Masterthesis

Simulation und Erprobung verschiedener Szenarien des Autonomen Fahrens am TurtleBot3

Fakultät für Elektrotechnik und Informatik
Hochschule Stralsund

eingereicht am 06. Oktober 2021

Autor:
Moritz Wilke

Matrikelnummer:
15214

Studiengang:
Informatik - Master (INFM)

Erstgutachter:
Prof. Dr. rer. nat. Christian Bunse

Zweitgutachter:
Dr.-Ing. Jöran Pieper

Danksagung

Zusammenfassung

Masterthesis im Studiengang Informatik
Hochschule Stralsund
Fakultät für Elektrotechnik und Informatik

Moritz Wilke

Simulation und Erprobung verschiedener Szenarien des Autonomen Fahrens am TurtleBot3

Erstgutachter: Prof. Dr. rer. nat. Christian Bunse
Zweitgutachter: Dr.-Ing. Jöran Pieper

Abstract

Master thesis in the course of computer science
University of Applied Sciences Stralsund
Faculty of Electrical Engineering and Computer Science

Moritz Wilke

Simulation und Erprobung verschiedener Szenarien des Autonomen Fahrens am TurtleBot3

First supervisor: Prof. Dr. rer. nat. Christian Bunse
Second supervisor: Dr.-Ing. Jöran Pieper

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

Glossar

Hex-Farbcode
: Hexadezimale Repräsentation der Rot-, Grün- und Blau-Kanalwerte einer Farbe.

LiDAR
: Steht für *light detection and ranging*. Hierbei werden Laserimpulse ausgesandt und das zurückgestreute Licht detektiert, um Fernmessungen durchzuführen.

Morphologische Bildverarbeitung
: Beschreibt ein Modell für digitale Bilder, welches auf Verbandstheorie und Topologie basiert. [1]

Odometrie
: Odometrie beschreibt ein Schätzverfahren der Position und Orientierung eines Objektes. Hierbei werden Rückschlüsse aus der Radumdrehung gezogen.

Python
: Python ist eine Programmiersprache, die sowohl objektorientierte als auch funktionale Programmierung unterstützt. Des Weiteren kann Python als Scriptsprache verwendet werden.

Universally Unique Identifier
: Eine als hexadezimal notierte, eindeutige 16-Byte Identifizierungsnummer.

Video-Frame
: Ein einzelnes Bild einer Bildsequenz (=Videosequenz).

1 Einleitung

1.1 Motivation

Das Autonome Fahren ist ein Konzept, welches mehr und mehr an Bedeutung gewinnt. Da mit der zunehmenden Digitalisierung auch eine höherer Grad der Automatisierung gefragt ist, um z.B. Logistik- oder Transport-Prozesse kostengünstiger und sicherer zu gestalten, ergeben sich hierbei auch Anforderungen an das maschinengesteuerte Fahren. Zum aktuellen Zeitpunkt kommt eine Hybrid-Form in modernen Fahrzeugen bereits zum Einsatz, wodurch unterschiedliche Fahrmannöver, wie z.B. das Einparken oder Halten einer Spur, teilautomatisiert werden. Damit das Fahren dauerhaft oder vollständig von einem Softwaresystem übernommen werden kann, muss es auch in nicht-alltäglichen Situationen nachvollziehbar entscheiden.

Um zwischen den einzelnen Ebenen der Autonomie zu unterscheiden, werden verschiedene Autonomiestufen definiert.

- Autonomiestufe 0: "Driver only", der Fahrer fährt selbst ohne Assistenzsysteme.
- Autonomiestufe 1: Der Fahrer wird bei der Bedienung des Fahrezeugs unterstützt, wie z.B. durch einen Tempomat.
- Autonomiestufe 2: Das Fahrzeug ist teilautomatisiert und bietet Assistenzsysteme für automatisiertes Einparken oder zum halten der Spur.
- Autonomiestufe 3: Einzelne Fahrmannöver, wie z.B. das Wechseln der Fahrspur, werden vom Fahrzeug automatisiert durchgeführt. Falls Handlungsbedarf für den Fahrer besteht, wird dieser innerhalb einer Vorwarnzeit zur Übernahme der Fahrzeugführung aufgefordert. Aktuell wird darauf hingearbeitet, Fahrzeuge dieser Autonomiestufe für den öffentlichen Straßenverkehr zuzulassen.
- Autonomiestufe 4: Das Softwaresystem übernimmt dauerhaft die Steuerung des Fahrzeugs. Falls der Fahrer die Fahrzeugführung übernehmen muss, wird dieser innerhalb einer Vorwarnzeit benachrichtigt.
- Autonomiestufe 5: Das Fahrzeug ist vollautomatisiert, ein Fahrer ist nicht länger erforderlich. [2]

Insbesondere für die Autonomiestufe 3 gab es bereits mehrere Projekte, die solche Systeme in der Praxis getestet haben. Im Juli 2014 gab es hierzu ein Pionierprojekt, bei welchem der Mercedes-Benz Future Truck 2025 auf einem gesperrten Autobahnteilstück bei Magdeburg autonom gefahren ist. Das System hat hierbei beispielsweise das mittige Fahren innerhalb der rechten Fahrspur sowie das Beschleunigen und Bremsen übernommen. Der Fahrer konnte sich somit anderen Aufgaben widmen, wie z.B. der Planung der nächsten Tour oder auch der Frachtkontrolle über digitale Displays. Durch den Wegfall mehrerer Bedienelemente, hat der Fahrzeugführer in diesem Beispiel auch mehr Platz im Innenraum. [3]

Zum Erreichen der Autonomiestufe 4 und 5 stellen sich hierbei neben der rechtlichen Grundlage auch technische Herausforderungen. Ein Beispiel hierfür ist die Erkennung von Wildtieren, die den Straßenverkehr behindern oder gefährden können. Der schwedische Autohersteller Volvo kann mit seiner Software beispielsweise die heimische Fauna bestehend aus Tieren wie Elchen, Rehen oder Rentieren zuverlässig erkennen, scheitert jedoch beispielsweise an Kängurus. [4] Dies zeigt, dass es unzählig viele Szenarien gibt, die es beim Autonomen Fahren zu betrachten gibt, um die Fahrsicherheit entsprechend zu gewährleisten. Hier gilt es auch darauf zu achten, dass die Sicherheit nicht nur für den Fahrzeugführer, sondern auch für Fußgänger, Radfahrer u.Ä. gesichert ist. Aus diesem Grund werden in dieser Thesis eher Randszenarien behandelt, die nicht im alltäglichen Straßenverkehr auftreten jedoch kritisch für die Sicherheit von Fahrzeugführer und Umwelt sind.

Abgesehen von öffentlichen Straßen gibt es auch andere Bereiche, in denen das Autonome Fahren Einzug erhalten hat. Ein Beispiel sind hier automatisierte Transportsysteme, wie sie z.B. in Waren- oder Krankenhäusern zu finden sind. Die Roboter sind hier zuständig, sich innerhalb des eines Gebäudes zu orientieren und einzelne Güter zu transportieren. Wichtig ist hierbei insbesondere, dass sich die Roboter nicht gegenseitig behindern oder zur Gefahr für den Menschen werden.

Die Orientierung in Gebäuden ist hierbei eine besondere Herausforderung, da es für diese im Regelfall keine digitalen Karten gibt, anhand der sich die Roboter orientieren können. Diese Karten werden somit teils manuell erstellt, wodurch sich Probleme in der Genauigkeit sowie Flexibilität offenbaren. Der Vermessungsprozess kann darüber hinaus mehrere Monate andauern, wie es z.B. im Greifswalder Klinikum der Fall war, als der Einsatz von TRANSCAR-Robotern vorbereitet wurde. [1]

[1] `https://webmoritz.de/2013/05/16/ameisen-im-klinikum/`

1.2 Zielstellung

Für das autonome Fahren existieren schon zuverlässige Lösungen, die im Alltag funktionieren. In der Thesis wird deshalb der Fokus auf Randszenarien gelegt, die im Kern die Erkennung und das Verhalten eines Fahrzeugs an einer ungeregelten Kreuzung beschreiben. Die Zielstellung ist, basierend auf dem Quellcode des "TurtleBot 3 AutoRace-Projekts [2], ungeregelte Kreuzungen zu identifizieren. Der Roboter soll hier feststellen, wo die Kreuzung beginnt und in welche Richtungen er fahren kann. Zusätzlich soll er erkennen, aus welchen Richtungen er mit Gegenverkehr rechnen muss. Umgesetzt wird dies mit Bildverarbeitung. Im zweiten Schritt soll der Roboter zusätzlich erkennen, ob andere Fahrzeuge in der Nähe sind mit denen eine Kollisionsgefahr besteht. Hierfür wird eine Technik genutzt, die an Connected Cars angelehnt ist. Im letzten Schritt sollen beide Komponenten zusammengeführt werden. Der Roboter soll sich in seiner Umgebung so verhalten, dass grundlegende Regeln des Straßenverkehrs, wie z.B. rechts vor links, umgesetzt werden. Darüber hinaus soll der Roboter auch potenzielle Regelbrüche, wie das nehmen der Vorfahrt, frühzeitig erkennen und eine Kollision vermeiden.

Grundsätzlich gab es in diese Richtung bereits Forschungen, jedoch waren diese Versuche auf Fahrzeuge innerhalb einer Fahrbahn begrenzt, bei denen versucht wurde, Kollisionen durch diverse Mannöver zu vermeiden [5]. Ein anderer Ansatz zu Erkennung von Kreuzungen war auf Basis von Punktwolken, die durch einen LiDAR-Sensor erstellt wurden. Dieser Ansatz setzt jedoch eine physische Fahrbahnbegrenzung, wie durch einen Bordstein, voraus [6].

[2] `https://github.com/ROBOTIS-GIT/TurtleBot3_autorace`

2 Grundlagen

2.1 Tools und Frameworks

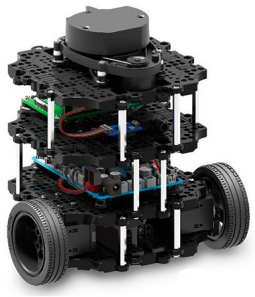
2.1.1 ROS

ROS (Robot Operating System) ist ein Framework, mit dem Roboter programmiert werden können. Entgegen dem Namen handelt es sich hierbei nicht um ein Betriebssystem, sondern vielmehr um eine Middleware. Im Zentrum von ROS stehen einzelne Nodes, welche die Programmlogik beinhalten. Die Nodes kommunizieren über sogenannte Topics mit Nachrichten. Diese Nachrichten können skalare Datentypen oder komplexe Objekte beinhalten. Eine einzelne Node sendet (=publish) oder empfängt (=subscribe) Nachrichten von einem oder mehreren Topics. Das grundlegende Paradigma erinnert somit stark an Reactive Programming. Einzelne Nodes lassen sich zur Laufzeit dynamisch starten und stoppen, wodurch eine komplexe, ressourcensparende Programmarchitekur möglich ist.

Die unterstützten Programmiersprachen sind hierbei Python und C++. Hierbei ist es auch möglich, Nodes hybrid mit C++ und Python zu entwickeln. Zusätzlich stehen dem Entwickler mehrere Bibliotheken zur Verfügung, um die Entwicklung zu vereinfachen. [7]

2.1.2 Gazebo

Gazebo ist eine Simulationssoftware, mit der Roboter in komplexen Umgebungen simuliert werden können. Für Gazebo existiert eine Integration für ROS, wodurch es möglich ist, die Roboter innerhalb einer Simulation über ROS-Nodes zu kontrollieren. Weiterhin gibt es viele bestehende Modelle, welche unter Anderem den TurtleBot 3 repräsentieren, sowie vorgefertige Welten, in denen das implementierte Verhalten erprobt werden kann. Zusätzlich nutzt Gazebo eine Physik-Engine, wodurch Schwerkraft oder Kollisionen berechnet werden, und eine Integration für verschiedenste Sensoren, die auf dem Roboter verbaut sind. Hierdurch ist es möglich, LiDAR- oder Kameradaten bereits in der Simulation auszuwerten. [8]



Quelle: https://www.roscomponents.com/1326-big_default_2x/turtlebot-3.jpg
Abbildung 2.1: TurtleBot 3 Burger

2.1.3 TurtleBot 3

Der TurtleBot 3 ist ein programmierbarer und mobiler Roboter. Grundlegend ist dieser Roboter anpassbar, bringt jedoch im Standard zwei Elektromotoren zur Bewegung, ein 360-Grad-LiDAR-System, eine Kamera sowie einen Raspberry Pi 3 als Computer mit. Den TurtleBot 3 gibt es in den Ausführungen Burger und Waffle. In der Thesis wird das Modell Burger genutzt. [9]

In Abbildung 2.1 ist der TurlteBot 3 Burger zu sehen. Ein weiteres Modell, welches käuflich erwerblich ist, ist der TurtleBot 3 Waffle. Im Rahmen dieser Thesis wird ausschließlich mit dem TurtleBot3 Burger gearbeitet.

2.1.4 OpenCV

OpenCV ist eine Bibliothek, welche Algorithmen und Hilfsfunktionen für die Verarbeitung von Bildern und Videos liefert. Die Bibliothek kann in den Programmiersprachen C, C++, Java und Python genutzt werden. OpenCV findet hauptsächlich Anwendung in den Feldern Computer Vision und Künstliche Intelligenz.

Grundlegend besteht bei OpenCV die Möglichkeit, ein Video-Frame zu analysieren und zu bearbeiten. Hierdurch besteht die Möglichkeit, Bilder für Folgeprozesse aufzubereiten oder diese direkt auszuwerten, um beispielsweise Objekte zu erkennen.

2.2 Softwaredesign

Wie bereits erwähnt, wird in dieser Thesis ein bestehendes Projekt aufgegriffen. Somit steht die Wahl der Programmarchitektur nicht frei, sondern wird in die bestehende Programmarchitektur eingegliedert.

In der bestehenden Architektur gibt es Core-, Detection- und Control-Nodes.

2.2.1 Core-Nodes

In den Core-Nodes werden die Detection- und Control-Nodes dynamisch gestartet und bei Bedarf gestopt. Hierfür werden diverse Topics der Detection-Nodes abonniert, wie z.B. der Erkennung eines Ampellichts, und bei Bedarf die entsprechende Control-Node gestartet. Zusätzlich wird hier der aktuelle Modus vorgehalten, der beschreibt, ob der Roboter beispielsweise der Fahrspur folgt oder versucht einzuparken. Die Core-Nodes sind nicht dafür verantwortlich, Daten zu verarbeiten oder den Roboter zu steuern.

Die bestehende Implementierung beinhaltet zwei Core-Nodes, *core_mode_decider* und *core_node_controller*. Die Node *core_mode_decider* reagiert auf die Detektion von Verkehrsschildern und wechselt je nach erkanntem Schild den Modus des Roboters. Der Fallback-Modus ist hier in jedem Fall *lane_following* - also die Detektion sowie das Fahren in der Fahrspur. Wenn der Roboter ein Schild zum einparken erkennt, wird der Roboter hier beispielsweise in den Modus *parking* versetzt. Die Node *core_node_controller* reagiert auf die Veränderung des Modus. Abhängig vom Modus werden hier die Nodes aktiviert oder deaktiviert.

2.2.2 Detection-Nodes

In den Detection-Nodes werden Sensor- oder Kameradaten sowie die odometrische Position des Roboters ausgewertet. Jede Detection-Node hat eine designierte Aufgabe, wie z.B. das erkennen einer Ampel oder der Fahrspur. Falls das designierte Objekt erkannt wird, wird ein Nachrichten-Objekt vorbereitet. In diesem Nachrichtenobjekt werden die Daten zusammengefasst, die benötigt werden, um eine entsprechende Entscheidung zu liefern. Bei der Fahrspurerkennung wird beispielsweise die berechnete Fahrspur-Mitte im Nachrichtenobjekt persistiert. Bei erfolgreicher Detektion wird das Nachrichtenobjekt an ein spezifisches Topic gesendet.

Die bestehende Implementierung beinhaltet sechs Detection-Nodes. Die Nodes, die für die Arbeit Bedeutung haben, sind *detect_lane* und *detect_sign*. Mit der Hilfe von Bildverarbeitungs-Algorithmen werden hier also die Fahrbahn (*detect_lane*) und verschiedene Verkehrsschilder (*detect_sign*) detektiert.

2.2.3 Control-Nodes

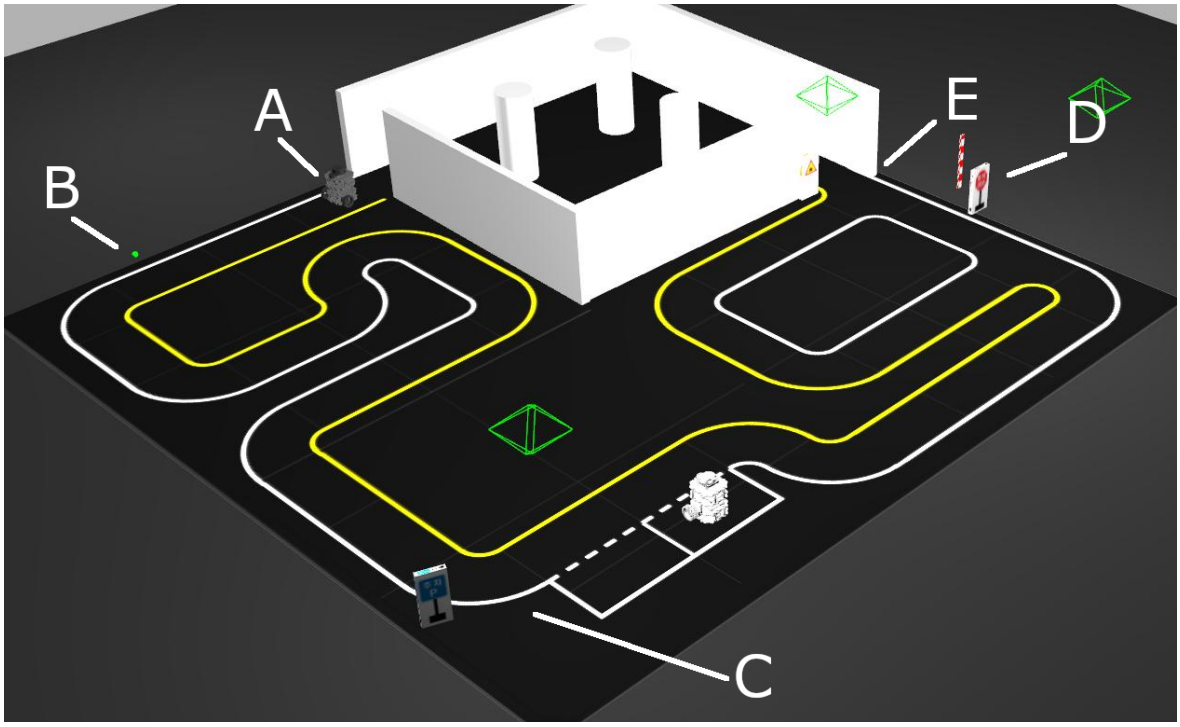
In den Control-Nodes wird der Roboter basierend auf den Erkenntnissen der Detection-Nodes gesteuert. Hierfür abonnieren die Control-Nodes zunächst die Topics der Detection-Nodes. Falls eine Nachricht empfangen wird, können auf Grundlage dieser Berechnungen durchgeführt werden um den Roboter schlussendlich zu steuern. Diese Steuerung erfolgt in der Regel über Twistnachrichten, die an andere Topics versendet werden. Diese Twistnachrichten beinhalten Daten, die die lineare und angulare Beschleunigung beschreiben.

Die bestehende Implementierung beinhaltet zwei Control-Nodes. Diese sind *control_lane* und *control_parking*. Lediglich *control_lane* ist im weiteren Verlauf der Thesis relevant. Hier wird wie bereits beschrieben die Farhbahn-Detektion abonniert. Durch diese wird das Zentrum der Fahrbahn bestimmt. Aus diesem Zentrum wird abgeleitet, wie sehr angular beschleunigt werden soll.

2.3 Simulation

Für das bestehende Projekt existiert bereits eine Simulation, in der die einzelnen Herausforderungen des TurtleBot AutoRace geprüft werden können. In Abbildung 2.2 ist hier der grundsätzliche Aufbau zu sehen. Diese digitale Version dient zum testen und implementieren der Software für den TurtleBot 3 und wurde real nachgebaut. Ein wichtiges Merkmal ist die Fahrspur, die im Laufe des Projekts beibehalten wurde. Hierbei ist die rechts Fahrbahnbegrenzung weiß (Hex-Farbcode #ffffff) und die linke Fahrbahnbegrenzung gelb (Hex-Farbcode #ffff01). Hierdurch ergibt sich, dass die Fahrbahn nicht bidirektional sondern nur in eine Richtung befahren werden kann. Am Punkt A in der Abbildung befindet sich der Roboter. Dieser startet mitten in der Fahrbahn und fährt entsprechend der Softwareumsetzung direkt zur ersten Herausforderung am Punkt B. Punkt B beschreibt die Ampel. Wenn der Roboter sich der Ampel nähert, wird die Ampel von grün auf gelb geschaltet. Die erwünschte Aktion vom Roboter ist, dass dieser langsamer werden soll. Wenn der Roboter noch näher an die Ampel fährt, wird diese schlussendlich rot. Der Roboter soll nun stehen bleiben, bis die Ampel wieder auf grün schaltet. Nach der Ampel wird das Erkennen und Einhalten der Fahrspur geprüft, indem Kurven gefahren werden, bis der Roboter an Punkt C ein Parkschild erkennen soll. Bei dieser Herausforderung soll der Roboter automatisch in der freien Parkfläche ein- und nach erfolgreicher Absolvierung wieder ausparken. Nachdem die Herausforderung abgeschlossen wurde, nähert sich der Roboter der Zollschranke an Punkt D. Hier soll der Roboter zunächst ein Schild erkennen, welches ihn auf die bevorstehende Aufgabe hinweist. Wenn der Roboter sich in der Nähe der Zollschranke befindet, wird

diese heruntergefahren. Der Roboter soll nun vor der Zollschrank warten, bis diese wieder oben ist. Wenn der Roboter dies geschafft hat, ist die letzte Herausforderung die Navigation durch den Tunnel an Punkt E. In diesem Tunnel kann die Kamera nicht genutzt werden, da es hier wenig Licht gibt. Der Roboter soll deshalb mit Hilfe des LiDAR-Systems kollisionsfrei durch den Tunnel navigieren.



Quelle: `https://emanual.robotis.com/assets/images/platform/turtlebot3/autonomous_driving/autorace_map_mission.png`

Abbildung 2.2: Karte des TurtleBot AutoRace 2017

2.4 Techniken

2.4.1 Objekterkennung

Wie bereits erwähnt, sollen in der Arbeit Kreuzungen explorativ erkannt werden. Das bedeutet, dass noch keine Karte oder ähnliches besteht, auf denen alle Kreuzungen aufgelistet sind. Um eine Kreuzung zu erkennen, ist eine Objekterkennung notwendig, welche auf den Kameradaten des TurtleBot 3 agiert.

Bildverarbeitung

Bei der Bildverarbeitung wird auf Grundlage von Bildverarbeitungs-Algorithmen versucht, Erkenntnisse aus Bilddaten zu gewinnen. Hierfür müssen die Bilder in der Regel bereinigt, gefiltert oder transformiert werden. Bei der Bereinigung ist das Hauptaugenmerk, Rauschen zu unterdrücken. Hierfür können die Bilddaten geglättet, gefiltert oder

morphologische Operationen angewendet werden. Insbesondere die morphologischen Operationen werden benötigen, um gewisse Strukture oder Figuren, die potenziell durch Bildrauschen beschädigt wurden, zu reparieren.

Bei der Filterung der Bilddaten können diverse Operationen angewendet werden, um Objektstrukturen hervorzuheben oder zu unterdrücken. Ein bekanntes Beispiel ist hier die Kantendetektion, die unter Anderem mit dem Sobel-Operator durchgeführt werden kann, welcher unter die Faltungsoperationen fällt. Hierbei wird eine 3x3-Matrix iterativ auf jeden Bildpunkt mit seinen umliegenden Pixeln multipliziert. Die einzelnen Produkte der Multiplikation werden aufsummiert, um den neuen Grauwert des Pixels vom ursprünglichen Bildpunkt zu berechnen. Das Resultat ist hierbei abhängig vom Aufbau der zu verschiebenden Matrix. Der Sobel-Operator G_x kann beispielsweise genutzt werden, um Vertikale Kanten zu detektieren, G_y hingegen wird eher horizontale Kanten herausfiltern. Da die Berechnung der Farbwerte für die Bildpunkte negativ sein kann, können Schwellwerte genutzt werden, um zu entscheiden, ob ein Pixel Teil einer Kante ist. In Kombination der vertikalen sowie horizontalen Filter mit einem Schwellwert können gute Resultate bei der Kantendetektion erzielt werden. [10]

$$G_x = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix} \; G_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}$$

Der letzte Teil der genutzten Bildverarbeitungsalgorithmen sind die Transformationen. Hierbei können die Bilddaten beispielsweise rotiert oder verschoben werden, um die Perspektive auf das Bild zu verändern. Ein Beispiel hierfür ist die Rotation, Dehnung oder Translation. Eine Rotation kann zum Beispiel hilfreich sein, wenn die Kamera, die auf ein Fahrzeug montiert ist, eine leichte Schräglage hat. Somit hätte auch das Urbild, welches bei der Bildverarbeitung analysiert wird, eine Schräglage. Durch eine Rotation entsprechend des Neigungswinkels kann das Bild hierbei korrigiert werden. Ein Bild kann am Nullpunkt beispielsweise durch Multiplikation mit der Matrix R um den Winkel α rotiert werden.

$$R = \begin{pmatrix} cos(\alpha) & sin(\alpha) \\ -sin(\alpha) & cos(\alpha) \end{pmatrix}$$

Direkte Algorithmen zur Objekterkennung in Bildern sind z.B. der Marr-Hildreth-Operator (auch Laplacian of Gaussian genannt) oder SIFT (Scale-invariant feature transform) in Kombination mit einem Feature-Matcher wie FLANN (Fast Library for

Approximate Nearest Neighbours). Hierfür ist es notwendig, dass ein Referenzbild vom zu detektierenden Objekt existiert. Wie bereits beschrieben gibt es eine Node, welche Verkehrsschilder detektiert. Im Hintergrund arbeitet diese Node mit diesem Prinzip. Als Referenzbilder sind hier diverse Verkehrsschilder hinterlegt. Der Prozess funktioniert vereinfacht beschrieben so, dass beispielsweise über SIFT einzelne Schlüsselpunkte sowohl im Referenzbild als auch im zu verarbeitenden Bild berechnet werden. Im Anschluss werden die Abweichung bzw. Distanzen zwischen diesen Schlüsselpunkten berechnet. Wenn ausreichend Abweichungen unter einer gewissen Schwelle liegen, ist das Referenzbild im zu verarbeitenden Bild enthalten. Dieser Ansatz funktioniert recht zuverlässig und kann auch Referenzbilder aus anderen Perspektiven erkennen. Die Kernproblematik hierbei ist jedoch, dass diese Referenzbilder existieren müssen. Für eine Detektion von Kreuzungen ist dieser Ansatz also nicht geeignet, da eine grenzenlose Menge an Referenzbilder erstellt und geprüft werden müsste.

Bei der Bearbeitung der Thesis wurden weitere Algorithmen eingesetzt. Diese werden an entsprechender Stelle erklärt. Dies hat den Hintergrund, dass die Nutzung und die Auswirkungen der Parametrisierung so besser beleuchtet werden kann. Im Kern beruhen diese Algorithmen auf den beschriebenen Prinzipien.

Der Vorteil bei der Nutzung von Bildverarbeitungs-Algorithmen liegt darin, dass die Detektion hierbei auf Strukturen beruht. Solange sich die Strukturen im Kern nicht ändern, kann das Programm durch eine Anpassung der Parametrisierung an neue Umgebungen angepasst werden. Zusätzlich intigriert sich der Bildverarbeitungs-Ansatz gut in das bestehende Projekt, da sämtliche existierende Detektionen ebenfalls mit OpenCV umgesetzt wurden.

Neuronale Netze

Ein anderer Ansatz zur Erkennung von Objekten ist die Analyse der Bilddaten in einem Neuronalen Netz. Ein solches Netz besteht aus mehreren Neuronen, welche miteinander verbunden sind. Die einzelnen Neuronen beinhalten hierbei im Regelfall numerische Werte. Die Verbindungen geben hierbei an, wie diese Werte zu wichten sind. Die Eingabedaten werden durch Eingabe-Neuronen vom Netz ausgewertet. In Ausgabe-Neuronen beinhalten hierbei die Ergebnisse.

Eine spezifische Architektur, um unter Anderem Bilddaten zu verarbeiten, ist das Convolutional Neural Network (CNN). Hierbei wird ein Bild in seine Farbkanäle zerlegt und mit Faltungsalgorithmen, ähnlich zum beschriebenen Sobel-Operator, verarbeitet. Mit Hilfe von ausreichenden Trainingsdaten, versucht die Maschine selbst die Faltungen zu erkennen, die notwendig sind, um die gewünschten Ausgaben zu erzielen.

Eine große Herausforderung ist hierbei, das Netz richtig zu konfigurieren. Bei einem Neuronalen Netz gibt es viele Parameter, um die Genauigkeit bei der Verarbeitung der Eingabedaten zu erhöhen, wie z.B. die Anzahl der Neuronen oder Schichten. Komplexere Parameter beschreiben die Fehlerfunktionen, um Lernfortschritte zu repräsentieren, oder die Aktivierungsfunktion. Die Aktivierungsfunktion ist dafür zuständig, den Wert vom vorhergehenden, verknüpften Neuron zu verarbeiten. Durch die Vielfalt an Parametern ist es schwierig, eine zuverlässige Genauigkeit zu erzielen. Jedoch ist es insbesondere beim Autonomen Fahren erstrebsam, eine Genauigkeit von annähernd 100% zu erreichen, um Unfälle zu vermeiden.

Aus diesem Grund wird der Ansatz zur Objekterkennung mit einem Neuronalen Netz nicht weiter verfolgt. Eine weitere Problematik ist, dass kein direkter Einfluss darauf genommen werden kann, welche Faltungen das Netz lernen würde. Somit besteht die Möglichkeit, dass eine Detektion von Kreuzungen ein komplett neues Netz erforder würde, wenn sich beispielsweise die Farbe der Fahrbahn ändert.

Der besondere Vorteil eines neuronalen Netzes wäre, dass es leicht auszuwechseln ist. Dies ist damit begründet, dass die Eingabe- sowie Ausgabe-Neuronen fix sind. Hierdurch würde das Programm deutlich an Flexibilität gewinnen. Zusätzlich würde sich dieser Ansatz für komplexere Kreuzungen eignen. Das kann bedeuten, dass es dort z.B. verschiedene Fahrspuren zum abbiegen oder verkehrsregelnde Elemente wie Ampeln gibt. Ein Versuch, der auf einer Kreuzung mit einer Trennung in zwei Straßen aufbaut, wurde bereits im Jahr 1995 durchgeführt [11]

2.4.2 Connected Cars

Zur Kommunikation zwischen den Fahrzeugen wird eine Kommunikationsmethode gewählt, die an Connected Cars angelehnt ist. Der Begriff Connected Cars beschreibt grundsätzlich den Datenaustausch von beispielsweise Positionsdaten mit anderen Verkehrsteilnehmern oder Infrastrukturen, um den Straßenverkehr effizienter und sicherer zu gestalten. Hier kann unter Anderem in folgende Kategorien unterschieden werden:

- Vehicle to Vehicle (V2V) - die Fahrzeuge kommunizieren ihre Position und Geschwindigkeit an andere Fahrzeuge
- Vehicle to Pedestrian (V2P) - Fahrzeuge und gefährdete Gruppen, wie z.B. Fußgänger oder Radfahrer, kommunizieren ihre Position und Geschwindigkeit untereinander
- Vehicle to Infrastructure (V2I) - Fahrzeuge und Komponenten der Infrastruktur, wie z.B. Ampeln oder Schilder, kommunizieren Daten untereinander

Für die Bearbeitung der Thesis ist hierbei insbesondere die V2V-Kommunikation von Bedeutung. Im bestehenden AutoRace-Projekt ist es bereits so, dass ein Roboter seine odometrischen Daten veröffentlicht. Diese Daten enthalten Angaben zur Position und zur linearen sowie angularen Beschleunigung. Um eine V2V-Kommunikation nachzuahmen, veröffentlicht jeder Roboter seine odometrischen Daten in einem globalen Topic. Zusätzlich zu den odometrischen Daten enthält veröffentlichte Nachricht einen für jeden Roboter eindeutigen Universally Unique Identifier (UUID).

2.5 Requirements-Engineering

2.5.1 Anforderungsanalyse

„Eine vom Benutzer benötigte Eigenschaft oder Fähigkeit, die eine Software erfüllen oder besitzen muss, um einen Vertrag, einen Standard, eine Spezifikation oder ein anderes formales Dokument zu erfüllen.“ [12]

Nach IEEE ist dies die Definition einer Anforderung. Nach dieser Definition lassen sich zwei verschiedene Klassen von Anforderungen ableiten. Die Fähigkeiten eines Systems sind hierbei die Funktinalen Anforderungen (FA). Die Eigenschaften eines Systems hingegen werden von den Nicht-Funktionalen Anforderungen (NFA) repräsentiert.

Funktionale Anforderungen

Funktionale Anforderungen werden durch sogenannte Anforderungsquellen identifiziert. Solch eine Anforderungsquelle kann eine Person, ein Prozess in einem Betrieb oder ein Dokument sein. Insbesondere die Personen, auch oft Stakeholder genannt, sind geeignet, um diverse Problemdomänen zu beschreiben. Anforderungen können darüber hinaus über andere Wege, wie z.B. Apprenticing oder Feldbeobachten, ermittelt werden. Eine Möglichkeit, um Funktionale Anforderungen zu beschreiben und zu dokumentieren sind sogenannte Anforderungsschablonen. Eine beispielhafte Vorgehensweise hierfür sind „Mustergültige Anforderungen - die SOPHIST Templates für Requirements“-Schablonen (MASTeR). [13][S. 219-225] Über verschiedene Schlüsselwörter kann hierüber eine Anforderung definiert werden. Grundsätzlich baut dieser Prozess auf vier Schritten auf.

Schritt 1 Es wird festgelegt, wie wichtig die Anforderung ist. Schlüsselwörter hierfür sind „Das System **muss**“ (hohe Priorität) oder aber „Das System **kann**“ (niedrige Priorität). Bei einer juristisch verbindlichen Anforderung sollte beispielsweise das Modalverb **muss** eingesetzt werden. „Das System muss...“ ist dann das erste Anforderungsfragment.

Schritt 2 Die geforderte Funktionalität wird identifiziert. Das Zentrum einer jeden Anforderung ist die Funktionalität, die erwünscht ist. Diese Funktionlitäten sind Prozesse, wie z.B. Vorgänge oder Tätigkeiten. Ein Beispiel hierfür ist „Das System muss **speichern**."

Schritt 3 Die Art der geforderten Funktionalität wird festgelegt. Für das Schreiben von Anforderungen sind hierbei drei Systemaktivitäten relevant: Selbsttätige Systemaktivitäten, Benutzerinteraktionen und Schnittstellenanforderungen. Aus „Das System muss **speichern**" wird so beispielsweise „A-S-001, Version 1: Selbsttätige Systemaktitivät. Das System löst eine Backuperstellung aus.". Anders wäre es, wenn dem Nutzer beispielsweise eine Möglichkeit zur Speicherung gegeben werden würde. Dann würde es eine zusätzliche Benutzerinteraktions-Anforderung geben.

Schritt 4 Das Objekt, für welches die Anforderung gefordert wird, muss identifiziert werden. In der definierten Anforderung A-S-001 ist beispielsweise nicht klar, was genau persistiert werden muss. Besser ist z.B. „Das System löst eine Backuperstellung der Kundendaten aus."

Schritt 5 - optional Es wird festgelegt, unter welchen Bedingungen die Funktionalität durchgeführt werden soll. Die vollständige Anforderung könnte dann so aussehen: „A-S-001, Version 1: Selbsttätige Systemaktitivät. Wenn es 02:00 Uhr ist, muss das System eine Backuperstellung der Kundendaten auslösen."

Nicht-Funktionale Anforderungen

Bei den Nicht-funktionalen Anforderungen wird zwischen den folgenden Kategorien unterschieden: [13][S. 268]

Anforderungen an die Benutzeroberfläche: bündelt Anforderungen zur Beschreibung, wie sich das System dem Benutzer darstellen soll

Technologische Anforderungen: schränken den Lösungsraum für die Realisierung dadurch ein, dass sie Lösungsvorgaben geben oder die Umgebung beschreiben, in der das System betrieben werden soll

Anforderungen an sonstige Lieferbestandteile: beschreiben alle Produkte, die neben dem eigentlichen System geliefert werden müssen

Anforderungen an durchzuführende Tätigkeiten: beschreiben den Entwicklungsprozess beziehungsweise einzelne, dem Entwicklungsprozess nachgelagerte, Tätigkeiten

Rechtlich-vertragliche Anforderungen: beschreiben Regelungen zwischen Auftraggeber und Auftragnehmer

Qualitätsanforderungen: Die Qualitätsmerkmale eines Systems sind im Standard ISO25010 [14] wie folgt definiert:

- Functional Suitability (Funktionalität): vollständig hinsichtlich Softwarefunktionen, funktional korrekt, angemessene Funktionalität
- Reliability (Zuverlässigkeit): ausgereifte Softwarequalität, Verfügbarkeit, Fehlertoleranz, Wiederherstellbarkeit
- Performance Efficiency (Effizienz): Laufzeitverhalten, Ressourcennutzung, Kapazitäten schonen
- Compatibility (Kompatibilität): Co-Existenz zu weiterer Software, Interoperabilität
- Usability (Benutzbarkeit): Erkennbarkeit, Erlernbarkeit, Bedienbarkeit, Schutz vor Fehlbedienung, Ästhetische Benutzerschnittstelle, Zugänglichkeit
- Maintainability (Wartbarkeit): Modularität, Wiederverwendbarkeit, Analysierbarkeit, Modifizierbarkeit, Testbarkeit
- Portability (Übertragbarkeit): Adaptivität, Installierbarkeit, Austauschbarkeit
- Security (Sicherheit): Datenschutz, Integrität, nicht manipulierbar, Authentizierbarkeit

2.5.2 Anforderungsmanagement

Durch Anforderungsmanagement können einzelne Anforderungen an ein System dokumentiert und nachvollziehbar abgelegt werden. Dies hat den Vortel, dass so zu einem späteren Zeitpunkt Erkenntnisse darüber gezogen werden können, wie eine einzelne Anforderung und somit auch Änderung im System zustande gekommen ist. Zusätzlich sind Anforderungen die Grundlage bei der Kommunikation mit Kunden und weiteren Projektbeteiligten.

Ein Anforderungsmanagement ist also dann relevant, wenn durch die Anforderungen ein System oder Projekt mit hoher Lebensdauer beschrieben wird. Ein anderer Faktor, der für ein ausgereiftes Anforderungsmanagement sprechen würde, wären mehrere Projektteilnehmer oder eine große Zahl an Anforderungen. [13, S. 368-372]

Da keiner dieser Punkte erfüllt ist, wird auf ein komplexes Anforderungsmanagement verzichtet. Alle Anforderungen an das System werden in dieser Thesis dokumentiert.

3 Konzeption

3.1 Szenarien

Die Szenarien haben eine ungeregelte Kreuzung gemeinsam. Ziel ist es, dass das Fahrzeug eine Kreuzung erkennt sowie mit potenziellem Gegenverkehr umgehen kann. Das Fahrzeug soll erkennen, in welche Richtungen es abbiegen darf, sich für eine Richtung entscheiden und das entsprechende Fahrmannöver ausführen. Zusätzlich sollen die odometrischen Koordinaten der Schlüsselpunkte einer Kreuzung berechnet werden. Das bedeutet, dass Start- sowie Endpunkte eines Fahrmannövers bekannt sein sollen.

Szenario 1 Das Fahrzeug erkennt eine ungeregelte Kreuzung, bei der es keinen weiteren Verkehr gibt. Es ist ein grundsätzliches Szenario und soll als Test dienen, dass der Roboter die grundlegende Fähigkeit verfügt, eine Kreuzung zu erkennen und das entsprechende Fahrmannöver durchzuführen.

Szenario 2 Das Fahrzeug erkennt eine ungeregelte Kreuzung sowie ein weiteres Fahrzeug, welches sich direkt vor ihm befindet. Das andere Fahrzeug soll erkannt und ein entsprechender Sicherheitsabstand eingehalten werden. Wenn die Kreuzung frei ist, soll das Fahrzeug entsprechend 3.1 fortfahren.

Szenario 3 Das Fahrzeug erkennt eine ungeregelte Kreuzung, sowie ein weiteres Fahrzeug, welches aus der rechten Fahrtrichtung kommt. Entsprechend der Straßenverkehrsordnung soll hier die Regel "rechts vor links"beachtet werden. Das Fahrzeug soll deshalb warten, bis das andere Fahrzeug die Kreuzung verlassen hat.

Szenario 4 Das Fahrzeug erkennt eine ungeregelte Kreuzung, sowie ein weiteres Fahrzeug, welches aus der linken Fahrtrichtung kommt. Entgegen 3.1 soll das Fahrzeug hier also sein Vorfahrtsrecht beachten.

Szenario 5 Das Fahrzeug erkennt eine ungeregelte Kreuzung, sowie ein weiteres liegengebliebendes Fahrzeug, welches die gewünschte Fahrtrichtung blockiert. Das Fahrzeug soll die Fahrt in eine andere Richtung fortsetzen.

3.2 Voraussetzungen

Die Arbeit beruht wie bereits beschrieben auf dem AutoRace-Projekt. Somit ist die erste Voraussetzung, dass die Fahrbahnen unidirektional sind. Im Detail bedeutet dies, dass der Roboter anhand der Fahrbahnbegrenzung erkennen kann, in welche Richtungen abgebogen werden darf. Für die Detektion von Gegenverkehr bedeutet dies, dass lediglich aus Richtungen, in die nicht abgebogen werden darf, Gegenverkehr zu erwarten ist.

Zusätzlich wird angenommen, dass die Fahrbahnbegrenzung wie bereits beschrieben rechts weiß und links gelb ist. Darauf aufbauend wird definiert, dass eine Kreuzung nicht unmittelbar nach einer Kurve beginnt. Der Roboter muss die Chance haben, die Kreuzung zu erkennen. Da die Kamera erhöht verbaut ist, hat der Roboter ein totes Sichtfeld und hätte somit keine Möglichkeit eine Kreuzung visuell zu erkennen, wenn diese unmittelbar nach einer Kurve beginnt. Eine Möglichkeit, die durch das AutoRace-Projekt besteht, ist eine Erkennung einer Kreuzung durch Schilder. Ähnlich zur realen Welt hätten hier Vorfahrsschilder oder neu entworfene Schilder genutzt werden können. Durch den beschriebenen Objekterkennungsansatz mit z.B. SIFT und FLANN besteht hier die Möglichkeit, eine Kreuzung direkt zu erkennen. Dies würde dazu führen, dass ein Kreuzungsszenario sicher eingeleitet werden kann. Jedoch hat dieser Ansatz zwei wesentliche Nachteile. Zunächst fehlen dem Roboter sämtliche Informationen über die Kreuzung, wie z.B. die Richtungen in die er abbiegen darf oder aber die Schlüsselpunkte. Ein weiterer Nachteil ist, dass es zusätzlichen Aufwand bei einer potenziellen Nutzung in der Realität bedeutet. Wenn an jeder Kreuzung Schilder stehen müssen, büßt die Software an Flexibilität ein. Aus diesen Gründen gilt die Voraussetzung, dass eine Krezung ausschließlich durch die Fahrbahnmakierung identifiziert wird.

Zur Kommunikation zwischen den Robotern wird die Abwandlung von Connected Cars als gegeben angesehen. Jeder Roboter, der in den Szenarien eingesetzt wird, kann mit allen anderen Robotern kommunizieren. Das bedeutet, dass z.B. auch ein liegengebliebender Roboter stehts seine Daten übermittelt und hierüber identifiziert werden kann.

3.3 Versuchsaufbau

Um die einzelnen Szenarien in Gazebo zu simulieren, wurden neue Simulationswelten bereitgestellt. Grundsätzlich basieren diese Welten auf der Simulation des TurlteBot AutoRace 2017. Der wesentliche Unterschied ist, dass sämtliche Aufgaben entfallen. Das bedeutet, dass die Ampel, der Parkplatz, der Tunnel sowie die Zollschranke entfernt wurden. Grundsätzlich ist die Version des AutoRace, auf der der Versuchsaufbau basiert,

nicht relevant, da für die Arbeit lediglich die Fahrbahn benötigt wird. Diese hat sich in den letzten Jahren nicht verändert.

Insgesamt wurden drei neue Welten entworfen. Die erste ist in Abbildung 3.1 zu sehen. Hier sind verschiedene Kreuzungen dargestellt. Die Intention ist hier, dass der Roboter am roten Pfeil in der entsprechenden Richtung startet. Bei jeder Kreuzung soll er erkennen, welche Richtungen für ein Abbiegemannöver erlaubt sind. Wenn die Detektion erfolgreich funktioniert, soll der Roboter das entsprechende Mannöver für jede erlaubte Richtung in separaten Durchlaufen durchführen. Eine Alternative, um die Detektion zu prüfen, wäre den Roboter auf statischen Kameraaufnahmen arbeiten zu lassen. Diese Kameraaufnahmen hätten mit entsprechenden Informationen beschrieben werden müssen, welche Richtungen erlaubt sind oder an welchen Koordinaten sich die Schlüsselpunkte der Kreuzung befinden. Zweiteres ist direkt abhängig von der aktuellen Position des Roboters, was die Arbeit mit statischen Bildern erschwert. Ein zusätzliches Manko ist, dass sich das statische Bild nicht verändert. Wie später zu sehen sein wird, zeigt die Realität, dass sich selbst beim Stillstand des Roboters in der simulierten Welt das aufgenommene Bild minimal verändert. Dieses Verhalten ist auch für die reale Welt anzunehmen, da beispielsweise die Vibration durch verbaute Geräte am TurtleBot 3 ausreichen würde, um Varianzen in einer Bildsequenz zu haben.

In dieser Simulation sind noch keine anderen Roboter enthalten. In dieser Welt soll ausschließlich die Kernfunktionalität, die Detektion und das Durchführen der Abbiegemannöver, simuliert und erprobt werden. Die Implementierung ist an dieser Stelle noch unabhängig vom AutoRace-Projekt, weshalb dieser Versuch auf neuentwickelten Nodes beruht.

Die zweite Welt ist in Abbildung 3.2 zu sehen. Hier sollen die einzelnen Szenarien getrennt simuliert und erprobt werden. Der erste Versuch wird darin bestehen, die neuentwickelten Nodes sowie die bestehende AutoRace-Implementierung zusammenzuführen. Der Roboter soll also die Spur halten, bis er eine Kreuzung erkennt und entsprechend den Modus wechseln. In diesem Modus soll er entsprechende Fahrmannöver an der Kreuzung durchführen. Nachdem dies absolviert ist, soll der Roboter wieder in den entsprechenden Modus wechseln, um die Fahrspur zu halten.

Im nächsten Schritt sollen in dieser Welt mehrere Instanzen des TurtleBot 3 fahren. Hier soll das Verhalten entsprechend der Szenarien von den Robotern durchgeführt werden. Hier ist das Ziel, dass das Verhalten an einem rudimentären Beispiel getestet werden kann.

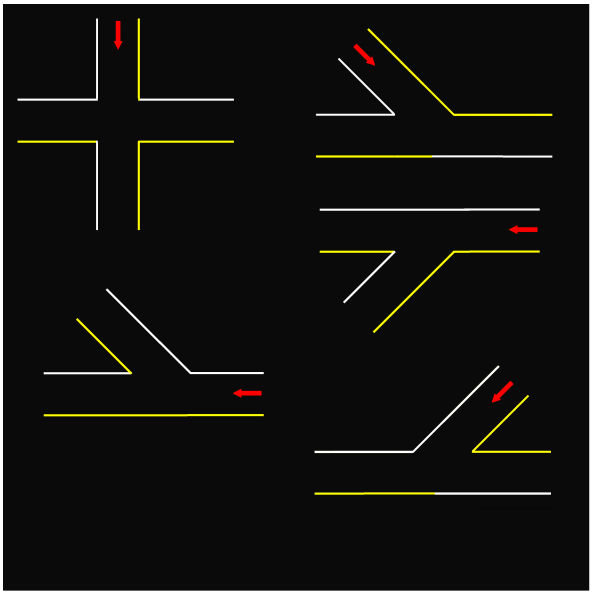


Abbildung 3.1: Karte der Testwelt

Der Hauptfokus in dieser Welt soll darauf liegen, dass die Robustheit des Roboters getestet werden kann. Wenn die Implementierung bereit ist, soll der Roboter für längere Zeit in der Welt fahren. Wenn die Software robust ist, dann wird der Roboter für die ganze Durchführungsdauer die Spur halten können und sich an den Kreuzungen richtig verhalten.

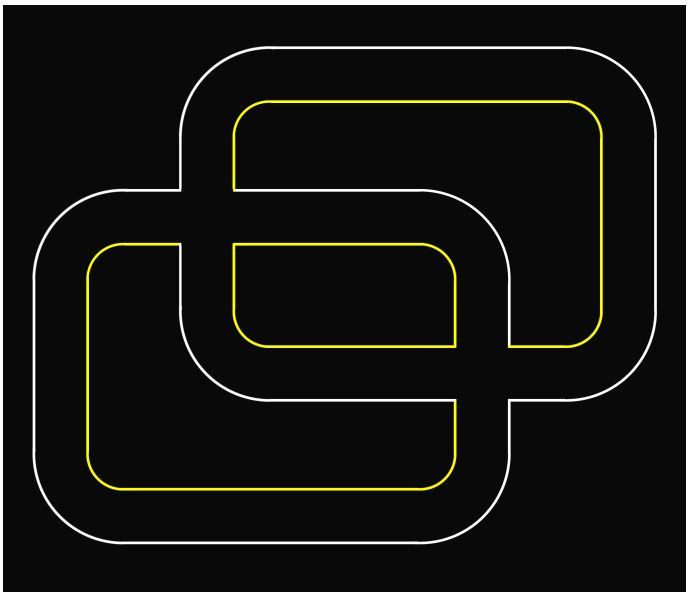


Abbildung 3.2: Karte der Testwelt mit Straßennetz

Die letzte Welt ist in Abbildung 3.3 zu sehen. In dieser Welt sind komplexere Kreuzungsszenarien mit aufgenommen. Auch hier ist das Ziel, die Robustheit der Software zu prüfen, indem mehrere Instanzen des TurtleBot 3 versuchen, durch das Straßennetz

zu navigieren ohne von der Spur abzukommen oder mit einem anderen TurtleBot zu kollidieren.

Die einzelnen Szenarien werden auch hier manuell nachgestellt und erprobt.

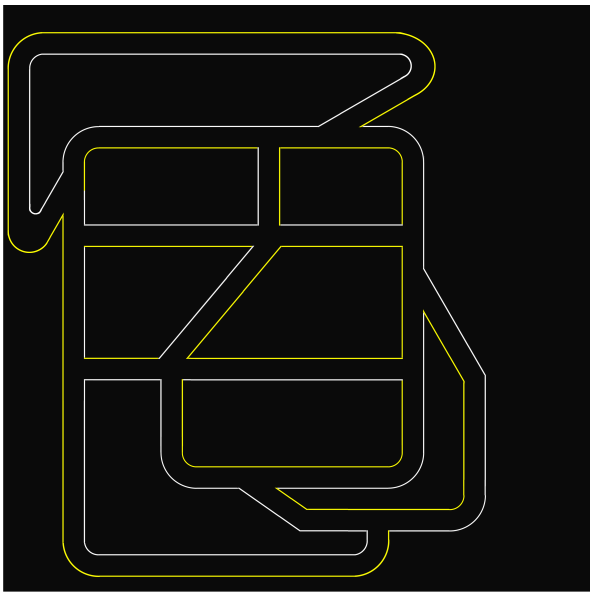


Abbildung 3.3: Karte der finalen Testwelt mit Straßennetz

3.4 Anforderungen

An dieser Stelle werden die Funktionalen Anforderungen (FA) beschrieben, welche die zu implementierenden Softwarekomponenten erfüllen müssen. Diese werden auf Basis des erläuterten FunktionsMASTeRs beschrieben. In der Tabelle 3.1 werden alle Funktionalen Anforderungen erfasst, die das System mit der neuen Funktionalität umsetzen soll. Die Anforderungen sind Grundlage für folgende Designentscheidungen und dienen als wesentliches Strukturelement für die Umsetzung.

Anforderungen die mit einem *D* beginnen, haben die Hauptaufgabe etwas zu detektieren. Diese Anforderungen sollten von Detection-Nodes umgesetzt werden. Die Anforderungen, die mit einem *C* beginnen, sollen den Roboter kontrollieren. Entsprechend werden diese Anforderungen von Control-Nodes umgesetzt.

Was in dieser Arbeit vernachlässigt wird, sind Nicht-Funktionale Anforderungen. Wie bereits erwähnt, beschreiben Nicht-Funktionale Anforderungen die Eigenschaften eines Systems. Viele dieser Eigenschaften sind bereits identifiziert worden, wie z.B. die Programmiersprache oder Frameworks. Grundsätzlich handelt es sich bei diesem System um keines, welches produktiv eingesetzt wird, wodurch eine Auseinandersetzung mit den Nicht-Funktionale Anforderungen des System nicht gerechtfertigt ist.

ID	Anforderung
D1	**Kreuzung erkennen** Selbsttätige Systemaktitivät Wenn sich der TurtleBot 3 einer Kreuzung nähert, muss das System diese erkennen. **Akzeptanzkriterien:** - das System stellt fest, dass es sich einer Kreuzung nähert und ändert den Modus
D2	**Richtungen identifizieren** Selbsttätige Systemaktitivät Wenn das System eine Kreuzung erkannt hat, muss das System die möglichen Fahrtrichtungen bestimmen. **Akzeptanzkriterien:** - das System erkennt die Fahrtrichtungen und weiß, welche Fahrtrichtungen für eine Weiterfahrt erlaubt sind
D3	**Schlüsselpunkte identifizieren** Selbsttätige Systemaktitivät Wenn das System eine Kreuzung erkannt und die Fahrtrichtungen identifiziert hat, muss das System die Schlüsselpunkte der Fahrtrichtungen bestimmen. **Akzeptanzkriterien:** - das System kann die Schlüsselpunkte einer Kreuzung berechnen
D4	**Gegenverkehr identifizieren** Selbsttätige Systemaktitivät Das System muss in der Lage sein, Gegenverkehr zu identifizieren. **Akzeptanzkriterien:** - das System überprüft zu jeder Zeit, ob aktuell Gegenverkehr herrscht - das System überprüft zu jeder Zeit, ob aktuell ein Fahrzeug im direkten Umfeld ist
C1	**Zur Kreuzung fahren** Selbsttätige Systemaktitivät Wenn das System die Schlüsselpunkte der Kreuzung berechnet hat, muss das System den Roboter zum Startpunkt der Kreuzung steuern. **Akzeptanzkriterien:** - der Roboter wird vom System mit einer minimalen Abweichung (unter 0.1 odom. Längeneinheit) zum Startpunkt gesteuert
C2	**Abbiegen** Selbsttätige Systemaktitivät Wenn der Roboter vom System zum Starpunkt gesteuert wurde und kein Gegenverkehr existiert, muss das System den Roboter in eine erlaubte Richtung abbiegen lassen. **Akzeptanzkriterien:** - der Roboter kann mit einer minimalen Abweichung (unter 0.1 odom. Längeneinheit) zum Ausgangspunkt einer Kreuzung gesteuert werden

Tabelle 3.1: Funktionale Anforderungen

3.5 Softwarekomponenten

Die zu implementierenden Softwarekomponenten sollen sich in das bestehende Projekt eingliedern. Dies vereinfacht die Lebenszyklen der zu implementierenden Nodes, da diese nach ähnlichen Programmabläufen gestartet oder beendet werden, wie es für die bestehenden Nodes der Fall ist. Die Core-Nodes werden deshalb erweitert und neue Control- sowie Detection-Nodes bereitgestellt.

3.5.1 Control-Nodes

Die Erweiterung der Core-Nodes beschränkt sich auf eine Erweiterung der Modi. Die Modi werden aktuell in einem Aufzählungstypen nachgehalten, *CurrenMode*. Dieser Aufzählungstyp wird durch den Wert *control_crossing* erweitert. Zusätzlich müssen die neuen Nodes abhängig vom Modus gestartet werden. Die neuen Nodes sollten in den Modi *lane_following* und *control_crossing* in jedem Fall aktiv sein.

3.5.2 Detection-Nodes

Insgesamt wird es zwei neue Detection-Nodes geben. Diese sind *detect_crossing* und *detect_traffic*. In Abbildung 3.4 ist das Aktivitätsdiagramm der Node *detect_crossing* dargestellt. Grundsätzlich soll diese Node die Anforderungen D1, D2 und D3 umsetzen. Auslöser dieser Aktivität ist ein eingehenden Bild der Kamera des TurlteBot 3. Auf diesem Kamerabild wird zunächst die Aktion "Kanten detektieren"durchgeführt. Ziel ist es, dass unwesentliche Bildelemente wie die Farbe oder Struktur der einzelnen Oberflächen aus dem Bild entfernt werden. Auf Basis des reduzierten Bildes können im Folgenden Schritt die Geraden berechnet werden, die die einzelnen Kanten repräsentieren. Entsprechend der Voraussetzungen kann davon ausgegangen werden, dass eine Kreuzung aus Geraden besteht, da ein Stück weit gerade Strecke vorhanden sein muss. Basierend auf dieser Prämisse, können aus den Kanten Geraden abgeleitet werden. Diese Geraden werden nun geclustert. Damit die relevanten Intersektionen berechnet werden können, müssen die Geraden in einer Form gruppiert werden. Alle Geraden, die eine Fahrbahnbegrenzung repräsentieren, sollten zusammen gruppiert werden. Im nächsten Schritt werden die Schnittpunkte und -winkel der Gruppen berechnet. Dies sind wichtige Informationen für den Roboter, um den Roboter gezielt abbiegen zu lassen. Weiterhin ist insbesondere der Schnittpunkt relevant, um die Schlüsselpunkte zu berechnen. Die Schlüsselpunkte sind hier für jede Richtung zu berechnen und bestehen aus dem Start-Punkt sowie dem End-Punkt für das Abbiegemannöver. Diese Schlüsselpunkte sind zunächst aus den Bildkoordinaten abgeleitet und beschreiben somit einen Punkt im konkreten Bild. Diese Koordinaten müssen im nächsten Schritt transformiert

werden, sodass sie den odometrischen Koordinaten der Welt entsprechen. Nachdem alle Kerninformationen berechnet wurden, muss bestimmt werden, welche Richtungen für eine Weiterfahrt erlaubt sind. Zusammenfassend werden die Daten noch gewichtet. Bei der Bildverarbeitung sind Abweichungen, auch wenn sie minimal sind, einzukalkulieren. Wie bereits beschrieben kann dieses Problem bereits durch leichte Vibrationen entstehen. Ein zusätzliches Risiko sind falsch erkannte Daten. Um einen guten Durchschnitt und falsche Erkenntnisse zu eliminieren, müssen ähnliche Resultate mit einer Wertung zusammengefasst werden. Diese Wertung beschreibt, wie wahrscheinlich es sich bei einem Resultat um eine Kreuzung handelt. Im letzten Schritt werden die Resultate an das Topic *detect/crossing/data* veröffentlicht.

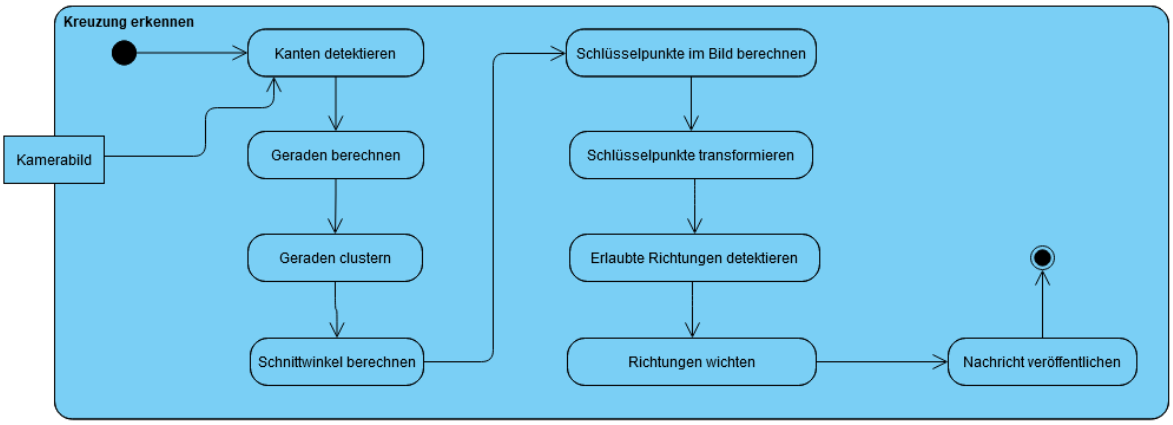


Abbildung 3.4: Akitivitätsdiagramm der Node *detect_crossing*

Die zu versendende Nachricht wird durch eine Liste des in 3.5 dargestellten ROS-Message-Formats repräsentiert. Diese Nachricht beinhaltet alle Informationen, die durch den im Aktivitätsdiagramm beschriebenen Prozess berechnet wurden. Zusätzlich sind hier die Informationen *angle_to_start* sowie *angle_at_start* als Felder der Nachricht deklariert. Diese Daten können ebenfalls aus den Schlüsselpunkten abgeleitet werden. Ziel hierbei ist zu wissen, in welche Richtung der Roboter fahren muss, damit der Startpunkt erreicht werden kann als auch wie der Roboter am Startpunkt ausgerichtet werden muss.

```
string direction
bool allowed
float32 angle_to_start
float32 angle_at_start
float32 start_x
float32 start_y
float32 angle_target
float32 angular_distance_target
```

```
float32 target_x
float32 target_y
```

Quelle: eigene Darstellung

Abbildung 3.5: Message-Format für die Kreuzungsdetektion *CrossingData*

Die Node *detect_traffic* wird zwei Aktivitäten zeitgleich ausführen und ist für die Anforderung D4 verantwortlich. Weiterhin ist zu bemerken, dass diese Node in jedem Modus aktiviert sein sollte. Die erste Aktivität ist in Abbildung 3.6 dargestellt. Auslöser dieses Programmablaufs ist eine erhaltene Nachricht der vom Roboter veröffentlichten odometrischen Daten. Diese Daten sollen die Informationsgrundlage für Connected Cars repräsentieren. Damit diese Daten für andere Roboter wiederverwendbar sind, ist es jedoch wichtig, dass diese Daten im Schritt "Daten taggen"gekennzeichnet werden. Als Kennzeichnung wird wie bereits beschrieben eine UUID verwendet. Diese wird beim Start dieser Node generiert und ist für jede Instanz der Roboter eindeutig. Über diese ID sind die Bewegungen aller Roboter für jeden einzelnen zuordenbar. Von der Idee ist diese Funktionalität in einer Detection-Node nicht richtig angesiedelt. Da jedoch hier auf Gegenverkehr und somit Kollisionen geprüft wird, ist es leichter, wenn die Daten auch in dieser Node produziert werden. Zusätzlich existiert in der bestehenden Implementierung kein Node-Typ, in den diese Verantwortung passen würde. Das Topic, an welches diese Daten veröffentlich werden, ist */detect/traffic/position*. Dieses Topic ist ein globales Topic. Das Nachrichtenformat ist in 3.7 dargestellt und beinhaltet lediglich die odometrischen Daten sowie die ID des Roboters.

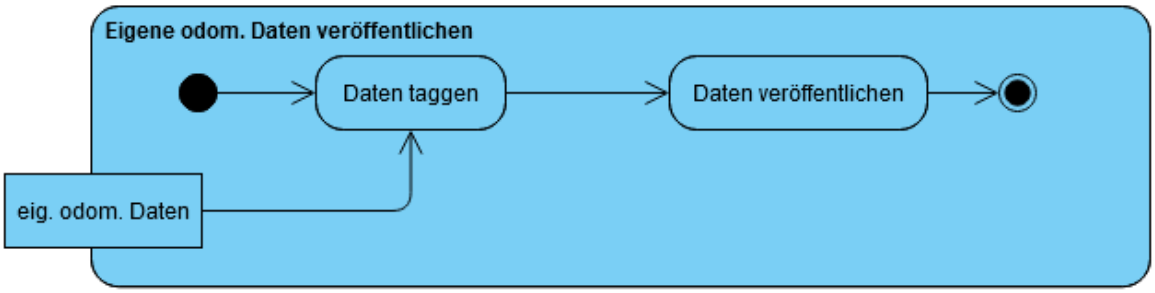


Abbildung 3.6: Akitivitätsdiagramm zum Taggen der Daten der Node *detect_traffic*

```
string vehicle_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

Quelle: eigene Darstellung

Abbildung 3.7: Message-Format für das Taggen der odom. Daten

Die konkrete Analyse des Verkehrs ist in Abbildung 3.8 zu sehen. Ausgangspunkt dieses Prozesses ist der Erhalt einer Nachricht mit getaggten odometrischen Daten eines

anderen TurtleBot 3. Zunächst werden anhand dieser Daten die Fahrbahnen berechnet. Hier wird angenommen, dass zu keinem Zeitpunkt eine angulare Beschleunigung von 0 existiert. Falls doch, soll diese durch einen Wert, der sich 0 annähert, substituiert werden. Unter diese Annahme gilt, dass sich die Roboter zu jedem Zeitpunkt auf einer Kreisbahn bewegen. Die einzelnen Fahrbahnen werden somit durch Kreise repräsentiert. Im zweiten Schritt werden die potenziellen Schnittpunkte dieser Kreise berechnet. Diese Schnittpunkte sind die Stellen, an denen beide Roboter kollidieren würden. Anschließend wird auf Basis der Schnittpunkte und der aktuellen Position auf dem Kreis die Distanz zwischen dem Roboter und der Kollisionspunkte berechnet. Falls hier eine Kollision möglich ist, soll eine Nachricht veröffentlicht und der Prozess beendet werden. Falls kein Schnittpunkt berechnet wurde, wird die Distanz zwischen dem aktuellen und anderem Roboter berechnet, ähnlich zu einem Radar-System. Wenn sich das andere Fahrzeug innerhalb einer von der Welt abhängigen Distanz befindet, soll geprüft werden, ob die beiden Fahrzeuge auf Basis ihrer aktuellen Ausrichtung aufeinander zusteuern. Wenn dies der Fall ist, wird ebenfalls eine Kollisions-Nachricht veröffentlicht und der Prozess beendet. Wenn sich die Roboter nicht auf Kollisionskurs befinden, jedoch nahe beieinander sind, soll eine Warn-Nachricht veröffentlicht werden. Grundsätzlich sind hier ledigliche Fahrzeuge interessant, die sich vor dem Roboter befinden.

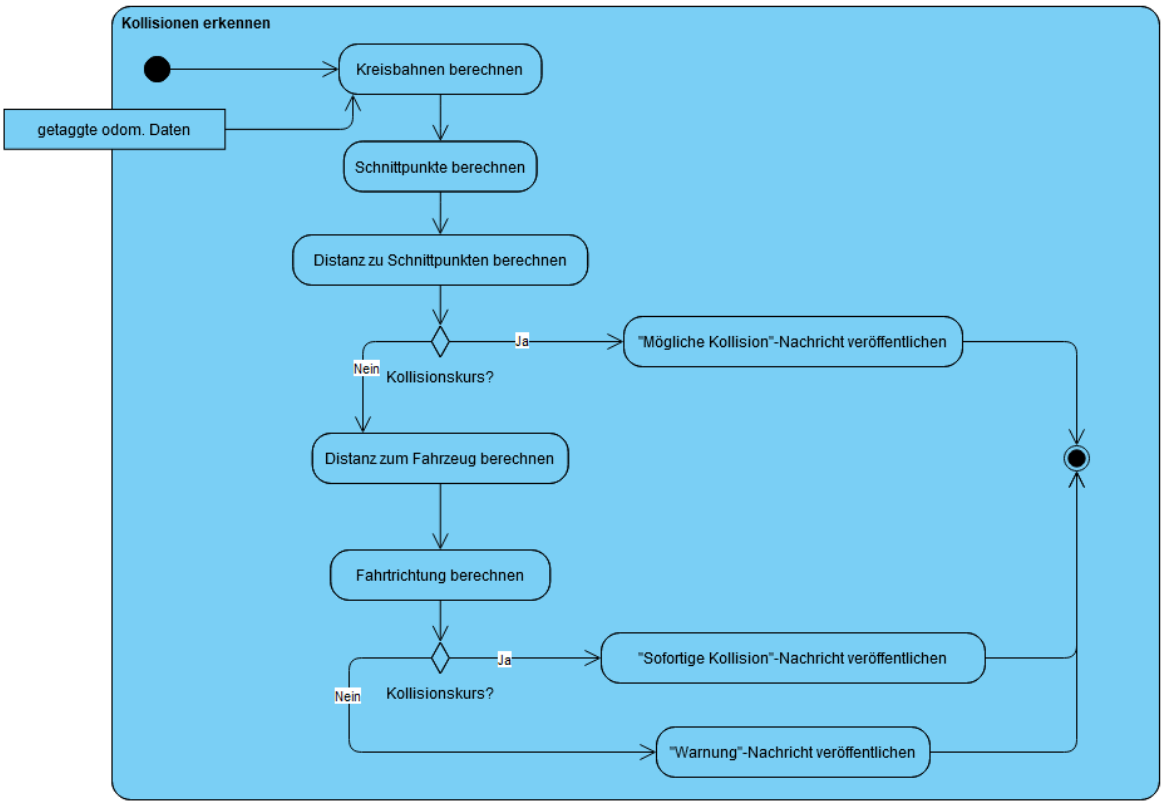


Abbildung 3.8: Akitivitätsdiagramm zur Kollisionserkennung der Node *detect_traffic*

In 3.9 wird das Nachrichtenformat für diesen Prozess definiert. Über das Feld *collision_type* wird identifizert, ob es sich um eine mögliche bzw. sofortige Kollision oder aber nur um eine Warnung handelt. Zusätzlich werden in dieser Nachricht die odometrischen Daten des eigenen Fahrzeugs als auch des geprüften Fahrzeugs veröffentlicht. Im Falle, dass es zu einer Kollision kommt, wird in dem Feld *time_till_collision* die Zeit und in *distance_till_collision* die Distanz bis zur Kollision übermittelt. Das Topic für diese Nachricht ist *detect/traffic/collision*.

```
int32 collision_type
geometry_msgs/PoseWithCovariance other_vehicle_pose
geometry_msgs/TwistWithCovariance other_vehicle_twist
geometry_msgs/PoseWithCovariance current_vehicle_pose
geometry_msgs/TwistWithCovariance current_vehicle_twist
int32 time_till_collision
float32 distance_till_collision
```

Quelle: eigene Darstellung

Abbildung 3.9: Message-Format für eine Kollision

3.5.3 Control-Nodes

Zusätzlich werden zwei Control-Nodes implementiert. Die erste Control-Node ist *control_traffic*, die zweite *control_crossing*. Ähnlich zu den Detection-Nodes sollte auch hier die Node *control_traffic* unabhängig vom Modus generell aktiv sein. Die konkrete Aufgabe dieser Node ist es, auf potenziellen Gegenverkehr zu reagieren und Kollisionen zu vermeiden. Zusätzlich hat diese Node die Verantwortung, Verkehrsregeln zu beachten und aufzulösen. Im Falle, dass sich zwei Roboter entsprechend 3.1 auf Kollisionkurs an einer Kreuzung befinden, soll diese Node die Situation durch die konkrete Verkehrsregel auflösen. In Abbildung 3.10 ist der Prozessablauf dieser Node abgebildet. Auslöser dieses Prozesses ist eine Nachricht der Node *detect_traffic*. Auf Grundlage dieser Daten wird zunächst ermittelt, ob es sich überhaupt um eine zu behandelnde Kollisionsnachricht handelt. Wenn dies der Fall ist, wird geschaut, wie dringend der Handlungsbedarf ist. Wenn die Roboter nah beieinander sind und aufeinander zusteuern, wird die Aktivität "Drossel max. Geschwindigkeitäusgeführt. Hierbei wird die maximale Geschwindigkeit auf 0 begrenzt, um den Roboter auf der Stelle anzuhalten. Diese Zustand wird beibehalten, bis keine Kollisionsgefahr mehr besteht und die Weiterfahrt frei fortgesetzt werden kann. Ähnlich ist es, wenn eine Warnung besteht, weil sich beispielsweise ein anderer Roboter vor dem gesteuerten Roboter befindet, der in die selbe Richtung fährt. Wenn der andere Roboter langsamer fährt, würde es auch hier zu einer

Kollision kommen. Entsprechend eines Sicherheitsabstands zwischen den Robotern sowie der Geschwindigkeit des anderen Roboters, soll eine neue Geschwindigkeit für den gesteuerten Roboter berechnet werden, sodass dieser nicht auffährt. Sobald auch hier eine freie Weiterfahrt möglich ist, da der andere Roboter beispielsweise abgebogen ist, soll die Geschwindigkeitsbegrenzung des gesteuerten Roboters aufgehoben werden.

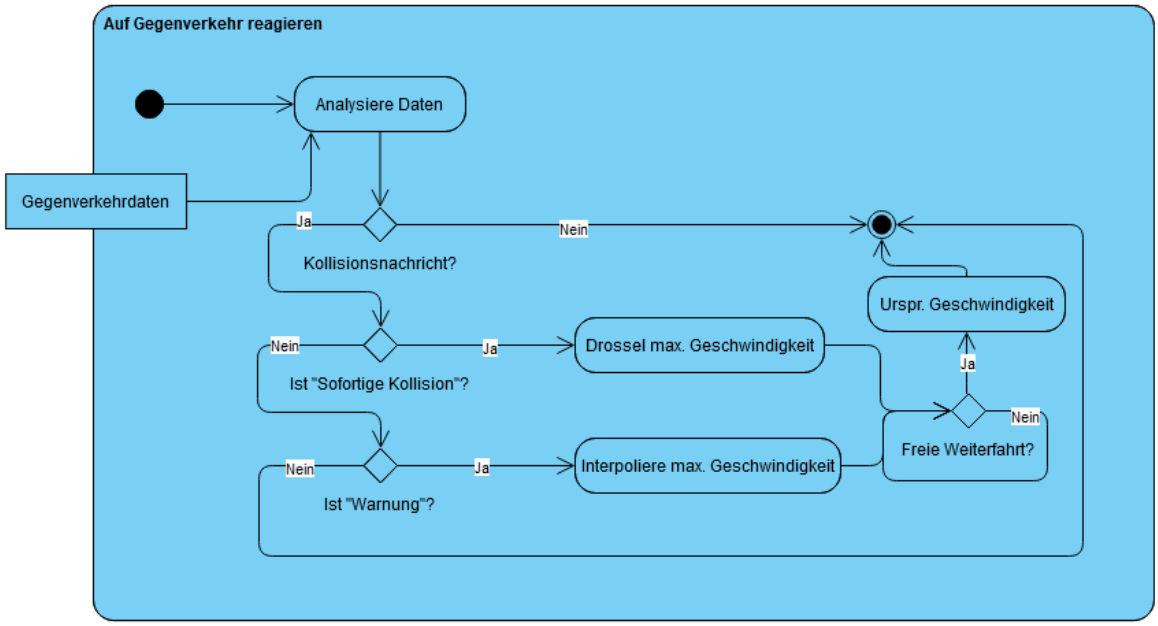


Abbildung 3.10: Akitivitätsdiagramm zur Kollisionssteuerung der Node *control_traffic*

Der Prozess der Node Node *control_crossing* ist in 3.11 abgebildet. Auslöser dieses Prozesses sind Nachrichten aus der Node *detect_crossing*. Die Hauptaufgabe dieser Node ist, den Roboter an einer Kreuzung zu steuern und den Roboter in eine Richtung abbiegen zu lassen. Dieser Prozess lässt sich in drei Schritte aufteilen. Wenn eine Kreuzung mit ihren Schlüsselpunkten erkannt wurde, soll der Robter zunächst an den Startpunkt der Kreuzung fahren. Wenn der Roboter diesen Punkt erreich hat, wird er gestoppt und es wird auf Gegenverkehr geprüft. Grundsätzlich hat die Node *control_traffic* diese Aufgabe. Wenn jedoch Gegenverkehr detektiert wurde und es sich dabei beispielsweise entsprechend 3.1) um einen Roboter handelt, der eine Richtung durch stillliegen blockiert, sollte der Roboter eine alternative Route fahren. Sobald kein Gegenverkehr mehr vorhanden ist, kann der Roboter mit dem konkreten Abbiegevorgang fortfahren. Hier soll der Roboter vom Startschlüsselpunkt zum Zielschlüsselpunkt fahren.

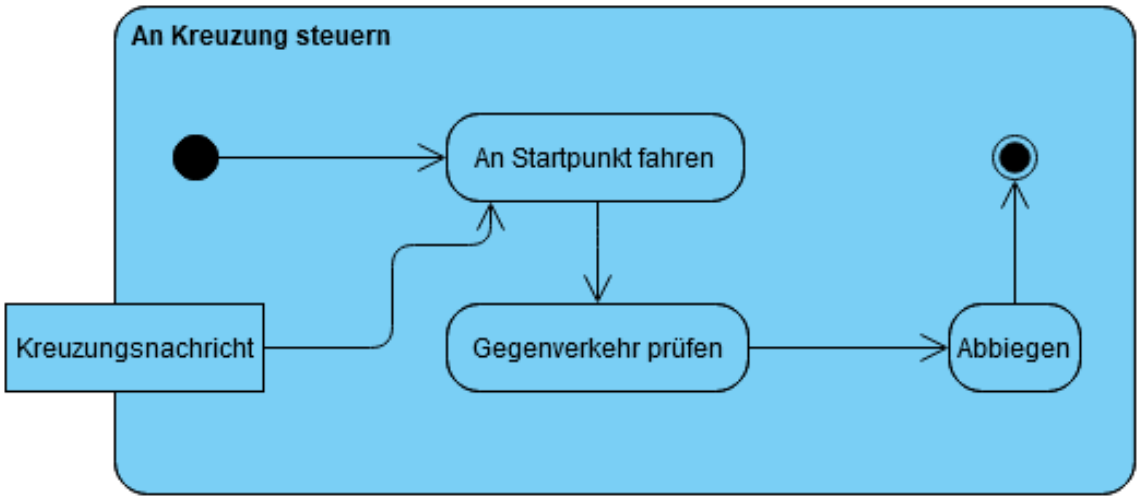


Abbildung 3.11: Akitivitätsdiagramm der Node *control_crossing*

4 Umsetzung

4.1 Umstrukturierung

Der erste Bestandteil der Umsetzung ist es, das bestehende AutoRace-Projekt geringfügig umzustrukturieren. Im Wesentlichen sind hier zwei Komponenten relevant: die Entfernung des irrelevanten Quellcodes sowie die Umstrukturierung des Lebenszyklus-Managements der einzelnen Nodes.

Im bestehenden Projekt gibt es viel Programmcode, der für die Bearbeitung der Thesis grundlos ausgeführt wird. Im Konkreten handelt es sich hierbei um die Detektion von Ampeln und Verkehrsschildern. Solche Infrastuktur-Elemente gibt es in den Simulationswelten der Thesis nicht, wodurch unnötige Rechenleistung beansprucht wird. Zusätzlich besteht die Gefahr, dass diese Nodes für ein unvorhersehbares Verhalten sorgen, indem durch falsche Detektionen Daten veröffentlicht werden die zu einer Steuerung des Roboters führen. Aus diesem Grund werden die entsprechenden Komponenten entfernt. Zusätzlich werden die Core-Nodes soweit angepasst, dass sämtliche Fallunterscheidung bezüglich des Modus auf das Wesentliche reduziert werden - das Halten der Fahrbahn und das Steuern an der Kreuzung.

Das Lebenszyklus-Management der Nodes sieht aktuell vor, dass diese dynamisch gestartet und bei Bedarf komplett beendet werden. Dies hat zufolge, dass der Roboter bei der Transition zwischen den einzelnen Modi warten muss, bis die Nodes wieder gestartet sind. Das ist ein unschönes Verhalten. In der Theorie gibt es für ROS ein Lebenszyklus-Management.[1] Das bestehende Projekt setzt auf die *roslaunch*-API, um Nodes dynamisch zu starten und zu beenden. Für diese wurde die Funktionalität nicht umgesetzt oder dokumentiert. Aus diesem Grund wird hier für jede Node ein neues Topic eingeführt. Über diese Topics wird mit Boolean-Nachrichten kommuniziert. Für die Node *control_crossing* wird beispielsweise das Topic */control_crossing/active* implementiert. Wenn hier beispielsweise ***TRUE*** empfangen wird, werden alle Subscriber der Node registiert. Wenn jedoch ein ***FALSE*** empfangen wird, werden alle Subscriber der Node abgemeldet. Hierdurch kann der Roboter fast nahtlos den Modus wechseln. Ein Nachteil dieses Vorgehens ist, dass so alle Instanzen einer Node in einem Namespace simultan

[1] `https://design.ros2.org/articles/node_lifecycle.html`

deaktiviert werden. Da für diese Thesis jedoch jede Node innerhalb ihres Namespaces einzigartig ist, ist dieser Sachverhalt unerheblich.

5 Ergebnisse

6 Fazit

Literatur

[1] *Morphologische Bildverarbeitung*, https://de.wikipedia.org/wiki/Mathematische_Morphologie, Stand: 08.08.2021.

[2] *CEDR Call 2014 DoRN Mobility & ITS*, http://www.bast.de/DE/BASt/Forschung/Forschungsfoerderung/Downloads/cedr_call_2014_2.pdf?__blob=publicationFile&v=2, Stand: 09.04.2021.

[3] *Mercedes-Benz Future-Truck 2025*, https://www.mercedes-benz.com/de/innovation/autonomous/selbststaendig-unterwegs-der-fern-lkw-der-zukunft/, Stand: 09.04.2021.

[4] *Driverless cars: Kangaroos throwing off animal detection software*, https://www.abc.net.au/news/2017-06-24/driverless-cars-in-australia-face-challenge-of-roo-problem/8574816, Stand: 10.04.2021.

[5] J. Baber, J. Kolodko, T. Noel, M. Parent und L. Vlacic, „Cooperative Autonomous Driving: Intelligent Vehicles Sharing City Roads“, Griffith University, Techn. Ber., 2005.

[6] Q. Zhu, L. Chen, Q. Li, M. Li, A. Nüchter und J. Wang, „3D LIDAR Point Cloud based Intersection Recognition for Autonomous Driving“, IEEE, Techn. Ber., 2012.

[7] *ROS*, https://www.ros.org/, Stand: 06.08.2021.

[8] *Gazebo*, http://gazebosim.org/, Stand: 06.08.2021.

[9] *Hands-on With TurtleBot 3, a Powerful Little Robot for Learning ROS*, https://spectrum.ieee.org/review-robotis-turtlebot-3, Stand: 06.08.2021.

[10] I. Sobel, „An Isotropic 3x3 Image Gradient Operator“, HP Labs Computer Vision & Graphics, Techn. Ber., 2014.

[11] T. Jochem, D. Pomerleau und C. Thorpe, „Vision-Based Neural Network Road and Intersection Detection and Traversal“, Carnegie Mellon University, Techn. Ber., 1995.

[12] IEEE, „Std 610.12-1990 (R2002, IEEE Standard Glossary of Software Engineering Terminology“, IEEE, Techn. Ber., 1990.

[13] C. Rupp, *Requirements-Engineering und -Management: Aus der Praxis von klassisch bis agil*. Wien: Carl Hanser Verlag GmbH & Co. KG, 2014.

[14] ISO/IEC, „ISO/IEC 25010 System and software quality models“, Techn. Ber., 2010.

Ehrenwörtliche Erklärung

Ich versichere, dass ich die vorliegende Masterarbeit mit dem Thema „Simulation und Erprobung verschiedener Szenarien des Autonomen Fahrens am TurtleBot3“ selbständig verfasst habe. Die Stellen, die anderen Werken wörtlich oder sinngemäß entnommen wurden, sind durch die Angabe der Quelle gekennzeichnet. Des Weiteren versichere ich, dass die vorliegende Masterthesis bei keiner anderen Prüfung vorgelegt wurde.

Rostock, den 17. Mai 2021

.....

Moritz Wilke