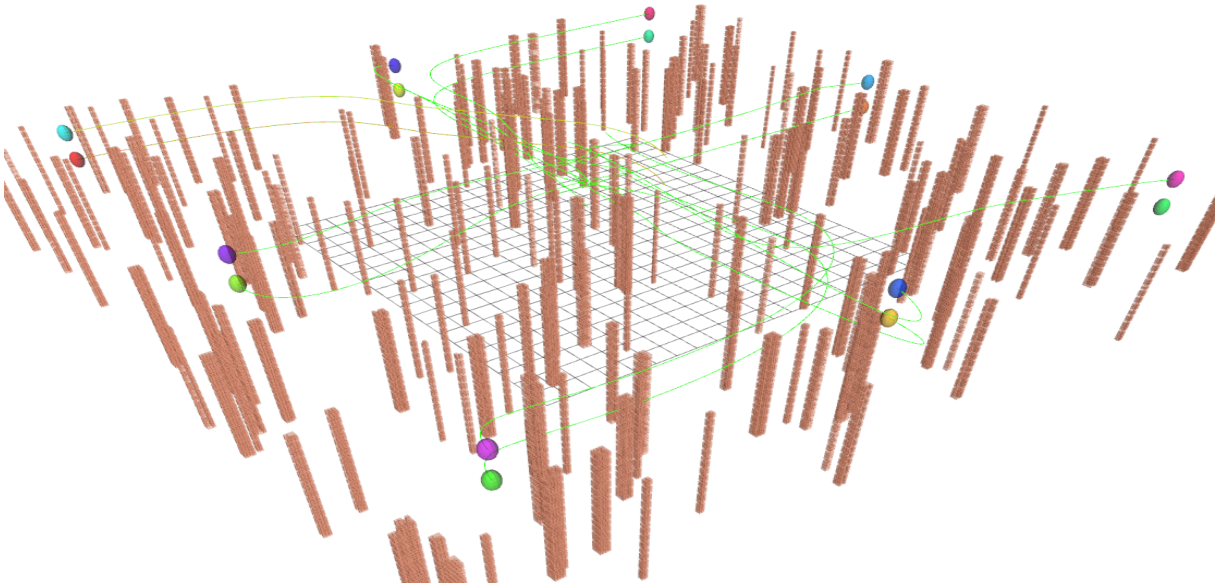
Konzeptionierung und Umsetzung eines Frameworks zur dreidimensionalen Routenplanung einer Flotte autonomer Flugroboter

Masterarbeit im Studiengang Mechatronik

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl für Fertigungsautomatisierung und Produktionssystematik
Prof. Dr.-Ing. J. Franke



Bearbeiter: Dong Wang Matrikelnr.: 22453645

Betreuer: Prof. Dr.-Ing. J. Franke
M.Sc M. Lieret

Abgabetermin: 07.09.2020

Bearbeitungszeit: 6 Monate

Erklärung

Ich versichere, dass ich die vorliegende Arbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den 30. Juli 2020

Dong Wang

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

3D	statische Methode
4D	dynamische Methode
ACO	Ameisenkolonie-Optimierung
BIP	binär-lineare Programmierung
EA	Evolutionary Algorithm
ECBS	Enhanced CBS
FCL	Flexible Collision Library
GA	genetischer Algorithmus
HIL	Hardware in the Loop
ILP	integrale lineare Programmierung
LQG	linear-quadratic-gaussian
MA	memetischer Algorithmus
MAPF	Multi-Agent Pfadfindung
MAVLink	Micro Air Vehicle Link
MD5	Message-Digest Algorithm 5
MILP	Mixed-Integer-lineare-Programmierung
MIQP	gemischt-ganzzahlige quadratische Programmierung
NLP	nichtlineare Programmierung
NN	neuronales Netz
ODE	Open Dynamics Engine
PRM	Probabilistic Road Maps
PSO	Partikelschwarmoptimierung
QP	quadratische Programmierung
ROS	Robot Operating System
RPAS	ferngesteuerten Flugzeugsystemen
RRT	Rapidly-exploring random tree
RRT*	Rapidly-exploring random tree*
RSFC	der relativ sichere Flugkorridor
SAPF	Single-Agent Pfadfindung
SCP	sequenzielle konvexe Programmierung
SFC	der sichere Flugkorridor
SFLA	shuffled frog-leaping algorithm
SIL	Software-in-the-Loop
TCP/IP	Transmission Control Protocol/Internet Protocol

UAV	Unbemannte Flugroboter
UDP	User Datagram Protocol

1 Einleitung

1.1 Hintergrund

Unbemannte Flugroboter (UAV) wurden in der Vergangenheit ausschließlich vom Militär eingesetzt. Inzwischen wurde der Einsatz von ferngesteuerten Flugzeugsystemen (RPAS) und kleinen Drohnen ausgeweitet, um zivile Aufgaben wie die Unterstützung von Such- und Rettungsaktionen [14], die Überwachung von Wetterlagen und Verkehrsströmen und die Bereitstellung von Gütern zu erledigen [15] und um als Plattform für Luftaufnahmen zu dienen. Drohnen werden eingesetzt, um Veränderungen in der Umwelt zu bewirken. Ein gutes Beispiel ist die Landwirtschaft, in der durch den Einsatz von Drohnen zum Besprühen von Feldern und zum Verfolgen von Pflanzenwachstumsmustern Effizienzsteigerungen erzielt werden können [16].

Einen der vielversprechendsten Einsatzbereiche für Drohnen stellen Logistiksysteme dar. Als Alternative zu starren Logistiksystemen und flurgebundenen Transportsystemen bietet sich daher der Einsatz autonomer Flugroboter zum Warentransport an. Hierdurch wird die Intralogistik um die dritte Dimension erweitert, zusätzlich kann der bislang ungenutzte Raumbereich oberhalb der bestehender Produktionssysteme in den Materialfluss einbezogen werden. Das entstehende mehrdimensionale Fördersystem zeichnet sich weiterhin durch eine hohe Flexibilität sowie hohe Geschwindigkeit aus. Der Einsatz von Drohnen könnte die Arbeitskosten drastisch senken und wird als potenzieller Störfaktor für die konventionelle Paketzustellungsbranche angesehen. Online-Händler und Lieferfirmen wie Amazon, DHL, FedEx, JD und Alibaba melden bereits Patente für die Entwicklung der mehrstufigen Erfüllung für UAV oder „Drohnen-Bienenstöcke“ an, die die Bereitstellung dieser Technologie in einer gebauten Umgebung ermöglichen würden [15][17].Es wurden in den letzten Jahren umfangreiche Forschungsarbeiten zum möglichen Einsatz von Drohnen in der Paketzustellung durchgeführt, vor allem im Bereich der logistischen Optimierung [18]. Zudem versprechen sich Logistikunternehmen von der Verwendung von Drohnen die Lösung des Problems der letzten Meile. Bislang stellt der Transportschritt vom letzten Logistikstützpunkt zum Empfänger den aufwendigsten und damit kostenintensivsten Abschnitt der Logistikkette dar[19].

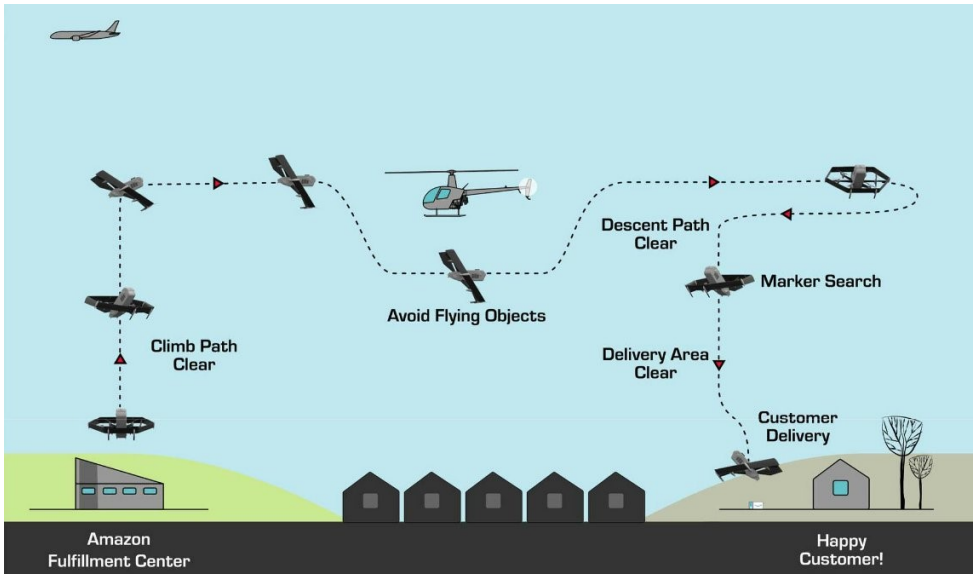


Abbildung 1.1: Lieferung durch Amazon Prime Air[1]

Abbildung 1.2: DHL Paketkopter[2]

1.2 Problemstellung

Der effiziente und zuverlässige Einsatz autonomer Multikopter in Industrieumgebungen setzt geeignete Routen voraus. Basierend auf einer vorab generierten dreidimensionalen Karte der Umgebung sowie während des Fluges erfasster Sensorinformationen wird eine kollisionsfreie Flugbahn durch die Fabrikhalle berechnet. Die Berechnung eines solchen Pfades mit minimalen Kosten von einem bekannten Startzustand zu einem bekannten Zielzustand für einen einzelnen Agenten unter Berücksichtigung von Hindernissen und unter Zeitbeschränkung wird als Single-Agent Pfadfindung (SAPF) bezeichnet [20].

Bei der Betrachtung der Zusammenarbeit mehrerer Drohnen müssen nicht nur die statischen Hindernisse aus der Karte berücksichtigt werden, sondern auch mögliche Kollisionen zwischen den UAV-Gruppen in einem Multi-Agent-System. Jede Drohne in diesem Multi-Agent-System hat ihren eigenen Ausgangspunkt und ihr eigenes Ziel und muss ihre Aufgaben unabhängig oder in Zusammenarbeit ausführen, und zwar ohne Störung anderer Drohnen. Die Routenplanung in einer komplexen Umgebung innerhalb kurzer Zeit und unter Vermeidung von Kollisionen mit Hindernissen oder anderen Agenten wird als Multi-Agent Pfadfindung (MAPF) bezeichnet [21][22][23].

Unterschieden wird dabei zwischen globaler Planung und lokaler Planung. Wenn die Umwelt bekannt ist, kann eine globale Pfadplanung offline stattfinden bevor die Roboter sich in Bewegung setzen. Die lokale Planung wird normalerweise online vollzogen und zielt darauf ab, dass die Roboter in Echtzeit Hindernisse vermeiden. Die Pfadplanung beschränkt sich auf den Hüllkörper um den Roboter herum mit dem Zweck, die nächste Bewegung des Roboters zu berechnen und unbekannte oder dynamische Hindernisse in der Nähe zu vermeiden und dabei die weit entfernten Hindernisse zu ignorieren, um eine schnelle Planung zu erreichen [24][11]. Diese Arbeit konzentriert sich auf das MAPF-Problem in bekannter Umgebung und im Offlinebetrieb.

1.3 Ziel der Arbeit

In dieser Arbeit soll ein Framework für die dreidimensionale Routenplanung einer Flotte von Flugrobotern vorgestellt werden. Folgendes sind die Ziele dieser Masterarbeit:

- Erarbeitung einer Lösung zur dreidimensionalen Routenplanung einer variablen Anzahl von Flugrobotern.
- Vorgegeben seien dabei stets eine Umgebungskarte in Form einer OctoMap, sowie die Start- und Zielpunkte der einzelnen Roboter.
- Ergebnis der Routenplanung sollen roboterspezifische Trajektorien sein, die kollisionsfrei sowie nach Flugstrecke optimiert sind.
- Die Kollisionsfreiheit kann dabei durch überschneidungsfreie Trajektorien oder zeitliche Anpassung erzielt werden.
- Bewertung der Robustheit und Leistungsfähigkeit des umgesetzten Routenplaners mittels Simulation, Ableitung von Ansätzen zur Optimierung der Flugbahnen und der zur Pfadgenerierung erforderlichen Rechenzeiten.

1.4 Aufbau und Vorgehensweise

Kapitel 2 bietet einen Literaturüberblick, um die aktuell vorherrschenden Ansätze zur Lösung dieses Problems aufzuzeigen. Kapitel 3 und 4 präsentieren die Grundlagen für die Entwicklung der angestrebten Systeme. Dazu zählen neben Robot Operating System, OctoMap, Mavros und Gazebo auch bestehende Algorithmen zur Routenplanung. Auch die relativen Bernsteinpolynome werden dargestellt. Der Hauptteil umfasst die Kapitel 5 und 6, die sich im Aufbau ähneln. In Kapitel 5 werden zwei Methoden beschrieben, um das MAPF-Problem zu lösen, nämlich die statische Methode (3D) ohne Berücksichtigung der Zeitdimension und die dynamische Methode (4D) mit Zeitdimension. Diese beiden Methoden werden in Kapitel 6 entwickelt und implementiert. Dem folgt in Kapitel 6 eine Übersicht der Simulationsarchitektur der beiden Methoden. Das Hauptaugenmerk bei der Entwicklung gilt der Bewertung der zwei Methoden. Weitere bedeutende Faktoren sind Rechenzeiten, Robustheit, Leistungsfähigkeit und Pfadkosten sowie Energieverbrauch. Zusätzlich werden die Vor- und Nachteile sowie Anwendungsfelder der Methoden vorgestellt. In Kapitel 6 fasst ein Fazit alle Ergebnisse der Evaluation in kurzer Form zusammen und bewertet anhand dieser das entwickelte Softwaremodul hinsichtlich Leistungsfähigkeit, Zuverlässigkeit und Anwendungsintegration. Kapitel 7 fasst abschließend die Anregungen für die weitere Forschung zusammen. Den Abschluss bildet ein kurzer Lebenslauf des Verfassers.

2 Literaturübersicht

2.1 zentralisierte Optimierungsmethode

Grundsätzlich gibt es zwei akademische Lösungsansätze für das MAPF-Problem: die zentralisierte Optimierungsmethode und die verteilte Optimierungsmethode [25][26]. In [27] formulierte D. Mellinger die zentralisierte Optimierungsmethode in ganzzahligen Einschränkungen für die quadratische Programmierung mit gemischten Ganzzahlen (MIQP) um. Aufgrund der rechnerischen Komplexität des MIQP sind jedoch mehr als 500 Sekunden erforderlich, um die Flugbahn von vier Agenten zu generieren. In [28] wird die sequenzielle konvexe Programmierung (SCP) vorgeschlagen, um die nicht konvexen Bedingungen durch konvexe zu ersetzen. SCP zeigt gute Leistung bei der Planung einer kleinen Anzahl von Quadrotoren, ist aber ungeeignet für ein großes Team und eine komplexe Umgebung. Robinson D Reed hat in [29] die nichtlineare Programmierung (NLP) mit sequenzieller Planung kombiniert, um nichtlineare Einschränkungen direkt zu behandeln. Diese Verwendung der sequenziellen Planungsmethode ermöglicht eine höhere Skalierbarkeit. Eine Einschränkung ist jedoch, dass für eine komplizierte Umgebung, z. B. überfülltes Lager, keine Lösung gefunden werden kann.

2.2 verteilte Optimierungsmethode

Eine verteilte Optimierungsmethode wird ebenfalls in Betracht gezogen, um die Gesamtplanungszeit durch Verteilung der Rechenlast zu reduzieren. Ansätze basierend auf linear-quadratic-gaussian (LQG) Hindernis [30][31] und gepuffertem Voronoi-Zellen [32] zeigen, dass ein kollisionsfreier Pfad in Echtzeit erzeugt werden kann. Solche verteilten Methoden vermögen jedoch nicht, Vollständigkeit und Abwesenheit von Deadlocks zu gewährleisten. In [33] schlagen Yu and LaValle eine Methode vor, die sowohl optimale Lösungen als auch eine hohe Effizienz garantiert. Um dieses Ziel zu erreichen, entwerfen sie basierend auf der integralen linearen Programmierung (ILP) neuartige und vollständige Algorithmen zur Optimierung für jedes der vier Ziele. Dann verbessern sie die Rechenleistung dieser Algorithmen durch die Einführung prinzipieller Heuristik. Die Kombination von ILP-Modell-basierten Algorithmen und Heuristiken erweist sich als äußerst effektiv und ermöglicht die Berechnung optimaler Lösungen für Probleme mit hunderten von Robotern

oft in wenigen Sekunden. Allerdings ist diese Methode auf zweidimensionale Modelle beschränkt. Bei drei- oder sogar vierdimensionalen Modellen wird diese Methode aufgrund der zunehmenden zeitlichen Komplexität extrem zeitaufwendig [33].

Diese Arbeit hat zum Ziel, eine verteilte Optimierungsmethode mit schnellem erkundenden zufälligen Baum (RRT*) in drei Dimensionen und mit konfliktbasiertem Suchen in vier Dimensionen zu entwickeln [11]. Dieses Modell für einen sicheren Flugkorridor (Englisch: safe flight corridor) wurde eingeführt, um den Rechenaufwand zu reduzieren [12].

3 Softwaregrundlagen

In diesem Kapitel werden die zur Umsetzung des in dieser Arbeit entwickelten Softwaremoduls benötigten Grundlagen näher betrachtet. Dazu zählt das Robot Operating System (ROS) zur Steuerung des eingesetzten Flugroboters und zur Ausführung der entwickelten Programmbausteine. Des Weiteren werden die zur Modellierung und Simulation des Flugroboters genutzte Open-Source-3D-Robotersimulator sowie die genutzten Softwarebibliotheken OctoMap und Flexible Collision Library (FCL) vorgestellt. Die OctoMap-Bibliothek implementiert einen 3D-Belegungsgitter-Mapping-Ansatz, der Datenstrukturen und Mapping-Algorithmen in C ++ bereitstellt, die besonders für die Robotik geeignet sind. FCL wird zur Kollisionserkennung verwendet. Schließlich folgt eine Betrachtung der eingesetzten mathematische Programmierlöser für lineare Programmierung.

3.1 ROS

3.1.1 ROS als Softwareframework zur Robotersteuerung

Das als Robot Operating System (ROS) bekannte flexible Software-Framework enthält eine Sammlung von Bibliotheken, Konventionen und Tools zum Programmieren einer Vielzahl von Roboterapplikationen und -anwendungen. Dies vereinfacht die Entwicklung komplexer und robuster Systeme und Automatisierungslösungen für eine Vielzahl von Entwicklungsteams und Roboterplattformen erheblich. Das Framework entstand aus einer Kombination von zwei Forschungsprojekten der Stanford University und der Willow Garage-Softwarearchitektur für die Implementierung von Servicerobotern(Personal Robotics Program). Aufgrund der weitgehend freien Zugänglichkeit und Verfügbarkeit einer großen Anzahl bereits implementierter Softwarelösungen für eine große Anzahl von Robotern und integrierbaren Sensoren, einer weltweit wachsenden Community von Entwicklern und Forschungsteams sowie des Open-Source-Konzepts ist ROS zu einer weitverbreiteten Plattform für eine Vielzahl von Forschungs- und Entwicklungsprojekten der Automatisierungstechnik und Robotik entwickelt geworden [34][35]. Grundsätzlich basiert ROS auf einem Basissystem, das verschiedene Module bereitstellt, um eine effiziente und erfolgreiche Implementierung und Anwendung verschiedener automatisierter Applikationen im Bereich der Robotik zu ermöglichen. Dazu gehören eine standardisierte Kommunikations-

infrastruktur, spezifische Bibliotheken für Robotik und Sensortechnologie sowie Hilfsmittel für die Visualisierung und Diagnose der entwickelten Softwaremodule. Dieses Kernsystem kann bei Bedarf auch mit sogenannten Softwarepaketen erweitert werden, die von der Entwicklergemeinschaft frei bereitgestellt oder vom Benutzer implementiert werden. Derzeit stehen über 3000 öffentlich verfügbare Pakete zur Integration zur Verfügung [36][35].

3.1.2 ROS Master

Das ROS-Framework besteht aus verschiedenen Teilen, nämlich Knoten, Nachricht, Thema, Dienste, und Server. Zur Verwaltung einzelner ROS-Knoten, Dienste, Klienten, Aktionenservern und Aktionenklienten dient der sogenannte ROS-Master. Der ROS-Master stellt den übrigen Knoten im ROS-System Namens- und Registrierungsdienste zur Verfügung. Er verfolgt Verlage und Abonnenten von Themen und Diensten. Die Rolle des Masters besteht darin, einzelnen ROS-Knoten zu ermöglichen, sich gegenseitig zu lokalisieren. Sobald sich diese Knoten gefunden haben, kommunizieren sie miteinander. Dieses zentrale Element jeder ROS-Anwendung steuert die Kommunikation mittels der Knoten, Dienste und Aktionen untereinander. Hierzu müssen sich alle Module vor Programmausführung bei diesem Master registrieren und angeben mit welchen Themen sie zur Laufzeit interagieren wollen. Die Interaktionen können das Senden und Empfangen von Nachrichten, Serviceanfragen und -antworten oder Ziel- und Statusveröffentlichungen enthalten. Auf diese Weise können sich einzelne Paketbausteine finden und eine Kommunikation untereinander aufbauen [35]. Abbildung 3.1 stellt schematisch dar, die bei Master registrierten ROS-Knoten sich erkennen und miteinander kommunizieren.

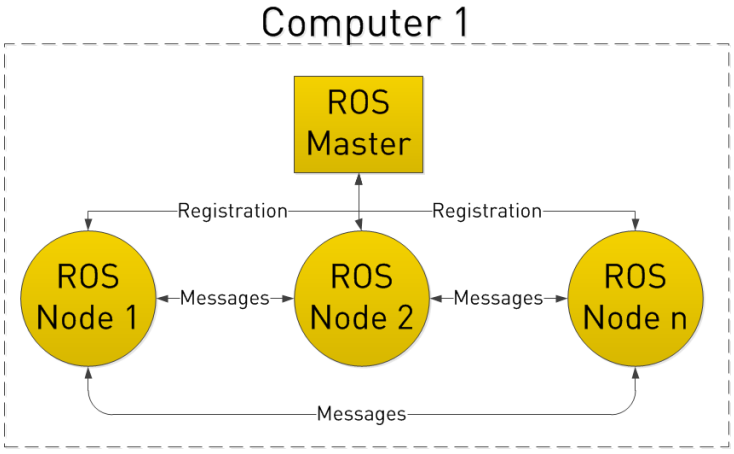


Abbildung 3.1: ROS Master

3.1.3 ROS Knoten

Zum besseren Verständnis des ROS-Prinzips werden in diesem Abschnitt das Grundkonzept und die Funktionsweise des bereitgestellten Kommunikationssystems erläutert. Jede auf dem Roboterbetriebssystem basierende Softwareanwendung besteht normalerweise aus einem oder mehreren sogenannten Knoten. Jeder Knoten stellt ein unabhängiges und ausführbares Programm dar, das beispielsweise einen Algorithmus, eine Berechnung oder andere nützliche Aufgaben zur Erreichung bestimmter Ziele übernimmt. Die einzelnen Programme kommunizieren über standardisierte Nachrichten über das Transmission Control Protocol/Internet Protocol (TCP/IP) und das User Datagram Protocol (UDP) miteinander. Dies stellt unter anderem sicher, dass einzelne Knoten auch auf verschiedenen Computern ausgeführt werden können, die nur in einem Netzwerk verbunden sind. So ist es beispielsweise möglich, Knoten auf unterschiedlichen Computern mit unterschiedlichen Anforderungen an die Leistung der bereitgestellten Hardware auszuführen. Auf diese Weise kann beispielsweise eine an einem Roboter angebrachte Kamera zur Umgebungserkennung auf dem Bordcomputer gestartet werden. Die eigentliche Bildverarbeitung und Weiterverarbeitung der Sensordaten erfolgt jedoch durch einen leistungsstarken Computer, wie Abbildung 3.2. Außerdem können die beim Master registrierten Knoten die verschiedene Themen abonnieren und veröffentlichen, damit die Nachrichten zwischen den Knoten in den zugeordneten Themen gesendet und empfangen werden.

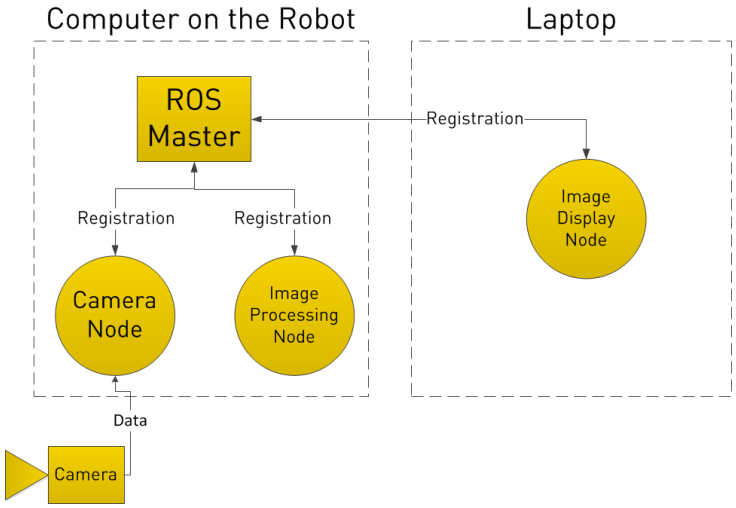


Abbildung 3.2: ROS Knoten

3.1.4 ROS Nachrichten

Nachrichten, die von einem Knoten empfangen oder gesendet werden können, werden in der Regel bestimmten Themenbereichen zugeordnet. Grundsätzlich können die Namen frei

gewählt werden, sollten jedoch durch einen eindeutigen Begriff wie Motor- oder Raddrehzahl beschrieben werden, der die in der Nachricht enthaltenen Daten eindeutig beschreibt. Für Nachrichten, die in der Robotik und Automatisierungstechnik häufig verwendet werden, wie z. B. Position und Positionsinformationen, existieren bereits vordefinierte Datenstrukturen. Weitere Knoten können die genannten Themen abonnieren und haben somit Zugriff auf alle Nachrichten, die dem jeweiligen Thema zugeordnet und veröffentlicht sind. Darüber hinaus können mehrere Themen innerhalb eines Knotens abonniert oder Nachrichten an eine beliebige Anzahl von Themen gesendet werden. Besteht ein Softwaremodul aus mehreren Knoten, ist es möglich, mehrere Nachrichten von verschiedenen Knoten zu einem Thema zu senden und ein Thema mit mehreren Knoten gleichzeitig zu abonnieren. Dieses Knoten- und Themenkonzept ermöglicht eine effiziente und einfache Verknüpfung, und zwar auch für Anwendungen, die in verschiedenen Programmiersprachen entwickelt wurden. Durch die Kapselung der einzelnen Operationen in verschiedenen Knoten wird die Komplexität des zugrunde liegenden Quellcodes im Vergleich zu einem einzelnen Programm erheblich reduziert. Dadurch ist genau bekannt, welche Softwarekomponente von welchem Knoten ausgeführt wird. Die Fehlertoleranz wird auch durch den modularen Aufbau der Software reduziert. Der Absturz eines isolierten Knotens führt normalerweise nicht zu einem vollständigen Systemausfall [35].

3.1.5 ROS Themen

ROS Themen sind Busse, über die Knoten Nachrichten austauschen können. Themen haben eine anonyme abonnieren- und veröffentlichen-Semantik, die die Produktion von Informationen von ihrem Verbrauch entkoppelt. Im Allgemeinen wissen die Knoten nicht, mit wem sie kommunizieren. Stattdessen abonnieren Knoten, die an Daten interessiert sind, das relevante Thema. Knoten, die Daten generieren, veröffentlichen das relevante Thema. Es kann mehrere Herausgeber und Abonnenten eines Themas geben. Das bedeutet, dass die Themen für unidirektionale Streaming-Kommunikation vorgesehen sind. Jedes Thema ist stark vom ROS-Nachrichtentyp abhängig, der zum Veröffentlichen verwendet wird, und Knoten können nur Nachrichten mit einem passenden Typ empfangen. Der Master erzwingt keine Typkonsistenz zwischen den Herausgebern, aber Abonnenten stellen nur dann einen Nachrichtentransport her, es sei denn, wenn die Typen übereinstimmen. Darüber hinaus prüfen alle ROS-Clients, ob eine aus den Nachrichtendateien berechnete kryptografische Hashfunktion vom Typ Message-Digest Algorithm 5 (MD5) übereinstimmt. Diese Überprüfung stellt sicher, dass die ROS-Knoten aus konsistenten Codebasen kompiliert wurden. Abbildung 3.3 zeigt, wie der Kameraknoten eine Nachricht unter dem ROS-Thema image-data veröffentlicht, damit diese von den dieses

Thema abonnierenden Knoten Image-Processing und Image-Display empfangen werden kann.

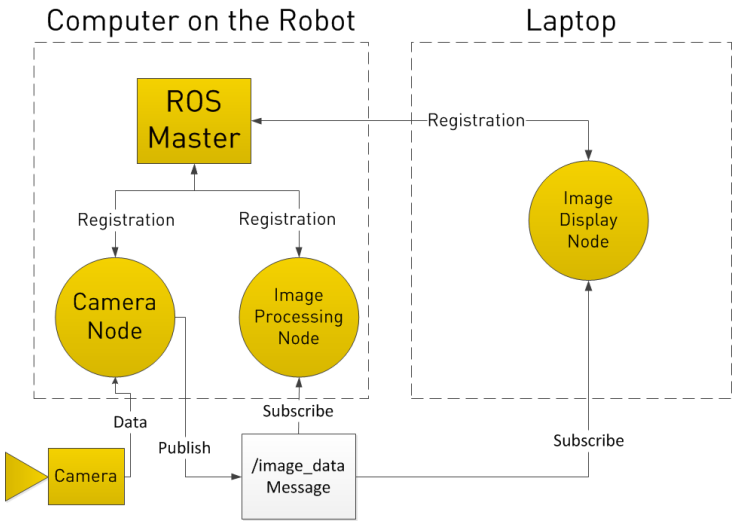


Abbildung 3.3: ROS Themen

3.1.6 ROS Service

Der ROS-Service stellt eine weitere Kommunikationsstruktur zur Verfügung, die vor allem für verteilte Systeme genutzt wird. Dabei wird ein Service von einem Knoten veröffentlicht. Dieser wird mit den bereits erläuterten Themen ebenfalls mit einem eindeutigen Namen versehen. Im weiteren Verlauf der Datenaustausches durch ein eindeutig definiertes Paar an Nachrichten. Hier können Nachrichten in Anfragen und Antworten gesendet werden. So initialisiert ein Knoten einen Service unter den Namen und ein Klient Knoten abonniert diesen Dienst durch das Senden einer Anfragenachricht. Anschließend wartet dieser auf eine eingehende Antwort. Der Service ist dabei nur solange aktiv, bis er die vom Klienten erhaltene Nachricht und das Endergebnis veröffentlicht hat. Bis zu einer erneuten Anfrage veröffentlicht er keine neuen Nachrichten mehr. Als typisches Beispiel für eine solche Servicebeziehung kann ein Bildverarbeitungsalgorithmus wie in Abbildung 3.4 angeführt werden. Dort fordert der Image-Processing Knoten zuerst image_data an, der Kameraknoten sammelt Daten von der Kamera und sendet dann die Antwort.

3.1.7 ROS Server

Ein weiterführendes komplexeres Kommunikationskonzept im Sinne des Service-Klient-Prinzips ist die Struktur Status-Action-Server/-Klient-Struktur. Die Funktionsweise ist

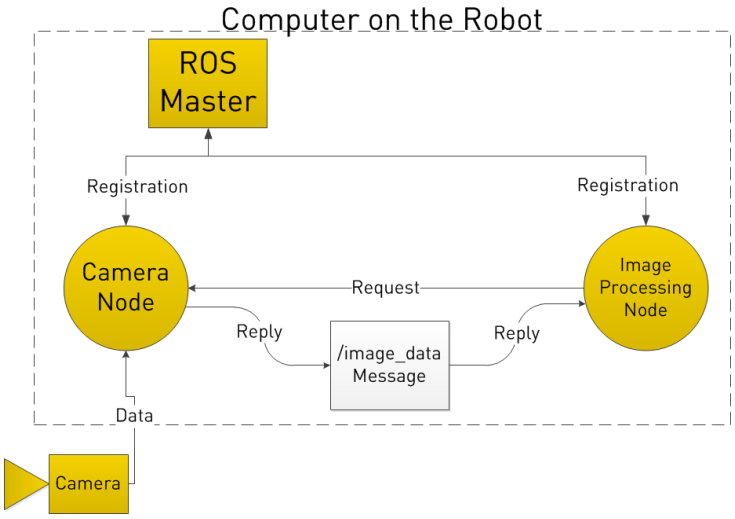


Abbildung 3.4: ROS Service

dabei der vorangehend erläuterten Service-/Klient-Kommunikation sehr ähnlich. Hierbei besteht der Basisaufbau aus einem Server- und einem Klient-Knoten. Der große Unterschied zur Service-Klient-Kommunikation besteht unter anderem darin, dass die vom Action-Server wahrnehmbare Aktion im Verlauf abgebrochen werden kann. Dieses Konzept ist vor allem für Aktionen geeignet, deren Ausführung über einen gewissen Zeitraum hinweg stattfindet. Dies ermöglicht es dem Benutzer, gestartete Aktionen durch direkten Eingriff wieder abzubrechen. Auch wird eine Feedback-Möglichkeit zur Kontrolle bereitgestellt. Die Kommunikation zwischen Action-Server und -Klient erfolgt dabei über das sogenannte „ROS Action Protocol“, welches hierarchisch über den ROS Nachrichten zur Kommunikation zwischen einfachen Knoten angesiedelt ist. Dazu müssen in einem anderen Dokument das zu einer Aktion gehörende Ziel, das Feedback und das Ergebnis festgehalten werden. Diese essenziellen Bausteine werden durch ROS Nachrichtentypen miteinander verknüpft. Nach dem Start des Action-Servers wird dieser aktiv und für den dazugehörigen Action-Klienten sichtbar. Erhält Letzterer eine positive Rückmeldung des Servers, wird ein neues Ziel bestimmt. Dies wird die Funktion des Action-Servers abonniert und damit die Programmausführung durch diese Aufgaben. Dies ist der Fall, der dem Server gehört. Zusätzlich wird für jedes Ziel eine sogenannte Status Maschine angelegt, die Informationen über den Status des zugeordneten Ziels gibt. Je nach gewünschter Komplexität sind hier unterschiedliche Ausgaben möglich. Meist wird allein zwischen den Statuswerten „Aktiv“, „Andauernd“ und „Fertig“ unterschieden. Auch das Senden eines neuen Ziels erfordert einen neuen Programmdurchlauf, da es grundsätzlich nicht möglich ist, dass mehrere Ziele für den gleichen Server aktiv sind. Ist der Programmablauf erfolgreich und somit das eingegebene Ziel erreicht, hat der Action-Server eine bestimmte definierte Ergebnisnachricht. Der Server erlaubt keinen Zugriff durch neue Nachrichten

solange kein neues Ziel zugesendet wird. Ein möglicher Anwendungsfall für die Kommunikation zwischen Action-Server und -Klient stellt das Ansteuern einer Position im Raum durch ein autonomes Fahrzeug dar. Dabei werden als Ziel die Koordinaten der Endposition bestimmt. Während der Fahrt kann das Fahrzeug Feedback beispielsweise in Form der aktuellen Position, Geschwindigkeit und Fahrzeit geben. Ist die Zielposition erreicht, kann diese als Ergebnis veröffentlicht werden, um das Fahrzeug anzuhalten oder ein neues Ziel zu übermitteln [35]. Abbildung 3.5 zeigt ein Beispiel für eine Klientenbeziehung in der Bildverarbeitung, die ausschließlich aktiv wird, sobald die Kamera eine neue Bilddatei übermittelt. Anschließend wird das bearbeitete Bild aus dem Kameraknoten einmalig veröffentlicht. Dann empfangt der Image-Processing-Knoten das bearbeitete Bild unter dem Thema Image_Data.

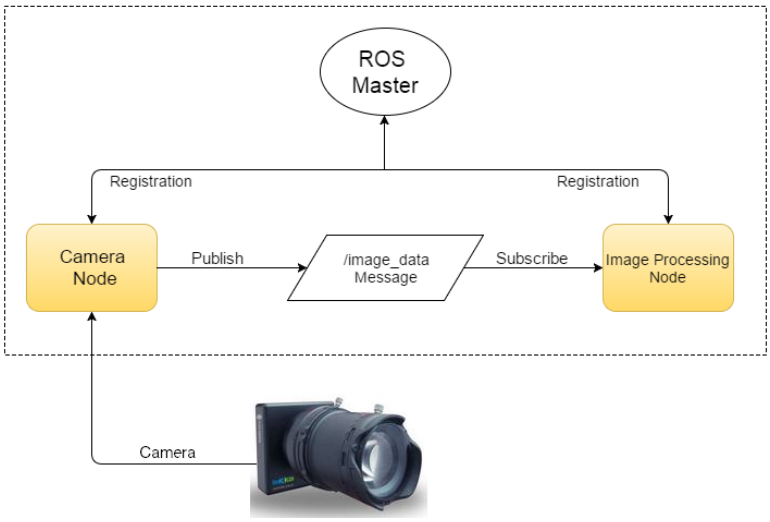


Abbildung 3.5: ROS Server

Die Verknüpfung von Knoten untereinander ist in Abbildung 3.6 illustriert. ROS-Metapakete beinhalten mehrere, thematisch oder funktionell zusammengehörige ROS-Pakete und werden zur Organisation von ROS-Paketen verwendet. ROS-Pakete stellen einen oder mehrere Knoten (Nodes) zur Verfügung, die Datenströme via multicast zu bestimmten Themengebieten (Topics) bereitstellen und auf Datenströme von anderen Knoten zugreifen (publish, subscribe). Knoten können zudem Dienste (Services) anbieten. Ein Dienst realisiert eine Ende-zu-Ende-Kommunikation nach dem Prinzip Anfrage/Antwort. Ein zentraler Master-Knoten registriert und verwaltet alle Knoten sowie deren Themengebiete und Dienste.

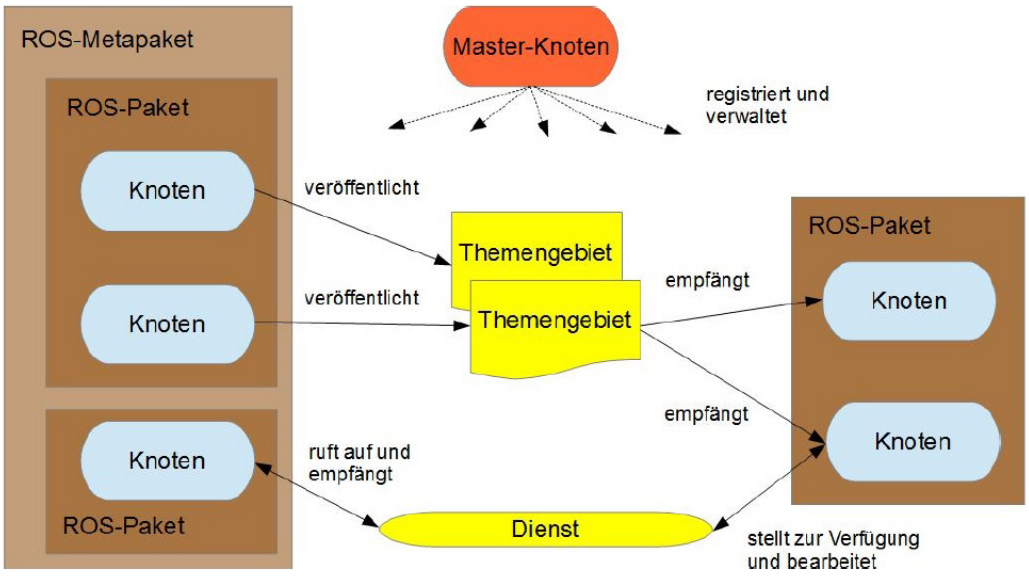


Abbildung 3.6: Schematischer Grundaufbau von ROS

3.2 MAVLink

Um eine Datenkommunikation zwischen dem ROS-Framework und der Drohne herzustellen, mittels der Befehle zur Drohne gesendet und Daten von der Drohne empfangen werden können, wird ein Kommunikationsprotokoll namens Micro Air Vehicle Link (MAVLink) eingeführt.

3.2.1 Grundlagen des MAVLinks-Protokolls

MAVLink ist ein Protokoll hauptsächlich für die Kommunikation mit kleinen unbemannten Fahrzeugen, die als Header-Nachrichten-Marshalling-Bibliothek konzipiert ist. Das MAVlink-Protokoll wurde 2009 von Lorenz Meier von der Computer Vision and Geometry Group der Eidgenössischen Technischen Hochschule Zürich unter der Open-Source-Lizenz LGPL veröffentlicht. Als Open-Source-Kommunikationsprotokoll auf höherer Ebene basiert Mavlink-Protokoll auf serieller Kommunikation, den Standards CAN-Bus und SAE AS-4 und soll eine Sende- und Empfangsregel formulieren und eine Prüfsummenfunktion für die Daten hinzufügen, die häufig verwendet werden, wenn kleine Flugzeuge mit Bodenstationen (oder anderen Flugzeugen) kommunizieren. Das Protokoll definiert die Regeln für die Parameterübertragung in Form einer Nachrichtenbibliothek, die verschiedene Arten unbemannter Luftfahrzeuge, z. B. Startflügelflugzeuge, unbemannte Drehflügler und unbemannte Fahrzeuge unterstützt. Wie das ROS-Framework folgt auch das MAVLink-Protokoll einem modernen hybriden Publish-Subscribe- und Punkt-zu-Punkt-

Entwurfsmuster: Datenströme werden als Themen gesendet/ veröffentlicht, während Unterprotokolle wie das Missionsprotokoll oder das Parameterprotokoll bei erneuter Übertragung Punkt-zu-Punkt sind.

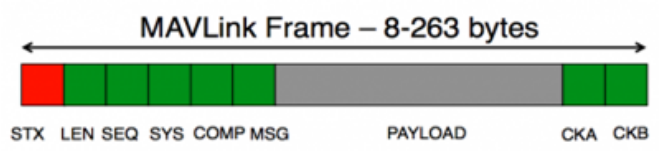
Nachrichten werden als XML-Dateien formatiert. Jede XML-Datei definiert den Nachrichtensatz, der von einem bestimmten MAVLink-System unterstützt wird, das auch als "Dialekt"bezeichnet wird. Der Referenznachrichtensatz, der von den meisten Bodenkontrollstationen und Autopiloten genutzt wird, ist im common.xml-Format definiert (die meisten Dialekte bauen auf dieser Definition auf).

Die MAVLink-Toolchain generiert anhand der XML-Nachrichtendefinitionen MAVLink-Bibliotheken für jede der unterstützten Programmiersprachen. Drohnen, Bodenkontrollstationen und andere MAVLink-Systeme verwenden die generierten Bibliotheken zur Kommunikation. Diese sind in der Regel MIT-lizenziert und können daher in jeder Closed-Source-Anwendung ohne Einschränkungen verwendet werden, ohne den Quellcode der Closed-Source-Anwendung zu veröffentlichen.

3.2.2 Nachrichtenformat

Wie in Abbildung 3.7 gezeigt, hat jeder Nachrichtenrahmen die gleiche Struktur. Das rote und die grünen Felder in der Grafik repräsentieren jeweils ein Datenbyte. Die Länge der Daten im grauen Feld ist nicht festgelegt.

In Version 1.0 wird FE als Startflag(stx) verwendet(rot markiert). Diese Flag ist nützlich, wenn der Empfänger des MAVLink-Nachrichtenrahmens eine Nachrichtendecodierung durchführt. Das zweite Feld repräsentiert die Bytelänge (len) der Playload (Nutzlast, grau markiert), die in der Nutzlast zu verwendenden Daten im Bereich von 0 bis 255. Das empfangende Ende des MAVlink-Nachrichtenrahmens kann diese Information mit der tatsächlich empfangenen Nutzdaten vergleichen, um deren Integrität zu verifizieren. Das dritte Feld stellt die Sequenznummer (seq) des aktuellen Nachrichtenrahmens dar. Jedes Mal, wenn eine Nachricht gesendet wird, wird der Wert dieses Bytes um 1 erhöht, bis die Zählung nach dem Maximalwert von 255 wieder bei 0 beginnt. Diese Sequenznummer wird vom Empfänger des MAVLink-Nachrichtenrahmens verwendet, um das Nachrichtenverlustverhältnis zu berechnen, das der Signalstärke entspricht. Das vierte Feld stellt die Systemnummer (sys) des Geräts dar, das diesen Nachrichtenrahmen gesendet hat. Die Standardsystemnummer ist 1, wenn PIXHAWK zum Flashen der PX4-Firmware verwendet wird. Die Systemnummer wird verwendet, um zu identifizieren, auf welchem Gerät die Nachricht vom Empfänger des MAVLink-Nachrichtenrahmens gesendet wird. Das fünfte Feld stellt die Einheitennummer (comp) des Geräts dar, das diesen Nachrichtenrahmen gesendet hat. Der Standardwert ist 50, wenn PIXHAWK zum Flashen



Byte Index	Content	Value	Explanation
0	Packet start sign	v1.0: 0xFE (v0.9: 0x55)	Indicates the start of a new packet.
1	Payload length	0 - 255	Indicates length of the following payload.
2	Packet sequence	0 - 255	Each component counts up his send sequence. Allows to detect packet loss
3	System ID	1 - 255	ID of the SENDING system. Allows to differentiate different MAVs on the same network.
4	Component ID	0 - 255	ID of the SENDING component. Allows to differentiate different components of the same system, e.g. the IMU and the autopilot.
5	Message ID	0 - 255	ID of the message - the id defines what the payload "means" and how it should be correctly decoded.
6 to (n+6)	Data	(0 - 255) bytes	Data of the message, depends on the message id.
(n+7) to (n+8)	Checksum (low byte, high byte)	ITU X.25/SAE AS-4 hash, **excluding packet start sign, so bytes 1..(n+6)** Note: The checksum also includes MAVLINK_CRC_EXTRA (Number computed from message fields. Protects the packet from decoding a different version of the same packet but with different variables).	

Abbildung 3.7: MAVLink Nachrichtenformat [3]

der PX4-Firmware verwendet wird. Die Einheitennummer, um die Einheit des Geräts zu identifizieren, das die Nachricht vom Empfänger des MAVLink-Nachrichtenrahmens empfangen hat. Es ist vorerst nutzlos. Das sechste Feld stellt die Nummer (msg) des Nachrichtenpakets in der Nutzlast dar. Diese unterscheidet sich von der Sequenznummer. Der Empfänger des MAVLink-Nachrichtenrahmens muss anhand dieser Nummer bestimmen, welche Nachricht in der Nutzlast platziert wird. Die letzten zwei Bytes sind 16-Bit-Prüfbits, CKB sind die oberen acht Bits und CKA sind die unteren acht Bits. Der Prüfcode wird vom CRC-16-Algorithmus(zyklische Redundanzprüfung) erhalten. Der Algorithmus führt die CRC-16-Berechnungfür die gesamte Nachricht durch (vom Startbit bis zum Ende der Nutzlast plus einem zusätzlichen MAVLINK_CRC_EXTRA-Byte), um einen 16-Bit-Prüfcode zu erhalten. Jede der zuvor erwähnten Nutzdaten in der Nutzlast (angezeigt durch die Nachrichtenpaketnummer) gibt einen MAVLINK_CRC_EXTRA an.

Dieser MAVLINK_CRC_EXTRA wird von der XML-Datei generiert, die den MAVLink-Code generiert. Wenn das Flugzeug und die Bodenkontrollstation unterschiedliche Versionen des MAVLink-Protokolls verwenden, sind die von den beiden Parteien berechneten Prüfcodes unterschiedlich, sodass die MAVLink-Protokolle zwischen verschiedenen Versionen nicht ordnungsgemäß zusammenarbeiten. Durch diese Methode wird das erhebliche Fehlerrisiko einer Kommunikation zwischen verschiedenen Versionen vermieden.

Gemäß dem Format dieser Nachrichten muss der Absender immer die Felder System-ID und Komponenten-ID ausfüllen, damit der Empfänger weiß, woher das Paket stammt. Die System-ID ist eine eindeutige ID für jedes Fahrzeug oder jede Bodenkontrollstation. Bodenkontrollstationen verwenden normalerweise eine hohe System-ID wie 255 und Fahrzeuge verwenden standardmäßig 1 (dies kann durch Setzen des Parameters SYSID_THISMAV geändert werden). Die Komponenten-ID für die Bodenkontrollstation oder den Flugregler lautet normalerweise 1. Andere MAVLink-fähige Geräte im Fahrzeug (etwa Begleitcomputer, Gimbal) sollten dieselbe System-ID wie der Flugcontroller verwenden, jedoch eine andere Komponenten-ID.

3.2.3 Nachrichtenfluss

In diesem Format verpackt wird die Nachricht zum Fahrzeug oder zur Bodenkontrollstation gesendet. Abbildung 3.8 zeigt ein Beispiel für den Nachrichtenfluss zwischen Bodenkontrollstation und Drohne. Sobald eine Verbindung zwischen den beiden hergestellt ist, sendet jedes Gerät (auch bekannt als System) die Heartbeat-Nachricht (orange Pfeile in der Abbildung). Diese wird im Allgemeinen verwendet, um anzuzeigen, dass das Gerät, das die Nachricht sendet, aktiv ist.

Sowohl das Flugzeug als auch die Bodenkontrollstation senden dieses Signal (normalerweise in der 1-Hertz-Frequenz), die Bodenkontrollstation und das Flugzeug bestimmen, ob sie das Flugzeug oder die Bodenkontrollstation verloren haben, je nachdem, ob das Heartbeat-Paket rechtzeitig empfangen wird. Wenn beide Geräte das Heartbeat-Paket des anderen empfangen haben, fordert die Bodenkontrollstation die gewünschten Daten (und die Rate) an, indem sie Nachrichten der folgenden Typen REQUEST_DATA_STREAM und COMMAND_LONG sendet. REQUEST_DATA_STREAM unterstützt das Festlegen der Rate von Nachrichten. COMMAND_LONG mit einem Befehl SET_MESSAGE_INTERVAL bietet eine genaue Kontrolle darüber, welche Nachrichten gesendet werden (und deren Rate), wird jedoch nur von ArduPilot 4.0 und höher unterstützt. Dann bekommt die Drohne die Anfordern-Nachricht und sendet die angeforderte Daten per MAVLink-Nachricht. Falls die Bodenkontrollstation die Nachricht mit Daten empfängt, ist dieser Kommunikationsprozess beendet. Falls nicht, wird die Anfordern-Nachricht erneut gesendet, bis diese Daten

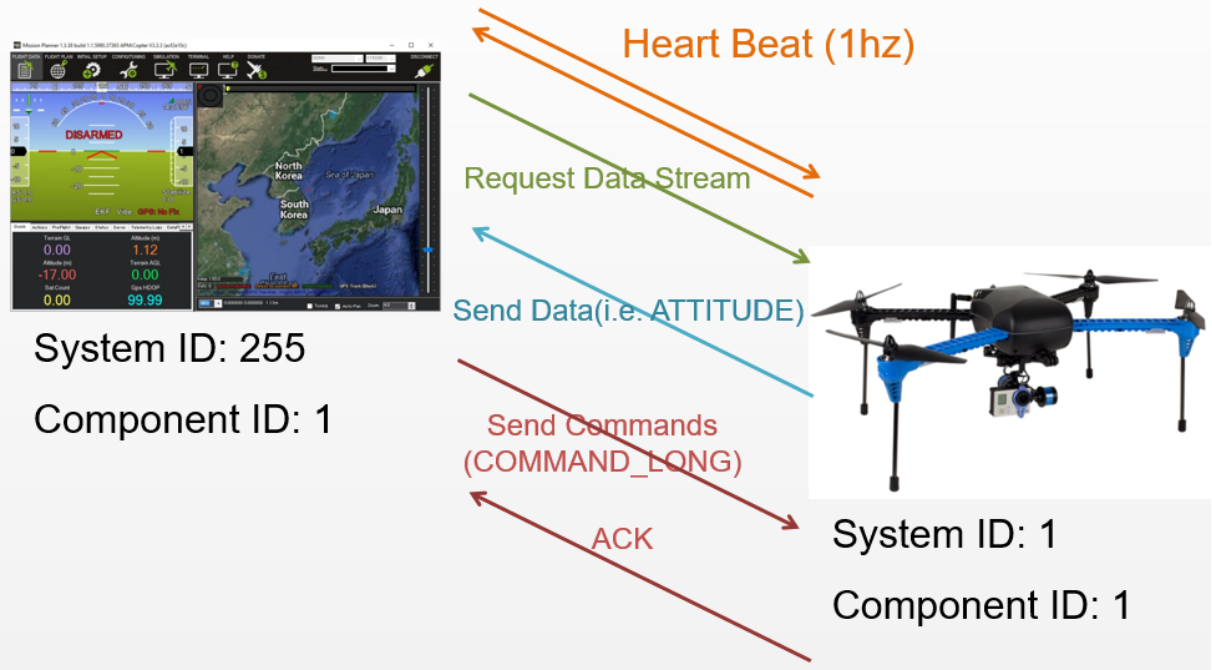


Abbildung 3.8: MAVLink Nachrichtenfluss [3]

empfangen werden. Wenn eine Bodenkontrollstation einen Befehl an das Flugzeug sendet (weinrote Pfeilspitze in der Abbildung) und dieser Befehl empfangen wird, sendet die Drohne eine ACK-Nachricht als Bestätigung zurück an die Bodenkontrollstation. Erhält die Kontrollstation keine ACK-Nachricht von der Drohne, ist das ein Hinweis dafür, dass die Drohne den Befehl möglicherweise nicht bekommen hat.

3.2.4 MAVROS

MAVROS ist ein ROS-Paket, das die erweiterbare MAVLink-Kommunikation zwischen ROS-Framework, MAVLink-fähigen Autopiloten und MAVLink-fähigen Bodenkontrollstationen ermöglicht. Der Hauptkommunikationsknoten des Pakets ist der MAVROS-Knoten, der das Thema mavros_msgs/Mavlink abonniert und das Thema mavros_msgs/Mavlink und diagnostic_msgs/DiagnosticStatus veröffentlicht, um die Daten und Befehle zwischen Drohnen oder Simulator und ROS zu übertragen. Neben den Hauptthemen gibt es viele Unterthemen, um die verschiedenen Kommunikationsfunktionen zu implementieren. So kann zum Beispiel das Thema geometry_msgs/PoseStamped die aktuelle Positionen der Drohne veröffentlichen und die Sollwertpositionen vom ROS an die Drohne weitergeben. Ein weiterer Knoten mavros_extras, ermöglicht die Ergänzung von ROS-Paketen durch zusätzliche Kommunikations-Plugins etwa für Schwingungen und Kameradaten, die in mavros_node nicht enthalten sind.

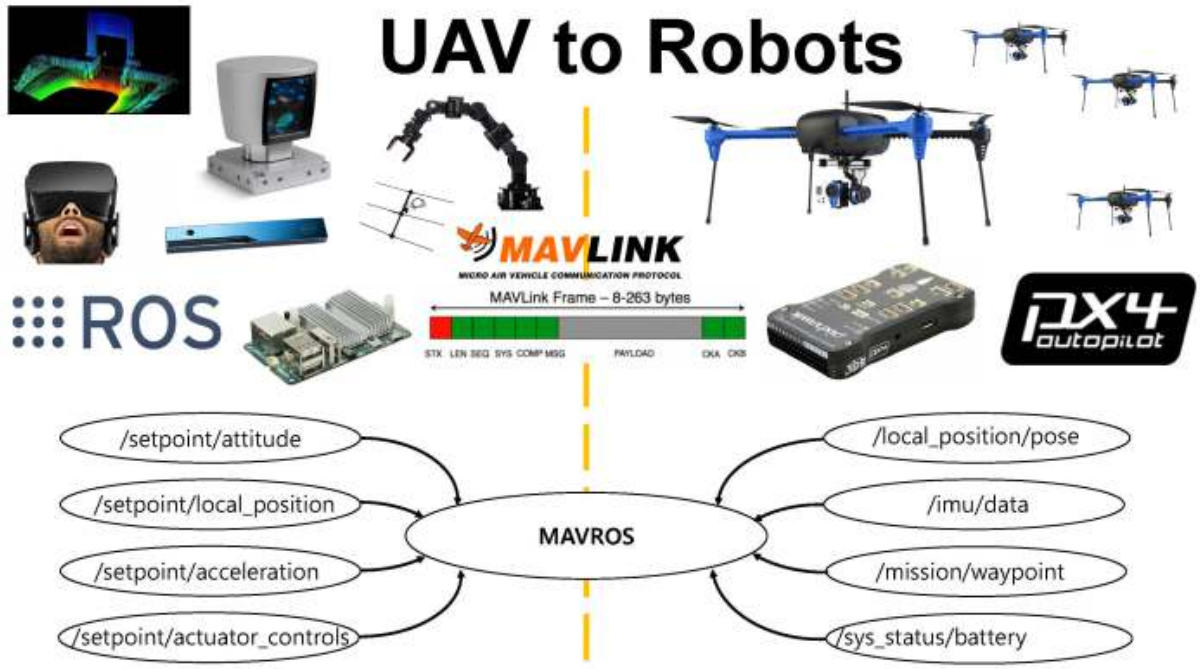


Abbildung 3.9: MAVROS [4]

3.3 Software in the Loop

Um die Implementierung des Algorithmus zu simulieren und die Software zu testen, wird die Methode Software-in-the-Loop (SIL) vorgestellt. SIL bezeichnet das Testen von ausführbarem Code wie Algorithmen (oder sogar eine gesamte Controller-Strategie), der gewöhnlich für ein bestimmtes mechatronisches System geschrieben wurde, in einer Modellierungsumgebung, um die Kosten zu verringern und dem Test zu vereinfachen. Für diese Arbeit werden die Umwelt und die Drohnen in Gazebo modelliert und die Verbindung zwischen dem Gazebo Simulator und ROS-Framework durch PX4 realisiert. Durch diese Vorgehensweise werden Kollisionsrisiken vermieden.

3.3.1 Gazebo

Gazebo Simulator ist ein Open-Source-3D-Robotersimulator, der von 2004 bis 2011 Bestandteil des Player Project war [37]. Im Jahr 2011 wurde Gazebo ein unabhängiges Projekt, das von Willow Garage unterstützt wurde. Im Jahr 2012 wurde die Open Source Robotics Foundation (OSRF) zum Verwalter des Gazebo-Projekts und änderte dessen Namen 2018 zu Open Robotics [38].

Gazebo integrierte die Physik-Engine Open Dynamics Engine (ODE), OpenGL-Rendering und Support-Code für die Sensorsimulation und die Akteursteuerung. Gazebo kann

mehrere Hochleistungs-Physik-Engines wie ODE oder Bullet verwenden (die Standardeinstellung ist ODE) und bietet eine realistische Präsentation von Umgebungen, einschließlich hochwertiger Darstellung von Licht, Schatten und Texturen. Der Simulator kann Sensoren modellieren, die die simulierte Umgebung erfassen, beispielsweise Laser-Entfernungsmesser, Kameras (einschließlich Weitwinkel) und Sensoren im Kinect-Stil.

3.3.2 PX4 Autopilot

Der im Rahmen dieser Arbeit verwendete Autopilot PX4 stellt eine breit einsetzbare Kontrollmöglichkeit für Flugroboter dar und ist mit weiteren Plattformen, wie ROS und den von ROS unterstützten Visualisierungswerkzeugen Gazebo und RViz kompatibel. Der PX4 Autopilot ist Teil eines Open-Source Projektes namens Dronecode Project, das auf kostengünstige autonome Flugzeuge ausgerichtet ist. Das Projekt startete 2009 und wird am Computer Vision and Geometry Lab der ETH Zürich weiterentwickelt und eingesetzt und vom Autonomous Systems Lab und dem Automatic Control Laboratory unterstützt. Es stellt eine umfangreiche Plattform für unbemannte Flugroboter zur Verfügung. Zusätzlich zum genannten Autopiloten und der Bodenkontrollstation bietet das Projekt das bereits eingeführte Kommunikationsprotokoll namens MAVLink für Flugroboter aller Art an. Derzeit wird das Dronecode Project weltweit von namhaften Entwicklungs-, Forschungs- und Industriepartnern unterstützt und getragen.

PX4 unterstützt sowohl die SITL-Simulation, bei der der Flightstack auf einem Computer (entweder auf demselben Computer oder einem anderen Computer im selben Netzwerk) ausgeführt wird, als auch die Hardware in the Loop (HIL) Simulation unter Verwendung einer Simulationsfirmware auf einer realen Flugcontroller-Tafel. In dieser Arbeit wird die SIL-Simulation beschrieben. Die Simulatoren wie Gazebo, jMAVSim und AirSim sind mit PX4 für die SIL-Simulation kompatibel. Von diesen unterstützen Gazebo und jMAVSim die Multifahrzeugsimulation. Wie oben beschrieben, dient für diese Arbeit Gazebo als Simulator.

Der Autopilot PX4 besteht aus zwei Hauptschichten: Der Flightstack ist ein Estimation- und Flight-Control-System, und die Middleware ist eine allgemeine Robotikschicht, die autonome Roboter jede Art unterstützen kann und interne/externe Kommunikation sowie Hardware-Integration bietet. Das Diagramm in Abbildung 3.11 bietet einen detaillierten Überblick über die Bausteine von PX4. Der obere Teil des Diagramms stellt Middleware-Blöcke dar, während der untere Teil die Komponenten des Flightstacks zeigt. Die Pfeile repräsentieren den Informationsfluss zwischen den Modulen. In der Realität gibt es viel mehr Verbindungen als gezeigt, und auf einige Daten (z. B. für Parameter) greifen die meisten Module zu. Module kommunizieren miteinander über einen Publish-Subscribe-

Nachrichtenbus mit dem Namen uORB. Die Verwendung des Publish-Subscribe-Schemas bedeutet erstens, dass das System reaktiv ist (also aktualisiert wird, wenn neue Daten verfügbar sind), zweitens, dass alle Operationen und die Kommunikation vollständig parallel ablaufen, und drittens, dass eine Systemkomponente Daten von überall auf threadsichere Weise verarbeiten kann.

Die Middleware besteht im Wesentlichen aus Gerätetreibern für eingebettete Sensoren, der Kommunikation mit der Außenwelt (Begleitcomputer, GCS usw.) und dem UORB-Publish-Subscribe-Nachrichtenbus. Darüber hinaus enthält die Middleware eine Simulationsschicht, mit der der PX4-Flugcode auf einem Desktop-Betriebssystem ausgeführt und eine computermodellierte Drohne in einer simulierten Umgebung gesteuert werden kann. Da die Module auf Nachrichtenaktualisierungen warten, definieren die Treiber normalerweise, wie schnell ein Modul aktualisiert wird. Die meisten IMU-Treiber testen die Daten mit einer Frequenz von 1 kHz, integrieren sie und veröffentlichen sie mit 250 Hz. Andere Teile des Systems, wie der Navigator, benötigen keine so hohe Aktualisierungsrate und laufen daher erheblich langsamer.

Der Flightstack ist eine Sammlung von Leit-, Navigations- und Steuerungsalgorithmen für autonome Drohnen. Er enthält Steuerungen für Starrflügel-, Multirotor- und VTOL-Flugzeugzellen sowie Schätzer für Fluglage und Position. Abbildung 3.10 zeigt eine Übersicht über die Bausteine des Flightstacks. Das Diagramm zeigt die gesamte Pipeline von Sensoren, RC-Eingang und autonomer Flugsteuerung (Navigator) bis hin zur Motor- oder Servosteuerung (Aktuatoren).

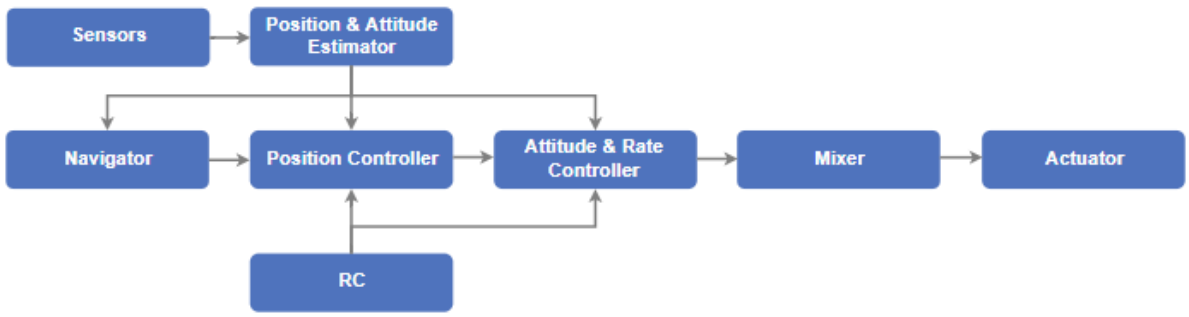


Abbildung 3.10: Flightstack

Ein Estimator nimmt eine oder mehrere Sensoreingaben und berechnet daraus einen Fahrzeugzustand (zum Beispiel anhand der IMU-Sensordaten). Eine Steuerung (Position Controller) ist eine Komponente, die einen Sollwert und eine Messung oder einen geschätzten Zustand (Prozessvariable) als Eingabe verwendet. Ziel ist es, den Wert der Prozessvariablen so anzupassen, dass er dem Sollwert entspricht. Der Output ist eine Korrektur, um diesen Sollwert schließlich zu erreichen. Beispielsweise nimmt der Positionsregler Positions-

sollwerte als Eingaben an, die Prozessvariable ist die geschätzte aktuelle Position, und die Ausgabe ist ein Lager- und Schubsollwert, der das Fahrzeug in Richtung der gewünschten Position bewegt. Ein Mixer nimmt Kraftbefehle entgegen (z. B. rechts abbiegen) und übersetzt sie in einzelne Motorbefehle, wobei sichergestellt wird, dass bestimmte Grenzwerte nicht überschritten werden. Diese Übersetzung ist spezifisch für jeden Fahrzeugtyp und hängt von verschiedenen Faktoren ab, wie z. B. den Motoranordnungen in Bezug auf den Schwerpunkt oder der Rotationsträgheit der Drohne.

3.3.3 Simulation

Alle Simulatoren kommunizieren mit PX4 über die Simulator-MAVLink-API. Diese API definiert eine Reihe von MAVLink-Nachrichten, die Sensordaten aus der simulierten Welt an PX4 liefern und Motor- und Aktorwerte aus dem Flugcode zurückgeben, der auf das simulierte Flugzeug angewendet wird. Abbildung 3.12 zeigt den Nachrichtenfluss in einer typischen SIL-Simulationsumgebung für einen der unterstützten Simulatoren.

Die verschiedenen Teile des Systems stellen UDP-Verbindungen her und können entweder auf demselben Computer oder auf einem anderen Computer im selben Netzwerk ausgeführt werden. Standardmäßig verwendet PX4 UDP-Ports für die MAVLink-Kommunikation mit Bodenkontrollstationen (z. B. QGroundControl), Offboard-APIs (z. B. MAVSDK, MAVROS) und Simulator-APIs (z. B. Gazebo). PX4 verwendet das normale MAVLink-Modul, um eine Verbindung zu Bodenstationen (die Port 14550 überwachen) und externen Entwickler-APIs wie MAVSDK oder ROS (die Port 14540 überwachen) herzustellen. Der lokale TCP-Port 4560 des Simulators wird für die Kommunikation mit PX4 verwendet. Die Simulatoren tauschen dann mithilfe der oben beschriebenen Simulator-MAVLink-API Informationen mit PX4 aus.

3.4 OctoMap

Die im Rahmen dieser Arbeit verwendete OctoMap-Bibliothek wurde von Kai M. Wurm und Armin Hornung an der Universität Freiburg im Jahr 2013 entwickelt und wird derzeit von Armin Hornung gepflegt. Die OctoMap-Bibliothek implementiert einen 3D-Belegungsgitter-Mapping-Ansatz, der Datenstrukturen und Mapping-Algorithmen in C++ bereitstellt, die besonders für die Robotik geeignet sind. Die Kartenimplementierung basiert auf einem Octree. Ein Octree ist eine hierarchische Datenstruktur für dreidimensionale räumliche Unterteilung. Jeder Knoten in einem Octree repräsentiert den enthaltenen Raum in einem kubischen Volumen, normalerweise als Voxel bezeichnet. Dieser Band wird

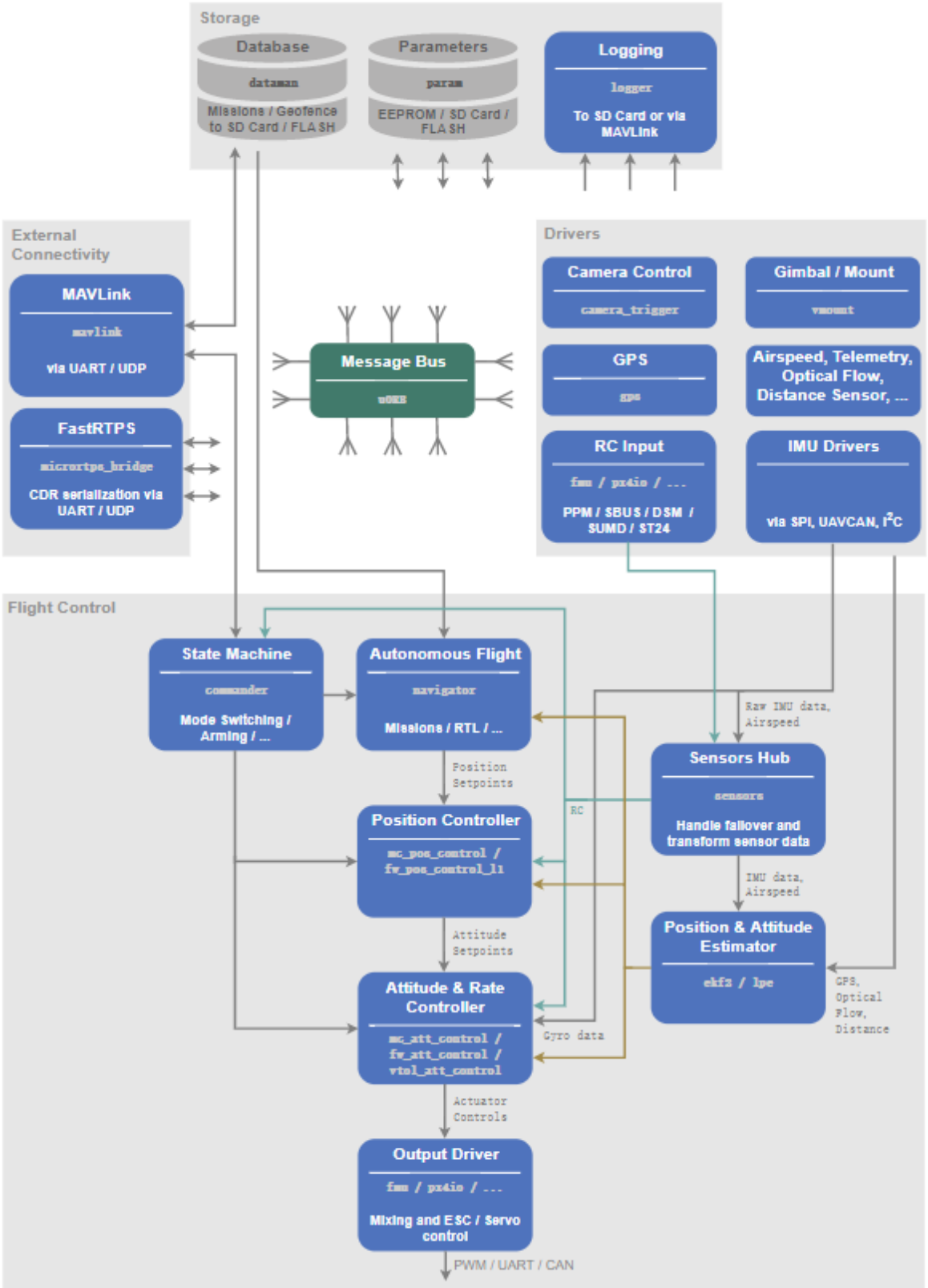


Abbildung 3.11: PX4 Architektur [39]

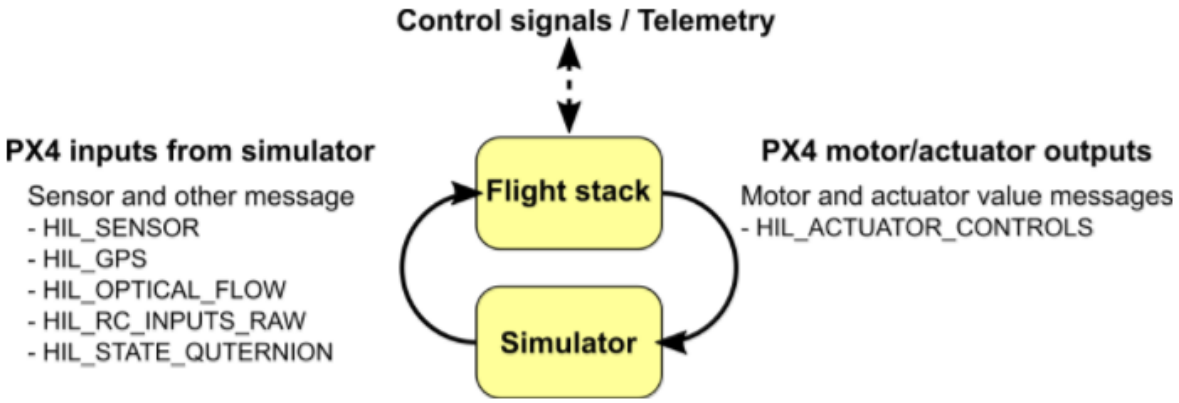


Abbildung 3.12: Simulator Nachrichtenfluss [39]

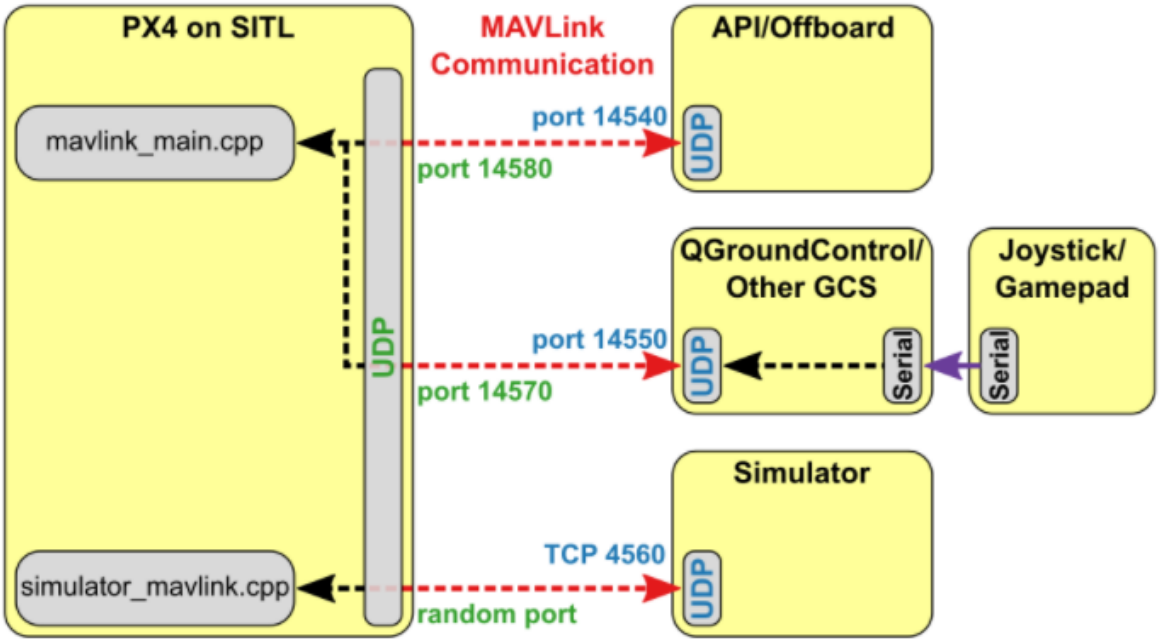


Abbildung 3.13: Software in the Loop Architektur [40]

rekursiv in acht Teilbände unterteilt, bis eine gegebene minimale Voxelgröße erreicht wird. Abbildung 3.14 stellt das Volumenmodell links und den entsprechenden Baum rechts dargestellt. Die minimale Voxelgröße bestimmt die Auflösung des Octree. Da ein Octree eine hierarchische Datenstruktur ist, kann der Baum auf jeder Ebene beschnitten werden, um eine gröbere Unterteilung zu erhalten, wenn die inneren Knoten entsprechend gepflegt werden. In Abbildung 3.15 werden besetzte Voxel in Auflösungen von 0,08 Metern, 0,64 Metern und 1,28 Metern angezeigt [5].

In ihrer einfachsten Form können Octrees die boolesche Eigenschaft modellieren, um die Belegung eines Volumens zu kartieren. Wenn ein bestimmtes Volumen als besetzt gemes-

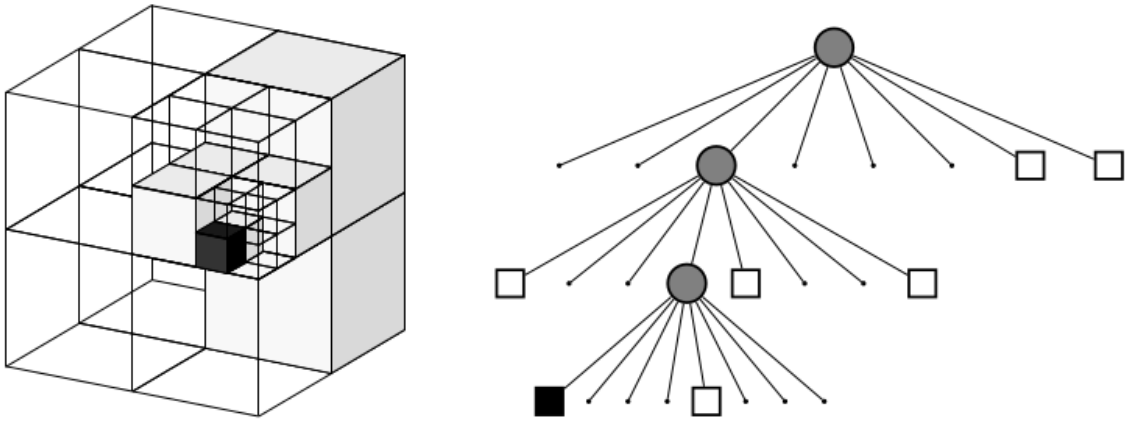


Abbildung 3.14: Octree

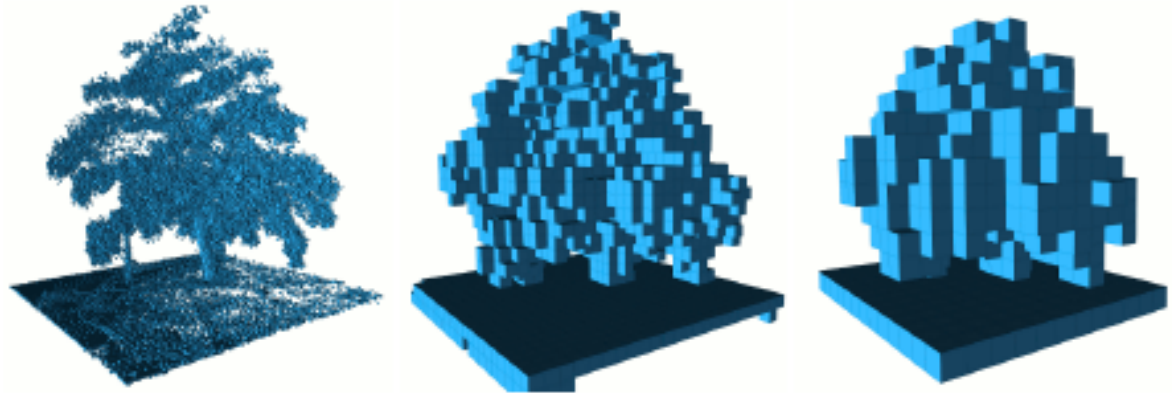


Abbildung 3.15: Octree-Auflösung [5]

sen wird, wird der entsprechende Knoten im Octree initialisiert. In der OctoMap werden Sensorablesungen mithilfe der von Moravec und Elfes eingeführten Belegungsgitterkartierung integriert [41]. Die Wahrscheinlichkeit $P(n \mid z_{1:t})$ eines Knotens n, der bei den Sensormessungen $z_{1:t}$ besetzt werden soll, ist wie folgt definiert:

$$P(n \mid z_{1:t}) = \left[1 + \frac{1 - P(n \mid z_t)}{P(n \mid z_t)} \frac{1 - P(n \mid z_{1:t-1})}{P(n \mid z_{1:t-1})} \frac{P(n)}{1 - P(n)}\right]^{-1} \tag{3.1}$$

Diese Aktualisierungsformel hängt von der aktuellen Messung z_t, einer vorherigen Wahrscheinlichkeit $P(n)$ und der vorherigen Schätzung $P(n \mid z_{1:t-1})$ ab. Der Term $P(n \mid z_t)$ bezeichnet die Wahrscheinlichkeit, dass Voxel n bei der Messung z_t besetzt wird. Dieser Wert ist spezifisch für den Sensor, der z_t erzeugt hat. Die übliche Annahme einer einheitlichen vorherigen Wahrscheinlichkeit führt zu $P(n) = 0,5$ und unter Verwendung der Log-Odds-Notation kann die Gleichung 6.1 umgeschrieben werden:

$$L(n \mid z_{1:t}) = L(n \mid z_{1:t-1}) + L(n \mid z_t), \quad L(n) = log\left[\frac{P(n)}{1 - P(n)}\right] \tag{3.2}$$

Diese Formulierung der Aktualisierungsregel ermöglicht schnellere Aktualisierungen, da die Multiplikation durch eine Addition ersetzt wird. Bei vorberechneten Sensormodellen müssen die Logarithmen während des Aktualisierungsschritts nicht berechnet werden. Jeder nicht initialisierte Knoten kann frei sein oder unbekannt in dieser booleschen Einstellung. Um diese Mehrdeutigkeit zu beheben, werden die freie Volumen im Baum dargestellt. Diese werden im Bereich zwischen dem Sensor und dem gemessenen erstellt Endpunkt, z. B. entlang eines mit Raycasting bestimmten Strahls. Bereiche die nicht initialisiert sind, werden als implizit unbekannter Raum modelliert. Abbildung 3.16 zeigt die Illustration eines Octrees, der freie und besetzte Knoten aus realen Lasersensordaten enthält (Links: Pointclouds, aufgenommen in einem Korridor mit einem kippbaren Laser-Entfernungsmesser. Mitte: Aus den Daten generierter Octree, der nur belegte Voxel anzeigt. Rechts: Visualisierung des Octree mit besetzten Voxeln (dunkel) und freien Voxeln (weiß). Die freien Bereiche werden erhalten, indem der Raum auf einem Strahl vom Sensorursprung zu jedem Endpunkt freigegeben wird. Verlustfreies Beschneiden führt zu Blattknoten unterschiedlicher Größe, die vor allem in den freien Bereichen rechts sichtbar sind). Die boolesche Belegungszustände oder diskrete Beschriftungen ermöglichen eine kompakte Darstellung des Octree. Wenn alle Kinder (Unterknoten) eines Knotens den gleichen Zustand haben (besetzt oder frei), können sie weggeschnitten werden. Dies führt zu einer wesentlichen Verringerung der Anzahl der Knoten, die im Baum gepflegt werden müssen.

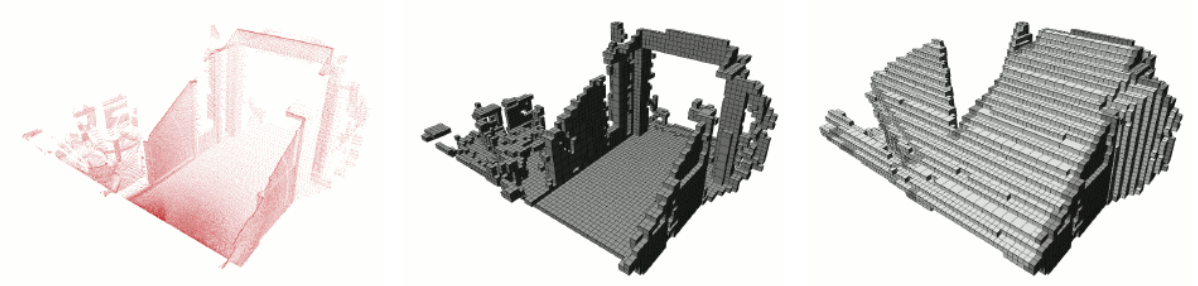


Abbildung 3.16: OctoMap-Auflösung[5]

Zusammenfassend lassen sich für die OctoMap-Bibliothek im Wesentlichen folgende Vorteile gegenüber anderen Map-Bibliotheken wie Pointclouds und Elevationmap festhalten:

- Vollständiges 3D-Modell: Die Karte kann beliebige Umgebungen ohne vorherige Annahmen modellieren. Die Darstellung modelliert belegte Raum sowie freien Raum.

Unbekannte Bereiche der Umgebung werden implizit in der Karte codiert. Da die Unterscheidung zwischen freiem und besetztem Raum für eine sichere Roboternavigation wesentlich ist, sind Informationen über unbekannte Bereiche von entscheidender Bedeutung z. B. für die autonome Erkundung einer Umgebung.

- Aktualisierbar: Es ist jederzeit möglich, neue Informationen oder Sensorwerte hinzuzufügen. Das Modellieren und Aktualisieren erfolgt auf probabilistische Weise. Dies erklärt Sensorrauschen oder Messungen, die aus dynamischen Veränderungen der Umgebung resultieren, z. B. aufgrund dynamischer Objekte. Darüber hinaus können mehrere Roboter zu einer Karte beitragen, und eine zuvor aufgezeichnete Karte kann um neu erkundete Gebiete erweitert werden.
- Flexibilität: Der Umfang der Karte muss nicht im Voraus bekannt sein. Stattdessen wird die Karte nach Bedarf dynamisch erweitert. Die Karte hat mehrere Auflösungen, so dass beispielsweise ein Planer auf hoher Ebene eine grobe Karte verwenden kann, während ein lokaler Planer möglicherweise mit einer feinen Auflösung arbeitet. Dies ermöglicht auch effiziente Visualisierungen, die von groben Übersichten bis hin zu detaillierten Nahansichten skalieren.
- Kompaktheit: Die Karte wird sowohl im Speicher als auch auf der Festplatte effizient gespeichert. Es ist möglich, komprimierte Dateien für die spätere Verwendung oder den bequemen Austausch zwischen Robotern bei beschränkter Bandbreite zu generieren [5].

3.5 FCL

Eine der Hauptaufgaben bei der Pfadplanung für mehrere Drohnen besteht in der Erkennung drohender Kollisionen. Für diese Aufgabe wurde die Flexible Collision Library entwickelt und im Jahr 2012 bei Jia Pan vorgestellt. Aus Anwendungssicht ist die FCL darauf ausgelegt, einheitliche und erweiterbare Schnittstellen für Kollisionen und Näherungsberechnungsalgorithmen bereitzustellen. Konzipiert wurde sie darüber hinaus zur Unterstützung verschiedener Datendarstellungen, einschließlich Dreiecksgitter und bekannter Formprimitive (z. B. Kugel, Zylinder). Um diese Ziele zu erreichen, modelliert die FCL alle Kollisionen und Annäherungsabfragen zwischen zwei Objekten als Abfrageprozess entlang einer hierarchischen Struktur. Wie Abbildung 3.17 zeigt, wird der FCl-Abfrageprozess in drei Schritten ausgeführt [6].

- Objektdarstellung: Objekte werden durch eine hierarchische, für spezifische Abfragen geeignete Struktur dargestellt. So werden etwa geometrische Grundformen (z. B. Kegel, Zylinder, Kugeln) von Knoten ersetzt, die einstufige Hierarchie mit

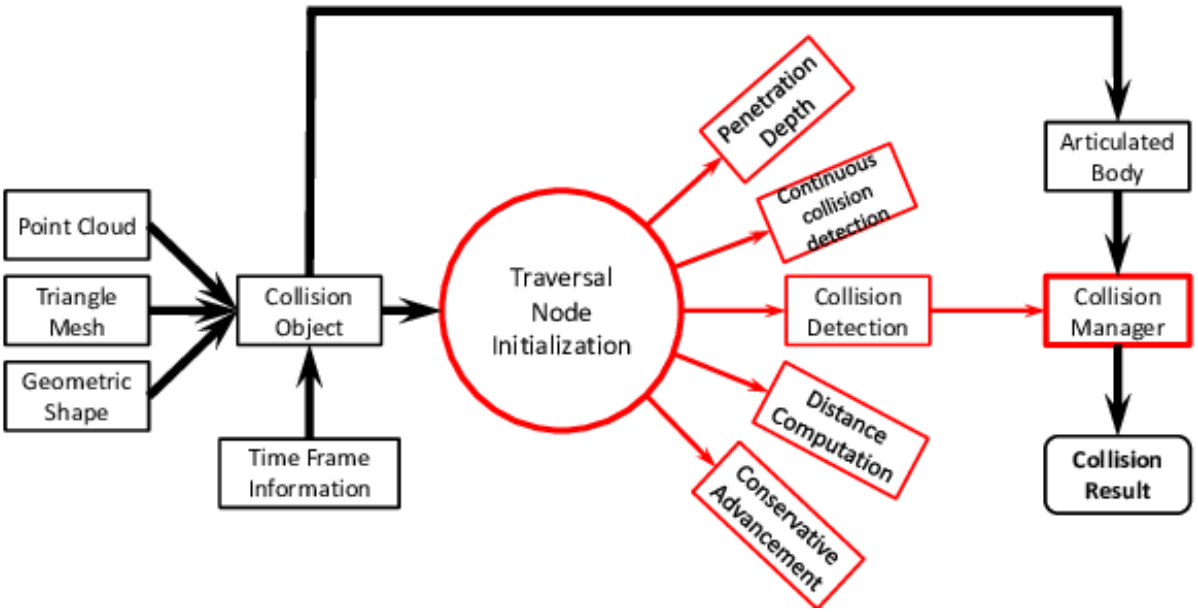


Abbildung 3.17: FCL-Architektur [6]

der entsprechenden Begrenzung sind. Beliebige geometrische Objekte werden unter Verwendung einer Begrenzungsvolumenhierarchie dargestellt. Die hierarchische Struktur und die Konfiguration des Objekts werden in einer Struktur namens CollisionObject erstellt, das auch die Formdarstellung verformbarer Modelle aus dem vorherigen Zeitschritt enthält.

- Die Initialisierung des Abfrageknotens: Im Abfrageknoten werden die vollständigen Informationen, die für eine bestimmte Abfrage erforderlich sind, gespeichert. Die Informationen unterscheiden sich von verschiedenen Objektdarstellungen. Für eine kontinuierliche Kollisionsabfrage müssen z. B. die Objektkonfiguration und Formdarstellung für den vorherigen Zeitrahmen gespeichert werden. Außerdem kann der Abfrageknoten die Strategie der Abfrage entscheiden. Erfordert beispielsweise der Knoten nur eine Ja-Nein-Antwort erfordert, stoppt die Kollisionsabfrage, sobald eine Kollision gefunden wurde. Eine solche vorzeitig beendete Strategie ist auf Trennungsabstandsabfragen nicht anwendbar.
- Hierarchieabfragen: Nach der Initialisierung des Abfrageknotens wird die Abfrage die hierarchische Struktur durchlaufen, um eine bestimmte Kollisions- oder Näherungsobjekte zu finden.

In Abbildung 3.17 zur FCL-Architektur repräsentieren die schwarzen Pfeile die Datenflüsse, einschließlich Aufbau der hierarchischen Struktur für jedes Objekt und Kollisionsmanager für mehrere Objekte. Die roten Pfeile stehen für Algorithmusfluss, einschließlich Vorbereitung des Abfrageknotens und hierarchischer Abfrage. Die Kollisionsabfragen für artikulierte Körper oder Umgebungen mit mehreren beweglichen/verformbaren Objekten

müssen effizient ausgeführt werden. In der FCL geschieht dies durch den Kollisionsmanager, der den SaP für die Kollisionserkennung mit N-Körpern Algorithmus verwendet zur Behandlung solcher Szenarien für verschiedene Abfragen.

3.6 Cplex

Da diese Arbeit sich mit dem MAPF-Problem befasst, geht es um die Pfadplanung für jede Drohne zur Vermeidung von Kollisionen. In die Planung müssen außerdem zusätzliche Einschränkungen einbezogen werden, z. B. Hindernisse, die Raumbegrenzung, die Geschwindigkeit, die Kapazität der Batterie, Ausgangspunkt und Endpunkt usw. Das MAPF-Problem kann in eine quadratische Programmierung (QP) umgewandelt werden. Zur Verbesserung der Recheneffizienz der QP wird für diese Arbeit das IBM ILOG CPLEX Optimization Studio verwendet (meist schlicht bezeichnet als CPLEX). Es handelt sich dabei um ein Programmsystem zur Modellierung und Lösung von Optimierungsproblemen mithilfe mathematischer Optimierung sowie der Constraint-Programmierung. CPLEX wurde 1987 von Robert Bixby als Implementierung des Simplex-Verfahrens zur Lösung linearer Optimierungsprobleme in der Programmiersprache C entwickelt. Es wurde erstmals im Jahr 1988 durch die von Bixby und Janet Lowe gegründete CPLEX Optimization Inc. kommerziell vertrieben und weiterentwickelt. Als einer der Marktführer im Bereich Operations Research wurde die CPLEX Optimization Inc. im Jahr 1997 durch ILOG aufgekauft. Neben einem kommandozeilen-basierten Solver stellt CPLEX auch die Modellierungssprache OPL und umfangreiche Bibliotheken mit Anbindung an die Programmiersprachen C, C++, Java und Python bereit. CPLEX Optimizer bietet flexible, leistungsfähige Solver für die mathematische Programmierung, die lineare Programmierung, die gemischt ganzzahlige Programmierung, die quadratische Programmierung und quadratisch beschränkte Programmierungsprobleme. Die Solver zeichnen sich durch einen verteilten parallelen Algorithmus für gemischt ganzzahlige Programmierung aus, um für die Lösung schwieriger Probleme mehrere Computer nutzen zu können.

Wie bereits erwähnt, ist die quadratische Programmierung ein Modell, in dem die Einschränkungen linearer Art sind, die Zielfunktion aber einen oder mehrere quadratische Terme enthalten kann. Wenn solche Probleme konvex sind, löst CPLEX sie normalerweise effizient in Polynomzeit. Nicht-konvexe QP sind jedoch ziemlich hart. Theoretisch werden sie als NP-hart charakterisiert. CPLEX wendet verschiedene Ansätze auf diese Probleme an, beispielsweise Ansätze Barriere-Algorithmen oder Verzweigungs- und gebundene Algorithmen. Insbesondere in der Branche und gebunden Ansatz gibt es keine theoretische Garantie für die Komplexität eines solchen Problems. Folglich kann die Lösung dieses

Problems nichtkonvexer QP mehr Rechnenzeit erfordern als die Lösung einer konvexen QP von vergleichbarem Wert.

4 Algorithmen

In diesem Kapitel werden verschiedene Algorithmen für die Pfadplanung und Trajektorienverarbeitung im Rahmen der Arbeit erläutert. Zuerst werden in einer Übersicht verschiedene Algorithmen für dreidimensionale Routenplanung zusammengefasst. Dann werden die in dieser Arbeit verwendeten Algorithmen zur Routenplanung vorgestellt, nämlich RRT, RRT*, CBS und A*. Am Ende des Kapitels wird ein Algorithmus namens Bernsteinpolynom zur Trajektorienverarbeitung eingeführt.

4.1 Übersicht der Algorithmen von Routen plannung

Algorithmen für die 3D-Pfadplanung sind seit dem letzten Jahrhundert entstanden. Inzwischen gibt es viele Methoden dreidimensionaler Routenplanung, angewendet für verschiedene Roboter und Umgebungen. z. B. Rapidly-exploring random tree (RRT), Probabilistic Road Maps (PRM), künstliches Potenzialfeld und Mixed-Integer-Programmierung. Entsprechend ihrer Eigenschaften lassen sich die Algorithmen in fünf Kategorien einordnen, wie in Abbildung 4.1 dargestellt [42].

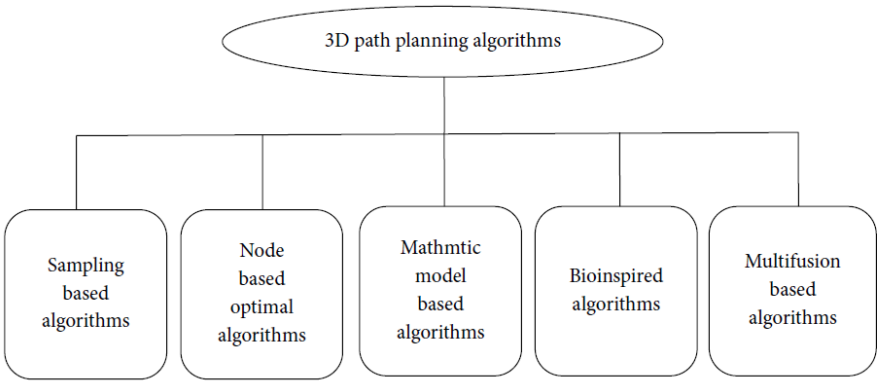


Abbildung 4.1: Klassifizierung von Pfadplanung

Stichprobenbasierte Algorithmen (Englisch: Sampling-based algorithms) benötigen einige bekannte Umweltinformationen in der mathematische Darstellung, um den Arbeitsbereich zu beschreiben. Diese Methode tastet die Umgebung normalerweise als Reihe von Knoten oder Zellen oder in anderen Formen ab. Dann werden die Stichprobeknoten mit den einschränkenden Faktoren kombiniert oder einfach nach dem Zufallsprinzip gesucht, um

einen realisierbaren Pfad zu erhalten. Die Elemente stichprobenbasierter Algorithmen sind in Abbildung 4.2 dargestellt.

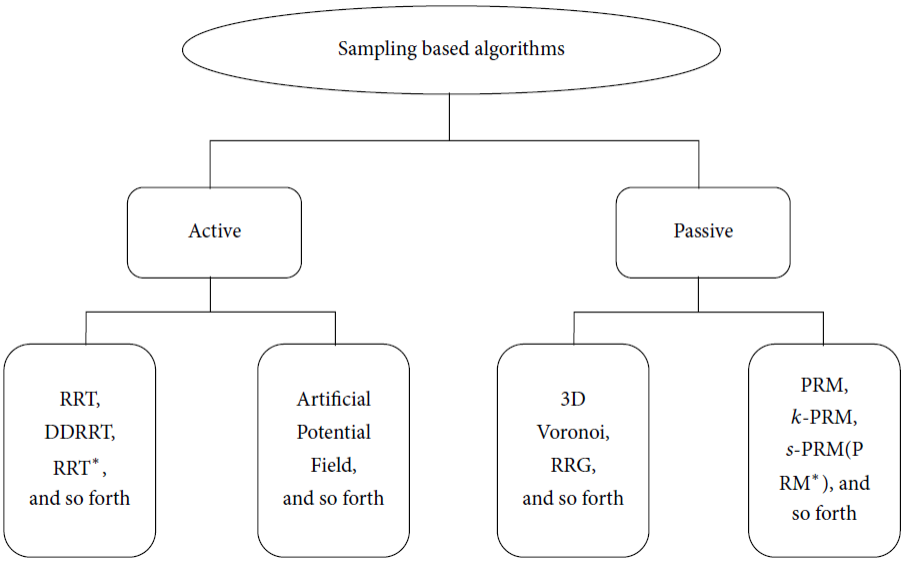


Abbildung 4.2: Stichprobenbasierte Algorithmen

Das Diagramm unterteilt die stichprobenbasierten Algorithmen in zwei Gruppen: aktiv und passiv. Aktiv umfasst Algorithmus wie RRT, die den machbaren Pfad zum Ziel durch ihr eigenes Verarbeitungsverfahren erreichen können. Passiv bedeutet Algorithmen wie PRM, die nur ein Straßennetz vom Start bis zum Ziel erzeugen, das viele mögliche Pfade enthält. Nach Kombination von Suchalgorithmen wird der bestmögliche Pfad gefunden. Algorithmen, die nicht selbständig den besten Pfad finden können (d. h. abhängig von anderen Algorithmen sind), werden als passiv klassifiziert.

Knotenbasierte optimale Algorithmen(Englisch: node-based optimal algorithms) teilen die Eigenschaft Stichprobenbasierter Algorithmen, die unter einer Reihe von Knoten (Zelle) in der Karte untersuchen. Allerdings wird hier die Karte mit einer zerlegten Grafik ersetzt. Deshalb kann diese Methode immer einen optimalen Weg finden. Abbildung 4.3 zeigt die typischen Elemente knotenbasierter optimaler Algorithmen. LPA* ist dabei der Algorithmus für lebenslange Planung A*, eine sich wiederholende Version von A* (d. h. mit der Fähigkeit, mit dynamischen Bedrohungen umzugehen). Basierend auf LPA* wird D* vorgeschlagen, das sich mit dynamischen Bedrohungssituationen befasst. Die Dijkstra-Algorithmen A* und D* werden traditionell als diskrete optimale Planung oder Straßenkartenalgorithmen klassifiziert.

Auf mathematischen Modellen basierende Algorithmen umfassen lineare Algorithmen und optimale Steuerung. Diese Methoden modellieren die Umgebung (kinematische Einschränkungen) sowie das System (dynamisch) und die Kostenfunktion, die mit den ki-

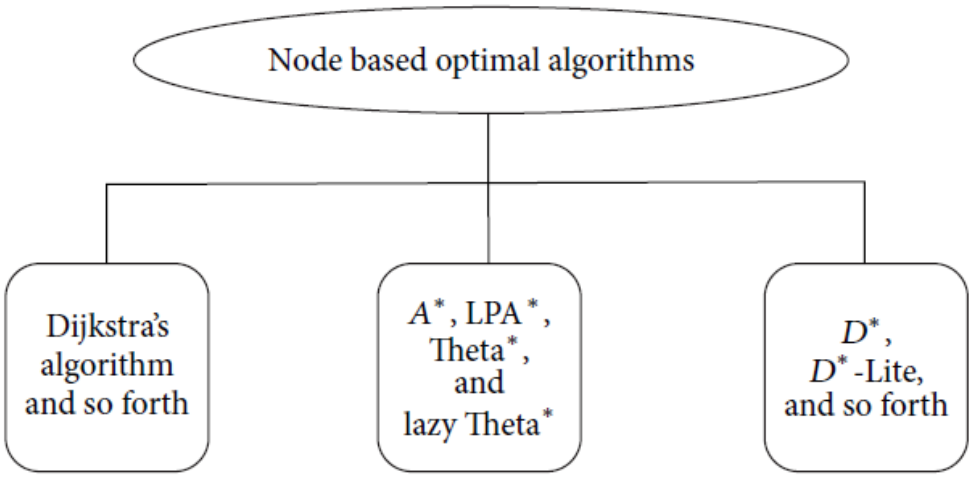


Abbildung 4.3: Knotenbasierte Algorithmen

nematischen und dynamischen Beschränkungsgrenzen verbunden ist, um eine optimale Lösung zu erreichen. Die Elemente auf mathematischen Modellen basierender Algorithmen werden in Abbildung 4.4 vorgestellt.

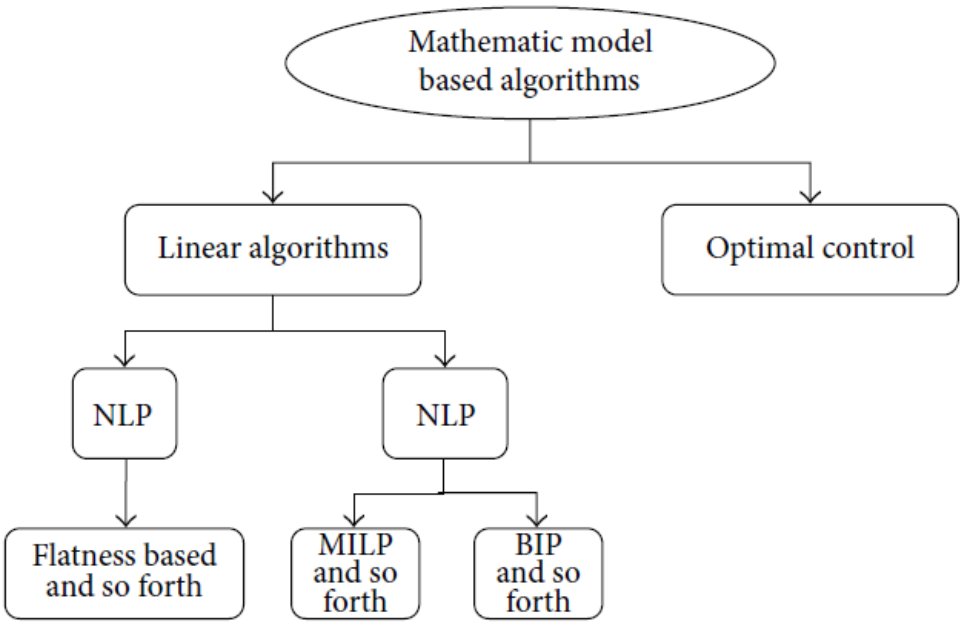


Abbildung 4.4: Auf mathematischen Modellen basierende Algorithmen

Dabei verwendet die auf Ebenheit basierende Methode, die zuerst von Chamseddine vorgeschlagen wurde, verschiedene Ebenheit und die Linerarisierung der nichtlinearen kinodynamischen Zwänge, um die Kontrollebenheit entlang des Referenzpfades sicherzustellen. Die Mixed-Integer-lineare-Programmierung (MILP) zeichnet sich durch eine starke Mo-

dellierungsfähigkeit aus. Die binär-lineare Programmierung (BIP) ist ein Sonderfall der linearen Programmierung, wobei die Variablen nur einen ganzzahligen Wert Null oder Eins haben.

Bioinspirierte Algorithmen stammen aus der Nachahmung biologischer Verhaltensweisen zur Lösung von Problemen. Diese Planungsmethoden lassen den Prozess des Aufbaus einer komplexen Umgebung aus und überwinden die Schwäche der auf mathematischen Modellen basierenden Algorithmen, die bei der Lösung von NP-harten Problemen mit großer Anzahl von Variablen und nichtlinearen Zielfunktionen häufig fehlschlagen. Abbildung 4.5 präsentiert eine Reihe typischer aktueller Methoden.

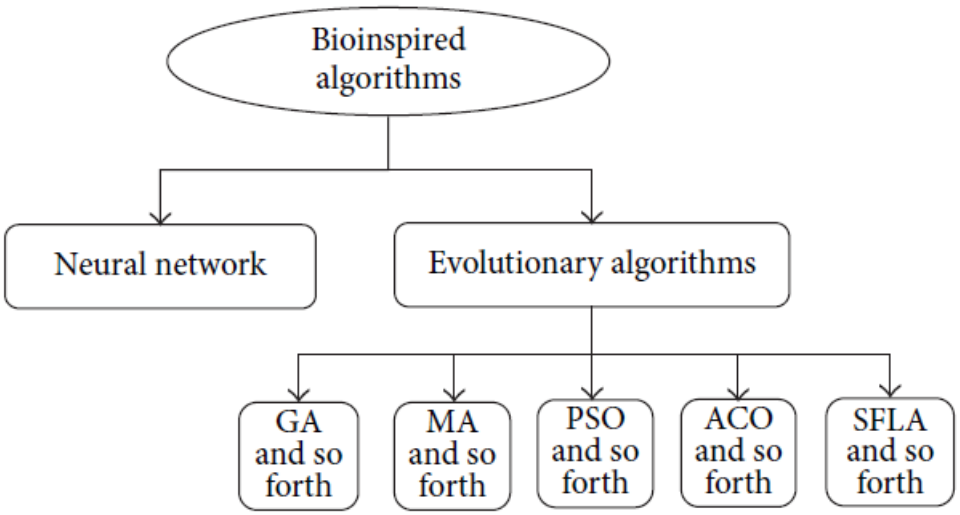


Abbildung 4.5: Bioinspiriert Algorithmen

Dabei ist der genetische Algorithmus (GA) auch die bekannteste populationsbasierte numerische Optimierungsmethode. Der memetische Algorithmus (MA) ist ein populationsbasierter heuristischer Explorationsansatz für kombinatorische Optimierungsprobleme. Die Partikelschwarmoptimierung (PSO) ist ein populationsbasierter stochastischer Optimierungsalgorithmus, und die Ameisenkolonie-Optimierung (ACO) ahmt das Verhalten der Ameise beim Finden des kürzesten Weges mit Pheromoninformation nach. Der Leapfrog-Algorithmus (Englisch: shuffled frog-leaping algorithm (SFLA)) ist eine Kombination von MA und PSO ist. Bei bioinspirierten Algorithmen wird zwischen dem Evolutionary Algorithm (EA) und dem neuronalen Netz (NN) unterschieden, die auf verschiedenen Ebenen analysiert werden.

Multifusion ist ein in jüngster Zeit entstehender Ansatz zur Verbesserung der Leistung von 3D-Pfad-Planungsalgorithmen. Die Algorithmen lassen sich miteinander kombinieren, um einen optimalen Weg zu planen (bessere Echtzeit oder optimale Leistung). Künstliche Potenzialfeldalgorithmen fallen z. B. normalerweise hinein lokale Minima ohne Naviga-

tionsfunktion oder andere Tricks. Probabilistische Straßenkarten können keine optimale Lösung erzeugen. Eine Kombination beider Algorithmen könnte die Nachteile beseitigen.

4.2 Routenplanungsalgorithmen in dieser Arbeit

Wie oben beschrieben, gibt es viele Algorithmen, die auf verschiedenen Grundprinzipien beruhen und je eigene Vor- und Nachteile aufweisen. In dieser Arbeit werden die Algorithmen RRT, CBS (conflict-based search) und A* zur Pfadplanung in drei Dimensionen genutzt. Im Folgenden werden deren Grundlagen und Eigenschaften erklärt.

4.2.1 RRT, RRT* und informierter RRT*

Der RRT Algorothmus wurde von LaValle vorgeschlagen [43]. Er versucht, die Probleme der Pfadplanung unter holonomen, nichtholonomen und kinodynamischen Bedingungen zu lösen. Der RRT durchsucht den Konfigurationsraum schnell, um einen Baum zu generieren, der Start- und Zielknoten verbindet. In jedem Schritt wird ein neuer Knoten abgetastet. Wenn die Erweiterung vom abgetasteten zum nächsten Knoten erfolgreich ist, wird dem Baum ein neuer Knoten hinzugefügt. Bei Anwendung dieser Methode in einer 3D-Umgebung, wird normalerweise davon ausgegangen, dass ein 3D-Konfigurationsraum $\chi = C$ vorhanden ist. Der Konfigurationsraum besteht aus zwei Teilen: einem festen Hindernisbereich $\chi_{obs} \subset \chi$, der vermieden werden muss, und einem hindernisfreien Bereich $\chi_{free} \subset \chi$, in dem sich die Roboter aufhalten müssen. Entsprechend des Konfigurationsraums enthält ein Pfadzustands-(oder Scheitelpunkt-) Satz P alle Abtastscheitelpunkte, die durch den RRT-Explorationsprozess erzeugt werden. Um den Algorithmus zu implementieren, müssen die folgenden Schritte befolgt werden (siehe in Abbildung 4.6). Die Cyan-Kreise stellen Hindernisregionen dar, die nicht passiert werden können. γ ist die maximale Schrittverlängerungslänge gemäß den angegebenen Einschränkungen und Kostenfunktionen.

- Schritt 1: Zuerst gibt es den Anfangszustand $\chi_{init} \in \chi_{free}$ in P als ersten Punkt. Dann wird ein Punkt zufällig in χ_{free} frei gewählt. Abbildung 4.6 zeigt zwei Zustände, die $x_{random1}$ und $x_{random2}$.
- Schritt 2: Dann wird ein Folgezustand, der auf dem neu erzeugten Zustand x_{random} in P basiert und einen festen Abstand zu diesem hat, ausgewählt. Der Punkt x_{near} wird als Elternzustand von x_{random} betrachtet.
- Schritt 3: x_{random} ist der Zustand, der die Richtung angibt, in die der nächste Punkt gehen soll. Unter der Berücksichtigung der kinodynamischen Einschränkungen wird

ein Regelungsfaktor in einer Kostenfunktion $\phi = f(x, y, z)$ hinzugefügt. Gemäß den Einschränkungen r und der Kostenfunktion ϕ wird der erreichbare Zustand x_{new} erstellt. Eine Prüfung ermittelt, ob x_{new} frei ist. Wenn es sich in χ_{free} befindet, wird es dem Pfad P hinzufügt. Andernfalls wird es weggelassen.

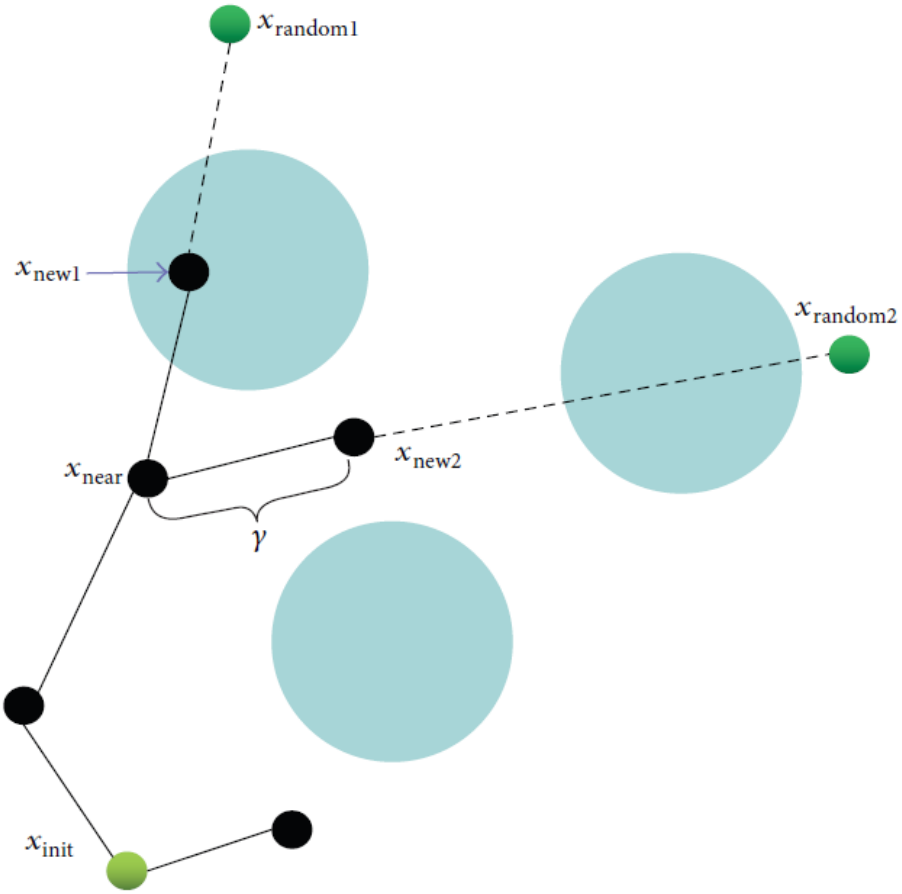


Abbildung 4.6: RRT

In Abbildung 4.6 liegt, der Punkt x_{new1} im Hindernis. Deshalb ist die Richtung nach $x_{random1}$ ungültig. Die Zustände $x_{random1}$ und x_{new1} werden gelöscht. Dann wird durch Wiederholung von Schritt 2 ein neuer Punkt $x_{random2}$ generiert. Nach Wiederholung von Schritt 3 wird der Punkt x_{new2} als gültiger Punkt gesetzt. Der RRT Algorithmus ist

folgendermaßen konstruiert:

Algorithm 1: RRT

Input: $RRT(z_{init})$

Output: $T = (V, E)$

```
1  T ← InitializeTree();
2  T ← InsertNode(∅, z_init, T);
3  for i = 0 to i = N do
4  |   z_rand ← Sample(i);
5  |   z_nearest ← Nearest(T, z_rand);
6  |   (z_new,U_new) ← Steer(z_nearest, z_rand);
7  |   if Obstaclefree(z_new) then
8  |   |   T ← InsertNode (z_min, z_new, T);
9  |   end
10 end
11 return T;
```

Nach Einführung in die Literatur wurden theoretische Grundlagen des RRT-Algorithmus vorgestellt, darunter die probabilistische Vollständigkeit [44] und die exponentielle Zerfallsrate der Ausfallwahrscheinlichkeit [45]. Insbesondere wurde gezeigt, dass der RRT-Algorithmus für Systeme mit verschiedenen Beschränkungen, nichtlinear Dynamik und nichtholonomer Zwänge sowie für rein diskrete oder hybride Systeme effektiv funktioniert. Zudem hat sich der RRT-Algorithmus auf verschiedenen experimentellen Roboterplattformen bewährt.

Während der RRT-Algorithmus in der Praxis gut funktioniert und die Vollständigkeit der Lösung garantieren kann, besteht das Problem, dass RRT Monte-Carlo-Zufallsstichproben nutzt, die jedoch kaum auf die Qualität der Ergebnisse achten. Es ist erwiesen, dass RRT-Algorithmen nicht asymptotisch optimal sind. Die Methode benötigt viel Zeit, um in überfüllten Umgebungen Auswege zu finden. Abbildungen 4.7, 4.8 und 4.9 zeigen RRT-Simulationen. Darin zeigt sich, dass die Erweiterung des RRT-Algorithmusbaums nicht optimal, völlig zufällig und ungerichtet ist. Wenn die Pfadplanung in einer komplexen Umgebung mit dichten Hindernissen oder engen Kanälen durchgeführt wird, erhöht sich die Berechnungszeit des Algorithmus erheblich, und der Algorithmusbaum weist mit zunehmender Pfadlänge eine große Redundanz auf. Abbildungen 4.8 und 4.9 zeigen, dass der gültige Pfad von Anfang bis Ende durch einen engen Korridor gehen muss. Nur wenn die Zufallsstichproben in engen Korridoren liegen, kann dar RRT-Baum effektiv erweitert werden.

Um die Qualität der Ergebnisse zu erhöhen, wird der erweiterte RRT-Algorithmus vor-

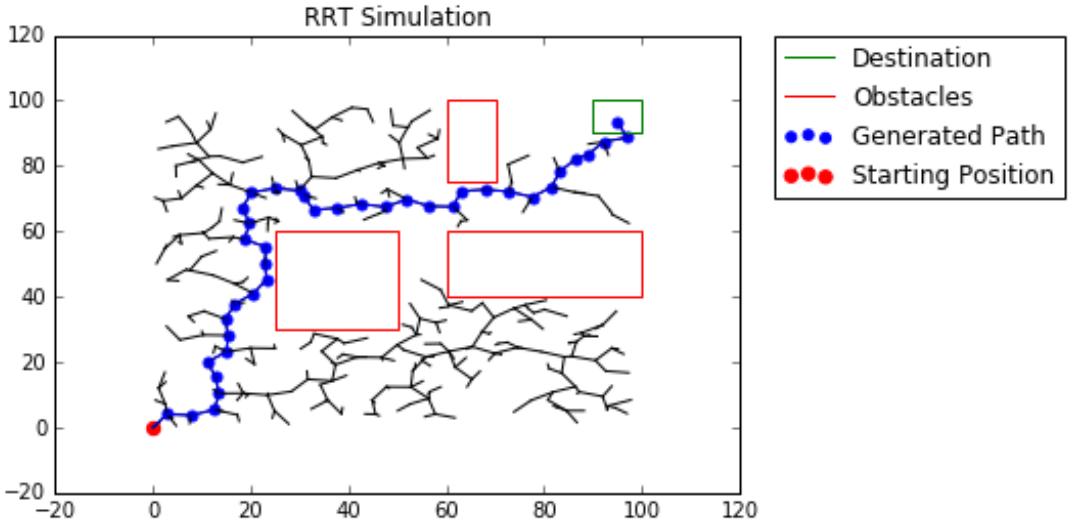


Abbildung 4.7: RRT Simulation [7]

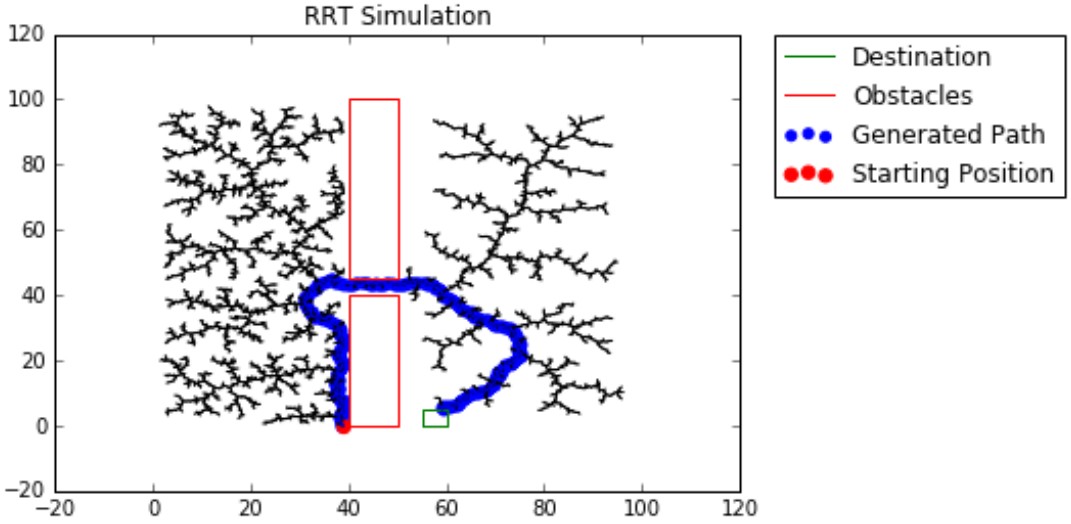


Abbildung 4.8: RRT Simulation [7]

gestellt, der als RRT*-Algorithmus bezeichnet wird und eine signifikante Verbesserung der Präzision der RRT erreicht. Es wird gezeigt, dass RRT* eine asymptotisch suboptimale Lösung liefert. Zum Unterschied von RRT werden die wahrscheinliche schlechte Verbindungen im RRT*-Baum entfernt. Dadurch werden die Lösungen so optimiert, dass sie kostengünstiger als RRT sind. RRT* erbt alle Eigenschaften von RRT und funktioniert ähnlich wie RRT. Es wurden jedoch zwei optimierte Funktionen eingeführt, die als Nahe-Nachbar-Operationen(Englisch: near neighbor search) und Neuverbindung-Operation (Englisch: rewire) bezeichnet werden. Die Nahe-Nachbar-Operation findet den besten Elternknoten für den neuen Knoten, bevor sie in den Baum eingefügt wird. Dieser Vorgang wird im Bereich einer Kugel mit dem Radius (R) durchgeführt. Durch die Neuverbindung-Operation wird der Baum innerhalb dieses Radius des Bereichs k neu

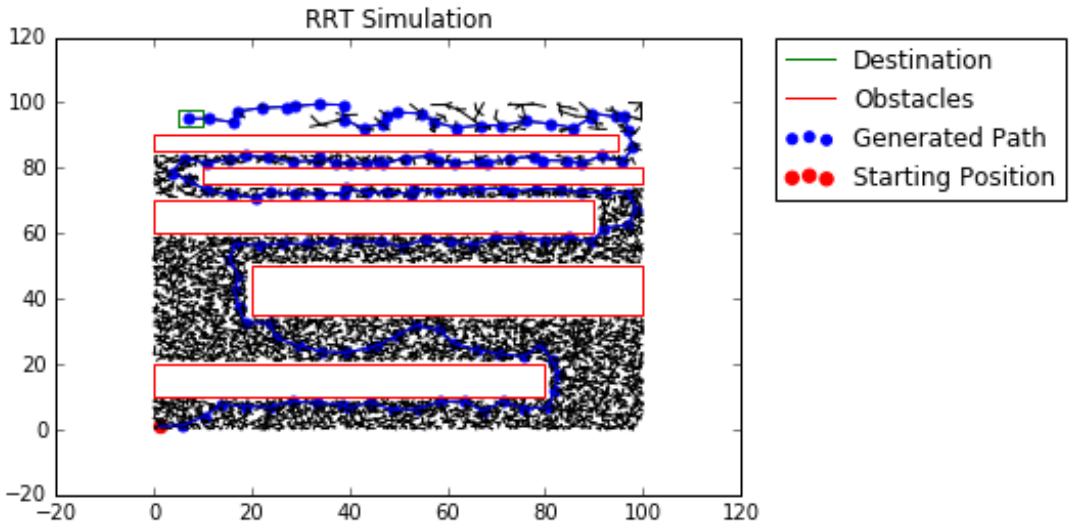


Abbildung 4.9: RRT Simulation [7]

erstellt, um die Kosten zwischen den Baumverbindungen zu minimieren [46][47].

Wenn mit dem Baum ein zufälliger Knoten hinzugefügt wird, wählt der RRT den nächsten Nachbarn als Elternknoten für diesen neuen Knoten aus. RRT* aber wählt den besten Nachbarn als Elternknoten für den neuen Knoten aus. Beim Finden des nächsten Nachbarn berücksichtigt RRT* alle Knoten innerhalb einer Nachbarschaft der Zufallsstichprobe. Hier wird die Kostenfunktion (p) definiert, um die Kosten des eindeutigen Pfades von X_{init} zu einem beliebigen Zustand $p \in P$ darzustellen. Zur Initialisierung wird der Wert von Kosten(X_{init}) auf null gesetzt. RRT* ermittelt dann die Kosten für die Verbindung zu jedem dieser Knoten. Der Knoten mit den niedrigsten Verbindungskosten zum Erreichen der Zufallsstichprobe wird als Elternknoten ausgewählt und dem Baum hinzugefügt [48].

Der RRT*-Algorithmus beginnt auf die gleiche Weise wie der RRT-Algorithmus. Bei der Auswahl des nächsten Nachbarn wählt der Algorithmus jedoch auch Knoten Q_{near} im Baum aus, die sich in der Nähe der Zufallsstichproben- q_{rand} befinden. Zeile 6 des Algorithmus 2 ist der erste Hauptunterschied zwischen RRT* und RRT. Anstatt den nächsten Nachbarn zur Zufallsstichprobe auszuwählen, wählt die Funktion *ChooseParent()* den be-

sten Elternknoten aus der Nachbarschaft der Knoten aus [46].

Algorithm 2: RRT*

Input: (q_{init})
Output: $T = (V, E)$

1 $T \leftarrow$ InitializeTree();
2 $T \leftarrow$ InsertNode($\varnothing, q_{init}, T$);
3 **for** $k = 1$ *to* N **do**
4 $\quad q_{rand} \leftarrow$ RandomSample(k);
5 $\quad q_{nearest} \leftarrow$ NearestNeighbor(q_{rand}, Q^{near}, T);
6 $\quad q_{min} \leftarrow$ ChooseParent($q_{rand}, Q^{near}, q_{nearest}, \triangle q$);
7 $\quad T \leftarrow$ InsertNode(q_{min}, q_{rand}, T);
8 $\quad T \leftarrow$ Rewire($T, Q_{near}, q_{min}, q_{rand}$);
9 **end**
10 return T;

Algorithmus 3 beschreibt die Funktion *ChooseParent()*. Diese Funktion verwaltet den Knoten mit den niedrigsten Gesamtkosten für das Erreichen von q_{rand}. In Zeile 1 von Algorithmus 3 wird der nächste Nachbar, $q_{nearest}$, als der Nachbar mit den niedrigsten Kosten q_{min} betrachtet. In Zeile 2 werden die Kosten, die mit dem Erreichen der neuen Zufallsstichprobe q_{rand} unter Verwendung von $q_{nearest}$ als Eltern verbunden sind, als die aktuell niedrigsten Kosten C_{min} gespeichert. Der Algorithmus durchsucht dann die Menge der Knoten in der Nähe von q_{rand}. Die Funktion *Steer()* in Zeile 4 von Algorithmus 3 gibt einen Pfad vom nahegelegene Knoten q_{near} zu q_{rand} zurück. Wenn dieser Pfad frei von Hindernissen ist und niedrigere Kosten als die aktuellen Mindestkosten hat, wird der nahe gelegene Knoten zum besten Nachbar, q_{min}, und diese Kosten werden zu den niedrigsten Kosten c_{min} (Zeilen 7–9 von Algorithmus 3). Wenn alle nahegelegenen Knoten untersucht wurden, gibt die Funktion den besten Nachbarn zurück. Der neue Zufallsknoten wird mit q_{min} als Elternknoten in den Baum eingefügt. Der nächste Schritt ist der zweite große Unterschied zwischen dem RRT*- und dem RRT-Algorithmus. Zeile 8 von Algorithmus 2

ruft die Funktion *Rewire()* auf [46].

Algorithm 3: RRT*-ChooseParent

Input: $(q_{rand}, Q^{near}, q_{nearest}, \triangle q)$

Output: q_{min}

```
1  q_min ← q_nearest;
2  c_min ← Cost(q_nearest) + c(q_rand) ;
3  for q_near ∈ Q_near do
4      q_path ← Steer(q_near, q_rand, △q);
5      if ObstacleFree(q_path) then
6          c_new ← Cost(q_near) + c(q_path);
7          if c_new < c_min then
8              c_min ← c_new;
9              q_min ← q_near;
10         end
11     end
12 end
13 return q_min;
```

Die in Algorithmus 4 beschriebene *Rewire*-Funktion ändert die Baumstruktur basierend auf dem neu eingefügten Knoten q_{rand}. Diese Funktion verwendet wiederum die nahe Nachbarschaft der Knoten Q_{near} als Kandidaten für die Neuverbindung. Die *Rewire*-Funktion verwendet die *Steer*-Funktion, um den Pfad abzurufen. Der Pfad beginnt am neuen Knoten q_{rand} und geht zum nahe gelegenen Knoten q_{near}. Wenn dieser Pfad frei von Hindernissen ist und die Gesamtkosten dieses Pfads niedriger als die aktuellen Kosten für das Erreichen von q_{near} sind (Zeile 3 von Algorithmus 3), ist der neue Knoten q_{rand} ein besserer Elternknoten als der aktuelle Elternknoten von q_{near}. Der Baum wird dann neu verkabelt, um die Verbindung zum aktuellen Elternknoten von q_{near} zu löschen und eine Verbindung hinzuzufügen, um q_{rand} zum Elternknoten von q_{near} zu machen. Dies erfolgt mit der Funktion *ReConnect* in Zeile 4 von Algorithmus 4. Die Funktionen *ChooseParent* und *Rewire* ändern die Struktur des Suchbaums im Vergleich zum RRT-Algorithmus. Der vom RRT generierte Baum weist Verzweigungen auf, die sich in alle Richtungen bewegen. Der vom RRT*-Algorithmus erzeugte Baum weist selten Zweige auf, die sich in Richtung

des Elternknoten zurückbewegen [49].

Algorithm 4: RRT*-Rewire

Input: $(T, Q_{near}, q_{min}, q_{rand})$

Output: T

```
for q_near ∈ Q_near do
    q_path ← Steer(q_rand,q_near);
    if ObstacleFree(q_path) and Cost(q_rand) + c(q_path) < Cost(q_near) then
        T ← ReConnect(q_rand, q_near, T)
    end
end
return T;
```

Die Funktion *ChooseParent()* stellt sicher, dass Pfade erstellt werden und sich immer vom Startort entfernen [50]. Die Funktion *Rewire()* ändert die interne Struktur des Baums, um sicherzustellen, dass interne Knoten keine unnötigen Schritte auf einem erkannten Pfad hinzufügen. Die Funktionen *ChooseParent()* und *Rewire* garantieren, dass die erkannten Pfade asymptotisch suboptimal sind, da diese Funktionen immer die Kosten minimieren, um jeden Knoten innerhalb des Baums zu erreichen[46].

Abbildung 4.10 verdeutlicht dies anhand eines Beispiels. Der ursprüngliche RRT-Baum ähnelt der linken Abbildung. Der gelbe Knoten repräsentiert q_{rand}, der aus den Zufallsstichproben generiert wird. Es gibt vier Knoten in der Nähe von q_{rand} als Q_{near}. Nach der Kostenberechnung und der Neuverbindung wird die Verbindung zwischen X_1 und X_2 gelöscht. Die Knoten X_2 und q_{rand} sind verbunden.

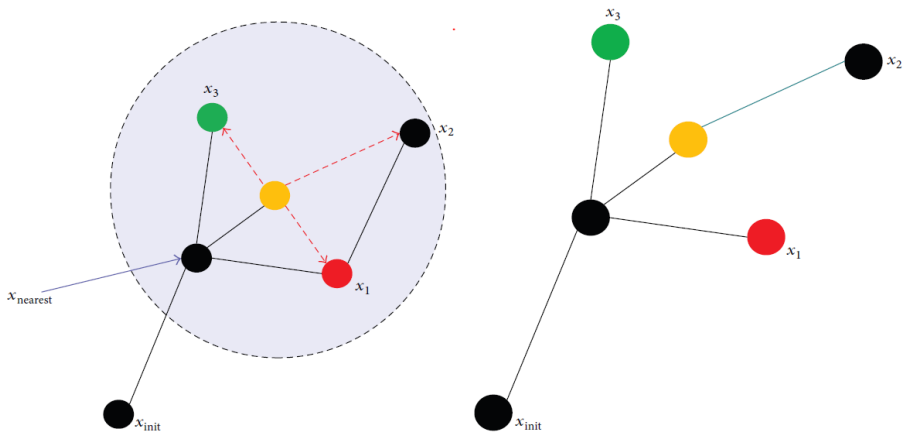


Abbildung 4.10: RRT*

Abbildung 4.11 zeigt die Simulation von RRT und RRT* mit derselben Umgebung. Beide Algorithmen wurden mit derselben Probensequenz 20.000 Mal ausgeführt. Die Kosten für den besten Pfad in der RRT und der RRT* betrugen 21,02 und 14,51 Sekunden. Es ist

ersichtlich, dass der RRT*-Algorithmus eine asymptotisch suboptimale Lösung erzeugt. Zwar verbessert RRT* die Ergebnisse gegenüber dem RRT-Algorithmus, aber der RRT* basiert genau wie RRT auf Zufallsstichproben. Das heißt, dass die Probenahme jeder Iteration gleichmäßig über die gesamte Umgebung verteilt ist. Wie in der Abbildung 4.11 zu sehen, sind die Probennahmen von RRT und RRT* gleichmäßig auf der Karte verteilt, anstatt in der Nähe der optimalen Pfade konzentriert zu sein. Probenahmen weit entfernt vom optimalen Pfad sind meist redundant. Wenn die Umgebung relativ weitläufig ist oder die Planung in hohen Dimensionen erfolgt, ist RRT* ineffizient.

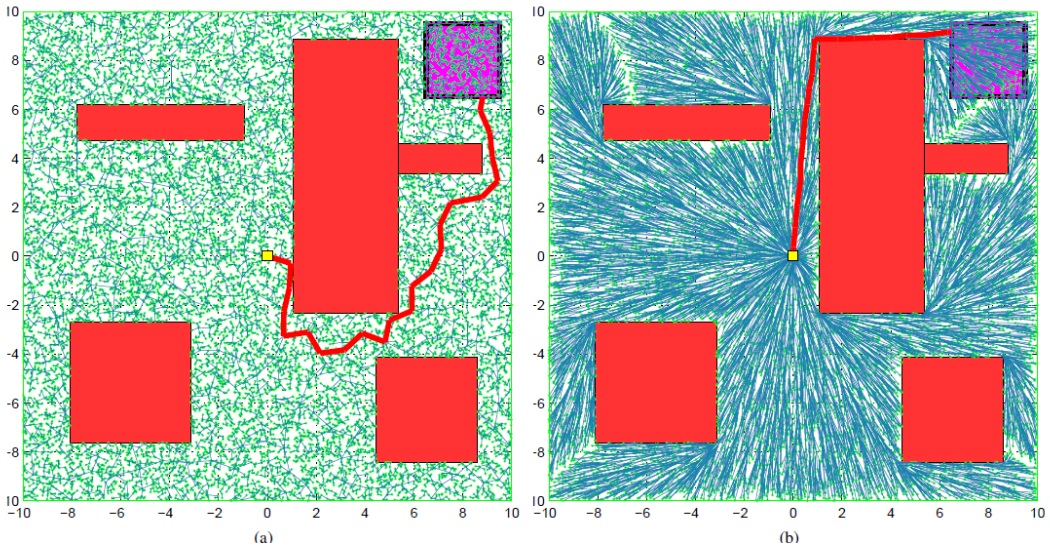


Abbildung 4.11: RRT und RRT* [8]

Daher wird im Folgenden der informierte RRT* vorgestellt, um die Vorteile der informierten inkrementellen Suche zu demonstrieren. Der informierte RRT* verhält sich wie RRT*, bis eine erste Lösung gefunden wird. Danach werden die Stichproben nur aus der Teilmenge von Zuständen abgetastet, die durch eine zulässige Heuristik definiert sind, um nach Möglichkeit die Lösung zu verbessern. Diese Teilmenge gleicht implizit die Ausbeutung mit der Exploration aus und erfordert keine zusätzliche Abstimmung (d. H. es gibt keine zusätzlichen Parameter) oder Annahmen (d. H. alle relevanten Homotopieklassen werden durchsucht).

Algorithmus 5 ist ein Beispielalgorithmus mit direkter, informierter Abtastung, Informed RRT*. Es fügt dem zuvor dargestellten RRT*-Algorithmus die Zeilen 3, 6, 7, 30 und 31 hinzu. Wie RRT*, sucht dieser nach dem optimalen Pfad zu einer Planung durch schrittweises Erstellen eines Baums im Zustandsraum $\tau = (V, E)$, bestehend aus einer Menge von Punkten $V \subseteq X_{free}$ und Kanten $E \subseteq X_{free} \times X_{free}$. Neue Punkte werden hinzugefügt, indem der Graph im freien Raum in Richtung zufällig ausgewählter Zustände vergrößert wird. Der Baum wird mit jedem neuen Punkt neu verbunden, sodass die Kosten für die

nahegelegenen Punkte minimiert werden. Der Algorithmus unterscheidet sich von RRT* darin, dass er sich nach dem Finden einer Lösung auf den Teil des Planungsproblems konzentriert, der die Lösung verbessern kann. Dies geschieht durch direkte Abtastung der ellipsoiden Heuristik. Sobald Lösungen gefunden wurden (Zeile 30), fügt der informierte RRT* sie einer Liste möglicher Lösungen hinzu (Zeile 31). Die Lösung der niedrigsten Kosten wird verwendet (Zeile 6), um $X_{\hat{f}}$ direkt zu berechnen und abzutasten (Zeile 7). Wie üblich wird das Minimum einer leeren Liste als unendlich angenommen.

Algorithm 5: Informierter RRT*

Input: (x_{start}, x_{goal})
Output: τ

$V \leftarrow \{x_{start}\}$;
$E \leftarrow \varnothing$;
$X_{soln} \leftarrow \varnothing$;
$\tau = (V, E)$;
for *iteration* = 1 *to* N **do**
  $c_{best} \leftarrow min_{x_{soln} \in X_{soln}} \{\text{Cost}(x_{soln})\}$;
  $x_{rand} \leftarrow \text{Sample}(x_{start}, x_{goal}, c_{best})$;
  $x_{nearest} \leftarrow \text{Nearest}(\tau, x_{rand})$;
  $x_{new} \leftarrow \text{Steer}(x_{nearest, x_{rand}})$;
  if *CollisionFree*$(x_{nearest, x_{new}})$ **then**
    $V \leftarrow \cup \{x_{new}\}$;
    $X_{near} \leftarrow \text{Near}(\tau, x_{new}, r_{RRT*})$;
    $x_{min} \leftarrow x_{nearest}$;
    $c_{min} \leftarrow \text{Cost}(x_{min}) + c \cdot \text{Line}(x_{nearest}, x_{new})$;
    for $\forall x_{near} \in X_{near}$ **do**
      $c_{new} \leftarrow \text{Cost}(x_{near}) + c \cdot \text{Line}(x_{near}, x_{new})$;
      if $c_{new} < c_{min}$ **then**
        if *CollisionFree*(x_{near}, x_{new}) **then**
          $x_{min} \leftarrow x_{near}$;
          $c_{min} \leftarrow c_{new}$;
    end
    $E \leftarrow E \{(x_{min}, x_{new})\} \cup$;
    for $\forall x_{near} \in X_{near}$ **do**
      $c_{near} \leftarrow \text{Cost}(x_{near})$;
      $c_{new} \leftarrow \text{Cost}(x_{new}) + c \cdot \text{Line}(x_{new}, x_{near})$;
      if $c_{new} < c_{near}$ **then**
        if *CollisionFree*(x_{new}, x_{near}) **then**
          $x_{parent} \leftarrow \text{Parent}(x_{near})$;
          $E \leftarrow \frac{E}{\{(x_{parent}, x_{near})\}}$;
          $E \leftarrow E \cup \{(x_{new}, x_{near})\}$;
    end
    if *InGoalRegion*(x_{new}) **then**
      $X_{soln} \leftarrow X_{soln} \cup \{x_{new}\}$;
end
return τ;

- *Sample*: Bei zwei Punkten $x_{from}, x_{to} \in X_{free}$, und einem maximalen heuristischen Wert, $c_{max} \in \mathbb{R}$, gibt die Funktion $Sample(x_{from}, x_{to}, c_{max})$ unabhängige und verteilte Samples aus dem Zustandsraum, $x_{new} \in X$, zurück, so dass die Kosten eines optimalen Pfades zwischen x_{from} und x_{to}, der x_{new} durchlaufen muss, kleiner als c_{max} sind (wie in Algorithmus 6). Bei den meisten Planungsproblemen $x_{from} \equiv x_{start}$, $x_{to} \equiv x_{goal}$, und die Zeilen 2 bis 4 von Algorithmus 6 werden zu Beginn des Problems einmal berechnet.
- *InGoalRegion*: Bei einem Punkt $x \in X_{free}$ gibt die Funktion *InGoalRegion* den Zustand *True* zurück, wenn der Punkt sich in der Zielregion X_{goal} befindet, andernfalls *False*. Eine gemeinsame Zielregion ist eine Kugel mit dem Radius r_{goal} um dem Ziel. D. h.$X_{goal} = \{x \in X_{free} \mid \|x - x_{goal}\|_2 \leqslant r_{goal}\}$.
- *RotationToWorldFrame*: Bei zwei Punkten als Brennpunkte eines Hyperellipsoids, $x_{from}, x_{to} \in X$, gibt die Funktion *RotationToWorldFrame* (x_{from}, x_{to}) die Rotationsmatrix $C \in SO(n)$ von der hyperellipsoid ausgerichteten Koordinate zur Weltkoordinate zurück. Wie bereits erwähnt, muss diese Rotationsmatrix bei den meisten Planungsproblemen zu Beginn des Problems nur einmal berechnet werden [9].
- *SampleUnitNBall*: Die Funktion *SampleUnitNBall* gibt ein einheitliches Sample aus dem Volumen einer Kugel mit einem Einheitsradius zurück, d. h. $x_{ball} \sim \upsilon(X)$.

Algorithm 6: Informiertes RRT*-Sample

Input: $(x_{start}, x_{goal}, c_{max})$
Output: x_{rand}

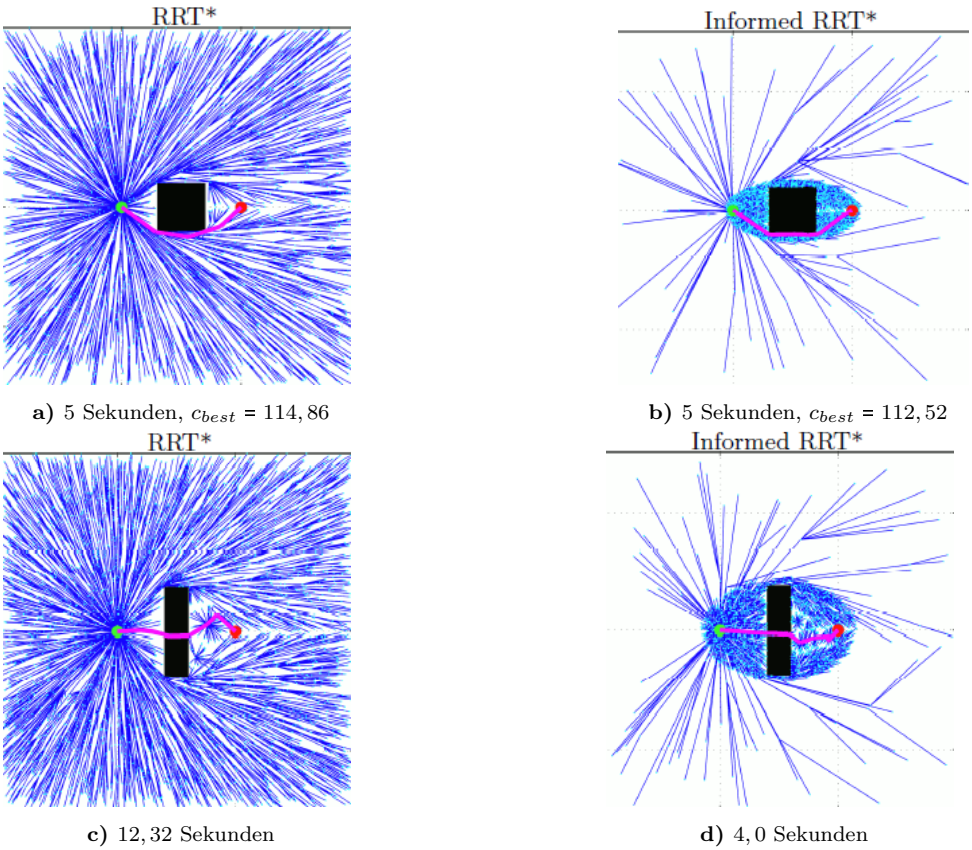
1 **if** $c_{max} < \infty$ **then**
2 $\quad c_{min} \leftarrow \|x_{goal} - x_{start}\|_2$;
3 $\quad x_{centre} \leftarrow \frac{(x_{start}+x_{goal})}{2}$;
4 $\quad C \leftarrow$ RotationToWorldFrame(x_{start}, x_{goal});
5 $\quad r_1 \leftarrow \frac{c_{max}}{2}$;
6 $\quad \{r_i\}_{i=2,\cdots,n} \leftarrow \frac{(\sqrt{c_{max}^2 - c_{min}^2})}{2}$;
7 $\quad L \leftarrow diag\{r_1, r_2, \cdots, r_n\}$;
8 $\quad x_{ball} \leftarrow$SampleUnitBall;
9 $\quad x_{rand} \leftarrow (CLx_{ball} + x_{centre}) \cap X$;
10 **end**
11 $x_{rand} \sim \upsilon(X)$;
12 return x_{rand};

Bei jeder Iteration muss der Rewire-Radius r_{RRT*} groß genug sein, um eine fast sichere asymptotische Konvergenz zu gewährleisten, aber klein genug, um einige gefügige Rewire-Kandidaten zu erzeugen. Der informierte RRT* tastet die Teilmenge des Planungspro-

blems, die die Lösung verbessern kann, gleichmäßig ab. Aus die informierte Teilmenge und die zugehörigen Punkten darin werden ein Rewire-Radius berechnet. Dieser aktualisierte Radius reduziert die erforderliche Neuverdrahtung und verbessert die Leistung vom informierten RRT* weiter.

Der informierte RRT* wird mit dem RRT* bei einer Vielzahl einfacher Planungsprobleme (Abbildungen 4.12) und zufällig erzeugter Umgebungen verglichen. Die Abbildungen 4.12 a) und 4.12 b) zeigen die Recheneffizienz des RRT* und des informierten RRT*, wobei die Breite des Hindernisses zufällig ausgewählt wurde. In fünf Sekunden finden beide Algorithmen eine supoptimale Lösung. Der informierte RRT* kann immer eine gegenüber RRT* signifikant reduzierte Domäne durchsuchen, was sowohl die Konvergenzrate als auch die Qualität der endgültigen Lösung erhöht. Im Vergleich dazu benötigt benötigt der RRT*-Algorithmus (4.12) erhebliche Rechenressourcen für die Erforschung von Regionen des Planungsproblems, die die aktuelle Lösung möglicherweise nicht verbessern. Abbildungen 4.12 c) 4.12 d) zeigen, dass beide Algorithmen Pfade durch eine Wand mit einem 3%-außermittigen Spalt finden. Durch Fokussieren des Suchraums auf die Teilmenge von Zuständen, die eine anfängliche Lösung verbessern können, die das Hindernis flankiert, kann der informierte RRT* in vier Sekunden einen Pfad durch die enge Öffnung finden, während RRT* 12,32 Sekunden benötigt. Abbildung 4.12 d) verdeutlicht, dass die vom informierten RRT* gesuchte Trajektorie in einer Ellipse konzentriert ist, während die von RRT* gesuchte Trajektorie divergent ist. Das bedeutet, dass der informierte RRT*-Algorithmus zielorientiert ist.

Zusammenfassend lässt sich festhalten, dass der informierte RRT* in der Lage ist, nahezu optimale Lösungen mit deutlich weniger Iterationen als der RRT* zu finden. Die direkte Abtastung der informierten Teilmenge erhöht die Dichte um die optimale Lösung schneller als die globale Abtastung und erhöht daher die Wahrscheinlichkeit, die Lösung zu verbessern und die Suche stärker weiter zu fokussieren. Im Gegensatz dazu weist RRT* über den gesamten Planungsraum eine gleichmäßige Dichte auf und verringert tatsächlich die Wahrscheinlichkeit, weitere Verbesserungen zu finden. Der informierte RRT* garantiert die gleiche Wahrscheinlichkeit für Vollständigkeit und Optimalität wie RRT* und verbessert gleichzeitig die Konvergenzrate und Qualität der endgültigen Lösung [9]. Außerdem zeigt die Methode weniger Abhängigkeit von der Zustandsdimension und dem Bereich des Planungsproblems. In dieser Arbeit wird daher für die Pfadplanung in drei Dimensionen der informierte RRT* verwendet.



a) 5 Sekunden, c_{best} = 114, 86

b) 5 Sekunden, c_{best} = 112, 52

c) 12, 32 Sekunden

d) 4, 0 Sekunden

Abbildung 4.12: Leistungsvergleich zwischen RRT* und informiertem RRT* [9]

4.2.2 A*-Algorithmus

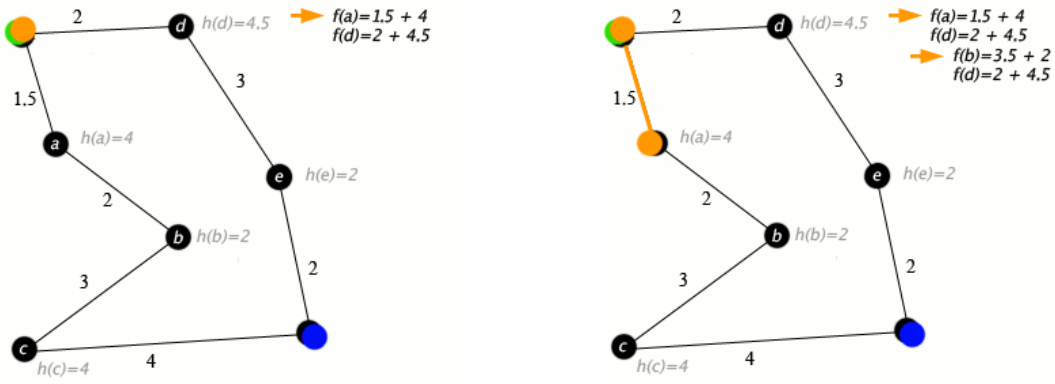
Der A*-Algorithmus (auch A*-Suche; Englisch: A* search algorithm) dient in der Informatik der Berechnung des kürzesten Pfades zwischen zwei Knoten in einem Graphen mit positiven Kantengewichten. Er wurde 1968 von Peter Hart, Nils J. Nilsson und Bertram Raphael beschrieben. Wie der informierte RRT*-Algorithmus, verwendet auch der A*-Algorithmus eine Heuristikfunktion (Schätzfunktion), um zielgerichtet zu suchen und damit die Laufzeit zu verringern. Der Algorithmus ist vollständig und optimal, d. h., dass immer eine optimale Lösung gefunden wird, falls eine solche existiert [10].

Im Gegensatz zum informierten RRT*-Algorithmus zielt die Heuristik des A*-Algorithmus nicht darauf ab, mögliche Lösungen zu finden, sondern darauf, den Abstand vom Start zum Ziel zu schätzen. Bei jeder Iteration des Hauptloop muss A* bestimmen, in welche Richtung ein Pfade erweitert werden soll. Dies geschieht auf Basis der geschätzten Kosten aus die Schätzfunktion sowie der Knoten der Open List und der Closed List. Die Knoten sind Punkte in einem Graphen, durch die Pfade verlaufen können. Der A*-Algorithmus untersucht immer jene Knoten zuerst, die wahrscheinlich schnell zum Ziel führen. Um den vielversprechendsten Knoten zu ermitteln, wird allen bekannten Knoten x jeweils ein

Wert $f(x)$ zugeordnet. Dieser entspricht der geschätzten Länge des Pfades vom Start zum Ziel unter Verwendung des betrachteten Knotens im günstigsten Fall. Als nächstes wird der Knoten mit dem niedrigsten f-Wert untersucht.

$$f(x) = g(x) + h(x) \tag{4.1}$$

Für einen Knoten x bezeichnet $g(x)$ die bisherigen Kosten vom Startknoten aus, um x zu erreichen, während $h(x)$ die geschätzten Kosten von x bis zum Zielknoten repräsentiert. Die verwendete Heuristik darf die Kosten nicht überschätzen. Für das Beispiel der Wegsuche ist die Luftlinie eine geeignete Schätzung. Die tatsächliche Strecke ist nie kürzer als die direkte Verbindung. Abbildung 4.13 zeigt die Funktion der Heuristik und die Berechnung des f-Werts. Die Zahlen auf den schwarzen Linien geben jeweils die Entfernung zwischen zwei Knoten an. Der h-Wert jedes Knotens ist der geschätzte Abstand des Knotens vom Zielknoten. Am Anfang wird der f-Wert für die Knoten a und d berechnet, um den günstigsten Knoten auszuwählen. Dabei ist $g(a)$ gleich $1,5$ und $g(d)$ gleich 2. Die Werte von $f(a)$ und $f(d)$ können mit den Werten von $h(a)$ und $h(b)$ verglichen werden. In Abbildung 4.13 ist $f(a)$ $5,5$und $f(d)$ $6,5$, also die größer als $f(a)$. Deswegen wird der Pfad zuerst in die a-Richtung erweitert. Es werden alle Nachfolgeknoten vom Startknoten betrachtet. Für den Knoten b wird der Wert von $f(b)$ mit $f(d)$ verglichen. $g(b)$ ist gleich $g(a)$ zuzüglich des Abstands zwischen a und b, d.h. $g(b)$ ist $3,5$ und $g(d)$ ist 2. Im Vergleich zu $f(d)$ ist $f(b)$ noch kleiner. Somit erstreckt sich der Pfad weiter zum b-Knoten.



a) A*-Schätzfunktion **b)** A*-Schätzfunktion

Abbildung 4.13: A*

Die Knoten werden während der Suche in drei verschiedene Klassen eingeteilt, nämlich unbekannte Knoten, bekannte Knoten und abschließend untersuchte Knoten. Die unbekannten Knoten wurden während der Suche noch nicht gefunden. Zu ihnen ist noch kein Weg bekannt. Jeder Knoten (außer dem Startknoten) ist zu Beginn des Algorithmus unbekannt. Zu den bekannten Knoten ist ein (möglicherweise suboptimaler) Weg bekannt.

Alle bekannten Knoten werden zusammen mit ihrem jeweiligen f-Wert in der *Open List* gespeichert. Aus dieser Liste wird immer der vielversprechendste Knoten ausgewählt und untersucht. Die Implementierung der *Open List* hat großen Einfluss auf die Laufzeit und wird oft als einfache Prioritätswarteschlange (z. B. binärer Heap) realisiert. Zu abschließend untersuchten Knoten ist der kürzeste Weg bekannt. Die abschließend untersuchten Knoten werden in der *Closed List* gespeichert, damit sie nicht mehrfach untersucht werden. Um effizient entscheiden zu können, ob sich ein Element auf der *Closed List* befindet, wird diese oft als Menge implementiert. Die *Closed List* ist zum Beginn leer [51] [52].

Jeder bekannte oder abschließend untersuchte Knoten enthält einen Zeiger auf seinen (bisher besten) Vorgängerknoten, damit der Pfad bis zum Startknoten zurückverfolgt werden kann. Wird ein Knoten x abschließend untersucht (auch expandiert oder relaxiert), so werden seine Nachfolgeknoten in die *Open List* eingefügt und x in die *Closed List* aufgenommen. Für neu eingefügte Nachfolgeknoten werden die Vorgängerzeiger auf x gesetzt. Ist ein Nachfolgeknoten bereits auf der *Closed List*, so wird er nicht erneut in die *Open List* eingefügt und auch sein Vorgängerzeiger nicht geändert. Ist ein Nachfolgeknoten bereits auf der *Open List*, so wird der Knoten nur aktualisiert (f-Wert und Vorgängerzeiger), wenn der neue Weg dorthin kürzer ist als der bisherige.

Sobald der Zielknoten abschließend untersucht wurde, endet dieser Durchlauf des Algorithmus. Der gefundene Weg wird mit Hilfe der Vorgängerzeiger rekonstruiert und ausgegeben. Falls die *Open List* leer ist, gibt es keine Knoten mehr, die untersucht werden könnten. In diesem Fall terminiert der Algorithmus, da es keine Lösung gibt. Bedingt durch die Vorgängerzeiger wird der gefundene Weg vom Ziel ausgehend rückwärts bis zum Start ausgegeben. Um den Weg in der richtigen Reihenfolge zu erhalten, können z. B. vor der Wegsuche Start und Ziel vertauscht werden. Somit wird vom eigentlichen Ziel zum Start gesucht und die Wegausgabe beginnt beim ursprünglichen Startknoten [53].

Die Diagramme in Abbildung 4.14 zeigen eine Wegfindung um ein Hindernis mittels A*-Suche. Die graue L-Form stellt das Hindernis dar, durch das der Pfad nicht verlaufen kann. Der hellgrüne Punkt in der oberen rechten Ecke ist das Ziel, und der Startpunkt befindet sich in der unteren linken Ecke. Bekannte Knoten sind hellblau umrandet, abschließend untersuchte Knoten sind ausgefüllt. Die Farbe letzterer markiert dabei die Entfernung zum Ziel; je grüner, desto weniger weit ist dieser vom Ziel entfernt. Zu beobachten ist, dass der A* zuerst in einer geraden Linie in Richtung Ziel strebt, bis er auf das Hindernis stößt. Erreicht er den Zielknoten, erkundet er zuerst noch alternative Knoten in der *Open List*, bevor er terminiert.

Es ist erwiesen, dass der A*-Algorithmus vollständig, optimal und optimal effizient ist [52]. D. h. A* expandiert eine minimale Anzahl an Knoten. Allerdings ist es immer schwie-

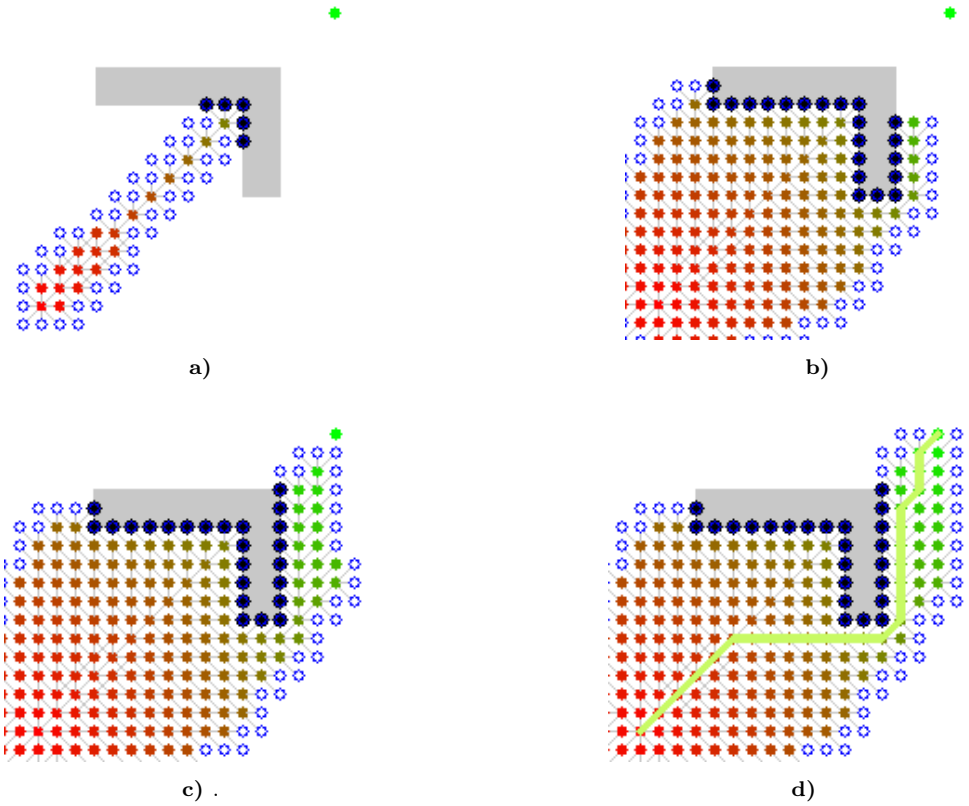


Abbildung 4.14: A*-Pfadplanung in zwei Dimensionen mit Gitter [10]

rig, die Heuristikfunktion auszuwählen. Ist die Heuristik nicht monoton, erhöht sich die Laufzeit exponentiell, da Knoten mehrfach verknüpft werden. Je genauer die Kostenabschätzung ist, desto weniger Knoten werden untersucht. Das Einstellen einer guten heuristischen Funktion kann die Genauigkeit und Recheneffizienz der Lösung erheblich verbessern. Außerdem ist der begrenzende Faktor bei A* oft nicht die Rechenzeit, sondern der Speicherplatz. Da alle bekannten Knoten im Speicher gehalten werden (*Open List* und *Closed List*), ist A* für viele Probleme nicht geeignet. Schon beim einfachen 15-Puzzle hat der komplette Graph bereits 16!=20.922.789.888.000 Knoten. Bei einem entsprechend langen Lösungsweg reicht der verfügbare Speicher nicht aus und A* kann keine Lösung finden [10].

4.2.3 CBS und ECBS

Der RRT-Algorithmus und der A*-Algorithmus konzentrieren sich auf die Routenplanung von Einzelagenten. Im Multi-Agenten Pfadfindung (MAPF) als Forschungsgegenstand dieser Arbeit sollen Pfade für mehrere Agenten mit jeweils unterschiedlicher Start- und Zielposition gefunden werden, so dass Agenten nicht kollidieren. Um das MAPF-

Problem zu lösen,wird im Folgenden die konfliktbasierte Suche (conflict-Based Search, CBS) vorgestellt, ein zweistufiger Algorithmus, bei dem die Suche auf hoher Ebene in einem Einschränkungsbaum (Constraint Tree (CT)) durchgeführt wird, dessen Knoten zeitliche und räumliche Einschränkungen für einen einzelnen Agenten enthalten. An jedem Knoten im Einschränkungsbaum wird eine Suche auf niedriger Ebene durchgeführt, um neue Pfade für alle Agenten unter den vom Knoten auf hoher Ebene vorgegebenen Einschränkungen zu finden. Im Gegensatz zu A*-basierten Suchvorgängen, bei denen der Suchbaum mit der Anzahl der Agenten exponentiell steigt, ist der Suchbaum von CBS in der Anzahl der Konflikte, die während des Lösungsprozesses auftreten, exponentiell. Das bedeutet, dass die Komplexität des Suchbaums mit steigender Anzahl von Konflikten exponentiell zunimmt. Der Zustandsraum von MAPF ist exponentiell in der Anzahl der Agenten. Im Gegensatz dazu ist bei einem Einzelagenten-Pfadfindungsproblem der Zustandsraum in der Diagrammgröße nur linear. Der CBS-Algorithmus löst das MAPF-Problem, indem es in eine große Anzahl von Einzelagenten-Pfadfindungsproblemen zerlegt wird. Jedes Problem ist relativ einfach zu lösen [11].

Im CBS-Algorithmus wird der Begriff Pfad nur im Kontext eines einzelnen Agenten verwendet. Die Lösung bedeutet eine Menge von k Pfaden für eine gegebene Menge von k Agenten. Eine Einschränkung für einen gegebenen Agenten a_i ist ein Tupel (a_i, v, t), bei dem es dem Agenten a_i verboten ist, den Punkt v zum Zeitpunkt t zu besetzen. Im Verlauf des Algorithmus werden Agenten mit Einschränkungen verbunden. Ein konsistenter Pfad für Agent a_i ist ein Pfad, der alle seine Einschränkungen erfüllt. Ebenso ist eine konsistente Lösung eine Lösung, die aus Pfaden besteht, so dass der Pfad für Agent a_i mit den Einschränkungen von a_i konsistent ist. Ein Konflikt ist ein Tupel (a_i, v, t), bei dem Agent a_i und Agent a_j zum Zeitpunkt t den Punkt v besetzen. Eine Lösung (von k Pfaden) ist gültig, wenn alle ihre Pfade keine Konflikte aufweisen. Eine konsistente Lösung kann ungültig sein, wenn diese Pfade trotz der Tatsache, dass sie mit den Einschränkungen der einzelnen Agenten übereinstimmen, immer noch Konflikte aufweisen. Die Schlüsselidee des CBS-Algorithmus besteht darin, eine Reihe von Einschränkungen für jeden Agenten zu erweitern und Pfade zu finden, die diesen Einschränkungen entsprechen. Wenn diese Pfade Konflikte aufweisen und daher ungültig sind, werden die Konflikte durch Hinzufügen neuer Einschränkungen gelöst. Der CBS-Algorithmus arbeitet auf zwei Ebenen. Auf hoher Ebene werden Konflikte gefunden und Einschränkungen hinzugefügt. Auf niedriger Ebene werden die Agentenpfade so aktualisiert, dass sie mit den neuen Einschränkungen übereinstimmen. Die beiden Teile werden im Folgenden ausführlicher beschrieben.

Auf hoher Ebene durchsucht der CBS-Algorithmus einen CT. Der CT ist ein Binärbaum, in dem jeder Knoten N die folgenden Datenfelder enthält.

- Eine Reihe von Einschränkungen (N.Einschränkungen). Die Wurzel des CT enthält einen leeren Satz von Einschränkungen. Das untergeordnete Element eines Knotens in der CT erbt die Einschränkungen des übergeordneten Knotens und fügt eine neue Einschränkung für einen Agenten hinzu.
- Eine Lösung (N.Lösungen). Eine Menge von k Pfaden, ein Pfad für jeden Agenten. Der Pfad für Agent a_i muss mit den Einschränkungen von a_i übereinstimmen. Solche Pfade werden auf der niedrigeren Ebene errechnet.
- Die Gesamtkosten (N.Kosten) der aktuellen Lösung (Summe aller Pfadkosten für einen einzelnen Agenten). Der Knoten N in dem CT ist ein Zielknoten, wenn die N.Lösung gültig ist, wenn also die Pfade für alle Agenten keine Konflikte aufweisen. Die hohe Ebene führt eine Best-First-Suche auf dem CT durch, bei der die Knoten nach ihren Kosten geordnet sind. Bindungen werden mithilfe einer Konfliktvermeidungstabelle (CAT) wie oben beschrieben unterbrochen.

Angesichts der Liste der Einschränkungen für einen Knoten N wird die Suche auf niedriger Ebene aufgerufen. Diese Suche gibt den kürzesten Pfad für jeden Agenten a_i zurück, der mit den Einschränkungen von a_i übereinstimmt. Sobald für jeden Agenten ein konsistenter Pfad in Bezug auf seine Einschränkungen gefunden wurde, werden diese Pfade in Bezug auf die anderen Agenten validiert. Die Validierung wird durchgeführt, indem die k Pfaden simuliert werden. Wenn alle Agenten ihr Ziel ohne Konflikte erreichen, wird dieser CT-Knoten N als Zielknoten deklariert und die aktuelle Lösung (N.Lösung), die die Pfade enthält, zurückgegeben. Wenn jedoch ein Konflikt $C = (a_i, a_j, v, t)$ für zwei oder mehr Agenten a_i und a_j während der Validierung gefunden wird, wird die Validierung angehalten und der Knoten als Nichtzielknoten deklariert.

Bei einem Nichtziel-CT-Knoten N, dessen Lösung N.Lösung einen Konflikt $C_n = (a_i, a_j, v, t)$ enthält, kann der Punkt v zum Zeitpunkt t in jeder gültigen Lösung höchstens bei einem der Konfliktagenten (a_i und a_j) besetzt werden. Daher muss mindestens eine der Bedingungen (a_i, v, t) oder (a_j, v, t) zu der Menge von Bedingungen in N.Bedingungen hinzugefügt werden. Um Optimalität zu gewährleisten, werden beide Möglichkeiten geprüft und N in zwei Kinder aufgeteilt. Beide Kinder erben die Menge der Einschränkungen von N. Das linke Kind löst den Konflikt durch Hinzufügen der Einschränkung (a_i, v, t) und das rechte Kind fügt die Einschränkung (a_j, v, t) hinzu.

Es ist zu beachten, dass für einen gegebenen CT-Knoten N nicht alle kumulativen Einschränkungen gespeichert werden müssen. Stattdessen kann der Knoten nur die letzte Einschränkung speichern und die anderen Einschränkungen fallenlassen, indem er den Pfad von N zur Wurzel über seine Vorfahren durchläuft. In ähnlicher Weise sollte die Suche auf niedriger Ebene nur nach Agent a_i durchgeführt werden, der der neu hinzu-

gefügten Einschränkung zugeordnet ist. Die Pfade anderer Agenten bleiben dieselben, da für sie keine neue Einschränkung hinzugefügt wurde. Die hohe Ebene des CBS wird in Algorithmus 7 dargestellt.

Algorithm 7: Die hohe Ebene von CBS

Input: MAPF Problem

Output: Lösungen der MAPF

```
1  R.constraints = ∅;
2  R.solution = find individual paths using the low-level();
3  R.cost = SIC(R.solution) insert R to OPEN;
4  while OPEN not empty do
5      P ← best node from OPEN // lowest solution cost;
6      Validate the paths in P until a conflict occurs;
7      if P has no conflict then
8          return P.solution // P is goal;
9      end
10     C ← first conflict C_n = (a_i, a_j, v, t) in P;
11     for agent a_i in C do
12         A ← new node;
13         A.constriants ← P.constriants +(a_i, s, t);
14         A.solution ← P.solution ;
15         Update A.solution by invoking low-level(a_i);
16         A.cost = SIC(A.solution);
17         insert A to OPEN;
18     end
19 end
```

Auf der niedrigen Ebene werden ein Agent, a_i und eine Reihe von zugehörigen Einschränkungen eingegeben, um einen optimalen Pfad für Agent a_i zu finden, der allen seinen Einschränkungen entspricht. Die anderen Agenten werden ignoriert. Diese Suche ist dreidimensional, da sie zwei räumliche Dimensionen und eine Zeitdimension umfasst. Jeder einzelne Agent-Pfadfindungsalgorithmus kann verwendet werden, um den Pfad für Agent a_i zu finden und gleichzeitig zu überprüfen, ob die Einschränkungen erfüllt sind. In dieser Arbeit wird der A*-Algorithmus mit einer perfekten Heuristik in beiden räumlichen Dimensionen verwendet. Für jeden CT-Knoten N wurde ein CAT auf niedriger Ebene verwendet. Er wird durch die aktuellen Pfade des Knotens N initialisiert. Wenn zwei Zustände auf niedriger Ebene dieselben f-Werte haben, wird der Zustand mit der geringsten Anzahl von Konflikten im CAT bevorzugt. Dies führt zu einer Lösung von höherer Qualität (weniger widersprüchliche Agenten) für jeden Knoten auf hoher Ebene.

Abbildung 4.15 zeigt ein Beispiel, in dem zwei Mäuse zu ihren jeweiligen Käsestücken gelangen müssen. Die entsprechende CT ist in rechter Abbildung dargestellt. Die Wurzel enthält einen leeren Satz von Einschränkungen. Die niedrige Ebene gibt nun eine optimale Lösung für jeden Agenten zurück (Zeile 2 von Algorithmus 7), $\langle S_1, A_1, C, G_1 \rangle$ für a_1 und $\langle S_2, B_1, C, G_2 \rangle$ für a_2. Somit betragen die Gesamtkosten dieses Knotens 6. Alle diese Informationen werden in diesem Knoten gespeichert. Die Wurzel wird dann in die *OPEN-Liste* eingefügt und als nächstes erweitert. Bei der Validierung der Zwei-Agenten-Lösung, die durch die beiden einzelnen Pfade (Zeile 7) gegeben ist, wird ein Konflikt gefunden, wenn beide Agenten zum Zeitpunkt 2 zum Scheitelpunkt C gelangen. Dies erzeugt den Konflikt $(a_1, a_2, C, 2)$. Infolgedessen wird die Wurzel als Nichtziel deklariert und zwei untergeordnete Elemente werden generiert, um den Konflikt zu lösen (Zeile 11). Das linke Kind fügt die Einschränkung $(a_1, C, 2)$ hinzu, während das rechte Kind die Einschränkung $(a_2, C, 2)$ hinzufügt. Die Suche auf niedriger Ebene wird jetzt aufgerufen (Zeile 15), um einen optimalen Pfad zu finden, der auch die neue Einschränkung erfüllt. Für das linke Kind muss a_1 einen Zeitschritt entweder bei S_1 (oder bei A_1) warten und der Pfad$\langle S_1, A_1, A_1, C, G_1 \rangle$ wird für a_1 zurückgegeben. Der Pfad für a_2, $\langle S_2, B_1, C, G_2 \rangle$ bleibt für das linke Kind unverändert. Die Gesamtkosten für das linke Kind betragen jetzt 7. In ähnlicher Weise wird das rechte Kind ebenfalls mit den Kosten 7 generiert. Beide Kinder werden zu *OPEN* hinzugefügt (Zeile 17). Im letzten Schritt wird das linke Kind für die Erweiterung ausgewählt, und die zugrunde liegenden Pfade werden validiert. Da keine Konflikte bestehen, wird das linke Kind als Zielknoten deklariert (Zeile 9) und seine Lösung als optimale Lösung zurückgegeben.

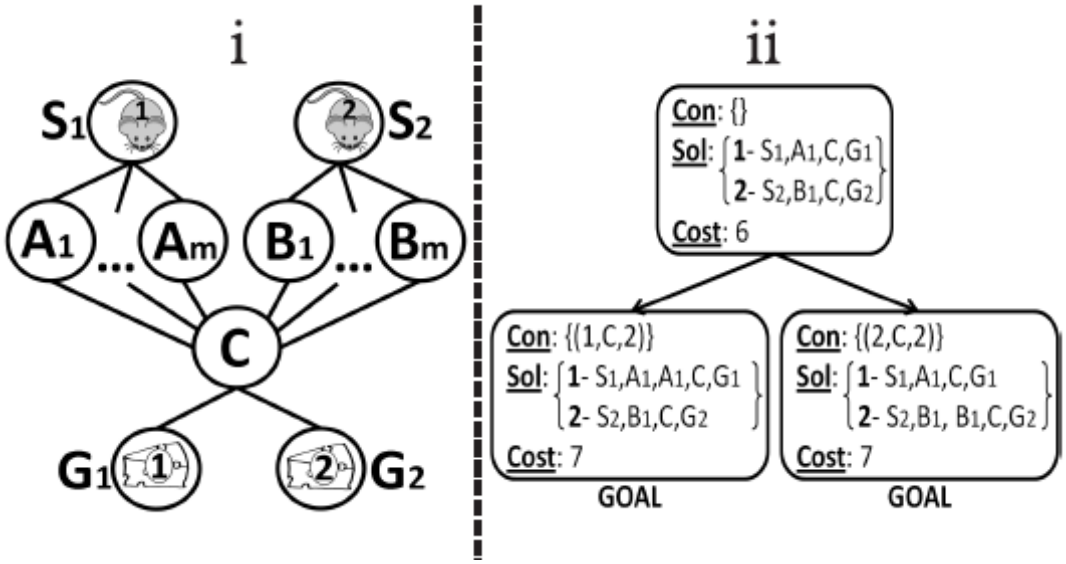


Abbildung 4.15: CBS-Pfadplanung [11]

Es lässt sich festhalten, eine sehr flexible zur optimalen Lösung des MAPF-Problems

darstellt, denn auf der niedrigen Ebene können bei Bedarf verschiedene Single-Path-Planungsalgorithmen (z. B. RRT und A*) verwendet werden. Auf hoher Ebene reduziert der Einschränkungsbaum die Zeitkomplexität für eine große Anzahl von Agenten erheblich. Eine große Herausforderung für den CBS-Algorithmus bestehlt allerdings, wenn die Umwelt kompliziert ist oder viele Konflikte zwischen Agenten auftreten. Um die Optimalität zu gewährleisten, führen sowohl die hohe- als auch die niedrige Ebene des CBS eine optimale Best-First-Suche durch. Wie bei jeder Best-First-Suche verursacht dies zusätzliche Arbeit, da die Knoten, die möglicherweise Lösungen sehr nahe liegen, verlassen werden, nur weil ihre Kosten sind hoch.

Enhanced CBS (ECBS) ist eine w-suboptimale Variante von CBS, deren Suchvorgänge auf hoher und niedriger Ebene eher eine Fokussuche als Best-First-Suche sind. Eine Fokussuche verwendet wie A* eine *OPEN*-Liste, deren Knoten n in aufsteigender Reihenfolge ihrer f-Werte $f(n) = g(n) + h(n)$ sortiert sind. Im Gegensatz zu A* verwendet eine Fokussuche mit dem Suboptimalitätsfaktor w und zwei Listen, *OPEN* und *FOCAL*. *OPEN* ist die reguläre *OPEN*-Liste des A*-Algorithmus. *FOCAL* enthält eine Teilmenge von Knoten aus *OPEN*. Die Fokussuche verwendet zwei beliebige Funktionen f_1 und f_2. f_1 definiert die Knoten, die sich in *FOCAL* befinden. $f_{1_{min}}$ ist der minimale f_1-Wert in *OPEN*. Bei einem Suboptimalitätsfaktor w enthält *FOCAL* alle Knoten n in *OPEN*, unter der Bedingung $f_1(n) \leq w \cdot f_{1_{min}}$. Mit f_2 wird bestimmt, welcher Knoten aus *FOCAL* erweitert werden soll. Wenn f_1 zulässig ist, wird garantiert, dass die zurückgegebene Lösung höchstens $w \times C$ ist.

Die *OPEN*-Liste besteht aus $OPEN_i$, das in der niedrigen ECBS-Ebene verwendet wird, wenn nach einem Pfad für Agent a_i gesucht wird. Der mit $f_{min}(i)$ bezeichnete minimale f-Wert in $OPEN_i$ ist eine Untergrenze für die Kosten des optimalen konsistenten Pfades für a_i (für den aktuellen CT-Knoten). Für einen CT-Knoten n, $LB(n) = \sum_{i=1}^{k} f_{min}(i)$, das bedeutet $LB(n) \leq n.cost \leq LB(n) \times w$. Im ECBS-Algorithmus gibt die niedrige Ebene für jeden erzeugten CT-Knoten n zwei Werte auf den hohen Pegel zurück, nämlich $n.Kosten$ und $LB(n)$. Es sei $LB = min(LB(n) \mid n \in OPEN)$, wobei *OPEN* aus der hohen Ebene ist. LB ist eindeutig eine Untergrenze für die optimale Lösung des gesamten Problems (c^*). *FOCAL* in ECBS wird in Bezug auf LB und $n.Kosten$ wie folgt definiert:

$$FOCAL = \{n \mid n \in OPEN, n.kosten \leq LB \cdot w\} \tag{4.2}$$

Da LB eine Untergrenze für C^* ist, haben alle Knoten in *FOCAL* die Kosten, die innerhalb des w-fachen der optimalen Lösung liegen. Sobald eine Lösung gefunden ist, betragen die Kosten höchstens $w \cdot C^*$. Der Vorteil von ECBS gegenüber CBS besteht darin, dass die

niedrige Ebene und die hohe Ebene mehr Flexibilität erhalten. Darüber hinaus stimmen alle gültigen Lösungen immer mit mindestens einem der CT-Knoten in *OPEN* überein. Aufgrund der systematischen Suchen werden ECBS-Algorithmen schließlich eine Lösung finden, wenn eine solche existiert. Somit ist ECBS auch vollständig [54][55].

4.3 Bernsteinpolynome

Im mathematischen Bereich der numerischen Analyse sind Bernsteinpolynome Polynome in Bernstein-Form, d. h. eine lineare Kombination von Bernstein-Basispolynomen. Die Bernsteinpolynome haben ihren Ursprung in der Approximationstheorie. Mit ihrer Hilfe konnte ihr Entdecker, Sergei Natanovich Bernstein, im Jahr 1911 einen konstruktiven Beweis für den Approximationssatz von Karl Weierstraß angeben. Ende der 1950er Jahre gab es erste Versuche, auf Bernsteinpolynomen basierende Methoden im Design von Kurven und Flächen einzusetzen. Paul de Faget de Casteljau bei Citroën und Pierre Bézier bei Renault nutzten die Bernsteinpolynome bei ihrer Entwicklung von Bézierkurven und legten damit den Grundstein des heutigen Computer Aided Design (CAD). In dieser Arbeit werden die Bernsteinpolynome verwendet, um nichtkonvexe Beschränkungen in konvexe Beschränkungen umzuwandeln und die Trajektorien zu optimieren [56].

Für $n \in N_0$ heißen die reellen Polynome $B_{i,n} : \mathbb{R} \to \mathbb{R}, t \mapsto \binom{n}{i} t^i (1-t)^{n-i}, 0 \leqslant i \leqslant n$ die Bernsteinpolynome vom Grad n. Durch affine Transformation (Abbildung des Intervalls $[0,1]$ auf ein beliebiges Intervall $[a,b]$) erhält man die verallgemeinerten Bernsteinpolynome $B_{i,n}^{[a,b]} : \mathbb{R} \to \mathbb{R}, t \mapsto \frac{1}{(b-a)^n}\binom{n}{i}(t-a)^i(b-t)^{n-i}$. Dabei bezeichnet $\binom{n}{i} = \frac{n!}{i!(n-i)!}$ den Binomialkoeffizienten [57]. Das Bernsteinpolynom ist die lineare Kombination von Bernstein-Basispolynomen, und das Bernstein-Basispolynom vom Grad n ist wie folgt definiert:

$$B_n^i(t) = \binom{n}{i} \cdot t^i \cdot (1-t)^{n-i} \tag{4.3}$$

Das Polynom, das aus der Bernstein-Basis besteht, wird als Bézierkurve bezeichnet und wie folgt geschrieben:

$$B_j(t) = c_j^0 b_n^o(t) + c_j^1 b_n^1(t) + \cdots + c_j^n b_n^n(t) = \sum_{i=0}^{n} c_j^i b_n^i(t) \tag{4.4}$$

Dabei ist $\left[c_j^0, c_j^1, \cdots, c_j^n\right]$ bezeichnet als c_j, die Menge der Kontrollpunkte des j-ten Stücks der Bézierkurve. Die Bézierkurve unterscheidet sich vom mononomischen Basispolynom durch folgende Eigenschaften [58].

- Endpunkt-Interpolation. Die Bézierkurve beginnt immer am ersten Kontrollpunkt, endet am letzten Kontrollpunkt und verpasst niemals andere Kontrollpunkte.
- Festes Intervall. Die auf die Variable t parametrisierte Bézierkurve ist auf $t \in [0, 1]$.
- Konvexe Hülle. Die Bézierkurve $B(t)$ besteht aus einem Satz von Kontrollpunkten c_i, die vollständig innerhalb der durch alle diese Kontrollpunkte definierten konvexen Hülle begrenzt sind.
- Hodograph-Eigenschaft. Die Ableitungskurve $B^{(1)}(t)$ einer Bézierkurve $B(t)$ wird als Hodograph bezeichnet und ist immer noch eine Bézierkurve mit den definierten Kontrollpunkten $c^{(1)}(i) = n \cdot (c_{i+1} - c_i)$, wobei n der Grad ist.

Tatsächlich können für eine Bézierkurve die Kontrollpunkte als Gewichte der Basis und auch die Basis als Gewichte der Kontrollpunkte angesehen werden. Da die Bézierkurve in einem festen Intervall $[0, 1]$ definiert ist, ist es erforderlich, dass ein Skalierungsfaktor s für jedes Stück der Trajektorie eingefügt wird, um die Parameterzeit t auf eine willkürlich zugewiesene Zeit für dieses Segment zu skalieren. Dementsprechend kann die stückweise Bernstein-Basis des m-Segments in einer Dimension μ aus x, y, z wie folgt geschrieben werden.

$$f_\mu(t) = \begin{cases} s_1 \sum_{i=0}^{n} c_{\mu 1}^i b_n^i(\frac{t - T_0}{s_1}), & t \in [T_0, T_1] \\ s_2 \sum_{i=0}^{n} c_{\mu 2}^i b_n^i(\frac{t - T_1}{s_2}), & t \in [T_1, T_2] \\ \vdots & \vdots \\ s_m \sum_{i=0}^{n} c_{\mu m}^i b_n^i(\frac{t - T_{m-1}}{s_m}), & t \in [T_{m-1}, T_m] \end{cases} \tag{4.5}$$

Dabei ist c_{ji} der i-te Kontrollpunkt des j-ten Segments der Trajektorie. $T_1, T_2, \cdots, T_m$ sind die Endzeiten jedes Segments. Die Gesamtzeit beträgt $T = T_m - T_o$. $s_1, s_2, \cdots, s_m$ sind die Skalierungsfaktoren, die auf jedes Stück der Bézierkurve angewendet werden, um das Zeitintervall von $[0, 1]$ zur Zeit $[T_{i-1}, T_i]$ in einem Segment zugeordnet. In der Praxis wird durch Multiplizieren eines Skalierungsfaktors in der Position jedes Kurvenstücks eine bessere numerische Stabilität für das Optimierungsprogramm erzielt [59][60].

5 Methoden

In diesem Kapitel werden zwei Methoden zur Lösung des MAPF-Problems vorgestellt, nämlich eine statische Methode in drei Dimensionen und eine dynamische Methode in vier Dimensionen. Im Rahmen dieser Arbeit wird die Pfadplanung basierend auf den Dimensionen des kartesischen Koordinatensystems als statische Methode in drei Dimensionen definiert, und die Pfadplanung basierend auf den drei räumlichen Dimensionen und der zeitlichen Dimension wird als dynamische Methode in vier Dimensionen definiert. Für die statische Methode steht die Kollisionsfreiheit durch überschneidungsfreie Trajektorien im Zentrum. Die Pfade der Agenten dürfen sich nicht überschneiden, d. h. es ist keine Überlappung von Pfaden erlaubt. Mit der dynamischen Methode kann dieses Problem durch zeitliche Anpassung gelöst werden. Neben der Ebene der Pfadplanung der statischen Methode wird eine Ebene der autonomen Entscheidungsfindung jedes Agenten hinzugefügt. So können zwei Agenten A und B im Fall einer Pfadüberschneidung z. B. entscheiden, dass A wartet bis B diesen Punkt durchquert hat. Im Folgenden wird zunächst das mathematische Modell definiert, um das MAPF-Problem zu beschreiben. Danach wird der Implementierungsprozess beider Methoden im Detail vorgestellt.

5.1 Mathematische Modelldefinition

Wie in Kapitel 1 erklärt, wird ein Multi-Agent-System aus N Quadrotoren betrachtet. Es wird angenommen, dass die Quadrotor/Agenten die gleiche dynamische Grenze und verschiedene Größen mit dem Radius $r_1, \cdots, r_N$ haben. Als Voraussetzung sind Vorkenntnisse des freien Raums F und des Hindernisses O in der 3D-Belegungskarte angegeben. Startpunkt und Zielpunkt des i-ten Quadrotors werden als s_i, g_i zugewiesen. Es wurde gezeigt, dass die Quadrotordynamik unterschiedlich flach ist und die Trajektorie in stückweisen Polynomen mit flachen Ausgaben in der Zeit t dargestellt werden kann [61]. Somit kann die Trajektorie des i-ten Quadrotors $p_i(t)$ in stückweisen M-Segment-Polynomen dargestellt werden. Durch die zwei Methoden wird für jeden Agenten eine kontinuierliche, glatte Trajektorie $p_i(t)$ erzeugt, die Kollisionen mit Hindernissen und anderen Agenten verhindert. Die maximale Geschwindigkeit und Beschleunigung des i-ten Quadrotors sind v^i_{max} bzw. a^i_{max}.

5.1.1 Darstellung der Trajektorie

Aufgrund der unterschiedlichen Ebenheit der Quadrotordynamik ist bekannt, dass die Flugtrajektorie des Quadrotors in einer Polynomfunktion mit flachen Ausgängen in der Zeit t dargestellt werden kann [61]. Es ist jedoch schwierig, die Beschränkungen der Kollisionsvermeidung auf Standardpolynombasis zu handhaben. Aus diesem Grund werden die Trajektorien von Quadrotoren im Rahmen dieser Arbeit mit Bernsteinpolynome umgesetzt. Das Bernsteinpolynom ist die lineare Kombination von Bernstein-Basispolynomen, und das Bernstein-Basispolynom vom Grad n ist wie folgt definiert:

$$B_{k,n}(t) = \binom{n}{k} t^k (1-t)^{n-k} \tag{5.1}$$

wobei $t \in [0,1]$ und $k = 0, 1, \cdots, n$. Die Trajektorie des i-ten Quadrotors $p_i(t) \in \mathbb{R}_3$ kann als stückweise Bernsteinpolynom im M-Segment wie folgt dargestellt werden:

$$p^i(t) = \begin{cases} \sum_{k=0}^{n} c^i_{1,k} B_{k,n}(\tau_1) & t \in [T_0, T_1] \\ \sum_{k=0}^{n} c^i_{2,k} B_{k,n}(\tau_2) & t \in [T_1, T_2] \\ \vdots & \vdots \\ \sum_{k=0}^{n} c^i_{M,k} B_{k,n}(\tau_M) & t \in [T_{M-1}, T_M] \end{cases} \tag{5.2}$$

wobei $\tau_m = \frac{t - T_{m-1}}{T_m - T_{m-1}}$, $c^i_{m,k}$ ist der k-te Kontrollpunkt des m-ten Segments der Trajektorie des i-ten Quadrotors ist, und T_{m-1},T_m sind die Start- und Endzeit des m-ten Segments. Somit besteht der Vektor des Optimierungsproblems, c, aus allen Kontrollpunkten von $p_i(t)$ für $i = 1, \cdots, N$. Wie in Kapitel 4 geschrieben, ist ein Tupel (a_i, v, t) zur Beschreibung der Einschränkung definiert. Dabei bezeichnet a_i die einzelnen Agenten und v die Koordinaten der Position. D. h. zu jeden Position v gibt es drei Achsen x, y, z für die räumliche Darstellung. Das Zeichen t steht für den aktuellen Zeitpunkt.

5.1.2 Einschränkungen der Dynamik

Die Zielfunktion der Dynamik des Quadrotors wird wie folgt definiert (5.3), um das Integral der ϕ-ten Ableitung der Trajektorie zu minimieren:

$$J = \sum_{i=1}^{N} \int_{T_0}^{T_m} \left\| \frac{\mathrm{d}^\phi}{\mathrm{d}t^\phi} p^i(t) \right\|_2^2 dt = c^T Q c \tag{5.3}$$

Dabei ist Q die hessische Matrix der Zielfunktion. In dieser Arbeit wird $\phi = 3$ gesetzt, um den Ruck der Trajektorie zu minimieren, so dass die Eingabe nicht aggressiv für den Quadrotor ist [62].

Die Trajektorie muss die Start- und Zielpunkte verbinden und soll bis zur $\phi - 1$-ten Ableitung kontinuierlich sein. Außerdem dürfen die Geschwindigkeit und Beschleunigung des Quadrotors die maximale Geschwindigkeit v^i_{max} und Beschleunigung a^i_{max} nicht überschreiten. Diese Einschränkungen können als affine Gleichheits- bzw. Ungleichheitsbeschränkungen geschrieben werden:

$$A_{eq}c = b_{eq} \tag{5.4}$$

$$A_{dyn}c \leqslant b_{dyn} \tag{5.5}$$

5.1.3 Einschränkungen zur Vermeidung von Hindernissen

Zur Vermeidung von statischen Hindernissen wird das Hinderniskollisionsmodell des i-ten Quadrotors definiert (5.6). Das Hinderniskollisionsmodell bestimmt einen Kollisionsbereich zwischen einem Quadrotor und statischen Hindernissen, um einen sicheren Abstand zu gewährleisten.

$$C^i_{obs} = \left\{ p \in \mathbb{R}_3 \mid \|p\|_2^2 \leqslant (r_2^i) \right\} \tag{5.6}$$

Der i-te Quadrotor muss die nachstehende Bedingung erfüllen, um nicht mit den Hindernissen zu kollidieren:

$$p^i(t) \oplus C^i_{obs} \subset F, \quad t \in [T_0, T_M] \tag{5.7}$$

Dabei ist $\oplus$ die Minkowski-Summe. Die graphische Darstellung dazu ist das linke Diagramm in Abbildung 6.1.

5.1.4 Einschränkungen zur Vermeidung von Interkollisionen

Ein Kollisionsbereich zwischen dem i-ten und j-ten Agenten kann mit einem Interkollisionsmodell $C^{i,j}_{inter}$ ausgedrückt werden (5.8).

$$C^{i,j}_{inter} = \left\{ p \in \mathbb{R}_3 \mid p^T E p \leqslant \left(r^i + r^j\right)^2 \right\} \tag{5.8}$$

Dabei ist E die diagonale Matrix $dia(\left[1, 1, 1/(c_{dw})^2\right])$, und c_{dw} ein Koeffizient zur Beschreibung des Abwind-Effekts/Downwash-Effekts (siehe Abbildung 6.2) [63]. Der i-te

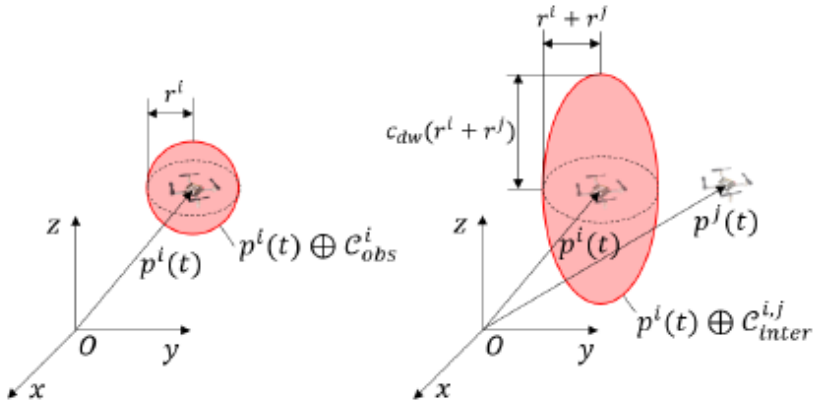


Abbildung 5.1: Hindernis-Kollisionsmodell und Inter-Kollisionsmodell

Agent kollidiert nicht mit dem j-ten Agenten, wenn die relative Trajektorie des j-ten Agenten in Bezug auf den i-ten Agenten, $p^{i,j}(t) = p^j(t) - p^i(t)$, die folgende Bedingung erfüllt(5.9). Abbildung 6.1 (rechts) ist eine graphische Darstellung des Interkollisionsmodells.

$$p^i(t) \cap C^{i,j}_{inter} = \varnothing, \quad t \in [T_0, T_M] \tag{5.9}$$

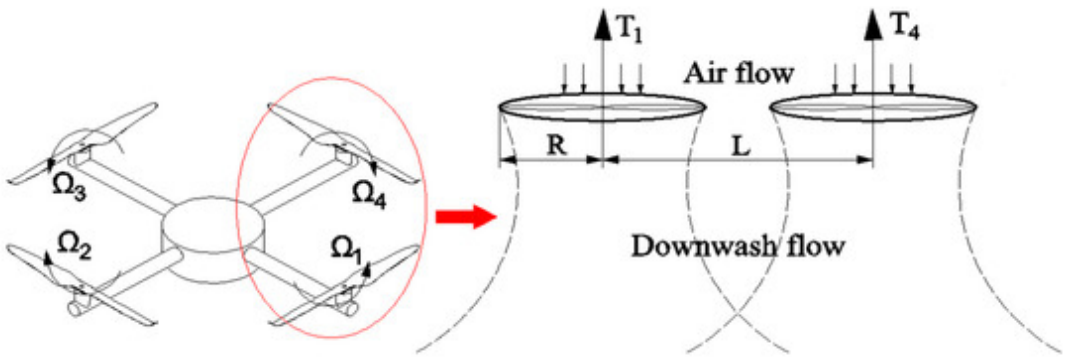


Abbildung 5.2: Abwind-Effekt/Downwash-Effekt

5.2 Statische Methode (ohne Zeitdimension)

Beim MAPF-Problem geht es darum, Kollisionen mit anderen Agenten sowie mit statischen Hindernissen zu vermeiden. Beim SAPF-Problem werden nur der einzelne Agent und statische Hindernisse betrachtet. Unter Verwendung des Pfadplanung-Algorithmus ist das SAPF-Problem viel einfacher als das MAPF-Problem zu lösen. Beim SAPF gibt

es keinen Einfluss dynamischer Hindernisse oder anderer Agenten, d. h. die Position der Hindernisse verändert sich im Zeitverlauf nicht. Die Lösungsansätze des SAPF-Problems konzentrieren sich auf die Methoden für nur die drei räumlichen Dimensionen, um Kollisionen mit statischen Hindernissen zu vermeiden.

Im Gegensatz dazu müssen die MAPF-Lösungsansätze dynamische Hindernisse berücksichtigen. Aus der Perspektive eines einzelnen Agenten sind die sich bewegenden anderen Agenten dynamische Hindernisse. Deswegen muss beim MAPF-Problem die Zeitdimension hinzugefügt werden, um dynamische Hindernisse zu vermeiden. Es ist bekannt, dass jede zusätzliche Dimension die Zeitkomplexität des Algorithmus exponentiell erhöht. Es gibt zwei Optimierungsmöglichkeiten bei der Lösung des MAPF-Problems, nämlich entweder die Anzahl der Dimensionen zu reduzieren oder die Algorithmuseffizienz zu erhöhen. Die statische Methode wird zur Verminderung der Zahl der Dimensionen verwendet. Statt der Echtzeittrajektorien werden die Pfade aller anderen Agenten als statische Hindernisse in der Karte gespeichert. Bei der Pfadplanung eins einzelnen Agenten werden dann nur statische Hindernisse, und dazu zählen Hindernisse in der Umwelt ebenso wie die Pfade der anderen Agenten. Somit wird das MAPF-Problem in ein SAPF-Problem umgesetzt. Als Nächstes wird diese statische Methode im Detail betrachtet, beginnend mit ihrer Architektur. Danach werden die vier Module der Methode vorgestellt, nämlich Kartenkonstruktion, FCL Kollisionserkennung, Pfadplanung und Trajektoriengenerierung.

5.2.1 Architektur statischer Methode

Abbildung 6.16 stellt die Architektur der statischen Methode dar. Die rechteckigen Blöcke repräsentieren die verschiedenen Module, und die Pfeile geben die Richtung des Informationsflusses an. Input und Output sind in den ovalen Feldern beschrieben. Zuerst wird die Umgebung in OctoMap modelliert, um eine 3D-Belegungsgitter-Map zu erhalten. Die Map-Information wird durch ROS-Knoten zum FCL-Modul gesendet. Zur Vereinfachung der Kollisionserkennung werden Hindernisse als Kuben, die Drohne als Kugel, und der Pfad als Zylinder modelliert. Dann wird der Pfad jeder Drohne durch den informierten RRT*-Algorithmus sequenziell geplant. Wird ein gültiger Pfad errechnet, so wird dieser als FCL-Kollisionsmodell gespeichert. Das Pfadplanungmodul generiert diskrete Pfade, wie die gestrichelte Linie gezeigt. Dabei sind s^1, s^2 die Startpunkte und g^1, g^2 die Zielpunkte für Agenten 1 und 2. Im letzten Modul werden die Pfade unter Berücksichtigung der Kinematik optimiert. Schließlich wird eine ausführbare Trajektorie für jede Drohne generiert, angezeigt durch die durchgezogene Linie.

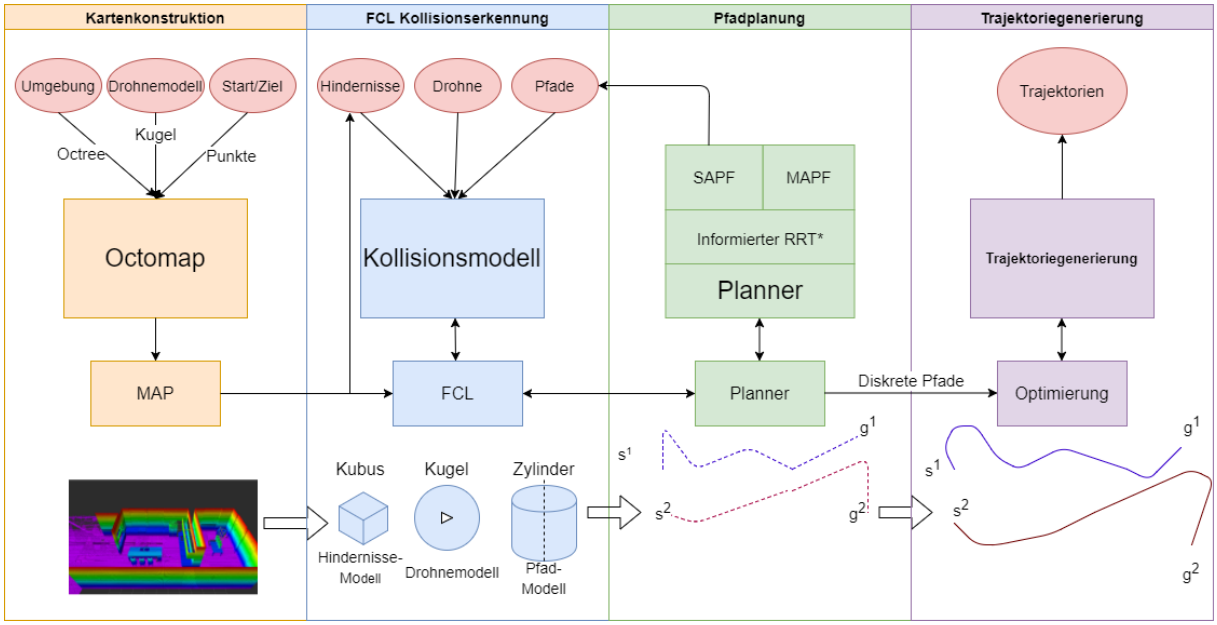


Abbildung 5.3: Architektur statischer Methode

5.2.2 Kartenkonstruktion

Als Voraussetzung für die Pfadplanung wird das Kartenkonstruktionsmodul verwendet, um die Daten einer bekannten Umgebung zu modellieren und zu speichern. Im Rahmen dieser Arbeit geht es primär um die Pfadplanung für Produktions- und Transportinfrastruktur. Häufige Anwendungsbereiche sind Lagerhäuser und Werkstätten oder Industrieparks. Für diese Arbeit werden ein Lagerhaus und ein Industriepark durch OctoMap modelliert. Wie das Kapitel 3.4 beschrieben, basiert OctoMap auf Octree und einer probabilistischen Belegungsschätzung. Es repräsentiert explizit nicht nur belegten Raum, sondern auch freie und unbekannte Bereiche. Die Wahrscheinlichkeit $P(n)$ beschreibt die Belegungsschätzung des Knotens n. Wenn der Knoten frei ist, gilt $P(n) = 0$. Wenn der Knoten belegt ist, gilt $P(n) = 1$. Je größer die Wahrscheinlichkeit ist, desto höher ist die Wahrscheinlichkeit dafür, dass dieser Knoten belegt ist. Ist $P(n) = 0,5$, so ist der Knoten unbekannt. Um das Kartenmodell weiter zu vereinfachen, gibt es im Rahmen dieser Untersuchung jedoch keine unbekannten Bereiche, so dass jeder Knoten nur zwei Zustände haben kann, nämlich frei oder belegt. $P(n)$ kann nur die Werte 0 und 1 annehmen. Die Wahrscheinlichkeiten ändern sich nicht mit der Zeit. Wenn der Pfadplanungsalgorithmus jeden Knoten in der Karte durchläuft, muss er nur eine Logik ausführen, um zu bestimmen, ob jeder Knoten belegt ist, es gibt keine Unsicherheit und die Berechnungskomplexität wird verringert. Algorithmus 9 verdeutlicht den Prozess der Kartenkonstruktion.

Der gesamte Raum θ besteht aus belegtem Raum ξ und freiem Raum ϱ.

$$\theta = \xi \cup \varrho, \quad \xi \cap \varrho = \varnothing \tag{5.10}$$

Zuerst wird das Octree mit einer vorgegebenen Auflösung initialisiert (Zeile 1). Dann werden alle belegte und freie Knoten im Octree registriert und aktualisiert (Zeilen 2-7). Schließlich wird eine 3D-Belegungskarte ε ausgegeben, und die Karte wird in eine Datei mit .bt als Suffix geschrieben. Die .bt-Datei kann durch den OctoMap-Server mit dem ROS-Framework verbunden sein. Die 3D-Belegungskarte wird über die ROS-Knoten in Form einer ROS-Nachricht an den Planer gesendet. Die Abbildung zeigt OctoMap-Visualisierung mit Rviz.

Algorithm 8: Kartenkonstruktion

Input: belegter Raum $\xi = \{\xi_1, \cdots, \xi_N\}$, freier Raum $\varrho = \{\varrho_1, \cdots, \varrho_M\}$, gesamter Raum θ

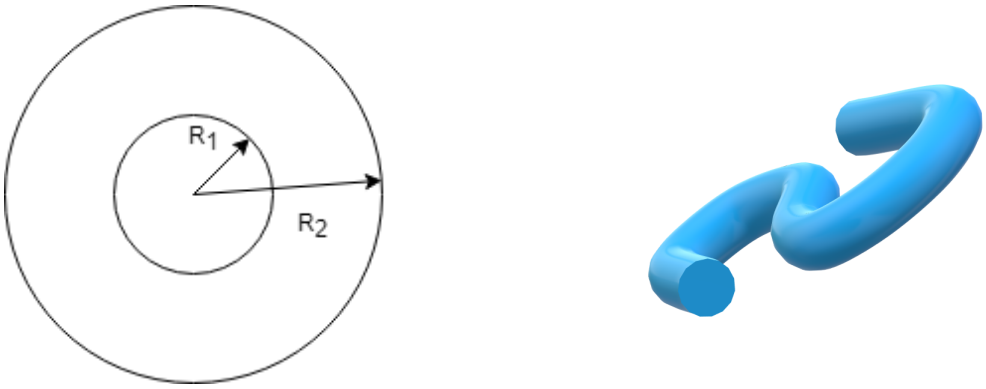
Output: 3D-Belegungskarte ε

```
1 tree ← Octree(resolution) ;
2 for i ← 1 to N do
3 |   tree.updateNode(ξ_i, true) ;
4 end
5 for i ← 1 to M do
6 |   tree.updateNode(ϱ_i, false) ;
7 end
8 tree.writeBinary(ε ) ;
```

5.2.3 FCL-Kollisionserkennung

Das FCL-Kollisionsmodell behandelt die Darstellung von Objekten in einer hierarchischen Datenstruktur, damit Kollisions- und Näherungsabfragen effizient ausgeführt werden können. In statischen Methoden werden drei FCL-Kollisionsmodelle dargestellt, nämlich das Hindernismodell, das Drohnenmodell und das Pfadmodell. In der Kartenkonstruktion wird die Umgebung in einer 3D-Belegungskarte modelliert, und die statischen Hindernisse werden im Octree gespeichert. Zur Vereinfachung der Kollisionserkennung werden die besetzten Räume, $P(n) = 1$, zwischen Pfad und Umgebung als Hindernismodelle beschrieben. Das Octree in OctoMap wird zu einem FCL-Octree konvertiert. Die Drohne wird im FCL als Kugel mit dem Radius R modelliert, dabei steht R für die Summe aus dem Radius der tatsächlichen Geometrie R_1 der Drohne und dem Sicherheitsabstand R_2. Der Sicherheitsabstand ist abhängig vom Aerodynamikfaktor und dem Antrieb der

Drohne, z.B. Windkraft oder Motor. Der Pfad wird als Zylinder mit demselben Radius R modelliert, der auf dem Pfad zentriert ist.



a) Drohne-Modell. **b)** Pfad-Modell

Abbildung 5.4: Kollisionsmodell

Die Kollisionserkennung ist im Flussdiagramm in Abbildung 6.17 dargestellt. Das Kollisionsmodell ist die Geometrie des Kollisionsobjekts, und der Zustand des Modells beschreibt die Kinematik des Kollisionsobjekts. Wenn zwei Kollisionsobjekte mit Kollisionsmodellen und Zuständen initialisiert sind, berechnet der FCL Manager die Überlappung der Objekte mit dem Hüllkörperalgorithmus, um festzustellen, ob eine Kollision vorliegt. Schließlich wird ein Kollisionsergebnis zurückgegeben. Durch logische Beurteilung kann der Algorithmus entscheiden, ob der aktuelle Pfad kollidiert, um zu bestimmen, ob der aktuelle Pfad erweitert oder ein neuer Pfad neu generiert werden soll.

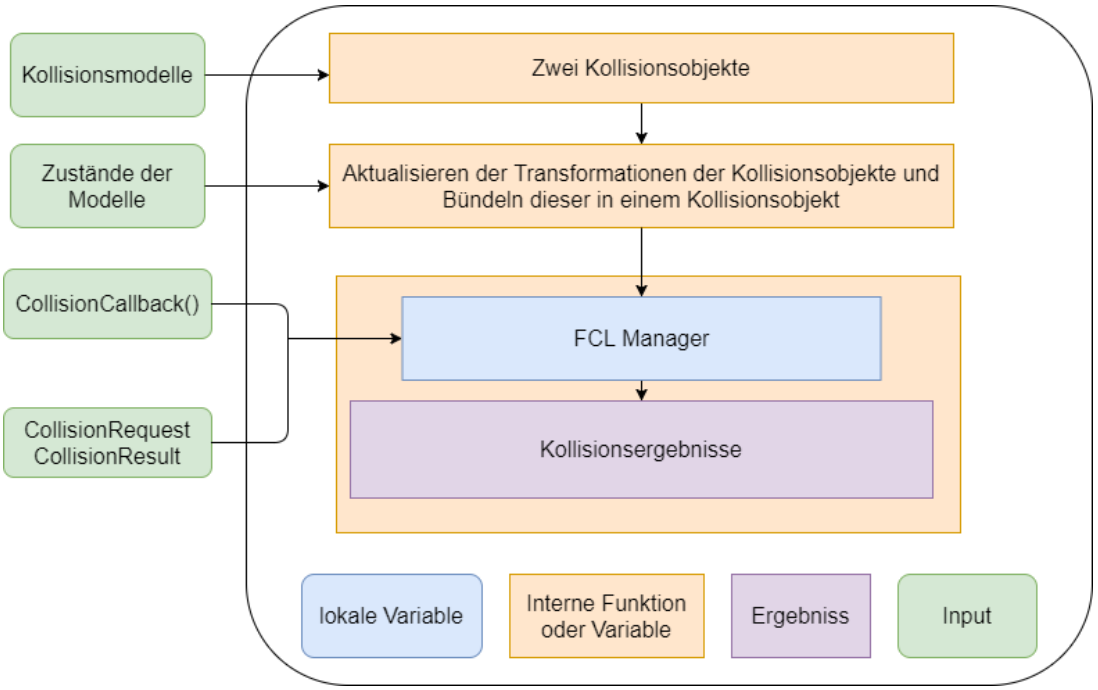


Abbildung 5.5: Architektur der FCL

5.2.4 Pfadplanung

Bei der SAPF-Pfadplanung geht um die Vermeidung statischer Hindernisse durch einen einzelnen Agenten. Der informierte RRT*-Algorithmus wird verwendet, um einen kollisionsfreien Pfad in einer vorgegebenen Umgebung zu generieren. Im Unterschied dazu beschäftigt sich die MAPF mit der Vermeidung von Kollisionen zwischen mehreren Agenten. Die zur SAPF-Problemlösung generierten Pfade werden als Pfadkollisionsmodell im Octree gespeichert, damit in der sequenziellen MAPF die Pfade anderer Agenten vermieden werden können.

Algorithmus 9 dient der Pfadplanung in der statischen Methode für Agenten i im Rahmen dieser Arbeit. Als Input sind der Startpunkt s^i, der Zielpunkt g^i, die 3D-Belegungskarte ε und die maximale Planungszeit T_{max} vorgegeben. Als Output generiert der Algorithmus eine Reihe von Wegpunkten $w_1^i, \cdots, w_n^i$ für Agenten i, die Startpunkt und Zielpunkt diskret verbinden, so dass gilt: $w_1^i = s^i$ und $w_n^i = g^i$. Außerdem bilden die Wegpunkte einen kollisionsfreien Pfad. Das bedeutet, dass keine benachbarten Punkte, die durch eine gerade Linie verbunden sind, mit Hindernissen kollidieren. Zuerst werden die FCL-Kollisionsmodelle initialisiert(Zeile 1). Dann wird der informierte RRT* verwendet, um einen Pfad von s^i nach g^i zu finden. Die Funktion *FCL.collide()* beurteilt, ob der Pfad mit die Hindernissen kollidiert (Zeile 4). Ist der Pfad kollisionsfrei, wird er in *PathOctree* gespeichert. Am Ende gibt der Algorithmus die Wegpunkte w^i und das Octree τ^i zurück.

Algorithm 9: SAPF statischer Methode

Input: Startpunkt s^i, Zielpunkt g^i für Agenten $i \in \{1, \cdots, N\}$, 3D-Belegungskarte ε, maximale Planungszeit T_{max}

Output: Wegpunkte $w^i = \left\{w_1^i, \cdots, w_n^i\right\}$ für Agenten i, τ^i

```
1 MapOctree ζ ← ε, PathOctree τ^i ← ∅, FCL.Drohne ← FCL.Sphere(R),
   runningTime t ;
2 for t ≤ T_max do
3 |   w^i ←InformedRRT*(s^i, g^i);
4 |   if !FCL.collide(w^i, ζ) then
5 |   |   τ^i ← w^i;
6 |   |   break;
7 |   end
8 end
9 return τ^i, w^i;
```

Algorithmus 10 beschreibt die Darstellung der Pfadplanung für mehrere Agenten. Es gibt insgesamt i Agenten, und für jeden Agenten gibt es einen Startpunkt s^i und einen

Zielpunkt g^i. Die sequenzielle Pfadplanung geschieht folgendermaßen: Für den ersten Agenten gibt es bei der Planung keinen anderen Pfad (Zeile 5). Für nachfolgende Agenten muss der zuvor geplante Pfad berücksichtigt werden (Zeilen 7-11). Daher ist eine Kollisionsprüfung zwischen dem Pfad des i-ten Agenten mit den vorherigen $i-1$ Pfaden durchzuführen. Tritt eine Kollision auf, muss der Pfad des i-ten Agenten neu geplant werden (Zeile 10). Wenn die Pfadplanung für alle i Agenten fertig ist, gibt der Algorithmus die Wegpunkte $W = \left\{w^1, \cdots, w^N\right\}$ zurück. Da jeder Pfad die Kollisionsprüfung für alle anderen Pfade durchführt, gibt es keine Pfadkollisionen. Schließlich werden die überschneidungsfreien Pfade generiert, die aus diskreten Wegpunkten bestehen.

Algorithm 10: MAPF statischer Methode

Input: Startpunkt s^i, Zielpunkt g^i für Agenten $i \in \{1, \cdots, N\}$, 3D-Belegungskarte ε, maximale Planungszeit t_{max}

Output: Wegpunkte $W = \left\{w^1, \cdots, w^N\right\}$

```
1  runningTime t;
2  for t ≤ t_max do
3      for i ← 1 to N do
4          if i = 1 then
5              SAPF(s^i, g^i);
6          else
7              SAPF(s^i, g^i);
8              for j ← (i − 1) to 1 do
9                  while FCL.collide(w^i, τ^j) do
10                     SAPF(s^i, g^i);
11                     j ← (i − 1);
12                 end
13             end
14         end
15     end
16 end
17 return W = {w^1, ···, w^N};
```

5.2.5 Trajektoriengenerierung

Nach der Pfadplanung werden aus diskreten Wegpunkten überschneidungsfreie Pfade generiert. Das bedeutet, dass die Pfade nicht kontinuierlich sind und der Abstand zwischen Zuständen im Pfad sehr groß sein kann. Außerdem berücksichtigen die Pfade keine dynamischen oder kinematischen Einschränkungen. Dies bedeutet, dass es zwischen den

Wegpunkten scharfe Kurven geben kann, wie Abbildung 6.20 zeigt. Deshalb ist es notwendig, dass die Pfade durch Interpolation und Glättung optimiert werden. Hier soll dazu die B-Spline-Funktion verwendet werden, die für den vorliegenden Fall vorteilhafte Eigenschaften besitzt, nämlich die lokale Steuerung und die konvexe Hülleneigenschaft, um eine ausführbare Trajektorie zu generieren [64] [65] [66].

Bei $n+1$ Kontrollpunkten $p_0, p_1, \cdots, p_n$ und Knotenvektoren $t_0, t_1, \cdots, t_m$ ist die B-Spline-Kurve $s(t)$ vom Grad k wie folgt definiert:

$$s(t) = \sum_{i=0}^{n} p_i N_{i,k}(t) \tag{5.11}$$

wobei $N_{i,k}(t)$ die B-Spline-Mischfunktion des Grades k ist, die rekursiv wie folgt ausgewertet werden kann:

$$N_{i,0}(t) = \begin{cases} 1 & wenn\ t_i < t < t_{i+1} \\ 0 & sonst \end{cases} \tag{5.12}$$

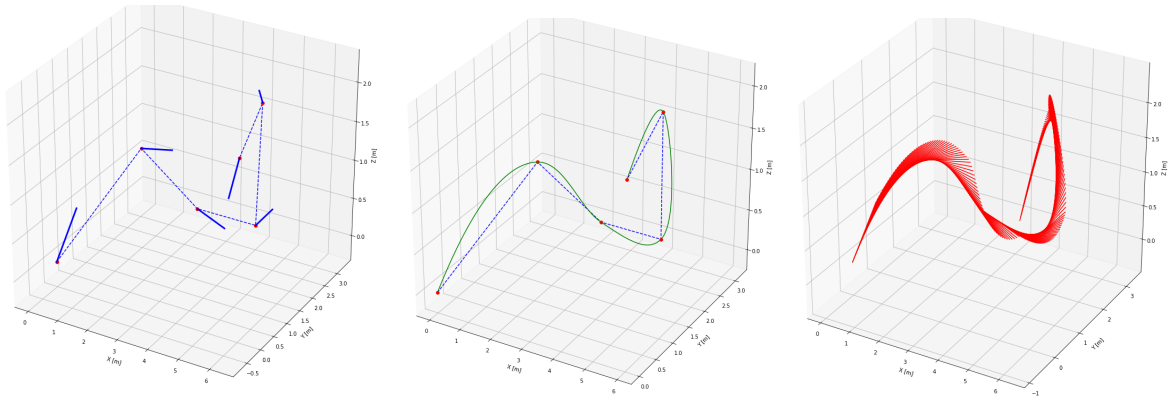
$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k} - t_i} N_{i,k-1}(t) + \frac{t_{i+k+1} - t}{t_{i+k+1} - t_{i+1}} N_{i+1,k-1}(t) \tag{5.13}$$

Die Gesamtzahl der Knoten sollte $m + 1 = n + k + 2$ erfüllen. Der einheitliche B-Spline ist eine spezielle Art von B-Spline, dessen Knoten gleichmäßig verteilt sind. Angenommen, der Knotenvektor wird mit der Äquidistanz $\triangle t$ getrennt. Das halboffene Intervall $[t_i, t_{i+1})$ wird als i-te Knotenspanne bezeichnet. Jede Knotenspanne wird mit $u = \frac{t-t_i}{\triangle t}$ normalisiert und für die i-te Knotenspanne sind nur $k+1$ Mischfunktionen ungleich Null, entsprechend $k+1$ Kontrollpunkten $p_{i-k}, \cdots, p_k$. Die $k+1$ Kontrollpunkte werden als Koordinatenmatrix einer Kontrollpunktspanne gestapelt $P_{i-k} := [p_{i-k} p_{i-k+1} \cdots p_i]^T \in \mathbb{R}^{(k+1)\times 3}$. Sei $j = i - k$, können die Position und die Ableitungen der B-Spline-Kurve, die der j-ten Kontrollpunktspanne entsprechen, wie folgt bewertet werden:

$$\frac{ds_j(u)}{d^l(u)} = \frac{1}{(\triangle t)^l} \frac{db^T}{d^l(u)} M_k P_j \tag{5.14}$$

Dabei bezeichnet l die Ordnung der Ableitung ($l = 0$ ist die Position), $b = \left[1\, u\, u^2 \cdots u^k\right]^T \in \mathbb{R}^{k+1}$ den Basisvektor, und $M_k = (m_{i,j} \in \mathbb{R}^{(k+1)\times(k+1)})$ die Mischmatrix, wobei $m_{i,j} = \frac{1}{k!}\binom{k}{k-i} \sum_{s=j}^{k} (-1)^{s-j} \binom{k+1}{s-j} (k-s)^{k-i}$. Gemäß 5.14 kann die Bewertung der Ableitungen der B-Spline-Kurve durch eine lineare Matrixmultiplikation in Bezug auf die Kontrollpunktspanne P_j ausgedrückt werden. Im Rahmen dieser Methode wird ein quintischer einheitlicher B-Spline ($k = 5$) verwendet, um die Kontinuität sicherzustellen [67] [68] [69] [70].

Die Diagramme in Abbildungen 5.6 zeigen den Optimierungsprozess durch B-spline. Zuerst werden die diskreten Wegpunkte als B-Spline-Kontrollpunkte bezeichnet. Entsprechend der Parametereinstellung werden einige Punkte zwischen jeweils zwei benachbarten Wegpunkten interpoliert, um eine glatte Trajektorie zu formulieren (Abbildung 5.6 b)). Dann wird die Steigung jedes Punktes auf der Kurve berechnet, um die kinodynamische Machbarkeit der Trajektorie sicherzustellen. Aus den Diagrammen ist ersichtlich, dass die endgültige Trajektorie durch die Kontrollpunkte verläuft. Die hinzugefügten Punkte bergen jedoch ein potenzielles Kollisionsrisiko. Genau genommen sollte für jeden interpolierten Punkt eine Kollisionserkennung durchgeführt werden. Um die Interpolation zu vereinfachen und die Effizienz zu verbessern, wird der Kollisionssicherheitsabstand so eingestellt, dass keine für die Interpolation ausgewählten Punkte diesen Abstand überschreiten. Schließlich wird eine kollisionsfreie, kinodynamische und glatte Trajektorie generiert.



a) Diskrete Wegpunkte nach Pfadplanung

b) Optimierung durch B-Spline

c) Steigung der Kurve

Abbildung 5.6: Trajektorie Optimierung durch B-Spline

5.3 Dynamische Methode (mit Zeitdimension)

Das Hauptaugenmerk der statischen Methode ist darauf gerichtet, durch die MAPF-Planung jede Überschneidung von Pfaden auszuschließen. Im Gegensatz dazu ist eine Pfadüberschneidung mit der dynamischen Methode möglich, da hier neben den drei räumlichen Dimensionen die zeitliche Dimension zur Verfügung steht, um Kollisionen zu vermeiden. In diesem 4D-Scenario geht es darum, jede Positionsüberlappung zwischen Agenten zu einem bestimmten Zeitpunkt zu verhindern. Wie bereits erwähnt, stellt die zusätzliche Dimension extrem hohe Anforderungen an die Recheneffizienz des Algorithmus. In dynamischen 4D-Methode werden der sichere Flugkorridor (SFC), der relativ sichere Flugkorridor (RSFC) zur Kollisionsvermeidung und die Dummy-Agenten zur Effizienzoptimierung verwendet. Im Folgenden wird die dynamische Methode im Detail be-

trachtet, beginnend mit ihrer Architektur. Dann werden die Kernmodule der Methode vorgestellt, nämlich die initiale Planung, der sichere Flugkorridor (SFC), der relativ sichere Flugkorridor (RSFC), Dummy-Agenten und die Zeitzuweisung. Die sogenannten Module wie Kartenkonstruktion, Trajektoriengenerierung und Trajektorienverfolgung gleichen denen der statischen Methode und werden hier nicht erneut betrachtet.

5.3.1 Architektur der dynamischen Methode

Die Architektur der dynamischen Methode ist in Abbildung 6.15 dargestellt. Die blauen Blöcke repräsentieren die Module, und die orangen Blöcke sind die Algorithmen oder Ergebnisse der Module. Die Pfeile geben die Richtung des Informationsflusses an. Zuerst wird aus dem Kartenkonstruktionsmodul die OctoMap erstellt, um die Umgebung zu modellieren. Anhand der OctoMap generiert der MAPF-Algorithmus ECBS eine initiale Trajektorie für jeden Agenten. Basierend auf dieser initialen Trajektorie werden zwei weitere Flugkorridore generiert, nämlich der sichere Flugkorridor und der relativ sichere Flugkorridor, um Kollisionen zwischen Hindernissen und Agenten zu vermeiden. Nach der Kombination beider Korridore wird ein absolut kollisionsfreier Korridor hergestellt. Anschließend werden virtuelle Agenten im Dummy-Agenten-Modul eingeführt, um die Komplexität der Optimierung zu verringern. Danach werden die Zeitsegmente für die Trajektorie im Zeitzuweisungsmodul verteilt. Am Ende entstehen realisierbare Trajektorien mit zeitlicher Anpassung.

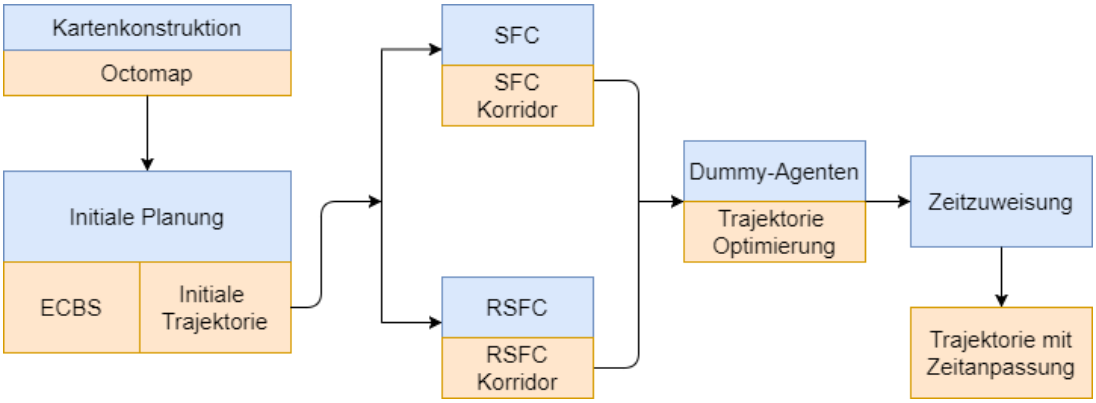


Abbildung 5.7: Architektur dynamischer Methode

5.3.2 Initiale Planung

Den Planungsprozess für einen einzelnen Quadrotor haben viele Forscher in die initiale Trajektorienplanung und die Verbesserung der Trajektorie unterteilt, und diese zweistufige Methode soll hier auf mehrere Agenten angewendet werden [22][71]. In Anlehnung daran

wird die initiale Trajektorie der dynamischen Methode zuna¨chst mithilfe eines graphbasierten MAPF-Algorithmus geplant. Die initiale Flugbahn des i-ten Quadrotors p^i_{init} wird als Reihe von Wegpunkten definiert, die Start- und Zielposition in einem Diagramm verbinden. In der MAPF ist die Kostenfunktion die Summe der Länge aller Flugbahnen. Es gibt viele MAPF-Algorithmen, wie HCA* [72], M* [73], CBS [11]. Unter diesen wird der Enhanced CBS (ECBS) [54] als diskreter Planalgorithmus für die initiale Flugbahn gewählt, da er in kurzer Zeit eine suboptimale Lösung unter der Grenze der Lösungskosten finden kann. Mit anderen Worten, wird es garantiert, dass die Kosten der Trajektorie niedriger als das c_w-fache der optimalen Kosten sind, wobei c_w ein benutzerdefinierter Begrenzungsfaktor ist[54]. Um das graphbasierte ECBS zu verwenden, übersetzt der diskrete Planer die 3D-Belegungskarte OctoMap in eine 3D-Gitterkarte. Nach der Übersetzung berechnet ECBS eine diskrete Trajektorie, die Start- und Zielpunkte verbindet. Wenn die Start- und Zielpunkte sich nicht auf der 3D-Gitterkarte befinden, werden die nächsten Gitterpunkte statt der ursprünglichen Punkte verwendet.

Eine initiale Trajektorie des i-ten Quadrotors, $p^i_{init} = \pi^i = \left\{\pi^i_0, \cdots, \pi^i_M\right\}$, ist definiert als Pfad, der die folgenden Bedingungen (5.15) für alle $m = 0, \cdots, M$ und $i \neq j$ erfüllt:

$$\pi^i_0 = s^i, \quad \pi^i_M = g^i \tag{5.15}$$

$$\left\langle \pi^i_{m-1}, \pi^i_M \right\rangle \oplus C^i_{obs} \subset F \tag{5.16}$$

$$\left\langle \pi^{i,j}_{m-1}, \pi^{i,j}_m \right\rangle \cap C^{i,j}_{inter} = \varnothing \tag{5.17}$$

Wobei $\left\langle \pi^i_{m-1}, \pi^i_M \right\rangle = \left\{\alpha\pi^i_{m-1} + (1-\alpha)\pi^i_m \mid 0 \leq \alpha \leq 1\right\}$ ein Liniensegment zwischen den Wegpunkten π^i_{m-1} und π^i_m ist, und $\pi^{i,j}_m = \pi^j_m - \pi^i_m$. (11) zeigt, dass die initiale Flugbahn frei von Hindernissen ist, und (12) bedeutet, dass die Agenten nicht mit anderen Agenten kollidieren, wenn sich alle Agenten mit konstanter Geschwindigkeit entlang ihrer initialen

Trajektorien bewegen.

Algorithm 11: Trajektorienplanung

Input: Startpunkt s^i, Zielpunkt g^i für Agenten $i \in \{1, \cdots, N\}$, 3D-Belegungskarte ε

Output: Gesamtflugzeit T, Trajektorie $p^i(t)$ für Agenten $i \in \{1, \cdots, N\}$, $t \in [0, T]$

$\pi = (\pi^1, , \pi^N) \leftarrow$ planInitialTraj$((s^{\forall i}, g^{\forall i}), \varepsilon)$;
for $i \leftarrow 1$ *to* N **do**
  $SFC^i = (SFC_0^i, \cdots, SFC_M^i) \leftarrow$ buildSFC(π^i, ε);
  for $j \leftarrow i + 1$ *to* N **do**
    $RSFC^{i,j} = (RSFC_0^{i,j}, \cdots, RSFC_M^{i,j}) \leftarrow$ buildRSFCπ^i, π^j;
  end
end
$p^0(t), \cdots, p^N(t) \leftarrow$ trajOpt$(\pi, SFC^{\forall i}, RSFC^{\forall i,j>i})$;
$T, p^0(t), \cdots, p^N(t) \leftarrow$ timeScale$(p^0(t), \cdots, p^N(t))$;
return $T, p^o(t), \cdots, p^N(t)$;

5.3.3 Der sichere Flugkorridor (SFC)

Der sichere Flugkorridor (SFC) wird bei der Pfadplanung benutzt, um den freien Raum in einer Karte zu modellieren [74]. Der SFC besteht aus verbunden konvexen Mengen und kann als lineare Ungleichungen zur Vermeidung von Hindernissen bei der quadratischen Programmierung (QP) dargestellt werden [75] [74] [59] [76]. Der SFC des i-ten Quadrotors, $SFC_1^i, \cdots, SFC_M^i$, ist als Sammlung konvexer Mengen definiert, die nicht mit Hindernissen kollidieren und sequenziell verbunden sind.

$$SFC_m^i \oplus C_{obs}^i \in F, \quad m = 1, \cdots, M \tag{5.18}$$

$$SFC_m^i \cap SFC_{m+1}^i \neq \varnothing, \quad m = 1, \cdots, M-1 \tag{5.19}$$

C_{obs}^i ist das Hinderniskollisionsmodell für den i-ten Quadrotor, das als Kugel mit dem Radius r^i definiert ist, der den Sicherheitsabstand zwischen einem Hindernis und einem Quadrotor darstellt. Die Trajektorie des i-ten Quadrotors ist frei von Hindernissen, wenn für beliebiges $t \in [0, T]$ existiert $m \in \{1, \cdots, M\}$, so dass $p^i(t) \in SFC_m^i$, wobei T die Gesamtflugzeit ist. Zuerst wird der SFC nach der Achsensuchmethode mit einer vordefinierten Größe an jedem Wegpunkt der initialen Trajektorie konstruiert. Mit Ausnahme der Start- und Zielpunkte erweitern die Wegpunkte sich in der vorherigen Wegpunktrichtung, um zwei konvexe Mengen zu verbinden. Alle Wegpunkte außer Start- und Ziel-

punkten werden auf der 3D-Gitterkarte ausgerichtet, so dass die Bedingung 5.14 erfüllt ist. Danach erweitert sich jeder Korridor in alle anderen Achsenrichtungen, bis er einen maximalen freien Raum hat. Schließlich werden die duplizierten Korridore gelöscht.

Algorithm 12: Erstellung des SFC (buildSFC)

Input: initiale Trajektorie π^i, 3D-Belegungskarte ε

Output: SFC $SFC^i = (SFC_1^i, \cdots, SFC_M^i)$

```
1  D ← {±x, ±y, ±z};
2  for m ← 1 to M do
3  |   SFC_m^i ← ⟨π_{m-1}^i, π_m^i⟩;
4  |   while D is not empty do
5  |   |   for μ in D do
6  |   |   |   if SFC_m^i cannot expand to direction μ then
7  |   |   |   |   D ← D \ μ;
8  |   |   |   end
9  |   |   end
10 |   |   expand SFC_m^i to all directions in D;
11 |   end
12 end
```

Algorithmus 12 zeigt die Erstellung des SFC. Der SFC wird auf $\left\langle \pi_{m-1}^i, \pi_m^i \right\rangle$ initialisiert, um die Bedingung() zu erfüllen (Zeile 3). Für alle Richtungen wird geprüft, ob der SFC erweiterbar ist (Zeilen 5-9), und eine Länge der Erweiterung ist vorgegeben (Zeile 10). Dieser Algorithmus gibt konvexe Mengen zurück, die der Definition von SFC entsprechen.

5.3.4 Der relativ sichere Flugkorridor(RSFC)

Im Vergleich zum SFC wird der relativ sichere Flugkorridor(RSFC) vorgestellt, um einen freien Raum für Ausweichmanöver zwischen zwei Agenten zu modellieren. Unter Verwendung der Eigenschaft des Bernsteinpolynoms, wandelt der RSFC die nichtkonvexen Beschränkungen in lineare Beschränkungen um. Somit kann dieses Verfahren eine stückweise Polynomtrajektorie optimieren, indem QP nur einmal verwendet wird, und es garantiert, dass eine praktikable Lösung von QP keine Kollision und keinen Deadlock verursacht. Die RSFC zwischen dem i-ten und dem j-ten Agenten sind als konvexe Mengen wie folgt definiert, $RSFC_1^{i,j}, \cdots, RSFC_M^{i,j}$, die nicht in den Kollisionsbereich zwischen dem i-ten

und dem j-ten Agent eindringen und sequenziell verbunden sind.

$$RSFC_m^{i,j} \cap C_{inter}^{i,j} = \varnothing, \quad m = 1, \cdots, M \tag{5.20}$$

$$RSFC_m^{i,j} \oplus RSFC_{m+1}^{i,j} \neq \varnothing, \quad m = 1, \cdots, M-1 \tag{5.21}$$

$C_{inter}^{i,j}$ ist ein Interkollisionsmodell, das rechteckiges Parallel-flach zum Körperrahmen des i-ten Quadrotors ausgerichtet ist, zwischen dem i-ten und dem j-ten Quadrotor. Es ist zu beachten, dass $C_{inter}^{i,j}$ für jedes Agentenpaar variieren kann, was bedeutet, dass es unterschiedliche Größen von Quadrotoren handhaben kann. In dieser Arbeit werden die Länge und Breite von $C_{inter}^{i,j}$ als $2(r^i + r^j)$ und eine Höhe als $2c_{dw}(r^i + r^j)$ zugewiesen, um den Downwash-Effekt zu berücksichtigen, wobei c_{dw} der Downwash-Koeffizient ist. Die Trajektorie des j-ten Quadrotors kollidiert nicht mit dem i-ten Quadrotor, wenn für beliebiges $t \in [0, T]$ existiert $m \in \{1, \cdots, M\}$, so dass $(p^j(t) - p^i(t)) \in RSFC_m^{i,j}$.

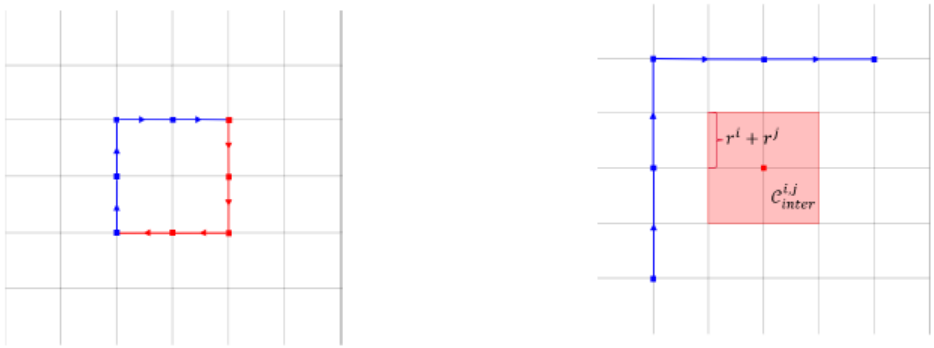
Die Konstruktion des RSFC ist in den Abbildungen 6.6 und 6.9 beschrieben, zur Vereinfachung dargestellt in einem 2D-Raum. Zunächst werden die initialen Trajektorien in relative Trajektorien für jedes Agentenpaar konvertiert. Die relative Trajektorie des i-ten und des j-ten Quadrotors $\pi^{i,j}$ kann durch Subtrahieren entsprechender Wegpunkte von zwei initiale Trajektorien erhalten werden, wie in Abbildung 6.6 dargestellt. Es gibt sechs RSFC-Kandidaten in Richtung $\pm x, \pm y, \pm z$, um die Anzahl der Entscheidungsvariablen im Optimierungsschritt zu reduzieren, und jeder RSFC-Kandidat $RSFC_\mu$ ist wie folgt definiert:

$$RSFC_\mu = \begin{cases} \left\{p \mid p \cdot n_\mu > r^i + r^j\right\}, & \mu = \pm x, \pm y \\ \left\{p \mid p \cdot n_\mu > c_{dw}(r^i + r^j)\right\}, & \mu = \pm z \end{cases} \tag{5.22}$$

n_μ ist ein Einheitsvektor in Richtung $\mu \in \{\pm x, \pm y, \pm z\}$. Für jeden Wegpunkt $\pi^{i,j}[k]$ in $\pi^{i,j}$, kann jeder RSFC ausgewählt sein, wenn die folgende Bedingung erfüllt ist:

$$\pi^{i,j}[k] \cdot n_\mu > 0 \tag{5.23}$$

Dann wird ein geeigneter RSFC aus den RSFC-Kandidaten ausgewählt. Redundante RSFC-Übergänge entlang der Wegpunkte können jedoch die Anzahl der Polynomsegmente und die Rechenzeit erhöhen. Abbildung 6.9 zeigt ein Beispiel dafür. Zur Erstellung einer glatten relativen Trajektorie ist es erforderlich, dass zwei Polynomsegmente als Ersatz für die relative Trajektorie dienen, wenn es einen Übergang von RSFC entlang der Wegpunkte gibt (z. B. Abbildung 6.7 $RSFC_1^{i,j} \rightarrow RSFC_2^{i,j}$). Aber wenn es drei Übergänge

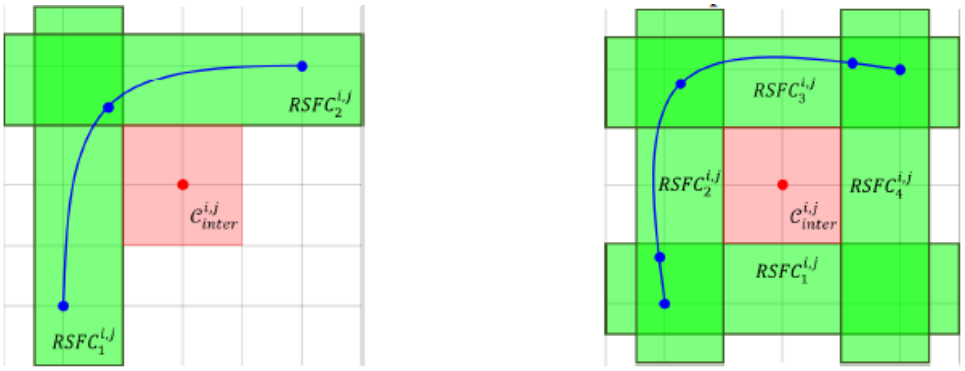


a) Initiale Trajektorien des i-ten (rot) und j-ten (blau) Quadrotors

b) Relative initiale Trajektorie des j-ten Quadrotors in Bezug auf den i-ten Quadrotor

Abbildung 5.8: Die relativ initiale Trajektorie [12]

gibt (wie in Abbildung 6.8 $RSFC_1^{i,j} \rightarrow RSFC_2^{i,j} \rightarrow RSFC_3^{i,j} \rightarrow RSFC_4^{i,j}$), müssen zwei zusätzliche Polynomsegmente geplant werden. Der Greedy-Algorithmus (13) kommt zur Anwendung, um die Anzahl der RSFC-Übergänge zu minimieren.



a) RSFC Konstruktion

b) Beispiel für einen redundanten RSFC-Übergang.

Abbildung 5.9: RSFC [12]

Algorithm 13: Erstellung des RSFCs (buildRSFC)

Input: π^i, π^j, 3D-Belegungskarte ε
Output: RSFC $RSFC^{i,j}$

```
1  l_max ← max(size(π^i), size(π^j)) ;
2  RSFC^{i,j} ← ∅;
3  for ∀μ ∈ {±x, ±y, ±z} do
4  |  initialize s_μ to 0 ;
5  end
6  for n ← 1 to l_max do
7  |  for ∀μ ∈ {±x, ±y, ±z} do
8  |  |  if (π^j[n] − π^i[n]) · n_μ > 0 then
9  |  |  |  if n = 1 then
10 |  |  |  |  s_μ[n] ← 1;
11 |  |  |  end
12 |  |  |  s_μ[n] ← s_μ[n − 1] + 1
13 |  |  end
14 |  end
15 end
16 n ← l_max;
17 μ_M ← argmax_μ(s_μ[n]);
18 RSFC^{i,j}.push_front (RSFC_{μ_M});
19 n ← n − s_{μ_M}[n];
20 while n > 0 do
21 |  μ_M ← argmax_{μ≠−μ_M}(s_μ[n]);
22 |  RSFC^{i,j}.push_front(RSFC_{μ_M});
23 |  n ← n − s_{μ_M}[n];
24 end
25 return RSFC^{i,j};
```

Der Algorithmus empfängt π^i und π^j als Eingabe und gibt $RSFC^{i,j}$ zurück. $RSFC^{i,j}$ ist als leeres Array initialisiert (Zeile 2), und s_μ ist als ein Array aller Nullen mit der Länge l_{max} initialisiert (Zeile 4-5). Nach der Initialisierung überprüft der Algorithmus die RSFC-Kandidaten mit (5.23) und speichert das Ergebnis in s_μ (Zeilen 8-13). Am Ende des relativen Pfads wird ein RSFC-Kandidat gefunden, der die maximale Anzahl von Wegpunkten enthält, und der Kandidat wird im $RSFC^{i,j}$ eingefügt (Zeilen 17-18). Danach geht es zum letzten Wegpunkt (Zeile 19). Der Algorithmus findet wieder das Maximum des Kandidaten, bis es den Startpunkt des relativen Pfades erreicht (Zeilen

20-24). Es ist zu beobachten, dass sich der neue Kandidat nicht auf der dem vorherigen Kandidaten gegenüberliegenden Seite befinden darf, da Quadrotoren nicht durch einen leeren Raum zwischen zwei gegenüberliegenden Kandidaten springen können (Zeile 21).

5.3.5 Dummy-Agenten

Das gleichzeitige Optimieren aller Kontrollpunkte von Polynomen kann ein Skalierbarkeitsproblem verursachen, da die zeitliche Komplexität des QP-Lösers $O(n^3)$ beträgt. Hier wird eine effiziente sequenzielle Optimierungsmethode unter Verwendung von Dummy-Agenten vorgestellt. Algorithmus 14 zeigt den Prozess der sequenziellen Optimierung.

Algorithm 14: Trajektorieoptimierung (trajOpt)

Input: initiale Trajektorie π, $SFC^{\forall i}$, $RSFC^{\forall i,j>i}$
Output: Trajektorie $p^i(t)$

1 $p_{dmy}(t) = (p^0_{dmy}(t), \cdots, p^N_{dmy}(t)) \leftarrow$ planDummy(π);
2 **for** $l \leftarrow 1$ *to* N_b **do**
3 $\quad b \leftarrow$ agents in l^{th} batch ;
4 $\quad p^b(t) \leftarrow$ solveQP($\pi^b, SFC^b, RSFC^{\forall i,j>i}, p^{\forall i \notin b}_{dmy}(t)$) ;
5 $\quad p_{dmy}(t) \leftarrow p(t)$;
6 **end**
7 return $p^o(t), \cdots, p^N(t)$;

Zunächst werden die Trajektorien für Dummy-Agenten $p_{dmy}(t)$ unter Verwendung der folgenden Kontrollpunkte $c^i_{m,k}$ erstellt (Zeile 1):

$$c^i_{m,k} = \begin{cases} \pi^i_{m-1}, & k = 0, \cdots, \phi - 1 \\ \pi^i_m, & k = n - (\phi - 1), \cdots, n \\ x \in \left\langle \pi^i_{m-1}, \pi^i_m \right\rangle, & \text{else} \end{cases} \tag{5.24}$$

Als Nächstes werden die Agenten in N_b Stapeln aufgeteilt und das QP-Problem für den

Stapel b sieht wie folgt aus (Zeilen 3-4) [61]:

$$\text{Minimieren } \; c^T Q c, \qquad c \in \mathbb{R}^{\frac{N}{N_b} M(n+1)} \tag{5.25}$$

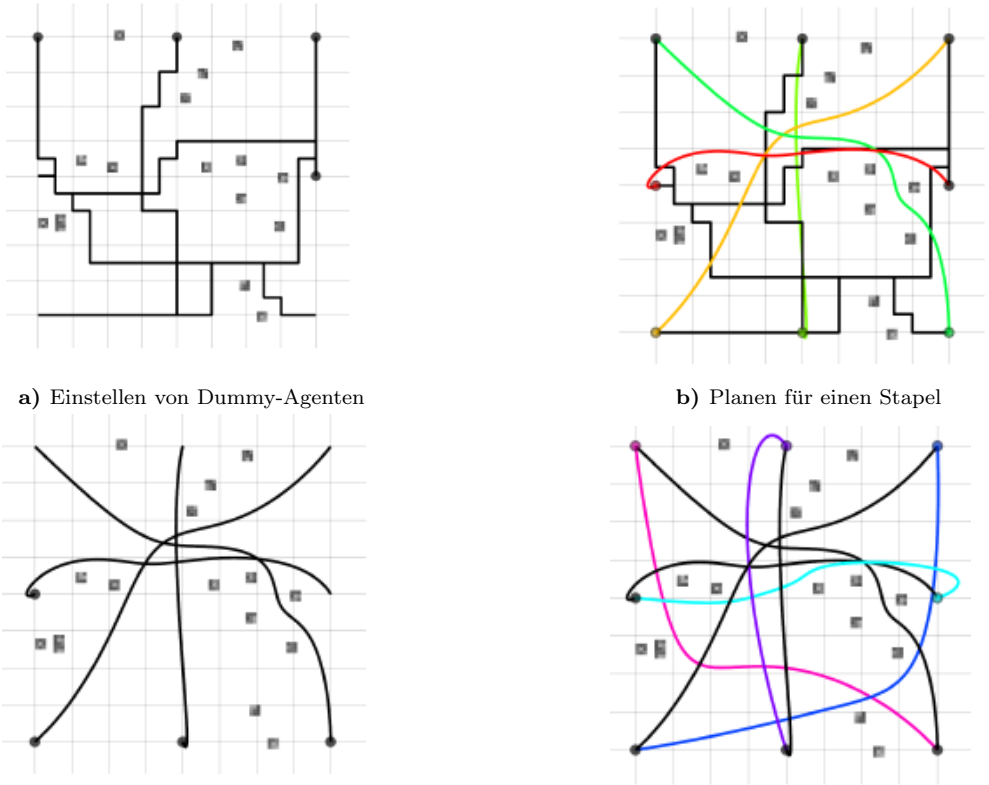
$$\text{Erfüllen } \; A_{eq} c = b_{eq}, \qquad A_{eq} \in \mathbb{R}^{\frac{N}{N_b}(M+1)\phi \times \frac{N}{N_b} M(n+1)} \tag{5.26}$$

$$c^i_{m,k} = \text{Kontrollpunkte von } p^i_{dmy}(t), \qquad \forall \notin b, m, k \tag{5.27}$$

$$c^i_{m,k} \in SFC^i_m, \qquad \forall \in b, m, k \tag{5.28}$$

$$c^j_{m,k} - c^i_{m,k} \in RSFC^{i,j}_m, \qquad \forall i, j > i, m, k \tag{5.29}$$

Die Anzahl der Ungleichheitsbeschränkungen ist $(N - \frac{1}{2}(\frac{N}{N_b} + 1))\frac{N}{N_b} M(n+1)$. $p^i_{dmy}(t)$ ist die Trajektorie für den i-ten Dummy-Agenten. Zuletzt werden die Trajektorien von Dummy-Agenten durch die zuvor geplanten Trajektorien ersetzt und die Trajektorie für den nächsten Stapel wird geplant (Zeile 5).



a) Einstellen von Dummy-Agenten

b) Planen für einen Stapel

c) Ersetzen der Dummy-Agenten durch den vorherigen Stapel

d) Planen für den nächsten Stapel

Abbildung 5.10: Dummy-Agenten [13]

Abbildung 6.14 zeigt den Gesamtprozess der Dummy-Agenten ($N_b = 2$). Dummy-Agenten werden als schwarze Kreise dargestellt, und Agenten im aktuellen Stapel werden als farbige Kreise dargestellt. Für jede Iteration werden die Trajektorien für den aktuellen Stapel

(Farblinie) geplant, die die Trajektorien von Dummy-Agenten (schwarze Linie) vermeidet. Für jede Iteration werden die Dummy-Agenten mit Ausnahme der Agenten im aktuellen Stapel eingestellt (Abbildung 6.10). Dann plant der Algorithmus die Pfade für den aktuellen Stapel, um Dummy-Agenten zu vermeiden (Abbildung 6.11). Danach werden die Agenten im aktuellen Stapel bei der nächsten Iteration als Dummy-Agenten verwendet (Abbildung 6.12). Am Ende der Iteration werden kollisionsfreie Trajektorien ohne Verklemmung gefunden, da alle Agenten so geplant sind, dass der vorherige Stapel vermieden wird (Abbildung 6.13). Diese sequenzielle Methode mit Dummy-Agenten kann eine bessere Skalierbarkeit erzielen, da die hohe zeitliche Komplexität des QP-Lösers vermieden wird. Wenn die Anzahl der Agenten zunimmt, während die Anzahl der Entscheidungsvariablen von QP beibehalten wird, indem die Anzahl des Stapels sich erhöht. Darüber hinaus ist es nach [13] erwiesen, dass die Methode keinen Optimierungsfehler aufgrund nicht realisierbarer Einschränkungen verursacht.

5.3.6 Zeitzuweisung

Nach der Konstruktion von SFC und RSFC ist es erforderlich, dass das Zeitsegment der stückweisen Polynomtrajektorie dem entsprechenden SFC und RSFC zugeordnet wird. $p_m^i(t)$ ist das m-te Segment der Trajektorie $p^i(t)$ in $t \in \left\{t_{m-1}^i, t_m^i\right\}$. Das Zeitsegment des i-ten Quadrotors ist folgendermaßen definiert:

$$t_s^i = \left[t_0^i, \cdots, t_M^i\right] \tag{5.30}$$

Für diese Arbeit werden die Trajektorien aller Agenten so eingestellt, dass sie das gleiche Zeitsegment t_s haben, um die konvexe Hülleneigenschaft des Bernstein-Basispolynoms zu verwenden. Dies kann jedoch zu vielen Entscheidungsvariablen führen und somit die Rechenzeit verlängern. Daher wird das folgende Verfahren verwendet, um die Anzahl von Entscheidungsvariablen zu verringern. Algorithmus 15 zeigt den Prozess zum Finden eines Zeitsegmentteils. Der Algorithmus empfängt SFC oder RSFC und die initialen oder relativen Trajektorien als Eingaben und sucht nach dem mittleren Wegpunkt zwischen dem Schnittpunkt zweier aufeinanderfolgender konvexer Mengen (Zeilen 11-13). Danach zeichnet der Algorithmus den Index dieses mittleren Wegpunkts auf, um ihn als den Ort zuzuweisen, an dem der SFC- oder RSFC-Übergang stattfindet (Zeile 14). Mit anderen Worten, der m-te SFC oder RSFC wird vor dem Zeitpunkt $(n + \left\lfloor \frac{count}{2} \right\rfloor) * t_{step}$ zugewiesen, und der *m+1*-ten SFC oder RSFC wird nach dem Zeitpunkt $(n + \left\lceil \frac{count}{2} \right\rceil) * t_{step}$ zugewiesen, wobei $n + \left\lfloor \frac{count}{2} \right\rfloor)$ der Index des mittleren Wegpunktes zwischen dem Schnittpunkt der m-ten und *m+1*-ten konvexen Mengen ist. Im SFC-Fall ist es garantiert, dass in zwei aufeinander folgenden SFC ein Wegpunkt vorhanden ist, da der Wegpunkt über die

Achsensuchmethode mit SFC verbunden ist. Im RSFC-Fall gibt es jedoch möglicherweise keinen Wegpunkt in einer Kreuzung zwischen zwei aufeinanderfolgenden RSFC (Zeile 17). In diesem Fall wird der Integerindex nicht mehr verwendet, da er eine nicht realisierbare Einschränkung darstellen kann, wenn SFC und RSFC sich gleichzeitig ändern. Stattdessen steht eine heuristische Methode zur Verfügung, die den RSFC-Übergang zeitverzögert, um die gleichzeitige Änderung von SFC und RSFC zu vermeiden (Zeile 18). Dies kann die Anzahl der Entscheidungsvariablen erhöhen, aber die Erfolgsrate nimmt beim Finden einer realisierbaren Flugbahn zu. Dieser Algorithmus gibt immer ein Array mit einer maximalen Größe von $2l_{max}$ zurück, so dass garantiert ist, dass die stückweise Trajektorie maximal $2l_{max}$ Segmente aufweist.

Algorithm 15: Zeitsegment zuweisen (findTimeSegment)

Input: initiale Trajektorie π oder relative Trajektorie $\pi^{i,j}$, Array von sequenziellen konvexen Mengen C, Zeitschritt t_{step}

Output: Zeitsegment t_{sp}

```
1  t_sp ← ∅;
2  m ← 1;
3  for n ← 1 to l_max do
4      if m ≥ size(C) then
5          break;
6      end
7      if π[n] ∈ (C[m] ∩ C[m + 1]) then
8          count ← 1;
9          while π[n + count] ∈ (C[m] ∩ C[m + 1]) and n + count ≤ l_max do
10             count ← count + 1;
11             t_sp.push_back((n + ⌊count/2⌋) ∗ t_step);
12             n ← n + ⌊count/2⌋;
13             m ← m + 1;
14         end
15     end
16     else if π[n] ∈ C[m + 1] then
17         t_sp.push_back((n + 0.5) ∗ t_step);
18         m ← m + 1;
19     end
20 end
21 return t_sp;
```

Nach der Erstellung des Zeitsegments werden alle Zeitsegmente kombiniert und sortiert.

Das bedeutet, dass doppelte Elemente gelöscht werden und das Gesamtzeitsegment generiert wird, indem die Startzeit und die Gesamtflugzeit an jedes Ende des kombinierten Arrays angehängt werden. Diese Methode kann die Größe des gesamten Zeitsegments reduzieren, indem die Elemente des Zeitsegmentteils so weit wie möglich überlappt werden. Durch Vergleichen von t_{sp} und t_s werden SFC und RSFC dem Zeitsegment zugeordnet.

6 Simulation und Evaluation

In diesem Kapitel werden die Simulation und Evaluation unter Verwendung des vorab erarbeiteten Absicherungskonzeptes sowie der implementierten Verarbeitungsarchitektur beschrieben. Im Detail erläutert werden sollen hierbei die Architektur der Simulation, die Simulation der statischen Methode und der dynamischen Methode sowie die Evaluation der beiden Methoden. Zuerst wird der Konstruktionsprozess der Simulation vorgestellt, nämlich die Verbindung zwischen ROS-Framework, PX4-Flugsteuerung und Gazebo-Simulator. Darauf folgt eine kurze Einführung in die Simulation der beiden Methoden. Den Abschluss bilden die Bewertung und Evaluation der Simulationsergebnisse unter Berücksichtigung verschiedener Faktoren, z. B. Rechenzeiten, Robustheit und Leistungsfähigkeit.

6.1 Simulation

Um die vom Pfadplaner generierten Trajektorien zu demonstrieren, ist eine Tracking-Steuerung notwendig, damit die Drohne gemäß dieser vorhandenen Trajektorie in Bewegung gehalten werden kann. Deshalb werden zunächst zwei Verfahren zur Tracking-Steuerung im Rahmen dieser Arbeit vorgestellt, nämlich die lineare Tracking-Steuerung und die PID-Tracking-Steuerung. Dem folgt eine Übersicht zur Architektur und zum Informationsfluss der Demonstration.

6.1.1 Trajektorienfolger

Der lineare Trajektorienfolger basiert auf der Differenz zwischen der Zieltrajektorienposition und der aktuellen Trajektorienposition, um die die Geschwindigkeit und Richtung der Bewegung zu bestimmen. Der Pfadpunkt auf der Trajektorie wird als Referenzpunkt p_r des Folgers verwendet, und die vom Simulator gemessene Position wird als Echtzeitposition p_0 verwendet. Entsprechend der Genauigkeit scannt der Knoten die beiden Positionen mit einer festgelegten Frequenz. Abbildung 6.1 zeigt die lineare Steuerung, wobei $v = (v_x, v_y, v_z)$ die Geschwindigkeit von p_o nach p_r ist. v_x, v_y, v_z sind die Geschwindigkeitskomponenten auf der X-Achse, der Y-Achse und der Z-Achse.

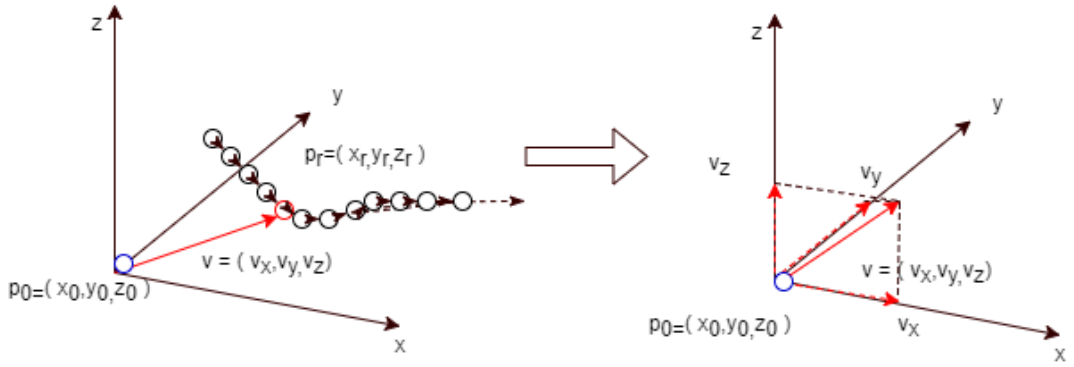


Abbildung 6.1: Der lineare Trajektorienfolger

Die euklidische Abstand $\triangle s$ zwischen p_0 und p_r ist gleich $p_r - p_0$. Die Durchschnittsgeschwindigkeit wird mit V_d bezeichnet. Dann werden die Geschwindigkeitskomponenten v_x, v_y, v_z wie folgt berechnet:

$$\triangle s = \sqrt{(x_r - x_0)^2 + (y_r - y_0)^2 + (z_r - z_0)^2} \tag{6.1}$$

$$v^2 = \sqrt{v_x^2 + v_y^2 + v_z^2} \tag{6.2}$$

$$v_x = \frac{x_r - x_0}{\triangle s} * V_d = \frac{x_r - x_0}{\sqrt{(x_r - x_0)^2 + (y_r - y_0)^2 + (z_r - z_0)^2}} * V_d \tag{6.3}$$

$$v_y = \frac{y_r - y_0}{\triangle s} * V_d = \frac{y_r - y_0}{\sqrt{(x_r - x_0)^2 + (y_r - y_0)^2 + (z_r - z_0)^2}} * V_d \tag{6.4}$$

$$v_z = \frac{z_r - z_0}{\triangle s} * V_d = \frac{z_r - z_0}{\sqrt{(y_r - y_0)^2 + (y_r - y_0)^2 + (z_r - z_0)^2}} * V_d \tag{6.5}$$

Im Unterschied zum linearen Trajektorienfolger basiert der PID-Trajektorienfolger auf nicht nur auf der Positionsdifferenz, sondern auch auf dem Geschwindigkeitsfehler, damit die Position bei Störeinflüssen möglichst gut eingehalten wird [77] [78] [79]. In Abbildung 6.2 ist die PID-Regelung dargestellt. Der PID-Regler ist von den Standard-Reglern am anpassungsfähigsten, verhindert bei konstantem Sollwert eine bleibende Regelabweichung bei Führungs- und Störgrößensprung und kann Verzögerungen der Regelstrecke kompensieren und damit die Regelstrecke vereinfachen. Durch die Integration von Positionsfehlern und die Regelung von Geschwindigkeitsfehlern wird die geregelte Beschleunigung ausgerechnet. Bei der Positionssteuerung wird bei einem Sollwertsprung für die Position meist ein Referenzprofil für Geschwindigkeit und Beschleunigung generiert. Entsprechend dem Sollwertverlauf und einem linearisierten Streckenverhalten wird eine Beschleunigungsvorsteuerung berechnet, die ohne weitere Stellgrößenanteile die linearisierte Strecke entsprechend den Referenzprofilen positionieren würde. Der eigentliche Regler, also die anderen Stellgrößenanteile, müssen dann nur noch Nichtlinearitäten und unbekannte äußere Einflüsse ausregeln.

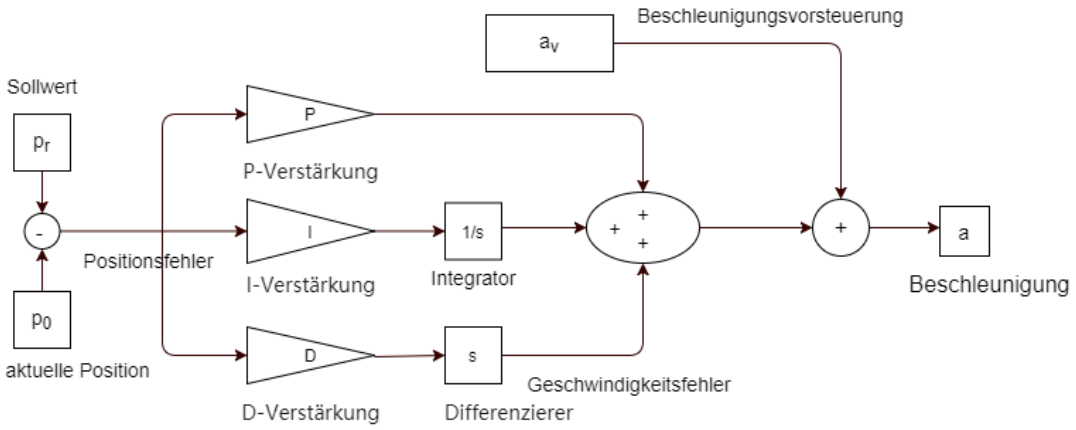


Abbildung 6.2: PID-Trajektorienfolger

6.1.2 Simulation der statischen Methode

Die Architektur der statischen Methode ist in Abbildung 6.3 dargestellt. Die Simulation und Visualisierung bestehen darin, die Kommunikation und Informationsverarbeitung zwischen verschiedenen Modulen im Rahmen des ROS-Frameworks zu realisieren. Im OctoMap-Modul wird eine Belegungskarte generiert, die Hindernisse unterschiedlicher Form und Volumen enthält. Durch den OctoMap-Knoten wird die Belegungskarte zum ROS-Knoten *multi_agents_Planner_3d* gesendet, damit der Pfadplaner die Hindernisse erkennen kann. Außerdem wird die Belegungskarte im Kartenkonverter-Modul nach *World*-file umgesetzt, um die Belegungskarte in Gazebo zu modellieren. Im Planermodul wird eine kollisionsfreie Trajektorie für jede Drohne geplant, und die Trajektorien werden durch ROS-Knoten *multi_agents_Planner_3d* zu ROS-Knoten *trajectory_controller_uav* weitergeleitet. Der Trajektorienfolger empfängt die Trajektorien und sendet die aktuellen Positionskontrollpunkte und Geschwindigkeit an Gazebo durch MAVROS und PX4. Gleichzeitig werden die aktuelle Bewegungszustände jeder Drohne von Gazebo an den Trajektorienfolger zurückgegeben, damit der Regler den Fehler reduzieren kann. Im *Gazabo_Gui* wird die Bewegung der Drohne angezeigt. Es ist sichtbar zu prüfen, ob die Drohne die Hindernisse vermeiden kann.

6.1.3 Simulation der dynamischen Methode

Abbildung 6.4 zeigt die Architektur der dynamischen Methode, die der statischen Methode ähnelt. Der Unterschied liegt im Planermodul, in dem der 4D-Planer die Trajektorien für jede Drohne berechnet. Der ROS-Knoten *multi_agents_Planner_4d* empfängt die Belegungskarte aus den OctoMap-Knoten. Nach der dynamischen Pfadplanung werden die Trajektorien an *trajectory_controller_uav* gesendet. Der Gazebo-Simulator erhält die ak-

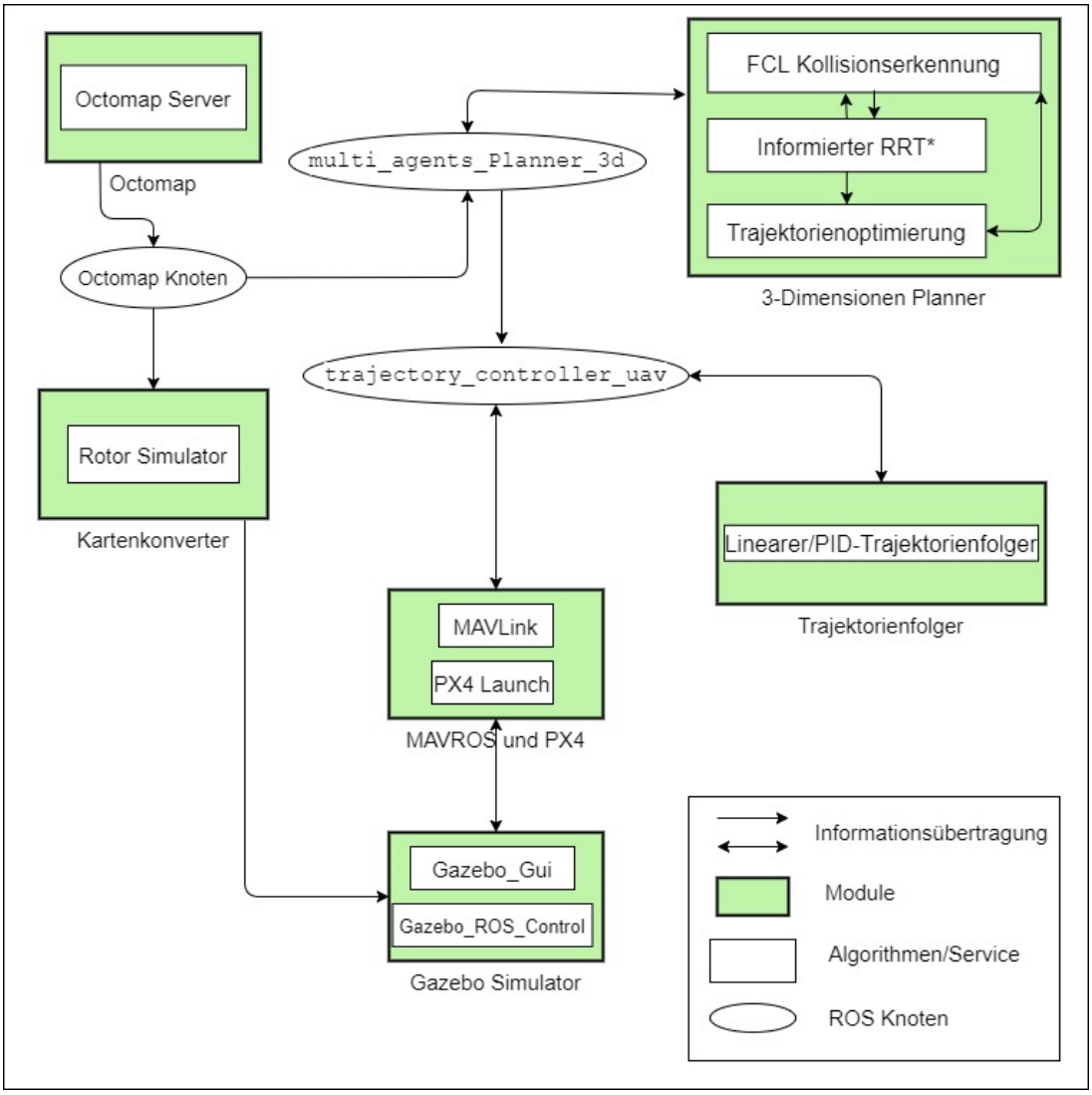


Abbildung 6.3: Architektur der Simulation der 3D-Methode

tuellen Positionskontrollpunkte und Daten zur Geschwindigkeit und Beschleunigung von MAVROS. Abschließend werden die Belegungskarte und der Bewegungszustand der Drohne in Gazebo simuliert.

6.2 Evaluation

In diesem Abschnitt werden unterschiedliche Faktoren der beiden Algorithmen analysiert. Untersucht werden dazu die Pfadkosten, Erfolgsraten, Rechenzeiten, Anzahl der Drohnen, Hindernisdichte und Robustheit, um die beiden Methoden zu vergleichen, ihre jeweiligen Vor- und Nachteile aufzuzeigen und den Anwendungsbereich jeder Methode zu erläutern. Zur Untersuchung der Abhängigkeit von der Hindernisdichte werden verschiedene 3D-Belegungskarten mit unterschiedlicher Anzahl

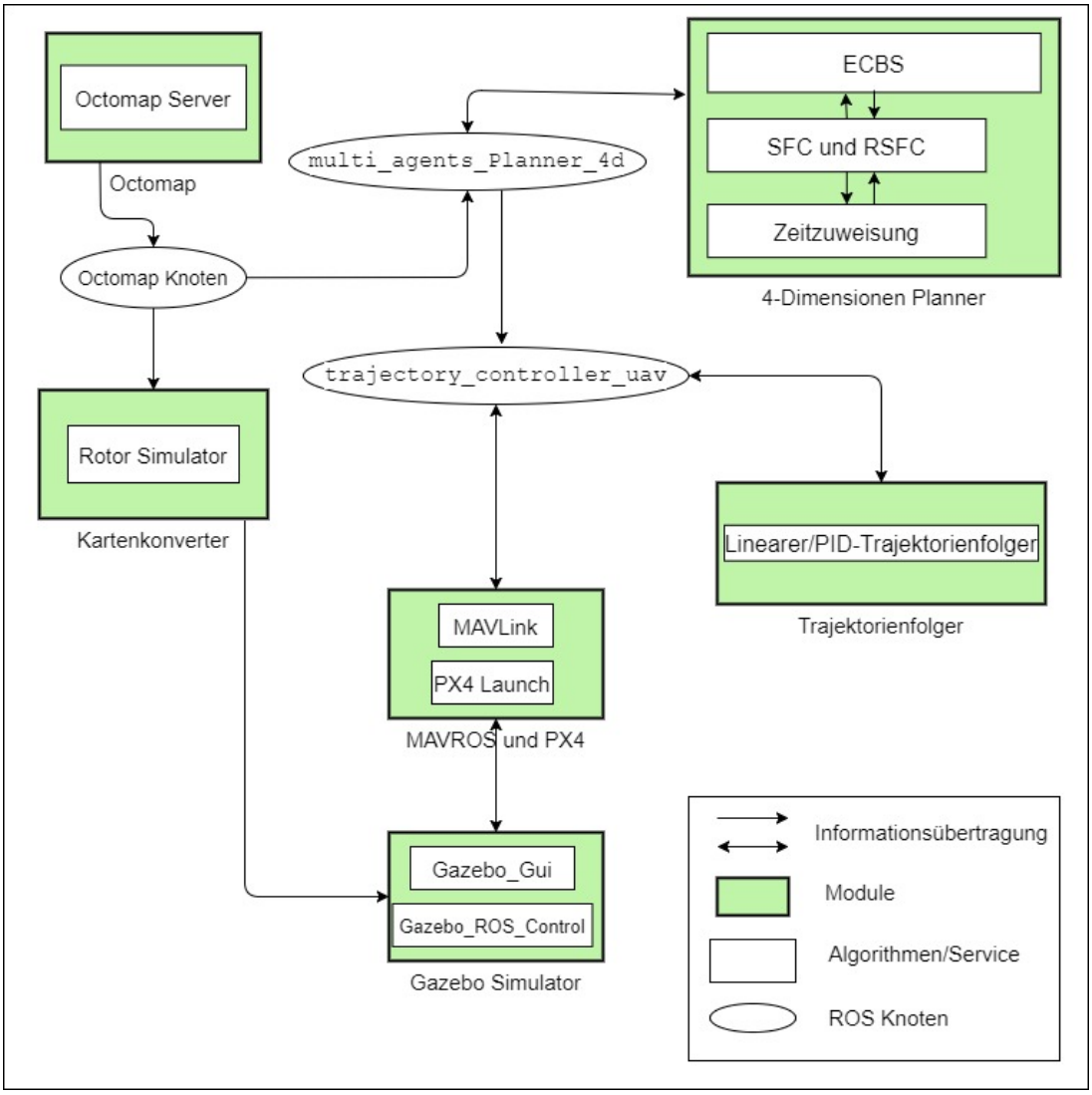
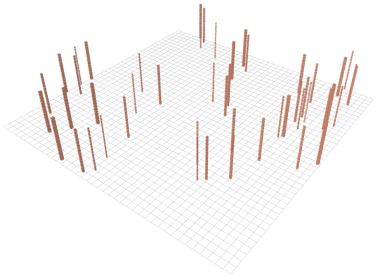


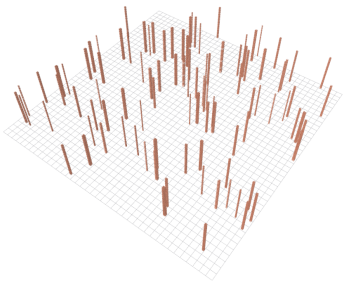
Abbildung 6.4: Architektur der Simulation der 4D-Methode

von Hindernissen generiert. Für diese Arbeit ist die 3D-Belegungskarte im Raum $\{(x, y, z) \,||\, x \in [-20, 20], y \in [-20, 20], z \in [0, 10]\}\,(m)$ definiert. Die Diagramme in Abbildung 6.5 zeigen Räume mit zylindrischen Hindernissen mit einer Höhe von 7 m und einem Durchmesser von 0,1 m bis 0,4 m in unterschiedlicher Anzahl und damit Dichte, von 50 (niedrige Dichte) bis 600 (extreme Dichte).

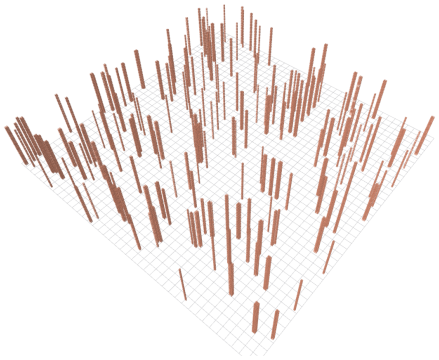
Zur Analyse der Qualität der Trajektorie wird der Begriff die Glätte des Pfades G wie folgend 6.6 dargestellt. Die Idee besteht darin, die Dreiecke zu betrachten, die durch aufeinanderfolgende Pfadsegmente s_{i-2}, s_{i-1}, s_i gebildet werden, und den Winkel zwischen diesen Segmenten unter Verwendung des Satzes von Pythagoras zu berechnen [80]. Dann wird der Außenwinkel für den berechneten Winkel durch die Pfadsegmente normalisiert und trägt zur Pfadglätte bei. Dabei ist a_i der Abstand von s_{i-2} bis s_{i-1}, b_i der Abstand von s_{i-1} bis s_i und c_i der Abstand von s_{i-2} bis s_i. Für einen geraden Pfad ist die Glätte



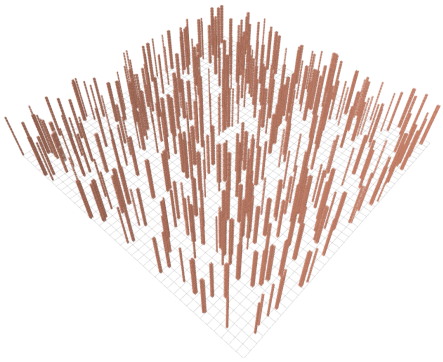
a) 50 zylindrisches Hindernisse (niedrig Dichte)



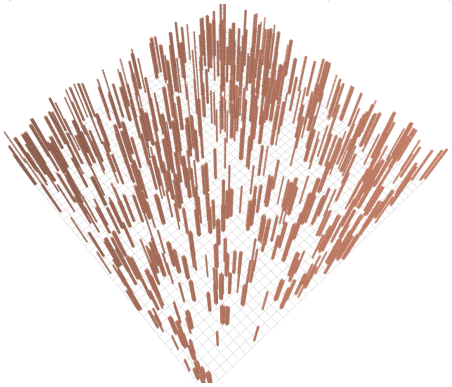
b) 100 zylindrisches Hindernisse (mittlere Dichte)



c) 200 zylindrisches Hindernisse (hohe Dichte)



d) 400 zylindrisches Hindernisse (sehr hohe Dichte)



e) 600 zylindrisches Hindernisse (extreme Dichte)

Abbildung 6.5: 3D-Belegungskarte mit unterschiedlicher Anzahl von Hindernissen

gleich 0.Je näher der Wert an 0 liegt, desto glatter ist der Pfad.

$$G = \sum_{i=2}^{n-1} \left(\frac{2(\pi - arccos(\frac{a_i^2 + b_i^2 - c_i^2}{2a_i b_i}))}{a_i + b_i} \right)^2 \tag{6.6}$$

Die vorgeschlagenen Methoden werden in C++ ausgeführt und auf einem PC mit Ubuntu 16.04 mit Intel Xeon(R) CPU E3-1230 V2 3.30GHz × 8 und Grafikkarte Quadro 600/PCIe/SSE2 simuliert. Das Gazebo 7 und Firmwire 1.8.2 werden verwendet, um den Bewegungszustand der Drohne in der tatsächlichen Umgebung zu modellieren und zu

visualisieren.

6.2.1 Evaluation der statischen Methode

Zur Evaluation der statischen Methode werden die Pfadkosten, die Rechenzeiten, die Anzahl der Drohnen, die Pfadglätte und die Hindernisdichte betrachtet. In der statischen Methode gibt es eine Zeitfunktion, die ein Zeitlimit bzw. die Anzahl der Sekunden definiert, die der informierte RRT* Algorithmus für die Planung verwenden darf. Je länger die Zeit ist, desto mehr Zustände werden vom Algorithmus abgetastet. Zuerst wird das Zeitlimit von 0,1 bis 40 Sekunden festgelegt, um die Beziehung zwischen Zeitlimit und die Qualität der Lösung zu analysieren. Abbildung 6.6 zeigt die Pfadkosten in Abhängigkeit vom Zeitlimit in der Karte mit 200 Hindernissen. Aufgrund der zufälligen Stichprobe des Algorithmus nimmt die Pfadlänge nicht notwendigerweise linear mit der Zeit ab, aber der Trend zeigt, dass die Pfadlänge mit der Zeit eine signifikante Abnahme aufweist. Wenn die Planungszeit zu lang ist, nimmt allerdings auch die Reaktionsgeschwindigkeit des gesamten Systems ab. Als Ausgleich zwischen beiden Faktoren wird das Zeitlimit in statischer Methode auf eine Sekunde gesetzt. In Abbildung 6.7 stellt den Zusammenhang

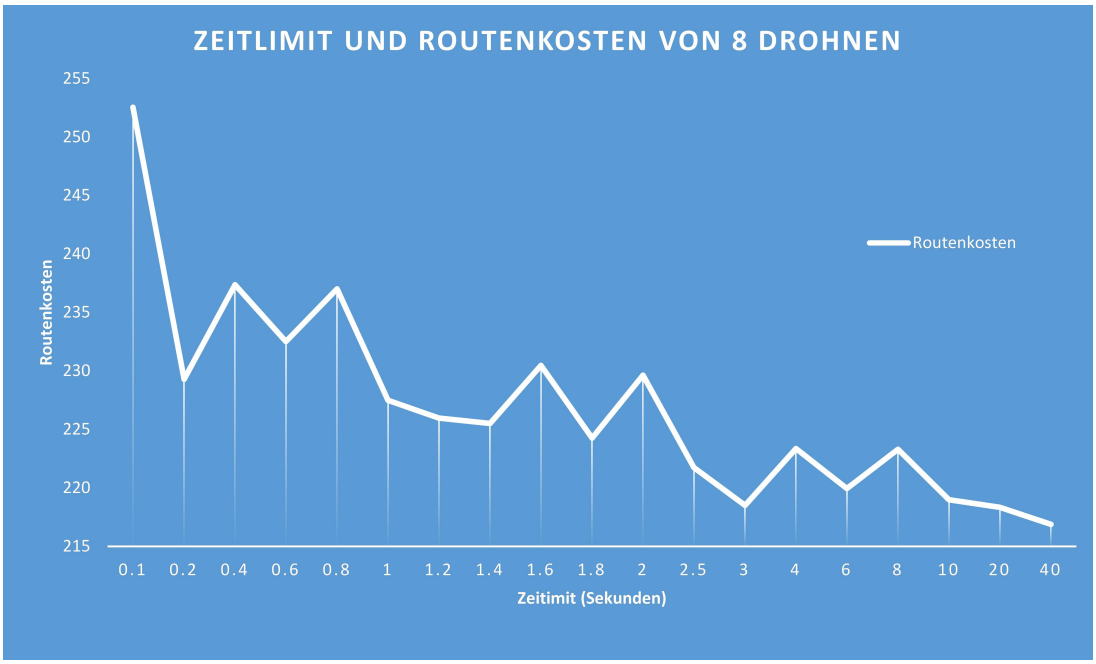


Abbildung 6.6: Die Beziehung zwischen Zeitlimit und Routenkosten

zwischen der Rechenzeit und der Anzahl der Drohnen dar. Dabei wird deutlich, dass die Rechenzeit mit der Anzahl der Drohnen steigt. Für jede Planung ist die Rechenzeit gleich Planungszeit plus Optimierungszeit. Die Planungszeit wurde wie oben beschrieben auf eine Sekunde gesetzt, und die Optimierungszeit ist abhängig von die Pfadlänge.

Die Rechenzeit für 64 Drohnen ist gleich der Planungszeit für 64 Drohnen plus Optimierungszeit für 64 Pfade. Die Planungszeit für 64 Drohnen ist 64 Sekunden und die Optimierungszeit ist in diesem Fall 14 Sekunden. Um die Rechenzeit zu reduzieren, gibt es zwei Möglichkeiten, nämlich die Reduktion der Planungszeit bzw. des Zeitlimits oder die Verbesserung der Optimierungseffizienz. Letztere ist in dieser Arbeit abhängig von den Parametern der B-Spline-Kurve, nämlich maximale Schritte *smoothSteps* und minimale Änderungen *pathMinChange*. Abbildung 6.8 skizziert den Zusammenhang zwischen der Optimierungszeit und den B-Spline Parametern. Die Optimierungszeit nimmt mit abnehmender *pathMinChange* und zunehmender *smoothSteps* zu. Abbildungen 6.9 zeigt die Pfade nach der B-Spline-Optimierung mit unterschiedlichen Werten für *pathMinChange* und *smoothSteps*. Die Diagramme a) bis d) zeigen, dass die Glätte des Pfades mit mehr Optimierungsschritten zunimmt. Die Diagramme e) bis h) zeigen den Prozess der kontinuierlichen Optimierung der Glätte des Pfades mit *pathMinChange* von 0.5 bis 0.001. Daraus lässt sich die Schlussfolgerung ziehen, dass kleinere *pathMinChange* und mehr *smoothSteps* zwar die Glätte des Pfades verbessern, dabei aber die Optimierungszeit verlängern. In der folgenden Simulation wird *pathMinChange* auf 0.001 und *smoothSteps* auf 5 konfiguriert.

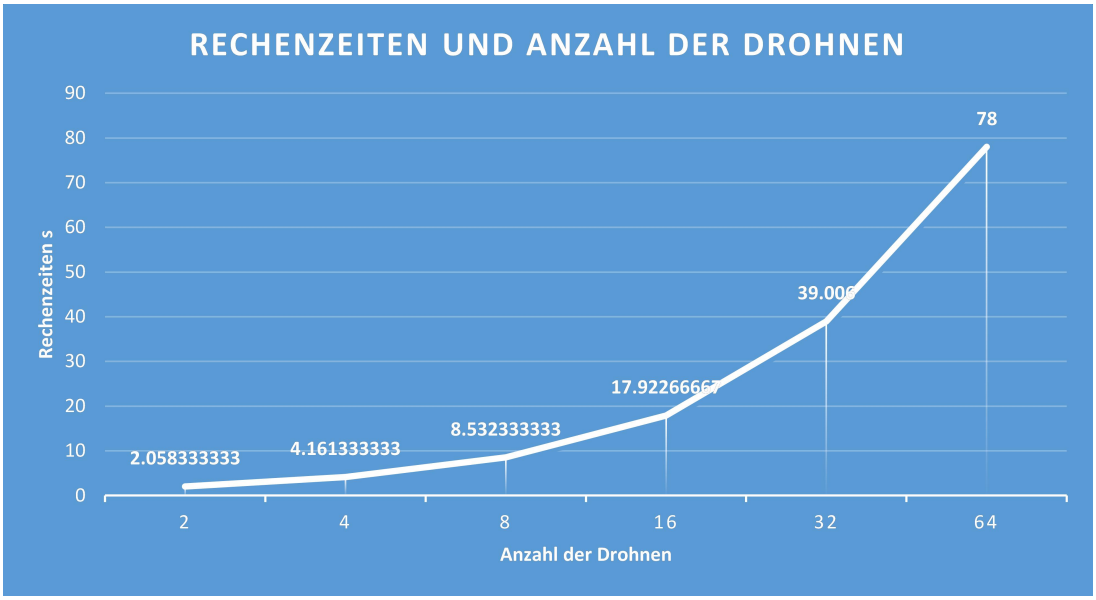
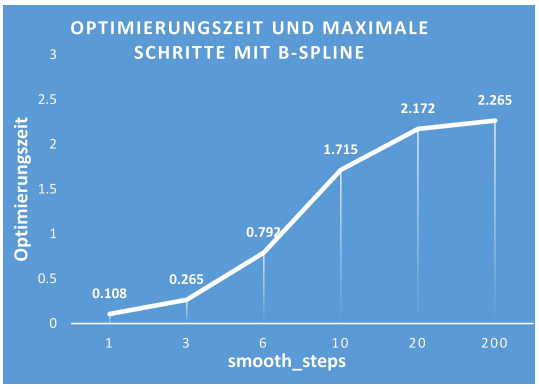
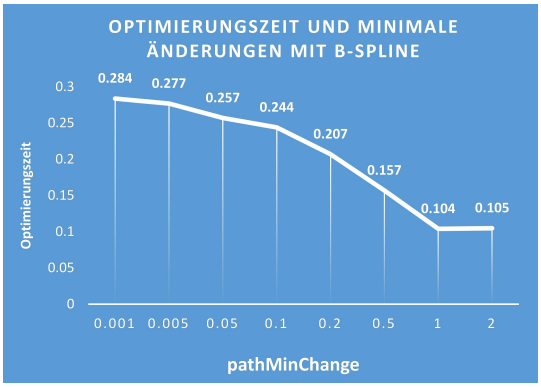


Abbildung 6.7: Rechenzeit und Anzahl der Drohnen

Abbildung 6.10 beschreibt die Glätte von 32 Drohnenpfaden, wenn *pathMinChange* = 0.001 und *smoothSteps* = 5 beträgt. Gemäß der Grafik liegt die Glätte der Pfade zwischen 0,1 und 0,6 und die Glätte aller 32 Pfade ist in diesem Intervall gleichmäßig verteilt. Das bedeutet, dass die sequenzielle Planung von der ersten bis zur 32. Drohne keine negativen Auswirkungen auf die Qualität des Pfades hat, wenn der Optimierungsparameter



a) Die Optimierungszeit und die maximale Schritte



b) Die Optimierungszeit und die minimale Änderungen

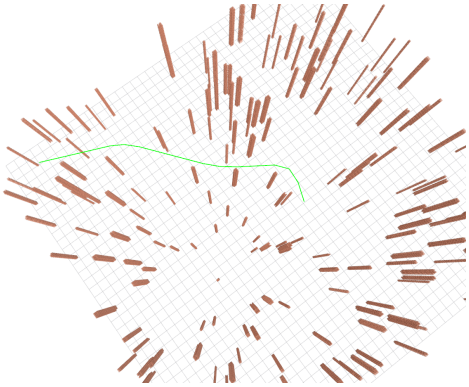
Abbildung 6.8: Die Optimierungszeit für 8 Drohnen mit B-Spline

gleich konfiguriert ist. Priorisierte Drohnen haben jedoch immer eine größere Auswahl an planbarem Raum, da die später geplanten Drohnenpfade die zuvor geplanten Pfade als Hindernisse berücksichtigen müssen. Deshalb wird der planbare Raum, der für nachfolgende Drohnen (niedrige Priorität) verfügbar ist, mit fortschreitender Planung kleiner. Theoretisch ist die Chance einer Lösung umso höher, je höher die Priorität der Drohne ist. Je niedriger der Rang der Drohnen im zeitlichen Ablauf, desto kleiner ist der Raum, den sie wählen können, und desto geringer ist die Wahrscheinlichkeit dafür, eine bessere Pfadlösung zu erhalten. Dieser Punkt wird in Kapitel 6.2.3 ausführlich beschrieben.

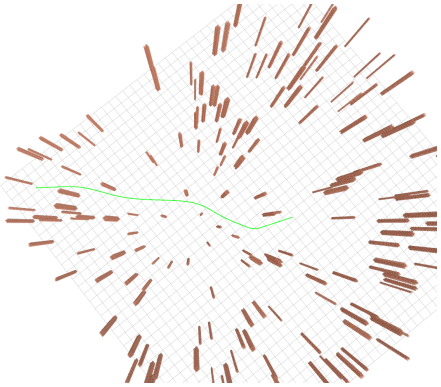
Abbildung 6.11 skizziert die Ergebnisse der Simulation, um die Abhängigkeit der Hindernisdichte zu analysieren. Das Zeitlimit ist 1 Sekunde und die Optimierungsparameter lauten $pathMinChange = 0.001, smoothSteps = 5$. Mit zunehmender Anzahl der Hindernisse in der Karte von 50 bis 600 steigen die Rechenzeit, die Pfadkosten und die Glätte. Dies zeigt, dass bei einer höheren Dichte von Hindernissen in der Umgebung auch die geplante Pfadlänge zunimmt, ebenso wie die für den Pfad erforderliche Optimierungszeit.

6.2.2 Evaluation der dynamischen Methode

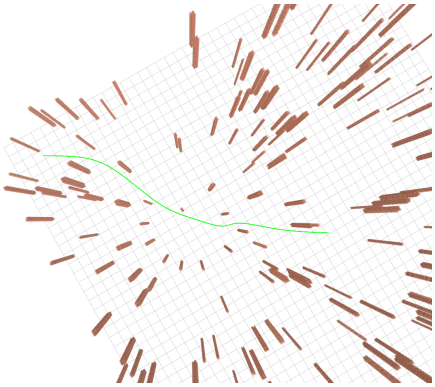
Zur Evaluation der dynamischen Methode werden der Mindestabstand zwischen den Drohnen, die Rechenzeiten, die Anzahl der Drohnen und die Hindernisdichte betrachtet. Wie in Kapitel 5.3.5 erläutert, werden Dummy-Agenten verwendet, um die sequenzielle Optimierungseffizienz zu verbessern. Deswegen ist der Stapel N_b ein wichtiger Faktor für dynamische Methode. Wenn es N Drohnen zur Pfadplanung gibt und der Stapel gleich N_b ist, beträgt die Anzahl der Pfade für jede Runde zur Optimierung $\frac{N}{N_b}$. Die Effizienz der Algorithmusoptimierung kann durch Ändern der Größe von N_b bewertet werden. Abbildung 6.12 zeigt die Rechenzeit der Pfadplanung für $N = 16, 32, 64, 128$ Drohnen in



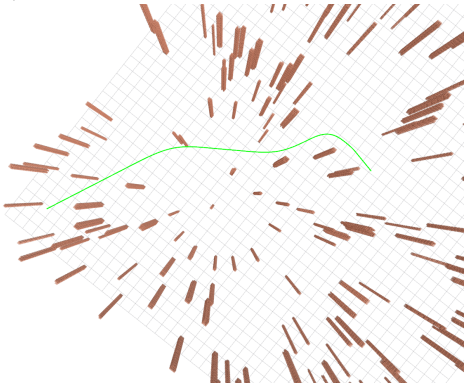
a) $smoothSteps = 1, pathMinChange = 0.001$



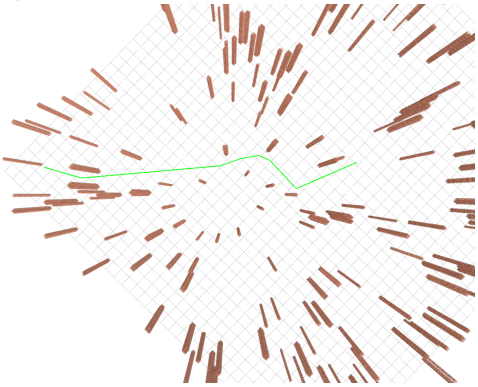
b) $smoothSteps = 5, pathMinChange = 0.001$



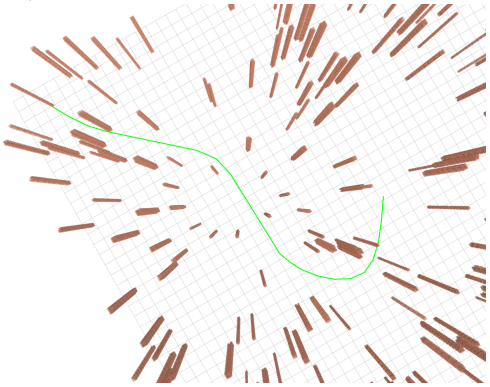
c) $smoothSteps = 10, pathMinChange = 0.001$



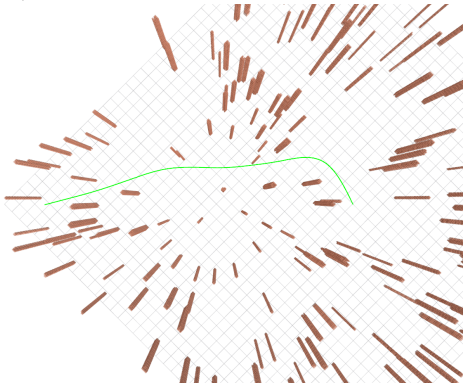
d) $smoothSteps = 50, pathMinChange = 0.001$



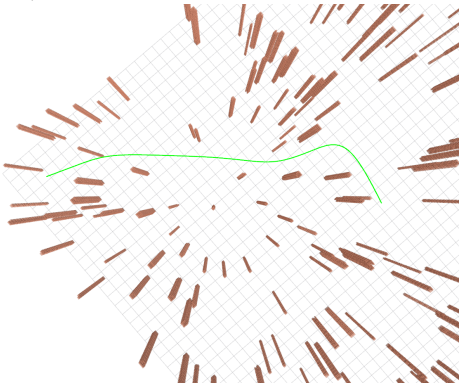
e) $smoothSteps = 5, pathMinChange = 0.5$



f) $smoothSteps = 5, pathMinChange = 0.1$



g) $smoothSteps = 5, pathMinChange = 0.01$



h) $smoothSteps = 5, pathMinChange = 0.001$

Abbildung 6.9: Optimierungszeit für 8 Drohnen mit B-Spline

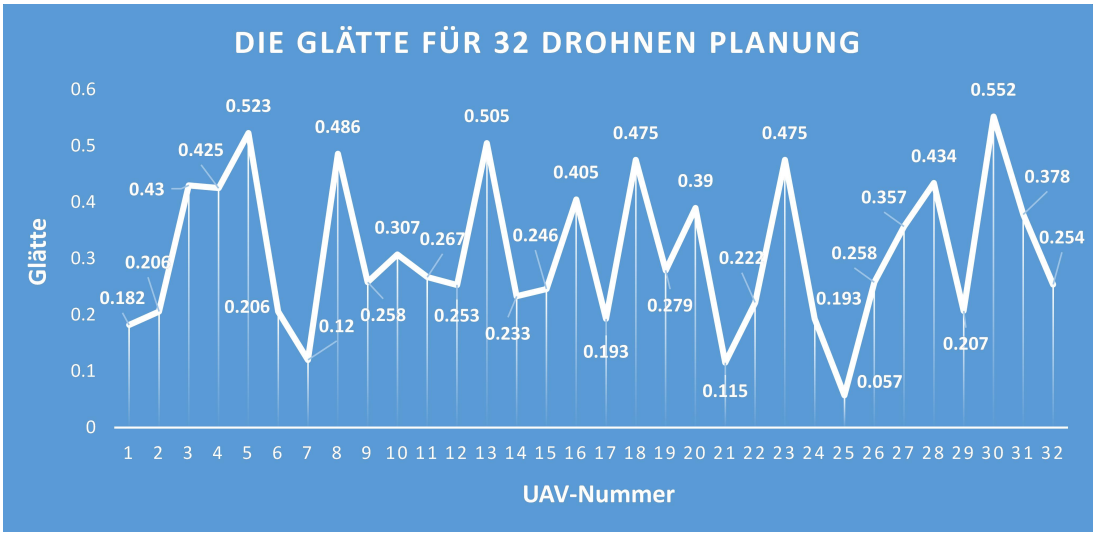


Abbildung 6.10: Die Glätte von 32 Drohnen Pfade

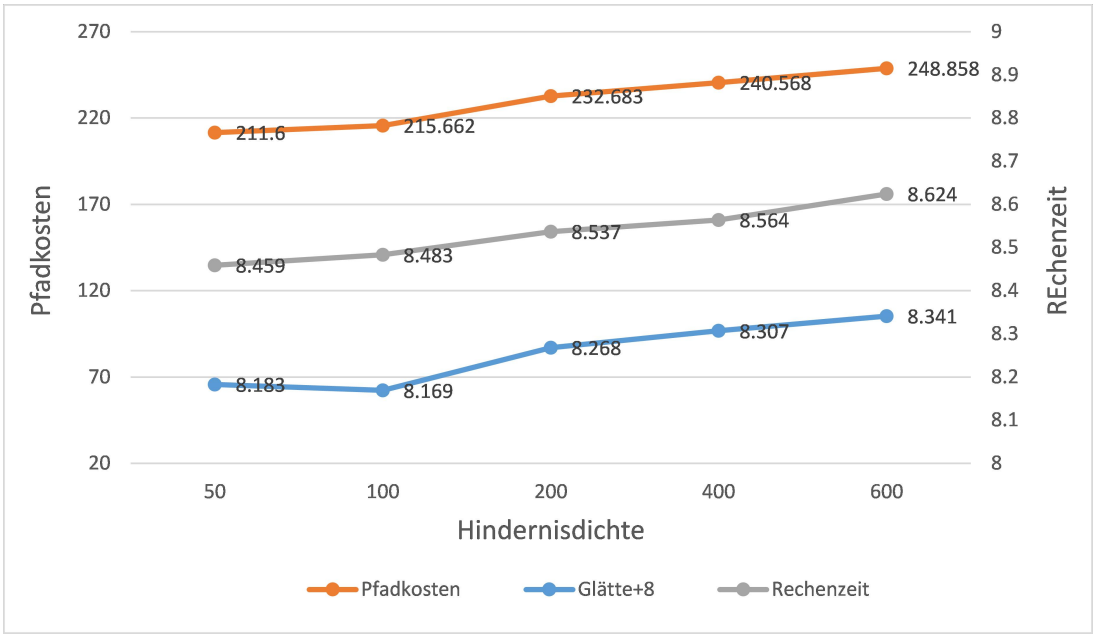
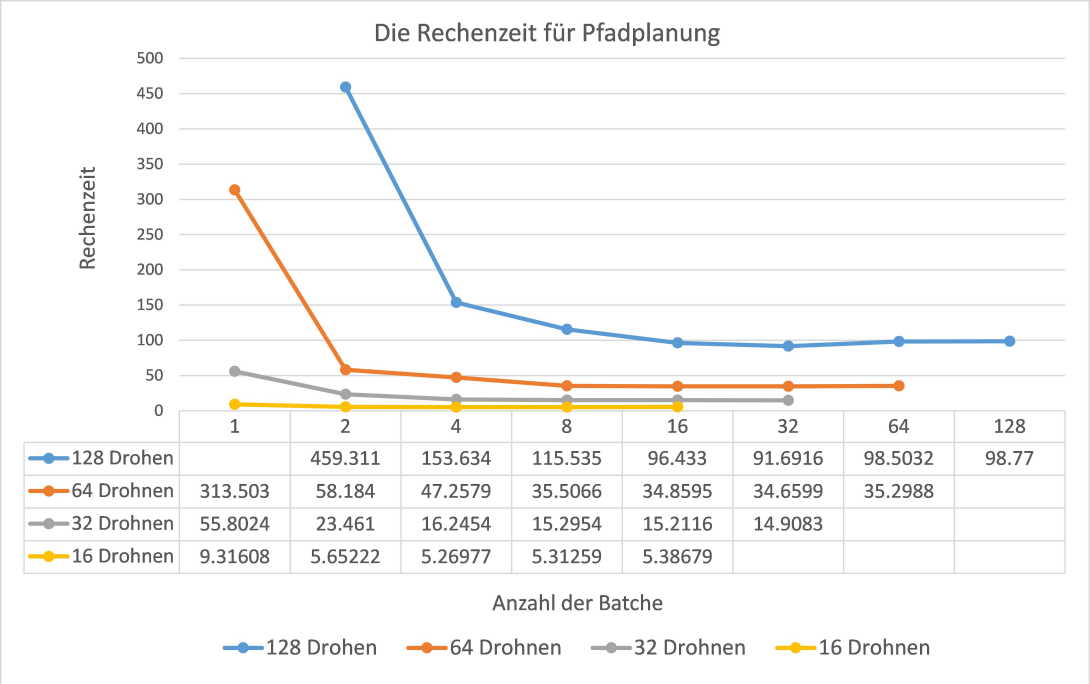


Abbildung 6.11: Evaluation zur statischen Methode in Abhängigkeit von Hindernisdichte

Abhängigkeit von N_b. Wenn $N_b = 1$ beträgt, wird die sequenzielle Optimierungsmethode nicht angewendet und alle Pfade werden gleichzeitig optimiert. In diesem Fall ergeben sich jeweils maximale Rechenzeiten für $N = 16, 32, 64, 128$ Drohnen. Wenn N_b von 1 auf 2 wechselt, wenn also die sequenzielle Optimierungsmethode zur Anwendung kommt, nimmt die Rechenzeit stark ab. Wenn N_b von 2 weiter steigt, nimmt die Rechenzeit langsam ab. Wenn $N_b = N$, erhöht die Rechenzeit sich wieder, denn in dieser Situation ist die Optimierungszeit jedes Schritts zwar sehr kurz, aber die Anzahl der Optimierungsschritte ist zu hoch, und das Produkt der beiden steigt. Mit anderen Worten, in diesem Fall ist der dominierende Faktor für die endgültige Rechenzeit nicht mehr die Optimierungszeit jedes

Schrittes, sondern die Anzahl der Optimierungsschritte. Wie aus dem Diagramm hervorgeht,, eignet sich die sequenzielle Optimierung $N_b \geqslant 2$ sehr gut für eine hohe Zahl von Drohnen ($N = 64, 128, \ldots$). In folgender Simulation wird der Stapel N_b auf $\frac{N}{N_b} = 4$ gestellt, was bedeutet, dass vier Pfade in einem Stapel optimiert werden. Abbildung 6.13 fasst die



	1	2	4	8	16	32	64	128
128 Drohen		459.311	153.634	115.535	96.433	91.6916	98.5032	98.77
64 Drohnen	313.503	58.184	47.2579	35.5066	34.8595	34.6599	35.2988	
32 Drohnen	55.8024	23.461	16.2454	15.2954	15.2116	14.9083		
16 Drohnen	9.31608	5.65222	5.26977	5.31259	5.38679			

Abbildung 6.12: Rechenzeit für Pfadplanung in Abhängigkeit von N_b, 200 Hindernissen

Beziehung zwischen der Rechenzeit und der Anzahl der Drohnen (2, 4, 8, 16, 32, 64, 128) zusammen. Mit der Verdoppelung der Anzahl von Drohnen erhöht sich die Rechenzeit schnell. Abbildung 6.14 stellt die Rechenzeit, den Mindestabstand und die Pfadkosten von 64 Drohnen in Abhängigkeit zur Hindernisdichte (50 bis 600 Hindernisse) dar. Der Mindestabstand ist der kleinste Abstand zwischen Drohnen während ihres Fluges gemäß dem geplanten Pfad. Die Pfadkosten sind die Summe der Pfadlängen aller 64 Drohnen. Es ist deutlich zu erkennen,dass dass in der Karte mit zunehmender Dichte von Hindernissen der Mindestabstand zwischen Drohnen über 1 Meter bleibt. Mit zunehmender Anzahl von Hindernissen erhöhen sich die Pfadkosten und die Rechenzeit entsprechend. Abbildung 6.15 zeigt den Pfadplanungs- und Simulationsprozess in der dynamischen Methode für 16 Drohnen. Die Startpunkte und Endpunkte sind auf $(0, 0, 5), (0, 0, 6), (2, 0, 5), (2, 0, 6), \ldots$ und $(16, 16, 5), (16, 16, 6), (16, 0, 5), (16, 0, 6), \ldots$ eingestellt. Die farbigen Kugeln repräsentieren die Drohnen und die Größe der Kugeln repräsentiert die geometrische Größe der Drohne. Zur Simulation der dynamischen Methode wird die maximale Geschwindigkeit der Drohne auf 2m/s eingestellt.

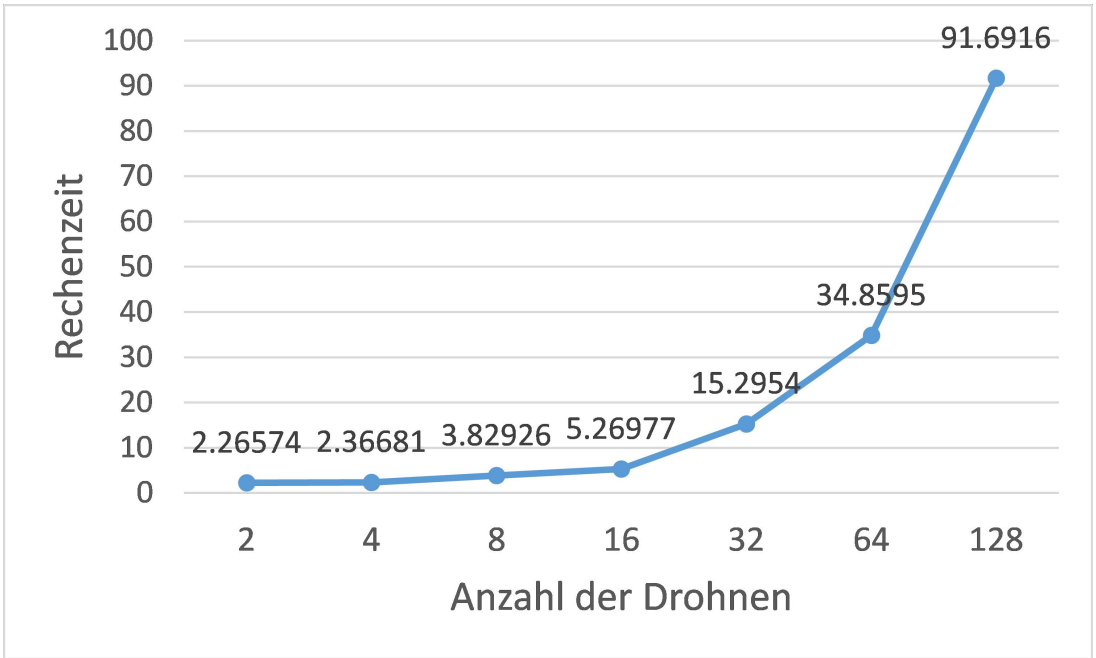


Abbildung 6.13: Rechenzeit für Pfadplanung in Abhängigkeit von der Anzahl der Drohnen

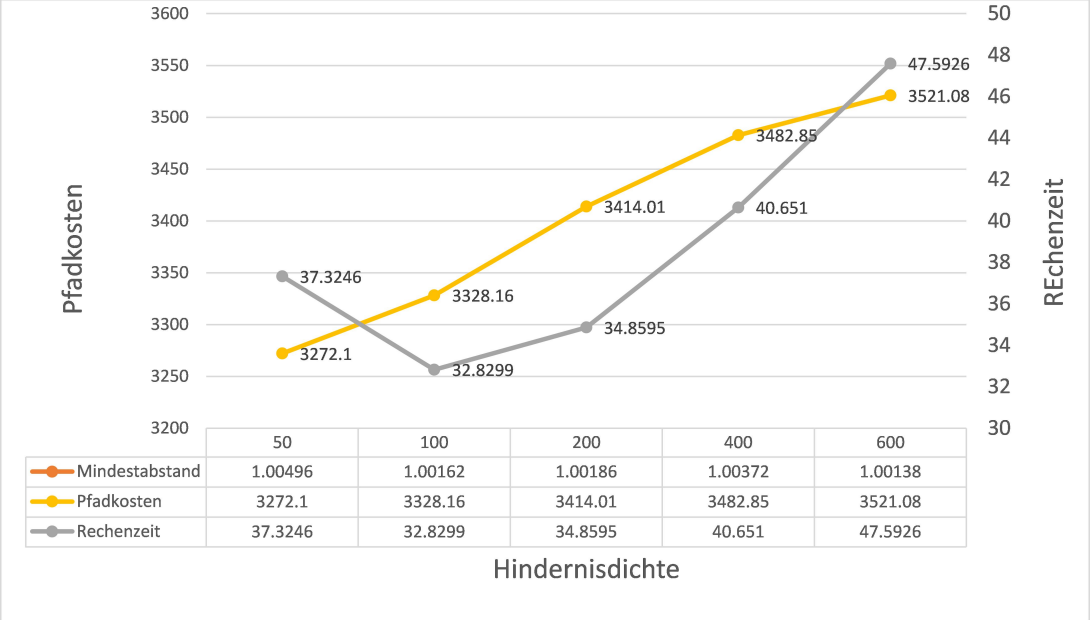


Abbildung 6.14: Rechenzeit und Pfadkosten für Pfadplanung (64 Drohnen) in Abhängigkeit von der Hindernisdichte

6.2.3 Bewertung der beiden Methoden

Im Folgenden werden die beiden in diesem Artikel untersuchten Methoden unter verschiedenen Aspekten verglichen, z. B. Pfadkosten, Rechenzeit und Hindernisdichte. Anschließend werden die Ergebnisse der beiden Planungsmethoden anhand einiger spezieller Karten analysiert. Abschließend werden die Vor- und Nachteile der beiden Methoden

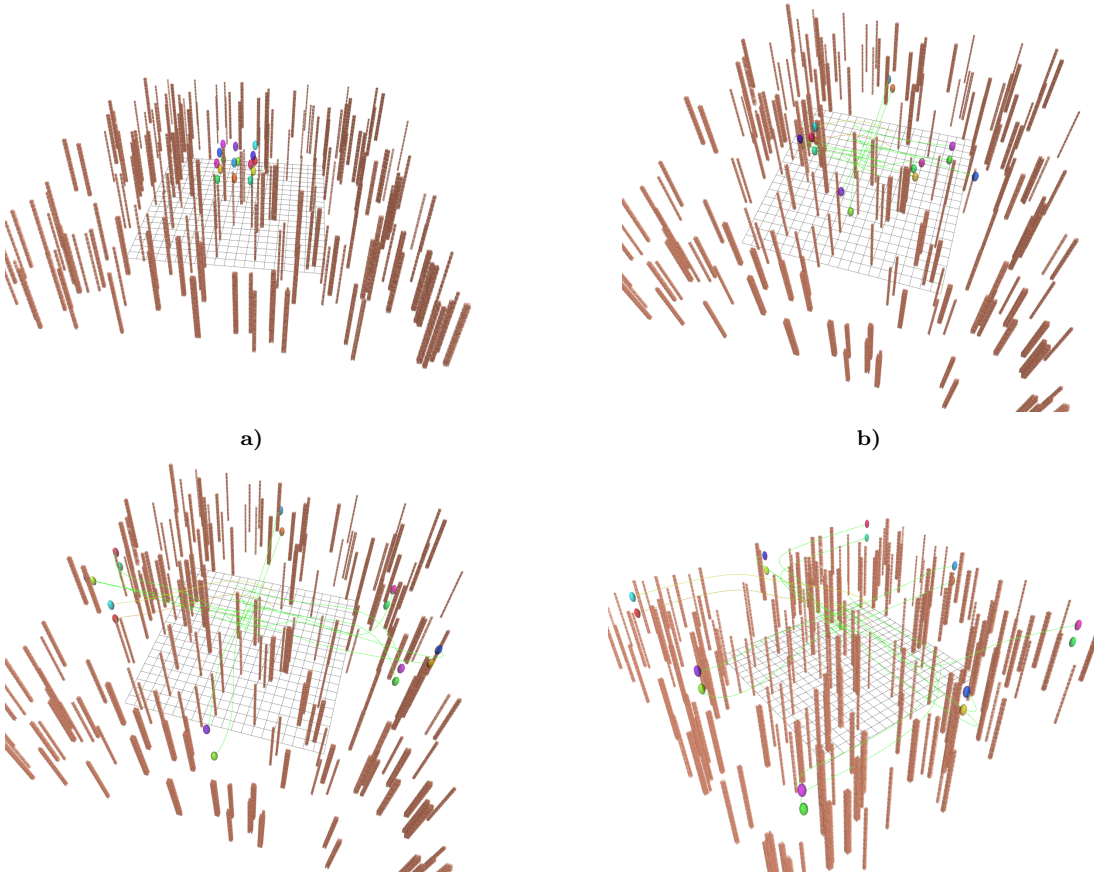
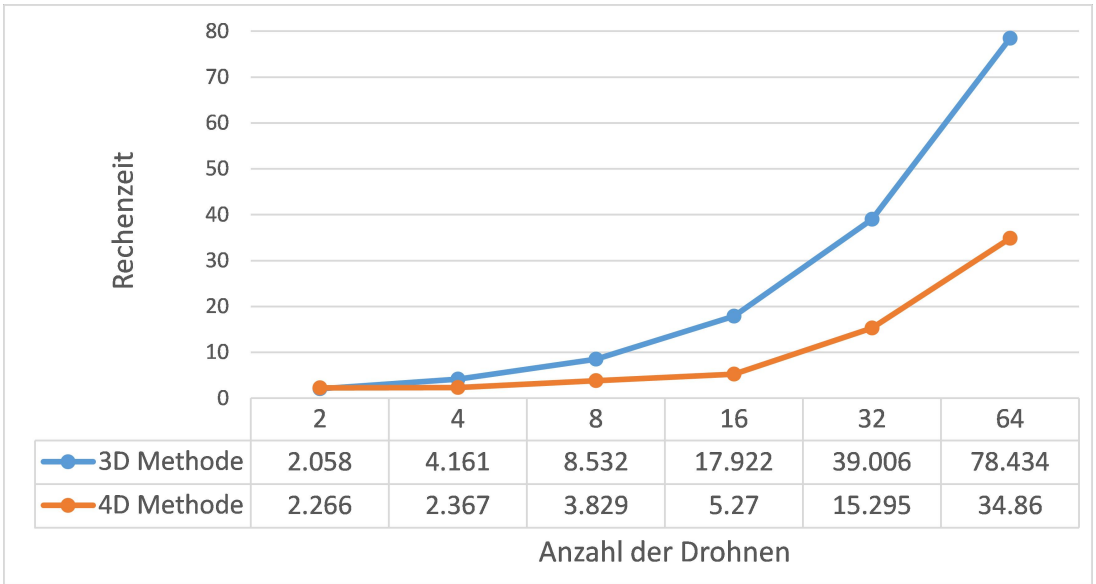


Abbildung 6.15: Pfadplanung und Simulation für 16 Drohnen

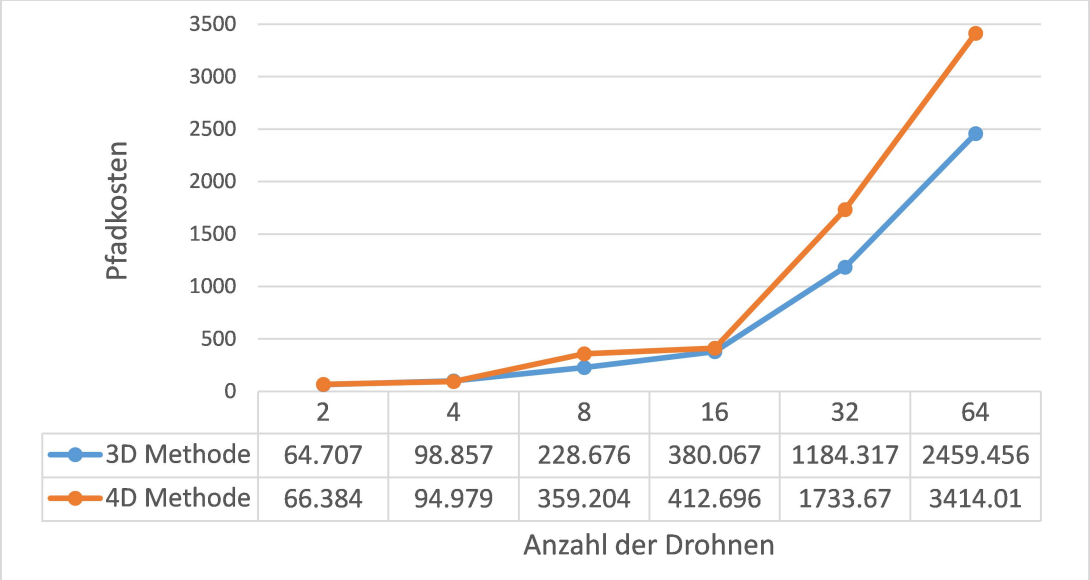
zusammengefasst. Zur Evaluation der beiden Methoden basieren alle Simulationsdaten auf derselben Karte mit demselben Start- und Endpunkt. Abbildungen 6.16 und 6.17 zeigen die Rechenzeit und gesamten Pfadkosten zunehmender Anzahl an Drohnen unter Verwendung der beiden Methoden. Aus der Grafik 6.16 ist es deutlich, dass die Berechnungseffizienz der dynamischen Methode(4D) höher als die der statischen Methode(3D) ist, insbesondere wenn die Anzahl der Drohnen relativ groß ist. Das liegt daran, dass die statische Methode(3D) eine sequenzielle Planung verwendet und für jede Drohne eine feste Rechenzeit braucht. Wenn die Anzahl der Drohnen exponentiell zunimmt, nimmt auch die Rechenzeit exponentiell zu. Im Gengensatz zur 3D-Methode verwendet die 4D-Methode eine globale Planung, die auf dem ECBS-Algorithmus und dem MAPF-Problem basiert. Außerdem werden die Trajektorien aller Drohnen in N_b Stapeln optimiert, sodass die Rechenzeit mit exponentiellem Wachstum der Anzahl der Drohnen einen linearen Wachstumstrend zeigt. Abbildung 6.17 bildet die Pfadkosten der beiden Methoden mit zunehmender Anzahl von Drohnen ab. Wenn die Anzahl der Drohnen nicht groß ist, $N \leq 16$, unterscheiden sich die durch die beiden Methoden erzeugten Pfadkosten nicht wesentlich. Wenn jedoch die Anzahl der Drohnen groß ist, $N > 16$, weist die statische

Methode Vorteile auf. Die erzeugte Pfadlänge ist geringer als die der 4D-Methode, und die relative Differenz vergrößert sich mit zunehmender Anzahl von Drohnen. Das liegt daran, dass die 3D-Methode versucht, für jede Drohne einen suboptimalen Pfad in der Karte zu finden. Die dynamische Methode ist darauf ausgerichtet, eine praktikable Lösung für alle Drohnen zu finden. Zur Evaluation der beiden Methoden in Abhängigkeit von



	2	4	8	16	32	64
3D Methode	2.058	4.161	8.532	17.922	39.006	78.434
4D Methode	2.266	2.367	3.829	5.27	15.295	34.86

Abbildung 6.16: Vergleich der Rechenzeit, 200 Hindernisse in der Karte



	2	4	8	16	32	64
3D Methode	64.707	98.857	228.676	380.067	1184.317	2459.456
4D Methode	66.384	94.979	359.204	412.696	1733.67	3414.01

Abbildung 6.17: Vergleich der Pfadkosten, 200 Hindernisse in der Karte

der Hindernisdichte dienen Abbildungen 6.18 und 6.19. Nimmt die Anzahl der Hindernisse auf der Karte zu, steigen deutlich erkennbar bei meiden Methoden die Pfadkosten und Rechenzeit. Die Wachstumsrate der Rechenzeit ist bei der 4D-Methode höher als bei

der 3D-Methode. Dies zeigt auch, dass die Anpassungsfähigkeit der 4D-Methode an die Umgebung mit dichten Hindernissen nicht so gut wie die der 3D-Methode ist.

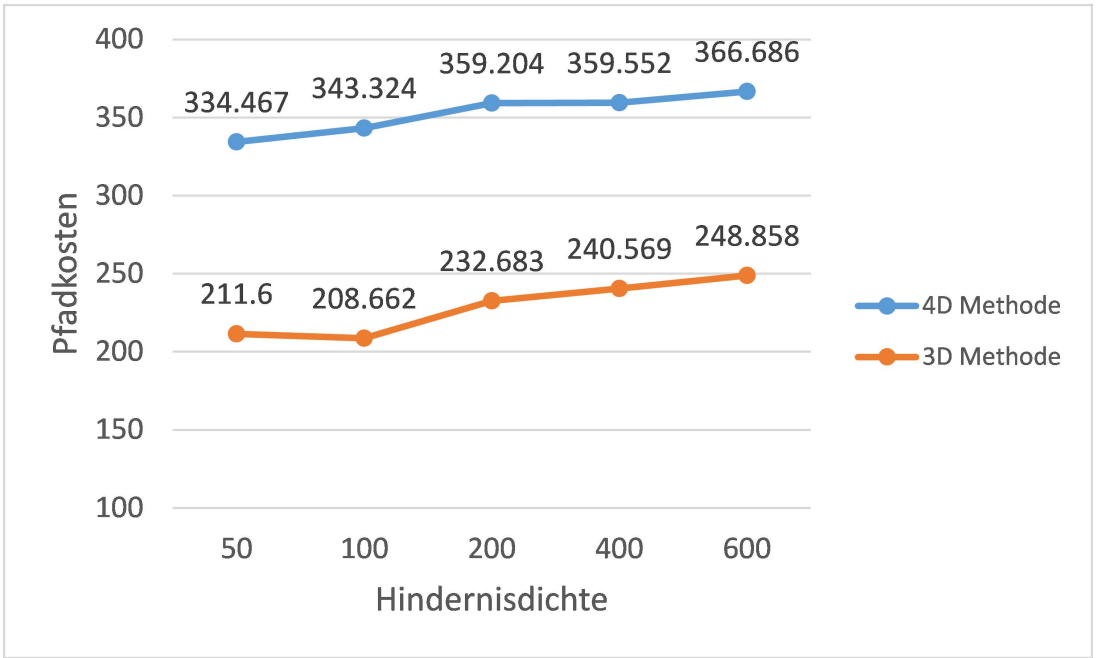


Abbildung 6.18: Vergleich der Pfadkosten in Abhängigkeit von Hindernisdichte

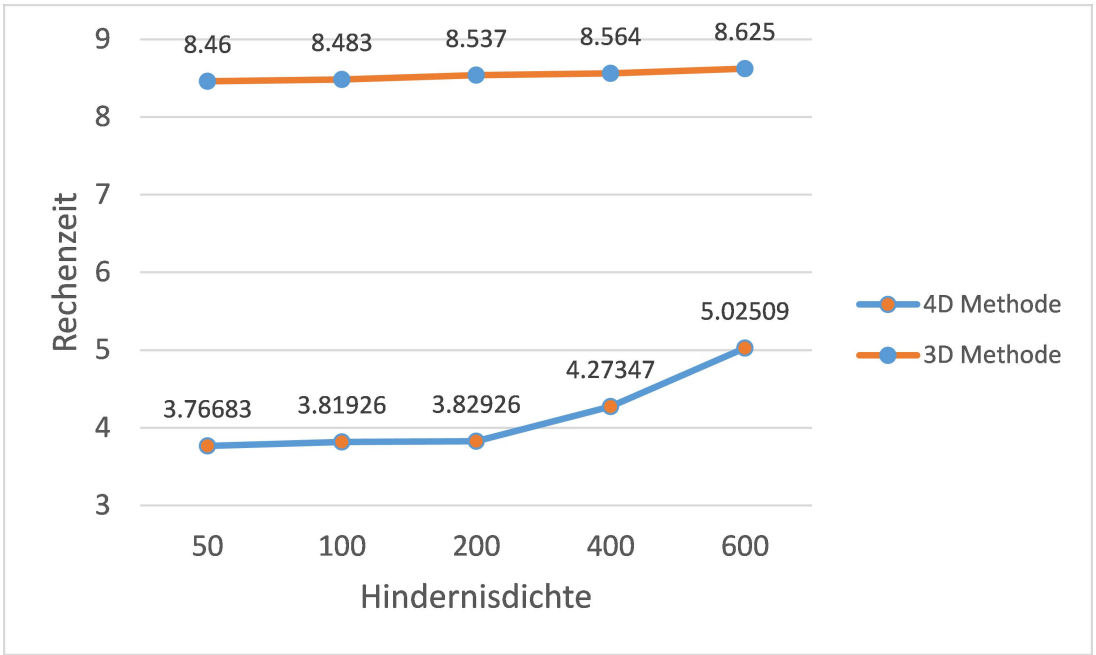
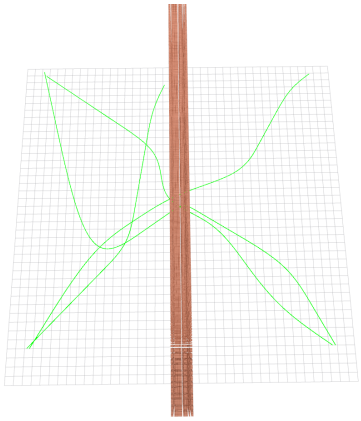


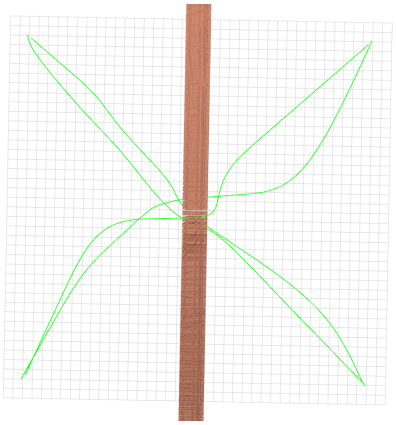
Abbildung 6.19: Vergleich der Rechenzeit in Abhängigkeit von Hindernisdichte

Zur Bewertung der Robustheit der beiden Methoden dienen die Karten in Abbildungen 6.20. In der Mitte des dargestellten Raumes liegt eine Wand (Länge: 40 m, Höhe: 10 m, Breite: 2 m) mit einem Fenster (Länge: 2 m, Höhe: 2 m, Breite: 2 m), die den Raum

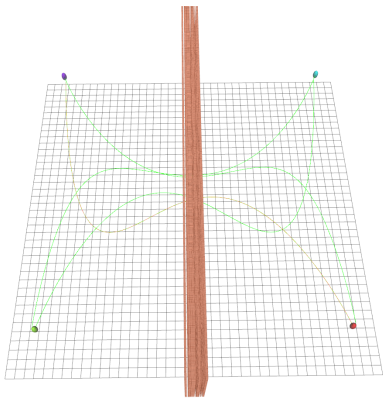
in zwei Hälften teilt. Vier Drohnen können nur durch das Fenster von einer Seite des Raumes auf die andere fliegen. Die Abbildung a) zeigt, dass bei einem Zeitlimit von eine Sekunde die 3D-Methode nur drei Pfade finden kann. Ein Pfad scheitert, da er nicht durch das Fenster führt. Bei einem Zeitlimit von drei Sekunden finden alle vier Drohnen einen Pfad. Im Gegensatz dazu werden vier mögliche Pfade bei der 4D-Methode bereits nach 2,48 Sekunden gefunden. Mit abnehmender Größe des Fensters nehmen in beiden Methoden die erforderliche Berechnungszeit zur Lösungsfindung sowie die Ausfallrate zu. Ist die Fläche des Fensters kleiner als die Querschnittsfläche, die vier Drohnen gleichzeitig aufnehmen kann, funktioniert die 3D-Methode nicht mehr. Die 4D-Methode kann jedoch so lange wirksam bleiben, bis die Fläche des Fensters kleiner als die Fläche ist, die eine einzelne Drohne benötigt. Wegen der zeitlichen Anpassung hat die 4D-Methode in extrem engen Kanälen eine höhere Erfolgsrate als die 3D-Methode.



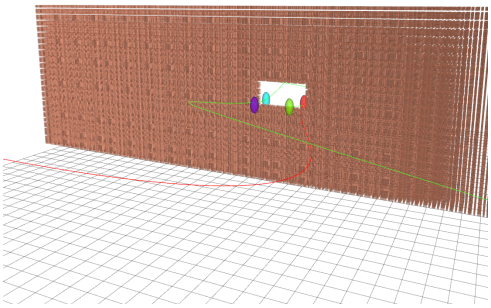
a) Pfadplanung für vier Drohnen mit 3D-Methode, Zeitlimit 1 Sekunde



b) Pfadplanung für vier Drohnen mit 3D-Methode, Zeitlimit 3 Sekunden, Pfadkosten = 212.559



c) Pfadplanung für vier Drohnen mit 4D-Methode, Pfadkosten = 217.888, Draufsicht



d) Pfadplanung für vier Drohnen mit 4D-Methode, Seitenansicht

Abbildung 6.20: Pfadplanung für vier Drohnen mit 3D und 4D Methode

Aus der Gegenüberstellung der zwei Methoden lassen sich folgende Schlussfolgerungen ziehen:

- Vorteile der 3D-Methode:

- Pfadkosten: Es können suboptimale Pfade für Drohnen gefunden werden. Mit zunehmender Planungszeit können kürzere Pfade berechnet werden.
- Sicherheit: Keine geplanten Pfade kreuzen sich, was den Sicherheitsfaktor der Drohnen während des Fluges verbessert.
- Rechenaufwand: Die Zeitdimension muss nicht berücksichtigt werden, was den Rechenaufwand verringert.
- Kinematische Anforderungen: Keine Anforderung für Geschwindigkeit und Beschleunigung jeder Drohne.

- Nachteile der 3D-Methode:
  - Rechenzeit: Bei hoher Anzahl von Drohnen ($N \geq 32$) ist die erforderliche Rechenzeit deutlich höher als unter der 4D-Methode.
  - Raumnutzung: Da keine Überschneidung von Pfaden zulässig ist, ist die Raumnutzungsrate auf der Karte niedrig.
- Vorteile der 4D-Methode:
  - Rechenzeit: Bei exponentiell steigender Anzahl der Drohnen erhöht sich die erforderliche Planungszeit nur linear.
  - Raumnutzung: Durch zeitliche Anpassungen verbessert diese 4D-Methode die Raumnutzungsrate.
- Nachteile der 4D-Methode:
  - Pfadkosten: Bei hoher Zahl von Drohnen ($N \geq 16$) sind die Pfadkosten höher als in der 3D-Methode.
  - Sicherheit: Die Zeit aller Drohnenflüge muss synchronisiert werden. Andernfalls besteht die Gefahr einer Kollision.
  - Kinematische Anforderung: Alle Drohnen haben die gleiche Geschwindigkeit.
- Anwendungsintegration:
  - Die 3D-Methode eignet sich für die Pfadplanung mit großer Entfernung und niedriger Hindernisdichte, z. B. bei der Warenlieferung.
  - Die 4D-Methode eignet sich für die Pfadplanung bei geringer Entfernung und hohe Hindernisdichte, z. B. für die unternehmensinterne Logistik.

7 Zusammenfassung und Ausblick

Im Folgenden werden zunächst die Ergebnisse dieser Arbeit als Antworten auf die zu Beginn als Schwerpunkte aufgeworfenen Fragen vorgestellt. Anschließend werden Möglichkeiten zur Weiterentwicklung der Ergebnisse aufgezeigt. Zusätzlich werden Ansätze vorgestellt, mit denen die im Teilprojekt (Intrafly) definierten Forschungsaspekte weiterverfolgt werden können.

7.1 Fazit

Im Rahmen dieser Arbeit wurde ein Softwaremodul zur dreidimensionalen Routenplanung einer variablen Anzahl von Flugrobotern entwickelt. Vorgegeben war dabei stets eine Umgebungskarte in Form einer OctoMap sowie die Start- und Zielpunkte der einzelnen Roboter. Zur Implementierung der vorgestellten Software wurden das Robot Operating System (ROS), die Softwarebibliotheken FCL und MAVLink sowie die Optimierungslösung Cplex verwendet, wie in Kapitel 3 erläutert. Zur Simulation wurden der Gazebo-Simulator und die PX4-Flugsteuerung vorgestellt. In Kapitel 4 wurden die in diesem Artikel verwendete Algorithmen dargestellt. Der RRT-Algorithmus und die von diesem abgeleiteten Algorithmen RRT* und informierter RRT* wurden für die Pfadplanung einer einzelnen Drohne verwendet. Im Gegensatz dazu dienten die Algorithmen A*, CBS und ECBS zur Pfadplanung für mehrere Drohnen.

Basierend auf der Erweiterung dieser beiden Arten von Algorithmen wurden zwei Methoden für die Planung kollisionsfreier sowie nach Flugstrecke optimierter roboterspezifischer Trajektorien vorgestellt: eine statische 3D-Methode und eine dynamische 4D-Methode. Zur Optimierung der Pfade dienten die am Ende von Kapitel 4 erläuterten Grundlagen der Bernsteinpolynome. Die entwickelte und in Kapitel 5 beschriebene Verarbeitungsarchitektur der beiden Methoden zeichnet sich durch einen modularen Aufbau aus, wobei die einzelnen Module unter Verwendung des ROS-Framework kommunizieren und unterschiedliche Module zusammenarbeiten. Abschließend wurde auf Basis der entwickelten Pfadplanung sowie der implementierten Verarbeitungsarchitektur ein Demonstrator aufgebaut und getestet, beschrieben in Kapitel 6. Die von einer 3D-Belegungskarte erfassten Umgebungsdaten bilden hierbei die Eingangsdaten für die beide Methoden, die es den Drohnen ermöglichen, Hindernissen im Raum dynamisch auszuweichen und Kollisionen

zwischen Drohnen zu vermeiden. Zusätzlich wurden die aus dem Pfadplaner erstellten Trajektorien über die Steuerung eines Trajektorienfolgers an den Simulator gesendet. Die Evaluation der beiden Methoden konzentrierte sich auf die Pfadkosten, die Rechenzeit, die Robustheit und die Hindernisdichte. Zum Schluss wurden die Anwendungsintegration sowie die Vor- und Nachteile der beiden Methoden zusammengefasst. Grundsätzlich sind die in dieser Arbeit entwickelten Methoden geeignet zur Lösung des MAPF-Problems und zur Trajektorienoptimierung. Die Evaluation zeigt die Leistungsfähigkeit der zwei Methoden in unterschiedlichen Karten und mit einer undefinierten Anzahl von Drohnen.

7.2 Ausblick

Nachfolgend werden zunächst die aktuellen Einschränkungen sowie die Verbesserungs- und Entwicklungspotenziale des im Rahmen der vorliegenden Arbeit realisierten Konzepts herausgearbeitet. Anschließend sollen Möglichkeiten vorgestellt werden, um den Pfadplaner unter Verwendung weiterer Optimierungsmethoden, zusätzlicher Daten und Informationen sowie weiterführender Algorithmen zu verbessern und für die industrielle Anwendung zu ertüchtigen.

7.2.1 Aktuelle Einschränkungen

Die Bewertung der beiden in diesem Artikel genannten Methoden befindet sich jedoch erst in der Simulationsphase. Für die reale Flugversuche müssen noch die folgende Punkte beachtet werden.

- Kartenkonstruktion: In dieser Arbeit werden die Umgebungen mit einer Belegungskarte modelliert, und die Hindernisse in der Karte werden durch Quader dargestellt. Die Kollisionserkennung in den Algorithmen basiert ebenfalls auf der Belegungskarte. In realen Umgebungen sind die Formen von Hindernissen komplexer. Deswegen ist eine Kartenkonstruktion mit höherer Genauigkeit für reale Flugversuche erforderlich.
- Trajektorienfolger: In Simulationen werden ideale Umgebungen betrachtet. In tatsächlichen Umgebungen müssen auch Windwiderstand, die Kommunikationslatenz zwischen Drohne und Host sowie Sensormessfehler berücksichtigt werden. Der Trajektorienfolger in dieser Arbeit ist nur für die Simulationsphase geeignet; für reale Flugversuche bedarf es eines Trajektorienfolgers mit höherer Genauigkeit.
- Zeitsynchronisation: Die 4D-Methode basiert auf zeitlicher Synchronisation, was bedeutet, dass die Zeitbeschriftungen aller Drohnen konsistent sein müssen. Bei

realen Flugversuchen ist es schwierig, sicherzustellen, dass die Zeit aller Drohnen synchronisiert wurde.

- Anzahl der Drohnen: Die 3D-Methode hat eine Begrenzung für die Anzahl der Drohnen.

7.2.2 Weiterentwicklung der Methoden

Für eine Weiterentwicklung des vorgestellten Moduls empfiehlt sich eine Verbesserung des Algorithmus zur MAPF-Problemlösung mit einer besseren Raumnutzung. Dabei sollten möglichst alle entwickelten Kollisionsvermeidungs- und Datenweiterleitungsmethoden untersucht werden. Ein Vorschlag für die Verbesserung der Raumnutzung der 3D-Methode besteht im Modellieren einer dynamischen Belegungskarte. Dies bedeutet, dass für alle Kuben in der Karte nicht nur die Ortsbezeichnungen mit dreidimensionalen Koordinaten, sondern auch die Zeitbezeichnungen verwendet werden, so dass der Pfadplaner die Positionen und die Zeitpunkte der Kuben abfragen kann, um festzustellen, ob der Kubus in diesem Zeitpunkt frei ist. Auch besteht die Möglichkeit, die 3D-Methode und 4D-Methode zu integrieren, um die Vorteile beider Methoden zu nutzen und die Nachteile zu eliminieren. Für die globale Planung würde dabei die 3D-Methode zuerst ausgeführt. Die 4D-Methode würde aufgerufen, um lokale Kollisionen zu vermeiden. Eine Integration dieser beiden Methoden klingt perfekt, aber die Synchronisation der Zeitdimension wäre immer noch eine große Herausforderung. Um die Flexibilität des Algorithmus zu verbessern, ließe sich die Methode eventuell um zusätzliche Entscheidungsoptionen für die Agenten erweitern, damit die Drohnen nicht nur durch Bewegung Kollisionen vermeiden, sondern auch schweben, warten und ihren Flug verzögern können.

Sollte es in Zukunft möglich sein, autonome Flugroboter mit einer den industriellen Normen entsprechenden Flugsteuerung auszustatten und auch die Software allezeit sicher und redundant zu gestalten, werden sich in den nächsten Jahren vor allem in Industrieunternehmen vielfältige Einsatzbereiche herausstellen. Vor allem hinsichtlich des wachsenden Digitalisierungsbewusstseins, aber auch im Hinblick auf Effizienzvorteile und ein sichereres Arbeitsumfeld, werden autonom fliegende Multikopterflotten in zukünftigen Produktions- und Lagerstätten großen Einfluss haben. Sie bieten insbesondere durch ihre flexible Einsetzbarkeit, die Nutzung bisher ungenutzter Lufträume und die damit verbundene Schnelligkeit deutliche Vorteile gegenüber bisher eingesetzten Technologien. Vor allem in einem vollständig vernetzten Internet der Dinge kann ihr gesamtes Potenzial optimal genutzt werden. So wird es in Zukunft möglich sein, Produkte nicht nur deutlich variabler und auf den Kunden hin angepasst zu produzieren, sondern auch gesamte Fertigungslinien durch geringere Produktions-, Stillstands- und Transportzeiten effizienter und schlanker

zu gestalten. Dadurch ließe sich der Ressourceneinsatz reduzieren und so die Fertigung industrieller Güter nachhaltiger gestalten. Zusätzlich würde die Arbeitsumgebung für den Menschen sicherer und bequemer.

Literaturverzeichnis

[1] Garrett-Glaser, Brian: *Amazon Seeks FAA Approval for Prime Air Drone Delivery*, [EB/OL], https://www.aviationtoday.com/2019/08/09/following-wing-ups-amazon-seeks-approval-prime-air-drone-delivery/ Accessed August 9, 2019.

[2] DHL: *DHL launches its first regular fully-automated and intelligent urban drone delivery service*, [EB/OL], https://www.dpdhl.com/en/media-relations/press-releases/2019/dhl-launches-its-first-regular-fully-automated-and-intelligent-urban-drone-del.html Accessed May 16, 2019.

[3] Team, ArduPilot Dev: *MAVLink Basics*, [EB/OL], https://ardupilot.org/dev/docs/mavlink-basics.html Accessed Juli 11, 2020.

[4] Lim, Jaeyoung: *PX4 Offboard Control Using MAVROS on ROS*, [EB/OL], https://www.codetd.com/article/2918485 Accessed August 28, 2018.

[5] Dronecode: *PX4 Architectural Overview*, [EB/OL], https://dev.px4.io/v1.9.0/en/concept/architecture.html Accessed August 28, 2019.

[6] Dronecode: *ROS with Gazebo Simulation*, [EB/OL], https://dev.px4.io/v1.9.0/en/simulation/ros_interface.html Accessed August 28, 2019.

[7] Hornung, Armin; Wurm, Kai M; Bennewitz, Maren; Stachniss, Cyrill; Burgard, Wolfram: *OctoMap: An efficient probabilistic 3D mapping framework based on octrees*, in Autonomous robots, 2013, Vol. 34 (3), S. 189–206.

[8] Pan, Jia; Chitta, Sachin; Manocha, Dinesh: *FCL: A general purpose library for collision and proximity queries*, in , 2012, S. 3859–3866.

[9] Yuan, Zhenyuan: *Motion Planning–Rapidly-exploring Random Tree (RRT)*, [EB/OL], https://sites.psu.edu/zqy5086/project-1/ Accessed April 4, 2020.

[10] Karaman, Sertac; Frazzoli, Emilio: *Incremental sampling-based algorithms for optimal motion planning*, in Robotics Science and Systems VI, 2010, Vol. 104 (2).

[11] Gammell, Jonathan D; Srinivasa, Siddhartha S; Barfoot, Timothy D: *Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic*, in , 2014, S. 2997–3004.

[12] Peter, Hart: *Motion Planning–Rapidly-exploring Random Tree (RRT)*, [EB/OL], `https://de.wikipedia.org/wiki/A*-Algorithmus` Accessed April 4, 2020.

[13] Sharon, Guni; Stern, Roni; Felner, Ariel; Sturtevant, Nathan R: *Conflict-based search for optimal multi-agent pathfinding*, in Artificial Intelligence, 2015, Vol. 219, S. 40–66.

[14] Park, Jungwon; Kim, H Jin: *Fast Trajectory Planning for Multiple Quadrotors using Relative Safe Flight Corridor*, in arXiv preprint arXiv:1909.02896, 2019.

[15] Park, Jungwon; Kim, Junha; Jang, Inkyu; Kim, H Jin: *Efficient Multi-Agent Trajectory Planning with Feasibility Guarantee using Relative Bernstein Polynomial*, in arXiv preprint arXiv:1909.10219, 2019.

[16] Hii, Michelle Sing Yee; Courtney, Patrick; Royall, Paul G: *An evaluation of the delivery of medicines using drones*, in Drones, 2019, Vol. 3 (3), S. 52.

[17] Jung, Sunghun; Kim, Hyunsu: *Analysis of amazon prime air uav delivery service*, in Journal of Knowledge Information Technology and Systems, 2017, Vol. 12 (2), S. 253–266.

[18] Bamburry, Dane: *Drones: Designed for product delivery*, in Design Management Review, 2015, Vol. 26 (1), S. 40–48.

[19] Park, Jiyoon; Kim, Solhee; Suh, Kyo: *A comparative analysis of the environmental benefits of drone-based delivery services in urban and rural areas*, in Sustainability, 2018, Vol. 10 (3), S. 888.

[20] Aurambout, Jean-Philippe; Gkoumas, Konstantinos; Ciuffo, Biagio: *Last mile delivery by drones: an estimation of viable market potential and access to citizens across European cities*, in European Transport Research Review, 2019, Vol. 11 (1), S. 30.

[21] Vlahovic, Nikola; Knezevic, Blazenka; Batalic, Petra: *Implementing delivery drones in logistics business process: Case of pharmaceutical industry*, in World Academy of Science, Engineering and Technology, International Journal of Social, Behavioral, Educational, Economic, Business and Industrial Engineering, 2016, Vol. 10 (12), S. 3981–3986.

[22] Yu, Jingjin; LaValle, Steven M: *Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics*, in IEEE Transactions on Robotics, 2016, Vol. 32 (5), S. 1163–1177.

[23] Čáp, Michal; Novák, Peter; Vokřínek, Jiří; Pěchouček, Michal: *Multi-agent RRT*: Sampling-based cooperative pathfinding*, in arXiv preprint arXiv:1302.2828, 2013.

[24] Hönig, Wolfgang; Preiss, James A; Kumar, TK Satish; Sukhatme, Gaurav S; Ayanian, Nora: *Trajectory planning for quadrotor swarms*, in IEEE Transactions on Robotics, 2018, Vol. 34 (4), S. 856–869.

[25] Ding, Xu Chu; Rahmani, Amir R; Egerstedt, Magnus: *Multi-UAV convoy protection: An optimal approach to path planning and coordination*, in IEEE transactions on Robotics, 2010, Vol. 26 (2), S. 256–268.

[26] Peng, Jung-Hao; Li, I-Hsum; Chien, Yi-Hsing; Hsu, Chen-Chien; Wang, Wei-Yen: *Multi-robot path planning based on improved D* Lite Algorithm*, in , 2015, S. 350–353.

[27] Boyarski, Eli; Felner, Ariel; Stern, Roni; Sharon, Guni; Tolpin, David; Betzalel, Oded; Shimony, Eyal: *ICBS: improved conflict-based search algorithm for multi-agent pathfinding*, in , 2015.

[28] Tang, Sarah; Kumar, Vijay: *Mixed integer quadratic program trajectory generation for a quadrotor with a cable-suspended payload*, in , 2015, S. 2216–2222.

[29] Mellinger, Daniel; Kushleyev, Alex; Kumar, Vijay: *Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams*, in , 2012, S. 477–483.

[30] Ritz, Robin; Müller, Mark W; Hehn, Markus; D'Andrea, Raffaello: *Cooperative quadrocopter ball throwing and catching*, in , 2012, S. 4972–4978.

[31] Robinson, D Reed; Mar, Robert T; Estabridis, Katia; Hewer, Gary: *An efficient algorithm for optimal trajectory generation for heterogeneous multi-agent systems in non-convex environments*, in IEEE Robotics and Automation Letters, 2018, Vol. 3 (2), S. 1215–1222.

[32] Bareiss, Daman; Van den Berg, Jur: *Reciprocal collision avoidance for robots with linear dynamics using LQR-obstacles*, in , 2013, S. 3847–3853.

[33] Van Den Berg, Jur; Wilkie, David; Guy, Stephen J; Niethammer, Marc; Manocha, Dinesh: *LQG-obstacles: Feedback control with collision avoidance for mobile robots with motion and sensing uncertainty*, in , 2012, S. 346–353.

[34] Zhou, Dingjiang; Wang, Zijian; Bandyopadhyay, Saptarshi; Schwager, Mac: *Fast, on-line collision avoidance for dynamic vehicles using buffered voronoi cells*, in IEEE Robotics and Automation Letters, 2017, Vol. 2 (2), S. 1047–1054.

[35] Yu, Jingjin; LaValle, Steven M: *Optimal multi-robot path planning on graphs: Structure and computational complexity*, in arXiv preprint arXiv:1507.03289, 2015.

[36] Joseph, Lentin: *Erratum to: Robot Operating System for Absolute Beginners: Robotics Programming Made Easy*, in , 2018, S. E1–E1.

[37] Foote, Tully: *ROS Documentation*, [EB/OL], `http://wiki.ros.org/Documentation` Accessed Juli 11, 2020.

[38] O'Kane, Jason M: *A gentle introduction to ROS*, 2014.

[39] Koenig, Nathan; Howard, Andrew: *Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator*, in , 2004, S. 2149–2154.

[40] Aguero, C.E.; Koenig, N.; Chen, I.; Boyer, H.; Peters, S.; Hsu, J.; Gerkey, B.; Paepcke, S.; Rivero, J.L.; Manzo, J.; Krotkov, E.; Pratt, G.: *Inside the Virtual Robotics Challenge: Simulating Real-Time Robotic Disaster Response*, in Automation Science and Engineering, IEEE Transactions on, 2015, Vol. 12 (2), S. 494–506.

[41] Moravec, Hans; Elfes, Alberto: *High resolution maps from wide angle sonar*, in , 1985, Vol. 2, S. 116–121.

[42] Yang, Liang; Qi, Juntong; Song, Dalei; Xiao, Jizhong; Han, Jianda; Xia, Yong: *Survey of robot 3D path planning algorithms*, in Journal of Control Science and Engineering, 2016, Vol. 2016.

[43] LaValle, Steven M: *Rapidly-exploring random trees: A new tool for path planning*, in , 1998.

[44] Kuffner, James J; LaValle, Steven M: *RRT-connect: An efficient approach to single-query path planning*, in , 2000, Vol. 2, S. 995–1001.

[45] Frazzoli, Emilio; Dahleh, Munther A; Feron, Eric: *Real-time motion planning for agile autonomous vehicles*, in Journal of guidance, control, and dynamics, 2002, Vol. 25 (1), S. 116–129.

[46] Connell, Devin; Manh La, Hung: *Extended rapidly exploring random tree–based dynamic path planning and replanning for mobile robots*, in International Journal of Advanced Robotic Systems, 2018, Vol. 15 (3), S. 1729881418773874.

[47] Noreen, Iram; Khan, Amna; Habib, Zulfiqar: *A comparison of RRT, RRT* and RRT*-smart path planning algorithms*, in International Journal of Computer Science and Network Security (IJCSNS), 2016, Vol. 16 (10), S. 20.

[48] Liu, Ben; Feng, Wenzhao; Li, Tingting; Hu, Chunhe; Zhang, Junguo: *A Variable-Step RRT* Path Planning Algorithm for Quadrotors in Below-Canopy*, in IEEE Access, 2020, Vol. 8, S. 62980–62989.

[49] Shi, Yangyang; Li, Qiongqiong; Bu, Shengqiang; Yang, Jiafu; Zhu, Linfeng: *Research on Intelligent Vehicle Path Planning Based on Rapidly-Exploring Random Tree*, in Mathematical Problems in Engineering, 2020, Vol. 2020.

[50] La, Hung M; Sheng, Weihua; Chen, Jiming: *Cooperative and active sensing in mobile sensor networks for scalar field mapping*, in IEEE Transactions on Systems, Man, and Cybernetics: Systems, 2014, Vol. 45 (1), S. 1–12.

[51] Hart, Peter E; Nilsson, Nils J; Raphael, Bertram: *A formal basis for the heuristic determination of minimum cost paths*, in IEEE transactions on Systems Science and Cybernetics, 1968, Vol. 4 (2), S. 100–107.

[52] Hart, Peter E; Nilsson, Nils J; Raphael, Bertram: *Correction to a formal basis for the heuristic determination of minimum cost paths*, in ACM SIGART Bulletin, 1972, (37), S. 28–29.

[53] DuchoĖ, František; Babineca, Andrej; Kajana, Martin; BeĖoa, Peter; Floreka, Martin; Ficoa, Tomáš; JurišicaA, Ladislav: *Path planning with modified a star algorithm for a mobile robot*, in Procedia Engineering, 2014, Vol. 96, S. 59–69.

[54] Barer, Max; Sharon, Guni; Stern, Roni; Felner, Ariel: *Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem*, in , 2014.

[55] Cohen, Liron; Uras, Tansel; Kumar, TK Satish; Koenig, Sven: *Optimal and Bounded-Suboptimal Multi-Agent Motion Planning*, in , 2019.

[56] Szasz, Otto: *Generalization of S. Bernstein's polynomials to the infinite interval*, in J. Res. Nat. Bur. Standards, 1950, Vol. 45 (3), S. 239–245.

[57] Lorentz, George G: *Bernstein polynomials*, American Mathematical Soc., 2013.

[58] Flores Contreras, Melvin Estuardo: *Real-time trajectory generation for constrained nonlinear dynamical systems using non-uniform rational b-spline basis functions*, in , 2008.

[59] Gao, Fei; Wu, William; Lin, Yi; Shen, Shaojie: *Online safe trajectory generation for quadrotors using fast marching method and bernstein basis polynomial*, in , 2018, S. 344–351.

[60] Preiss, James A; Hönig, Wolfgang; Ayanian, Nora; Sukhatme, Gaurav S: *Downwash-aware trajectory planning for large quadrotor teams*, in , 2017, S. 250–257.

[61] Mellinger, Daniel; Kumar, Vijay: *Minimum snap trajectory generation and control for quadrotors*, in , 2011, S. 2520–2525.

[62] Mueller, Mark W; Hehn, Markus; D'Andrea, Raffaello: *A computationally efficient motion primitive for quadrocopter trajectory generation*, in IEEE Transactions on Robotics, 2015, Vol. 31 (6), S. 1294–1310.

[63] Lei, Yao; Wang, Hengda: *Aerodynamic Optimization of a Micro Quadrotor Aircraft with Different Rotor Spacings in Hover*, in Applied Sciences, 2020, Vol. 10 (4), S. 1272.

[64] Unser, Michael; Aldroubi, Akram; Eden, Murray: *B-spline signal processing. I. Theory*, in IEEE transactions on signal processing, 1993, Vol. 41 (2), S. 821–833.

[65] Prautzsch, Hartmut; Boehm, Wolfgang; Paluszny, Marco: *Bézier and B-spline techniques*, Springer Science & Business Media, 2002.

[66] Berglund, Tomas; Brodnik, Andrej; Jonsson, Håkan; Staffanson, Mats; Soderkvist, Inge: *Planning smooth and obstacle-avoiding B-spline paths for autonomous mining vehicles*, in IEEE Transactions on Automation Science and Engineering, 2009, Vol. 7 (1), S. 167–172.

[67] Ding, Wenchao; Gao, Wenliang; Wang, Kaixuan; Shen, Shaojie: *An efficient b-spline-based kinodynamic replanning framework for quadrotors*, in IEEE Transactions on Robotics, 2019, Vol. 35 (6), S. 1287–1306.

[68] Qin, Kaihuai: *General matrix representations for B-splines*, in The Visual Computer, 2000, Vol. 16 (3-4), S. 177–186.

[69] Tsai, Ching-Chih; Huang, Hsu-Chih; Chan, Cheng-Kai: *Parallel elite genetic algorithm and its application to global path planning for autonomous robot navigation*, in IEEE Transactions on Industrial Electronics, 2011, Vol. 58 (10), S. 4813–4821.

[70] Koyuncu, Emre; Ure, N Kemal; Inalhan, Gokhan: *A probabilistic algorithm for mode based motion planning of agile unmanned air vehicles in complex environments*, in IFAC Proceedings Volumes, 2008, Vol. 41 (2), S. 2661–2668.

[71] Xu, Yang; Lai, Shupeng; Li, Jiaxin; Luo, Delin; You, Yancheng: *Concurrent optimal trajectory planning for indoor quadrotor formation switching*, in Journal of Intelligent & Robotic Systems, 2019, Vol. 94 (2), S. 503–520.

[72] Holte, Robert C; Grajkowski, Jeffery; Tanner, Brian: *Hierarchical heuristic search revisited*, in , 2005, S. 121–133.

[73] Wagner, Glenn; Choset, Howie: *M*: A complete multirobot path planning algorithm with performance bounds*, in , 2011, S. 3260–3267.

[74] Liu, Sikang; Watterson, Michael; Mohta, Kartik; Sun, Ke; Bhattacharya, Subhrajit; Taylor, Camillo J; Kumar, Vijay: *Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments*, in IEEE Robotics and Automation Letters, 2017, Vol. 2 (3), S. 1688–1695.

[75] Tang, Sarah; Kumar, Vijay: *Safe and complete trajectory generation for robot teams with higher-order dynamics*, in , 2016, S. 1894–1901.

[76] Debord, Mark; Hönig, Wolfgang; Ayanian, Nora: *Trajectory planning for heterogeneous robot teams*, in , 2018, S. 7924–7931.

[77] Choi, Youngjin; Chung, Wan Kyun: *PID trajectory tracking control for mechanical systems*, Springer Science & Business Media, 2004.

[78] Sadeghzadeh, Iman; Mehta, Ankit; Zhang, Youmin; Rabbath, Camille-Alain: *Fault-tolerant trajectory tracking control of a quadrotor helicopter using gain-scheduled PID and model reference adaptive control*, in , 2011, Vol. 2.

[79] Liu, Changlong; Pan, Jian; Chang, Yufang: *PID and LQR trajectory tracking control for an unmanned quadrotor helicopter: Experimental studies*, in , 2016, S. 10845–10850.

[80] Šiér, Zbyněk; Jüttler, Bert: *Constructing acceleration continuous tool paths using Pythagorean hodograph curves*, in Mechanism and machine theory, 2005, Vol. 40 (11), S. 1258–1272.