiert ausgeführt. Die Entscheidung, ob ein Objekt weiter untersucht wird, wird anhand von Objekterkennungsresultaten getroffen.

Um das 3D-Modell von einem kleinen Objekt erfolgreich zu konstruieren, muss das Objekt von unterschiedlichen Seiten untersucht werden. Für die Untersuchung werden die Daten mithilfe der Tiefenkamera gesammelt. Bei der Bewegung von der Kamera um das zu untersuchende Objekt herum, entstehen Punktwolken, die die Oberfläche des Objekts aus unterschiedlichen Seiten repräsentieren. Die erstellten Punktwolken müssen zuerst in eine große Punktwolke zusammengefügt werden. Damit entsteht ein Problem dieser Modellierung, das als Registrierungsproblem bezeichnet wird. Der Prozess, bei dem die Punktwolken zusammengefügt werden, heißt Punktwolkenregistrierung. Die Registrierung kann mithilfe von unterschiedlichen Algorithmen durchgeführt werden. Einer von diesen Algorithmen ist der Iterative Closest Point Algorithmus oder ICP. Es werden zwei unterschiedliche Varianten von ICP präsentiert [25] [26]. Die erste Variante nutzt die Idee von paarweiser Punktwolkenregistrierung mit weiterer Optimierung, die andere nimmt die Punktwolken in der Reihe, wie sie aufgenommen wurden, und registriert sie nacheinander in eine große Punktwolke.

Die Registrierung von Punktwolken ist nur ein Teil der gesamten 3D-Objektmodellierung. Die Punktwolke wird nach der Registrierung zu einem festen Polygon umgewandelt. Die Erstellung von diesem Polygon wird durch zwei unterschiedliche Algorithmen präsentiert: Alpha-Shape Algorithmus [27] und Poisson Surface Rekonstruktion [16]. Die Software wird komplett parametrisierbar erstellt. Die beschriebene Algorithmen können in unterschiedlichen Variationen zusammen benutzt werden.

Im nächsten Kapitel werden die in der Arbeit genutzte Soft- und Hardware detailliert beschrieben. Kapitel 3 enthält die Vorgehensweise, die die Verläufe von allen erstellten Algorithmen vorstellt. Kapiteln 4 und 5 beschreiben die mathematischen Grundlagen von Registrierungsalgorithmen und den 3D-Modell Erstellungsalgorithmen. Im Kapitel 6 werden die erstellten Algorithmen miteinander verglichen, und die Ergebnisse gezeigt. Im letzten Kapitel werden alle Ergebnisse und wichtige Erkenntnisse der Arbeit kurz zusammengefasst.

2 Anforderungen

In diesem Kapitel wurden die Anforderungen für die Software für aktive Szeneexploration aufgelistet. Weiter folgt eine Liste von Anforderungen für das gesamte System und für die Systemkomponente:

1. Das System soll mit Parametern einstellbar sein.

   Mit diesen Parametern soll es möglich sein, die Algorithmen auszuwählen, mit denen die Punktwolkenzusammenführung und die 3D-Modellierung durchgeführt werden. Die Parameter für den Alpha Shape Algorithmus und Poisson Surface Rekonstruktion sollen auch eingestellt werden können.

2. Die Tiefenkamera soll die Punktwolken und Bilder vom Objekt sammeln.

   Während der Objektuntersuchung wird die Tiefenkamera, die auf der Husky Plattform eingebaut ist, die Punktwolken und Bilder vom Objekt aufnehmen.

3. Das System soll automatisiert die Modellierung vom Objekt durchführen.

   Das heißt, dass das System ohne menschlicher Beteiligung die Objekte untersucht, Informationen wie die Punktwolken und Bilder sammelt und anschließend das Objekt vollständig in 3D modelliert.

4. Die Punktwolke soll nur das zu untersuchende Objekt darstellen.

   Das heißt, dass auf der Punktwolke soll sich nur das Objekt befinden. Der Hintergrund und die Umgebung vom Objekt sollen nicht auf der Punktwolke erscheinen.

5. Die Tiefenkamerasichtweite soll auf 40cm gesetzt werden.

   Die Tiefenkamera soll keine Punkte auf der Punktwolke abbilden, die sich weiter als 40cm vom Kamera befinden. Das garantiert, dass der Hintergrund und die Objektumgebung nicht sichtbar werden.

6. Es sollen maximal 90 Punktwolken pro Objekt gespeichert werden.

Es dürfen nicht mehr als 90 Punktwolken aufgenommen werden, damit die Punktwolkenzusammenführung nicht zeitaufwendig ist.

7. Es soll ein neuronales Netz für die Objekterkennung gebaut werden.

   Das neuronale Netz wird versuchen die Objekte vor der Untersuchung zu erkennen.

8. Es sollen drei Bilder vom Objekt, die aus unterschiedlichen Perspektiven aufgenommen wurden, für die Erkennung benutzt werden.

   Es werden drei Bilder mit der Tiefenkamera gesammelt, die klassifiziert werden müssen. Die Bilder werden von unterschiedlichen Seiten des Objekts (vorne, links, rechts) aufgenommen. Das Objekt heißt von Netz erkannt, wenn alle drei Bilder mindestens zu 90 % mit der gleichen Klasse erkannt werden, sonst ist das Objekt als nicht erkannt bezeichnet.

9. Vor der Untersuchung soll das System versuchen das Objekt zu erkennen.

   Je nach Erfolg der Klassifizierung des Objekts wird enterscheiden, ob das Objekt weiter untersucht und modelliert werden muss.

10. Die vom neuronalen Netz erkannten Objekte sollen nicht modelliert werden.

    Alle Objekte, die vom neuronalen Netz erkannt werden, werden nicht weiter untersucht und die 3D-Modelle für diese Objekte werden nicht gebaut.

11. Die Untersuchung wird nur für unerkannte Objekte durchgeführt.

    Unerkannte Objekte werden weiter mit der Tiefenkamera untersucht, es werden Punktwolken von diesen Objekten gesammelt und zusammengefügt. Die Punktwolken werden dann weiter in ein festes 3D-Objekt umgewandelt.

12. Die Punktwolken sollen separat von Bildern gespeichert werden.

    Um die Datenverwaltung zu erleichtern, werden die Punktwolken und Bilder getrennt gespeichert. Die Punktwolken- und Bildspeicherung wird in dem Kapitel 4.2 dargestellt.

13. Der Roboterarm soll das Objekt von allen Seiten untersuchen.

    Der Roboterarm wird eine Kreisbewegung um das Objekt machen, damit die Tiefenkamera, die auf dem Arm befestigt ist, das Objekt von allen Seiten aufnehmen kann. Dadurch werden die Punktwolken von allen Objektseiten gesammelt und das Objekt kann erfolgreich und vollständig modelliert werden. Die Kreisbewegung erfolgt mit dem

festen Pfad. Die Implementierung vom Kreisbewegungalgorithmus ist in dem Kapitel 4.6 präsentiert.

14. Die gespeicherten Punktwolken sollen zu einer großen Punktwolke zusammengefügt werden.

    Mithilfe der Punktwolkenregistrierungsalgorithmen werden die gesammelten Punktwolken zu einer großen Punktwolke zusammengefügt. Die Implementierung dieser Algorithmen ist in dem Kapitel 4.4 zu sehen. Die mathematischen Grundlagen dazu sind in dem Kapitel 5 erläutert.

15. Die große Punktwolke soll in ein festes 3D-Objekt umgewandelt werden.

    Die zusammengefügten Punktwolken werden mithilfe des Alpha Shape Algorithmus und Poisson Surface Rekonstruktion in ein festes 3D-Objekt umgewandelt. Die Implementierungen zu diesen Algorithmen sind in dem Kapitel 4.5 gezeigt. Die mathematischen Grundlagen sind in dem Kapitel 6 zu finden.

16. Erstellte 3D-Objekte sollen zusammen mit den Punktwolken gespeichert werden.

    Die 3D-Objekte, die nach der Untersuchung entstehen, werden zusammen mit den Punktwolken gespeichert.

17. Die Pfade zu den erstellten Objekten sollen in einer Datei gespeichert werden.

    Damit die 3D-Modelle später identifiziert und gefunden werden können, wird eine Datei mit dem Typ csv erstellt, die die Pfade zu den 3D-Modellen und die Namen der Objekte enthält.

18. Objekte, die modelliert werden sollen, dürfen sich nicht nahe an Wänden befinden.

    Wenn sich das Objekt neben der Wand befindet, wird es unmöglich sein, das Objekt von hinten abzuscannen. Ohne diesen Scan werden die Objekte nicht vollständig (ohne den hinteren Teil des Objekts) modelliert.

Die aufgelisteten Anforderungen können mit folgendem Use Case abgebildet werden.

Angonommen ist die Husky Plattform vor einem Objekt positioniert. Dann soll das Objekt zuerst klassifiziert werden. Dafür werden drei Bilder von unterschiedlichen Perspektiven aufgenommen (vorne, links, rechts). Diese Bilder werden danach mit dem neuronalen Netz klassifiziert. Wenn alle Bilder zu der gleichen Klasse gehören, wird das Objekt als erkannt bezeichnet, ansonsten ist das Objekt nicht erkannt und die Untersuchung wird fortgesetzt. Für die Untersuchung wird sich der Roboterarm mit der Tiefenkamera um das Objekt bewegen.

Die Tiefenkamera sammelt Punktwolken und Bilder vom Objekt von unterschiedlichen Seiten. Die Punktwolken und Bilder werden dann in unterschiedlichen Ordnern gespeichert. Die gesammelten Punktwolken werden mithilfe des Registrierungsalgorithmus ICP zusammengefügt. Die registrierte Punktwolke wird danach mit dem Alpha Shape Algorithmus und der Poisson Surface Rekonstruktion in ein festes 3D-Objekt umgewandelt. Das Objekt wird im gleichen Ordner wie die Punktwolken gespeichert. Der Pfad zu diesem Objekt wird in eine csv Datei hinzugefügt. Der Name vom Objekt kann später manuell zur Datei hinzugefügt werden. Damit wird das Objekt gelabelt. Mit den gesammelten Bildern und dem Label aus der Datei kann das neuronale Netz für dieses Objekt trainiert werden und es später erfolgreich erkennen.

Um die Anforderungen zu erfüllen, wurden die benötigten Skripte in unterschiedlichen Dateien geschrieben. Die Scripte wurden separat für die Objekterkennung, Punktwolken- und Bildspeicherung, Roboterbewegung, Punktwolkenzusammenführung und 3D-Modellierung erstellt. Der Aufbau von des Systems und dieser Skripte werden in dem Kapitel 4 erläutert. Zusätzlich wird die Kommunikation zwischen Hardwarekomponenten in diesem Kapitel vorgestellt.

3 Hardware und Softwaresetup für autonome Objektmodellierung

In diesem Kapitel werden die verwendete Hard- und Software näher beschrieben. Zuerst werden die benutzten Softwarekomponenten beschrieben. Zum Anfang wird die Open3D Software vorgestellt. Sie wird für die Punktwolkenbearbeitung und die 3D-Modellierung benutzt. Danach wird die Software für die Tiefenkamera vorgestellt. Es wird gezeigt, welche Komponenten diese Software hat und welche davon in der Bachelorarbeit benutzt wurden. Weiter wird die Software ROS Melodic mit dazugehörigen Komponenten: *rviz* und *MoveIt* beschrieben. Der zweite Teil dieses Kapitels beschreibt die Hardwarekomponenten, die in der Arbeit benutzt wurden. Zuerst wird die Tiefenkamera beschrieben. Die Tiefenkamera wird für die Aufnahme von Bildern und Punktwolken benutzt. Die zweite Komponente ist Nvidia Jetson AGX Xavier, der Rechner, der für die Punktwolkenverarbeitung benutzt wird. Die Roboterplattform Husky und der Greifarm werden auch in diesem Kapitel präsentiert und beschrieben. Am Ende werden noch die Definitionen von den in dieser Arbeit verwendeten Datenstrukturen Punktwolke und Polygonnetz erläutert.

3.1 Open3d

Open3D ist eine Open-Source-Bibliothek, die die Entwicklung der Software für 3D-Daten unterstützt. Open3D stellt unterschiedliche Datenstrukturen und Algorithmen für die 3D-Modellierung und Verarbeitung in C++ und Python bereit. Die Bibliothek besitzt Funktionen für Punktwolkenbearbeitung und -registrierung, zusätzlich ermöglicht es die Punktwolken zu visualisieren. [28]

Die Erstellung von 3D-Objekten aus Punktwolken wird ebenfalls durch die Open3D-Bibliothek realisiert. Dafür wird die Bibliothek *Registrierung* benutzt. Open3D bietet Implementierungen von mehreren Registrierungsmethoden: Globale Registrierung, paarweise lokale Verfeinerung und Mehrwegregistrierung mittels **Pose Graph Optimierung** an. Alle Registrierungsmethoden nutzen ICP Algorithmus für die Ausrichtung von unterschiedlichen Punktwolken. Im Rahmen dieser Arbeit werden auf Basis von Open3D angebotenen Methoden zwei weitere

Methoden entwickelt. Eine Methode macht die Paarweiseregistrierung von Punktwolken mit zusätzlicher Optimierung mittels Pose Graph. Die andere Methode nutzt einfache Punktwolkenregistrierung mittels ICP Punkt-zu-Ebene Registrierung ohne Pose Graph Optimierung.

Open3D unterstützt auch Machine Learning Technologie Open3D-ML. Open3D-ML ist eine Erweiterung von Open3D für maschinelle 3D-Lernaufgaben. Sie baut auf der Open3D-Kernbibliothek auf und erweitert sie um maschinelle Lernwerkzeuge für die 3D-Datenverarbeitung. In dieser Arbeit werden Open3D-ML Funktionalitäten nicht benutzt. Grundsätzlich werden nur Funktionen für Punktwolkenvisualisierung, -Registrierung und 3D-Modell Erstellung verwendet.

3.2 Intel Realsense SDK

Bevor die Punktwolken mit Open3D bearbeitet werden, müssen sie erstellt und gespeichert werden. Die Punktwolken werden mithilfe der Intel RealSense Technologie erstellt. Die Intel RealSense Technologie ist eine Reihe von Technologien, mit denen Maschinen und Geräte die Funktionen zur Tiefenwahrnehmung erhalten. Die Technologien von Intel werden in autonomen Drohnen, Robotern, AR/VR und Smart-Home-Geräten eingesetzt. Intel RealSense SDK ist eine plattformübergreifende Bibliothek für Intel RealSense Tiefenkameras (D400-Serie und SR300) und die T265-Tracking-Kamera. Das SDK ermöglicht Tiefen- und Farb-Streaming und bietet intrinsische und extrinsische Kalibrierungsinformationen an. ([29]) Die Bibliothek bietet auch synthetische Streams (Punktwolke, Tiefe farblich ausgerichtet und umgekehrt) sowie eine integrierte Unterstützung für die Aufzeichnung und Wiedergabe von Streamingsitzungen. Intel Realsense SDK arbeitet mit einer Intel RealSense Kamera D435 zusammen.

Das SDK wird mit unterschiedlichen Wrappern realisiert. Grundsätzlich wird das SDK mit C++ verwendet, es gibt aber auch die Möglichkeit die Python Bindung und ROS Bindung zu nutzen. Um die Kamera mit Python oder C++ zu nutzen, muss sie mit dem Kabel zum Rechner gekoppelt werden. Da die Kamera auf dem Roboter befestigt ist und die Roboterbewegung durchgeführt wird, ist die Nutzung vom Kabel nicht erwünscht. Im Rahmen der Arbeit werden die Befehle an die Kamera mithilfe eines ROS Wrapper geschickt ([30]). Der Wrapper ist mit den Intel RealSense Kameras Versionen R200, F200, SR300 und D400 kompatibel. Er kann mit drei Versionen von ROS benutzt werden: Kinetic, Melodic und Noetic. Der ROS Wrapper enthält vier Pakete. *librealsense* ist der zugrundeliegende Bibliothekstreiber für die Kommunikation mit Intel Realsense R200, F200 und SR300 Kameras. *realsense_camera* enthält den ROS Intel Realsense Kameraknoten. Dieser Knoten schickt Daten und Befehle an die Kamera, die das

librealsense Paket nutzt. *librealsense2* ist ein SDK, das die Kommunikation mit den Kameras SR300 und D400 erstellt. *realsense2_camera* ist ein Knoten, der Informationen und Befehle an die Kameras schickt, die mit librealsense2 arbeiten. Der Kameraknoten wird mit dem Befehl *roslaunch realsense2_camera rs_camera.launch* angesteuert. Dadurch werden alle Kamerasensoren gestreamt und die Informationen aus diesen Sensoren werden zu den entsprechenden ROS-Themen veröffentlicht. Stream-Auflösungen und Bildraten können optional als Parameter für die Datei *rs_camera.launch* bereitgestellt werden. Die Publisher Topics, die erwacht werden, sind von der Kamera und den Parametern, die gesetzt wurden, abhängig. Um die Kamera richtig zu konfigurieren und eine Verbindung herzustellen, werden folgende Parameter benutzt:

- *align_depth* Wenn dieser Parameter zum Wert *TRUE* gesetzt ist, wird ein zusätzliches Topic publiziert, das alle Bilder mit Tiefenwerten zusammenfügt.
- *filters* Dieser Parameter erlaubt unterschiedliche Filter auf das Kamerabild anzuwenden. Dabei wird der Parameter auf das Wert *pointcloud* gesetzt. Mit diesem Parameter wird ein Topic *camera/depth/color/points* publiziert, damit wird es möglich, die Tiefenwerte aus dem Kamerabild zu extrahieren und als Punktwolke abzuspeichern.
- *clip_distance* Wenn dieser Parameter gesetzt ist, werden alle Tiefenwerte, die größer als der gesetzte Wert sind, aus dem Bild entfernt.

Die oben beschriebenen Parameter erlauben die Information für Punktwolkenerstellung aus der Tiefenkamera zu holen.

3.3 ROS Melodic

ROS (Robot Operating System) ist ein Open-Source, Meta-Betriebssystem für einen Roboter. Es stellt Dienste zur Verfügung, welche von einem Betriebssystem erwartet werden: Hardwareabstraktion, Gerätetreiber, Utilityfunktionen, Interprozesskommunikation und Paketmanagement.

ROS Computation Graph ist ein Peer-to-Peer Netzwerk von ROS Prozessen, welche gemeinsam eine Aufgabe erfüllen. Das Framework kann unabhängige Prozesse initiieren, diese Prozesse heißen Knoten. Die Knoten kommunizieren direkt untereinander, die Verbindungsinformationen erhalten sie vom Master ([31]). Die Knoten kommunizieren über Nachrichten, eine Nachricht ist eine Datenstruktur mit typisierten Feldern. Die Nachrichten werden über einen Publish-Subscribe Mechanismus verteilt. Ein Knoten veröffentlicht Nachrichten zu einem Thema (Topic), das über einen Namen identifiziert wird. Ein anderer Knoten, welcher

diese Nachrichten empfangen will, abonniert das Thema. Damit erfolgt die Kommunikation zwischen den Knoten im ROS Framework. Mehrere Knoten können gleichzeitig Publisher und Subscriber sein, und ein Knoten kann mehrere Themen veröffentlichen und abonnieren. Die Publisher und Subscriber im Framework sind einander nicht bekannt. Damit wird die Erzeugung und der Konsum von Informationen entkoppelt. Die Knoten müssen nicht individuell initialisiert werden, stattdessen werden sogenannte *launch* Dateien benutzt. Damit können mehrere Knoten gleichzeitig initialisiert werden.

Es werden unterschiedliche Werkzeuge in ROS Framework integriert. Die in dieser Arbeit verwendeten Instrumente sind *rviz* und *MoveIt*.

3.3.1 *rviz*

rviz steht für ROS Visualisation, ein dreidimensionales Visualisierungswerkzeug für ROS. Es hilft den Benutzern zu visualisieren, was der Roboter in der Simulation sieht, und wie er sich verhält.

3.3.2 *MoveIt*

Die MoveIt ist eine Open-Source-Plattform für die Manipulation von Robotern. Mithilfe dieser Plattform wird die Bewegung vom Roboter zunächst geplant und anschließend ausgeführt ([32]). Auch wird die inverse Kinematik vom Roboter in MoveIt automatisch ausgerechnet. Das ermöglicht dem Benutzer die Positionen für eine beliebige Pose des Roboters zu generieren. Die MoveIt Plattform kann bei der Ausführung vom Bewegungsplan auch die Trajektorie auf Kollisionen überprüfen und automatisch die Planung ändern, um diese zu vermeiden.

Für die Kommunikation innerhalb von ROS wird bei MoveIt ein Knoten mit dem Namen *move_group* benutzt. Dieser Knoten dient als Integrator: Er zieht alle einzelnen Komponenten zusammen, um eine Reihe von ROS-Aktionen und -Diensten bereitzustellen, die die Benutzer verwenden können. Der Zugriff für diesen Knoten kann auf drei Wegen erfolgen: mit C++ Paket *move_group_interface*, mit Python Paket *moveit_commander* und mithilfe vom ROS GUI *rviz*.

Insgesamt benötigt der Knoten *move_group* drei Informationen aus ROS:

1. URDF - Universal Robot Description Format ist eine XML-Datei, die im ROS benutzt wird und die Elemente vom Roboter beschreibt. Die URDF-Dateien werden benötigt, damit das ROS die Simulationen verstehen und ausspielen kann, bevor ein Benutzer den Roboter tatsächlich anwendet.

2. SRDF - Semantik Robot Description Format ist eine weitere XML-Datei, die URDF ergänzt und die Gelenkgruppen, Standardroboterkonfigurationen, zusätzliche Informationen zur Kollisionsprüfung und zusätzliche Transformationen angibt, die möglicherweise erforderlich sind, um die Pose des Roboters vollständig festzulegen.

3. Moveit Konfigurationen - *move_group* sucht auf dem ROS-Param-Server nach anderen für MoveIt spezifischen Konfigurationen, einschließlich Gelenkgrenzen, Kinematik, Bewegungsplanung, Wahrnehmung und anderen Informationen. Konfigurationsdateien für diese Komponenten werden vom MoveIt-Setup-Assistenten automatisch generiert und im Konfigurationsverzeichnis des entsprechenden MoveIt-Konfigurationspakets für den Roboter gespeichert.

3.4 Intel RealSense Tiefenkamera D435

Im folgenden Abschnitt werden die technischen Eigenschaften der benutzten Tiefenkamera aufgelistet. Intel RealSense Tiefenkamera D435 ist Teil einer Tiefenkamerafamilie der D400 Serie, sie benutzt Stereovision, um die Tiefe auszurechnen. Die D435 ist eine USB-betriebene Tiefenkamera und besteht aus zwei Tiefensensoren, einem RGB-Sensor und einem Infrarotprojektor (Siehe die Abbildung 3.1). Die Maximale Tiefenausgangsauflösung der D435 beträgt 1280x720, die maximale Tiefenbildrate 90 FPS. Die Tiefenwahrnehmung bei der Kamera variiert zwischen 10 cm und 10 m ([1]).

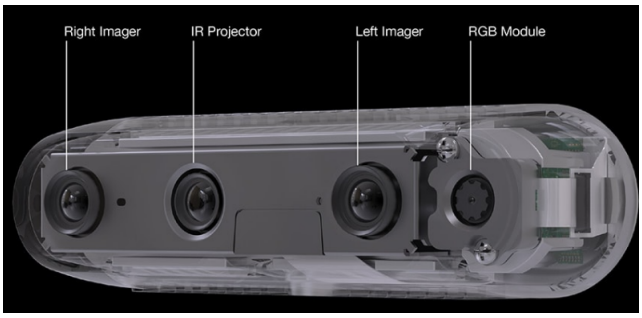


Abbildung 3.1: Die Kamera für Objektscan: Realsense Tiefenkamera D435 [1]

Im Rahmen dieser Bachelorarbeit wird diese Kamera für die Wahrnehmung und Erstellung von Punktwolken benutzt. Die Steuerung von der Kamera wird von der Intel RealSense SDK 2.0 übernommen. Die Kommunikation zwischen der Kamera und dem Rechner erfolgt durch einen ROS Wrapper. Alle nötigen Einstellungen für die Kamera werden bei der Erstellung vom ROS Kameraknoten gesetzt. Wichtig ist, die Weite von der Tiefenwahrnehmung zu ändern.

Sie wird auf 40 cm gestellt, und alle Objekte, die mehr als 40cm entfernt sind, werden auf der Wolke nicht dargestellt. Dies verbessert die Genauigkeit von der Messung und vermeidet die Entstehung von Störungen auf der Punktwolke.

3.5 Nvidia Jetson AGX Xavier

Als Rechner für die Kommunikation mit dem Roboter und für die Punktwolkenverarbeitung dient das NVIDIA Jetson AGX Xavier Developer Kit. (Abbildung 3.2) Es besteht aus einem Rechner, der für die autonomen Maschinen benutzt werden kann, und der die Leistung einer GPU-Workstation in einem eingebetteten Modul unter 30 Watt liefert. Jetson AGX Xavier wurde für Roboter, Drohnen und andere autonome Maschinen entwickelt ([2]). Das AGX Xavier Board

Abbildung 3.2: Rechner für die Punktwolkenregistrierung und 3D-Modellierung: Nvidea Jetson Xavier AGX [2]

liefert sehr gute GPU Leistungen, weshalb es in dieser Arbeit auch als Hardware zum Training Neuronaler Netze verwendet wird. Als Betriebssystem auf dem Board wird Ubuntu 18.04 benutzt. Als GPU Modul wird für das Board ein von Nvidia entwickelter graphischer Prozessor Volta mit 512 Recheneinheiten und Tensor-Recheneinheiten benutzt. Außerdem ist das Xavier Board mit 8 ARM-v8.2-64-Bit-CPU-Kerne und 32 GB vom Arbeitsspeicher ausgerüstet.

3.6 Universal Robots Greifarm

Um die Informationen über das Objekt, das abgescannt wird, zu bekommen, wird zusätzlich zur Realsense Tiefenkamera auch ein Greifarm benutzt. Der Greifarm ist von der Firma Universal Robots hergestellt.

Universal Robots ist ein dänischer Hersteller von industriellen, kollaborierenden Leichtbaurobotern - sogenannten Cobots. Die erste Produktreihe besteht aus drei Cobots dem UR3, dem UR5 und dem UR10. Alle drei Modelle sind sechsgelenkige Roboterarme mit einem Gewicht von 11 kg, 18 kg und 28 kg ([33]). Die Hubkraft von UR3 und UR5 Cobots sind 3 und 5 kg, der Arbeitsradius sind 500 mm und 850 mm. Der UR10 verfügt über eine Hubkraft von 10 kg bei einem Radius von 1300 mm. Die Cobots von Universal Robots finden Anwendung sowohl in der Metallbau-, in der Automobilbau- und in der Logistikindustrie, als auch in der Pharma-, der Kosmetik- und Medizinbranche ([34]).

Im Projekt wird eine Roboterplattform mit dem Namen Husky verwendet. (Abbildung 3.3)

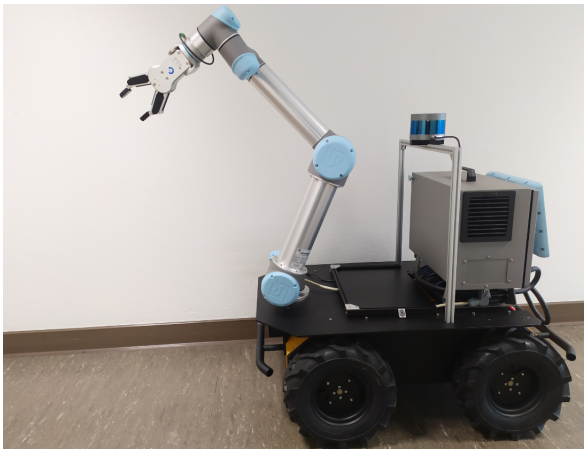


Abbildung 3.3: Plattform mit Roboterarm UR5 für Objektuntersuchung: Husky Plattform für TIQ Projekt [3]

Die Husky Plattform besitzt einen UR5 Metall Roboterarm, der mit einem flexiblen Zweifinger-Robotergreifer RG6 ausgerüstet ist. Auf der Plattform befindeen sich ebenfalls ein 3D-Laser-Scanner RD-LiDAR-16, ein Orientierungssensor UM7 und ein GPS U-Blox 7 ([3]). Zusätzlich besitzt die Plattform zwei Realsense D435 Tiefenkameras, die 3D-Messungen erlauben. Die eine Kamera ist in die Plattform integriert, die andere Kamera ist auf dem Roboterarm eingebaut. Diese Kamera übernimmt die Aufgabe des Scanners und wird die Objekte scannen. Der Greifarmbewegt sich um das zu scannende Objekt herum, damit die Tiefenkamera Informationen von allen Seiten vom Objekt aufnehmen kann. Die Bewegung wird mithilfe von MoveIt und der

Software Universal Robots ROS Driver realisiert. Der Treiber nutzt das ROS um die Steuerungsbefehle an den Roboter zu schicken. Um die Roboterplattform zu starten, wird eine Launchdatei *ur5_bringup.launch* gestartet. Um die Datei auszuführen, muss die Beschreibung vom Roboter in einer anderen XML-Datei definiert werden. Damit wird nicht nur der Arm, sondern das ganze UGV Husky gestartet, weshalb nicht die Standardbeschreibung zum Arm genutzt werden kann. In der XML-Datei, die Husky beschreibt, werden alle Hardwarekomponente des Husky aufgelistet. Damit wird das Husky Plattform gestartet und kann Aufgaben erledigen. Zusätzlich wird die vorher berechnete Inverse Kinematik benutzt, damit alle Bewegungen vom Arm richtig gemessen, wahrgenommen und durchgeführt werden können.

3.7 Datenstrukturen

3.7.1 Punktwolke

Eine Punktwolke ist eine der Hauptdatenstrukturen, die im Rahmendieser Arbeit genutzt werden. Die Punktwolke ist eine Menge von Punkten im Vektorraum, die sich ungeordnet in diesem Raum befinden. Die Erzeugung von Punktwolken erfolgt grundsätzlich über das Scanning-Verfahren. Beim Scannen eines Bereichs, zum Beispiel eines Objekts, erfasst der Laserscanner, in diesem Fall die Tiefenkamera, eine große Anzahl von Punkten, die von Kanten und Flächen dieses Objekts reflektieren.

Abbildung 3.4: Visualisierung vom Becher als Punktwolke [4]

Anhand der registrierten x-, y- und z- Koordinaten aller Punkte wird eine genaue 3D-Darstellung des gescannten Objekts erstellt. Open3D implementiert eine Datenstruktur, die Punktwolke darstellen kann *PointCloud* (Abbildung 3.4). Diese Datenstruktur enthält die Infor-

mationen über die Punkte, ihre Koordinaten, die Farbe und die Normalen. Diese Informationen sind ausreichend, um alle benötigten Manipulationen mit Punktwolken zu machen.

Es gibt unterschiedliche Bereiche, in denen Punktwolken angewendet werden können. Zunehmend werden sie in Echtzeit für Roboter und autonomfahrende Computer eingesetzt, um ihre Umgebung zu verstehen und durch sie zu navigieren. Die Daten, die man durch Punktwolken erhält, eignen sich besonders gut für die Erkennung und Identifikation von Oberflächen und Objekten: zum Beispiel von anderen Autos, Straßenschildern und der Fahrbahnmarkierung. Abbildung 3.5 zeigt ein Beispiel für die Punktwolke gezeigt, die mithilfe des Velodyne Sensor aufgenommen wurde ([5]).

Abbildung 3.5: Die Punktwolkenanwendung beim Navigieren [5]

Die Punktwolken werden auch im Bereich Geomorphologie eingesetzt, dabei können mit Punktwolken die Bodenerosionsprozesse untersucht werden ([35]). Um den Bodenabtrag zu quantifizieren, werden 3D-Punktwolken zu unterschiedlichen Zeitpunkten aufgenommen und miteinander verglichen. Archäologie ist noch ein Bereich, wo Punktwolken erfolgreich eingesetzt werden können. Durch die Analyse von Oberflächenformen wird dabei auf vergangene Siedlungsstrukturen geschlossen. Eine weitere Einsatzmöglichkeit von 3D-Punktwolken in der Archäologie ist die Modellierung von historischen Stätten ([36]).

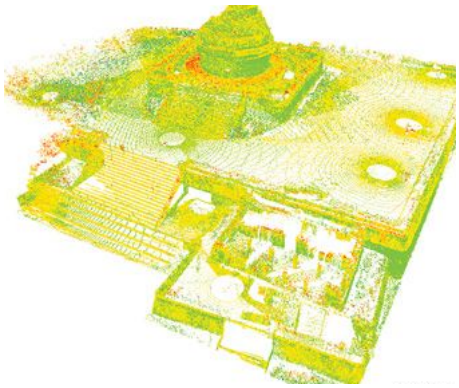


Abbildung 3.6: Die Punktwolkenanwendung in der Archäologie[6]

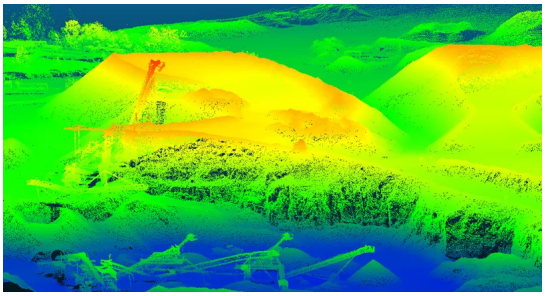


Abbildung 3.7: Die Darstellung von Landschaft als Punktwolken [7]

Die Punktwolken werden besonders häufig in der Computergrafik verwendet. Damit können die gescannten Objekte und Flächen am Computer dargestellt werden. Punktwolken sind auch zur Modellierung geeignet, dabei werden die ursprünglichen, geschlossenen Oberflächen mittels Oberflächenrekonstruktion wiederhergestellt. Die entstehenden Oberflächen bestehen meistens aus Polygonen. Die Punktwolken können für die geometrischen Messungen oder für die Identifizierung von gescannten Objekten verwendet werden. Sie haben auch in der Medizin eine Anwendung gefunden. Sie werden benutzt, um ein Oberflächenmodell zu erzeugen, um Verletzungen zu dokumentieren und diese zu rekonstruieren. Eine weiterführende Methode ist die Kombination vom Laserscanning und dem radiologischen Verfahren. Somit kann ein virtuelles 3D-Modell eines Körpers erstellt werden. In diesem Modell werden alle Verletzungen (innere und äußere) sichtbar [37]. Die Punktwolken haben außer der bereits genannten Vorteilen auch Nachteile. Ein Nachteil ist das Problem bei der Speicherung solcher Strukturen. Um ein komplettes Objekt vollständig mit Punktwolken darzustellen und zu visualisieren, müssen die Punktwolken aus verschiedenen Stellen des Objekts zusammen verbunden werden. Jede dieser Punktwolken benötigt mindestens 2 bis 5 Mb Speicherplatz. Nachdem alle Punktwolken verbunden werden, enthält die erstellte Punktwolke noch mehr Punkten und verbraucht dadurch noch mehr Speicherplatz, um gespeichert werden zu können.

3.7.2 Polygonnetz

Polygonnetz oder Polygon Mesh ist eine Sammlung von Punkten, Kanten und Facetten, die zusammen ein festes 3D-Objekt darstellen können. Die Facetten von solchen Netzen werden mit Dreiecken oder Vierecken erstellt.

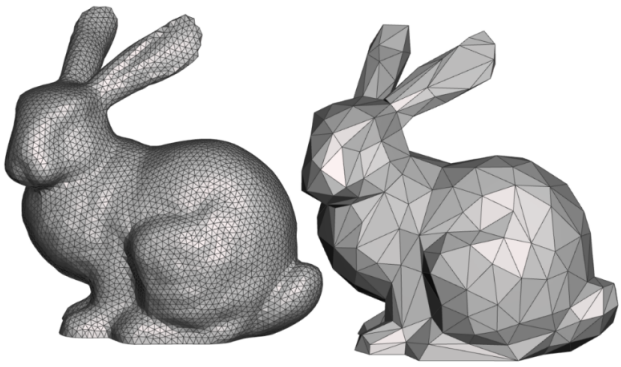


Abbildung 3.8: Beispiele für ein Polygonnetz [8]

Ein Netz kann die folgenden Eingeschaften haben, aber für keines ist ein Polygonnetz erforderlich [38]:

1. Strukturiertheit: Ein Polygonnetz wird als strukturiert bezeichnet, wenn jeder innere Punkt die gleiche Anzahl anliegender Kanten und Flächen hat.

2. Regularität: Ein Polygonnetz ist regulär, wenn die Kantenlängen in jede Richtung konstant sind. Diese Eigenschaft baut auf der Strukturiertheit auf.

3. Orthogonalität: Ein Polygonnetz wird als orthogonal bezeichnet, wenn die Netzkanten rechte Winkel bilden. Die Orthogonalität baut auf der Eigenschaft der Strukturiertheit und der Regularität auf.

Objekte, die mit Polygonnetzen erstellt wurden, müssen verschiedene Arten von Elementen speichern. Dazu gehören Vertexe, Kanten und Facetten.

1. Vertex wird mit einem Punkt repräsentiert. Dieser Punkt kann außer der Position noch zusätzlich Farbe, Normale und Texturkoordinate haben.

2. Kante ist eine Linie, die zwei Vertexe miteinander verbindet. Kanten können gerichtet und ungerichtet sein.

3. Facette ist ein geschlossener Satz von Kanten, bei dem eine Dreiecksfläche drei Kanten und eine Quad-Fläche vier Kanten hat. Ein Polygon ist eine komplanare Menge von Facetten.

Polygone können mit unterschiedlichen Datenstrukturen repräsentiert werden: zum Beispiel Face-vertex Polygone, Winged-edge, Half-edge.

4 Kommunikation und Algorithmen in der Szenenexploration

In diesem Kapitel wird die gesamte Funktionalität des geschriebenen Programms erklärt. Zuerst wird das ausgewählte Architekturmodell beschrieben. Es wird gezeigt, wie die benutzten Software- und Hardwarekomponenten miteinander kommunizieren und die erwünschten Aufgaben autonom erledigen können. In diesem Kapitel wird gezeigt, wie die Nachrichtenübertragung zwischen den Hardwarekomponenten erfolgt und wie die Punktwolken und Bilder gespeichert werden. Zusätzlich wird ein neuronales Netz für die Objekterkennung präsentiert. In dem Kapitel wird gezeigt, wie dieses Netz trainiert wurde und welche Datenmenge dafür benutzt wurde. Am Ende dieses Kapitels werden die Algorithmen für die Objektanalyse, Punktwolkenregistrierung und 3D-Objektmodellierung beschrieben. Es wird gezeigt, wie diese Algorithmen mit der benutzten Software in dieser Bachelorarbeit implementiert werden.

4.1 Server-Client Verbindung

In der Arbeit werden zwei unterschiedlichen Umgebungen benutzt, eine mit der Python Version 2.7 und eine andere mit der Python Version 3.6. Ein Rechner, der die Rolle des Clients impliziert, wird für die Robotersteuerung mit ROS benutzt. Der andere Rechner, der als Server bezeichnet wird, steuert die Kamera. Zusätzlich werden dort Punktwolken und Bilder gespeichert, und das 3D-Objekt erstellt. Beim Start vom Server werden die Parameter für das System gesetzt. Mit diesen Parameter werden die benötigte Algorithmen ausgewählt, zusätzlich werden solche Parametern wie alpha für Alpha-Shape Algorithmus und Tiefe für Poisson Surface Rekonstruktion ermittelt (Anforderung 1). Diese Einordnung ist erforderlich, weil die ROS Melodic, die für die Roboterarmsteuerung benutzt wird, nur mit Python 2.7 arbeiten kann. Die Open3D Bibliothek für die Punktwolken Registrierung und die 3D-Objekterstellung benötigt hingegen Python 3.6. In der Abbildung 4.1 ist die Verbindung von allen Hardware Komponenten der Arbeit gezeigt.

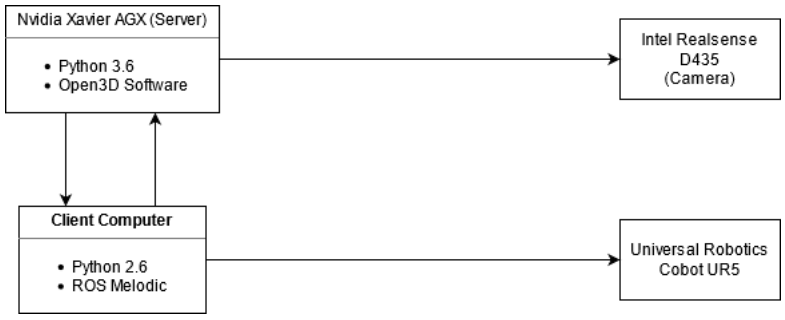


Abbildung 4.1: Die Verbindung zwischen die Hardware und Software Komponenten

Die Kommunikation erfolgt durch die Nachrichtenübertragung zwischen Server und Client. Die erste und einzige Nachricht, die vom Benutzer manuell geschickt werden muss, ist *start*. Diese Nachricht startet dan gesamten Prozess, weitere Nachrichtenübertragungen erfolgen automatisch und sind von dem Objekterkennungsresultat abhängig. Sobald *start* geschickt wurde, werden zwei Ordner für die Objektdaten erstellt. An der Clientseite, beim Empfang von *start*, wird das Skript gestartet, der Bilder für die Objekterkennung sammelt. Dieses Skript steuert den Roboterarm und schickt Nachrichten an den Server, damit Bilder mit der Kamera aufgenommen werden. Das Skript bewegt den Arm zuerst zur Linkeseite des Objekts und schickt die Nachricht *foto*. Danach wird der Arm zur Vordersiete vom Objekt bewegt und es wird eine weitere *foto* Nachricht geschickt und ein Bild gemacht. Anschließend wird ein Bild von der rechten Seite aufgenommen und die Bilder klassifiziert. Der Roboterarm wird nach der Aufnahme zurück zu Vorderseite des Objekts bewegt.

Wenn alle drei Bilder derselben Klasse zugeordnet sind, wird kein Scan verlangt, ansonsten wird das Objekt abgescannt. Wenn ein Scan erforderlich ist, schickt der Server die Nachricht *scan* und damit wird das Skript mit der Kreisbewegung gestartet. Zuerst wird der Roboter hinten links vom Objekt positioniert, dann sendet der Client eine Nachricht *sc*.Der Roboter sich langsam um das Objekt im Kreis bewegen, bis er hinten rechts vom Objekt ankommt. Dann schickt der Client eine Nachricht *stop*. Wenn der Server diese Nachricht *sc* erhält, werden mithilfe der Python Bibliothek *subprocess* zwei Prozesse gestartet. Beide Prozesse steuern die Tiefenkamera. Ein Prozess nimmt auf und speichert Punktwolken für die Registrierung, das andere parallel speichert die Bilder vom Objekt. Die Bilder von unerkannten Objekten werden später benutzt, um das ML-Modell mit neuen Daten zu trainieren. Sobald die Nachricht *stop* angekommen ist, sind beide Prozesse terminiert. Danach wird mit der 3D-Objekterstellung begonnen. Wenn das Objekt erstellt und gespeichert ist, wartet der Server auf die neue *start* Nachricht, um weitere Objekte zu untersuchen. Abbildung 4.2 zeigt die Kommunikation zwischen Server und Client grafisch.

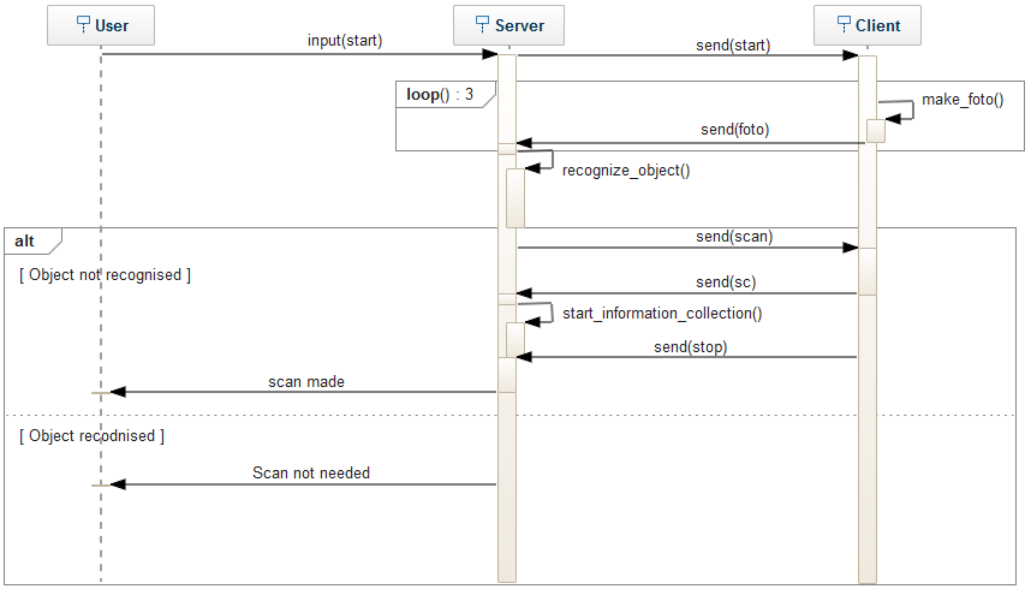


Abbildung 4.2: Sequenz Diagramm von der Server-Client Kommunikation

4.2 Punktwolken- und Bilderspeicherung

Bevor die Objektanalyse und das Scanning stattfindet, müssen alle benötigten ROS Knoten, mit denen die Kommunikation durchgeführt wird gestartet werden. Zuerst wird der Husky Knoten gestartet, damit der Roboterarm und die Tiefenkamera erreichbar sind. Um mit der Kamera zu kommunizieren, muss ein Kameraknoten gestartet werden. Der Befehl, um diesen zu starten, ist in der Abbildung 4.3 gezeigt.

```
nvidia@nvidia-desktop:~$ roslaunch realsense2_camera rs_camera.launch filters:=p
ointcloud allign_depth:=true enable_pointcloud:=true clip_distance:=0.4
```

Abbildung 4.3: Befehl um den Kameraknoten zu starten

Mit diesen Parametern werden alle Punkte, die nicht weiter als 40 cm von der Kamera entfernt sind, zu den Punktwolken hinzugefügt (Anoforderung 5). Damit wird sichergestellt, dass die Hintergrund und die Umgebung vom Objekt nicht auf der Punktwolke abgebildet werden (Anforderung 4) Um die Punktwolken zu speichern, wird das Topic */camera/depth/color/points* abonniert, die benötigten Daten befinden sich in der Nachricht *PointCloud2*. Diese Daten werden in eine Array umgewandelt, auf Basis dieser Daten wird mithilfe der Open3D Software die Punktwolke erstellt und gespeichert. Um die Bilder zu speichern, wird das Topic */camera/color/image_raw* abonniert. Die Daten besitzen den Typen *Image*, sie werden mithilfe

des *scipy.misc* Pakets gespeichert. Damit werden die Punktwolken und Bilder vom Objekt mithilfe der Tiefenkamera gesammelt und gespeichert (Anforderung 2).

Alle Daten werden in unterschiedlichen Ordnern, die beim Serverstart erstellt werden, gespeichert (Anforderung 12). Für die Punktwolken wird ein Ordner mit dem Namen *Object + die aktuelle Zeit* erstellt. Dort werden alle gesammelten Punktwolken, die registrierte Punktwolke und die erstellten 3D-Modellen gespeichert. Die Bilder werden im Ordner *ObjectFotos + aktuelle Zeit* gespeichert. Diese Zuordnung vereinfacht die zukünftige Verwaltung von gescannten Objekten. Learning Modells genutzt. In der Abbildung 4.4 wird die beschriebene Datenverwaltung als Flussdiagramm dargestellt.

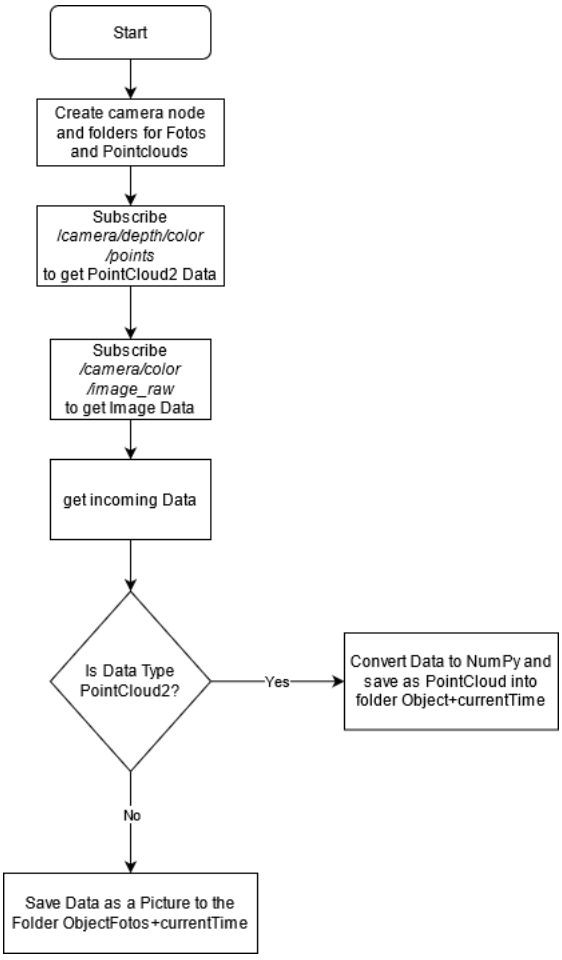


Abbildung 4.4: Flussdiagramm für Datenverwaltung

4.3 Objekterkennung

Sobald die ersten drei Bilder vom Objekt gespeichert sind, müssen diese klassifiziert werden (Anforderung 7). Im Folgenden wird die genutzte Architektur des Machine Learning Modells, die Datenmenge und der zugehörige Algorithmus für die Erkennung beschrieben.

4.3.1 MobileNetV2

Für die Objekterkennung wird das ML Modell MobileNetV2 benutzt ([39]). Das Modell besteht aus Blöcken, die als "Bottleneck Residual Block"bezeichnet werden. Jeder dieser Blöcke verfügt über drei Schichten. Die erste Schicht ist eine 1x1 Convolutional Schicht, die auch Expansionsschicht genannt wird. Diese Schicht vergrößert die Anzahl von Kanälen in den Daten, bevor sie in die nächste Schicht übertragen werden. Die Größe der Datenerweiterung wird wird durch den Wert des Expansionsfaktors angegeben. Im grundlegenden MobileNetV2 Modell wird der Expansionsfaktor 6 benutzt. Wenn ein Tensor mit 32 Kanälen durch die Expansionsschicht gelangt, haben die Daten $32 * 6 = 192$ Kanäle.

Die nächste Schicht im Block ist die Depthwise Convolutional Schicht. Bei dieser Schicht werden Filter auf die eingehenden Daten angewendet. Bei der Depthwise Convolutional Schicht werden für jeden Eingangskanal die Filter separat angewendet, die Kanäle werden nicht zusammen genutzt, wie bei der normalen Convolutional Schicht. Jeder Eingang wird mit dem jeweiligen Filter gefaltet und danach werden alle Kanäle zurück zusammengestapelt.

Die letzte Schicht im Block ist die sogenannte Bottleneck Schicht. Diese Schicht projiziert Daten mit hoher Anzahl von Kanälen zum Tensor mit kleinerer Anzahl von Kanälen. Zum Beispiel, wenn nach den ersten zwei Schichten 144 Kanäle entstehen, wird die Bottleneck Schicht diese Anzahl zu 24 reduzieren.

Der gesamte Block ist gleichzeitig ein Residual Block. Ein Residual Block ist ein Stapel von Schichten, die so festgelegt sind, dass die Ausgabe einer Schicht zu der Ausgabe von einer anderen Schicht, die sich tiefer im Block befindet, addiert wird. Diese Struktur ist ähnlich zu der Struktur von ResNet. Eine hohe Anzahl der Schichten in einem Netz führt zur Steigerung des Fehlerpotenzials beim Training. Die Residual Block Strukturen erlauben viele Schichten im Netz zu haben, ohne das Fehlerpotenzial beim Training zu steigern. Jede der beschriebenen Schichten wird von einer Batch Normalization und Aktivierungsfunktion ReLU6 gefolgt. Nur die letzte dritte Bottleneck Schicht hat kein ReLU6 am Ende. Die Architektur besteht aus 17 Blöcken, die in eine Reihe eingeordnet sind. Danach folgen eine 1x1 Convolutional Schicht, Global Average Pooling Schicht und eine Klassifikationsschicht am Ende. Die Anzahl der Parameter variiert zwischen 1.7 Millionen und 6.9 Millionen.

In der Arbeit zusätzlich zu originaler ModelNetV2 wird noch ein Dropout angewendet, Die letzte Schicht ist eine Klassifizierungsschicht mit 10 Klassen und einem Softmax als Aktivierungsfunktion. Da das ausgewählte ModelNetV2 Modell bereits mit einer ImageNet Datenmenge trainiert war, werden die Schichten vom ModelNetV2 eingefroren. Die Gewichte werden dort im Laufe des Trainings nicht geändert. In der Abbildung 4.5 wird eine kurze Zusammenfassung für das Modell gegeben.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
keras_layer (KerasLayer)     (None, 1280)              2257984
_________________________________________________________________
dropout (Dropout)            (None, 1280)              0
_________________________________________________________________
dense (Dense)                (None, 10)                12810
=================================================================
Total params: 2,270,794
Trainable params: 12,810
Non-trainable params: 2,257,984
```

Abbildung 4.5: Die Schichten von verwendete ModelNetV2 Modell

4.3.2 Datenmenge aus ImageNet

Die Datenmenge für das Modelltraining wird mithilfe von Bildern aus ImageNet erstellt [9].Die ImageNet Datenmenge war nicht direkt in der Bachelorarbeit benutzt. Die Bilder für zugehörige Klassen waren aus ImageNet manuell heruntergeladen. Danach waren diese Bilder in eine Datenmenge zusammengestellt und für das Training benutzt. Jede Klasse besteht aus 400-450 Bildern. Insgesamt enthält die Datenmenge ungefähr 4500 Bilder. Die Bilder werden für jede Klasse in die Trainings- und Validierungsmenge aufgeteilt. Von der gesamten Menge werden 40 Prozent zu der Validierungsmenge zugeordnet. Alle Bilder sind zu zehn unterschiedlichen Klassen zugeteilt. Das Modell wird mit Klassen aus Abbildung 4.6 trainiert.

```
{'Bottle': 0, 'Bucket': 1, 'Carton_Box': 2, 'Cell_Phone': 3, 'Chair': 4
, 'Flower Pot': 5, 'Laptop': 6, 'Monitor': 7, 'Mug': 8, 'Trash_Can': 9}
```

Abbildung 4.6: 10 Klasse mit dennen das ModelNetV2 traniert war

Es wurden nur die Objekte von relativ kleinen Größen ausgewählt. Alle Bilder, die im Training benutzt werden, worden aus ImageNet Datenmenge heruntergeladen. Die Beispiele für die Bilder sind in der Abbildung 4.7 gezeigt.

Abbildung 4.7: Beispiele für die Bilder, die für das Modelltraining benutzt wurden [9]

4.3.3 Modelltraining

Das Modell wurde mit der obenbeschriebenen Datenmenge trainiert. Aus 2715 Bilder entsteht eine Trainingsmenge und aus 1802 Bilder - eine Validierungsmenge. Das Modell wurde mit 60 Epochen trainiert, jede Epoche benötigt ungefähr 60 Sekunden um abgeschlossen zu sein. Mit diesen Einstellungen wurde eine Genauigkeit von 92.05 % erreicht, und der Loss beträgt 0.26. In der Abbildung 4.9 ist die Genauigkeit während des Trainings präsentiert. In der Abbildung 4.8 ist Confisionmatrix zu sehen. Es ist zu erkennen, dass das Modell alle Objekte außer den Becher (Mug) mit der Wahrscheinlichkeit 1 erkennen kann. In der Abbildung 4.10 sind die Resultate des Erkennungs nach dem Training gezeigt.

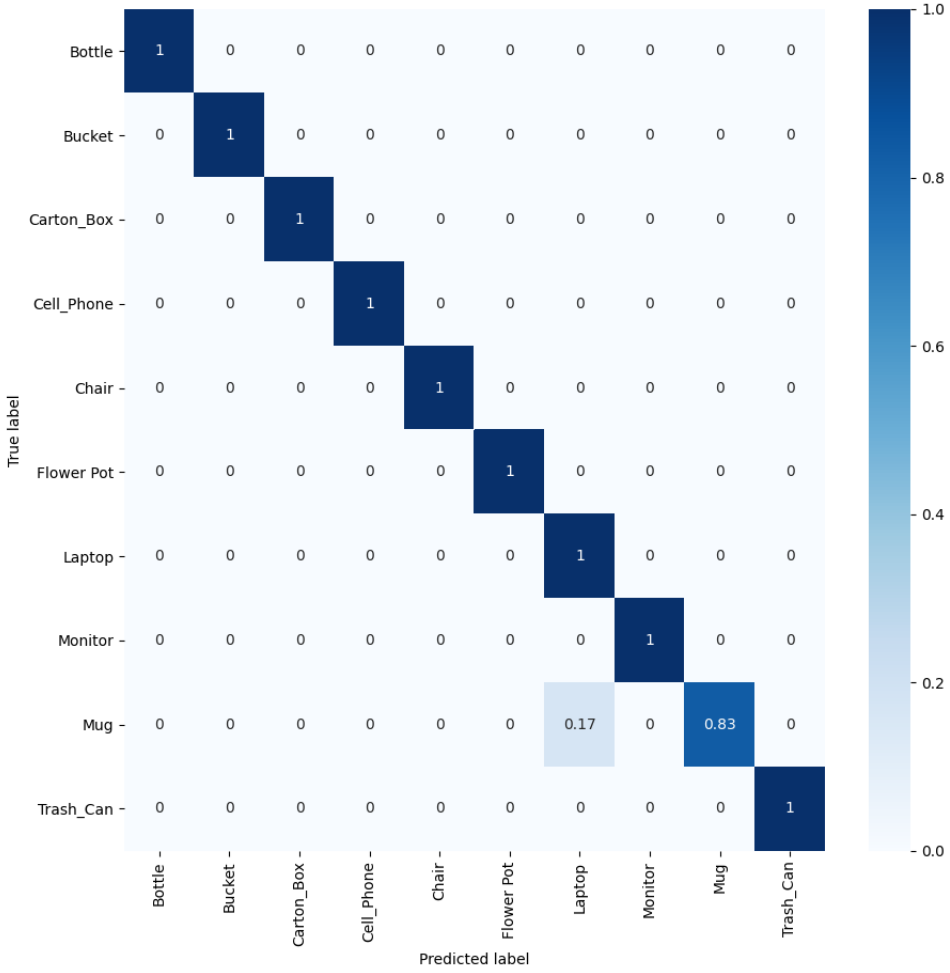


Abbildung 4.8: Confusionmatrix von erstellte neuronales Netzl

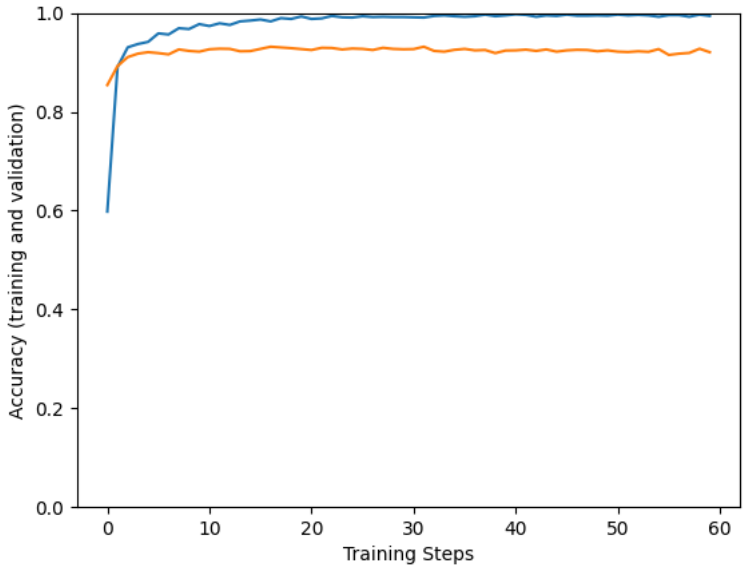


Abbildung 4.9: Grafik für Entwiklung von Genauigkeit während des Trainings

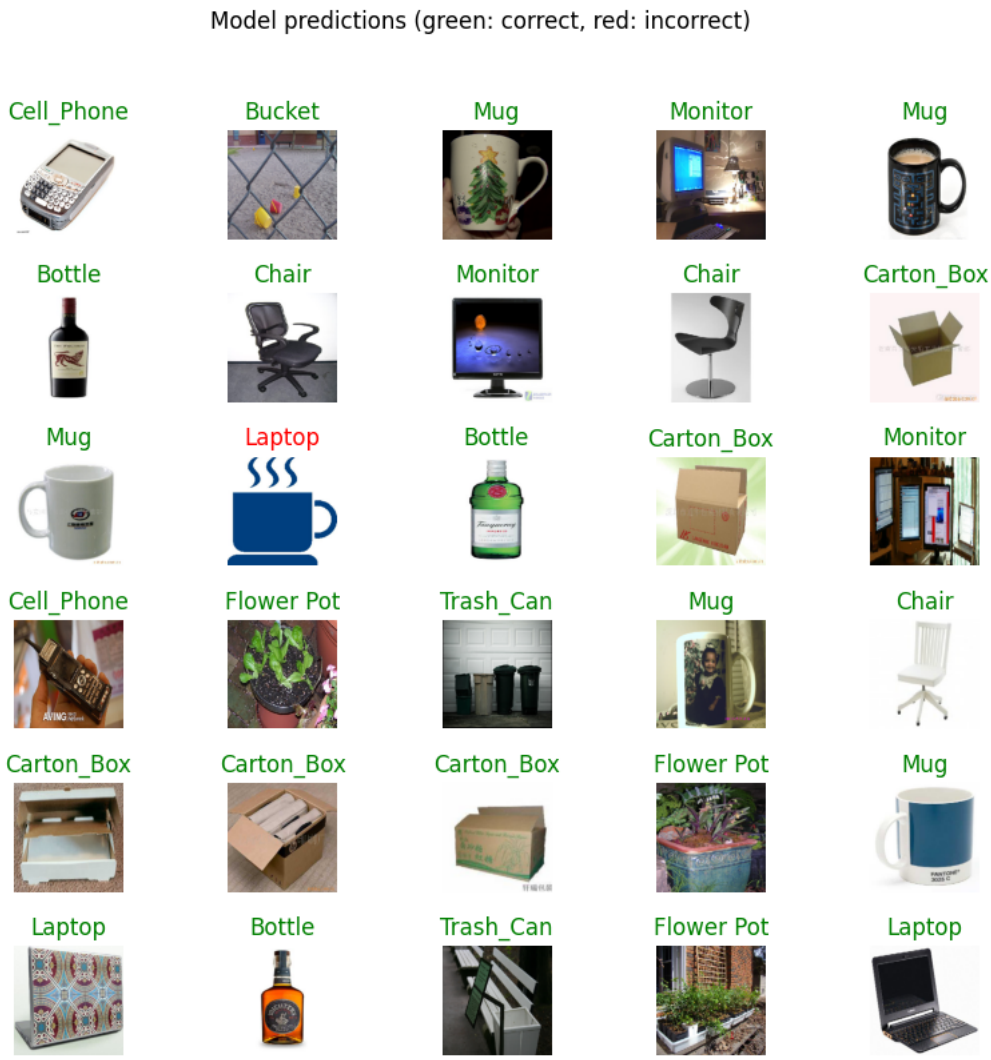


Abbildung 4.10: Klassifiziereungsresultate des Netzes

4.4 Objektanalyse und Registrierung

Bevor das Objekt abgescannt wird, muss zuerst entschieden werden, ob ein Scan benötigt wird, dafür werden zunächst Bilder vom Objekt aus drei unterschiedlichen Perspektiven gemacht. Es werden Bilder von der Vorderseite, der linken Seite und der rechten Seite des Objekts aufgenommen. Nachdem die Bilder aufgenommen sind, werden deren Klassen mithilfeeines

Machine Learning Netzes ermittelt (Anforderung 8). Die ermittelten Klassen werden analysiert, wenn alle drei Bilder die gleiche Klasse haben, wird das Objekt als erkannt bezeichnet und der Scan ist nicht erforderlich (Anforderung 10).

Wenn weniger als drei Bilder die gleiche Klasse haben, wird das Objekt nicht erkannt und es wird abgescannt (Anforderung 11). Dafür bewegt sich der Roboterarm mit integrierter Tiefenkamera um das Objekt (Anforderung 13). Bei der Bewegung werden Punktwolken und Bilder vom Objekt gesammelt. Durchschnittlich werden 70-90 Bilder und Punktwolken aufgenommen (Anforderung 6). Nachdem die Daten vom Objekt gesammelt wurden, werden die Punktwolken zu einer großen Punktwolke zusammengefügt. Das Flussdiagramm, das die Analyse und Untersuchung beschreibt ist in der Abbildung 4.11 zu sehen.

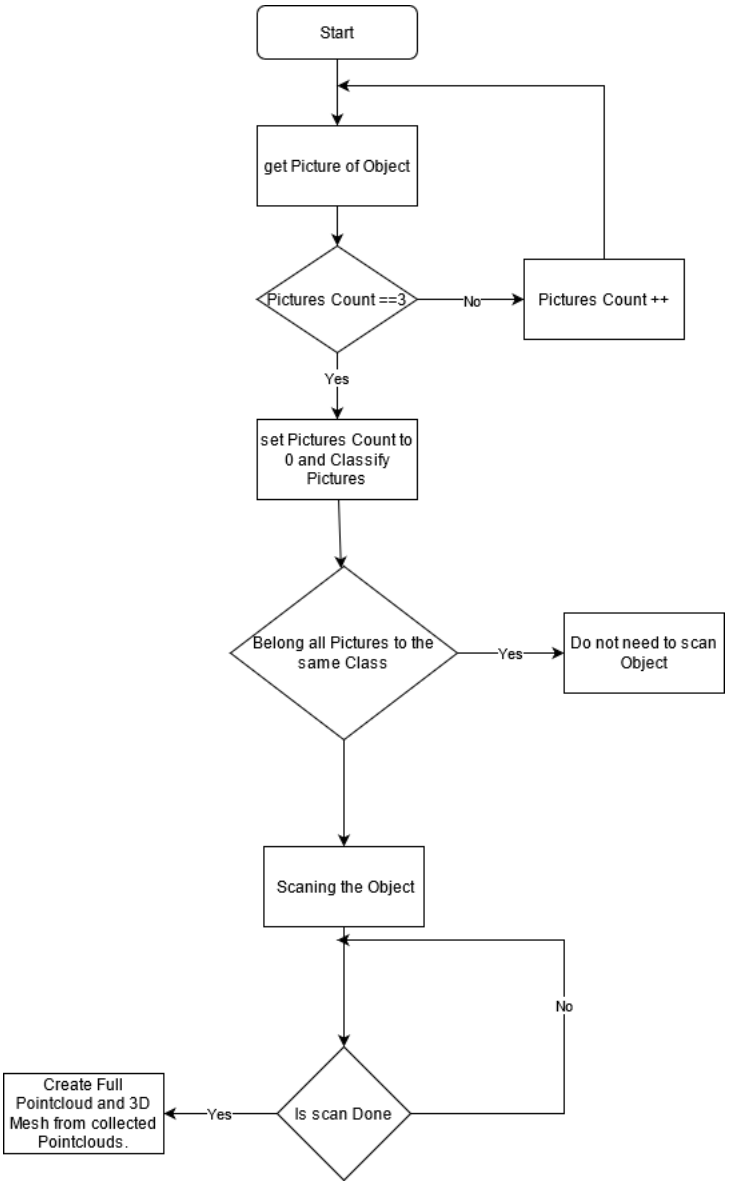


Abbildung 4.11: Das Flussdiagramm, das der Ablauf von Objekt Untersuchung darstellt

Im Rahmen dieser Arbeit werden zwei unterschiedliche Methoden für die Punktwolkenregistrierung verglichen. Eine Methode nutzt die Punkt-zu-Ebene Registrierung von Punktwolken ohne Pose Graph Optimierung. Die andere Methode wird mithilfe von paarweiser Punkt-zu-Ebene Registierung mit Pose Graph Optimierung realisiert. Die mathematische Grundlagen zu beiden Methoden sind im Kapitel 4 dargestellt. Beide Methoden werden die gesammelten Punktwolken zu einer großen Punktwolke zusammenfügen (Anforderung 14)

Punkt-zu-Ebene Registrierung ohne Pose Graph Optimierung

Bei der Punkt-zu-Ebene Registrierung ohne Pose Graph Optimierung werden zuerst alle zum Objekt gehörige Punktwolken gelesen. Die Punktwolken werden in gleicher Reihenfolge gelesen, in der sie aufgenommen wurden. Danach erfolgt die Registrierung dieser Punktwolken mithilfe der Open3D Bibliothek und dem Punkt-zu-Ebene ICP Algorithmus. Zuerst werden die ersten zwei Punktwolken registriert. Diese zwei Punktwolken bilden die Basis für die weitere Registrierung. Alle weiteren Punktwolken werden mit dieser Basis weiter zusammengefügt und werden diese Basis erweitern. Vor der Registrierung werden die Punktwolken vereinfacht, damit sie weniger Punkte enthalten. Dies vereinfacht und beschleunigt die Registrierung ohne Verlust wichtiger Informationen. Die Vereinfachung wird mit der Punktwolkenvoxelization erreicht. Es wird mit der Open3D Methode *voxel_down_sample(size)* gemacht. Der Parameter size bezeichnet einen Voxelizationswert und wird bei diesem Algorithmus auf 0.002 gesetzt. Auf Basis vom Voxelizationswert werden die Korrespondenzdistanzen ermittelt. Da die Punkt-zu-Ebene Methode die Normalen für die Registrierung braucht, werden für jede Punktwolke zuerst die Normalen berechnet. Die Berechnung wird mit Open3D Methode *estimate_normals()* gemacht.

Die Registrierung erfolgt in zwei Schritten. Der erste Schritt ist die grobe Registrierung mit einer Korrespondenzdistanz von $10 * voxel\_size$ und einer Identitätsmatrix, das Ergebnis ist eine neue Korrespondenzmatrix. Im ersten Schritt wird die sogenannte initiale Registrierung von den Punktwolken gemacht. Der nächste Schritt ist eine weitere Registrierung von ausgewählten Punktwolken, dafür werden andere Parameter benutzt. Die Korrespondenzdistanz ist $1.5 * voxel\_size$ und als Korrespondenzmatrix wird die Matrix aus dem ersten Schritt benutzt. Im zweiten Schritt werden die Distanzen zwischen den korrespondierenden Punkten in den beiden Punktwolken noch mehr verkleinert und die Registrierung dadurch genauer. Der zweite Schritt gibt eine Transformationsmatrix zurück, mit der eine Punktwolke transformiert wird. Um das Endergebnis der Registrierung von zwei Punktwolken zu erhalten, wird zu der transformierten Punktwolke eine andere Punktwolke aufaddiert. Damit werden die Punktwolken zusammengefügt. Die Registrierung wird mit der Methode aus Open3D *registration_icp* gemacht. Als Parametern erhält diese Methode zwei Punktwolken, eine Korrespondenzdistanz, TransformationsMatrix und das benötigte Algorithmus (Punkt-zu-Punkt oder Punkt-zu-Ebene).

Paarweise Punkt-zu-Ebene Registrierung mit Pose Graph Optimierung

Die zweite Methode nutzt ebenfalls die Punkt-zu-Ebene ICP Registrierung, aber die Punktwolken werden mithilfe vom Pose Graph optimiert. Genau wie beim erstem Skript werden

zuerst alle Punktwolken gelesen. Die Punktwolken werden auch mit der Voxelization vereinfacht und am Anfang wird der Pose Graph initialisiert. Der erste Knoten in dem Pose Graph ist eine Identitätsmatrix. Dieser Knoten wird am Ende für die Optimierung als ein globaler Raum benutzt. Der Knoten wird mit folgendem Befehl zum Pose Graph hinzugefügt: *pose_graph.nodes.append(node)*. Die Knoten sind die Teile von der gesamten Geometrie und repräsentieren die Transformationsmatrizen, die die Punktwolken zum globalen Raum transformieren. Jede Punktwolke wird dann paarweise mit jeder anderer Punktwolke registriert. Genau wie im ersten Skript erfolgt die Registrierung in zwei Schritten: grobe Registrierung mit Korrespondenzdistanz von $10 * voxel\_size$ und Identitätsmatrix, und eine verfeinerte Registrierung mit Korrespondenzdistanz von $1.5 * voxel\_size$, als Matrix wird die Transformationsmatrix aus dem ersten Schritt genommen. Die Voxelizationswert wird bei diesem Algorithmus auf das Wert 0.004 gesetzt. Nur die Transformationen von benachbarten Punktwolkenpaaren erstellen einen Knoten, der zum Pose Graph hinzugefügt wird.

Die anderen Teile vom Pose Graph sind die Kanten. Die Kanten repräsentieren die Transformation zwischen zwei Knoten und besitzen die Information über die Transformationsmatrix, die zwei Punktwolken registriert. Um einen Graph zu optimieren, werden die Kanten in zwei unterschiedliche Klassen unterteilt. Alle Kanten, die Information über zwei benachbarten Punktwolken besitzen, sind Odometriekanten. Alle anderen Kanten werden als Schlaufenverschlusskanten bezeichnet. Die Kanten werden mit folgendem Befehl erstellt **pipelines.registration.PoseGraphEdge()**. Die Kanten mit *uncertain=False* sind die Odometriekanten. Wenn die Variable den Wert *True* hat, ist diese Kante eine Schlaufenverschlusskante. Nachdem der Pose Graph erstellt wird, erfolgt die Optimierung mithilfe des Levenber-Marquardt Algorithmus. Es wird eine *edge_prune_threshold* Variable gesetzt. Alle Kanten, die diesen Threshold nicht überschneiden, werden aus dem Pose Graph entfernt, danach wird eine gesamte Punktwolke erstellt. Um diese Punktwolke zu erstellen, wird jede Punktwolke mit einer korrespondierender Pose aus dem Pose Graph transformiert und zur gesamten Punktwolke hinzugefügt. Die mathematische Beschreibung und eine detaillierte Erklärung dieser Methode wird in Kapitel 4 erläutert.

Zusätzlich wird die erstellte Punktwolke mit zwei weiteren Optimierungsskripten verarbeitet. Ein Skript entfernt den Boden aus dem Objekt und das zweite Skript entfernt die Defekte und unnötige Punkten aus dem gesamten Bild. Diese Optimierungen sind erforderlich, damit das 3D-Objekt keine unnötigen Punkte erhält, und die 3D-Erstellung schneller ist. Dafür waren zwei Methoden entwickelt *groundRemove()* und *outliner_remove()*. *groundRemove()* bekommt Pfad zu registrierte Punktwolke als Parameter. Mit der Methode aus Open3D *segment_plane*, wird den Boden aus der Punktwolke entfernt. Andere Methode bekommt auch Pfad zu registrierte

Punktwolke als Parameter und nutzt *remove_statistical_outlier()* um die unnötigen Punkten aus der Punktwolke zu entfernen.

4.5 3D Polygon Erstellung

Sobald alle Punktwolken registriert sind und eine gesamte Punktwolke erstellt ist, wird diese Punktwolke zu einem festem Polygon umgewandelt (Anforderung 15). Die Umwandlung kann mit zwei unterschiedlichen Algorithmen erfolgen, damit werden zwei 3D-Objekte erstellt. Das erste Skript nutzt den Alpha Shape Algorithmus für die Objekterstellung. Es benötigt zwei Variablen: eine ist die Punktwolke und die andere ist ein Wert von Alpha, womit das Objekt erstellt wird. Das Alpha kann beim Serverstart ausgewählt werden, der Standandwert für Alpha ist auf 0.007 gesetzt. Die Erstellung erfolgt mit dem Befehl *create_from_point_cloud_alpha_shape(pcd, alpha)*.

Das zweite Skript führt die 3D-Objekterstellung mit dem Poisson Algorithmus durch. Das Skript benötigt zwei Variablen: eine ist die Punktwolke und die andere ist die Tiefe vom 3D-Objekt. Je größer der Tiefenparameter ist, desto detaillierter ist das Objekt. Der Befehl für die Erstellung ist: *create_from_point_cloud_poisson(pcd, depth=9)*. Es werden zwei Objekte zurückgegeben: ein 3D-Objekt selbst und die Menge von Dichten. In dieser Menge stehen alle Vertex des Objekts mit der Anzahl von Punkten aus der Punktwolke, die zu diesem Vertex gehört. Es wird ein Polygon mit Dichten erstellt, daraus werden die Vertexe bestimmt, deren Dichte nicht groß genug ist. Derartige Vertexe werden aus dem originalen Polygon entfernt. Im Skript werden alle Vertexe, die einen Dichtenwert kleiner als 0.2 haben, aus dem originalen Polygon entfernt. Alle mathematische Grundlagen für die beiden 3D-Modellierungsmethoden sind im Kapitel 5 beschrieben.Das erstellte Polygon wird in einer Datei mit dem Typ *.obj* in die gleichen Ordner wie Punktwolken gespeichert (Anforderung 16). Dieser Objekttyp ist Standard für die 3D-Objekte und kann mit unterschiedlichen Programmen gelesen werden. Der Pfad zu diesen Objekten wird in eine csv Datei hinzugefügt. Die Datei enthält zwei Spalten eine für den Pfad und die andere für den Objektnamen, der manuell in Datei hinzugefügt werden soll, zusätzlich beinhaltet die Datei alle bisher abgescannte Objekte zusammen mit Objektnamen.

4.6 Roboterarm Bewegung

Um das Objekt zu scannen, muss der Roboterarm sich um das Objekt herum bewegen. Das wird mit der richtigen Setzung von Koordinaten und der Orientierung von der UR5 Roboterarm Pose erreicht. Die Kreisbewegung ist von der aktuellen Gelenkposition des Arms, dem

Kreisradius und dem gewünschten Winkel, mit dem der Arm gedreht werden soll, abhängig. Die Ausrechnung von der Position erfolgt durch einfache trigonometrische Funktionen und wird mit folgenden Gleichungen berechnet:

$$XPos = x + cos(\alpha) * r \tag{4.1}$$

$$YPos = y + sin(\alpha) * r \tag{4.2}$$

In den Gleichungen 4.1 und 4.2 x und y Parameter sind Offsets von den aktuellen Gelenkkoordinaten. Die Variable r ist der Radius vom Bewegungskreis. α ist der Winkel im Radianten. Die z Koordinate wird leicht erhöht, damit das Objekt von oben auch angemessen untersucht ist. Bei der Implementierung vom Bewegungsskript wird z mit dem Wert 0.12 erhöht. Der Radius des Kreises wird auf den Wert 0.3 gesetzt.

Zusätzlich zu den Koordinaten muss auch die Orientierung des Gelenks richtig berechnet werden. Dies erfolgt mit einer Quaternion. Die Quaternion wird auf Basis vom angegebenen Winkel berechnet. Sie hat drei wichtige Parameter: roll, pitch und yaw. Neue Quaternion wird aus den Eulerwinkeln berechnet, dafür wird die Methode *quaterniuon_from_euler()* benutzt. Der Roboterarm besitzt vorne einen Greifer. Damit der Greifer keinen Einfluss auf die Kreisbewegung nimmt, wird der Arm 90 Grad in y Richtung gedreht. Um den Arm zu drehen, wird der roll Parameter auf den Wert PI/2 gesetzt. Zusätzlich wird dazu ein kleiner Offset von 0.35 addiert, damit das Objekt von oben besser gescannt werden kann. Der dritte Parameter yaw wird auf den Wert von a + P I gesetzt. In der Abbildung 4.11 ist ein kreisförmiger Bewegungsvorgang (rot) nach links gezeigt.

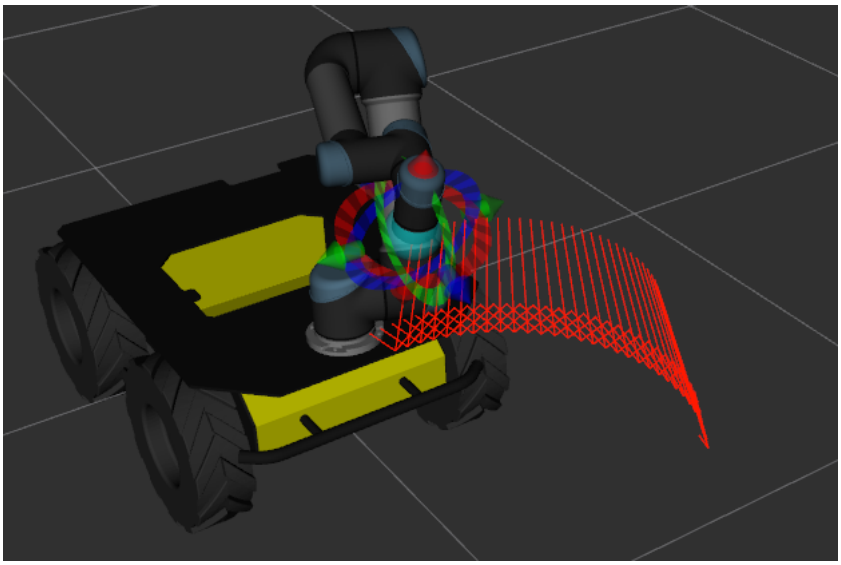


Abbildung 4.12: Die Darstellung vom Pfad des Roboters bei der Kreisbewegung

Es müssen aktuelle Gelenkpositionen, Radius des Kreises, Start- und Endewinkel gegeben werden, um die Bewegung durchführen zu können. Um Bilder für die Objekterkennung zu sammeln, werden drei unterschiedliche Winkeln in Radianten benutzt. Am Anfang ist es PI, damit wird der Roboter vor dem Objekt positioniert. Zweite Wert ist 0.5 * PI, damit wird der Roboter links vom Objekt positioniert. Beim nächsten Schritt bewegt sich der Roboter zurück zur Startposition und mit dem Wert 1.5 * PI wird er sich zur rechten Seite des Objekts bewegen. Für den Objektscan werden andere Winkelwerte genommen, damit das Objekt auch von hinten abgescannt werden kann. Der Wert bei der Bewegung nach links wird auf 0.3 * PI gesetzt, bei der Bewegung nach rechts beträgt der Winkel das Wert 1.7 * PI.

Um die Pose vom Roboterarm zu ändern wird auf die Nachricht *geometry_msgs.msg.PoseArray()* zugegriffen. Diese Nachricht wird modifiziert und mit ROS publiziert. Zusätzlich wird mithilfe der ROS Methode *compute_cartesian_path()*, die die Pose als Parameter bekommt, ein Bewegungsplan erstellt. Die Methode liefert auch den Wert, der zeigt auf wie viel Prozent der Plan ausgeführt wird. Nur wenn dieser Wert eine 1 liefert, wird der Plan ausgeführt. Der Bewegungsplan wird danach mit der Methode *execute()* ausgeführt.

Für die erfolgreiche Kreisbewegung muss sich das Objekt vor dem Roboter befinden. Der Roboter soll zuerst in eine Startpose fahren, damit die Bewegungsschritte erfolgreich abgeschlossen werden können. Wenn die Kreisbewegung mit der ausgewählten Startpose nicht durchführbar ist, muss die Startpose manuell geändert werden. Da das Objekt auch von hinten abgescannt werden soll, darf das Objekt nicht zu nahe an Wänden stehen (Anforderung 18)

5 Punktwolken Registrierung

Um eine vollständige dreidimensionale Repräsentation eines Modells zu erhalten, ist es notwendig Punktwolken aus verschiedenen Blickwinkeln zu erzeugen. Die Punktwolken sollen anschließend in ein gemeinsames Koordinatensystem überführt und zueinander ausgerichtet werden. Diese Zuordnung ist notwendig, um ein aufgenommenes Objekt vollständig und der Realität entsprechend repräsentieren zu können.

Beim Überführen von den Punktwolken in ein gemeinsames Koordinatensystem werden die erfassten Punktkoordinaten aller Punktwolken so transformiert, dass deren Punkte möglichst nah zueinander bewegt werden. Das heißt, dass die Skalierungen und Verzerrungen in den unterschiedlichen Punktwolken angeglichen werden müssen. Die von der Kamera gesammelten Punktwolken befinden sich bereits im gleichen Koordinatensystem und besitzen den gleichen Maßstab, weshalb eine Überführung nicht mehr notwendig ist. Das heißt aber nicht, dass die Punktwolken bereits richtig zusammengefügt sind.

Wenn ein Punkt einer Szene aus unterschiedlichen Positionen aufgenommen wird, tritt dieser in verschiedenen Punktwolken auf. Sind die Punktwolken nicht zueinander ausgerichtet, besitzt er in jeder Punktwolke eine andere Koordinate. Diese Punkte werden als korrespondierende Punkte oder Korrespondenzen bezeichnet. Das Finden von Korrespondenzen, sowie die richtige Verschiebung (Translation) und Verdrehung (Rotation) der Punktwolken, wird als Registrierung bezeichnet. Bei der Registrierung werden die korrespondierenden Punkte einer Punktwolke so transformiert, dass sie möglichst nah zu korrespondierenden Punkten in der anderen Punktwolke stehen. Im Rahmen dieser Arbeit wird der automatisierte Ansatz zum Lösen des Registrierungsproblems verwendet. Dieses Verfahren heißt Iterative Closest Point Algorithmus (ICP). In diesem Kapitel wird zuerst das Registrierungsproblem und danach zwei unterschiedliche Varianten von ICP beschrieben.

5.1 Registrierungsproblem

Es werden zwei Punktwolken gegeben P und Q. Diese repräsentieren einen Teil von dem zu modellierenden Objekt. Beide Punktwolken können mathematisch wie folgt definiert werden:

$$\begin{aligned} P &= \{p_i | p_i \in \mathbb{R}^3, 0 < i \leq n, n \in \mathbb{N}\} \\ Q &= \{q_i | q_i \in \mathbb{R}^3, 0 < i \leq m, m \in \mathbb{N}\} \end{aligned} \tag{5.1}$$

Die Transformation von den Punkten in beiden Punktwolken wird durch eine Rotationsmatrix R und einen Translationsvektor $\vec{t}$ beschrieben, wo

$$\begin{aligned} R &= R_x * R_y * R_z \\ R_x &= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix}, \alpha \in \mathbb{R}, 0 \leq \alpha < 360 \\ R_y &= \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 1 & \cos(\beta) \end{pmatrix}, \beta \in \mathbb{R}, 0 \leq \beta < 360 \\ R_z &= \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix}, \gamma \in \mathbb{R}, 0 \leq \gamma < 360 \\ \vec{t} &= \begin{pmatrix} t_x \\ t_y \\ t_z \end{pmatrix}, t_x, t_y, t_z \in \mathbb{R} \end{aligned} \tag{5.2}$$

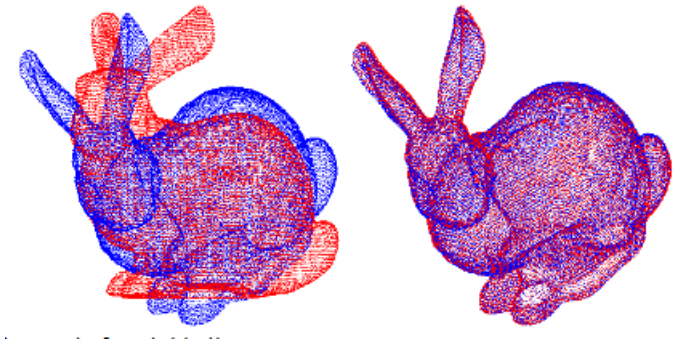


Abbildung 5.1: Die Visualisierung vom Punktwolkenproblem (links) und die Lösung dieses Problems (rechts) [10]

Bevor die Registrierung erfolgen kann, werden die Korrespondenzen zwischen zwei Punktwolken definiert. Die Korrespondenzen können als ein Paar von Punkten beider Punktwolken, deren Punktkoordinaten in dem zu untersuchenden Objekt möglichst identisch sein sollen, beschrieben werden. Mathematisch wird folgendes definiert:

$$\hat{c} = (c_p, c_q), c_p \in P, c_q \in Q \tag{5.3}$$

Die Variable $\hat{c}$ beschreibt ein Korrespondenzpaar von zwei Punkten. Das Registrierungsproblem wird in der Abbildung 5.1 visualisiert. Das Problem wird erst dann gelöst, wenn R und $\vec{t}$ so ausgerechnet sind, dass für die Menge von Korrespondenzen C gilt:

$$C = \{(c_{k_p}, c_{k_q}) | c_{k_p} = R \cdot c_{k_q} + \vec{t}, 1 \leq k \leq |C|\} \tag{5.4}$$

Da beide Punktwolken nicht identisch sind, kann die Gleichheit nicht erreicht werden. Um das Problem zu lösen, wird die Gleichung 5.4 um einen Grenzwert $\epsilon > 0$ erweitert [40].

$$C = \{(c_{k_p}, c_{k_q}) | \, \|c_{k_p} - R \cdot c_{k_q} + \vec{t}\|_2 \leq \epsilon, 1 \leq k \leq |C|\} \tag{5.5}$$

5.2 Iterative Closest Point (ICP)

In diesem Abschnitt werden zwei Varianten von ICP Algorithmen mathematisch vorgestellt. Eine klassische Variante, die im Jahr 1992 von Besl und McKay entwickelt war, zeigt die Registrierung von zwei Punktwolken auf Basis von der Punkt-zu-Punkt Korrespondenz ([41]). Die zweite Variante kommt von Chen und Medioni und wurde im Jahr 1991 entwickelt, dabei wird die Registrierung auf Basis von der Punkt-zu-Ebene Korrespondenz gezeigt ([25]).

5.2.1 ICP Punkt-zu-Punkt Registrierung

Bei der Punkt-zu-Punkt Registrierung werden zuerst die Korrespondenzpunkte bestimmt, dafür wird die euklidische Distanz zwischen einem Punkt in einer Quellenpunktwolke und in einer Zielpunktwolke bestimmt. Die Punkte mit kleinster Distanz werden als Korrespondenzpunkte bezeichnet.

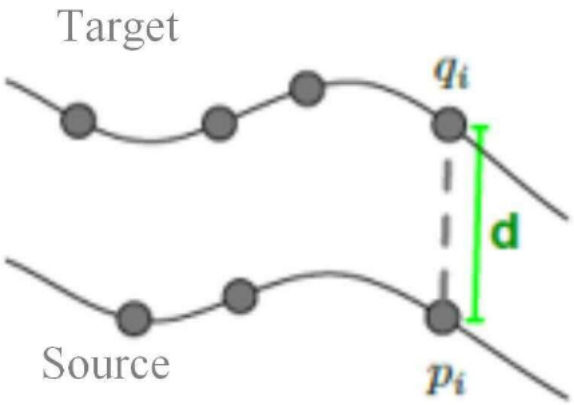


Abbildung 5.2: Visualisierung von Punktwolkenregistrierung mit Punkt-zu-Punkt ICP [11]

In der Abbildung 5.2 ist eine Punkt-zu-Punkt Registrierung visualisiert. Die Punkte q_i und p_i sind Korrespondenzpunkte und d ist die Distanz zwischen diesen Punkten. Um die Registrierung durchzuführen, werden Transformationsmatrizen bestimmt, die diese Distanz minimieren können.

Sei $\vec{r_1} = (x_1, y_1, z_1)$ und $\vec{r_2} = (x_2, y_2, z_2)$ Punkte in unterschiedlichen Punktwolken, dann kann die euklidische Distanz zwischen diesen Punkten so definiert werden:

$$d(\vec{r_1}, \vec{r_2}) = \|\vec{r_1} - \vec{r_2}\| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{5.6}$$

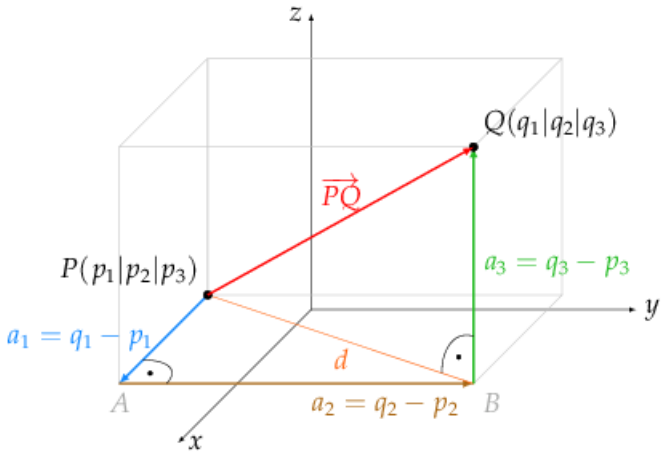


Abbildung 5.3: Darstellung von euklidischer Distanz in 3D-Ebene [12]

In der Abbildung 5.3 ist die euklidische Distanz in 3D visualisiert. Wenn es eine Punktwolke Q mit N_a Punkten gibt, mit $q_i \in Q$ und $i = 1....., N_a$, dann ist die Distanz zwischen dem Punkt $\vec{p}$ und der Punktwolke Q:

$$d(\vec{p}, Q) = \min_{i \in \{1,...,N_i\}} d(\vec{p}, \vec{q_i}) \tag{5.7}$$

Auf Basis der obenvorgestellten Formeln wird eine Menge von Korrespondenzpunkten erstellt. Die erstellte Menge wird für die Registrierung benutzt. Die Fehlerfunktion bei der Punkt-zu-Punkt Transformation ist:

$$E(T) = \sum_{(p,q) \in C} \|p - T(q)\|_2 \tag{5.8}$$

mit C als Menge der Korrespondenzpaare und $T_i(q) = R_i \cdot q + t_i$. Der Fehler E beschreibt, wie weit zwei Punktwolken von einer korrekten Ausrichtung entfernt sind. Die Tranformationsmatrix, die den kleinsten Fehler aufweist, wird auf eine Punktwolke angewendet. Der Algorithmus wird dann abgebrochen, wenn der Fehler E kleiner oder gleich dem vordefiniertem Grenzwert ist, oder wenn die maximale Anzahl von Iterationen erreicht wird.

Die Variablen R und t werden mit dem Algorithmus auf Basis von den Quoternionen ausgerechnet, der im Jahr 1987 von Horn beschrieben war. [42]

Der komplette Punkt-zu-Punkt ICP Algorithmus sieht wie folgt aus:

1. Berechnung von Korrespondenzpunkten

2. Bestimmung der Transformationsmatrix mit kleinstem Fehler E
3. Anwendung der Transformationsmatrix auf die Punktwolke $P_{k+1} = T(P_k)$
4. Wenn $E \leq \theta$, wo θ der vordefinierte Grenzwert ist, oder die Anzahl von Iterationen erreicht ist, wird der Algorithmus abgebrochen, sonst zurück zu 2.

5.2.2 Punkt-zu-Ebene Registrierung

Bei dieser Registrierungsmethode werden am Anfang wie in der Punkt-zu-Punkt Registrierung die Korrespondenzpunkte bestimmt. In diesem Verfahren für die Korrespondenz werden die Punkte ausgewählt, die sich in beiden Punktwolken auf möglichst glatter Fläche befinden. Dies erhöht die Wahrscheinlichkeit, dass es Überschneidungen zwischen der Normalen von einem Punkt in der Quellenpunktwolke und in der Zielpunktwolke gibt. Auf diesem Überschneidungspunkt wird die Tangente gebaut. Der korrespondierende Punkt wird mit dem Algorithmus so transformiert, dass der Abstand zwischen der Tangente und diesem Punkt möglichst klein ist [25]. Mathematisch kann dieses Problem so definiert werden:

$$E^k(T) = \sum_{i=1}^{N} d_s^2(T^k p_i, S_i^k) \tag{5.9}$$

In der Gleichung 5.9 steht T für die Transformationsmatrix bei der Iteration k, p_i ist ein Korrespondenzpunkt aus der Punktwolke P, d_s ist eine Distanz zwischen dem Punkt und der Ebene. Die Variable S_i^k wird wie folgt beschrieben:

$$S_i^k = \{s | n_{qi}^k \cdot (q_i^k - s) = 0\}$$

Diese Variable repräsentiert eine Tangente der Punktwolke Q im Punkt q_j^k. n_{qj}^k ist die Normale der Fläche Q aus dem Punkt q_j^k. Dieser Punkt ist Schnittpunkt von Q mit der Gerade $T^{k-1} l_i$

$$q_j^k = (T^{k-1} l_i) \cap Q$$

l_i ist eine Normale von p_i in der Punktwolke P

$$l_i = \{a | (p_i - a) \times n_{pi} = 0\}$$

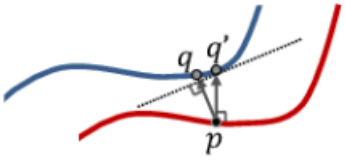


Abbildung 5.4: Visualisierung von Punktwolkenregistrierung mit Punkt-zu-Ebene Registrierung [13]

Die Punkt-zu-Ebene Registrierung kann in folgenden Schritten gemacht werden:

1. Bestimmung der Korrespondenzpunkten $p_i \in P(i = 1...\mathbb{N})$.
2. Bestimmung der Normalen in Korrespondenzpunkten.
3. Bei jeder Iteration k werden folgende Schritte wiederholt, bis der Fehler $E(T) \leq e$ ist (e ist ein selbst definierter Grenzwert), oder bis die Anzahl von Iterationen erreicht ist.
   - Für jeden Punkt und Normale die Transformationsmatrix T^{k-1} anwenden.
   - Den Überschneidungspunkt q_i^k zwischen Q und der Normalen finden.
   - Die Tangente S_i^k auf dem Überschneidungspunkt ausrechnen.
   - Suche nach T,welches den Fehler E in Gleichung 5.9 mithilfe der Methode der kleinsten Quadrate minimiert.

In diesem Projekt werden zwei unterschiedliche Registrierungsmethoden benutzt, eine davon nutzt genau solche Art von Punktwolkenregistrierung. Die andere Methode nutzt paarweise Registrierung auf Basis von Punkt-zu-Ebene Registrierung mit Pose Graph Optimierung.

5.2.3 Pose Graph Optimierung

Bei dieser Methode wird jede Punktwolke paarweise mit jeder anderen registriert. Die Registrierung erfolgt mit Punkt-zu-Ebene Algorithmus. Diese Art von Registrierung braucht eine zusätzliche Optimierung mit dem Pose Graph. Es wird eine Menge von Transformationen $\mathbb{T} = \{T_i\}$ erstellt, die Transformationen zwischen Punktwolkenpaaren aufweist. Bei der paarweise Registrierung besteht die Möglichkeit, dass die Registrierung von den weit entfernten Punktwolken zu falschen Transformationen führt, damit kann das gesamte Objekt falsch registriert werden. Die Kanten im Pose Graph werden in zwei Klassen unterteilt. Im Pose Graph

werden die Knoten als die Punktwolken repräsentiert und die Kanten sind die Transformationen zwischen diesen Punktwolken. Beim Aufbau vom Graph wird der Überlappungsbereich von den Punktwolkenpaaren beachtet. Wenn zwei Punktwolken nebeneinander stehen, wird die Transformationsmatrix von diesen Punktwolken als Odometry Kante bezeichnet. Die Transformationsmatrix, die zwei nicht nebeneinander stehende Punktwolken registrieren, gehört zu einer anderen Kantenklasse,der Schlaufenverschlusskantenklasse. Die Schlaufenverschlusskanten, die fehlerhaften Transformationen enthalten, werden aus dem Graph entfernt. Dieser Ansatz wurde im Jahr 2015 von Sungjoon Choi präsentiert ([26]). Um solche Kanten zu finden, wird ein Linienprozess $\mathbb{L} = l_{ij}$ beim Pose Graph eingesetzt. Um die falsch zusammengefügte Kanten zu finden werden die Informationsmatrizen benutzt. Die Open3D Software berechnet die Matrizen beim Aufruf der Methode $get\_information\_matrix\_from\_point\_clouds()$. Der Wertebereich von l_{ij} ist [0,1]. Wenn dieser Wert für die Kante groß ist, dann kann diese Kante als richtige definiert werden, und sie wird für die finale Registrierung benutzt. Bei der Optimierung werden alle Kanten mit Linienprozessgewicht $l_{ij} < \theta$, wo θ ein benutzerdefinierter Grenzwert ist, aus dem Graph entfernt. Diese Variable zeigt, ob eine Transformation, die zu entsprechender Schlaufenverschlusskante zugeordnet ist, eine korrekte Transformation ist, und ob sie die gesamte Objektregistrierung nicht stören wird. Die Optimierung erfolgt mithilfe des Levenberg-Marquardt Algorithmus. Die Optimierung des Pose Graph wird anhang von zwei Parameter Vektoren gemacht. Ein Vektor enthält die Knoten und Kanten des Pose Graph, der andere Vektor beinhaltet die liniaren Gewichte von allen Kanten. Die Optimierung erfolgt in zwei Phasen. Zuerst wird der Vektorparameter mit linearen Gewichten optimiert und die Kanten mit kleinem Gewicht werden aus dem Graph entfernt. In dieser Phase werden die Transformationen, die in den Kanten gespeichert sind, nicht benutzt und verändert. In der zweiten Phase werden die Kanten vom Pose Graph optimiert damit man eine noch bessere Registrierung von Punktwolken bekommt. Hier werden die Gewichte nicht mehr benutzt, für diese Phase werden die Transformationen von Punktwolken verändert damit die Registrierung bessere Ergebnisse bekommt.

Levenberg-Marquardt Algorithmus

Der Levenberg-Marquardt-Algorithmus, benannt nach Kenneth Levenberg und Donald Marquardt, ist ein numerischer Optimierungsalgorithmus zur Lösung nichtlinearer Ausgleichs-Probleme mit Hilfe der Methode der kleinsten Quadrate.[43] Dieses Verfahren kombiniert das Gauß-Newton-Verfahren mit dem Gradientenverfahren. Im Gegensatz zum Gauß-Newton-Verfahren nutzt dieser Algorithmus die zusätzliche Variableλ, die die Minimalisierung verbessert. Der Algorithmus ist Iterativ.

Sei f(x) eine nichtlineare Funktion mit $f(x) \in \mathbb{R}^m$ und $x \in \mathbb{R}^n$, die minimiert werden soll, gegeben. Es muss ein x gefunden werden, für welches die Funktion den minimalen Wert hat. Dafür muss folgende Gleichung gelöst werden:

$$\begin{aligned} x_{k+1} &= x_k - S_k \\ k &= 0, 1, 2, 3, ..., M \\ S_k &= (H_k + \lambda \cdot I)^{-1} \nabla f(x_k) \end{aligned} \tag{5.10}$$

H_k ist eine Hesse-Matrix von $f(x_k)$, mit $H_k \in \mathbb{R}^{n*n}$, ∇ ist ein Gradient von der Funktion $f(x_k)$. Die Variable S_k mit $S_k \in \mathbb{R}^n$ ist der Schritt, der hilft das nächste x, also x_{k+1} auszurechnen. Wenn nach der Iteration der neu berechnete Wert von der Funktion kleiner als der vorherige Funktionswert ist, wird λ für die nächste Iteration verkleinert. Hier sind die Ähnlichkeiten vom LMA zum Gauß-Newton-Verfahren gut zu erkennen. Wenn der neue Wert größer als vorheriger Wert ist, wird λ vergrößert - dieses Schritt weist eher Ähnlichkeiten zum Gradientverfahren auf. Seien $x \in \mathbb{R}^n$ und $f(x) \in \mathbb{R}^m$, denn geht die Algorithmus wie folgt vor:

1. Startwerte im Parametervektor x_0 und λ auswählen
2. Berechne $f(x_0)$
3. Berechne H_k und $\nabla f(x_k)$
4. Berechne S_k
5. x_{k+1} bestimmen
6. Wenn $|f(x_{k+1})| < |f(x_k)|$, dann λ wird verkleinert, ansonsten vergrößert.
7. Prüfen, ob die Funktion den vordefinierten Grenzwert erreicht hat. Wenn der Grenzwert erreicht ist, werden Iterationen abgebrochen und der aktuelle Wert für x wird als beste Wert zurückgegeben, sonst wird vom Punkt 2 wieder berechnet.

.

Um den Pose Graph zu optimieren wird die Fehlfunktion von folgender Art minimiert:

$$F(x) = \sum_{(i,j) \in C} e(x_i, x_j, z_{ij})^T \Omega_{ij} e(x_i, x_j, z_{ij}) \tag{5.11}$$

$$x^* = \arg\min_x F(x) \tag{5.12}$$

Die Variable x ist ein Parametervektor $x^T = (x_1^T, ..., x_n^T)$, wo x_i Knoten im Graph repräsentiert. Die Variable z_{ij} beschreibt die Transformation zwischen zwei Posen i und j. Mit anderen Wörtern z_{ij} ist eine Kante zwischen zwei Posen. Variable Ω ist eine Informationsmatrix der Transformation zwischen i und j. Die Funktion $e(x_i, x_j, z_{ij})$ ist eine Vektorfehlerfunktion, die misst, wie gut die Parameterblöcke x_i und x_J die Bedingung z_{ij} erfüllen. Diese Funktion beschreibt in Vektorform die Abweichung zwischen den Posen x_i und x_j nachdem x_i mit z_{ij} transformiert ist. Je kleiner der Wert von dieser Vektorfehlerfunktion ist, desto besser passen die Posen zusammen. Die Menge C ist eine Menge von Posen, die mit Kanten z_{ij} verbunden sind. Der Wert von x^* ist genau der Wert, der zu erreichen ist. Weiter wird die Vektorfehlerfunktion als $e(x_i, x_j, z_{ij}) = e(x)$ beschrieben. Genauso wie im oberen Beispiel wird die initiale Variable $\hat{x}$ ausgewählt. Die Idee ist, die Fehlerfunktion durch ihre Taylor-Approximation erster Ordnung um die aktuelle anfängliche Schätzung zu approximieren.

$$e_{ij}(\hat{x} + \triangle x) \simeq e_{ij}(\hat{x}) + J_{ij}\triangle x \tag{5.13}$$

J_{ij} ist eine Jacobi-Matrix von Funktion $e_{ij}(x)$ an der Stelle von $\hat{x}$. Die Matrix hat nur an den Stellen x_i und x_j, alle anderen Stellen haben dan Wert 0. Wenn die Gleichung 5.13 auf die Gleichung 5.11 angewendet wird, können mit der Jacobi-Matrix die Systemmatrix H und der Vektor b berechnet werden. Diese Variablen werden für die Optimierung benötigt.

$$F(\hat{x} + \triangle x) = (e_{ij}(\hat{x}) + J_{ij}\triangle x)^T \Omega_{ij}(e_{ij}(\hat{x}) + J_{ij}\triangle x) \tag{5.14}$$

$$= e_{ij}^T \Omega_{ij} e_{ij} + 2e_{ij}^T \Omega_{ij} J_{ij} \triangle x + \triangle x^T J_{ij}^T \Omega_{ij} J_{ij} \triangle x \tag{5.15}$$

$$H = \sum_{ij} H_{ij} = \sum_{ij} J_{ij}^T \Omega_{ij} J_{ij} \tag{5.16}$$

$$b = \sum_{ij} b_{ij} = \sum_{ij} e_{ij}(x)^T \Omega_{ij} J_{ij} \tag{5.17}$$

$$c = \sum_{ij} c_{ij} = e_{ij}^T \Omega_{ij} e_{ij} \tag{5.18}$$

Die Matrix H repräsentiert auch die Adjazenzmatrix vom Pose Graph. Alle nebeneinanderstehende Posen landen in der Diagonale von dieser Matrix und alle andere Werte können als Schlaufenverschlusskanten bezeichnet werden. Der Koeffizientvektor b wird an allen Stellen Nullen haben, außer an Stellen von x_i und x_j. Wenn man alle b Vektoren addiert, entsteht ein b Vektor, der für jeden Knoten im Graph einen Wert hat. Um die Gleichung 5.15 in x zu

minimieren, wird diese Gleichung nach $\triangle x$ abgeleitet und auf Null gesetzt. Die Ableitung wird danach nach $\triangle x$ aufgelöst, und daraus entsteht folgende Gleichung:

$$\triangle x = \frac{-b}{H} \tag{5.19}$$

$$x^* = \hat{x} + \triangle x \tag{5.20}$$

Die Variable $\triangle x$ kann man als S_k aus der Gleichung 5.10 bezeichnen.

Bei jeder Iteration werden der Vektor b, die Matrix H, $\triangle x$ und $\hat{x}$ neu berechnet, solange bis das Konvergenzkriterium nicht erfüllt wird.

Das oben beschriebene Verfahren wird auch bei der Software Open3D benutzt. Eine detaillierte Erklärung von der Optimierung wurde von Rainer Kümmerle im Jahr 2011 präsentiert ([44]). Das oben beschriebene Verfahren ist ein allgemeiner Ansatz zur Minimierung multivariater Funktionen. In der Erklärung wurde angenommen, dass der Raum des Parameters x euklidisch ist. Diese trifft für Pose Graphen jedoch häufig nicht zu. Die Parameterblöcke x können aus einem Translationsvektor t_i und einer Rotationskomponente α_i bestehen. Der Translationsvektor bildet einen euklidischen Raum, die Rotationskomponenten bilden aber keine euklidischen 2D- oder 3D-Räumen. Dadurch werden sie oft mithilfe von Rotationsmatrizen oder Quaternionen beschrieben. Diese Repräsentationen von Posen können nicht direkt für die Optimierung benutzt werden. Die Posen sollen in eine Vektorform umgewandelt werden. Das Inkrement $\triangle x$, das für das updaten der Posen benutzt wird, soll auch vor dem Update zur Form vom Parameterblock umgewandelt werden. Eine detaillierte Beschreibung von Optimierung von Pose Graphen mit nicht euklidischem Raum wurde vom Irvin Aloise and Giorgio Grisetti ([45]) präsentiert.

6 3D-Modell Erstellung

Nachdem die gesammelten Punktwolken mithilfe von dem oben erklärten Algorithmus zu einer großen Punktwolke registriert werden, sollen sie zu einem festen 3D-Mesh umgewandelt werden. Im Rahmen dieser Arbeit werden zwei unterschiedliche Algorithmen von 3D-Mesh Erstellung benutzt: Alpha Shapes und Poisson Surface Reconstruction.

6.1 Alpha Shapes Algorithmus

Der Alpha Shapes Algorithmus wurde zuerst von Herbert Edelsbrunner präsentiert. Die Alpha Shapes sind eine Verallgemeinerung der konvexen Hüllen eines Punktsatzes.

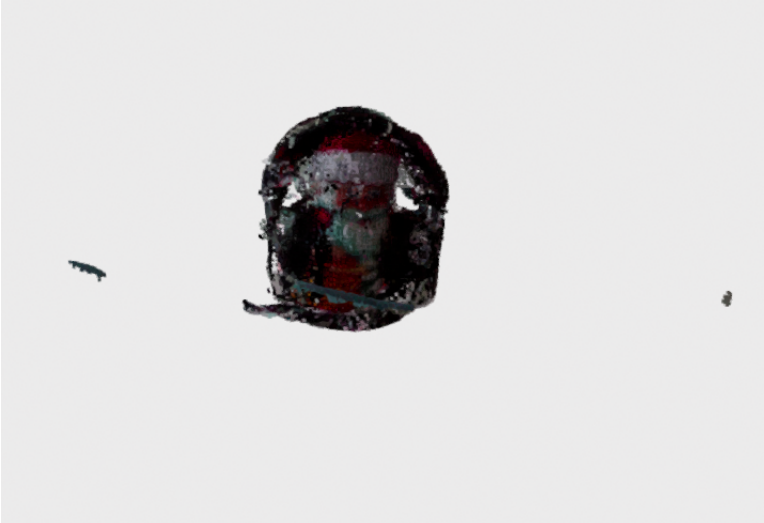


Abbildung 6.1: Komplett registrierte Punktwolke für weitere 3D-Modellierung

Sei S eine endliche Punktenmenge in $\mathbb{R}^3$ (Siehe Abbildung 6.1 und α ist eine Zahl mit $0 \leq \alpha \leq \infty$. Die α-Shape von S ist ein Polytop, der sowohl nicht unbedingt ein Konvex, als auch nicht unbedingt verbunden ist. Für $\alpha = \infty$ ist α-Shape eine Konvex-Hülle von Punktsatz S. [27]

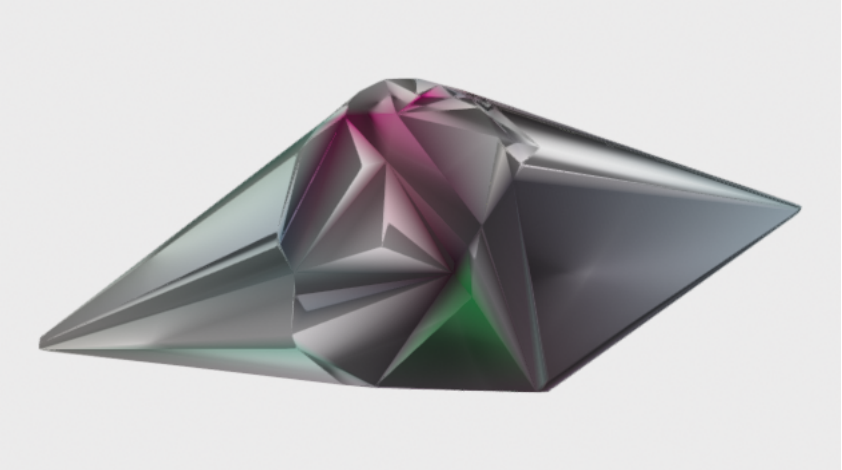


Abbildung 6.2: Darstellung von Konvex-Hülle von Punktwolke mit dem Alpha Shape Algorithmus und α=1

Mit abnehmendem α werden in den Shapes die Lochräume entwickelt. Diese Lochräume können die Löcher in dem 3D-Modell bilden. Einfacher gesagt, ein Stück vom Polytop kann verschwinden, wenn die sich bildenden Sphären mit dem Radius α keinen Punkt aus S beinhalten. Bei α = 0 besteht die α-Shape nur aus Punkten in der Punktmenge S.

Abbildung 6.3: Das Modell erstellt mit dem Alpha Shape Algorithmus und α=0.001, das die Lochräumen enthält

Beim Beispiel in der Abbildung 6.3 ist zu sehen, dass bereits bei Alpha von 0.001 im Modell eine große Anzahl von Lochräumen entsteht.

Bei der Erstellung eines 3D-Mesh mithilfe der Alpha Shape Methode müssen zuerst k-Simplices in der Punktmenge S definiert werden. Diese k-Simplices werden mithilfe von der Delaunay Triangulation miteinander mit Kanten verbunden. Danach wird ein Alpha Komplex erstellt und alle erstellten Alpha Komplexe repräsentieren das erwünschte 3D Mesh. Die k-Simplices sind eine beliebige Teilmenge T$\subseteq$ S mit der Größe $|T| = k + 1$, wo $0 \leq k \leq 3$, sie sind eine Konvex-Hülle von T. Es gibt einen besonderen Typ von k-Simplices, die α-exponiert heißen. Die k-Simplices heißen α-exponiert, wenn eine α-Sphäre existiert, die leer ist, und die Punkte aus der k-Simplices Menge an der Grenze von dieser Sphäre liegen. Das k für solche k-Simplices befindet sich im Bereich von 0 bis 2.

Abbildung 6.4: α-exponiert

Abbildung 6.5: Nicht α-exponiert

Abbildung 6.6: Darstellung von zwei Arten von k-Simplices [14]

Aus diesen α-exponierten k-Simplices entsteht die Grenze des resultierenden α-Shape. Es werden Mengen F_k definiert, die aus α-exponierten k-Simplices mit 0$\leq$k$\leq$2 entstehen. Insgesamt ist das α-Shape von der Punktmenge S ein Polytope, die Grenze von welchem aus den Dreiecken in der Menge $F_{2\alpha}$, den Kanten in der Menge $F_{1\alpha}$ und den Punkten (Vertecies) aus der Menge $F_{0\alpha}$ entsteht.

Der nächste wichtige Begriff für die α-Shape Methode ist die sogenannte Delaunay Triangulation, die nach dem russischen Geometer Boris Delaunay benannt ist. Die Triangulation im Kontext der α-Shape Methode kann wie folgt beschrieben werden: für $0 \leq k \leq 3$ wird die Menge von den k-Simplices F_k mit σ_t=conv(T) (steht für Konvexhülle), T $\subseteq$ S und $|T| = k + 1$ definiert, wo die k-Simplices α-exponiert sind. Für jedes Dreieck aus der Triangulation existiert also ein Kreis mit dem Radius α, auf dessen Grenzen die Dreieckspunkte liegen und in dem sich keine andere Punkte befinden. Dann ist die Delaunay Triangulation von S ein Simplizial-

komplex, definiert mit Tetraeder aus F_3, Dreiecken aus F_2, Kanten aus F_1 und Punkten aus F_0. Anschließend ist jede Facette der α-Shape ist ein Simplizial der Delaunay Triangulation der Punktmenge S. In einfachen Wörtern die Delaunay Triangulation ist ein Verfahren, in dem Punkte zu Dreiecken so vernetzt werden, dass innerhalb des Kreises auf dem die drei Dreieckspunkte liegen, keine anderen Punkte enthalten sind.

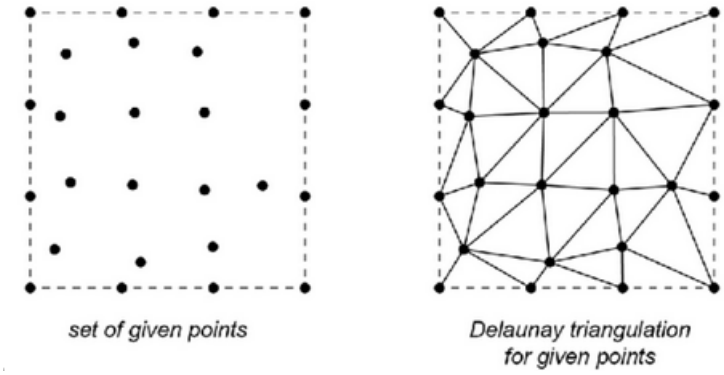


Abbildung 6.7: Nicht triangulierte Punktmenge (links) Delaunay Triangulation von dieser Punktmenge [15]

Die letzte Komponente der α-Shape Methode ist die Erstellung von α-Komplexen. Diese Komplexen repräsentieren anschließend das gewünschte 3D-Modell. Da alle Facetten von α-Shape Teile von Simplizialkomplex aus Delaunay Triangulation sind, ist der Innenraum von α-Shape trianguliert bei Tetraeder von diesem Simplizialkomplex. Das α-Komplex ist also ein Teilkomplex der Delaunay Triangulation. Der α-Komplex $\mathbb{C}$ ist eine Sammlung von geschlossenen k-Simplices mit $0 \leq k \leq 3$, die zwei Bedingungen erfüllen:

1. Wenn k-Simplex $\in \mathbb{C}$, dann enthält $\mathbb{C}$ alle Facetten, die auch zum k-Simplex gehören.
2. Wenn der Umkreis des k-Simplex mit dem Radius r< α keine weiteren Punkte aus der Punktmenge beinhaltet, dann gehört dieser k-Simplex zu α-Shape.

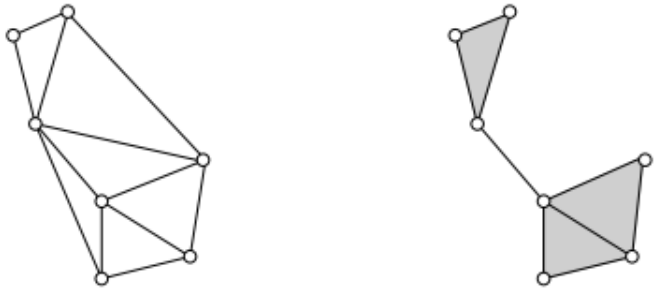


Abbildung 6.8: Beispiel für die Erstellung eines α-Komplexes aus triangulierten Simplices [14]

In der Abbildung 6.8 ist ein Beispiel eines α-Komplexes gezeigt, links sind die Delaunay Triangulation Mengen und rechts sind die Komplexen, die aus dieser Triangulation stammen.

Sei zu jedem k-Simplex aus dem Simplizialkomplex der Delaunay Triangulation ein Ball *b* mit Radius *r* zugeordnet. Die Punkte vom Simplex liegen auf den Grenzen des Balls. Für $1 \leq k \leq 3$ und $0 \leq \alpha \leq \infty$ kann die Menge $G_{k,\alpha}$ definiert werden. Die Menge enthält k-Simplices aus Simplizialkomplex, für welchen der Ball *b* leer ist, und $r \leq \alpha$. Mit diesem Wissen der α-Komplex von S, als $\mathbb{C}_\alpha$ bezeichnet, ist ein Simplizialkomplex, wessen k-Simplices entweder in der Menge $G_{k,\alpha}$ sind, oder sie (k+1)-simplices von $\mathbb{C}_\alpha$ verbinden. Die unterliegende Fläche von $\mathbb{C}_\alpha$ ist eine Vereinigung von allen Simplices von $\mathbb{C}_\alpha$. Diese Vereinigung ist auch die gewünschte Polytope von dem 3D-Objekt.

Abbildung 6.9: 3D-Modell erstellt vom Alpha Shape Algorithmus mit α = 0.005

In der Abbildung 6.9 ist eine erfolgreiche Alpha Shape Rekonstruktion zu sehen. Die Rekonstruktion wurde auf Basis der Punktwolke aus der Abbildung 6.1 erstellt. Es ist zu erkennen, dass bei diesem Verfahren im Modell viele Defekte entstehen. Um sie zu entfernen, müssen extra Optimierungen auf das Polygon angewendet werden.

6.2 Poisson Surface Rekonstruktion

Diese Methode der 3D-Objekterstellung nutzt eine implizite Funktion. Anhang von Resultaten einer 3D-Indikatorfunktion $\mathbb{X} : \mathbb{R}^3 \rightarrow \{0, 1\}$ wird zwischen zwei Werten unterschieden: Wert 1 für Punkte innerhalb der 3D-Objektfläche und 0 für die Punkte außerhalb des 3D-Objekts (Abbildung 6.10). Das rekonstruierte Objekt wird durch Extrahieren einer geeigneten Isofläche erhalten.

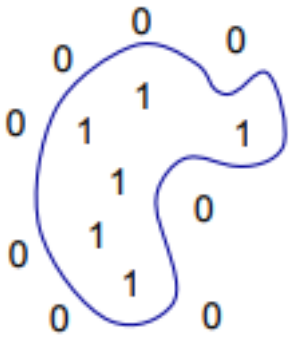


Abbildung 6.10: Die Visualisierung von Werten der Indikatorfunktion innerhalb und außerhalb des 3D-Objekts [16]

Der Gradient der Indikatorfunktion ist ein Vektorfeld, der gleich 0 überall, außer in den Punkten neben der Fläche, ist. Das Hauptproblem dieses Ansatzes ist, eine Funktion $\mathbb{X}$ zu finden, deren Gradient am besten zum Vektorfeld von Punkten approximiert. Wenn auf diesem Gradienten der Divergenzoperator angewendet wird, wird das Problem zu dem Standard Poisson Problem umgewandelt. Es muss eine Scalarfunktion $\mathbb{X}$ berechnet werden, deren Divergenz vom Gradienten gleich zur Divergenz vom Vektorfeld $\vec{V}$ ist mit $\vec{V} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$. Das Vektorfeld $\vec{V}$ ist mit Normalen von jedem Punkt in der Punktwolke definiert, und ist nicht Null neben diesen Punkten.

$$\triangle \mathbb{X} = \nabla \vec{V} \tag{6.1}$$

Um so eine Rekonstruktion durchzuführen, wird eine Punktwolke S mit Punkten $p \in \mathbb{R}^3$ benötigt, für diese Punkte werden die Normalen $\vec{N}$ ausgerechnet. Angenommen liegen diese Normalen auf oder neben einer Oberfläche von unbekanntem Modell M. Das Ziel der Methode ist die triangulierte Annäherung an die Oberfläche durch eine Approximation der Indikatorfunktion des Modells und durch Extrahieren der Isofläche.

Um die Rekonstruktion erfolgreich durchführen zu können, muss die Indikatorfunktion eine genaue Darstellung in der Nähe der rekonstruierten Oberfläche haben. Die Werte von der Indikatorfunktion, die sich weit entfernt von der Objektoberfläche befinden, können vernachlässigt werden. Dafür wird ein Octree gebaut, der sowohl für die Indikatorfunktion, als auch für die Lösung des Poisson Problems benutzt wird. Um den Octree zu bauen, werden die Positionen der Punkte aus den Punktwolken benutzt. Zu jedem Knoten im Octree wird eine Funktion F_o zugeordnet. Für die Auswahl des Baumes und der Funktion müssen folgende Bedingungen erfüllt werden:

1. Das Vektorfeld $\vec{V}$ kann effizient mit der Summe der Funktionen F_o repräsentiert werden.
2. Die Indikatorfunktion, die als Summe der Funktionen F_o dargestellt ist, soll nahe zur Objektoberfläche präzise und effizient ausgewertet werden.
3. Das Poisson Problem, das mit Funktionen F_o ausgedrückt ist, kann effizient gelöst werden.

Der Octree wird mithilfe von der Punktmenge S aus der Punktwolke und maximaler Tiefe des Baums D definiert. Jeder Punkt aus der Menge wird beim Octree in der Tiefe D in einen Blattknoten fallen. Mithilfe der Knoten im Octree wird eine Basisfunktion $F : \mathbb{R}^3 \to \mathbb{R}$ definiert. Zu jedem Knoten wird ein Einheitsintegral "Knotenfunktion"definiert:

$$F_o(q) \equiv F(\frac{q - o.c}{o.w})\frac{1}{o.w^3} \tag{6.2}$$

Die Variablen o.c und o.w repräsentieren das Zentrum und die Breite von dem Knoten o, wo o.c $\in \mathbb{R}^3$ und 0.w $\in \mathbb{R}$ sind. Die Variable q ist ein Punkt aus der Punktmenge. Die Menge von diesen Funktionen bildet einen Funktionenraum $\mathcal{F}_{\mathcal{O},F}$. Die Funktion F(q) ist eine Basisfunktion, definiert als n-te Faltung des Box Filters mit sich selbst:

$$F(x, y, z) \equiv (B(x)B(y)B(z))^{*n} \tag{6.3}$$

$$B(t) = \begin{cases} 1, & |t| < 0.5 \\ 0, & sonst \end{cases}$$

In Open3D und Kazhdan Implementierungen wird n = 3 gesetzt ([16]).

Nachdem die Basisfunktionen für jeden Knoten definiert sind, soll ein Vektorfeld definiert werden. In dem Ansatz, den die Open3D Implementierung nutzt, wird dafür die trilineare Interpolation benutzt, damit wird die Approximation zum Gradientfeld von der Indikatorfunktion so beschrieben:

$$\vec{V}(q) \equiv \sum_{s \in S} \sum_{o \in Ngbr_D(s)} a_{o.s} F_o(q) s.\vec{N} \tag{6.4}$$

die $Ngbr_D(s)$ sind acht benachbarten D-Tiefe Knoten, die sich am nächsten zu s.p befinden, wo s.p ein Punkt aus der Punktwolke ist, die $a_{o.s}$ sind die trilinear interpolierte Gewichte. $s.\vec{N}$ ist eine Normale vom Punkt s.

Der nächste Schritt ist das Poisson Problem zu formulieren und zu lösen. Zuerst muss sichergestellt werden, dass die Funktionen sich $\triangle \mathbb{X}$ und $\nabla \vec{V}$ in eine Raum $\mathcal{F}_{\mathcal{O},F}$ befinden. Für die Lösung dieses Problems muss für die Funktion $\mathbb{X}$ eine Projektion von $\triangle \mathbb{X}$ auf den Raum $\mathcal{F}_{\mathcal{O},F}$ gefunden werden, die am nächsten zur Projektion von $\nabla \vec{V}$ ist. Das Problem kann vereinfacht werden, wenn für die Lösung auch noch der Octree mit Funktionen F_o benutzt wird. Damit wird das Problem bei der Kazhdan Implementierung auf die Minimierung folgender Gleichung vereinfacht:

$$\sum_{o \in \mathcal{O}} ||\langle \triangle \mathbb{X} - \nabla \vec{V}, F_o \rangle||^2 = \sum_{o \in \mathcal{O}} ||\langle \triangle \mathbb{X}, F_o \rangle - \langle \nabla \vec{V}, F_o \rangle||^2 \tag{6.5}$$

Für dieses Problem ist ein $|\mathcal{O}|$-dimensionaler Vektor $\mathcal{V}$, wo $|\mathcal{O}|$ die Größe des Baums ist, gegeben. Die Koordinaten von diesem Vektor an der Stelle o sind $\mathcal{V} = \langle \nabla \vec{V}; F_o \rangle$. Es ist eine Funktion $\mathbb{X}$ zu finden, für welche der Vektor, der durch die Projektion des Laplacewertes von $\mathbb{X}$ auf jedes F_o entsteht, so nah wie möglich an $\mathcal{V}$ liegt.

Dies kann auch in Matrizen ausgedrückt werden. Die Gleichung $\mathbb{X} = \sum_o x_o F_o$ wird nach $x \in \mathbb{R}^{|\mathcal{O}|}$ aufgelöst. Danach wird Matrix L mit Dimensionen $|\mathcal{O}| \times |\mathcal{O}|$ definiert und zwar so, dass für alle Werte $o, o' \in \mathcal{O}$ die Stelle (o, o') in der Matrix L wie folgt aussehen wird:

$$L_{o,o'} = \langle \frac{\partial^2 F_o}{\partial x^2}, F_{o'} \rangle + \langle \frac{\partial^2 F_o}{\partial y^2}, F_{o'} \rangle + \langle \frac{\partial^2 F_o}{\partial z^2}, F_{o'} \rangle \tag{6.6}$$

Genau zwischen diesem Skalarprodukt und dem Vektor $\mathcal{V}$ soll der minimale Wert gefunden werden. Die Lösung von $\mathbb{X}$ bezieht sich auf die Lösung von folgender Gleichung:

$$\min_{x \in \mathbb{R}^{|\mathcal{O}|}} ||Lx - \mathcal{V}||^2 \tag{6.7}$$

,

Der letzte Schritt bei der Erstellung vom 3D-Objekt mit Poisson Rekonstruktion ist das Ausschneiden der Isosurface mit dem Objekt aus dem gesamten Modell. Dafür wird ein Isovalue ermittelt. Danach wird die benötigte Isosurface der ausgerechneten Indikatorfunktion extrahiert. So wird ein Isovalue gewählt, dass die extrahierte Oberfläche am nächsten zu den Positionen von Punkten aus den Punktwolken ist. Dazu wird $\mathbb{X}$ an den Positionen von diesen Punkten bewertet. Für das Extrahieren der Fläche wird der Durchschnitt der Werte verwendet.

$$\delta M \equiv \{q \in \mathbb{R}^3 | \mathbb{X}(q) = \gamma\} \text{ with } \gamma = \frac{1}{|S|} \sum_{s \in S} \mathbb{X}(s.p) \tag{6.8}$$

Diese Wahl des Isovalue hat einen Vorteil, weil die Isosurface, die mit dieser Isovalue extrahiert wird, unabhängig von der $\mathbb{X}$ Skalierung ist. Das Extrahieren der Isosurface erfolgt mit dem Marching Cubes Algorithmus ([46]).

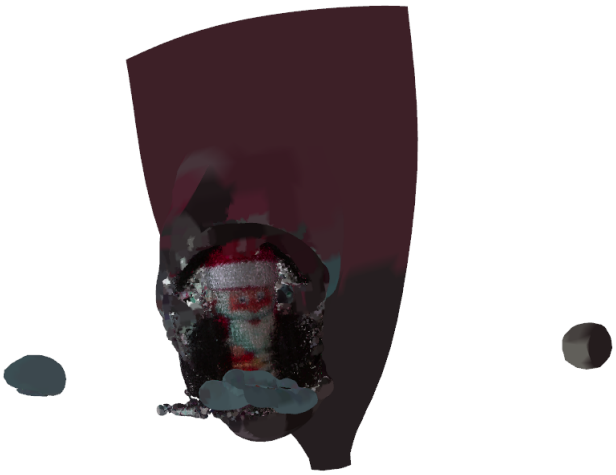


Abbildung 6.11: IsoSpace vom Poisson Modell bevor die Punktdichten berechnet wurden

Nachdem die Isofläche extrahiert ist, können in dem erstellten Objekt ungenaue Stellen und Defekte vorkommen. Die Defekte können an solchen Stellen entstehen, wo eigentlich keine Fläche sein soll und sich nur ganz wenige Punkte aus Punktwolken befinden. Diese Stellen heißen die Stellen mit kleiner Punktdichte, sie werden aus dem erstellten 3D-Objekt auch extrahiert. Die kleine Punktdichte bedeutet, dass das Vertex aus dem Objekt nur mit einer geringeren Anzahl von Punkten aus der Punktwolke unterstützt ist. Die Open3D Implementie-

rung von dieser Rekonstruktionsmethode liefert auch die Punktdichte zurück. Mit den Werten der Punktdichten kann auch ein weiteres 3D-Modell erstellt werden, siehe Abbildung 6.12

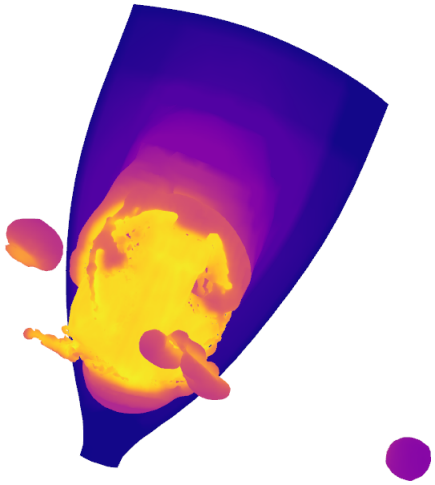


Abbildung 6.12: Poisson Modell mit dargestellten Punktdichten

Nachdem alle nicht gewünschten Vertexe extrahiert sind, kann mit dem oben genannten Beispiel aus der Abbildung 6.13 folgendes 3D-Modell entstehen:

Abbildung 6.13: Poisson Modell ohne Punkten mit kleinen Punktdichten

7 Auswertung

In diesem Kapitel geht es um die Resultate und die Produktivität von implementierten Registrierungsmethoden, 3D-Objektmodellierungsmethoden und dem gesamten System. Um den Prozess von der Untersuchung und Erkennung zu starten, wurden Server und Client mit allen benötigten ROS Knoten gestartet. Server schickt die Nachricht Start an Client. Wenn diese Nachricht an Client ankommt, wird weitere Interaktion mit dem System nicht nötig (Anforderung 3). Laut den Anforderungen, die in dem Kapitel 2 präsentiert wurden, soll das System parametrisierbar sein. Beim Start vom Server wurden die Parameter für die 3D-Modellierung an das System angegeben. Damit wurden die benötigten Algorithmen ausgewählt und die Parameter für Alpha-Shape und Poisson Surface Rekonstruktion definiert (Anforderung 1). Die zu untersuchenden Objekte wurden mit dem neuronalen Netz klassifiziert (Anforderung 7, Anforderung 9). Dafür werden zuerst drei Bilder vom Objekt aus unterschiedlichen Perspektiven aufgenommen (Anforderung 8). Alle ausgewählten Objekte wurden vom Netz nicht erkannt. Diese Auswahl von Objekten wurde bewusst gemacht, weil es nur unerkannte Objekte mit dargestelltem System untersucht und modelliert werden sollen (Anforderung 10, Anforderung 11). Alle Objekte befinden sich nicht neben der Wand, damit der Scan erfolgreich durchgeführt werden kann (Anforderung 18). Für den Scan wird der Roboterarm mit der Tiefenkamera vor dem Objekt positioniert. Der Roboterarm mit der Kamera wird sich um das Objekt herumbewegen (Anforderung 13). Damit werden die Punktwolken und Fotos von den unterschiedlichen Seiten vom Objekt gesammelt und für die weitere Verarbeitung in separaten Ordnern gespeichert. Die Untersuchung dauert im Durchschnitt 40 bis 50 Sekunden. Alle Bilder und Punktwolken werden in separaten Ordnern gespeichert (Anforderung 12). Insgesamt wurden sechs Objekte für die Tests ausgewählt. Vier davon wurden mit dem Roboter untersucht, für die anderen zwei Objekte wurde die Untersuchung ohne den Roboter gemacht, die Bewegung um das Objekt wurde manuell durchgeführt. In der Abbildung 7.1 sind die zu untersuchenden Objekte visualisiert.

Schachtel

Weihnachtsmann

Flasche

Osterhase

Stofftier

Schuh

Abbildung 7.1: Fotos von Objekten die für Tests benutzt wurden

Nachdem die Untersuchung abgeschlossen wurde, wurden die gesammelten Punktwolken zusammen in eine große Punktwolke registriert (Anforderung 14). Zuerst wurde die Registrierungsmethode mit paarweiser Registrierung und Pose Graph Optimierung angewendet,

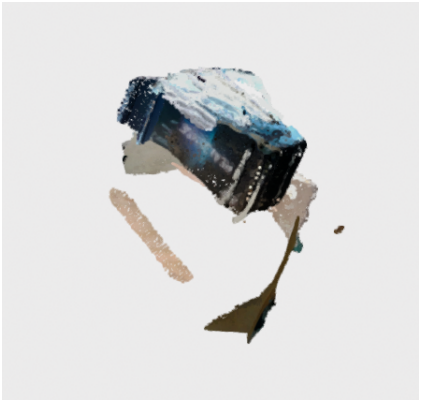
und im zweiten Versuch wurden die Punktwolken mit der Methode ohne Pose Graph Optimierung und paarweiser Registrierung erfasst. In dem Algorithmus ohne Optimierung beträgt die Anzahl von Iterationen des ICP Algorithmus 700. Im anderen Algorithmus ist die Anzahl von Iterationen 30. Im nächsten Schritt wurde aus der registrierten Punktwolke ein festes 3D-Objekt erstellt (Anforderung 15). Es wurden zwei 3D-Objekte aus der Punktwolke erstellt: eines mit Alpha Shape Algorithmus und ein anderes mit Poisson Surface Rekonstruktion. Der Parameter Alpha wurde auf den Wert 0.007 gesetzt, Tiefe des Baums bei der Poisson Surface Rekonstruktion beträgt 9. Sowohl für die Registrierung als auch für die 3D-Modellerstellung wurde ein Rechner NVIDIA Jetson AGX Xavier benutzt, der im Kapitel 2 beschrieben wurde. Auf diesem Rechner werden die Bilder, Punktwolken und erstellte 3D-Objekte gespeichert (Anforderung 6). Der Pfad zu erstellten 3D-Objekten wird zu einer csv Datei hinzugefügt (Anforderung 17).

Das erste untersuchte Objekt war eine Schachtel. Nach der Untersuchung entstanden 51 Punktwolken. (Beispiele in der Abbildung 7.2). Die erstellten Punktwolken visualisieren das Objekt und es ist kein Hintergrund zu sehen (Anforderung 4). Um das zu erreichen, wurde die Tiefenkamerasicht auf 40cm gesetzt (Anforderung 5). Das heißt, es wurden keine Punkte gebildet, die sich weiter als 40cm von der Kamera befinden. Diese Einstellung wurde auch für andere Tests übernommen.

Abbildung 7.2: Drei der Punktwolken, die nach der Untersuchung von der Schachtel gespeichert wurden

Die Registrierung dieser Punktwolken mit der Optimierung braucht ungefähr 267 Sekunden. Die daraus entstandene Punktwolke ist in der Abbildung 7.3 (links) gezeigt. Der Algorithmus ohne Optimierung brauchte 170 Sekunden, das Ergebnis ist auch in der Abbildung 7.3 (rechts) zu sehen.

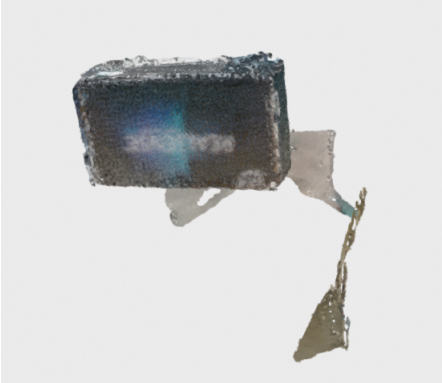
Optimierter Registrierung



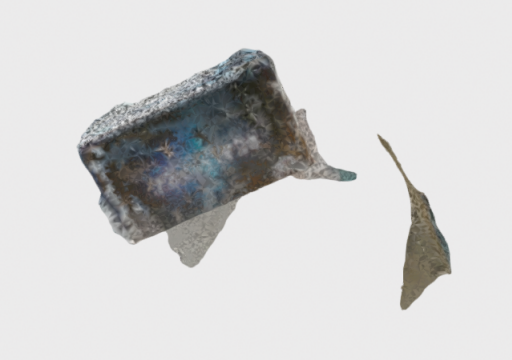
Registrierung ohne Optimierung

Abbildung 7.3: Registrierung vom Objekt Schachtel

Es ist zu sehen, dass dieses Objekt nicht ohne die Punktwolkenoptimierung vollständig modelliert werden kann. Die weitere Anmerkung zu den Resultaten ist die Zeit. Der Algorithmus ohne Optimierung war ungefähr 94 Sekunden schneller als der Algorithmus mit Optimierung. Wegen vergleichbar schlechterer Registrierung wurde für die Polygonerstellung trotzdem die optimierte Punktwolke benutzt. Die 3D-Polygone von dieser Punktwolke sind in der Abbildung 7.4 zu finden.



Schachtel erstellt mit Poisson Rekonstruktion



Schachtel erstellt mit Alpha Shape

Abbildung 7.4: 3D-Objekte, die mit der Schachtelpunktwolke erstellt wurden

Der zweite Test wurde mit der kleinen Weihnachtsmannfigur mit den Kopfhörern darauf gemacht. Im Gegensatz zur Schachtel sieht dieses Objekt von allen Seiten unterschiedlich aus. (Siehe Abbildung 7.1). Die Anzahl von Punktwolken nach der Untersuchung beträgt 35. In der Abbildung 7.5 sind manche Punktwolken gezeigt.

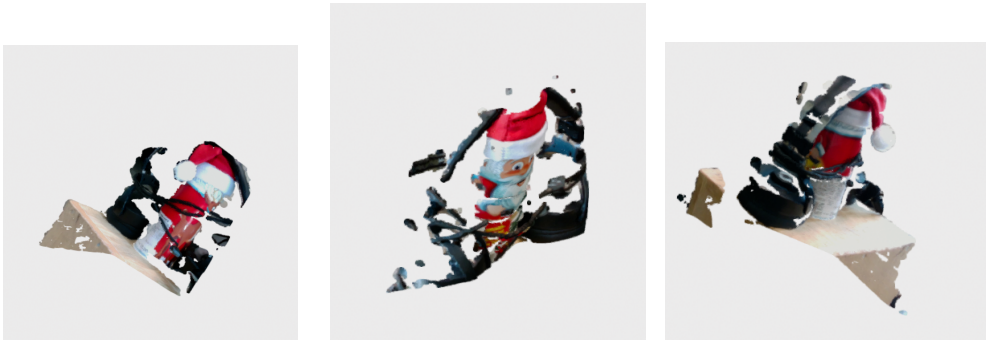
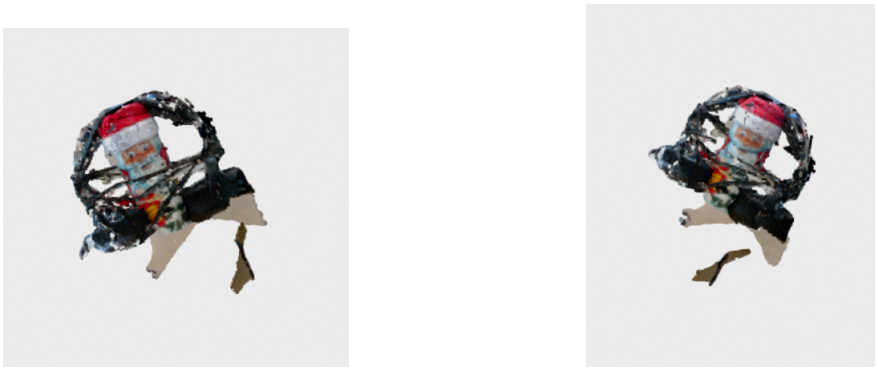


Abbildung 7.5: Punktwolken, die nach der Untersuchung vom Weihnachtsmann gespeichert wurden

Diese Punktwolken wurden wieder mit zwei Algorithmen registriert. Der Algorithmus mit der Optimierung braucht für die Registrierung ungefähr 150 Sekunden. Die Registrierung ohne Optimierung erfolgt aber nur in ungefähr 30 Sekunden, was offensichtlich schneller ist. Die registrierten Punktwolken sind in der Abbildung 7.6 zu sehen.



Die Punktwolke erstellt mit der Registrierung mit Optimierung

Die Punktwolke erstellt mit der Registrierung ohne Optimierung

Abbildung 7.6: Registrierte Punktwolke vom Weihnachtsmann

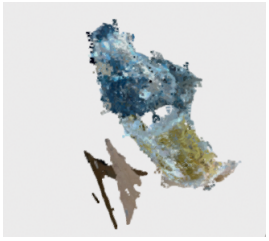
Beide Registrierungsmethoden liefern ähnliche Resultate. Der Algorithmus ohne Optimierung ist aber fünfmal schneller als der Algorithmus mit Pose Graph Optimierung. Deswegen wird die Behauptung aufgestellt, dass in den meisten Fällen für nicht symmetrische Objekte, die Optimierung und paarweise Registrierung nicht notwendig sind. Dies ist aber auch von der Qualität der gesammelten Punktwolken und von den Koordinaten der Punkte abhängig. Die 3D-Objekte, die während dieses Tests entstanden sind, wurden in der Abbildung 7.7 dargestellt.

Weihnachtsmann 3D-Objekt erstellt mit Poisson Rekonstruktion

Weihnachtsmann 3D-Objekt erstellt mit Alpha Shape

Abbildung 7.7: 3D-Objekte aus der Punktwolke vom Weihnachtsmann

Für den nächsten Test wurde eine transparente runde Flasche modelliert. Die Registrierungen haben 175 Sekunden für den Algorithmus mit Optimierung und 66 Sekunden für den Algorithmus ohne Optimierung gedauert. Die Resultate sind in der Abbildung 7.8 zu sehen.



Flaschepunktwolke, die mit der Registrierung mit Optimierung erstellt wurde

Flaschepunktwolke, die mit der Registrierung ohne Optimierung erstellt wurde

Abbildung 7.8: Flasche, die als registrierte Punktwolke dargestellt wird

Mit diesem Test wurde gezeigt, dass in dieser Arbeit verwendeten Registrierungsansätze für transparente oder symmetrische Objekte nicht anwendbar sind. Bei transparenten Objekten wird die Entfernung zum Objekt falsch berechnet, und die Punktwolken bilden die Oberfläche des Objekts nicht korrekt ab. Die Registrierung von diesen Punktwolken führt zu einem komplett unvollständigen Modell des Objekts. Bei symmetrischen Objekten entsteht das Problem bei den Koordinaten der Punkte in den Punktwolken. Die Koordinaten werden immer in Bezug auf das Kamera-Koordinatensystem berechnet und werden dadurch für viele durch die Untersuchung gesammelten Punktwolken ähnlich sein. Bei der Registrierung werden deswegen alle Punktwolken einfach aufeinander registriert und eine symmetrische Form wird

nicht erreicht. Das Objekt wird trotz der Untersuchung von allen Seiten nur von einer Seite modelliert.

Das letzte Objekt, das mithilfe des Greifarms untersucht wurde, ist der Osterhase (Abbildung 7.1). Die Registrierung wurde mithilfe von 59 gesammelten Punktwolken gemacht. Der Ansatz mit Pose Graph Optimierung brauchte für die Registrierung 371 Sekunden, der Ansatz ohne Optimierung brauchte 140 Sekunden. Zwei registrierte Objekte sind in der Abbildung 7.9 zu sehen.

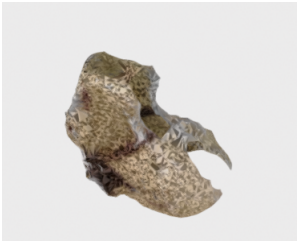
Osterhase registriert mit Optimierung

Osterhase registriert ohne Optimierung

Abbildung 7.9: Osterhasepunktwolke, die nach der Registrierung entsteht

Es ist zu erkennen, dass im Objekt ein paar Punkte im hinteren Teil fehlen. In diesem Teil ist das Objekt schmal und deswegen fehlen bei der Registrierung an dieser Stelle einige Registrierungspunkte. Die 3D-Polygone sind in der Abbildung 7.10 visualisiert.

Osterhase 3D-Modell, das mit Poisson Rekonstruktion erstellt wurde.



Osterhase 3D-Modell, das mit Alpha Shape erstellt wurde

Abbildung 7.10: 3D-Objekte aus Osterhasepunktwolke

Bei dem Polygon, das mit der Poisson Surface Rekonstruktion erstellt wurde, ist der Mangel an Punkten deutlich erkennbar. Bei dem Polygon, das mit dem Alpha Shape Algorithmus erstellt wurde, kann das Alpha so gestellt werden, dass die fehlenden Punkte nicht mehr erkennbar sind. Dies kann zu schlechterer Qualität des 3D-Polygons führen.

Die letzten zwei Objekte Stofftier und Schuh wurden ohne UR5-Greifarm modelliert. Die Untersuchung erfolgte manuell mit der Handbewegung. Für das Stofftier wurden 70 Punktwolken gesammelt. Für die Modellierung von dem Schuh wurden 58 Punktwolken gespeichert. Die Ergebnisse von beiden bereits registrierten Punktwolken sind in der Abbildung 7.11 visualisiert.

Stofftier, das mit Optimierung registriert wurde

Stofftier, das ohne Optimierung registriert wurde

Schuh, der ohne Optimierung erstellt wurde

Schuh, der mit Optimierung erstellt wurde

Abbildung 7.11: Registrierte Punktwolken für Schuh und Stofftier

Die Stofftier-Punktwolkenregistrierung mit Optimierung dauerte 341 Sekunden, der andere Ansatz brauchte nur ungefähr 106 Sekunden. Für den Schuh brauchte der Ansatz mit Pose Graph Optimierung 300 Sekunden, der andere Algorithmus hat 104 Sekunden benötigt. Für den Algorithmus ohne paarweiser Registrierung und Pose Graph Optimierung ist diese Untersuchungsart allerdings nicht geeignet, weil mit der Hand zu große Unterschiede zwischen Punktkoordinaten bei den Punktwolken entstehen können. Zusätzlich ist die Geschwindigkeit, anderes als beim Roboter, bei der manuellen Untersuchung nicht konstant. Beide Objekte

wurden aber erfolgreich mit dem Pose Graph Optimierungsalgorithmus registriert und zu 3D-Polygonen umgewandelt (siehe Abbildung 7.12).

Stofftier 3D-Modell mit Poisson Rekonstruktion

Schuh 3D-Modell Poisson Rekonstruktion

Abbildung 7.12: 3D-Objekt für Schuh und Spielzeug

Der durchgeführte Test zeigt, dass die 3D-Objekterstellung auch ohne Greifarm mit in dieser Arbeit dargestellten Algorithmen möglich ist. In der Tabelle 7.1 sind die Laufzeiten der Tests für alle Objekte zusammengefasst. In der Abbildung 7.13 sind die Ergebnisse als Balkendiagramm dargestellt. Alle Angaben sind in Sekunden.

Objekt	Registrierung mit Optimierung	Registrierung ohne Optimierung
Schachtel	266-268	169-171
Stofftier	340-342	106-108
Weihnachtsmann	149-151	29-30
Flasche	176-177	66-68
Hase	370-373	137-140
Schuh	300-303	103-106

Tabelle 7.1: Testresultate von Registrierung

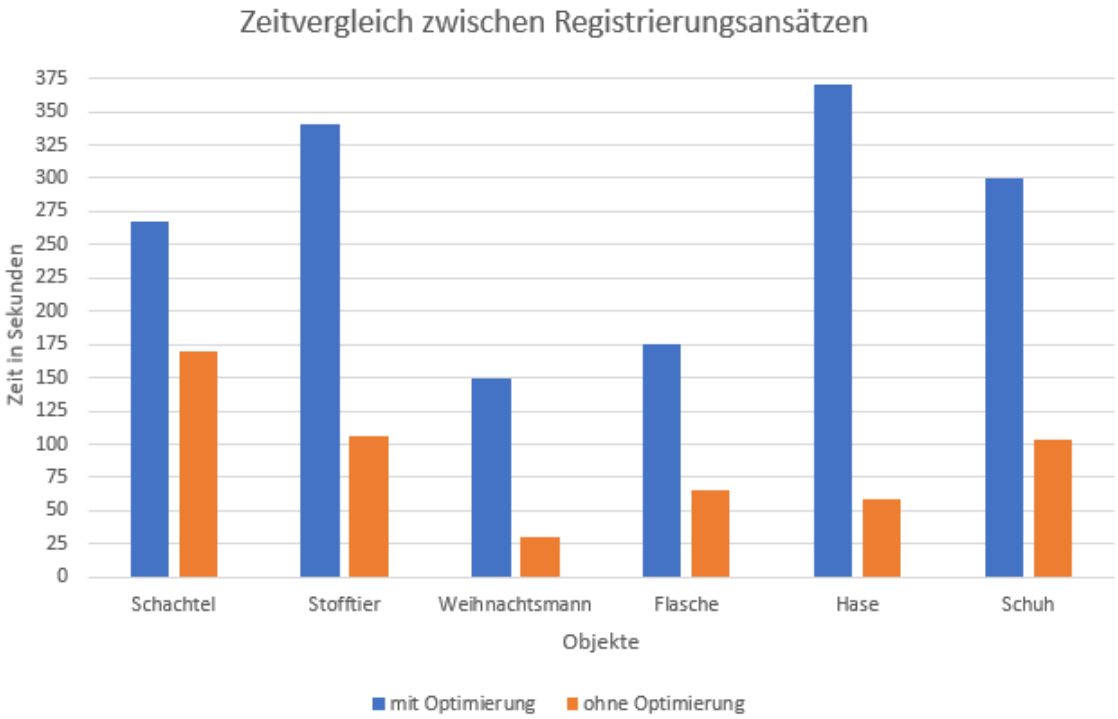


Abbildung 7.13: Balkendiagramm mit Ergebnissen aus der Tabelle 6.1

Mit diesen Tests wurde gezeigt, welche Objekttypen mit den in dieser Arbeit beschriebenen Algorithmen modelliert werden können. Es ist zu sehen, dass der Algorithmus mit Pose Graph Optimierung zeitaufwendiger als Algorithmus ohne Optimierung ist. Dieser Algorithmus ist langsamer, weil er paarweise die Punktwolken registriert im Gegensatz zu anderem Algorithmus, der die Punktwolken aufeinander registriert. Andererseits ist die daraus entstehende Registrierung in manchen Fällen, wie im Beispiel mit der Schachtel, besser, als die Registrierung ohne Optimierung. Beide Registrierungsalgorithmen sind aber für vollkommen symmetrische und transparente Objekte nicht geeignet, und können diese Objekte nicht modellieren. Außerdem wurde mit den beiden letzten Tests gezeigt, dass die Ansätze auch mit manueller Objektuntersuchung angewendet werden können. Diese Objekte und die Objekte mit schmalen Stellen hatten auch Punktmängel, und dadurch Löcher in den 3D-Polygonen. Mit dem Greifarm erfolgt die Untersuchung autonom und genauer, dadurch wird die Registrierung besser und die Punktmängel werden für einige Objekte behoben.

Während der Tests wurde festgestellt, dass die Registrierung von Punktwolken von der Untersuchungsgeschwindigkeit abhängig ist. Die Geschwindigkeit muss so ausgewählt werden, dass die Koordinaten der Punkte sich in den gesammelten Punktwolken nicht groß voneinan-

der unterscheiden. Ansonsten können die Korrespondenzpunkte zwischen den Punktwolken schlecht ausgerechnet werden, und die Punktwolken werden falsch registriert.

Mit diesen Tests ist auch zu erkennen, dass die Anforderungen aus dem Kapitel 3 erfolgreich in das System umgesetzt sind. Nachdem der Prozess gestartet wurde, braucht er keine Interaktion mit Menschen und ist somit autonom (Anforderung 3). Die Objekte wurden zuerst klassifiziert (Anforderungen 7, 9). Auf Basis von Klassifizierungsresultaten wurde entschieden, ob die Objekte untersucht werden müssen (Anforderungen 10, 11). Die gesammelten Punktwolken wurden zu einer großen Punktwolke zusammengefügt und danach wurden 3D-Modelle erstellt (Anforderungen 14, 15). Die Pfade zu diesen Modellen wurden in eine Datei mit dem Typ csv gespeichert (Anforderung 17). Die Datei enthält zwei Spalten: eine für den Pfad zum Objekt und die andere für das Label, das manuell hinzugefügt werden soll.

8 Zusammenfassung

In der dargestellten Bachelorarbeit wurde eine Software entwickelt, die autonome 3D-Objektmodellierung durch Szenenexploration durchführt. Dafür sollen die zu modellierenden Objekte von allen Seiten untersucht werden. Die Untersuchung erfolgte mit einer Husky-Plattform und einem Roboterarm UR5. Die Objekte wurden mithilfe einer Tiefenkamera Intel Realsense D435 untersucht. Die Steuerung von dem Arm und der Kamera erfolgte mittels ROS Melodic. Der Prozess teilt sich zwischen zwei Rechnern. Die Kommunikation zwischen diesen Rechnern erfolgt durch eine Server-Client Architektur. Der Server gibt dem Client Befehle, steuert die Kamera und die Informationsspeicherung. Der Client sendet Befehle zum Roboter und steuert die Kreisbewegung um das Objekt herum. Die Kamera speichert die Fotos und Punktwolken während der Untersuchung. Der Prozess der autonomen Untersuchung des Objekts beginnt zuerst mit der Objekterkennung. Für die Erkennung wurde das Modell MobileNetV2, das mit Bildern aus ImageNet Menge trainiert wurde, benutzt. Die Objekte wurden nur dann untersucht und modelliert, wenn sie nicht von dem Modell erkannt wurden.

Die 3D-Modellierung erfolgte mithilfe von gesammelten Punktwolken. Dafür wurden diese Punktwolken zusammen in eine große Punktwolke zusammengeführt. Dabei ist das Problem aufgetreten, wie man die Punkte richtig zueinander orientiert und transformiert, damit sie am Ende ein vollständiges Objekt repräsentieren. Dieses Problem heißt Registrierungsproblem. Im Rahmen dieser Arbeit wurden zwei unterschiedliche Ansätze zur Lösung dieses Problems entwickelt. Als Basis für beide Algorithmen wurde der Punkt-zu-Ebene ICP Algorithmus benutzt. Der Unterschied zwischen beiden Algorithmen ist die Reihenfolge, in der die Punktwolken zusammen registriert werden. Der Algorithmus ohne Pose Graph Optimierung nahm die Punktwolken in der gleichen Reihenfolge, in der sie gespeichert wurden. Damit wurden alle Punktwolken aufeinander registriert. Der zweite Algorithmus erfolgte mit paarweiser Registrierung aller Punktwolken. Zusätzlich wurde bei diesem Algorithmus eine Pose Graph mit Punktwolkentransformationen erstellt und nach der Untersuchung mithilfe des Levenberg-Marquardt Algorithmus optimiert. Als Hilfsmittel für die Entwicklung dieser Algorithmen wurde die Software Open3D benutzt. Diese Software stellte die Implementierung des Punkt-zu-Ebene ICP bereit.

Während der zweiten Phase der 3D-Modellierung wurde aus registrierten Punktwolken ein festes 3D-Polygon erstellt. Im Rahmen dieser Arbeit wurden zwei unterschiedliche Algorithmen für die Polygonerstellung implementiert. Der erste Algorithmus ist eine Alpha Shape Methode für die Polygonerstellung. Der zweite Algorithmus benutzt Poisson Surface Rekonstruktion, um Polygone zu konstruieren. Bei der Erstellung des Polygons mithilfe des ersten Algorithmus entstehen häufig unnötige Dreiecken an den Stellen, wo die Dichte der Punkte aus der Punktwolke niedrig ist. Diese Polygone brauchen zusätzlich weitere Verarbeitung, um die 3D-Objekte besser darzustellen. Der zweite Algorithmus berücksichtigt die Stellen mit kleiner Punktdichte nicht, dort entstehen keine Dreiecke. Es gibt aber Situationen, wo nach der Registrierung kleine Punktmengen an der Objektoberfläche gebildet werden. Das gilt für die Objekte, die an den Seiten schmal sind. An diesen Stellen werden bei der Poisson Surface Rekonstruktion kleine Löcher produziert, was aber beim Alpha Shape Algorithmus mit angemessenem Alpha Parameter nicht passieren wird. Die Software lässt sich parametrisieren, und dadurch kann man unterschiedliche Algorithmen sowohl für die Registrierung, als auch für die 3D-Polygonerstellung nutzen.

Die Funktionalität der Software wurde mit den durchgeführten Tests geprüft. Die Ansätze sind für unterschiedliche Objekte anwendbar. Die Tests wurden für unterschiedliche Formen von Objekten durchgeführt. Bei der Registrierung von transparenten oder symmetrischen Objekten wurden einige Probleme entdeckt. Bei den transparenten Objekten wurden die Punkte an der durchsichtigen Objektoberfläche in den Punktwolken nicht abgebildet. Dadurch ist es nicht möglich die entstehenden Punktwolken zusammen zu registrieren. Bei symmetrischen Objekten haben die Punktwolken immer gleiche Koordinaten in Bezug auf die Kamera. Dadurch wird bei der Registrierung stets nur eine Seite des Objekts modelliert. Eine mögliche Lösung für dieses Problem ist, dass die Punktwolken bei der Untersuchung bereits richtig rotieren, die Rotation kann bezüglich der Kameraposition gemacht werden. Durch die Rotation werden alle Punktwolken eine sogenannte Initiale Registrierung erhalten, damit sollte es möglich sein auch symmetrische Objekte zu modellieren. Es wurden Tests gemacht, wo das Objekt auch ohne Roboter nur mit der Hand untersucht wurde. Mit diesen Tests wurde gezeigt, dass die Untersuchung auch ohne Greifarm manuell durchgeführt werden kann.

Die dargestellte Software ist gut für die Modellierung von kleinen Objekten auf Basis von Punktwolkenregistrierung geeignet. Die in dieser Arbeit dargestellten Ansätze können für die meisten Objekte das Registrierungsproblem lösen, und diese Objekte können erfolgreich in 3D-Polygone umgewandelt werden. Die Software ist vollkommen parametrisierbar. Ein Benutzer kann die Objektmodellierung mit unterschiedlichen Registrierungsalgorithmen, mit unterschiedlicher Tiefe bei der Poisson Surface Rekonstruktion und mit unterschiedlicher

Alpha bei dem Alpha Shape Algorithmus durchführen. Dadurch können für unterschiedliche Objekte die Parameter gefunden werden, die die Modellierung genauer und effizienter machen.

Literaturverzeichnis

[1] "Intel RealSenseD435," (Zugriffsdatum: 04.02.2021). [Online]. Available: https://www.intelrealsense.com/depth-camera-d435/

[2] "Jetson agx xavier developer kit," (Zugriffsdatum: 13.02.2021). [Online]. Available: https://developer.nvidia.com/embedded/jetson-agx-xavier-developer-kit

[3] Maximilian Mang, Nils Schönherr, "Universal robots productshusky platform," (Zugriffsdatum: 13.02.2021). [Online]. Available: https://autosys.informatik.haw-hamburg.de/platforms/2020husky/

[4] B.-S. Hua, M.-K. Tran, and S.-K. Yeung, "Pointwise convolutional neural networks," 2018.

[5] "Velodyne Lidar," (Zugriffsdatum: 05.02.2021). [Online]. Available: https://www.roboticsbusinessreview.com/unmanned/unmanned-ground/velodyne-touts-latest-lidar-advances-autonomous-vehicle-partnerships/

[6] P. Daukantas, "Adding a new dimension: Lidar and archaeology," *Optics Photonics News*, vol. 25, pp. 32–39, 2014.

[7] (Zugriffsdatum: 04.04.2021). [Online]. Available: https://altigator.com/en/lidar-point-cloud-landscape2_sm/

[8] (Zugriffsdatum: 04.04.2021). [Online]. Available: http://www.gradientspace.com/tutorials/2017/8/30/mesh-simplification

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[10] "PS 1 - Rigid shape registration," (Zugriffsdatum: 04.02.2021). [Online]. Available: http://www.lix.polytechnique.fr/~maks/Verona_MPAM/TD/TD1/

[11] Y. Pan, "Target-less registration of point clouds: A review," 2019.

[12] (Zugriffsdatum: 04.02.2021). [Online]. Available: https://www.mathematik-oberstufe.de/vektoren/ko/k3d-punkte-abstand.html

[13] S.-I. Choi, U. Wijenayake, and S.-Y. Park, "Head pose tracking using gpu based real-time 3d registration," 08 2013.

[14] Kaspar Fischer, "Introduction to alpha shapes," (Zugriffsdatum: 13.02.2021). [Online]. Available: https://graphics.stanford.edu/courses/cs268-11-spring/handouts/AlphaShapes/as_fisher.pdf

[15] W. Rokicki and E. Gawell, "Voronoi diagrams – architectural and structural rod structure research model optimization," *MAZOWSZE Studia Regionalne*, vol. 2016, pp. 155–164, 09 2016.

[16] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," ser. SGP '06. Goslar, DEU: Eurographics Association, 2006, p. 61–70.

[17] "What is 3D Modeling What's It Used For?" (Zugriffsdatum: 27.06.2021). [Online]. Available: https://conceptartempire.com/what-is-3d-modeling/

[18] S. Hacker and H. Handels, "Repräsentation und visualisierung von 3d-formvarianten von organen für die medizinische ausbildung," 01 2006, pp. 276–280.

[19] A. P. Placitelli and L. Gallo, "3d point cloud sensors for low-cost medical in-situ visualization," 11 2011.

[20] J. Benavides, G. Jiménez, M. Sánchez Romero, E. García, g. LOZANO MEDINA, and J. A. Esquivel, "3d modelling in archaeology: The application of structure from motion methods to the study of the megalithic necropolis of panoria (granada, spain)," *Journal of Archaeological Science: Reports*, vol. 10, pp. 495–506, 12 2016.

[21] F. Poux, R. Neuville, L. Van Wersch, G.-A. Nys, and R. Billen, "3d point clouds in archaeology: Advances in acquisition, processing and knowledge integration applied to quasi-planar objects," *Geosciences*, vol. 7, p. 96, 09 2017.

[22] Stephan Pareigis,Tim Tiedemann, Maximilian De Muirier, "Test area intelligent quartier mobility (tiq)," (Zugriffsdatum: 19.06.2021). [Online]. Available: https://autosys.informatik.haw-hamburg.de/project/smartmobility/

[23] S. Kriegel, "Autonomous 3d modeling of unknown objects for active scene exploration," Ph.D. dissertation, Technische Universität München (TUM), 2015. [Online]. Available: https://elib.dlr.de/97250/

[24] S. Kriegel, C. Rink, T. Bodenmüller, and M. Suppa, "Efficient next-best-scan planning for autonomous 3d surface reconstruction of unknown objects," *Journal of Real-Time Image Processing*, vol. 10, 12 2013.

[25] Y. Chen and G. Medioni, "Object modeling by registration of multiple range images," *Image Vision Comput.*, vol. 10, pp. 145–155, 01 1992.

[26] Sungjoon Choi, Q. Zhou, and V. Koltun, "Robust reconstruction of indoor scenes," pp. 5556–5565, 2015.

[27] H. Edelsbrunner and E. Mucke, "Three-dimensional alpha shapes," *ACM Transactions on Graphics*, vol. 13, 10 1994.

[28] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3D: A modern library for 3D data processing," *arXiv:1801.09847*, 2018.

[29] "Intel RealSense SDK," (Zugriffsdatum: 04.02.2021). [Online]. Available: https://www.intelrealsense.com/sdk-2/

[30] "Intel RealSense ROS Wrapper," (Zugriffsdatum: 04.02.2021). [Online]. Available: https://dev.intelrealsense.com/docs/ros-wrapper

[31] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org

[32] I. A. Sucan and S. Chitta, "Moveit," (Zugriffsdatum: 05.02.2021). [Online]. Available: https://moveit.ros.org

[33] "Universal robots," (Zugriffsdatum: 13.02.2021). [Online]. Available: https://www.universal-robots.com

[34] "Universal robots products," (Zugriffsdatum: 13.02.2021). [Online]. Available: https://www.universal-robots.com/products/

[35] A. Meijer, J. Heitman, J. White, and R. Austin, "Measuring erosion in long-term tillage plots using ground-based lidar," *Soil and Tillage Research*, vol. 126, pp. 1–10, 01 2013.

[36] Benavides, José and Jiménez, G. and Sánchez Romero, Margarita and García, E. and Martín, S. and LOZANO MEDINA, Águeda and Esquivel, Jose Antonio, "3d modelling in archaeology: The application of structure from motion methods to the study of the megalithic necropolis of panoria (granada, spain)," *Journal of Archaeological Science: Reports*, vol. 10, pp. 495–506, 12 2016.

[37] U. Buck, "Digitale photogrammetrie und laserscanning in der forensik," 2006.

[38] N. Jens, "Verfahren zur ad hoc-modellierung und -simulation räumlicher feder-masse-systeme für den einsatz in virtual reality-basierten handhabungssimulationen," *Fraunhofer IPK*.

[39] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 06 2018, pp. 4510–4520.

[40] S. Kontschak, "Analyse paarweiser und punktbasierter Registrierungsverfahren auf hochaufgelösten, sphärischen 3D-Punktwolken," Diploma Thesis, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Germany, January 2012.

[41] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[42] B. K. P. Horn, "Closed-form solution of absolute orientation using unit quaternions," vol. 4, no. 4, pp. 629–642, Apr 1987.

[43] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, no. 2, pp. 431–441, 1963.

[44] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*, 2011, pp. 3607–3613.

[45] I. Aloise and G. Grisetti, "Matrix difference in pose-graph optimization," 2018.

[46] W. Lorensen and H. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *ACM SIGGRAPH Computer Graphics*, vol. 21, pp. 163–, 08 1987.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Autonome 3D-Objektmodellierung durch aktive Szenenexploration mit einem UR5 Greifarm

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

_____	_____	_____
Ort	Datum	Unterschrift im Original