IFS INSTITUTE FOR SOFTWARE

OST Ostschweizer Fachhochschule

Big Data Management von Punktwolken

Bachelorarbeit

Studiengang Informatik
OST - Ostschweizer Fachhochschule
Campus Rapperswil-Jona

Frühjahrssemester 2022

Autoren:	Reto Ehrensperger Christian Rutzer
Betreuer:	Prof. Stefan F. Keller
Projektpartner:	Prof. Dr. Dejan Šeatović

Version:	17. Juni 2022

Abstract

Light Detection And Ranging (LiDAR) ist eine Methode, die verwendet werden kann, um Objekte oder Umgebungen als Punktwolke aufzunehmen. In Kombination mit Simultaneous Localization And Mapping (SLAM) ist es mobilen Robotern möglich ihre eigene Position in einer dieser Umgebungen zu bestimmen. Damit können sie verwendet werden, um autonom Punktwolken ihres Umfelds zu erstellen.

Das Institute for Lab Automation and Mechatronics der OST (ILT) hat ihren Roboterhund Spot mit einem LiDAR-Sensor und einem Mini-Computer mit SLAM-Algorithmus ausgestattet. Damit sollen in Zukunft Punktwolken unterschiedlicher Indoor-Umgebungen erstellt werden. Um diesem Ziel näher zu kommen, wird eine Software-Komponente benötigt, die Punktwolken zentral verwaltet und noch während der Aufnahme deren Qualität überprüft werden kann. Dafür wurde im Rahmen dieser Arbeit eine Applikation namens "3D Data Management" entwickelt. Sie ermöglicht es mit mobilen Clients (zum Beispiel Roboter) aufgenommene Punktwolken über eine Schnittstelle an einen Server zu senden. Die empfangenen Daten werden inkrementell in einer Webapplikation dargestellt und können so beinahe in Echtzeit betrachtet werden.

Mobile Clients können 3D-Daten in Form von Punktwolken (Point Clouds) oder Polygonnetze (Meshes) im PLY-Dateiformat an die REST-API der Webapplikation übermitteln. Die übertragenen Daten werden auf dem Server zunächst unverändert als PLY-Datei verwaltet. Zusätzlich werden zu jeder Aufnahme zugehörige Metadaten wie Name, Ort, Zeitpunkt, Dateigrösse und Vorschau in einer PostgreSQL-Datenbank gespeichert. Für die unmittelbare Darstellung der aufgenommenen Punkte im Webbrowser (JavaScript und Three.js) wird die Webapplikation per Websocket-Verbindung mit dem Backend (Python und Django) über neue Aufnahmen benachrichtigt. Am Ende einer Aufnahmesession können die Teilaufnahmen zu einer einzelnen Datei zusammengefügt und als Ganzes verwaltet, heruntergeladen und weiterverarbeitet werden.

Die Funktionalität der Applikation wurde am Ende des Projekts mit dem Roboterhund Spot des ILT getestet. Die Systemtests haben gezeigt, dass die Software alle gewünschten Anwendungsfälle abdeckt. Sie leistet damit einen wichtigen Beitrag zur Vision des ILT. Die Modulare Softwarearchitektur der Applikation bildet eine gute Grundlage für Weiterentwicklungen.

Abstract English

Light Detection And Ranging (LiDAR) is a method that can be used to record objects or environments as a point cloud. In combination with Simultaneous Localization And Mapping (SLAM), it is possible for mobile robots to determine their own position in one of these environments. This means that mobile robots can be used to autonomously create point clouds of their environment.

The Institute for Lab Automation and Mechatronics (ILT) has equipped their robot dog Spot with a LiDAR sensor and a mini-computer with a SLAM algorithm in order to be able to create point clouds of different environments in the future. To get closer to this goal, a software component is needed that allows the point clouds to be managed centrally and their quality to be checked while they are still being recorded. For this purpose, an application called "3D Data Management"was developed within the scope of this work. It enables point clouds recorded with mobile clients (e.g. robots) to be sent to a server via an interface and WLAN. The received data is displayed incrementally in a web application and can thus be viewed almost in real time.

Mobile clients can transmit 3D data in the form of point clouds or meshes in PLY file format to the REST API of the web application. The transmitted data is initially managed on the server unchanged as a PLY file. In addition, associated metadata such as name, location, time, file size and preview are stored in a PostgreSQL database for each recording. For the immediate display of the recorded points in the web browser (JavaScript and Three.js), the web application is notified of new recordings via a websocket connection with the backend (Python and Django). At the end of a recording session, the partial recordings can be merged into a single file and managed as a whole, downloaded and further processed.

The functionality of the application was tested at the end of the project with the robot dog Spot of the ILT. The system tests have shown that the software covers all desired use cases. It thus makes an important contribution to the vision of 3D indoor mapping and navigation.The modular software architecture of the application forms a good basis for further developments.

Management Summary

Aufgabenstellung

In dieser Arbeit wurde eine Applikation namens "3D Data Management" entwickelt, die es mobilen Clients , wie zum Beispiel dem Roboterhund Spot, ermöglicht aufgenommene Punktwolken über eine Schnittstelle an einen Server zu senden. Die empfangenen Daten werden inkrementell in einer Webapplikation dargestellt und können so beinahe in Echtzeit betrachtet werden. Am Ende einer Aufnahmesession können die Teilaufnahmen zu einer einzelnen Datei zusammengefügt und als Ganzes heruntergeladen und weiterverarbeitet werden.
Vielfach werden diese Punktwolken mit *Light Detection And Ranging* (*LiDAR*)-Sensoren aufgenommen. *LiDAR* ist eine Form des dreidimensionalen Laserscanning, bei der die Distanzen der Punkte mit Hilfe der Reflektion der Umgebung vermessen werden kann. In Kombination mit *Simultaneous Localization And Mapping* (*SLAM*)-Algorithmen können mobile Roboter ihre eigene Position in dieser aufgenommenen Umgebung bestimmen.

Vor dieser Arbeit wurde der vierbeinige Roboterhund Spot des *ILT* mit einem *LiDAR*-Sensor und einem Mini-Computer mit *SLAM*-Algorithmus ausgestattet und von einem Masterstudenten optimiert. In Zukunft soll dieser Roboterhund verwendet werden, um autonom Punktwolken unterschiedlicher Umgebungen, wie einsturzgefährdeten Gebäuden, welche für einen Menschen nur schwer oder gefahrvoll zugänglich sind, zu erstellen. Um diesem Ziel näher zu kommen, wird eine Software-Komponente benötigt, die Punktwolken zentral verwaltet und noch während der Aufnahme deren Qualität überprüft werden kann.

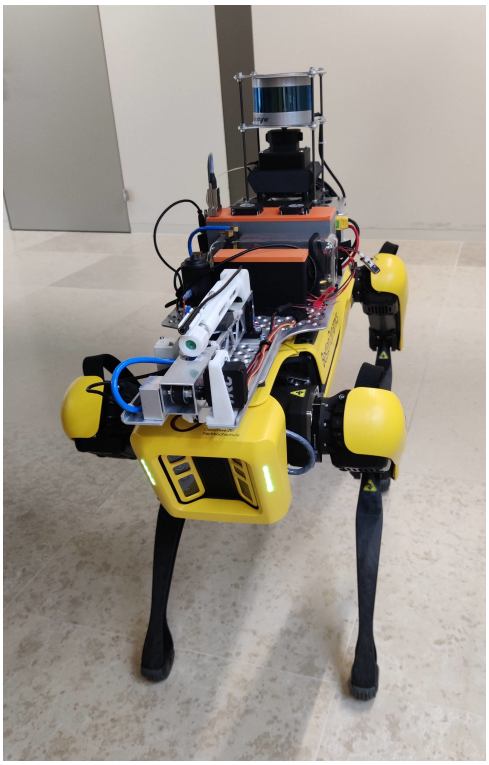


Abbildung 1: Boston Dynamics Spot mit Sensoreinheit

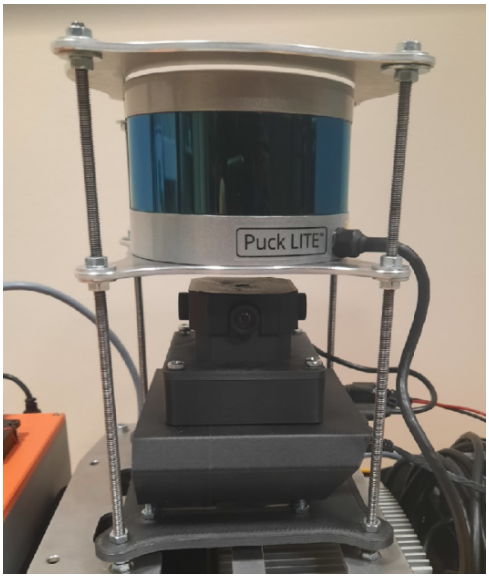


Abbildung 2: Sensoreinheit mit *LiDAR*-Sensor und $360°$ Kamera (Kümin, 2022)

Vorgehen und Technologie

Zu Beginn dieser Arbeit wurden verschiedene Ansätze für die Verwaltung der 3D-Daten, sowie deren Darstellung im Browser evaluiert. Dabei wurden der benötigte Speicherplatz und die Zugriffszeiten der Ansätze inklusive verschiedener Dateiformate verglichen. Der dateibasierte Ansatz im *PLY*-Dateiformat hatte dabei das beste Verhältnis zwischen Speichergrösse und Zugriffszeit.
Für die Darstellung von 3D-Daten wurden verschiedene browserbasierte Möglichkeiten geprüft. Die JavaScript-Bibliothek *Three.js* hat sich dabei als optimal erwiesen, da unter anderem ein dynamisches Darstellen von *PLY*-Dateien möglich ist.

Damit die Teile der Applikation als eigenständige Docker-Container betrieben werden können, wurden diese auf die folgenden drei Komponenten, welche je einen Container bilden, aufgeteilt. Der Collector und Processor bilden zusammen das Backend, der Viewer das Frontend.

Der Collector wurde in Python mit Hilfe des *Django* Frameworks in Kombination mit dem Django REST Framework entwickelt. Er bildet die zentrale Komponente der gesamten Applikation. Der entsprechende Container stellt eine *REST-API* zur Verfügung, über welche 3D-Daten in Form von Punktwolken (Point Clouds) oder Polygonnetze (Meshes) im *PLY*-Dateiformat, hoch- und heruntergeladen werden können. Die übertragenen Daten werden zunächst unverändert als *PLY*-Datei verwaltet. Zusätzlich werden zu jeder Aufnahme zugehörige Metadaten wie Name, Ort und Zeitpunkt in einer PostgreSQL-Datenbank gespeichert, damit nach diesen Informationen die Aufnahmen gefiltert und sortiert werden kann.

Der Processor ist für das Zusammenfügen der Teilaufnahmen zuständig. Dies bedeutet, dass mehrere *PLY*-Dateien zu einer einzelnen Datei zusammengefügt werden können. Das Zusammenfügen der Teilaufnahmen wird mit der *Open3D*-Bibliothek realisiert, welche viele zusätzliche Funktionen für das Verarbeiten von 3D-Daten mitbringt. Durch die lose Kopplung zum Collector, kann der Processor einfach mit zusätzlichen Funktionen erweitert und wiederverwendet werden.

Der Viewer wurde mit Hilfe der *React*-Bibliothek in Kombination mit *Three.js*, welches für die Darstellung der Punktwolken im Browser verwendet wird, implementiert. Da *Three.js* das direkte Darstellen von *PLY*-Dateien erlaubt, muss keine Präprozessierung durchgeführt werden und eine Aufnahme kann mit weniger Verzögerung im Browser dargestellt werden. Für die unmittelbare Darstellung der aufgenommenen Punkte, wird der Viewer per *Websocket*-Verbindung mit dem Collector über neue Aufnahmen benachrichtigt.

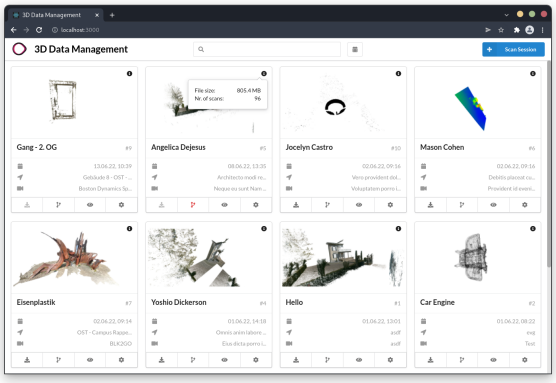


Abbildung 3: Viewer Übersicht

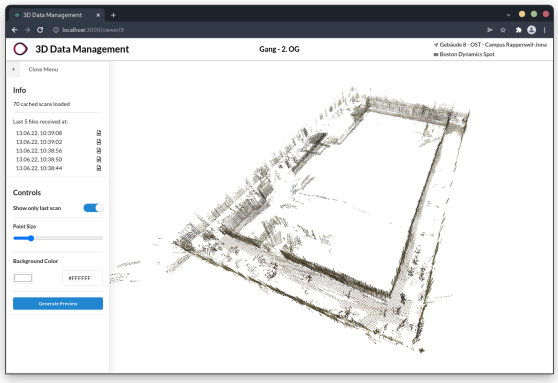


Abbildung 4: Darstellung eines Modells

Ergebnis

Am Ende konnte die entwickelte Applikation mit Hilfe von Florin Kümin, welcher sich in seiner Masterarbeit mit der Aufnahme der 3D-Daten mit mobilen Robotern befasst hat, getestet werden. Dafür wurde mit dem mobilen Roboter Spot der Gang des zweiten Obergeschosses im Gebäude 8 der *OST* Campus Rapperswil aufgenommen. Die Systemtests haben gezeigt, dass die Software alle gewünschten Anwendungsfälle abdeckt. Ein Problem, welches festgestellt werden musste, ist die Performance bei der Darstellung von riesigen Punktwolken. Dabei gibt es je nach Gerät, auf welchem die Webapplikation geöffnet wird, Verzögerungen bei der Darstellung. Dieses Problem trifft ab etwa 30 Millionen Punkte auf. Eine mögliche Lösung dieses Problems, wäre eine Reduktion der Punkte unmittelbar nach dem Hochladen. Diese Manipulation der Daten würde jedoch bedeuten, dass mehr Zeit zwischen Aufnahme und Darstellung der Daten im Browser vergeht.

Mit dieser Arbeit wurde ein wichtiger Beitrag zur Vision des *ILT* beigetragen. Die modulare Softwarearchitektur der Applikation bildet eine gute Grundlage für Weiterentwicklungen. Eine Möglichkeit wäre ein Hochladen und Manipulieren von Punktwolken, welche mit Consumer Geräten, wie einem iPad Pro mit integriertem *LiDAR*-Sensor, aufgenommen wurden.

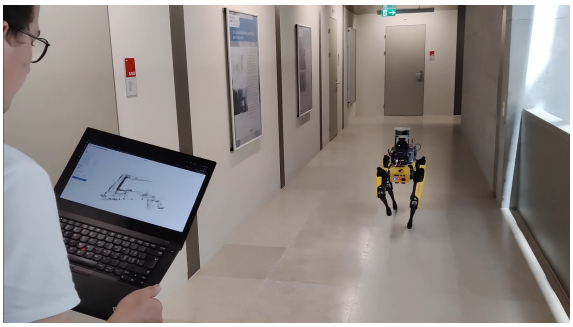


Abbildung 5: Systemtest mit dem Boston Dynamics Spot

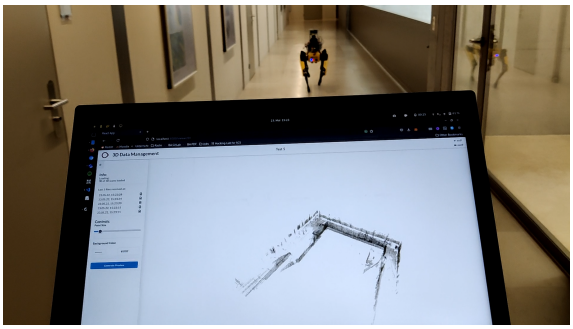


Abbildung 6: Darstellung der Punktwolke im Systemtest

Inhaltsverzeichnis

Abstract **2**

Abstract English **3**

Management Summary **4**

1 Einleitung **9**
1.1 Problemstellung / Vision . . . 9
1.2 Ziel der Arbeit . . . 9
1.3 Rahmenbedingungen . . . 9
1.4 Begriffserklärung . . . 10
1.4.1 Spot . . . 10
1.4.2 Scan . . . 10
1.4.3 Session . . . 10

2 Stand der Technik **11**
2.1 Technologische Grundlagen . . . 11
2.1.1 Punktwolken . . . 11
2.1.2 Meshes (Polygonnetze) . . . 12
2.1.3 Light detection and ranging (*LiDAR*) . . . 12
2.1.4 Simultaneous localization and mapping (*SLAM*) . . . 12
2.2 Verwandte Arbeiten . . . 13
2.2.1 Geoprocessing mit Handheld-Laser-Scanner erfassten Point Cloud-Daten (Nauli & Sennhauser, 2021) . . . 13
2.2.2 Autonomous exploration and color mapping of dynamic environments (Kümin, 2022) . . . 14

3 Analyse und Design **15**
3.1 Anforderungsspezifikation . . . 15
3.1.1 Use Case - Übersicht . . . 15
3.1.2 Use Case - Diagramm . . . 16
3.1.3 Use Case - Details . . . 17
3.1.4 Nicht-funktionale Anforderungen . . . 20
3.2 Evaluation - Verwaltung der 3D-Daten . . . 21
3.2.1 Beschreibung der Ansätze . . . 21
3.2.2 Vergleich der Ansätze . . . 23
3.2.3 Fazit . . . 25
3.3 Evaluation - Technologie Stack . . . 26
3.3.1 Backend Framework . . . 26
3.3.2 Frontend Framework . . . 26
3.3.3 Point Cloud / Mesh Viewer . . . 27
3.3.4 Library für das Zusammenfügen von Scans . . . 32
3.4 Architekturübersicht . . . 33
3.4.1 Container . . . 33
3.4.2 Datenfluss . . . 35
3.5 User Interface Mockup . . . 36

4 Realisierung **37**
4.1 Collector . . . 37
4.1.1 Datenspeicherung . . . 37

4.1.2 Beschreibung der Endpoints . . . 38
4.1.3 Verwendete Libraries . . . 41
4.2 Processor . . . 42
4.2.1 Separierung von Collector und Processor . . . 42
4.2.2 Beschreibung des Endpoints . . . 42
4.2.3 Verwendete Libraries . . . 42
4.3 Viewer . . . 43
4.3.1 Übersicht . . . 43
4.3.2 Technologien . . . 45
4.4 Deployment . . . 47
4.4.1 Übersicht . . . 47
4.4.2 Konfiguration . . . 48
4.4.3 Lokales Deployment . . . 48

5 Qualitätssicherung 49
5.1 Softwaretest . . . 49
5.1.1 Collector + Processor . . . 49
5.1.2 Viewer . . . 49
5.2 Statische Codeanalyse . . . 50
5.2.1 Reultat . . . 50
5.3 Systemtests . . . 51
5.3.1 Livetest . . . 51
5.3.2 Use Cases . . . 52

6 Schlussfolgerung 53
6.1 Einschränkungen . . . 53
6.1.1 Viewer Performance bei riesigen Punktwolken . . . 53
6.1.2 Initiale Ausrichtung eines Modells . . . 53
6.2 Weiterentwicklung . . . 54
6.2.1 Authentisierung . . . 54
6.2.2 Unterstützung zusätzlicher Dateiformate . . . 54
6.2.3 Erweiterung des Processor . . . 54
6.2.4 Unterstützung unterschiedlicher Auflösung . . . 54
6.3 Fazit . . . 55
6.4 Danksagung . . . 56

Glossar 56

Akronyme 57

7 Verzeichnisse 59
7.1 Abbildungsverzeichnis . . . 59
7.2 Tabellenverzeichnis . . . 60
7.3 Listings . . . 60
7.4 Literaturverzeichnis . . . 61

8 Anhang 62
8.1 Aufgabenstellung . . . 62
8.2 Projektplan . . . 63
8.3 Risikomanagement . . . 68
8.4 Manual . . . 72

1 Einleitung

Light Detection And Ranging (*LiDAR*)-Sensoren und *Simultaneous Localization And Mapping* (*SLAM*) ermöglichen es mobilen Robotern gleichzeitig eine Karte seiner Umgebung erstellen zu können, wie auch seine Position innerhalb dieser Karte zu bestimmen.

1.1 Problemstellung / Vision

Das *Institute for Lab Automation and Mechatronics* (*ILT*) der *OST* besitzt einen Roboterhund Spot von *Boston Dynamics*, welcher mit einem *LiDAR*-Sensor ausgestattet wurde. In Zukunft soll dieser Roboterhund verwendet werden können, um autonom Punktwolken unterschiedlicher Umgebungen, wie einsturzgefährdeten Gebäuden, welche für einen Menschen nur schwer oder gefahrvoll zugänglich sind, zu erstellen.

Das *ILT* wünscht sich deshalb die Möglichkeit die Qualität der Aufnahmen des Roboterhund Spots kontrollieren zu können, ohne sich dabei selbst in umgehender Nähe des mobilen Roboters befinden zu müssen.

1.2 Ziel der Arbeit

Ziel der vorliegenden Arbeit ist die Entwicklung einer Applikation, die es dem *ILT* erlaubt, die aufgenommenen Punktwolken hochzuladen, umgehend darzustellen und zu verwalten. Zusätzlich sollen diese Aufnahmen auch heruntergeladen und weiterverarbeitet werden können.

Die Ergebnisse der Arbeit (schriftlicher Projektbericht, Code, Dokumentation) werden am Ende den Projektbeteiligten zur Verfügung gestellt.

1.3 Rahmenbedingungen

Die Arbeit soll nach den Konzepten und Wünschen des *ILT* und *Institut für Software* (*IFS*) implementiert werden. Explizit sollen keine veraltete oder aufgegebene Libraries verwendet werden. Architektonisch ist die Applikation so zu entwickeln, dass diese lokal auf dem Ubuntu-Server des *ILT* ausgeführt werden kann. Ebenfalls wurde vorgegeben das Backend in Python zu implementieren.
Für die Umsetzung der Funktionalitäten auf dem Roboter ist das *ILT* zuständig. Am Schluss der Arbeit wird ein Python-Skript zur Verfügung gestellt, mit welchem Dateien an die *REST-API* übermittelt werden können.

1.4 Begriffserklärung

Im Rahmen dieser Arbeit gibt es oft verwendete Begriffe, welche aber auch anderweitig bekannt sind. Deshalb wird in diesem Abschnitt die Bedeutung der Begriffe, wie sie in dieser Arbeit verwendet werden, genauer erklärt.

1.4.1 Spot

Der Spot ist ein mobiler vierbeiniger Roboter von *Boston Dynamics*. Dieser kann sich ohne Probleme in unterschiedlichem und schwer zugänglichem Gelände bewegen. Damit können routinemässige Aufgaben, wie zum Beispiel die sichere und genaue Datenerfassung, automatisiert werden (Boston Dynamics, 2022a).

Abbildung 7: Spot von *Boston Dynamics* (Boston Dynamics, 2022b)

1.4.2 Scan

Ein Scan stellt eine einzige Aufnahme eines *LiDAR*-Sensors dar. Der Begriff verweist auf die hochgeladene *PLY*-Datei selbst, aber auch die dazu gehörigen Metadaten. Diese beinhalten ein Verweis auf die angehörige Session, den Namen der Datei, die Grösse einer Datei in Byte, sowie das Datum und die Uhrzeit, wann der Scan bei der Applikation hochgeladen wurde.

1.4.3 Session

Eine Session stellt die gesamte Aufnahme eines realen Objekts dar. Grundsätzlich besteht sie aus mehreren Scans und wiederum eigenen Metadaten, die eine Session beschreiben. Diese beinhalten den Namen des Objekts, Standort des Objekts, den Namen des Geräts, mit dem die Session aufgenommen wurden, Datum und Uhrzeit des zuletzt eingetroffenen Scans, sowie den Status der zusammengefügten Datei. Zusätzlich kann sie auf eine zusammengefügte Datei sowie auf ein Vorschaubild verweisen.

2 Stand der Technik

In diesem Abschnitt werden technologische Grundlagen, sowie verwandte Arbeiten, die im Rahmen dieser Arbeit verwendet wurden, vorgestellt.

2.1 Technologische Grundlagen

2.1.1 Punktwolken

Eine Punktwolke, ist eine Menge von mehreren, meist mit einem Laser, aufgenommenen Punkten. Ein Punkt besteht aus X-, Y- und Z-Koordinate und kann zusätzliche Features wie Farbwerte beinhalten. Je nach grösse eines Objekts müssen für eine optimale Darstellung eines 3D-Bilds mehrere Millionen Punkte aufgenommen werden.

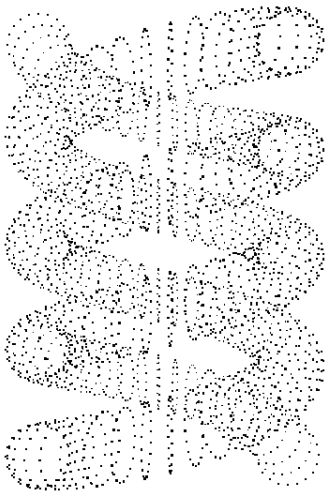


Abbildung 8: Helix als Punktwolke (John Burkardt, 2012)

2.1.2 Meshes (Polygonnetze)

Ein Mesh ist eine Sammlung von Punkten, Kanten und Flächen, die verwendet wird, um 3D-Modelle darzustellen (Petty, o. J.). Unterschiedliche Algorithmen können verwendet werden, um anhand von Punktwolken die Kanten und Flächen zu berechnen. Meist werden die Flächen in Form von Dreiecken dargestellt. Ein Vorteil von Meshes gegenüber einer Punktwolke besteht darin, dass für die Darstellung eines Objekts eine Vielzahl von Punkten eingespart werde kann. Durch die Berechnung von optimalen Polygonen, können Flächen von mehreren Punkten mit drei Vektoren dargestellt werden.

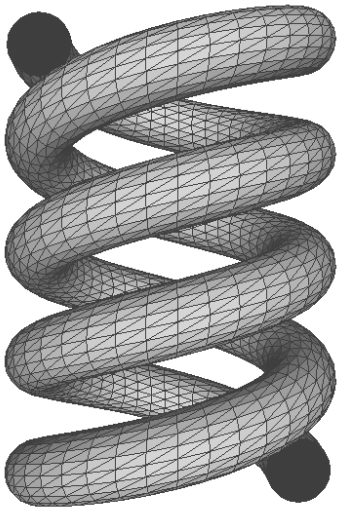


Abbildung 9: Helix als Mesh (John Burkardt, 2012)

2.1.3 Light detection and ranging (*LiDAR*)

LiDAR ist eine Technologie, die für Erkennen von Oberflächen-Merkmalen verwendet wird. Dabei werden Laserstrahlen entsendet und von Oberflächen reflektiert, womit die Distanz eines Punkts berechnet werden kann. Die aufgenommenen Daten werden oftmals mit anderen Sensoren, wie Kamerabilder für Farbwerte, kombiniert. Aus diesen Daten entstehen Punktwolken (Dempsey, 2022).

2.1.4 Simultaneous localization and mapping (*SLAM*)

SLAM ist eine Methode, welche es mobilen Robotern erlaubt die eigene Position einer simultan aufgenommenen Umgebung zu bestimmen. Grob gesagt werden dafür zwei technologische Komponenten benötigt. Eine Komponente, welche Aufnahmen der Umgebung erstellt und eine Komponente, welche die eigene Position in dieser Umgebung berechnet (MathWorks, o. J.).

2.2 Verwandte Arbeiten

2.2.1 Geoprocessing mit Handheld-Laser-Scanner erfassten Point Cloud-Daten (Nauli & Sennhauser, 2021)

Die Bachelorarbeit "Geoprocessing mit Handheld-Laser-Scanner erfassten Point Cloud-Daten" wurde von Denis Nauli und Nadine Sennhauser im Frühjahrssemester 2021 an der *OST* im Studiengang Informatik geschrieben. Die Arbeit kann grob in folgende zwei Themenbereiche aufgeteilt werden:

Handheld-Laser-Scanner Test
Dabei wurden verschiedene Handlaserscanner an drei unterschiedlichen Objekten getestet. Es hat sich gezeigt, dass professionelle Geräte einen hohen Preis haben, jedoch gegenüber einem iPad Pro eine deutlich bessere Genauigkeit der aufgenommenen Punkte erzielen.
Dieser Test war für diese Arbeit nicht direkt relevant. Dennoch ist es interessant um verschiedene Scanner kennenzulernen und zu sehen, wie unterschiedlich die Qualität der aufgenommenen Punktwolken sein kann.

Pointcloud Browser
Zusätzlich zum Handheld-Laser-Scanner Test wurde eine Webapplikation entwickelt, die das Hochladen, Visualisieren, sowie das Teilen von Punktwolken ermöglicht. Dabei wurde auf das gezippte LAS Format, .LAZ, für die Speicherung der Punktwolken gesetzt. Für die Visualisierung wurde der Potree-Viewer verwendet.
Da die Arbeit von Nauli und Sennhauser (2021) ein ähnliches Thema behandelt hat, wurden die dabei verwendeten Technologien ebenfalls evaluiert.

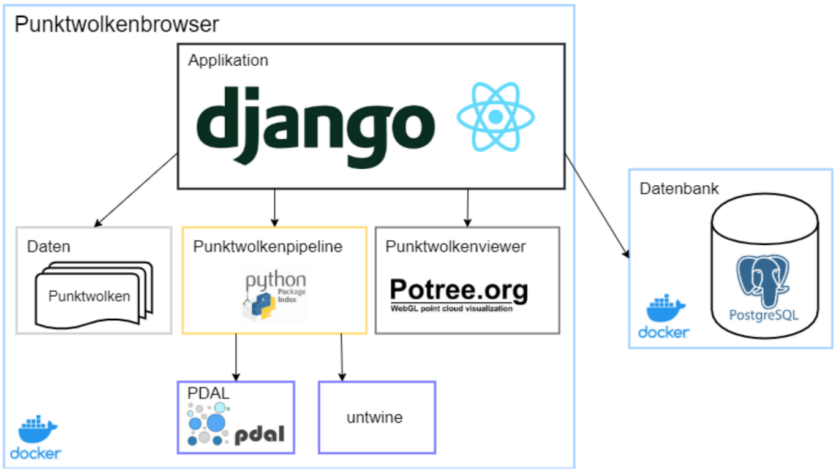


Abbildung 10: Pointcloud Browser Übersicht
(Nauli & Sennhauser, 2021)

2.2.2 Autonomous exploration and color mapping of dynamic environments (Kümin, 2022)

Die Masterarbeit "Autonomous exploration and color mapping of dynamic environments" wurde von Florin Kümin im Jahr 2022 als Masterarbeit im Studiengang Master of Science in Engineering an der *OST* geschrieben.

Sie befasste sich mit der Frage, ob die heutigen Werkzeuge in der mobilen Robotik ausreichen, um ein System zu entwickeln, welches in der Lage ist in einem Gebäude eine 3D-Umgebung aufzuzeichnen. Dabei wurde Open-Source-Software und kommerzielle Hardware verwendet. Das entwickelte System nutzt den vierbeinigen Roboterhund Spot, welcher mit einem *LiDAR*-Sensor und 360° Kamera ausgestattet wurde. Damit dynamische Objekte wie zum Beispiel Menschen aus den räumlichen Messungen entfernt werden können, wurde eine bildbasierte Pipeline, welche ein Deep Neural Network beinhaltet, verwendet.

Durch verschiedene Tests im Innen- und Aussenbereich, sowie in einfacher und komplexer Umgebung konnten wichtige Erkenntnisse und Schwächen des Systems gesammelt werden. Durch die modulare Entwicklung der Sensoreinheit und der Softwarekomponenten wird es in Zukunft möglich sein, gezielt bestimmte Teile zu optimieren.

Trotz eingeschränkten Ressourcen konnte in der Arbeit von Florin Kümin ein leistungsfähiges und funktionierendes System entwickelt werden, welches interessante 3D-Daten für diese Arbeit liefern konnte. Im Systemtest am Ende der Arbeit konnten beide Systeme kombiniert werden und die vom Roboterhund Spot aufgenommenen Daten direkt in der mit dieser Arbeit entwickelten Applikation visualisiert werden.

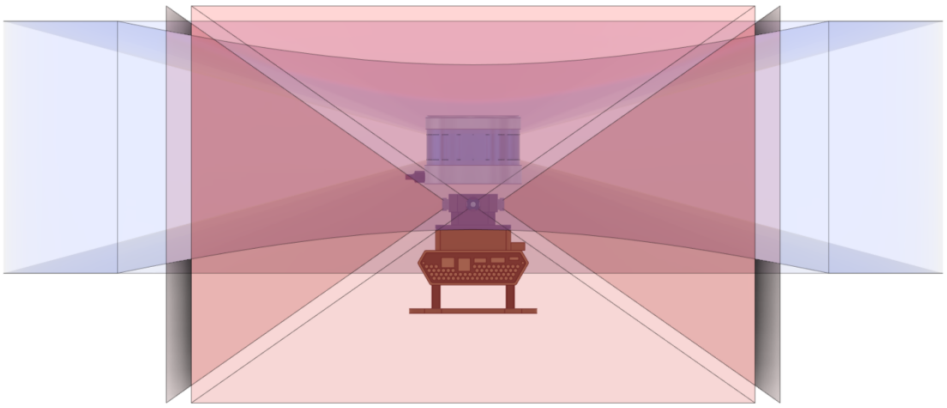


Abbildung 11:
Illustration Sensoreinheit
(Kümin, 2022)

Velodyne VLP16 (oben), vier Kameras (Mitte) und Jetson Nano (unten)
Markierte Bereiche: *LiDAR* Aussicht (blau), Kamera Aussicht (rot)

3 Analyse und Design

3.1 Anforderungsspezifikation

3.1.1 Use Case - Übersicht

Tabelle 1: Use Cases - Übersicht

Nr.	Use Case	Prio
1.1	Benutzer kann CRUD Operationen für Sessions inklusive Metadaten durchführen	Muss
1.2	Mobiler Roboter kann *PLY*-Dateien zu einer existierenden Session hochladen	Muss
1.3	Mehrere Punktwolken können zu einer Datei zusammengefügt werden	Muss
1.4	Vom mobilen Roboter hochgeladene *PLY*-Dateien werden inkrementell im Browser angezeigt	Muss
1.5	Zusammengefügte Punktwolken / Meshes können herunterladen werden	Muss
1.6	Benutzer kann in einer Session-Übersicht nach verschiedenen Metadaten suchen und filtern	Muss
1.7	Benutzer kann ein Vorschaubild einer Punktwolke / Mesh generieren	Soll
1.8	Benutzer kann sich authentisieren	Soll
1.9	Session kann für alle, nur eine Gruppe oder nur einem Benutzer zugänglich gemacht werden	Soll
1.10	Benutzer kann einen quadratischen Ausschnitt einer Punktwolke / Mesh herunterladen	Soll
1.11	Benutzer kann über Browser Punktwolken / Meshes als Datei hochladen	Kann

Aktoren

- **Benutzer**: Benutzer, welcher im Besitz eines *LiDAR*-Systems ist und Punktwolken aufnimmt.
- **Mobiler Roboter**: System, auf welchem die Daten eines *LiDAR*-Sensors verarbeitet werden.
- ***OSM* Mapper**: Person, welche das OpenStreetMap Projekt mit 3D-Daten unterstützen möchte.

3.1.2 Use Case - Diagramm

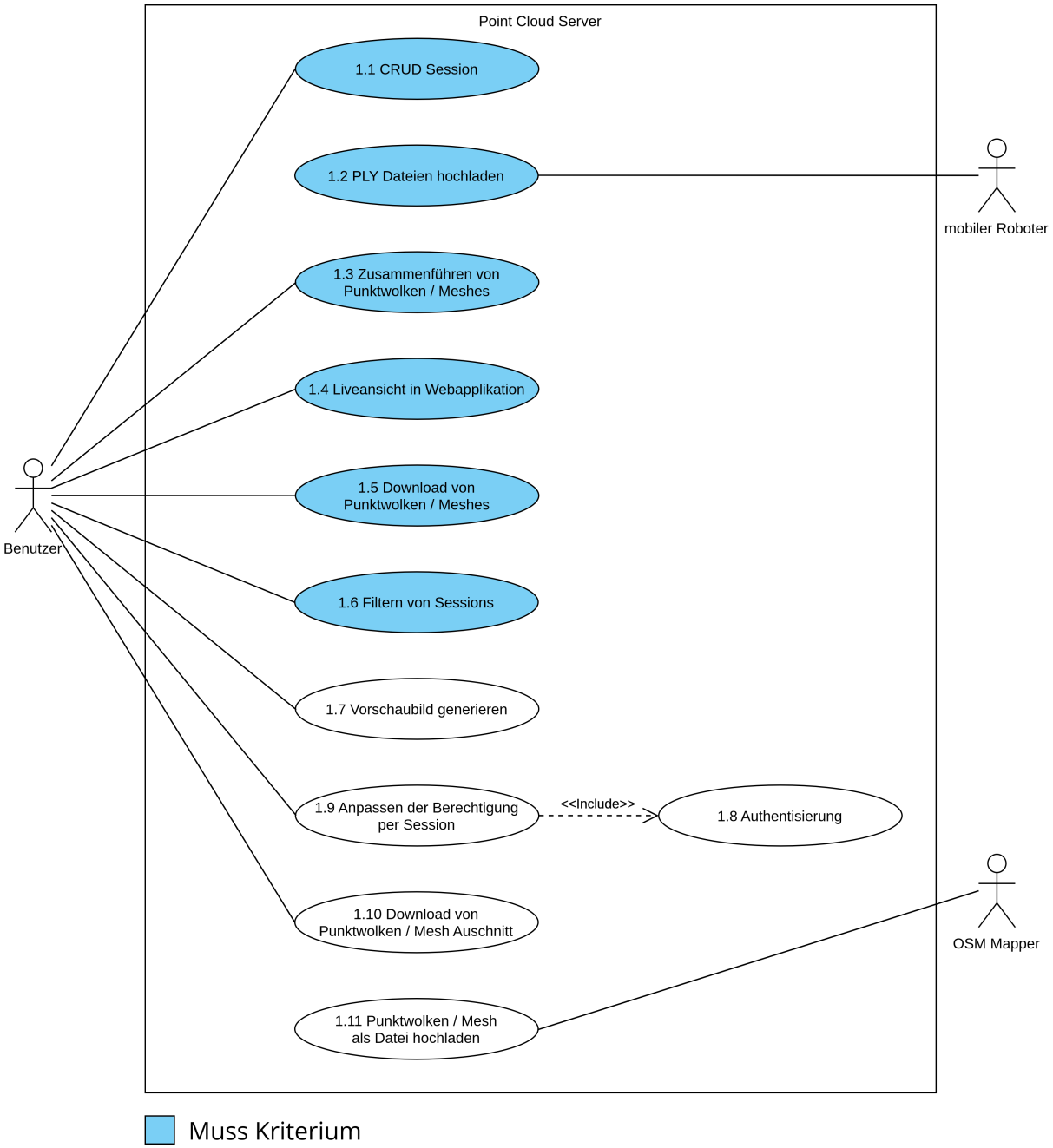


Abbildung 12: Use Case Diagramm

3.1.3 Use Case - Details

ID	1.1
Name	**CRUD Session**
Beschreibung	Benutzer kann CRUD Operationen für Sessions inklusive Metadaten durchführen
Actors	- Benutzer
Voraussetzungen	Für Read, Update und Delete muss Session vorhanden sein
Ablauf	1. Benutzer füllt Formular mit allen benötigten Daten aus 2. Benutzer erhält Session ID, mit der Dateien an die API gesendet werden können 3. Benutzer kann in der Session-Übersicht die Session updaten und löschen

ID	1.2
Name	***PLY*-Dateien hochladen**
Beschreibung	Ein mobiler Roboter kann *PLY*-Dateien hochladen
Actors	- mobiler Roboter
Voraussetzungen	Session vorhanden
Ablauf	1. *LiDAR* System sendet *PLY*-Datei an *REST-API* 2. API prüft, ob hochgeladene Datei im *PLY*-Format ist 3. API speichert Datei auf Dateisystem und Metadaten in Datenbank

ID	1.3
Name	**Zusammenfügen von Punktwolken / Meshes**
Beschreibung	Mehrere Punktwolken / Meshes können zu einer Datei zusammengefügt werden
Actors	- Benutzer
Voraussetzungen	Nutzbare Daten vorhanden
Ablauf	1. Prozess zum Zusammenfügen einzelner Punktwolken wird durch Benutzer ausgelöst 2. API speichert zusammengefügte Datei auf dem Dateisystem 3. Ende des Prozesses im UI erkennbar

ID	1.4
Name	**Liveansicht in Webapplikation**
Beschreibung	Vom mobilen Roboter hochgeladene *PLY*-Dateien werden inkrementell im Browser dargestellt
Actors	- Benutzer
Voraussetzungen	Punktwolken / Mesh werden von einem mobilen Roboter hochgeladen
Ablauf	1. Benutzer öffnet Modellansicht einer Session 2. Beim Upload einer neuen 3D-Datei wird die Ansicht automatisch aktualisiert

ID	1.5
Name	**Download von Punktwolken / Mesh**
Beschreibung	Benutzer kann zusammengefügte Punktwolken / Meshes herunterladen
Actors	- Benutzer
Voraussetzungen	Prozess zur Zusammenfügung ist abgeschlossen
Ablauf	1. *PLY*-Datei der gewünschten Session wird heruntergeladen

ID	1.6
Name	**Filtern von Sessions**
Beschreibung	Benutzer kann in einer Session-Übersicht nach verschiedenen Metadaten suchen und filtern
Actors	- Benutzer
Voraussetzungen	Sessions vorhanden
Ablauf	1. Durch Aktivierung verschiedener Filter oder Suche werden nur den Kriterien entsprechende Sessions angezeigt

ID	1.7
Name	**Vorschaubild generieren**
Beschreibung	Benutzer kann ein Vorschaubild einer Punktwolke / Mesh generieren
Actors	- Benutzer
Voraussetzungen	Session inklusive 3D-Daten vorhanden
Ablauf	1. Benutzer öffnet Modellansicht 2. Benutzer positioniert 3D-Modell in gewünschter Form 3. Durch Drücken eines Buttons wird ein Vorschaubild generiert

ID	1.8
Name	**Authentisierung**
Beschreibung	Benutzer kann sich authentisieren
Actors	- Benutzer
Ablauf	1. Benutzer meldet sich per Social Login an

ID	1.9
Name	**Anpassen der Berechtigung per Session**
Beschreibung	Session kann für alle, nur eine Gruppe oder nur einem Benutzer zugänglich gemacht werden
Actors	- Benutzer
Voraussetzungen	Authentisierung vorhanden
Ablauf	1. Benutzer erstellt eine neue Session 2. Berechtigung kann beim Erstellen oder nachträglich angepasst werden

ID	1.10
Name	**Download von Punktwolken / Mesh Ausschnitt**
Beschreibung	Benutzer kann einen quadratischen Ausschnitt einer Punktwolke / Mesh herunterladen
Actors	- Benutzer
Voraussetzungen	Session mit Daten vorhanden
Ablauf	1. Über ein Formular können gewünschte Parameter bestimmt werden 2. *PLY*-Datei des gewünschten Ausschnitts kann heruntergeladen werden

ID	1.11
Name	**Punktwolken / Mesh als Datei hochladen**
Beschreibung	Benutzer kann über Browser Punktwolken / Meshes als Datei hochladen
Actors	- *OSM* Mapper
Voraussetzungen	Benutzer ist angemeldet
Ablauf	1. Über ein Formular können *PLY*-Dateien hochgeladen werden

3.1.4 Nicht-funktionale Anforderungen

Tabelle 2: Nicht-funktionale Anforderungen

Nr.	Bezeichnung	Beschreibung
1	Installability	Das entwickelte Front- und Backend müssen dockerisiert und als Container lauffähig sein.
2	Testability	Vor jedem Merge müssen alle Testfälle erfolgreich ausgeführt werden
3	Performance	Punktwolken müssen nach der Aufnahme innerhalb von maximal 5 Sekunden in der Webapplikation angezeigt werden.

3.2 Evaluation - Verwaltung der 3D-Daten

Für die Verwaltung der 3D-Daten wurde ein dateibasierter Ansatz inklusive verschiedener Dateiformaten und ein datenbankbasierter Ansatz anhand von Speichergrösse und Zugriffszeit verglichen. Dabei wurden verschiedene Tests durchgeführt, womit das optimale Speicherformat bestimmt werden konnte.

3.2.1 Beschreibung der Ansätze

3.2.1.1 Dateibasierter Ansatz

Klassisch werden Punktwolken in binären Dateien gespeichert und von spezialisierter Software verarbeitet. Um Abfragen zu den gespeicherten Daten machen zu können, werden Metadaten in einer Datenbank gespeichert.

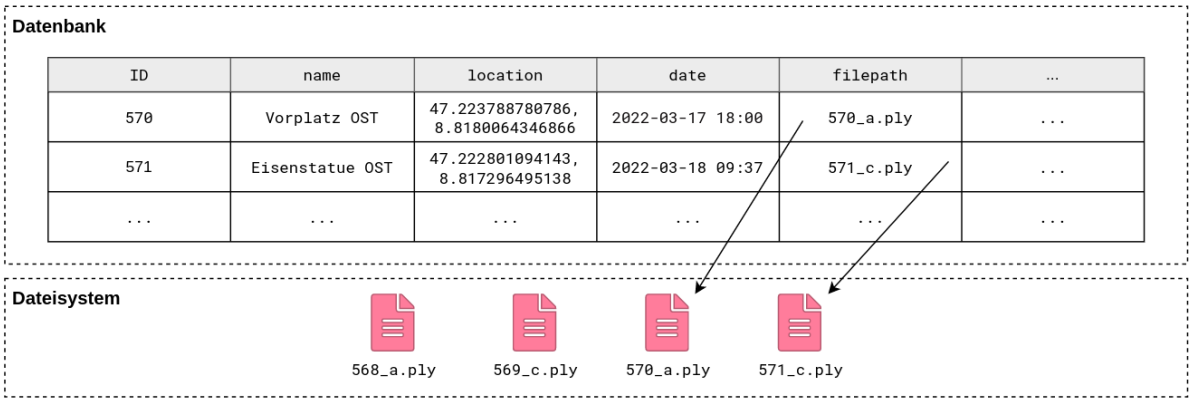


Abbildung 13: Übersicht Punktwolken als Datei

Vorteile

- Bessere Performance, da Zugriff ohne Datenbank Layer
- Speicher wird nur vom Dateisystem limitiert

Nachteile

- Sicherheits- und Zugriffskontrollen erschwert
- Unsynchronisierte Daten:
  - Datenbankeinträge, deren externe Daten nicht vorhanden sind
- Mehrere "Points of Failure" möglich

3.2.1.2 Datenbankbasierter Ansatz

Neben dem Dateibasierten Ansatz können Punktwolken auch direkt in einer Datenbank gespeichert werden. Um dies in einer relationalen Datenbank wie PostgreSQL effizient zu machen, müssen die Daten in sogenannte Patches unterteilt werden.

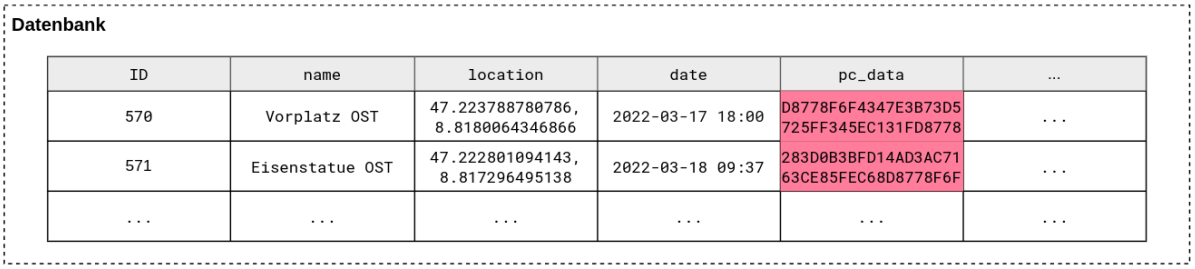


Abbildung 14: Übersicht Punktwolken in Datenbank

Vorteile

- Sicherheits- und Zugriffskontrollen vereinfacht
- Versionskontrolle einfacher
- *ACID* gewährleistet
- zentrale Backups möglich

Nachteile

- Schlechtere Performance als Dateibasierter Ansatz
- Speicheranforderungen sind höher

3.2.2 Vergleich der Ansätze

Das *PLY*-Format ermöglicht sowohl das Speichern von Punktwolken wie auch von Meshes. Als Absicherung, dass sich das *PLY*-Format als geeignet erweist, wurden folgend unterschiedliche Datei Formate für die Speicherung von Punktwolken miteinander verglichen.

Dabei wurden die nachfolgenden Versuche mit jeweils einer kleinen (Hand), mittelgrossen (Auto) und einer grossen (Garage) Punktwolke durchgeführt.

Punktwolke Hand
≈400'000 Punkte

Punktwolke Auto
≈1 Million Punkte

Punktwolke Garage
≈27.8 Millionen Punkte

Die Nachfolgenden Versuche wurden auf einem Lenovo Thinkpad mit Intel Core i7-8550U 1.8GHz CPU und 32GB 2400 MHz RAM durgeführt.

3.2.2.1 Speicherplatz

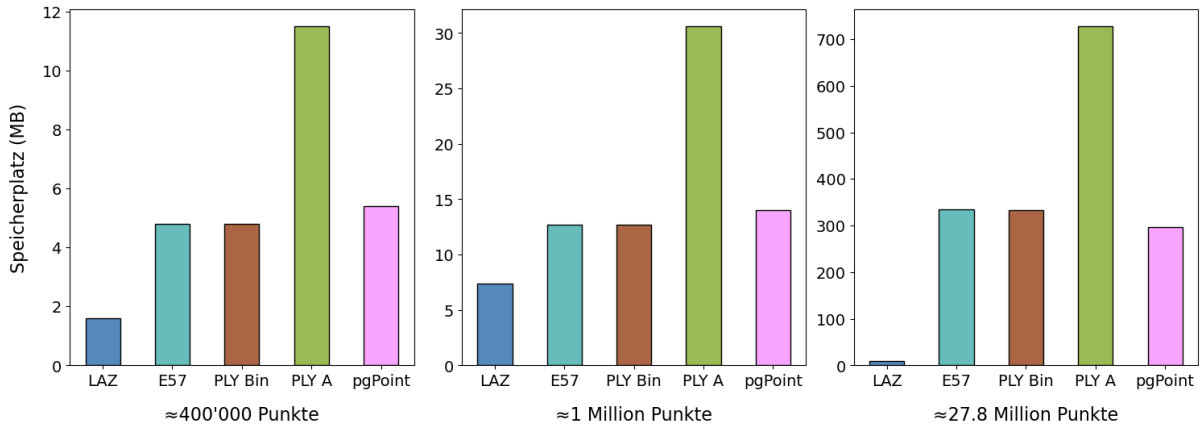


Abbildung 15: Vergleich Speicherplatz

Tabelle 3: Vergleich Speicherplatz - Messwerte in MB

	≈400'000	≈1 Million	≈27.8 Millionen
LAZ	1.6	7.4	9.5
E57	4.8	12.7	335
***PLY* Binär**	4.8	12.7	333.6
***PLY* ASCII**	11.5	30.6	727.5
pgPointcloud	5.4	14	296

3.2.2.2 Zugriffszeit

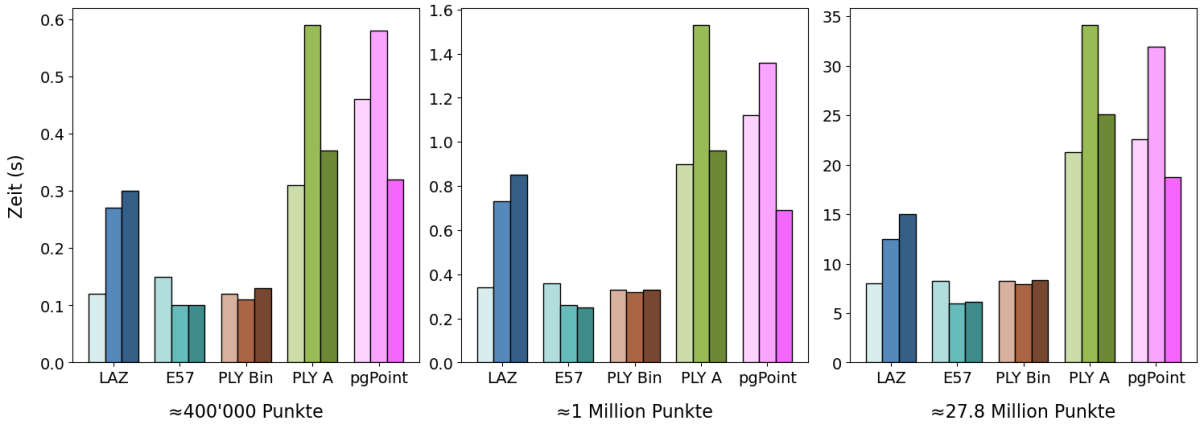
Um die Zugriffszeiten verschiedener Formate zu überprüfen, wurden jeweils die folgenden drei Schritte durchgeführt:

1. Einlesen der Punktwolke mit Hilfe einer *PDAL*-Pipeline
2. Filtern der Punktwolke. In diesem Schritt wurde jeder Zehnte Punkt ausgelesen und zu einer neuen Punktwolke hinzugefügt
3. Abspeichern der verkleinerten Punktwolke

Ablauf

- **P2F = *PLY* zu Format**
  → *PLY* Datei lesen → Dezimierung → Format z.B. E57 schreiben
- **F2F = Format zu Format**
  → Format z.B. E57 lesen → Dezimierung → Format z.B. E57 schreiben
- **F2P = Format zu *PLY***
  → Format z.B. E57 lesen → Dezimierung → *PLY* Datei schreiben

Ergebnisse



hell = P2F, mittel = F2F, dunkel = F2P

Abbildung 16: Vergleich Zugriffszeit

Tabelle 4: Vergleich Zugriffszeit - Messwerte in Sekunden

Hand	***PLY* zu Format**	**Format zu Format**	**Format zu *PLY***
LAZ	0.12	0.27	0.3
E57	0.15	0.1	0.1
***PLY* Binär**	0.12	0.11	0.13
***PLY* ASCII**	0.31	0.59	0.37
pgPointcloud	0.46	0.58	0.32

Auto	***PLY* zu Format**	**Format zu Format**	**Format zu *PLY***
LAZ	0.34	0.73	0.85
E57	0.36	0.26	0.25
***PLY* Binär**	0.33	0.32	0.33
***PLY* ASCII**	0.9	1.53	0.96
pgPointcloud	1.12	1.36	0.69

Garage	***PLY* zu Format**	**Format zu Format**	**Format zu *PLY***
LAZ	8.05	12.46	14.97
E57	8.26	6.01	6.14
***PLY* Binär**	8.24	7.96	8.35
***PLY* ASCII**	21.26	34.13	25.11
pgPointcloud	22.58	31.97	18.78

3.2.3 Fazit

Die verschiedenen Versuche haben aufgezeigt, dass es erhebliche Unterschiede zwischen den verschiedenen Dateiformaten für Punktwolken gibt. Das LAZ-Format braucht im Vergleich zu den anderen Formaten mit Abstand am wenigsten Platz. Die beiden binären Formate E57 und *PLY* Binär brauchen ungefähr gleich viel Platz wie die Variante mit PostgreSQL-Datenbank mit *PgPointcloud* Erweiterung. Der Vorteil von *PLY*-ASCII besteht darin, dass er menschenlesbar ist, braucht aber entsprechend viel Speicherplatz.
Die kleine Speichergrösse des komprimierten LAZ-Formats wirkt sich auf die Verarbeitungszeit aus, welche im Vergleich zu den anderen binären Dateiformaten höher ist. Das *PLY*-ASCII-Format braucht nicht nur am meisten Speicher, sondern braucht auch am längsten für die Verarbeitung. Der Datenbankansatz brauchte im Vergleich zu den binären Dateiformaten durchschnittlich mehr als doppelt so lange, um verarbeitet zu werden.

Beschluss - Verwaltung der 3D-Daten

Anhand der Erkenntnisse wurde entschieden, die Dateien als binäres *PLY*-Format zu speichern. Damit nicht bei jeder Anfrage das gesamte Dateisystem durchsucht werden muss, sollen gewisse Metadaten in einer PostgreSQL-Datenbank abgespeichert werden. Weitere Vorteile des *PLY*-Format sind:

- Punktwolken und Meshes speichern
- Definition von zusätzlichen Parametern z.B. Farbe möglich
- Verbreitung in handelsüblicher Software

3.3 Evaluation - Technologie Stack

3.3.1 Backend Framework

Es stehen verschiedene Frameworks zur Verfügung, welche alle ihre Stärken und Schwächen haben. Nauli und Sennhauser (2021) haben in ihrer Arbeit "Geoprocessing mit Handheld-Laser-Scanner erfassten Point Cloud-Daten" verschiedene Frameworks anhand von Kriterien wie Community oder Funktionalität verglichen. Da dieser Vergleich bereits sehr ausführlich durchgeführt wurde, wird in dieser Arbeit auf einen erneuten Vergleich verzichtet.

Beschluss - Backend Framework

Durch den Entschluss von (Nauli & Sennhauser, 2021), dass Django das optimale Framework für ihre Problemstellung ist, wurde entschlossen, die API dieser Arbeit auch mit dem Django Framework zu entwickeln. Dies bringt den zusätzlichen Vorteil, dass gewisse Code Abschnitte, welche von (Nauli & Sennhauser, 2021) umgesetzt wurden, wiederverwendet werden können.

3.3.2 Frontend Framework

Eine Anforderung an die Arbeit ist, dass das Frontend eine im Browser lauffähige Applikation sein soll. Im Projektteam konnte in vorherigen Projekten bereits Erfahrungen in *React* und Vue.js gesammelt werden. Nauli und Sennhauser (2021) haben sich ebenfalls mit diesen beiden Frameworks befasst, weshalb deren Evaluation ebenfalls in die Entscheidung eingeflossen ist.

Da aus den oben genannten Gründen kein klarer Favorit bestimmt werden konnte, wurde entschieden, das Framework mit der besseren Unterstützung für, die unter 3.3.3 evaluierte Library, *Three.js* zu verwenden.

Beschluss - Frontend Framework

Anhand der wöchentlichen Downloads und der Anzahl GitHub Stars lässt sich ableiten, dass die *Three.js* Unterstützungsbibliothek React-Three-Fibre (*React*) vielfach öfter eingesetzt wird als TroisJS (Vue.js). Aufgrund der grösseren Beliebtheit und entsprechend grösseren Community hinter React-Three-Fibre wurde entschieden, das Frontend mit *React* zu realisieren.

3.3.3 Point Cloud / Mesh Viewer

3.3.3.1 Potree

Potree ist ein gratis Open Source WebGL basierter Punktwolken-Renderer für grosse Punktwolken (Schütz, 2020).

Tabelle 5: Facts and Figures - Potree

Entwickler	TU Wien - Insitute of Visual Computing
Projekttreiber	Sponsoren (z.B. Synthesis Technology Ltd, GeoCue Group Inc)
Lizenz	FreeBSD
GitHub Stars	2'900
Release-Zyklus	Unregelmässig - ca. 1x pro Jahr

Abbildung 17: Screenshot - Potree

Tabelle 6: Technische Informationen - Potree

Anzeigen von Punktwolken	Ja
Anzeigen von Meshes	Nein
Dateiformat für Viewer	EPT
Dateiformat für Konverter	e57, LAS, OBJ, *PLY* usw.
Konverter 1. X zu LAZ 2. LAZ zu EPT	 1. PDAL 2. Entwine, Untwine, Potree Converter
dynamisches Laden neuer Dateien	Ja

3.3.3.2 3D Heritage Online Presenter

3DHOP ist ein Open Source Software Paket für die Erstellung von interaktiven webbasierten Präsentationen von 3D-Modellen (Callieri, 2016).

Tabelle 7: Facts and Figures - 3DHOP

Entwickler	National Research Council Italy, Visual Computing Lab
Projekttreiber	Forschung - 3DCOFORM Projekt
Lizenz	GPL
GitHub Stars	116
Release-Zyklus	ca. alle zwei Jahre

Abbildung 18: Screenshot - 3DHOP

Tabelle 8: Technische Informationen - 3DHOP

Anzeigen von Punktwolken	Nein
Anzeigen von Meshes	Ja
Dateiformat für Viewer	NXZ, *PLY*
Dateiformat für Konverter	*PLY*, OBJ
Konverter	Nexus
dynamisches Laden neuer Dateien	Nein

3.3.3.3 Three.js

Das Ziel von *Three.js* ist eine leicht zu verwendende, leichtgewichtige und cross Browser 3D Library zur Verfügung zu stellen (Cabello, 2021).

Tabelle 9: Facts and Figures - *Three.js*

Entwickler	Ricardo Cabello / Open Source Community
Projekttreiber	Open Source Community
Lizenz	MIT
GitHub Stars	80'900
Release-Zyklus	ca. 1x pro Monat

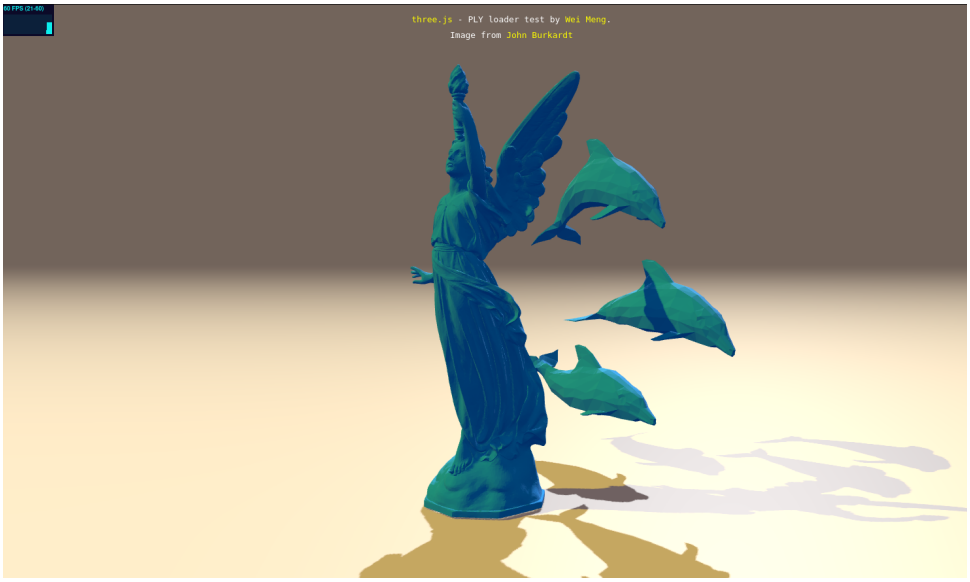


Abbildung 19: Screenshot - *Three.js*

Tabelle 10: Technische Informationen - *Three.js*

Anzeigen von Punktwolken	Ja
Anzeigen von Meshes	Ja
Dateiformat für Viewer	*PLY*, OBJ, PCD usw.
Konverter	Import mit Hilfe von Loader
dynamisches Laden neuer Dateien	Ja

3.3.3.4 Deck.gl

Deck.gl ist ein WebGL basiertes Framework für die visuelle explorative Datenanalyse von grossen Datensätzen (Chen, 2020).

Tabelle 11: Facts and Figures - Deck.gl

Entwickler	Vis.gl - Urban Computing Foundation
Projekttreiber	Urban Computing Foundation
Lizenz	MIT
GitHub Stars	9'900
Release-Zyklus	Laufend

Abbildung 20: Screenshot - Deck.gl

Tabelle 12: Technische Informationen - Deck.gl

Anzeigen von Punktwolken	Ja
Anzeigen von Meshes	Ja
Dateiformat für Viewer	*PLY*, OBJ, PCD, LAS usw.
Konverter	loaders.gl
dynamisches Laden neuer Dateien	unbekannt

3.3.3.5 Fazit

Da die Point Cloud / Mesh Viewer zwingend webbasiert und Open Source sein mussten, war die Auswahl stark eingeschränkt und es wurden nur die zuvor beschriebenen Viewer verglichen. Um die Funktionalität zu prüfen, wurde jeweils ein Prototyp anhand eines Beispielprojekts gebaut.

Schnell hat sich gezeigt, dass die Umsetzung mit *Three.js* den anderen Viewer in praktisch allen Punkten überlegen ist. Aus diesem Grund wurde darauf verzichtet eine genaue Bewertung aller Punkte vorzunehmen.

Vorteile von *Three.js*:

- Sehr grosse Verbreitung und aktive Community
- MIT Lizenz
- Regelmässige Updates
- Anzeigen von Punktwolken und Meshes
- Import verschiedener Dateiformaten möglich
- dynamischer Import möglich

Nachteile:
Durch die unzähligen Features von *Three.js* ist der Aufwand für eine einfache Darstellung einer Punktwolke oder Mesh um einiges höher als bei den anderen Viewern. Zum Beispiel muss selbst bestimmt werden, von welcher Position im Koordinatensystem ein Objekt angeleuchtet wird oder wie sich die Kamera um ein Objekt bewegen soll.

Beschluss - Point Cloud / Mesh Viewer

Auf Grund der oben genannten Gründe wurde entschieden, *Three.js* als Point Cloud / Mesh Viewer zu verwenden. *Three.js* bietet gegenüber den Alternativen Viewern einen grossen Vorteil bezüglich aktiver Community und allgemeiner Erweiterbarkeit.

3.3.4 Library für das Zusammenfügen von Scans

Das Zusammenfügen von Scans ist die aktuelle Hauptaufgabe des Processors. Dies ermöglicht es mehrere Scans in einer einzelnen Datei zusammenzuführen, welche später von den Benutzern heruntergeladen werden kann.

Nach eigener Recherche im Internet und mit Absprache mit dem Projektpartner wurden die drei Libraries, *PDAL* (Contributors, 2020) und *Open3D* (Zhou, Park & Koltun, 2018a) genauer angeschaut.

Es wurde für jede Library ein kleines Test-Projekt erstellt, in welchem die Funktionalitäten und Anwendung der Library genauer überprüft wurden.

3.3.4.1 Kriterien

Um eine Übersicht über die Vor- und Nachteile der Libraries zu bekommen, wurde eine Tabelle erstellt, mit den wichtigsten Kriterien, die an die Library gestellt werden.

Tabelle 13: Vergleich der Libraries für die 3D-Daten Verarbeitung

Kriterium	PDAL	Open3D
PLY-Unterstützung	Ja	Ja
Programmiersprache	Python	Python
Zusammenfügen von Punktwolken und Meshes	Nur Punktwolken	Punktwolken & Meshes
Lizenz	BSD License	MIT License

3.3.4.2 Resultat

Durch die Evaluation hat sich herausgestellt, dass sowohl *PDAL*, wie auch *Open3D* für die Anwendungsfälle tauglich sind. Beide Libraries bieten zusätzliche Funktionen für das Verarbeiten von Punktwolken und Meshes, womit der Processor jeweils auch einfach erweitert werden kann.

Beschluss - Library für das Zusammenfügen von Scans

Schlussendlich wurde die *Open3D*-Library ausgewählt, da diese neben Punktwolken auch Meshes Zusammenfügen kann. Zusätzlich wird die Library auch von Florin Kümin für seine Master-Thesis verwendet, womit am *ILT* für allfällige Weiterentwicklungen bereits Knowhow vorhanden ist.

3.4 Architekturübersicht

3.4.1 Container

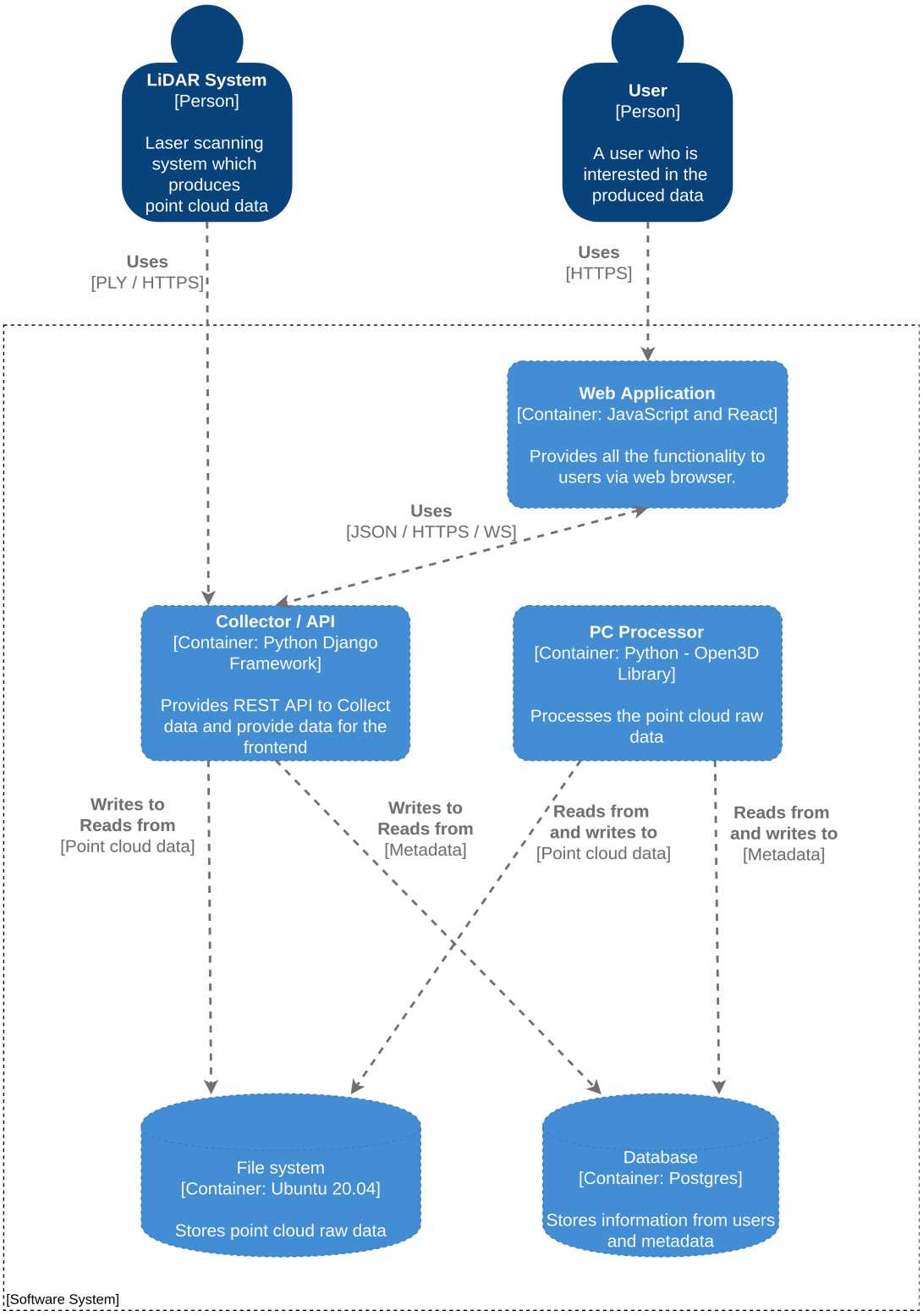


Abbildung 21: Architektur Übersicht

Libraries zur Modifizierung von 3D-Daten brauchen meist viel Speicherplatz und führen zu entsprechend schwergewichtigen Container Images. Aus diesem Grund wurde entschieden, jeweils eigenständige Container für Collector und Processor zu implementieren. Ein zusätzlicher Vorteil von zwei getrennten Containern ist die Möglichkeit, das System auf einem Rechner mit eingeschränkten Ressourcen betreiben zu können. Dabei kann auf den Processor Container verzichtet werden und nur die Basisfunktionen der API genutzt werden.

3.4.1.1 Kommunikation

Grundsätzlich läuft die Kommunikation der gesamten Applikation über HTTPS und über *REST-API* des Collectors. Dies funktioniert für allgemeine Abfragen, welche bei bestimmten Aktionen des Benutzers ausgelöst werden. Für die Liveansicht einer Punktwolke / Mesh, muss ein System zum Update der vorhanden Aufnahmen umgesetzt werden. Dabei gibt es zwei Möglichkeiten für die Umsetzung eines solchen Systems:

Polling
Beim Polling wird nach einer bestimmten Zeit jeweils eine neue Anfrage an den Server gemacht. In der untenstehenden Grafik wird jeweils jede Sekunde eine Anfrage gestartet. Im schlimmsten Fall kann eine Verzögerung von über einer Sekunde eintreffen, bis die aktuelle Aufnahme im Frontend empfangen wird.

Websocket
Eine *Websocket*-Verbindung ist dauerhaft und Daten können bidirektional zwischen Front- und Backend ausgetauscht werden. Aus diesem Grund können die Daten aus dem Backend direkt beim Eintreffen an das Frontend übermittelt werden.

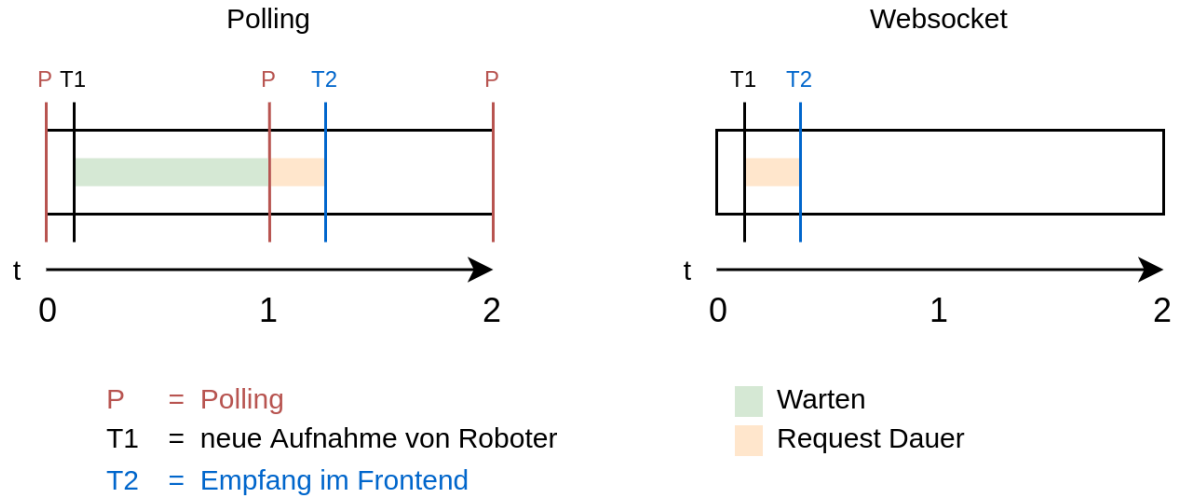


Abbildung 22: Polling vs. *Websocket*

3.4.2 Datenfluss

Beim Datenfluss wird zwischen Liveansicht und On-Demand Ansicht unterschieden.

Live

Um die aktuellen Aufnahmen möglichst zeitnah im Frontend anzeigen zu können, werden diese unbearbeitet vom Frontend heruntergeladen und angezeigt.

On-Demand

Wenn eine Session beendet wurde, können alle einzelnen Aufnahmen zusammengeführt und als komplette Datei heruntergeladen werden.

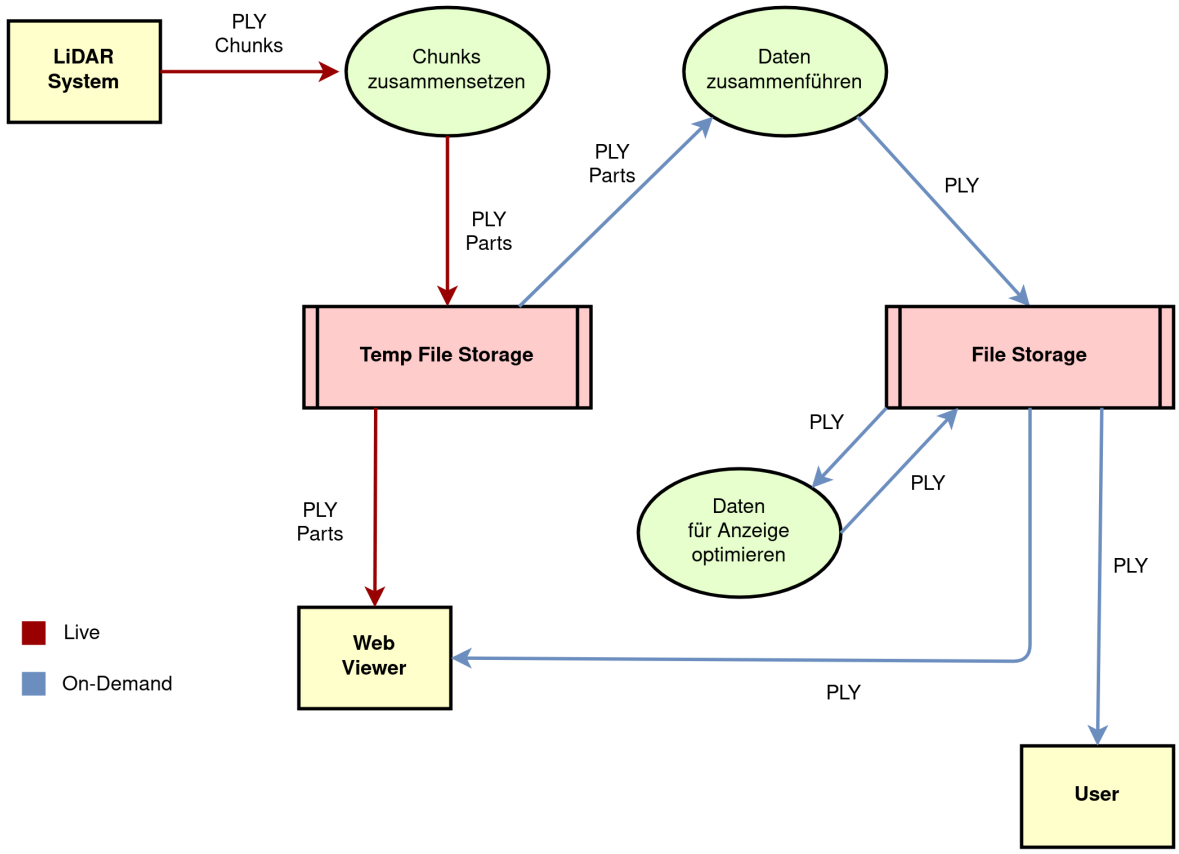


Abbildung 23: Übersicht Datenfluss

3.5 User Interface Mockup

Der wichtigste Teil des User Interfaces ist die Ansicht eines Modells. In dieser soll ein 3D-Modell gezeichnet werden und mit Steuerungselementen wie zum Beispiel Zoom bewegt werden können. Zusätzlich soll ein Übersichts- und ein Detailscreen mit entsprechenden Metadaten implementiert werden. Um alle Anforderungen mit dem Projektpartner besprechen zu können, wurden folgende Mockups erstellt.

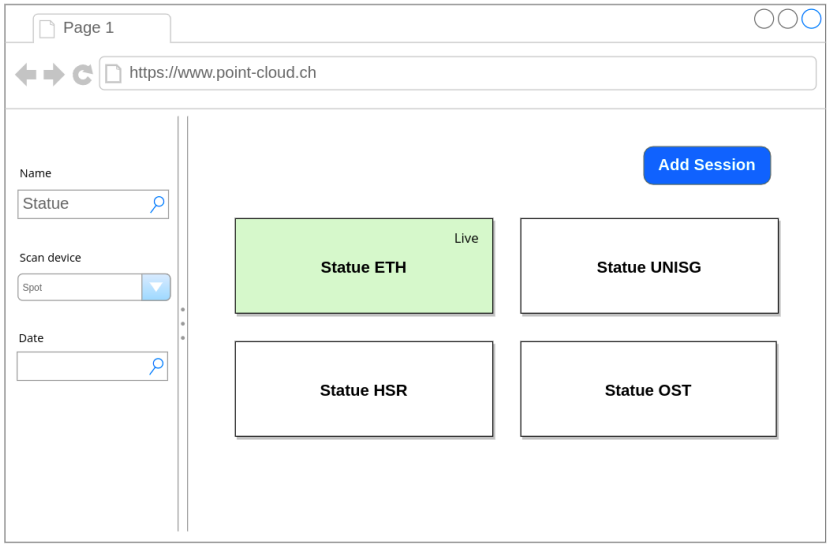


Abbildung 24: Mockup - Übersichtsscreen

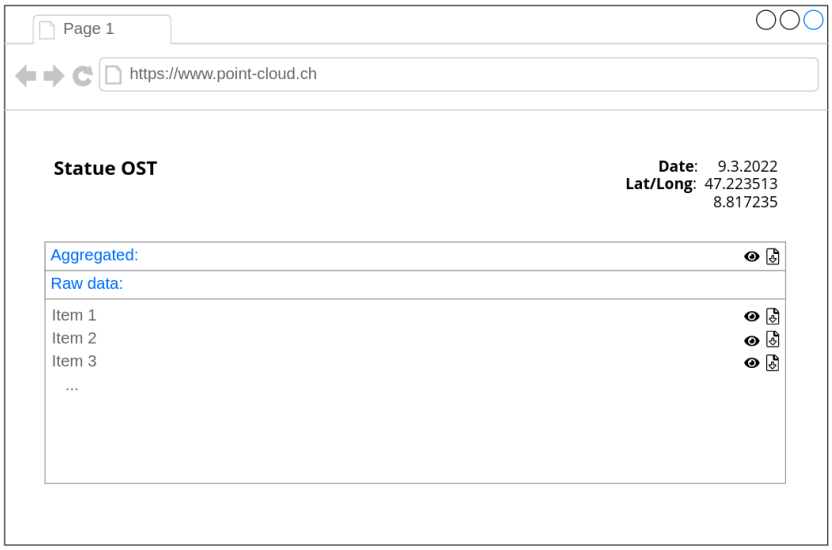


Abbildung 25: Mockup - Detailscreen

4 Realisierung

Die Applikation wurde in drei Software-Komponenten aufgeteilt. Namentlich dem Collector, Processor und Viewer. Dabei bilden der Collector und Processor mit der Datenbank das Backend und der Viewer das Frontend der Applikation.

4.1 Collector

Der Collector ist die zentrale Komponente des Backends. Er stellt die Endpoints zur Verfügung, kommuniziert mit der Datenbank, speichert Dateien auf dem lokalen Dateisystem und delegiert arbeiten dem Processor weiter.

Der Collector wurde mit Python und der Verwendung des Django Frameworks sowie zusätzlichen Libraries, welche unter 4.1.3 genauer beschrieben sind, implementiert. Die Verwendung und das Aufsetzen des Collectors sind im README des GitLab-Repositories genauer beschrieben.

4.1.1 Datenspeicherung

Die Metadaten der hochgeladenen Punktwolken oder Mesh-Dateien werden auf einer PostgreSQL-Datenbank gespeichert und die Dateien auf dem lokalen Dateisystem abgelegt.

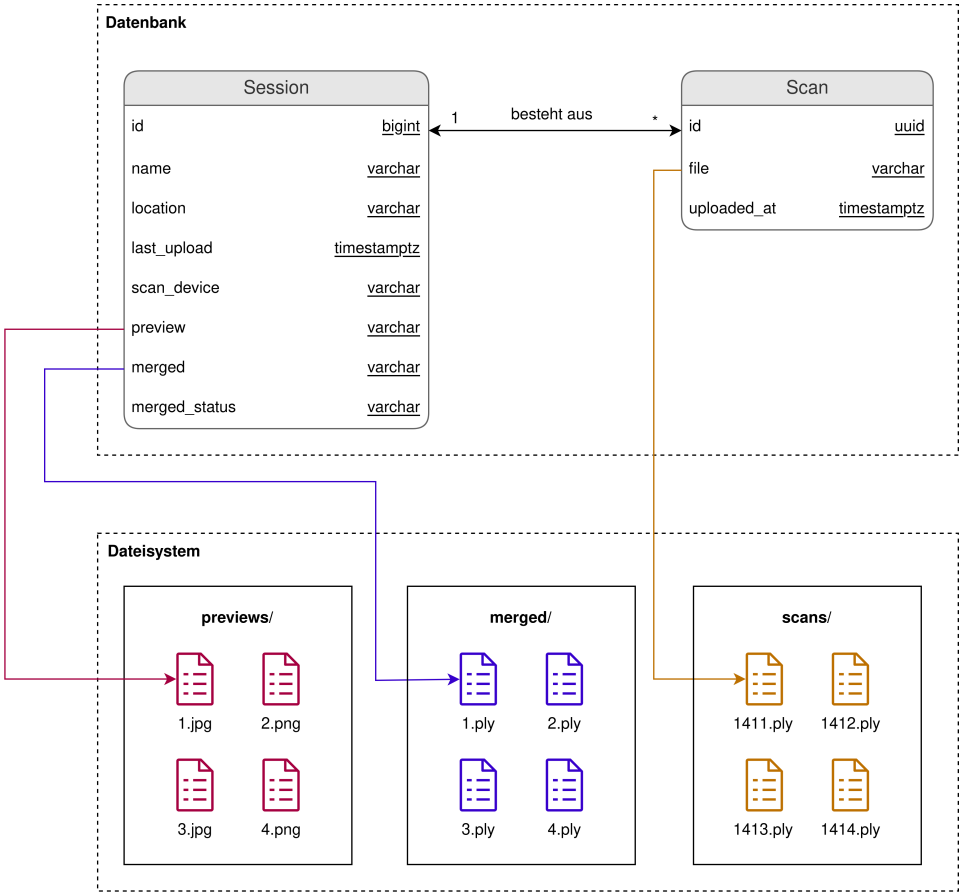


Abbildung 26: Zusammenspiel von Datenbank und Dateisystem

4.1.2 Beschreibung der Endpoints

Der Collector stellt Endpunkte nach den Prinzipien von REST zur Verfügung. Zum Beispiel können Informationen mit der GET-Methode abgefragt und neue Objekte mit der POST-Methode erstellt werden. In den folgenden Unterthemen werden die Endpoints zum Zusammenfügen mehrerer Scans zu einem und zum Hochladen eines Scans genauer beschrieben.

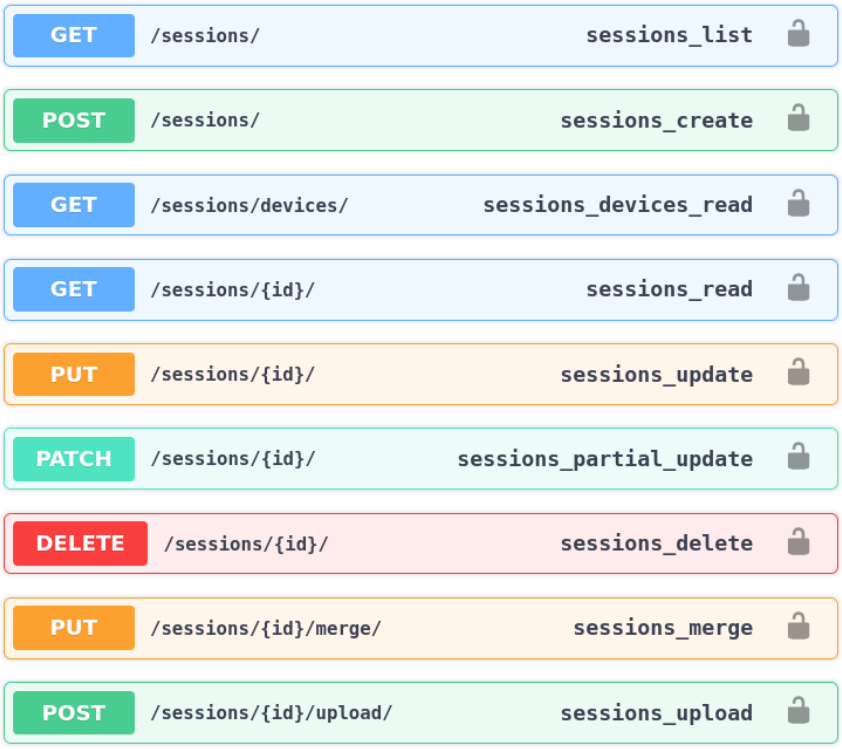


Abbildung 27: *OpenAPI* Spezifikation der Collector Endpoints

Die Endpoints sind nach der *OpenAPI* Spezifikation beschrieben und wie in Abbildung 27 ersichtlich. Diese wurde mit Tools von Swagger erstellt und ist unter dem URL-Pfad /swagger/ ersichtlich.

4.1.2.1 sessions_merge

Über den Endpoint *sessions_merge* können alle Scans einer Session zu einer einzigen Datei zusammengefügt werden. Im folgenden Sequenzdiagramm wird das Zusammenspiel der einzelnen Komponenten genauer dargestellt.

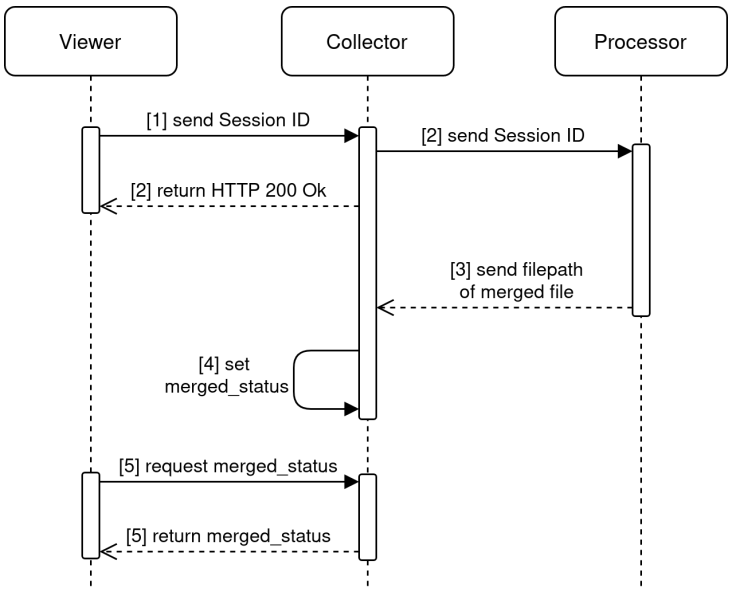


Abbildung 28: Sequenzdiagramm zum Zusammenfügen mehrerer Scans

Ablauf

1. Der Viewer sendet über den sessions_merge-Endpoint die Session ID, der Session, welche zusammengefügt werden soll
2. Der Collector sendet asynchron die Session ID weiter zum Processor. Damit der Viewer nicht auf eine Antwort warten muss, wird gleichzeitig dem Viewer mit dem HTTP Statuscode 200 geantwortet.
3. Der Processor fügt alle Scans der Session zu einer einzigen Datei zusammen und speichert diese lokal auf dem Dateisystem. Dem Collector wird mit dem Dateipfad, der eben zusammengefügten Datei, geantwortet
4. Anhand der Antwort vom Processor wird das Property merged_status entsprechend gesetzt, sowie die Metadaten der Session in der Datenbank aktualisiert.
5. Über das Property merged_status erfährt der Viewer den aktuellen Zustand

4.1.2.2 sessions_upload

Der *session_upload*-Endpoint erlaubt dem Benutzer Scans zu einer bereits bestehenden Session hinzuzufügen.

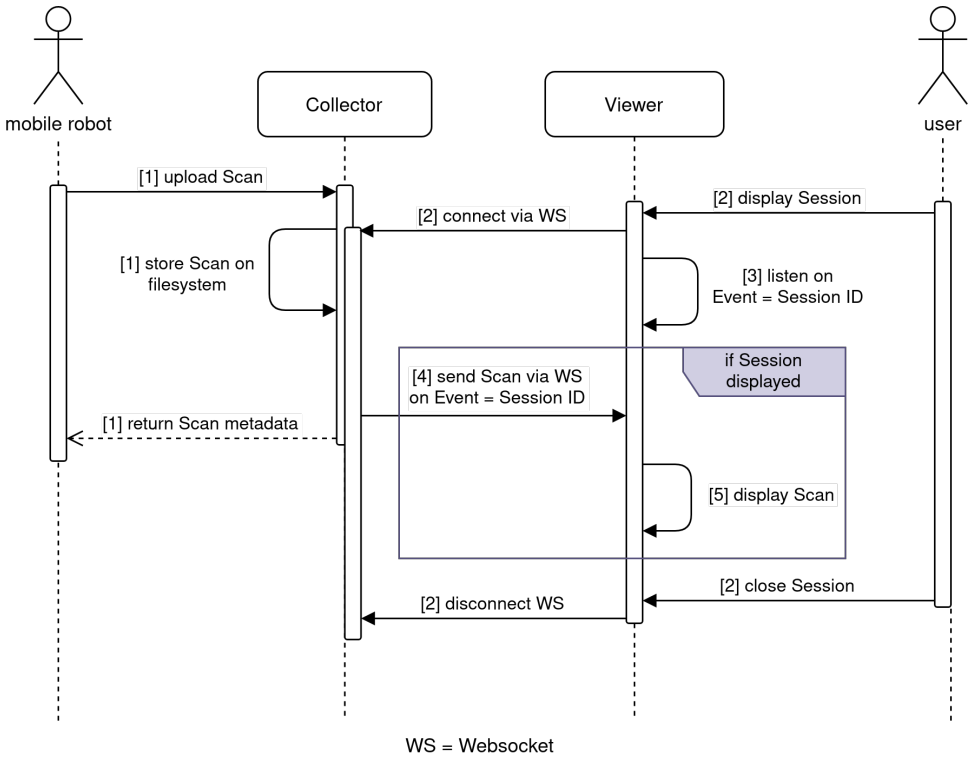


Abbildung 29: Sequenzdiagramm zum Hochladen eines Scans

Ablauf

1. Ein Actor lädt einen Scan über den *session_upload*-Endpoint hoch, welcher auf dem lokalen Dateisystem gespeichert wird. Dem Actor, wird mit den Metadaten des Scans geantwortet.
2. Wenn im Viewer eine Session dargestellt werden soll, wird eine *Websocket*-Verbindung mit dem Collector erstellt. Die Verbindung wird beim Schliessen der Darstellung wieder geschlossen.
3. Der Viewer registriert, dass er sich nur für Events mit der Session ID, der aktuelle dargestellten Session, interessiert.
4. Der Collector wird als multicast den erhaltenen Scan an alle *Websocket*-Verbindungen, welche für den Event mit der aktuellen Session ID registriert sind, weiterleiten.
5. Auf dem Viewer werden die Scans inkrementell dargestellt.

4.1.3 Verwendete Libraries

4.1.3.1 Django REST framework

Django REST framework ist ein mächtiges und flexibles Toolkit für das Erstellen von RESTful Web APIs mit *Django* (django REST framework, o. J.).

Es erlaubt unter anderem das Implementieren von Serializers für jede Model-Klasse. Mit diesen kann die JSON-Darstellung der Model-Klassen kontrolliert oder die erhaltenen Daten validiert werden, welche über die Endpoints versendet werden.

Zusätzlich können für die Views Klassen verwendet werden, welche Anhand der Models und der Serializers oft verwendete Endpoints zur Verfügung stellt. Wie zum Beispiel die CRUD-Operation einer Session.

Tabelle 14: Informationen - Django REST framework

Dokumentation	https://www.django-rest-framework.org/
GitHub	https://github.com/encode/django-rest-framework
Package Repository	https://pypi.org/project/djangorestframework/

4.1.3.2 python-socketio

Python-socketio ist eine Python Implementation von Socket.IO Clients und Servers (Grinberg, 2018). Mit der Library wird ein Socket.IO Server aufgesetzt, mit dem sich das Frontend verbinden kann, um über den aktuell hochgeladenen Scan informiert zu werden. Dabei wird beim Empfangen eines Scans ein Multicast, mit der Session ID als Event-Name, versendet. Der gesamte Ablauf ist in der Abbildung 29 genauer ersichtlich.

Tabelle 15: Informationen - python-socketio

Dokumentation	https://python-socketio.readthedocs.io/
GitHub	https://github.com/miguelgrinberg/python-socketio
Package Repository	https://pypi.org/project/python-socketio/

4.1.3.3 drf-yasg

DRF-YASG ist ein Akronym und heisst ausgeschrieben Django Rest Framework - Yet Another Swagger Generator. Diese Library wurde verwendet, um die in Abbildung 27 dargestellte *OpenAPI* Spezifikation der Endpoints zu erstellen.

Tabelle 16: Informationen - drf-yasg

Dokumentation	https://drf-yasg.readthedocs.io/en/stable/
GitHub	https://github.com/axnsan12/drf-yasg
Package Repository	https://pypi.org/project/drf-yasg/

4.2 Processor

Der Processor ist für das Zusammenfügen mehrerer Punktwolken oder Meshes zu einer Datei verantwortlich. Er besitzt einen Endpoint, welcher aber ausschliesslich über den Collector und nicht direkt aus dem Frontend angesprochen wird.

Der Processor wurde in Python und mit Hilfe des *Django* Frameworks, Django Rest Framework, sowie der *Open3D*-Library implementiert.

4.2.1 Separierung von Collector und Processor

Aktuell ist es mit dem Processor möglich mehrerer Scans zu einem grossen 3D-Modell zusammenzufügen. In Zukunft könnten zusätzliche Funktionen wie das Berechnen von Mesh-Strukturen implementiert werden. Diese Aufgaben sind je nach 3D-Modell sehr rechen- und zeitintensiv. Durch die Verwendung der *Open3D*-Library wird das Docker Container Image über 2.8 Gigabyte gross. Mit der Trennung von Collector und Processor kann selber entschieden werden, ob die zusätzlichen Funktionen, welche im Processor implementiert wurden, verwendet werden möchten. Ohne Processor Container kann eine sehr leichtgewichtige Lösung mit der Grundfunktionalität betrieben werden.

4.2.2 Beschreibung des Endpoints

Aktuell besitzt der Processor einen Endpoint, über welchen die Scans einer Session zusammengefügt werden. Nach dem Zusammenfügen eines 3D-Modells wird die entsprechende Datei auf dem Dateisystem abgelegt und der Pfad als Antwort retourniert.

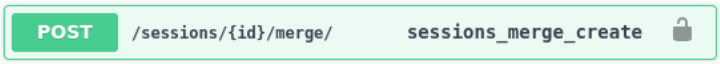


Abbildung 30: OpenAPI Spezifikation - Processor Endpoint

4.2.3 Verwendete Libraries

4.2.3.1 Django Rest Framework

Django Rest Framework wird für die Implementierung des einen Endpoints verwendet. Für weitere Informationen siehe 14

4.2.3.2 Open3D

Open3D ist eine Open Source-Library für die 3D-Daten Prozessierung (Zhou, Park & Koltun, 2018b). Die Library bietet eine Vielzahl an Funktionen, um 3D-Daten zu manipulieren. Im Processor Container wird die Library verwendet, um mehrere Scans zu einer Datei zusammenzufügen.

Tabelle 17: Informationen - *Open3D*

Dokumentation	https://www.django-rest-framework.org/
GitHub	https://github.com/encode/django-rest-framework
Package Repository	https://pypi.org/project/djangorestframework/

4.3 Viewer

Neben den beiden Backend Komponenten, die für die Verarbeitung und Verwaltung der 3D-Daten zuständig sind, wurde eine Frontend Komponente entwickelt. Über diese ist es möglich Punktwolken oder Meshes darzustellen, welche über den Collector hochgeladen wurden.

4.3.1 Übersicht

Die Viewer Applikation besteht aus zwei Teilen, die über die folgenden URL zu erreichen sind:

Tabelle 18: Viewer Komponenten

Pfad	Komponente
[base_url]/	Session Übersicht
[base_url]/**viewer/[sessionID]**	Modellansicht der entsprechenden Session

4.3.1.1 Session Übersicht

In der Übersicht werden alle aufgenommenen Sessions, inklusive Preview Bild, und den wichtigsten Metadaten angezeigt. Diese können per Suchfeld gefiltert und nach verschiedenen Kriterien sortiert werden. Über einen Dialog kann eine neue Session hinzugefügt, eine bestehende bearbeitet oder inklusive 3D-Daten gelöscht werden. Ebenfalls ist es möglich die 3D-Daten, über das Backend, zusammenzufügen. Diese zusammengefügte Datei kann später über den entsprechenden Button heruntergeladen werden.

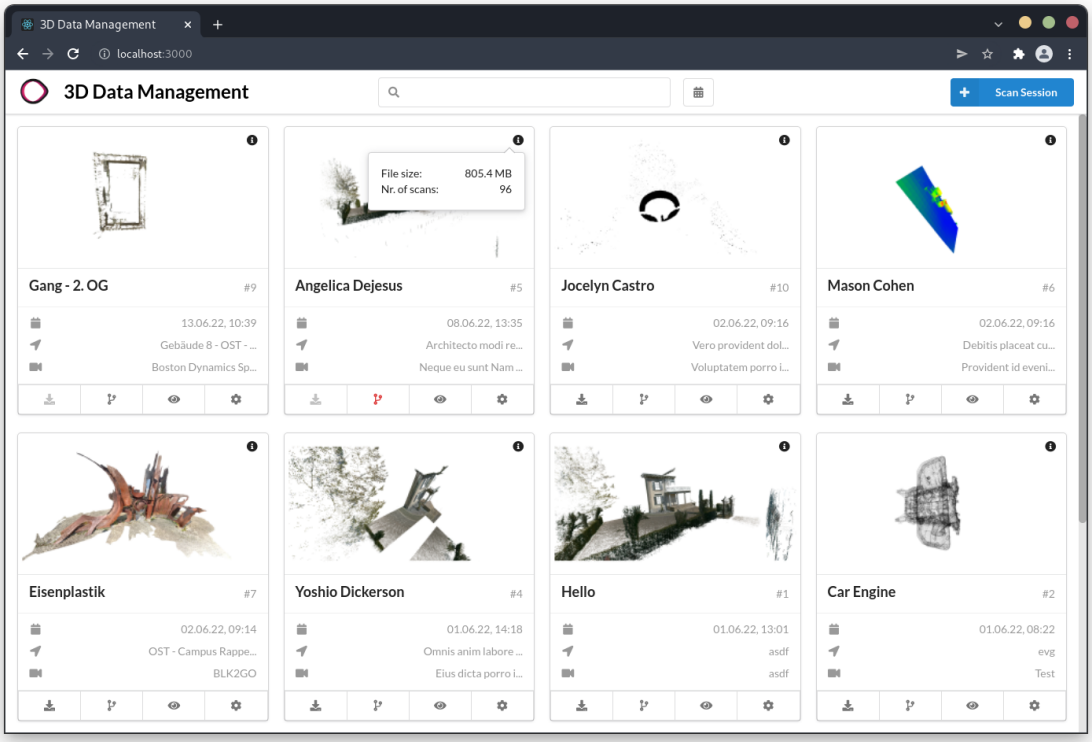


Abbildung 31: Screenshot - Viewer Übersicht

4.3.1.2 Modellansicht

In der Modellansicht werden die aufgenommenen Daten angezeigt und können von verschiedenen Seiten und Entfernung betrachtet werden. Zur Überprüfung wann der letzte Scan übermittelt wurde, wird jeweils der Zeitpunkt der letzten fünf Aufnahmen eingeblendet.

Über die Steuerelemente sind folgende Anpassungen am Modell möglich:

- Nur neuster Scan einblenden
- Grösse eines einzelnen Punkts anpassen
- Hintergrundfarbe anpassen
- Vorschaubild für Übersicht generieren

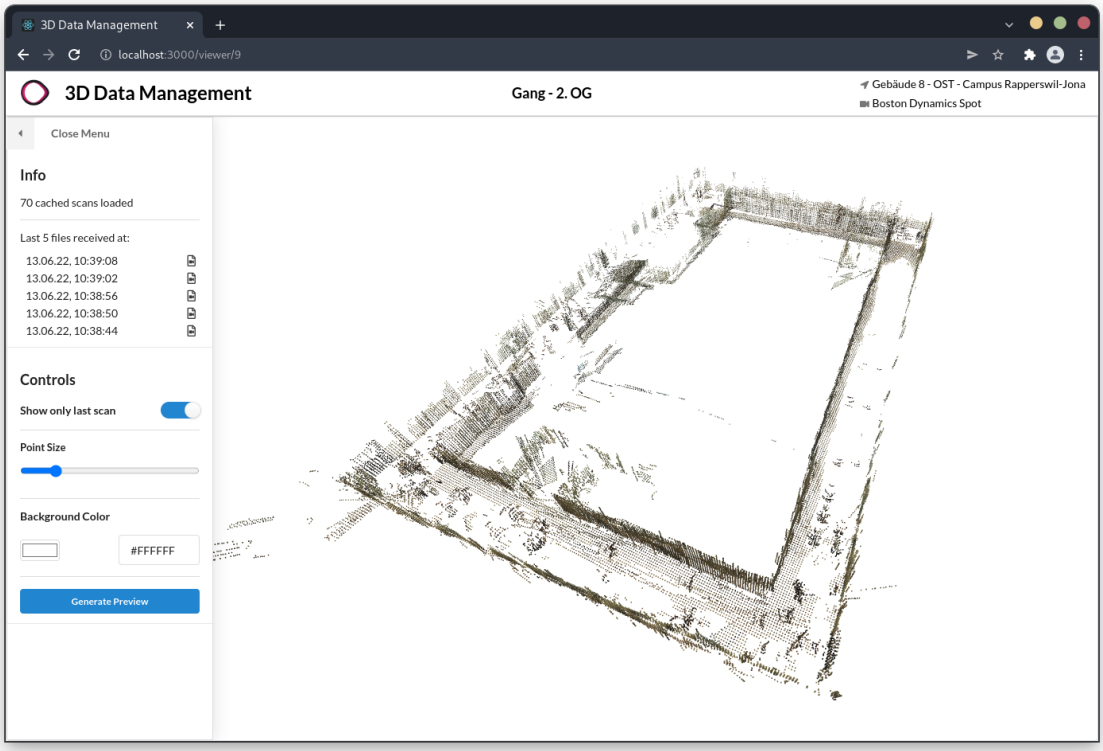


Abbildung 32: Screenshot - Viewer Modellansicht

4.3.2 Technologien

Die Grundstruktur des Frontend wurde mit Hilfe der *React*-Library implementiert. In der Evaluation Phase wurde entschieden, die 3D-Modelle mittels der 3D Computer Grafik Library *Three.js* darzustellen.

Für die einfachere und übersichtlichere Implementierung wurden verschiedene Libraries verwendet. Nachfolgend werden diese inklusive der Vorteile, welche sie mit sich bringen, beschrieben.

4.3.2.1 React Three Fibre

Das react-three-fibre Paket ist ein *React* Renderer für *Three.js*. React Three Fibre wandelt komponentenbasierte, deklarative Logik in pures *Three.js* JavaScript um. Dies führt zu deutlich übersichtlicherem Code und es kann bekannte, komponentenbasierte *React* Logik verwendet werden.

Tabelle 19: Informationen - React Three Fibre

Dokumentation	https://docs.pmnd.rs/react-three-fiber/getting-started/introduction
GitHub	https://github.com/pmndrs/react-three-fiber
Package Repository	https://www.npmjs.com/package/@react-three/fiber

4.3.2.2 Zustand

Wenn in *React* Daten zwischen verschiedenen Komponenten geteilt werden sollen, können diese direkt einer untergeordneten Komponente übergeben werden. Wenn dies nur vereinzelt oder direkt von Eltern zu Kind Komponente passiert, ist dies eine gute Lösung und der dazugehörige Code bleibt immer noch übersichtlich und gut lesbar. Bei komplexeren Beziehungen zwischen den Komponenten wird dies schnell unübersichtlich und oft müssen Daten über Zwischenkomponenten weitergegeben werden, weil die Zielkomponente nicht ein direktes Kind ist.

Um die oben beschriebenen Probleme zu lösen, wird in grösseren Applikationen oft eine State Management Library zur Hilfe genommen. Es wurde eine kurze Evaluation zwischen Redux, *React* Context API und Zustand gemacht. Aus folgenden Gründen wurde schlussendlich Zustand verwendet:

- Minimale Grösse (unter 1 kB)
- Zugriff auf State per Hooks
- Gleicher Entwickler wie react-three-fibre
- Weniger Boilerplate Code als bei Redux / Context API

Tabelle 20: Informationen - Zustand Library

Dokumentation	https://docs.pmnd.rs/zustand/introduction
GitHub	https://github.com/pmndrs/zustand
Package Repository	https://www.npmjs.com/package/zustand

4.3.2.3 Semantic UI

Semantic ist ein Entwicklungsframework, um ansprechende, responsive und benutzerfreundliche User Interfaces zu gestalten. Durch die Verwendung von Semantic UI kann viel Zeit im Design von Steuerelementen eingespart werden. Zusätzlich zum ansprechenden Design werden nicht-standardisierte Elemente wie Dropdown Elemente, Suchfelder oder spezielle Buttons zur Verfügung gestellt. Diese Elementen können als *React* Komponenten eingebunden werden, womit der Quellcode übersichtlich bleibt.

Tabelle 21: Informationen - Semantic UI

Dokumentation	https://semantic-ui.com/
GitHub	https://github.com/Semantic-Org/Semantic-UI
Package Repository	https://www.npmjs.com/package/semantic-ui-react

4.4 Deployment

Als Entwicklungsumgebung wurden alle Docker Container auf einem Server der *OST* betrieben. Für die kontinuierliche Integration wurden die entsprechenden Funktionen der *OST* GitLab Instanz verwendet.

4.4.1 Übersicht

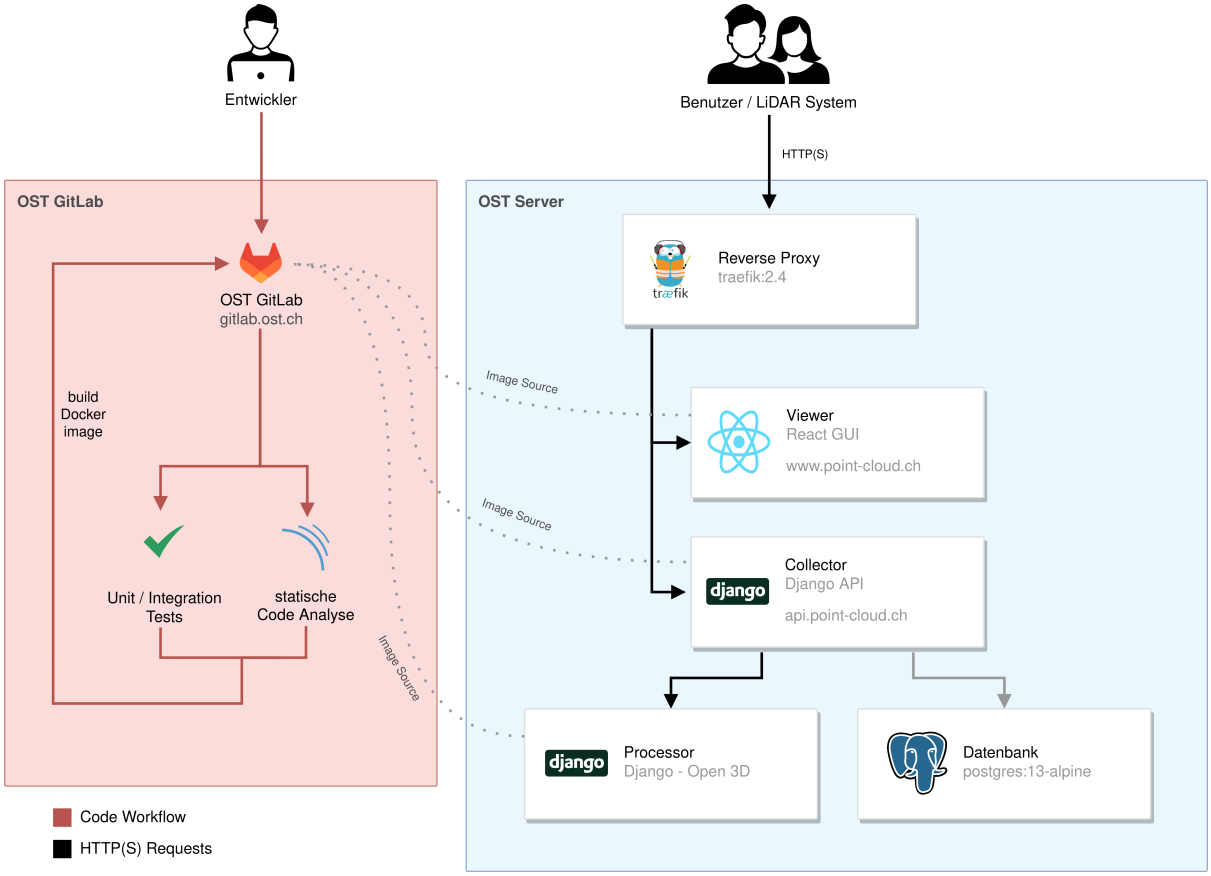


Abbildung 33: Deployment Diagramm

Server

Wie in den vorherigen Kapiteln beschrieben, wurden drei verschiedene Komponenten entwickelt, welche jeweils in einem eigenen Docker Container ausgeführt werden. Da diese alle auf dem gleichen Server lauffähig sein sollen, wurde ein Traefik Reverse Proxy verwendet, um die Requests an die jeweiligen Container weiterzuleiten.
Da alle Requests zum Processor über den Collector Container führen, ist dieser nur innerhalb des lokalen Docker Netzwerks erreichbar. Das gleiche gilt für die Datenbank.

GitLab

Bei jedem Push Befehl ins entsprechende Git-Repository werden die automatisierten Tests und eine statische Code-Analyse mit Hilfe von SonarQube durchgeführt. Wenn diese den Kriterien entsprechend erfolgreich ausgeführt werden, wird ein Docker Image erstellt und auf der GitLab Container Registry gespeichert. Beim entsprechenden Befehl wird dieses von dort heruntergeladen und auf dem Server ausgeführt.

4.4.2 Konfiguration

Die Konfiguration, welche vor dem Erstellen der Docker Images gemacht werden muss, wird in den jeweiligen README Dateien im entsprechenden Git-Repository beschrieben.

Die zur Verfügung gestellten *Docker Compose* Dateien bieten die Möglichkeit zu bestimmen, in welchem Ordner die 3D Dateien, Logs und statischen Dateien, gespeichert werden sollen.

Collector Volumes Konfiguration:

```
api:
  volumes:
    - [PFAD AUF SERVER]:/code/collector/media/
    - [PFAD AUF SERVER]:/code/logs/
    - [PFAD AUF SERVER]:/code/static/
```

Listing 1: *Docker Compose* Konfiguration - Collector

Processor Volumes Konfiguration:

```
proc:
  volumes:
    - [PFAD AUF SERVER]:/code/collector/media/
    - [PFAD AUF SERVER]:/code/logs/
```

Listing 2: *Docker Compose* Konfiguration - Processor

Der Pfad zum /media Ordner muss in beiden Containern gleich angegeben werden. Sonst kann der Processor nicht verwendet werden!

4.4.3 Lokales Deployment

Für das lokale Deployment ohne Domänenname wurde eine separate Docker-Compose Datei zur Verfügung gestellt. Dabei werden alle Requests auf /api an den Collector Container weitergeleitet. Alle anderen Pfade landen beim Viewer Container.

Tabelle 22: Lokales Deployment

Pfad	Ziel
/api	Collector
*****	Viewer

5 Qualitätssicherung

5.1 Softwaretest

Um allfällige Softwarefehler frühzeitig zu Erkennen und nach grösseren Refactorings sicherzustellen, keine Funktionen gelöscht zu haben, wurden die verschiedenen Teile ausführlich mit Unit und Integration Tests getestet.

5.1.1 Collector + Processor

Unit-Tests

Für die Unit-Tests im Backend wurde das Python Standard Library Module "unittest" verwendet. Neben der standardmässigen Integration in Python bietet sie zusätzliche Funktionen, wie zum Beispiel das erstellen von Mock-Objekten.

Integration-Tests

Für die Integration-Tests wurde die Hilfsklassen des Django REST frameworks verwednet. Damit können einfache Request ausgelöst und deren Antwort überprüft werden.

5.1.2 Viewer

Unit- und Integration-Tests

Alle Tests, die im Frontend automatisiert ausgeführt werden, wurden mit Hilfe des "Cypress Test Frameworks" umgesetzt. Cypress ist ein universelles Hilfsmittel, dass von Unit- bis zu Ende zu Ende Tests verwendet werden kann. Dadurch kann bei jedem Durchlauf ein Benutzer simuliert werden, der das gesamte User Interface bedient.

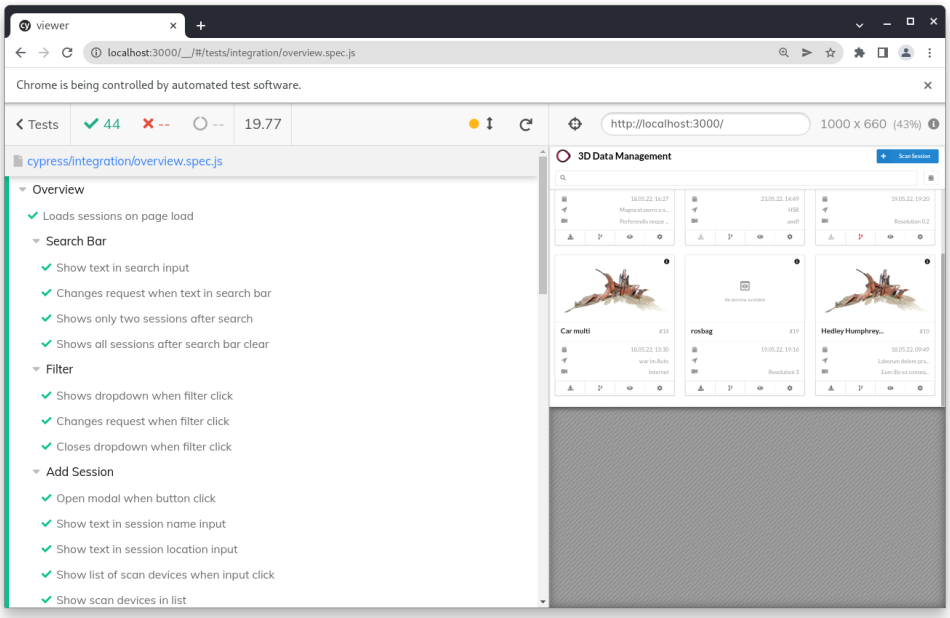


Abbildung 34: Screenshot der automatisierten Cypress Integration Tests

5.2 Statische Codeanalyse

Mit einer statischen Analyse von Quellcode können bestimmte Arten von Fehler entdeckt und die Qualität von Quellcode bestimmt werden. Zu schlechter Qualität von Quellcode führen zum Beispiel schlechte Namensgebung bei Variablen, lange unübersichtliche Methode oder doppelter Code.

Für die statische Codeanalyse wurde SonarQube verwendet. Es ist eine Open Source Plattform für die statische Code Analyse von 29 Programmiersprachen. Inklusive Python und TypeScript (SonarQube, 2021). Damit konnte eine Instanz verwendet werden, um alle Komponenten zu analysieren.

5.2.1 Reultat

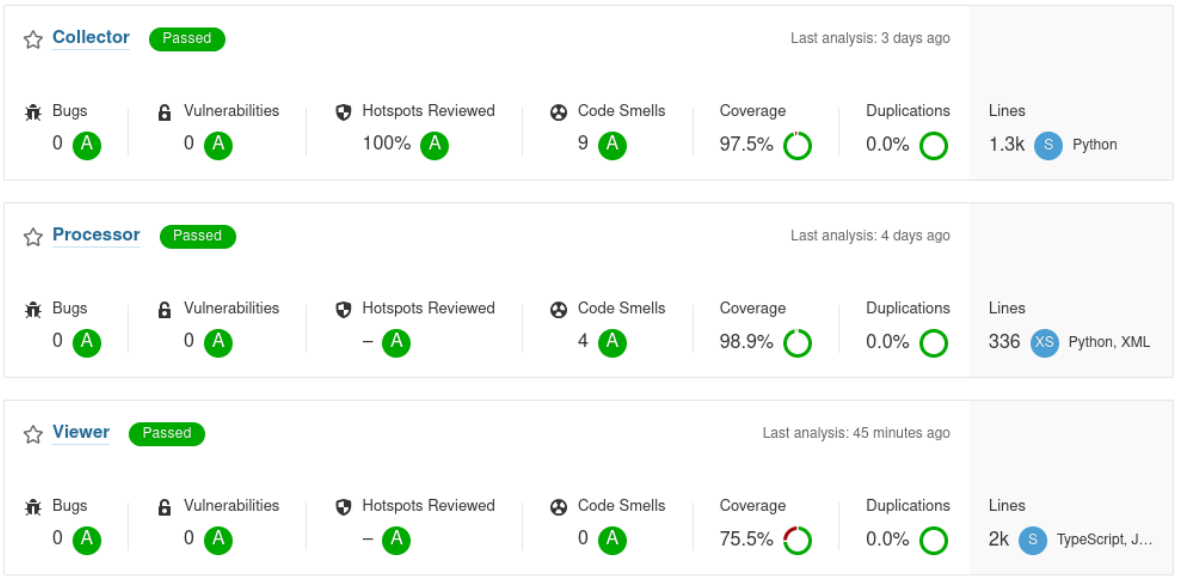


Abbildung 35: Resultat SonarQube

Komponente	Abdeckung	Anzahl Testsfälle
Collector	≈ 98 %	94
Processor	≈ 99 %	8
Viewer	≈ 76 %	44

5.3 Systemtests

5.3.1 Livetest

Um zu prüfen, ob das entwickelte Produkt wirklich funktioniert, wurde kurz vor Ende des Projekts ein Systemtest mit dem Roboterhund Spot des *ILT* durchgeführt. Dabei wurden mehrere Aufnahmen des Gangs im Gebäude 8 der *OST* durchgeführt.

Bis zu diesem Zeitpunkt wurden die Tests jeweils mit sogenannte *Rosbags* durchgeführt. Diese Dateien erlauben es zu einem späteren Zeitpunkt den Roboterhund Spot lokal auf einem Computer zu simulieren. Dadurch konnte die Funktionalität des gesamten Systems fortlaufend während der Entwicklung überprüft werden.

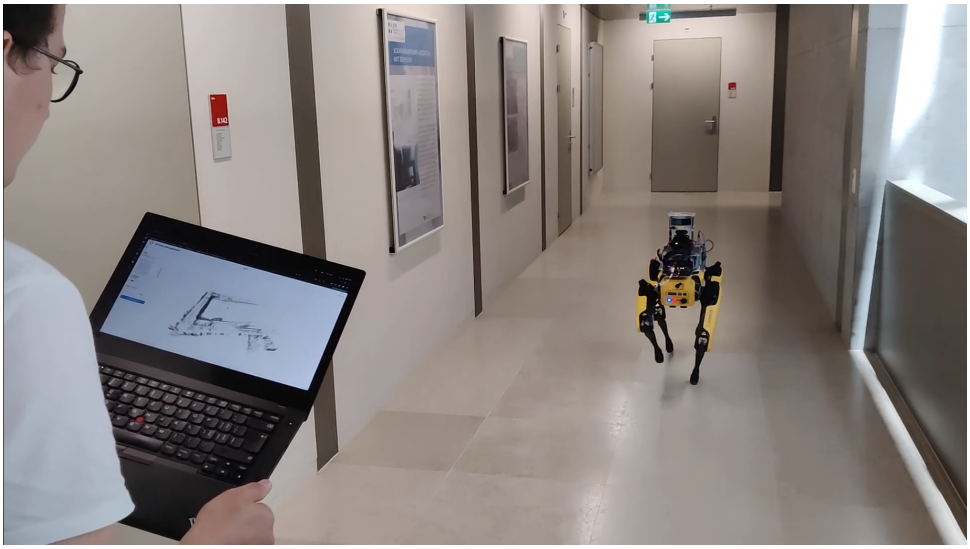


Abbildung 36: Systemtest - 1. Foto

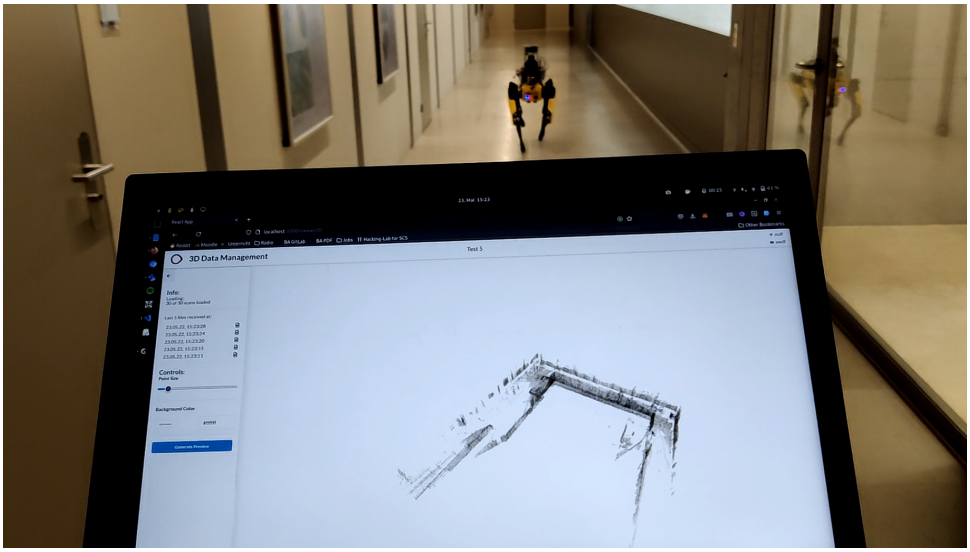


Abbildung 37: Systemtest - 2. Foto

5.3.2 Use Cases

Tabelle 23: Use Cases - Status

Nr.	Use Case	Erfüllt?
1.1	Benutzer kann CRUD Operationen für Sessions inklusive Metadaten durchführen	✓
1.2	Mobiler Roboter kann *PLY*-Dateien zu einer existierenden Session hochladen	✓
1.3	Mehrere Punktwolken können zu einer Datei zusammengefügt werden	✓
1.4	Vom mobilen Roboter hochgeladene *PLY*-Dateien werden inkrementell im Browser angezeigt	✓
1.5	Zusammengefügte Punktwolken / Meshes können herunterladen werden	✓
1.6	Benutzer kann in einer Session-Übersicht nach verschiedenen Metadaten suchen und filtern	✓
1.7	Benutzer kann ein Vorschaubild einer Punktwolke / Mesh generieren	✓
1.8	Benutzer kann sich authentisieren	
1.9	Session kann für alle, nur eine Gruppe oder nur einem Benutzer zugänglich gemacht werden	
1.10	Benutzer kann einen quadratischen Ausschnitt einer Punktwolke / Mesh herunterladen	
1.11	Benutzer kann über Browser Punktwolken / Meshes als Datei hochladen	

Das wichtigste Ziel der Arbeit war es die geforderten Muss-Kriterien zu erfüllen. Dabei war die Entwicklung der Grundfunktionalität mit Use Case "1.2 *PLY*-Datei hochladen" und "1.4 Liveansicht in Webapplikation" am aufwendigsten. Nach der erfolgreichen Implementierung dieser Use Cases wurde zusammen mit dem Projektpartner ein Refinement der offenen Punkte durchgeführt. Dabei wurde ein Vorschaubild als Use Cases mit dem grössten Nutzen bestimmt. Am Schluss fehlte leider die Zeit für die Umsetzung weiterer Funktionen.

6 Schlussfolgerung

6.1 Einschränkungen

In der vorgegebenen Projektdauer konnten viele Funktionen sehr zufriedenstellend umgesetzt werden. Bei folgenden zwei Problemen konnten aus Zeitgründen keine optimale Lösung gefunden werden.

6.1.1 Viewer Performance bei riesigen Punktwolken

Die Applikation hat bei riesigen Punktwolken (ab etwa 30 Millionen Punkten) Performance-Probleme, die sich negativ auf die Bedienung des Viewers auswirken. Die Reaktionszeiten werden dabei so gross, dass das angezeigte Modell nur unangenehm langsam bewegt werden kann. Da dieses Problem ebenfalls in der nativen Desktopanwendung MeshLab auftritt, wird davon ausgegangen, dass die wohl eleganteste Lösung eine Obergrenze für die maximal Anzahl der dargestellten Punkte ist.

6.1.2 Initiale Ausrichtung eines Modells

Wenn ein bereits gespeichertes 3D-Modell dargestellt wird, kann es vorkommen, dass dieses nicht zentriert oder zu gross im Viewer angezeigt wird. Diese Zentrierung wird aktuell mit Hilfe der react-three-fibre Library umgesetzt. Um ein optimales Ergebnis zu erzielen, müsste wahrscheinlich die Bounding Box eines Modells am Ende einer Aufnahme berechnet und in die Datenbank gespeichert werden.

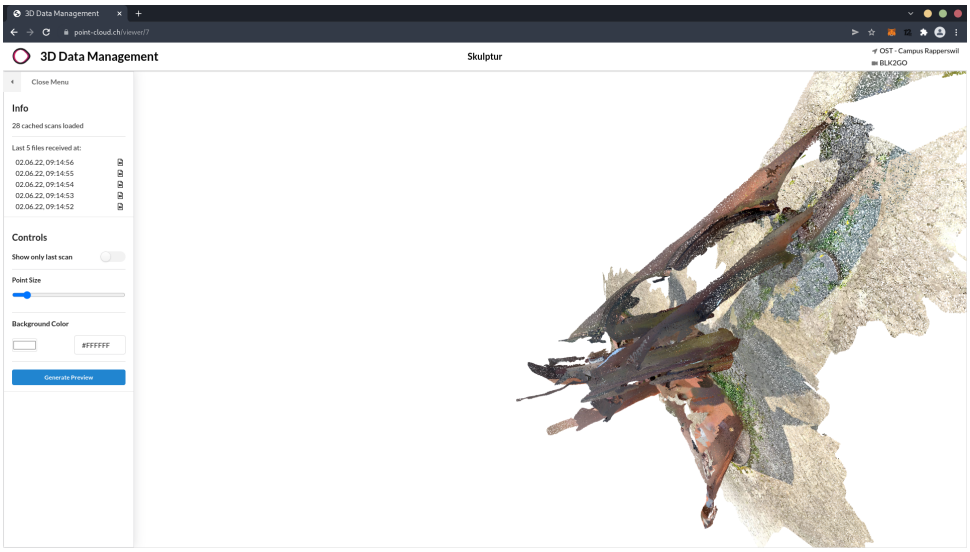


Abbildung 38: Screenshot - Probleme bei initialer Ausrichtung

6.2 Weiterentwicklung

6.2.1 Authentisierung

Bevor die Applikation öffentlich im Internet zur Verfügung gestellt werden soll, wäre es sinnvoll, ein Authentisierungssystem zu implementieren. Als Möglichkeit könnten Punktwolken als privat oder öffentlich hochgeladen werden. Zusätzlich wäre die Integration eines Social Logins wie zum Beispiel GitHub eine einfache Lösung zur Authentisierung.

6.2.2 Unterstützung zusätzlicher Dateiformate

Momentan ist lediglich der Upload und das Verarbeiten von Dateien im *PLY*-Format möglich. Um die Applikation generischer zu gestalten und den Benutzern mehr Freiheit zu bieten, könnte mit weiteren 3D-Objekt Dateiformaten erweitert werden. Bei der Evaluation von verschiedenen Libraries wurde bereits darauf geachtet, ob diese mehrere Dateiformate unterstützen.

6.2.3 Erweiterung des Processor

Der Processor wird zurzeit nur für das Zusammenfügen von mehreren *PLY*-Dateien zu einer *PLY*-Datei verwendet. Die dafür verwendete Library, *Open3D*, verfügt über viele zusätzliche Funktionalitäten für das Verarbeiten von 3D-Daten. Beispielsweise bietet die Library Funktionalitäten, um aus Punktwolken Meshes zu berechnen.

6.2.4 Unterstützung unterschiedlicher Auflösung

Die Applikation stellt aktuell jeweils alle hochgeladenen Punkte im Browser dar. Durch die beschränkte Leistung des Browsers ist bei sehr grossen Punktwolken mit Performance Problemen zu rechnen. Ein Ansatz zur Lösung dieses Problems wäre die Limitierung der Punktwolken auf eine Obergrenze an Anzahl Punkten. Hierfür könnte die Anzahl an Punkten nach dem Upload entsprechend reduziert werden.

6.3 Fazit

In dieser Arbeit wurde eine Applikation entwickelt, über die 3D-Daten verwaltet und dargestellt werden können. Für die Verwaltung der Dateien steht den Benutzer ein Dashboard aller hochgeladenen Punktwolken zur Verfügung, über welches Punktwolken dargestellt oder heruntergeladen werden können. Besonderer Wert wurde auf eine Liveansicht und das zugehörige inkrementelle Darstellen von aktuellen Aufnahmen eines *LiDAR* Scanners gelegt. Zusätzlich besteht die Möglichkeit ein Vorschaubild aus dem aktuell dargestellten Modell zu generieren.

Zu Beginn der Arbeit wurden verschiedene Ansätze für die Entwicklung der Applikation evaluiert. Dabei hat sich herausgestellt, dass die Kombination aus Dateiablage und Speicherung der Metadaten in einer Datenbank, sich am besten für die Verwaltung der Punktwolken eignen. Als Austauschformat wurde zusammen mit dem Projektpartner das *PLY*-Dateiformat ausgewählt. Ein Vorteil dieses Formats besteht in der Möglichkeit eines binären Speicherformats, womit die Dateigrösse klein bleibt, die Zugriffszeit aber dennoch schnell ist. Zusätzlich können *PLY*-Dateien sowohl Punktwolken wie auch Meshes enthalten. Als Library für die Darstellung der 3D-Daten im Browser hat sich *Three.js* als sehr geeignet erwiesen. Durch die unzähligen Möglichkeiten, welche die Library bietet, war eine Einarbeitung entsprechend aufwendig.

Die evaluierten Technologien haben sich für die gewünschten Anwendungsfälle als geeignet erwiesen. Die meisten Funktionen konnten damit wie gewünscht umgesetzt werden. Das grösste Problem stellen aktuell Performance Probleme mit der Darstellung von riesigen Punktwolken (ab etwa 30 Millionen Punkte) dar. Um dieses Problem zu lösen, müsste nach dem Upload einer Datei eine Reduktion der Anzahl Punkte durchgeführt werden. Durch diese Berechnung würde eine grössere Verzögerung zwischen Zeitpunkt der Aufnahme und Darstellung im Browser entstehen.

Mit dieser Arbeit kann ein wichtigen Beitrag zur Vision des ILT beigetragen werden. Die modulare Softwarearchitektur der Applikation bildet eine gute Grundlage für Weiterentwicklungen.

6.4 Danksagung

Wir bedanken uns bei...

Prof. Stefan F. Keller für die Betreuung und Unterstützung während der gesamten Arbeit.

Prof. Dr. Dejan Šeatović für das Erklären technischer Themen und die wertvollen Inputs.

Florin Kümin für die Unterstüzung bei verschiedenen Tests mit dem Roboterhund Spot.

Marius Zindel für das LaTeX-Template.

Glossar

Boston Dynamics Robotik-Unternehmen. 9, 10, 59

Django Python Webframework. 5, 41, 42
Docker Compose Tool zur Definierung von Multi-Container Applikationen. 48, 60, 72

Open3D Python Library für 3D-Daten Manipulation. 5, 32, 42, 54, 60
OpenAPI Standard zur Beschreibung von REST-Schnittstellen. 38, 41, 59
OpenStreetMap Open Source Geodaten-Projekt. 58
OST Ostschweizer Fachhochschule. 6, 9, 13, 14, 47, 51

PDAL Library für 3D-Daten Manipulation. 24, 32
pgPointcloud Punktwolken Extension für PostgreSQL. 25
Polygon Datei Format Dateiformat für Punktwolken oder Meshes. 58

React JavaScript Frontend Library. 5, 26, 45, 46
REST-API Paradigma für Softwarearchitektur von Schnittstellen. 5, 9, 17, 34
Robot Operating System (ROS) Set von Frameworks für Roboter-Softwareentwicklung. 57
rosbags Dateiformat zur Speicherung von *Robot Operating System (ROS)* Nachrichten. 51, 70

Three.js JavaScript Grafik Library. 5, 26, 29, 31, 45, 55, 59, 60

Websocket Kommunikations-Session zwischen Browser und Server. 5, 34, 40, 59

Akronyme

ACID Atomicity, Consistency, Isolation, Durability. 22

IFS Institut für Software. 9
ILT Institute for Lab Automation and Mechatronics. 4, 6, 9, 32, 51, 70

LiDAR Light Detection And Ranging. 4, 6, 7, 9, 10, 12, 14, 15, 17, 55, 59

OSM *OpenStreetMap*. 15, 19

PLY *Polygon Datei Format*. 5, 10, 15, 17, 18, 19, 23, 24, 25, 27, 28, 29, 30, 32, 52, 54, 55

SLAM Simultaneous Localization And Mapping. 4, 7, 9, 12

7 Verzeichnisse

7.1 Abbildungsverzeichnis

1 Boston Dynamics Spot mit Sensoreinheit . . . 4
2 Sensoreinheit mit *LiDAR*-Sensor und $360°$ Kamera (Kümin, 2022) . . . 4
3 Viewer Übersicht . . . 5
4 Darstellung eines Modells . . . 5
5 Systemtest mit dem Boston Dynamics Spot . . . 6
6 Darstellung der Punktwolke im Systemtest . . . 6
7 Spot von *Boston Dynamics* (Boston Dynamics, 2022b) . . . 10
8 Helix als Punktwolke (John Burkardt, 2012) . . . 11
9 Helix als Mesh (John Burkardt, 2012) . . . 12
10 Pointcloud Browser Übersicht (Nauli & Sennhauser, 2021) . . . 13
11 Illustration Sensoreinheit (Kümin, 2022) . . . 14
12 Use Case Diagramm . . . 16
13 Übersicht Punktwolken als Datei . . . 21
14 Übersicht Punktwolken in Datenbank . . . 22
15 Vergleich Speicherplatz . . . 23
16 Vergleich Zugriffszeit . . . 24
17 Screenshot - Potree . . . 27
18 Screenshot - 3DHOP . . . 28
19 Screenshot - *Three.js* . . . 29
20 Screenshot - Deck.gl . . . 30
21 Architektur Übersicht . . . 33
22 Polling vs. *Websocket* . . . 34
23 Übersicht Datenfluss . . . 35
24 Mockup - Übersichtsscreen . . . 36
25 Mockup - Detailscreen . . . 36
26 Zusammenspiel von Datenbank und Dateisystem . . . 37
27 *OpenAPI* Spezifikation der Collector Endpoints . . . 38
28 Sequenzdiagramm zum Zusammenfügen mehrerer Scans . . . 39
29 Sequenzdiagramm zum Hochladen eines Scans . . . 40
30 OpenAPI Spezifikation - Processor Endpoint . . . 42
31 Screenshot - Viewer Übersicht . . . 43
32 Screenshot - Viewer Modellansicht . . . 44
33 Deployment Diagramm . . . 47
34 Screenshot der automatisierten Cypress Integration Tests . . . 49
35 Resultat SonarQube . . . 50
36 Systemtest - 1. Foto . . . 51
37 Systemtest - 2. Foto . . . 51
38 Screenshot - Probleme bei initialer Ausrichtung . . . 53
39 Projektplan . . . 63
40 Zeit je Meilenstein . . . 66
41 Risikomanagement - 25.02.2022 . . . 68
42 Risikomanagement - 27.03.2022 . . . 69
43 Risikomanagement - 01.05.2022 . . . 70

7.2 Tabellenverzeichnis

1 Use Cases - Übersicht . . . 15
2 Nicht-funktionale Anforderungen . . . 20
3 Vergleich Speicherplatz - Messwerte in MB . . . 23
4 Vergleich Zugriffszeit - Messwerte in Sekunden . . . 25
5 Facts and Figures - Potree . . . 27
6 Technische Informationen - Potree . . . 27
7 Facts and Figures - 3DHOP . . . 28
8 Technische Informationen - 3DHOP . . . 28
9 Facts and Figures - *Three.js* . . . 29
10 Technische Informationen - *Three.js* . . . 29
11 Facts and Figures - Deck.gl . . . 30
12 Technische Informationen - Deck.gl . . . 30
13 Vergleich der Libraries für die 3D-Daten Verarbeitung . . . 32
14 Informationen - Django REST framework . . . 41
15 Informationen - python-socketio . . . 41
16 Informationen - drf-yasg . . . 41
17 Informationen - *Open3D* . . . 42
18 Viewer Komponenten . . . 43
19 Informationen - React Three Fibre . . . 45
20 Informationen - Zustand Library . . . 45
21 Informationen - Semantic UI . . . 46
22 Lokales Deployment . . . 48
23 Use Cases - Status . . . 52
24 Meilensteine . . . 63
25 Zeitplan . . . 65
26 Soll-Ist-Zeit-Vergleich . . . 66
27 Qualitätsmassnahmen . . . 67
28 Risiken im Detail . . . 71

7.3 Listings

1 *Docker Compose* Konfiguration - Collector . . . 48
2 *Docker Compose* Konfiguration - Processor . . . 48

7.4 Literaturverzeichnis

Boston Dynamics. (2022a). *Transformative Mobility* . https://www.bostondynamics.com/products/spot. (Abgerufen am 14. Juni 2022)

Boston Dynamics. (2022b). https://www.bostondynamics.com/sites/default/files/2021-10/spot-explorer-web-sm.png. (Abgerufen am 14. Juni 2022)

Cabello, R. (2021). *Three.js GitHub ReadMe.* https://github.com/mrdoob/three.js. (Abgerufen am 11. April 2022)

Callieri, M. (2016). *3DHOP GitHub ReadMe.* https://github.com/cnr-isti-vclab/3DHOP. (Abgerufen am 11. April 2022)

Chen, X. (2020). *Deck.gl Webseite.* https://deck.gl/. (Abgerufen am 5. Juni 2022)

Contributors, P. (2020, August). *Pdal point data abstraction library.* Zugriff auf https://doi.org/10.5281/zenodo.2556737 doi: 10.5281/zenodo.2556737

Dempsey, C. (2022). *Lidar Explained.* https://www.geographyrealm.com/lidar-explained/. (Abgerufen am 25. Mai 2022)

django REST framework. (o. J.). *django REST framework.* https://www.django-rest-framework.org/. (Abgerufen am 03. Juni 2022)

Grinberg, M. (2018). *python-socketio.* https://python-socketio.readthedocs.io/en/latest/. (Abgerufen am 26. Mai 2022)

John Burkardt. (2012). https://people.sc.fsu.edu/~jburkardt/data/ply/ply.html. (Abgerufen am 15. Juni 2022)

Kümin, F. (2022). Autonomous exploration and color mapping of dynamic environments.

MathWorks. (o. J.). *What is SLAM?* https://ch.mathworks.com/de/discovery/slam.html. (Abgerufen am 05. Juni 2022)

Nauli, D. & Sennhauser, N. (2021). *Geoprocessing mit Handheld-Laser-Scanner erfassten Point Cloud-Daten.* https://eprints.ost.ch/id/eprint/939/. (Abgerufen am 9. März 2022)

Petty, J. (o. J.). *What is a Polygon Mesh?* https://conceptartempire.com/polygon-mesh/. (Abgerufen am 05. Juni 2022)

Schütz, M. (2020). *Potree GitHub ReadMe.* https://github.com/potree/potree. (Abgerufen am 11. April 2022)

SonarQube. (2021). *Code Quality and code Scurity.* https://www.sonarqube.org/. (Abgerufen am 26. Mai 2022)

Zhou, Q.-Y., Park, J. & Koltun, V. (2018a). Open3D: A modern library for 3D data processing. *arXiv:1801.09847*.

Zhou, Q.-Y., Park, J. & Koltun, V. (2018b). *Open3D: A modern library for 3D data processing.* http://www.open3d.org/. (Abgerufen am 25. Mai 2022)

8 Anhang

8.1 Aufgabenstellung

Aufgabenstellung: Big Data Management von Punktwolken

- Bachelorarbeit im Frühlingssemester 2022 Bachelor Informatik
- Autoren: Reto Ehrensperger & Christian Rutzer
- Betreuer: Prof. Stefan Keller, IFS OST, Prof. Dejan Seatovic, ILT OST
- Industriepartner: -

Einleitung

Lidar-Sensoren (Abk. für Light detection and ranging) und SLAM-Methoden (Simultaneous Localization and Mapping) ermöglichen es mobilen Robotern - wie den (noch nicht wasserfesten) "Roboterhund" Spot der ILT OST mit NVIDIA Jetson Nano - gleichzeitig eine Karte seiner Umgebung (outdoor) erstellen wie auch seine Position innerhalb dieser Karte zu bestimmen.

Diese Karte – d.h. Punktwolken (Point Clouds) - schickt der Jetson Nano periodisch einem hier zu realisierenden Backend. Diese rechnet daraus eine grössere Karte der Umgebung und verwaltet sie. Typischerweise werden dabei die Punktwolken zu 3D Meshes (v.a. Dreiecksnetze) umgewandelt und in bestehende Daten desselben Zeitraums integriert.

Ziele

Ziel der Arbeit ist die Realisierung eines Backends, welches Punktwolken über eine REST-Schnittstelle entgegennimmt und dem Benutzer ermöglicht, diese über eine Webapplikation darzustellen.

Vorgehen

Es sollen folgende Teilschritte ausgeführt werden:

1. **Einarbeitung** in die BA “Geoprocessing mit Handheld-Laser-Scanner erfassten Point Cloud-Daten" von Sennhauser & Nauli (https://eprints.ost.ch/id/eprint/939/), sowie in entsprechende Technologien (z.B. MeshLab, CloudCompare, PDAL etc.) und in die Thematik der Punktwolken und Meshes. Vergleich verschiedener Möglichkeiten zur Verwaltung der Daten.
2. **Prototyp** für das Uploaden und Verwalten von Punktwolken.
3. **Realisierung** einer Webapplikation, bei der Benutzer mit den Punktwolken interagieren und diese darstellen können.
4. **Ausbau** der Webapplikation damit 5 bis 10 Poweruser mit der Webapplikation arbeiten können. Filtern der Punktwolken und zusätzlichen Features für ein benutzerfreundliches Interface.

8.2 Projektplan

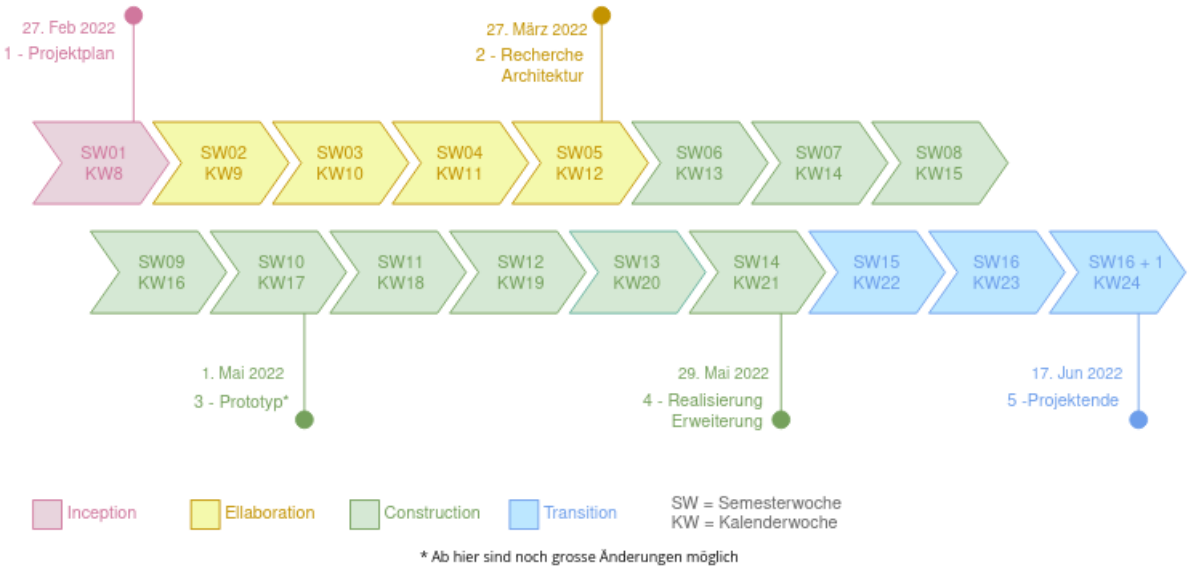


Abbildung 39: Projektplan

8.2.1 Meilenstein Details

Tabelle 24: Meilensteine

Nr.	Meilenstein	Datum	Produkte
1	Projektplan	27. Februar 2022	• Regeltermine definiert • Meilensteine sind definiert • Risikomanagement wurde erstellt • Zeitliche Planung wurde definiert • Qualitätsmassnahmen sind definiert
2	Recherche / Architektur	27. März 2022	• Funktionaler Scope mit Use Cases ist definiert • Nicht-funktionale Anforderungen wurden erfasst • Entwicklungs-Tools sind definiert und aufgesetzt • Aufgabenstellung ist fertig definiert • SW-Stacks evaluiert • Schnittstelle von ROS zu Backend evaluiert • Kern-Architektur des Gesamtsystem wurde verifiziert

3	Prototyp	1. Mai 2022	• Anzeige von 3D Meshes im Frontend ermöglichen • Senden von Punktwolken von NVIDIA Jetson Nano an Backend ist möglich • Zusammensetzen von Punktwolken auf dem Backend ist möglich
4	Realisierung / Erweiterung	29. Mai 2022	• Senden von Meshes von NVIDIA Jetson Nano an Backend ist möglich • Integration von bestehenden Daten ist auf dem Backend möglich • Performance Tests erstellt und evaluiert • Produkt fertig
5	Projektende	17. Juni 2022	• Schlussbericht geschrieben • Plakat erstellt • Abstract geschrieben • Broschüren-Abstract geschrieben • Management Summary geschrieben • Dokumentation fertigstellen • Präsentation vorbereitet

8.2.2 Zeitplan

Für die Bachelorarbeit werden 12 ECTS zugerechnet. Mit einem Arbeitsaufwand von 30 Stunden pro ECTS wird ein Gesamtaufwand von 720 Stunden erwartet. Diese werden wie folgt eingeplant:

Tabelle 25: Zeitplan

SW	KW	Datum von	Datum bis	Phase	Zeit
1	8	21.02.2022	27.02.2022	Inception	2x 20h
2	9	28.02.2022	06.03.2022	Elaboration	2x 20h
3	10	07.03.2022	13.03.2022	Elaboration	2x 20h
4	11	14.03.2022	20.03.2022	Elaboration	2x 20h
5	12	21.03.2022	27.03.2022	Elaboration	2x 20h
6	13	28.03.2022	03.04.2022	Construction	2x 20h
7	14	04.04.2022	10.04.2022	Construction	2x 20h
8	15	11.04.2022	17.04.2022	Construction	2x 20h
9	16	18.04.2022	24.04.2022	Construction	2x 20h
10	17	25.04.2022	01.05.2022	Construction	2x 20h
11	18	02.05.2022	08.05.2022	Construction	2x 20h
12	19	09.05.2022	15.05.2022	Construction	2x 20h
13	20	16.05.2022	22.05.2022	Construction	2x 20h
14	21	23.05.2022	29.05.2022	Construction	2x 20h
15	22	30.05.2022	05.06.2022	Transition	2x 20h
16	23	06.06.2022	12.06.2022	Transition	2x 20h
16 + 1	24	13.06.2022	17.06.2022	Transition	2x 40h

Total:	720h

8.2.3 Soll-Ist-Zeit-Vergleich

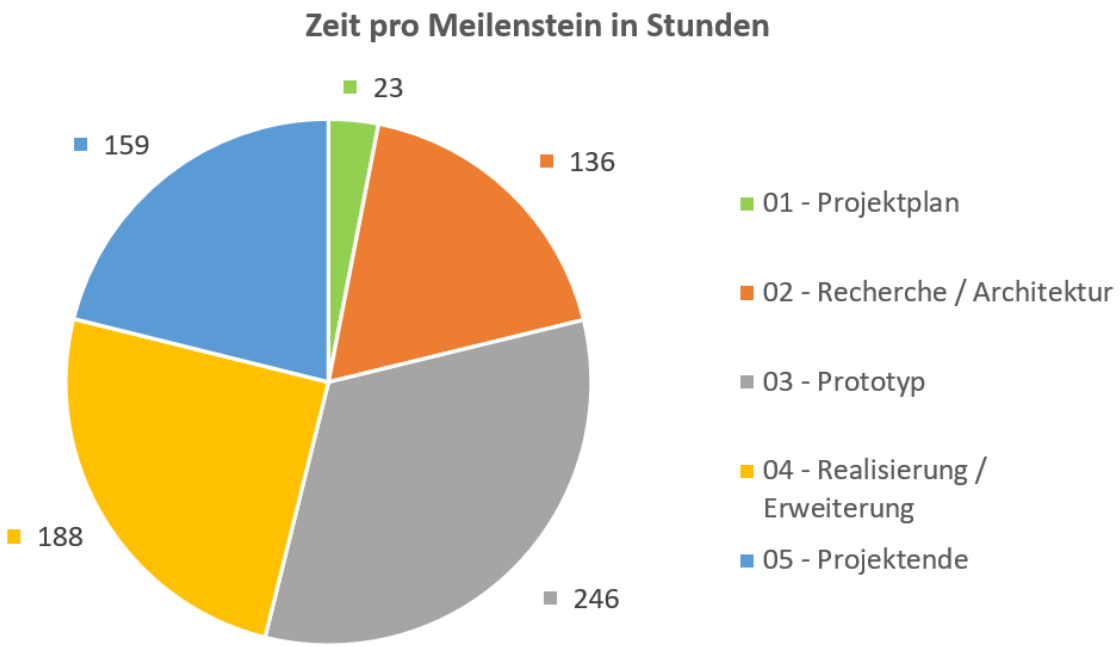


Abbildung 40: Zeit je Meilenstein

Tabelle 26: Soll-Ist-Zeit-Vergleich

Meilenstein	Soll	Ist
01 - Projektplan	40 h	23 h
02 - Recherche / Architektur	160 h	136 h
03 - Prototyp	200 h	246 h
04 - Realisierung / Erweiterung	160 h	188 h
05 - Projektende	160 h	159 h
Total	720 h	752 h

Aus der Tabelle ist ersichtlich, dass vorallem mehr Zeit für die Implementierung der Software aufgewendet wurde. Dies lässt sich daraus schliessen, dass Libraries verwendet wurden, welche noch nicht bekannt waren und somit das Knowhow zuerst erarbeitet werden musste.

8.2.4 Projektmanagement

Um das Beste aus dem agilen und klassischen Projektmanagement zu kombinieren, wurde Scrum+ als passende Methode ausgewählt. Scrum+ ist eine Kombination aus Rational Unified Process (RUP) und Scrum. Für die Langzeitplanung wird das Projekt in die vier Phasen von RUP (Inception, Elaboration, Construction und Transition) unterteilt. Innerhalb der Phasen werden die Funktionen iterativ am Prinzip von Scrum entwickelt.

8.2.5 Qualitätsmassnahmen

Tabelle 27: Qualitätsmassnahmen

Massnahme	Zeitraum	Ziel
Besprechung mit Betreuer / Projektpartner	Wöchentlich	Austausch über Projektstand, Fehler frühzeitig erkennen, Probleme zeitnah lösen
Automatische Software Tests	Merge Request	Funktionalität von Code sicherstellen
Continous Integration	Merge Request	Verhindern von "Integration hell"
Code Reviews	Merge Request	Sicherstellen von Clean Code, Verständnis der Software fördern
Definition of done	Issue	Sicherstellen von Verschiedenen Massnahmen

8.2.6 Definition of Done

- Automatische Tests erfolgreich
- CI erfolgreich durchlaufen
- Dokumentation aktualisiert
- Coding Guidelines eingehalten
- Code review gemacht

8.3 Risikomanagement

Beim Abschluss eines Meilensteins werden die Risiken evaluiert, entfernt oder wenn nötig neue Risiken hinzugefügt. Dabei werden je nach Einschätzung folgende Massnahmen getroffen:

- akzeptabel: keine Aktion notwendig
- ALARP (as low as reasonably practicable): Risiko muss laufend beobachtet werden
- inakzeptabel: Risiko muss durch Gegenmassnahmen verkleinert werden

8.3.1 Risiken Stand 25.02.2022

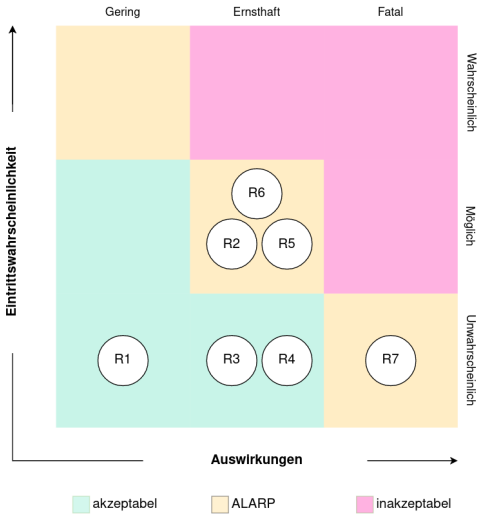


Abbildung 41: Risikomanagement - 25.02.2022

8.3.2 Risiken Stand 27.03.2022

- (R2) Verringerung der Eintrittswahrscheinlichkeit: Recherche wurde mit Betreuer und Projektpartner besprochen
- (R4) Risiko entfernt: Dieses Risiko existierte nie. Während der Entwicklung hat der Verlust von Daten auf dem Server keine Auswirkung.
- (R5) Abstufung der Auswirkung: Erhalt von Punktwolken, welche bei der Implementierung als versuche verwendet werden können

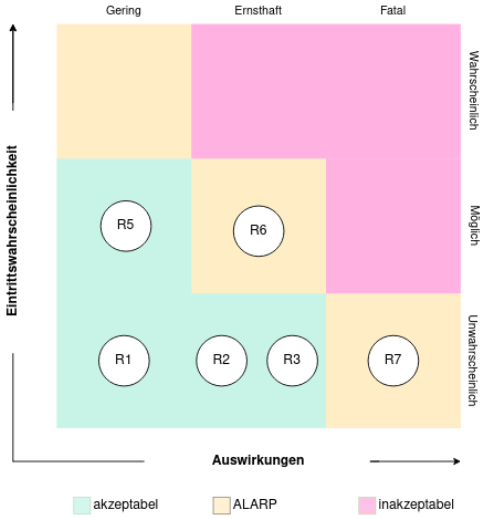


Abbildung 42: Risikomanagement - 27.03.2022

8.3.3 Risiken Stand 01.05.2022

- (R1) Verringerung der Eintrittswahrscheinlichkeit: Prototyp wurde präsentiert und Anforderungen sind klar
- (R3) Verringerung der Eintrittswahrscheinlichkeit: Prototyp Präsentation hat gezeigt, dass gewünschte Anforderungen implementiert werden
- (R5 & R6) Abstufung der Auswirkung und Eintrittswahrscheinlichkeit: Beim Besuch beim *ILT* konnten *Rosbags* aufgenommen werden, wodurch der Spot lokal simuliert werden kann
- (R7) Abstufung der Auswirkung: Applikation ist dockerisiert, womit sie ohne grossen Aufwand lokal aufgesetzt werden kann

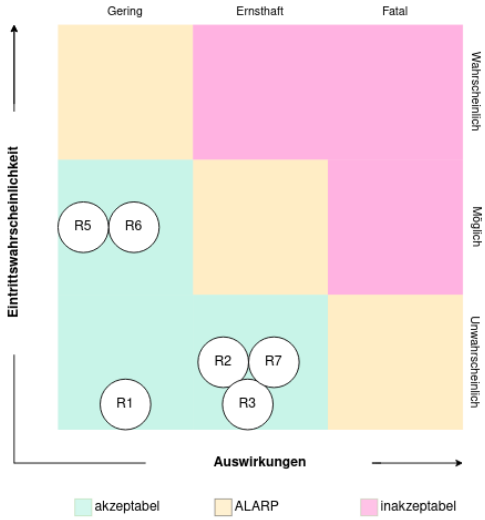


Abbildung 43: Risikomanagement - 01.05.2022

8.3.4 Risiken Stand 29.05.2022

Es wurden keine Risken mehr evaluiert.

8.3.5 Beschreibungen der Risiken

Tabelle 28: Risiken im Detail

Nummer	Beschreibung	Gegenmassnahmen
R1	Anforderungsänderungen des Auftraggebers	Es werden wöchentliche Sitzungen mit dem Betreuer durchgeführt. Verwendung von SCRUM+ (RUP + SCRUM).
R2	Annahmen nicht zutreffend, z.B. unzutreffende Recherche / Analyse	Die Recherchen und Analyse werden am Anfang des Projektes gemacht und bei Unklarheiten mit den Betreuern besprochen.
R3	Grosse Abweichungen gegenüber den Anforderungen	Es werden wöchentlich Sitzungen mit dem Betreuer durchgeführt. Zudem wird bei Halbzeit des Projektes den aktuellen Stand präsentiert.
R4	Datenverlust	Es werden wöchentliche Backups der bestehenden Daten erstellt.
R5	Roboterhund Spot ist nicht verfügbar oder fällt aus	Es kann ein Handheld-Laser-Scanner verwendet werden oder mit Beispiel-Dateien gearbeitet werden.
R6	Defekte oder Ausfall der zusätzlichen Infrastruktur (ROS oder NVIDIA Jetson Nano) auf dem Roboterhund	Es kann ein Handheld-Laser-Scanner verwendet werden oder mit Beispiel-Dateien gearbeitet werden.
R7	Ausstieg der Server Infrastruktur	Alle Files werden in einem Versionierungstool abgespeichert.

8.4 Manual

8.4.1 Einzelne Komponenten

Die Installationsanleitung der jeweiligen Komponenten ist als README-Datei im jeweiligen GitLab-Repository abgelegt.

8.4.2 Gesamtapplikation

Für das Deployment der Gesamtapplikation wurden verschiedene *Docker Compose* Konfigurationen erstellt und im Repository "Scripts and Deployment" abgelegt.

8.4.3 Script für Upload von 3D-Daten

Das Python-Script, mit welchem Punktwolken von einem mobilen Roboter an die Applikation übermittelt werden kann, ist im "Scripts and Deployment" Repository abgelegt.