Aufbau eines virtuellen Umweltmodelles für die Entwicklungen von Algorithmen für die sichere Personenerkennung in Automated Guided Vehicle Appikatonen

Projektarbeit

für die Prüfung zum

Bachelor of Science

des Studienganges Informationstechnik

an der

Dualen Hochschule Baden-Württemberg Karlsruhe

von

Lorenz Scherrer

Abgabedatum 24. September 2023

Bearbeitungszeitraum	12 Wochen
Matrikelnummer	8809469
Kurs	tinf21b3
Ausbildungsfirma	SICK AG Waldkirch
Betreuer der Ausbildungsfirma	Manfred Haberer
Gutachter der Studienakademie	Prof. Dr. Jürgen Vollmer

Erklärung

Ich versichere hiermit, dass ich meine Projektarbeitmit dem Thema: »Aufbau eines virtuellen Umweltmodelles für die Entwicklungen von Algorithmen für die sichere Personenerkennung in Automated Guided Vehicle Appikatonen« selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt. _____

_____

Ort Datum Unterschrift

Sofern vom Dualen Partner ein Sperrvermerk gewünscht wird, ist folgende Formulierung zu verwenden:

Sperrvermerk

Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungsprozesses und des Evaluationsverfahrens zugänglich gemacht werden, sofern keine anderslautende Genehmigung vom Dualen Partner vorliegt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Algorithmen

Formelverzeichnis

Abkürzungsverzeichnis

Kapitel 1

Einleitung

Das Ziel dieser Abreit ist es ein virtuelle Umgebung zu erstellen um für die Entwicklung von Algorithmen für die sichere Personenerkennung in Automated Guided Vehicle. Es sollen Umweltdaten von Automated Guided Vehicle Applikationen zur Entwicklung generiert werden. Die Umweltdaten sollen zum testen und Validieren von Sicherheitskonzepten. So sollen Worst-Case-Szenarien definiert werden und die Robustheit der Softwarealgorithemen geprüft werden.

Der Vorteil ist es das so viele mehr testdaten generiert werden könne ohne aufwendige Test Scenerien in einer realen Werkshalle aufzubauen.

1.1 Problemstellung

Die Problemstellung bindet sich im DSM ein. Es sollen die Sceneie aufgebaut werden und die Sensordaten dem gRPC Server bereitgestellten werden.

Es gib keine Simulation in der man Gefahrensitzuationen testen kann. Ohne Sie in echten Hallen zu testen.

Im weiteren verlauf kann der Algorithmus der die Sicherheitsfelder der AGVs berechnet an ein Instutut geschickt werden der diesen Valiediert. Oder ein in der Simulation getesten Algorithmus in echten Szenarien getestet werden. Hier können die Sicherheitsalghrutmen erstmals getestet werden.

1.2 Zielsetzung

Daten übermittel und ein AGv in Unity erstellen.

1.3 Fragestellung

Wie kann ein AGV in Unity in erstellt werden und dessen Meschanische Daten über gRPC mit der Ausenwelt geteilt werden und auch Daten erhalten?

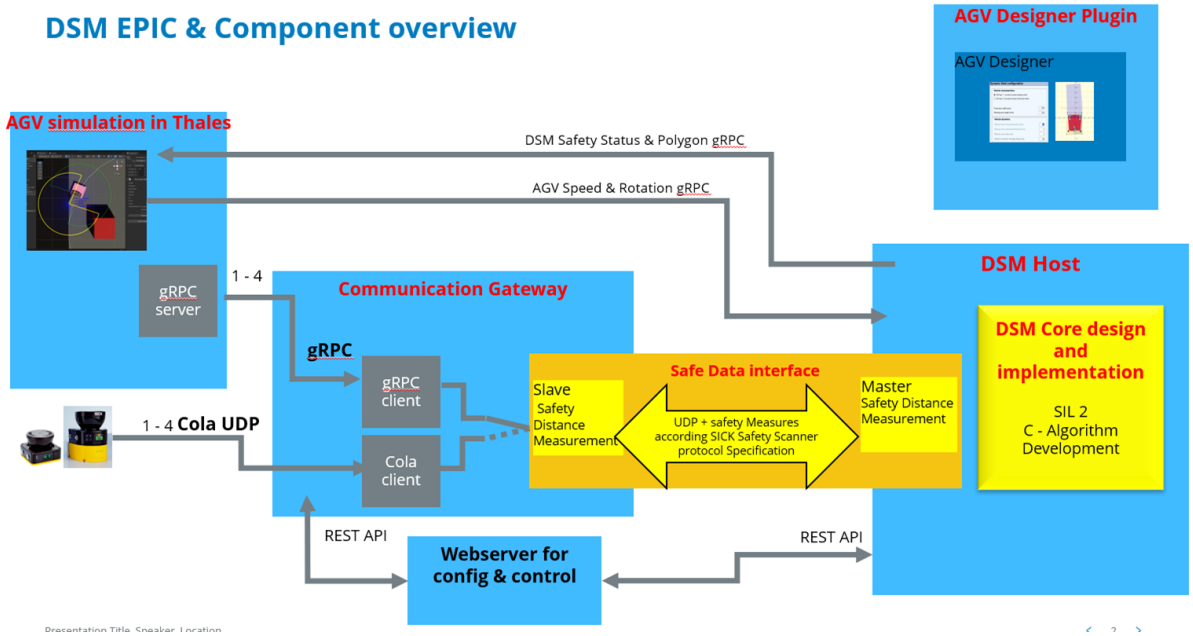

Abbildung 1.1: DSM EPIC & Component overview

Kapitel 2

Planung und Kontrolle

2.1 Zeitmanagament

2.1.1 Meilensteinen und Etappen

Innerhalb ersten Meilensteins liegt der Fokus auf der Umsetzung einer funktionsfähigen Schnittstelle zwischen dem gRPC Server und Client, um Daten in beiden Richtungen zu übertragen. Ein zentrales Ziel ist es, Thales Sensordaten von insgesamt 4 Sensoren über den gRPC Server vom Client abrufen zu können. Die Datenabfrage sollte in der Lage sein, mit einer Aufnahmefrequenz von mindestens alle 40 Millisekunden durchgeführt zu werden.
Zusätzlich zu den Sensordaten sollen auch andere Parameter über den gRPC Server übertragen werden, wie beispielsweise der Lenkwinkel oder die Geschwindigkeit eines zukünftig erstellten AGVs. Dieser Meilenstein bildet einen wesentlichen Schritt, um die grundlegende Funktionalität des Projektes zu etablieren und legt den Grundstein für die weitere Entwicklung im Projektverlauf.
Der Meilenstein umfasst die Erstellung der Fahrphysik für die AGVs. Dies beinhaltet die Realisierung von realistischen Bewegungsabläufen und Steuerungsmechanismen, die den Charakter der jeweiligen AGV-Typen authentisch wiedergeben.
Innerhalb dieses Meilensteins ist die Entwicklung mehrerer Arten von AGVs geplant, wobei mindestens eine Variante umgesetzt wird. In der Anfangsphase werden verschiedene AGVs entworfen, darunter ein AGV, das auf einem herkömmlichen Auto basiert, gefolgt von einem dreirädrigen Gabelstapler und schließlich einer Panzerrolle.
Die Steuerung und Fahranweisung der AGVs wird über ein Script in der Unity-Umgebung gesteuert. Dieses Script gibt die Geschwindigkeit, Beschleunigung und den Lenkeinschlag der Steuerachse vor.

Die erfolgreiche Umsetzung dieses Meilensteins wird die Basis für die funktionale und realistische Integration der AGVs in das Forschungsprojekt legen. Durch die Entwicklungsarbeit im zweiten Meilenstein wird die Grundlage für weiterführende Tests und Analysen der AGVs geschaffen, die zur Erreichung der gesamten Projekts beitragen.

Innerhalb dieses Meilensteins ist das Hauptziel die Erstellung eines Polygonzugs, der den Bremsweg des AGVs abbildet. Dieser Polygonzug soll den Verlauf des Bremsvorgangs in einer geometrischen Darstellung wiedergeben.

Nach der Erstellung des Polygonzugs muss überprüft werden, ob dieser direkt auf das AGV-Modell übertragen werden kann. Es ist von Bedeutung zu ermitteln, ob die Daten des Polygonzugs in das Simulationssystem eingeführt werden können, um eine korrekte Darstellung des Bremsverhaltens zu ermöglichen.
Falls eine direkte Übertragung des Polygonzugs nicht möglich ist, wird eine alternative Methode in Betracht gezogen. Hierbei wird die Länge des Bremsschlauchs in Verbindung mit dem Lenkeinschlag genutzt, um den Bremsweg auf eine plausible Weise darzustellen.
Die erfolgreiche Erfassung des AGV-Bremswegs als Polygonzug oder über alternative Darstellungsformen wird eine wichtige Grundlage für die Simulation des Bremsverhaltens in späteren Forschungsphasen bilden. Dieser Meilenstein trägt dazu bei, das Verhalten des AGVs unter verschiedenen Bedingungen besser zu verstehen und die Gesamtfunktionalität des Modells zu verbessern.

Im Rahmen dieses Meilensteins wird das Hauptziel verfolgt, die Signalübertragung vom gRPC-Client zum AGV zu implementieren. Das Signal soll die Abbremsung des AGVs auslösen, wenn es empfangen wird.
Es ist von entscheidender Bedeutung, die Steuerungsfunktion im AGV zu konfigurieren, damit sie auf das empfangene Signal reagiert und die notwendigen Maßnahmen zur Abbremsung einleitet.
Die erfolgreiche Umsetzung dieses Meilensteins wird die Fähigkeit des AGVs zur externen Steuerung und zur Initiierung einer Abbremsung über das gRPC-Client-Signal demonstrieren. Diese Funktionalität kann im weiteren Verlauf der Forschungsarbeit dazu verwendet werden, verschiedene Szenarien und Anwendungen zu testen, bei denen eine externe Steuerung des AGVs erforderlich ist.

2.1.2 Zeitplan

Das Projekt startet am 3. Juli und die Abgabe der schriftlichen Projektarbeit ist am 21. September. Nach der Berücksichtigungen der Abwesenheiten umfasst das Projekt 35 Arbeitstage.

In den ersten fünf Tagen des Projekts steht die Vorbereitung und Planung im Vordergrund. Hierbei werden die genauen Anforderungen und Ziele des Projekts geklärt. Es erfolgt eine detaillierte Aufschlüsselung der Arbeitsschritte sowie die Definition der benötigten Ressourcen.
Vom sechsten bis zum fünfzehnten Tag wird der erste Meilenstein erreicht - die Implementierung der gRPC-Server-Client-Schnittstelle. Ziel ist es, die bidirektionale Datenübertragung zwischen dem Server und Client zu ermöglichen.
Vom sechzehnten bis zum einundzwantigsten Tag liegt der Fokus auf dem zweiten Meilenstein, der die Umsetzung der AGVs beinhaltet. Es wird die Fahrphysik für verschiedene AGV-Typen erstellt, beginnend mit einem AGV, das sich wie ein herkömmliches Auto verhält. Die Bewegungsabläufe und Steuerung werden getestet und optimiert.
Vom zweiundzwanzigsten bis zum sechsundzwanzigsten Tag wird die Umsetzung der AGVs fortgesetzt. Ein Gabelstapler-AGV wird entwickelt, um dreirädrige Bewegungen zu simulieren. Parallel dazu wird die Vorbereitung für die Umsetzung des dritten AGV-Typs, einer Panzerrolle, getroffen.
Vom siebenundzwanzigsten bis zum neunundzwanzigsten Tag wird der dritte Meilenstein erreicht. Ein Polygonzug wird erstellt, um den Bremsweg des AGVs darzustellen. Die Übertragbarkeit dieser Daten auf das Modell wird geprüft, alternativ wird die Berechnung des Bremswegs anhand des Bremsschlauchs in Betracht gezogen.

Die letzten beiden Tage, der dreißigste und der zweidreißigste Tag, widmen sich dem vierten Meilenstein. Hierbei steht die Implementierung der externen Abbremsung des AGVs durch ein Signal vom gRPC-Client im Fokus. Die Steuerungsfunktion im AGV wird entsprechend konfiguriert und getestet, um sicherzustellen, dass das AGV zuverlässig auf das Signal reagiert.

Aufgabe	**Anfang**	**Ende**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**
Planung+ Aufgabe Prio 1	12.07	14.07												
Aufgabe Prio 2	17.07	21.07												
Abwesenheit	24.07	01.09												
Aufgabe Prio 2	04.09	08.09												
Aufgabe Prio 3	11.09	15.09												
Aufgabe Prio 4+Doku	18.09	29.09												
Praxisbericht + Kollogium Vorbereitung	18.09	22.09												
Kollogium	25.09	29.09												

2.2 Risikoanalyse

Im Rahmen des Projekts wurden potenzielle Risiken identifiziert, die während der Umsetzung der Meilensteine und des 35-tägigen Zeitplans auftreten könnten. Diese Risiken wurden analysiert und bewertet, um angemessene Maßnahmen zur Risikobewältigung zu identifizieren.

Ein hoch bewertetes Risiko besteht in technischen Herausforderungen bei der Implementierung der gRPC-Server-Client-Schnittstelle (Meilenstein 1). Die Eintrittswahrscheinlichkeit wird als mittel eingestuft, aber die Auswirkungen könnten hoch sein, da dies die Grundlage für die Kommunikation im Projekt bildet. Ähnlich verhält es sich mit dem Risiko unvorhergesehener Schwierigkeiten bei der Signalübertragung für die externe Abbremsung (Meilenstein 4). Hierbei könnte die Eintrittswahrscheinlichkeit mittel sein, aber die Auswirkungen wären hoch, da die externe Steuerung des AGVs beeinträchtigt werden könnte.

Die Umsetzung der AGV-Typen (Meilenstein 2) birgt das Risiko von Schwierigkeiten bei der Erstellung der Fahrphysik, da die Eintrittswahrscheinlichkeit als hoch eingeschätzt wird und die Auswirkungen mittel sein könnten. Zudem besteht die Gefahr von zeitlichen Verzögerungen bei der Umsetzung der AGV-Typen, da eine mittlere Eintrittswahrscheinlichkeit und mittlere Auswirkungen gegeben sind.

In Bezug auf den dritten Meilenstein, die Erfassung des Bremsweg-Polygonzugs, besteht das Risiko technischer Inkompatibilität bei der Übertragung, obwohl die Eintrittswahrscheinlichkeit niedrig ist, könnten die Auswirkungen mittel sein. Zusätzlich könnte das Risiko bestehen, dass sich die Anforderungen an den Bremsweg ändern, was jedoch als gering eingeschätzt wird.

Die mögliche Fehlkommunikation zwischen dem gRPC-Server und dem Client (Meilenstein 1) sowie unerwartete Schwierigkeiten bei der Konfiguration der Steuerungsfunktion für die externe Abbremsung (Meilenstein 4) könnten sich als hohe Risiken mit mittleren Auswirkungen erweisen.

Ein weiteres bedeutendes Risiko betrifft die mangelnde Verfügbarkeit von Ressourcen wie Hardware oder Software während des gesamten Projekts. Hierbei wird eine hohe Eintrittswahrscheinlichkeit angenommen, verbunden mit hohen Auswirkungen auf den Projektfortschritt.

Die Risikoanalyse zeigt auf, dass eine gründliche Planung und regelmäßige Überwachung der Projektschritte unerlässlich ist, um auf mögliche Risiken rechtzeitig reagieren zu können. Es empfiehlt sich, für jedes identifizierte Risiko geeignete Maßnahmen zur Risikominderung und -bewältigung zu entwickeln.

Kapitel 3

Grundlagen

3.1 gRPC in Unity

RPC steht für "Remote Procedure Call"(zu Deutsch: "Fernprozeduraufruf"). Es handelt sich um ein Protokoll und ein Konzept in der Informatik, das es ermöglicht, Funktionen oder Methoden auf entfernten Computern oder Prozessen aufzurufen, als ob sie auf dem lokalen Computer ausgeführt würden. RPC ermöglicht die Kommunikation zwischen verteilten Systemen und wird häufig in Client-Server-Anwendungen, Netzwerkdiensten und verteilten Systemen eingesetzt.[Bengel 2004]

Die Grundidee hinter RPC ist, dass ein Client-Programm eine Funktion oder Methode auf einem entfernten Server aufruft, als ob sie lokal vorhanden wäre. Der RPC-Mechanismus kümmert sich um die Details der Kommunikation, der Datenübertragung und der Rückgabewerte. Der Aufruf einer entfernten Funktion ähnelt daher stark einem normalen Funktionsaufruf in der Programmierung.[Bengel 2004]

RPC erleichtert die Entwicklung von verteilten Anwendungen, da Entwickler sich nicht um die Details der Netzwerkkommunikation kümmern müssen. Es gibt verschiedene RPC-Implementierungen und Protokolle, darunter gRPC, XML-RPC, JSON-RPC und andere, die in verschiedenen Umgebungen und Anwendungsfällen eingesetzt werden können.[Bengel 2004]

gRPC ist ein leistungsstarkes Remote Procedure Call (RPC)-Framework, das von Google entwickelt wurde. Es wird häufig in verteilten Systemen und Microservices-Architekturen verwendet, um die Kommunikation zwischen verschiedenen Diensten zu erleichtern.

Eine der herausragenden Eigenschaften von gRPC ist die Verwendung des HTTP/2-Protokolls als Transportmechanismus. HTTP/2 bietet signifikante Verbesserungen gegenüber HTTP/1.x, darunter die Möglichkeit, mehrere Anfragen und Antworten über eine einzige TCP-Verbindung zu multiplexen, Header-Komprimierung und verbesserte Leistung.

Ein weiterer großer Vorteil von gRPC ist seine plattformübergreifende Unterstützung. Es ist in einer Vielzahl von Programmiersprachen verfügbar, darunter C++, Java, Python, Go, Ruby, C# und mehr. Dies ermöglicht die Entwicklung von Client- und Serveranwendungen in verschiedenen Sprachen, die miteinander kommunizieren können.

Um Schnittstellen und Datenstrukturen eindeutig und plattformübergreifend zu definieren, verwendet gRPC die Protobuf (Protocol Buffers)-Sprache als IDL (Interface Definition Language). Diese starke Typisierung ermöglicht es Client und Server, genau zu wissen, welche Daten erwartet werden, ohne auf textbasierte Formate wie JSON oder XML angewiesen zu sein.

Eine weitere Stärke von gRPC ist die Unterstützung von bidirektionaler Streaming-Kommunikation. Das bedeutet, dass sowohl der Client als auch der Server Daten gleichzeitig senden und empfangen können, was für Anwendungsfälle wie Chat-Anwendungen oder Echtzeit-Spiele sehr nützlich ist.

Die Authentifizierung und Sicherheit sind ebenfalls integrale Bestandteile von gRPC. Es bietet Funktionen zur Authentifizierung und Verschlüsselung, um die Kommunikation zwischen Client und Server sicher zu gestalten. Unterschiedliche Authentifizierungsmechanismen wie OAuth, JWT und TLS werden unterstützt.

Ein weiterer Vorteil von gRPC ist die automatische Codegenerierung. Auf Grundlage der in Protobuf definierten Schnittstellen und Nachrichten generiert gRPC automatisch Client- und Servercode, was die Entwicklung und Wartung von gRPC-Anwendungen erleichtert.

Insgesamt bietet gRPC eine effiziente und plattformübergreifende Möglichkeit, die Kommunikation zwischen verschiedenen Komponenten in verteilten Systemen zu verwalten. Es wird in einer Vielzahl von Anwendungen eingesetzt, von Mikrodiensten-Architekturen über IoT-Geräte bis hin zu Cloud-basierten Anwendungen.

In Unity ist gRPC eine Implementierung des gRPC-Frameworks, das speziell für die Verwendung in Unity-Projekten entwickelt wurde. Es ermöglicht die Integration von gRPC-Kommunikation in Unity-Anwendungen, um die Interaktion zwischen Unity-Anwendungen und entfernten Servern oder Diensten zu erleichtern.

Die Verwendung von gRPC in Unity erfordert in der Regel die Integration von gRPC-Paketen oder -Bibliotheken in Ihr Unity-Projekt sowie die Erstellung von gRPC-Clientcode, um Serveraufrufe durchzuführen. Dadurch können Unity-Entwickler die Vorteile von gRPC nutzen, um die Kommunikation mit Servern oder Diensten in ihren Spielen oder Anwendungen zu vereinfachen und zu optimieren.

3.2 Thales-Framework

Die Virtualisierung von Anwendungen mittels Modellierung und Simulation eröffnet eine Vielzahl von Vorteilen gegenüber der Verwendung physischer Anwendungen. Diese Vorteile erstrecken sich auf verkürzte Entwicklungszyklen und eine gesteigerte Zuverlässigkeit der Anwendungen. In diesem Kontext strebt das Thales-Framework an, eine effiziente Plattform bereitzustellen, welche es den Anwendern ermöglicht, ihre eigenen virtuellen Anwendungen auf effektive Weise zu entwickeln.

Ein zentrales Ziel von Thales besteht darin, Werkzeuge zur zügigen und effizienten Erstellung von Anwendungsmodellen zur Verfügung zu stellen. Dies wird durch die Umsetzung eines Gemeinschafts- und Paket-basierten Ansatzes ermöglicht, wobei Artifactory als Paketregister fungiert. Diese Herangehensweise erleichtert die rasche gemeinsame Nutzung von Modellressourcen innerhalb der gesamten Anwendergemeinschaft.

Thales nutzt zur Erreichung hoher Simulationseffizienz und -genauigkeit Sensormodelle, welche von Grafikprozessoren (GPUs) unterstützt werden.

Durch die Anwendung von Protobuf und gRPC wird die Integration in unterschiedliche Programmiersprachen und Plattformen ermöglicht. Auf diese Weise ist der Zugriff auf die Anwendung über diverse Programmiersprachen und Plattformen hinweg realisierbar.

Des Weiteren gestaltet sich die Erweiterung von Thales durch Unity-Pakete als unkompliziert.

Ein weiterer hervorstechender Aspekt in der Entwicklung von Thales ist die Minimierung des Eigenentwicklungsanteils durch die verstärkte Nutzung existierender Bibliotheken und Tools. Dieser Ansatz trägt zur Effizienzsteigerung bei der Umsetzung des Frameworks bei.[**.02.09.2023**]

3.3 Senoren in der VirtuellenUmgebung

Im Thales-Paket sind vordefinierte LIDAR-Scanner als Fertigteile enthalten, die mühelos in der entsprechenden Szene instanziiert werden können. Alternativ besteht die Möglichkeit, einen generischen LIDAR-Scanner zu erstellen und entsprechend zu konfigurieren, indem die Systemdiagrammkomponente direkt genutzt wird. Die Nutzung von vorgefertigten Elementen, wie in der Sektion SSpezifische LIDAR-Prefabs"dargestellt, bietet eine zügige und unkomplizierte Möglichkeit, die Arbeit mit virtuellen LIDAR-Sensoren aufzunehmen. Dies erweist sich insbesondere dann als hilfreich, wenn der gewünschte Sensor bereits in Thales verfügbar ist. Falls die Absicht besteht, LIDAR-Scanner zu verwenden, die nicht im bereitgestellten Paket enthalten sind, oder wenn generell LIDAR-Geräte in einem frühen Entwicklungsstadium erforscht werden sollen, bietet es sich an, die generische LIDAR-Scanner-Komponente zu nutzen. Für Fälle, in denen eine raffiniertere Sensormodellierung angestrebt wird, besteht sogar die Möglichkeit, eine individuelle Systemdiagrammkomponente zu erstellen. Referenzimplementierungen hierzu sind in den Systemgraph-Komponenten innerhalb von Thales verfügbar.

Rasterisierung basierende LIDAR-Scanner

Diese Implementierung nutzt Rasterisierungs-Shader als Backend-Technologie und erfordert keine RTX-Hardware. Obwohl es in gewisser Hinsicht seine Grenzen hat, ist es oft eine vernünftige Wahl, insbesondere wenn Sie keine hohe Simulationsgenauigkeit benötigen.

RTX basierende LIDAR-Scanner

Das RTX-basierte Lidar-Scanner-Modell nutzt Echtzeit-Raytracing und erfordert daher eine Grafikkarte, die RTX unterstützt. Sie erhalten diese Implementierung, indem Sie das GenericLidarScannerRTX-Systemdiagramm auswählen.

Kapitel 4

Implementierung

4.1 Implementierung der Schnittstelle

Die ThalesApi wird verwendet um die Sensordaten der Test Sceneie weiterzuleiten. Die ThalesApi gRPC verwendet das Protocol Buffers (protobuf) Datenformat, das binär kodiert ist und daher effizienter als JSON oder XML ist. Dies führt zu schnellerer Datenübertragung und geringerem Ressourcenverbrauch, was in Spielen und anderen Unity-Anwendungen wichtig ist.
Das gRPC Protokocol welches die ThalesApi verwendet bietet Unterstützung für eine Vielzahl von Programmiersprachen, darunter C#, was die Integration in Unity erleichtert. Dies ermöglicht es, Client-Server-Kommunikation nahtlos zwischen Unity und anderen Serveranwendungen in verschiedenen Sprachen herzustellen. Die Clients können in vielen Sprachen geschrieben werden. Im DSM Projekt wird ein C++ CLient verwendet.
Mit gRPC können Sie starke Typisierung für Ihre Dienstaufrufe verwenden. Dies bedeutet, dass Sie spezifische Methodenaufrufe mit stark typisierten Eingabe- und Ausgabeparametern definieren können, was die Entwicklung und Wartung Ihrer Anwendung erleichtert.
gRPC unterstützt bidirektionale Streaming-Kommunikation, was bedeutet, dass sowohl der Client als auch der Server Daten an den anderen senden können, ohne auf eine Anfrage des anderen zu warten. Dies ist besonders nützlich für Multiplayer-Spiele und Echtzeit-Anwendungen.
gRPC bietet Tools zur automatischen Codegenerierung, um Client- und Serverstubs sowie die erforderlichen Datenklassen basierend auf Ihren Protobuf-Definitionen zu erstellen. Dies vereinfacht die Entwicklung und verhindert Fehler durch manuelle Codeerstellung. Dies wird vorallem für die CustumApi genutzt, um beliebige Variablen zuübertragen.
Die ThalesApi basiert auf gRPC. Die ThalesApi erstellt einen gRPC Server dessen Services von einem Client abgerufen werden.

Sobald die ThalesApi der Scene hinzugefügt wird können die Sensordaten mit dem Client abgerfagt werden.

4.1.1 Übertragung von den Sensordaten

Die Sensor daten werden über den Client abgerufen.

Algorithmus 4.1: Sample Code Listing C++

```
int main(int argc, char** argv) {
    auto consoleOutput = std::make_shared<sick::dsm::consumers::
        ConsoleOutputConsumer>();
```

```
3     auto slaveProxy0 = std::make_shared<sick::dsm::proxies::SlaveProxy>("127.0.0
          .1", 5555, 6555);
4     slaveProxy0->setSendImmediately(true).setMaxSlices(1).setSliceBeamCount(2200
          ).setMtuSize(1460).setIndex(0).setIdentifier({127, 0, 0, 1, 0x15, 0xb3,
          0, 0}).initialise();
5     auto slaveProxy1 = std::make_shared<sick::dsm::proxies::SlaveProxy>("127.0.0
          .1", 5556, 6556);
6     slaveProxy1->setSendImmediately(true).setMaxSlices(1).setSliceBeamCount(2200
          ).setMtuSize(1460).setIndex(0).setIdentifier({127, 0, 0, 1, 0x15, 0xb4,
          0, 0}).initialise();
7     auto slaveProxy2 = std::make_shared<sick::dsm::proxies::SlaveProxy>("127.0.0
          .1", 5557, 6557);
8     slaveProxy2->setSendImmediately(true).setMaxSlices(1).setSliceBeamCount(2200
          ).setMtuSize(1460).setIndex(0).setIdentifier({127, 0, 0, 1, 0x15, 0xb5,
          0, 0}).initialise();
9     auto slaveProxy3 = std::make_shared<sick::dsm::proxies::SlaveProxy>("127.0.0
          .1", 5558, 6558);
10    slaveProxy3->setSendImmediately(true).setMaxSlices(1).setSliceBeamCount(2200
          ).setMtuSize(1460).setIndex(0).setIdentifier({127, 0, 0, 1, 0x15, 0xb6,
          0, 0}).initialise();
11    std::string connectionString = "192.168.100.1:30718";
12    std::shared_ptr<grpc::Channel> channel = sick::dsm::producers::
          ThalesGrpcClient::getInstance(connectionString).getChannel();
13
14    std::string scannerObjectPath0 = "BasicVehicle/microScan3-maxRange-
          Rasterization-0";
15    std::string scannerObjectPath1 = "BasicVehicle/microScan3-maxRange-
          Rasterization-1";
16    std::string scannerObjectPath2 = "BasicVehicle/microScan3-maxRange-
          Rasterization-2";
17    std::string scannerObjectPath3 = "BasicVehicle/microScan3-maxRange-
          Rasterization-3";
18    auto scanner0 = std::make_shared<sick::dsm::producers::ThalesScanner>(
          scannerObjectPath0, channel);
19    auto scanner1 = std::make_shared<sick::dsm::producers::ThalesScanner>(
          scannerObjectPath1, channel);
20    auto scanner2 = std::make_shared<sick::dsm::producers::ThalesScanner>(
          scannerObjectPath2, channel);
21    auto scanner3 = std::make_shared<sick::dsm::producers::ThalesScanner>(
          scannerObjectPath3, channel);
22    scanner0->addConsumer(slaveProxy0);
23    scanner1->addConsumer(slaveProxy1);
24    scanner2->addConsumer(slaveProxy2);
25    scanner3->addConsumer(slaveProxy3);
26
27    auto recorder0 = std::make_shared<sick::dsm::consumers::FileRecorder>("
          recorder0.bin");
28    auto recorder1 = std::make_shared<sick::dsm::consumers::FileRecorder>("
          recorder1.bin");
29    auto recorder2 = std::make_shared<sick::dsm::consumers::FileRecorder>("
          recorder2.bin");
30    auto recorder3 = std::make_shared<sick::dsm::consumers::FileRecorder>("
          recorder3.bin");
31
32
33    sick::dsm::proxies::MasterProxy masterProxy0(5555, {127, 0, 0, 1, 0x15, 0xb3
          , 0, 0});
```

```
34      masterProxy0.addConsumer(recorder0);
35      // masterProxy0.addConsumer(consoleOutput);
36      masterProxy0.start();
37
38      sick::dsm::proxies::MasterProxy masterProxy1(5556, {127, 0, 0, 1, 0x15, 0xb4
            , 0, 0});
39      masterProxy1.addConsumer(recorder1);
40      // masterProxy1.addConsumer(consoleOutput);
41      masterProxy1.start();
42
43      sick::dsm::proxies::MasterProxy masterProxy2(5557, {127, 0, 0, 1, 0x15, 0xb5
            , 0, 0});
44      masterProxy2.addConsumer(recorder2);
45      // masterProxy2.addConsumer(consoleOutput);
46      masterProxy2.start();
47
48      sick::dsm::proxies::MasterProxy masterProxy3(5558, {127, 0, 0, 1, 0x15, 0xb6
            , 0, 0});
49      masterProxy3.addConsumer(recorder3);
50      // masterProxy3.addConsumer(consoleOutput);
51      masterProxy3.start();
52
53      scanner0->start();
54      scanner1->start();
55      scanner2->start();
56      scanner3->start();
57
58      int counter = 0;
59      std::thread t1([&](){
60          while (true) {
61              scanner0->produce();
62              scanner1->produce();
63              scanner2->produce();
64              scanner3->produce();
65              std::this_thread::sleep_for(std::chrono::milliseconds(500));
66          }
67      });
68      std::thread t2([&](){
69          while (true) {
70              masterProxy0.produce();
71              masterProxy1.produce();
72              masterProxy2.produce();
73              masterProxy3.produce();
74              std::this_thread::sleep_for(std::chrono::milliseconds(500));
75          }
76      });
77      t1.join();
78      t2.join();
79      return 0;
80  }
```

Der Code beginnt mit Include-Deklarationen, die die notwendigen Header-Dateien für die Verwendung von C++-Standardbibliotheksfunktionen und anderen Bibliotheken wie gRPC und der ßick::dsmBibliothek importieren.

Im weiteren Verlauf des Codes werden verschiedene Variablen und Objekte deklariert. Dazu

gehören Objekte wie consoleOutput, slaveProxy0, slaveProxy1, slaveProxy2, slaveProxy3, channel, scanner0, scanner1, scanner2, scanner3, recorder0, recorder1, recorder2, recorder3, masterProxy0, masterProxy1, masterProxy2 und masterProxy3. Diese Objekte repräsentieren verschiedene Teile der Anwendung, darunter die Kommunikation mit Laserscannern, die Aufzeichnung von Daten und die Verwendung von gRPC für die Kommunikation.

Die Slave-Proxies (slaveProxy0, slaveProxy1, slaveProxy2, slaveProxy3) werden konfiguriert, um Verbindungen zu bestimmten IP-Adressen und Ports herzustellen. Diese Proxies werden wahrscheinlich verwendet, um Daten von den Laserscannern zu empfangen und zu verarbeiten.

Eine gRPC-Verbindung (channel) zu einer bestimmten IP-Adresse und Port wird erstellt. Dies wird wahrscheinlich für die Kommunikation mit einem entfernten Server verwendet.

Es werden Scanner-Objekte (scanner0, scanner1, scanner2, scanner3) erstellt und für die Verarbeitung von Daten aus den Laserscannern konfiguriert. Sie fügen auch Slave-Proxies als Verbraucher hinzu, um die verarbeiteten Daten weiterzuleiten.

Recorder-Objekte (recorder0, recorder1, recorder2, recorder3) werden erstellt. Diese Objekte zeichnen Daten in Dateien auf.

Master-Proxies (masterProxy0, masterProxy1, masterProxy2, masterProxy3) werden konfiguriert, um Daten von den Slave-Proxies zu empfangen und wahrscheinlich weitere Verarbeitung oder Aufzeichnung durchzuführen.

Es werden zwei Threads (t1 und t2) erstellt, die kontinuierlich Daten von Scannern und Master-Proxies produzieren. Diese Threads laufen in einer Endlosschleife und führen die entsprechenden Produktionsschritte aus.

4.1.2 Übertragung von Variablen

4.2 Umsetzung der Fahrphysik

4.2.1 Lenkung

Der Lenk mechanismus wird durch den Ackermannwinkel umgesetzt. Der Ackermannwinkel wird verwendet weil das Äußere Rat und das innere Rad einen anderen Wendekreis. Wenn die beiden Räder den gleichen Einschlagwinkel haben würde das Rad mit der geringeren haftung zum boden den Kontakt zudiesem verlieren.
Beide Räder sollen bei Kurvenfahrt auf einer Kreisbahn mit gleichem Mittelpunkt rollen, dem Momentanpol der Bewegung. Bei der Drehschemellenkung ist das automatisch der Fall. Bei einer Einzelradlenkung muss das innere Rad, das näher am Mittelpunkt liegt und dem äußeren vorauseilt, stärker einlenken als das äußere. Bei der Achsschenkellenkung müssen die Räder unterschiedlich geschwenkt werden, das außen liegende weniger stark als das innen liegende. Das erreicht man mit der Bildung des sogenannten Lenktrapezes aus dem Achskörper, den beiden leicht nach innen weisenden Lenkhebeln an den Achsschenkeln und einer Spurstange, die kürzer als der Achskörper ist. Dadurch entstehen beim Schwenken der Räder ungleich lange wirksame

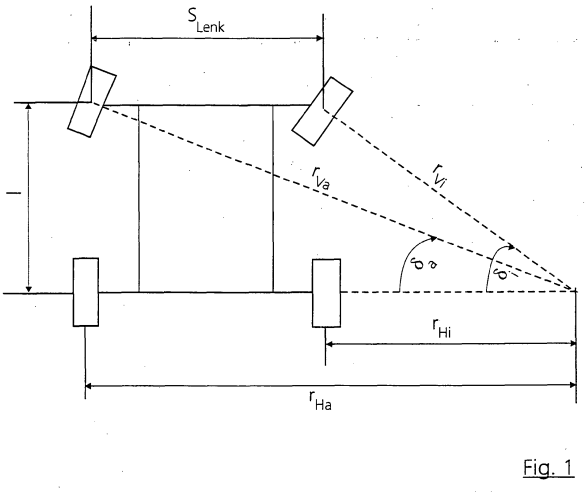

Abbildung 4.1: Achsschenkellenkung bei Starrachsen

Hebelarme, so dass sich die Verlängerungen aller Radachsen ungefähr im Kurvenmittelpunkt schneiden (Ackermann-Prinzip). Ein Hinweis auf ein richtig dimensioniertes Lenktrapez ist die Tatsache, dass sich die verlängerten Lenkhebel an den Achsschenkeln in Geradeausstellung in der Mitte der Hinterachse treffen (siehe oben in obiger Abbildung).

Wenn nach rechts gesteuert wird werden diese für den Winkel der beiden Räder verwendetet:

$$\text{Linkes Rad} = \arctan^{-1}\left(\frac{\text{Achsenabstand}}{\text{KurvenRadius} + \frac{\text{RadAbstand}}{2}}\right) \tag{4.1}$$

$$\text{Rechtes Rad} = \arctan^{-1}\left(\frac{\text{Achsenabstand}}{\text{KurvenRadius} - \frac{\text{RadAbstand}}{2}}\right) \tag{4.2}$$

Im ScrWheelcontroller Script wird der Ackermannwinkel berechnet wird. Der berechnete Ackermannwinkel wird an die vordernen beiden Räder weitergegeben. Die Räder werden zuvor in einer Liste hinzugefügt. Es brauchte ein Controller Script verwendet um die Struktur übersichtlicher zugestalten. Im Wheelscripts werden public bools angelegt die im Inpector dort wird fest gelegt um welches Rad es sich handelt.

```
1  private void manuelSteering()
2  {
3
4      if (steerInput > 0)
5      {
6          ackermannAngleLeft = Mathf.Rad2Deg * Mathf.Atan(wheelBase / (turnRadius
               + (rearTrack / 2))) * steerInput;
7          ackermannAngleRight = Mathf.Rad2Deg * Mathf.Atan(wheelBase / (turnRadius
               - (rearTrack / 2))) * steerInput;
8      }
9      else if (steerInput < 0)
10     {
11         ackermannAngleLeft = Mathf.Rad2Deg * Mathf.Atan(wheelBase / (turnRadius
               - (rearTrack / 2))) * steerInput;
12         ackermannAngleRight = Mathf.Rad2Deg * Mathf.Atan(wheelBase / (turnRadius
               + (rearTrack / 2))) * steerInput;
```

```
13      }
14      else
15      {
16          ackermannAngleLeft = 0;
17          ackermannAngleRight = 0;
18      }
19  }
```

4.2.2 Beschleunigung

4.2.3 Steuerung

Die Publicvariablen sollen verändert werden.

Kapitel 5

Ergebnis

5.1 Diskussion

Literatur

Bengel, Günther [2004]. *Grundkurs Verteilte Systeme: Grundlagen und Praxis des Client-Server-Computing - Inklusive aktueller Technologien wie Web-Services u. a. - Für Studenten und Praktiker*. 3., verbesserte und erweiterte Auflage. Springer eBook Collection Computer Science and Engineering. Wiesbaden: Vieweg+Teubner Verlag. ISBN: 9783322803399. DOI: 10.1007/978-3-322-80339-9 [siehe S. 8].

Änderungen

2020/03/13 Tippfehler korrigiert
aktuelle Formulierungen aus der Prüfungsordnung Technik übernommen
Formatdatei erklärt

2017/10/06 Anpassung an neuer Versionen diverse Pakete.

2016/03/16 Auf UTF-8 umgestellt, Indices.

2010/04/12 ToDo-Markierungen mit dem `\todo`-Kommando.

2010/01/27 Anhang (`appendix`), Selbständigkeits-Erklärung, `framed`-Paket.

2010/01/21 Abkürzungen (`acronym`), `table` und `tabular` benutzt, unübliche Pakete beigelegt.

2010/01/18 Code-Listings (`listings`), Literaturreferenzen `biblatex`)

2010/01/11 Initiale Version.

Liste der ToDo's