B. Sc. Hauke Petersen

# Selektionmechanismen für roboter- und szenarienspezifische Pfadplanung


INF FAKULTÄT FÜR INFORMATIK

Institut für Verteilte Systeme

Masterarbeit

# Selektionmechanismen für roboter- und szenarienspezifische Pfadplanung

Autor: B. Sc. Hauke Petersen

Professor: Junior-Prof. Dr.-Ing. Sebastian Zug

Professor: Prof. Dr.-Ing. habil. Arndt Lüder

Betreuer: M. Sc. Martin Seidel

Wintersemester 2016/17

**Petersen, Hauke:** *Selektionmechanismen für roboter- und szenarienspezifische Pfadplanung*
Masterarbeit, Otto-von-Guericke-Universität
Magdeburg, 2016/17.

# Kurzreferat

Die Einsatzgebiete autonomer mobiler Roboter erweitern sich stetig. In zahlreichen Bereichen des Lebens werden sie eingesetzt, ob als Serviceroboter in öffentlichen Gebäuden, Logistiksystem im Großhandel oder als Rasenmäher in heimischen Gärten. Daher ist die Weiterentwicklung der Hard- und Software von elementarer Bedeutung. Schon jetzt ist auf internationalen Wettbewerben, wie dem RoboCup, eine deutlicher Fortschritt zu erkennen. Dabei ist die automatisierte Navigation eines der forschungsintensivsten Gebiete der Robotik. Durch sie wird der optimale Betrieb von mobilen Robotersystemen sichergestellt. In der vorliegenden Arbeit wird eine innovative Schnittstelle, in Form eines Plug-ins, zwischen der *Open Motion Planning Library* und dem *Robot Operating System* entwickelt, deren Einsatzgebiet sich im Bereich der Pfadplanung von holonomen Robotern bis hin zu Systemen mit kinematischen Einschränkungen erstreckt. Die Pfadplanungsalgorithmen der *Library* werden unter Verwendung des Plug-ins getestet und miteinander in eigens dafür erstellten Szenarien verglichen. Für die Auswertung der erhobenen Daten wurde ein, auf Hyperbeln basierendes, Analyseverfahren entworfen. Auf Grundlage der gewonnenen Ergebnisse ließen sich Regelsätze ableiten, die fundamental für die Entwicklung zukünftiger automatisierter Planerselektionen sind.

# Abstract

The applications of autonomous mobile robots are constantly expanding. They are used in multiple areas of everyday life, whether as service robots in public buildings, as logistic systems in wholesale trade or as lawnmowers in local gardens. Therefore the further development of hardware and software is of utter importance. A major progress is clearly visible when participating in international competitions, such as the RoboCup. Thereby automated navigation is one of the most research-intensive areas of robotics. It ensures an optimal operation of mobile robotic systems. In the present paper, an innovative interface, in form of a plug-in is developed as a connection between the *Open Motion Planning Library* and the *Robot Operating System*, which includes the path planning of holonomic robots and systems with kinematic constraints. The path planning algorithms of the library are tested using the plug-in and compared with each other in scenarios specifically created for this purpose. A hyperbola-based analysis method was developed for the analysis of the collected data. Based on these obtained results, it was possible to derive rules which are fundamental for the developing automated planner selections in the future.

# Inhaltsverzeichnis

**Abbildungsverzeichnis** **IV**

**Tabellenverzeichnis** **V**

**Abkürzungsverzeichnis** **VIII**

**1 Einleitung** **1**

**2 Grundlagen** **3**

2.1 Projekte . . . 3
2.1.1 Robocup@Work . . . 4
2.1.2 Autonome Fahrräder in Urbanen Szenarien . . . 5
2.1.3 Vergleich . . . 7
2.2 Fortbewegung von mobilen Systemen . . . 8
2.2.1 Roboter mit Rädern . . . 8
2.2.2 Roboter mit Beinen . . . 10
2.2.3 Fliegende Roboter . . . 11
2.2.4 Weitere Roboter . . . 13
2.3 Umgebungserfassung von Robotern . . . 14
2.3.1 Sensoren . . . 14
2.3.2 Navigation und Kartendarstellungen . . . 16
2.4 The Robot Operating System . . . 22
2.4.1 ROS Navigation Stack . . . 23
2.4.2 Costmap . . . 24
2.4.3 Plug-ins im ROS-Kontext . . . 26
2.5 The Open Motion Planning Library . . . 27
2.5.1 Konfigurationsräume . . . 27
2.5.2 Pfadplaner der OMPL . . . 29

**3 Konzept** **35**
3.1 Plug-in Entwurf . . . 35
3.2 Auswahl der Szenarien . . . 37
3.3 Vergleich der Planer . . . 38

**4 Umsetzung** **41**
4.1 Testszenarien . . . 41
4.1.1 RoboCup@Work Weltmeisterschaft 2017 . . . 41
4.1.2 Campus der Otto-von-Guericke-Universität . . . 43
4.2 Anforderungsanalyse . . . 43
4.3 OMPL Planer Plug-in . . . 46
4.3.1 Initialisierung . . . 46
4.3.2 Ablauf der Planung . . . 47
4.3.3 Kollisionsabfrage . . . 48
4.3.4 Zusätzliche Funktionen . . . 49
4.4 Versuchsanordnung . . . 50
4.4.1 Ablauf der Experimente . . . 50
4.4.2 Hard- und Softwarespezifikationen . . . 51

**5 Evaluation** **53**
5.1 Vorbetrachtungen . . . 53
5.2 Experiment 1: Einflusses der Parameter . . . 54
5.3 Experiment 2: Vergleich der Planungsalgorithmen . . . 56
5.3.1 Erläuterung der Darstellung . . . 56
5.3.2 Auswertung der Ergebnisse . . . 58
5.4 Experiment 3: Vergleich der restlichen Planer . . . 63
5.5 Experiment 4: Einsatz im OvGU-Szenario . . . 64
5.6 Fazit . . . 66

**6 Zusammenfassung und Ausblick** **69**
6.1 Zusammenfassung . . . 69
6.2 Ausblick . . . 70

**Literaturverzeichnis** **73**

# Abbildungsverzeichnis


2.1 Kuka youBot des Team robOTTO . . . . . 3
2.2 RoboCup@work Arena . . . . . 4
2.3 AFiUS Fahrrad . . . . . 5
2.4 Elbauenpark Magdeburg . . . . . 6
2.5 Beispiel eines *omni-wheels* [52] . . . . . 9
2.6 Roboter mit Rädern . . . . . 10
2.7 Roboter mit Beinen . . . . . 11
2.8 Fliegende Roboter . . . . . 12
2.9 Hybride Roboter . . . . . 13
2.10 Beispiele für aktive und passive Sensoren . . . . . 15
2.11 Beispielumgebung für verhaltensbasierte Navigation . . . . . 16
2.12 Unterschied verhalten-/ kartenbasierte Navigation . . . . . 17
2.13 Kontinuierliche Darstellung . . . . . 19
2.14 exakte Zelldekomposition . . . . . 20
2.15 Approximierte Zelldekomposition . . . . . 21
2.16 ROS-*node-graph* . . . . . 22
2.17 Schematische Abbildung der move_base . . . . . 23
2.18 Inflation Funktion der *costmap* . . . . . 24
2.19 *Occupancy grid map* und darauf aufgebaute *costmap* . . . . . 25
2.20 Beispiel für Plug-ins . . . . . 26
2.21 Dubins Pfad . . . . . 29
2.22 Planer der OMPL . . . . . 30
2.23 Aufbau eines Random Tree . . . . . 30
2.24 Beispiel einer *Probabilistic Roadmap* . . . . . 32

3.1 Konzept für die Kombination aus OMPL und ROS . . . . . 36
3.2 Darstellung der OMPL API . . . . . 37

4.1 WM-Karte - *Occupancy grid* der Robocup@work Weltmeisterschaft 2017 . . . . . 42
4.2 Teil des Campus der OvGU Magdeburg . . . . . 44
4.3 UML Diagramm der OMPL Plug-in Umsetzung . . . . . 47
4.4 Ausschnitt des ROS *rviz* Visualisierungstools während der Pfadplanung . . . . . 49

5.1 Beispiel für das Fehlverhalten des *Dubins* Konfigurationsraums . 54
5.2 Vergleich der Planungskonfigurationen . . . . . 55
5.3 Erläuterung der Darstellungen . . . . . 57
5.4 Ergebnisse - PRM*-Planer - WM-Karte . . . . . 59
5.5 Ergebnisse - LazyPRM*-Planer - WM-Karte . . . . . 59
5.6 Ergebnisse - RRT*-Planer - WM-Karte . . . . . 61
5.7 Ergebnisse - FMT*-Planer - WM-Karte . . . . . 61
5.8 Ergebnisse - Informed RRT*-Planer - WM-Karte . . . . . 62
5.9 Ergebnisse - T-RRT-Planer - WM-Karte . . . . . 62
5.10 Vergleich der Hyperbelkonstanten - WM-Karte . . . . . 63
5.11 Darstellung der Ergebnisse auf der OvGU-Karte . . . . . 65
5.12 Vergleich der Hyperbelkonstanten - OvGU-Karte . . . . . 66

# Tabellenverzeichnis

2.1 Vergleich des Robocup@work mit dem AFiUS-Projekt . . . . . 7

4.1 Ausgewählte Wege der WM-Karte . . . . . 42
4.2 Ausgewählte Wege der OvGU-Karte . . . . . 43
4.3 Eigenschaften der Szenarien . . . . . 45
4.4 Anordnung der Experimente für die Evaluation . . . . . 50
4.5 *Hard-* und *Software* Spezifikationen des Test PC . . . . . 52

5.1 Anzahl der erfolgreichen Planungen bei 100 Versuchen . . . . . 66

# Abkürzungsverzeichnis

| | |
|---|---|
| **AFiUS** | Autonome Fahrräder in Urbanen Szenarien |
| **API** | Application Programming Interface |
| **APS** | Active Pixel Sensor |
| **BIT*** | Batch Informed Trees |
| **CD** | Compact Disk |
| **FMT*** | Fast Marching Tree algorithm |
| **GDB** | GNU Debugger |
| **LazyPRM*** | Lazy Probabilistic Roadmap Star |
| **LBTRRT** | Lower Bound Tree Rapidly-Exploring Random Trees |
| **OMPL** | Open Motion Planning Library |
| **OvGU** | Otto-von-Guericke-Universität |
| **PC** | Personal Computer |
| **Pkw** | Personenkraftwagen |
| **ROS** | Robot Operating System |
| **PRM** | Probabilistic Roadmap |
| **PRM*** | Probabilistic Roadmap Star |
| **RRT** | Rapidly-Exploring Random Trees |

**RRT*** Rapidly-Exploring Random Trees Star

**SPARS** SPArse Roadmap Spanner algorithm

**SPARS2** SPArse Roadmap Spanner algorithm 2

**SST** Sparse Stable Rapidly-Exploring Random Trees

**T-RRT** Transition-based Rapidly-Exploring Random Trees

**UAV** Unmanned Aerial Vehicles

**UML** Unified Modeling Language

**WM** Weltmeisterschaft

# 1 Einleitung

*"I'll be back."*

– T-800*, Autonomer Mobiler Roboter*

Die Einsatzgebiete autonomer Robotersysteme sind in den letzten Jahren exponentiell gewachsen. Schon für das Jahr 2025 wird vorausgesagt, dass jede vierte Tätigkeit in der Industrie von einem Roboter durchgeführt wird [28]. Dabei soll der Markt für mobile autonome Robotersysteme auf weltweit über zehn Milliarden US-Dollar bis zum Jahr 2020 anwachsen [36].

Schon auf Wettbewerben wie dem RoboCup sind die Weiterentwicklungen der mobilen Robotik deutlich spürbar. Dennoch ist Potenzial für weitere Verbesserungen der Hard- und Software erkennbar. Ein essentielles Feld der Forschung an autonomen mobilen Robotern ist die Navigation. Aufgrund ihrer Bedeutsamkeit wurden viele Verfahren entwickelt, einen Pfad zwischen der derzeitigen Position des Roboters und seinem Ziel zu planen. Die Gesamtheit dieser Algorithmen bietet ein breites Spektrum von Ansätzen, die alle mit positiven und negativen Aspekten einher gehen. Bis heute konnte kein Planer entwickelt werden, der alle Vorteile vereint und die Nachteile hinter sich lässt.

Daher ist das Leitmotiv der Forschung die Erschaffung eines Algorithmus, der anhand der Charakteristiken des infrage kommenden Roboters, den Aspekten seiner Aufgabenstellung und den Einflussfaktoren der Umgebung automatisiert, den richtigen Planungsansatz wählt und für die Navigation einsetzt.

Von diesem Motiv ausgehend leitet sich die Motivation dieser Arbeit ab. Der erste Schritt besteht darin, die modernen Planungsalgorithmen zu untersuchen und zu verglichen, um sie in Einsatzbereiche aufzugliedern. Dabei konzentriert sich diese Arbeit auf folgende Fragen: Ist ein Pfadplanungsframework mit Hilfe des weltweit verbreiteten Robot Operating Systems und der umfangreichen Open Motion Planning Library realisierbar, welches für unterschiedliche Roboter, mit und ohne kinematische Einschränkungen, Pfade generieren kann? Und

ist es darüber hinaus möglich, mit diesem Framework die enthaltenen Pfadplanungsalgorithmen zu charakterisieren und Regelsätze aufzustellen, welche die Entwicklung von automatisierten Auswahlprogrammen vorantreiben?

Um diese Fragen zu beantworten, folgt nach der Erörterung der benötigten Grundlagen im ersten Kapitel dieser Arbeit, ein Konzept für ein Pfadplanungswerkzeug, dass auf der Grundlage der genannten Softwaretools entwickelt wird. Es wird auf die nötigen Randbedingungen eingegangen und verschiedene Aspekte gegeneinander abgewogen. Die Realisierung der konzeptionierten Ideen und die Umsetzung von Versuchsanordnungen schließen sich im Kapitel vier daran an. Letztere werden im darauffolgenden fünften Kapitel evaluiert. Die Arbeit schließt mit einer Zusammenfassung der Erkenntnisse und einem Ausblick auf mögliche weiterführende Forschung.

# 2 Grundlagen

In diesem Kapitel werden die für das Konzept und die Umsetzung benötigten Grundlagen beleuchtet und definiert.

Zu Beginn erfolgt eine Erläuterung der zugrundeliegenden Projekte dieser Arbeit, anhand derer Unterschiede und Gemeinsamkeiten aufgezeigt werden, die für das Konzept von Bedeutung sind. Darauf folgt eine generelle Übersicht über die Fortbewegung und Umgebungserfassung mobiler Robotersysteme. Das Kapitel schließt mit der Beschreibung, der für die Umsetzung gebrauchten Robotersysteme.

## 2.1 Projekte

In diesem Abschnitt werden der Wettkampf RoboCup@Work und das universitäre Projekt Autonome Fahrräder in Urbanen Szenarien (AFiUS) vorgestellt. Diese Projekte bilden die Grundlage der Konzeptfindung sowie Umsetzung des zu erstellenden Pfadplanungs-Plug-ins. Die dort zur Anwendung kommenden autonomen Robotersysteme werden anschließend beschrieben.

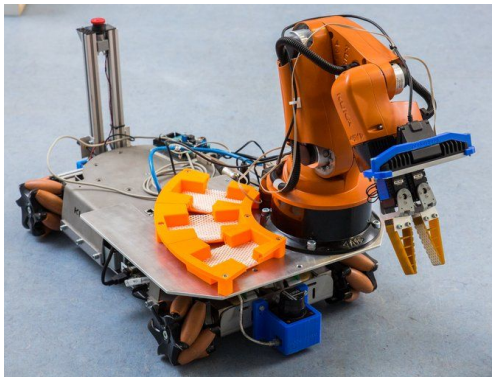


Abb. 2.1: Kuka youBot des Team robOTTO

### 2.1.1 Robocup@Work

"Die RoboCup@Work Liga befasst sich mit der Forschung und Entwicklung für den Einsatz von Robotern (mobilen Manipulatoren) im industriellen Kontext [...]. Roboter sollen künftig komplexe Aufgaben in Zusammenarbeit mit Menschen erfüllen, beispielsweise bei der Fertigung, Automatisierung oder der allgemeinen Logistik. Reale industrielle Herausforderungen sollen die Grundlage bilden für eine robuste mobile Manipulation, die skalierbar, also in weitaus größerem Maßstab einsetzbar, sein soll. Der Roboter soll firm sein in den Disziplinen Navigation, Objekterkennung, Manipulation und Planung." [53]

Für diesen Wettkampf wird vom Team robOTTO der Otto-von-Guericke-Universität (OvGU) Magdeburg ein KUKA youBot verwendet (Abbildung 2.1). Dieser wurde vom Team modifiziert, um den Anforderungen der Problemstellung gerecht zu werden. Dazu gehört zum einen die Erweiterung um zwei Laserscanner, mit denen es dem youBot möglich ist, sich in der Arena zu lokalisieren. Zum anderen wurde an dem Arm eine 3D-Kamera angebracht, die dazu dient, industriellen Objekte zu erkennen und zu unterscheiden.

Abb. 2.2: Eine RoboCup@Work Arena des RoboCup 2015 in Hefei, China

Die Wettkampfarenen, in denen sich der Roboter bewegt, sind mit 40 cm hohen Wänden aus aluminium-umrahmten Sperrholz abgegrenzt, welche über Öffnungen zum Ein- und Ausfahren der Roboter verfügen, wie in Abbildung 2.2 zu erkennen ist. Innerhalb der Arena befinden sich eine Auswahl an Objekten, die Ziele oder Hindernisse darstellen können. Mögliche Ziele der Roboter sind Tische, Fließbänder oder Drehtische, vor denen sich der Roboter platzieren muss, um die darauf befindlichen Gegenstände zu manipulieren. Die Hindernisse können entweder vor Beginn des Wettbewerbs bekannt sein oder erst kurz vor

einem Durchlauf in der Arena platziert werden. Im ersten Fall können die Hindernisse fest in die jeweilige Karte verzeichnet werden und der Roboter kann bereits in seiner Pfadplanung Kollisionen mit diesen vermeiden. Im zweiten Fall hingegen wird der Roboter mit unbekannten Hindernissen konfrontiert und seine Reaktion auf diese beurteilt. Unbekannte dynamische Hindernisse kamen bei den Wettkämpfen bisher nicht vor, sind allerdings für die Zukunft nicht ausgeschlossen [14, S. 5f.].

### 2.1.2 Autonome Fahrräder in Urbanen Szenarien

Beim AFiUS Projekt steht die Automatisierung und Autonomisierung des gesamten Fahrprozesses von Fahrrädern (siehe Abbildung 2.3) im Fokus. In diesem Rahmen soll zu experimentellen Zwecken folgendes Szenario geschaffen werden:

Abb. 2.3: AFiUS Fahrrad

Im Elbauenpark in Magdeburg (siehe Abbildung 2.4) soll es möglich sein, an einer beliebigen Stelle auf den vorgesehenen Wegen, ein Fahrrad anzufordern. Das angeforderte Rad begibt sich darauf hin selbständig von seinem derzeitigen Standort zur Rufstelle. Dort angekommen, kann die anfordernde Person das Fahrrad wie ein herkömmliches Rad mit elektronischer Unterstützung nutzen und es wiederum an jeder beliebigen Stelle im Park aus seinen Diensten entlassen.

Daraufhin wechselt das Fahrrad zurück in den autonomen Betrieb und fährt zum nächsten Aktionsort. Das Fahrrad entscheidet dabei anhand des Akkustands und der Auftragslage, ob es in das Lager, zu einer Ladestation oder zum nächsten Auftragsort fährt.

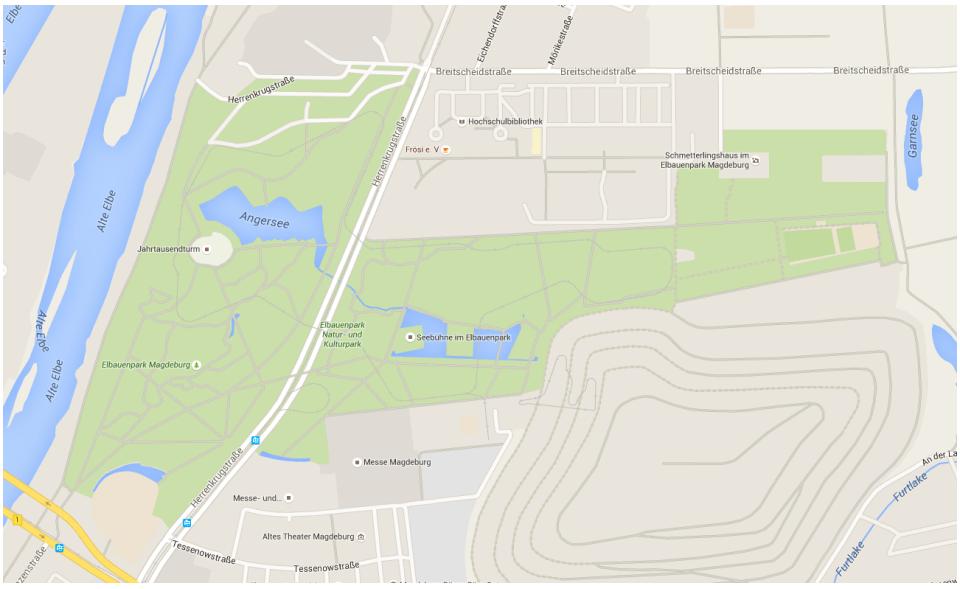



Abb. 2.4: Der Elbauenpark Magdeburg (grün dargestellt) soll als Versuchsgelände für AFiUS genutzt werden. [49]

In Zukunft soll dieses Szenario zeitgleich mit drei Fahrrädern realisiert werden, die sich diese Aufgaben teilen. Dementsprechend ergeben sich verschiedene Teilaufgaben: Die Lokalisierung innerhalb des 90 ha großen Parks, die Navigation zur Rufstelle, die Aufgabenverteilung und die Kommunikation zwischen den verfügbaren Rädern über große Entfernungen hinweg.

Im Hinblick auf die Pfadplanung muss zudem beachtet werden, dass weitere Störfaktoren/ Herausforderungen hinzu kommen. Zum einen stellen Besucher des Parks (Erwachsene, Kinder, Hunde) dynamische Hindernisse dar, welche die Pfadplanung erschweren, da ihre Handlungen, ob absichtlich oder unabsichtlich, nicht vorhersehbar sind. Zum anderen muss sowohl die Planung der Wege über große Distanzen mit der Detailplanung in bestimmten Situationen in Einklang gebracht werden. Ein mögliches Beispiel könnte folgendermaßen aussehen: Ein sich im Lager befindliches Fahrrad wird gerufen. Um das Lager verlassen zu können, muss es an zwei Fahrrädern vorbei navigieren, die zum selben Zeitpunkt in das Lager fahren. Zeitgleich kommt es an dieser Stelle zur

Kommunikation bzw. Koordination zwischen diesen. Anschließend muss das Fahrrad zum mehrere hundert Meter entfernten Ziel fahren. Diese Interaktionen müssen in einem angemessenem zeitlichen Verhältnis stehen, da der Rufende nicht unverhältnismäßig lang auf das Fahrrad warten soll [39].

### 2.1.3 Vergleich

In Tabelle 2.1 werden die beiden Projekte gegenübergestellt.

Tabelle 2.1: Vergleich des Robocup@work mit dem AFiUS-Projekt

| | **Robocup@work** | **AFiUS** |
|---|---|---|
| **Aufgabe** | Lösen industrieller/ logistischer Aufgaben im Rahmen eines Wettkampfes | Bereitstellung eines Personentransportvehikels an beliebigen Stellen des Elbauenparks auf "Zuruf" |
| **Umgebung** | Arenen, bestehend aus Tischen, Wänden mit Toren, Förderbändern und ähnlichen Objekten | Elbauenpark Magdeburg |
| **Kartengröße** | min: 2 × 4 m<br>max: 10 × 12 m | Park West: ca. 400 × 920 m<br>Park Ost: 1120 × 530 m |
| **Maße der Roboter** | 0,7 × 0,4 m | 2,09 × 0,7 m |
| **Art der Karte** | statische Karte mit vorher nicht verzeichneten statischen Hindernissen, dynamische Hindernisse (in Zukunft) möglich | statische Karte mit vielen möglichen statischen und dynamischen Hindernissen |
| **Art der Roboter** | holonome, autonome Roboterplattform mit 4 Rädern | nicht-holonomes, autonomes Fahrrad mit 3 Rädern |
| **Anzahl der Roboter** | eins<br>(in Zukunft: mehr möglich) | eins<br>(in Zukunft: mehr möglich) |
| **Lokalisation** | Laserscanner & Odometrie | GPS, Odometrie & Kamerasystem |

Hierbei fällt besonders auf, dass die Kartengröße des AFiUS Projekts gravierend größer, als die des RoboCup@work ist. Zudem weißt das Fahrrad im Gegensatz zum youBot Einschränkungen in der Bewegungsfreiheit auf, die bei der Navigation bzw. Pfadplanung berücksichtigt werden müssen.

Auch die Umgebungen, in denen sich die Roboter bewegen, sind grundverschieden. Besonders in Hinsicht auf die statischen und dynamischen Hindernisse, welche sich auf den Ablauf der Navigation auswirken.

Die einzige Gemeinsamkeit der beiden Projekte ist das Kriterium der Roboteranzahl, da vorerst nur jeweils ein System betrachtet wird.

Die in der Tabelle genannten Abmessungen der Roboter beziehen sich auf die in der OvGU Magdeburg befindlichen Systeme und können durch Umbauten etc. von Standards abweichen.

## 2.2 Fortbewegung von mobilen Systemen

Mobile Robotersysteme werden in der heutigen Zeit in vielen verschiedenen Aufgabenbereichen eingesetzt. In diesen müssen sie sich, je nach Bauart und Problemstellung, in unterschiedlichen zwei- und dreidimensionalen Umgebungen orientieren und Wege planen. Nicht zuletzt fordern die einzelnen Disziplinen des RoboCup von den mannigfaltigen Robotern, dass sie beispielsweise in schwer begehbarem Gelände nach Lebenszeichen suchen, im Haushalt behilflich sind, Industrieaufgaben übernehmen oder in Fußballturnieren gegeneinander antreten.

Dieser Abschnitt stellt eine Auswahl verschiedenartiger mobiler Roboter vor und kategorisiert diese anhand ihrer Fortbewegungsmechaniken.

### 2.2.1 Roboter mit Rädern

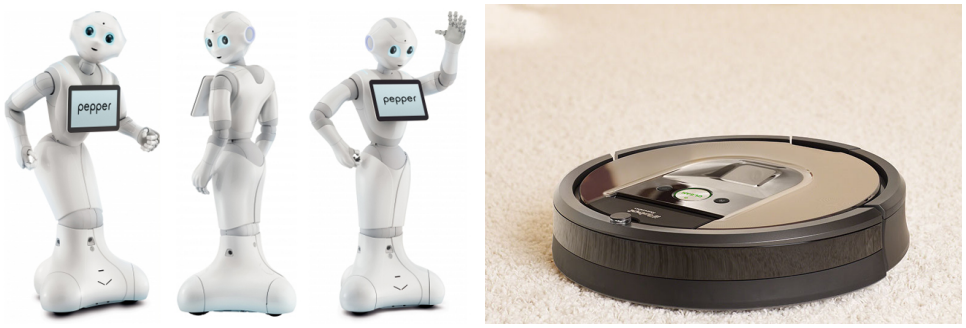
Das Rad ist der namenhafteste von Menschen erfundene Mechanismus zur Fortbewegung. Entgegen des vergleichsweise simplen Aufbaus bietet das Rad eine hohe Effizienz in Bezug auf das Verhältnis von Kraftaufwand zu Geschwindigkeit. Da die meisten radgetriebenen Roboter über drei Räder und somit drei Auflagepunkten verfügen, ist eine Dysbalance ausgeschlossen. [35, S. 32]

Abb. 2.5: Beispiel eines *omni-wheels* [52]

Der Forschungsschwerpunkt liegt hier, bei den eng miteinander verflochtenen Teilgebieten Traktion bzw. Stabilität, Manövrierbarkeit und Kontrollierbarkeit. Ein autonomfahrender Personenkraftwagen (Pkw) bewegt sich lediglich entlang seiner Hauptachse und ist in Kurvenfahrten von der Bewegung in Richtung dieser abhängig. Der Pkw charakterisiert somit eine Art von Plattform mit eingeschränkten kinematischen Eigenschaften. Dem entgegengesetzt verkörpert beispielsweise der Logistikroboter Robotino eine holonome Plattform. Dieser fährt mittels drei sogenannter *omni-wheels*[1](dt: Allseitenrad) (Abbildung 2.5) unabhängig in die zwei Dimensionen der Ebene und ist in der Lage, fahrtrichtungsunabhängig Rotationen durchzuführen. Im Gegensatz zum Auto ist hierfür eine weitaus komplexere Ansteuerung der Räder notwendig. Zudem werden die Seitenführungskräfte von den Onmi-Wheels schlechter aufgenommen [35, S. 36f.].

Weitere Beispiele für radbetriebene Roboter sind unter anderem Staubsauger-roboter und Serviceroboter, welche in Abbildung 2.6 dargestellt sind. In [35, S. 31-45] und [34, S. 391-410] wird auf den Aufbau verschiedener Radtypen, die Auswirkung der Radkonfiguration auf die Freiheitsgrade sowie die bereits erwähnte Stabilität, Manövrierbarkeit und Kontrollierbarkeit eingegangen.

[1]Räder, die zusätzlich zu ihrer Hauptachse in eine weitere Richtung abrollen können [35, S. 32].




(a) Softbank Aldeberan Pepper [55] (b) iRobot Roomba Staubsaugerroboter [48]

Abb. 2.6: Roboter mit Rädern

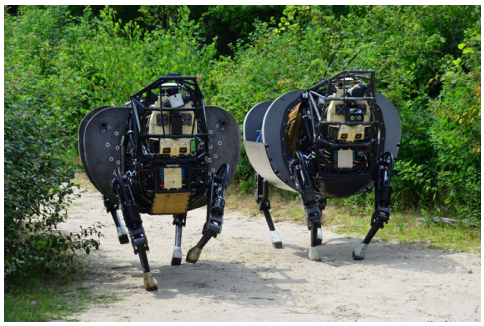
### 2.2.2 Roboter mit Beinen

Eine weitere Art der Fortbewegung bilden die mobilen Roboter mit Beinen. Der Bodenkontakt dieser Roboter wird durch eine Reihe von Punktverbindungen charakterisiert. Die Beschaffenheit des Bodens zwischen den Kontaktpunkten ist somit unwichtig. So können zum Beispiel Löcher im Untergrund oder Stufen je nach Beschaffenheit besser überwunden werden, als von Robotern mit Rädern. Dadurch ist eine Steigerung der Anpassbarkeit und Manövrierbarkeit in unwegsamem Gelände möglich [35, S. 16f.].

Je nach Einsatzmöglichkeit steigt auch die Komplexität des Roboteraufbaus, da die Beine, welche in der Regel mehrere Freiheitsgrade aufweisen, das Gesamtgewicht des Roboters tragen müssen. Dabei ist es von Bedeutung, dass zur Fortbewegung eines oder mehrere der Beine den Bodenkontakt aufheben, ohne dass der Roboter seine Balance verliert. Zusätzlich müssen die Beine in der Lage sein, den Körper auf und ab zu bewegen. Für eine hohe Manövrierbarkeit ist es zudem notwendig, dass die Gliedmaßen Arbeit in mehrere Richtungen verrichten können [34, S. 363f.].

Im Bereich des RoboCup ist der Softbank Aldeberan Nao (Abbildung 2.7a) eine der populärsten Umsetzungen eines zweibeinigen Roboter. Darüber hinaus wird bei Forschungsinstituten wie Bosten Dynamics unter anderem an vier- und mehrbeinigen Robotern geforscht (Abbildung 2.7b). Generell steigt mit der Anzahl der Beine die Stabilität, allerdings ebenfalls die kinematischen Bedingungen des Systems.

(a) Softbank Aldeberan NAO [54]



(b) Bosten Dynamics LS3 - Legged Squad Support Systems [41]

Abb. 2.7: Roboter mit Beinen

In [34, S. 361-386] wird eine Einführung in die Historie der Roboter mit Beinen gegeben sowie die Mechanismen verschiedener mehrbeiniger Roboter erläutert. Des Weiteren wird in [35, S. 16-21] im Allgemeinen auf diese Art von Robotern eingegangen und weitere Beispiele erörtert. Das Paper [31] von Marc Raibert et al. geht auf eine der neueren Entwicklungen von Bosten Dynamics ein.

### 2.2.3 Fliegende Roboter

Eine gänzlich andere Fortbewegungsmöglichkeit verkörpern die fliegende Roboter. Sie werden auch Unmanned Aerial Vehicles (UAV) oder, im Volksmund, Drohnen genannt und repräsentieren derzeit eine der entwicklungsstärksten Roboterbranchen. Der Einsatz fliegender Roboter im Bereich des Militärs spielt, vor allem in den Vereinigten Staaten, eine große Rolle. Dort werden Sie zur Spionage, Aufklärung, Überwachung und schnelle Angriffe eingesetzt. Auch im zivilen Sektor lassen sich immer mehr Anwendungsmöglichkeiten finden, so sprühen diese Roboter beispielsweise im landwirtschaftlichen Bereich Felder, unterstützen bei der Wartung von Pipelines und Gebäuden oder sind in diversen Sicherheitsanwendungen im privaten oder staatlichen Sinne nützlich [29, S. 7f.].

Vorteile dieser unbemannten Systeme sind unter anderem die geringe Baugröße, die sich sich aus der Platzersparnis eines Piloten ergibt. Hinzu kommen die Einsatzmöglichkeiten in verseuchten oder gefährlichen Gebieten. Nicht zuletzt sind auch dies Gründe für den starken Aufschwung dieser Roboterart in den letzten Jahren.

Generell werden die UAV [29, S. 10f.] nach in vier Hauptkategorien unterteilt:

(a) Predator Drohne [29]

(b) Syma X8C Venture Quadrocopter [43]

Abb. 2.8: Fliegende Roboter

**UAV mit statischen Flügeln** sind unbemannte Flugzeuge und benötigen wie diese Rollfelder oder ähnliches zum Starten und Landen. Sie sind meist dazu gedacht, lange Strecken und in großer Höhe zu fliegen. Eins der bekanntesten Systeme dieser Art ist die Predator-Drohne (siehe Abbildung 2.8a).

**UAV mit rotierenden Flügeln** ähneln meist Hubschraubern und haben wie diese die Fähigkeit, senkrecht abzuheben und eine hohe Manövrierbarkeit sowie die Möglichkeit in der Luft zu "stehen". Durch diese Fähigkeiten ist dieser fliegende Roboter im zivilen Bereich sehr praktisch anwendbar und hier am weitesten verbreitet. Es gibt eine Reihe unterschiedlicher Konfigurationen, am bekanntesten ist der Quadrocopter (siehe Abbildung 2.8b).

**Ballons bzw. Zeppeline** sind mit Gas gefüllt, welches leichter als Luft ist. Somit schweben sie ohne aktiven Antrieb, wodurch sie lange in der Luft bleiben können. Sie haben in der Regel keine schnellen Antriebe und sind generell größer als andere Drohnen.

**UAV mit schwingenden Flügeln** haben flexible Flügel, welche für den Antrieb sorgen und sind meist beim Aussehen und Bewegungsablauf von Vögeln und Insekten inspiriert.

Zwischen diesen Kategorien sind auch Hybride möglich oder Systeme, die einen Mechanismus zum Starten bzw. Landen benutzen und einen zum Fliegen.

### 2.2.4 Weitere Roboter

Neben den in den vorhergehenden Abschnitten genannten Robotersystemen gibt es viele weitere Roboterarten. Der Anzahl der Einsatz- und Forschungsgebiete für autonome und teilautonome mobile Roboter ist keine Obergrenze gesetzt. Es wird an schwimmenden [26], tauchenden [5] und springenden [3] Systemen geforscht. Roboter werden genutzt, um Tunnel zu graben [17] und bei Operationen zu assistieren [21].

(a) Salamander Roboter [15]

(b) Boston Dynamics Handle [47]

Abb. 2.9: Hybride Roboter

Auch hybride Systeme, die sich zwischen den genannten Kategorien bewegen, sind in der Industrie und Forschung beliebt. Es gibt beispielsweise an Amphibien orientierte Roboter zum Einsatz im Wasser und an Land [15]. Außerdem wurden Roboter entwickelt, die sich dank einer Kombination aus Rädern und Beinen sehr schnell fortbewegen und unebenes Terrain ausgleichen können [12] (siehe Abbildung 2.9).

Der youBot, der für diese Arbeit genutzt wird, stellt ebenfalls eine solche Kombination dar. Er besteht aus einer mobilen Plattform mit einem Fünf-Achs-Arm. Daher sind bei seinen Bewegungsabläufen sowohl die drei Freiheitsgrade der Ebene als auch die sechs Freiheitsgrade im Raum des Arms zu berücksichtigen. Bei ersteren ist er nicht eingeschränkt, da seine Basis holonom ist, Letztere allerdings kann er nur durch die Bewegung der Basis erreichen, da sein Arm alleine mindestens eine bewegliche Achse zu wenig hat. Ganzheitliche Bewegungen des Roboters erfordern durch diese Art des Aufbaus eine komplexe

Planung der Bewegungsabläufe. Dies wiederum verlangt dem Roboter eine hohe Rechenleistung ab. Daher hat sich das Team robOTTO dafür entschieden, Bewegungen der mobilen Plattform und des Arms sequenziell oder getrennt voneinander zu planen und auszuführen.

Diese Arbeit behandelt die Planung von Pfaden in der Ebene. Dazu muss der Roboter sich in seiner Umwelt per Sensoren zurechtfinden und eine Kartendarstellung besitzen, die seine Umgebung ausreichend abbildet. Diese Themen werden im Abschnitt 2.3 erläutert.

## 2.3 Umgebungserfassung von Robotern

In diesem Abschnitt werden zunächst die verschiedenen Arten von Sensoren erläutert, die es Robotersystemen ermöglichen sich zu lokalisieren. Anschließend werden die verhaltens- und kartenbasierte Navigation sowie verschiedene Kartendarstellungen erläutert.

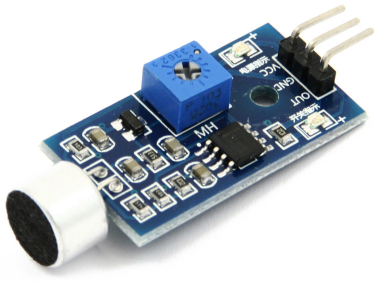
### 2.3.1 Sensoren

Für bewegliche autonome Robotersysteme ist es von Bedeutung, Information über ihre Umgebung aufzunehmen und sich in ihr zu lokalisieren. Um dies zu realisieren nehmen die Sensoren spezifische Informationen der Umgebung auf und geben diese an den Roboter zur Auswertung weiter.

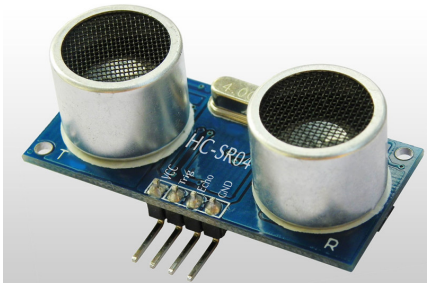
In [35, S. 89f.] werden Sensoren in zwei Kategorien eingeteilt, exterozeptiv und propriozeptiv. Zudem findet in den Kategorien jeweils eine Unterteilung in aktive bzw. passive Sensoren statt. Nachfolgend werden alle vier Varianten vorgestellt.

**Propriozeptive Sensoren** nehmen Werte des Systeminneren auf. Zu diesen zählen unter anderem Drehzahl- bzw. Geschwindigkeitsmesser, Winkelmesser, Odometrie, Beschleunigungs- und Lagesensoren.

**Exterozeptive Sensoren** dokumentieren die Umgebung des Roboters. Beispielhaft hierfür sind Entfernungsmesser, Lichtstärkensensor und Lautstärkenmesser.




(a) Beispiel eines Ultraschallsensors [51]




(b) Beispiel eines Geräuschsensors / Mikrophones [56]

Abb. 2.10: Beispiele für aktive und passive Sensoren

**Passive Sensoren** zeichnen sich dadurch aus, dass sie Umgebungszustände aufnehmen und in elektrische Signale umwandeln. Vertreter sind beispielsweise Thermometer, Mikrofone (Abbildung 2.10a) und Active Pixel Sensor (APS) Kameras.

**Aktive Sensoren** wirken im Gegensatz zu passiven Sensoren aktiv auf ihre Umgebung ein und messen die Reaktion auf diese. Meist liefern ihre Messungen vergleichsweise bessere Ergebnisse in Bezug auf die Genauigkeit und den Datenumfang als passive Sensoren. Allerdings besteht bei ihnen das Risiko, durch die Messung die zu messende Größe zu beeinflussen. Zudem ist es möglich, dass ähnliche Signale, beispielsweise von anderen aktiven Sensoren in der Nähe, die eigene Messung stören. Beispiele für aktive Sensoren sind unter anderem Ultraschallsensoren (Abbildung 2.10b, Infrarot- , Laserreichweitenmesser.

Alle Sensoren haben gemeinsam, dass ihre Werte mit Unsicherheiten, Messrauschen und Treppeneffekten belegt sind. Ein GPS-Sender ist beispielsweise durch die Reflexion von Gebäudewänden innerhalb einer Stadt nur auf mehrere Meter genau. Diese Genauigkeit genügt für ein Auto, welches sich ausschließlich auf der Straße bewegt, ist jedoch für ein Robotersystem mit der maximalen Größe eines Menschen ungenügend. Zudem sind GPS-Sensoren nur unter freiem Himmel anwendbar.

Sensoren müssen immer nach den Einsatzanforderungen des jeweiligen Robotersystems gewählt werden, sodass der Roboter zum einen seine Aufgabe mit gültigen Information nachgehen kann und zum anderen keine Störungen, wie beispielsweise ungewollte Kollisionen mit der Umgebung, aufweist.

### 2.3.2 Navigation und Kartendarstellungen

Wie die nötigen Informationen über die Situation des Roboters abstrahiert und in einer Kartendarstellung abgespeichert werden, wird in diesem Abschnitt erörtert. Dazu wird zunächst die Navigation in zwei Arten untergliedert.

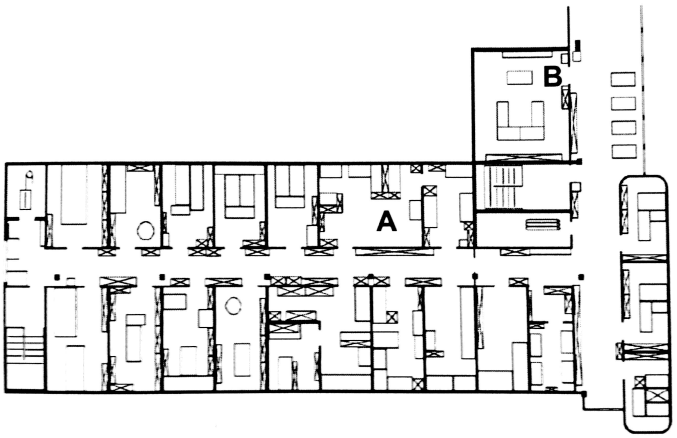



Abb. 2.11: Beispielumgebung für verhaltensbasierte Navigation [35, S. 192]

Neben der kartenbasierten Navigation, bei der eine geometrische Karte verwendet wird, ist es in einigen Situation möglich, eine verhaltensbasierte Navigation zu nutzen, um ein bestimmtes Ziel zu erreichen. Abbildung 2.11 zeigt eine Umgebung, in der ein fiktiver Roboter von Raum A zu Raum B fahren soll. An dieser Stelle ist eine verhaltensbasierte Navigation möglich, bei der das Robotersystem lediglich der linken oder rechten Wand folgen, Hindernissen ausweichen und erkennen muss, wann es sich in Raum B befindet. In Abbildung 2.12 werden die Abläufe beider Arten der Navigation dargestellt. Die Vorteile verhaltensbasierter Navigation sind, dass eine Lokalisation des Roboters in seiner Umgebung nicht nötig ist, da er solange weiter macht bis er am Ziel ist und dass durch Wegfallen der Planung Rechenzeit gespart wird. Natürlich kann dieses unüberlegte Vorgehen dazu führen, dass die Ausführung wesentlich länger dauert, da kein optimaler Weg gefahren wird [35, S. 191f.].

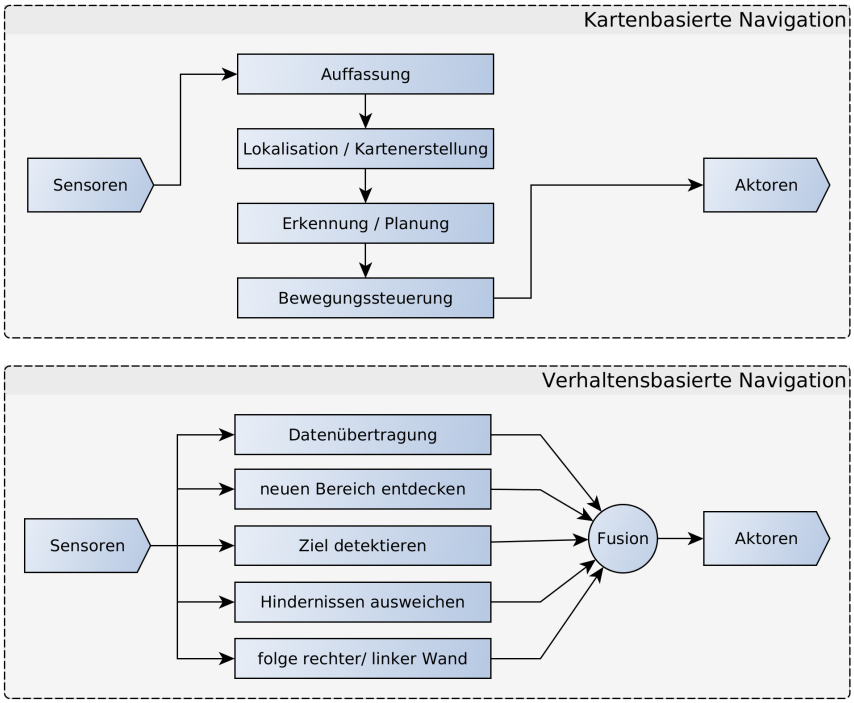



Abb. 2.12: Unterschied verhalten-/ kartenbasierte Navigation, nach [35, S. 193]

Bei kartenbasierter Navigation wird eine Form der namensgebenden Karte angelegt und/oder genutzt. In dieser muss sich der Roboter lokalisieren und Wege zwischen der derzeitiger Position und dem Ziel planen. Der Vorteil einer Karte ist, dass sie auch von einem Menschen gelesen werden kann und somit ein Mittel der Mensch-Maschine-Kommunikation darstellt. So ist es für den Anwender möglich, dem Roboter für neue, von ihm unerforschte Gebiete eine Karte zu übergeben, in der sich der Roboter zurecht findet. Allerdings muss berücksichtigt werden, dass die Karte eine Abstraktion der Umgebung darstellt und das Robotersystem sich somit mehr auf das Modell verlässt, als auf die wirkliche Realität [35, S. 193f.].

Die in Abschnitt 2.3.1 genannte Unsicherheit der Sensoren wirkt sich direkt auf die Lokalisation aus. Daher ist die Pose des Roboters in der Karte immer mit Ungenauigkeiten belastet. Bei einer bekannten Startposition und eindeutigen Landmarken in der Umgebung des Roboters, kann die Wahrscheinlichkeit, die absolute Position des Roboters zu kennen, erhöht werden. Demzufolge wird hier meist mit nur einer Annahme über die Pose des Roboters gearbeitet. Bei einer unbekannten Startposition eines *kidnapped robot* (dt. entführter Roboter) Problems muss davon ausgegangen werden, dass der Roboter sich an einer von

mehreren Positionen auf der Karte befindet. Aus diesem Grund muss die Lokalisation dies berücksichtigen, um durch Auswertung der Sensorinformationen die wirkliche Position zu bestimmen. [35, S. 195-199]

In [38, S. 157 ff.] werden verschiedene Lokalisierungsverfahren genauer erläutert und in [22, S. 452 ff.] wird auf den Umgang mit Unsicherheiten eingegangen.

Folgende fundamentale Fragen der kartenbasierten Navigation müssen bei der Gestaltung der Karte beantwortet werden:

- Welche Informationen müssen in der Karte abgebildet werden?
- Woher kommen die Informationen?
- Welche Genauigkeit bzw. Auflösung muss die Karte besitzen?
- Wie werden Hindernisse eingetragen, die der Roboter zwar überwinden kann, dafür allerdings sein Verhalten anpassen muss (klettern oder springen statt laufen)?
- Wie werden Objekte dargestellt, mit denen interagiert wird oder welche vermieden werden sollen?
- Wie viele Dimensionen muss die Karte abbilden?

Nach [35, S. 200] gibt es drei grundlegende Kriterien, die bei der Auswahl einer Kartendarstellung berücksichtigt werden müssen.

1. Die Genauigkeit der Karte muss der Genauigkeit entsprechen, mit der der Roboter sein Ziel erreichen muss.
2. Die Feinheiten der Eigenschaften der Karte und der darin enthaltenen Objekte muss der Genauigkeit der Sensoren entsprechen.
3. Die Komplexität der Kartendarstellung hat einen direkten Einfluss auf die Rechenleistung, die gebraucht wird, um die Karte abzubilden bzw. sich in ihr zu lokalisieren und zu planen.

Im Folgenden sollen einige wichtige Kartendarstellungen beschrieben werden:

#### Kontinuierliche Darstellung

Bei der kontinuierlichen Darstellung wird die Umgebung exakt abgebildet. Hierfür werden die Positionen aller Objekte präzise in die Karte eingetragen. Diese

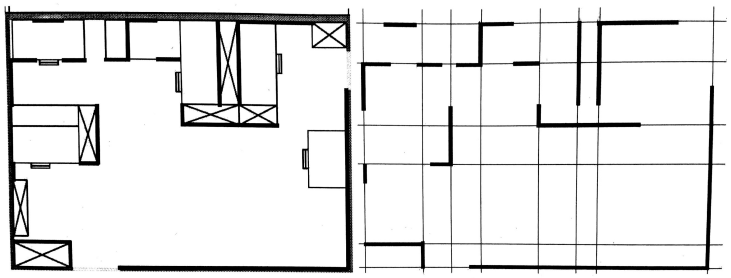


Abb. 2.13: Beispiel für kontinuierliche Darstellung mit allen Objekten (links) und daraus reduzierter Liniendarstellung (rechts) [35, S. 202]

Darstellung wird meist nur in zweidimensionalen Karten genutzt, da weitere Dimensionen den Verarbeitungsaufwand exorbitant vergrößern würden. Aus diesem Grund wird die kontinuierliche Darstellung meist nur in geschlossenen Umgebungen genutzt, bei denen die Größe des Raums sowie die Anzahl und Position aller Objekte bekannt ist. Dies begründet sich auch darin, dass der Speicherplatzbedarf von der Komplexität der in der Karte befindlichen Objekte abhängt und somit der Nachteil dieser Art der Darstellung ist.[35, S. 200]

Um das Speichervolumen und die benötigte Performance zu senken, kann die Darstellung auf die für die Sensoren wichtigen Merkmale reduziert werden. Wenn ein Laserscanner eingesetzt wird, um den Roboter zu lokalisieren, können die umliegenden Objekte beispielsweise auf ihre Außenlinien reduziert werden (Liniendarstellung). Dieser Sachverhalt ist in Abbildung 2.13 dargestellt. [35, S. 201]

Der Vorteil der kontinuierlichen Darstellung liegt in der Möglichkeit, das Robotersystem sehr präzise in der Umgebung zu bestimmen, sowohl bei einer Einals auch bei einer Mehr-Posen-Hypothese.[35, S. 202]

### Dekompositions Darstellung

Die Dekompositions Darstellung bildet die zweite Art der Kartendarstellungen. Diese wird auch Zelldekomposition (eng.: *cell decomposition*) genannt. Sie basiert auf der Zerlegung des freien Arbeitsraumes des Roboters in kleine, einfach zu beschreibende Regionen, den sogenannten Zellen. Eine Unterteilung wird solange durchgeführt, bis der Roboter innerhalb einer Zelle jede Konfiguration ungehindert einnehmen kann. Danach wird der *connectivity graph* (dt. Verbindungsgraph) erstellt, der die Nachbarschaftsbeziehungen zwischen den

Zellen beschreibt. Die Knoten des Graphen repräsentieren die freien Zellen und werden nur miteinander verbunden, wenn zwei nebeneinander liegen. Auf diese Weise entsteht ein freier, kontinuierlicher Pfad [35, S. 203].

Die Zelldekomposition gliedert sich in drei Unterkategorien: die exakte, die adaptiv approximierte und die fest approximierte Zelldekomposition, auf die nachfolgend eingegangen wird.

**Exakte Zelldekomposition** teilt den freien Raum in Zellen ein, die diesem exakt[2] entsprechen. Die Grenzen der einzelnen Zellen werden hier anhand bestimmter Kriterien gezogen, beispielsweise wenn die Umgebung einen Einfluss auf die Bewegung des Roboters hat [35, S. 204].

In Abbildung 2.14 wird eine exakte Zelldekomposition gezeigt. Hier ist der freie Raum von einem Polygon umgeben und es befinden sich drei polygonale Hindernisse in ihm. Der freie Raum wird in Zellen der Form von Dreiecken und Trapezen eingeteilt.

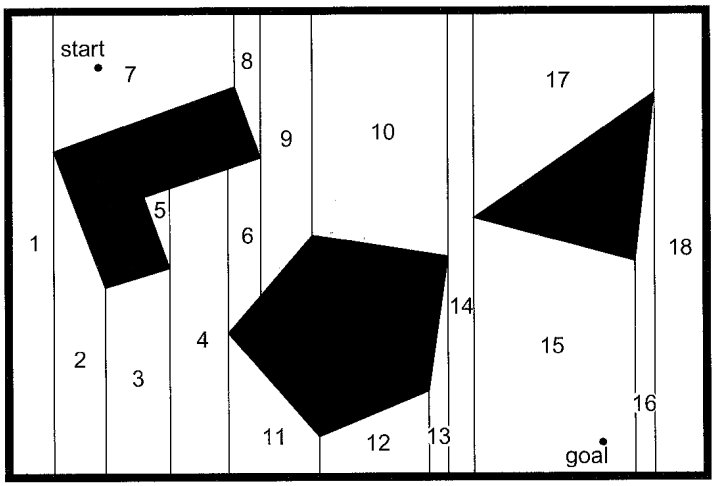



Abb. 2.14: Einteilung einer Karte in exakte Zellen. [35, S. 204]

**Adaptiv approximierte Zelldekomposition** separiert den freien Raum in Zellen, deren Form vorgegeben ist, beispielsweise Quadrate. Die Übergänge sind hierbei von dem Roboter unabhängig und haben für die physikalische Gegebenheit keine Bedeutung.

[2] "exakt" bezieht sich in diesem Fall auf die gegebene Karte, nicht auf die reale Welt, welche nur approximiert werden kann.

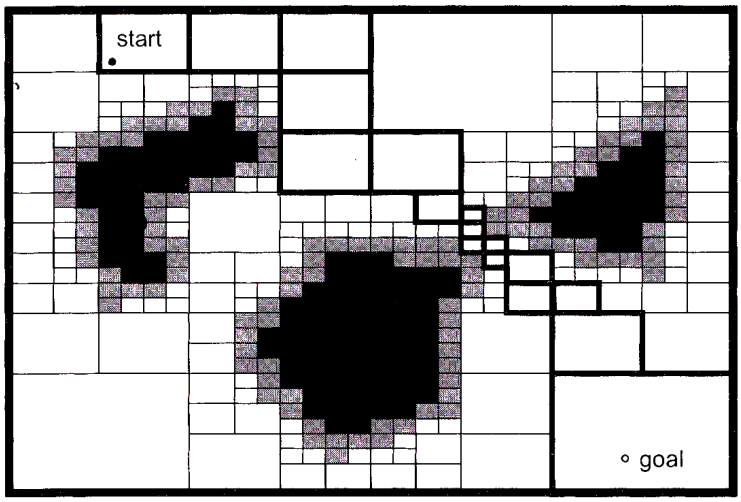



Abb. 2.15: Einteilung einer Karte in approximierte Zellen. [35, S. 206]

Die Abbildung 2.15 zeigt die approximierte Zelldekomposition innerhalb des Arbeitsraums. Der Raum wird rekursiv in kleinere Rechtecke unterteilt, wodurch bei jeder Zerlegung vier neue identische Rechtecke entstehen.[3] Wenn sich in den Zellen sowohl Hindernisse als auch freier Raum befinden, wird die Zerlegung solange vorgeführt, bis an den Rändern von Hindernissen eine feinere Auflösung entsteht. Ab einem gewissen Feinheitsgrad wird die Dekomposition gestoppt und der *connectivity graph* von der Start- zur Zielzelle verbunden (dick-schwarz umrandete Kästchen). Ist dieses erfolgreich, wird ein entsprechender Pfad erstellt. Andernfalls ist entweder die Auflösung des Verfahrens nicht fein genug oder es existiert kein Pfad. [35, S. 205]

Bei manchen Methoden der Zelldekomposition wechseln sich die Suche mit der Zerlegung der Zellen ab: immer wenn die Verbindung des *connectivity graph* fehlschlägt, wird die Zerlegung der Zellen verfeinert und erneut gesucht. [22, S. 14 ff.]

**Fest approximierte Zelldekomposition** ähnelt der adaptiven Variante mit dem Unterschied, dass keine rekursive Zerteilung stattfindet. Stattdessen wird die Karte mit einer vordefinierten Zellgröße zerlegt. Die verbreitetste Umsetzung ist die sogenannte *occupancy grid map* (Abbildung 2.19a). Die einzelnen Zellen können hier die Werte *frei*, *belegt* oder *unbekannt*

[3] Dieses spezielle Verfahren wird *quadtree* genannt, da ein "Suchbaum" entsteht, bei dem von jedem Ast vier neue Äste abgehen.

annehmen. Jedoch besteht die Gefahr, dass bei einer zu groben Auflösung kleinere Durchgänge und enge Passagen zwischen Objekten als *belegt* gekennzeichnet werden. [35, S. 207]

Trotz dieses Umstandes sind *occupancy grid maps* weit verbreitet und werden auch im ROS *Navigation Stack* für den *map_server* benutzt, da die Umsetzung einfach und der Speicherbedarf der Karte fest ist. Diese Karte bildet die Grundlage für die *costmap*, welche in Abschnitt 2.4.2 erläutert wird.

## 2.4 The Robot Operating System

Robot Operating System (ROS) ist ein flexibles Softwaregerüst für Roboter, welches 2007 im Rahmen des Standford-AI-Robot-Projektes am Stanford Artificial Intelligence Laboratory entwickelt wurde. Bis heute wird es, in erster Linie durch das Robotikinstitut Willow Garage, weiterentwickelt. Die in dieser Arbeit genutzte Version trägt den Namen *Indigo Igloo* und ist damit die achte Distributionsversion von ROS, welche am 22. Juli 2014 veröffentlicht wurde. [42]

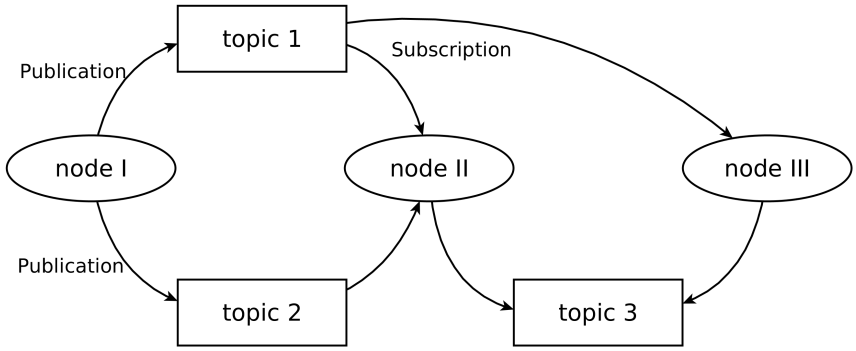



Abb. 2.16: Beispielhaftes Schema des *publisher- /subscriber*-Prinzip im ROS *node-graph*

ROS ist eine Sammlung von Werkzeugen, Bibliotheken und Richtlinien, deren Ziel es ist, die Entwicklung von komplexer und robuster Robotersoftware zu vereinfachen und dabei eine große Palette verschiedener Roboter abzudecken. Es wurde darauf geachtet, dass die Software die Möglichkeit bietet, dass mehrere Personen bzw. Teams mit ähnlichen Projekten Fortschritte untereinander austauschen können. Aus diesem Grund besitzt ROS eine modulare Struktur.

Aufgerufenen Programmen ist es beispielsweise möglich über ein *publisher-/subscriber*-Prinzip zu kommunizieren, welches in ROS über einen *node graph* dargestellt wird (Abbildung 2.16) [44].

Programme werden als *node* dargestellt und reagieren nur auf die *topics*, die sie auslesen sollen. Gleichzeitig geben sie selbst Informationen durch *topics* unter vorgeschriebenen Namen aus. Das bedeutet, dass *nodes* jederzeit ausgetauscht werden können, solange sie sowohl zum *publishen* (dt.: herausgeben) als auch zum *subscriben* (dt.: abonnieren) die selben Namen und Typen verwenden wie der ausgetauschte *node*.

### 2.4.1 ROS Navigation Stack

Der *Navigation Stack* bildet eine Gruppe von ROS Programmen, welche eigens für die Navigation von Robotern in der Ebene entwickelt wurden. Abbildung 2.17 zeigt einen schematischen Aufbau des *Stacks*, bei dem die *move_base node* eine zentrale Rolle spielt. Sie besteht aus funktionalen Einheiten, die Informationen von außen benötigen, wie zum Beispiel die Karte des *map_servers* und die Sensordaten. Anhand dieser Informationen und der Zielvorgabe *move_base_simple/goal* werden die Geschwindigkeiten *cmd_vel* für die Motor-Controller des Roboters errechnet.

Die einzelnen *nodes* innerhalb der *move_base* sind sogenannte Plug-ins, welche im Abschnitt 2.4.3 erläutert werden. Im nächsten Abschnitt werden die *costmaps* beschrieben, welche für die Navigation benötigt werden.

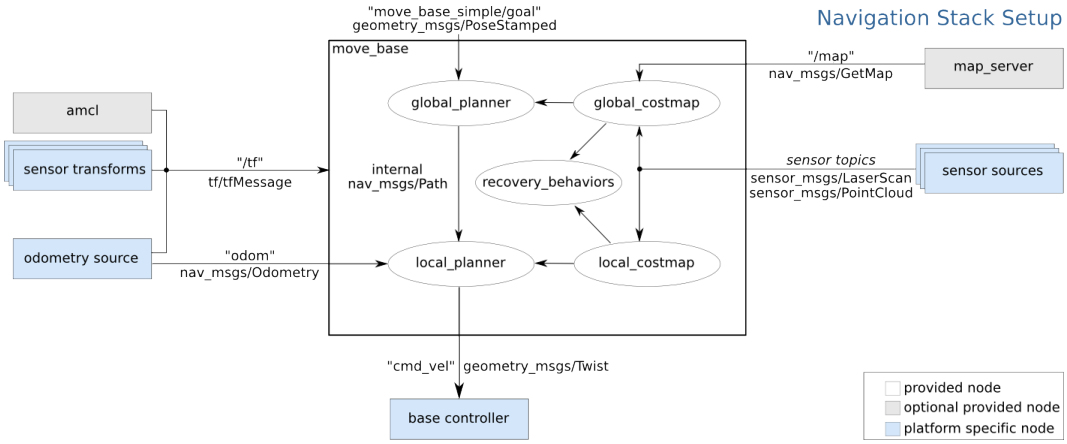



Abb. 2.17: Schematische Abbildung der move_base [27]

### 2.4.2 Costmap

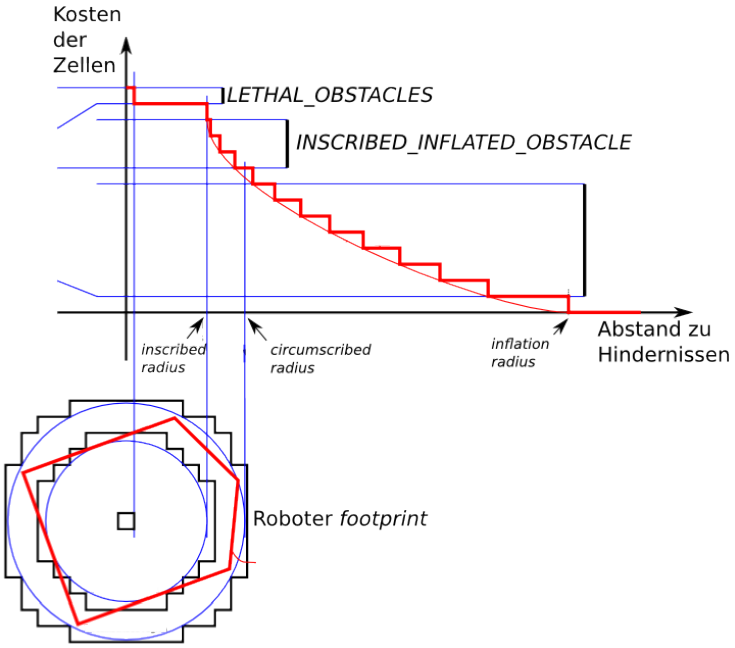
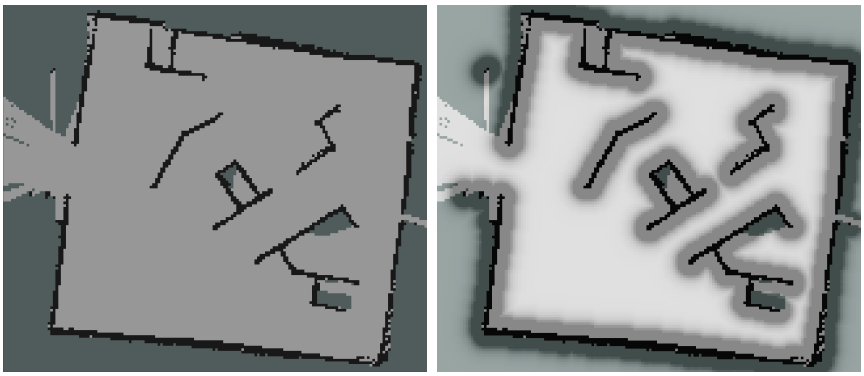



Abb. 2.18: Inflation Funktion der *costmap* [13]

Allgemein ist eine *costmap* die Zuordnung von Kosten zu jedem Punkt in einer Karte. Dadurch wird eine Wertung über die Erreichbarkeit einer Stelle in der Karte abgegeben, wobei Punkte mit kleineren Kosten zugänglicher sind. Ziel ist es, durch die Summe der Kosten aller im Roboterpfad befindlichen Punkte, eine Aussage über die Pfadgüte zu treffen und somit verschiedene geplante Wege miteinander vergleichen zu können.

In ROS können entsprechende Sensordaten in die *costmap* einfließen, um erkannte Hindernisse als Kosten darzustellen. Gleichzeitig ist es möglich, eine statische Karte mit bekannten Hindernissen zur Verfügung zu stellen. Diese Karte ist das in Abschnitt 2.3.2 unter fest approximierte Zelldekomposition beschriebene *occupancy grid map* (Abbildung 2.19a). Jedem Punkt der kontinuierlichen Ebene wird mittels der Karte und ihrer entsprechenden Auflösung in $[m/px]$ eine Zelle bzw. Pixel zugewiesen. Bei einer Abfrage eines Punktes ist die entsprechende Zelle entweder *frei* oder *belegt*.

In Abbildung 2.18 wird gezeigt, wie die *costmap* aufgebaut ist. Die Kosten in jedem Punkt werden durch einen 8 Bit Integer dargestellt und können somit zwischen null und 255 liegen. Dabei werden sogenannten "LETHAL_OBSTACLES",



(a) Beispiel einer *occupancy grid map*

(b) Beispiel einer *costmap*

Abb. 2.19: *Occupancy grid map* und darauf aufgebaute *costmap*

also den erkannten bzw. eingetragenen Hindernissen, der Wert 254 zugewiesen. Alle Hindernisse werden um den Wert des inneren Radius des Roboterpolygons erweitert. Dem entstandenen Bereich wird der Wert 253 zugewiesen ("INSCRIBED_INFLATED_OBSTACLE"). Außerhalb dieses Bereiches nehmen die Kosten nach der folgenden Funktion exponentiell ab:

$$c = e^{-1.0 \cdot k \cdot (d-r)} \cdot (c_i - 1), \tag{2.1}$$

- wobei $c$ die Kosten der betrachteten Zelle sind,
- $k$ ein Skalierungsfaktor ist, der frei gewählt werden kann,
- $d$ ist der Abstand zum nächsten Hindernis,
- $r$ ist der innere Radius des Roboterpolygons und
- $c_i$ ist der oben genannte Wert "INSCRIBED_INFLATED_OBSTACLE", welcher 253 entspricht.

Die Kosten werden durch diese Funktion bis zu einem festgelegten Radius, dem "inflation_radius", berechnet. Außerhalb dieses Radius erhalten alle Punkte bzw. Zellen den Kostenwert null. Unbekannten Stellen in der Karte wird der Wert 255 zugewiesen ("NO_INFORMATION") [13].

### 2.4.3 Plug-ins im ROS-Kontext

Plug-ins unterstreichen die flexible Modularität von ROS. Sie sind wie *nodes* einzelne Programme und verhalten sich analog zu diesen. Während der Entwicklung von Programmen muss nur die Deklaration der Funktionen feststehen, die das Plug-in später bereitstellt. Die Definition der Funktionen kann nachträglich erfolgen.

Dazu wird eine Klasse angelegt, die als Interface zwischen dem Plug-in und dem Programm dient. Dabei stellt ersteres eine gewisse Funktion zur Verfügung, welche von letzterem genutzt wird. Das Programm instanziiert ein Objekt von dieser Klasse und das Plug-in muss somit von dieser Interfaceklasse erben. Diese wiederum legt durch abstrakte Funktionen fest, welche Parameter dem Plug-in übergeben und welche Rückgaben von ihm erwartet werden. Somit ist das Programm über das Interface von den Plug-ins entkoppelt und diese sind frei austauschbar.

Im Beispiel der *move_base* und dem globalen Planer Plug-in sind zwei abstrakte Funktionen in der *BaseGlobalPlaner*-Klasse festgelegt. Die erste dient der Initialisierung und wird beim Start des Programms aufgerufen. Sie bekommt die aktuelle *costmap* übergeben und hat keine Rückgabewerte. Die zweite Funktion stellt die eigentliche Planungsfunktion dar, welche die Start- und Zielpose bekommt und den fertigen Plan zur Weiterverarbeitung durch den lokalen Planer zurück geben muss. In Abbildung 2.20 ist das mit zwei bereits im ROS *Navigation Stack* vorhandenen Planer Plug-ins dargestellt.

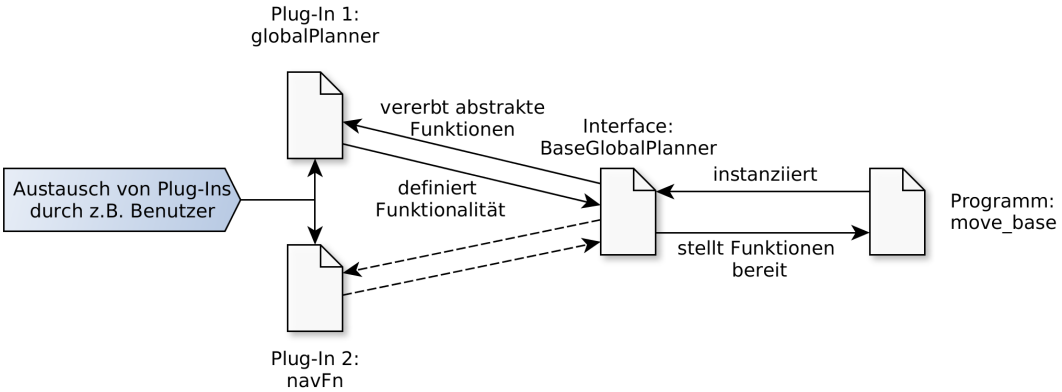



Abb. 2.20: Beispiel für Plug-ins anhand der *move_base*

Die genaue Umsetzung des globalen Planer Plug-ins für diese Arbeit wird in Abschnitt 4.3 erläutert. Zunächst wird die für das Plug-in benötigte Pfadplanungsbibliothek beschrieben.

## 2.5 The Open Motion Planning Library

Die Open Motion Planning Library (OMPL) ist ein universelles Pfadplanungswerkzeug. Es ist 2012 im *Kavraki Lab* der Rice University in Houston, Texas entstanden. OMPL wurde nicht für einen speziellen Roboter entwickelt, sondern soll Entwicklern bei der Umsetzung von Pfadplanungen verschiedener Roboter unterstützen. Daher wird sie unter der BSD-Lizenz kostenfrei zur Verfügung gestellt. Sie enthält die Grundkonzepte von Pfadplanung, wie Zustandsräume, verschiedene Planer und Interfaces für Kollisionsabfrage etc.. Der Grund für dieses Design ist, Robotersysteme mit beliebigen Randbedingungen, wie beispielsweise beliebiger Anzahl von Achsen, abbilden zu können. Unabhängig davon, ob es sich um eine Plattform mit Rädern, Beinen oder einen festen Arm handelt. Letzterer wurde bereits durch das *MoveIT!* Paket, welches OMPL beinhaltet, in ROS umgesetzt.[37]

In dieser Arbeit wird die Version 1.2.1 genutzt, welche am 01. Juli 2016 veröffentlicht und von den Entwicklern als stabil gekennzeichnet wurde.

Zunächst sollen in den folgenden Abschnitten einige der grundlegenden Aspekte der Pfadplanung und wie sie in der OMPL umgesetzt sind erläutert werden.

### 2.5.1 Konfigurationsräume

Konfigurationsräume werden auch Zustandsräume oder *statespaces* genannt. Sie beschreiben die Zustände aller Gelenke und Aktoren eines Roboters. Anders als beim Arbeitsraum, welcher alle Punkte umfasst, die der Roboter physisch erreichen kann, muss es sich beim Konfigurationsraum nicht zwangsläufig um einen geometrischen Raum handeln. Die Anzahl der Dimensionen entspricht der Anzahl der Aktoren. Beispielsweise hat ein 5-Achs-Roboterarm einen fünfdimensionalen Zustandsraum. Diese Arbeit konzentriert sich auf die Navigation in der Ebene, bei der drei Dimensionen benötigt werden. Jeweils eine für die Translationen in der $x$-$y$-Ebene und eine für die Rotation $\theta$.

Weiterhin ist zwischen dem gesamten Konfigurationsraum $C$ und dem für den Roboter zu erreichenden $C_{free}$ zu unterscheiden. Eine Pfadplanung muss innerhalb des freien Raums einen Weg suchen, ohne dass Punkte oder Übergänge zwischen diesen im blockierten Teil von $C$ liegen. Des Weiteren können bestimmte Bedingungen die Übergänge zwischen Posen in $C_{free}$ weiter einschränken, falls dies nötig ist. Dem AFiUS Fahrrad ist es beispielsweise nicht möglich, sich quer zu seiner Hauptachse in der Ebene zu bewegen, da es zu den in Abschnitt 2.2.1 beschriebenen Plattformen mit kinematischen Einschränkungen gehört.

Um diese Unterscheide zu berücksichtigen, bietet die OMPL mehrere vorgefertigte Zustandsräume für die typischen Pfadplanungsszenarien an. Sollten diese nicht dem betrachteten Szenario entsprechen, ist es möglich, weitere hinzuzufügen. Für die Anwendungsfälle dieser Arbeit sind allerdings schon die entsprechenden Konfigurationsräume vorhanden.

Um die holonome Basis des youBots abzubilden wird der *SE2 statespace* und für das AFiUS Fahrrad der *Dubins* und der *ReedsShepp statespace* verwendet, die nachfolgend definiert werden.

**SE2 statespace** besteht aus den Dimensionen $x$, $y$ und $\theta$. Da keine Einschränkungen bestehen, ist dieser Konfigurationsraum für die Pfadplanung in der Ebene für holonome Roboter geeignet. Die Gesamtlänge des Pfades wird von der OMPL wie folgt berechnet:

$$l(x, y, \theta) = \sum_{n=1}^{m} \sqrt{(x_n - x_{n-1})^2 + (y_n - y_{n-1})^2 + 0.5(\theta_n - \theta_{n-1})^2}, \quad (2.2)$$

wobei $m$ die Anzahl von Posen im Pfad ist, $x$ und $y$ die Einheit $[m]$ und $\theta$ die Einheit $[rad]$ haben. Damit ist die Gesamtlänge nicht allein von der Pfadlänge der beiden Translationen $x$ und $y$ abhängig, sondern auch von der Rotation. Sie trifft somit keine Aussage über die geometrische Länge des Pfades, sondern ist einheitenloses Bewertungskriterium.

**Dubins- und ReedsShepp statespace** bauen auf dem *SE2* Konfigurationsraum auf. Sie benutzen die gleichen Dimensionen und die gleiche Bewertung der Gesamtlänge. Der Unterschied ist, dass es Planern innerhalb dieses Raumes nur möglich ist, Dubins- bzw. Reeds-Shepp-Pfade zu planen.

Dubins Pfade setzen sich lediglich aus geraden Pfaden und Kurven auf einem festgelegten Kreisradius zusammen, wie in Abbildung 2.21 zusehen

ist. Außerdem wird angenommen, dass sich das mobile System ausschließlich vorwärts bewegt. Kann es sich zudem rückwärts bewegen, handelt es sich um einen Reeds-Shepp-Pfad. [32]

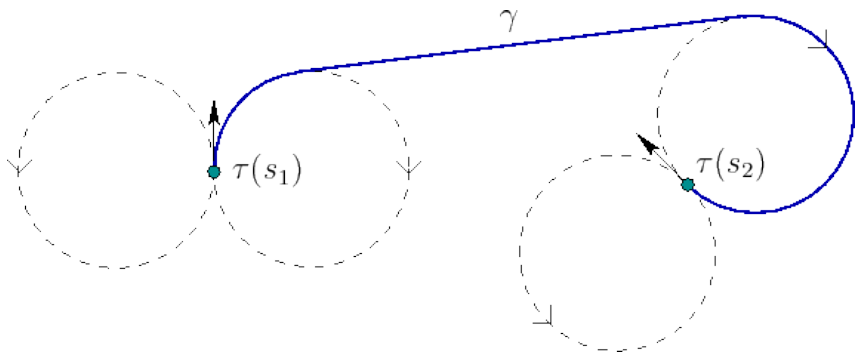



Abb. 2.21: Dubins Pfad, $\tau(s_1)$ und $\tau(s_2)$ sind jeweils Rechtskurven mit dem gleichen Radius, $\gamma$ ist der Gerade Pfad, der die beiden verbindet.[40]

### 2.5.2 Pfadplaner der OMPL

In diesem Abschnitt werden Pfadplanungsalgorithmen vorgestellt, die in der OMPL enthalten sind. Alle Planer gehören entweder den *tree-based* oder den *roadmap-based* Algorithmen an (Abbildung 2.22). Die Grün eingefärbten Planer sind optimierende Planer. Ihre Besonderheit besteht darin, dass sie auf ein Optimalitätskriterium hinarbeiten, während die anderen Planer ihre Berechnung einstellen, sobald sie einen gültigen Pfad gefunden haben. Standardmäßig ist dies das Kriterium, den kürzesten Pfad zu finden. In den genannten geometrischen Konfigurationsräumen wird versucht, das Ergebnis der Formel 2.2 zu minimieren bis die Höchstdauer für Planungen, welche frei einstellbar ist, erreicht ist. Andere mögliche Kriterien wären das Maximieren des Abstandes zu Hindernissen oder das Unterschreiten einer definierten Pfadlänge. Auch gewichtete Kombinationen aus diesen sind möglich.

Zunächst werden der Rapidly-Exploring Random Trees (RRT) und Probabilistic Roadmap (PRM) Planer im Detail erklärt, da die ihnen zugrundeliegenden Algorithmen die Basis für die weiteren Planer der OMPL bilden.

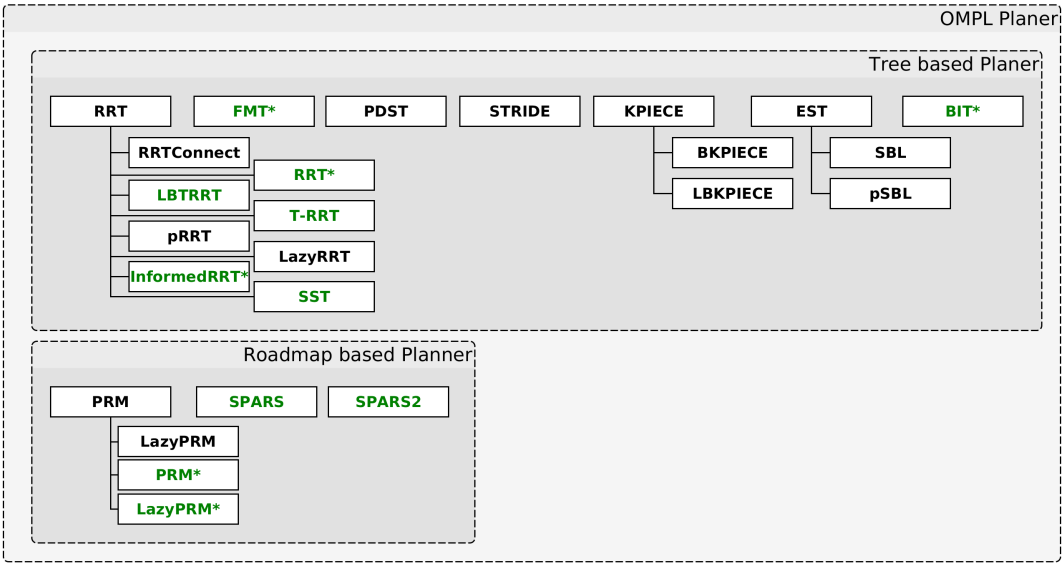



Abb. 2.22: Planer der OMPL - Eingeordnet in die Kategorien *tree-based* und *roadmap based*. Die Hierarchie entspricht [46].

### Rapidly-Exploring Random Trees (RRT) Planer

Der RRT wurde im Jahr 1998 von Steven M. Lavalle eingeführt. Zu dieser Zeit waren bereits die auf Zufall basierenden Planungsverfahren *randomized potential fild algorithm* [2] und *probabilistic roadmap algorithm* [1] bekannt. Diese sind allerdings nicht oder nur mit Anpassungen für nicht-holomome Probleme ausgelegt. Daher hat Lavalle einen weiteren zufallsbedingten Algorithmus entwickelt, welcher sowohl holonome als auch nicht-holomome Planungsprobleme löst.

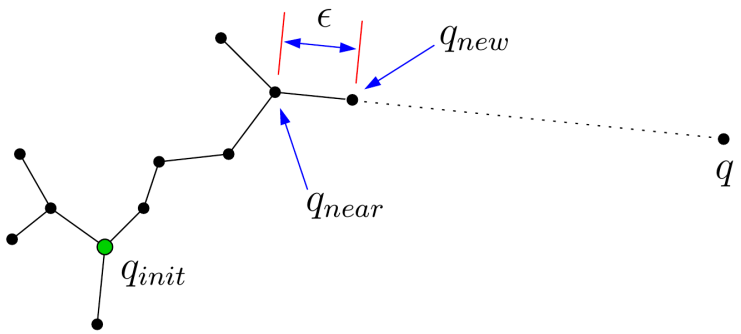



Abb. 2.23: Aufbau eines Random Tree im 2D-Raum [20]

Dabei wird im Konfigurationsraum $C$ ein kontinuierlicher Weg zwischen einer Startpose $q_{init}$ und einer Zielpose $q_{goal}$ gesucht. Hierbei wird generell angenommen, dass es sich bei $C$ um einen geometrischen 2D- oder 3D-Raum handelt[4]. Der Ablauf der Planung ist in Abbildung 2.23 dargestellt. Anfangs wird von $q_{init}$ aus eine neue zufällige Pose $q_{new}$ hinzugefügt. Es wird überprüft, ob $q_{new} \in C_{free}$ liegt, wobei $C_{free}$, wie in Abschnitt 2.5.1 erwähnt, für den Teil des Zustandsraumes steht, in dem sich keine Hindernisse befinden. Ist dies der Fall, wird $q_{new}$ dem *tree* hinzugefügt, wodurch dieser "wächst". Sollte dies nicht der Fall sein, wird eine neue Pose von $q_{init}$ aus gesucht. Dieser Vorgang wird solange wiederholt, bis ein Pfad zwischen $q_{init}$ und $q$ besteht. [23]

### Probabilistic Roadmap (PRM) Planer

Die Entwickler der OMPL verweisen bei ihrem Ansatz der PRM auf die Veröffentlichung [19]. Hier wird in Abschnitt 3 der Ablauf der Planung erläutert. Sie umfasst zwei Phasen, die Lern- und die Überprüfungsphase.

Die Lernphase wiederum ist in die Schritte Konstruktion und Expansion unterteilt, welche nacheinander ausgeführt werden. Im ersten Schritt wird ein leerer Graph $R = (N, E)$ angelegt. Dann wird per Zufall eine Pose $c$ generiert und überprüft, ob sie im freien Konfigurationsraum $C_{free}$ liegt. Ist dies der Fall, wird sie in die Liste $N$ aufgenommen. Dieser Vorgang wiederholt sich, wobei ständig überprüft wird, ob die jeweils neue Pose $c$ mit den bereits vorhandenen Posen $n$ kollisionsfrei verbunden werden kann. Ist dies möglich, so wird die Verbindung $(c, n)$ in $E$ eingetragen und nach weiteren Verbindungen gesucht, bis entweder alle Verbindungen innerhalb eines bestimmten Abstandes zu $c$ gefunden wurden, eine bestimmte Anzahl an Verbindungen zu $c$ gefunden wurde oder beides. So wächst $R$ an und füllt $C_{free}$ nach und nach aus. Im Expansionsschritt wird $R$ erweitert, indem weitere Posen in der Nähe bereits getesteter Posen in $N$ erstellt und überprüft werden. Dies wird solange wiederholt, bis jede der ursprünglichen Posen eine festgelegte Menge Nachbarn hat. Nach diesem Vorgang bedeckt $R$ einen großen Teil von $C_{free}$.

In der Überprüfungsphase kommen die Start- und Zielposen $q_{init}$ und $q_{goal}$ hinzu. Wie in Abbildung 2.24 dargestellt, werden $q_{init}$ und $q_{goal}$ mit den nächstliegendsten Posen,$q'$ und $q''$, im Graph $R$ verbunden. Dabei muss diese Verbindung

[4] Andere, nicht geometrische, Zustandsräume sind durchaus möglich. Diese sind für die Pfadplanung der zu betrachtenden Roboter allerdings zu vernachlässigen.

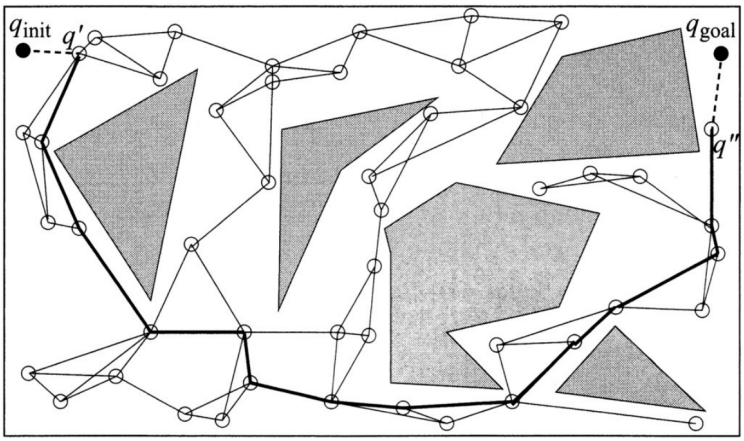



Abb. 2.24: Beispiel einer *Probabilistic Roadmap* [6]

kollisionsfrei sein, ansonsten wird versucht eine Verbindung mit der jeweils zweit nächsten Pose aufzubauen etc.. War dies erfolgreich, ist der Weg durch den Graphen aufgrund der Verbindungen untereinander bekannt. [19]

Sollte keine gültige Verbindung zwischen Roadmap und Start- bzw. Zielposen gefunden werden, kann sich die Karte durch eine weitere Lernphase erweitern. Generell ist es möglich, die Überprüfungsphase mit einem kleinen Graphen anzufangen und nur bei Bedarf zu erweitern.

### Optimierende Planer

Für die Evaluation dieser Arbeit werden die in Abbildung 2.22 grün markierten, optimierenden Planer verwendet. Da die übrigen Planer lediglich einen Pfad von Start- zu Zielpose finden, unabhängig von der Länge des Pfades, sind sie weder für das RoboCup- noch für das AFiUS-Szenario geeignet.

Die optimierenden Planer werden kurz vorgestellt:

**Probabilistic Roadmap Star (PRM*) Planer** Während der PRM-Planer die Lernphase nach einer bestimmten Anzahl von Posen im Graph beendet und zur Überprüfungsphase übergeht, versucht der PRM*-Planer weitere Posen zu finden, um einen optimalen Pfad zu generieren [19].

**Lazy Probabilistic Roadmap Star (LazyPRM*) Planer** Dieser Planer verfolgt die gleiche Strategie wie der PRM*-Planer, versucht allerdings die Anzahl an Kollisionsprüfungen zu minimieren [4].

**Rapidly-Exploring Random Trees Star (RRT*) Planer** Durch stetige Wiederholung des RRT Algorithmus und Vergleich der sich ergebenden Pfade, gelingt es dem RRT*-Planer zu einem optimalen Pfad zu konvergieren [18].

**Informed RRT* Planer** Beim Informed RRT*-Planer wird der Suchbaum aufgeteilt. In diesen Unterbäumen wird durch Hinzufügen neuer Pfade und stutzen des vorhandenen Graphen nach einer besseren Lösung gesucht [10].

**Batch Informed Trees (BIT*) Planer** Durch die Verbindung eines gerichteten, optimalen Suchalgorithmus wie A* [25] und eines zufallsbasierten Planers wie RRT kann der BIT*-Planer die Vorteile beider Welten nutzen, um schnellst möglich günstige Wege zu finden [11].

**Lower Bound Tree Rapidly-Exploring Random Trees (LBTRRT) Planer** Dieser Planer bietet nach [33] günstigere Wege als der RRT und schnellere Laufzeiten als der RRT* Planer. Dabei wird neben dem Hauptgraphen im Suchbaum ein Nebengraph, der *lower-bound graph*, für die Suche genutzt.

**Sparse Stable Rapidly-Exploring Random Trees (SST) Planer** Der SST-Planer baut ebenfalls auf dem RRT Algorithmus auf. Er versucht während der Suche nach einem optimalen Pfad, den Suchbaum klein zu halten, indem er überflüssige Posen und Verbindungen verwirft [24].

**Transition-based Rapidly-Exploring Random Trees (T-RRT) Planer** Die Umsetzung des T-RRT-Planer in der OMPL versucht nicht sich genau an ein Optimalitätskriterium zu halten, sondern die Planungszeit zu reduzieren [7] [46].

**SPArse Roadmap Spanner algorithm (SPARS) Planer** Analog zum PRM-Planer berechnen auch diese Algorithmen einen Graphen, um anschließend in diesem nach einem gültigen Pfad zu suchen. Allerdings verlassen sie die Lernphase früher mit weniger Posen und Verbindungen [9] [8].

**Fast Marching Tree algorithm (FMT*) Planer** Dieser Planer wurde speziell für Konfigurationsräume mit vielen Dimensionen entwickelt und nutzt Algortihmen, die sowohl *tree-based* als auch *roadmap-based* sind [16].

# 3 Konzept

Wie in der Motivation bereits angekündigt, wird in der vorliegenden Arbeit eine Möglichkeit gesucht, die Planer der OMPL zu testen und Regelsätze für die Pfadplanung aufzustellen. Dies ist von grundlegender Bedeutung, da dies den ersten Schritt in Richtung einer autonomen Planerauswahl darstellt. Nur durch die Konzipierung geeigneter Tests wird es möglich, konkrete Aussagen zur Eignung der Planer unter verschiedenen Bedingungen zu tätigen.

Dazu werden in diesem Kapitel zunächst Überlegungen zur Umsetzung einer Schnittstelle zwischen den *Softwareframewoks* ROS und OMPL angestellt. Anschließend folgt eine Vorbetrachtung der Szenarien und der Vergleichbarkeit der Planer.

Bei einer vollständigen Neukonzeptionierung der Navigation unter ROS würden die in Abbildungen 2.17 dargestellten Softwarebausteine neu entwickelt werden müssen. Dazu gehören neben der eigentlichen Pfadplanung, unter anderem die Aufnahme und Verarbeitung von Sensorinformationen, die Abstraktion einer Karte und die Lokalisation in dieser. Dies würde den zeitlichen Rahmen dieser Arbeit übersteigen. Eine naheliegendere und effizientere Lösung ist die Verwendung des ROS zur Verfügung gestellten *Navigation Stack*, welcher über Schnittstellen zur Einbindung von Plug-ins verfügt. Somit wird es möglich, Drittsoftware, wie die OMPL, in ROS einzusetzen.

Nachfolgend wird ein Entwurf eines solchen Plug-ins konzeptioniert.

## 3.1 Plug-in Entwurf

Der *Navigation Stack* und seine Komponente *move_base*, welche in Abschnitt 2.4.1 erläutert werden, sind für eine Umsetzung neuer Planungskonzepte geeignet.

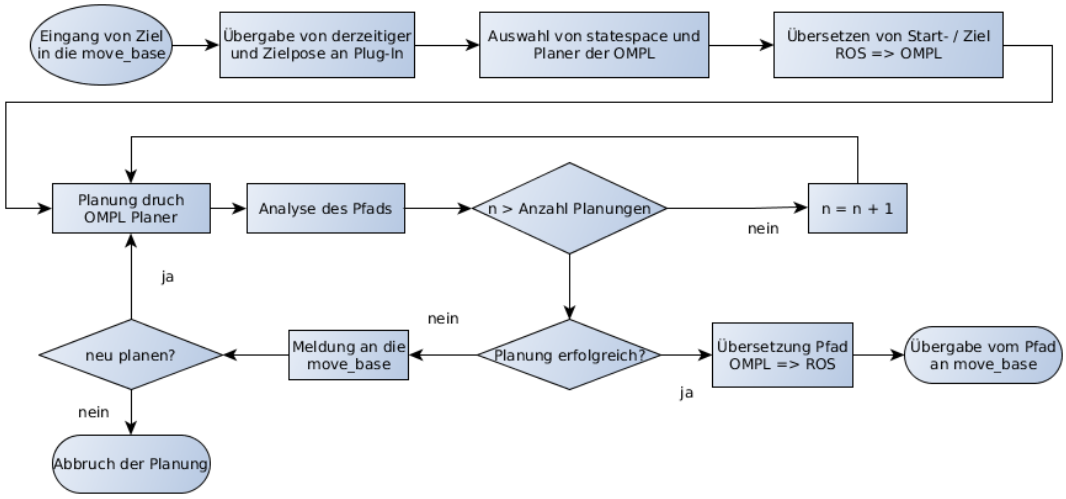



Abb. 3.1: Konzept für die Kombination aus OMPL und ROS

In Abbildung 3.1 wird ein Flussdiagramm für eine mögliche Umsetzung eines OMPL-basierten Plug-ins für ROS gezeigt.

Dabei muss auf die Funktionsweise und die Dateitypen von beiden Systemen eingegangen werden. Ein Plug-in muss die ROS eigenen Posen übersetzen können, damit es möglich ist, sie den Planern der OMPL zu übergeben. Analog dazu muss der geplante Weg in das entsprechende ROS-Format gebracht werden, damit es der *move_base* möglich ist, diesen weiter zu verarbeiten.

Zudem ist es nötig, ausgewählte Variablen zu definieren, wie zum Beispiel die Planer sowie die verschiedenen Konfigurationsräume, in denen sie eingesetzt werden oder die Anzahl an gewünschten Planungen. Diese Einstellungen müssen einfach zugänglich sein, um sie schnell und möglicherweise in Zukunft automatisiert anzupassen.

Für die Analyse der entstandenen Pfade müssen die entsprechenden Daten sowohl während der Planung erhoben als auch danach ausgelesen werden können. Es ist realisierbar, das Plug-in durch einen weiteren *node* überwachen zu lassen, welches die Zeit zwischen Start und Abschluss einer Planung aufnimmt und die entstandenen Pläne abruft. Allerdings bieten die Klassen der OMPL bereits fertige Methoden, die diese Funktionalitäten umsetzen. Daher wäre es logisch, die entsprechenden Funktionen direkt im Plug-in aufzurufen und die Daten zur Analyse in externe Dateien zu speichern.

Des Weiteren ist es notwendig, die Objekte der OMPL zu initialisieren, die für eine Pfadplanung benötigt werden. Die Abbildung 3.2 ist [45] nachempfunden

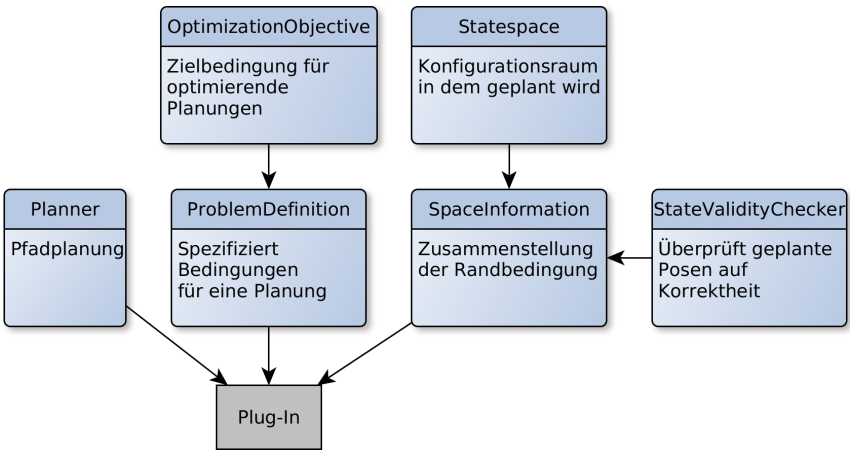



Abb. 3.2: Darstellung der OMPL API

und stellt diese Objekte dar. Wobei die OMPL über keine eigene Implementierung einer Kollisionsabfrage verfügt. Daher muss diese den Bedürfnissen der Aufgabe entsprechend gestaltet werden.

Der trivialste Ansatz wäre die Überprüfung des inneren oder äußeren Radius des Roboters auf Kollision mittels der *costmap*. Allerdings würden in beiden Fällen falsch-negative bzw. falsch-positive Ergebnisse generiert, welche entweder dazu führen, dass der Roboter Pfade nicht plant, die möglich wären, oder in einem Zusammenstoß enden. Daher soll hier der sogenannte *footprint*[1] mit Hilfe der vom Planer gewählten Orientierungen überprüft werden, um zu ermitteln, ob es sich um eine gültige Pose handelt.

Steht die Gestaltung des Plug-ins fest, müssen nun Überlegungen zur Auswahl der Szenarien getroffen werden, mit Hilfe derer die Planner getestet werden sollen.

## 3.2 Auswahl der Szenarien

Hier gäbe es die Möglichkeit, analog zu [30] abstrakte Karten wie zum Beispiel Labyrinthe zu generieren und Punkte sowie Orientierungen zufällig festzulegen, um Posen für die Planung zu erstellen. Allerdings ist vor dem Hintergrund der zugrundeliegenden Projekte für diese Arbeit die Wahl von realistischen Szenarien naheliegender. Somit entfällt auch die Suche nach zufälligen Posen

[1] Der Umriss des Roboters auf der Karte.

in der Karte, da beispielsweise die Aufgaben des RoboCups Posen vorgeben, in denen sich der Roboter befinden muss.

Daher sollen auf Grundlage des Vergleiches der Projekte in Abschnitt 2.1.3 Szenarien gefunden werden, die den jeweiligen Anforderungen entsprechen. Dabei ist zu bedenken, dass sich beide Projekte in Größe der Umgebung, Manövrierbarkeit der mobilen Systeme und Zeitkontingent der Aufgaben sehr voneinander unterscheiden.

Nach der Festlegung geeigneter Szenarien müssen Überlegungen zum Vergleich der Planer angestellt werden.

## 3.3 Vergleich der Planer

Ein Vergleich der OMPL-Planer mit den vorhandenen Planer Plug-ins von ROS scheint naheliegend. In dieser Arbeit wird allerdings aus folgenden Gründen bewusst darauf verzichtet.

Die Planer von ROS basieren auf deterministischen Suchalgorithmen, welche in den zwei Dimensionen der Ebene nach einem Weg zwischen Start und Ziel suchen. Die Orientierung wird außer acht gelassen. Derweil haben die probabilistischen Ansätze der OMPL durch Einbeziehung der Rotation einen weitaus größeren Suchraum.

Die Rotation während der Navigation wird von den ROS-Planer auf eine nachgeschaltete Komponente ausgelagert, während die Pläne der OMPL die Rotation bereits vorgeben.

Da die Gestalt des Roboters bei Planungen mit den ROS-eigenen Algorithmen auf einen Kreis mit festem Radius reduziert wird und darüber hinaus die Hindernisse um diesen Radius erweitert werden, ist es den Algorithmen nur möglich für einfache Robotergeometrien einen idealen Weg zu finden. Im Gegensatz dazu lässt die OMPL durchdachte Überprüfungsstrategien zu, die es in Verbindung mit der Orientierung ermöglichen, Wege für komplexe Robotergeometrien zu generieren.

Unter den genannten Bedingungen finden die deterministischen Suchalgorithmen von ROS für eine Kombination von Posen immer den gleichen Pfad nach der gleichen Zeit. Die OMPL-Planer versuchen hingegen durch das zufällige

Hinzufügen von Posen einen Pfad zwischen Start und Ziel zu erstellen. Über eine Vielzahl von Iterationsschritten und Vergleichen der Pläne nähern sich diese einem Optimum an, welches mit der zur Verfügung stehenden Zeit korreliert.

Zudem sind die ROS-Planungsalgorithmen nicht in der Lage, Pfade für nichtholonome mobile Systeme zu planen, während dies der OMPL, mit den entsprechend vorgefertigten Konfigurationsräumen, möglich ist.

Aus den genannten Gründen wird sich diese Arbeit auf den Vergleich der Planungsalgorithmen innerhalb der OMPL konzentrieren. Dazu müssen Kriterien und Experimente erarbeitet werden, die eine Gegenüberstellung der Planer ermöglicht.

Aufgrund der Anzahl von Planern, Roboterkonfigurationen, mögliche Szenarien und Planungsposen ergibt sich eine unüberschaubare Menge an Kombinationsmöglichkeiten für Tests. Daher ist es nötig, diese Bedingungen im Vorfeld einzuschränken. Die Tests sollen vorrangig im Kontext des youBot-Szenarien stattfinden, da bei einer omnidirektionalen Plattform im Industriekontext die Generalisierbarkeit besser umsetzbar ist.

# 4 Umsetzung

Die im letzten Kapitel erarbeitet Konzepte sollen in diesem Kapitel umgesetzt werden.

Dazu werden zunächst die entworfenen Szenarien vorgestellt. Darauf aufbauend folgt die Anforderungsanalyse, welche die wichtigsten Bewertungskriterien für den Vergleich und die Bewertung der Planer liefert. Anschließend folgt die Erläuterung des eigens für die ROS-*move_base* entworfenen Plug-ins. Die Erläuterung der Versuchsanordnung für die Evaluation schließt das Kapitel ab.

## 4.1 Testszenarien

In diesem Abschnitt werden die beiden Szenarien beschrieben, mit denen das Plug-in und die Pfadplanungsalgorithmen der OMPL getestet werden. Diese sind dem jeweiligen Aufgabenbereich der zugrundeliegenden Projekte nachempfunden.

### 4.1.1 RoboCup@Work Weltmeisterschaft 2017

In diesem Szenario muss der youBot verschiedene Tische anfahren. Die Karte in Abbildung 4.1 wurde während der Weltmeisterschaft (WM) des RoboCup@Work 2017 in Leipzig vom youBot mittels seinen Laserscannern aufgezeichnet. Auf ihr sind die Posen vor den Tischen sowie Anfangs- und Endpose der Arena mit grünen Pfeilen markiert. Es ergeben sich insgesamt 306 mögliche Kombinationen aus Start- und Zielposen. Von diesen werden die in Tabelle 4.1 für die Planungen repräsentativ ausgewählt.

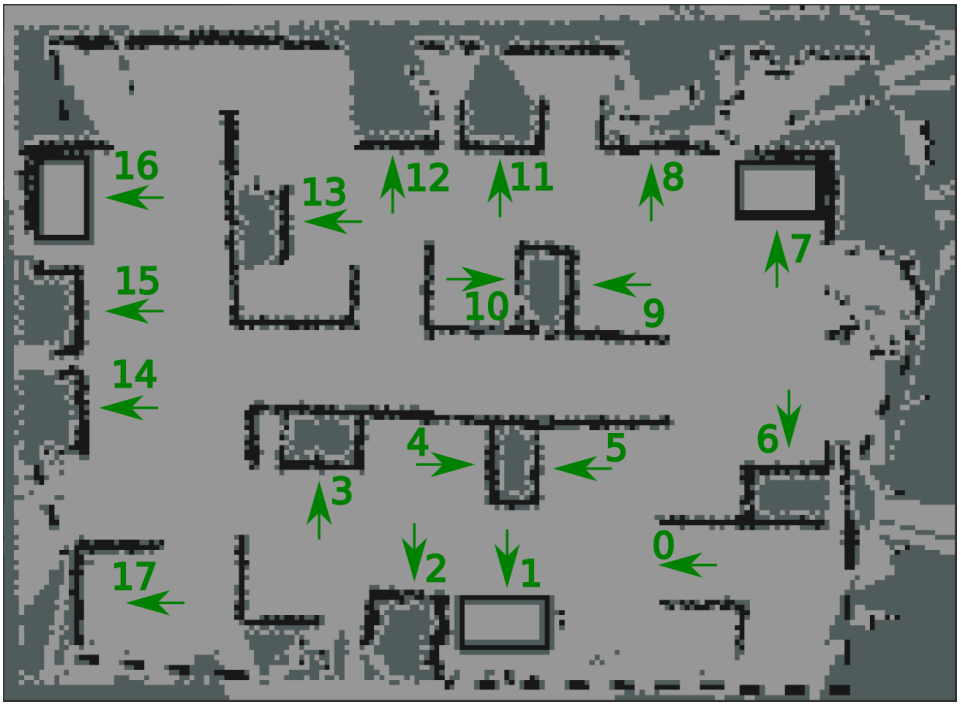



Abb. 4.1: WM-Karte - *Occupancy grid* der Robocup@work Weltmeisterschaft 2017

Tabelle 4.1: Ausgewählte Wege der WM-Karte

| **Start** | **Ziel** | **Begründung** |
|---|---|---|
| 01 | 02 | Minimale Gesamtlänge |
| 08 | 01 | Durchschnittliche Gesamtlänge |
| 17 | 07 | Maximale Gesamtlänge |
| 02 | 13 | Maximale Gesamtwinkeländerung |
| 14 | 15 | Minimale Gesamtwinkeländerung |
| 09 | 17 | Maximale Weglänge |
| 12 | 13 | Minimale Weglänge |
| 06 | 08 | Verhältnis von Länge [m] zu Rotation [rad] $\approx 1$ |

### 4.1.2 Campus der Otto-von-Guericke-Universität

Das zweiten Szenario stellt den östlichen Teil des Campus der Otto-von-Guericke-Universität Magdeburg dar. Das AFiUS Fahrrad wird zu bestimmten Gebäuden beordert und soll sich entsprechend fahrbereit ausrichten. Die gewünschten Posen sind in der Abbildung 4.2b ebenfalls mit grünen Pfeilen markiert. Die Karte wird anhand eines Satellitenbildes erstellt, welches in Abbildung 4.2a zusehen ist. Die Gebäude und unbefahrbaren Gelände wie Straßen und diverse Grünflächen werden in der *occupancy grid map* (Abschnitt 2.3.2) durch belegte Zellen ersetzt oder von diesen umschlossen.

Auch hier wird eine Vorauswahl der Start- und Zielposen anhand der in der Tabelle 4.2 ersichtlichen Kriterien getroffen.

Tabelle 4.2: Ausgewählte Wege der OvGU-Karte

| **Start** | **Ziel** | **Begründung** |
|---|---|---|
| 00 | 07 | Maximale Gesamtlänge |
| 03 | 06 | Durchschnittliche Gesamtlänge |
| 05 | 01 | Durchschnittlicher Abstand zu Hindernissen minimal |
| 06 | 02 | Maximale Gesamtwinkeländerung |

Abschließend werden die Eigenschaften der beiden Szenarien in Tabelle 4.3 zusammengefasst. Es ist zu erkennen, dass die Karte des OvGU-Szenarien wesentlich größer und gröber aufgelöst ist.

## 4.2 Anforderungsanalyse

Um einen Vergleich der Pfadplanungsalgorithmen zu ermöglichen, werden in diesem Abschnitt Anforderungen beschrieben, die von den Planern bestmöglich erfüllt werden müssen. Dabei leiten sich die Anforderungen von den angesprochenen Einsatzszenarien ab.

**Kollisionsfreiheit** Dieses Kriterium muss jeder geplante Pfad erfüllen, um als gültig gewertet zu werden. Ein Pfad, der bereits zum Zeitpunkt der Planung für eine ungewollte Berührung mit den statischen Hindernissen der Karte führt, ist unbrauchbar. Daher darf der minimale Abstand des Pfades den inneren Radius des Roboters nicht unterschreiten.


(a) Satellitenaufnahme[50]


(b) OvGU-Karte - *Occupancy grid*

Abb. 4.2: Teil des Campus der OvGU Magdeburg

Tabelle 4.3: Eigenschaften der Szenarien

| Eigenschaft | WM-Szenario | OvGU-Szenario |
|---|---|---|
| Breite | 190 px | 1642 px |
| Höhe | 139 px | 974 px |
| Auflösung | 0.05m/px | 0.2174 m/px |
| Roboter | youBot | AFiUS Fahrrad |
| Statespace | SE2 | Dubins & ReedsShepp |
| Kurvenradius | - | 1,5 m |
| Verhältnis unbefahrbar/gesamt | 30% | 54% |

**Exakter und vollständiger Pfad** Die Planer müssen einen nachvollziehbaren Weg zwischen Start- und Zielpose planen. Dabei ist es erforderlich, dass die geplanten Posen den Anforderungen der Manövrierbarkeit des jeweiligen Konfigurationsraums entsprechen.

**Gesamtlänge des Pfades in der Ebene** Sowohl im Wettbewerbskontext des RoboCup als auch bei der Umsetzung des AFiUS Projekt ist es nötig, dass die Gesamtlänge der Teilstrecken zwischen den Posen minimal ist. Bei einem Wettkampf der @Work Liga besteht meist eine Zeitbegrenzung von fünf bis zehn Minuten, um komplette Aufgaben zu erfüllen [14, S. 44]. Diese Zeit sollte nicht mit dem Fahren unnötig langer Wege verschwendet werden. Daher ist dieses Kriterium einer der Hauptaspekte bei der Vergleichbarkeit der Planer und ihrer Konfigurationen.

**Planungszeit** Darunter wird die Verarbeitungszeit verstanden, die zwischen dem Eingang eines neuen Ziels und der Ausgabe eines vollständigen Plans vergeht. Neben der Gesamtlänge des Pfades bildet die Planungszeit den zweiten Hauptaspekt für die Vergleichbarkeit der Planer der OMPL. Dies begründet sich darin, dass sich diese beiden Kriterien direkt auf die Gesamtausführungsdauer der Navigation des Roboters auswirken.

**Gesamtwinkeländerung über den Pfad** Die Summe aller Winkeländerungen der Posen entlang des geplanten Pfades beschreibt die Gesamtwinkeländerung über den Pfad. Für das AFiUS Fahrrad ist dieses Kriterium bedeutend, da durch die kinematischen Abhängigkeiten zwischen Translation und Rotation (Abschnitt 2.2.1), welche vom *Dubins* bzw. *ReedsShepp*

*statespace* berücksichtigt werden (Abschnitt 2.5.1), die Weglänge direkt von der Drehung abhängt. Für eine holonome Plattform wie den youBot ist dieses Kriterium nebensächlich, vorausgesetzt der Roboter bewegt sich, wie in diesem Fall, in alle Richtungen gleich schnell.

## 4.3 OMPL Planer Plug-in

Auf Grundlage des in Abschnitt 3.1 beschriebenen Konzepts, wurde ein auf der OMPL besierendes Plug-in für den ROS *Navigation Stack* entwickelt. Dieses wurde in C++ geschrieben und der vollständige Quellcode befindet sich auf der Compact Disk (CD), die dieser Arbeit beigefügt ist. Es handelt sich dabei um die funktionale Einheit des globalen Planers der *move_base*.

Abbildung 4.3 zeigt eine Unified Modeling Language (UML) Darstellung, welche die Interaktion des Plug-ins mit den Komponenten der move_base und OMPL widerspiegelt. Das Plug-in füllt die abstrakten Funktionen des Interface mit Funktionalität und stellt sie zur Verfügung, sodass sie vom *Navigation Stack* genutzt werden können. Auf der anderen Seite ruft die Funktion die Klassen der OMPL auf und nutzt deren Funktionalität.

Nachfolgend werden die für den Betrieb nötigen Funktionen des Plug-ins vorgestellt.

### 4.3.1 Initialisierung

Die Initialisierungsfunktion lädt die benötigten Parameter aus einer Konfigurationsdatei. Mit diesen Parametern wird unter anderem festgelegt, welcher *statespace* und Planer verwendet wird. Außerdem werden die maximale Dauer eines Planungsvorgangs, wie viele Planungsversuche der Planer durchführt und welchen Wenderadius ein nicht-holomomer Roboter benutzt definiert.

Zusätzlich werden einige Parameter des *costmap*-Plugin geladen, wie zum Beispiel der *costscaling_factor* und der *robot_footprint*. Anhand dieser Parameter und der in Abschnitt 2.4.2 beschriebenen Formel 2.1 lassen sich die Kosten errechnen, die dem äußeren Radius des Roboters entsprechen (*circumscripted radius*). Diese werden für die in Abschnitt 4.3.3 beschriebene Prüfung auf Kollision mit einem Hindernis benötigt. Außerdem werden die Grenzen für den *statespace*

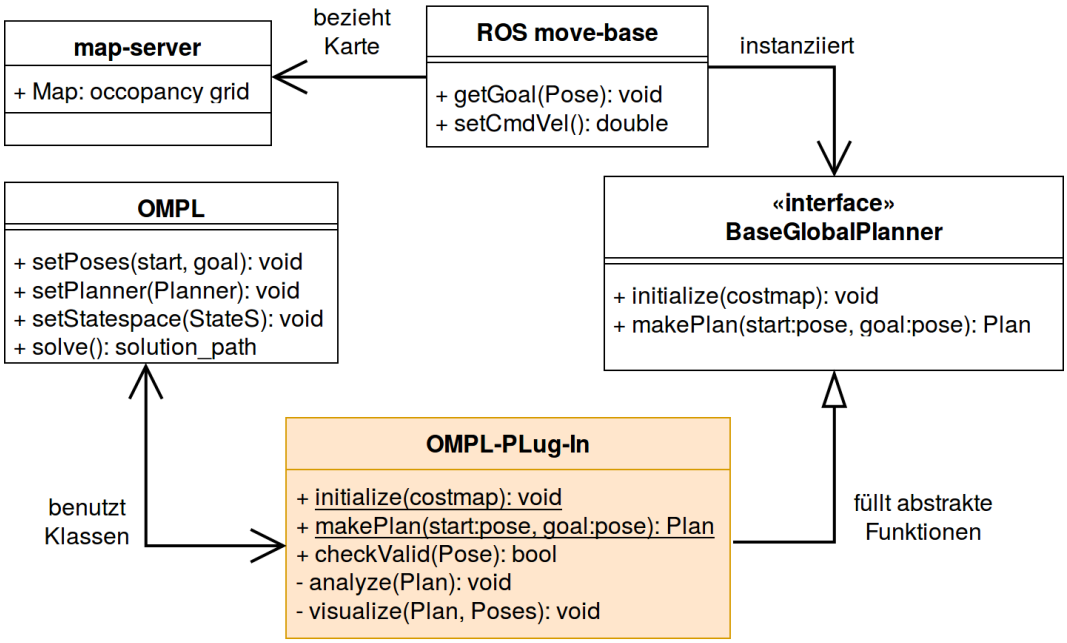



Abb. 4.3: UML Diagramm der OMPL Plug-in Umsetzung

anhand der Höhe und Breite der *costmap* festgelegt. Anschließend werden die in Abschnitt 3.1 dargestellten Objekte der OMPL instanziiert. Sie sind nötig, um das Planungsproblem mit den Mechaniken der OMPL zu bearbeiten. Die Initialisierung wird mit der Instanziierung des Planers abgeschlossen.

### 4.3.2 Ablauf der Planung

Die Planungfunktion beginnt mit der Übergabe der Start- und Zielpose an den Planer. Abhängig vom jeweiligen Planer werden zufällig neue Posen im Konfigurationsraum berechnet. Diese werden mit der in Abschnitt 4.3.3 beschriebenen Funktion überprüft. Ist die Pose unzulässig, wird sie verworfen und eine neue Pose errechnet und geprüft. Andernfalls wird sie dem Plan hinzugefügt und von ihr aus nach einer weiteren Pose gesucht. Dieser Vorgang variiert stark zwischen den zugrunde liegenden Planungsalgorithmen (Abschnitt 2.5.2).

Hat der Planer erfolgreich, je nach Anzahl der maximalen Versuche, einen oder mehrere Pfade gefunden, ist es möglich, überflüssige Punkte zu entfernen. Zusätzlich besteht die Option benötigte Zwischenpunkte hinzu zufügen. Letzteres ist bei Planungen im *Dubins* und *ReedsShepp* Konfigurationsraum notwendig,

um die entsprechenden Kurven für nicht-holomome mobile Systeme zu generieren. Danach werden die einzelnen Posen des Pfades in ein ROS-kompatibles Format umgewandelt und der *move_base* bzw. dem *local_planner* übergeben. Darüber hinaus wird der Plan und seine Posen an dieser Stelle *gepublished* (Abschnitt 2.4), damit beispielsweise eine Visualisierung möglich ist.

### 4.3.3 Kollisionsabfrage

Während der Planungen kann es zu Kollisionen kommen. Daher muss nach der Planung eine Kollisionsabfrage stattfinden. Das Plug-in benötigt zur Überprüfung der von OMPL errechneten Posen eine Funktion, die vom Planer aufgerufen wird. Der Planer übergibt der Funktion den neuen errechneten Mittelpunkt des Roboters und die Orientierung der Pose. Diese wird darauf in mehreren Stufen überprüft:

Zunächst wird mit der *getCost()* Methode abgefragt, welche Kosten in der *costmap* an diesem Punkt eingetragen sind. Sind die Kosten größer gleich den Kosten im *inscribed radius* oder gleich den Kosten eines Hindernisses, wird der Punkt direkt verworfen. Sind die Kosten kleiner als die Kosten im *circumscribed_radius*, befinden sich im gesamten äußeren Radius des Roboters keine Hindernisse. Die Pose des Roboters ist somit unabhängig von der Orientierung zulässig und wird in den Plan aufgenommen. Wenn die Kosten an dem überprüften Mittelpunkt zwischen den Kosten des *circumsribed* und *inscribed radius* liegen, geht die Überprüfung in die nächste Stufe.

Das Polygon des Roboters wird mit der vom Planer ermittelten Orientierung um den derzeitigen Mittelpunkt gelegt. Von dort aus ermittelt die Funktion mittels der *costmap* die Kosten jedes Polygonen-Punktes. Sollte einer der überprüften Punkte die Kosten eines Hindernisses aufweisen, befindet sich der Roboter in Kollision und der Mittelpunkt bzw. die dazu gehörige Pose wird für die Pfadplanung verworfen. Liegt keiner der Punkte in einem Hindernis, so wird die Pose in den Plan übernommen.

Sollten die Kosten an einem Punkt dem NO_INFORMATION-Wert der *costmap* entsprechen, also keine Informationen enthalten, wird die Pose nur verwendet, wenn ein entsprechender Parameter (plan_through_unknown) gesetzt ist. Ansonsten wird sie verworfen.

### 4.3.4 Zusätzliche Funktionen

Das Plug-in verfügt über eine Reihe zusätzlicher Funktionen. Diese dienen in ihrer Gesamtheit der Auswertung der Pfade und der Visualisierung. Daher ist es beispielsweise möglich, Pfade, samt aller in ihnen befindlichen Posen, mit dem ROS eigenen Visualisierungswerkzeug *rviz* zu veranschaulichen (siehe Abbildung 4.4). Darüber hinaus kann für jede Planung eine Auswertungsdatei mit den Eigenschaften der Pfade angelegt werden. Beispiele können auf der beiliegenden CD eingesehen werden.

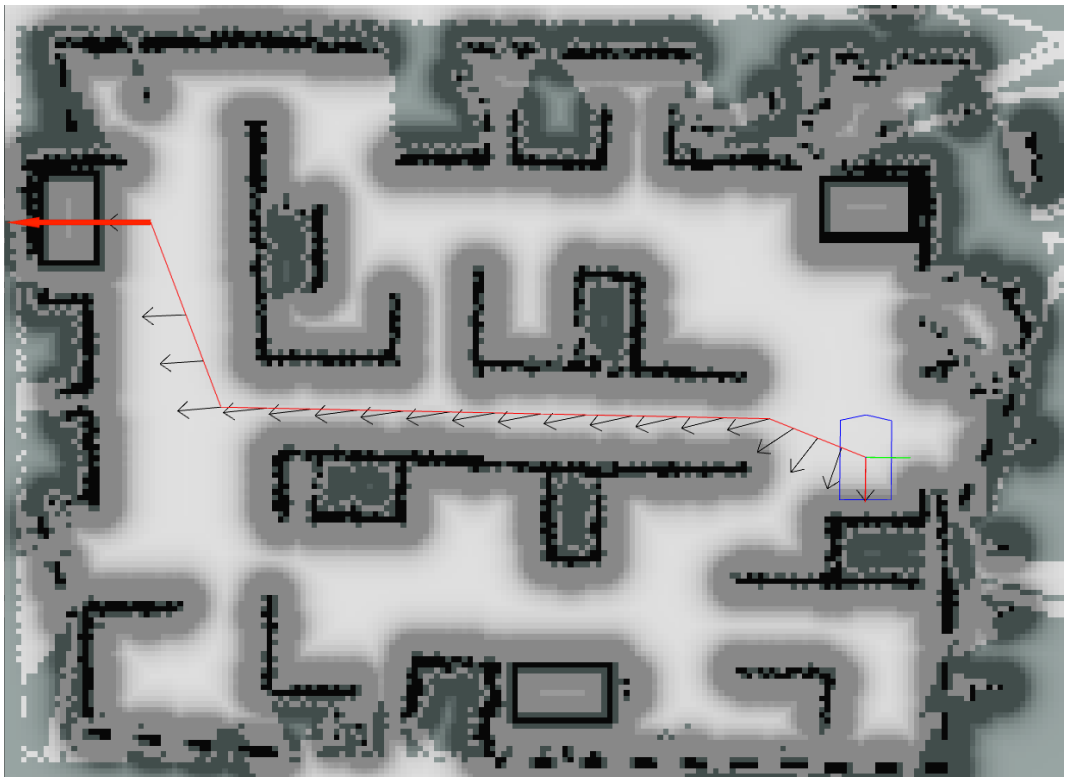


Abb. 4.4: Ausschnitt des ROS *rviz* Visualisierungstools während der Pfadplanung

Nachdem das Plug-in in Bezug auf die genannten Anforderungen entwickelt wurde, werden die Tests durchgeführt.

## 4.4 Versuchsanordnung

Nachfolgend wird die Gestaltung der Experimente sowie die benutzte Hard- und Software beschrieben.

### 4.4.1 Ablauf der Experimente

Da die OMPL über eine Reihe von Planern und diese Planer wiederum über Konfigurationsparameter verfügen, die sich auf Ihr Planungsverhalten auswirken, laufen die Experimente gestaffelt ab. In Tabelle 4.4 ist der Ablauf der geplanten Experimente abgebildet. Zunächst ist das Ziel, bestimmte Planer und Konfigurationen auszuschließen, wenn diese eindeutig unzureichender sind als andere oder nicht den Mindestforderungen entsprechen.

Tabelle 4.4: Anordnung der Experimente für die Evaluation

| | **Experiment** | **Szenario** | **Bewertungskriterien** |
|---|---|---|---|
| **1.** | Gegenüberstellung verschiedener Konfigurationsparameter | RoboCup Weltmeisterschaft | Pfadlänge und Planungszeit |
| **2.** | Pfadplanung der optimierenden Planer, die ein Optimierungskriterium unterstüzen | | Pfadlänge und Planungszeit |
| **3.** | Planung der optimierenden Planer, die eine Abbruchszeit unterstützen | | Pfadlänge |
| **4.** | Planung auf ausgewählten Strecken | OvGU Campus | Pfadlänge, Planungszeit und Winkeländerung |

Wie in Abschnitt 2.5.2 bereits dargelegt, sind nur die optimierenden Planer der OMPL für die angeführten Szenarien geeignet. Der Grund dafür ist, dass die Planer auf probabilistischen Verfahren beruhen und somit immer unvorhersagbare Suchbäume aufstellen. Somit ist auch die Länge des Zielpfades und die Lage der in ihm befindlichen Posen unvorhersehbar. Es ist möglich, dass der erstbeste Pfad, der die Kriterien **Kollisionsfreiheit** und **exakter und**

**vollständiger Pfad** erfüllt, sich als völlig unzureichend bei der Pfadlänge herausstellt.

Die optimierenden Planer benötigen ein Abbruchkriterium. In der Regel ist das die eingestellte maximal zulässige Planungszeit, welche beim WM-Szenario mit 30 und im OvGU-Szenario mit 50 Sekunden gewählt wurde. Dadurch ergibt sich, dass alle Planer zur gleichen Zeit die Planung einstellen und man die Planungszeit nicht miteinander vergleichen kann. Daher wurden im Vorfeld zu den Experimenten alle 306 möglichen Pfade der WM-Karte und 56 möglichen Pfade der OvGU-Karte mit dem RRT*-Planer mit einer maximalen Planungsdauer von 60 (WM) bzw. 90 (OvGU) Sekunden geplant. Für diese Pfade wurde die in Abschnitt 2.5.1 angegebene Gesamtlänge ausgelesen. Sie wird als Referenz für die ideale Länge angenommen. Das zweite Abbruchkriterium neben der Zeit ist, dass die Planer einen Wert von 110% der Ideallänge unterschreiten müssen.

Die Annahme des beschriebenen Pfades als Vorgabe wird durch die OMPL-Dokumentation [46] und durch [18] gestützt. Bei ausreichender Planungsdauer nähert sich der Algorithmus dem idealen reellen Pfad asymptotisch an.

Somit ist gewährleistet, dass Pläne, die das Abbruchkriterium der Länge nicht unterschreiten, verworfen werden und neu geplant werden müssen, womit sich die Planungszeit verlängert. Dadurch ist es möglich, beide Anforderungenskriterien (Weglänge und Planungszeit) zu überprüfen.

Nicht alle Planer benutzen das zweite Kriterium. Daher wird zunächst mit denen geplant, die das Längenkriterium nutzen. Anhand der Planungszeiten wird anschließend eine Abbruchzeit für die verbleibenden Planer gefunden.

Die Planungen erfolgen auf ausgewählten Strecken der WM-Karte. Es wird eine Hierarchie nach den in Abschnitt 4.2 aufgestellten Hauptkriterien **Planungszeit** und **Pfadlänge in der Ebene** erarbeitet.

Abschließend werden die besten Planer benutzt, um ausgewählte Strecken auf der OvGU-Karte zu planen. Da die Karte deutlich größer ist, werden sowohl längere Pfade als auch Planungszeiten erwartet.

### 4.4.2 Hard- und Softwarespezifikationen

Die Experimente finden auf einem Computer der Universität Magdeburg statt, dessen Spezifikationen in Tabelle 4.5 abgebildet sind. Zur Planung wird ein Ro-

Tabelle 4.5: *Hard-* und *Software* Spezifikationen des Test PC

| **Spezifikation** | **Beschreibung** |
|---|---|
| Betriebssystem | Ubuntu 14.04 mit Linux-Kernel 4.4.0 |
| Main Board | Gigabyte X79-UD3 |
| Prozessor | Intel Core i7-3930K Hexa-core mit 3,2 (bis 3,8) Ghz 12mb Cache |
| Grafikkarte | Radeon HD 6970 mit 2 Gb Arbeitsspeicher |
| Arbeitsspeicher | 8 Gb |

boter auf dem selben Personal Computer (PC) simuliert. Die in ROS enthaltene Simulationssoftware "Gazebo" wird dafür verwendet.

Wie in den Abschnitten 2.4 und 2.5 beschrieben, wird die ROS Version *Indigo Igloo* (veröffentlicht: 22. Juli 2014) und die OMPL Version 1.2.1 (veröffentlicht: 01. Juli 2016) genutzt. Der ROS *Navigation Stack* ist auf dem Stand vom 23. August 2016 im *jade-branch*.

# 5 Evaluation

In diesem Kapitel werden die Ergebnisse der durchgeführten Pfadplanungen ausgewertet. Die Versuche wurden sequentiell durchgeführt. Dabei wirken sich die stufenweise erlangten Erkenntnisse auf die fortlaufenden Experimente aus.

## 5.1 Vorbetrachtungen

Während der Durchführung der Versuche ist es zu einigen Komplikationen gekommen, die nachfolgend erläutert werden.

Zunächst konnten die drei, im Abschnitt 2.5.2 aufgeführten, Planer nicht in Betrieb genommen werden, da sie nicht funktionsfähig waren. Im Folgenden werden die Probleme näher erläutert:

**BIT*-Planer** Dieser Planer bringt das Plug-in während der Initialisierung zum Absturz. Ein *backtrace* mit dem GNU Debugger (GDB) ergab, dass der Planer während der *setup*-Methode der OMPL ausfällt. Eine Fehlermeldung wird dabei nicht ausgegeben.

**LBTRRT- und SST-Planer** Beide Planer sind nach der ersten Planung abgestürzt. Hier stellte sich bei der Suche mit GDB heraus, dass die Planer in der *solve*-Methode der OMPL ausfallen. Ebenso wie beim BIT*-Planer wird hierbei keine Fehlermeldung ausgegeben.

Darüber hinaus war es im OvGU-Szenario nicht möglich, den **Dubins state-space** zu benutzen. Die Planungen sämtlicher Algorithmen waren fehlerhaft oder unvollständig. In Abbildung 5.1 ist eine solche Fehlplanung abgebildet. Der Pfad unterliegt hier zum einen nicht dem vorgegebenen Radius der Planung. Zum anderen verläuft er durch eindeutig "belegt" gekennzeichnete Zellen der Karte und ist somit vor dem Kriterium der vollständigen und exakten Pfade unbrauchbar. Darüber hinaus wurde der Pfad als gültig gekennzeichnet

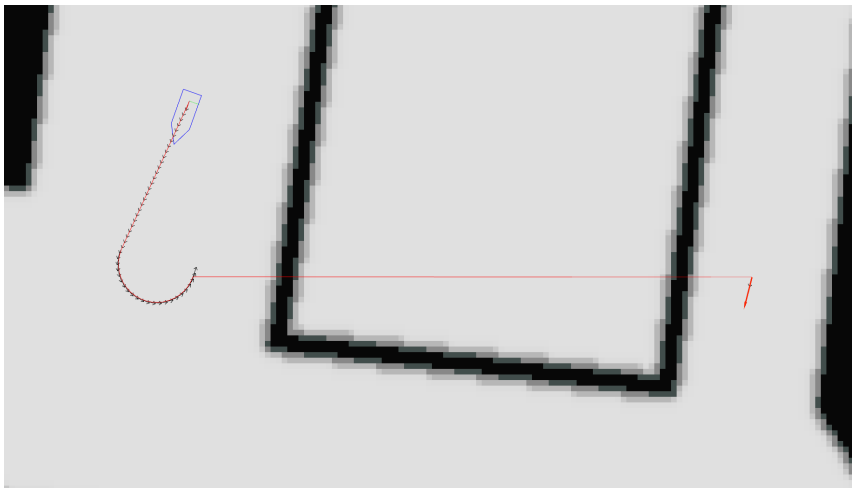


Abb. 5.1: Beispiel für das Fehlverhalten des *Dubins* Konfigurationsraums

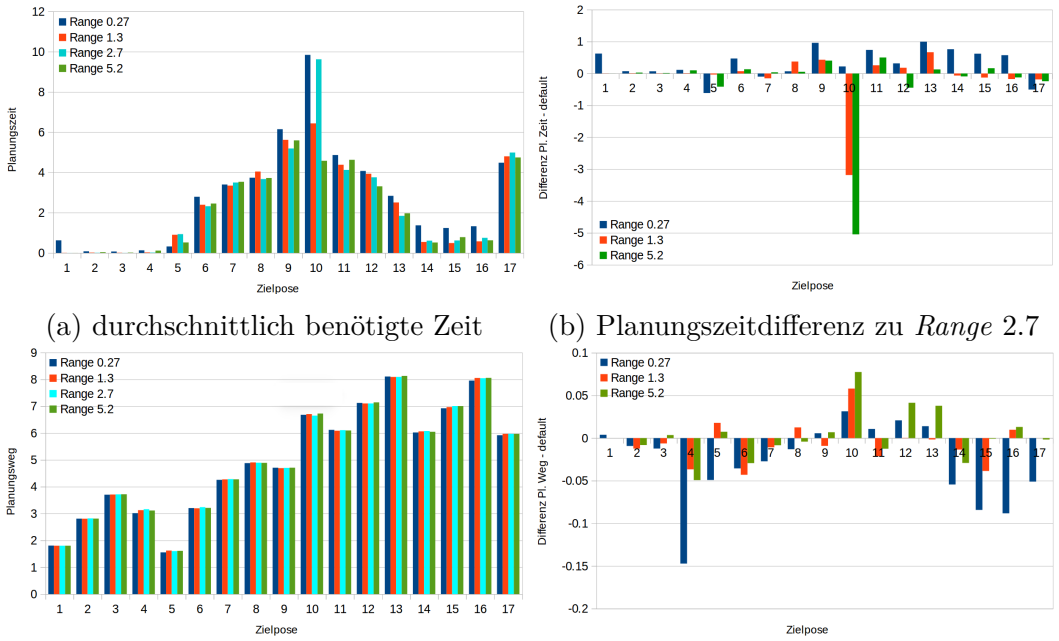
und der *move_base* zurückgegeben. In einer realen Fahrtplanung kann solches Fehlverhalten fatale Auswirkungen haben.

Gezielte Versuche, die Funktion der Planer und des *statespace* unter Berücksichtigung der Dokumentation der OMPL[46] mittels Parametrisierung wiederherzustellen, waren nicht erfolgreich. Daher wäre es notwendig, eine intensive Fehlersuche im Quellcode der OMPL durchzuführen. Da das den Rahmen der Arbeit übersteigt, wurde auf die Nutzung genannter Komponenten verzichtet. Allerdings sind die aufgetretenen Störungen den Hauptentwicklern der OMPL mitgeteilt worden. Bis zum Zeitpunkt der Abgabe lag keine Antwort vor.

## 5.2 Experiment 1: Einflusses der Parameter

In der OMPL Dokumentation [46] werden verschiedene Parameter erläutert, die Einfluss auf das Verhalten der Planer haben. Durch den Wegfall der angesprochenen Planer reduzieren sich die *tree based* Planer auf diejenigen, die den gleichen Parametersatz benutzen. Während die *roadmap based* Planer keine Parameter benutzen, welche zur Änderung für den Anwender in der Dokumentation gekennzeichnet wurden.

Stellvertretend für die genannten *tree based* Planer wurde in diesem Experiment der Einfluss des Paramters *Range r* auf die Planungsdauer und Weglänge des RRT*-Planers untersucht. Dieser Parameter beschreibt den maximalen




(a) durchschnittlich benötigte Zeit

(b) Planungszeitdifferenz zu *Range* 2.7

(c) durchschnittlich geplante Weglänge

(d) Weglängendifferenz zu *Range* 2.7

Abb. 5.2: Vergleich der Planungskonfigurationen

Abstand einer zufälligen neuen Pose innerhalb des Suchraums von der zuletzt geplanten. Dabei wird der Abstand nach der in Formel 2.2 angegebenen Gesamtlänge berechnet. Ist dieser Parameter nicht angegeben, wird er in der OMPL automatisch aus den Eigenschaften des Konfigurationsraums errechnet.

In dem WM-Szenario beträgt dieser Standardwert rund $r = 2,7$. Dieser Wert wird in diesem Experiment mit dem Doppelten, dem Halben und einem Zehntel des Werts verglichen.

Dazu hat der Planer von der Startposition des RoboCup Wettkampfes (in Abbildung 4.1 die Null) Wege zu allen Tischen und zur Endposition einhundert Mal geplant. Diese Pläne weisen ein breites Spektrum an Weglängen und Planungszeiten auf, deren Mittelwerte in Abbildung 5.2a und 5.2c gezeigt werden. Der Standardwert $r = 2,7$ ist in cyan gekennzeichnet. Zum Vergleich werden in den daneben liegenden Abbildungen 5.2b und 5.2d die jeweiligen Differenzen mit den Ergebnissen des standard-parametrisierten Planers dargestellt.

Nimmt der Parameter den Wert $r = 0,27$ an, ist die zeitliche Differenz zum Standardplaner am größten und beträgt je nach Planung zwischen einer halben und einer Sekunde mehr, als dies bei diesem der Fall ist. Dafür unterbietet

diese Parametrisierung in den meisten Fällen die mittlere Weglänge des Standardplaners. Allerdings handelt es sich hierbei um Differenzen von maximal 15 cm, welche, bei einer Kartenauflösung von 5 cm/px und Weglängen von mehreren Metern, zu vernachlässigen sind.

Die anderen Planungen mit $r = 5.2$ und $r = 1.3$ unterscheiden sich indes kaum von Standardwert. Einzig auffällig sind die Spitzen bei der Planung vom Start zu Pose Zehn. Da dieser Ausschlag jedoch einmalig bei einer Vergrößerung bzw. Verkleinerung des Wertes ist, wird er vernachlässigt.

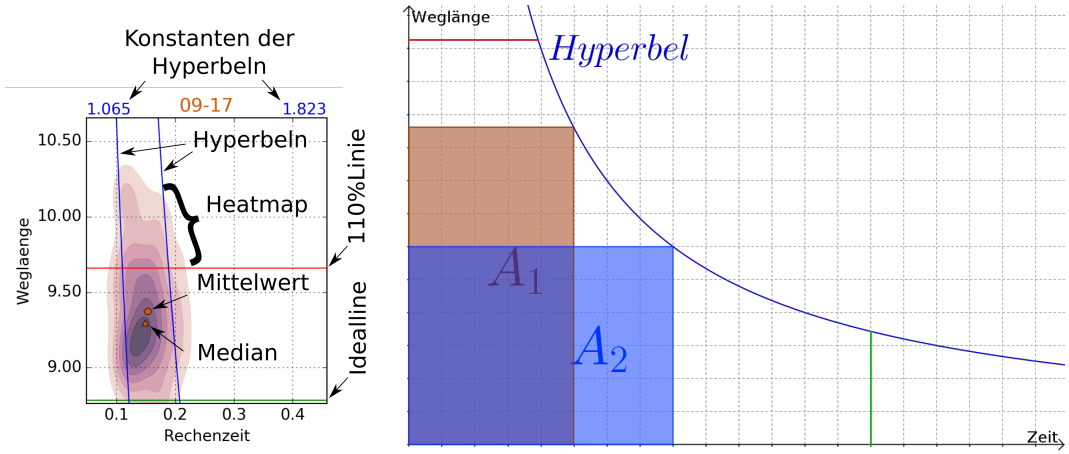
Generell ist über die Gesamtheit der Planungen keine eindeutige Tendenz zur Veränderung von Planungszeit und Weglänge zu erkennen. Daher wurde in den darauffolgenden Experimenten die Standard-Parametrisierung verwendet.

## 5.3 Experiment 2: Vergleich der Planungsalgorithmen

In diesem Experiment wurden die in Abschnitt 2.5.2 aufgezählten Planer verglichen. Dazu mussten die Planer Pfade zwischen den in Tabelle 4.2 aufgezählten Posen errechnen. Dabei galten die in Abschnitt 4.4.1 erläuterten Bedingungen. Analog zum Experiment 1 wurde jeder Weg von jedem Planer mit einhundert Wiederholungen geplant. Die Ergebnisse dieser Planungen sind in den Abbildungen 5.4, 5.5, 5.6, 5.7, 5.8 und 5.9 dargestellt. Vor der Auswertung der Ergebnisse erfolgt zunächst eine Erläuterung der Darstellungen in Abbildung 5.3a.

### 5.3.1 Erläuterung der Darstellung

Auf der Abszisse ist die Zeit, auf der Ordinate die Weglänge abgebildet. Mittig über dem Diagramm befindet sich die Kombination aus Start und Ziel (braun; in diesem Fall Start: 09, Ziel: 17) Die Ergebnisse der Planungen sind in sogenannten *Heatmaps* abgebildet (Abb. 5.3a lila Wolke), welche zweidimensionale Histogramme darstellen. Die Farbe der *Heatmap* nimmt an Intensität zu, je mehr Planungsergebnisse (in Bezug auf Zeit und Weglänge) an einem Ort liegen. Zusätzlich wurden ein braunes Dreieck für den Median und ein brauner Kreis für den Mittelwert eingezeichnet. Zur Orientierung wurden darüber hinaus




(a) Veranschaulichung der Grafiken

(b) Beispiel einer Hyperbel

Abb. 5.3: Erläuterung der Darstellungen

mit einer grünen und einer roten Linie die ideale Länge des jeweiligen Pfades und 110% dieser Ideallänge eingezeichnet. Abschließend wurden jeweils zwei Hyperbeln (blaue Graphen) in die Darstellungen eingebracht. Die Formel für diese lautet:

$$l = \frac{k_{1,2}}{t}, \tag{5.1}$$

wobei $l$ die Weglänge und $t$ die Planungszeit charakterisiert. Die Konstanten $k_{1,2}$ wurden wie folgt errechnet: Jede Weglänge eines Pfades wurde mit der dazugehörigen Planungszeit multipliziert. Dadurch entstand eine Liste von Werten, welche nach der Größe geordnet wurde. Da es sich um probabilistische Verfahren handelt, sind die Werte in der Liste entsprechend gestreut. Um dem entgegen zu wirken, wurden die obersten und untersten zehn Prozent der Ergebnisse abgeschnitten. Das verbleibende Maximum stellt $k_1$, das verbleibende Minimum $k_2$ dar. Zwischen den Hyperbeln befinden sich somit 80 % der Ergebnisse. Die Werte von $k_1$ und $k_2$ sind als blaue Zahlen über dem Koordinatensystem abgebildet, wobei der linke Wert immer der unteren-linken und der rechte Wert der oberen-rechten Hyperbel zuzuordnen ist.

In Abbildung 5.3b wird das Konzept dieser Hyperbeln noch einmal erläutert. Auf der Abszisse ist wiederum die Zeit und auf der Ordinate die Weglänge abgebildet. Die dargestellten Flächen $A_1$ und $A_2$ unter der Hyperbel sind stets gleich groß, da es sich hierbei um eine zur Winkelhalbierenden des ersten

Quadranten ($y = x$) symmetrisch verlaufenden Hyperbel handelt. Wird dieser Sachverhalt auf die gesamte Ausführungsdauer einer Navigation übertragen, ergibt sich ein indirekt proportionaler Zusammenhang zwischen Pfadlänge und Planungszeit. Die Ausführungsdauer setzt sich dabei aus der Planungszeit und der Zeit zum Abfahren des Weges zusammen.

Folgendes Szenario wäre also denkbar: Planer A plant in kürzerer Zeit als Planer B einen Pfad. Jedoch ist der Pfad des Planer A deutlich länger als der von Planer B. Da aber der Roboter von Planer A früher startet, kann er den Zeitverlust bezüglich der verlängerten Wegstrecke ausgleichen und erreicht zeitgleich mit dem anderen Roboter das Ziel.

Es muss berücksichtigt werden, dass die Steigung der Ausführungsgeschwindigkeit angepasst werden muss. Für die Auswertung der Ergebnisse genügt die festgelegte Hyperbel.

Mit den Hyperbeln und deren Konstanten $k_{1,2}$ wurde die Qualität der Pfadplanungen untereinander verglichen und die Streuung innerhalb einer Pfadplanung eingeschätzt. Sie dienen zudem der Übersichtlichkeit, da sich sowohl die Wertebereiche sowie die Skalierung der Achsen der Koordinatensysteme aufgrund der großen Verteilung der Ergebnisse in den Abbildungen stetig ändern.

Allerdings hat das Modell Einschränkungen, die als grüne Linie für die Zeit und als Rote Linie für den Weg in Abbildung 5.3b dargestellt sind. Die Planer dürfen weder für kurze Wege unbegrenzt rechnen, noch darf ein unbegrenzt langer Weg in kürzester Zeit geplant werden.

### 5.3.2 Auswertung der Ergebnisse

In Abbildung 5.4 werden die Ergebnisse des PRM*-Planers dargestellt. In den oberen vier Durchläufen wurden kurze Wege und in den vier unteren lange Pfade geplant. Bei den oberen Diagrammen findet sich keine Streuung. Außerdem wurde in kürzester Zeit bis an die Ideallänge heran geplant.

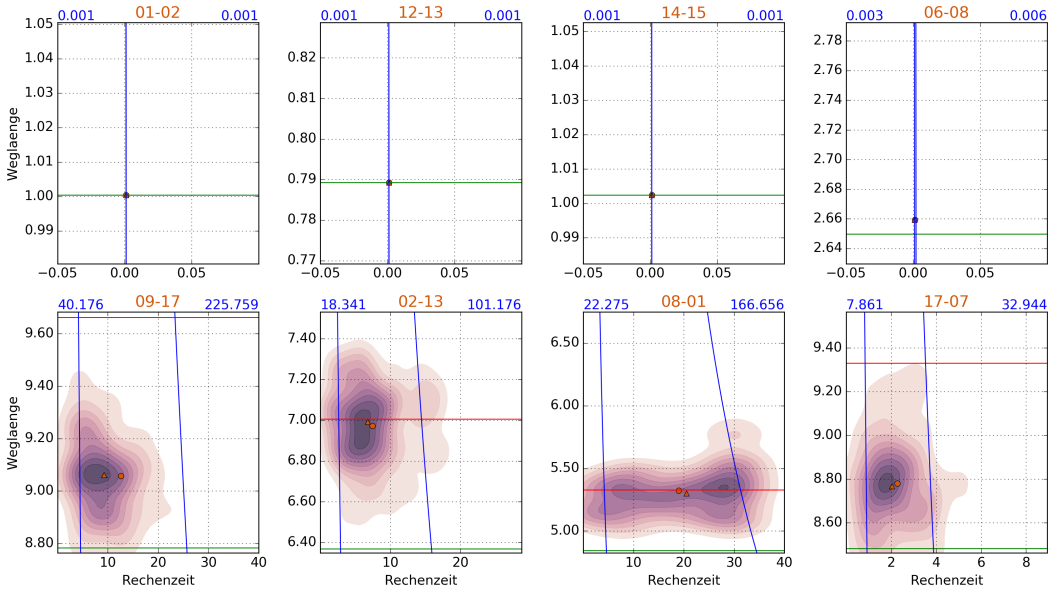



Abb. 5.4: Ergebnisse - PRM*-Planer - WM-Karte

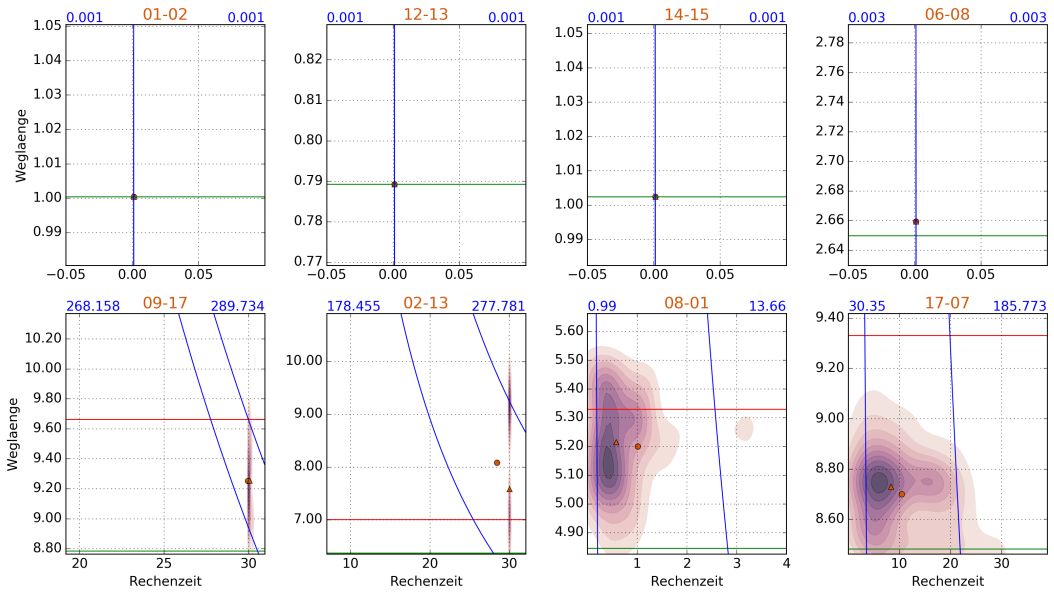



Abb. 5.5: Ergebnisse - LazyPRM*-Planer - WM-Karte

Die Pläne für 09-17 und 17-07 liegen beide unter der geforderten roten Linie, während ersterer dabei eine größere Verteilung zwischen den Ergebnissen aufweist. Die Pläne 02-13 und 08-01 liegen teilweise oberhalb der geforderten 110 %, während letzterer auch die 30 Sekundengrenze überschreitet.

In der nachfolgenden Abbildung 5.5 erreicht der zweite *roadmap* Planer bei den kurzen Strecken ähnliche Ergebnisse wie der PRM*. Bei den längeren Pfaden erreicht er nur bei den Planungen von 08-01 und 17-07 sehr gute bis akzeptable Werte, während die restlichen Planungen die Zeitgrenze überschreiten.

Der RRT*-Planer (Abbildung 5.6) erreichte in allen Pfaden schnell Ergebnisse und überschritt dabei allerdings bei 08-01 und 02-13 die geforderten 110 %. Bei Letzterem liegen die durchschnittlichen Weglängen über der Grenze.

In Abbidlung 5.7 zeigt der FMT* Algorithmus zwar kleinere Streuungen bei den kurzen Pfaden, welche allerdings keine gravierenden Auswirkungen auf die Zeit oder den Weg haben. Außerdem war es ihm möglich, bei allen Pfaden, bis auf 02-13, innerhalb kürzester Zeit sehr kurze Wege zu generieren.

Der Informed RRT*-Planer generiert in diesem Szenario sehr ähnliche Werte wie der RRT* in Bezug auf Zeit, Weglänge und Hyperbelkonstanten. Während der T-RRT Algorithmus in Abbildung 5.9 bei den kurzen und langen Pfaden die Höchstweglänge nicht einhält. Dafür gelang es ihm in jeder Planungsaufgabe zügig einen Pfad zu finden.

Die erste Gemeinsamkeit, die sich erkennen lässt, ist die durchweg qualitativ hochwertige Performance bei kurzen Pfaden. Der einzige Planer, der hier durch längere Pfade auffällt, ist der T-RRT. Dieser ist, wie in Abschnitt 2.5.2 bereits benannt, per Definition nicht stark an Optimierungskriterien gebunden. Die zweite Gemeinsamkeit bildet das Unvermögen die geforderten 110% des Versuches 02-13 mit allen Planungen zu unterschreiten.

Aufgrund dieser Ergebnisse kann darauf geschlossen werden, dass die Kombination beider Posen als anspruchsvoll angesehen wird. Bei den weiteren Aufgaben ist keine Gemeinsamkeit festzustellen. Beispielsweise hat der FMT* Planer bei 09-17 und 08-01 die kleinsten Hyperbelwerte, während LazyPRM* und PRM* sehr hohe Zahlen aufweisen. Die Utalität der Hyperbeln zum Vergleich der Algorithmen wird bei LazyPRM* und PRM* mit der Aufgabe 17-07 hervorgehoben.

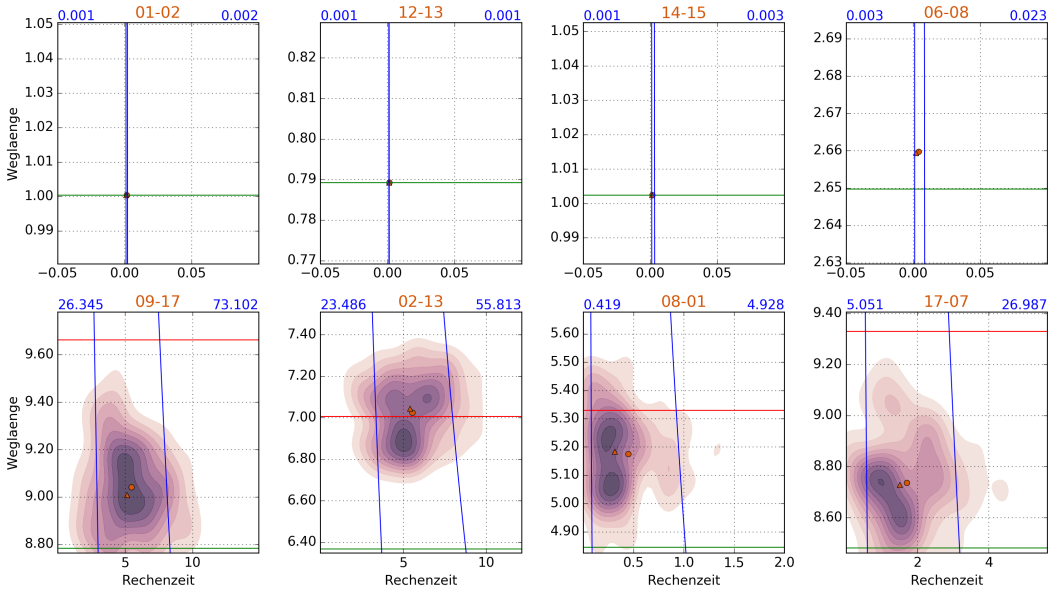



Abb. 5.6: Ergebnisse - RRT*-Planer - WM-Karte

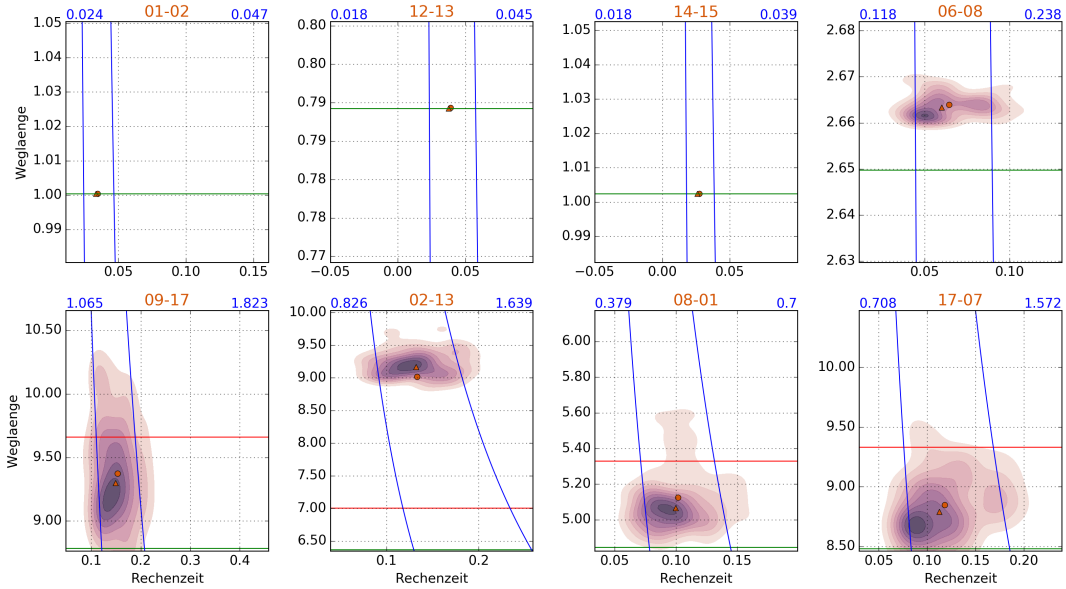



Abb. 5.7: Ergebnisse - FMT*-Planer - WM-Karte

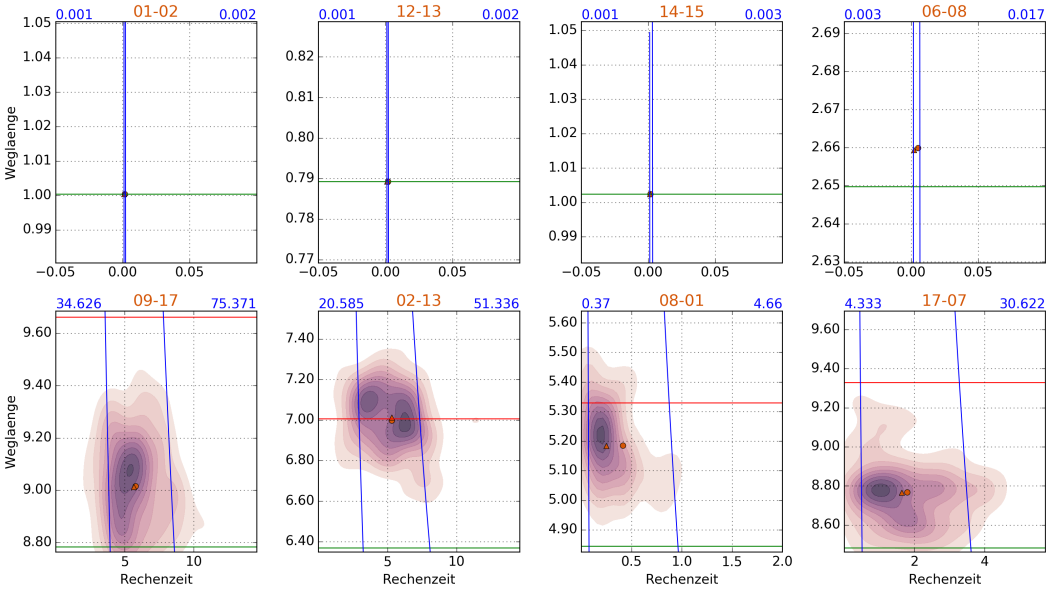



Abb. 5.8: Ergebnisse - Informed RRT*-Planer - WM-Karte

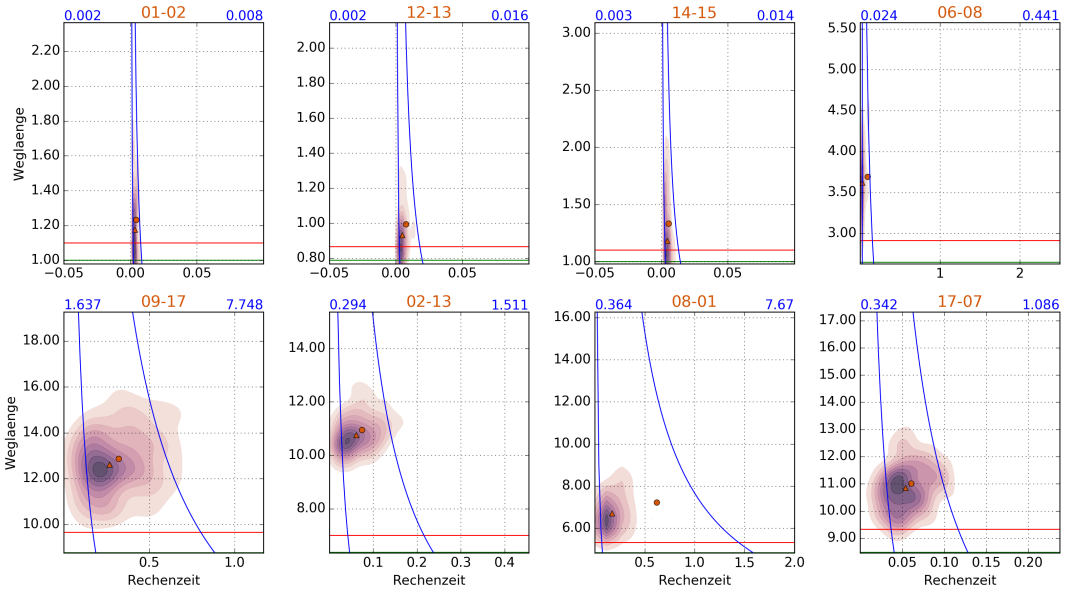



Abb. 5.9: Ergebnisse - T-RRT-Planer - WM-Karte

Die *Heatmaps* haben hier ungefähr die gleiche Form sowie Position und die Weglängenskalierung ist ebenfalls sehr ähnlich. Da aber die Zeitspanne des LazyPRM* wesentlich größer ist, liegen die *Heatmaps* global gesehen nebeneinander, was bedeutet, dass der LazyPRM*-Planer länger gebraucht hat.

Um alle Planer miteinander zu vergleichen, werden die rechten Hyperbelkonstanten bzw. Flächeninhalte, in denen sich 90 % der Ergebnisse befinden, in einer Tabelle gegenüber gestellt (siehe Abbildung 5.10). Die Ergebnisse wurden bezogen auf Ihr Qualität eingefärbt. Es bilden sich drei Gruppen aus diesem Vergleich. FMT* und T-RRT haben durchgehend niedrige Werte, RRT* und Informed RRT* liegen in der Mitte und LazyPRM* und PRM* haben deutlich höhere Werte.

| | 01-02 | 12-13 | 14-15 | 17-07 | 09-17 | 02-13 | 08-01 | 06-08 |
|---|---|---|---|---|---|---|---|---|
| InformedRRTstar | 0.002 | 0.002 | 0.003 | 30.622 | 75.371 | 51.336 | 4.66 | 0.017 |
| FMT | 0.047 | 0.045 | 0.039 | 1.572 | 1.823 | 1.639 | 0.7 | 0.238 |
| LazyPRMstar | 0.001 | 0.001 | 0.001 | 185.773 | 289.734 | 277.781 | 13.66 | 0.003 |
| PRMstar | 0.001 | 0.001 | 0.001 | 32.944 | 225.759 | 101.176 | 166.656 | 0.006 |
| RRTstar | 0.002 | 0.001 | 0.003 | 26.987 | 73.102 | 55.813 | 4.928 | 0.023 |
| TRRT | 0.008 | 0.016 | 0.014 | 1.086 | 7.748 | 1.511 | 7.67 | 0.441 |

Abb. 5.10: Vergleich der Hyperbelkonstanten - WM-Karte

Für die folgenden Experimente werden aufgrund dieser Erkenntnisse nur die leistungsfähigeren Gruppen in Betracht gezogen.

## 5.4 Experiment 3: Vergleich der restlichen Planer

In diesem Experiment sollten die aus dem zweiten Experiment übernommenen Planer mit den beiden optimierenden Planern SPARS und SPArse Roadmap Spanner algorithm 2 (SPARS2) verglichen werden. Diese wurden nicht in den vorangegangen Versuch aufgenommen, da sie kein Optimierungkriterium außer einer Abbruchszeit akzeptieren.

Um sie mit den Planern aus dem zweiten Experiment vergleichen zu können, wurde folgende Bedingung festgelegt: die SPARS Planer müssten in der längsten Zeit (sowie einer Toleranz von 20 %) der bereits ausgewählten Planer, mindestens genauso kurze Weglängen erreichen.

Beim Ausführen dieses Experiments ist es den SPARS Planern nicht gelungen innerhalb der geforderten Zeit vollständige und exakte Pläne aufzustellen. Somit werden sie im letzten Experiment im OvGU-Szenario nicht berücksichtigt.

## 5.5 Experiment 4: Einsatz im OvGU-Szenario

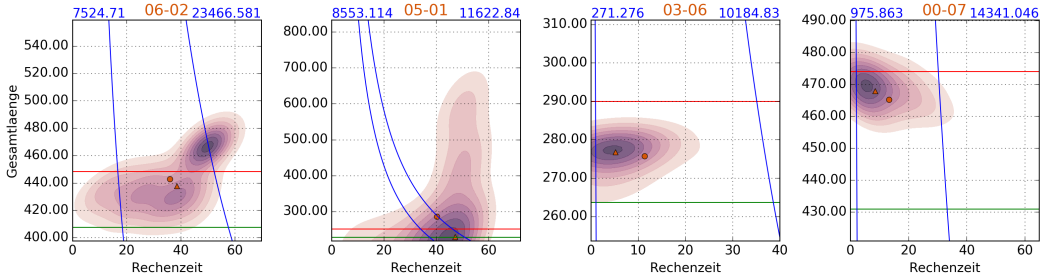
Im letzten Experiment dieser Arbeit wurden die besten vier Planer der vorangegangen Versuche im OvGU-Szenario getestet und die erhobenen Daten entsprechend analysiert. Im Unterschied zu den Experimenten in der WM-Karte wurden hier Wege für eine nicht-holonome Plattform berechnet. Daher wirkt sich die Winkeländerung auf die Weglänge aus. In Abbildung 5.11 ist deshalb auf der Ordinate die Weglänge durch die Gesamtlänge nach Formel 2.2 ersetzt worden.

Ein Aspekt, der in diesem Szenario wesentlich gravierender ist als bei den Experimenten auf der WM-Karte, ist die Überprüfung eines **exakten und vollständigen Pfades** und der **Kollisionsfreiheit**. Sind diese Kriterien nicht erfüllt, werden die Ergebnisse der Planungen nicht in den Diagrammen in Abbildung 5.11 berücksichtigt. In Tabelle 5.1 wird die jeweilige Anzahl der Planungen gezeigt, die diese Kriterien erfüllen. Der Informed-RRT*-Planer wurde in der Tabelle beim Pfad von 05 nach 01 rot markiert, da er lediglich 28 von 100 Planungen erfolgreich absolviert hat. Während die anderen Planer mindestens 82 erfolgreiche Planungen aufweisen. Somit sind die Ergebnisse dieses Planers für diese Aufgabe nicht aussagekräftig.

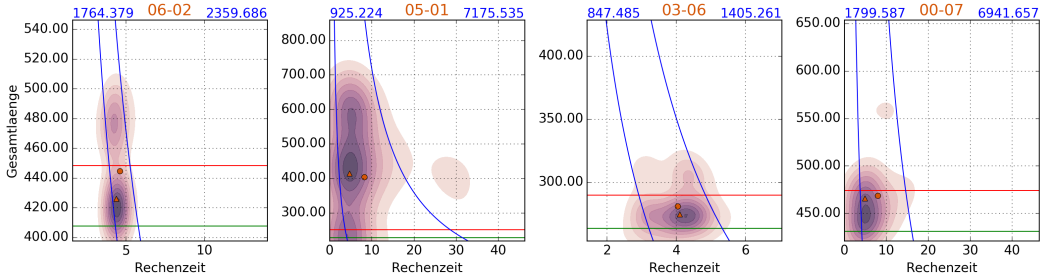
Der Mittelwert und der Median der übrigen Planer liegen bei dieser Aufgabe über dem geforderten Gesamtlängenwert, teilweise mit Abweichungen von 200 % zum Idealwert. Zudem stoßen T-RRT und RRT* Planer teilweise an die zeitliche Begrenzung. Daher ist anzunehmen, dass die Kombination aus Start- und Zielpose anspruchsvoller ist, als beispielsweise 03-06, bei der alle Planer günstige Weglängen oder Planungszeiten aufweisen.

Dies zeigt auch die Gegenüberstellung der rechten Hyperbeln, welcher in Abbildung 5.12 dargestellt ist. An dieser Stelle wird deutlich, dass die Werte für den Weg 05-01 bei allen Planern überdurchschnittlich hoch sind im Vergleich zu den Werten von 03-06 und das, obwohl sich die Ideallängen mit 228.67 (05-01) und 263.65 (03-06) nur um etwa 15 % unterscheiden.

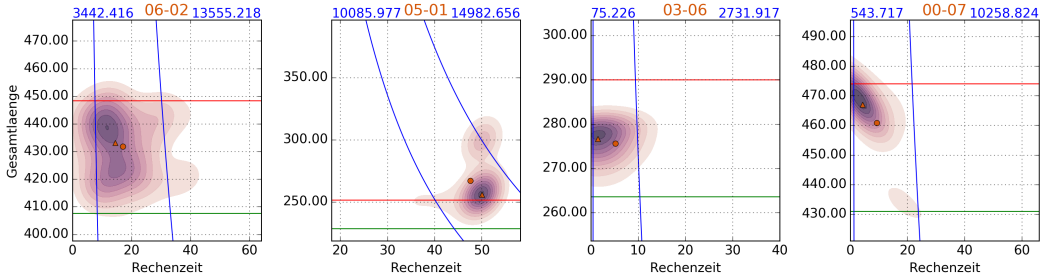
Gleichzeitig wird eine Grenze des Hyperbelvergleichs als Werkzeug für die Gegenüberstellung der Pfade untereinander aufgezeigt. Dies ist nur möglich, da die idealen Gesamtlängen sich ähneln. Wird nun Aufgabe 03-06 mit 00-07 verglichen, muss berücksichtigt werden, dass die Ideallänge von 00-07 bei 430.98 liegt.



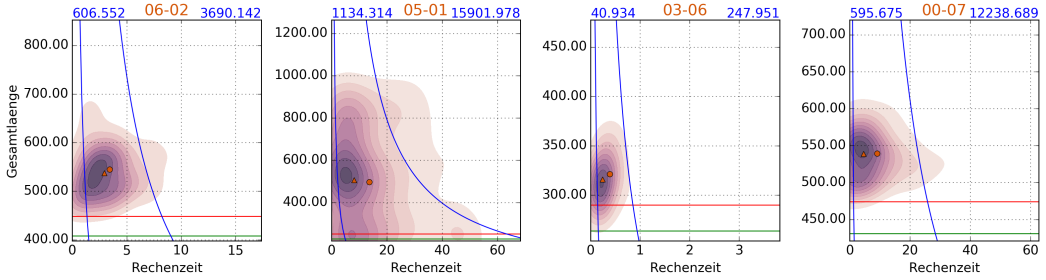

(a) Ergebnisse - Informed RRT*-Planer - OvGU-Karte




(b) Ergebnisse - FMT*-Planer - OvGU-Karte




(c) Ergebnisse - RRT*-Planer - OvGU-Karte




(d) Ergebnisse - T-RRT-Planer - OvGU-Karte

Abb. 5.11: Darstellung der Ergebnisse auf der OvGU-Karte

Tabelle 5.1: Anzahl der erfolgreichen Planungen bei 100 Versuchen

| | 06-02 | 05-01 | 03-06 | 00-07 |
|---|---|---|---|---|
| **Informed RRT*** | 85 | 28 | 94 | 79 |
| **FMT*** | 96 | 88 | 98 | 89 |
| **RRT*** | 94 | 94 | 98 | 92 |
| **T-RRT** | 88 | 82 | 95 | 75 |

Insgesamt lässt sich bei der Betrachtung der Hyperbeln feststellen, dass in diesem Experiment der FMT* Planer bei annähernd allen Planungsaufgaben günstigere Ergebnisse im Hinblick auf Planungszeit und Gesamtlänge erzielte. Danach folgen der T-RRT Planer mit schnelleren Planungszeiten sowie höheren Gesamtlängen und der RRT* mit längerer Rechendauer und kürzerer Gesamtlängen. Der Informed RRT* Planer weißt neben einer hohen Ausfallrate bei der Aufgabe 05-01 ebenfalls eine schlechtere Performance gegenüber den anderen auf.

| | 06-02 | 05-01 | 03-06 | 00-07 |
|---|---|---|---|---|
| **InformedRRTstar** | 23466.58 | 11622.84 | 10184.83 | 14341.05 |
| **FMT** | 2359.67 | 7175.54 | 1405.26 | 6941.66 |
| **RRTstar** | 13442.42 | 14982.66 | 2731.92 | 10258.83 |
| **TRRT** | 3690.14 | 15901.98 | 247.95 | 12238.69 |

Abb. 5.12: Vergleich der Hyperbelkonstanten - OvGU-Karte

## 5.6 Fazit

Eines der Ziele dieser Arbeit war es, anhand der Betrachtung der OMPL-Planer in geometrischen Szenarien Regelsätze aufzustellen, welche zur Umsetzung einer automatisierten Planerauswahl beitragen. Mittels der vorgestellten Experimente können Aussagen getroffen werden, die eine Einschätzung über die OMPL und ihre Planungsalgorithmen zulassen und aus denen sich die genannten Regelsätze ableitet.

1. **Für die Durchführung einer Planung, bei der die Start- und Zielpose im Konfigurationsraum nahe beieinander liegen, ist die Wahl des Planungsalgorithmus belanglos.** Wie in Experiment

2 gezeigt wurde, finden alle Planer für diese Aufgabe innerhalb kürzester Zeit optimale Wege. Die Ausnahme bildet hier der T-RRT Algorithmus.

2. **Bei Pfadplanungsaufgaben in der Ebene, in denen die Rotation berücksichtigt wird, sind in Bezug auf die Performance die *tree* basierenden den *roadmap* basierenden Planern der OMPL vorzuziehen.** Dies hat sowohl der Hyperbelvergleich in Experiment 2 als auch die Tatsache, dass es den SPARS Planern in Experiment 3 nicht gelungen ist innerhalb der vorgegebenen Zeit Pfade zu finden, gezeigt.

3. **Die Größe des Suchraums und die Einschränkungen des Konfigurationsraums wirken sich signifikant auf die Planungszeiten aus.** Das wird durch die gravierenden Unterschiede in der Performance der Planungsalgorithmen in den beiden Szenarien deutlich.

4. Neben der Auflösung **wirkt sich vor allem die Beschaffenheit der Karten zwischen den Start- und Zielposen auf die Planungsdauer und die geplante Weglänge der Pfade aus.** Daher ist es bei der Umsetzung einer automatischen Planerauswahl essentiell, eine Metrik zu entwickeln, welche die verschiedenen Charakteristiken der jeweils eingesetzten Karte bewertet. In den Experimenten konnten anhand der Ergebnisse Pfade erkannt werden, die sich in ihrem Anspruch an die Planer deutlich unterscheiden.

5. **Die geschaffenen Flächen unterhalb der Hyperbeln dienen als effektives Mittel zur Bewertung der Ergebnispfade.** Allerdings müssen die genannten Einschränkungen des Vergleichs berücksichtigt werden. Zudem muss untersucht werden, wie die Hyperbelformel anzupassen ist, um den Einfluss bestimmter Randbedingungen einzubeziehen.

6. **Die Auswirkungen verschiedener Parameter auf die Leistung der Planer muss weiter untersucht werden.** Durch die in Abschnitt 5.1 angesprochenen Umstände konnten keine Tendenzen festgestellt werden.

Die Regelsätz sind unter Berücksichtigung des Umfangs dieser Arbeit anzusehen und sollten durch weitere Experimente geprüft werden. Alle erhobenen Daten der durchgeführten Experimente werden auf die im Anhang beigelegten CD für weitere Auswertungen übertragen.

Im Kapitel 6 wird auf die Möglichkeiten zur Fortsetzung der Arbeit eingegangen.

# 6 Zusammenfassung und Ausblick

In diesem Kapitel werden die Ergebnisse der Arbeit zusammengefasst und es wird ein Ausblick gegeben, wie die Forschung fortgesetzt werden kann.

## 6.1 Zusammenfassung

In der vorliegenden Arbeit wurde eine innovative Schnittstelle zwischen der OMPL und dem ROS *Navigation Stack* geschaffen. Dabei wurde ein Plug-in für die *move_base* geschrieben, welches die Einbindung der OMPL-Planer ermöglicht. Dieses unterschiedet sich durch die Berücksichtigung der Orientierung und der komplexen Validitätsabfrage deutlich von den in der *move_base* vorhandenen Planungs-Plug-ins. Zusätzlich erweitert es den ROS *Navigation Stack* um eine Pfadplanung für nicht-holonome mobile Systeme. Darüber hinaus nimmt das Plug-in von jedem geplanten Pfad Daten, wie Weglänge, Winkeländerung und Abstand zu Objekten in der Umgebung auf und stellt diese für eine Auswertung zur Verfügung.

Zudem wurde ein Konzept für den Vergleich der Planungsalgorithmen geschaffen und umgesetzt. Dieses beinhaltet die Kreation von Testszenarien inklusive Karten, Anfahrtsposen und Wegen zwischen diesen. Es erfolgte die Erarbeitung eines Kriterienkatalogs, anhand dessen die geplanten Wege bewertet und verglichen wurden. Dies geschah in einer Versuchsanordnung bestehend aus vier sequenziell durchgeführten Experimenten.

Die Experimente dienten der Erkenntnisgewinnung über das Verhalten und die Performance der Planer. Es ist ein Bewertungsansatz auf Grundlage von Hyperbeln entwickelt worden, der die Ergebnisse der Planungen sowohl in Planungszeit und Pfadlänge vergleichbar machte als auch konkrete Aussagen über die Hierarchie der Planer in den Szenarien zuließ.

Allerdings konnten während der praktischen Umsetzung Einschränkungen festgestellt werden. Es war nicht möglich, drei Planungsalgorithmen und einen

Konfigurationsraum der OMPL zu nutzen. Somit konnte der Einfluss von Parameteränderungen an den Planern nicht im ausreichendem Umfang geprüft werden, um konkrete Aussagen zu treffen.

Im Zuge der Auswertung der Experimente sind Regelsätze aufgestellt worden, welche das Verhalten der Planer auf verschiedene Bedingungen beschreiben.

## 6.2 Ausblick

Wie bereits in der Motivation beschrieben, stellt die Arbeit den ersten Schritt in Richtung einer automatisierten Planerauswahl dar. Von diesem Punkt aus ergeben sich diverse Möglichkeiten die gewonnen Erkenntnisse auszubauen.

Anhand der entstandenen Regelsätze sollten weitere Versuchsanordnungen gestaltet werden, um diese zu erweitern. Da die Planer auf probabilistischen Verfahren basieren, sind sehr große Testreihen nötig, um konkrete Aussagen treffen zu können. Es ist möglich, die benötigte Menge an Daten durch weitere Planungsreihen in den bereits vorgestellten Szenarien bzw. auf neu aufgenommenen Karten zu erzeugen.

Des Weiteren müssen die aufgetretenen Probleme mit den Planern und dem Konfigurationsraum der OMPL adressiert und behoben werden, um sie für weitere Versuche im vollen Umfang nutzen zu können. Mit dem erarbeiteten Plug-in ist der Grundstein für die Nutzung der OMPL in ROS bereits gelegt. Zudem ist die OMPL ein erweiterbares *open source* Projekt. Deshalb wäre es sinnvoll, neue Planungsalgorithmen und Konfigurationsräume entsprechend den eigenen Bedürfnissen einzupflegen.

Darüber hinaus muss die ROS *move_base* um einen passenden lokalen Planer erweitert werden, welcher die errechneten Posen nutzt und den Umgang mit mobilen sowie unvorhergesehenen Hindernissen definiert. Das gilt sowohl für omnidirektionale als auch für Robotersysteme mit kinematischen Einschränkungen. Anhand der Ergebnisse dieser Arbeit ist zu erkennen, dass bei kleinen Suchräumen schnell optimale Ergebnisse verzeichnet werden können. Aus diesem Grund würde sich ebenfalls der Einsatz der OMPL anbieten.

Insbesondere sollte das entstandene Plug-in in den realen Projekten getestet werden. Bisher war es mit dem *Navigation Stack* nicht möglich, Pfade

für nicht-holonome Robotersysteme zu planen. Das Plug-in bietet somit innovative Möglichkeiten für den Einsatz des AFiUS Fahrrades. Es ist davon auszugehen, dass Anwendungstests dementsprechend zu einer Vielzahl wichtiger Erkenntnisse führen.

Während der Experimente wurde unter anderem der Einfluss von anspruchsvollen Pfaden auf die Leistung der Planer angesprochen. Für die automatisierte Planerauswertung ist es unabdingbar eine Metrik zu schaffen, die automatische Bewertungen von Karten in Hinblick auf den Schwierigkeitsgrad ermöglicht.

Hierzu würde es sich anbieten, Kartendarstellungen auf Charakteristiken zu untersuchen, welche wiederkehrende Effekte bei der Pfadplanung aufweisen. Bestimmte Parameter, wie beispielsweise das Verhältnis von freien Flächen zu Hindernissen und der mittlere Abstand aller Objekte auf der Karte, sind hierbei besonders zu berücksichtigen. Zusätzlich muss das Konzept der costmaps überdacht und möglicherweise um neue Eigenschaften erweitert werden. Gegebenenfalls lassen sich, auf dieser Ebene der Navigation, Verkehrsregeln für die Pfadplanung aufstellen. Die Anwendbarkeit für urbane Szenarien, wie dem AFiUS-Projekt, mit mehreren Verkehrsteilnehmern und schwierigen Passagen wird dementsprechend verbessert. Zusätzlich profitieren industrielle und logistische Anwendungsgebiete ebenfalls von einer Verbesserung der Kartendarstellungen.

# Literaturverzeichnis


[1] Nancy M Amato and Yan Wu. A randomized roadmap method for path and manipulation planning. In *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, volume 1, pages 113–120. IEEE, 1996.

[2] Jerome Barraquand and Jean-Claude Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6):628–649, 1991.

[3] Sarah Bergbreiter and Kristofer SJ Pister. Design of an autonomous jumping microrobot. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 447–453. IEEE, 2007.

[4] Robert Bohlin and Lydia E Kavraki. Path planning using lazy prm. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 521–528. IEEE, 2000.

[5] Vitaly Bokser, Carl Oberg, G Sukhatme, and Aristides Requicha. A small submarine robot for experiments in underwater sensor networks. *Center for Embedded Network Sensing*, 2004.

[6] H. Choset. Probabilistic roadmap path planning. Präsentation MIT Press.

[7] Didier Devaurs, Thierry Siméon, and Juan Cortés. Enhancing the transition-based rrt to deal with complex cost spaces. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4120–4125. IEEE, 2013.

[8] Andrew Dobson and Kostas E Bekris. Improving sparse roadmap spanners. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 4106–4111. IEEE, 2013.

[9] Andrew Dobson, Athanasios Krontiris, and Kostas E Bekris. Sparse roadmap spanners. In *Algorithmic Foundations of Robotics X*, pages 279–296. Springer, 2013.

[10] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2997–3004. IEEE, 2014.

[11] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3067–3074. IEEE, 2015.

[12] Erico Guizzo and Evan Ackerman. Boston dynamics officially unveils its wheel-leg robot: "best of both worlds". *IEEE Spectrum*, 2007. Website: `http://spectrum.ieee.org/automaton/robotics/humanoids/boston-dynamics-handle-robot`.

[13] Alex Henning. costmap_2d. `http://wiki.ros.org/costmap_2d`, 2015. [Online, visited 27.03.2017].

[14] Nico Hochgeschwender, Robin Kammel, Gerhard Kraetzschmar, Walter Nowak, Asadollah Norouzi, Benjamin Schnieders, and Sebastian Zug. *RoboCup@Work Rulebook*. RoboCup Technical Committee, Januar 2017.

[15] Auke Jan Ijspeert, Alessandro Crespi, Dimitri Ryczko, and Jean-Marie Cabelguen. From swimming to walking with a salamander robot driven by a spinal cord model. *science*, 315(5817):1416–1420, 2007.

[16] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International journal of robotics research*, 34(7):883–921, 2015.

[17] Mao Jun, Li Jian-gang, Li Wei-kang, and Hao Zhi-yong. Kinematics analysis and simulation of profiling memory cutting for tunnel robot in complex environment. *Procedia Earth and Planetary Science*, 1(1):1411–1417, 2009.

[18] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[19] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[20] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[21] Yik San Kwoh, Joahim Hou, EA Jonckheere, and Samad Hayati. A robot with improved absolute positioning accuracy for ct guided stereotactic brain surgery. *IEEE Transactions on Biomedical Engineering*, 35(2):153–160, 1988.

[22] Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

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

[24] Yanbo Li, Zakary Littlefield, and Kostas E Bekris. Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5):528–564, 2016.

[25] Maxim Likhachev, David I Ferguson, Geoffrey J Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *ICAPS*, pages 262–271, 2005.

[26] JH Long, TJ Koob, K Irving, K Combie, V Engel, N Livingston, A Lammert, and J Schumacher. Biomimetic evolutionary analysis: testing the adaptive value of vertebrate tail stiffness in autonomous swimming robots. *Journal of Experimental Biology*, 209(23):4732–4746, 2006.

[27] David V. Lu. move_base. `http://wiki.ros.org/move_base`, 2016. [Online, visited 27.03.2017].

[28] marketsandmarkets.com. Mobile robots market by environment (aerial, ground, and marine), component (hardware and software), application

(professional service and personal service), and geography (north america, europe, asia-pacific, and the row) - global forecast to 2020. *marketsandmarkets.com*, 2015.

[29] Kenzo Nonami, Farid Kendoul, Satoshi Suzuki, Wei Wang, and Daisuke Nakazawa. *Autonomous flying robots: unmanned aerial vehicles and micro aerial vehicles*. Springer Science & Business Media, 2010.

[30] Hauke Petersen. Pfadplanung und ausführung für einen mobilen roboter im kontext des robocup wettbewerbs, 2015.

[31] Marc Raibert, Kevin Blankespoor, Gabriel Nelson, Rob Playter, and T Bigdog Team. Bigdog, the rough-terrain quadruped robot. In *Proceedings of the 17th world congress*, volume 17, pages 10822–10825. Proceedings Seoul, Korea, 2008.

[32] James Reeds and Lawrence Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific journal of mathematics*, 145(2):367–393, 1990.

[33] Oren Salzman and Dan Halperin. Asymptotically near-optimal rrt for fast, high-quality motion planning. *IEEE Transactions on Robotics*, 32(3):473–483, 2016.

[34] Bruno Siciliano and Oussama Khatib, editors. *Springer handbook of robotics*. Springer, 2016.

[35] Roland Siegwart, Illah Reza Nourbakhsh, and Davide Scaramuzza. *Introduction to autonomous mobile robots*. MIT press, 2011.

[36] Harold L. Sirkin, Michael Zinser, and Justin Rose. The robotics revolution: The next great leap in manufacturing. *bcg.perspectives*, 2015. `https://www.bcgperspectives.com/content/articles/lean-manufacturing-innovation-robotics-revolution-next-great-leap-manufacturing/`.

[37] Ioan A Sucan, Mark Moll, and Lydia E Kavraki. The open motion planning library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, 2012.

[38] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[39] Sebastian Zug and Martin Seidel. Autonome Fahrräder in Urbanen Szenarios Projektantrag. Otto-von-Guericke Universität Magdeburg.

[40] Dubins path. http://planning.cs.uiuc.edu/node788.html, 2012. [Online, visited 27.03.2017].

[41] Ls3 - legged squad support systems. http://www.bostondynamics.com/robot_ls3.html, 2016. [Online, visited 27.03.2017].

[42] Ros history. http://www.ros.org/history/, 2016. [Online, visited 27.03.2017].

[43] Syma x8c venture. http://mainwww.chiassociatesinc.netdna-cdn.com/wp-content/uploads/2014/11/Screen-Shot-2015-10-30-at-1.14.32-PM.png, 2016. [Online, visited 27.03.2017].

[44] about ros. http://www.ros.org/about-ros/, 2017. [Online, visited 25.03.2017].

[45] Api overview. http://ompl.kavrakilab.org/api_overview.html, 2017. [Online, visited 27.03.2017].

[46] Available planners. http://ompl.kavrakilab.org/core/planners.html, 2017. [Online, visited 27.03.2017].

[47] Boston dynamics handle. https://img.vidible.tv/prod/2017-02/28/58b4bef74d96932fb5538872/58b4c0a7ad5cd352eab435c3_o_U_v1.png?w=636&h=361, 2017. [Online, visited 27.03.2017].

[48] irobot roomba. http://www.irobot.de/Haushaltsroboter/staubsaugen, 2017. [Online, visited 27.03.2017].

[49] Karte Elbauenpark. https://www.google.de/maps/@52.138632,11.672518,16z?hl=de, 2017. [Online, visited 27.03.2017].

[50] Karte Universität. https://www.google.de/maps/@52.1392835,11.6464737,240m/data=!3m1!1e3, 2017. [Online, visited 27.03.2017].

[51] Mikrophone. https://www.geras-it.de/documents/image/13/138/138.jpg, 2017. [Online, visited 25.03.2017].

[52] Omni-wheel. http://www.microrobo.com/images/100mm-aluminum-mecanum-wheel-right-14092-3.jpg, 2017. [Online, visited 25.03.2017].

[53] Robocup german open 2017. `https://www.robocupgermanopen.de/de/major/atwork`, 2017. [Online, visited 25.03.2017].

[54] Softbank aldeberan nao. `https://www.ald.softbankrobotics.com/en/cool-robots/nao`, 2017. [Online, visited 27.03.2017].

[55] Softbank aldeberan pepper. `http://www.haijapan.com/article/details/593`, 2017. [Online, visited 27.03.2017].

[56] Ultraschallsensor. `http://www.mikrocontroller-elektronik.de/wp-content/uploads/2016/03/Ultraschallsensor-HC-SR04-Ultrasonic-Sensor-800x534.jpg`, 2017. [Online, visited 25.03.2017].

# Selbstständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als den angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Diese Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Magdeburg, den 30. März 2017 B. Sc. Hauke Petersen