

Dynamic Rail Map

Projektdokumentation – Gruppe 43

Alicia Gleichmann <alicia.gleichmann@stud.tu-darmstadt.de>
Greys Fankyeyeva <greys.fankyeyeva@stud.tu-darmstadt.de>
Leonhard Steinecke <leonhard.steinecke@stud.tu-darmstadt.de>
Severin Pelikan <severin.pelikan@stud.tu-darmstadt.de>
Simon Riese <simon.riese@stud.tu-darmstadt.de>

Teambegleitung:
Maximilian Julius Höck <maximilian.hoeck@stud.tu-darmstadt.de>

Auftrag:
M.Sc. Alexander Kroth <alexander.kroth@tu-darmstadt.de>
Fachbereich für Fahrzeugtechnik - FZD (FB16)

31. März 2024



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Bachelor-Praktikum
Wintersemester 2023/24
Fachbereich Informatik

Inhaltsverzeichnis

1 Projektbeschreibung	3
2 Qualitätssicherung	6
2.1 Qualitätsziel 1: Wartbarkeit	6
2.1.1 Erweiterbarkeit	6
2.1.2 Lesbarkeit	7
2.2 Qualitätsziel 2: Kompatibilität	9
2.3 Qualitätsziel 3: Effizienz	10
3 Abweichende QS-Ziele	14
4 Projektverlauf und projektgefährdende Ereignisse	15
5 Softwarelizenz	16

1 Projektbeschreibung



Abbildung 1.1: Bild der InnoTram, bereitgestellt von Alexander Kroth, 2023

Motivation und Kontext: Unser Bachelor-Praktikum ist Teil des Forschungsprojekts **IKaS** („Intelligenter Kreuzungsassistent für Straßenbahnen auf Basis von Intentionsschätzung“) des Fachgebiets Fahrzeugtechnik (FZD) der Technischen Universität Darmstadt.

Im Rahmen des IKaS-Projekts wird an Methoden zur Reduktion von Unfällen mit Straßenbahnbeteiligung in Kreuzungs- und Einmündungsbereichen geforscht. Dies soll durch ein intelligentes Fahrassistenzsystem umgesetzt werden, welches das Verhalten anderer Verkehrsteilnehmer*innen antizipieren und so auf mögliche Kollisionen reagieren kann.

Zum Entwickeln und Trainieren dieses Systems werden Daten aus dem Straßenverkehr benötigt, welche vom InnoTram Projekt bereitgestellt werden. Die InnoTram ist die Forschungstram der HEAG mobilo GmbH und verfügt über viele zusätzliche Sensoren, mit denen sie die benötigten Trainingsdaten im Darmstädter Straßenverkehr aufnehmen kann. Um das Fahrassistenzsystem auf das Erkennen anderer Verkehrsteilnehmer*innen trainieren zu können, benötigen die Forscher*innen ein Abbild der statischen Umgebung des Darmstädter Schienennetzes als Referenz.

Ein*e Forscher*in des IKaS-Projekts wird das von uns implementierte System auf den gesammelten Daten

mehrerer Fahrten der InnoTram ausführen, um diese Referenz-Karte zu erhalten.

Projektziele: Unser System soll nun die oben genannte Referenz-Karte als eine punktbasierte 3D-Karte aus den LiDAR-Daten der InnoTram erstellen. LiDAR steht für „Light detection and ranging“ und kann vereinfacht als eine Kamera beschrieben werden, die anstelle von Farben Entfernung aufzeichnet.

Der Ablauf der Kartierung kann in drei Unterziele aufgeteilt werden:

Zuerst müssen die LiDAR-Daten mehrerer Fahrten angepasst und kombiniert werden, um eine kontinuierliche Karte zu erhalten.

Aus dieser müssen im nächsten Schritt alle dynamischen Objekte entfernt werden. Als dynamisches Objekt zählt alles, was sich während der Aufnahme bewegt. Zum Beispiel fahrende Autos oder auch Fußgänger*innen, die über den Luisenplatz laufen.

Im darauffolgenden Schritt sollen auch alle semi-dynamischen Objekte entfernt werden. Dabei handelt es sich um Objekte, die in den Aufnahmen oft an denselben Stellen stehen, aber nicht verlässlicher Bestandteil der statischen Umgebung sind. Beispiele dafür sind Autos, die an roten Ampeln halten, Personen, die an Haltestellen warten oder auch die Stände des Weihnachtsmarkts.

Der Fokus unseres Bachelor-Praktikums liegt besonders darauf, ein System zu entwickeln, welches den ersten Schritt, also die Erstellung einer kontinuierlichen Karte, sauber umsetzt. Für das Entfernen dynamischer und semi-dynamischer Objekte soll vor allem Vorarbeit geleistet werden, eine konkrete Umsetzung ist aber nicht gefordert.

Umsetzung: Dieses Format bietet ein modulares System, um Daten von mehreren Sensoren zusammen mit genauen Zeitstempeln des Aufnahmezeitpunkts in einer Datei zu speichern.

Das erste Unterziel (die Kombination der LiDAR-Daten über mehrere Fahrten) wird mit dem von uns implementierten CLI-Programm „Scan Combiner“ realisiert. Dieser besteht aus mehreren Komponenten, die alle in einer stark parallelisierten Pipeline integriert sind.

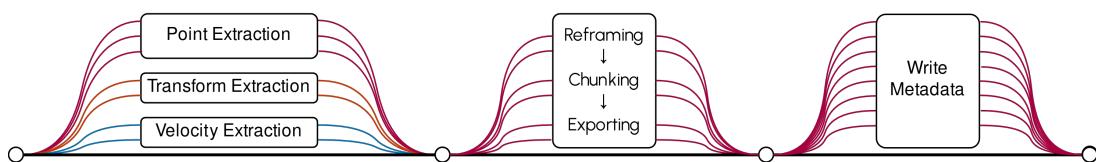


Abbildung 1.2: vereinfachte Programm Pipeline des „Scan Combiner“s

Im ersten Schritt (die „Extraction“-Blöcke links in der Grafik) werden die relativ zur Bahn aufgenommenen Punkt-Entfernungen sowie die mit Zeitinformationen versehenen Positionen, Rotationen und Geschwindigkeiten einer Fahrt geladen und gefiltert.

Danach nutzt der „Scan Combiner“ die erwähnten Zeitstempel zusammen mit Splineinterpolation, um zu ermitteln, wo genau sich die Bahn zum Zeitpunkt der Entfernungsmessung befunden hat.

Durch die Kombination dieser Position mit der Entfernungsmessung kann nun der „Scan Combiner“ den vom LiDAR getroffenen Punkt einer Oberfläche in die globale 3D-Karte transformieren.

Im darauffolgenden Schritt wird die entstandene Punktwolke in quadratische Zellen aufgeteilt. Die Aufteilung bietet den Vorteil, dass einzelne Teile der Karte geladen und verarbeitet werden können, ohne mehrere Milliarden Punkte auf einmal laden zu müssen.

Im finalen Schritt werden die Zellen in separaten Ordnern mit den dazugehörenden Metadaten gespeichert. Die Metadaten umfassen Informationen darüber, aus welchen Datensätzen die Punktwolken entstanden sind, von welchen Positionen aus die Punkte aufgenommen wurden, sowie von welcher Version und mit welcher Konfiguration diese vom „Scan Combiner“ verarbeitet wurden.

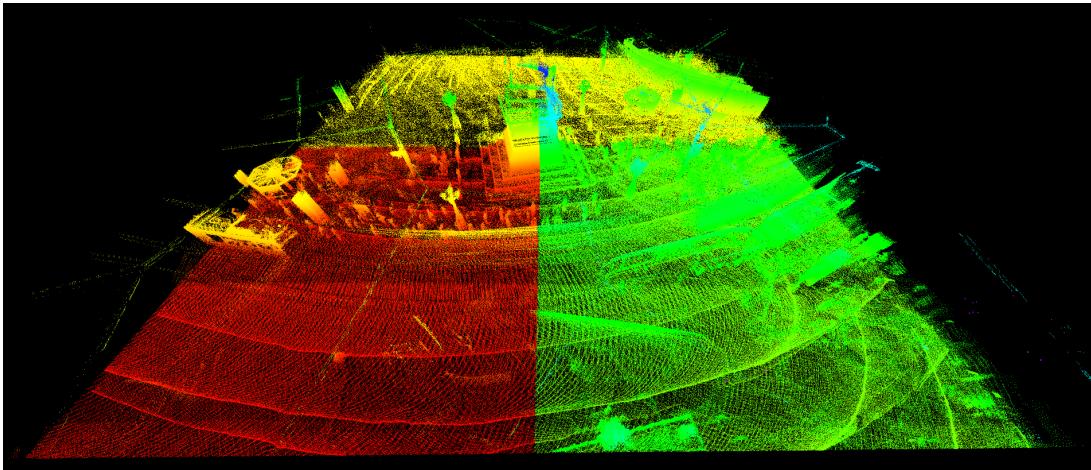


Abbildung 1.3: Ansicht einiger vom „Scan Combiner“ erstellten Zellen rund um das Ludwigsmonument.

Für das Entfernen dynamischer Objekte haben wir uns nach umfangreicher Recherche, für zwei externe Projekte entschieden: „Removert“ und „3DTK“ der Universität Würzburg. Da beide Programme eigene Vor- und Nachteile mit sich bringen, haben wir für diese beiden Schnittstellen erstellt und in unsere Pipeline integriert.

Für die Entfernung von semi-dynamischen Objekten haben wir ein Grundkonzept erarbeitet, das auf Filterung mittels voxelbasierten 3D-Masken beruht, wobei Voxel Volumenelemente sind, sozusagen das 3D äquivalent zu Pixeln. Voxelbasierte 3D-Masken werden durch eine Sammlung dieser Voxel repräsentiert und können so Diskretisierungen beliebiger dreidimensionaler Formen darstellen.

In diesem System kann ein*e Nutzer*in mithilfe von Domäne Wissen in einem Voxeleditor volumetrische Masken erstellen. Punkte, welche in diese Masken fallen, werden dann automatisch aus den Punktwolken entfernt.

Technologien: Da das gesamte IKaS-Projekt auf Ubuntu 20.04 mit ROS1 (einem Framework zur Entwicklung von Roboter Systemen) basiert, liegen diese gesammelten Daten in dem vom Framework bereitgestellten Rosbag-Format vor.

Dieses Format bietet ein modulares System, um Daten von mehreren Sensoren zusammen mit genauen Zeitstempeln des Aufnahmezeitpunkts in einer Datei zu speichern.

Um die von ROS1 bereitgestellten Bibliotheken zum Lesen dieser Rosbags nutzen zu können und die extrahierten Daten performant verarbeiten zu können, haben wir uns für C++ 17 als Programmiersprache entschieden. Wir nutzen das PCD-Format des weitverbreiteten **PCL** (Point Cloud Library) Frameworks, um die Kompatibilität der 3D-Karte mit möglichst vielen Schnittstellen zu ermöglichen.

Aus dem gleichen Grund werden die Metadaten als JSON-Dateien gespeichert. Hier wird die Bibliothek RapidJSON verwendet, um schnelles Lesen und Schreiben von den Dateien zu gewährleisten.

2 Qualitätssicherung

Das IKaS-Projekt beschäftigt sich mit der Reduktion von Unfällen mit Straßenbahnbeteiligung, also hat die Qualität des von uns entwickelten Systems einen indirekten Einfluss auf die Verkehrsteilnehmer*innen. Da es sich bei IKaS um ein kritisches System handelt, spielt auch die Qualitätssicherung unseres Projekts eine große Rolle, um die Sicherheit der Verkehrsteilnehmer*innen zu gewährleisten. Da das IKaS-Projekt ein großes Forschungsprojekt ist, ist es von Bedeutung sicherzustellen, dass das von uns entwickelte System kompatibel zu den anderen Komponenten des IKaS-Projekts ist. Aus demselben Grund muss unser System wartbar sein, sodass die zukünftigen Entwickler*innen das System einfach weiterverwenden und erweitern können. Ebenfalls ist Effizienz bedeutsam, damit die beschränkten Ressourcen des Projekts zielgerichtet verwendet werden können.

2.1 Qualitätsziel 1: Wartbarkeit

Wartbarkeit ist ein langfristiger Aspekt, der beschreibt, wie einfach und effizient die Software erweitert und weiterverwendet werden kann. Zusätzlich steht Wartbarkeit dafür, wie effizient die Leistung der Software verbessert werden kann und wie problemlos die Softwareteile an eine veränderte Umgebung beziehungsweise an veränderte Anforderungen angepasst werden können.

Bezug zum Projekt: Unser Arbeitgeber legt großen Wert darauf, dass das von uns entwickelte System effizient anpassbar und von zukünftigen Entwickler*innen einfach zu erweitern und weiterzuverwenden ist. Dies ist notwendig, da sich Ansprüche und die interne Herangehensweise an das Forschungsprojekt IKaS verändern können. Aus diesen Gründen wurde das Unterziel Erweiterbarkeit gewählt. Da es sich bei IKaS um ein Forschungsprojekt handelt, muss der Code besonders verständlich sein, um eventuelle Forschungsergebnisse nachvollziehen zu können. Außerdem müssen sich regelmäßig neue Mitarbeitende in das IKaS-Projekt einarbeiten. Als Teil von IKaS ist Lesbarkeit somit auch bei unserem Projekt ein notwendiges Unterziel.

2.1.1 Erweiterbarkeit

Maßnahme: Als erste Maßnahme zur Sicherung von Erweiterbarkeit werden am Anfang jeder Iteration Design Meetings durchgeführt. Während des Design Meetings wird diskutiert und überlegt, wie genau die User Stories implementiert werden sollen. Dafür werden Leitfragen genutzt, die dabei helfen, die Implementationsentscheidungen mit dem Fokus auf Erweiterbarkeit zu treffen und festzuhalten. Diese Maßnahme trägt dazu bei, dass während der Entwicklung des Systems die Erweiterbarkeit bei Implementationsentscheidung explizit miteinbezogen wird. Während der Design Meetings wird sowohl festgelegt, was erweitert werden kann als auch, wie die Erweiterbarkeit mit bestehenden Komponenten umgesetzt wird. Die ausgearbeiteten Protokolle

nach den Design Meetings werden den Entwickler*innen, die das Projekt übernehmen, auch einen Überblick geben, worauf besonderen Wert bei jeweiliger User Story gelegt wurde, um die Entscheidungsprozesse besser nachvollziehen zu können.

Als Hauptmaßnahme zur Sicherung von Erweiterbarkeit werden in jeder Iteration Code Reviews durchgeführt. Mit dieser Maßnahme wird regelmäßig überprüft, ob der neu geschriebene Code den Kriterien entspricht, die von dem restlichen System bereits erfüllt sind. Eine ausführliche Checkliste ist der Leitfaden für einen konsistenten und einheitlichen Aufbau von Komponenten. Die Code Reviews tragen zum Erreichen der Erweiterbarkeit bei, indem regelmäßig überprüft wird, ob die im Design Meeting festgelegten Vorgaben zur Erweiterbarkeit korrekt umgesetzt wurden.

Prozessbeschreibung: Nachdem die User Stories für die Iteration von unserem Arbeitgeber festgelegt wurden, findet am nächsten Tag ein Design Meeting statt. Das Design Meeting wird mit dem gesamten Entwicklerteam durchgeführt. In einem Design Meeting wird die grobe Implementation der neuen User Stories geklärt und besprochen, wie das Qualitätsziel Erweiterbarkeit umgesetzt werden kann. Dies umfasst Fragen zur Funktionalität, die zu verwendeten Schnittstellen/Funktionen und wie die Erweiterbarkeit genau gewährleistet werden kann. Um sicherzustellen, dass alle Entwickler*innen die gleiche und klare Vorstellung davon haben, worauf sie bei der Implementierung achten müssen, werden die Antworten auf diese Leitfragen in einem Design Meeting Protokoll aufgeschrieben. Zusätzlich wird das Datum, an dem der Implementationszeitraum der Iteration endet und die Code Reviews der User Stories stattfinden, festgelegt. Danach kann mit der Implementierung der User Story begonnen werden.

An dem beim Design Meeting festgelegten Datum wird ein Code Review durchgeführt, um den Code generell zu überprüfen und sicherzustellen, dass die erforderlichen Maßnahmen für Erweiterbarkeit durchgeführt wurden. In dem Code Review wird die Implementation der User Stories von dem gesamten Entwicklerteam mithilfe einer Checkliste überprüft. Die Hauptverantwortung für das Code Review einer User Story übernimmt dabei die gleiche Person, die im Unterziel Lesbarkeit als Code Buddy der User Story eingeteilt wurde. Wenn im Code Review Fehler gefunden werden, müssen diese behoben und erneut überprüft werden, ansonsten darf die User Story nicht akzeptiert werden. Wenn ein Code Review erfolgreich abgeschlossen wurde, gilt die Maßnahme für die entsprechende User Story als bestanden und wird in den automatisierten Tests zur Kompatibilität miteinbezogen.

2.1.2 Lesbarkeit

Maßnahmen: Um die Lesbarkeit zu gewährleisten, setzen wir statische Code-Analyse, automatische Code-Formatierung und das Code Buddy System ein. Das Code Buddy System ist hier unsere Hauptmaßnahme.

Die statische Code-Analyse führen wir durch den Linter clang-tidy aus. Das statische Code-Analyse Werkzeug hilft uns dabei, aus der Semantik hervorgehende stilistische Probleme zu finden und zu korrigieren. Indem der Linter den Code auf festgelegte stilistische Entscheidungen prüft, kann er helfen, gut strukturierten und konsistenten Code zu implementieren, was zu besserer Lesbarkeit führt.

Der von uns genutzte Codeformatierer, zur automatischen Formatierung, ist clang-format. Dieser formatiert den Code bei jedem Speichern und jedem Commit des Codes nach von uns festgelegten Regeln selbstständig. Da dadurch ein einheitlicher, sauberer Stil erzwungen wird, ist der Code lesbarer.

Das Code Buddy System ist eine Abwandlung der Paarprogrammierung, wobei der Code Buddy beim Code Buddy System nicht selbst mit implementiert. Jeder User Story wird neben Entwickler*innen auch ein Code Buddy zugewiesen, der sich ebenfalls in das Thema einarbeitet und als erster*erste Ansprechpartner*in zur Verfügung steht. Diese*r bekommt jedoch im Gegensatz zur normalen Paarprogrammierung auch eigene User Stories zur Implementation zugeteilt sowie einen dazugehörigen Code Buddy.

Der Code Buddy führt bei jedem Push ein kurzes Review durch, zu dem er sich Notizen macht. Diese werden in regelmäßigen Statusmeetings erörtert, in denen ebenfalls der Stand der Implementation besprochen und mögliche Fragen geklärt werden. Auch außerhalb der Meetings steht der Buddy für kurzfristig aufkommende Fragen bereit. Wobei versucht wird, diese während der Meetings zu klären, wenn es die Dringlichkeit zulässt. In diesem Fall ist der Code Buddy speziell für die Überprüfung der Stilrichtlinien im kurzen Review zuständig. Auch ist er dafür verantwortlich zu überprüfen, dass der Linter in der IDE des*der Entwicklers*in integriert und aktiviert ist sowie dass der Codeformatierer erfolgreich auf den Code angewendet wurde. Somit stellt er übersichtlichen, konsistenten Code und die Einhaltung der anderen Maßnahmen sicher, was gut lesbaren Code garantiert.

Prozessbeschreibung: Der Linter clang-tidy wurde vor Anfang der Implementation bei allen Entwicklern*innen in CLion, die IDE unserer Wahl, eingebunden.

Es wird in jeder Iteration vom Code Buddy überprüft, dass clang-tidy aktiviert ist und genutzt wird.

In der ersten Iteration wurde eine clang-format Datei erstellt, die an unsere Stilrichtlinien angepasst wurde. Diese wurde in das Repository integriert und formatiert den Code selbstständig bei Speicherung und Push des Codes.

Es wird in jeder Iteration vom Code Buddy überprüft, dass der Codeformatierer korrekt auf den Code angewendet wurde.

Am Anfang jeder Iteration wird jeder User Story neben den Entwickler*innen auch ein Code Buddy zugewiesen. Hierbei wird darauf geachtet, dass wenn eine feste Kombination von einer*einem oder mehreren Entwickler*innen an mehreren User Stories arbeitet, erhält diese denselben Code Buddy, um die Statusmeetings zu vereinen und so deren Anzahl verringern zu können.

Der Code Buddy arbeitet sich nun ebenfalls in das Thema der User Story ein, um als erster*erste Ansprechpartner*in bei Fragen und Problemen zur Verfügung zu stehen.

Wenn nun Code gepusht wird, überprüft der Code Buddy diesen auf Lesbarkeit anhand einer in der ersten Iteration festgelegten Checkliste. Hierbei wird auch überprüft, ob der Codeformatierer auf den Code angewandt wurde. Falls das nicht der Fall sein sollte, muss dies im Statusmeeting umgehend nachgeholt werden. Die Checkliste haben wir in Markdown geschrieben und umfasst einfach überprüfbare und präzise gehaltene Ja/Nein-Fragen, die einfach abgehakt werden können, sollte die Bedingung erfüllt sein.

Im Laufe der Iteration werden dann wöchentliche Statusmeetings abgehalten, was in unserem Fall zwei Statusmeetings pro Iteration sind, da wir uns auf ein Iterationsintervall von zwei Wochen geeinigt hatten.

Eine Checkliste wird bis zum nächsten Meeting beibehalten und im Falle neuer Pushes aktualisiert. Dies beinhaltet das Abhaken von neu erfüllten Punkten und das Überprüfen, ob sich der Status von bereits erfüllten Punkten verändert hat. Für die Lesbarkeit muss ausschließlich der Code überprüft werden, der sich seit der letzten Überprüfung verändert hat.

In den Statusmeetings werden die erstellten Checklisten anhand des CEDAR-Feedback Modells besprochen und die Fehler werden je nach Größe direkt im Meeting oder nach den Meetings korrigiert. Diese Korrektur wird wieder von den Code Buddies überprüft.

Die Integration des Linters in die IDE wird bei jedem Statusmeeting überprüft und daraufhin ebenfalls in der Checkliste abgehakt. Falls der Linter nicht integriert sein sollte, muss dies noch während des Meetings geändert

und mögliche Fehlermeldungen des Linters umgehend korrigiert werden. Auch mögliche aufgekommene Fragen, die nicht wegen akuter Dringlichkeit vorher geklärt wurden, und Implementationsprobleme werden in den Statusmeetings besprochen.

Die Statusmeetings werden anhand einer Protokollvorlage stichpunktartig protokolliert.

Wenn der Code Buddy am Ende des zweiten Statusmeetings keine Einwände mehr hat oder alle Fehler bestätigt korrigiert wurden, gilt der Code der User Story als lesbar und kann zum Code Review, welches zum Unterziel Erweiterbarkeit gehört, weitergereicht werden.

2.2 Qualitätsziel 2: Kompatibilität

Die Kompatibilität bezieht sich darauf, inwiefern die interne Kommunikation zwischen den Softwarekomponenten konfliktfrei vonstatten geht und ob sie ohne einander zu beeinträchtigen, die Ressourcen verwenden können.

Bezug zum Projekt: Für das IKaS-Projekt spielt die Interoperabilität eine besonders große Rolle. Das von uns entwickelte Projekt muss zu den bestehenden Systemen von IKaS kompatibel sein, damit eine konfliktfreie Zusammenarbeit gewährleistet ist. Dies bezieht sich besonders darauf, dass die Sensordaten korrekt eingelesen werden und die Komponenten unseres Systems fehlerfrei untereinander kommunizieren können. Außerdem sollen in Zukunft weitere Systeme auf unserem Projekt aufbauen, weshalb gängige Ausgabeformate für problemlose Kommunikation verwendet werden müssen.

Maßnahme: Um die Kompatibilität mit den bestehenden Systemen von IKaS sowie zwischen den einzelnen Komponenten unseres Projekts zu gewährleisten, werden automatisierte Integrationstests verwendet. Dabei handelt es sich um spezielle Programme, die die Integration verschiedener Softwarekomponenten automatisch testen und Rückmeldung geben, ob dies erfolgreich war. Diese Maßnahme ist geeignet, um Kompatibilität nachzuweisen, da sie eine klassische Nutzung der verschiedenen Komponenten auf echten Datensätzen automatisiert simulieren und dabei Fehler im Ablauf selbstständig erkennen kann. So kann das fehlerfreie Einlesen der Daten und eine funktionierende Kommunikation zwischen den Komponenten sichergestellt werden.

Prozessbeschreibung: Die automatisierten Tests müssen bei der Implementation einer User Story bereits entsprechend erweitert oder angepasst werden. Ausgeführt werden sie einmal pro Iteration nach allen Code Reviews. Mit einbezogen werden nur die User Stories bei denen das Code Review erfolgreich abgeschlossen wurde. Um unbearbeitete Sensordatensätze verwenden zu können, werden die Tests lokal, mittels eines Python-Skripts, von Alicia Gleichmann ausgeführt. Das Skript führt den Scan Combiner einmal für alle unterstützten Datenformate aus, wobei originale Sensordaten als Eingabe dienen. Die Ausgabedaten werden dann als Eingabe von Programmen genutzt, zu denen sie kompatibel sein sollen(?). Bei dem UOS-Format ist das der Peopleremover von 3DTK. Auf den Daten im KITTI-Format wird der Remover und nach Fertigstellung auch das kitti2pcl-Programm ausgeführt. Die Metadaten werden mit der JSON-Library von Python überprüft. Ein Test gilt als erfolgreich, wenn das entsprechende Programm fehlerfrei zu Ende läuft. Bei der erfolgreichen Ausführung aller Tests gilt die Maßnahme als bestanden und es werden die manuellen Tests zur Sicherung der Effizienz durchgeführt. Wenn es zu Fehlern kommt, arbeitet das Entwicklungsteam gemeinsam an einer Lösung. Danach werden die Tests erneut von Alicia Gleichmann durchgeführt. Sollten die Fehler bis zum Ende der Iteration nicht gefunden und behoben worden sein, muss unser Arbeitgeber über die fehlerhaften Komponenten informiert und ein Plan zum weiteren Vorgehen erstellt werden.

2.3 Qualitätsziel 3: Effizienz

Die Effizienz eines Systems bezieht sich auf die ökonomische Nutzung der zur Verfügung stehenden Ressourcen. Im Kontext von Softwaresystemen kann ein effizienteres System die gleichen Datentransformationen mit zum Beispiel weniger Speicherauslastung, Laufzeit beziehungsweise Stromverbrauch umsetzen.

Bezug zum Projekt: Die im IKaS-Projekt eingesetzte InnoTram kann täglich bis zu 5TB an Rohdaten produzieren. Um solche großen Datenmengen kontinuierlich analysieren zu können, müssen diese effizient verarbeitet werden. Diese Analyse soll auf einem Server der FZD durchgeführt werden. Allerdings soll es auch möglich sein, zu Testzwecken diese Analyse auf einem Desktop-System mit 16GiB an Arbeitsspeicher in angemessener Zeit durchzuführen. Deshalb haben wir uns mit unserem Arbeitgeber auf das QS-Ziel Effizienz geeinigt.

Die im IKaS-Projekt eingesetzte InnoTram kann täglich bis zu fünf Terabyte an Rohdaten produzieren. Unser Arbeitgeber möchte, dass diese Daten kontinuierlich verarbeitet werden können. Um dies ohne Zeitverzug zu erreichen, müssen die Sensordaten effizient/Laufzeit-effizient verarbeitet werden. Diese Verarbeitung soll regelmäßig auf einem Server der FZD durchgeführt werden. Da der Server auch für andere Projekte genutzt wird, ist unserem Arbeitgeber wichtig, dass das System durch die Nutzung von effizienten Algorithmen und Datenstrukturen unnötige Operationen vermeidet und damit die Auslastung möglichst gering hält. Allerdings soll es auch für Testzwecke möglich sein, diese Analyse auf einem Desktop-System mit 16GiB an Arbeitsspeicher durchzuführen, weshalb Speichereffizienz erforderlich ist. Um die Einhaltung dieser Effizienzziele im Entwicklungsprozess zu gewährleisten, haben wir uns mit unserem Arbeitgeber auf das QS-Ziel Effizienz geeinigt.

Maßnahme: Um sicherzustellen, dass das System in der Lage ist, die täglich entstehenden Messdaten abzuarbeiten, ohne in Verzug zu geraten, werden manuelle Tests zur Einhaltung von Laufzeit und Arbeitsspeicherauslastung-Limits, sowie Performanzanalysen, zum Vorbeugen von Problemen in zukünftigen Iterationen, ausgeführt. Bei den manuellen Tests zur Einhaltung der Limits handelt es sich um Messungen der Laufzeit und Speicherauslastung beim Verarbeiten von echten Rohdaten auf der Zielhardware, gefolgt von einem Vergleich mit den Maximalwerten. Durch das Testen in dieser realitätsnahen Umgebung kann mit hoher Konfidenz sichergestellt werden, dass das System im tatsächlichen Anwendungsfall alle Limits einhalten wird.

Nach dem Prüfen der Einhaltung dieser Limits werden manuelle analytische Tests zur Effizienz durchgeführt, die potenzielle Engpässe und Designprobleme identifizieren sollen. Diese umfassen Messungen davon, welchen Zeitanteil der gesamten Laufzeit Komponenten eingenommen haben und eine Einschätzung, ob die Teillaufzeiten für die Aufgabe der Komponente angemessen sind.

Die Analyse kann im Falle des Nicht-Bestehens der Limit-Tests dabei helfen, die Quelle des Problems zu finden und die Ausarbeitung einer Lösung zu unterstützen. Doch auch wenn die Tests bestanden sind, unterstützt die Analyse den weiteren Entwicklungsprozess durch Vorschläge zu einer performanteren Implementation. Dies hilft vor allem dabei, Engpässe in Komponenten früh zu erkennen und zu beheben, bevor weitere Abhängigkeiten zu dieser geschaffen werden, die eine Änderung zeitlich teuer machen. Gerade im Kontext der begrenzten Arbeitszeit und des festen Abgabetermins des Bachelor-Praktikums würden solche großen Änderungen nicht ins Zeitbudget passen.

Prozessbeschreibung: Ursprünglich war geplant, diese Tests auf der Zielhardware der FZD durchzuführen, um möglichst genaue Messergebnisse zu erhalten. Da unser Arbeitgeber jedoch nicht in der Lage war, uns Zugriff auf Server Hardware des FZDs zu geben, mussten die Tests (nach Absprache mit unserem Arbeitgeber)

auf einem privaten Server System eines Gruppenmitglieds umgesetzt werden. Das genannte System ist bereits über 10 Jahre alt und stellt damit die unterste Schwelle für zu erwartende Hardware dar. Konkret bedeutet dies, dass wenn die Limits für Laufzeit und Arbeitsspeicherauslastung auf dieser Hardware eingehalten werden, diese auch auf der Zielhardware des FZDs eingehalten werden können.

Trotz des Alters sind alle für die Effizienzanalyse relevanten Komponenten wie eine mehrkernige, SIMD-fähige CPU, ausreichend Arbeitsspeicher und eine SSD-Festplatte verbaut. Dies garantiert zu einem hohen Grad, dass Performanzverbesserungen aufgrund von Testergebnissen auf dieser Hardware auch auf die Zielhardware übersetzbare sind.

Alle Effizienztests und Messungen für Analysen finden auf dem folgenden System statt:

Komponenten	Spezifikation
System:	PowerEdge T620
CPU(s):	24 Cores, 48 Threads 1,8Ghz - 3,5Ghz (2x Intel(R) Xeon(R) CPU E5-2697 v2)
Cache:	384KiB L1, 3MiB L2, 30MiB L3
Memory:	256GiB (16x 16GiB DDR3 ECC 1600 MT/s)
Drive:	223GiB SATA SSD
OS:	Ubuntu 20.04.6 LTS
Kernel:	Linux 5.15.0-101-generic
Architecture:	x86-64

Um die Desktop-Messungen auch auf diesem System durchführen zu können, muss die Anzahl der Threads im Programm auf 12 gesetzt werden und der „time-per-block“ Parameter so eingestellt werden, dass das System nicht das Arbeitsspeicherlimit überschreitet.

Die Limit-Tests sowie die Effizienzanalyse werden am Ende jeder Iteration nach den bestandenen Kompatibilitätstests von Leonhard Steinecke durchgeführt. In einem Testdurchlauf müssen die Desktop- und Serversysteme folgende Rosbags verarbeiten:

Dateiname	Aufnahmedauer
Recorder_2022-08-26-08-59-35_c_1.bag	25,2489min
Recorder_2022-08-26-10-25-02_c_1.bag	25,1569min
Recorder_2022-08-26-12-28-09_c_1.bag	22,8333min
Recorder_2022-08-28-12-28-55_c_1.bag	40,63min
Recorder_2022-08-28-13-49-15_c_1.bag	19,8891min
insgesamt:	133,758min

Daraus ergeben sich die folgenden (Laufzeit-)Limitierungen für die Durchläufe:

System	Thread Anzahl Limit	Arbeitsspeicher Limit (in GiB)	maximale Laufzeit (in min)
Server	48	50 (14GiB für OS)	133,758
Desktop	12	10 (6GiB für OS)	267,516

Die für die Tests und Analyse benötigten Messdaten werden vom Tester in zwei separaten Programmdurchläufen gesammelt.

Der erste Durchlauf wird mit „GNU Time“ /usr/bin/time -v \$COMMAND auf Laufzeit und maximale

Speicherauslastung geprüft.

Im zweiten Durchlauf wird `perf record -a -g $COMMAND` verwendet, um genaue Informationen über die Verteilung der Laufzeiten auf die Komponenten des Systems zu sammeln.

Um zu überprüfen, ob das Programm die Limits zur Laufzeit und Speicherauslastung einhält, wird die „GNU Time“ Messung aus dem ersten Durchlauf verwendet. Falls die Laufzeit oder Speicherauslastung 90% des Limits überschreiten, muss die Messung mit „GNU Time“ auf zwei weiteren Durchläufen die Limits unterschreiten, um den Test zu bestehen.

Für die folgende Effizienzanalyse soll das Programm Hotspot verwendet werden, um die Cycle Messungen aus dem zweiten Durchlauf zu überprüfen. Dafür müssen zunächst die 25 Funktionen mit den meisten Cycles den Komponenten des Programms zugeordnet werden. Im nächsten Schritt muss der*die Testende bewerten, ob der Laufzeitanteil dieser Komponenten ihrer Aufgabe angemessen ist oder gewisse Komponenten oder Teile derer Engpässe darstellen. Falls Engpässe identifiziert wurden, müssen die damit verbundenen Komponenten genannt und der Lösung des Problems eine Dringlichkeit zugewiesen werden.

Um diese Analyse möglichst zielgerichtet zu halten, soll sich hierbei besonders auf neu hinzugefügte oder in der Iteration angepasste Komponenten konzentrieren werden.

Falls die festgelegten Limits bei einem Test nicht eingehalten wurden, muss zunächst abgeschätzt werden, ob das Problem bis zum Ende der Iteration behoben werden kann oder die Behebung dieser eine weitere Iteration in Anspruch nimmt.

Diese Entscheidung darf erst nach der Effizienzanalyse getroffen werden, da diese ausführliche Informationen zur Quelle und möglichen Lösungsansätzen bietet.

Falls das Problem voraussichtlich nicht mehr in der Iteration behoben werden kann, muss dies im Meeting am Anfang der nächsten Iteration mit unserem Arbeitgeber abgesprochen werden und ggf. die User Story um eine Iteration verlängert werden.

Weitere Vorschläge zur Anpassung der Performanz, die nicht wegen eines Überschreitens von Limits umgehend umgesetzt werden müssen, sollen in Entscheidungen beim nächsten Design Meeting einfließen.

Beim Protokollieren der Maßnahme soll aus Lesbarkeitsgründen folgendes Format eingehalten werden:

```
> **Datum:** D.M.Y  
> **Durchgeführt von:** ?  
  
## Komponenten im Fokus  
<!---- Komponenten, die sich während der Iteration verändert haben und besonders überprüft werden müssen -->  
  
### Befehle:  
'`bash  
# server settings  
<!---- Server Befehl -->  
  
# desktop settings  
<!---- Desktop Befehl -->  
`'  
  
### Messwerte  
  
Relevante GNU Time Ergebnisse mit Server Einstellungen:  
`'go  
Elapsed (wall clock) time (m:ss):  ?:?:?  
Maximum resident set size (kbytes): ?  
Percent of CPU this job got:    ?% / 4800%  
`'  
  
Relevante GNU Time Ergebnisse mit Desktop Einstellungen:  
`'go  
Elapsed (wall clock) time (m:ss):  ?:?:?  
Maximum resident set size (kbytes): ?  
Percent of CPU this job got:    ?% / 1200%  
`'  
  
Server Cycle-Distribution:  
<!---- Screenshot aus dem Program 'Hotspot' der mit 'perf record' aufgenommenen Cycle-Distribution des Server Durchlaufs -->
```

Desktop Cycle-Distribution:
<!-- Screenshot aus dem Program 'Hotspot' der mit 'perf record' aufgenommenen Cycle-Distribution des Desktop Durchlaufs -->

Checkliste

- Server Limits

- [] Werden die Daten innerhalb der maximalen Laufzeit verarbeitet?
 - [] Bleibt die Speicherauslastung unter dem festgelegten Limit?
- Desktop Limits
- [] Werden die Daten innerhalb der maximalen Laufzeit verarbeitet?
 - [] Bleibt die Speicherauslastung unter dem festgelegten Limit?
- Cycle-Analyse
- [] Werden in den Hotspot-Codeabschnitten essenzielle Operationen ausgeführt, oder stellen sie vermeidbare Engpässe dar?
 - [] Welche Teile des System sollten angepasst werden, um die Laufzeit oder Speicherauslastung zu reduzieren?
 - [] Welche Dringlichkeit wird diesen Anpassungen zugewiesen?

Konsequenzen

<!-- Welche Konsequenzen wurden entschieden und wurden diese umgesetzt? -->

3 Abweichende QS-Ziele

Alle genannten QS-Ziele konnten wie geplant durchgeführt und durch Maßnahmen gesichert werden.

4 Projektverlauf und projektgefährdende Ereignisse

Projektverlauf: Das erste Planungstreffen mit dem*der Auftraggeber*in fand am 07.11.2023 statt. Die Ergebnisse des Projekts wurden am 12.03.2024 im finalen Treffen übergeben.

- Iteration 1: 07.11.2023 – 15.11.2023
- Iteration 2: 15.11.2023 – 29.11.2023. *Velocity:* 23
- Iteration 3: 29.11.2023 – 19.12.2023. *Velocity:* 26
- Iteration 4: 19.12.2023 – 11.01.2024. *Velocity:* 20
- Iteration 5: 11.01.2024 – 22.01.2024. *Velocity:* 11
- Iteration 6: 22.01.2024 – 06.02.2024. *Velocity:* 20
- Iteration 7: 06.02.2024 – 22.02.2024. *Velocity:* 22
- Iteration 8: 22.02.2024 – 12.03.2024. *Velocity:* 23

Abweichende Zeiten: Leonhard Steinecke ist 2 Wochen krankgeschrieben worden, weswegen er eine geringere Stundenzahl hat.

Projektgefährdende Ereignisse: In Iteration 3 auf Iteration 4 ist Leonhard Steinecke, vom 8.12.2023 bis zum 22.12.2023, 2 Wochen krank war. Atteste wurden per E-Mail an das Organisationsteam geschickt. Die aufgenommenen Rosbags der Bahn werden in einem Datensilo gespeichert, zu dem wir bis zum Ende des Projekts durch die Neueinarbeitung und Urlaub des Netzwerkadmins keinen Zugriff hatten. Jedoch hat uns Alexander Kroth repräsentative Rosbags auf einer externen Festplatte zur Verfügung gestellt, welche wir dann zum Testen verwendet haben.

Aus dem gleichen Grund konnte uns bis zum Ende kein Zugriff auf den Server gegeben werden, der die Zielhardware darstellen sollte.

5 Softwarelizenz

Der für dieses Projekt erstellte Code steht im Einvernehmen aller Beteiligten unter folgender Lizenz:

EUPL 1.2 - <https://joinup.ec.europa.eu/collection/eupl/eupl-text-eupl-12>