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, 26. April 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.

26. April 2018

Bernd Sahre

Inhaltsverzeichnis

1 Einleitung

Schon die Griechen der Antike träumten von autonomen, selbstfahrenden Fahrzeugen [TODO Literaturverweis]. Mittlerweile ist die Forschung in diesen Bereichen so weit fortgeschritten, dass dieser Traum schon bald Wirklichkeit werden könnte. Dabei werden unterschiedliche Ansätze verfolgt, 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 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 Pfadplanung wird hauptsächlich das Prinzip elastischer Bänder (Time-Elastic-Bands, [vgl. 2]) zur Erzeugung von *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 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 der Arbeit

(Arbeitstitel) [TODO doppelung mit struktur]

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 [4], ein Algorithmus zur Findung eines Pfades zum Ziel durch unbekanntes Gelände, wird bei [Querverweis] noch weiter erläutert
RRT*	Eine verbesserte Variante des RRT, bei dem die Pfade optimiert werden. Asymptoptisch optimal [**?**]
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	

1.1.4 Struktur

[TODO Querverweise] Nach einer kurzen Hinführung zum Thema werden zuallererst die Grundlagen 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. Im vierten Kapitel 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.

1.1.5 Wichtige Quellen und deren Beitrag zum Thema

Es existieren[TODO RRT-Quellen [vgl 4], 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 [3] 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 3]. 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 4]. 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 4]. 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:

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 beleuchtet [TODO Querverweis].
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. [TODO Bild]
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 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, vlg. [TODO Literarur LaValle Kap 3]:

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 [TODO Bild RRT einfügen]. 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 [TODO Literaturverweis Kap4 Alg6 Frazzoli].

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]

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. Deshalb wird nun der *RRT** Algorithmus im nächsten Kapitel angepasst, um auf dem Auto anwendbar zu sein.

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, bevor 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 ÄutoNOMOS 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 FAULHABER 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 Auto-

NOMOS 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 [**?**]. 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.
Topics stellen nur eine unidirektionale Verbindung zur Verfügung. Für die Abwicklung von zum Beispiel *Remote Procedure Calls* sind sogenannte Services zuständig. Diese ermögliche, eine Antwort auf eine bestimmte Anfrage nach dem Client Server Prinzip zurückzusenden.

3.3 APIs und bereits vorhandene Knoten

Das Dahlem Center for Machine Learning and Robotics entwickelte ROS-Pakete für die Steuerung autonomer Autos. Diese Pakete und daraus resultierenden ROS-Nodes können dazu genutzt werden, den 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 fubMigController

Dieser Knoten lauscht auf das *Topic* "planned_path"

3.4 Algorithmen: notwendige Anpassungen

3.4.1 RRT*

Dazu sind Änderungen besonders an zwei Stellen nötig:

1. Es muss bei der Auswahl des Vaterknotens überprüft werden, ob der eingefügte Knoten über den Vaterknoten überhaupt erreicht werden kann.

2. Beim Rewiring - dem Neuverknüpfen der Knoten - wird nicht die alte Verbindung zum Vaterknoten gelöscht. Stattdessen wird ein neuer Knoten erzeugt, der als Vaterknoten den eingefügten Knoten `x_new` hat. Dies ist nötig, weil

- Datenstruktur Nodes
- Berechnung der Orientierung
- Rewiring

3.5 Metrik und Kostenfunktion

Bestrafen für harte Lenkung bzw. (starke) Lenkänderung Belohnung für geradeausfahren und kurze Wege

3.6 Dokumentation der Durchführung und entstandener Artefakte

3.7 Verwendete Metriken

3.8 Beschreibung besonderer Schwierigkeiten und wie diese umgangen wurden

3.9 (Evaluation - nur wenn ich dafür Zeit habe)

3.9.1 Tests und Testdatensätze/Szenarien für die Software)

3.9.2 Korrektheitsbeweise

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] 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.

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

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

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}
}
```