Freie Universität Berlin

Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Arbeitsgruppe Robotik

Untersuchung der Effizienz von RRT* bei autonomen Autos

Bernd Sahre
Matrikelnummer: 4866892
besahre@zedat.fu-berlin.de

Betreuer:	Prof. Dr. Daniel Göhring
Eingereicht bei:	Prof. Dr. Daniel Göhring
Zweitgutachter:	Prof. Dr. Raul Rojas

Berlin, 1. Mai 2018

Zusammenfassung

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

1. Mai 2018

Bernd Sahre

Inhaltsverzeichnis

1 Einleitung

Schon die Griechen der Antike träumten von autonomen, selbstfahrenden Fahrzeugen, wie in Homers Ilias geschildert. Mittlerweile ist die Forschung in diesen Bereichen so weit fortgeschritten, dass dieser Traum schon bald Wirklichkeit werden könnte. Zuvor müssen jedoch etliche Herausforderungen bewältigt werden. Eine davon ist, das Auto sicher durch den Straßenverkehr zu bringen. Dabei heißt sicher, dass das Auto während der Fahrt weder sich noch andere Verkehrsteilnehmer gefährdet. Neben der Sicherheit ist jedoch auch das möglichst schnelle Erreichen des Ziels wichtig, bei dem das Auto seinen Weg durch eine Umgebung mit statischen und dynamischen, das heißt sich bewegenden Hindernissen finden muss. Dies wird durch einen Pfadplaner gewährleistet, der -informell gesprochen - aus der eigenen Position, dem Zielbereich und unter Berücksichtigung aller statischen und sich bewegenden Hindernissen einen sicheren Pfad zum Ziel findet. Dieser Pfad ist dann durch das Auto abfahrbar.
Um diese Aufgabe zu meistern, existieren unterschiedliche Ansätze, um dem Auto je nach Anwendungsfall bei gegebenen Ziel eine *Trajektorie* vorzuschlagen. Die jeweiligen Algorithmen unterscheiden sich in Ausführungszeit, Genauigkeit, Sicherheit und berechnen unterschiedlich optimale Pfade.
Das Dahlem Center for Machine Learning and Robotics untersucht maschinelles Lernen und Anwendungen intelligenter Systeme. Dazu haben sich vier Arbeitsgruppen der Freien Universität Berlin zusammengeschlossen:

- Intelligent Systems and Robotics (Prof. Dr. Raúl Rojas)
- Autonomos Cars (Prof. Dr. Daniel Göhring)
- Artifical and Collective Intelligence (Prof. Dr. Tim Landgraf)
- Logic and automatic proofs (Christoph Benzmüller)

Ein Forschungsgebiet ist die Entwicklung und Analyse autonomer Autos. Zur oben beschriebenen Pfadplanung wird in der Arbeitsgruppe hauptsächlich das Prinzip elastischer Bänder (Time-Elastic-Bands, [vgl. 2]) zur Erzeugung abfahrbarer *Trajektorien* benutzt. Doch auch die Untersuchung und Analyse anderer Algorithmen ist interessant, um zu überprüfen, ob sich eine vertiefte Forschung in diesen Bereichen lohnt.
Diese Bachelorarbeit untersucht einen dieser anderen Algorithmen zur Pfadplanung, *RRT**, auf seine Tauglichkeit, über eine vorgegebene Fahrbahn mit Hindernissen eine abfahrbare, sichere und möglichst optimale *Trajektorie* zu finden.

1.1 Aufbau und Struktur

[TODO Querverweise] Nach einer kurzen Hinführung zum Thema werden zuallererst die Grundlagen (2) besprochen, um das Verständnis der nachfolgenden Kapitel zu erleichtern. Danach wird die Umsetzung des Algorithmus auf das Auto sowie die technischen Details und Hintergründe beschrieben (3). Im vierten Kapitel (4) werden die Ergebnisse der Testfahrten vorgestellt und bewertet. Zum Schluss werden in einem Fazit alle Schlussfolgerungen nochmals zusammengefasst und ein Ausblick auf

weitere mögliche Forschungsmöglichkeiten gegeben.

1.1.1 Problemanalyse

Diese Arbeit untersucht die Effizienz des *RRT**-Algorithmus unter Anwendung bei autonomen Autos. Dazu fährt das Auto eine vorgegebene Strecke ab [TODO Bild einfügen]. Auf dieser Strecke werden erst statische, dann sich bewegende Hindernisse platziert.
Das Auto soll diese Strecke in möglichst kurzer Zeit mit einem möglichst optimalen Pfad abfahren, ohne eines dieser Hindernisse zu berühren. Dazu sollte das Auto den Algorithmus idealerweise 30 Mal pro Sekunde ausführen, mindestens aber vier Mal pro Sekunde.
Die Effizienz, Sicherheit und Effektivität des Algorithmus wird bei verschiedenen Eingabeparametern untersucht und bewertet.

1.1.2 Anmerkung zur Gestaltung der Arbeit

[TODO Plagiat mit Prof abklären (ist ne 1:1 Kopie)] Für die im Folgenden verwendeten personenbezogenen Ausdrücke wurde, um die Lesbarkeit der Arbeit zu erhöhen, ausschließlich die männliche Schreibweise gewählt. Desweiteren werden eine Reihe von englischen Bezeichnungen und Fachwörtern verwendet, um einerseits dem interessierten Leser das Studium der fast ausschließlich englischen Fachliteratur zu erleichtern und andererseits bestehende Fachbegriffe nicht durch die Übersetzung zu verfälschen. Diese Begriffe wurden im Gegensatz zum restlichen Text in kursiver Schrift formatiert.

1.1.3 Glossar

Begriff	Erklärung
Trajektorie	Die Strecke, die dem Low-Level-Planer des Autos übergeben wird
Low-Level-Planer	Steuert direkt die Motoren des Autos, Lenkung und Antrieb, um eine vorgegebene Trajektorie möglichst genau abzufahren.
RRT	Rapidly-Exploring Random Tree [6], ein Algorithmus zur Findung eines Pfades zum Ziel durch unbekanntes Gelände, wird in Kapitel 2.1 noch weiter erläutert
RRT*	Eine verbesserte Variante des RRT, bei dem die Pfade optimiert werden. Asymptoptisch optimal, erläutert in Kapitel 2.2
ROS	Robot Operating Systems, eine Open-Source Sammlung an Software-Bibliotheken und Werkzeugen zur Kreation von Anwendungen zur Robotik.
Lidar	Light detection and ranging; Radarscanner am Auto, für Abstandsmessung zu Hindernissen.
Nonholonomic Robots	Roboter, die gewissen kinematischen Einschränkungen unterworfen sind. Ein Auto zum Beispiel kann sich nicht in jede beliebige Richtung bewegen, sondern ist z.B. durch den maximalen Lenkwinkel und den Wenderadius eingeschränkt und kann nicht jeden beliebigen Punkt sofort, mit nur einem Schritt, erreichen.
Dynamische Umgebung	Eine Umgebung mit dynamischen, also sich bewegenden oder veränderlichen Hindernissen.
Kinodynamic planning	Beschreibt eine Klasse von Problemen bei der physikalische Einschränkungen wie Geschwindigkeit, Beschleunigung und Kräfte zusammen mit kinematischen Einschränkungen (Hindernisvermeidung) berücksichtigt werden müssen.
hochdimensionale Probleme	[TODO]
randomisierte Algorithmen	[TODO]
Kinematik	
Odroid	
Arduino Nano	
Gyroskop	
Odometrie	
Remote Procedure Call	
Rewiring	Neuverknüpfung eines RRT*-Baumes. Die Begriffe Rewiring und Neuverknüpfung werden synomym gebraucht. Nach Hinzufügung eines Knotens K zum Baum werden Nachbarknoten überprüft, ob diese durch K besser erreichbar sind als vorher. Falls ja, wird K als Vaterknoten ausgewählt.
k-nearest neighbour	
radius k-nearest neighbour	

1.1.4 Wichtige Quellen und deren Beitrag zum Thema

Es existieren[TODO RRT-Quellen [vgl 6], RRT*-Quellen, Übersichtsliteratur]
Nun werden wir uns den wissenschaftlichen Grundlagen der Arbeit widmen.

2 Grundlagen

Dieses Kapitel führt die verwendeten Algorithmen und Berechnungen ein.
Mit dem A*-Algorithmus wurde schon in den 60er Jahren ein Werkzeug für die Pfadplannung von Robotern eingeführt[TODO Literaturverweis]. Pfadplannung bedeutet hier, dass ein Roboter mit festgelegten kinematischen Einschränkungen in einer bestimmten, definierten Umgebung von einem Startzustand zu einem Zielzustand mithilfe von Steuerungseingaben gelangen kann, ohne die physikalischen Gesetze und die der Umgebung (keine Kollision mit Hindernissen) zu verletzen. Der A*-Algorithmus löst dieses Problem mit vielen Einschränkungen, indem er in einem Graphen den kürzesten Weg zwischen zwei Knoten findet. Allerdings benötigt A* diesen Graphen zur Berechnung des Weges und ist aufgrund des hohen Speicherplatzbedürfnisses für *hochdimensionale Probleme*, d.h. für Probleme mit den oben genannten Einschränkungen, nicht geeignet[TODO Zitat].
In nachfolgender Zeit wurden *randomisierte Algorithmen* [TODO weitere Erklärungen zu randomisierten Alg und deren Vorteile] entwickelt, die diese Probleme nicht mehr hatten, wie der *randomized potential field* Algorithmus [5] und der *probalistic roadmap* Algorithmus [1]. Doch auch diese waren nicht allgemein auf *nonholonomic Robots* anwendbar und lösten oft nur spezifische Probleme unter ganz bestimmten Bedingungen. Der Erfolg des *randomized potential field* Algorithmus beispielsweise hing stark von der Wahl einer passenden Heuristik ab [vgl. Kap 3.4 5]. Während sich bei einfachen Ausgangsbedingungen die Heuristik noch einfach finden lies, wurde dies bei komplexen, dynamischen Umgebungen mit Hindernissen, physikalischen und kinematischen Bedingungen zu einer großen Herausforderung.
1998 schließlich führte Steven LaValle den *RRT*-Algorithmus ein, der die oben genannten Einschränkungen umgehen sollte.

2.1 RRT

LaValle erkannte die sowohl die Vorteile von randomisierten Algorithmen als auch die Nachteile der existierenden Algorithmen [vlg. Kap 1 6]. Insbesondere störten ihn die fehlende Skalierbarkeit vieler Algorithmen in komplexere Umgebungen, da diese Algorithmen damit nur unter gewissen Vorbedingungen effizient einsetzbar waren. Der *probalistic roadmap* Algorithmus [1] beispielsweise [TODO] .
Bei der Entwicklung von *RRT* wurde deshalb viel Wert auf Einfachheit, Allgemeingültigkeit und damit auf Skalierbarkeit gelegt [vlg. Kap 3 6]. Bevor wir jedoch genauer auf die Vorteile des Algorithmus eingehen und warum dieser hier gewählt wurde, folgt jetzt erstmal eine kurze Erklärung der Funktionsweise. RRT baut einen Baum auf, indem zufällig gewählte Punkte unter Berücksichtigung einer Metrik verbunden werden. Der Algorithmus mit dem *RRT* T mit den Eingabeparametern Größe K, Metrik M, Bewegungsfunktion u, und Startzustand `x_init` funktioniert folgendermaßen:

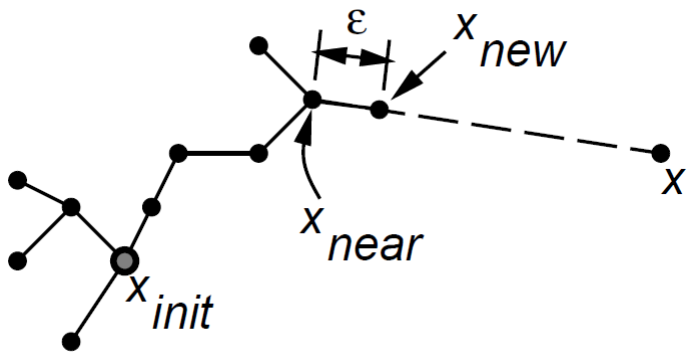

Abbildung 1: Die EXTEND Funktion [**?**]

2.1.1 Funktionsweise RRT

[TODO in Quellcode Literaturverweis]

```
1 BUILD_RRT(K, M, u, x_init)
2   T.init (x_init)
3   for k=1 to K do
4     x_rand = RANDOM_STATE();
5     EXTEND(T, x_rand);
6   Return T;
```

```
1 EXTEND(T, x)
2   x_near = NEAREST_NEIGHBOR(x, T, M);
3   x_new = project(x, x_near, u);
4   if (Collisionfree(x_new, x_near, u) then
5     T.add_vertex(x_new);
6     T.add_egde(x_near, x_new, u_new);
7     Return Extended;
8   else
9     Return Trapped;
```

Der Baum wird anfangs mit dem Startzustand `x_init` initialisiert. Anschließend wird in K Iterationen der Baum T aufgebaut, indem mit `x_rand` ein zufälliger Punkt ausgewählt und mit `EXTEND(T, x_rand)` dem Baum hinzugefügt wird.
Die Funktion `EXTEND(T, x)` ermittelt zunächst mithilfe der Metrik M den nächsten Nachbar von x. Diese Metrik kann von einer einfachen euklidischen Distanz bis hin zur komplexen Einberechnung verschiedener kinetmatischer Bedingungen alles beinhalten. Die hier verwendete Metrik wird später noch in Kapitel 3.5 beleuchtet.
Ist der nächste Nachbar `x_near` gefunden, wird von diesem aus mit `project(x, x_near, u)` ein Schritt der Länge ϵ in Richtung x durchgeführt und an dieser Stelle der neue Knoten `x_new` erzeugt.
Nun wird mit `Collsionfree(x_new, x_near, u)` überprüft, ob `x_new` oder die Bewegung u zu `x_new` hin mit Hindernissen kollidiert oder diesen zu Nahe kommt. Falls dies nicht der Fall ist, werden sowohl der neu entstandene Knoten `x_new` als auch die Kante von `x_near` zu `x_new` dem Baum T hinzugefügt.
Falls `x_new` oder die Bewegung u zu `x_new` mit Hindernissen kollidiert oder diesen

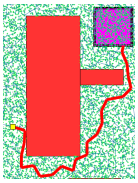

Abbildung 2: Ausschnitt eines RRT Graphs mit 20.000 Knoten [**?**]

zu Nahe kommt, wird der Knoten `x_new` verworfen und die Funktion `EXTEND(T,x)` beendet.

2.1.2 Vor- und Nachteile von RRT

Die Rapidly-Exploring Random Trees haben einige Eigenschaften, die für Bewegungsplanung von Robotern von großem Vorteil sind, wie LaValle in [TODO Literarur LaValle Kap 3] schreibt:

1. Ein *RRT* breitet sich sehr schnell in unerforschte Bereiche des Statusraums aus. Dadurch können Pfade schnell gefunden werden und es wird schnell eine mögliche (wenn auch nicht optimale) Lösung gefunden.

2. Die Verteilung der Knoten im Baum entspricht der Verteilung, wie diese Knoten erzeugt wurden; dies führt zu konsistentem Verhalten. Unter anderem kann dadurch das Wachstum des Baumes in eine bestimmte Richtung gesteuert werden (z.B. zum Ziel hin)

3. Ein *RRT* ist probabilistisch vollständig, das heißt mit zunehmender Laufzeit konvergiert die Wahrscheinlichkeit, keinen Pfad zum Ziel zu finden, gegen null

4. Ein *RRT* ist sowohl einfach zu implementieren als auch einfach in der Analyse, was es ermöglicht die Performance einfach zu analysieren und zu verbessern

5. Ein *RRT* ist immer mit sich selbst verbunden, und das bei einer minimalen Kantenanzahl

6. Ein *RRT* kann als Pfadplanungsmodul interpretiert werden, was die Kombination mit anderen Werkzeugen zur Bewegungsplanung möglich macht

Leider existieren neben den oben genannten Vorteilen auch etliche Nachteile. Eines der größten ist, dass ein *RRT* nicht den optimalen Pfad zurückliefert, da einmal gesetzte Knoten ihren Vaterknoten nicht mehr ändern können. Dadurch kann, auch wenn eine bessere Knotenfolge vom Start zum Ziel bestehen würde, diese nicht ausgewählt werden.
Deshalb wurde von Sertac Karaman und Emilio Frazzoli aufbauend auf *RRTs* der Algorithmus *RRT** eingeführt, welcher diesen Nachteil ausgleicht.

2.2 RRT*

Im Gegensatz zu einem *RRT* führt ein *RRT** die zwei folgenden Neuerungen ein:

1. Auswahl eines passenden Vaterknotens bei Hinzufügen des Knotens zum Baum
2. Neuverknüpfung des Baumes

Diese Neuerungen resultieren in einer veränderten `EXTEND(T,x)` Funktion, siehe [**?**].

2.2.1 Funktionsweise RRT*

```
1  EXTEND(T,x)
2    x_nearest = NEAREST_NEIGHBORS(x, T, M);
3    x_new = project(x, x_nearest, u);
4    if (Collisionfree(x_new, x_near, u) then
5      T.add_vertex(x_new);
6      x_min= x_nearest;
7      c_min = x_nearest.cost + cost(x_nearest, x_new);
8      X_NEAR = NEAR_NEIGHBORS(x, T, r);
9      foreach x_near in X_NEAR do
10       if Collisionfree(x_near, x_new) && x_near.cost + cost(
             x_near, x_new) < c_min then
11         x_min = x_near;
12         c_min = x_near.cost + cost(x_near, x_new);
13     T.add_egde(x_min, x_new);
14     foreach x_near in X_NEAR do
15       if Collsionfree(x_new, x_near) && x_new.cost + cost(x_new
             , x_near) < x_near.cost then
16         T.del_edge(x_near_parent, x_near);
17         T.add_edge(x_new, x_near);
18     Return Extended;
19   else
20     Return Trapped;
```

Während wie beim Erstellen eines *RRT* auch bei *RRT** zuerst der nächste Nachbar als Vaterknoten festgelegt wird, folgt daraufhin eine Speicherung der nächsten Nachbarn von `x_new` in einem gewissen Radius r in der Liste `X_NEAR`. Es wird `x_min` als der Abstand zum nächsten Knoten gesetzt. Die Funktion `x_nearest.cost` liefert die Kosten, um vom Startknoten zu `x_nearest` zu kommen, zurück, während die Funktion `cost(x_nearest, x_new` die Kosten des Pfades von `x_nearest` zu `x_new` berechnet. Als vorläufiger Startwert beinhaltet `c_min` demnach die Kosten, um vom Startknoten aus zu `x_new` zu kommen.
Die Liste `X_NEAR` wird nun durchiteriert, um den besten Nachbarn, also den mit den geringsten Kosten, für `x_new` zu finden. Dazu wird jedesmal verglichen, ob (sofern `x_new` überhaupt durch `x_near` erreichbar ist) die Kosten, um `x_new` zu erreichen, geringer sind als die bisher geringsten Kosten. Wurde die Liste durchiteriert, wird der beste gefundene Nachbar für `x_new` als Vaterknoten gesetzt, also eine Kante zwischen `x_new` und `x_near` gezogen.
Nachdem so ein Pfad vom Startknoten zu `x_new` gebildet wurde, wird überprüft,

für welche Knoten in der Liste `X_NEAR` wiederum der in diesem Schritt hinzugefügte Knoten `x_new` die Gesamtkosten vom Startknoten aus verringern würde. Dazu wird wieder `X_NEAR` durchiteriert und bei allen Knoten `x_near`, wo der Weg über `x_new` geringere Kosten verursacht, `x_new` als Vaterknoten gesetzt.

2.2.2 Vor- und Nachteile RRT*

Der erste Unterschied zu einem *RRT* ist, dass nicht der nächste Nachbar als Vaterknoten gesetzt wird, sondern der mit den besten Kosten. Je nach Wahl des Radius *r* kann hier einiges an Ümweg"gespart werden.
Der Hauptunterschied ist jedoch die Neuverknüpfung bereits bestehender Knoten über `x_new`. Ein Nachteil des *RRTs* war auch, dass Knoten, die aus bereits gut mit Knoten gefüllten Regionen neu hinzugefügt wurden den Baum nicht wirklich bereichert hatten. Dies ändert sich nun, denn jeder von Knoten umgebene neu hinzugefügte Knoten verbessert die Kosten der meisten Knoten, sofern sie durch `x_new` besser erreichbar ist. Dies führt sogar soweit, dass ein *RRT** asymptotisch optimal ist, d.h. bei genügend langer Laufzeit der Pfad zum Optimum konvergiert.[TODO Literaturverweis]. Je nach Art der Kostfunktion kann der RRT* auch mit bestimmten Eigenschaften ausgestattet werden, z.B. präferiertes Wachsen in bestimmte Richtungen.

2.3 Nicht-holonomische Einschränkungen

Leider ist dieser Algorithmus so nicht eins zu eins auf ein Auto umsetzbar, da dieses bestimmten kinematischen Bedingungen unterworfen ist. Es kann sich zum Beispiel nicht in jede beliebige Richtung bewegen (z.B. seitlich) und hat abhängig vom Lenkradius gewisse Punkte, die nicht in einem Schritt erreichbar sind. Wie die Einschränkungen genau aussehen, welche Rahmenbedingungen an Hardware und Software gegeben waren und inwieweit der RRT*-Algorithmus angepasst werden musste, behandelt das nächste Kapitel.

3 Umsetzung

Bevor wir uns den notwendigen Anpassungen des *RRT**-Algorithmus widmen können, müssen wir die kinematischen und physikalischen Beschränkungen des Autos analysieren. Anschließend wird das verwendete Framework ROS - Robot Operating Systems - vorgestellt, wonach wir uns mit der Wahl einer geeigneten Metrik und Kostenfunktion beschäftigen.

3.1 Hardwareausstattung der Autos

Das Dahlem Center for Machine Learning and Robotics arbeitet mittlerweile mit dem Modelfahrzeug AutoNOMOS Mini v3 (1:10). Der Hauptcomputer auf dem Auto ist ein *Odroid*(XU4 64GB) mit Ubuntu Linux als Betriebssystem und ROS (Robot Operation Systems) als Steuerungssystem [**?**].
Motorisiert ist das Auto mit einem bürstenlosen [TODO??] DC-Servomotor FAUL-

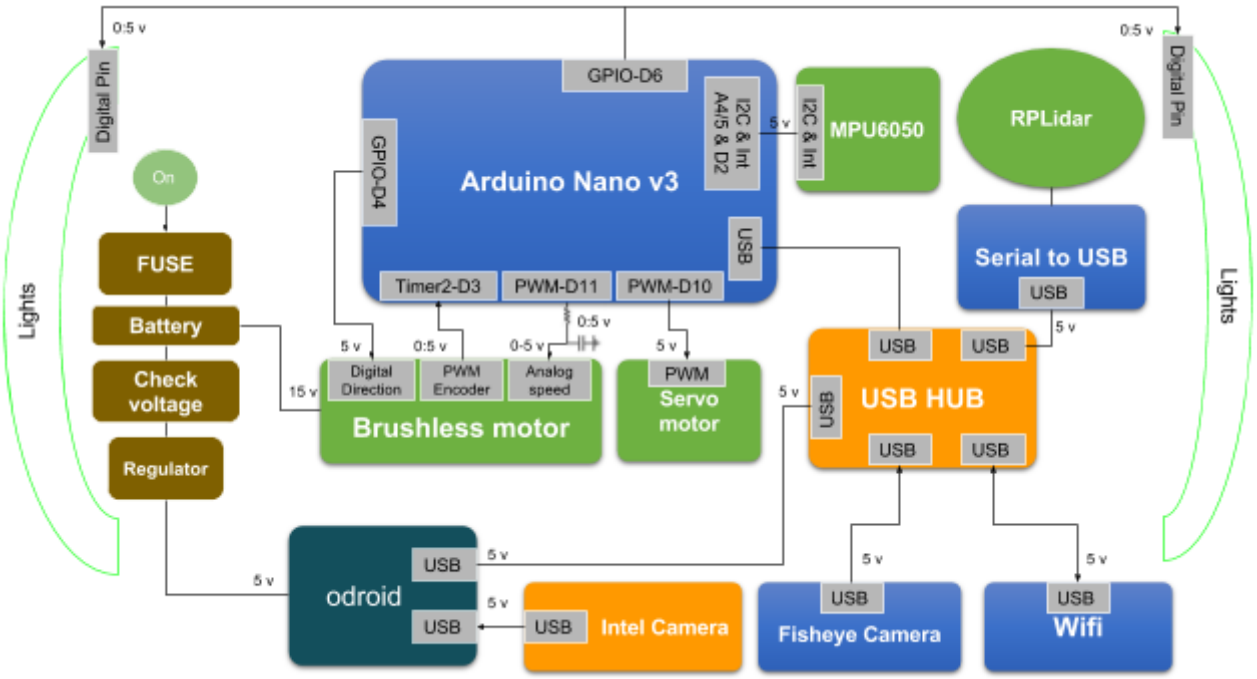

Abbildung 3: Überblick über die Module des AutoNOMOS Mini v3

HABER 2232. Die Lenkung wird von dem Servomotor HS-645-MG übernommen, beide Motoren werden mithilfe einer *Arduino Nano* Platine gesteuert.
Zur Wahrnehmung der Umgebung besitzt das AutoNOMOS Mini v3 mit dem RPLIDAR A2 360 einen rotierenden Laserscanner, der in der Lage ist, die Umgebung des Autos auf Hindernisse zu überprüfen. Als Rückgabewert liefert der RPLIDAR pro Gradwinkel den Wert, wie weit das nächste Hindernis in dieser Richtung entfernt ist, also insgesamt 360 Werte (einen pro Winkel).
Auf dem oberen Teil des Autos befestigt ist das *Kinect-type stereoscopic system* (Intel RealSense SR300), welches eine Wolke aus 3D Punkten liefert, die dazu benutzt werden kann, Hindernisse zu erkennen. Außerdem kann die Kamera des *Kinect-type* Sensors dazu benutzt werden, Fahrbahnmarkierungen und Objekte direkt vor dem Auto zu lokalisieren.
Der letzte äußere Sensor, auch am oberen Teil des Autos angebracht, ist die Fischaugen-Kamera. Diese zeigt nach oben, zur Decke, und kann dazu benutzt werden bestimmte markante, feststehende Objekte zu lokalisieren, damit das AutoNOMOS Mini v3 sich auch innerhalb von Räumen orientieren kann. Dazu kann eine GPS Navigationseinheit simuliert werden, indem die an der Decke angebrachten vier Lampen in unterschiedlichen Farben leuchten.
Die Sensoren sind entweder via USB 3.0 an der Hauptplatine oder direkt am *Odroid* angeschlossen.
An inneren Sensoren besitzt das AutoNOMOS Mini v3 eine MPU6050, die einen Beschleunigungssensor und ein *Gyroskop* enthält. Mithilfe dieser MPU kann das AutoNOMOS Mini v3 seine Orientierung, seine Richtung im Raum bestimmen. Außerdem können Messungen zur *Odometrie* ergänzt werden.

Das AutoNOMOS Mini v3 wird über eine 14,8 V Batterie mit Energie versorgt.

3.2 Software: ROS - Robot Operating Systems

ROS stellt Bibliotheken und Werkzeuge zur Verfügung, die Software-Entwicklern helfen sollen, Robotik Anwendungen zu kreieren [3]. Unter anderem beinhaltet ROS Gerätetreiber, Bibliotheken, Visualisierungswerkzeuge, Paketmanagement und vieles mehr. ROS ist Open Source und unter der BSD Lizenz verfügbar.

3.2.1 Architektur

Mithilfe von ROS können so genannte *Nodes*, ausführbare Programme, erzeugt werden, die über so genannte *Topics* kommunizieren können. Dies passiert über einen anonymisierten Publisher/Subscriber Mechanismus, das heißt Daten generierende Knoten können auf relevanten *Topics* Nachrichten senden, und interessierte Knoten können von relevanten *Topics* Nachrichten empfangen.
Für jedes *Topic* ist dabei auch die Nachrichtenart definiert, die für dieses *Topic* veröffentlicht und von diesem *Topic* empfangen werden. Dies können neben simplen Datentypen auch komplexe, selbst definierte Datenstrukturen sein. Dabei wird nur dieser eine, vorher festgelegte Datentyp der Nachricht vom *Topic* akzeptiert. *Topics* stellen nur eine unidirektionale Verbindung zur Verfügung. Für die Abwicklung von zum Beispiel Remote Procedure Calls sind sogenannte Services zuständig. Diese ermöglichen, eine Antwort auf eine bestimmte Anfrage nach dem Client Server Prinzip zurückzusenden.

3.3 APIs und Einbettung zu bereits vorhandene Knoten

Das Dahlem Center for Machine Learning and Robotics entwickelte ROS-Pakete für die Steuerung ihrer autonomer Fahrzeuge. Diese Pakete und daraus resultierenden ROS-Nodes können dazu genutzt werden, den von mir entwickelten RRT*-Pfadplaner möglichst gut einzubetten. So kann durch das visuelle indoor GPS die Position des Autos bestimmt werden, die der Pfadplaner für seine Berechnungen braucht. Die resultierende *Trajektorie*, die der Pfadplaner entwickelt, wird einem Steuerungsknoten übergeben, der diese *Trajektorie* in Motorbefehle, also Beschleunigungen und Lenkungen, umsetzt.

3.3.1 Bestimmung der Odometry und visual GPS

[TODO was benutzte ich jetzt?]

3.3.2 Steuerungsknoten fubController

Dieser Knoten lauscht auf das *Topic* `"planned_path"`. Auf `"planned_path"` kann eine *Trajektorie* publiziert werden, die dann mithilfe des Steuerungsknotens vom Auto abgefahren wird. Dabei kümmert sich dieser Steuerungsknoten allerdings nicht um etwagige Hindernisse, die mit dem Auto kollidieren könnten. Somit muss der Pfadplaner selbst alle Kollisionen mit Hindernissen ausschließen.

Das Format der *Trajektorie* wurde von der Arbeitsgruppe der FU Berlin definiert und besteht aus

- `std_msgs/Header`: Hier wird die aktuelle Zeit gespeichert.
- `string child_frame_id`: [TODO]
- `fub_trajectory_msgs/TrajectoryPoint[] trajectory`: Eine Liste aus Trajektorie-Punkten.

Ein Trajektorien-Punkt symbolisiert einen abzufahrenden Knotenpunkt und wiederum besteht aus

- `geometry_msgs/Pose pose`: Hier sind Position und Orientierung des Autos gespeichert.
- `geometry_msgs/Twist velocity`:Hier wird die Geschwindigkeit des Autos an diesem Punkt gespeichert.
- `geometry_msgs/Twist acceleration` Hier wird die Beschleunigung des Autos an diesem Punkt gespeichert.

Die Position wird in x-Position und y-Position angegeben, ausgehend von einer Ecke des Raumes. Die Orientierung wird durch ein *Quaternion* dargestellt, bei dem durch vier Werte die Drehung in jeder Richtung des Raumes genau bestimmt ist. Allerdings genügt uns die Drehung um die z-Achse, also die Drehrichtung über die Vertikalachse, weshalb dieses Quaternion in einen Winkel, der sogenannten Gierung, umgerechnet wird.
Nachdem die notwendige Infrastruktur erläutert wurde, folgt eine kurze Betrachtung der kinematischen und physikalischen Einschränkungen, denen das AutoNOMOS Mini v3 unterworfen ist. Anschließend können wir uns endlich dem Kernstück der Arbeit, dem eigentlichen RRT*-Pfadplaner, widmen.

3.4 Kinematische und physikalische Einschränkungen

Im Gegensatz zu holonomischen Fahrzeugen sind die Möglichen Pfade eines Autos nicht so einfach zu berechnen. Deshalb stellen wir zuerst zur Orientierung ein mathematisches Modell des Autos auf und definieren die Randbedingungen, unter denen es agiert. Die mathematischen Definitionen der Randbedingungen und des Autos sind in der Bachelorarbeit von Lukas Gödicke [4] gut beschrieben und werden hier (übersetzt) übernommen.

3.4.1 Randbedingungen

Die Position des Autos bezieht sich auf den Mittelpunkt zwischen den Hinterrädern mit den Koordinaten x und y. Der Winkel $\theta \in S, S = [0, 2\pi]$ steht für die Ausrichtung des Autos, der Winkel $\phi \in S$ für die Ausrichtung der Räder (=Lenkwinkel). Wir nehmen an, dass das Auto sich auf einer flachen Ebene bewegt. Somit ist der Arbeitsraum definiert durch $C = \mathbb{R} \times \mathbb{S}$. Ein Zustand des Autos wird beschrieben durch

$q \in C = (x, y, \theta)$.
Zur Vereinfachung der Rechnungen wird angenommen, dass die Räder bei Bedarf sich sofort, ohne Zeitverlust, in die gewünschte Position begeben. Da der Lenkwinkel keinen Einfluss auf den Momentanzustand des Autos hat wird dieser in der Zustandsbeschreibung nicht aufgeführt. Zusätzlich kommt noch hinzu, dass das Auto zur Testzwecken bestimmte Geschwindigkeiten nicht überschreiten darf, sodass der minimale Kurvenradius nur durch den Lenkwinkel beschränkt ist.

3.4.2 Einschränkungen durch das Auto

Ein Auto hat eine Möglichkeit sich fortzubewegen: Es kann beschleunigen. Eine negative Beschleunigung wäre Bremsen oder sogar Rückwärtsfahren. Zusätzlich zu dieser einen Möglichkeit, den Zustand des Autos zu ändern, kann das Auto das Resultat der Beschleunigung ändern, indem der Lenkwinkel angepasst wird. Dieser reicht von ϕ_{max} bis $-\phi_{max}$, den maximalen Lenkwinkeln des Autos. Je nach Lenkwinkel i_ϕ, Geschwindigkeit i_g und Anfangsausrichtung des Autos θ ändern sich verschiedene Parameter des Zustandes $q(x, y, \theta)$:

$$\begin{aligned} \dot{x} &= i_g cos\phi \\ \dot{y} &= i_g sin\phi \\ \dot{\theta} &= \frac{i_g}{L} tan(i_\phi) \end{aligned}$$

L ist hierbei der Radstand, also der Abstand zwischen Vorder- und Hinterachse.

3.4.3 Konsequenzen

Ein normaler *RRT** besitzt *Trajektorien* zum Ziel, die für ein Auto aufgrund der oben benannten Beschränkungen nicht abfahrbar sind. So entstehen beispielsweise diskrete Ecken und Kanten die für ein sich kontinuierlich bewegendes Auto unmöglich zu bewältigen ist. Somit ist die erste Aufgabe, die Pfade im *RRT** für das Auto irgendwie abfahrbar zu machen.
Dazu hatte ich am Anfang unterschiedliche Ansätze:

1. RRT* normal ausführen. Sobald ein Pfad zum Ziel gefunden: Diesen so bearbeiten, dass dieser vom Auto zu bewältigen ist

2. RRT* normal ausführen, jedoch mit geeignetem Algorithmus dafür sorgen, dass das Auto jeden Knoten erreichen kann

3. Nur Knoten hinzufügen, die von ihrem Vaterknoten aus unter oben genannten Bedingungen stets erreichbar sind

Das Problem des ersten Ansatzes war, das unter Umständen je nach Umgebung ein Pfad nicht nur wegen bestimmter Ecken und Kanten nicht abfahrbar ist, sondern auch wegen Kurven, die Aufgrund des Lenkradius des Autos nicht zu bewältigen wären. Es wäre zwar möglich, für kurze Zeit die geplante Trajektorie zu verlassen und nach Ende der Kurve wieder zu ihr zu stoßen, allerdings ist es unter diesen Umständen

schwierig eine Hindernisfreiheit zu garantieren.
Die zweite Variante war Thema der Bachelorarbeit meines oben bereits erwähnten Kommolitonen David Gödicke [4]. Dieser benutze für den Pfad zwischen zwei Kurven Dubins Curves [**?**], bei denen drei Einstellungen aus linkem maximalem Lenkeinschlag, rechtem maximalen Lenkeinschlag und Geradeausfahren dreimal hintereinander verkettet werden . Damit kann zwar jeder Punkt des Arbeitsbereiches erreicht werden, allerdings war die Berechnung zu aufwändig und langsam, was für Echtzeit Szenarien im Straßenverkehr ein hartes Ausschlusskriterium ist [vergleiche 4, Kapitel 7].
Deshalb entschloss ich mich für die dritte Variante:

3.5 RRT*-Pfadplaner für Automobile

Diese beinhaltete, bei der Erstellung des RRT* vom Auto nicht erreichbare Knoten gar nicht erst zuzulassen. Das Auto sollte also von einem Knoten zum nächsten mit nur einer Lenkeinstellung direkt fahren können. Dies sollte die Berechnungszeit stark verringern.
Zum Auschluss der Knoten verwendete ich zuerst eine Heuristik: Ein nächster Nachbar N für einen neu hinzugefügten Knoten K kam nur dann in Frage, falls dieser den neu hinzugefügten Knoten K auch erreiche konnte. Dazu wurde die Ausrichtung des Autos im potentenziellen Elternknoten N mit dem Richtungsvektor zwischen den beiden Knoten verglichen [TODO Zeichnung]. Somit war K gültig, falls der Winkel α zwischen der Ausrichtung von N und dem direktem Weg zum Knoten K (Richtungsvektor) einen bestimmten Wert ω nicht überschritt. Ungültige Knoten wurde verworfen. [TODO in Formel zsmfassen]
Als nächster Schritt wird, wie in Kapitel 2.2 geschildert, der Knoten K an den nächsten (gültigen) Nachbarn N projiziert und der Vaterknoten bestimmt.

3.5.1 Bestimmung des Vaterknotens

Dabei muss bemerkt werden, dass alle Punkte außerhalb der Wendekreise des Autos theoretisch mit einer Lenkeinstellung erreichbar sind. Allerdings fährt dabei das Auto sehr große Kreise und Umwege, die nicht nur weniger effizient sind, sondern auch Probleme bei der Hindernisvermeidung aufwerfen. Um dafür zu sorgen, dass das Auto diese großen Kreise meidet, wird die maximale Distanz zwischen zwei verbundenen Knoten so hinunter gesetzt, das nicht mehr als rechtwinklige Drehungen der Ausrichtung des Autos möglich sind [TODO Bild mit erreichbarem Raum + Bild mit beschränktem Raum].
Nun wird aus allen nächsten Nachbar innerhalb des Radius R der mit den besten Kosten gewählt. Ausgeschlossen müssen natürlich alle Punkte, die innerhalb der Wendekreise des Autos liegen. Ausgerechnet wurde dies, indem vom potentiellen Vaterknoten N aus zwei Mittelpunkte der Wendekreise bestimmt wurden [Bild/Formel hinzufügen]. Dann wurde davon aus mithilfe des Radiuses des Wendekreises bestimmt ob der Punkt K jetzt drin lag oder nicht.
Nachdem erfolgreich ein Vaterknoten bestimmt und dem neu hinzugefügtem Knoten K zugewiesen wurde, muss noch die Ausrichtung oder Orientierung bestimmt werden, die da Auto im Knoten K hat.

3.5.2 Bestimmung der Orientierung

Da das Auto im Vaterknoten N eine bestimmte Ausrichtung hat und mit nur einer Lenkeinstellung zum nächsten Knoten K gelangt, ist dort die Ausrichtung dadurch festgelegt und berechnet sich aus dem Winkel θ und dem Richtungsvektor von N nach K. [TODO Zeichnung + Erklärung] Nach Berechnung der Orientierung sind alle notwendigen Operationen an K abgeschlossen. Nun wird von K aus das Neuverknüpfen des Baumes, das Rewiring, durchgeführt.

3.5.3 Rewiring

[TODO Pseudo code] Der Knoten K wurde neu in den Baum hinzugefügt. Nun wird überprüft, ob bereits vorhandene Knoten besser erreichbar sind, wenn der Weg über K gewählt wird. Dazu werden zunächst im Radius R alle in Frage kommenden Nachbarn ausgewählt. Aussortiert werden alle, die entweder vom Knoten K aus nicht erreichbar sind (gleiche Überprüfung wie oben) oder aber bei denen K keine Verbesserung bewirkt also die Kosten über den Knoten K gleich oder kleiner sind.
Wenn man von den jetzt noch ausgewählten Knoten M einfach K als Vater hinzufügen würde, müsste jeweils die Orientierung der Knoten, der Winkel θ, geändert werden. Das Auto gelangt auf anderem Wege zu diesen Knoten und hat somit eine andere Ausrichtung als vorher. Allerdings kann es dadurch vorkommen, das wenn der Knoten $m \in M$ eine neue Orientierung hat, Kinder von m nicht mehr durch m erreichbar sind [TODO Zeichnung]. Deshalb wird, anstatt die Orientierung zu ändern, einfach ein neuer Knoten hinzugefügt, der zwar die gleicher Koordinaten hat wie m, aber eine andere Orientierung. Dies wird mit allen von K erreichbaren Knoten durchgeführt. Alle somit zum Baum neu hinzugefügten Knoten sind durch K besser erreichbar und können nun selbst zur Neuverknüpfung des Baumes beitragen. Somit wird der Algorithmus des Neuverknüpfens rekursiv auf alle im vorherigen Schritt durch das Rewiring neu hinzugefügten Knotens angewendet.
Der Vorteil dieses Mechanismus ist, dass mögliche Optimierungen schnell durch den Baum wandern und keine Knoten ungültig werden. Außerdem müssen nicht so viele randomisierte Punkte erzeut werden, was der Laufzeit zu Gute kommt. Nachteile sind allerdings eine geringere Anzahl von räumlich verteilten Punkten (es liegen Punkte "übereinander"). Zudem kann dieses Verfahren viel Zeit beanspruchen, ohne viel Effekt zu haben, wenn besonders viele Punkte im Baum existieren und Bereiche weit abseits der Zielregion neu verknüpft werden.

3.6 Metrik und Kostenfunktion

Eine besondere Bedeutung kommt beim Rewiring der Kostenfunktion zu. Denn neben der Aufgabe, für möglichst kurze und optimale Pfade zu sorgen, kann mithilfe der Kostenfunktion für besonders gut abfahrbare Pfade gesorgt werden.
Wenn durch die Orientierung des Vaterknotens die Orientierung des Kindknotens bereits festgelegt wird, können aus eigentlich geraden Strecken sehr ungünstige Pfade entstehen.
In Abbildung 1 [TODO anpassen?] wird als Kostenfunktion der euklidische Abstand benutzt. Knoten C wählt aus B1 und B2 den nächsten Nachbarn aus, das ist B2. Leider

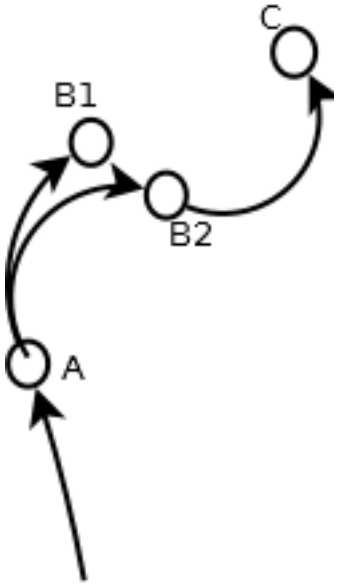

Abbildung 4: einfache euklidische Kostenfunktion

entsteht dadurch eine sehr kurvige Route, bei der vom fast maximalem Lenkeinschlag nach rechts auf den maximalen Lenkeinschlag der anderen Seite gewechselt werden muss. Neben Nachteilen des Komforts, der Sicherheit und längerem Weg wird auch die Ungenauigkeit höher. Anfangs wurde angenommen, dass der Lenkwinkel sich sofort ändern kann. Während bei kleinen Änderungen diese Annahme nur für kleine Fehler sorgt, sind die Auswirkungen bei solch starken Änderungen deutlicher spürbar.
In Abbildung zwei hingegen wurde eine bessere Kostenfunktion gewählt, sodass über B1 gehend ein effizienterer, kürzerer Pfad entsteht. Die Kostenfunktion muss dabei die Ausrichtung des Autos beim Vaterknoten berücksichtigen. [TODO Auswahl] Eine Möglichkeit wäre dann, die tatsächliche zu fahrende Strecke des Autos zu benutzen und dadurch effizientere Pfade zu bevorzugen. Eine andere Möglichkeit ist, zu harte Richtungsänderungen zu bestrafen und kleine $\Delta\theta$ zu bevorzugen.

3.7 Datenstruktur

RRT* hat eine amortisierte Laufzeit von O(n log(n)). Um diese Laufzeit auch in der Praxis zu erreichen, muss auf spezielle Datenstrukturen zurückgegriffen werden. Insbesondere müssen mehrfach sowohl der nächste Knoten als auch die nächsten innerhalb eines Radius ermittelt werden. Für die sogenannte *k-nearest neighbour* Suche und *radius k-nearest neighbour* existieren bereits viele wissenschaftliche Untersuchungen und auch empfohlene Datenstrukturen, wie zum Beispiel k-d-Bäume [**?**]. Leider benötigt der RRT* Algorithmus eine dynamische Datenstruktur, da Punkte erst nach und nach hinzugefügt werden. Eine weitere Einschränkung waren fehlende Implementierungen, die auf die Anwendungsfälle von RRT* zugeschnitten waren.
Aufgrund dieser Einschränkungen wurde keine komplizierte, unter Umständen bessere Datenstruktur gewählt. Stattdessen wurde ein einfaches Gitter gewählt, deren

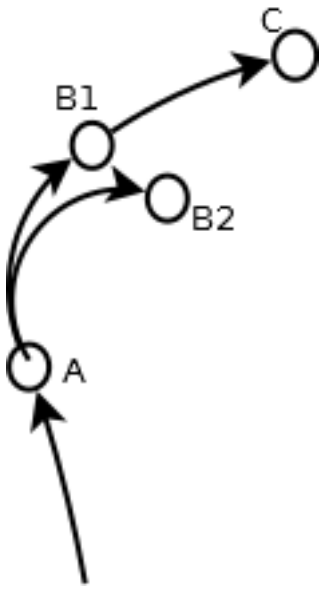

Abbildung 5: erweiterte Kostenfunktion

Achsen die x- und y-Werte des einzufügenden Punktes repräsentierte. Dadurch fallen Punkte, die nah beeinander sind, in die gleiche oder benachbarte Zellen. Je nach Wahl der Größe der Zellen müssen deutlich weniger Knoten überprüft werden. Ein weiterer Vorteil des Gitters ist das schnelle Einfügen eines Knotens: Dadurch dass das Gitter statisch bleibt können Arrays als Rahmen gewählt werden und die Laufzeit zum Einfügen von Knoten liegt bei O(1).

3.8 Dokumentation der Durchführung und entstandener Artefakte

3.9 Beschreibung besonderer Schwierigkeiten und wie diese umgangen wurden

[TODO] Erwähnung technischer schwierigkeiten a la visual gps, fub controller

3.9.1 Tests und Testdatensätze/Szenarien für die Software)

3.9.2 Korrektheitsbeweise

3.10 (Evaluation - nur wenn ich dafür Zeit habe)

4 Zusammenfassung

...

Literaturverzeichnis

[1] Nancy M. Amato and Yan Wu. A randomized Roadmap Method for Path and Manipulation Planning. In *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, ICRA, pages 113–120. IEEE, 1996.

[2] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[3] Arbeitsgruppe Robotics FU Berlin. Hardware (AutoNOMOS Model v3). https://github.com/AutoModelCar/AutoModelCarWiki/wiki/Hardware-(AutoNOMOS-Model-v3).

[4] C.Rösmann, F.Hoffmann, and T.Bertram. Timed-elastic-bands for time-optimal point-to-point nonlinear model predictive control. In *European Control Conference (ECC)*, 2015.

[5] Lester Eli Dubin. ON PLANE CURVES WITH CURVATURE, 1961.

[6] Open Source Robotics Foundation. Robot Operating System. http://wiki.ros.org/.

[7] Lukas Gödicke. Rrt based path planning in static and dynamic environment for model cars, March 2018.

[8] Jean-Claude Latombe Jerome Barraquand. Robot Motion Planning: A Distributed Representation Approach, 1991.

[9] Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[10] Steven M. LaValle, James J. Kuffner, and Jr. Rapidly-exploring random trees: Progress and prospects, 2000.

[11] S.Karaman and E.Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Robotics: Science and Systems*, 2010.

A Anhang

Quellcode der LaTeX-Klasse `agse-thesis`:[1]

```
\NeedsTeXFormat{LaTeX2e}
\ProvidesClass{agse-thesis}[2017/05/23 v0.1 AGSE Thesis]

%%% Read options
% Language: Default is German
\newcommand{\lang}{ngerman}
\DeclareOption{de}{\renewcommand{\lang}{ngerman}}
\DeclareOption{en}{\renewcommand{\lang}{english}}

% Font family: Default is LaTeX's lmodern
\newcommand{\fonttype}{plain}
\DeclareOption{serif}{      \renewcommand{\fonttype}{serif}}
\DeclareOption{plain}{      \renewcommand{\fonttype}{plain}}
\DeclareOption{sans-serif}{\renewcommand{\fonttype}{sans-serif}}

% Document type: Default is article (twosided)
\newcommand{\baseClass}{article}
\DeclareOption{article}{%
    \renewcommand{\baseClass}{article}
    \PassOptionsToClass{twoside}{article}
}
\DeclareOption{book}{\renewcommand{\baseClass}{book}}

\newcommand{\useparskip}{no}
\DeclareOption{parskip}{\renewcommand{\useparskip}{yes}}
\DeclareOption{noparskip}{\renewcommand{\useparskip}{no}}

\DeclareOption*{\PassOptionsToClass{\CurrentOption}{\baseClass}}
\ProcessOptions\relax
\LoadClass[11pt,a4paper]{\baseClass}

% Load required language
\RequirePackage[\lang]{babel}

% Load required font
\RequirePackage{xifthen}
\ifthenelse{\equal{\fonttype}{plain}}{
    \RequirePackage{lmodern}
}{}
\ifthenelse{\equal{\fonttype}{serif}}{
    \RequirePackage[sc]{mathpazo}
    \linespread{1.05}  % Palladio needs more leading (space between
        lines)
}{}
\ifthenelse{\equal{\fonttype}{sans-serif}}{
    \RequirePackage{paratype}
    \renewcommand*\familydefault{\sfdefault}
}{}
\RequirePackage[T1]{fontenc}
```

[1] Es ist nicht üblich, den gesamten produzierten Quellcode bei einer Abschlussarbeit in Textform abzugeben.

```
% Allow unicode in input files
\RequirePackage[utf8]{inputenc}

% Set layout
\RequirePackage[
    inner=3.4cm,
    outer=3cm,
    top=3cm,
    marginparwidth=2.5cm,
    marginparsep=0.1cm
]{geometry}

\ifthenelse{\equal{\useparskip}{yes}}{
    \RequirePackage{parskip}
}{}

% Header and Footer Style
\RequirePackage{fancyhdr}
\pagestyle{fancy}
\fancyhead{}
\fancyhead[OR]{\slshape\nouppercase{\rightmark}}
\fancyhead[EL]{\slshape\nouppercase{\leftmark}}
\fancyfoot{}
\fancyfoot[C]{\thepage}
\renewcommand{\headrulewidth}{0pt}

% Display Chapter and Section for book class
\ifthenelse{\equal{\baseClass}{book}}{
    \renewcommand{\chaptermark}[1]{\markboth{%
        \chaptername\ \thechapter.\ #1}{\chaptername\ \thechapter.\
            #1}}
}{%
% Display Section and Subsection for article class
    \renewcommand{\sectionmark}[1]{\markboth{%
        \thesection.\ #1}{\thesection.\ #1}}
}

% PDF settings
\usepackage[%
    pdfstartview=FitH,
    linktocpage,
% two lines below = color links
    colorlinks=true,
    citecolor=blue!20!black!30!green,
% two lines below = don't color links
    %colorlinks=false,
    %pdfborder={0 0 0},
]{hyperref}

% Tables
\usepackage{tabularx}
\newcolumntype{L}[1]{>{\raggedright\arraybackslash}p{#1}}

% Misc
\RequirePackage{fancyref}
\RequirePackage{url}
\RequirePackage{makeidx}
```

```
\RequirePackage[pdftex]{graphicx}

%%% BibTeX
\RequirePackage[numbers,sort&compress]{natbib}
\RequirePackage[nottoc]{tocbibind}
\bibliographystyle{plain}

% Java Code Listing Style
\RequirePackage{xcolor}
\RequirePackage{listings}
\definecolor{darkblue}{rgb}{0,0,.6}
\definecolor{darkgreen}{rgb}{0,0.5,0}
\definecolor{darkred}{rgb}{0.5,0,0}
\lstset{%
    language=Java,
    basicstyle=\ttfamily\small\upshape,
    commentstyle=\color{darkgreen}\sffamily,
    keywordstyle=\color{darkblue}\rmfamily\bfseries,
    breaklines=true,
    tabsize=2,
    xleftmargin=3mm,
    xrightmargin=3mm,
    numbers=none,
    frame=single,
    stringstyle=\color{darkred},
    showstringspaces=false
}

% Custom commands
\newcommand\zb{z.\,B.\ }
\renewcommand\dh{d.\,h.\ }
\newcommand{\mailto}[1]{\href{mailto:#1}{#1}}

\RequirePackage{pgfkeys}
\pgfkeys{
    student/id/.estore in = \studentID,
    student/mail/.estore in = \coverpageMail,
    thesis/type/.estore in = \thesisType,
    thesis/type = Bachelorarbeit,
    thesis/date/.estore in = \thesisDate,
    thesis/date = \today,
    thesis/advisor/.estore in = \advisor,
    thesis/examiner/.estore in = \firstExaminer,
    thesis/examiner/2/.estore in = \secondExaminer,
    thesis/group/.estore in = \groupName,
    thesis/group = {Arbeitsgruppe Software Engineering},
    title/size/.store in = \titleFontSize,
    abstract/separate/.estore in = \separateAbstract,
}

% Define abstract environment for book class
\ifthenelse{\equal{\baseClass}{book}}%
    {\newenvironment{abstract}%
        {\begin{center}\textbf{\small\abstractname}\end{center}\
            quotation\small}%
        {\endquotation}%
```

```
    }{}

% (Re)define frontmatter and mainmatter
\ifthenelse{\equal{\baseClass}{book}}{
    \let\frontmatterOrig\frontmatter
    \renewcommand{\frontmatter}{
        \frontmatterOrig
        \pagestyle{plain}
    }
    \let\mainmatterOrig\mainmatter
    \renewcommand{\mainmatter}{
        \mainmatterOrig
        \pagestyle{fancy}
        \setcounter{page}{1}
    }
}{
    \newcommand{\frontmatter}{
        \pagestyle{plain}
        \pagenumbering{roman}
        \setcounter{page}{1}
    }
    \newcommand{\mainmatter}{
        \pagestyle{fancy}
        \pagenumbering{arabic}
        \setcounter{page}{1}
    }
}

\RequirePackage{xstring}
\RequirePackage{etoolbox}
\newcommand{\coverpage}[2][]{
    \pgfkeys{#1}
    \pagestyle{empty}

    \ifcsdef{separateAbstract}{\mbox{}\vspace{15mm}}{\mbox{}}

    \begin{center}
        \LARGE
        \textbf{Freie Universität Berlin}

        \vspace{4mm}

        \normalsize
        \thesisType{} am Institut für Informatik der Freien Universitä
            t Berlin

        \vspace{2mm}

        \groupName

        \ifcsdef{separateAbstract}{\vspace{25mm}}{\vspace{13mm}}

        \ifcsdef{titleFontSize}{}{%
            \StrLen{\thesisTitle}[\titleLength]
            \ifthenelse{\titleLength > 100}{%
                \let\titleFontSize\LARGE
            }{%
```

```
                \let\titleFontSize\huge
            }
        }
        \titleFontSize\thesisTitle

        \ifcsdef{separateAbstract}{\vfill}{\vspace{13mm}}

        \Large
        \studentName \\
        \normalsize
        Matrikelnummer: \studentID\\
        \mailto{\coverpageMail}

        \vspace{4mm}

        \begin{tabular}{rl}
            \ifcsdef{advisor}{Betreuer: & \advisor\\}{}
            Eingereicht bei: & \firstExaminer \\
            \ifcsdef{secondExaminer}{Zweitgutachter: & \secondExaminer
                \\}{}
        \end{tabular}

        \vspace{4mm}

        Berlin, \thesisDate
    \end{center}

    \ifcsdef{separateAbstract}{\cleardoublepage\frontmatter}{\vfill}
    \begin{abstract}
        #2
    \end{abstract}
    \cleardoublepage
    \ifcsdef{separateAbstract}{}{\frontmatter}
}
```