Aufgabenbasierte Rekonfiguration und Adaption modularer Serviceroboter

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Dipl.-Inform. Lars Pfotzer

aus Karlsruhe

Datum der mündlichen Prüfung:	17. Juli 2018
Erster Gutachter:	Prof. Dr.-Ing. Rüdiger Dillmann
Zweiter Gutachter:	Prof. Dr. rer. nat. Karsten Berns

Danksagung

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Forschungszentrum Informatik (FZI) in der Abteilung Interaktive Diagnose und Servicesysteme (IDS). Herrn Prof. Dr.-Ing. R. Dillmann danke ich besonders für die Anregung zu dieser Arbeit, die wissenschaftliche Förderung, die stets vorhandene Diskussionsbereitschaft und für die Übernahme des Hauptreferates.

Für die freundliche Übernahme des Korreferates gebührt mein ganz besonderer Dank Herrn Prof. Dr. rer. nat. Karsten Berns vom Robotics Research Laboratory der Technischen Universität Kaiserslautern. Dem Vorsitzenden des Prüfungsausschusses gilt ebenfalls mein Dank für die kritische Durchsicht des Manuskriptes. Meinen Eltern, die mir durch ihre Redlichkeit und Strebsamkeit immer Vorbild waren, die mir Heimat und Geborgenheit aber auch Freiheit und Interesse für Neues gaben, danke ich von tiefstem Herzen.

Karlsruhe, im Juli 2018 *Lars Pfotzer*

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Nach schwerwiegenden Naturkatastrophen oder Unglücken, wie beispielsweise der Hurrikan Kathrina in den USA (August 2005), der Tsunami in Tōhoku, Japan (März 2011) mit der darauf folgenden Nuklearkatastrophe in Fukushima oder das Erdbeben in Nepal (April 2015), sind meistens Einsatzkräfte der Feuerwehr, des Roten Kreuzes, des technischen Hilfswerks und anderen Hilfsorganisationen im Einsatz. Zur Unterstützung der Rettungs- und Hilfskräfte vor Ort, wird dabei immer häufiger Spezialequipment eingesetzt, unter anderem auch Robotersysteme. Auf gleiche Weise können Einsatzkräfte bei kleineren Unfällen und Ereignissen im täglichen Leben durch Serviceroboter unterstützt werden. Zu den vielseitigen Anwendungsbereichen zählen vor allem die Suche und Rettung von Personen, die Analyse und Bergung von Gefahrstoffen sowie die Inspektion und Wartung. Der Einsatz von Robotern ist insbesondere dann erforderlich, wenn die Gefahren für Rettungskräfte zu groß werden, wie z. B. bei der Entschärfung von Sprengstoffen oder der Gefahreneinschätzung von einsturzgefährdeten Gebäuden. Bei dieser Art von Einsatzszenarien müssen autonome Serviceroboter oft mit zuvor unbekannten Hindernissen und Strukturen, wie z. B. Trümmerteile von eingestürzten Wänden oder quer liegende Eisenträger und Holzbalken zurecht kommen. Zudem können während einer Mission besondere Umweltbedingungen vorliegen, wie z. B. Feuer, Rauch oder Wassereinbruch und unvorhergesehene Ereignisse eintreten, wie z. B. herab fallende Gegenstände und Trümmer. Solch komplexe Einsatzszenarien stellen immense Anforderungen an eingesetzte Roboter und erfordern Eigenschaften wie *Anpassungsfähigkeit*, *Flexibilität*, *Autonomie* und *Feldtauglichkeit*.

Diese Arbeit verfolgt den Anspruch, die Fähigkeiten aktueller Serviceroboter bezüglich dieser Anforderungen zu steigern und gleichzeitig die nötigte Robustheit der Hardware und Software für den realen Einsatz zu gewährleisten. Um dieses Ziel zu erreichen, werden Möglichkeiten zur Rekonfiguration und Adaption modularer Serviceroboter analysiert und implementiert. Darüber hinaus wird die Übertragbarkeit der Konzepte und Methoden auf andere rekonfigurierbare Systeme wie z. B. modulare Satelliten und deren Einsatz im Orbit untersucht.

1.2 Problemstellung

Zur Steigerung der Flexibilität sowie der Ausfallsicherheit durch Modularität und Redundanz sind Ansätze von rekonfigurierbaren und selbstrekonfigurierbaren Robotern sehr vielversprechend. In diesem Forschungsgebiet der Robotik existieren Fragestellungen, die bislang noch nicht vollständig gelöst wurden. Der Artikel *Modular Self-Reconfigurable Robot Systems - Grand Challenges of Robotics* [1] zeigt eine Reihe offener Fragen und Probleme auf. Insbesondere in den Bereichen Planung und Steuerung ist die Entwicklung der im Folgenden dargestellten Verfahren und Methoden erforderlich, um einen Einsatz von rekonfigurierbaren Robotern unter realen Bedingungen ermöglichen:

- Algorithmen zur Planung paralleler Bewegungen zur Manipulation und Lokomotion mit und ohne Hindernissen
- Algorithmen zur Planung der hinsichtlich Zeit und Energie optimalen Rekonfiguration mit und ohne Hindernissen
- Algorithmen zur Bestimmung der optimalen Konfiguration für eine gegebene Aufgabe und Umgebung
- Effiziente und skalierbare (asynchrone) Kommunikation zwischen einer großen Anzahl von Modulen

Basierend auf diesen Erkenntnissen und eigenen Erfahrungen aus der Teilnahme an der Katastrophenschutzübung TARANIS 2013 [2], wurden die für diese Arbeit relevanten Problemstellungen für den Einsatz von rekonfigurierbaren Robotern in einem Such- und Rettungsszenario untersucht. Dazu wurde zunächst der Ablauf realer Einsätze am Beispiel der Erkundung von einsturzgefährdeten Gebäuden generalisiert und anhand des in Abbildung 1.1 skizzierten Prozesses strukturiert.

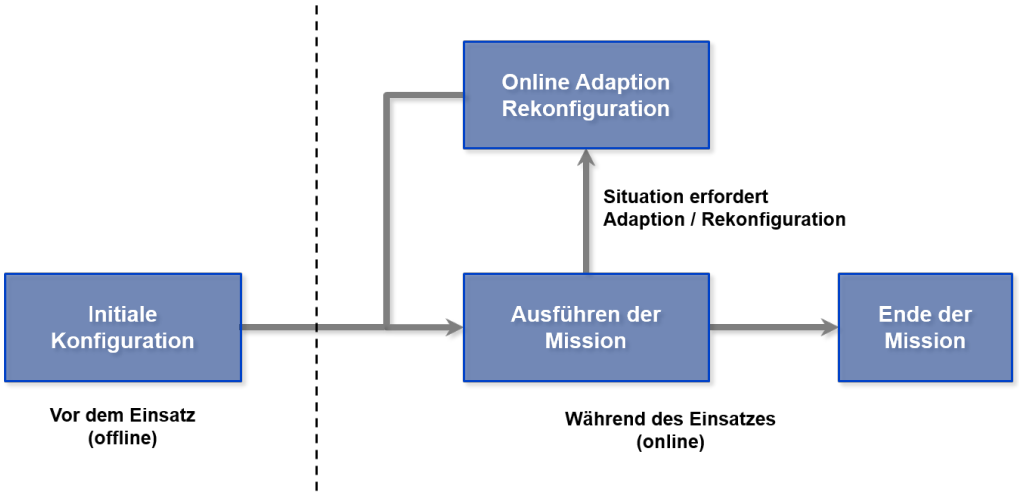


Abb. 1.1: Idealisierter Einsatzablauf eines rekonfigurierbaren Roboters.

Die identifizierten Problemstellungen lassen sich rudimentär in *Vor dem Einsatz* und *Während des Einsatzes* unterteilen. Vor dem Einsatz ist es erforderlich, die Roboterkonfiguration[1] im Hinblick auf das geplante Einsatzszenario bezüglich der Aufgabenstellung und Einsatzumgebung zu optimieren. Hierbei sollte bereits beim Design eines modularen Roboter auf eine robuste und für den Einsatz im Feld geeigneten Aufbau der Hardware sowie eine flexible und anpassungsfähige Steuerung geachtet werden.

Um einen zuverlässigen Betrieb des Robotersystems zu gewährleisten, kann es auf Grund von unvorhersehbaren Ereignissen oder Situationen während des Einsatzes notwendig sein, eine Rekonfiguration des Robotersystems oder eine Anpassung der Fortbewegung durchzuführen. Solche Situationen oder Ereignisse können z. B. Hindernisse sein, die der Roboter mit seiner aktuellen Konfiguration nicht überwinden kann oder durch Umwelteinflüsse hervorgerufene Beschädigungen von Sensoren oder Aktoren des Roboters. Zur Rekonfiguration von mehreren mobilen Robotersystemen während des Einsatzes wird eine robuste mechanische Schnittstelle zum An- und Abkoppeln sowie Verfahren zur selbständigen Rekonfiguration benötigt. Zusätzlich wird eine Navigations- und Bewegungsplanung benötigt, die in der Lage ist, die Fähigkeiten des Roboters bereits bei der Planung zu berücksichtigen.

Die aus der Analyse des Einsatzszenarios gewonnenen Problemstellungen lassen sich zu den im Folgenden dargestellten Kernthemen dieser Arbeit zusammenfassen:

1. **Optimierung des Hardwaredesigns und der Steuerungssoftware** zur Steigerung der Flexibilität und Robustheit in realen Einsätzen
2. **Bestimmung von optimalen Roboterkonfigurationen** hinsichtlich der Aufgabenstellung und der Umgebung in verschiedenen Einsatzszenarien
3. **Anpassung der Fortbewegung an die Umgebung** durch biologisch-inspirierte Lokomotionsarten und deren automatischen Selektion
4. **Konfigurationsabhängige Navigation und Lokalisierung** zur Überwindung von Hindernissen in unstrukturiertem Gelände

1.3 Beitrag dieser Arbeit

Die vorliegende Arbeit erweitert die Fähigkeiten aktueller Robotersysteme durch innovative Hardware- und Softwarelösungen zur Rekonfiguration und Adaption modularer Serviceroboter. Das entwickelte Verfahren zur Synthese und Optimierung von Roboterkonfigurationen und deren Morphologie bezüglich verschiedener Missionsparameter, wie Einsatzgebiet und Umweltbeschaffenheit, steigert

[1] Eine Roboterkonfiguration beschreibt in diesem Kontext die Hardwarestruktur (Morphologie) eines modularen Robotersystems und ist nicht zu verwechseln mit der häufig verwendeten Gelenkwinkelkonfiguration.

die Flexibilität sowie die Anwendungsmöglichkeiten von rekonfigurierbaren Robotern. Zur Vereinfachung der Modellierung und Durchführung der einzelnen Optimierungsschritte wird eine interaktive Benutzerschnittstelle bereitgestellt. Die implementierte Methode zur Navigations- und Bewegungsplanung ermöglicht das adaptive Überwinden von Hindernissen unter Berücksichtigung der Roboterkonfiguration sowie Umweltbedingungen. Im Zusammenspiel mit biologisch inspirierten Fortbewegungsarten wird die Autonomie und Feldtauglichkeit modularer Serviceroboter verbessert. Zusammengefasst erweitert diese Arbeit den Stand der Technik primär durch die folgenden Konzepte und deren algorithmische Umsetzung:

1. Entwurf von **rekonfigurierbaren Multi-Roboter Systemen**, insbesondere einer **flexiblen Steuerungsarchitektur** und **Hardwareschnittstelle**
2. **Syntheseprozess** zur aufgabenspezifischen **Generierung und Optimierung** modularer **Roboterkonfigurationen**
3. Verfahren zur **Navigations- und Bewegungsplanung** sowie **Adaption der Fortbewegung** unter Berücksichtigung der aktuellen **Roboterkonfiguration** und der **Umwelt**
4. Erweiterung des **RRT* Pfadplanungsalgorithmus** um einen **Sekundären Nächsten Nachbar Raum (SSN)**

1.4 Aufbau der Arbeit

Diese Arbeit strukturiert sich gemäß der vier Forschungsschwerpunkte wie folgt: Nach der Einleitung folgt in Kapitel 2 der aktuelle Stand der Technik und eine Beschreibung der Grundlagen. Kapitel 3 zeigt den Entwurf von rekonfigurierbaren Multi-Roboter-Systemen (RMRS) sowie die Optimierung des Hardwaredesigns und der Steuerungssoftware anhand von KAIRO 3. In Kapitel 4 wird die Konzeption und Umsetzung eines innovativen Verfahrens zur Generierung und Optimierung von Roboterkonfigurationen vorgestellt, das sich für den offline und online Einsatz mit RMRS eignet. Zur Anpassung an unvorhersehbare Hindernisse oder Ereignisse werden in Kapitel 5 biologisch inspirierte Fortbewegungsarten und der Wechsel zwischen diesen analysiert und implementiert. Kapitel 6 beschreibt die Konzeption und Umsetzung einer Methode zur Navigations- und Bewegungsplanung unter Berücksichtigung der aktuellen Roboterkonfiguration und dessen Fähigkeiten zur Überwindung von Hindernissen. Die für alle Kernthemen durchgeführten Experimente und deren Ergebnisse werden in Kapitel 7 vorgestellt und diskutiert. Die Zusammenfassung dieser Arbeit und ein Ausblick findet sich in Kapitel 8.

2 Stand der Technik

2.1 Einordnung modularer und selbst-rekonfigurierbarer Robotersysteme

Ein **modulares Robotersystem** ist üblicherweise aus mehreren homogenen oder heterogenen Bausteinen (sogenannten Building Blocks) zusammengesetzt. Zwischen den einzelnen Bausteinen befinden sich einheitliche Schnittstellen zur Übertragung von mechanischen Kräften und Momenten, Energie und Daten innerhalb des Robotersystems. Mit Hilfe dieser Schnittstellen entsteht die Möglichkeit, durch Veränderung der Bausteinanordnung, unterschiedliche Formen und Kinematiken zu erzeugen [1]. In diesem Kontext wird eine feste Bausteinanordnung als *Konfiguration* und die Veränderung einer solchen Konfiguration als *Rekonfiguration* bezeichnet.

Modulare selbst-rekonfigurierbare Roboter (MSRR), können als echte Teilmenge der modularen Roboter angesehen werden. Bei diesen Systemen lassen sich einzelne Bauteine oder Gruppen von Bausteinen insbesondere während des laufenden Betriebs austauschen und umordnen. Erfolgt die Rekonfiguration durch den Roboter selbst (ohne externe Einflüsse), so spricht man von Selbst-Rekonfiguration. Diese Robotersysteme können sich während eines Einsatzes an verschiedene vorher unbekannte Gegebenheiten anpassen und auf unvorhersehbare Ereignisse reagieren [3].

Die in der Forschung existierende Vielzahl unterschiedlicher modularer selbstrekonfigurierbarer Robotersysteme erfordert eine Kategorisierung. Murata und Kurokawa [4] klassifizieren MSRR nach der **Art der Bausteinanordnung** in die drei Kategorien *gitterförmig* (lattice type), *kettenförmig* (chain type) und *hybrid* (hybrid). Die Klasse der gitterförmigen MSRR umfasst Systeme bei denen die Bausteine in einer sich wiederholenden und symmetrischen Struktur angeordnet sind, vergleichbar mit Kristallen oder biologischen Zellen. Wie in Abbildung 2.1 (a) dargestellt, befinden sich alle Bausteine in der durch die Geometrie festgelegten 2-D oder 3-D Gitterstruktur. Bei kettenartigen MSRR hingegen, sind Bausteine mit mindestens zwei Schnittstellen in einer überwiegend seriellen Struktur angeordnet, ähnlich wie bei schlangenartigen Robotern. Durch Bausteine mit mehr als zwei Schnittstellen werden Verzweigungen und somit baumartige Strukturen, wie in Abbildung 2.1 (b) ermöglicht. Als hybride MSRR wird eine Kombination aus gitter- und kettenförmigen Bausteinanordnungen bezeichnet, welche die Eigenschaften beider Klassen kombiniert.

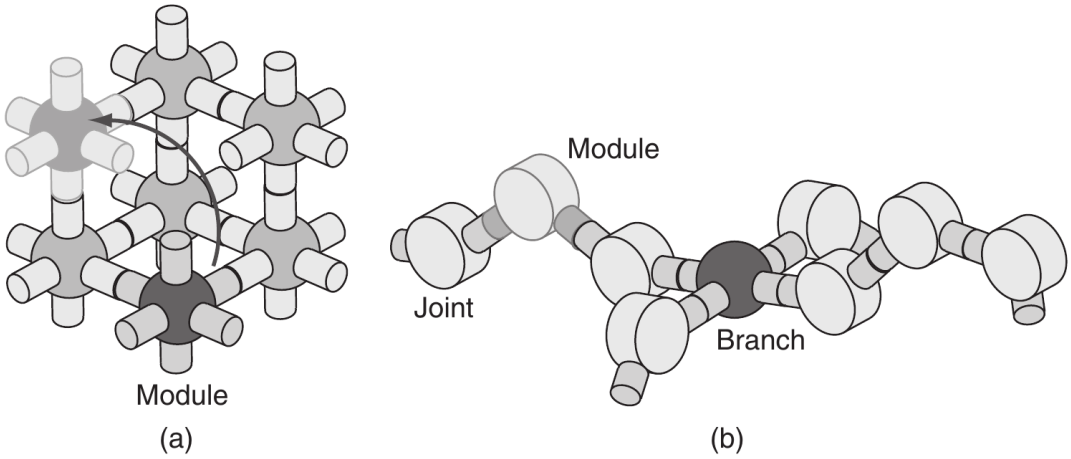


Abb. 2.1: Vergleich zwischen (a) gitterförmigen und (b) kettenartigen modularen selbst-rekonfigurierbaren Robotern [4].

Yim et al. [3] unterscheiden in einer Taxonomie ebenfalls die drei Klassen *gitterartige*, *ketten- oder baumartige* und *mobile* Roboterarchitekturen. Darüber hinaus werden selbst-rekonfigurierbare Systeme auch nach der **Art der Rekonfiguration** unterteilt. Bei der *deterministischen Rekonfiguration* ist jederzeit bekannt, wo sich die einzelnen Module befinden und die Dauer des Umformens zwischen zwei Konfigurationen kann exakt bestimmt werden. Die *stochastische Rekonfiguration* hingegen benutzt Folgen von Zufallsversuchen um die Pose von Modulen zu verändern. Dabei sind die Posen der einzelnen Module nur dann exakt bestimmbar, wenn diese mit der Basisstruktur verbunden sind. Dadurch kann die Ausführungszeit eines Rekonfigurationsvorgangs lediglich geschätzt werden.

Chennareddy et al. [5] zeigen eine aktuelle Erweiterung der MSRR Klassifikation von Yim et al. mit Fokus auf verschiedenen Hardware Architekturen. Neben der Einordnung nach der Art der Anordnung (Struktur) und der Art der Rekonfiguration wird zusätzlich die **Art der Fortbewegung** (Lokomotion) und der **Formfaktor** vom MSRR unterschieden. Die Klassifizierung nach der Art der Fortbewegung trennt zwischen externer (External), koordinierter (Coordinated) und mobiler (Mobile) Lokomotion. Systeme in der Klasse *External* besitzen meistens keine eigenen Antriebe und benötigen Benutzereingriffe, Umwelteinflüsse oder Störungen von außen für die Fortbewegung bzw. Rekonfiguration. Zur Klasse mit *koordinierter Lokomotion* zählen MSRR, die lediglich Gelenke und keine eigenen Räder oder Ketten zur Fortbewegung besitzen. Für das initiale Verbinden der Bausteine bzw. das Zusammenfügen von mehreren Bausteingruppen ist in gewissem Maße das Eingreifen eines Benutzers erforderlich. Die Klasse der *mobilen Lokomotion* umfasst MSRR die Sensor- und Aktor-Bausteine besitzen, beispielsweise Bausteine mit Kameras und Radantrieben. Mobile MSRR sind somit in der Lage sich selbstständig fortzubewegen und zu rekonfigurieren. Bei der Einordnung nach dem Formfaktor werden die drei Kategorien *Mikro*, *Mini* und *Makro*

unterschieden. Chennareddy bezeichnet Systeme die für das menschliche Auge nicht sichtbar sind als Mikro. MSRR deren Module erkennbar und kleiner als 5 cm sind, werden als Mini und alle anderen Systeme als Makro kategorisiert.

Abbildung 2.2 zeigt die Taxonomie von Gomez et al. [6], welche die Klassifikation von MSRR weiter unterteilt und existierende Robotersysteme in diese Hierarchie einordnet. Insbesondere werden gitterförmige MSRR nach ihrer räumlichen Struktur 2-D und 3-D unterschieden. Kettenartige MSRR werden in die beiden Topologie Klassen 2-D & 3-D sowie 1-D eingeteilt. 1-D Strukturen werden weiter unterteilt in schlangenartige und Serpentine (Rad- oder Kettenantrieb) MSRR.

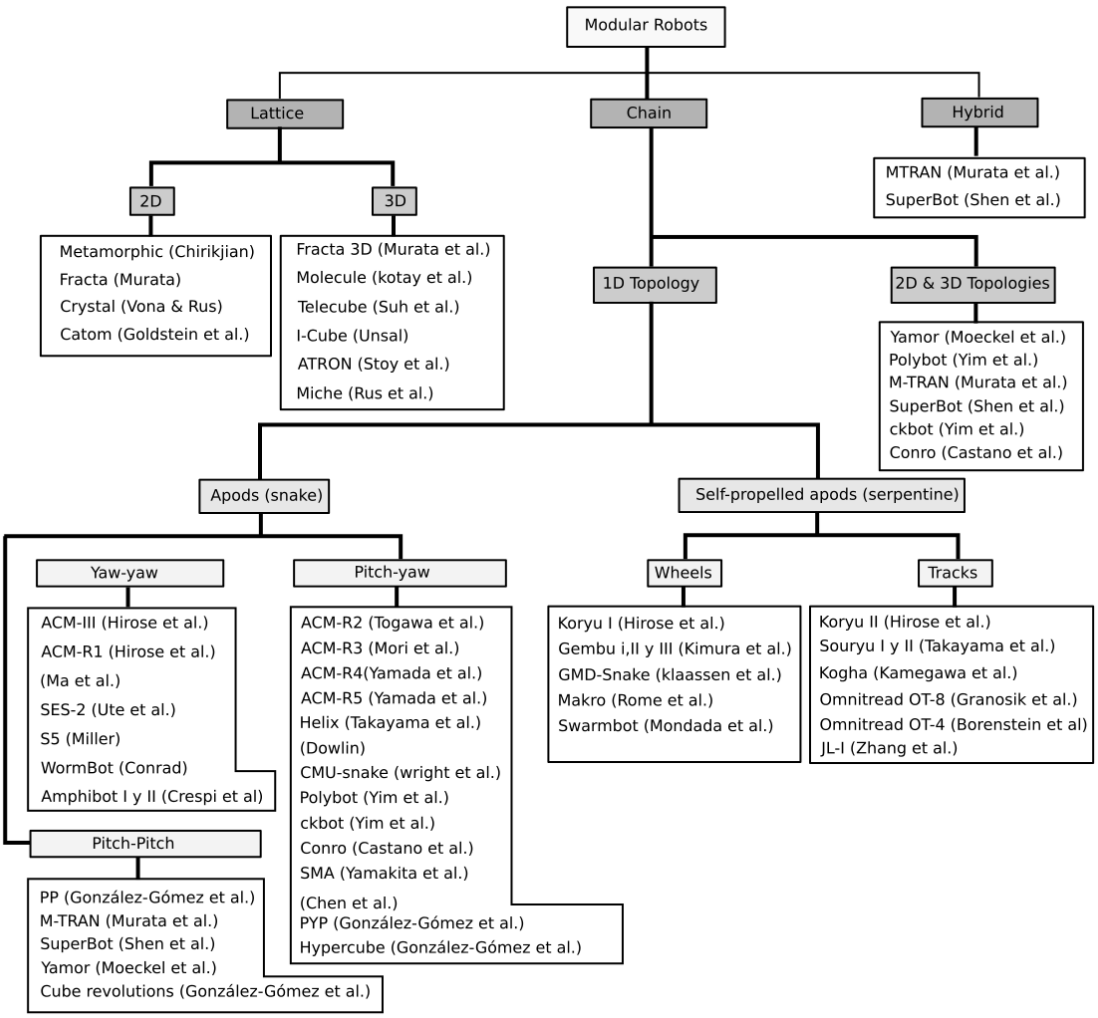


Abb. 2.2: Taxonomie modularer und selbstrekonfigurierbarer Roboter [6].

2.1.1 Hardwareschnittstellen zur Rekonfiguration

Eine der wichtigsten Komponenten bei MSRR ist die Hardwareschnittstelle zum Verbinden und Trennen der einzelnen Bausteine. Neben dem Herstellen einer fes-

ten mechanischen Verbindung (mechanische Schnittstelle) werden ebenfalls elektrische Schnittstellen zur Energieübertragung und Datenschnittstellen zur Kommunikation zwischen den einzelnen Bausteine benötigt. Aktuelle Ansätze zur mechanischen und magnetischen Kopplung von MSRR werden in der Arbeit von Müller [7] untersucht und analysiert. Darauf aufbauend gibt die Bachelorarbeit von Gerdes [8] einen Überblick über existierende Schnittstellen zur Energie- und Datenübertragung und vergleicht insbesondere solche, die sich in die mechanische Schnittstelle integrieren lassen. Eine weitere Übersicht von Schnittstellen und eine Bewertung dieser speziell für den Einsatz in modularen rekonfigurierbaren Satelliten, wird von Göller et al. [9] gezeigt. Neben der Analyse aktueller physikalischer Verbindungsmechanismen werden ebenfalls die elektrische Kopplung und Datenkopplung (Kommunikation) untersucht. Als Schnittstellen zur Energieübertragung werden am häufigsten Steckverbinder, Kontaktflächen oder Schleifringe eingesetzt. Einige MSRR nutzen Akkus innerhalb der einzelnen Module um eine unabhängige Energieversorgung sicherzustellen und verzichten auf elektrische Schnittstellen. Zur Übertragung von Daten werden meist die zur Stromversorgung vorhandenen Schleifringe oder Steckverbinder mitbenutzt. Alternativ wird auch oft eine drahtlose Kommunikation mit Hilfe von Infrarot LEDs oder einer Funkübertragung realisiert.

2.2 Bestimmung der optimalen Roboterkonfiguration

2.2.1 Repräsentation von Konfigurationen

Zur Darstellung und Speicherung von Konfigurationen modularer Robotersysteme gibt es bereits viele verschiedene Ansätze. Manche Datenstrukturen sind für den Einsatz von Algorithmen spezialisiert, andere Datenformate sind besonders allgemeingültig ausgelegt, um möglichst viele verschiedene Systeme darstellen zu können. Sowohl Yang und Chen [10] als auch Wu et al. [11] verwenden Assembly Incidence Matrizen (AIM) zur Beschreibung von Konfigurationen modularer Roboter. Diese Matrixdarstellung beschreibt Verbindungen zwischen verschiedenen Modulen durch Angabe der benachbarten Schnittstellen. Dabei wird eine Verbindung als Tupel von Schnittstellenindices in einer Matrix gespeichert. Als Erweiterung der AIMs wurde die Configuration Coupling Matrix (CCM) von Dong und Li [12] vorgestellt. Diese Matrixrepräsentation beschreibt Schnittstellen, Orientierung und Längen aufeinander folgender Module kettenförmiger, modularer Manipulatoren. Die *Robot Formation Language* [13] wurde speziell für die Darstellung kollektiver Roboter (eine Art rekonfigurierbarer Schwarmroboter) entwickelt. Bei der Spezifikation dieser formalen Beschreibungssprache wurde besonders auf die Möglichkeit zur maschinellen Verarbeitung in Optimierungsalgorithmen geachtet. Darüber hinaus existieren auch einige XML basierte Beschreibungssprachen zur Darstellung von Roboterkonfigurationen, wie bei-

spielsweise URDF (Unified Robot Description Language)[1] und SRDF (Semantic Robot Description Language) [14].

Einige der vorgestellten Datenstrukturen sind besonders auf den Einsatz in Optimierungsalgorithmen spezialisiert wohin gegen andere Datenformate auf eine besonders übersichtliche und visuell lesbare Darstellung komplexer und vielfältiger MSRR ausgelegt sind.

2.2.2 Algorithmen zur Konfigurationsgenerierung und -optimierung

Für die Optimierung modularer Robotersysteme werden bereits biologisch-inspirierte Methoden eingesetzt, die den Prozess der natürlichen Evolution nachbilden. Zur Generierung aufgabenspezifischer Konfigurationen für MSRR Roboter gibt es bereits einige vielversprechende Ansätze. Chris Leger veröffentlichte 1999 eine Methode zur automatischen Synthese von Roboterkonfigurationen [15]. Mit Hilfe eines evolutionären Ansatzes optimiert das Verfahren sowohl die Eigenschaften von heterogenen Modulen wie Abmessungen und Materialien als auch die Anordnung dieser. Dabei werden aufgabenspezifische Konfigurationen erzeugt, die während der Synthese anhand der Aufgabe simuliert und evaluiert werden. Der Optimierungsprozess ist grundsätzlich auf den Bereich der mobilen Manipulation von Objekten beschränkt. Aufgrund der detaillierten Simulation aller erzeugter Konfigurationen ergeben sich sehr lange Laufzeiten für diesen Ansatz. Das Verfahren eignet sich deshalb insbesondere für einen offline Einsatz.

Ein weiterer Ansatz zur aufgabenorientierten Optimierung modularer Roboterkonfigurationen wurde von Yang, Chen et al. vorgestellt [10]. Der Schwerpunkt dieser Arbeit liegt auf der Erzeugung von anwendungsspezifischen Kinematiken modularer Manipulatoren für den Einsatz in der industriellen Fertigung. Deshalb beschränken sich Yang und Chen auf kettenartige Konfigurationen. Es werden die bereits in Kapitel 2.2.1 vorgestellten AIMs für die Darstellung der Konfigurationen verwendet. Die Optimierung der Bausteinanordnung erfolgt mittels eines genetischem Algorithmus. Bei der Evaluation ergeben sich ebenfalls lange Laufzeiten und die Beschränkung auf je zwei Schnittstellen pro Baustein.

Winkler et al. [16] beschreiben in einen genetischen Algorithmus, der mithilfe der in Kapitel 2.2.1 vorgestellten *Robot Formation Language* Konfigurationen erzeugt. Bei dieser Methode werden sowohl die Anzahl und Anordnung der Bausteine als auch die Steuerung des Robotersystems iterativ verbessert. Dazu wird die Fortbewegungsart jeder einzelnen Konfiguration in einer Simulationsumgebung evaluiert und die Steuerungsparameter für die Gelenke angepasst. Als Optimierungskriterium wird hierbei die Geschwindigkeit der Fortbewegung verwendet. Zur Steigerung der Variation von Konfigurationen wurde in spezielles Abstands-

[1] http://www.ros.org/wiki/urdf

und Unterschiedlichkeitsmaß eingeführt. Aufgrund der physikalischen Simulation jedes einzelnen Individuum in einer virtuellen Umgebung, ergeben sich für diesen Ansatz lange Laufzeiten.

Aus dem aktuellen Stand der Technik lässt sich folgern, dass die Bestimmung geeigneter Konfigurationen heterogener MSRR für Aufgaben und Anwendungen im realen Einsatz noch nicht vollständig gelöst ist. Insbesondere erschweren die langen Laufzeiten den online Einsatz der aktuellen Verfahren.

2.3 Fortbewegungsarten und Adaption an die Umwelt

Bewegungsmuster und Fortbewegungsarten werden bei Robotersystemen oft von entsprechenden biologischen Vorbildern, wie Säugetieren oder Insekten, abgeleitet, da die Natur diese mit den Verlauf der Zeit optimiert und perfektioniert hat. Insbesondere für schlangenartige Robotersysteme gibt es bereits verschiedene Ansätze zur biologisch motivierten Fortbewegung.

Ein Ansatz zur rollen-basierten Steuerung [17] erzeugt feste Bewegungsmuster mit Hilfe vordefinierter Sequenzen von Gelenkwinkeln. Jedes Modul für sich fährt entsprechend seiner Rolle die vorgegebenen Gelenkwinkel an. Ein Synchronisationssignal steuert die zeitliche Ausführung auf den einzelnen Modulen. Variable Bewegungsmuster werden durch Generierung und Modulation von sinusförmigen Wellen erzeugt [18, 19]. Komplexere Verfahren basieren auf zentralen Mustergeneratoren [20, 21] oder so genannten Backbone Curves [22], welche mithilfe von Neuronen verschiedene adaptive Bewegungen erzeugen. Gomez et al. [23] analysiert die vielfältigen Bewegungsmöglichkeiten von modularen schlangenartigen Robotern und zeigt, dass es mithilfe eines zentralen Mustergenerators möglich ist, fünf verschiedene Bewegungsmuster für diese Art von Robotern zu erzeugen.

Im Gegensatz zur wellenförmigen Fortbewegung hat Chen et al. [24] ein Modell zur Lokomotion von raupenartigen Robotern mit linear angetriebenen Modulen vorgestellt. Zur Steuerung der abwechselnden Kontraktion und Ausdehnung wird bei dieser Methode ein endlicher Zustandsautomat eingesetzt. Ein weiterer Ansatz [25] zeigt die Entwicklung eines raupenartigen Roboterprototyp bei dem die Bewegung bereits in der Struktur kodiert ist. Durch Verwendung spezieller Mikrostrukturen und Formgedächtnismaterialien, die durch Spulen angeregt werden können, wird eine omega-förmige Bewegung, ähnlich der Fortbewegung von Raupen, erzeugt.

Bei Robotern, deren Aufbau und Kinematik unterschiedliche Bewegungsarten ermöglicht, kann das Wechseln zwischen diesen vorteilhaft sein. In Abhängigkeit von Einsatzort und Gelände sind bestimmte Fortbewegungsarten besser geeignet als andere. Für die Auswahl der passenden Lokomotion kann unter anderem die Bodenbeschaffenheit in Betracht gezogen werden. Zur Erkennung des Bodenkontakt bei schlangenartigen Robotern, entwickelten Yang et al. [26] taktile

Sensoren zur Messung der Reibung zwischen Roboter und Boden. Die Ergebnisse sehen sehr vielversprechend aus, lassen sich jedoch nur auf Roboter mit direktem und kontinuierlichem Bodenkontakt anwenden. Ein weiterer Ansatz untersucht und vergleicht verschiedene Methoden zur kamerabasierten Terrain Klassifikation des Geländes [27].

2.4 Navigations- und Bewegungsplanung mobiler Roboter

Im Bereich der mobilen Roboternavigation existiert bereits eine Vielfalt von Methoden und Algorithmen zur Bewegungs- und Pfadplanung für unterschiedliche Anwendungen. Die Planungsmethoden reichen von lokalen Ansätzen, wie dem Potentialfeldverfahren [28] bis hin zur graphbasierten Planung mit Bewegungsprimitiven [29, 30]. Bei der Bewegungs- und Pfadplanung in vieldimensionalen Zustandsräumen sind Verfahren, die auf Stichproben basieren (auch als Sampling bezeichnet) besonders effizient. Algorithmen, wie der Rapidly-exploring Random Tree (RRT) von LaValle [31] oder der RRT-Connect von Kuffner [32] finden immer einen möglichen Pfad, jedoch gibt es keine Garantie für dessen Qualität. Als Erweiterung des Standard RRT, führte Karaman et al. [33, 34] den asymptotisch optimalen Planungsalgorithmus RRT* ein und gibt außerdem einen Einblick in die entsprechende Sampling Theorie. Weitere Methoden zur Bewegungs- und Pfadplanung werden unter anderem beschrieben durch Latombe et al. [35], LaValle et al. [36], Siciliano et al. [37] und Coenen et al. [38].

Auf Grund der meist sehr großen Anzahl an Freiheitsgraden, werden für schlangenartige Roboter oft biologisch inspirierte Planungsmethoden eingesetzt. Hirose et al. [39] und Chirikjian et al. [40] zeigen die Berechnung und Generierung simultaner Bewegungen für hoch-redundante Robotersysteme. Darüber hinaus existieren hybride Systeme, wie OmniThread [41], Wheeeler [42], KOHGA [43] und KAIRO 3 [44], die sich sowohl klassisch rad- bzw. kettengetrieben als auch schlangenartig fortbewegen können. Die Fortbewegung solcher radgetriebener schlangenartiger Systeme kann durch den *follow-the-leader* Ansatz von Yamada et al. [45] und Scholl et al. [46] erreicht werden. Eine weitere Methode ist als *n-trailer* bekannt und wird von Granosik et al. [47], Murugendran et al. [48] und Altafini et al. [49] beschrieben.

Konventionelle Navigations- und Planungsverfahren sind häufig auf mobile und radgetriebene Roboter ausgelegt und optimieren deshalb die geplanten Pfade zur Vermeidung von Hindernissen und Objekten in der Umwelt. Im Gegensatz dazu, sind biologisch inspirierte Robotersysteme, wie KAIRO 3 (Kapitel 2.5) und LAURON V [50] technisch in der Lage über Hindernisse zu klettern. In diesem Kontext beschreibt Kano et al. [51] die *Scaffold-Based* Lokomotion, bei der explizit Hindernisse in der Umgebung für die Fortbewegung eines Roboters genutzt werden. Der Ansatz wurde in Form einer dezentralen Steuerung mit dem schlangenartigen Roboter HAUBOT III umgesetzt. Das Steuerungsprinzip *EARLI (Exten-*

ded Asymmetrical Reverse Lateral Inhibition) [52] erzeugt eine Vorwärtsbewegung durch Aufbringen eines Drehmoments auf die Kontaktpunkte zwischen dem Roboter und Hindernissen in seiner Umgebung. Die Auswertung dieser Hindernisbasierten Fortbewegung erfolgt in einer Physiksimulation. Liljeback et al. [53] erläutern einen Ansatz zur Steigerung der Vortriebskraft bei schlangenartiger Fortbewegung durch das Rotieren benachbarter Module. Die Regelung nutzt dabei die Hindernisse in der Umgebung um eine Vorwärtsbewegung des Roboters zu erzielen.

Die meisten der betrachteten Verfahren zur Hindernis-basierten Fortbewegung konzentrieren sich auf die Ebene der Steuerung oder Regelung. Die Fähigkeit Hindernisse überwinden zu können, steigert jedoch ebenfalls den Aufwand bei der Bewegungs- und Pfadplanung. Eine weitere Herausforderung speziell bei MSRR ist die Fähigkeit zur Rekonfiguration und insbesondere die Veränderung der Kinematik. Bisher betrachten und unterstützen nur sehr wenige Navigationsverfahren eine Anpassung der Roboterkonfiguration während der Laufzeit, zumal in Abhängigkeit der Roboter Kinematik unterschiedliche Bewegungen möglich sind, die wiederum spezialisierte Planungsstrategien erfordern.

2.5 Modularer schlangenartiger Serviceroboter KAIRO 3

Der am FZI Forschungszentrum Informatik entwickelte KAIRO 3 (Karlsruher Autonomer Inspektions-ROboter 3) ist, wie sein Vorgänger KAIRO II [54], ein biologisch inspiriertes Robotersystem. Dank des modularen kettenförmigen Aufbaus ist KAIRO 3 in der Lage mehrere Körpersegmente anzuheben (siehe Abbildung 2.3 links). KAIRO 3 kann sich ebenfalls durch schmale Bereiche und Öffnungen hindurch bewegen, wie in Abbildung 2.3 rechts dargestellt. Auf Grund dieser Fähigkeiten ist KAIRO 3 für Inspektions- und Wartungsarbeiten und insbesondere für die Suche und Rettung in schwer zugänglichen oder für Menschen gefährlichen Umgebungen geeignet. Die Einsatzgebiete reichen von Industrieanlagen über einsturzgefährdete Gebäude bis hin zu toxisch bzw. radioaktiv belasteten Gebieten.

2.5.1 Hardwareaufbau

Der modulare Hardwareaufbau von KAIRO 3 besteht aus zwei geometrisch unterschiedlichen Arten von Bausteinen: *Antriebsmodule* und *Gelenkmodule*.

Antriebsmodule, wie in Abbildung 2.4a darstellt, dienen dem Vortrieb des Roboters und besitzen dazu jeweils zwei voneinander unabhängige Räder. Zur Realisierung enger Kurvenfahren, wird jedes Rad über einen eigenen Motoren angesteuert. Auf Grund der sehr kompakten Bauweise des Antriebssystems [55] steht in jedem Antriebsmodul freier Platz für die Integration von Sensoren, Aktoren,

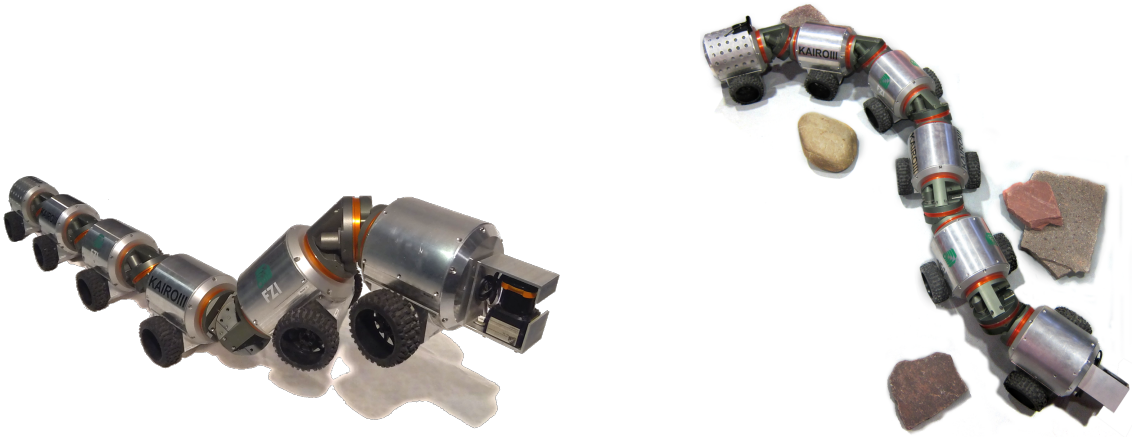


Abb. 2.3: KAIRO 3: Modularer schlangenartiger Inspektionsroboter.

Rechner und Energieversorgung sowie zum Transport von Nutzlasten zur Verfügung. Ein Antriebsmodul ohne zusätzliche Komponenten wiegt 4 kg.

(a) Antriebsmodul (b) Gelenkmodul

Abb. 2.4: Modularer Aufbau von KAIRO 3 bestehend aus abwechselnd miteinander verbundenen Abtriebs- und Gelenkmodulen.

Gelenkmodule, wie in Abbildung 2.4b gezeigt, verbinden jeweils zwei Antriebsmodule miteinander und besitzen drei rotatorische Freiheitsgrade (DOF). Jede der drei Gelenkachsen *Alpha*, *Beta* und *Gamma*, wird aktiv über integrierte Motoren angesteuert. Abbildung 2.5 veranschaulicht die Anordnung der *Alpha* und *Gamma* Gelenke entlang der Mittelachse des Roboters, wobei das *Beta* Gelenk um 45° dazu rotiert ist. Diese spezielle Anordnung der Gelenkachsen ermöglicht die Durchführung eines Kabelstrangs zur Strom- und Datenübertragung zwischen den Modulen und ebenfalls das Abknicken eines Gelenkmoduls um bis zu 90°. Mit Hilfe sogenannter Harmonic Drive Getrieben wird eine kompakte Bauweise und die benötigte Kraft der Gelenke erreicht. Das Gesamtgewicht eines Gelenkmoduls inklusive Motoren und Getriebe beträgt 4,5 kg. Eine Kombination aus der Gelenkkraft mit der besonderen Kinematik, befähigt KAIRO 3 zum Anheben einzelner Modulen, wie in Abbildung 2.3 links dargestellt.

Mit einer Konfiguration bestehend aus elf Modulen (sechs Antriebsmodule und

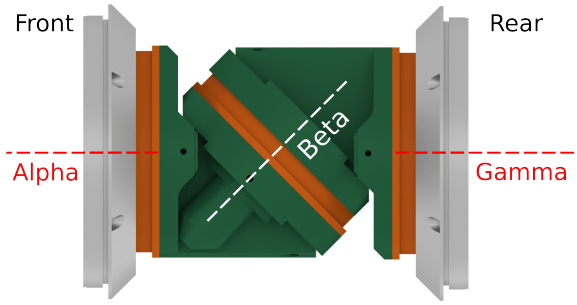


Abb. 2.5: Anordnung der drei Achsen eines KAIRO 3 Gelenkmoduls.

fünf Gelenkmodule), erreicht der Roboter eine Gesamtlänge von 1,8 m bei einem Gewicht von 47 kg. Die Komplexität bei der Steuerung ist auf Grund mehrerer verketteter Freiheitsgrade, z. B. 27 DOF bei der Beispielkonfiguration mit jeweils zwei DOF pro Antriebsmodul und drei DOF pro Gelenkmodul, sehr hoch. Dementsprechend wird für KAIRO 3 die im Folgenden vorgestellte Steuerungsmethode verwendet.

2.5.2 Steuerungsarchitektur

Die hierarchische Steuerungsarchitektur von KAIRO II [54] wird in dieser Arbeit als Grundlage für die Steuerung von KAIRO 3 genutzt. Abbildung 2.6 zeigt die Unterteilung in die vier Ebenen *Missionsplanung*, *Bewegungsplanung*, *Ausführung* und *Basissteuerung* sowie den Daten- und Kontrollfluss zwischen den einzelnen Ebenen.

Die *Missionsplanung* auf der obersten Ebene ermöglicht die Vorgabe von Steuerungsbefehlen und die Auswahl von Manövern an Hand einer grafischen Benutzeroberfläche. Zur Planung von übergeordneten Aktionen und Aufgaben, kann die von Ziegenmeyer entwickelte semantische Missionssteuerung [56] einsetzt werden.

Mit Hilfe der *Bewegungsplanung* wird ein durch die Missionsplanung vorgegebenes Manöver von der Manöverkontrolle ausgeführt. Ein Manöver wie z.B. *Freie Fahrt*, *Inspektion* oder *Stufe* beschreibt die Abfolge von Bewegungsabläufen. Basierend auf den Sensordaten der unteren Ebenen und dem aktuellen Zustand des Roboters, wird eine virtuelle Schiene erzeugt, auf der sich der Roboter fortbewegt. Die Funktionsweise der virtuellen Schiene wird im folgenden Abschnitt näher erläutert.

Das Befahren der virtuellen Schiene zur Fortbewegung von KAIRO II wird durch die Ebene *Ausführung* realisiert. Die inverse Kinematik des Roboters rechnet die Positionen der virtuellen Schiene in Gelenkwinkel um. Sensorwerte von der Basissteuerung werden vorverarbeitet und an die Bewegungsplanung weitergegeben. Zusätzlich lassen sich während der Bewegung dynamische Verzerrungen, wie z. B. das Anheben oder Drehen von Modulen erreichen.

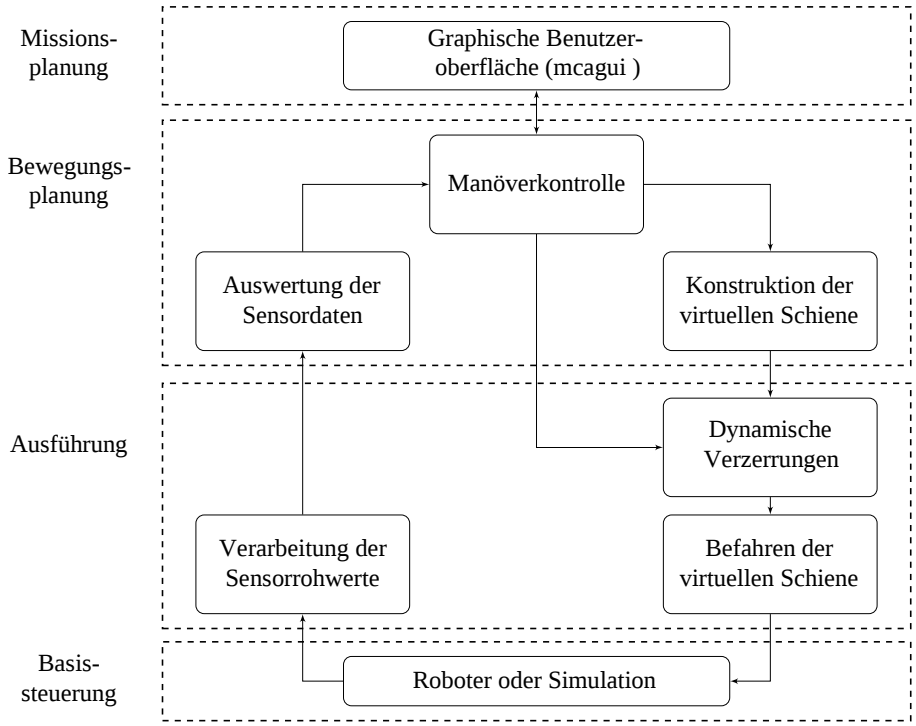


Abb. 2.6: Hierarchische Steuerungsarchitektur von KAIRO II [54].

Die *Basissteuerung* überträgt Steuersignale und liest Sensorwerte des zugrundeliegenden Roboters. Durch eine Hardwareabstraktion können verschiedene reale oder simulierte Robotersysteme an die Steuerung angebunden werden. Eine Simulationsumgebung ermöglicht es Algorithmen und neue Verfahren zu testen, bevor diese auf dem echten Roboter ausgeführt werden [54].

Virtuelle Schiene

Zur Vereinfachung der Steuerung von kinematisch hoch-redundanten Robotersystemen wie KAIRO II, bietet sich der Einsatz einer virtuellen Schiene [46, 57] an. Dabei wird zunächst ein Polygonzug definiert, der als Trajektorie für die Fortbewegung des Roboters dient. Abbildung 2.7 zeigt ein Beispiel einer virtuellen Schiene mit Wegpunkten, die aus Posen im $\mathrm{SE}\,2$ Raum bestehen. Die Strecken vor der ersten Pose (first rail base) sowie nach der letzten Pose (last rail base) werden jeweils in das Unendliche verlängert.

Der von Scholl [57] entwickelte NSVS-Algorithmus bildet einen Roboter mit n Modulen auf eine virtuelle Schiene ab, indem die Mittelpunkte aller Gelenkmodule (*kinks*) auf den durch die Schiene definierten Pfad gelegt werden. Abbildung 2.8 stellt die Projektion eines schlangenartigen Roboters bestehend aus 3 Antriebs- und 2 Gelenkmodulen auf eine geradlinige virtuelle Schiene dar. Vor und nach dem ersten bzw. letzten Antriebsmodul werden sogenannte

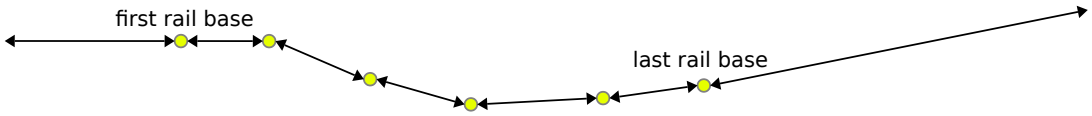


Abb. 2.7: Definition einer virtuellen Schiene mit gelb markieren Wegpunkten, die einen Pfad zwischen dem ersten (*first rail base*) und letzten Wegpunkt (*last rail base*) beschreiben.

virtuelle Gelenke (*kink #0* und *kink #3*) eingefügt um eine Richtungsvorgabe bei Vorwärts- bzw. Rückwärtsfahrt zu gewährleisten.

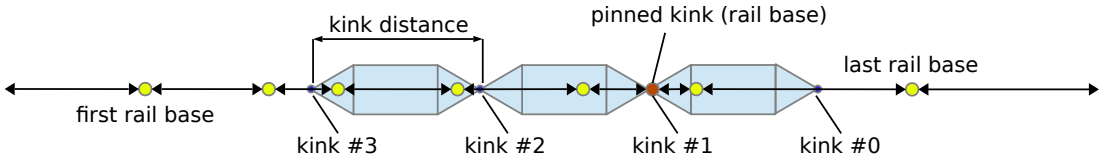


Abb. 2.8: Definition einer geradlinigen virtuelle Schiene mit einem darauf platzierten Roboter. Die Mittelpunkte der Gelenkmodule werden als *kink* und der Abstand zwischen zwei benachbarten Gelenken wird als *kink distance* bezeichnet. Mit dem rot markierten Bezugspunkt (*pinned kink*) wird der Roboter über die Schiene bewegt.

Abbildung 2.9 veranschaulicht die Vorwärtsbewegung eines radgetriebenen schlangenartigen Roboters auf einer geradlinigen virtuellen Schiene. Für die Steuerung der Bewegung werden die Geschwindigkeiten der einzelnen Räder sowie die Winkel für die jeweiligen Gelenkmodule benötigt. Dazu werden ausgehend vom Bezugspunkt *pinned kink* zunächst die kartesischen Positionen der Gelenkmittelpunkte durch Interpolation der jeweils benachbarten Wegpunkte der Schiene berechnet. Aus den Mittelpunkten der Gelenkmodule lassen sich die kartesischen Positionen der Antriebsmodule ableiten. Mit Hilfe dieser Modulpositionen und der inversen Kinematik können iterativ die zur Fortbewegung benötigten Steuergrößen bestimmt werden.

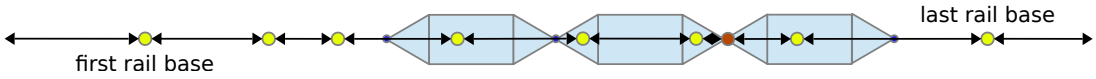


Abb. 2.9: Vorwärtsbewegung des Roboters auf einer geradlinigen virtuellen Schiene.

Eine kontinuierliche Bewegung des Roboter wird durch Einfügen neuer Wegpunkte oder ganzen Pfadsegmenten vor dem letzten Wegpunkt (*last rail base*) gewährleistet. Kurvenfahrten, wie in Abbildung 2.10 dargestellt, werden durch das Auslenken der Schiene in die entsprechende Richtung realisiert, wobei jede Richtungsänderung das Einfügen eines zusätzlichen Wegpunktes erfordert. Beim Befahren der Scheine befinden sich die Gelenkmittelpunkte jederzeit exakt auf der Schiene, jedoch können die Mittelpunkte der Antriebsmodule die Schiene insbesondere bei Kurvenfahrten verlassen. Dadurch beschreiben die Mittelachsen der

Antriebe eine glatte Kurvenbahn während die Gelenkmodule dem Verlauf der virtuellen Schiene folgen.

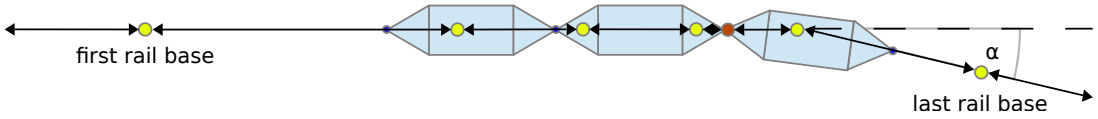


Abb. 2.10: Vorwärtsbewegung des Roboters auf einer virtuellen Schiene mit Kurve. Die Strecke zwischen dem letzten (*last rail base*) und vorletzten Wegpunkt ist um den Winkel α ausgelenkt.

In Kapitel 6.4.1 wird eine Erweiterung der bisher implementierten virtuellen Schiene vorgestellt, die 6-D Wegpunkte mit zusätzlichen Sollwerten für die Drehwinkel des jeweils folgenden Gelenkmoduls kombiniert.

3 Entwurf rekonfigurierbarer Multi-Roboter-Systeme (RMRS)

Aktuelle Serviceroboter für die Inspektion und Wartung sowie der Suche und Rettung von Menschen in Katastrophenszenarien, sind oft sehr robuste und monolithische Systeme. Viele dieser Robotersysteme sind allerdings in realen Einsätzen bisher noch nicht flexibel und effizient genug, da ihre Anpassungsfähigkeit an veränderliche Umweltbedingungen oder Aufgabenstellungen noch nicht hinreichend realisiert wurde. Die Anpassungsfähigkeit sowie Flexibilität und Redundanz kann durch den Einsatz selbst-rekonfigurierbarer Roboter deutlich gesteigert werden. Jedoch existieren solche Robotersysteme hauptsächlich als prototypische Systeme in Forschungs- und Entwicklungslaboren. Für die Anwendung unter den Bedingungen eines realen Einsatzes, sind selbst-rekonfigurierbare Roboter bisher noch nicht oder nur sehr eingeschränkt geeignet. Dies begründet sich dadurch, dass neben einer robusten und gleichzeitig flexiblen Hardware, auch besondere Problemstellungen bei der Entwicklung von Algorithmen zur Planung und Steuerung von Bewegungen sowie der Rekonfiguration entstehen. Abbildung 3.1 stellt monolithische und selbst-rekonfigurierbare Roboter gegenüber und verdeutlicht deren Vor- und Nachteile [44].

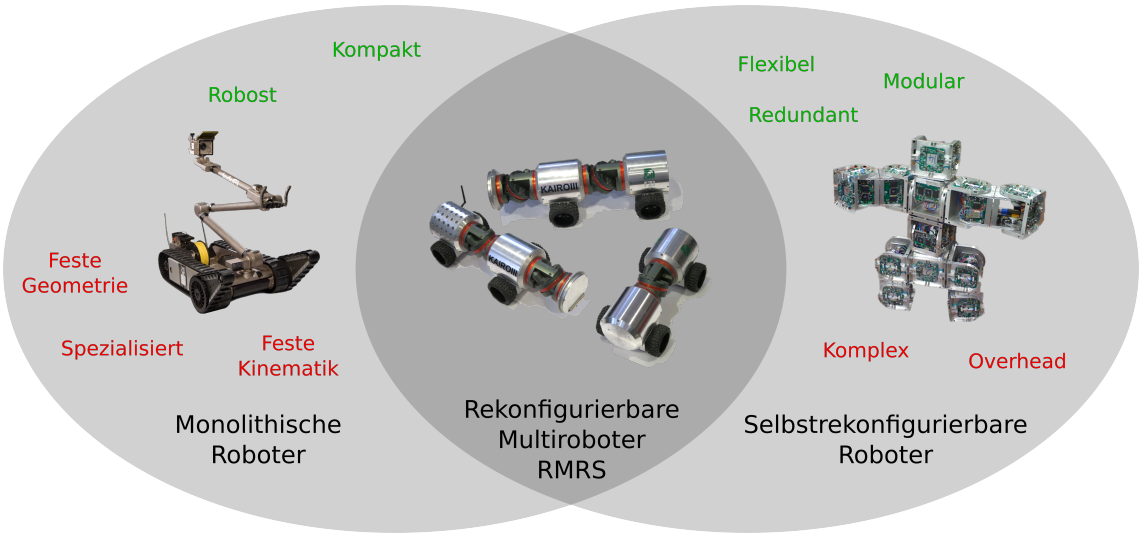


Abb. 3.1: Vergleich zwischen klassisch monolithischen und selbst-rekonfigurierbaren Robotern. Rekonfigurierbare Multi-Roboter-Systeme (RMRS) kombinieren die Vorteile beider Systeme.

In dieser Arbeit werden sogenannte *Rekonfigurierbare Multi-Roboter-Systeme*

(RMRS) betrachtet, die einen Mittelweg zwischen monolithischen Robotern und selbst-rekonfigurierbaren Systemen bilden. RMRS vereinen die Vorteile der **Modularität**, **Flexibilität** und **Redundanz** sowie der **Robustheit** und **Kompaktheit** beider Klassen. Die Einordnung von RMRS wird in Abbildung 3.1 verdeutlicht.

3.1 Erweiterung der Klassifizierung für RMRS

Die in der Literatur vorgestellte Klassifizierung modularer und rekonfigurierbarer Systeme nach der Art der verwendeten Module (Kapitel 2.1) ist bei der Beschreibung von RMRS meinst nicht ausreichend. Neben der Unterscheidung zwischen der Homogenität und der Heterogenität von Systemen lässt sich die Klassifikation um eine geometrische und funktionale Sicht erweitern.

3.1.1 Verfeinerung der Klassifikation

Die Klassifikation nach der Art der Module kann zusätzlich in **Geometrie**, **Funktionalität** und **Schnittstellen** unterteilt werden. Für jede dieser drei Eigenschaften der Module, lässt sich eine Aussage bezüglich der Heterogenität bzw. Homogenität treffen.

- Geometrie: *Homogen* vs. *Heterogen*
- Funktionalität: *Homogen* vs. *Heterogen*
- Schnittstellen: *Homogen* vs. *Heterogen*

Zusätzlich lässt sich durch Betrachtung der Granularität der Konfigurationsklassifikation eine zusätzliche Sicht bzw. Ebene einführen. Jede der drei oben betrachteten Eigenschaften kann entweder auf das gesamte System (**Systemebene**) oder auf einzelne Modulklassen (**Modulebene**) angewendet werden. Die folgenden drei Beispiele verdeutlichen die Unterscheidung zwischen System- und Modulebene sowie die Klassifikation nach Geometrie, Funktionalität und Schnittstellen.

Beispiel 1: KAIRO 3 Gesamtsystem

Bei der Betrachtung von KAIRO 3 als Gesamtsystem (Systemebene) werden zwei verschiedene Modultypen differenziert: Gelenkmodule und Antriebsmodule. Beide Module unterscheiden sich in ihrer Geometrie, Funktionalität und Schnittstellen.

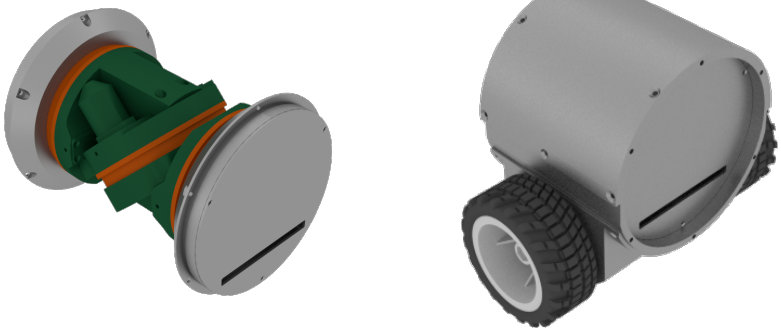
Abbildung 3.2 zeigt die geometrischen Unterschiede beider Modultypen: Gelenkmodule besitzen eine kompakte Form und drei Drehachsen im Vergleich zur

Granularität der Klassifikation:
Systemebene

Geometrie:
Heterogen

Funktionalität:
Heterogen

Schnittstellen:
Heterogen



Gelenkmodul Antriebsmodul

Abb. 3.2: Betrachtung von KAIRO 3 auf Systemebene.

zylindrischen Form und den Rädern des Antriebsmoduls. Die Funktionalität von Gelenkmodulen besteht in der Veränderung des Drehwinkels, Antriebsmodule hingegen sorgen für den Vortrieb des Roboters. Die Schnittstellen werden auf Grund der aktiven und passiven Seite ebenfalls als Heterogen eingeordnet.

Beispiel 2: KAIRO 3 Antriebsmodule

Ein Vergleich der Antriebsmodule von KAIRO 3 untereinander, lässt in Abbildung 3.3 erkennen, dass die Geometrie der Module identisch ist. Auf

Granularität der Klassifikation:
Modulebene

Geometrie:
Homogen

Funktionalität:
Heterogen

Schnittstellen:
Heterogen

Batteriemodul Rechnermodul

Abb. 3.3: Betrachtung der Antriebsmodule von KAIRO 3 (Modulebene).

Grund unterschiedlicher Komponenten innerhalb der Module können diese jedoch verschiedene Aufgaben erfüllen, was zur Heterogenität bezüglich der Funktionalität führt. Die Schnittstellen auch auf der Modulebene sind weiterhin Heterogen, da die mechanische Schnittstelle aus einer aktiven und einer passiven Seite besteht. Somit können Antriebsmodule mit jeweils zwei aktiven Schnittstellen nicht untereinander verbunden werden.

Beispiel 3: Rekonfigurierbare Satellitenbausteine

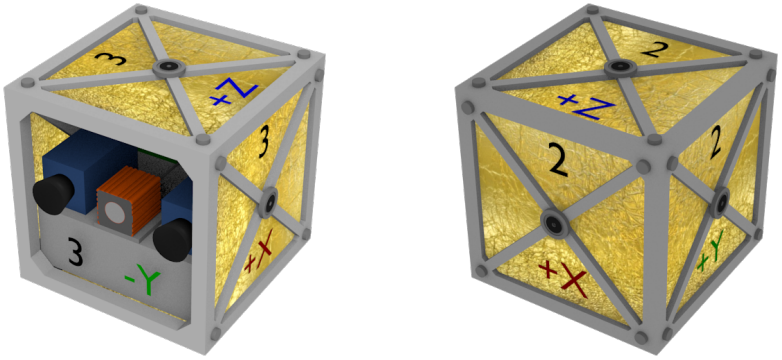
Im Gegensatz zu den KAIRO 3 Modulen sind die in Abbildung 3.4 dargestellten würfelförmigen modularen Satellitenbausteine in ihrer grundlegenden Geometrie immer gleich [58]. Dennoch unterscheiden sich die Module nach ihrem inneren Aufbau und der Funktionalität. So kann beispielsweise ein Kamerabaustein verschiedene Sensoren beinhalten, wobei die rechenintensive Auswertung der Daten in einem separaten Rechnerbaustein stattfindet. Auf Grund des androgynen Design der Schnittstellen[1] sind diese sowohl auf System- als auch auf Modulebene homogen klassifiziert.

Granularität der Klassifikation:
Systemebene

Geometrie:
Homogen

Funktionalität:
Heterogen

Schnittstellen:
Homogen



Kameramodul Rechnermodul

Abb. 3.4: Betrachtung der Geometrie, Funktionalität und Schnittstellen Eigenschaften am Beispiel eines modularen Satellitenbaukastens auf Systemebene.

3.2 Grundlegender Aufbau von RMRS

Rekonfigurierbare Multi-Roboter-Systeme bestehen aus mehreren mobilen Robotern, die wiederum aus standardisierten homogenen oder heterogenen Modulen aufgebaut sind. Zwischen diesen Bausteinen befinden sich kompatible Hardware-Schnittstellen, die ähnlich wie bei selbst-rekonfigurierbaren Systemen, zum Verbinden und Trennen einzelner Module dienen. Im Gegensatz zu den meisten selbst-rekonfigurierbaren Robotern, wird bei RMRS jedoch möglichst viel Intelligenz und Autonomie in den einzelnen Modulen vereint, so dass bereits wenige Bausteine einen eigenständigen Roboter bilden. Zusätzlich können mehrere gleichartige mobile Roboter mit Hilfe der Schnittstellen untereinander gekoppelt werden. Dieser größere Verbund kann entsprechend der Zusammensetzung der Module, verschiedene Ressourcen, wie z. B. Energie oder Rechenleistung teilen und von der vorhandenen Redundanz profitieren.

[1] Alle Schnittstellen sind identisch aufgebaut und können 90° rotationssymmetrisch untereinander verbunden werden.

Durch den Bedarf an nur wenigen Bausteinen für einen eigenständigen mobilen Roboter, wird ebenfalls die Anzahl der insgesamt benötigten Schnittstellen reduziert, was eine Verbesserung der Robustheit des Gesamtsystem mit sich bringt. Die große Herausforderung bei der Entwicklung eines RMRS besteht darin, einen adäquaten Mittelweg zwischen einer möglichst hohen Flexibilität und Robustheit des Gesamtsystems zu finden.

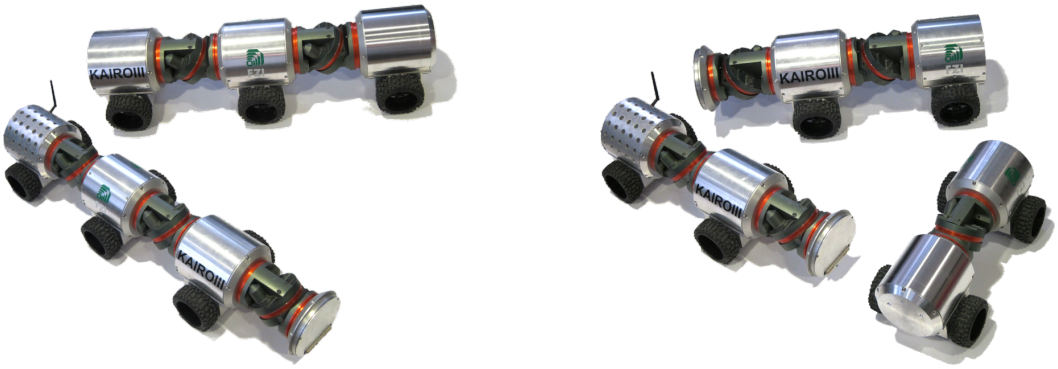
3.2.1 Realisierung eines RMRS am Beispiel von KAIRO 3

Modulare schlangenartige Roboter mit zusätzlichen Rad- oder Kettenantrieben, wie KAIRO 3 (Abbildung 3.5), eignen sich besonders gut als Basis für RMRS. Auf Grund des modularen kettenförmigen Aufbaus besitzt diese Art von Robotern meist einen geringen Querschnitt bzw. Durchmesser und kann somit durch enge Öffnungen und Hohlräume gelangen wie z. B. Wartungsschächte oder Trümmer eines eingestürzten Gebäudes. Zusätzlich besitzen schlangenartige Roboter meist Gelenke zwischen den Modulen, die eine sehr flexible Kinematik ermöglichen. Diese Roboter können sich somit auf sehr engem Raum beziehungsweise um sehr enge Kurven bewegen. Durch den modularen Aufbau mit zusätzlichen Rädern ist es zudem möglich solche Robotersysteme in ihrer Länge d. h. Geometrie variabel zu gestalten. Einen Verbund aus mehreren physikalisch getrennten Robotern kann Aufgaben gleichzeitig an unterschiedlichen Orten erledigen und bei Bedarf Module austauschen oder kombinieren. Eine Steigerung der *Flexibilität* bei RMRS, wird durch das Anordnen funktionaler Module zu unterschiedlichen Konfigurationen aus einem oder mehreren Robotern erreicht.

Abb. 3.5: Konfiguration eines kombinierten Roboters mit insgesamt elf Modulen.

Abbildung 3.5 und Abbildung 3.6 verdeutlichen diese Flexibilität von RMRS an Beispiel dreier unterschiedlicher Konfigurationen von KAIRO 3. Die Robustheit des Robotersystems wird durch den Einsatz redundanter Module sowie Hardware-Schnittstellen erreicht, die es ermöglichen einzelne z. B. defekte Module auszutauschen. Damit kann der Betrieb des Roboters auch im Falle eines Defekts oder Ausfall von Sensoren oder Aktoren sichergestellt werden [44].

Je nach Einsatzszenario eignen sich unterschiedliche Roboterkonfiguration besser. Mehrere kleinere Robotersysteme (siehe Abbildung 3.6b) sind in der Lage parallel verschiedene Bereiche des Einsatzgebietes abzusuchen. Jedoch können



(a) Konfiguration aus zwei Robotern mit jeweils drei Antriebsmodulen.

(b) Konfiguration aus drei einzelnen Robotern mit je zwei Antriebsmodulen.

Abb. 3.6: Drei unterschiedliche Roboterkonfigurationen am Beispiel des rekonfigurierbaren Multi-Roboter-Systems KAIRO 3.

diese Teilsysteme, bestehend aus nur wenigen Bausteinen, keine größeren Hindernisse überwinden. Für diesem Fall ist eine lange schlangenartige Roboterkonfiguration, wie in Abbildung 3.5 dargestellt, deutlich besser geeignet. Durch eine entsprechende Hardware-Rekonfiguration mehrerer kleiner Roboter zu einer längeren Roboterkonfiguration kann eine Umwandlung bzw. Transformation des Roboters während eines Einsatzes realisiert werden. Diese Flexibilität ermöglicht es auf zuvor unbekannte Ereignisse zu reagieren und Robotersysteme an unvorhersehbare Situationen anzupassen. Die folgenden Kapitel zeigen Methoden und Komponenten mit deren Hilfe sich ein solches RMRS umsetzen lässt.

3.3 Flexible Steuerungsarchitektur für RMRS

Die Realisierung eines rekonfigurierbaren Multi-Roboter Systems erfordert eine flexible und adaptive Steuerungssoftware, die unabhängig von der Anzahl verwendeter Bausteine ist und sich selbständig an verschiedene Konfigurationen anpasst. Insbesondere zur Online-Rekonfiguration muss es möglich sein, die Steuerung während der Laufzeit anzupassen, um auf das An- und Abzukoppeln von einzelnen Bausteinen bzw. Roboterteilsystemen (Bausteinketten) flexibel reagieren zu können. Um diese besonderen Anforderungen von RMRS erreichen zu können, wird ein modulares und anpassungsfähiges Framework für den flexiblen Datenaustausch benötigt.

Zunächst wurden verschiedene vorhandene Robotik Frameworks im Hinblick auf die speziellen Anforderungen von RMRS untersucht. Dabei wurden insbesondere die sechs Frameworks **ROS** [59], **ROS 2** [60], **MCA 2.6** [61], **RoboComp** [62], **SmartSoft** [63] und **Kamaro** anhand der Eigenschaften *Modularität*, *Flexibilität*, *Dezentralität*, *Wiederverwendbarkeit*, *Stabilität*, *Echtzeitfähigkeit*, *Programmiersprachen*, *Kompatibilität*, *Anwenderfreundlichkeit*, *Verbreitung*, *Lizenz* und *OpenSour-*

ce Verfügbarkeit miteinander verglichen. Es hat sich gezeigt, dass ROS 2 die meisten der gestellten Anforderungen erfüllt. Trotz des noch sehr frühen Entwicklungsstandes von ROS 2, wurde es für den Entwurf einer flexiblen und anpassbaren Steuerung für RMRS ausgewählt [64].

Abbildung 3.7 verdeutlicht am Beispiel von KAIRO 3 das Konzept zur Steuerung dreier unabhängiger Roboter. Die Roboterteilsysteme (*Hardware Teil 1, 2 und 3*) bestehen jeweils aus mehreren Bausteinen, einem integrierten Steuerungsrechner und einer eigenen Energieversorgung. Jeweils eine Instanz der Steuerungssoftware (*Software Teil 1, 2 und 3*) wird auf dem entsprechenden Steuerungsrechner ausgeführt. Die Teilkomponente (*HW Abstraktion*) erkennt die einzelnen Hardwaremodule des Roboters und empfängt Sensordaten bzw. sendet Steuersignale. Die Berechnung der Kinematik und inversen Kinematik wird mit Hilfe der in Kapitel 2.5.2 vorgestellten virtuellen Schiene getrennt für jeden Hardware Teil realisiert. Zur Steuerung des jeweiligen Gesamtsystems, wird die (*Lokale Steuerung*) eingesetzt, welche unter anderem die Fahrgeschwindigkeit und den Lenkwinkel vorgibt. Jede Instanz der Steuerungssoftware passt sich entsprechend der aktuell verwendeten Anzahl an Bausteinen und deren Topologie an. Einzelne Module oder Modulketten ohne eigene Steuerung können mit Hilfe der in Kapitel 3.4 gezeigten Schnittstelle an eines der aktiven Teilsysteme gekoppelt werden.

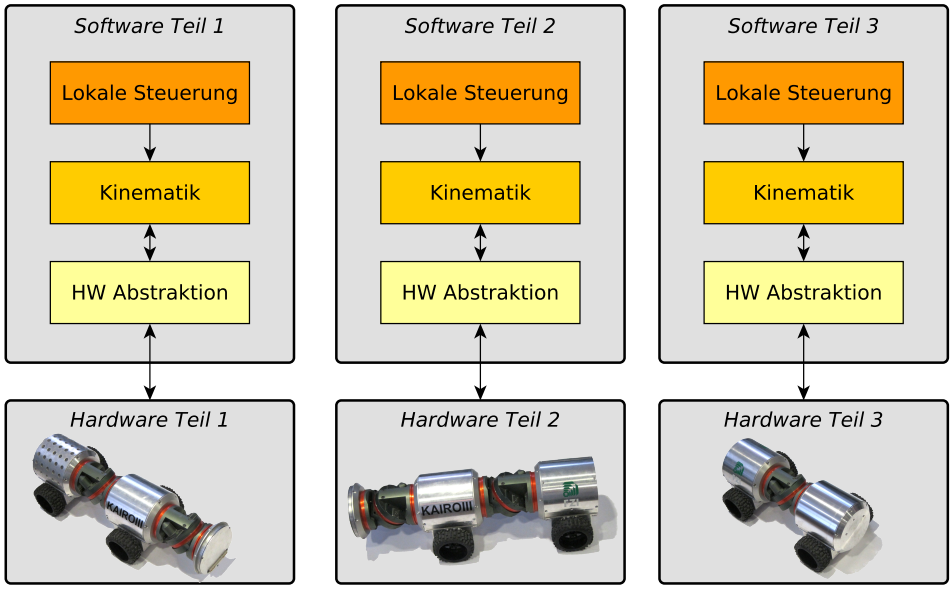


Abb. 3.7: Struktur der flexiblen Steuerungssoftware bei drei unabhängigen Robotern am Beispiel von KAIRO 3 [64].

Die Verbindung von Robotersystemen mit jeweils eigenen Steuerungen erfordert neben der Kopplung der Hardwaremodule ebenfalls eine Verknüpfung der Softwaremodule. Abbildung 3.8 zeigt am Beispiel von KAIRO 3 eine Kombination aus drei eigenständigen Teilsystemen zu einem gesamten Robotersystem.

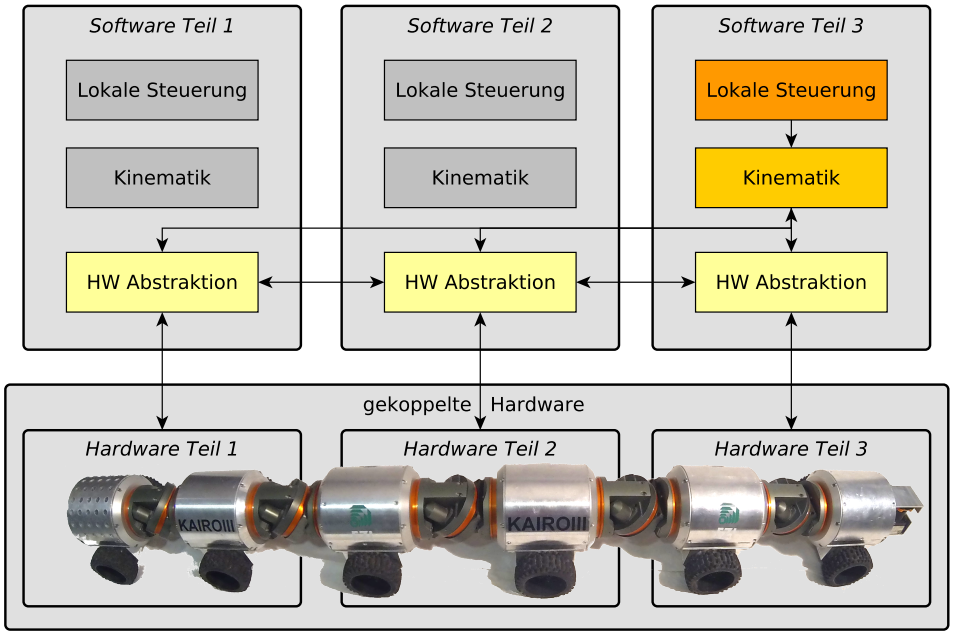


Abb. 3.8: Struktur der flexiblen Steuerung eines kombinierten Roboters am Beispiel von KAIRO 3 [64].

Es wird deutlich, dass die Komponente *HW Abstraktion* weiterhin getrennt auf der entsprechenden Hardware ausgeführt wird. Die *Kinematik* Berechnung mit Hilfe der virtuellen Schiene erfolgt in einer zentralen Instanz die zur Synchronisierung Daten- und Steuersignale mit den einzelnen Hardwarekomponenten austauscht. Für die Steuerung des Gesamtsystems insbesondere zur Bewegungsvorgabe für das erste Modul wird lediglich eine Instanz der *Lokale Steuerung* benötigt. Die nicht benötigten Softwaremodule werden inaktiv geschaltet. Welche der Instanzen die Kinematik Berechnung sowie die Gesamtsteuerung übernimmt wird untereinander ausgehandelt. Beim Ausfall einer Steuerungseinheit oder der Softwarekomponenten zur Gesamtsteuerung bzw. Kinematik Berechnung können mit Hilfe eines Watchdog die entsprechenden inaktiven Instanzen eines anderen Moduls aktiviert werden. Dies trägt zur Erhöhung der Ausfallsicherheit des Gesamtsystem bei.

Speziell für die Steuerung von rekonfigurierbaren Multi-Robotersystemen (RMRS), wurden sogenannte Entitäten entwickelt und innerhalb der ROS 2 Architektur umgesetzt. Entitäten ermöglichen die direkte Abbildung von Hardwarekomponenten (Sensoren und Aktoren) auf entsprechende Softwaremodule und können nach der in Abbildung 3.9 dargestellten Baumstruktur eingeteilt werden. Ausgehend von der Basisklasse *EntityBase* wird die generische Klasse *Entity* abgeleitet, welche die Basisklasse um eine Kommunikationsschnittstelle mit einer austauschbaren Datenstruktur *MessageType* erweitert. Die davon abgeleiteten Klassen JointSensor, JointActor, DriveSensor und DriveActor repräsentieren konkrete Hardware Sensoren und Aktoren und bilden die kleinsten nicht unterteilbaren Komponenten. Zur Strukturierung und Hierarchisierung

eines Robotersystems wurden die Klassen *Unit*, *Part* und *Robot* eingeführt. Die davon abgeleiteten Klassen *DriveUnit*, *JointUnit*, *Segment* und *VirtualRobot* ermöglichen das Verknüpfen von Instanzen anderer Klassen zu einem Gesamtsystem. Ein *Segment* repräsentiert die Kombination eines Antriebsmoduls und eines Gelenkmoduls, welche als eigenständige Einheit betrachtet werden kann.

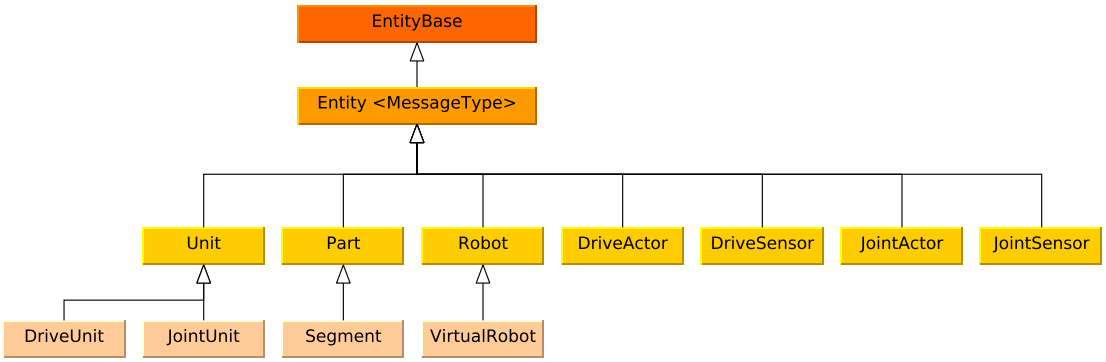


Abb. 3.9: Klassifikation von Entitäten am Beispiel der Steuerungsarchitektur von KAIRO 3 [64].

Zur Abbildung der Struktur bzw. des Hardwareaufbaus eines Robotersystems, werden die für jede Hardwarekomponente instanziierten Softwaremodule in der Steuerungsarchitektur miteinander verknüpft. Durch die Möglichkeit der Veränderung dieser Verbindungen wird die für RMRS benötigte Flexibilität der Steuerung erreicht. Abbildung 3.10 zeigt am Beispiel von KAIRO 3 die baumartig verknüpften Entitäten von zwei unabhängigen Robotern, die als virtueller Roboter (*VirtualRobot*) im einem einzigen RMRS zusammengefasst sind. Durch den Einsatz von Entitäten lässt sich eine Hierarchisierung und somit eine klare Trennung zwischen Daten-, Funktion- und Anzeigeschicht erreichen [64].

3.3.1 Kommunikation und Bussysteme zur Umsetzung der flexiblen Steuerung

Ein essentieller Bestandteil einer flexiblen und verteilen Steuerung ist die effiziente Kommunikation zwischen den einzelnen Softwaremodulen und Bausteinen eines Robotersystems. Zur Realisierung der oben vorgestellten Steuerungskonzepte wurde das Robotik Framework ROS 2 gewählt. Die Kommunikationsarchitektur von ROS 2 basiert auf dem Data Distribution Service (DDS) [65] und ermöglicht eine modulare und verteilte Softwarestruktur [66]. Mit Hilfe dieser Kommunikationsschicht können sich einzelne in einem Netzwerk befindliche ROS Module (Nodes) finden, verbinden und über gemeinsame Kommunikationskanäle Daten austauschen. Die Spezifikation eigener Datenformate bzw. Nachrichten (Messages) ermöglicht die Übertragung von Basisdatentypen für Steuersignale bis hin zu komplexen Datenstrukturen oder großen Datenmengen wie z. B.

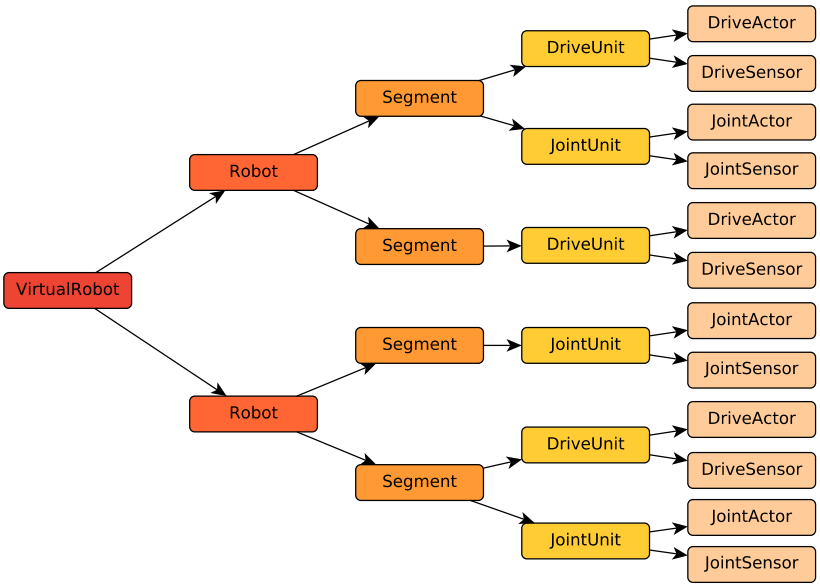


Abb. 3.10: Aufbau und Strukturierung der flexiblen Steuerungssoftware basierend auf dem Entitäten Konzept am Beispiel von KAIRO 3 [64].

Punktewolken von Lasersensoren. Entsprechend der Wichtigkeit der Daten können für die einzelnen Datenkanäle unterschiedliche Qualitätsparameter (Quality of Service) eingestellt werden, die unter anderem festlegen wie mit Datenverlust oder Datenbeschädigungen umgegangen wird. Die Kommunikation zwischen den einzelnen Nodes in einem ROS 2 Softwaresystem erfolgt dabei Netzwerk transparent und ist echtzeitfähig. Insbesondere die Hardware Abstraktionsschicht und die dabei eingesetzten Bussysteme und Protokolle sind stark von den verwendeten Hardware- und Elektronikkomponenten abhängig. Aus diesem Grund werden im Folgenden zwei unterschiedliche Möglichkeiten zur Umsetzung aufgezeigt.

Realisierung der Kommunikation mittels CAN-Bus

Bedingt durch den besonderen Hardwareaufbau von KAIRO 3 werden in jedem Hardwaremodul spezielle Steuerungsplatinen (UCoMs) [67] verwendet. Diese jeweils aus einem Mikrocontroller und einem FPGA bestehenden Steuerungsboards dienen zur Regelung von bis zu drei Motoren. Zudem wird das Auslesen der entsprechenden Sensorik zur Bestimmung von Gelenkpositionen, Drehgeschwindigkeiten und Motorströmen ermöglicht. Die Steuerungsplatinen werden untereinander mit einem CAN-Bus verknüpft und erhalten jeweils eine eindeutige Kennung (ID). Die Softwarekomponente *HW Abstraktion* sammelt Informationen aller am CAN-Bus angeschlossenen Module und erzeugt entsprechende Software Entitäten für alle Sensoren und Aktoren. Das zur Kommunikation auf dem CAN-Bus eingesetzte Protokoll wurde speziell für die hardware-nahe Regelung von Robotern entwickelt [67]. Die Bestimmung der Modulanordnung (Topologie)

erfolgt durch eine spezielle *Init* Signalleitung parallel zum CAN-Bus und eine Softwareroutine zum Ermitteln der Modulsequenz. Basierend auf der Modulsequenz wird ein Abhängigkeitsgraph der Entitäten erzeugt, wie in Abbildung 3.10 dargestellt. Beim An- oder Abkoppeln von Hardwaremodulen wird eine direkte Verbindung des CAN-Bus hergestellt bzw. getrennt und die Initialisierungsroutine zur Erkennung der Topologie wird erneut ausgeführt. Die Entitäten und der Abhängigkeitsgraph werden entsprechend an die neue Topologie angepasst. Die Integration einer entsprechenden Datenkopplung des CAN-Bus in die Schnittstelle wird in Kapitel 3.4.2 am Beispiel von KAIRO 3 gezeigt. Das entsprechende ROS 2 Softwaremodul (*HW Abstraktion*) wird jeweils auf einem eingebetteten Rechner in einem der Antriebsmodule ausgeführt und stellt die gesammelten Informationen der übergeordneten Kinematik Berechnung und der Robotersteuerung zur Verfügung.

Realisierung der Kommunikation am Beispiel modularer Satelliten

Die im Rahmen des iBOSS Projekts [9] entwickelten modularen Satellitenbausteine können mit den integrieren Schnittstellen mechanisch und elektrisch gekoppelt werden. Zur Datenübertragung zwischen den Bausteinen wird ein optischer Ethernet-Bus eingesetzt, der die einzelnen Bausteine vernetzt. Ein verwalteter Switch in jedem Baustein verbindet die bis zu sechs Schnittstellen und den internen Steuerungsrechner miteinander. Zudem ermöglicht der Switch die automatische Auflösung von Ringschlüssen durch das Spannbaum-Protokoll (STP). Die exakte Zuordnung der Ethernet Schnittstellen auf die jeweiligen Seitenflächen eines Bausteins ermöglicht das Netzwerkverwaltungsprotokoll (SNMP). Dank dieser Protokolle ist es möglich die Nachbarbausteine und deren Positionen aus der Busstruktur des Netzwerk zum ermitteln. Für die Bestimmung der Anordnung des Gesamtsystem wird zusätzlich noch die Orientierung der einzelnen Bausteine benötigt, da die Schnittstellen jeweils um 90° rotationssymmetrisch gekoppelt werden können. Dazu sind spezielle optische Sensoren in die Seitenfläche der Bausteine integriert, die den entsprechenden Rotationsversatz liefern [9]. Die gewonnen Informationen zu den Nachbarbausteinen werden mit Hilfe der Softwarekomponente zur Hardwareabstraktion (*HW Abstraktion*) vorgehalten. Die *HW Abstraktion* verwaltet zudem alle in einem Baustein vorhandenen Komponenten und erzeugt entsprechende Entitäten für diese. Diese Kommunikationsebene wird mit Hilfe der oben vorgestellten flexiblen Steuerungsarchitektur umgesetzt, die auf der ROS 2 Kommunikation beruht. Basierend auf den Nachbarschaftsbeziehungen kann eine Topologie bzw. ein Abhängigkeitsgraph des Gesamtsystem aufbaut werden, sodass die einzelnen Bausteine gezielt miteinander kommunizieren und z. B. Steuerungs- und Telemetriedaten austauschen können. Das implementierte Protokoll für die Kommunikation zwischen den einzelnen Softwarekomponenten nutzt ROS 2 Nachrichten (Messages, Services und Actions) die aus teilweise vor- und selbstdefinierten Datentypen bzw. Datenstrukturen bestehen.

Konzept einer kombinierten Kommunikation und verteilten Steuerung

Die zuvor vorgestellte direkte Kopplung des CAN-Bus am Beispiel von KAIRO 3 hat den Vorteil, dass die gesamte Kommunikation zur Regelung der Motoren und Synchronisierung der einzelnen Hardwaremodule in Echtzeit ausgeführt wird. Dank der Kopplung des Datenbus ist es ebenfalls möglich, passive Module, die keinen eigenen Steuerrechner besitzen, anzudocken und anzusteuern. Diese Art der Verbindung des Bussystems erschwert jedoch die Realisierung einer echten verteilten Steuerung. Wie in Abbildung 3.8 dargestellt, laufen alle Daten auf einem Steuerungsrechner zusammen, der die Kinematik Berechnung und Gesamtsystemsteuerung durchführt. Die Steuerungsrechner in den anderen Hardwaremodulen bleiben ungenutzt. Zudem erweist sich die Kopplung und Reinitialisierung des CAN-Bus bei jedem Kopplungsvorgang als aufwendig.

Aus diesen Gründen wird im Folgenden eine Kombination der oben vorgestellten Bussysteme und Kommunikationskanäle diskutiert. Auf der untersten Ebene wird weiterhin der hardware-nahe CAN-Bus verwendet. Dieser wird jedoch nicht über die Schnittstellen verbunden sondern jeweils lokal für jedes Segment des Roboters getrennt betrieben. Die kleinste Einheit besteht aus je einem Antriebs- und Gelenkmodul. Mindestens ein Antriebsmodul pro Segment besitzt einen eigenen Steuerungsrechner der alle an einem lokalen CAN-Bus angeschlossenen Hardwaremodule verknüpft. Der Steuerrechner führt zusätzlich ein ROS 2 Softwaremodul zur Hardware Abstraktion aus, welches die lokale Topologie über die CAN-Bus Initialisierung bestimmt und entsprechende Entitäten in einem Abhängigkeitsgraph erzeugt. Die einzelnen Steuerrechner werden untereinander mittels Ethernet-Bus und entsprechender Switches in den Antriebsmodulen verbunden, wobei die Datenschnittstelle von KAIRO 3 eine physikalische Verbindung des Ethernet-Bus herstellt. Neben einer direkten elektrischen Kopplung ist ebenfalls eine drahtlose Verbindung des Ethernet-Bus möglich. Mit Hilfe des Netzwerk zwischen den Steuerrechnern und dem dezentral organisieren ROS 2 Framework können die Softwaremodule Netzwerk transparent miteinander kommunizieren. Die Ermittlung der Anordnung der einzelnen Teilsysteme wird wie bei der rein Ethernet-Bus basierten Methode, mittels STP Protokoll und einem Algorithmus zur Bestimmung der Busstruktur im Netzwerk realisiert.

Diese Struktur und Vernetzung der Hardware Abstraktion macht es ebenfalls möglich die Kinematik Berechnung auf die einzelnen Steuerrechner zu verteilen. Auf Grund der linearen Kinematik von KAIRO 3 ermöglicht der iterative NSVS-Algorithmus die Berechnung der Gelenkwinkel und Radgeschwindigkeiten aller mit dem lokalen CAN-Bus verbunden Antriebs- und Gelenkmodule. Die benötigten Informationen, wie Position und Orientierung der jeweiligen Nachbarsegmente werden in jedem Zeitschritt über ROS Nachrichten synchronisiert. Alle Softwaremodule zur Kinematik Berechnung benötigen ebenfalls Zugriff auf die virtuelle Schiene. Diese wird entsprechend global von der übergeordneten Gesamtsystemsteuerung erzeugt und bei jeder Aktualisierung als ROS Nachricht im

Netzwerk verteilt. Das Softwaremodul zur Steuerung des Gesamtsystems ist weiterhin monolithisch aufgebaut und kann auf einem beliebigen Steuerungsrechner ausgeführt werden. Die hochfrequenten Sensor- und Steuersignale für die Regelung der Antriebe werden gemeinsam mit den Daten der virtuellen Schiene und Sensoren zur Umwelterfassung über den selben Ethernet-Bus übertragen. Mit Hilfe der ROS 2 Parameter zur Servicequalität kann sichergestellt werden, dass die Regelungsdaten immer Vorrang vor den Nutzdaten haben.

3.4 Hardwareschnittstelle für RMRS

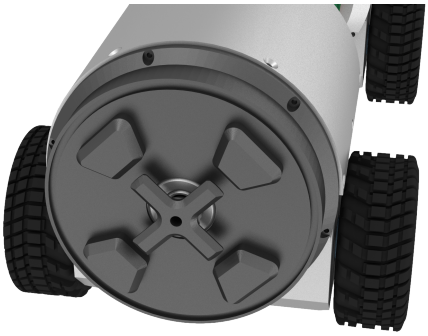
Robuste Hardwareschnittstellen zum An- und Abkoppeln einzelner Module bzw. von Bausteingruppen sind neben der flexiblen und adaptiven Steuerung einer der wichtigsten Bestandteile von rekonfigurierbaren Robotersystemen. Die Anforderungen an eine Hardwareschnittstelle bzw. ein Kopplungssystem können abhängig von den Modulen und des Robotersystems stark variieren. Deshalb werden meistens speziell auf ein RMRS zugeschnittene Schnittstellen entwickelt und implementiert.

3.4.1 Mechanische Schnittstelle

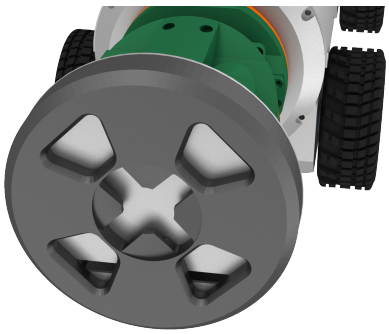
Bei der Konzeption und Entwicklung der mechanischen Schnittstelle für KAIRO 3 wurde vorausgesetzt, dass sich trotz des hohen Gewichts mindestens drei Module gleichzeitig durch den Roboter anheben lassen. Dies erfordert eine besondere Stabilität der mechanischen Schnittstelle zur Übertragung der hohen Kräfte. Für das zuverlässige und sichere Verbinden der beiden Schnittstellenpaare, sollen Ungenauigkeiten und Positionierungsfehler während des Kopplungsvorgangs ausgeglichen werden. Die Schnittstelle soll in der Lage sein einen translatorischen Versatz $d_{translatorisch}$ zur Mittelachse von mindestens 5 mm bei einem maximalen axialen Rotationswinkel $\omega_{rotatorisch}$ von 1° zu kompensieren. Zusätzlich wird gefordert, das ein Winkelversatz α zur Mittelachse von bis zu 2° ausgeglichen werden kann.

Konzept der mechanischen Schnittstelle für KAIRO 3

Zunächst wurden verschiedene mechanische Kopplungsmechanismen, die den oben genannten Anforderungen entsprechen, untersucht und daraus ein speziell für KAIRO 3 optimiertes Design abgeleitet [7]. Der grundlegende Aufbau der mechanischen Schnittstelle besteht, wie in Abbildung 3.11 dargestellt, aus zwei gegensätzlichen Seiten, einer aktiven und einer passiven Schnittstelle. Die Kopplung der beiden Seiten miteinander, erfolgt nach dem Prinzip eines Bajonettverschluss. Im Zentrum der aktiven Schnittstelle befindet sich ein drehbarer Riegel (Abbildung 3.11a), der ein- und ausgefahren werden kann.



(a) 3D Modell der aktiven Schnittstelle an einem KAIRO 3 Antriebsmodul.



(b) 3D Modell der passiven Schnittstelle an einem KAIRO 3 Gelenkmodul.

Abb. 3.11: Gerenderte 3D Ansichten der aktiven und passiven KAIRO 3 Schnittstelle.

Dieser Riegel passt in ein entsprechendes Gegenstück der passiven Schnittstelle (Abbildung 3.11b) und lässt sich durch einen Anschlag blockieren. Die Bewegung des Riegels wird durch eine Hohlwelle und entsprechenden Lagerungen realisiert. Eine detaillierte Beschreibung des Kopplungsvorgang und dessen Ablauf folgt im nächsten Abschnitt. Zur Vereinfachung der Positionierung während der Kopplung des Schnittstellenpaar wurden spezielle Strukturen mit Fasen und Aussparungen auf den jeweils gegenüberliegenden Seiten angebracht. Diese ermöglichen eine Kompensation von Ungenauigkeiten bei der Positionierung beider Schnittstellen. Der maximale Versatz d_{max} der sich ausgleichen lässt, kann mit Hilfe der Gleichung 3.1 berechnet werden. Bei der konzipierten Schnittelle erreicht dieser Wert unter der Annahme von $d_{translatorisch} = 5$ mm, $\omega_{rotatorisch} = 1°$ und $r_{Struktur} = 120$ mm theoretisch bis zu 7 mm [7].

$$d_{max} = d_{translatorisch} + r_{Struktur} \cdot \sin(\omega_{rotatorisch}) \approx 7\ mm \tag{3.1}$$

Kopplungsvorgang

3.4.2 Elektrische und Datenschnittstelle

Der in Abbildung 3.12 dargestellte Ablauf eines Kopplungsvorgangs der mechanischen Schnittstelle wird in vier grundlegende Schritte unterteilt. In Schritt 1 ist die Schnittstelle geöffnet und der Riegel in seiner Nullstellung. Im zweiten Schritt fährt die Schnittstelle durch eine relativ Bewegung des Roboters zusammen. Dabei wird der Riegel durch die kreuzförmige Öffnung der passiven Schnittstelle geführt. In Schritt 3 wird der Riegel gedreht bis die Rotation durch den Anschlag in der passiven Schnittstelle blockiert wird. Bei dieser Drehung bewegen sich die Flügel des Riegels nach dem Prinzip eines Bajonettverschluss in die feste Struktur neben der Öffnung der passiven Schnittstelle. Durch weiteres drehen des

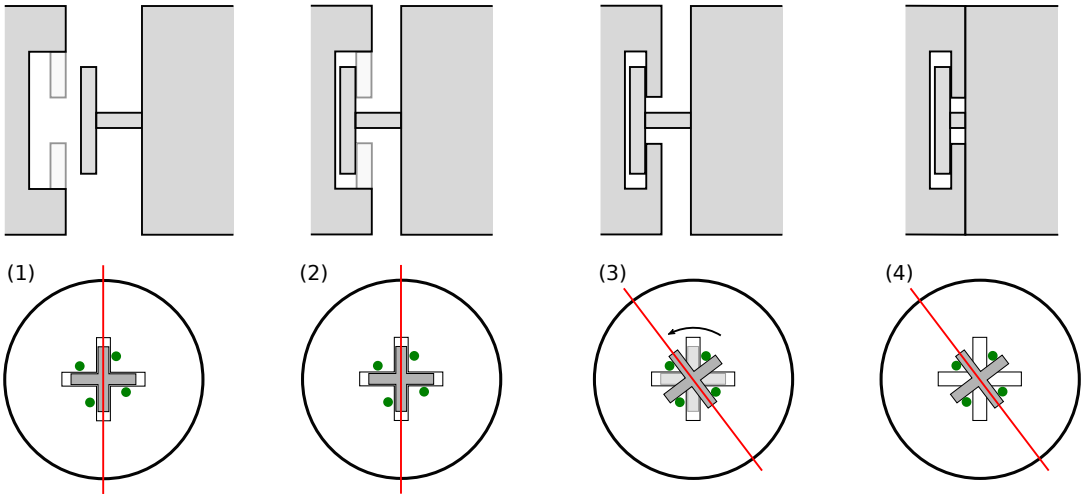


Abb. 3.12: Ablauf eines Kopplungsvorgangs während des Schließens der Schnittstelle. Die obere Reihe zeigt jeweils die Seitenansicht der passiven und aktiven Schnittstelle. Darunter ist die entsprechende Rotation des Riegel in der Vorderansicht dargestellt [8].

blockierten Riegel, entsteht in Schritt 4 eine translatorische Bewegung[2] die das Schnittstellenpaar zusammen zieht und somit eine kraft-schlüssige Verbindung herstellt. Zum Lösen der Schnittstelle wird der Riegel in die entgegengesetzte Richtung gedreht. Der Riegel dreht sich nach dem Lösen zurück in die Nullstellung über der kreuzförmigen Öffnung und wird dort von der passiven Schnittstelle arretiert. Ist der Riegel komplett ausgefahren, kann der Roboter getrennt werden [8].

Zusätzlich zur mechanischen Verbindung müssen bei RMRS meist Daten und Energie zwischen den einzelnen Modulen übertragen werden. Drahtlose Verbindungen bieten auf den ersten Blick die flexibelste und einfachste Möglichkeit zur Datenübertragung. Jedoch gibt es den entscheidenden Nachteil der Unsicherheit z. B. durch Übertragungsstörungen oder Latenzen. Aus diesem Grund wird vor allem bei echtzeitfähigen Regelungen auf einen physikalischen Kontakt in Form von Datenleitungen mit entsprechenden Steckverbindungen zurückgegriffen. Neben der Kommunikation ist die Übertragung elektrischer Energie ein weiterer sehr wichtiger Aspekt bei RMRS. Im Normalfall benötigen die meisten Module eines RMRS im operativen Betrieb eine kontinuierliche Stromversorgung. Eine weit verbreitete und flexible Lösung nutzt Akkumulatoren bzw. Batterien in jedem einzelnen Modul. Diese Variante ermöglicht den autarken Betrieb der Module ohne Energieübertragung. Jedoch beeinflussen ein unterschiedlich hoher Energieverbrauch sowie verschiedene Ladezustände einzelner Batterien die

[2]Die translatorische Bewegung entsteht durch Lagerung des Riegel in einer Hohlwelle mit Innengewinde. Bei Rotation der Hohlwelle drehen sich sowohl Hohlwelle als auch Riegel mit der gleichen Geschwindigkeit. Wird der Riegel blockiert, dreht sich die Hohlwelle weiter und das Gewinde zwischen beiden erzeugt eine translatorische Schraubbewegung ähnlich einer Spindel.

Komplexität und die gesamte Einsatzdauer eines solchen Systems. Zudem steigt der Aufwand zum Laden der Batterien, da jedes einzelne Modul geladen werden muss. Diese Nachteile können durch Einführung einer elektrischen Schnittstelle verbessert werden. In Kombination mit dem Einsatz dedizierter Energieversorgungsmodule, dem Ausgleich der Ladestände zwischen den Modulen und dem gleichzeitigen Laden aller Module steigert dies die Flexibilität und Handhabbarkeit des Gesamtsystems.

Konzept der elektrischen und Datenschnittstelle für KAIRO 3

Die Übertagung von Daten und elektrischer Energie zwischen den Modulen von KAIRO 3 wird, wie in Abbildung 3.13 schematisch veranschaulicht, durch eine Kombination aus direkten elektrischen Kontakten und einer drahtlosen Verbindung realisiert. Die Übertragungskanäle zur Durchleitung der Stromversorgung

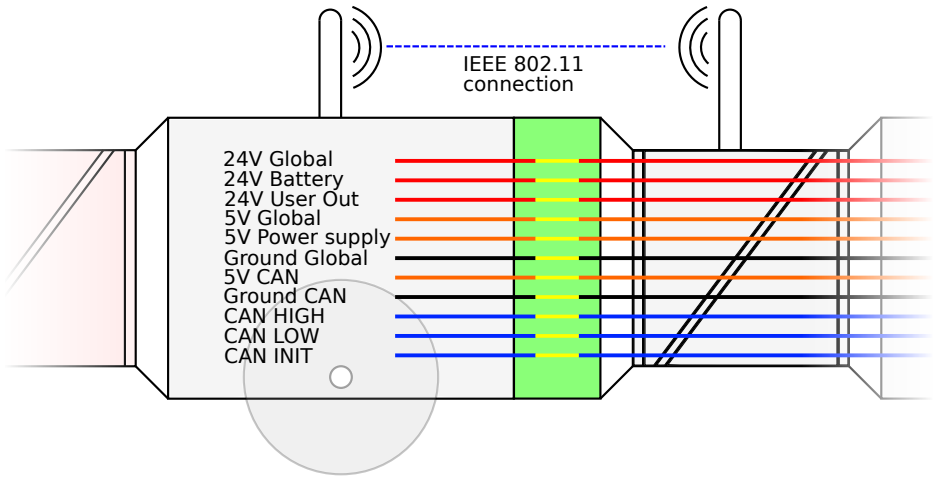


Abb. 3.13: Kanäle zur direkten Daten- und Energieübertragung zwischen jeweils zwei benachbarten KAIRO 3 Modulen [8].

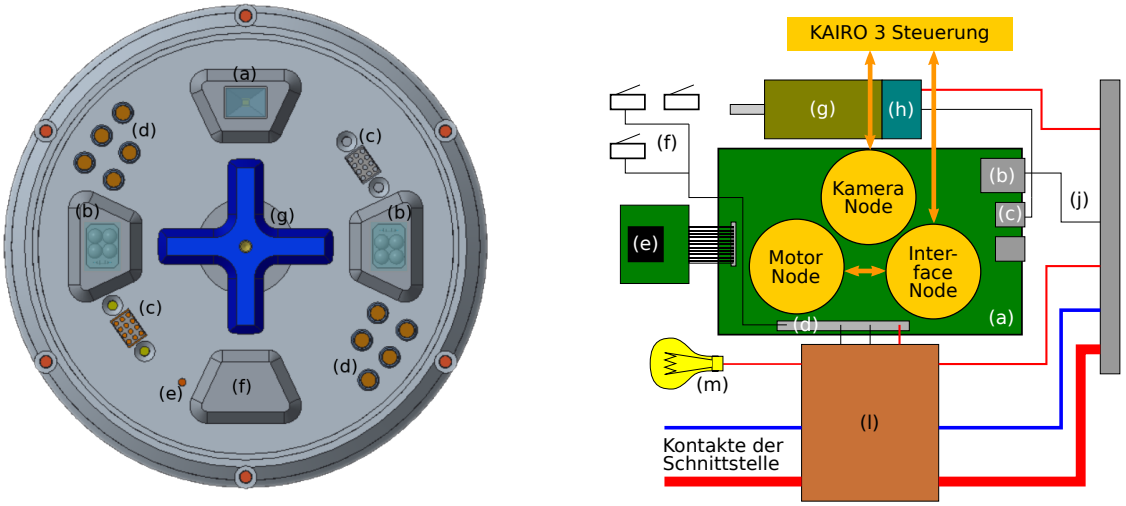
(*24 V Global*, *24 V Battery*, *24 V User Out*, *5 V Global* und *Ground Global*) erfordern hochstromfähige Kontakte. Diese wurden in Form von Federkontakten selbst entwickelt und an die Geometrie der Schnittstelle von KAIRO 3 angepasst. Die Alternative einer Übertragung z. B. mittels Induktion wäre für die benötigten Ströme von bis zu 10 A in der momentanen Umsetzung zu komplex und sehr aufwändig in das bestehende System zu integrieren.

Zur Gewährleistung der Synchronität der gekoppelten Module und zur Übertragung von Sensor- und Steuersignale zur hardwarenahen Regelung des Roboters, sollen größere Latenzen und Ausfälle bei der Datenübertragung vermieden werden. Dies kann am Besten mit Hilfe einer direkten und physikalischen Verbindung der Signalleitungen des Bussystems zwischen den Modulen erreicht werden. Aus diesem Grund werden zur Übertragung geringerer Ströme

(*5 V Power Supply*, *5 V CAN*, *Ground CAN*) und zur Kopplung der Signalleitungen des CAN-Bus von KAIRO 3 (*CAN HIGH*, *CAN LOW* und *CAN INIT*) ebenfalls Federkontakte eingesetzt. Diese besitzen jedoch einen geringeren Querschnitt und sind für weniger Leistung ausgelegt. Anstelle des CAN-Bus können diese Signalleitungen auch zur Kopplung des Ethernet-Bus genutzt werden. Dies ermöglicht die Realisierung der in Kapitel 3.3.1 vorgestellten Ethernet-basierten Kommunikation zwischen den Robotersegmenten.

Die Kommunikation zwischen mehreren entkoppelten Robotern erfolgt mittels drahtloser Kommunikation in Form einer Ethernet-Bus Verbindung nach dem Standard IEEE 802.11. Wie in Kapitel 3.3 beschrieben, findet über diese Verbindung ein Austausch der Positionen und Hardwarekonfigurationen von physikalisch getrennten Robotersystemen statt. Für den Austausch dieser Daten wird keine Datenübertragung mit harten Echtzeitanforderungen benötigt.

Die Anordnung der Hochstrom-Federkontakte (d) sowie der Federkontakte zur Datenübertragung (c) auf der aktiven Schnittstelle ist in Abbildung 3.14a dargestellt. Zur Steigerung der Fehlertoleranz und Redundanz wurden die Kontakte zur Stromübertragung um 180° gedreht dupliziert. Die Kontakte auf der passiven Schnittstelle sind dementsprechend exakt gegenüberliegend angeordnet. Weitere Sensorik sowie das Steuerungskonzept der Schnittstelle wird im folgenden Abschnitt 3.4.3 erläutert [8].



(a) Vorderansicht der aktiven Schnittstelle und deren Komponenten.

(b) Schaubild der Elektronik und Sensoren in der aktiven Schnittstelle.

Abb. 3.14: Integration der Elektronikkomponenten und Sensorik zur Steuerung der KAIRO 3 Schnittstelle [8].

3.4.3 Steuerung und Sensorik der Schnittstelle

Das Konzept der integrierten mechanischen, elektrischen und Datenschnittstelle ist in Abbildung 3.14a dargestellt. Zur Steuerung der aktiven Schnittstelle und zur Unterstützung des Kopplungsvorganges wurden neben den redundanten Federkontakten (c) und (d) ebenfalls eine Kamera (a) und zwei LED Beleuchtungen (b) eingebaut. Sowohl die Kamera als auch die Beleuchtung werden für das in Kapitel 3.5 beschriebene Autonome Docking von zwei eigenständigen Robotern benötigt. Zusätzlich beinhaltet die Schnittstelle noch Kontaktschalter (e) sowie Endschalter für die minimale und maximale Position des Riegel (g). Spezielle mechanische Strukturen (f) dienen dem Ausgleich von Ungenauigkeiten beim Koppeln der Schnittstelle.

Mit Hilfe der in Abbildung 3.14b gezeigten Elektronik- und Softwarekomponenten wird die Steuerung der aktiven Schnittstelle realisiert. Die Komponente (a) stellt einen Einplatinen-Computer dar (z. B. Raspberry Pi), auf dem die Schnittstellensteuerung ausgeführt wird. Die Stromversorgung (b) wird durch die *5 V Global* Leitung der elektrischen Schnittstelle (j) sichergestellt. Zur Bewegung des Riegel wird der Motor (g) mit integrierter Regelung (h) über eine serielle Schnittstelle (c) angesteuert und über *24 V Global* mit Strom versorgt. Eine selbst entwickelte Platine (l) ermöglicht das An- und Abschalten aller stromführenden Kontakte und der CAN-Bus Leitungen sowie das Schalten der LED Beleuchtung (m). Die Platine (l) sowie die Endschalter (f) werden über die frei konfigurierbaren Ein-/Ausgabe Kontakte (d) des eingebetteten Computers angesteuert bzw. ausgelesen. Die Kamera (e) ist direkt über eine integrierte Schnittstelle mit dem Einplatinen-Computer verbunden [8].

Software zur Steuerung der Schnittstelle

Die Software zur Steuerung der Schnittstelle wurde aus Gründen der Flexibilität und Kompatibilität mit der Steuerung von KAIRO 3 im Framework ROS realisiert. Die in Abbildung 3.14b dargestellte *Motor Node* sendet und empfängt Daten zur Regelung des Motors und steuert damit die vorwärts und rückwärts Bewegung des Riegel. Eine *Interface Node* implementiert den in Abbildung 3.15 dargestellten Zustandsautomaten. Die geöffnete Schnittstelle befindet sich im Zustand *Undocked* und wechselt nach erfolgreichem Schließen (docking) zu *Docked*. Das Öffnen (undocking) erfolgt invertiert. In einem Fehlerfall während des Öffnens, Schließens oder Initialisierens, wechselt die Steuerung in den *Error* Zustand. Aus dem Fehlerzustand kann in den undefinierten Zustand *Undefined* (unbekannte Position des Riegel) gewechselt werden. Durch Ausführen der Initialisierungsroutine wechselt der Zustandsautomat bei erfolgreicher Bestimmung, in den entsprechenden Zustand *Docked* oder *Undocked*. Abhängig von den Sensordaten der Schnittstelle und der Position des Riegels werden unterschiedliche Drehrichtungs- und Geschwindigkeitsvorgaben für den Motor erzeugt. Zudem

werden die stromführenden Kontakte nach erfolgreichem Verbinden bzw. vor dem Trennen der Schnittstelle entsprechend geschaltet [8].

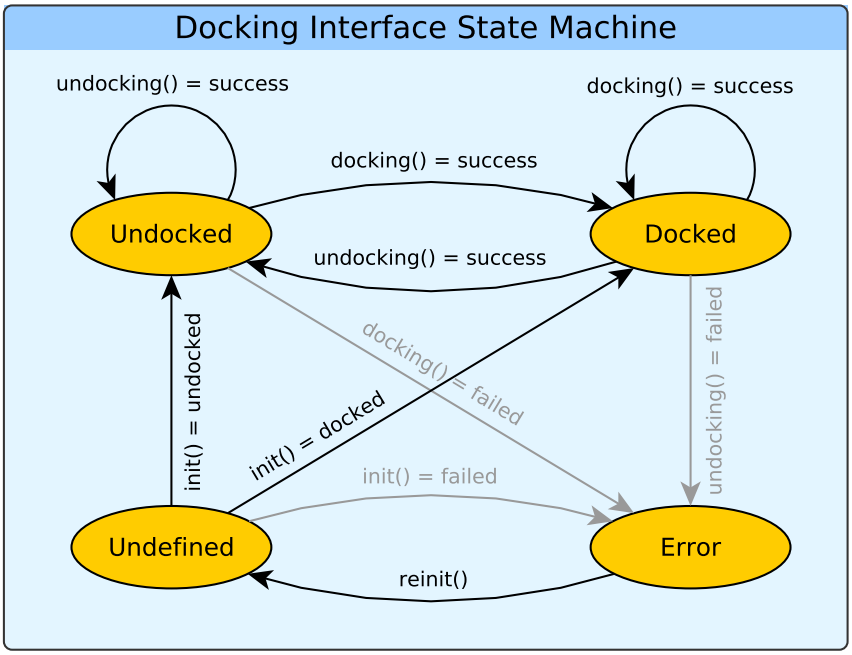


Abb. 3.15: Zustandsautomat zur Steuerung der mechanischen Schnittstelle [8].

3.5 Autonomes Docking mit Visual Servoing

Das Trennen einzelner Module oder eines Teilsystems von einem RMRS lässt sich bereits mit Hilfe der vorgestellten Hardware-Schnittstelle und der flexiblen Steuerungssoftware realisieren. Ein weiterer Bestandteil eines RMRS, stellt das selbstständige Finden und Zusammenführen zweier unabhängiger Robotersysteme dar. Dazu werden verschiedene sensorbasierte Ansätze untersucht und ein kamerabasiertes Visual Servoing für das autonome Koppeln von RMRS entwickelt. Es wird angenommen, dass eine Kamera in die aktive Schnittstelle (Abbildung 3.11a) integriert ist und sich ein zweiten Roboter mit der passiven Schnittstelle (Abbildung 3.11b) im Bildbereich befindet. Der Roboter mit der Kamera, auch aktiver Roboter genannt, wird auf den passiven Roboter zu bewegt, wobei versucht wird den passiven Roboter stets im Bild des ersten Roboters zu halten. Die für das kamerabasierte Visual Servoing benötigten Koordinatensysteme sind wie in Abbildung 3.16 definiert. Das Roboter Koordinatensystem, *Roboter Frame (RF)* befindet sich in der Mitte des ersten Antriebsmoduls. Das *Kamera Frame (KF)* des aktiven Roboters ist relativ dazu nach oben vorne (X und Z Achse) verschoben und je 90° um die Y und Z Achse rotiert. Das *Objekt Frame (OF)* befindet sich im Zentrum der Schnittstelle des passiven Roboters [68].

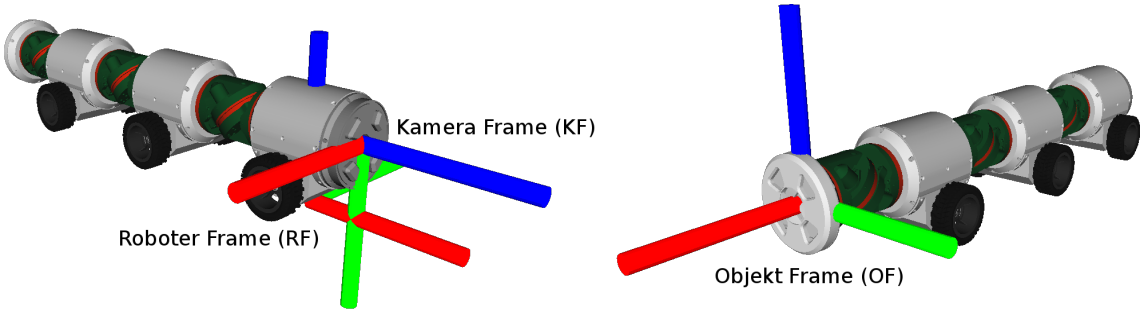


Abb. 3.16: Koordinatensysteme zur Bestimmung der relativen Pose zwischen aktivem und passivem Roboter.

Zur Realisierung des autonomen Docking wurden die folgenden Randbedingungen festgelegt:

- Direkter Sichtkontakt zwischen beiden Robotersystemen,
- Maximaler Abstand von 2 m und
- Gier-Winkelversatz von höchstens 90°

Der konzeptionelle Aufbau des Visual Servoing Systems für das Docking von RMRS ist in Abbildung 3.17 dargestellt. Mit Hilfe eines Detektors wird eine Initialisierung für das Tracking erzeugt. Das Tracking liefert kontinuierlich die Pose der passiven Schnittstelle an die Bewegungssteuerung welche Steuerbefehle für die Fortbewegung des aktiven Roboters generiert. Eine detailliertere Beschreibung der Teilsysteme und deren Umsetzung wird den nachfolgenden Abschnitten gegeben.

3.5.1 Detektion der initialen Pose

Die für das RMRS KAIRO 3 entwickelte Docking Methode verwendet einen Detektor zur anfänglichen Bestimmung einer relativen Roboterpose, die als Initialisierung für ein Trackingsystem dient. Zur Detektion wird ein *Haar Feature-based Cascade Classifier* der OpenCV [69, 70] Bibliothek genutzt um das Kamerabild nach vorher trainierten Features zu durchsuchen. Dazu wurden Bilder der passiven Schnittstelle aus verschiedenen Entfernungen, Orientierungen und mit unterschiedlichen Beleuchtungsverhältnissen trainiert. Das Training erfolgt ebenfalls mit den Tools der OpenCV. Als Ausgabe liefert der Detektor zunächst eine Liste rechteckiger Bildausschnitte mit den möglichen Vorkommen eines Roboters mit einer passiven Schnittstelle. Abbildung 3.18a zeigt die erkannten Bildausschnitte nach einer zusätzlichen Filterung und Bewertung. Aus den Ergebnissen der Detektion, wird der am Besten bewertete Bildausschnitt genutzt, um die initiale Pose für das Tracking zu berechnen [68].

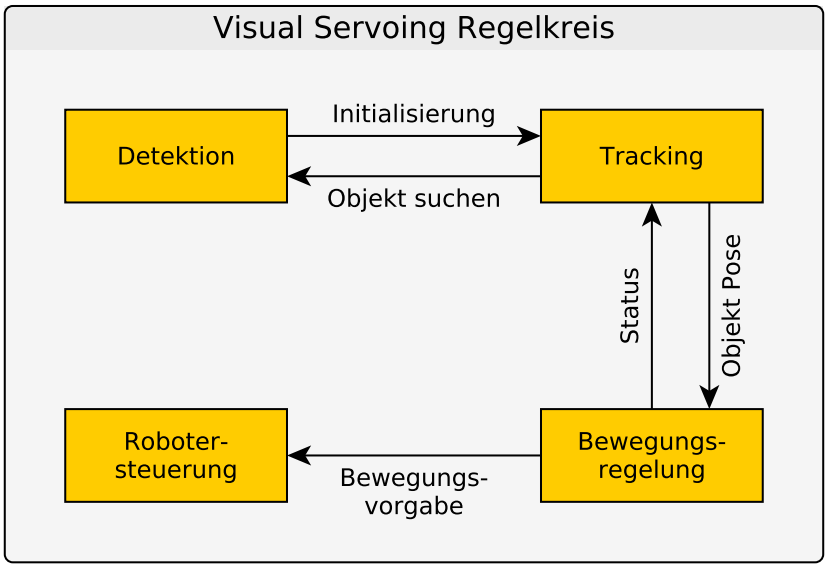


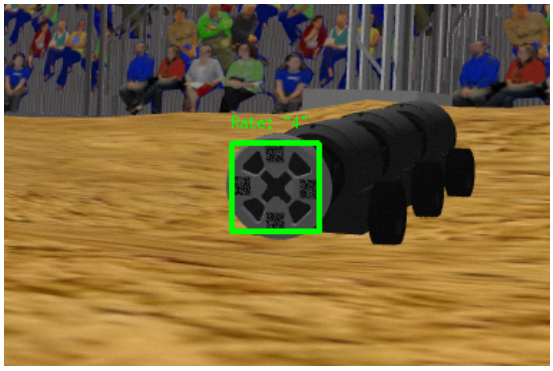
Abb. 3.17: Grundlegender Aufbau des autonomen, kamerabasierten Docking von mehreren Robotern mittels Visual Servoing [68].

3.5.2 Tracking des passiven Roboters

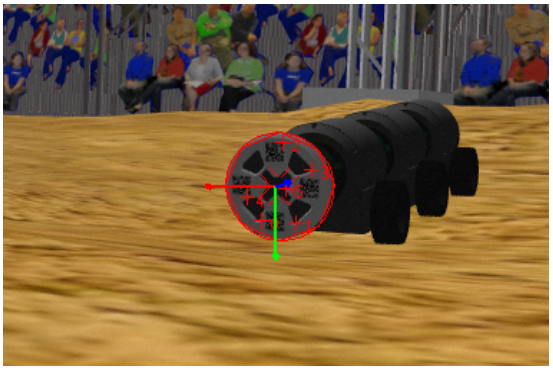
Die aus der anfänglichen Erkennung gewonnene Pose wird für die Initialisierung des Trackingsystems verwendet. Mit Hilfe des Tracking ist es möglich, kontinuierlich die relative Pose des passiven Roboter im Kamerabild zu bestimmen. Diese Information wird genutzt, um den aktiven Roboter in Richtung des passiven Roboter zu bewegen. Für die Umsetzung des Tracking wurde der Visp Auto Tracker [71] modifiziert und erweitert. Insbesondere wird ein modellbasierter Ansatz verwendet, der mittels CAD Daten der passiven Schnittstelle Features für das Tracking generiert. Abbildung 3.18b zeigt die durch das modellbasierte Tracking extrahierten Features und die daraus geschätzte Pose des passiven Roboters. Im Falle eines Trackingverlustes oder einer zu hohen Kovarianz, wird eine erneute Detektion durchgeführt und das Tracking neu initialisiert. Die kontinuierliche Ausgabe des Tracking besteht aus relativen Posen zwischen den beiden Robotern, die zur Bewegung des aktiven Roboters genutzt wird [68].

3.5.3 Bewegen des aktiven Roboters

Das Docking zweier Roboter mittels Visual Servoing wird durch das Bewegen des aktiven Roboters in Richtung des passiven Roboters realisiert. Bei KAIRO 3 wird dazu die bereits vorhandene Steuerung mittels virtueller Schiene benutzt. An Hand der aktuellen relativen Pose aus dem Tracker werden in jedem Aktualisierungsschritt neue Zwischenpunkte als Navigationsziele der virtuellen Schiene und eine entsprechende Fahrgeschwindigkeit berechnet. Bei der Berechnung werden benötigte Winkel und Posen berücksichtigt, so dass die beiden Roboter



(a) Detektierter Bildausschnitt der passiven Schnittstelle (grünes Rechteck) und zugehörige Bewertung.



(b) Extrahierte Features des Tracking (rot) und 3-D Pose des passiven Roboters.

Abb. 3.18: Visuelle Darstellung der Detektion und des Tracking für das autonome Docking am Beispiel von KAIRO 3 [68].

vor dem Verbinden der Schnittstelle exakt senkrecht zueinander stehen. Befinden sich die beiden Roboter in einem Abstand von weniger als 12 cm, kann die passive Schnittstelle nicht mehr vollständig im Kamerabild erkannt werden, was zu einer schlechteren Detektion durch das modellbasierte Tracking führt. Deshalb kann zur entsprechenden Feinjustierung für das finale Docking auf ein markerbasiertes Tracking umgeschaltet werden. Dazu wurde ein kleiner Marker auf die passive Schnittstelle aufgebracht, der in einem Bereich von 1 cm bis 15 cm gut von der Kamera erkannt werden kann. Schlussendlich kann die Schnittstelle geschlossen werden, sobald die in Kapitel 3.4.1 beschriebenen Ausgleichsstrukturen ineinander gegriffen und somit das Schnittstellenpaar zueinander ausgerichtet wurde.

3.6 Zusammenfassung

In diesem Kapitel wurden monolithische und selbst-rekonfigurierbare Robotersysteme gegenübergestellt sowie deren Vor- und Nachteile diskutiert. Die durch Kombination der Vorteile beider Roboterklassen gebildete Schnittmenge, wird durch den eingeführten Begriff der rekonfigurierbaren Multi-Roboter-Systeme (RMRS) bezeichnet. Eine Verfeinerung der Klassifizierung modularer Systeme auf Basis der RMRS wird am Beispiel von KAIRO 3 und mit Hilfe modularer Satellitenbausteine verdeutlicht. Des Weiteren wurden Verfahren und Methoden zur Realisierung von rekonfigurierbaren Multi-Roboter-Systemen am Beispiel von KAIRO 3 vorgestellt. Mit Hilfe der implementierten flexiblen Steuerungsarchitektur, der Hardwareschnittstelle zum Koppeln und Entkoppeln von Modulen sowie dem Visual Servoing basierten Verfahren zum autonomen Finden und Docken der Schnittstelle, wird die Umsetzung der Selbst-Rekonfiguration von RMRS am Beispiel von KAIRO 3 ermöglicht.

4 Generierung und Optimierung von Roboterkonfigurationen

Die Analyse des Einsatzszenario in Kapitel 1.2 hat gezeigt, dass es für RMRS mit vielen unterschiedlichen Modulen bzw. Bausteinen notwendig ist, die Roboterkonfiguration bezüglich der Aufgaben und Umweltbedingungen eines Einsatzes zu optimieren. Um die bestmögliche und auf einen speziellen Einsatz abgestimmte Konfiguration zu erhalten, sollte die Optimierung unmittelbar vor einem Einsatz, mit den entsprechenden Missions- und Umgebungsparametern durchgeführt werden. Auf Grund der Vielzahl an unterschiedlichen Bausteinen und der großen Varianz an Parametern, ergibt sich bereits eine Vielfalt an verschiedenen Bausteinkonfigurationen für ein einzelnes Robotersystem. Betrachtet man zusätzlich RMRS, welche einen Verbund aus mehreren unabhängigen modularen Robotern darstellen, so vergrößert sich der Suchraum zur Bestimmung der bestmöglichen Konfiguration enorm. Die beschränkte Zeit vor dem Einsatz des RMRS, erfordert jedoch eine sehr schnelle Bereitschaft und somit eine effiziente Lösung des Optimierungsproblems. Um dieses Ziel zu erreichen, wird im Folgenden ein mehrstufiger Prozess zur Generierung und Optimierung von Bausteinkonfigurationen vorstellt.

4.1 Synthese von Roboterkonfigurationen

Angelehnt an die Synthese von integrierten Hardware- und Softwarelösungen im Bereich Entwurfsautomatisierung elektronischer Systeme (EDA), wurde ein Verfahren zur Generierung und Optimierung modularer, rekonfigurierbarer Robotersysteme entwickelt. Abbildung 4.1 zeigt den grundlegenden Prozessablauf der sich in mehrere Teilschritte untergliedert, die von der Modellierung bis hin zum fertigen System aus Bausteinen reichen. Als Ausgangspunkt der Synthese dient ein Bausteinkatalog mit standardisierten Bausteinen und eine Spezifikation von Anforderungen in Form von einsatz- bzw. missionsspezifischen Parametern. Eine an den Bausteinkatalog angelehnte Wissensbasis wird während der Synthese zur Extraktion von Regeln und Restriktionen eingesetzt. Basierend auf diesen abgeleiteten Bedingungen und Regeln findet eine Optimierung der Auswahl und Anordnung von Hardwarebausteinen zu einen funktionsfähigen Robotersystem (Bausteinkonfiguration) statt.

Im folgenden Kapitel 4.2 wird zunächst näher auf die Modellierung und Spezifikation des Bausteinkatalogs sowie der Regeln und Restriktionen eingegangen. Die Speicherung und Verarbeitung der modellierten Informationen und Daten

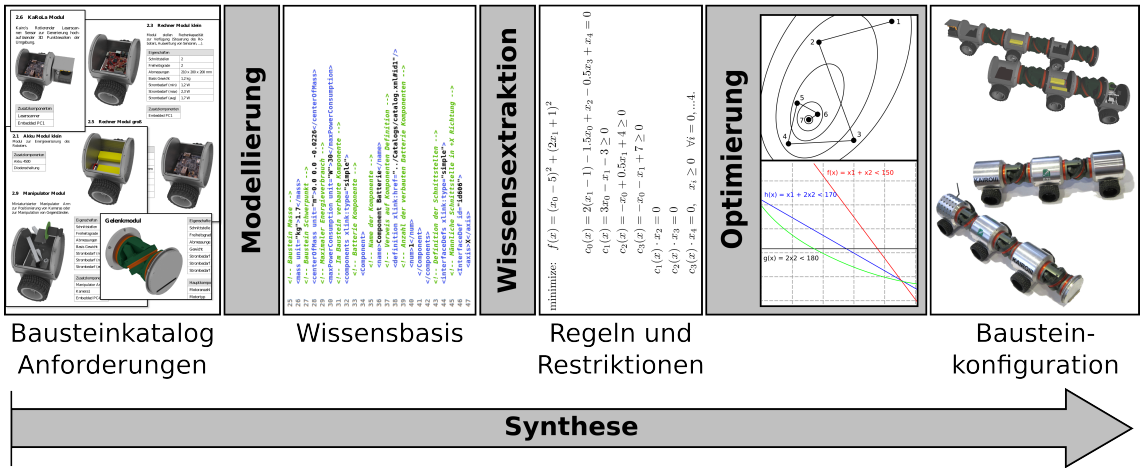


Abb. 4.1: Synthese zur Generierung und Optimierung von Konfigurationen für modulare rekonfigurierbare Systeme.

wird an Hand von zwei verschiedenen wissensbasierten Ansätzen verglichen und diskutiert. Darauf aufbauend beschreibt Kapitel 4.3 die einzelnen Teilschritte der Synthese sowie die eingesetzten Optimierungsverfahren und Algorithmen. Verschiedene Anwendungsfälle und eine grafische Benutzerschnittstelle zur einfacheren Handhabung der Synthese werden in Kapitel 4.4 vorgestellt.

4.2 Modellierung

Ein wichtiger Bestandteil der automatischen Generierung von Bausteinkonfigurationen ist die detaillierte Modellierung von Bausteinen bzw. Robotermodulen in einem Bausteinkatalog, deren Eigenschaften sowie Bedingungen und Regeln zur Auswahl und Anordnung der Bausteine. Die bei der Modellierung anfallenden Daten sollen effizient gespeichert werden, so dass gleichzeitig gute Maschinenlesbarkeit als auch Verständlichkeit für einen menschlichen Benutzer erreicht wird. Diese Anforderungen lassen sich mit Hilfe einer Wissensbasis erfüllen. Für die Realisierung dieser Wissensbasis wurden zwei unterschiedliche Ansätze untersucht. Die Modellierung und Speicherung der Informationen mit Hilfe einer Ontologie (OWL Standard des W3C-Konsortiums [72]) wurde in [73] gezeigt und analysiert. Eine OWL Ontologie bietet die Vorteile einer strukturierten und klassenorientierten Modellierung sowie die einfache Inferenz von Regeln durch bereits vorhandene Software Werkzeuge. Der zweite Ansatz nutzt ein individuelles XML-basiertes Datenformat, welches durch eine XML Schema Definitionen (XSD) spezifiziert wird. Diese Methode bietet gegenüber einer OWL Ontologie den Vorteil, dass keine spezialisierte Software zum Bearbeiten und Inferieren der Daten benötigt wird. Zudem ist bei der Implementierung der Algorithmen eine direkte Datenanbindung mit Zugang zu den Datenstrukturen möglich [74]. Im Folgenden wird die Modellierung mittels individueller Datenstruktur näher vorgestellt.

4.2.1 XSD/XML basierte Datenanbindung

Zur Vereinfachung der Datenanbindung in Form von XML Dateien, wird das Werkzeug CodeSynthesis XSD [75] verwendet. Dieser OpenSource und plattformübergreifende W3C XML Schema Compiler erzeugt C++ Klassen aus einzelnen oder einem Verbund von XSD Spezifikationen. Der XSD Compiler generiert eine Domain spezifische Datenstruktur in Form einer C++ Bibliothek, die das Parsen und Serialisieren von XML Dateien ermöglicht. Für den Zugriff auf die gespeicherte Daten, können somit Typen und Funktionen benutzt werden, die semantisch mit anwendungsspezifischen Domänen korrespondieren. Dadurch wird der Datenzugriff und ebenso der Austausch zwischen verschiedenen Programmen deutlich erleichtert. Mit Hilfe dieses Werkzeug werden die im Folgenden vorgestellten Datenstrukturen modelliert und für den Datenaustausch zwischen den einzelnen Synthese Schritten genutzt.

4.2.2 Bausteinkatalog

Der Bausteinkatalog bildet die Basis für die Modellierung der Bausteine bzw. Robotermodule. Um einen möglichst detaillierte Beschreibung der Bausteine zu erlauben, wurde der Bausteinkatalog in zwei Bereiche untergliedert. Der erste Bereich beschreibt zunächst Komponenten, die in die einzelnen Bausteine verbaut werden können. Diese Komponenten sind z. B. verschiedene Sensoren, Motoren, Steuerungsrechner oder Komponenten zur Energieversorgung. Der zweite Bereich des Bausteinkatalogs beschreibt die verfügbaren Bausteine und Robotermodule mit ihren Eigenschaften, den verbauten Komponenten, Schnittstellen und Ressourcen [74].

Abbildung 4.2 zeigt einen Ausschnitt eines Bausteinkatalogs mit der Definition eines Batteriebausteins für den modularen Roboter KAIRO 3. Neben Eigenschaften wie Name, Abmessung, Masse, Masseschwerpunkt und maximaler Energieverbrauch wird auch ein geometrisches Modell basierend auf CAD Daten spezifiziert. Des Weiteren wird eine Liste aller im Baustein verbauten Komponenten mit Referenzen auf die entsprechende Definition der Komponente angegeben. Eine Liste der mechanischen Schnittstellen zum Verbinden mit anderen Bausteinen vervollständigt die Spezifikation eines Bausteins. Das Hinzufügen und Ändern von Komponenten- und Bausteindefinitionen lässt sich dank des XML basierten Datenformats einfach durch einen Texteditor bewältigen. Zur weiteren Erleichterung der Bearbeitung wurde zusätzlich ein grafischer Editor entwickelt, der den Benutzer beim Anlegen und Modifizieren von Bausteinkatalogen mit vielen Komponenten und Bausteinen unterstützt. Der vollständige für KAIRO 3 modellierte Bausteinkatalog wird im Anhang A.1.1 dargestellt.

```
<!-- Batteriebaustein -->
<BuildingBlockDef id="id101">
  <!-- Baustein Name -->
  <name>KAIRO 3 Batterie Modul klein</name>
  <!-- Geometrisches Modell -->
  <geometry id="id501" xlink:href="../Models/kairo3_drive_batteries.3ds"/>
  <!-- Baustein Abmessungen -->
  <size unit="m">0.146 0.255 0.211</size>
  <!-- Baustein Masse -->
  <mass unit="kg">1.7</mass>
  <!-- Baustein Schwerpunkt -->
  <centerOfMass unit="m">0.0 0.0 -0.0226</centerOfMass>
  <!-- Maximaler Energieverbrauch -->
  <maxPowerConsumption unit="W">30</maxPowerConsumption>
  <!-- Im Baustein verbaute Komponente -->
  <components xlink:type="simple">
    <!-- Batterie Komponente -->
    <Component>
      <!-- Name der Komponente -->
      <name>Batterie klein</name>
      <!-- Verweis auf Komponenten Definition -->
      <definition xlink:href="../Catalogs/catalog.xml#id1"/>
      <!-- Anzahl der verbauten Batterie Komponenten -->
      <num>1</num>
    </Component>
  </components>
  <!-- Definition der Schnittstellen -->
  <interfaceDefs xlink:type="simple">
    <!-- Aktive Schnittstelle in +X Richtung -->
    <InterfaceDef id="id601">
      <axis>X</axis>
      <connection>Male</connection>
      <direction>Positive</direction>
      <faceId>0</faceId>
      <offset unit="m">0 0 0</offset>
    </InterfaceDef>
  [...]
</BuildingBlockDef>
```

Abb. 4.2: Definition eines KAIRO 3 Batteriebausteins mit seinen Eigenschaften, Komponenten und Schnittstellen im Bausteinkatalog.

4.2.3 Bedingungen für die Bausteinauswahl

In der Vorbereitungsphase einer Mission bzw. eines Einsatzes, soll ein modulares rekonfigurierbares Robotersystem an die konkreten Anforderungen des Einsatzes angepasst und optimiert werden. Zur Definition der erforderlichen Bedingungen für die Bausteinauswahl, werden zunächst die Bausteine im Bausteinkatalog in vier funktionale Gruppen unterteilt:

Missionsbausteine sind zwingend für die Erfüllung der Mission notwendig, wie z. B. ein Baustein der den zentralen Sensor der Mission enthält.

Betriebsbausteine stellen den Betrieb des Gesamtsystems sicher. Hierunter fallen Bausteine mit speziellen Fähigkeiten, wie z. B. Antriebsbausteine mit Rädern zur Fortbewegung.

Skalierbare Bausteine enthalten Komponenten mit Ressourcen die abhängig von der spezifischen Mission an den Bedarf bzw. Verbrauch des Gesamtsystems angepasst werden können. Ein Batteriebaustein mit veränderlicher Energiemenge kann als skalierbar angesehen werden.

Strukturbausteine sind spezielle passive Bausteine die durch ihre Struktur dem Gesamtsystem sowohl Stabilität als auch Transportkapazitäten für Ressourcen zur Verfügung stellen.

Auf Basis dieser Bausteingruppen lassen sich die zu spezifizierenden Anforderungen ableiten, die bei der Bausteinauswahl berücksichtigt werden müssen. Diese Anforderungen können wiederum in die vier Klassen Ressourcen-, Komponenten-, Typen- und Bausteinanforderungen einteilen. Für jede dieser Klassen wurde mit Hilfe des XSD Format eine separate Anforderungsliste modelliert. Abbildung 4.3 zeigt einen Auszug aus der entsprechenden XML Schema Definition. Für jede der Anforderungstypen wird ein komplexer XSD Datentyp mit den entsprechenden Eigenschaften und Attributen angelegt. Mehrere Elemente eines Anforderungstyps werden jeweils in einer Liste zusammengefasst und in einer geordneten Sequenz aus Ressourcen-, Komponenten-, Typen- und Bausteinanforderungen dargestellt.

```
<!-- Definition der Anforderungsliste -->
<xs:complexType name="RequirementListType">
  <xs:complexContent>
    <xs:sequence>
      <xs:element default="ANY" name="orbit" type="Orbit"/>
      <!-- Liste der Ressourcenanforderungen -->
      <xs:element maxOccurs="unbounded" minOccurs="0" name="resourceRequirement" type="ResourceRequirementType"/>
      <!-- Liste der Komponentenanforderungen -->
      <xs:element maxOccurs="unbounded" minOccurs="0" name="componentRequirement" type="ComponentRequirementType"/>
      <!-- Liste der Typenanforderungen -->
      <xs:element maxOccurs="unbounded" minOccurs="0" name="typeRequirement" type="TypeRequirementType"/>
      <!-- Liste der Bausteinanforderungen -->
      <xs:element maxOccurs="unbounded" minOccurs="0" name="buildingBlockRequirement" type="BuildingBlockRequirementType"/>
    </xs:sequence>
  </xs:complexContent>
</xs:complexType>
<!-- Definition der Ressourcenanforderungen -->
<xs:element abstract="false" name="ResourceRequirement" substitutionGroup="Resource" type="ResourceRequirementType"/>
<xs:complexType name="ResourceRequirementType">
  <xs:complexContent>
    <xs:extension base="ResourceType">
      <xs:sequence/>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
<!-- Definition der Komponentenanforderungen -->
<xs:element abstract="false" name="ComponentRequirement" type="ComponentRequirementType"/>
<xs:complexType name="ComponentRequirementType">
  <xs:complexContent>
    <xs:sequence>
      <xs:element name="comp_id" type="xs:ID"/>
      <xs:element default="1" name="count" type="CountType"/>
    </xs:sequence>
  </xs:complexContent>
</xs:complexType>
[...]
<!-- Definition einer Ressource -->
<xs:element abstract="false" name="Resource" substitutionGroup="VSD:NamedModelInstance" type="ResourceType"/>
<xs:complexType name="ResourceType">
  <xs:complexContent>
    <xs:extension base="VSD:NamedModelInstanceType">
      <xs:sequence>
        <xs:element name="res_id" type="ResourceIDType"/>
        <xs:element default="false" name="atomic" type="xs:boolean"/>
        <xs:element name="amount" type="xs:double"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

Abb. 4.3: Auszug der Modellierung von missionsspezifischen Anforderungen, die als Eingabe für die Bausteinauswahl dienen.

Die Spezifikation der Anforderungen erfolgt durch die Vorgabe einer Liste notwendiger Komponenten, wie z. B. Sensoren zur Umwelterfassung und Inspektion bzw. Aktoren zur Fortbewegung. Darüber hinaus können missionsspezifische Bausteine und Bausteintypen festgelegt werden. Zusätzlich können die benötigten Ressourcen, wie beispielsweise Energie und Rechenleistung vorgegeben werden. Jeder Baustein im Katalog kann eine oder mehrere dieser Ressourcen bereit-

stellen oder verbrauchen. Insbesondere für Ressourcen gibt es eine Unterteilung in nicht-atomar und atomar, die festlegt, ob eine bestimmte Ressourcenanforderung von exakt einem Baustein bzw. einer Komponente erfüllt werden muss (atomar) oder auf mehrere Bausteine verteilt werden kann (nicht-atomar).

Das resultierende System ist bezüglich der ausgewählten Bausteine *gültig*, wenn alle spezifizierten Anforderungen erfüllt sind, und es wird als *optimal* bezeichnet, wenn das System ebenfalls minimal skaliert ist [73].

4.2.4 Eigenschaften und Regeln für die Bausteinanordnung

Zur optimalen Anordnung von Bausteinen in einer Konfiguration, werden verschiedene Bausteineigenschaften sowie Ursachen und Wirkungen zwischen mehreren Bausteinen modelliert. Bereits beim Erstellen des Bausteinkatalog werden jedem Baustein die entsprechenden Eigenschaften und Ursachen zugeordnet. Im Kontext rekonfigurierbarer Multi-Roboter-Systeme wurden die folgenden Klassen an Eigenschaften festgelegt:

Bodenkontakt (*GroundContactType*): Definiert, ob ein Baustein Bodenkontakt benötigt und auf welcher Bausteinseite (z. B. Radantriebsbaustein).

Sichtfeld (*FieldOfViewType*): Baustein besitzt ein Sichtfeld mit einem einstellbarem Öffnungswinkel, das nicht verdeckt werden darf (z. B. Sensorbaustein).

Bewegungsrichtung (*MoveDirectionType*): In Abhängigkeit von der Bewegungsrichtung des Roboters, benötigt ein Baustein eine spezifische Ausrichtung.

Sonneneinstrahlung (*SunlightType*): Bausteine die empfindlich gegenüber direkter Sonneneinstrahlung sind besitzen diese Eigenschaft (z. B. Kamerabausteine).

Um Abhängigkeiten zwischen mehreren Bausteinen beschreiben zu können, wurden die folgenden Klassen von Ursachen und Wirkungen als Eigenschaften definiert:

Strahlung (*RadiationType*): Bausteine die in einem bestimmten Frequenzspektrum Strahlung abgeben oder durch die Strahlung anderer Bausteine beeinflusst werden (z. B. Baustein mit Beleuchtung blendet Kamerabaustein).

Beobachtungspunkt (*ViewPointType*): Mehrere Bausteine benötigen gleiche Ausrichtung zur Fokussierung auf den selben Punkt im Raum (z. B. zwei Kamerabausteine bilden ein Stereokamerasystem).

Magnetfeld (*MagneticFieldType*): Bausteine die ein bestimmtes Magnetfeld besitzen beeinflussen andere Bausteine (z. B. starker Elektromotor beeinflusst einen Lagesensorbaustein).

Energieübertragung (*EnergieFlowType*): Bausteine die sehr viel Energie benötigen sollten in der Nähe von Batteriebausteinen angeordnet werden.

Datenfluss (*DataFlowType*): Bausteine die sehr große Datenmengen erzeugen sollten in der Nähe von Bausteinen zur Datenverarbeitung angeordnet werden.

Wärmeübertragung (*HeatFlowType*): Bausteine die sehr viel Wärme erzeugen sollten in der Nähe von Bausteinen zur Wärmeabfuhr angeordnet werden.

Aus den vorgestellten Ursachen und Bausteineigenschaften können entsprechende Regeln für die Anordnung der Bausteine abgeleitet werden. Die Methodik zur Überführung der Eigenschaften in Regeln wird in Kapitel 4.3.2 dargestellt. Für die Modellierung der Regeldefinitionen wird ein getrennter Regelkatalog verwendet, der den Bausteinkatalog referenziert. Diese Regeldefinitionen lassen sich in drei Klassen einteilen:

Global: Missionsspezifische Regeln die sich auf alle Bausteine beziehen.

Baustein-relativ: Regeln, die zwischen zwei oder mehreren Bausteinen wirken.

Baustein-spezifisch: Regeln, die sich auf individuelle Bausteine beziehen.

Neben der Einteilung in Klassen wird für jede Regel mit Hilfe einer zweiten Dimension festgelegt, ob diese Regel notwendig (N) oder empfohlen (E) ist. Eine notwendige Regel muss bei der Bausteinanordnung unbedingt eingehalten werden um eine gültige Konfiguration zu erhalten. Eine empfohlene Regel liefert meist ein besseres Ergebnis, kann im Falle von gegensätzlichen Regeln unerfüllt bleiben, wenn dafür eine andere notwendige Regel erfüllt wird [73]. Die im Folgenden beschrieben Regeltypen zur Bausteinanordnung wurden im Rahmen dieser Arbeit modelliert und implementiert.

Globale Regeln

Globale Regeln sind für das Gesamtsystem gültig und beziehen sich damit immer auf die Menge aller ausgewählten Bausteine.

Konfigurationsraum: Legt die Dimensionen des Konfigurationsraum (x, y, z) zur Optimierung fest.

- Maximale Ausdehnung (N): Begrenzt die maximale Ausdehnung eines zu optimierenden Systems und ermöglicht ebenfalls das Sperren einzelner Raumachsen zur Erzeugung flächiger oder kettenartiger Konfigurationen.

Cluster: Beeinflusst die Aufteilung des Systems auf mehrere Teilsysteme.

- Anzahl Baustein Cluster (N): Erzeugt ein monolithisches System oder mehrere unabhängige Teilsysteme durch Gruppenbildung.

Flüsse: Definiert den notwendigen Ressourcenaustausch zwischen den einzelnen Bausteinen.

- Stromfluss (N): Dimensioniert Quellen, Senken und Leitfähigkeit des Bausteinnetzes für elektrische Energie.
- Wärmefluss (N): Dimensioniert Quellen, Senken und Leitfähigkeit des Bausteinnetzes für thermische Energie.
- Datenfluss (N): Dimensioniert Quellen, Senken und Leitfähigkeit des Bausteinnetzes für den Datentransport.

Anzahl Verbindungen: Legt die nötigen Eigenschaften für das Verbinden der Bausteine mittels Schnittstellen fest.

- Anzahl ungültiger Verbindungen = 0 (N): Bei der Anordnung der Bausteine müssen zwei benachbarte Bausteine jeweils eine Schnittstelle zur Kopplung besitzen. Eine Verbindung ist ungültig, wenn eine der beiden angrenzenden Bausteinseiten keine kompatible Schnittstelle besitzt.
- Anzahl gültiger Bausteinverbindungen = MAX (E): Das Maximieren der möglichen Verbindungen sorgt für eine kompakte Konfiguration.
- Ressourcentransport = MAX (E): Bausteine mit großem Ressourcenbedarf oder hoher Ressourcenbereitstellung sollten möglichst direkt oder über ein Bussystem aus Strukturbausteinen verbunden werden.

Baustein-relative Regeln

Baustein-relative Regeln beschreiben Einschränkungen die sich auf die Beziehung zwischen zwei oder mehreren spezifischen Bausteinen beziehen. Diese Regeln können durch Inferenz aus den Bedienungen der ausgewählten Bausteine ermittelt werden.

Relativer Abstand: Beschreibt Entfernungseinschränkungen zwischen zwei oder mehreren Bausteinen bei der Anordnung der Bausteine.

- Abstand zwischen Bausteinen = MIN (MAX) (E): Minimiere (maximiere) die Entfernungen zwischen zwei Bausteinen ohne spezifischen Schwellwert.
- Minimal- / Maximalabstand zwischen Bausteinen (N): Explizite Mindest- bzw. Maximalentfernung zwischen zwei Bausteinen.

Relative Orientierung: Beschreibt Rotationseinschränkungen zwischen zwei oder mehreren Bausteinen bei der Anordnung der Bausteine.

- Relativer Orientierungsoffset (N): Expliziter relativer Orientierungsunterschied zwischen zwei Bausteinen. Die gleiche Orientierung beider Bausteine ist durch einen Null-Offset möglich.

- Nicht Orientierungsoffset (N): Orientierung zwischen zwei Bausteinen darf nicht gleich dem gegeben Offset sein.

Baustein-spezifische Regeln

Baustein-spezifische Regeln sind jeweils einem spezifischen Baustein zugeordnet und werden bereits bei der Modellierung des Bausteinkatalogs festgelegt.

Bausteinbereich: Spezifiziert Bereiche außerhalb eines Bausteins in denen spezielle Einschränkungen für alle anderen Bausteine gelten.

- Sichtkegel (N): Der Sichtbereich eines Sensors darf nicht durch andere Bausteine verdeckt werden.
- Kollision (E): Anbauteile die über das Bausteinraster hinausragen dürfen nicht mit anderen Bausteinen oder Sichtkegeln kollidieren.

Absolute Orientierung: Absolute Ausrichtung eines Bausteins in Bezug auf das globale Koordinatensystem.

- Spezifische Orientierung (N): Baustein muss mit einem spezifischen Orientierungsoffset zum globalen Koordinatensystem ausgerichtet sein.
- Nicht spezifische Orientierung (N): Baustein darf nicht mit dem spezifizierten Orientierungsoffset zum globalen Koordinatensystem orientiert sein.

Absolute Lage: Absolute Position eines Bausteins in Bezug auf das globale Koordinatensystem bzw. eine Seite des Systems.

- Spezifische Seite (N): Baustein muss an einer spezifischen Seite des Gesamtsystems angeordnet sein.
- Nicht spezifische Seite (N): Baustein darf nicht an der spezifizierten Seite platziert werden.
- Hülle (N): Baustein muss auf der Hülle des Gesamtsystems liegen.
- Nicht Hülle (N): Baustein muss im Inneren des Gesamtsystems verbaut sein.
- Lage im System = MIN (MAX) (E): Ein Baustein soll möglichst weit innen oder außen liegen.

Zur Modellierung des Regelkatalogs mit den oben beschriebenen Regeltypen wird ein XSD/XML basiertes Datenformat verwendet. Dadurch lässt sich der Regelkatalog gleichermaßen wie der Bausteinkatalog problemlos erweitern. Ein Auszug der konkret implementierten Regeln sind im Anhang A.2 tabellarisch aufgeführt. Für jeden Regeltyp wird eine entsprechende Fitnessfunktion oder ein Algorithmus zur Berechnung der Güte benötigt, wie in Kapitel 4.3.3 beschrieben. Mit Hilfe dieser Fitness kann bei der Optimierung die Einhaltung der Regel sowie die Güte der einzelnen Konfiguration überprüft werden.

4.3 Optimierungsverfahren und Algorithmen

Abgeleitet aus dem in Kapitel 4.1 vorgestellten Konzept zur Synthese, veranschaulicht Abbildung 4.4 die Umsetzung des Syntheseprozess zur Generierung und Optimierung von Roboterkonfigurationen. Ausgangspunkt der Synthese sind ein nach Kapitel 4.2 modellierter Bausteinkatalog mit zugehörigen Bedingungen und Regeln sowie eine durch den Benutzer vorgegebene Anforderungsliste. Der interaktive Syntheseprozess untergliedert sich dabei in die drei Schritte *Bausteinauswahl*, *Regelinferenz* und *Bausteinanordnung*, die sich jeweils durch eigene Optimierungsmethoden unterscheiden. Im Folgenden werden die zur Umsetzung untersuchen Algorithmen und Verfahren diskutiert.

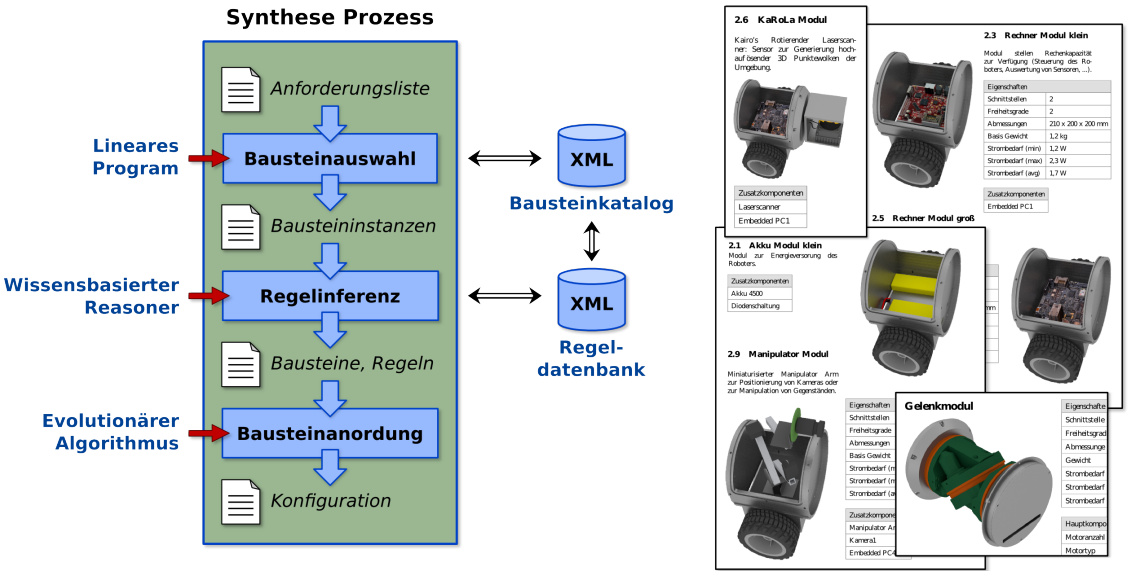


Abb. 4.4: Missionsspezifische Generierung und Optimierung von Roboterkonfigurationen mit Hilfe eines mehrstufigen Syntheseprozess (links) basierend auf einem Bausteinkatalog mit Regeldatenbank (rechts).

4.3.1 Bausteinauswahl

Ziel der Bausteinauswahl ist es eine minimale Menge an Bausteinen aus dem Katalog auszuwählen, welche die Anforderungen der Mission bzw. des Einsatzes erfüllen. Als Eingabe für die Bausteinauswahl dient der Bausteinkatalog mit entsprechenden Regeln und eine Anforderungsliste, bestehend aus *Ressourcen-*, *Komponenten-*, *Typen-* und *Bausteinanforderungen*, die durch den Benutzer festgelegt werden können. Die Ausgabe besteht aus einer Liste mit Bausteininstanzen, die jeweils konkrete physikalische Bausteine darstellen und die entsprechende Bausteindefinitionen aus dem Bausteinkatalog referenzieren. Eine solche Bausteinliste wird als *optimal* angesehen, wenn alle Anforderungen erfüllt sind und keine weitere Lösung existiert, die weniger Bausteine, Komponenten oder Ressourcen benötigt.

Bausteinvorauswahl

Vor der eigentlichen Auswahl und Instanziierung von Bausteinen wird zunächst geprüft, ob sich die gestellten Anforderungen durch den gegeben Bausteinkatalog prinzipiell erfüllen lassen. Die Anforderungen werden als nicht erfüllbar deklariert, falls im Bausteinkatalog kein Baustein existiert, der eine spezielle Bausteintypen-, Komponenten- oder Ressourcenanforderung erfüllt. Sind die Anforderungen realisierbar, so werden bereits Bausteine ausgewählt, die zwingend erforderlich sind und keiner Optimierung bedürfen, wie beispielsweise explizit vorgegebene missionsspezifische Bausteine. Zusätzlich werden Bausteine ausgewählt, die zur Erfüllung einer bestimmten Anforderung essentiell sind. Die noch verbleibenden Anforderungen und die Ressourcen, die durch die bereits ausgewählten Bausteine bereit gestellt werden, dienen als Eingabe für den Algorithmus zur Bausteinauswahl.

Zur Lösung des Bausteinauswahl-Problems wurden drei verschiedene Algorithmen untersucht: Eine *Brute-Force-Suche*, einen *Greedy-Ansatz* und die *Ganzzahlige lineare Optimierung*.

Brute-Force-Suche

Bei der Brute-Force Optimierung werden, unter Variation der Bausteinanzahl, iterativ alle möglichen Kombinationen von Bausteinen aus dem Bausteinkatalog auf die Einhaltung der gestellten Anforderungen geprüft. Eine optimale Lösung liegt jedoch nicht nur in der Einhaltung der Anforderungen, sondern auf Grund von unterschiedlichen Bausteinkosten, auch in der Auswahl einer möglichst geringen Anzahl an Bausteinen mit minimalen Kosten. Zum Finden einer optimalen Lösung müssen alle Kombinationen aus maximal $\lfloor c_s/c_{min} \rfloor$ Bausteinen untersucht werden, wobei $c_s \geq 0$ die Kosten der besten gefunden Lösung und $c_{min} \geq 0$ die Kosten des günstigsten Bausteins im Katalog beschreiben. Die daraus folgende

Menge an zu bewertenden Kombinationen lässt sich unter Annahme der Bausteinanzahl k und einer Größe des Katalogs von n Bausteinen wie folgt berechnen:

$$\binom{n+k-1}{k} = \frac{n\,(n+1)\,...\,(n+k-1)}{k!} \tag{4.1}$$

Bei einer Menge von 20 ausgewählten Bausteinen aus einem Katalog mit 100 Bausteinen ergeben sich bereits $5,36 \cdot 10^{18}$ zu überprüfende Kombinationen [76].

Greedy-Algorithmus

Greedy-Algorithmen verwenden meistens eine Heuristik, um effizient und schrittweise eine Lösungsmenge zu optimieren. Diese Art von Optimierungsalgorithmen findet immer eine gültige Lösung, die jedoch nicht notwendigerweise auch eine optimale Lösung darstellt. Auf Grund der oft suboptimalen Ergebnisse wird zusätzlich ein Backtracking verwendet. Dazu wird in Situationen, die keine Verbesserung zu vorherigen Lösungen erkennen lassen (lokales Minima), eine vermeidlich schlechtere Lösung weiter optimiert. Dies wird allerdings durch zusätzliche Rechenzeit erkauft. Im Falle der Bausteinauswahl wird mit Hilfe der im Folgenden erläuterten Heuristiken eine möglichst optimale Anzahl an Bausteinen ausgewählt [76].

1. Betrachte Bausteine, die noch offene Anforderungen erfüllen bzw. noch benötigte Ressourcen bereitstellen. Füge den Baustein hinzu, bei dem die quadratische Summe der verbleibenden Ressourcenanforderungen minimiert wird.

2. Der Algorithmus bricht ab, wenn jede einzelne noch verbleibende Anforderung größer oder gleich zu den verbleibenden Anforderungen der vorangegangen Teillösungen ist und somit keine gültige Lösung gefunden werden kann. Im Falle des Backtracking, wird an dieser Stelle eine vorherige Teillösung gewählt und der Baustein mit der nächst-schlechteren Bewertung zur Lösungsmenge hinzugefügt.

3. Eine zusätzliche Heuristik versucht eine gültige Lösung weiter zu verbessern, in dem je zwei Bausteine durch einen Baustein, der ebenfalls die Anforderungen erfüllt, ersetzt werden.

Ganzzahlige lineare Optimierung

Die *ganzzahlige lineare Optimierung*, auch als ILP (Integer Linear Programming) bezeichnet, ermöglicht die Optimierung einer linearen Zielfunktion für komplexe

Gleichungssysteme. Ein lineares Programm gehört zur Klasse der *NP-schweren* Probleme und wird meist durch eine Menge von Ungleichungen der Form

$$\max \{c^T x | Ax \leq b, x \geq 0, x \in \mathbb{Z}^n\}, \tag{4.2}$$

$$\text{mit } a_i \cdot x = \sum_{j=1}^{n} a_{ij} x_j \leq b_i \tag{4.3}$$

dargestellt. Die Variablen der Gleichungen dürfen dabei lediglich ganzzahlige und nicht-negative Werte annehmen. Geometrisch betrachtet, bilden diese Gleichungen einen polyederförmigen Lösungsraum. Während der Optimierung, werden jeweils die Hyperebenen $x|c^T x = 0$ in Richtung des Vektors c verschoben, bis diese den Rand des optimalen Polyeders erreichen. Sind alle Hyperebenen am Rand des Lösungsraumes angelangt, erhält man die Menge der optimalen Lösungen. Im schlechtesten Fall, können sich bei dieser Art der Optimierung sehr lange Rechenzeiten ergeben. Im praktischen Einsatz hat sich jedoch gezeigt, dass dieser Extremfall nur sehr selten eintritt [77]. Insbesondere aktuelle Optimierungswerkzeuge, wie GLPK [78], SCIP [79] oder CPELX [80] ermöglichen eine sehr effiziente Berechnung und Lösung des Problems.

Die Realisierung der Bausteinauswahl als ganzzahliges Programm erfordert die Abbildung der Anforderungen auf entsprechende Gleichungen. Zur Minimierung der benötigten Ressourcen, Komponenten und Bausteine wird die zur Maximierung gezeigte Gleichung 4.2 leicht modifiziert:

$$\min \{c^T x | Ax \geq b, x \geq 0, x \in \mathbb{Z}^n\} \tag{4.4}$$

Eine Überführung der Anforderungen auf Gleichungen ist auf Grund der verschiedenen Arten und insbesondere für atomare Ressourcen nicht trivial. Betrachtet man zwei gleiche Ressourcen-Anforderungen mit je 50 Einheiten und zwei Bausteine, die jeweils 60 und 40 Einheiten der Ressource anbieten, würde das Problem nach Gleichung 4.4 wie folgt beschrieben:

$$A = (-40, -60) \text{ und}$$
$$b = -100.$$

Es existieren zwei Möglichkeiten $x_1 = (1, 1)^T$ und $x_2 = (0, 2)^T$ die Gleichung $Ax \leq b$ optimal bezüglich der minimalen Anzahl an Bausteinen lösen. Im Falle einer atomaren Ressource ist jedoch die erste Lösung ungültig, da die geforderten 50 Einheiten auf zwei Bausteine $40 + 10$ aufgeteilt werden müssten. Aus diesem Grund, werden alle atomaren Ressourcen-Anforderungen getrennt voneinander betrachtet und virtuelle Ressourcen und Bausteine eingeführt. Die virtuellen Ressourcen $r_{a_1}, ..., r_{a_k}$ werden für mehrere, unterschiedlich große Anforderungen $a_1, ..., a_k$ der Ressource r eingeführt. Für alle Kombinationen der virtuellen Ressourcen werden mehrere virtuelle Bausteine B_i aus dem Baustein B erzeugt. Tabelle 4.1 verdeutlicht diesen Sachverhalt an Hand eines Beispiels mit einem Baustein B der 50 Einheiten einer atomaren Ressource anbietet. Zusätzlich

Baustein	r_{a_1}	r_{a_2}
B_1	5	0
B_2	3	1
B_3	2	2
B_4	0	3

Tabelle 4.1: Beispiel für die Erzeugung virtueller Ressourcen und Bausteine für eine atomare Ressource r, einen Baustein B sowie den Anforderungen a_1, a_2.

werden zwei Anforderungen $a_1 = 10$ sowie $a_2 = 15$ der selben atomaren Ressource definiert. Insgesamt werden für die beiden getrennt betrachteten Ressourcen-Anforderungen r_{a_1} und r_{a_2} vier virtuelle Bausteine eingeführt. Für jede der Kombinationen wird festgelegt, wie viele der Anforderungen von einem der virtuellen Bausteine erfüllt werden können. Zum Beispiel liefert Baustein B_2 genügend Ressourcen zur Erfüllung von drei r_{a_1} Anforderungen und einer r_{a_2} Anforderung ($3 \cdot 10 + 1 \cdot 15 = 45 \leq 50$). Die in Tabelle 4.1 bestimmen Variationen werden direkt für die Aufstellung der Gleichungen des linearen Programms verwendet [76].

Die Überführung der Baustein-, Typen-, und Komponenten-Anforderungen gestaltet sich etwas einfacher. Für konkrete Anforderungen von missionsspezifischen Bausteinen kann direkt die minimal benötigte Anzahl der Baustein-ID als Gleichung angegeben werden (z. B. bei mindestens zwei Bausteinen B_3 lautet die Gleichung $B_3 \geq 2$). Bei *Typen-Anforderungen* werden zunächst virtuelle Ressourcen r_{t_i} für alle im Katalog vorhandenen Bausteintypen t_i eingeführt. Jeder Baustein des Typs t_i stellt exakt eine Einheit der Ressource r_{t_i} zur Verfügung. Entsprechend der anforderten Bausteintypen, werden die Gleichungen t_i für die virtuellen Ressourcen-Anforderungen r_{t_i} erzeugt. Für die Erfassung der *Komponenten-Anforderungen* werden ebenfalls virtuelle Ressourcen r_{u_i} für alle Komponenten u_i angelegt und jedem Baustein B, der eine oder mehrere Komponenten enthält, die entsprechende Anzahl virtueller Ressourcen r_{t_i} zugewiesen. Die Abbildung auf Gleichungen u_i für das ILP wird im folgenden Beispiel verdeutlicht [76].

Beispiel zur Erstellung eines linearen Programms Der in Tabelle 4.2 dargestellte Bausteinkatalog bestehend aus den drei Bausteinen B_1, B_2 und B_3 und die in Tabelle 4.3 spezifizierte Anforderungsliste dienen als Grundlage für dieses Beispiel.

Für alle Bausteine ist jeweils festgelegt, welche der drei Ressourcen $R_1^{(a)}$, R_2 und R_3 bereitgestellt oder benötigt werden. In der Anforderungsliste werden jeweils eine Einheit ($R_1 : 1$) und zwei Einheiten ($R_1 : 2$) der atomaren Ressource $R_1^{(a)}$ gefordert. Aus diesem Grund werden zwei virtuelle Ressourcen r_1a_1 und r_1a_2 erstellt. Da die Bausteine B_1 und B_2 die atomare Ressource R_1 bereitstellen, werden

			Ressourcen		
Baustein	Typ	Komponenten	$R_1^{(a)}$	R_2	R_3
B_1	T_1	C_1	2	-1	0
		C_2	0	-1	1
B_2	T_1	C_1	3	-1	0
		C_3	0	-2	3
B_3	T_2	C_4	-1	3	0

Tabelle 4.2: Vereinfachter Bausteinkatalog mit drei Bausteinen, vier Komponenten und drei Ressourcen. Die Ressource R_1 ist atomar. Positive Werte entsprechen angebotenen und negative Werte benötigten Ressourcen.

Baustein-Anforderungen	1x B_3
Typen-Anforderungen	2x T_1
Komponenten-Anforderungen	1x C_3
Ressourcen-Anforderungen	$R_1 : 1$
	$R_1 : 2$
	$R_2 : 2$

Tabelle 4.3: Spezifizierte Baustein-, Typen-, Komponenten- und Ressourcen-Anforderungen.

für beide je zwei[1] virtuelle Bausteine B_1^{v1} und B_1^{v2} sowie B_2^{v1} und B_2^{v2} angelegt. Zusammen mit der Baustein-Anforderung eines B_3 Bausteins ergibt sich die folgende Kostenfunktion zur Minimierung:

$$\text{cost: } B_1^{v1} + B_1^{v2} + B_2^{v1} + B_2^{v2} + B_3$$

Zusätzlich zur Kostenfunktion werden die für das ILP benötigten Randbedingungen aus den Anforderungen wie folgt abgeleitet. Zur Einhaltung den Typen-Anforderungen aus Tabelle 4.3 werden die folgenden Gleichungen formuliert. Gleichung t1 leitet sich aus der Forderung nach mindestens zwei Bausteinen des Typs T_1 ab, wobei jeder der vier virtuellen Bausteine von Typ T_1 ist. Die Gleichung t2 entsteht aus der Baustein-Anforderung 1x B_3.

$$\text{t1: } B_1^{v1} + B_1^{v2} + B_2^{v1} + B_2^{v2} \geq 2$$
$$\text{t2: } B_3 \geq 0$$

Die Spezifikation der Komponenten-Anforderung 1x C_3 resultiert in den folgenden Gleichungen. Die Terme c1 und c2 stellen sicher dass keine negative Anzahl

[1] Die angebotene Menge der R_1 Ressource lässt sich für beide Bausteine exakt in zwei Teilmengen für r_1a_1 und r_1a_2 aufteilen

an Bausteinen ausgewählt werden. Da die virtuellen Bausteine B_2^{v1} und B_2^{v2} jeweils die geforderte Komponente C_3 enthalten muss Gleichung c3 die Mindestanzahl 1 erfüllen. Der Term c4 entsteht wiederum aus der Baustein-Anforderung 1x B_3.

$$\begin{aligned}
&\text{c1: } B_1^{v1} + B_1^{v2} + B_2^{v1} + B_2^{v2} \geq 0 \\
&\text{c2: } B_1^{v1} + B_1^{v2} \geq 0 \\
&\text{c3: } B_2^{v1} + B_2^{v2} \geq 1 \\
&\text{c4: } B_3 \geq 0
\end{aligned}$$

Die Gleichungen der Ressourcen-Anforderungen ergeben sich jeweils aus den Summen der pro Baustein benötigten oder bereitgestellten Ressourcen (siehe Tabelle 4.2).

$$\begin{aligned}
&\text{r1a1: } 2 \cdot B_1^{v1} + 3 \cdot B_2^{v1} - B_3 \geq 1 \\
&\text{r1a2: } B_1^{v2} + B_2^{v2} \geq 1 \\
&\text{r2: } -2 \cdot B_1^{v1} - 2 \cdot B_1^{v2} - 3 \cdot B_2^{v1} - 3 \cdot B_2^{v2} + 3 \cdot B_3 \geq 2 \\
&\text{r3: } B_1^{v1} + B_1^{v2} + 3 \cdot B_2^{v1} + 3 \cdot B_2^{v2} \geq 0
\end{aligned}$$

Zusätzlich werden noch die unteren Grenzen der Bausteintanzahlen festgelegt:

$$B_1^{v1} \geq 0,\ B_1^{v2} \geq 0,\ B_2^{v1} \geq 0,\ B_2^{v2} \geq 0,\ B_3 \geq 1$$

Das in Abbildung 4.5 zeigt das resultierende Lineare Programm für das minimale Beispiel und dient als Eingabe für den SCIP Linear Solver. Die für dieses Beispiel gefundene optimale Lösung besteht aus zwei B_1, einem B_2 und drei B_3 Bausteinen. Bei komplexeren Problemen mit deutlich größeren Bausteinkatalogen und längeren Anforderungslisten können die Gleichungssysteme schnell recht groß und unübersichtlich werden [76].

Das Resultat dieses ersten Optimierungsschritt, besteht aus einer minimalen Menge von Bausteininstanzen zur Erfüllung der spezifizierten Anforderungen. Mit Hilfe des allgemeinen XML Datenaustauschformates wird diese optimierte Bausteinliste an den nächsten Syntheseschritt, der Regelinferenz, weitergegeben.

4.3.2 Regelinferenz

Nach der Bestimmung der optimalen Menge an benötigten Bauteinen folgt der Syntheseschritt zur Regelinferenz (siehe Syntheseprozess in Kapitel 4.3). An Hand der ausgewählten Bausteininstanzen werden die in Kapitel 4.2.4 vorgestellten globalen, Baustein-spezifischen und Baustein-relativen Regeln zur Optimierung aus den Bausteineigenschaften sowie deren Ursachen und Wirkungen abgeleitet. Insbesondere die Baustein-relativen Regeln sind nicht

```
Minimize
  cost: B1v1 + B1v2 + B2v1 + B2v2 + B3
Subject to
  t1: B1v1 + B1v2 + B2v1 + B2v2 >= 2
  t2: B3 >= 0
  c1: B1v1 + B1v2 + B2v1 + B2v2 >= 0
  c2: B1v1 + B1v2 >= 0
  c3: B2v1 + B2v2 >= 1
  c4: B3 >= 0
  r1a1: 2 B1v1 + 3 B2v1 - B3 >= 1
  r1a2: B1v2 + B2v2 >= 1
  r2: -2 B1v1 -2 B1v2 -3 B2v1 -3 B2v2 +3 B3 >= 2
  r3: B1v1 + B1v2 +3 B2v1 +3 B3v2 >= 0
Bounds
  B1v1 >= 0
  B1v2 >= 0
  B2v1 >= 0
  B2v2 >= 0
  B3 >= 0
General
  B1v1
  B1v2
  B2v1
  B2v2
  B3
End
```

Abb. 4.5: Datenformat als Eingabe für das Optimierungswerkzeug SCIP [79].

immer direkt ersichtlich und lassen sich nur sukzessive aus den modellierten Eigenschaften schlussfolgern bzw. inferieren. Für diesen Regelinferenzschritt werden im Folgenden zwei unterschiedliche Ansätze wissensbasierter Reasoner untersucht und diskutiert.

Ontologie-basierte Modellierung und Reasoning

Bei diesem Ansatz erfolgt die Modellierung des Bausteinkatalog sowie der Bedingungen und Regeln in einer OWL-basierten Ontologie. Der Vorteil einer Ontologie besteht in der formalen Beschreibung, die es durch logisches Schließen bzw. Inferieren ermöglicht, automatisch Wissen zu schlussfolgern. Dazu werden die verschiedenen Bausteintypen modelliert um daraus konkrete Bausteine abzuleiten. Schließlich werden die Regeln aus den Eigenschaften und Beziehungen der Bausteininstanzen inferiert. Zur Inferenz von Wissen können Reasoning-Werkzeuge wie z. B. die Open-Source Bibliothek *FaCT++ (Fast Classification of Terminologies)* [81] genutzt werden. Beispiele für die Umsetzung einer solchen Ontologie sowie das Inferieren von Regeln wird im Kontext modularer und rekonfigurierbarer Satelliten gezeigt [73].

Individueller problemspezifischer Reasoner

Mit Hilfe der in Kapitel 4.2 eingeführten XSD/XML basierten Datenstruktur wurde ein individueller und problemspezifischer Reasoner entwickelt. Betrachtet man die drei Regeltypen global, Baustein-spezifisch und Baustein-relativ, so

wird deutlich, dass globale Regeln unabhängig von den einzelnen Bausteinen sind und direkt während der Spezifikation der Missionsanfoderungen festgelegt werden können. Ausgehend von den Bausteineigenschaften sowie Ursachen und Wirkungen die für die ausgewählten und instanziierten Bausteine definiert sind, werden entsprechende Baustein-spezifische oder Baustein-relative Regeln abgeleitet. Je nach Art der Eigenschaft, kann diese direkt in eine oder mehrere Baustein-spezifische Regeln überführt werden. Dazu werden für jede Klasse an Eigenschaften eine Funktion zur Zuordnung entsprechender Regeln und deren Parameter definiert. Die Umsetzung einer solchen Überführungsfunktion lässt sich am Besten mit Hilfe eines Beispiels verdeutlichen. Abbildung 4.6 zeigt den Auszug aus einem Bausteinkatalog der die Eigenschaften eines KAIRO 3 Baustein festlegt. Dieser Baustein enthält eine Infrarot Kamera und definiert

```
<properties xlink:type="simple">
  <Properties:FieldOfView>
    <Properties:type>"FieldOfViewType"</Properties:type>
    <Properties:angleDegree>"120"</Properties:angleDegree>
    <Properties:faceIndex>"0"</Properties:faceIndex>
  </Properties:FieldOfView>
  <Properties:GroundContact>
    <Properties:type>"GroundContactType"</Properties:type>
    <Properties:faceIndex>"3"</Properties:faceIndex>
  </Properties:GroundContact>
  <Properties:Radiation>
    <Properties:type>"RadiationType"</Properties:type>
    <Properties:interactionType>"affected"</Properties:interactionType>
    <Properties:intensity_min>"100"</Properties:intensity_min>
    <Properties:intensity_max>"1e6"</Properties:intensity_max>
    <Properties:unit>"nm"</Properties:unit>
  </Properties:Radiation>
</properties>
```

Abb. 4.6: Spezifikation der drei Bausteineigenschaften *FieldOfView*, *GroundContact* und *Radiation* am Beispiel des KAIRO 3 Infrarot Kamera Baustein.

deshalb eine Eigenschaft von Typ Sichtfeld (*FieldOfViewType*) mit den Parametern Öffnungswinkel (*angleDegree*) und Bausteinseite (*faceIndex*). Basierend auf dieser Eigenschaft wird eine Instanz der Baustein-spezifische Regel *Sichtkegel (FieldOfView)* mit den entsprechenden Parametern erzeugt (siehe Abbildung 4.7). Da es sich bei dem Baustein um ein KAIRO 3 Antriebsmodul handelt, ist

```
<rules:FieldOfView>
  <rules:ruleId>"FieldOfView␣120deg␣0face"</rules:ruleId>
  <rules:ruleType>"FieldOfView"</rules:ruleType>
  <rules:buildingBlockId>"id15"</rules:buildingBlockId>
  <rules:angleDegree>"120"</rules:angleDegree>
  <rules:faceIndex>"0"</rules:faceIndex>
</rules:FieldOfView>
<rules:AbsoluteOrientation>
  <rules:ruleId>"GroundContact"</rules:ruleId>
  <rules:ruleType>"AbsoluteOrientation"</rules:ruleType>
  <rules:buildingBlockId>"id15"</rules:buildingBlockId>
  <rules:inverse>"false"</rules:inverse>
  <rules:angleXDegree>"0"</rules:angleXDegree>
  <rules:angleYDegree>"0"</rules:angleYDegree>
  <rules:angleZDegree>"-1"</rules:angleZDegree>
</rules:AbsoluteOrientation>
```

Abb. 4.7: Abgeleitete Regeln *FieldOfView* und *AbsoluteOrientation* der entsprechenden Eigenschaften *FieldOfView* und *GroundContact* aus Abbildung 4.6.

zudem die Eigenschaft Bodenkontakt (*GroundContact*) spezifiziert. Diese wird

in eine Absolute Orientierungsregel (*AbsoluteOrientation*) überführt. Der Parameter Bausteinseite (*faceIndex*) gibt an, dass die negative Z Richtung in lokalen Baustein-Koordinatensystem in Richtung Boden zeigen soll. Dies spiegelt sich in der Fixierung der Achsen X ($angleXDegree = 0$) und Y ($angleYDegree = 0$) wieder. Eine Rotation um die Z Achse ist in jeweils 90° Schritten erlaubt ($angleZDegree = -1$).

Eine echte Inferenz ist lediglich bei den Ursachen und Wirkungen notwendig die zwischen mehreren Bausteinen wirken und zu den sogenannten Bausteinrelativen Regeln führen. Dabei wird nach direkten und indirekten Abhängigkeiten zwischen jeweils zwei der ausgewählten Bausteinen gesucht.

Direkte Abhängigkeiten sind Bausteineigenschaften die in unmittelbarer Relation zueinander stehen. Zum Beispiel strahlt Baustein A Licht mit einer Wellenlänge von 380 - 750 nm und der Baustein aus Abbildung 4.6 ist empfindlich gegenüber dieser Strahlung (Eigenschaft *Radiation*). Aus diesem Zusammenhang lässt sich die in Abbildung 4.8 dargestellte Regel für die Einhaltung einer relativen Orientierung ableiten: Diese Regel veranlasst, dass beide Bausteine nicht direkt gegen-

```
<rules:RelativeOrientation>
  <rules:ruleId>"Light_380-750nm"</rules:ruleId>
  <rules:ruleType>"RelativeOrientation"</rules:ruleType>
  <rules:buildingBlockId>"id15"</rules:buildingBlockId>
  <rules:inverse>"true"</rules:inverse>
  <rules:buildingBlock2Id>"id11"</rules:buildingBlock2Id>
  <rules:angleXDegree>"-1"</rules:angleXDegree>
  <rules:angleYDegree>"0"</rules:angleYDegree>
  <rules:angleZDegree>"180"</rules:angleZDegree>
</rules:RelativeOrientation>
```

Abb. 4.8: Abgeleitete Regel *RelativeOrientation* der relativen Eigenschaften *Radiation* zweier Bausteine.

einander orientiert werden, um das Blenden des Sensors durch die Beleuchtungseinheit zu verhindern.

Indirekte Abhängigkeiten werden meist über mehrere Stufen abgeleitet. Diese Art von Relation zwischen zwei Bausteinen ist jedoch wegen der speziellen Modellierung nur sehr selten anzutreffen.

Zur Realisierung des problemspezifischen Reasoners wurde ein Algorithmus implementiert der alle Eigenschaften der Bausteine auf entsprechende Relationen prüft und die daraus folgenden Regeln bzw. Zwangsbedingungen ableitet. Die Liste der inferierten Regeln wird in einem XML basierten Format zusammen mit den ausgewählten Bausteinen an den nächsten Syntheseschritt, der Bausteinanordnung, weitergereicht.

4.3.3 Optimierung der Bausteinanordnung

Nach der Auswahl einer optimalen Menge an Bausteinen sollen diese zu einem für die Anwendung möglichst optimalen Gesamtsystem (Bausteinkonfiguration)

unter Einhaltung der expliziten und impliziten Regeln kombiniert werden. Auf Grund der Vielzahl an Bausteinen und Schnittstellen in einem solchen System, ergibt sich für die Anordnung der Bausteine im 6-D Raum (Position und Orientierung) ein extrem großer Suchraum. Zur Lösung dieses Optimierungsproblems können verschiedenste Algorithmen verwendet werden. Grundsätzlich lassen sich jedoch die Klasse der *deterministischen Verfahren* von den *stochastischen Methoden* unterscheiden. Um möglichst schnell ein Optimum zu finden, folgen deterministische Verfahren, wie z. B. Hill-Climbing [82], zielgerichtet einem Gradienten, was jedoch durch lokale Minima zu suboptimalen Ergebnissen führen kann. Um aus lokalen Minima zu entkommen, können diese Methoden um weitere deterministische oder nicht-deterministische Verfahren wie z. B. Probabilistic Hill Climbing [83] oder Simulated Annealing [84] ergänzt werden. Ein Vorteil deterministischer Verfahren besteht darin, dass alle Schritte der Optimierung jederzeit nachvollziehbar sind, jedoch setzt dies auch eine stetig differenzierbare Optimierungsfunktion voraus. Auf Grund der sehr komplexen Optimierungsfunktion, insbesondere durch die verschiedenen Bewertungsfunktionen für die extrahierten Regeln zur Anordnung der Bausteine, kann für dieses Optimierungsproblem keine stetige Differenzierbarkeit garantiert werden. Deshalb bietet sich für die Lösung des vorliegenden Optimierungsproblem ein evolutionärer Algorithmus an, der zur Klasse der *nicht-deterministischen* Monte-Carlo Verfahren gehört [73].

Diskretisierung des Suchraumes

Auf Grund der vielen möglichen Freiheitsgrade bei der Anordnung der Module im euklidischen 6-D Raum (Positionen und Orientierungen), wurde zur Vereinfachung der Optimierung eine Diskretisierung des Suchraumes vorgenommen. Hierzu wird der Suchraum in Abhängigkeit von der Bausteingröße in ein festes Raster eingeteilt, wodurch sich die Anzahl an unterschiedlichen Position für jeden Baustein auf die Anzahl der Gitterplätze innerhalb des Suchraumes beschränkt. Gleichzeitig werden die Rotationen jeweils auf 90° Schritte beschränkt, so dass pro Baustein genau $n_{rot} = 24$ Orientierungen möglich sind. Diese Einschränkung wirkt sich zwar auf die möglichen Geometrien der einzelnen Module aus, insgesamt bleibt die Konfigurationsgenerierung jedoch allgemein genug, um alle unterschiedlichen Strukturklassen von rekonfigurierbaren Systemen (gitterförmig, baumartig, schlangenartig und mobil) zu unterstützen.

Mit der beschriebenen Diskretisierung des Suchraumes lässt sich eine Komplexitätsabschätzung der Optimierung erreichen. Die möglichen Lagen für einen Baustein n_{pose} hängen von der Anzahl möglicher Positionen n_{pos} und der möglichen Orientierungen n_{rot} ab:

$$n_{pose} = n_{rot} \cdot n_{pos}$$
$$n_{pos} = l \cdot b \cdot h$$

mit l, b und h als Länge, Breite und Höhe des euklidischen Suchraums. Da jeder Baustein mit n_{rot} möglichen Rotationen auf einem der n_{pos} Position im Suchraum platziert werden kann, ergibt sich Gesamtgröße des Zustandsraumes für m Bausteine zu:

$$(n_{rot})^m \cdot \frac{n_{pos}!}{(n_{pos} - m)!} \tag{4.5}$$

Für ein System bestehend aus $m = 20$ Bausteinen ergeben sich insgesamt $n_{pos} = 20 \cdot 20 \cdot 20 = 8000$ Bausteinpositionen mit jeweils $n_{rot} = 24$ Rotationen. Der maximale Suchraum besteht somit aus mehr als 10^{105} Kombinationen:

$$24^{20} \cdot \frac{8000!}{(8000 - 20)!} > 10^{27} \cdot 10^{78} = 10^{105} \tag{4.6}$$

Auf Grund der hohen Komplexität für bereits $m = 20$ Bausteine und insbesondere hinsichtlich der Skalierbarkeit für größere Anzahlen von Bausteinen, bieten sich wie Eingangs erwähnt, stochastische Suchverfahren besonders gut an.

Evolutionärer Optimierungsalgorithmus

Evolutionäre Optimierungsalgorithmen [82] sind dem natürlichen Evolutionsprozess nach empfunden und zielen darauf ab, eine Population von Individuen über viele Generationen durch Veränderungen bzw. Mutationen zu optimieren. Ein Individuum wird in diesem Anwendungsfall durch eine Bausteinkonfiguration, der Anordnung der ausgewählten Bausteine, repräsentiert. Mehrere dieser Individuen bilden zusammen die Population, die durch den evolutionären Algorithmus optimiert wird. Die Optimierung bzw. Evolution wird mit Hilfe einer Fitnessfunktion zur Bewertung der einzelnen Konfigurationen gesteuert. Individuen mit guter Fitness können ihre Eigenschaften durch Mutation an eine neue Generation weitergehen bzw. vererben. Durch iterative Anwendung von Bewertung, Selektion und Mutation auf die Population wird eine neue Generation von Individuen erzeugt. In jeder Iteration verändert sich die Population mehr in Richtung Optimum. Dabei können sowohl feste als auch variable Populationsgrößen gewählt werden. Mit Hilfe von Parametern kann bestimmt werden wie viele Individuen direkt oder durch Mutationen in die nächste Generation übernommen oder zufällig neu generiert werden. Viele neu erzeugte Konfigurationen sowie eine große Anzahl verschiedener Mutationen pro Individuum ermöglichen gleichermaßen eine breite Abdeckung des Suchraumes (Exploration) und eine zielgerichtete lokale Optimierung. Für eine effiziente Optimierung bietet es sich an mit einer Exploration des Suchraumes zu starten und im Laufe der Evolution den Fokus auf eine zielgerichtete lokale Suche zu verschieben. Dies kann durch Variation der Anzahl neu erzeugter Individuen erreicht werden. Neben der Aufteilung der Population haben ebenfalls die Initialisierung, die Art der Mutation und die Strenge der Selektion großen Einfluss auf die Suche.

Initialisierung

Bei der Initialisierung einer neuen Population werden entsprechend der Populationsgröße möglichst unterschiedliche Individuen angelegt. Ein Individuum wird aus zufällig angeordneten Bausteinen mit jeweils einer 6-D Pose (Position und Orientierung) instanziiert. Hierzu lassen sich verschiedene Ansätze unterscheiden. Die einfachste Methode erzeugt komplett zufällige Anordnungen der Bausteine, wobei selbst ungültige Konfigurationen mit Überlappungen von Bausteinen erlaubt sind. Für eine schnellere Konvergenz können die einzelnen Konfigurationen jedoch unter Nutzung von Vorwissen so gewählt werden, dass diese bezüglich einfachen Regeln gültig sind. Dies bedeutet, dass sich alle Bausteine innerhalb des gegebenen Konfigurationsraumes befinden müssen, es keine Überlappungen zwischen den Bausteinen geben darf und nach Möglichkeit bereits absolute Orientierungsbedingungen eingehalten werden. In dieser Arbeit wurden beide Varianten untersucht und es zeigt sich, dass eine zielgerichtete Initialisierung die Optimierung deutlich beschleunigen kann (siehe Kapitel 7.2.3).

Ein Ablaufplan zur initialen Platzierung der Bausteine bei der Erzeugung eines neuen Individuum ist in Abbildung 4.9 dargestellt. Als Eingabe dient die Größe des Konfigurationsraumes $cs = (x\ y\ z)$, die Menge der ausgewählten Bausteine M_{BS} und die Menge verfügbarer Positionen M_{pos} im Konfigurationsraum. Die Liste möglicher Positionen wird zu Beginn mit genau einer Position im Zentrum des Konfigurationsraumes initialisiert. Befinden sich noch Bausteine in der Liste M_{BS}, wird zufällig ein Baustein und eine Position aus der Liste M_{pos} ausgewählt. Für die gewählte Position werden zufällig alle möglichen Orientierungen nach einer passenden durchsucht. Konnte keine passende Orientierung gefunden werden, wird nach einer alternativen Position mit wiederum passender Orientierung gesucht. Konnte eine passende Position und Orientierung gefunden werden, wird der Baustein aus der Liste M_{BS} gelöscht und zum Individuum mit der Position und Orientierung hinzugefügt. Zudem wird die gewählte Position aus der Liste M_{pos} entfernt und die neuen freien Nachbarpositionen werden hinzugefügt. Der Algorithmus endet, wenn die Liste M_{BS} leer ist und somit alle Bausteine platziert wurden. Bei der in dieser Arbeit entwickelten Implementierung der *GenerateComposed* Methode wurde besonderen Wert auf eine effiziente Ausführung gelegt, um auch große Populationen schnell initialisieren zu können. Der entsprechende Pseudocode ist in Algorithmus 1 im Anhang A.3 dargestellt.

Zusätzlich zur Initialisierung von neuen Individuen, kann diese Methode ebenfalls zur Generierung neuer zufälliger Konfigurationen während der Optimierung verwendet werden.

Fitnessfunktion

Bei der Optimierung mit Hilfe eines evolutionären Algorithmus, ist es essentiell eine fundierte Kosten- bzw. Fitnessfunktion zur Bewertung der einzelnen Individuen aufzustellen. Die Gesamtfitness setzt sich dabei meist aus der gewichteten

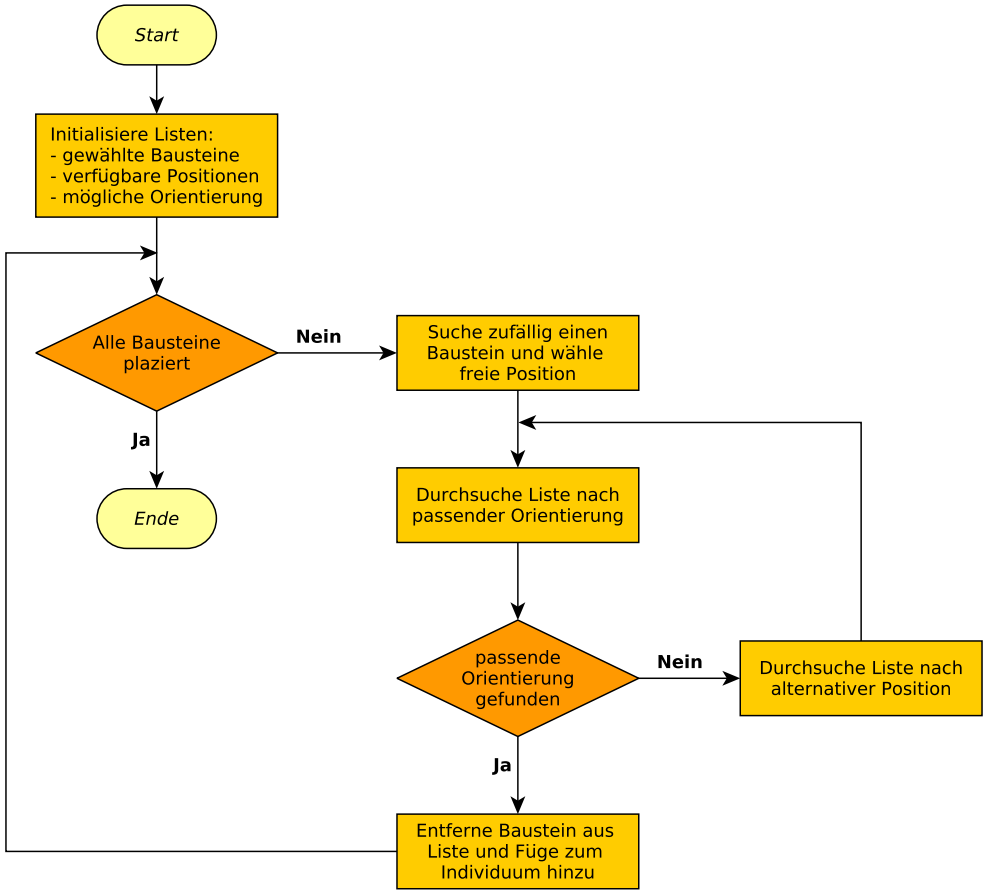


Abb. 4.9: Das Flussdiagramm zeigt den grundlegenden Ablauf zum erzeugen neuer Individuen mit der *GenerateComposed* Methode. Eine Menge von 1000 einzelner Individuen bildet zusammen die Population zur Optimierung.

Summe mehrerer Teilfunktionen zusammen. Gleichung 4.7 zeigt die in dieser Arbeit entwickelte Fitnessfunktion F, bestehend aus mehreren speziellen Teilfunktionen zur unabhängigen Bewertung einzelner Bedingungen zur Bausteinanordnung. Der Einfluss der jeweiligen Teilfunktionen auf die Gesamtfitness F lässt sich durch die Gewichte $\alpha, \beta, \gamma, ..., \iota$ einstellen.

$$\begin{aligned} F = {} & \alpha \cdot f_{Cluster} + \beta \cdot f_{Cluster\_Struktur} + \gamma \cdot f_{Fluss} + \delta \cdot f_{Verbindung} \\ & + \varepsilon \cdot f_{Rel\_Abstand} + \zeta \cdot f_{Rel\_Orientierung} \\ & + \eta \cdot f_{Spez\_Position} + \theta \cdot f_{Spez\_Orientierung} + \iota \cdot f_{Verdeckung} \end{aligned} \tag{4.7}$$

Jede Teilfunktion repräsentiert dabei exakt eine Bedingung, die aus den bereits eingeführten Regeltypen abgeleitet ist. Zur Auswertung der einzelnen Teilfunktionen dienen sogenannte Fitnesstests, die durch entsprechende Algorithmen repräsentiert sind. Dies ermöglicht unter anderem die einfache Erweiterung um zusätzliche Regeln und Regeltypen sowie deren zugehörigen Algorithmen und Auswertefunktionen.

Fitness-Tests (Metriken der Optimierung)

Die Fitness-Tests dienen zur Bestimmung der Qualität bzw. Kosten spezieller Bedingungen die als Teilfunktion in die Gesamtfitness einfließen. Meist ist die Auswertung dieser Bedienungen sehr komplex und lässt sich nicht direkt über mathematische Funktionen abbilden. Für diese Bedingungen wurden die im Folgenden vorgestellten Tests in Form von Algorithmen entwickelt und implementiert. Zur besseren Vergleichbarkeit werden die Teilfunktionen mit Hilfe der Gleichung 4.8 auf Werte zwischen 0 und 1 normiert. Der Term E_x entspricht dabei dem absoluten Fehler der für die jeweilige Bedingung x[2] des zu bewertenden Individuum berechnet wurde. Da in diesem Fall 0 der besten Fitness entspricht, handelt es sich bei f_x und somit auch bei der Gesamtfunktion F um eine invertierte Fitness bzw. einer Kostenfunktion.

$$f_x = 1.0 - e^{-E_x} \tag{4.8}$$

- Cluster-Test: Die Überprüfung, ob eine Konfiguration aus der gewünschten Anzahl unabhängiger Teilsysteme besteht, wird durch Berechnen der vorhandenen Baustein-Cluster erreicht. Dazu wird der zur Bestimmung der Informationsverteilung in Netzwerken häufig eingesetzte *Flooding-Algorithmus* verwendet. Zunächst wird ein zufälliger Baustein ausgewählt und alle seine Nachbarbausteine markiert. Für alle markierten Bausteine werden iterativ alle Nachbarn markiert, bis kein direkter Nachbarn mehr gefunden wird. Existieren noch nicht markierte Bausteine, befinden diese sich in einem neuen Cluster. Der Algorithmus untersucht die verbleibende Teilmenge auf weitere Cluster, bis alle Bausteine markiert sind. Als Ausgabe des Algorithmus erhält man die Anzahl vorhandener Bausteincluster, die zur Verwendung als Fitnessterm aufbereitet und normiert wird. Aus der Differenz der Clusterzahl $C_{verfügbar}$ zum Erwartungswert $C_{erwartet}$ und einer Normierung zwischen 0 und 1 wird der Fitnesswert $f_{Cluster}$ bestimmt:

$$f_{Cluster} = 1.0 - e^{-(|C_{verfügbar} - C_{erwartet}|)} \tag{4.9}$$

- Fluss-Test: Die Übertragung von verschiedenen Ressourcen wie Energie, Daten oder Wärme zwischen einzelnen Bausteinen und im Gesamtsystem wird mit Hilfe von Flusstests analysiert und bewertet. Um die komplexen Zusammenhänge mit parallelen Übertragungspfaden in einem Bausteinverbund zu berechnen, bietet sich der *Ford-Fulkerson Algorithmus* an. Die Methode wird benutzt, um auf Basis eines gewichteten Graphen den maximalen Fluss einer entsprechenden Ressource zwischen verschiedenen Bausteinen zu bestimmen.
- Verbindungs-Test: Der Verbindungs-Test prüft die korrekte und optimale Verbindung aller Bausteine im Bausteinverbund. Dazu werden jeweils die

[2]Die Normierungsfunktion gilt für alle x aus der Menge {*Cluster*, *Cluster_Struktur*, *Fluss*, *Verbindung*, *Rel_Abstand*, *Rel_Orientierung*, *Spez_Position*, *Spez_Orientierung*, *Verdeckung* }

Zustände angrenzender Seitenflächen von zwei benachbarten Bausteinen untersucht. Jede Seitenflächen besitzt dabei einen der in Abbildung 4.10 dargestellten Zustände. Eine Verbindung wird als gültig bezeichnet, wenn zwei androgyne Schnittstellen *eConAndrogyn* oder je eine aktive *eConMale* und eine passive Schnittstelle *eConFemale* aufeinander treffen. Zusätzlich werden zwei zusammentreffende Seitenflächen ohne Schnittstelle (*eConNone*) weder als gültig noch als ungültig angesehen. Alle anderen Kombinationen werden als ungültig deklariert, wobei jede Seitenfläche mit dem Zustand *eConBlocked* als blockiert gezählt wird, sobald ein anderer Baustein an diese Seite angrenzt. Gleichung 4.10 zeigt die Berechnung des Verbindungsfehlers $E_{Verbindung}$ durch die Kombination der beiden Terme V_{fehler} und $V_{kompakt}$. Der Fehlerterm V_{fehler} wird zur Vermeidung bzw. Minimierung ungültiger $V_{ungültig}$ und blockierter Verbindungen $V_{blockiert}$ benutzt. Um möglichst kompakte Konfigurationen zu erzeugen wird der Term $V_{kompakt}$ zur Maximierung der Anzahl gültiger Verbindungen benutzt. Dazu wird $V_{gültig}$ durch die Anzahl aller möglichen Verbindungen $V_{möglich}$ dividiert.

```
enum eConState
{
  eConNone,      // Seite ohne Schnittstelle
  eConBlocked,   // Seite darf nicht verdeckt werden
  eConAndrogyn,  // Seite mit einer androgynen Schnittstelle
  eConMale,      // Seite mit einer aktiven Schnittstelle
  eConFemale,    // Seite mit einer passiven Schnittstelle
  eConDimension
}
typedef ConState;
```

Abb. 4.10: Definition der möglichen Zustände von Bausteinseitenflächen.

$$\begin{aligned} E_{Verbindung} &= V_{fehler} + V_{kompakt} \\ V_{fehler} &= \left(V_{ungültig} + V_{blockiert}\right) \\ V_{kompakt} &= e^{-\left(V_{gültig}/V_{möglich}\right)} \end{aligned} \tag{4.10}$$

- Abstands-Test: Zur Überprüfung von Baustein-relativen Abstandsbedingungen, die jeweils zwischen zwei Bausteinen definiert sind, wird der Abstands-Test genutzt. Dabei wird für jede Abstandsbedingung zunächst die *Manhatten-Distanz* zwischen den Bausteinen berechnet. Abhängig von der Art der Bedingung (minimaler oder maximaler Schwellenwert) wird die entsprechende Differenz zum vorgegeben Schwellenwert als Fehlerwert betrachtet. Da für einen Baustein mehrere relative Abstandsbedingungen definiert sein können, summieren sich die jeweiligen Fehlerwerte zu $E_{Rel\_Abstand}$ auf. Der resultierende Fitnesswert $f_{Rel\_Abstand}$ wird wiederum durch Normierung zwischen 0 und 1 mit Hilfe der Gleichung 4.8 gewonnen.
- Orientierungs-Test: Der Orientierungs-Test bewertet für alle Orientierungsbedingungen die Orientierungsunterschiede zwischen jeweils zwei Baustei-

nen oder zu einer absoluten Orientierung. Dazu werden zunächst die Winkeldifferenzen für alle drei Raumachsen bestimmt und mit einem vorgegeben Offset verglichen. Die Abweichungen aller drei Achsen zu dem Offset werden aufsummiert und wiederum mit dem entsprechenden Gesamtfehler verrechnet. Der Test lässt sich sowohl für Baustein-relative $E_{Rel\_Orientierung}$ als auch absolute Orientierungsbedingungen $E_{Abs\_Orientierung}$ einsetzen. Beide Fehlerwerte werden entsprechend Gleichung 4.8 normiert.

- Positions-Test: Zur Überprüfung der Baustein-spezifischen Positionierung (absolute Lage) wurde der Positions-Test eingeführt. Dabei wird geprüft, ob sich ein Baustein an einer bestimmten absoluten Position befindet, die relativ zum globalen Systemkoordinatensystem betrachtet wird. Der Fehlerwert $E_{Abs\_Position}$ wird durch die Manhattan-Distanz zwischen der Bausteinposition und der gewünschten Zielposition bestimmt. Durch Normierung mit Gleichung 4.8 erhält man den Fitnesswert $f_{Abs\_Position}$ zwischen 0 und 1.
- Verdeckungs-Test: Der Verdeckungs-Test dient zur Vermeidung von Abschattungen (z. B. bei Sensoren) durch andere Bausteine. Für alle Seitenflächen von Bausteinen kann eine entsprechende Verdeckungsbedingung mit einem Öffnungswinkel zwischen $0°$ und $180°$ spezifiziert werden. Kein anderer Baustein darf sich innerhalb dieses Sichtbereiches (FOV) befinden. Zur Überprüfung wird zunächst die Entfernung entlang der Mittelachse des Sichtkegels zum Kegelursprung berechnet. Zusammen mit dem spezifizierten Öffnungswinkel lässt sich der Kegelradius an der entsprechenden Entfernung bestimmen. Ein anderer Baustein befindet sich im Sichtbereich, wenn sein orthogonaler Abstand zur Kegel-Mittelachse kleiner ist als der Kegelradius. Entsprechend des Abstandes zur Kegel-Mittelachse wird der Fehlerwert $E_{Verdeckung}$ erhöht. Der Fitnesswert $f_{Verdeckung}$ wird wiederum über die Gleichung 4.8 normalisiert.

Die vorgestellten Fitness-Tests ermöglichen für jedes Individuum die Überprüfung und Bewertung der in Kapitel 4.2.4 modellierten Regeln zur Bausteinanordnung.

Mutation

Ein weiterer wichtiger Bestandteil der evolutionären Optimierung ist das Verändern einzelner Individuen in jeder Iteration. Diese Variation, auch Mutation genannt, erfolgt mit Hilfe von Mutationsoperationen. Für die Optimierung von Bausteinkonfiguration wurden in dieser Arbeit die drei Mutationen *Move*, *Rotate* und *Swap* definiert. Die Operationen werden jeweils auf die einzelnen Bausteine eines Individuums angewendet und sind im Folgenden näher beschrieben.

- *Move*-Operation: Die *Move*-Operation bewegt einen einzelnen Baustein von seiner aktuellen Position auf einen freien Platz, der mindestens eine Verbindung mit einem anderen Baustein hat. Zur Bestimmung eines freien Platzes

werden alle Nachbarpositionen der vorhandenen Bausteine bewertet und die Position mit der besten Bewertung ausgewählt. Die Bewertung erfolgt durch Betrachtung der potentiellen Verbindungen mit den Nachbarbausteinen, wie in Gleichung 4.11 gezeigt. Der vollständige Algorithmus zum Bewegen von Bausteinen ist in Algorithmus 5 im Anhang A.4 dargestellt.

$$f_{pos} = V_{gültig} + randomInt\left[1, V_{gültig}\right] - 2 \cdot \left(V_{ungültig} + V_{blockiert}\right) \tag{4.11}$$

- *Rotate*-Operation: Das Rotieren von Bausteinen zur Variation eines Individuums wird mit der *Rotate*-Operation durchgeführt. Ein Baustein wird unter Berücksichtigung seiner Abmessungen gedreht, in dem eine Orientierung zufällig aus der Liste aller möglichen Rotationen ausgewählt wird. Auf Grund der gitterförmigen Anordnung der Bausteine sind die Möglichkeiten auf jeweils 90° Winkeldifferenz pro Achse beschränkt. Ist für den zu rotierenden Baustein eine absolute Orientierungsbedingung definiert, so werden lediglich die unter dieser Randbedingung möglichen Drehungen betrachtet. Insgesamt werden zehn Versuche unternommen bevor Algorithmus 6 im Anhang A.4 abbricht und der Baustein seine bisherige Orientierung beibehält.

- *Swap*-Operation: Mit Hilfe der *Swap*-Operation werden zwei zufällig ausgewählte Bausteine vertauscht mit dem Ziel die Bausteinkonfiguration zu mischen ohne die äußere Geometrie zu verändern. Das Tauschen zweier exakt gleich großer Bausteine erfolgt dabei ohne weitere Überprüfung durch einfaches austauschen der Positionen. Sind die ausgewählten Bausteine unterschiedlich groß, so wird die im Anhang A.4 in Algorithmus 7 dargestellte Methode angewendet. Dabei werden entsprechend der Bausteingrößen weitere Nachbarbausteine ausgewählt, bis beide Tauschmengen die gleiche Geometrie haben. Die geometrisch identischen Bausteinmengen können nun ihre Positionen tauschen.

In jeder Mutationsphase können die Mutationsoperationen *Move*, *Rotate* oder *Swap* mehrfach mit verschiedenen Bausteinen einzelner Individuen durchgeführt werden. Auf jedes ausgewählte Individuum werden insgesamt $1..n$ Mutationen angewendet, wobei n der Bausteinanzahl pro Individuum entspricht. Für jede Mutation wird zunächst zufällig ein Baustein selektiert. Über die Zufallsvariable op im Intervall $[0, 1]$ wird darauf hin zufällig eine Mutationsoperation bestimmt, wobei die Verteilung entsprechend der Schwellenwerte *prop_move*, *prop_rotate*, *prop_swap* erfolgt. Es hat sich gezeigt, dass eine gleichmäßige Verteilung zu je $1/3$ gute Ergebnisse liefert. Durch die probabilistische Variation der Anzahl an Mutation und zufälligen Auswahl von Bausteinen wird der Suchraum möglichst gut exploriert.

Abhängig von der Populationsgröße wird in jeder Generation jedoch nur ein gewisser Teil der Individuen mutiert, da Mutationen grundsätzlich der zielgerichteten lokalen Optimierung dienen. Um eine bessere Exploration und damit eine

Konvergenz des Algorithmus in Richtung globalem Optimum zu erreichen, werden 10 % der Population durch zufällig neu erzeugte Individuen ersetzt. Eine neue Population besteht somit aus $1/3$ der besten Individuen der Vorgängerpopulation, $1/10$ (10 %) neu erzeugter Individuen und $17/30$ aus mutierten Individuen. Somit ist ebenfalls immer eine konstante Anzahl Individuen pro Population sichergestellt. Gute Ergebnisse konnten mit einer Populationsgröße von $p = 1000$ Individuen erzielt werden. Mit Hilfe eines zusätzlichen Gewichtungsfaktor kann das Verhältnis aus mutierten und neuen Individuen während der Laufzeit variiert werden.

Abbruchkriterium

Um festzustellen, wann die Optimierung beendet werden kann, können mehrere Abbruchkriterien spezifiziert werden, abhängig davon ob eine Lösung als akzeptabel angesehen wird oder keine weitere Verbesserung mehr zu erwarten ist. Bei der Optimierung von Bausteinkonfigurationen wurden insgesamt zwei verschiedene Abbruchkriterien definiert. Einerseits wird die Optimierung beendet, sobald eine gewisse Anzahl an Iterationen i_{max} überschritten ist. Dadurch wird sichergestellt, dass immer eine Lösung gefunden wird. Zudem wird die Qualität der Population mit Hilfe zweier Merkmale überprüft: Liegt die Fitness der besten n Individuen unter einem Schwellwert f_{min} oder ist die Fitness der besten m Individuen identisch, so kann davon ausgegangen werden, dass keine weitere Verbesserung gefunden werden kann.

Auf Grund der in Kapitel 7.2.3 durchgeführten Versuche und Tests hat sich gezeigt, dass die Anzahl Iterationen auf $i_{max} = 1000$ begrenzt werden kann. Dabei wurde ein Fitnessschwellenwert von $f_{min} = 1.0$ für die besten $n = 33$ % Individuen gewählt. Zudem wird der Algorithmus bei mehr als $m = 80$ % identischer Individuen abgebrochen.

Als Ergebnis der Optimierung, wird die Menge aller Konfigurationen zusammen mit einer Bewertung der einzelnen Optimierungskriterien bzw. Bedingungen ausgegeben, so dass dem Anwender verschiedene Konfigurationen zur Auswahl stehen.

Interne Datenstrukturen und Parallelisierung

Zur Beschleunigung der Optimierung wurde bei der Implementierung darauf geachtet, dass alle zeitkritischen Funktionen wie das Generieren neuer Individuen (*generateComposed*), das Berechnen der Fitness (*calcFitness*) und das Mutieren ausgewählter Individuen (*mutate*) parallelisierbar sind. Dazu wurde der gesamte Quellcode in C++ implementiert und insbesondere Datentypen und Datenstrukturen der Standard Template Library (STL) verwendet. Mit Hilfe des Frameworks OpenMP [85] konnten rechenintensive Schleifen und Iteratoren unter Ausnutzung aller Prozessorkerne eines Multi-Kernsystems parallelisiert

werden. Abbildung 4.11 zeigt beispielhaft die Parallelisierung der Fitnessberechnung. Durch Verwendung der STL Standard Datentypen wird sichergestellt, dass keine Speicherzugriffsprobleme durch die parallele Ausführung auf mehreren Prozessoren auftreten. Die Evaluierung in Kapitel 7.2.3 veranschaulicht und diskutiert die dadurch erzielten Laufzeitverbesserungen.

```
// --------- calculate fitness of current population P(t) ----------
#pragma omp parallel for
for (size_t i = start_index; i < m_population.size(); ++i)
{
  m_population[i]->calcFitness(m_config_space);
}
```

Abb. 4.11: Verwendung des OpenMP [85] Pragma *omp parallel for* zur Parallelisierung der Fitness-Berechnung aller Individuen einer Population.

4.4 Benutzerschnittstelle und Anwendungsfälle

Bei der Konzeptionierung und Implementierung der Synthese von Bausteinkonfigurationen für modulare und rekonfigurierbare Systeme, wurde darauf geachtet, dass eine Anpassung an verschiedene Robotersysteme mit wenig Aufwand realisierbar ist. Durch die Generalisierung der Konzepte ist es ebenfalls möglich den Prozess für unterschiedliche Anwendungsdomänen einzusetzen. In dieser Arbeit wurde neben der Generierung modularer Roboterkonfigurationen auch die Optimierung rekonfigurierbarer Satelliten untersucht. Entsprechend des gewählten Anwendungsfeldes wird zwischen Computer-Aided-Robot-Design (CARD)[3] und Computer-Aided-Satellite-Design (CASD)[4] unterschieden.

4.4.1 Computer-Aided-Robot-Design (CARD)

Das Werkzeug für den computerunterstützten Roboterentwurf (CARD) stellt dem Anwender eine interaktive Schnittstelle zur Verfügung, die durch den Syntheseprozess aus Kapitel 4.3 führt. Der Benutzer hat die Möglichkeit die jeweiligen Eingaben und Zwischenergebnisse vor und nach jedem Syntheseschritt in einer eigenen grafischen Oberfläche zu überprüfen und anzupassen. Diese Oberflächen bilden zusammengefasst das CARD Werkzeug, einen interaktiven Wizard zur Generierung und Optimierung rekonfigurierbarer Roboterkonfigurationen.

Im ersten Schritt des CARD Werkzeug (siehe Abbildung 4.12) spezifiziert der Anwender die Anforderungen basierend auf der durchzuführenden Mission und der zu erwartenden Umweltbedingungen. Neben der *Art der Mission* und der *Gelände Beschaffenheit* werden Informationen bezüglich der *geplanten Einsatzdauer*

[3] engl. für computerunterstützter Roboterentwurf
[4] engl. für computerunterstützter Satellitenentwurf

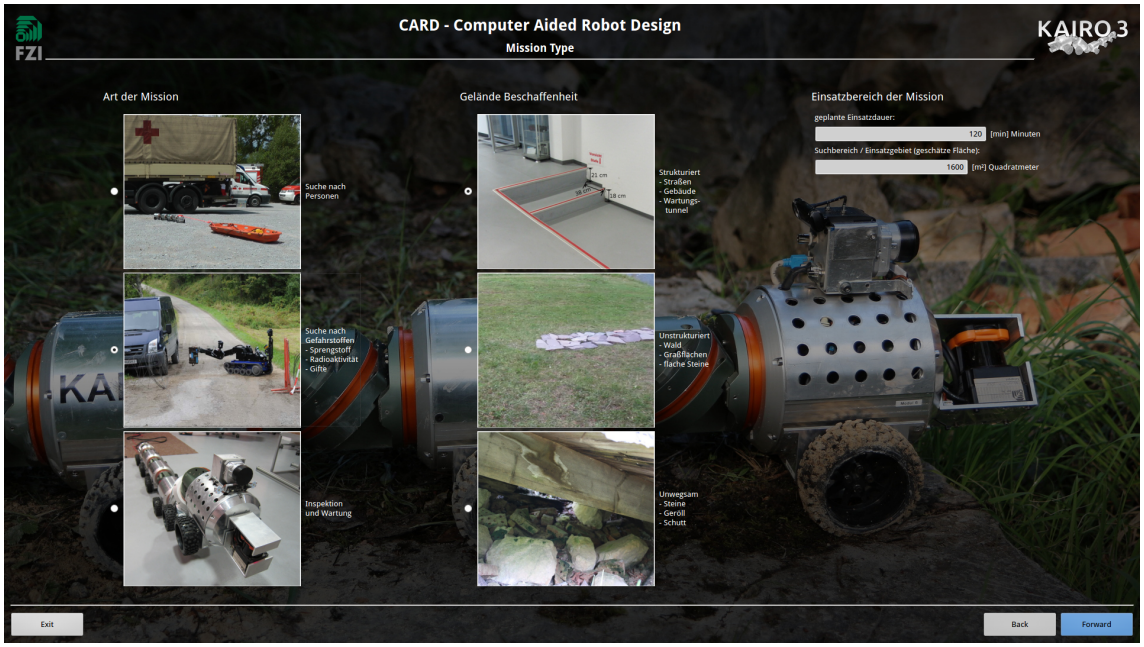


Abb. 4.12: Der erste Schritt des CARD Werkzeug ermöglicht das Festlegen der Missionsparameter und Rahmenbedingungen.

sowie der *Größe des Einsatzgebietes* als Eingabe benötigt. Basierend auf den spezifizierten Missionsparametern werden entsprechende Anforderungen (*Baustein-*, *Typen-*, *Komponenten-* und *Ressourcen-Anforderungen*) und globale Regeln erzeugt. Die Anforderungen können im folgenden Schritt des CARD Werkzeug begutachtet und editiert werden.

Die Benutzeroberfläche in Abbildung 4.13 zeigt den zweiten Schritt der interaktiven Robotersynthese. Auf der linken Seite befindet sich eine Liste aller im Katalog spezifizierten Komponenten und Ressourcen. Auf der rechten Seite werden die zur Durchführung des geplanten Einsatz vorgeschlagenen Komponenten und minimalen Ressourcen aufgelistet. Darunter befinden sich jeweils die Eigenschaften der aktuell ausgewählten Komponente oder Ressource. Der Anwender hat die Möglichkeit die Vorauswahl anzupassen und spezifische Komponenten hinzuzufügen. Sind alle Anforderungen festgelegt, wird die in Kapitel 4.3.1 erläuterte automatische Bausteinauswahl gestartet.

Das Ergebnis der automatischen Bausteinauswahl wird im dritten Schritt des CARD Werkzeugs dargestellt. Die Oberfläche in Abbildung 4.14 enthält auf der linken Seite eine Liste aller Bausteine aus dem Bausteinkatalog. Auf der rechten Seite sind die selektierten Bausteine aufgelistet. Diese erfüllen die in den vorherigen Schritten spezifizierten Anforderungen und enthalten zusätzliche, für den Betrieb notwendige Ressourcen und Komponenten. Durch das Selektieren der Bausteine werden entsprechende Eigenschaften, wie zum Beispiel beinhaltete Komponenten, Masse und Größe angezeigt. Der Anwender hat die Möglichkeit weitere Bausteine auszuwählen oder zu entfernen bevor der nächste Syntheseschritt, die in Kapitel 4.3.2 erläuterte Regelinferenz, ausgeführt wird.

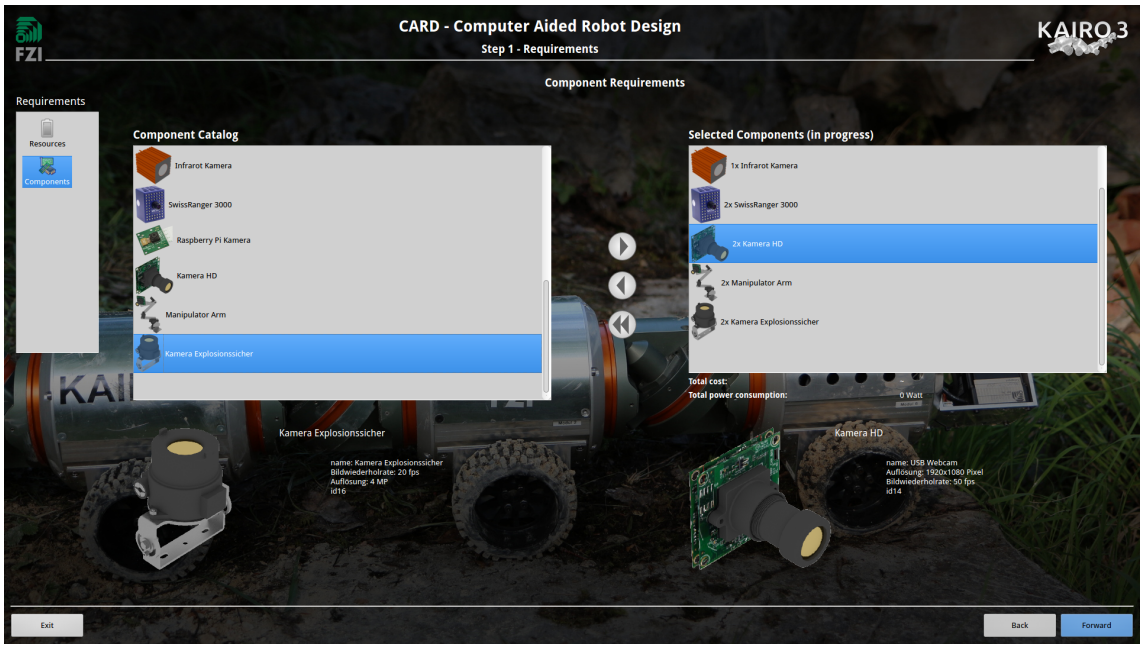


Abb. 4.13: Die zweite CARD Oberfläche dient zur Spezifikation von Ressourcen und Komponenten Anforderungen.

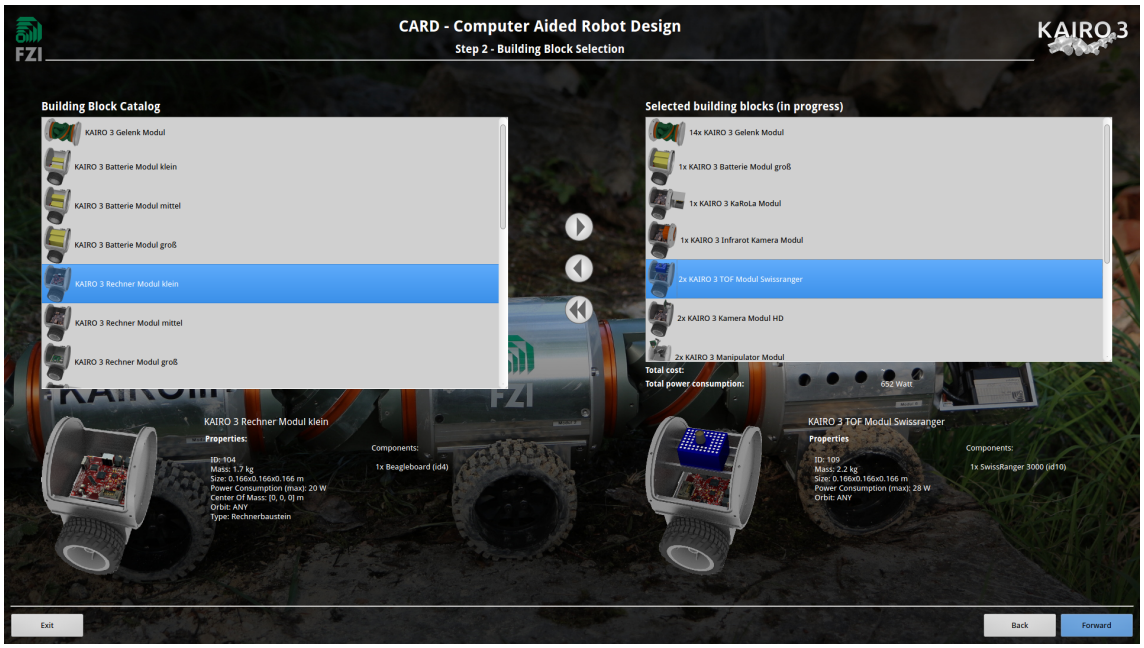


Abb. 4.14: Der dritte CARD Schritt erlaubt die Überprüfung der für die Gesamtkonfiguration ausgewählten Bausteine.

Nach dem Auflösen der Abhängigkeiten zwischen den Bausteinen und dem Ableiten der Regeln zur Anordnung der Bausteine, werden die entsprechenden Regeln wie in Abbildung 4.15 dargestellt. Für jede der drei Regelklassen Global,

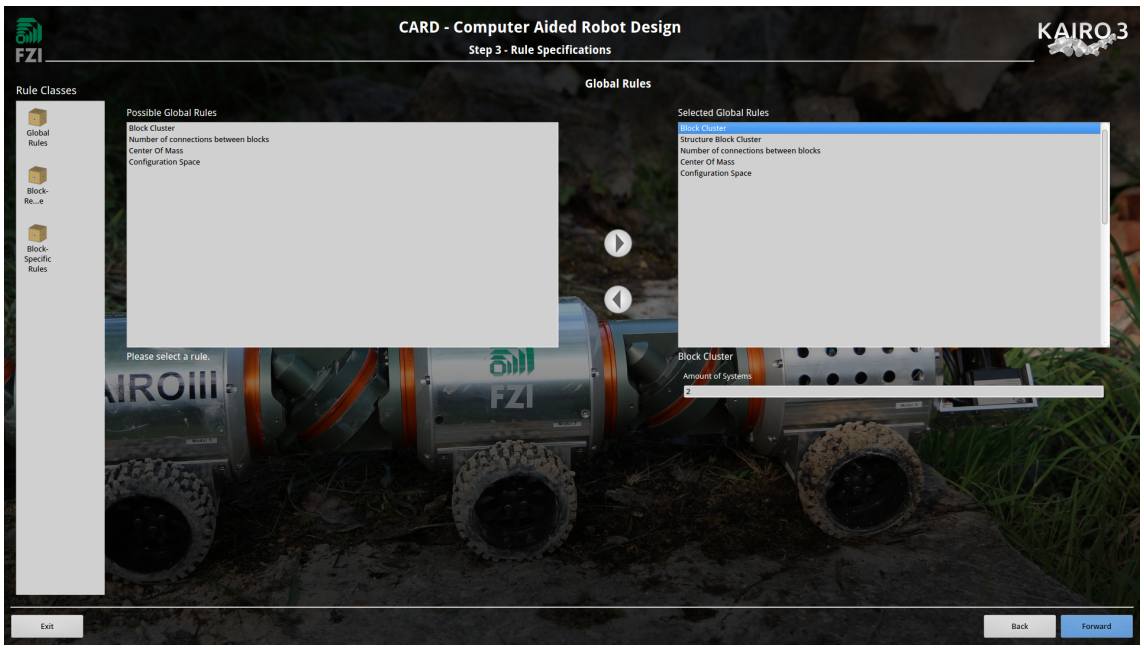


Abb. 4.15: Die Regelspezifikation im vierten CARD Schritt, gestattet dem Benutzer, vorhandene Regeln zu modifizieren, zu entfernen oder neue hinzufügen.

Baustein-relativ und Baustein-spezifisch existiert eine separate Liste abgeleiteter Regeln. Die Regelklasse sowie die entsprechenden Regeln werden auf der linken Seite der Oberfläche angezeigt. Auf der rechten Seite kann der Anwender globale Optimierungseinstellungen, wie z. B. den Konfigurationsraum oder die Anzahl von Bausteincluster verändern. Zudem lassen sich Abhängigkeiten zwischen Bausteinen oder Baustein-spezifische Regeln, wie z. B. der Sichtkegel eines Sensors hinzufügen, ändern oder entfernen. Die im vorherigen Schritt bestimmten Bausteine werden zusammen mit den Regeln im folgenden Syntheseschritt verwendet, um die Anordnung der Bausteine wie in Kapitel 4.3.3 beschrieben, zu optimieren.

Der letzte Schritt des CARD Werkzeug visualisiert die Ergebnisse des Syntheseprozess. Abbildung 4.16 zeigt die Ergebnisoberfläche, die dem Benutzer eine Liste der zehn besten Konfigurationen zur Auswahl stellt. Für eine selektierte Konfiguration werden dem Anwender die Optimierungsergebnisse hinsichtlich Einhaltung der Regeln sowie ein 3-D Modell der Bausteinanordnung angezeigt. Durch Selektion einzelner Bausteine innerhalb der 3-D Ansicht werden zusätzliche Informationen zum entsprechenden Baustein eingeblendet.

Während des gesamten interaktiven Prozess zur Generierung von Bausteinkonfigurationen kann der Anwender jeder Zeit zu den vorherigen Schritten zurück

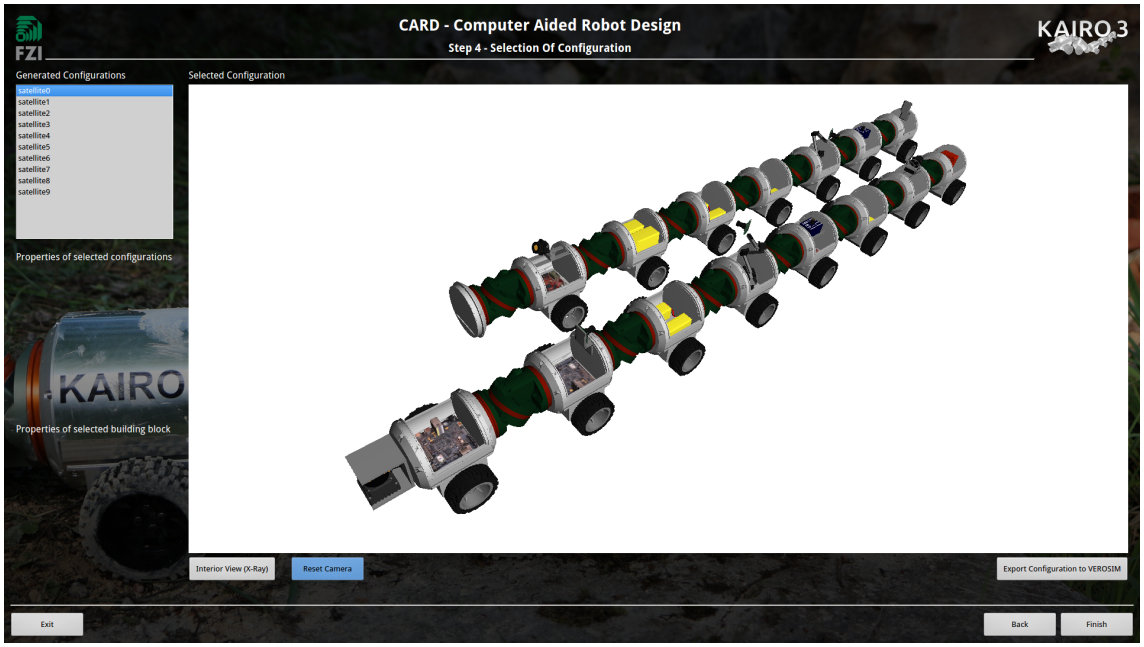


Abb. 4.16: Visualisierung der Optimierungsergebnisse mit Hilfe einer 3-D Ansicht der finalen Konfigurationen.

kehren, die getroffene Auswahl bzw. Spezifikation anpassen und den entsprechenden Optimierungsschritt erneut ausführen.

4.4.2 Computer-Aided-Satellite-Design (CASD)

Die vorgestellten Algorithmen und Verfahren zur Erzeugung und Optimierung von Roboterkonfigurationen lassen sich ebenfalls auf eine Vielzahl an unterschiedlichen modularen Systemen übertragen. In Folgenden wird dies beispielhaft an Hand modularer und rekonfigurierbarer Satelliten für den Einsatz im Orbit gezeigt. Als Grundlage für den Syntheseprozess dient ein Bausteinkatalog mit verfügbaren Satellitenkomponenten und -bausteinen sowie deren Eigenschaften und entsprechenden Regeln. Eine vollständige Liste der modellierten Satellitenbausteine und deren Eigenschaften befindet sich im Anhang A.1.2. Die Synthese folgt basierend auf den Eingabedaten den selben Schritten wie bei der Erzeugung von Roboterkonfigurationen. Die Algorithmen der einzelnen Optimierungsschritte wurden bis auf kleine Veränderungen an der Kosten- und Auswertungsfunktionen direkt übernommen. Zur Eingabe von Daten und insbesondere zur interaktiven Ausführung der Konfigurationsgenerierung modularer Satellitensysteme, wurde basierend auf der bereits vorgestellten interaktiven grafischen Benutzerschnittstelle CARD das sogenannte CASD Werkzeug entwickelt. Abbildung 4.17 und 4.18 zeigen das auf die Optimierung modularer Satelliten angepasste Benutzerinterface des zweiten und dritten Syntheseschritt.

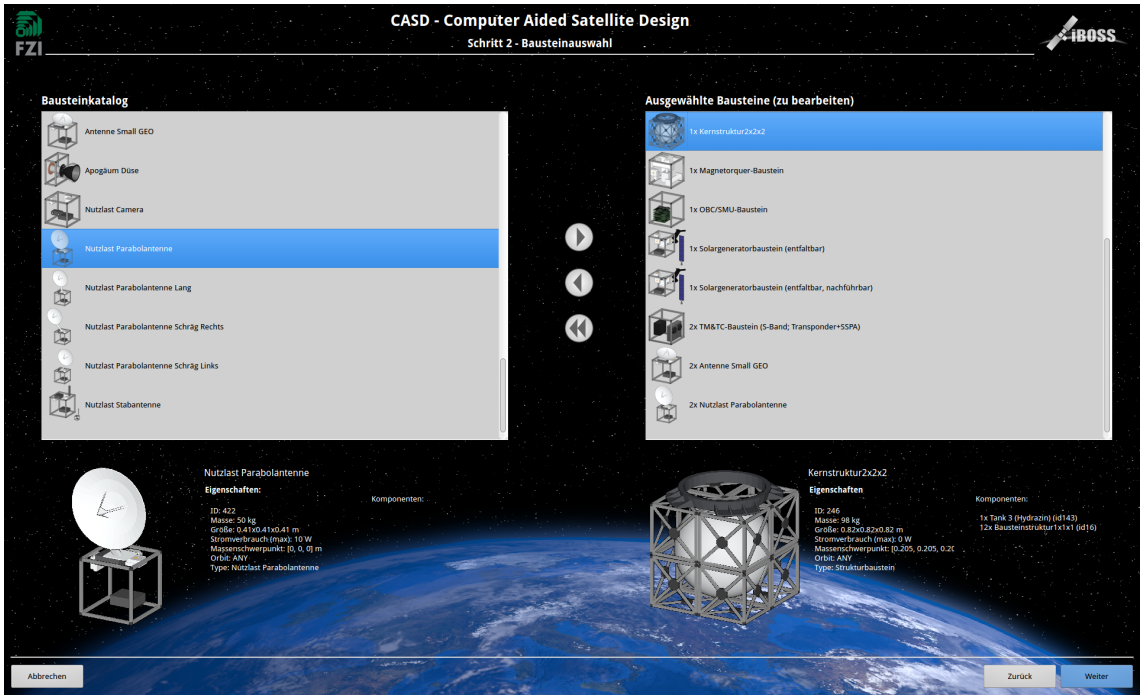


Abb. 4.17: Der zweite CASD Schritt ermöglicht, wie beim CARD Werkzeug, die Bearbeitung der für die Gesamtkonfiguration benötigten Bausteine.

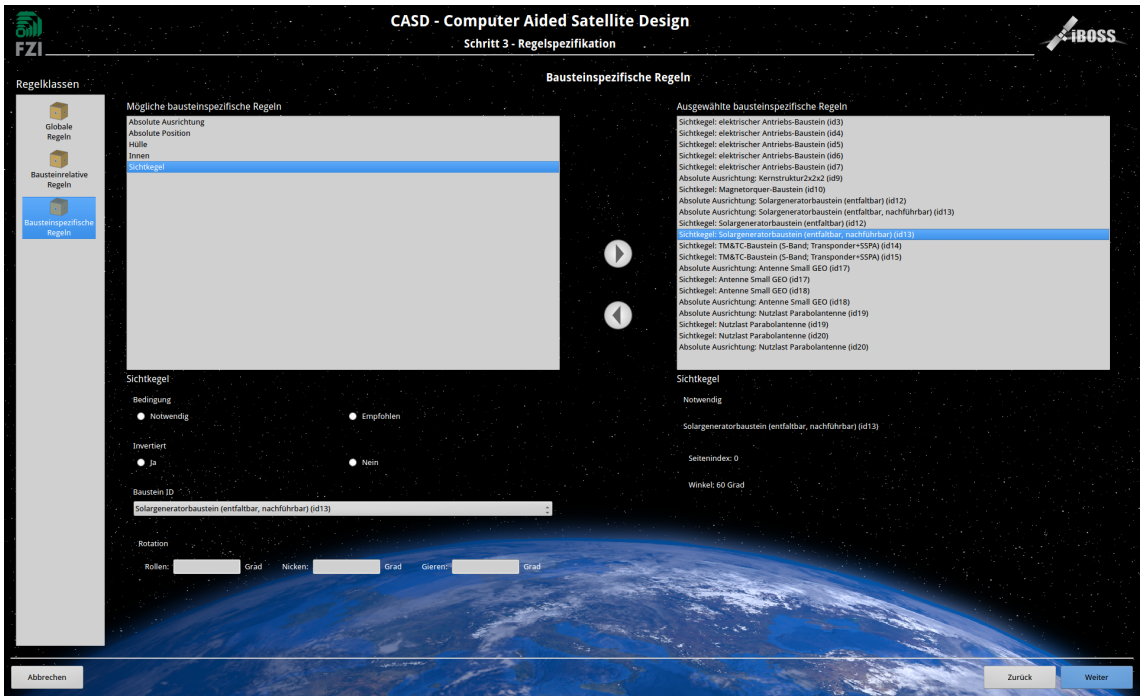


Abb. 4.18: Die Regelspezifikation im dritten CASD Schritt, ermöglicht dem Benutzer das Bearbeiten von Regeln zur Bausteinanordnung.

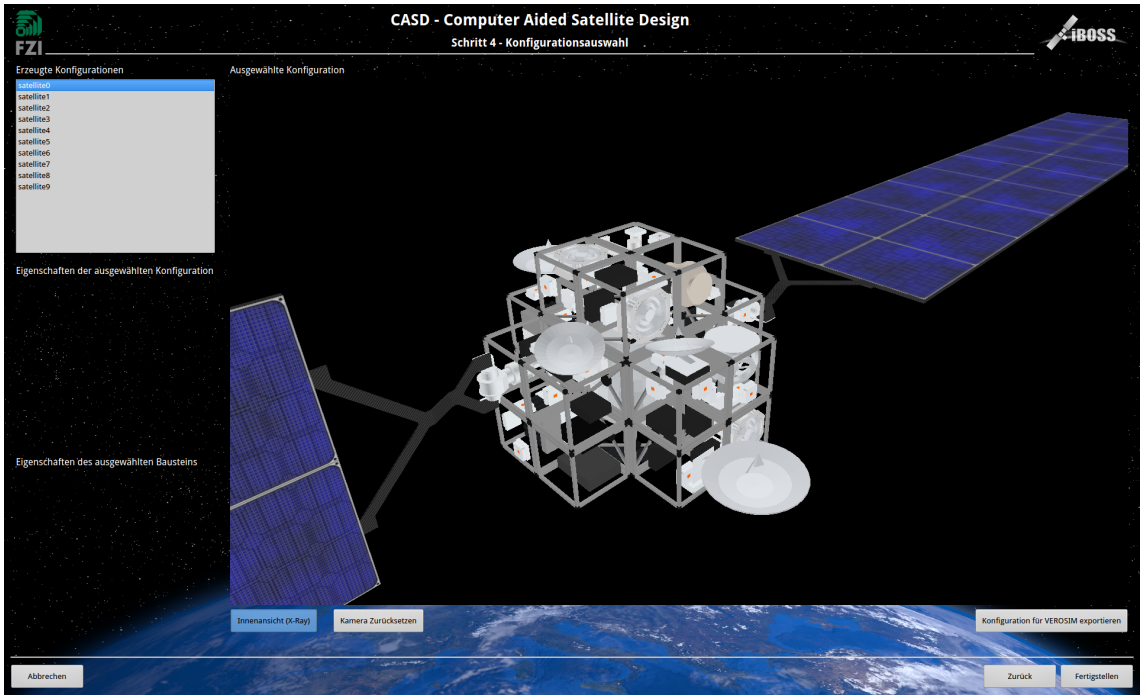


Abb. 4.19: Zur erweiterten Analyse der generierten Konfigurationen, ermöglicht der *X-Ray* Modus dem Anwender einen Blick in das Satelliten Innere.

Im Unterschied zum CARD Werkzeug verfügt die 3-D Ansicht über zwei verschiedene Ansichtsarten. Die Satellitenkonfigurationen werden zunächst als geschlossenes Modell dargestellt um einen realistischen Eindruck zu vermitteln. Im sogenannten *X-Ray* Modus werden die Bausteinmodelle teilweise transparent visualisiert. Somit erhält der Benutzer einen besseren Einblick in das Innere von Satellitenkonfiguration mit sehr vielen Bausteinen. Dabei werden, wie in Abbildung 4.19 gezeigt, die innen liegenden Bausteine und deren Komponenten sichtbar. Hat der Benutzer eine Auswahl getroffen, so kann die gewünschte Konfiguration im einheitlichen XML-Datenformat (siehe Kapitel 4.2) exportiert werden. Dadurch kann eine Konfiguration auch außerhalb des CASD Werkzeug weiterverwendet werden, beispielsweise um eine Evaluierung mit einem Simulationsprogramm durchzuführen.

4.5 Zusammenfassung

Dieses Kapitel beschreibt die Generierung und Optimierung von Konfigurationen modularer und rekonfigurierbarer Roboter. Mit Hilfe des entwickelten Syntheseprozess können Roboterkonfigurationen in einem mehrstufigen Optimierungsprozess erzeugt werden. Als Eingabe für den Prozess dienen ein Baustein- und Regelkatalog sowie Missionsparameter und -anforderungen. Zunächst wird die Modellierung des Bausteinkatalog, der Bedingungen für die Bausteinauswahl und der Eigenschaften sowie der daraus abgeleiteten Regeln für die Bausteinan-

ordnung detailliert erläutert. Darauf aufbauend werden verschiedene Optimierungsverfahren und -algorithmen zur Realisierung der drei Optimierungsstufen der Synthese diskutiert. Für die Bausteinauswahl (erster Syntheseschritt) werden die drei Verfahren *Brute-Force-Suche*, *Greedy-Algorithmus* und *Ganzzahlige lineare Optimierung* untersucht und beispielhaft implementiert. Der zweite Syntheseschritt, die Regelinferenz, kann mit Hilfe einer Ontologie-basierten Modellierung und entsprechendem Reasoning oder als individueller problemspezifischer Reasoner umgesetzt werden. Beide Möglichkeiten werden dargestellt und diskutiert. Für den letzten Schritt der Synthese, der eigentlichen Optimierung der Bausteinanordnung wird ein evolutionärer Algorithmus vorgeschlagen. Es werden die essentiellen Aspekte Initialisierung, Fitnessfunktion, Fitness-Tests (Metriken), Mutationsoperationen und Abbruchkriterium beschrieben. Die Darstellung des CARD Werkzeug verdeutlicht die Umsetzung der Robotersynthese in einer interaktiven Benutzerschnittstelle die ein reales Werkzeug zur Konfigurationsgenerierung von RMRS bietet. Darüber hinaus wird mit dem CASD Werkzeug gezeigt, dass sich die Synthese auch auf andere Anwendungsfälle, wie der Optimierung rekonfigurierbarer Satellitensysteme übertragen lässt.

5 Konfigurationsabhängige Adaption der Lokomotion

Die sichere Fortbewegung eines Robotersystems in unstrukturiertem Gelände ist für den Einsatz von RMRS in Katastrophenszenarien und Rettungsmissionen unerlässlich. Radgetriebene Roboter können sich auf ebenem und festem Untergrund sehr effizient fortbewegen, haben jedoch oft Schwierigkeiten in unwegsamem Gelände wie Grasflächen, Steinen oder weichem Sand. Im Gegensatz dazu eignen sich schlangenartige Robotersysteme sehr gut für unstrukturiertes Gelände, sind aber meist langsamer als radgetriebene Systeme. Aus diesem Grund werden hier hybride Robotersysteme betrachtet, die einen modularen Aufbau ähnlich wie schlangenartige Systeme ausweisen. Zusätzlich zu den Gelenkmodulen verfügen einige Module über Räder für eine schnellere Fortbewegung. Diese Roboter ähneln in ihrem strukturellen Aufbau dem von Raupen. In diesem Zusammenhang werden verschiedene biologische Bewegungsarten analysiert und die Übertragung dieser auf schlangenartige Robotersysteme untersucht. Weiterhin werden Möglichkeiten zur Messung der Bodenbeschaffenheit diskutiert und mit Hilfe verschiedener Messgrößen die Umsetzung einer automatischen Auswahl und Adaption der Fortbewegungsart dargestellt.

5.1 Omega-Lokomotion

Schmetterlingslarven, wie die Spannerraupe (Geometra papilionaria), gehören zur Gattung der Raupen und besitzen eine ganz ausgefallene und erstaunliche Art der Fortbewegung. Auf Grund fehlender Bauchbeinpaare, benutzen Spannerraupen die vorderen und hinteren Beinpaare, um sich abwechselnd festzuhalten und den gesamten Körper zu einer vertikalen Schleife (ähnlich einem Ω) zu formen. Deshalb wird diese Fortbewegungsart auch als Omega (Ω) Lokomotion bezeichnet [86]. Abbildung 5.1 (links) zeigt eine Spannerraupe jeweils im zusammengezogenen Zustand (oben) und im ausgestreckten Zustand (unten). Der Bewegungsablauf ist ebenfalls in Abbildung 5.1 (rechts) schematisch dargestellt und kann ausgehend vom ausgestreckten Zustand in drei Schritte unterteilt werden: *Zusammenziehen*, *Suchen* und *Ausstrecken* [87].

Zur Übertragung dieser Fortbewegungsart auf Robotersysteme, ist es zunächst notwendig die Stabilität während des gesamten Bewegungsablauf zu betrachten. Insbesondere der Schritt *Suchen* erfordert, dass ein technisches System in der Lage

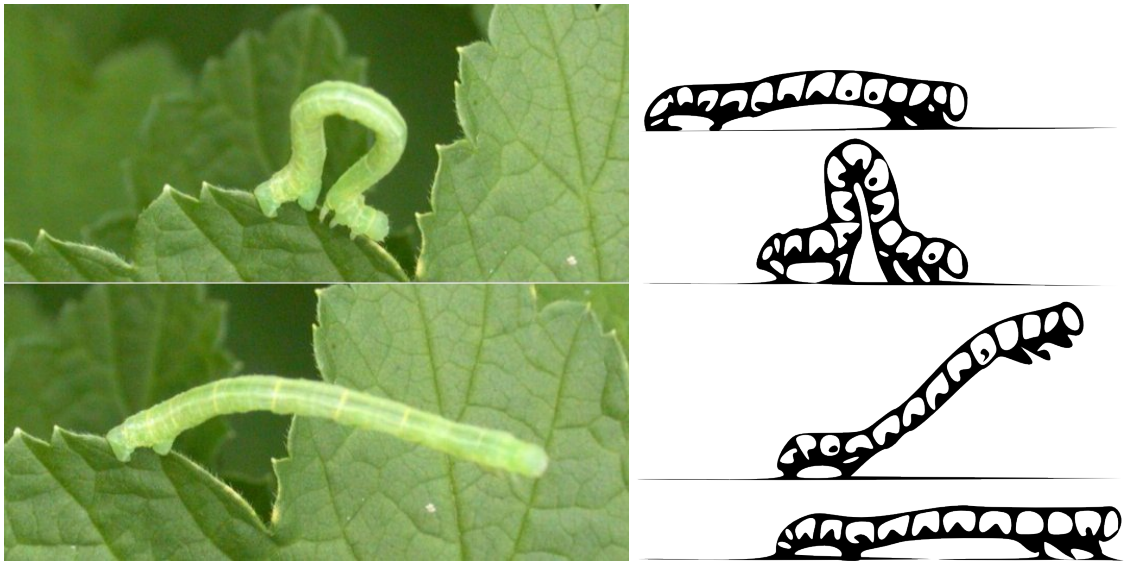


Abb. 5.1: Das linke Foto zeigt die besondere Fortbewegungsart einer Spannerraupe (Geometra papilionaria) in ihrer natürlichen Umgebung [86]. Rechts ist der Ablauf der Ω-förmigen Bewegung mit seinen drei Phasen *Zusammenziehen*, *Suchen* und *Ausstrecken* dargestellt [88].

ist, die hinteren Körpersegmente auf dem Boden zu fixieren, sodass sich die vorderen Segmente (mehr als zwei drittel des Systems) frei in der Luft bewegen können. Da dies mit aktuellen Robotern nur sehr eingeschränkt realisierbar ist, wird hier als Vereinfachung angenommen, dass sich sowohl die vorderen als auch die hinteren Segmente eines Roboters stets gleichzeitig auf dem Boden befinden. Dadurch wird der dreistufige Bewegungsablauf auf die zwei Schritte *Zusammenziehen* und *Ausstrecken* reduziert, jedoch kann die statische Stabilität während des gesamten Ω Bewegungsablauf sichergestellt werden. Betrachtet man zusätzlich die seitliche Stabilität gegen Verkippungen, wird dies durch die Räder mit einem entsprechenden Radstand erreicht.

Die Umsetzung auf einem radgetriebenen schlangenartigen Roboter wie KAIRO 3, wird durch einen Zustandsautomaten mit den zwei Zuständen *Zusammenziehen* und *Ausstrecken* erreicht. Während sich die Steuerung im Zustand *Zusammenziehen* befindet, wird der mittlere Bereich des Roboters durch entsprechendes Bewegen der Gelenke[1], nach oben bewegt. Gleichzeitig werden die vorderen Module auf dem Boden fixiert (durch Einschalten der Motorbremse der Antriebsräder) und die hinteren Module durch Antreiben der Räder nach vorne bewegt. Dies unterstützt die Gelenke der mittleren Module beim Anheben. Die maximale Höhe des so geformten Ω, hängt in erster Linie von der Anzahl der Module und somit der Konfiguration ab. Haben die mittleren Module die gewünschte Höhe erreicht, geht der Zustandsautomat in den Zustand *Ausstrecken*

[1]Die Bewegung und Ansteuerung der Gelenke wird an dieser Stelle nicht betrachtet. Es wird davon ausgegangen, dass die Robotersteuerung die Vorgabe von Höhenverschiebungen für einzelne Module, sogenannte Z-Shifts, erlaubt (siehe Kapitel 2.5.2).

über. In diesem Zustand werden die hinteren Module fixiert, die vorderen Räder angetrieben und die mittleren Module mit Hilfe der Gelenke langsam abgesenkt. Hat der Roboter seine ausgestreckte Position erreicht, beginnt der Bewegungszyklus von vorne. Durch Invertieren der Bewegungsrichtung sowie der Fixierung der Module, kann von einer Vorwärtsbewegung auf eine Rückwärtsbewegung umgeschaltet werden [87].

Die maximale pro Bewegungszyklus zurückgelegte Entfernung, wird in Abhängigkeit von der Anzahl an Robotermodulen n als Funktion $mmd(n)$ (*Maximum Moving Distance*) wie folgt definiert:

$$mmd(n) = 2\left\lfloor \frac{n-3}{2} \right\rfloor s - 2\sqrt{\left(\left\lfloor \frac{n-3}{2} \right\rfloor s\right)^2 - h_{\max}}$$

Die zurückgelegte Entfernung hängt dabei vor allem von dem Abstand zweier aufeinander folgender Gelenke (auch als Segmentlänge s bezeichnet), der maximalen Höhe h_{max} und der Modulanzahl n ab [87]. Eine Auswertung der tatsächlich erreichten Entfernung für einen Bewegungszyklus wird in Kapitel 7.3.1 gezeigt.

5.2 Raupenartige Lokomotion

Eine weitere in der Biologie sehr gängige Fortbewegungsart, lässt sich bei Raupen bzw. Larven von Schmetterlingen oder Faltern beobachten. Abbildung 5.2 (links) zeigt beispielhaft eine Raupe der Eastern Panthea Moth. Der zugehörige Bewegungsablauf in Abbildung 5.2 (rechts) zeigt, dass diese Raupen ihre Füße nur indirekt zur Fortbewegung nutzen. Vielmehr wird die Bewegung durch Schrumpfen und Ausdehnen von mehreren Körpersegmenten erreicht.

Für eine Vorwärtsbewegung beginnen die hinteren Körpersegmente sich zu verkürzen. Diese Kompression wird Segment für Segment nach vorne weiter propagiert, bis sich schließlich der Kopf um die Länge der Kompression nach vorne schiebt. Bei dieser wellenförmigen Bewegung werden die Beine der einzelnen Segmente leicht angehoben und ebenfalls nach vorne geschoben. Durch den segmentartigen Aufbau lässt sich diese Fortbewegungsart auch auf modulare schlangenartige Roboter übertragen. Zumal viele Roboter dieser Art, wie z. B. KAIRO 3, nicht die Möglichkeit besitzen, Module linear zu verkürzen oder auszudehnen, wird eine Verkürzung des Roboters durch das Anheben eines Gelenkmoduls zwischen zwei Antriebsmodulen erreicht. Der Bewegungsablauf beginnt mit dem Anheben des letzten Robotermodul. Danach wird iterativ jeweils das nächste Modul angehoben und das vorherige Modul abgesenkt. Sobald das vorletzte Modul abgesenkt wurde, wird auch der Kopf gesenkt und der gesamte Bewegungszyklus wiederholt sich. Die Erzeugung dieser Bewegung kann durch den in Abbildung 5.3 dargestellten Zustandsautomaten mit den vier Zuständen

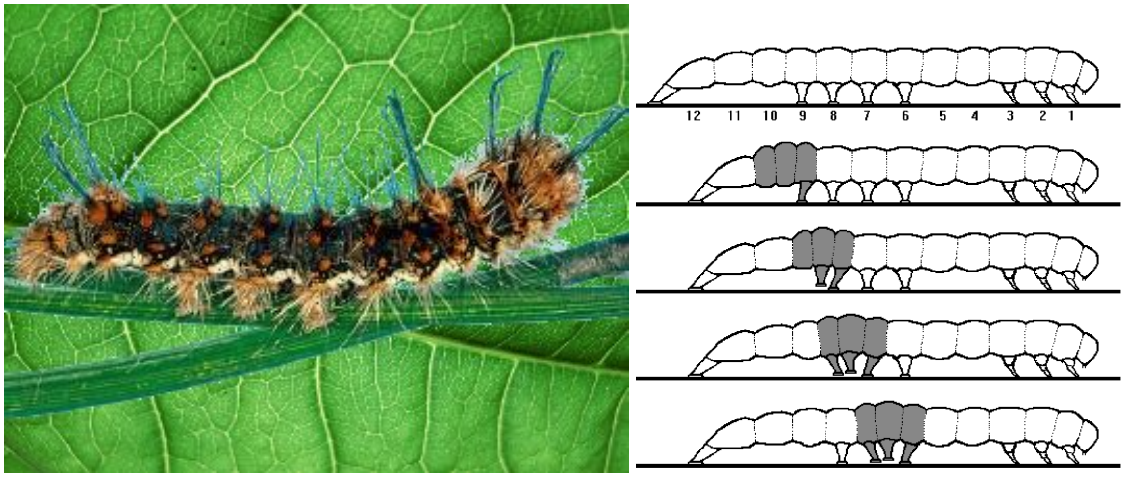


Abb. 5.2: Das linke Foto zeigt eine Raupe der Eastern Panthea Moth (panthea furcilla), die sich zu einem Nachtfalter verwandelt und zur Familie der Eulenfalter (Noctuidae) gehört. Der rechts skizzierte Bewegungsablauf dieser Raupenart besteht im wesentlichen aus dem linearen Zusammenziehen und Ausdehnen von Körpersegmenten [86, 88].

Letztes Anheben, *Nächstes Anheben*, *Vorheriges Absenken* und *Letztes Absenken*, beschrieben werden.

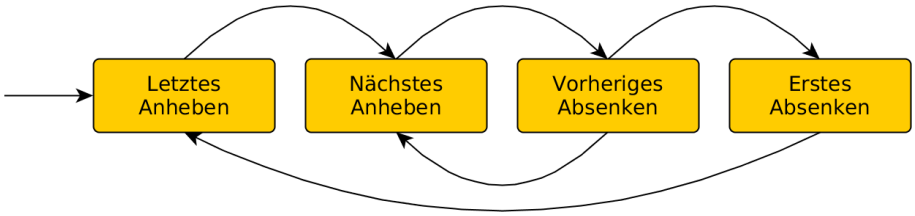


Abb. 5.3: Abstrahierter Zustandsautomat einer Raupenbewegung bestehend aus vier Zuständen.

Dieser Ablauf ermöglicht eine wellenförmige Fortbewegung des Roboters, ohne den Einsatz der Antriebsräder. Um jedoch die Gesamtlänge des Roboters während des Anhebens und Absenkens konstant zu halten, muss das entsprechende Nachbargelenk eine Ausgleichsbewegung durchführen. Aus diesem Grund werden das erste und letzte Modul im Zustandsautomaten separat behandelt [87]. Kapitel 5.2.1 verdeutlicht die Notwendigkeit einer solchen Ausgleichsbewegung.

5.2.1 Berechnung der Ausgleichsbewegung

Zur Veranschaulichung der Ausgleichsbewegung zeigt Abbildung 5.4 die Situation nach dem Absenken des Gelenkmoduls J_0 (*Vorheriges Absenken*). Das Gelenkmodul J_0 befindet sich auf Höhe h_0. Im nächsten Schritt soll das Gelenkmodul J_2 angehoben werden (*Nächstes Anheben*). Beim schrittweisen Verschieben des

Scheitelpunktes von J_1 nach J_2 kann die Länge $L = l_0 + l_1 + l_2$ lediglich konstant gehalten werden, wenn die Höhen h_1 und h_2 simultan angepasst werden. Im Fol-

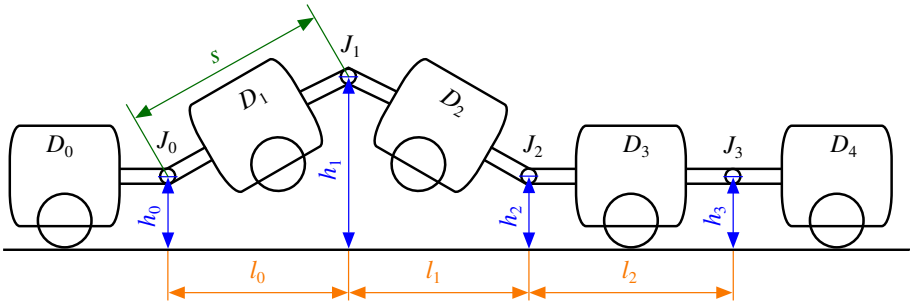


Abb. 5.4: Berechnung einer Ausgleichsbewegung zur Stabilisierung der Roboterlänge während der Raupenbewegung [87].

genden werden zwei unterschiedliche Arten zur Berechnung der Ausgleichsbewegung für die raupenartige Fortbewegung beschrieben: Einen geometrischen Ansatz zur Berechnung einer exakten Lösung und eine numerische Approximation zur schnellen Berechnung.

Geometrische Berechnung der Ausgleichsbewegung

Als Ausgangspunkt für die Berechnung einer exakten geometrischen Lösung dient die bereits in Abbildung 5.4 vorgestellte Situation nach dem Absenken von Gelenk J_0 (*Vorheriges Absenken*). Im darauf folgenden Zustand (*Nächstes Anheben*) wird das Gelenk J_2 angehoben, wie in Abbildung 5.5 geometrisch vereinfacht dargestellt. Der Abstand s entspricht der Länge eines Robotersegmentes[2] und

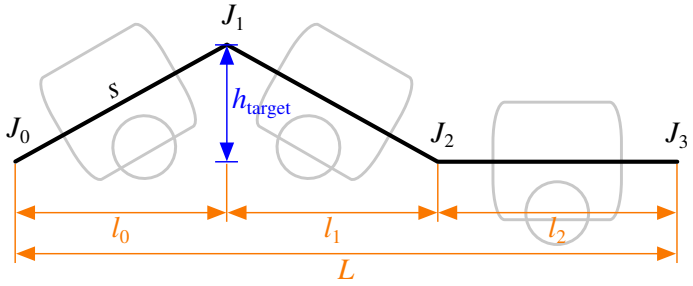


Abb. 5.5: Ausgangssituation für die geometrische Berechnung der Ausgleichsbewegung [87].

h_{target} ist die manuell festgelegte maximale Steighöhe eines Gelenkmoduls. In Abbildung 5.6 ist zu erkennen, dass ein Anheben von Gelenk J_2 zu einem Anstieg der Höhe h_2 führt. Dies erfordert bei konstanter Länge L eine Veränderung der Höhe h_1.

[2]Die Länge eines Robotersegment, bestehend aus einem Antriebs- und einem Gelenkmodul, wird aus dem Abstand zwischen den Mittelpunkten zwei aufeinander folgender Gelenkmodule ermittelt.

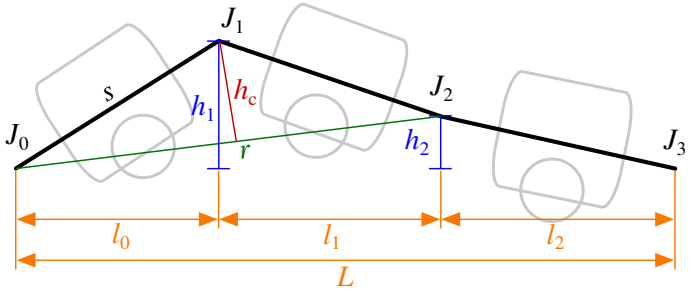


Abb. 5.6: Kontinuierliche Berechnung der Abstände r und h_c in Abhängigkeit der Höhe h_2 während des Anhebens des Gelenkes J_2 [87].

Zur kontinuierlichen Berechnung der Höhe h_1, wird zunächst die Gleichung zur Längenberechnung $L = l_0 + l_1 + l_2$ zwischen J_0 und J_3 aufgestellt. Die Gesamtlänge L entspricht der aufsummierten Abstände zwischen den Gelenken l_0, l_1 und l_2 und kann gleichgesetzt werden mit der Projektion der Strecken J_0 und J_1 sowie J_1 und J_2 auf die Grundfläche ($\sqrt{s^2 - h_{target}^2}$) addiert zur Strecke J_2 und J_3 (s):

$$L = l_0 + l_1 + l_2 = s + 2\sqrt{s^2 - h_{target}^2} \tag{5.1}$$

Das Ziel der Ausgleichsbewegung ist es, die Länge L in allen Zeitschritten konstant zu halten. Durch Einsetzen der Länge $l_2 = \sqrt{s^2 - h_2^2}$ in Gleichung 5.1 lässt sich die Länge $l_{0,1} = l_0 + l_1 = L - l_2$ zwischen J_0 und J_2 bestimmen. Mit Hilfe der Länge $l_{0,1}$ und der Höhe h_2 können die Abstände $r = \sqrt{l_{0,1}^2 - h_2^2}$ und $h_c = \sqrt{s^2 - \left(\frac{r}{2}\right)^2}$, wie in Abbildung 5.6 zu sehen, ermittelt werden.

Zur Berechnung der Höhe h_1 in Abhängigkeit von der Höhe h_2, wird der Abstand h_c auf die z-Achse projiziert: $h_p = h_c \cdot \frac{l_{0,1}}{r}$. Abbildung 5.7 zeigt einen vergrößerten Ausschnitt zur Verdeutlichung dieser Projektion und der resultierenden Berechnung von $h_1 = \frac{h_2}{2} + h_p$ durch Addition der projizierten Höhe h_p und der halben Höhe h_2.

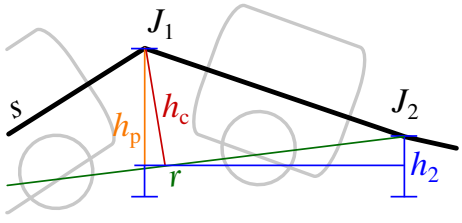


Abb. 5.7: Vergrößerte Ansicht aus Abbildung 5.6 zur Darstellung der projizierten Höhe h_p und der Berechnung von h_1 aus h_p und der halben Höhe h_2 [87].

Insgesamt lässt sich die Ausgleichsbewegung für h_1 in Abhängigkeit von h_2, s und $l_{0,1}$ mit Gleichung 5.2 zu jedem Zeitpunkt während der Aufwärtsbewegung von Gelenk J_2 berechnen [87].

$$h_1 = \frac{h_2}{2} + \sqrt{s^2 - \left(\frac{\sqrt{l_{0,1}^2 - h_2^2}}{2}\right)^2} \cdot \frac{l_{0,1}}{\sqrt{l_{0,1}^2 - h_2^2}} \tag{5.2}$$

Numerische Berechnung der Ausgleichsbewegung

Neben der geometrischen Berechnung der Ausgleichsbewegung, wurde ebenfalls eine approximierte numerische Lösung entwickelt. Im ersten Schritt wird Gleichung 5.1 zur Längenberechnung $L = l_0 + l_1 + l_2$ zwischen J_0 und J_3 verwendet. Der Abstand zwischen zwei benachbarten Gelenken J_i und J_{i+1} lässt sich in Abhängigkeit von der Segmentlänge s und den Höhen h_i und h_{i+1} als $l_i = \sqrt{s^2 - (h_i - h_{i+1})^2}$ darstellen. Setzt man dies in die Gleichung zur Längenberechnung ein, erhält man Gleichung 5.3.

$$L = \sqrt{s^2 - (h_0 - h_1)^2} + \sqrt{s^2 - (h_1 - h_2)^2} + \sqrt{s^2 - (h_2 - h_3)^2} \tag{5.3}$$

Durch eine Reihe von Annahmen kann die Länge L berechnet werden:

1. Annahme: Alle Höhen h_0 bis h_3 werden relativ zur Mittelachse des Roboters gemessen $=> h_0 = 0\ mm$.
2. Annahme: Die Gelenke J_0 und J_3 bleiben während der gesamten Bewegung konstant auf einer Höhe $=> h_0 = h_3 = 0\ mm$.
3. Annahme: Die Segmentlänge s beträgt 330 mm und maximale Höhe h_{max} ist auf 150 mm beschränkt $=> h_{max} = 150\ mm$ und $s = 330\ mm$.

Da die Länge L zu jedem Zeitpunkt der Bewegung konstant sein soll, lässt sich L für den in Abbildung 5.4 dargestellten Zustand mit $h_0 = h_2 = h_3 = 0\ mm$, $h_1 = 150\ mm$ und $s = 330\ mm$ berechnen.

Die berechnete Länge L wird in die Gleichung 5.3 eingesetzt. Es gelten nach wie vor die getroffenen Annahmen, jedoch werden nun h_1 und h_2 variabel gehalten. Löst man die so entstandene Gleichung nach h_1 auf, erhält man eine polynomiale Funktion, die jedoch einer Parabel sehr ähnelt. Durch Approximation auf ein Polynom dritten Grades erhält man Gleichung 5.4.

$$h_1 = -0.00000794 \cdot h_2^3 - 0.00181056 \cdot h_2^2 + 0.48173159 \cdot h_2 + 150.0173769 \tag{5.4}$$

Mit Hilfe dieses Polynoms lässt sich die Ausgleichsbewegung für h_1 in Abhängigkeit von h_2 effizient zu jedem Zeitpunkt während der Aufwärtsbewegung von Gelenk J_2 berechnen. Die Gleichung 5.4 kann durch Verallgemeinerung (Austausch der Variablen h_1 und h_2 durch h_x und h_{x+1}) für alle Gelenke in der Kette genutzt werden.

Erreichen die Gelenke J_1 und J_2 die gleiche Höhe, wird sowohl beim numerischen als auch beim geometrischen Ansatz, ein Zustandswechsel von *Nächstes Anheben* auf *Vorheriges Absenken* durchgeführt. Der neue Zustand führt eine Abwärtsbewegung für das Gelenk J_1 aus, die durch kontinuierliche Verringerung der Höhe h_1 erreicht wird. Analog zum Zustand *Nächstes Anheben* muss auch eine entsprechende Ausgleichsbewegung für Gelenk J_2 berechnet werden, um die Länge L konstant zu halten. Dazu kann ebenfalls Gleichung 5.4 beziehungsweise Gleichung 5.2 verwendet werden, jedoch zur Bestimmung der Höhe h_2 in Abhängigkeit von h_1.

Zur Vervollständigung der raupenartigen Fortbewegung, werden die Zustände *Nächstes Anheben* und *Vorheriges Absenken* abwechselnd so lange ausgeführt, bis die wellenförmige Bewegung das erste Gelenk Modul (den Kopf) des Roboters erreicht hat (siehe Zustandsautomat in Abbildung 5.3). Im letzten Schritt des Bewegungszyklus, wird das erste Modul durch den Zustand *Erstes Absenken* unabhängig von den benachbarten Modulen abgesenkt. Danach beginnt der gesamte Bewegungszyklus durch Anheben des letzten Moduls, von vorne. Eine Rückwärtsbewegung des Roboters kann anstelle einer Vorwärtsbewegung durch einfache Invertierung des Bewegungszyklus erreicht werden [87].

Zur Auswertung der hier diskutierten Ω und raupenartigen Bewegungen auf einem modularen schlangenartigen Roboter, wurden beide Verfahren sowohl in der Simulation als auch auf dem Roboter KAIRO 3 implementiert und evaluiert. Die Ergebnisse werden in Kapitel 7.3.1 diskutiert.

5.3 Erkennung der Bodenbeschaffenheit

Eine zuverlässige Fortbewegung von radgetriebenen schlangenartigen Robotern in unwegsamem und unstrukturiertem Gelände kann nur durch die Adaption der Bewegungsmuster bzw. durch das Umschalten der Fortbewegungsart erreicht werden. Im Falle eines Einsatzszenario bei dem der Roboter autonom seine Aufgabe erfüllt, muss der Roboter in der Lage sein, selbstständig die am Besten geeignet Fortbewegung auszuwählen. Dazu ist die Erkennung der Bodenbeschaffenheit zur Bewertung der aktuellen Fortbewegungsart auf dem Terrain notwendig. Der Fokus liegt hierbei weniger auf der Klassifikation des Geländes, beispielsweise nach Steinen, Sand oder Asphalt, vielmehr benötigt man eine Aussage über die Effektivität der Fortbewegung bzw. wie gut sich eine Fortbewegungsart für das Gelände eignet. Zur Bestimmung der Effektivität der Fortbewegung eignet sich speziell bei radgetriebenen schlangenartigen Robotern, wie KAIRO 3, die Messung der Motorströme der einzelnen Räder. Da bereits viele Antriebssysteme die Messung von Motorströmen unterstützen, wird bei diesem Ansatz keine zusätzliche oder externe Sensorik benötigt. Zur eigentlichen Bestimmung der Bodenbeschaffenheit werden die beiden Größen *Unebenheitsmaß* und *Festigkeitsmaß* im Folgenden definiert.

5.3.1 Unebenheitsmaß

Das *Unebenheitsmaß* stellt einen Indikator zur Evaluierung der Unebenheit des Geländes direkt unter dem Roboter dar. Zur Berechnung werden die Motorströme aller aktiv angetriebenen Räder eines schlangenartigen Roboters verwendet. Die Grundidee dabei ist, dass bei sehr schwierigem und unwegsamem Gelände, die Motorströme eines Rades sehr stark variieren, da sie auf Grund der veränderlichen Bodenhaftung zu verschiedenen Zeitpunkten unterschiedlichen viel Kraft aufbringen müssen. Diese Erkenntnis kann man sich zu Nutze machen und während der Bewegung des Roboters, über einen zeitlichen Verlauf der Motorströme, den Unebenheitswert R berechnen. Zur Berechnung wird die gleitende Standardabweichung (*MSTD*)[3] eingesetzt, die auch aus dem Finanzbereich zur Messung von Marktschwankungen bekannt ist. Die Berechnung der *MSTD* δ_i aus den Motorstromwerten erfolgt in zwei Schritten: Zunächst wird der gleitende Mittelwert ($M_{i,avg}$)[4], separat für jedes Antriebsmodul i, aus den Motorströmen X_j beider Antriebsräder über die gegeben Zeitspanne Δt, mit Gleichung 5.5, berechnet.

$$M_{i,avg} = \frac{X_{i,1} + \cdots + X_{i,n}}{n}, \text{mit } n = \text{Anzahl Motorstromwerte in } \Delta t \tag{5.5}$$

Danach wird der gleitende Mittelwert ($M_{i,avg}$) verwendet, um die *MSTD* δ_i mit Gleichung 5.6 für Antriebsmodul i einzeln zu berechnen.

$$\delta_i = \sqrt{\frac{(X_1 - M_{i,avg})^2 + (X_2 - M_{i,avg})^2 + \cdots + (X_n - M_{i,avg})^2}{n}} \tag{5.6}$$

Der resultierende Indikator für die Bodenunebenheit R wird durch ein arithmetisches Mittel aus den δ_i Werten aller Module kombiniert: $R = \frac{\delta_1 + \ldots + \delta_s}{s}$, mit $s =$ Anzahl Räder. Durch Variation der Zeitfenster Länge kann die pro Zeitschritt zu berücksichtigende Anzahl an Messwerten (Motorstromwerte) eingestellt werden. Eine kleinere Fensterlänge führt dabei zu einer sich schneller ändernden Unebenheit, ein längeres Fenster hingegen ergibt einen gefilterten Verlauf [89].

5.3.2 Festigkeitsmaß

Neben der Unebenheit, ist die Festigkeit des Terrain ein weiterer Indikator zur Beschreibung der Bodenbeschaffenheit. Ein radgetriebener Roboter sinkt bei weichen Materialien wie beispielsweise Sand sehr schnell ein und kann sich auf Grund festgefahrener Räder nicht mehr weiter fortbewegen. Das sogenannte *Festigkeitsmaß* S wurde definiert, um genau diese Eigenschaften des Bodens zu beschreiben. Eine naheliegende Möglichkeit zur Detektion, ob sich ein Roboter festgefahren hat, ist der Einsatz einer Kamera und der Bestimmung des optischen

[3] Vom englischen *Moving Standard Deviation* abgeleitet
[4] Vom englischen *Moving Average* abgeleitet

Flusses. Wenn sich die Pose des Roboters trotz Vortrieb der Räder nicht oder nur geringfügig ändert, kann daraus geschlossen werden, dass die Lokomotion auf dem Terrain versagt. Da diese Methoden einen zusätzlichen Sensor benötigen, oftmals störanfällig für sich bewegende Objekte in der Umgebung sind und die Detektion erst erfolgt, wenn der Roboter sich bereits festgefahren hat, wird hier ein Verfahren vorgestellt, dass ebenfalls die interne Sensorik des Roboter nutzt, um im Voraus die Festigkeit des Bodens zu prüfen.

Zur Bestimmung der Festigkeit wird ein spezielles Manöver des Roboters durchgeführt. Bei schlangenartigen Robotern mit Rädern oder Ketten besteht meistens die Möglichkeit ein oder mehrere Module mit Hilfe aktiver Gelenke anzuheben. Dies wird ausgenutzt um im stehenden Zustand einen Bodentest durchzuführen. Zunächst wird der Kopf des Roboters angehoben, so dass die Räder des ersten Moduls keinen Bodenkontakt mehr haben (siehe Abbildung 5.8 links). Die Antriebsräder ohne Bodenkontakt werden in Fahrtrichtung gedreht und der

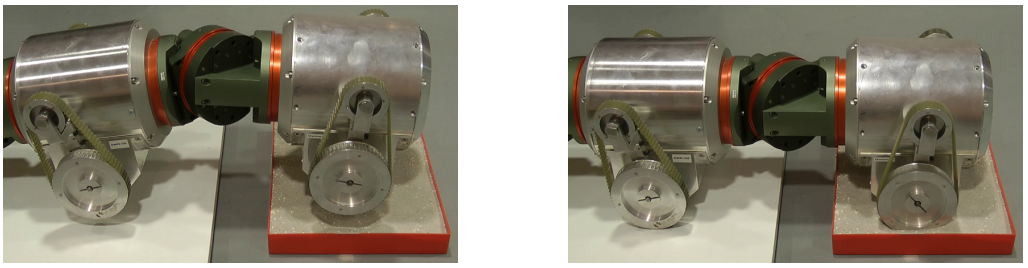


Abb. 5.8: Spezielles Manöver zur Bestimmung der Festigkeit des Bodens.

Kopf langsam abgesenkt. Währenddessen bleiben die Motorbremsen der restlichen Module angeschaltet und die Motorstromwerte der vorderen Räder werden aufgezeichnet. Treffen die Räder auf einen harten und griffigen Boden, so steigt der Motorstrom rapide an. Bei einem weichen Boden hingegen, zum Beispiel aus Sand, wie in Abbildung 5.8 rechts dargestellt, graben sich die Räder ein und der Motorstrom steigt nur sehr langsam an. Das Manöver wird beendet sobald der Motorstrom einen Schwellenwert überschreitet, oder der Roboter seinen Kopf eine maximale Tiefe in den Boden gedrückt hat. Zur Bestimmung des *Festigkeitsmaß* S wird zunächst nach der Methode der kleinsten Fehlerquadrate (Least Square Line Fitting) über alle während des Manövers gemessenen Motorströme, eine Regressionsgerade ermittelt. Die Steigung σ dieser Gerade wird als *Festigkeitsmaß* S definiert [89].

Durch eine Kombination der beiden Indikatoren *Unebenheitsmaß* und *Festigkeitsmaß* lässt sich, lediglich durch die Messung der Motorströme, bereits eine Aussage über die Beschaffenheit des Geländes vor bzw. unter dem Roboter ableiten. Eine detaillierte Auswertung der Messergebnisse findet sich in Kapitel 7.3.2 und Kapitel 7.3.3. Bei der Bestimmung dieser Parameter geht es lediglich darum zu bestimmen, wie zuverlässig sich die radgetriebene Fortbewegung auf dem aktuellen Terrain eignet, um entsprechend eine alternative Fortbewegungsart zu verwenden.

5.4 Selektion und Adaption der Fortbewegung

Mit Hilfe der erfassten Werte für das *Unebenheitsmaß* R und das *Festigkeitsmaß* S ist es nun möglich eine entsprechende Fortbewegungsart für den Roboter auszuwählen bzw. die Fortbewegung zu wechseln. Hierzu wird zunächst eine Bewertung der verfügbaren Fortbewegungsarten bezüglich verschiedener Bodenbeschaffenheiten benötigt. Die für KAIRO 3 implementierten Bewegungsmöglichkeiten *Radgetrieben*, *Omega Lokomotion* und *Raupenartige Lokomotion* [87], wurden bezüglich verschiedener Eigenschaften evaluiert und bewertet [89]. Eine zusammenfassende Auswertung ist in Tabelle 5.1 dargestellt.

Fortbewegungsart	**Räder**	**Omega**	**Raupe**
Geschwindigkeit	+	o	-
Unstrukturiertes Gelände	-	o	+
Weicher Untergrund	-	o	+
Stabilität	+	-	o
keine Räder	-	o	+

Tabelle 5.1: Vergleich der Fortbewegungsarten von KAIRO 3 bezüglich unterschiedlicher Bodenbeschaffenheiten und Eigenschaften.

Auf Grund der meist höheren Geschwindigkeit auf ebenem Gelände, ist die radgetriebene Fortbewegung die bevorzugte Bewegungsart. Die *Omega* Lokomotion hingegen eignet ich besonders für unstrukturiertes Gelände mit festem Untergrund wie beispielsweise Grasflächen oder unbefestigte Wege. Für extrem unwegsames Gelände oder nachgiebige Oberflächen, wie zum Beispiel sandige Bereiche oder Waldgebiete, eignet sich die raupenartige Fortbewegung am Besten, da hierfür keine Räder zur Fortbewegung benötigt werden.

Zur automatischen Auswahl der am Besten geeigneten Fortbewegungsart, wird vor dem Start und nach einer parametrierbaren zurückgelegten Entfernung, das Manöver zur Prüfung der Bodenfestigkeit durchgeführt. Während der radgetriebenen Fortbewegung kann kontinuierlich die Unebenheit bestimmt werden. Basierend auf den aktuellen Werten des *Unebenheitsmaß* R und des *Festigkeitsmaß* S kann eine Auswahl der Fortbewegung getroffen werden. Diese Selektion lässt sich im einfachsten Falle durch entsprechende Schwellenwerte realisieren. Tabelle 5.2 zeigt die für KAIRO 3 implementierten Schwellenwerte [89].

Fortbewegungsart	**Räder**	**Omega**	**Raupe**
Unebenheitsmaß R	< 15.0	≥ 15.0	≥ 15.0
Festigkeitsmaß S	≥ 0.5	≥ 0.5	< 0.5

Tabelle 5.2: Implementierte Schwellenwerte zur Selektion der Fortbewegungsart.

5.4.1 Umsetzung mit KAIRO 3

Für die Umsetzung der Berechnung der Bodenbeschaffenheit, wurde die Steuerungssoftware von KAIRO 3 um zwei neue Softwaremodule *TerrainRoughness* und *TerrainStrength* erweitert. Beide Klassen befinden sich bezogen auf die hierarchische Struktur der *Adaptive Control* innerhalb der sogenannten *SensorGroup*. Als Eingaben erhalten die beiden Softwaremodule jeweils die Motorstromwerte der Antriebsräder. Die Klasse *TerrainRoughness* berechnet kontinuierlich das Unebenheitsmaß R, nach der oben vorgestellten Gleichung 5.6. Im Gegensatz dazu, ist die *TerrainStrength* Klasse lediglich aktiv, wenn das Manöver zur Bestimmung des Festigkeitsmaß gestartet wurde. Dieses Manöver wurde in Form eines neuen Zustandes *StateGroundTest* in die Manöver Steuerung (*ManeuverControl*) von KAIRO 3 hinzugefügt. Dieses Manöver führt die oben beschriebene Sequenz zur Messung der Bodenfestigkeit aus. Gleichzeitig zeichnet das *TerrainStrength* Modul die entsprechenden Motorströme auf. Ist die Ausführung des Manövers beendet, wird das Festigkeitsmaß S aus den aufgezeichneten Motorströmen ermittelt. Der errechnete Wert bleibt bis zur nächsten Messung gültig.

Die automatische Selektion der Fortbewegung, wurde durch Erweiterung der Steuerungssoftware um das MCA2 Modul *MotionChooser* realisiert. Abbildung 5.9 zeigt die hierarchische Anordnung des *MotionChooser* über der *ManeuverControl*. Als Eingabe erhält der *MotionChooser* jeweils die aktuellen Werte des Unebenheitsmaß R und Festigkeitsmaß S. Basierend auf diesen Werten, des aktuellen Roboterzustands und der in Tabelle 5.2 gezeigten Schwellenwerten, weist der *MotionChooser* die *ManeuverControl* an, die aktuelle Fortbewegung zu wechseln [89].

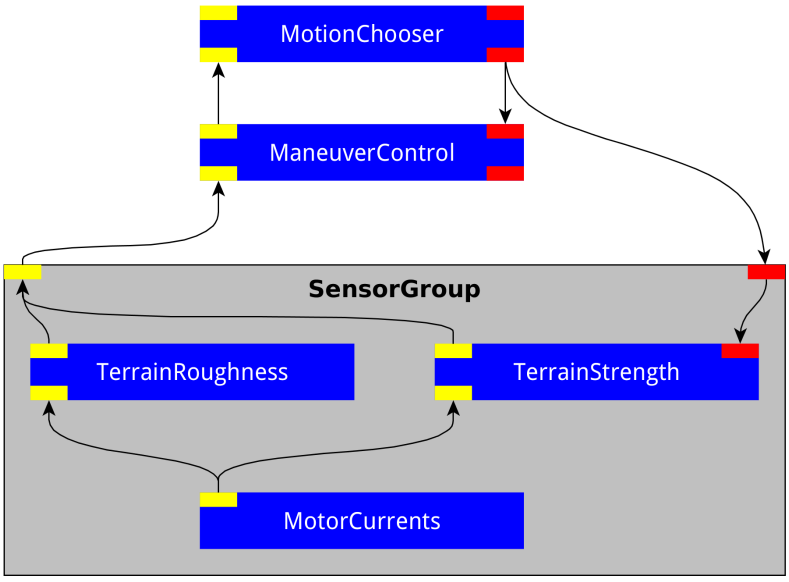


Abb. 5.9: Software Struktur des implementierten Teilsystems zur Adaption der Fortbewegung von KAIRO 3.

Besitzt der Roboter zusätzlich integrierte optische Sensorik, kann diese genutzt werden, um die Qualität der Ergebnisse zu verbessern. Insbesondere die Erkennung von Hindernissen und größeren Vertiefungen, kann dazu beitragen, die Auswahl der Besten Fortbewegungsart zu unterstützen.

5.5 Zusammenfassung

In diesem Kapitel zur konfigurationsabhängigen und umweltbasierten Adaption der Lokomotion wurden die zwei biologischen Fortbewegungsarten *Omega-Lokomotion* und *Raupenartige Lokomotion* analysiert und deren Umsetzung für modulare schlangenartige Roboter am Beispiel von KAIRO 3 aufgezeigt. Insbesondere für die raupenartige Lokomotion, wurde eine geometrische und numerische Möglichkeit zur Berechnung einer notwendigen Ausgleichsbewegung benachbarter Module beschrieben. Weiterhin wurde zur Bestimmung der Bodenbeschaffenheit mit Hilfe interner Sensorik, das *Unebenheitsmaß* R und das *Festigkeitsmaß* S eingeführt. An Hand dieser beiden Messgrößen, wurde die Selektion und Adaption der Fortbewegungsart diskutiert und eine beispielhafte Umsetzung durch Erweiterung der adaptiven Steuerung von KAIRO 3 erläutert. Die Evaluationsergebnisse hierzu finden sich in Kapitel 7.3.

6 Konfigurationsabhängige Navigations- und Bewegungsplanung

Ein weiterer Baustein zur Erhöhung der Autonomie und Flexibilität rekonfigurierbarer Serviceroboter ist die intelligente Navigations- und Bewegungsplanung in unstrukturiertem Gelände mit Hindernissen. Dies umfasst die Handhabung und Steuerung von RMRS mit vielen Freiheitsgraden (DOF) und veränderlicher Konfigurationen. Im Detail bedeutet dies, dass komplexe Roboterbewegungen in der euklidischen Gruppe SE 3 (Position und Orientierung im 3-D Umgebungsraum) geplant werden können, die es einem RMRS in Abhängigkeit seiner aktuellen Konfiguration erlaubt, Hindernisse autonom zu bewältigen. Insbesondere bei rekonfigurierbaren Robotern kann die Variation der Roboterkonfiguration einen großen Einfluss auf die Navigations- und Bewegungsplanung ausüben. An Hand des folgenden Szenario wird die Tragweite dessen verdeutlicht. Der in Abbildung 6.1 (links) dargestellte Roboter, bestehend aus fünf Modulen, ist nicht in der Lage das vor ihm befindliche Hindernis zu überwinden. In diesem Fall muss ein Weg gesucht werden, um das Hindernis zu umgehen. Bei einer Konfiguration mit zusätzlichen Modulen, wie in Abbildung 6.1 (rechts) dargestellt, kann der Roboter über das Hindernis klettern. Daher wurden in dieser Arbeit bereits bei der Bahnplanung unterschiedliche Bewegungsmöglichkeiten in Abhängigkeit von der aktuellen Roboterkonfiguration berücksichtigt.

Abb. 6.1: Einfluss der Roboterkonfiguration auf die Navigations- und Bewegungsplanung.

Basierend auf dem PlexNav [90] Planungsframework und der adaptiven Steuerung [44, 54] von KAIRO 3, wurde eine integrierte Steuerungsmethode zur Navigations- und Bewegungsplanung sowie der Bewegungsausführung für schlangenartige RMRS entwickelt. Dieser Ansatz ermöglicht es, komplexe Trajektorien entsprechend der Roboterkinematik und -konfiguration zu generieren und auszuführen. Abbildung 6.2 zeigt einen Überblick über die vorgeschlagene

Systemarchitektur, die das Planungsproblem in mehrere Teilschritte zerlegt und somit auch das individuelle Neuplanen ermöglicht [91].

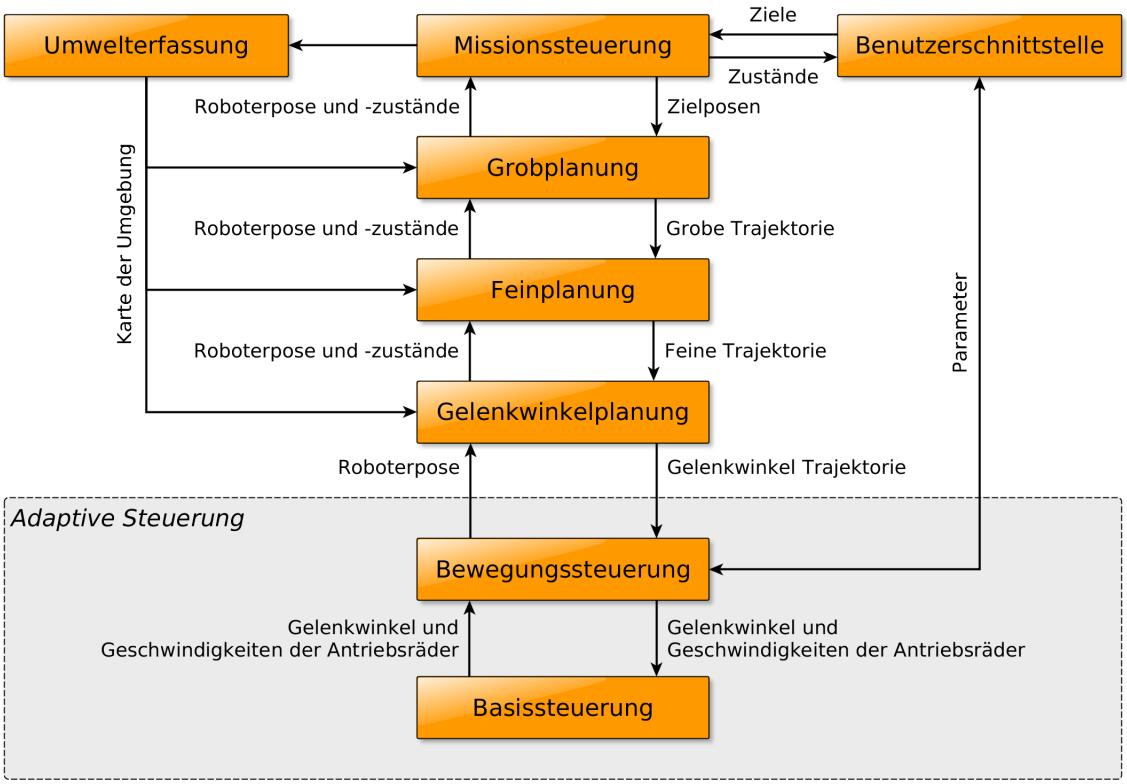


Abb. 6.2: Systemarchitektur der integrierten Navigations- und Bewegungsplanung für rekonfigurierbare schlangenartige Roboter [91].

6.1 Missionssteuerung und Benutzerschnittstelle

Die in Abbildung 6.2 zentral dargestellte *Missionssteuerung* übernimmt die übergeordnete Koordination des Planungssystems. Insbesondere wird der Ablauf einer Mission gesteuert und überwacht, um die durch die *Benutzerschnittstelle* vorgegebenen Missionsziele und -parameter zu erfüllen. Als Eingabe für das Planungssystem dient eine Liste von Navigationszielen, die jeweils als 6-D Zielposen $(x, y, z, \alpha, \beta, \gamma)$ dargestellt sind. Zusätzlich steuert die *Missionssteuerung* die Erfassung und Modellierung der Umwelt sowie die Erstellung einer Umgebungskarte. Zur Verbesserung der Qualität und Abdeckung der Umwelterfassung, auch bei variabler Anordnung der Sensoren, wurde die *Missionssteuerung* um sogenannte *Kartierungsmanöver* erweitert. Diese speziell an die Roboterkinematik anpassbaren Bewegungsabläufe, lassen sich durch einen Plugin-Mechanismus in die *Missionssteuerung* integrieren und können als spezielle Planungsziele ausgeführt werden. Als Auslöser für ein solches *Kartierungsmanöver* können sowohl externe Ereignisse wie die Erkennung markanter Umgebungsmerkmale als auch zeitlich oder räumlich wiederkehrende Impulse dienen.

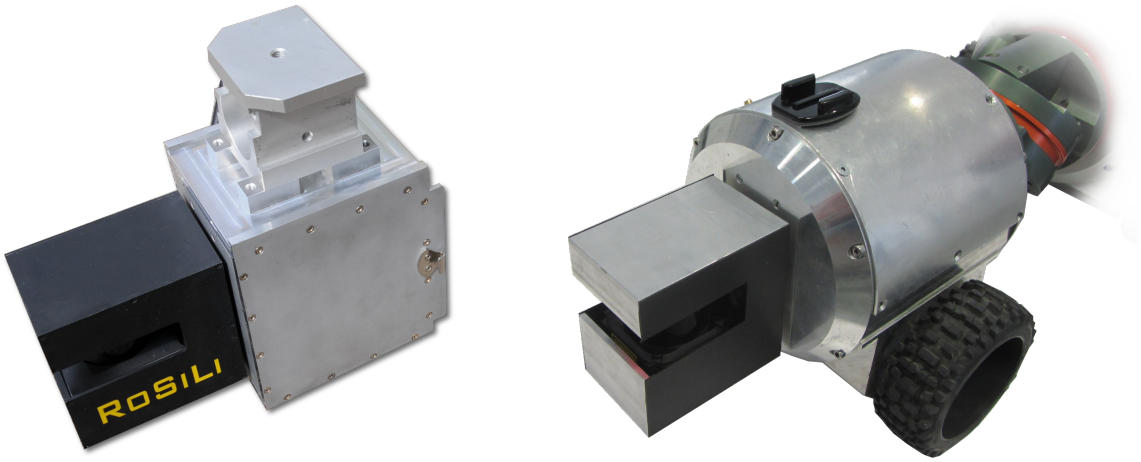
6.1.1 Kartierungsmanöver

Insbesondere schlangenartige Roboter, wie KAIRO 3, besitzen meist eine geringe Höhe, so dass Umweltsensoren lediglich bodennah angebracht werden können. Bei vielen Sensoren führt die niedrige Position oft zur Verdeckung von Objekten. Ein spezielles *Kartierungsmanöver* für schlangenartige Roboter, hebt den Kopf des Roboters inklusive des Sensors vertikal an und ermöglicht somit einen besseren Überblick über die Umgebung. Hierbei wird insbesondere die Reichweite der Sensoren und die Sichtbarkeit von Objekten verbessert. Zudem können befahrbare Bodenflächen präziser erkannt werden, da mit optischen Messprinzipien bei sehr spitzen Winkeln auf Oberflächen oft Messfehler und Ausreißerpunkte entstehen [91].

6.2 Umwelterfassung und Lokalisierung

Die Wahrnehmung und Kartierung der Umwelt sowie die *Lokalisierung* eines Roboters im erstellten Umweltmodel ist in Abbildung 6.2 durch den Block *Umwelterfassung* dargestellt. Zur Generierung und Speicherung des Umweltmodells wird die im PlexNav Framework integrierte Kartenrepräsentation PlexMap [90] eingesetzt. Diese generalisierte 2.5-D Karte ermöglicht es, 6-D Punktewolken im sehr verbreiteten PCL (Point Cloud Library) Datenformat zu verarbeiten. Somit kann die PlexMap Daten von einer Vielzahl unterschiedlicher 3-D Sensoren in einer einheitlichen Kartenrepräsentation integrieren. Speziell für den Einsatz auf schlangenartigen Robotern wurde das rotierende 3-D Laserscanner System KaRoLa [92] entwickelt, dass sich durch seine leichte und kompakte Bauform auszeichnet. KaRoLa lässt sich, wie in Abbildung 6.3b zu sehen, in bestehende Robotersysteme integrieren oder als eigenständiges Sensorsystem verwenden (Abbildung 6.3a). In Abbildung 6.4 wird beispielhaft eine mit KaRoLa aufgezeichnete Punktewolke dargestellt.

Eine 3-D Punktewolke wird durch Diskretisierung in Zellen und Speicherung der Höhenwerte in die 2.5-D *PlexMap* Kartenrepräsentation transferiert, wobei die ursprüngliche Punktewolke ebenfalls gespeichert wird. Die Zellgrößen werden dabei dynamisch an die Anzahl und Streuung der vorhandenen Messpunkte angepasst. Für jede Zelle wird der mittlere Höhenwert aus allen Messpunkten innerhalb der Zelle gespeichert. Mit Hilfe einer Hauptkomponentenanalyse und einer Ebenenschätzung der Messpunkte innerhalb einer Zelle, werden die Unebenheit und Steigung jeder Zelle bestimmt. Das PlexNav Framework integriert ebenfalls die Registrierung mehrerer nacheinander und an unterschiedlichen Posen aufgenommenen Punktewolken. Das Registrieren einer neuen Punktewolke gegen die bereits vorhandene Karte erfolgt durch einen modifizierten ICP basierten Algorithmus, wobei die Roboter Odometrie als grobe Referenz eingesetzt wird. Zusätzlich zur Registrierung einzelner Punktewolken wird ein Loop-Closing auf der gesamten Karte durchgeführt, um Unsicherheiten der Roboterlokalisierung zu korrigieren. Die *Umwelterfassung* bietet somit ein integriertes Kartierungs- und



(a) Autarkes Sensorsystem RoSiLi. (b) KaRoLa integriert in KAIRO 3.

Abb. 6.3: Kompakter rotierender Laserscanner zur Aufnahme hochauflösender und präziser 3-D Punktewolken. Durch den großen Öffnungswinkel von 270° und einer Reichweite von bis zu 30 m können Punktewolken mit mehr als einer Million Messpunkte aufgenommen werden.

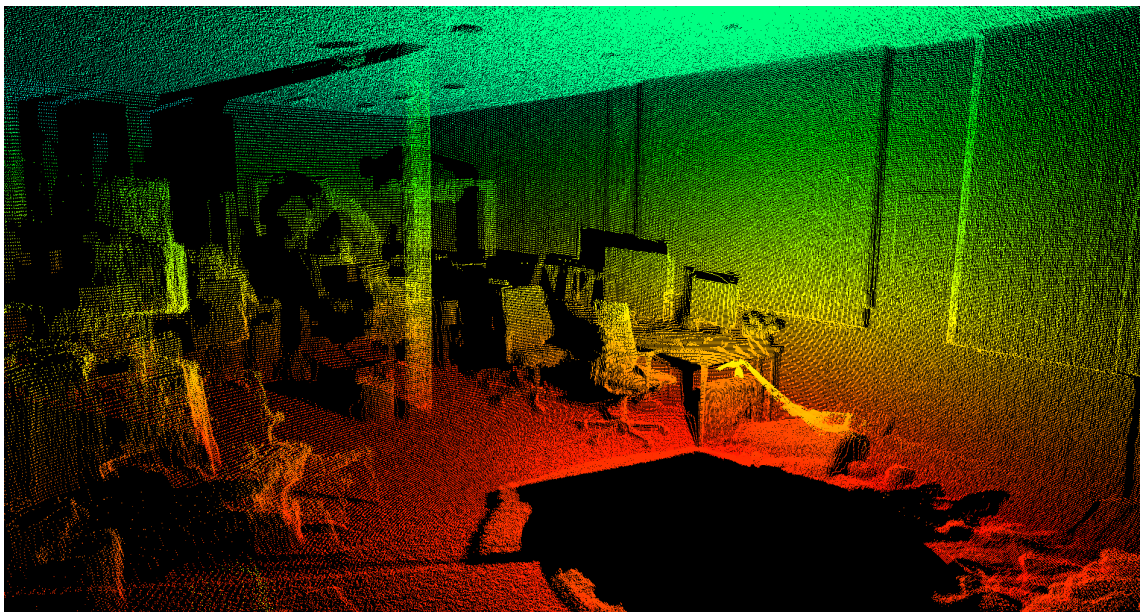


Abb. 6.4: Ausschnitt einer 3-D Punktewolke, die das Roboterlabor zeigt. Der Farbverlauf von rot nach grün kodiert die Höhe entlang der z-Achse.

Lokalisierungssystem, dass von der im Folgenden vorgestellten *Trajektorienplanung* verwendet wird [90, 93].

6.3 Trajektorienplanung

Methoden zur Pfad- und Bewegungsplanung erzeugen im Allgemeinen verschiedene Arten von Trajektorien, die von der Steuerung eines Robotersystems ausgeführt werden können. Insbesondere für schlangenartige Robotersysteme mit vielen Freiheitsgraden[1], erfordern schnelle Planungsverfahren wie Rapidly-Exploring Random Trees (RRT) [31] oder Randomized Kinodynamic Planning [94] um effizient mit dem hoch dimensionalen Planungsraum zurecht zu kommen. Diese Ansätze benutzen oft vollständige mathematische Modelle inklusive Nebenbedingungen der Roboterkinematik um alle Gelenke und Posen bei der Planung der Bewegung zu berücksichtigen. Diese vollständigen Modelle führen oft zu einer hohen Rechenlast insbesondere bei der Berechnung der inversen Kinematik. Betrachtet man rekonfigurierbare Roboter mit einer veränderlichen Modulanzahl und Kinematik, wächst die Dimension des Zustandsraumes für die Planung mit steigender Anzahl an Modulen und Gelenken rapide an.

Auf Grund dieser Nachteile bei der Verwendung eines vollständigen mathematischen Modells für die *Trajektorienplanung*, wird ein Verfahren vorgestellt, dass auf einem reduzierten Planungsraum arbeitet und die Komplexität der Planung in mehrere Schritte aufteilt. Wie in Abbildung 6.2 dargestellt, ist das Planungsverfahren in die drei Teilbereiche *Grobplanung*, *Feinplanung* und *Gelenkwinkelplanung* aufgeteilt. Der Planungsraum ist als konstant angenommen, insbesondere bei einer veränderlichen Anzahl an Robotermodulen und somit auch eine veränderliche Anzahl an Freiheitsgraden, da die Planung lediglich 6-D Posen für das erste Robotermodul berücksichtigt. Diese Vereinfachungen sind bei schlangenartigen Robotern durch die Kombination der Planung mit einem *follow-the-leader* Ansatz zur Bewegungssteuerung möglich. Im Zusammenhang mit dem Robotersystem KAIRO 3, wird diese Art der Bewegungssteuerung auch als *Virtuelle Schiene* bezeichnet und ist in Kapitel 2.5.2 beschrieben. Auf diese Weise ist das integrierte Navigations- und Bewegungsplanungssystem unabhängig von einer veränderlichen Anzahl an Modulen und gleichzeitig in der Lage verschiedene Konfigurationen von schlangenartigen Robotern zu berücksichtigen.

6.3.1 Grobplanung und Feinplanung

Die *Grobplanung* bestimmt einen allgemeinen Pfads von einer gegebenen Startpose zu einer Zielpose bestehend aus 6-D Posen des Roboterursprungs[2]. Bei der

[1] KAIRO 3 hat 27 Freiheitsgrade in den Gelenken und Antrieben plus 6 räumliche Freiheitsgrade zur Positionierung und Orientierung der Roboterbasis

[2] Am Beispiel von KAIRO 3 befindet sich der Roboterursprung im Zentrum des ersten Antriebsmoduls

Grobplanung sind Posen außerhalb des Sensorhorizonts des Roboter erlaubt. Diese Art der Planung wird deshalb auch als *over-the-horizon* Planung bezeichnet.

Im zweiten Schritt, der sogenannten *Feinplanung*, werden jeweils Teilbereiche des groben Pfades verfeinert, die sich innerhalb bereits bekannter Bereiche der Karte befinden. Der resultierende Pfad besteht ebenfalls aus 6-D Posen und berücksichtigt bereits die Fähigkeiten des Roboters, Hindernisse zu überwinden. Diese Fähigkeiten basieren auf den gegebenen Umweltinformationen und der aktuellen Roboterkonfiguration. Der Pfad beinhaltet jedoch keine Informationen darüber, wie der Roboter dem Pfad folgen bzw. Hindernisse überwinden kann.

Beide Planungsschritte, *Grobplanung* und *Feinplanung* wurden mit Hilfe einer erweiterten Variante des RRT* Algorithmus [95] implementiert. Die verwendete Kostenfunktion, wurde speziell für modulare schlangenartige Roboter optimiert und wird in Kapitel 6.5.1 erläutert. Diese Kostenfunktion passt sich an die Anzahl der verwendeten Robotermodule bzw. Roboterkonfiguration an und beeinflusst somit den geplanten Pfad. Beispielsweise wird ein kurzer schlangenartiger Roboter hohe Kosten für große Hindernisse erhalten. Ein langer Roboter hingegen erhält geringere Kosten, wenn er kinematisch in der Lage ist das Hindernis zu überwinden. Weitere Verbesserungen der Planungsergebnisse, insbesondere für nicht holonome Bewegungen, werden durch die in Kapitel 6.5.2 dargestellte *Sekundärer Nächster Nachbar Raum* Erweiterung des RRT* erreicht. Die Hauptunterschiede zwischen *Grobplanung* und *Feinplanung* sind die modifizierten Parameter der Kostenfunktion und eine feinere Schrittweite [91].

6.3.2 Gelenkwinkelplanung

Im dritten Planungsschritt, der *Gelenkwinkelplanung*, wird ein Teilbereich des verfeinerten Pfades verwendet, um einen geglätteten und an das Gelände beziehungsweise an die Oberfläche von Hindernissen angepassten Pfad entsprechend der Robotergeometrie und -kinematik zu erzeugen. Da dieser Schritt sehr stark von den eingesetzten Robotermodulen abhängt, muss dies für unterschiedliche Arten von Robotern angepasst werden. In dieser Arbeit wird ein modularer schlangenartiger Roboter mit Radantriebs- und Gelenkmodulen, wie KAIRO 3, betrachtet. Die *Gelenkwinkelplanung* wird wiederum in sechs aufeinanderfolgende Teilschritte untergliedert, die in Abbildung 6.5 dargestellt sind.

Initialisierung

Zunächst wird bei der *Initialisierung* der aktuelle Roboterzustand (die Pose des ersten Antriebsmoduls) aus der virtuellen Schiene abgeleitet. Mit Hilfe dieser Pose wird eine Plausibilitätsprüfung des nächsten Pfadsegments durchgeführt, welches von der *Feinplanung* erzeugt wurde. Dazu werden die Gierwinkeländerungen $\Delta\gamma$ und die absoluten Nickwinkeländerungen $\Delta\beta$ zwischen zwei jeweils aufeinander folgenden Posen getestet, wobei die aktuelle Roboterpose als Startpose

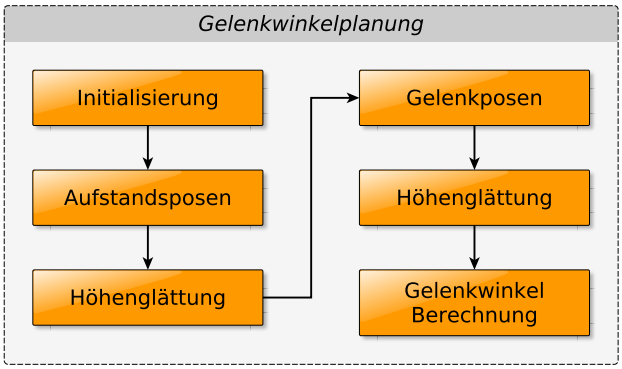


Abb. 6.5: Ablauf der Gelenkwinkelplanung [91].

dient. Überschreitet einer der beiden Winkeländerungen seinen Schwellenwert, so werden die entsprechenden Posen auf dem Pfad entfernt. Im Extremfall, kann dies dazu führen, dass alle Punkte entfernt werden, wodurch automatische eine Neuplanung der *Feinplanung* ausgelöst wird. Des Weiteren wird bei der *Initialisierung* ein Kartenausschnitt aus der *PlexMap* geladen, der den entsprechenden Pfadabschnitt abdeckt. Dieser Kartenausschnitt wird bei den weiteren Schritten der *Gelenkwinkelplanung* benötigt [91].

Aufstandsposen

Ausgehend von der aktuellen Roboterpose, werden mit einer festen Schrittweite neue Zustände in Richtung der nächsten Zwischenpose erzeugt, bis die Zielpose des Pfadabschnittes erreicht ist. Für diese Zustände werden sogenannte *Aufstandsposen* basierend auf dem *PlexMap* Kartenausschnitt bestimmt, um den Roboter auf den Boden zu projizieren. Dazu werden bei Robotern mit Radantrieb die Bodenkontaktpunkte der Räder verwendet. Mit Hilfe der x, y Koordinaten und des Gierwinkels γ lassen sich für jede Pose des aktuellen Pfadabschnittes die Radaufstandspunkte aus der Karte ableiten. Die Aufstandspunkte und die 2-D Pose werden verwendet um den Rollwinkel und die Höhe des Antriebsmoduls zu bestimmen. Der Rollwinkel α ergibt sich aus der Höhendifferenz des rechten und linken Radaufstandspunktes (h_r und h_l) und dem Abstand zwischen den beiden Rädern d:

$$\alpha = arcsin\left(\frac{h_r - h_l}{d}\right) \tag{6.1}$$

Mit Hilfe des Rollwinkels α, kann die resultierende Höhe z für den Mittelpunkt des Antriebsmoduls aus den Radaufstandspunkten (h_r und h_l) und der um α rotierten Distanz c abgeleitet werden. Dies entspricht einer Projektion der Höhe c auf die Bodenfläche, wobei c den orthogonalen Abstand zwischen den Radachsen und dem Mittelpunkt des Antriebsmoduls beschreibt:

$$z = \frac{h_r + h_l}{2} + cos\,(\alpha) \cdot c \tag{6.2}$$

Bei Robotersystemen ohne Räder können hier auch ausgewählte Kontaktpunkte, beispielsweise Auflagepunkte auf dem Boden oder Fußpunkte bei Laufrobotern, verwendet werden [91].

Höhenglättung

Zum Ausgleich von Modellungenauigkeiten der Karte und um das sichere Übersteigen von Hindernissen zu ermöglichen, werden die Höhenwerte benachbarter Posen geglättet. Diese Glättung umfasst drei Filterschritte und eine Interpolation [96]:

1. Akausaler Medianfilter: Entfernt grobe Ausreißer unter besonderer Behandlung der Randbereiche des Pfades.
2. Schwellenwertfilter: Unterdrückt Schwankungen die durch Kartenungenauigkeiten auf Grund von Sensorrauschen auftreten können.
3. Lineare Interpolation der Höhen: Erzeugt Geradenstücke an Stellen vor und nach scharfen Kanten (beispielsweise an Stufen oder Hindernissen).
4. Akausaler Mittelwertfilter: Erzeugt fließende Übergänge zwischen interpolierten Geradenstücken und geglätteten Bereichen.

Gelenkposen

Die nach der *Höhenglättung* erhaltene Trajektorie kann nicht direkt von der virtuellen Schiene in der *Basissteuerung* ausgeführt werden, da die virtuelle Schiene durch die Mittelpunkte der Gelenkmodule (Knickstelle) verläuft, die aktuelle Trajektorie jedoch *Aufstandsposen* bezogen auf den Mittelpunkt des ersten Antriebsmoduls beschreibt. Deshalb werden die geplanten *Aufstandsposen* in diesem Schritt in *Gelenkposen* transformiert. Dazu müssen zunächst die Nickwinkel β und Gierwinkel γ der einzelnen *Aufstandsposen* aus dem Richtungsvektor zwischen der vorherigen Knickstelle und der Pose selbst berechnet werden. Durch Projektion der Aufstandspose mit dem entsprechenden Richtungsvektor eine halbe Modullänge nach vorne, erhält man die zur Aufstandspose zugehörige Gelenkpose. In einigen Fällen wird die Gelenkpose durch die Transformation in den Boden oder zu nahe an ein Hindernis verschoben. Dies wird durch eine erneute Höhenanpassung basierend auf den Kartendaten korrigiert. Zudem können bei großen Änderungen des Nickwinkels β oder Gierwinkels γ *Gelenkposen* entgegen der Bewegungsrichtung des Roboters verschoben werden. Wird bei einer Überprüfung der Winkeldifferenzen von zwei aufeinander folgenden Posen ein Übertritt der maximal zulässigen Winkeländerungen (Abhängig von der eingestellten

Schrittweite) festgestellt, so wird die betroffene Pose verworfen. Die resultierende *Gelenkposen* Trajektorie wird nochmals durch eine *Höhenglättung* verfeinert, bevor die abschließende Erzeugung der *Gelenkwinkel* statt findet [96].

Gelenkwinkel

Nach der Glättung und Anpassung des 6-D Pfades an das Gelände, werden zusätzlich die *Gelenkwinkel* α, β und γ für jeden Zwischenpunkt des Pfades mit Hilfe der inversen Kinematik des Roboters berechnet und gespeichert. An Stellen, die eine Veränderung der Sollgelenkwinkel erfordert, müssen zusätzliche *Gelenkposen* hinzugefügt werden. Dies tritt hauptsächlich in Teilbereichen des Pfades auf, die sich in der Nähe von Hindernissen befinden. Die Vorgabe von Gelenkwinkeln für das jeweilige Gelenkmodul durch die Trajektorie selbst, ermöglicht unter anderem eine Prädiktion der Gelenkbewegung. Somit entsteht z. B. vor Steigungen eine vorausschauende Rotation der Gelenke[3], was die Ausführungszeit zum Anheben der Module verkürzt. Zusätzlich ermöglicht die explizite Angabe der Sollgelenkwinkel eine Synchronisierung der einzelnen Gelenkmodule mit ihren Nachbarmodulen bei der Ausführung des Pfades.

Die erzeugte Trajektorie aus 6-D Posen, erweitert um *Gelenkwinkel*, beschreibt den räumlichen Verlauf des Knickpunktes und die Rotation der α, β und γ Achsen eines Gelenkmoduls. Basierend auf dieser Trajektorie wird die virtuelle Schiene in der *Bewegungssteuerung* aufgebaut und ausgeführt [96].

6.4 Bewegungssteuerung und Basissteuerung

Die in der *Gelenkwinkelplanung* erzeugten Bewegungstrajektorien, bestehend aus 6-D Posen und den Sollgelenkwinkeln, werden an die *Bewegungssteuerung* weitergegeben. Die Sollgelenkwinkel α, β und γ jeder Pose sind für die virtuelle Knickstelle vor dem ersten Robotermodul definiert. Die *Bewegungssteuerung* bildet diese Trajektorien auf die im Folgenden vorgestellte erweiterte virtuelle Schiene ab und interpoliert die Gelenkwinkel zwischen den einzelnen Posen, während der Roboter entlang der Schiene bewegt wird [96].

6.4.1 Erweiterung der virtuellen Schiene

Die ursprüngliche Implementierung der virtuellen Schiene in der adaptiven Steuerung arbeitet lediglich mit 3-D Posen, wie in Kapitel 2.5.2 erläutert [54]. Zur Realisierung von vertikalen Bewegungen, die es ermöglichen in z Richtung Hindernisse und Stufen zu überwinden, werden 6-D Posen für jeden

[3] Zum vertikalen Anheben eines KAIRO 3 Moduls müssen die *Alpha* und *Gamma* Gelenke um jeweils 90° rotiert werden (siehe Kapitel 2.5.1).

Wegpunkt gespeichert. Darüber hinaus wurde die erweiterte virtuelle Schiene eingeführt, die zusätzlich die Sollgelenkwinkel α, β und γ für jede Pose vorhält. Abbildung 6.6 skizziert beispielhaft einen Roboter bestehend aus drei Antriebsmodulen und zwei Gelenkmodulen, der sich entlang einer erweiterten virtuellen Schiene bewegt. 6-D Posen mit zusätzlichen Gelenkwinkeln sind als gelbe Kno-

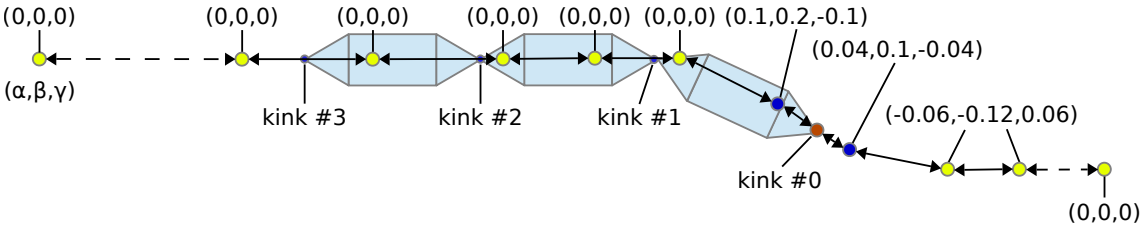


Abb. 6.6: Abstrahierte Skizze einer erweiterten virtuellen Schiene.

ten dargestellt. Knickstellen, einschließlich der virtuellen Knickstellen vor und hinter dem Roboter, sind mit kink #0 bis kink #3 nummeriert, wobei kink #0 rot markiert ist. Die aktuelle Pose und Gelenkwinkel für die virtuelle Knickstelle vor dem Roboter (kink #0) errechnet sich durch Interpolation zwischen den beiden Posen der virtuellen Schiene vor und hinter der Knickstelle (blau markierte Knoten). Gleichung 6.3 beschreibt die Interpolation der Sollgelenkwinkel $\vec{\psi}$ in Abhängigkeit der benachbarten Posen $\vec{p}$ und Winkel, wobei die Indizes $prev$ und $next$ die jeweiligen Vorgänger und Nachfolgenden Knoten auf der virtuellen Schiene und $curr$ die aktuelle Pose bezeichnen [96].

$$\vec{\psi} = \vec{\psi}_{prev} + \frac{|\vec{p}_{curr} - \vec{p}_{prev}|}{|\vec{p}_{next} - \vec{p}_{prev}|} \cdot \left(\vec{\psi}_{next} - \vec{\psi}_{prev}\right) \tag{6.3}$$

6.4.2 Regelung der Fahrgeschwindigkeiten

Zur automatischen Regulierung der Geschwindigkeit wurde ein alternierender Geschwindigkeitsregler entwickelt, der aus zwei Proportionalreglern besteht. Ein Regler dient zum Beschleunigen wobei der zweite Regler die Geschwindigkeit entsprechend den Gelenkrotationen abbremst. Damit ist es möglich die aktuelle Fahrgeschwindigkeit jederzeit und automatisch an die Geschwindigkeit der Gelenke anzupassen. Vor allem bei großen Änderungen des Gelenkwinkels während einer kurzen Fahrstrecke wird der Vortrieb verlangsamt. Die einzelnen Radgeschwindigkeiten, insbesondere für die linken und rechten Räder, werden getrennt durch eine spezielle Differentialantriebsfunktion berechnet, die von den benachbarten Modulen abhängt. Damit ist es möglich das Durchrutschen der Räder bei engen Kurven deutlich zu verringern. Die aus der Bewegungssteuerung erzeugten Geschwindigkeiten für die einzelnen Räder und Positionen für die Gelenke, werden von der verteilten Basissteuerung auf dem Roboter ausgeführt [96].

6.5 Erweiterter RRT* Algorithmus

Innerhalb der einzelnen Planungsschritte (*Grob-* und *Feinplanung*) der Navigations- und Bewegungsplanung wird eine erweiterte Variante des RRT* Algorithmus eingesetzt. Karaman et al. [34] beschreiben die grundlegende Funktionsweise des Standard RRT* Algorithmus und den Einsatz einer Kostenfunktion, die anwendungsbezogene Parameter zur Optimierung verwendet. Neben den (weichen) Optimierungskriterien existieren (harte) Zwangsbesinnungen die verhindern, dass der Planungsalgorithmus unzulässige Bewegungen erzeugt, die zum Beispiel mit einem Hindernis kollidieren oder die mögliche Gelenkwinkeleinschränkungen des Roboters verletzten. In den folgenden Abschnitten werden die Anpassungen der Kostenfunktion und die *Sekundärer Nächster Nachbar Raum* Erweiterung des RRT* für die Verbesserung der Planungsergebnisse mit nicht-holonomen Roboter erläutert.

6.5.1 Kostenfunktion für nicht-holonome Roboter

Auf Grund der nicht-holonomen Eigenschaften eines radgetriebenen schlangenartigen Roboters, wurde die in PlexNav [90] vorgeschlagene Kostenfunktion angepasst und zusätzliche Randbedingungen eingeführt. Die Kosten um den Zustand s_2 aus dem Zustand s_1 zu erreichen, können mit Hilfe der Gleichung 6.4, bestimmt werden.

$$\begin{aligned} cost_{s_1,s_2} = \text{DIST}(s_1, s_2)\frac{1}{N}\sum_{q=0}^{N} C(q) + \epsilon\frac{(\text{YAW}(s_1, s_2)+1)^2 - 1}{\text{DIST}(s_1, s_2)\cdot\pi} \\ + \zeta \cdot \text{ORIENTATIONDIFF}(s_2, s_t) \end{aligned} \tag{6.4}$$

Die Kostenfunktion besteht aus drei mathematischen Termen. Der erste Kostenterm wurde von der in PlexNav definierten Kostenfunktion übernommen. Dieser faktorisiert die durchschnittlichen Kosten $C(q_i)$ für alle Zwischenzustände q_i zwischen s_1 und s_2 mit der Euklidischen Distanz beider Zustände. $C(q)$ ist das Maximum der gewichteten Kosten für

- die Höhendifferenz zwischen den Radaufstandspunkten zweier benachbarter Antriebsmodule in der Karte[4],
- die Unebenheit des Geländes, welche durch die *PlexMap* gegeben ist und
- die Zellengröße der Karte, abhängig von der Anzahl der Messpunkte in dem entsprechenden Kartenbereich.

[4]Die maximal überwindbare Höhendifferenz wird automatisch an die aktuelle Roboterkonfiguration (Anzahl der Module beziehungsweise Länge des Zentralkörpers) angepasst.

Kosten für Steigungen aus der Karte, werden zunächst auf Grund der Fähigkeit schlangenartiger Roboter, vertikale Stufen einer gewissen Höhe zu erklimmen, nicht berücksichtigt. Jedoch werden die Fähigkeiten des Roboters betrachtet, Hindernisse in Abhängigkeit von seiner Konfiguration zu überwinden.

Zusätzlich zu der bereits bestehenden Gleichung, wurden ein zweiter und dritter Kostenterm eingefügt, um die nicht-holonomen Einschränkungen[5] von radgetriebenen schlangenartigen Robotern abzubilden. Der zweite Term berechnet die Orientierungsdifferenz (Gierwinkel γ) zwischen zwei benachbarten Zuständen in Abhängigkeit von ihrer Euklidischen Distanz normiert mit π. Dies erhöht die Gesamtkosten wenn große Orientierungsänderungen bei kurzen zurückgelegten Entfernungen auftreten. Der dritte Term verhindert ein überschreiten des maximal möglichen Kurvenradius, insbesondere wird dadurch ein auf der Stelle drehen des Roboter vermieden. Dazu wird die Orientierungsdifferenz zwischen s_2 und des Zielzustandes s_t berechnet. Beide Terme werden jeweils mit Gewichtungsfaktoren ϵ und ζ versehen [91].

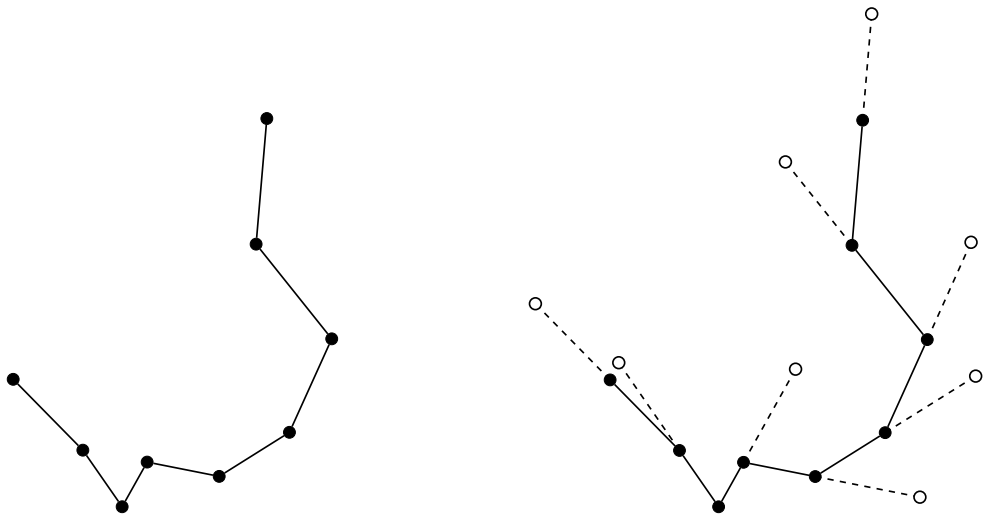
6.5.2 Sekundärer Nächster Nachbar Raum

Die Modifikationen und Erweiterungen der RRT* Kostenfunktion, insbesondere die Nebenbedingungen für nicht-holonome Bewegungen, führen dazu, dass sehr viele erzeugte Samples verworfenen werden. Dieses Problem tritt hauptsächlich während der Suche nach einer geeigneten Verbindung (Nächste-Nachbar-Suche) zwischen neu erzeugten Samples und dem existierenden RRT Baum auf. Dieser Schritt ist sehr entscheidend bei der Suchbaumerzeugung in allen RRT Planungsverfahren. Abhängig von den verschiedenen Möglichkeiten zur Verbindung der Knoten treten viele unterschiedliche Effekte auf. Ein Beispiel dafür ist die Nichtoptimalität des Standard RRT Algorithmus im Vergleich zum RRT*, der durch leichte Änderungen der Verbindungsregel von Knoten, asymptotisch optimale Lösungen findet. Im Falle der normalen Euklidischen Nächste-Nachbar-Suche mit der oben verwendeten Kostenfunktion wird die Nebenbedingung maximaler Lenkwinkel γ häufig verletzt und das entsprechende Sample verworfen. Dies führt zu einer schlechten Performance der Planung und verlängert die benötigte Planungszeit erheblich. Auf Grund dieser Problematik wurde ein *Sekundärer Nächster Nachbar Raum* (*SNN*) eingeführt [91]. Der *SNN* Raum ist Vergleichbar mit einer Projektion der Nächste-Nachbar Verbindung für neu erzeugte Samples in Richtung des Wurzelknotens des RRT*. Die Projektion führt zu Verbindungen bzw. Kanten mit Winkeln kleiner als der maximale Lenkwinkel (γ).

Für den RRT* mit *SNN* Erweiterung, werden zwei separate jedoch verbundene Räume bzw. Datenstrukturen angelegt. Der primäre Raum beinhaltet alle Kno-

[5] Als nicht-holonom bezeichnet man die Einschränkung, einen Roboter oder ein Fahrzeug nicht auf der Stelle drehen zu können, sondern eine bestimmte Vorwärts- oder Rückwärtsbewegung ausführen zu müssen, um Kurvenfahrten auf engstem Raum durchzuführen.

ten[6] und Kanten des klassischen RRT* Baumes, wie in Abbildung 6.7a dargestellt. Neben dem primären Raum wurde ein sekundärer Raum hinzugefügt, der für jeden primären Knoten exakt einen sekundären Knoten beinhaltet (siehe Abbildung 6.7b). Samples des RRT* Baum und deren Verknüpfungen im primären Raum werden als schwarz gefüllte Knoten und Linien dargestellt. Kanten im sekundären Raum (gestrichelten Linien) beschreiben die Beziehung zwischen einem sekundären Knoten (weiß gefüllte Knoten) und dem zugehörigen primären Knoten. Mit Hilfe dieser Verknüpfung, ist es möglich auf den korrespondierenden Knoten des jeweils anderen Raumes zuzugreifen. Da der Wurzelknoten keine Orientierung besitzt kann für diesen auch kein sekundärer Knoten angelegt werden.



(a) Baumstruktur eines klassischen RRT* Algorithmus.

(b) RRT* mit *SNN* Erweiterung.

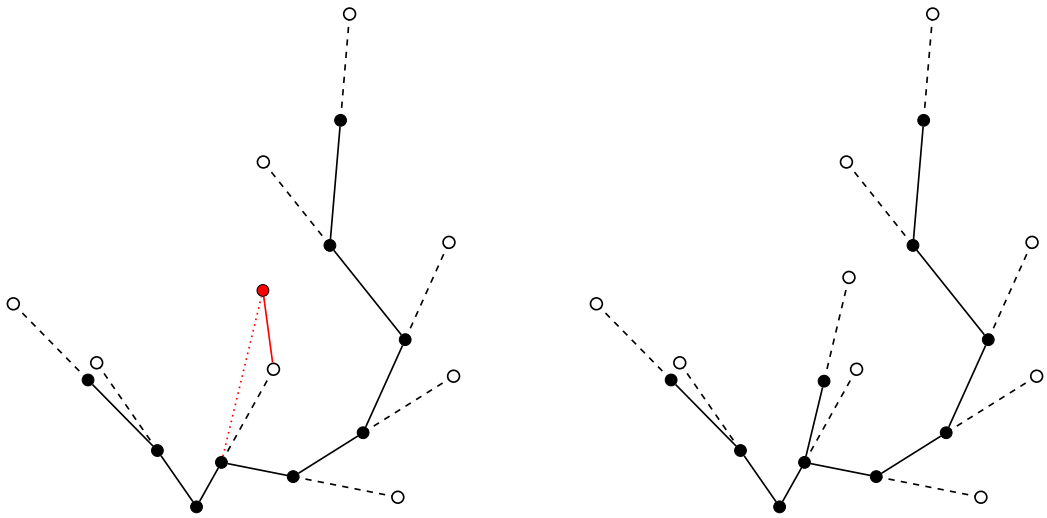
Abb. 6.7: *Sekundärer Nächster Nachbar Raum* Erweiterung des RRT* Algorithmus.

Sekundäre Knoten werden relativ zum primären Knoten um eine parametrierbare Entfernung δ verschoben. Die Richtung dieser Verschiebung δ definiert sich als Richtungsvektor der Kante zwischen dem primären Knoten und dessen Vorgängerknoten im primären Raum. Bei der Festlegung der Verschiebung δ muss lediglich berücksichtigt werden, dass durch die *SNN* Erweiterung, die maximale Änderung der Orientierung ϕ zwischen einem neu hinzugefügten Sample und seinem gefundenen Vorgängerknoten, vom Verhältnis der maximalen RRT Schrittweite s und der Verschiebung δ abhängt. Der damit erzielte Schwellenwert für Orientierungsänderungen ist sowohl für Gier- (γ) als auch Nickwinkel (β) wirksam und wird immer kleiner als 180° sein, wenn $\delta \geq s$:

$$\phi = arcsin\left(\frac{s}{\delta}\right) \tag{6.5}$$

[6]Jedes RRT* Sample entspricht einem Knoten im RRT* Baum und enthält eine 6-D Pose, die den Roboterursprung repräsentiert.

Zur Integration der *SNN* Datenstruktur in den RRT* wurden die Methoden zum Einfügen neuer Knoten und zum Neuverknüpfen von vorhandenen Knoten entsprechend angepasst. Abbildung 6.8a zeigt das Einfügen eines neuen zufällig erzeugten Sample (roter Knoten). Zunächst wird für das neue Sample eine nächste Nachbarn Suche im sekundären Raum durchgeführt. Der gefundene nächste Nachbar ist durch eine rote Linie mit den neuen Knoten verbunden. Mit Hilfe der Beziehung zwischen dem primären und sekundären Raum kann der korrespondierende Knoten im primären Raum als Vorgänger für das neue Sample ausgewählt werden. Abbildung 6.8a verdeutlicht dies durch die gestrichelte rote Linie. Auf Basis des neuen Sample und des selektierten Vorgängerknotens wird ein neuer Knoten im primären Raum angelegt. Dieser neue Knoten wird, wie in Abbildung 6.8b dargestellt, ausgehend von seinem Vorgängerknoten mit der festgelegten RRT Schrittweite s in Richtung des Samples verschoben. Abschließend wird ein korrespondierender sekundärer Knoten im *SNN* Raum angelegt.



(a) Nächste Nachbarn Suche findet im *SNN* Raum statt.

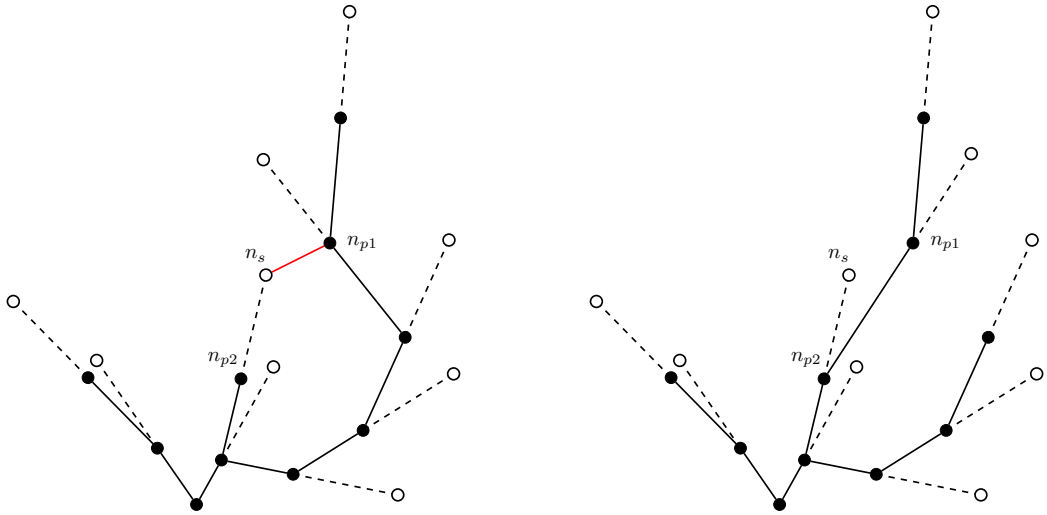
(b) Einfügen neuer Knoten in den primären und den *SNN* Raum.

Abb. 6.8: Einfügen neuer Knoten (Sampling) in den RRT* Baum mit *SNN* Raum.

Der Prozess zur Neuverknüpfung vorhandener Knoten[7] nutzt den *SNN* Raum um kürzere Pfade zu finden. Zunächst wird für jeden Knoten n_s im *SNN* Raum der nächste Nachbar n_{p1} im primären Raum gesucht. Eine Neuverknüpfung ist möglich, wenn der gefundene nächste Nachbar n_{p1} nicht direkt mit dem zu n_s korrespondierenden primären Knoten n_{p2} verbunden ist. Abbildung 6.9a zeigt ein Beispiel, bei dem diese Regel verletzt ist. Im Falle einer Neuverknüpfung wird zunächst die Verbindung zwischen dem nächsten Nachbarn Knoten n_{p1} und seinem Vorgängerknoten im primären RRT* Raum entfernt. Danach wird eine neue Kante zwischen dem Knoten n_{p1} und seinem korrespondierender nächsten Nachbar n_{p2} im primären Raum hergestellt. Anschließend müssend die entsprechenden sekundären Vertreter der modifizierten Knoten zur Wahrung der Konsistenz

[7] Eine Neuverknüpfung vorhandener Knoten erzeugt günstigere Pfade entlang der Baumstruktur durch Umordnen des Baumes [33]

an die neue Baumstruktur angepasst werden. Der resultierende RRT* Baum mit *SNN* Raum ist in Abbildung 6.9b dargestellt. Die Überprüfung von Kollisionen erfolgt wie beim klassischen RRT* weiterhin im primären Raum [91].



(a) Nächste Nachbarn Suche im primären Raum für jeden Knoten aus dem sekundären Raum.

(b) Neuverknüpfung der gefundenen Knotenpaare im primären Raum und Aktualisierung des sekundären Raum.

Abb. 6.9: Neuverknüpfung von Knoten beim RRT* mit *SNN*.

6.6 Zusammenfassung

In diesem Kapitel wurde, basierend auf dem PlexNav Planungsframework und der adaptiven Steuerung von KAIRO 3, ein Ansatz zur Navigations- und Bewegungsplanung für rekonfigurierbare schlangenartige Roboter vorgestellt. Das vorgestellte Kartierungsmanöver ermöglicht eine verbesserte 3-D Erfassung der Umwelt in einer Karte. Diese *PlexMap* wurde als Grundlage für die Trajektorienplanung mit Hilfe eines modifizierten RRT* genutzt. Unter Berücksichtigung der Roboterkonfiguration werden entsprechende Pfade zum Überwinden von Hindernissen geplant. Zur Ausführung der geplanten Trajektorien wurde eine Erweiterung der virtuellen Schiene für die Bewegungssteuerung vorgestellt. Die *Sekundärer Nächster Nachbar Raum* Erweiterung des RRT* Algorithmus sowie die speziell für rekonfigurierbare Roboter modifizierte Kostenfunktion wurden getrennt betrachtet.

7 Experimente und Ergebnisse

Dieses Kapitel beschreibt die Durchführung von Experimenten und Tests zur Evaluierung der in dieser Arbeit entwickelten und implementierten Konzepte. Für jedes der vier Kernthemen *Optimierung des Hardwaredesigns und der Steuerungssoftware*, *Bestimmung von optimalen Roboterkonfigurationen*, *Anpassung der Fortbewegung an die Umgebung* und *Konfigurationsabhängige Navigation und Lokalisierung* werden die aus den Experimenten gewonnenen Ergebnisse und Resultate diskutiert.

7.1 Entwurf rekonfigurierbarer Multi-Roboter-Systeme

7.1.1 Flexible Steuerungsarchitektur für RMRS

Zur Evaluierung der flexiblen Steuerungsarchitektur wurden Tests mit verschiedenen Hardware Konfigurationen sowohl in der Simulation als auch mit dem Roboter KAIRO 3 durchgeführt. Insbesondere die Anforderungen an eine Steuerung für rekonfigurierbare Robotersysteme wurden auf Grundlage der Implementierung mit ROS 2 untersucht. Kernanforderungen sind das An- und Abkoppeln einzelner Robotermodule sowie die Trennung eines Roboters in mehrere unabhängige Teilsysteme. Im Folgenden wird die Durchführung von drei verschiedenen Testszenarien zur Überprüfung der gestellten Anforderungen erläutert.

Szenario 1 : Validierung des Entitäten-basierten Ansatzes

Dieses Testszenario evaluiert die grundlegende Funktionsweise der in Kapitel 3.3 eingeführten verteilten Softwarearchitektur. Insbesondere wird das Konzept der Verwendung sogenannter Entitäten zur Erzeugung einer Sensor-Aktor-Baumstruktur validiert. Dazu wurde die Steuerungssoftware auf einem einzelnen Roboter bestehend aus zwei Antriebsmodulen und einem Gelenkmodul gestartet (Abbildung 7.1 rechts). Die Module sind mechanisch und elektrisch miteinander verbunden, so dass die Steuerungsplatinen über den CAN-Bus ausgelesen werden können. Die zentrale Steuerungssoftware erzeugt auf Basis der erhaltenen Informationen, die in Abbildung 7.1 (links) dargestellte Baumstruktur der Aktor und Sensor Entitäten.

Roboter 1

- *id110* Antriebsmodul
  - *id111* Antrieb Aktor
  - *id112* Antrieb Sensor
- *id120* Gelenkmodul
  - *id121* Gelenk Aktor
  - *id122* Gelenk Sensor
- *id130* Antriebsmodul
  - *id131* Antrieb Aktor
  - *id132* Antrieb Sensor

Abb. 7.1: Baumstruktur der erzeugten Entitäten für die Hardwarekomponenten einer KAIRO 3 Konfiguration bestehend aus drei Modulen.

Für alle Antriebsmodule wurden die vorhandenen Hardwarekomponenten identifiziert und entsprechende Softwareeinheiten (Entitäten) zur Steuerung der Motoren (*Antrieb Aktor*) und zum Auslesen der Sensordaten (*Antrieb Sensor*) in der Steuerungssoftware generiert. Analog dazu wurden die Softwareeinheiten *Gelenk Aktor* und *Gelenk Sensor* automatisch für die Hardwarekomponenten des Gelenkmoduls erstellt. Die Initialisierungsroutine des CAN-Bus wird genutzt um festzustellen in welcher Reihenfolge die Module untereinander gekoppelt sind. Basierend auf diesen Verbindungsinformationen, konnten die gefundenen Entitäten erfolgreich zu der oben gezeigten Baumstruktur angeordnet werden. Die erhaltene Roboterstruktur wird von der übergeordneten adaptiven Robotersteuerung genutzt um die einzelnen Sensoren und Aktoren auszulesen bzw. anzusteuern.

Szenario 2 : Kopplen zweier physikalisch getrennter Roboter

Mit der in Kapitel 3.4 entwickelten Schnittstelle können einzelne Module bzw. Roboter gekoppelt oder getrennt werden. Da zum Zeitpunkt der Evaluierung die Schnittstelle nicht Einsatzbereit war, wurde der Kopplungsvorgang simuliert. Zusätzlich zum Roboter aus Szenario 1 wurde auch eine lokale Instanz der Steuerungssoftware auf einem zweiten Roboter mit einer Konfiguration aus zwei Antriebsmodulen und zwei Gelenkmodulen gestartet. Beide Steuerungsinstanzen sind über ein drahtloses Netzwerk miteinander verbunden und tauschen Informationen über die Hardwarekonfiguration aus. Durch die simulierte Kopplung der Roboter wird eine der beiden lokalen Robotersteuerungen deaktiviert. Die noch aktive Steuerung übernimmt dann die Steuerung des gesamten Robotersystems (siehe Abbildung 3.8 in Kapitel 3.3). Die Sensor- und Aktuatoreinheiten

beider Roboter werden schließlich in einer gemeinsamen Baumstruktur zusammengefasst.

Szenario 3 : Trennen der zuvor gekoppleten Roboter

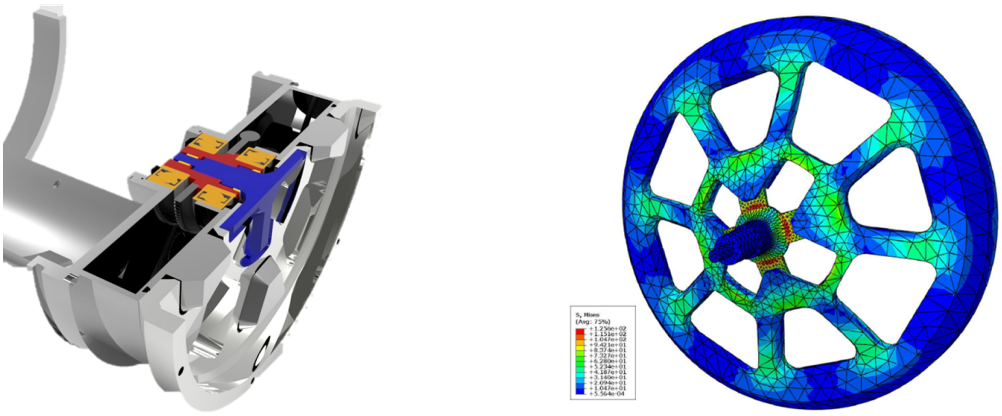
Nach der Trennung der zuvor hergestellten Kopplung des Robotersystems, übernimmt die passive lokale Steuerungsinstanz erneut die Kontrolle über den zweiten Roboter. Insbesondere wurde die Verbindung der Sensor-Aktor Baumstruktur gelöst, so dass sich beide Roboter unabhängig voneinander bewegen können. Dabei wurde die Struktur beider Steuerungen durch Verschieben der entsprechenden Entitäten zwischen den Steuerungsinstanzen adaptiert [64].

Durch die Umsetzung mit Hilfe des ROS 2 Frameworks basierend auf dem Data Distribution Service wird für die hardware-nahe Steuerung keine zentrale Softwareeinheit benötigt. Dadurch konnte das Ziel einer verteilten Steuerungsarchitektur erreicht werden. Diese Dezentralisierung ermöglicht den Einsatz von Hardwareeinheiten (Embedded PC) in jedem Modul des Roboters, die über ein Netzwerk (z. B. Ethernet) miteinander verbunden sind. Bei der Kopplung werden Entitäten, die durch den Kopplungsprozess hinzugefügt oder durch Entkopplung entfernt wurden, automatisch durch das Gesamtsystem registriert. Die entsprechenden Steuerungsinstanzen werden von einem Überwachungsprogramm aktiviert oder deaktiviert und flexibel auf eine der vorhandenen Steuerungseinheiten verteilt.

Die durchgeführten Tests zeigten, dass die eingesetzte flexible Steuerungssoftware in der Lage ist, sich an veränderte Bedingungen durch An- und Abkoppeln einzelner Robotermodule sowie Aufteilen eines Roboters anzupassen.

7.1.2 Hardwareschnittstelle für RMRS

Um das grundlegende Funktionsprinzip der in Kapitel 3.4 entwickelten Hardware-Schnittstelle zu verifizieren, wurde in einem ersten Schritt das CAD-Modell aller relevanten Komponenten aus Abbildung 7.2a erstellt. Die zentralen Komponenten sind im Wesentlichen der blau dargestellte Bajonettverschluss, die rot markierte Hohlwelle und die orange dargestellten Lager. Das 3-D Model wurde ebenfalls zur Optimierung der auftretenden Kräfte und Belastungen genutzt. Alle kritischen Bauteile wurden einer Analyse mittels der Finite-Elemente-Methode (FEM) unterzogen und bezüglich Stabilität und Kraftverteilung optimiert. Abbildung 7.2b zeigt die Verteilung der auf die Schnittstelle wirkenden Kräfte. Es ist deutlich zu erkennen, dass die größten Kräfte (rote Flächen) an den Flanken des Riegels wirken [7].



(a) CAD Model der gekoppelten Schnittstelle.

(b) Optimierung der kritischen Schnittstellenbauteile mit Hilfe der FEM-Analyse.

Abb. 7.2: CAD Konstruktion und FEM Optimierung der KAIRO 3 Schnittstelle [7].

Evaluierung mittels 3-D gedrucktem Prototyp

In einem zweiten Schritt wurden alle selbst entworfenen und konstruierten Komponenten der Schnittstelle 3-D gedruckt und ein Prototyp mit Hilfe von Zukaufteilen erstellt. Anhand dieses Prototyps wurden die Hardware-Schnittstelle sowie alle mechanischen und elektrischen Komponenten getestet. Abbildung 7.3 zeigt die endgültige Version der aktiven und passiven Schnittstelle sowie eine Nahaufnahme der Federstifte. Dank des 3-D Drucks konnten notwendige Konstruktions- und Designanpassungen sehr kostengünstig und schnell umgesetzt werden. Eine detailliertere Betrachtung der einzelnen Komponenten sowie Tests zur Robustheit der Schnittstelle werden im Folgenden vorgestellt.

Abb. 7.3: Zusammengebauter Prototyp der KAIRO 3 Schnittstelle. Die Hauptbestandteile der aktiven (links) und passiven Schnittstelle (mittig) wurden 3-D gedruckt. Die Federkontakte zur Datenübertragung (rechts) sind mit einer 3-D gedruckten Montagehilfe in die Schnittstelle integriert [8].

Evaluierung der elektrischen und Datenschnittstelle

Die Tests der elektrischen Schnittstelle wurden mit Hilfe eines Versuchsaufbau durchgeführt. Zur Evaluierung des eigen entwickelten Elektronikboard wurde ein Steuerungscomputer mit der Testsoftware bespielt. Mit diesem Testaufbau wurde das Durchschalten und Trennen der stromführenden Leitungen sowie das Verbinden und Abschalten der Kommunikationsleitungen des CAN-Bus getestet. Die Testergebnisse haben gezeigt, dass es mit den ausgewählten MOSFETs möglich ist, hohe Ströme von mehr als 10 A bei 24 V zu schalten und damit die Stromversorgung zwischen zwei gekoppelten Modulen herzustellen. Auch die Übertragung der hochfrequenten CAN-Bus Signale wird durch die Schnittstellenelektronik ermöglicht.

Im zweiten Schritt wurden die Federkontakte und Pins im eingebauten Zustand an der Schnittstelle überprüft. Die zugekauften Federkontakte für die Datenübertragung stellen einen sehr guten Kontakt her und ermöglichen das Überbrücken eines potenziellen Spalts von bis zu 5 mm zwischen der aktiven und passiven Schnittstelle. Die zur Stromübertragung selbst entwickelten Federkontakte können die gewünschten Spannungen und Ströme übertragen jedoch wurde festgestellt, dass die Kontakte beim Eindrücken leicht verkanten. Dies ist der prototypischen Fertigung mittels 3-D Druck geschuldet und kann durch einen Nachtbearbeitungsschritt behoben werden. Zudem besitzen die verwendeten Federn eine zu hohe Federkraft, so dass das Zusammendrücken der Schnittstelle sehr viel Kraft erfordert. Dieses Problem kann durch Änderung der Federn verbessert werden [8].

Evaluierung der Sensorik und Schnittstellensteuerung

Zur Detektion verschiedener Zustände der Schnittstelle werden mehrere Endschalter verwendet. Der in die Kontaktfläche integrierte Drucktaster zeigte bei den Tests zuverlässig an, ob die aktive und passive Schnittstelle Kontakt haben. Eine Steigerung der Ausfallsicherheit kann durch die Verwendung von zwei oder mehreren redundanten Schaltern erreicht werden. Zur Bestimmung der axialen Position des Riegels sind zwei Schalter am inneren Ende der Welle angebracht. Mit Hilfe dieser Endschalter konnte während der Tests immer die komplett ausgefahrene bzw. eingefahrene Position des Riegels bestimmt werden.

Die Funktionsfähigkeit der Schnittstellensteuerung wurde durch Tests mit dem Prototypen der aktiven Schnittstelle iterativ verbessert. Wenn das System in einem unbekannten Zustand eingeschaltet wird, erfolgt die automatische Initialisierungsfunktion. Bei allen Tests konnte die Position des Riegels durch die Initialisierung bestimmt werden und der Zustandsautomat aus Abbildung 3.15 befand sich entsprechend im Zustand *Docked* oder *Undocked*. Die korrekte Funktionsweise der Schnittstellensteuerung konnte durch mehrfaches Öffnen und Schließen der Schnittstelle bei gleichzeitiger Überwachung der Zustandsübergange des Zustandsautomaten überprüft und somit verifiziert werden [8].

Evaluierung der mechanischen Schnittstelle und des Kopplungsvorgang

Bereits während der Montage der mechanischen Bauteile wurde das Funktionsprinzip des Bajonettverschlusses bestehend aus Riegel, Hohlwelle, Lagerungen und Antriebszahnrädern überprüft und optimiert. Mit dem Einbau des Antriebsstrang bestehend aus Motor, Getriebe und Übersetzung, konnten erste erfolgreiche Tests zum Ein- und Ausfahren des Riegels durchgeführt werden. Versuche zur Zuverlässigkeit des Kopplungsvorgangs wurden mit dem in KAIRO 3 integrierten Prototypen der Schnittstelle durchgeführt. Dazu wurden zwei KAIRO 3 Roboter jeweils mit einer aktiven und passiven Schnittstelle in einem Abstand von 30 cm aufgestellt und mit einer manuellen Steuerung aufeinander zu bewegt. Die aktive Schnittstelle befand sich jeweils im geöffneten Zustand und wurde automatisch geschlossen sobald ein Kontakt der beiden Schnittstellen durch den Sensor erkannt wurde. Bei insgesamt fünf von sechs Versuchen konnte erfolgreich eine Kopplung herstellt und danach wieder gelöst werden. Bei einem Versuch führte ein zu großer Winkelversatz ($> 10°$) zum Verkanten des Riegels in der Struktur der passiven Schnittstelle. Ein Positionsversatz von bis zu 5 mm konnte dank der integrierten Führungsstrukturen problemlos ausglichen werden. Während allen Experimenten hat die Mechanik zur Bewegung des Riegels zuverlässig funktioniert und es konnte stets eine kraftschlüssige Verbindung hergestellt werden [8].

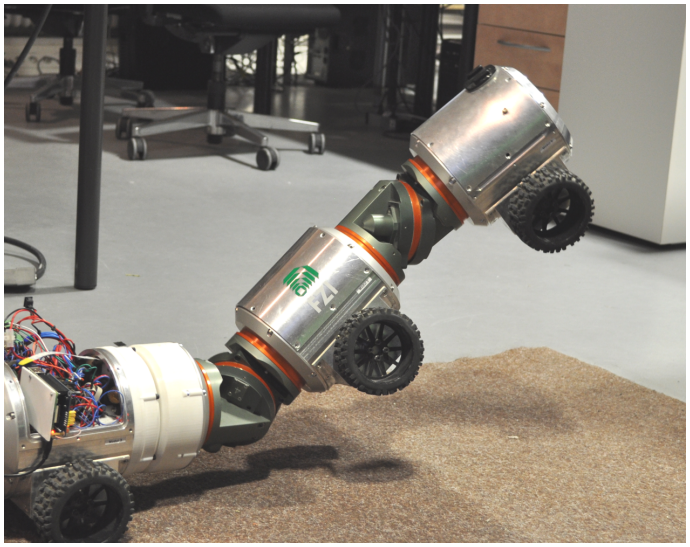


Abb. 7.4: Experiment zur Robustheit des Prototypen der KAIRO 3 Schnittstelle.

Um die Kraftübertragung und Robustheit des Schnittstellendesigns zu bewerten, wurden im gekoppelten Zustand die ersten drei Module von KAIRO 3 angehoben. Wie in Abbildung 7.4 zu sehen, befindet sich die Schnittstelle zwischen dem vierten und fünften Modul, so dass sehr große Kräfte auf die gekoppelte Schnittstelle wirken. Auf Grund der hohen Kräfte bildet sich ein Spalt von fast 5 mm am oberen Ende Schnittstelle. Trotzdem halten alle mechanischen Bauteile und die Kopplung der großen Belastung stand. Für den 3-D gedruckten Prototypen

war das nicht vorausgesetzt und übertrifft sogar die Erwartungen. Mit der erfolgreichen Evaluierung des Prototypen ist davon auszugehen, dass eine aus Metall gefertigte Version der entwickelten Schnittstelle alle gestellten Randbedingen erfüllt.

7.1.3 Autonomes Docking mit Visual Servoing

Die Evaluierung des in Kapitel 3.5 vorgestellten Verfahren zum automatischen Docking von RMRS wurde in der Robotik Simulationsumgebung Gazebo durchgeführt. Dazu wurden zwei Instanzen von KAIRO 3 mit einer aktiven und passiven Schnittstelle in eine virtuelle Umgebung eingefügt. Bilder zum Einlernen des Detektors und zur Evaluierung des Tracking wurden mit Hilfe einer simulierten Kamera in der aktiven Schnittstelle aufgenommen. Die Fortbewegung des aktiven Roboters wurde durch die Anbindung der adaptiven Steuerung an die Physiksimulation realisiert. Der Detektor wurde mit zwei verschiedenen Sätzen an Testdaten trainiert. Zur Evaluierung der Qualität der Detektoren wurden 14 Testreihen mit jeweils verschiedenen Startpositionen und -orientierungen durchgeführt. Mit Hilfe des ersten Detektors konnte die Schnittstelle in 12 von 14 Versuchen erkannt werden. Durch Anwendung des zweiten Detektors konnte die Detektion verbessert werden und die Schnittstelle bei allen 14 Versuchen erfolgreich erkannt [68].

Das Tracking Verfahren sowie der gesamte Visual Servoing Prozess wurde, wie in Abbildung 7.5 veranschaulicht, in einer Simulationsumgebung mit jeweils drei verschiedenen Startpositionen getestet. Der passive Roboter wurde bei

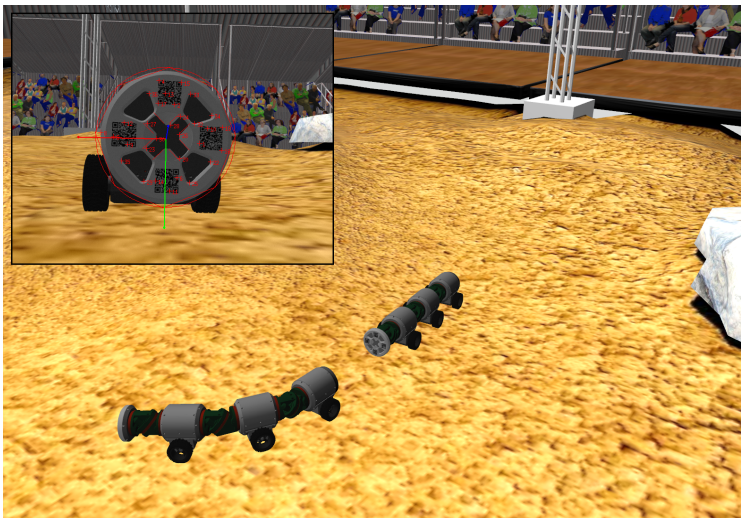


Abb. 7.5: Simulation des Visual Servoing Prozess zum automatischen Docking von zwei KAIRO 3 Robotern.

allen Tests im Koordinatenursprung an der Pose $\begin{bmatrix} X & Y & YAW \end{bmatrix}$ mit $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$ angeordnet. Unter der Annahme, dass die aktive sowie passive Schnittstellen

beide Roboter zueinander zeigen, befindet sich die erste Pose des aktiven Roboters in 1,5 m Entfernung zum passiven Roboter bei $\begin{bmatrix}1,5 & 0 & 0\end{bmatrix}$. Mit Hilfe der zweiten Pose $\begin{bmatrix}1,5 & 0,5 & 0\end{bmatrix}$ wurde der Ausgleich eines seitlichen Versatzes von 0,5 m getestet. zur Evaluierung der Robustheit des Verfahrens enthält die dritte Pose $\begin{bmatrix}1,5 & 0,5 & 45°\end{bmatrix}$ zusätzlich einen Winkelversatz von 45°. Basierend auf den Positionsdaten des Trackers wird der aktive Roboter durch die Steuerung in Richtung des passiven Roboters bewegt bis die beiden Schnittstellen aufeinander treffen. Während des Visual Servoing Prozess zur Annäherung der beiden Roboter wurden zur Bewertung kontinuierlich die Ist- und Sollwerte der Roboterpose (*X*, *Y* und *YAW*) aufgezeichnet. Abbildung 7.6 zeigt beispielhaft die Abweichung bzw. den Fehler zwischen den Ist- und Sollwerten ausgehend von der dritten Startpose $\begin{bmatrix}1,5 & 0,5 & 45°\end{bmatrix}$. Insgesamt wurden jeweils fünf Versuche für jede der drei Startpositionen durchgeführt. Für die erste Position konnte bei allen fünf Versuchen die passive Schnittstelle erfolgreich erreicht werden. Bei den anderen beiden Testreihen waren jeweils drei bzw. zwei Tests erfolgreich. Die Fehlschläge lassen sich durch einen zu großen Winkelversatz (*YAW*) am Ende des Prozess erklären. Die Problematik der Winkelabweichung tritt insbesondere für Startpositionen mit einem initialen Winkelversatz auf. Dies durch die Initialisierung des Trackers verursacht, da der Detektor lediglich Bildkoordinaten ohne Orientierung liefert. Für die *X* und *Y* Positionen wurden hingegen sehr genaue und stabile Ergebnisse erzielt [68].

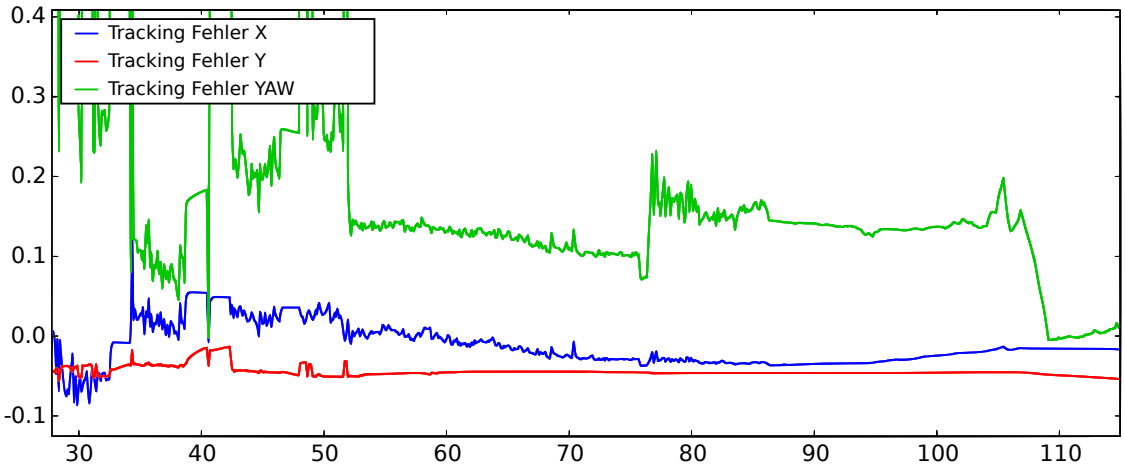


Abb. 7.6: Auswertung der Abweichung zwischen den Roboter Ist- und Sollwerten während des Visual Servoing Prozess.

7.2 Generierung und Optimierung von Roboterkonfigurationen

Die Bewertung der Verfahren zur Konfigurationsgenerierung und -optimierung für modulare rekonfigurierbare Robotersysteme ist in vier Unterkapitel unterteilt, in denen zunächst die drei Teilschritte des Syntheseprozesses einzeln betrachtet werden. Der letzte Abschnitt untersucht die Leistungsfähigkeit des Gesamtsystems zur Konfigurationsoptimierung. Alle gezeigten Tests basieren auf einem Bausteinkatalog, der für den rekonfigurierbaren Serviceroboter KAIRO 3 entwickelt wurde. Zusätzlich wird die Generalisierung und Übertragung der Implementierung auf andere Systeme am Beispiel von modularen Satelliten dargestellt und evaluiert.

7.2.1 Vergleich der Algorithmen zur Bausteinauswahl

Zur Realisierung der Bausteinauswahl des in Kapitel 4.3 vorgestellten Syntheseprozess wurden drei verschiedene Algorithmen implementiert und untersucht. Dieser Abschnitt vergleicht die Rechenzeiten und Erfolgsraten des Brute-Force-Algorithmus, des Greedy-Algorithmus und der ganzzahligen linearen Optimierung und fasst die gewonnen Resultate zusammen. Die im Folgenden für die Bausteinauswahl gezeigten Laufzeitmessungen wurden unter Linux auf einem Intel(R) Core(TM) i7-2820QM (2,3 GHz) erstellt.

Brute-Force-Algorithmus

Der in Kapitel 4.3.1 beschriebene Brute-Force-Algorithmus findet immer eine optimale Lösung, jedoch zeigt sich, dass die Laufzeit sehr ineffizient ist. Bei einem Bausteinkatalog mit 15 Bausteinen sowie drei Ressourcen-, zwei Komponenten- und zwei Typen-Anforderungen dauert die Ausführung bereits 54 min. Würde man für einen realen Einsatzfall von einer Obergrenze von 1000 Bausteinen ausgehen, so können sich Laufzeiten von bis zu mehreren Tagen ergeben. Der Brute-Force-Algorithmus ist aus diesem Grund nicht für eine effiziente Bausteinauswahl geeignet [76].

Greedy-Algorithmus

Greedy-Algorithmen besitzen die Eigenschaft auf Grund von lokalen Minima oftmals keine optimale Lösungen zu finden. Deshalb wurde bei der in Kapitel 4.3.1

vorgestellten Implementierung ein Backtracking verwendet. Durch diesen zusätzlichen Aufwand ist die Laufzeit des Algorithmus im schlechtesten Falle genau gleich wie die des Brute-Force-Algorithmus. Zur Auswertung des Greedy-Algorithmus wurde der selbe Testdatensatz wie bei der Evaluierung des Brute-Force-Algorithmus eingesetzt. Bei den insgesamt 200 Testfällen konnte jeweils in weniger als 40 ms eine Lösung gefunden werden. Daraus ergibt sich im Regelfall, mit zunehmender Komplexität, eine etwas schneller als linear ansteigende Ausführungszeit. Damit scheint der Greedy-Algorithmus auf Grund seiner schnellen Laufzeit und der Tatsache immer eine Lösung zu finden, für die Bausteinauswahl geeignet zu sein. Jedoch gibt es den entscheidenden Nachteil, dass lediglich 75 % der gefunden Lösungen optimal sind. Bei 25 % der Lösungen ist mindestens ein Baustein mehr als nötig enthalten [76].

Ganzzahlige lineare Optimierung

Um die in Kapitel 4.3.1 vorgestellte ganzzahlige lineare Optimierung zu implementieren und zu bewerten, wurde der nicht-kommerzielle SCIP-Solver [79] für gemischte ganzzahlige Programme verwendet. Für den verwendeten Testdatensatz liefern der Brute-Force-Algorithmus und das ganzzahlige lineare Programm (LP) die gleiche Lösung, aber mit Hilfe des linearen Programms wird die optimale Lösung deutlich schneller gefunden. Der Vergleich des Greedy-Algorithmus mit der linearen Optimierung erfolgt mit Hilfe eines erweiterten Testdatensatz. Insgesamt wurden 250 Testfälle mit verschiedenen Bausteinkatalogen und Anforderungslisten durchgeführt. Bei der Auswertung der Tests wurde festgestellt, dass die Laufzeiten der linearen Optimierung und des Greedy-Algorithmus hauptsächlich von der Größe des Bausteinkatalogs abhängig sind. Für alle Tests erfordert die lineare Optimierung eine maximale Laufzeit von 109 s bei 4942 Einträgen im Bausteinkatalog. Der Greedy-Algorithmus hingegen, braucht im Schnitt immer länger als das LP und liefert dabei nur in 28 % aller Testfälle eine optimale Lösung [76].

Basierend auf der Auswertung der Tests lässt sich schlussfolgern, dass die ganzzahlige lineare Optimierung die besten Resultate liefert. Zudem lassen sich durch unterschiedliche Gewichtungen der Bausteine im Katalog[1], Faktoren wie z. B. Bausteingewicht oder Kosten berücksichtigen. Eine solche Gewichtung ist beim Brute-Force-Algorithmus oder Greedy-Algorithmus nicht direkt möglich.

7.2.2 Methoden der Regelinferenz

Zur Evaluierung der beiden Methoden zur Regelinferenz (siehe Kapitel 4.3.2), wurde neben der Laufzeit insbesondere die Komplexität und Erweiterbarkeit der Modellierung neuer Regeln bzw. Regelklassen betrachtet.

[1] Die Gewichtung der Bausteine wurde bei der Evaluierung des LP als äquivalent angenommen.

Evaluierung des Ontologie-basierten Reasoning

Die Umsetzung mittels Ontologie-basierter Modellierung, bietet den Vorteil neue Regeln und Eigenschaften zu modellieren und zuvor unbekannte Zusammenhänge zu inferieren ohne eine Änderung am Programmcode durchführen zu müssen. Diese Erweiterbarkeit und Flexibilität bringt jedoch einen erhöhten Aufwand und Komplexität bei der Modellierung mit sich. Abbildung 7.7 verdeutlicht an Hand von zwei Beispielen die Inferenz von Baustein-relativen Regeln zwischen zwei Individuen.

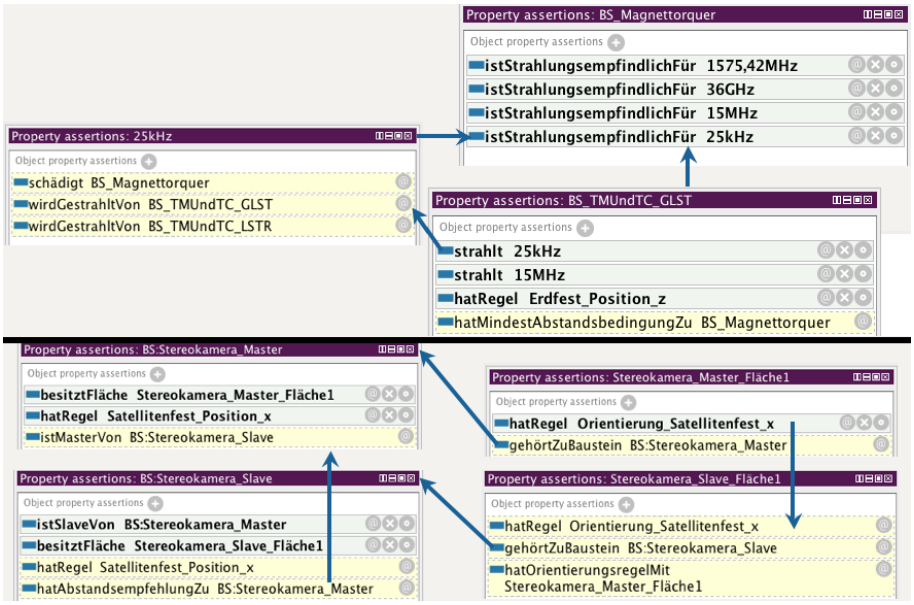


Abb. 7.7: Beispiel für die Modellierung von Abhängigkeiten zwischen Bausteinen und dem Ableiten von Regeln aus diesen [73].

Eine Auswertung der Laufzeit wurde an Hand mehrerer Testreihen mit unterschiedlichen Anzahlen an Bausteinen und Regeln durchgeführt. Für jede Testreihe wurde ein Mittelwert der Ausführungszeiten aus zehn Testläufen gebildet. Tabelle 7.1 zeigt die für das Ontologie-basierte Reasoning gewonnenen Testdaten. Die in der Ontologie durch das Reasoning erzeugten Beziehungen und Zusammenhänge werden in Abbildung 7.8 dargestellt. Je nach Modellierung können zur unbekannte Zusammenhänge automatisch erschlossen werden.

Für eine große Anzahl an Bausteinen und Regeln sind keine Messwerte vorhanden, da das Erzeugen der ausgewählten Baustein Instanzen in der Ontologie bei großen Bausteinmengen deutlich mehr Zeit beansprucht als das ableiten der Regeln. Aus diesem Grund war es nicht möglich, Tests mit dem ontologiebasierten Reasoner bei mehr als 550 Bausteinen durchzuführen.

# Bausteine	# Regeln	Laufzeit [s]
8	8	3,41
32	43	3,80
51	555	4,27
106	2.362	5,18
223	10.724	9,18
513	21.805	20,36
—	—	—

Tabelle 7.1: Durchschnittliche Laufzeiten der Regelinferenz mit dem Ontologie-basierten Reasoning.

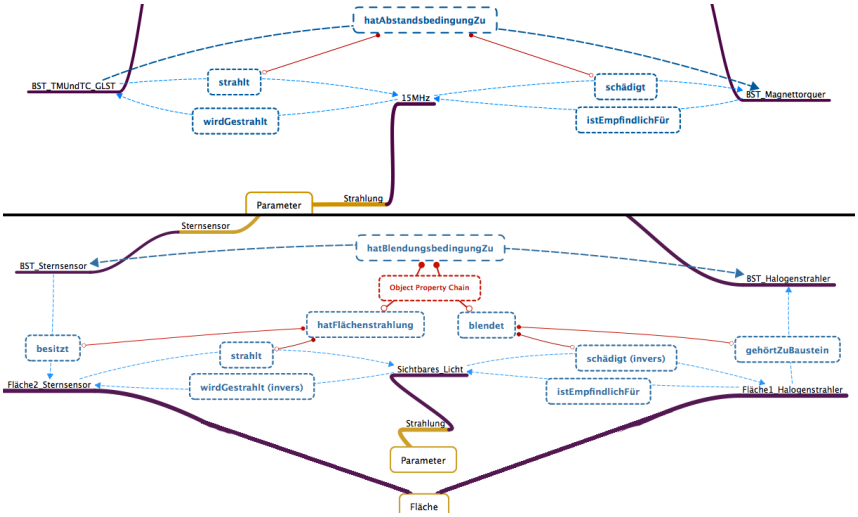


Abb. 7.8: Abgeleitete Beziehungen und Regeln zwischen zwei Bausteinen [73].

Evaluierung des individuellen problemspezifischen Reasoning

Im Gegensatz dazu, lässt sich mit der in Kapitel 4.2 dargestellten Modellierung, das Schlussfolgern und Ableiten von Bedingungen mit einem individuellen problemspezifischen Reasoner realisieren. Die Vorteile dieses Ansatzes sind vor allem die deutlich einfachere und intuitive Handhabung der Modellierung. Durch die spezielle Anpassung an das Ableiten von Baustein-relativen Regeln, ergibt sich ebenfalls ein Laufzeitvorteil gegenüber allgemeingültiger Reasoner wie z. B. *FaCT++*. Der Nachteil dieser Methode besteht darin, dass Anpassungen an den Regeln oder neu modellierte Zusammenhänge zu einer Modifikation der Implementierung führen können [73].

Durch den Vergleich der in Tabelle 7.1 und Tabelle 7.2 dargestellten Ausführungszeiten beider Inferenz Verfahren, zeigt sich ein deutlicher Laufzeitvorteil des individuellen problemspezifischen Reasoners. Der relative Anstieg der Laufzeit mit zunehmender Anzahl an Bausteinen und abgeleiteter Regeln ist bei

# Bausteine	# Regeln	Laufzeit [s]
8	18	0,08
32	40	0,10
51	871	0,13
106	4.211	0,25
223	20.399	0,95
430	20.578	1,10
850	81.686	5,56

Tabelle 7.2: Durchschnittliche Laufzeiten der Regelinferenz mit dem individuellen problemspezifischen Reasoning.

beiden Verfahren ähnlich groß. Die Anzahl der abgeleiteten Regeln hängt besonders von der Art der Bausteine (Bausteintyp) und den Beziehungen zwischen den Bausteinen ab. Insbesondere wurden in Tabelle 7.2 insgesamt 81.686 Regeln für 850 Bausteine abgeleitet. Bei dieser großen Zahl an Regeln wurden 80.000 Regeln durch relative Beziehungen zwischen zwei Bausteintypen A und B mit jeweils 400 Instanzen vom Typ A und 200 Instanzen vom Typ B erzeugt. Dies zeigt deutlich den Einfluss der ausgewählten Bausteintypen auf die Anzahl der angeleiteten Regeln [73].

Für jede Testreihe wurden zehn Durchläufe durchgeführt und jeweils der Mittelwert aus den Laufzeiten gebildet. Die Tests zur Regelinferenz wurden unter Linux auf einem Intel(R) Core(TM) i7-4900MQ (2,80GHz) durchgeführt. Beide Inferenz Verfahren wurden als Single-Thread Programme ausgelegt und nutzen nur einen CPU Core.

7.2.3 Evaluierung der Bausteinanordnung

Die Evaluierung und Bewertung der Bausteinanordnung mittels evolutionärer Optimierung umfasst mehrere Teilaspekte. Zunächst werden zwei verschiedene Methoden zur Initialisierung und Erzeugung neuer Individuen an Hand von Testfällen verglichen. Darauf aufbauend wird die Ausführungszeit der evolutionären Optimierung betrachtet und die Laufzeitverbesserung durch Parallelisierung diskutiert. Im Anschluss wird der Ablauf der Optimierung und die Qualität der Ergebnisse mit Hilfe eines Beispiels bewertet.

Initialisierung und Erzeugung von Individuen

Die Initialisierung und das Erzeugen neuer Individuen wird bei evolutionären Algorithmen im Regelfall durch zufälliges auswählen der zu optimierenden Eigenschaften bzw. Variablen erreicht. Der in Kapitel 4.3.3 dargestellte Algorith-

mus hingegen erzeugt neue Individuen unter Berücksichtigung grundlegender Randbedingungen zur Anordnung der Bausteine. Zum Einen wird ein kompaktes Cluster aus Bausteinen erzeugt, indem neue Bausteine ausgehend vom Zentrum an die vorhandenen Bausteine angefügt werden. Zusätzlich wird bei der Rotation der Bausteine bereits die Einhaltung der absoluten Orientierung angestrebt.

In dieser Arbeit wurden beide Verfahren, eine gänzlich zufällige und eine zielgerichtete Initialisierung implementiert. Beim Vergleich der in Tabelle 7.3 und Tabelle 7.4 gezeigten Testreihen[2], ist ein klarer Vorteil bei der Optimierung mit gezielter Erzeugung neuer Individuen zu sehen. Die gezielte Initialisierung der Startpopulation dauert zwar im Vergleich zur zufälligen Erzeugung deutlich länger, jedoch ist die kombinierte Laufzeit aus Initialisierung und Optimierung besser. Zudem werden weniger Iterationen benötigt, bis die Optimierung konvergiert bei gleichzeitig besserer oder ähnlicher Bewertung (Fitness).

Testreihe	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6
Anzahl Bausteine	8	33	55	106	51	146
Anzahl Regeln	18	35	43	69	871	307
Initialisierung [s]	0,024	0,078	0,126	0,239	0,110	0,314
Optimierung [s]	0,389	8,697	31,374	182,005	228,509	1846,91
Bewertung (Fitness)	0,843	1,158	2,505	2,764	4,136	5,121
Anzahl Iterationen	12	66	99	213	750	981

Tabelle 7.3: Ergebnisse der Optimierung mit zufälliger Erzeugung neuer Individuen.

Testreihe	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6
Anzahl Bausteine	8	33	55	106	51	146
Anzahl Regeln	18	35	43	69	871	307
Initialisierung [s]	0,031	0,307	1,105	5,912	0,851	12,191
Optimierung [s]	0,204	4,117	19,420	117,535	101,341	36,942
Bewertung (Fitness)	0,843	0,746	2,600	2,693	5,193	5,4
Anzahl Iterationen	6	27	60	126	308	17

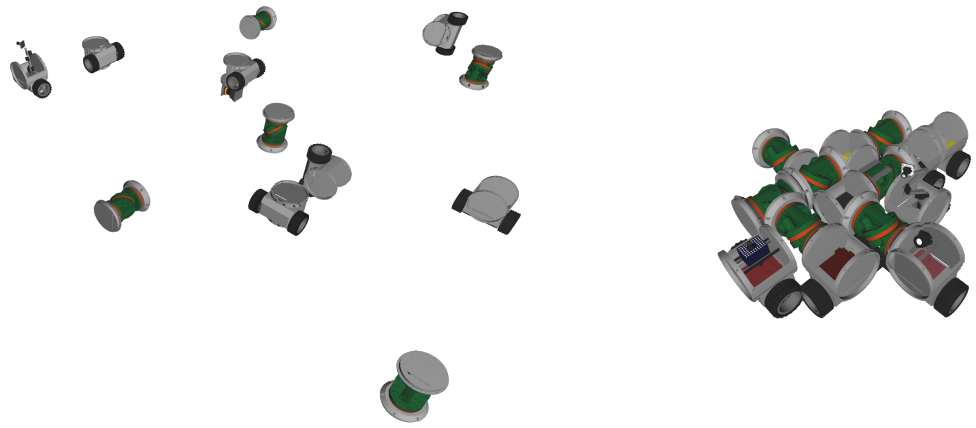
Tabelle 7.4: Ergebnisse der Optimierung mit gezielter Erzeugung neuer Individuen.

Die in Abbildung 7.9 dargestellten Individuen verdeutlichen bereits optisch den Unterschied zwischen den Startkonfigurationen. Eine Betrachtung der Bewer-

[2]In jeder Testreihe wurden zehn einzelne Testdurchläufe durchgeführt und alle dargestellten Messwerte wurden aus dem Mittelwert dieser Testdurchläufe gebildet.

tung (Fitnesswert) dieser Individuen bestätigt, dass gezielt generierte Individuen im Durchschnitt besser bewertet werden als zufällig erzeugte Individuen. Die durchschnittliche Fitness zufällig erzeugter Individuen erreicht lediglich $3,15$. Im Vergleich dazu liegt die Bewertung der gezielt generierten Individuen im Durchschnitt bei $1,9$.

Durch eine zielgerichtete Initialisierung kann dementsprechend eine deutliche Verbesserung bei der Erzeugung der Startpopulation erreicht werden. Diese ermöglicht eine schnellere Konvergenz zu einem Optimum ohne jedoch die Variation der Individuen zu sehr einzuschränken. Das Erzeugen neuer Individuen während der Optimierung wird entsprechend ebenfalls mit dem selben Algorithmus durchgeführt.



(a) Beispiel mit zufälliger Initialisierung. (b) Beispiel der zielgerichteten Initialisierung.

Abb. 7.9: Vergleich der Startpopulationen mittels unterschiedlicher Methoden zur Erzeugung neuer Individuen.

Laufzeit Betrachtung der Optimierung

Die Ausführung der evolutionären Optimierung wird sehr stark durch die Anzahl der Bausteine und Regeln beeinflusst. Insbesondere bei Regeln zwischen zwei Bausteintypen, können sehr viele Bedingungen entstehen (siehe Kapitel 7.2.2). Betrachtet man die Tests aus Tabelle 7.4, so wird ebenfalls deutlich, dass nicht nur die Anzahl der Regeln und Bausteine sondern auch deren Art einen Einfluss auf die Ausführungszeit hat. Je nach Art der abgeleiteten Regeln, werden entsprechende Fitness-Tests zur Bewertung genutzt, die verschiedene Laufzeiten mit sich bringen. Zudem werden Bausteine mit unterschiedlichen Geometrien, sogenannte Mehrfachbausteine, unterstützt. Für diese Mehrfachbausteine werden spezielle Strategien bei der Mutation angewendet, die auf Grund einer höheren Komplexität mehrere Iterationen bei der Positionierung

und Orientierung benötigen und somit längere Ausführungszeiten verursachen können.

Durch den Einsatz der in Kapitel 4.3.3 erläuterten Parallelisierung konnte eine deutliche Verbesserung der Ausführungszeit erreicht werden. Durch die parallele Ausführung auf mehreren Prozessorkernen konnten insbesondere die rechenintensiven Funktionen des evolutionären Algorithmus beschleunigt werden. Tabelle 7.5 vergleicht die Rechenzeiten der evolutionären Optimierung an Hand von vier Testfällen jeweils mit und ohne Parallelisierung. Der dargestellte Faktor beschreibt den Laufzeitvorteil der parallelen gegenüber der regulären Ausführung. Im Mittel ist eine Reduktion der Rechenzeit von mehr als 50 %, bei höherer Komplexität sogar mehr als 60 % möglich. Die beschriebene Parallelisierung auf mehrere Rechnerkerne ist sehr stark von der verwendeten Hardware abhängig. Alle Tests zur Evaluierung des Laufzeitverhaltens der evolutionären Optimierung wurden auf einem Intel(R) Core(TM) i7-4900MQ (2,80GHz) mit Hyper-Threading durchgeführt. Beim Einsatz künftiger Prozessoren mit einer größeren Anzahl von Kernen und höheren Taktraten ist eine weitere Minimierung der Ausführungszeit zu erwarten. Darüber hinaus besteht die Möglichkeit der Portierung auf aktuelle Grafikprozessoren mit deutlich mehr Kernen.

Testreihe	Test 1	Test 2	Test 3	Test 4
Anzahl Bausteine	8	16	32	64
Anzahl Regeln	18	34	38	82
Laufzeit ohne Parallelisierung [s]	0,569	5,314	30,141	217,894
Laufzeit mit Parallelisierung [s]	0,376	2,718	9,649	79,265
Laufzeitvorteil (Faktor)	1,51	1,96	3,12	2,75

Tabelle 7.5: Verbesserung der Laufzeiten der evolutionären Optimierung durch paralleler Ausführung auf mehreren Prozessorkernen.

Auswertung der evolutionären Optimierung

Die Auswertung des evolutionären Algorithmus erfolgt durch Analyse des Optimierungsverlaufes und der Bewertungsfunktion. An Hand von Beispielen wird die Bewertung (Fitness) einzelner Individuen in der Population zu verschiedenen Iterationsschritten untersucht. Zusätzlich wird die Zusammensetzung der Bewertungsfunktion aus den einzelnen Fitness Tests betrachtet.

Abbildung 7.10 visualisiert den Ablauf eines evolutionären Optimierungsprozess mit insgesamt 40 Schritten (Iterationen). Das am besten bewertete Individuum in der Population wird nach $2, 3, 5, 9, 20, 23$ und 24 Iterationen dargestellt. Die Bildsequenz veranschaulicht die Entwicklung der Bausteinanordnung von der Anfangskonfiguration (links oben) bis hin zu einer optimalen Endkonfiguration (rechts unten), die einen funktionalen Roboter unter Einhaltung aller Regeln

darstellt. In Abbildung 7.11 sind für jede Iteration die Bewertung der gesamten Population[3] (rot) und des jeweils besten Individuums (blau) aufgezeigt.

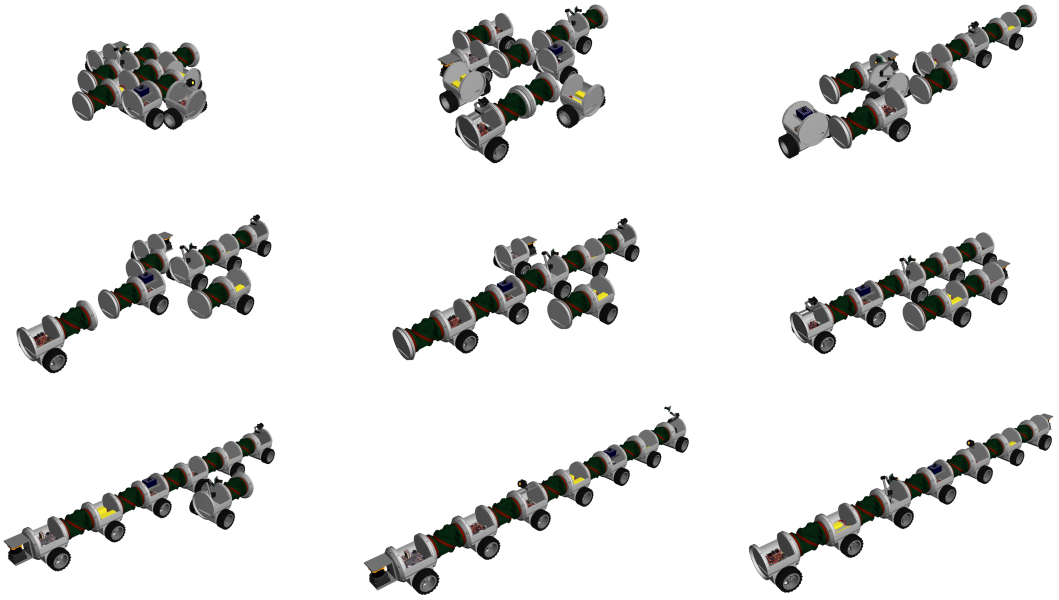


Abb. 7.10: Darstellung der jeweils besten Konfiguration (Individuum) während des Optimierungsprozess.

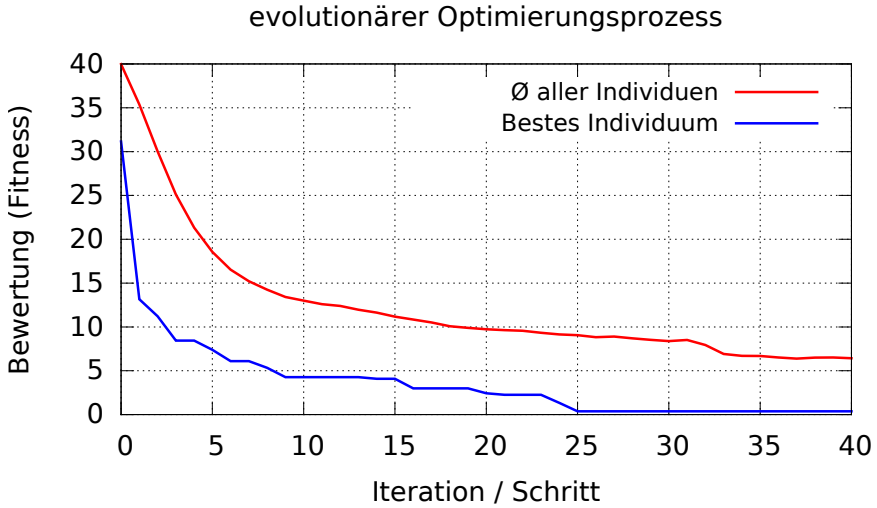


Abb. 7.11: Verlauf der Bewertung während der evolutionären Optimierung.

Bereits bei der Initialisierung ist zu erkennen, dass alle Bausteine auf Grund der gezielten Erzeugung von Individuen, bereits in ihrer Vorzugsrichtung orientiert sind. Somit sind alle absoluten Regeln zur Positionierung und Orientierung erfüllt. Dies spiegelt sich auch in den einzelnen Komponenten der Bewer-

[3]Die Bewertung der gesamten Population entspricht der durchschnittlichen Fitness aller Individuen in der Population.

tungsfunktion wieder. Tabelle 7.6 zeigt während des gesamten Optimierungsprozess keine Verletzung der absoluten Positionierung (*Abs Pos*) und Orientierung (*Abs Rot*) für das Individuum mit der jeweils besten Bewertung. Die Spalten in Tabelle 7.6 entsprechen den einzelnen Komponenten der Fitnessfunktion und stehen für *Cluster*, *Verbindungen*, *absolute Position*, *absolute Rotation*, *relative Entfernung*, *relative Orientierung*, *Blickfeld* sowie *Massenschwerpunkt*.

ID	Clust	Verb	Abs Pos	Abs Rot	Rel Entf	Rel Ori	FoV	CoM
0	0	30,43	0	0	0,632	0	0	0,299
2	7,99	2,516	0	0	0,632	0	0	0,281
3	4,99	2,435	0	0	0,864	0	0	0,543
5	5,99	0,435	0	0	0,864	0	0	0,351
9	2,95	0,361	0	0	0,864	0	0	0,358
20	1,98	0,339	0	0	0	0	0	0,442
23	1,86	0,339	0	0	0	0	0	0,213
24	0	0,318	0	0	0,864	0	0	0,615
40	0	0,318	0	0	0	0	0	0,221

Tabelle 7.6: Fitness der einzelnen Teilkomponenten der Bewertungsfunktion für die in Abbildung 7.10 dargestellten Iterationsschritte (*ID*).

Bei genauerer Betrachtung der Fitness der gesamten Population fällt auf, dass diese am Anfang sehr schnell fällt. Der Grund dafür ist die rapide Verbesserung der Verbindungen zwischen den Bausteinen auf Kosten der dafür ansteigenden Clusterverstöße. Erst nach Schritt 24 werden mehr Individuen durch Mutationen erzeugt, die aus der vorgegeben Anzahl an Clustern besteht und somit verbessert sich ebenfalls die Bewertung der gesamten Population stetig.

Zwischen den Schritten 23 und 25 ist in Abbildung 7.11 ein deutlicher Sprung der Bewertung des besten Individuums von $2,257$ auf $0,374$ zu beobachten. Dieser Sprung ist ebenfalls in Abbildung 7.10 zwischen den letzten drei Bildern zusehen und lässt sich durch eine sprunghafte Verbesserung der Cluster Bewertung erklären (siehe Tabelle 7.6). Bei allen Konfiguration vor dem Sprung stimmt die Anzahl der vorhandenen Cluster nicht mit der Vorgabe überein[4].

7.2.4 Leistungsfähigkeit des gesamten Syntheseprozess zur Konfigurationsoptimierung

Dieser Abschnitt betrachtet die zuvor einzeln getesteten Komponenten des Syntheseprozesses als Gesamtsystem und bewertet dessen Leistungsfähigkeit. Mit Hilfe des entwickelten Benutzerwerkzeugs CARD (siehe Kapitel 4.4.1) werden

[4]In diesem Beispiel wurde die Clusterzahl auf 1 festgelegt.

die Zwischen- und Endergebnisse der jeweiligen Optimierungsschritte auf Basis von drei Testszenarien betrachtet. Die Eingabedaten der einzelnen Tests für das CARD-Werkzeug sind in Tabelle 7.7 aufgeführt.

Szenario	Mission	Gelände	Einsatzdauer	Einsatzgebiet
Test 1	Gefahrstoffsuche	unstrukturiert	30 min	60 m^2
Test 2	Inspektion	strukturiert	120 min	600 m^2
Test 3	Personensuche	unwegsam	60 min	160 m^2

Tabelle 7.7: Test Szenarien zur Evaluierung des Syntheseprozess an Hand von Anwendungsbeispielen.

Testszenario 1: Gefahrstoffsuche in unstrukturiertem Gelände

In diesem Szenario wurden die Tabelle 7.7 gelisteten Parameter als Benutzereingabe für das CARD Werkzeug verwendet. Mit Hilfe der spezifizierten Rahmenbedingungen der Aufgabe wurden im **ersten Schritt** des Syntheseprozess die im folgenden aufgelisteten globalen Regeln festgelegt:

Globale Regeln

- *Cluster:* 1
- *Anzahl Verbindungszahl:* maximal
- *Massenschwerpunkt:* zentral
- *Konfigurationsraum:* $[50, 50, 1]$ (Flach)
- *Einsatzgebietes:* 60 m^2
- *Einsatzdauer:* 30 min

Zusätzlich wurden basierend auf den Missionsparametern die folgenden Anforderungen an Ressourcen, Komponenten oder speziellen Bausteintypen ausgewählt:

Komponentenanforderungen

- *1x id10:* Tiefenbild Kamera SwissRanger 3000
- *1x id13:* Kamera SD
- *1x id14:* Kamera HD
- *1x id15:* Manipulator Arm (5 Achsen)
- *1x id16:* Kamera Explosionssicher

Schritt 2: Aus den spezifizierten Anforderungen wurden die im Folgenden aufgelisteten Bausteine ausgewählt und instanziiert. Für die fünf Komponentenanforderungen wurden jeweils die Bausteine mit den entsprechenden Komponenten ausgewählt. Zusätzlich sind noch sechs Gelenk- und ein Batteriebaustein aufgelistet. Diese wurden automatisch ausgewählt um den Ressourcenbedarf der anderen Bausteine zu decken. Batterie Bausteine werden entsprechend dem Energiebedarf aller Bausteine in Abhängigkeit von der Missionsdauer ausgewählt. Bei verschiedenen Varianten (z. B. unterschiedlichen Batterie Kapazitäten) wird eine minimale Anzahl an Bausteinen mit der minimal nötigen Menge an Ressourcen gewählt. Die Instanziierung der Gelenkbausteine erfolgt auf Basis einer entsprechenden Ressource, die von jedem Antriebsbaustein benötigt wird.

Baustein Instanzen

- *6x id100:* KAIRO 3 Gelenk Baustein
- *1x id103:* KAIRO 3 Batterie Baustein groß
- *1x id109:* KAIRO 3 Tiefenbild Kamera Baustein
- *1x id111:* KAIRO 3 Kamera Baustein SD
- *1x id112:* KAIRO 3 Kamera Baustein HD
- *1x id113:* KAIRO 3 Manipulator Arm Baustein
- *1x id115:* KAIRO 3 Kamera Baustein Explosionssicher

Schritt 3: Basierend auf den zuvor spezifizierten globalen Regeln und den ausgewählten Bausteinen wurden die spezifischen, relativen und globalen Regeln durch den Inferenzschritt automatisch abgeleitet (siehe Kapitel 4.3.2). Auf Grund der Vielzahl an Regeln wird im Folgenden lediglich ein Auszug der wichtigsten Regeln dargestellt. Für alle Bausteine mit einem optischen Sensor wurde aus der entsprechenden Spezifikation im Bausteinkatalog eine Sichtkegelbedingung mit einem vorgegebenen Öffnungswinkel erstellt. Darüber hinaus wurden für alle Antriebsbausteine absolute Orientierungsbedingungen hinzugefügt, um sicherzustellen, dass die Räder im globalen Koordinatensystem korrekt auf den Boden ausgerichtet sind.

Baustein-spezifische Regeln

- *Sichtkegel 150°:* KAIRO 3 Tiefenbild Kamera Baustein
- *Sichtkegel 170°:* KAIRO 3 Kamera Baustein SD
- *Sichtkegel 170°:* KAIRO 3 Kamera Baustein HD
- *Sichtkegel 170°:* KAIRO 3 Manipulator Arm Baustein
- *Sichtkegel 170°:* KAIRO 3 Kamera Baustein Explosionssicher
- *Absolute Orientierung* KAIRO 3 Batterie Baustein groß
- *Absolute Orientierung* KAIRO 3 Tiefenbild Kamera Baustein

- *(...)*

Neben den Bedingungen für einzelne Bausteine werden im Inferenz Schritt insbesondere Abhängigkeiten die zwischen mehreren Bausteinen wirken abgeleitet. Im aufgezeigten Testszenario beinhaltet der *KAIRO 3 Kamera Baustein SD* neben einer Kamera auch eine Beleuchtungseinheit. Andere Sensoren wie die *Tiefenbild Kamera* sind entsprechend empfindlich gegenüber direkter Lichteinstrahlung. Mit Hilfe von relativen Orientierungsregeln kann sichergestellt werden, dass zwei Bausteine mit den entsprechenden Komponenten nicht aufeinander ausgerichtet werden. Die Bedingungen in der folgenden Tabelle verhindern, dass die Sensoren des *Tiefenbild Kamera Baustein*, *Kamera Baustein HD* und *Kamera Baustein Explosionssicher* nicht in Richtung der Beleuchtung des *Kamera Baustein SD* zeigen.

Baustein-relative Regeln

- *Relative Orientierung (Lichtempfindlichkeit):*
  Tiefenbild Kamera Baustein -> Kamera Baustein SD
  Kamera Baustein HD -> Kamera Baustein SD
  Kamera Baustein Explosionssicher -> Kamera Baustein SD

Schritt 4: Auf Grundlage der ausgewählten Bausteine und der abgeleiteten Bedingungen wird im letzten Schritt des Syntheseprozess die Anordnung der Bausteine zu einem Robotersystem, wie in Kapitel 4.3.3 beschrieben, durchgeführt. Die evolutionäre Optimierung liefert nach 26 Iterationen die in Abbildung 7.12 dargestellte Konfiguration mit einer Fitness von $0,319023$. Es lässt sich deutlich

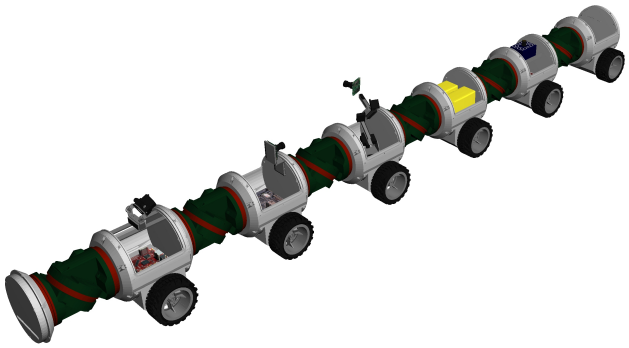


Abb. 7.12: Ergebnis des Syntheseprozess für die Suche von Gefahrstoffen in unstrukturiertem Gelände am Beispiel des RMRS KAIRO 3.

die schlangenartige Morphologie aus insgesamt 12 Bausteinen erkennen, welche in der vorliegenden Form besonders für unstrukturiertes Gelände wie Grasflächen, Wald und flache Steine geeignet ist. Die in den Bausteinen enthaltenen Komponenten stellen eine grundlegende Ausstattung für den Einsatz zur Suche nach Gefahrstoffen dar. Weitere individuelle Bausteine z. B. Sensoren für bestimmte Gift- oder Sprengstoffe können durch den Benutzer während der Synthese hinzugefügt werden.

Die explosionssichere Kamera ermöglicht den Betrieb des Roboters z. B. nach einer gezielten oder unbeabsichtigten Gefährdungsexplosion. Der Manipulatorarm wird zur Probenahme von giftigen Substanzen oder in Kombination mit einer Kamera zur gezielten Untersuchung und Gefahrenanalyse eingesetzt. Die Betriebsdauer von mindestens 30 min wird durch die insgesamt 8000 mAh fassenden Batterien sichergestellt. Bei einem maximalen Verbrauch von 8390 mAh und einer minimalen Restmenge von 10 % reichen die Batterien für mehr als 50 min. Alle zuvor spezifizierten Rahmenbedingungen der Mission sind somit erfüllt. Die Funktionsweise des Roboters kann durch eine zusätzliche Simulation verifiziert werden.

Testszenario 2: Strukturierte Inspektion und Wartung

Im zweiten Szenario wurden die Eingabeparameter (Tabelle 7.7) für eine Mission zur Inspektion und Wartung in einer strukturierten Umgebung verändert. Die globalen Regeln nach dem **ersten Syntheseschritt** wurden demnach wie im Folgenden aufgelistet, abgeleitet. Insbesondere ist zu erkennen, dass sich die Anzahl der Cluster auf drei erhöht hat. Eine Inspektion mit drei separaten Robotern erscheint bei einem Einsatzgebiet von 600 m^2 in einer Zeit von 120 min effektiv. Entsprechend der Anzahl an Robotern wurde ebenfalls die Art und Menge der für eine Inspektion notwendigen Komponenten automatisch angepasst.

Globale Regeln

- *Cluster:* 3
- *Anzahl Verbindungszahl:* maximal
- *Massenschwerpunkt:* zentral
- *Konfigurationsraum:* $[50, 50, 1]$ (Flach)
- *Einsatzgebietes:* 600 m^2
- *Einsatzdauer:* 120 min

Komponentenanforderungen

- *1x id8:* Laserscanner Hokuyo UTM-30LX
- *1x id10:* Tiefenbild Kamera SwissRanger 3000
- *3x id14:* Kamera HD
- *3x id15:* Manipulator Arm (5 Achsen)

Schritt 2: Durch die Bausteinauswahl wurden insgesamt 22 Bausteine an Hand der globalen Regeln und Komponentenanforderungen instanziiert. Zur Realisierung von drei eigenständigen Robotern die jeweils eine Komponente *Kamera HD*

und *Manipulator Arm* enthalten sowie insgesamt einen *Laserscanner Hokuyo UTM-30LX* und eine *Tiefenbild Kamera SwissRanger 3000* wurden die im Folgenden aufgelisteten Bausteine ausgewählt. Neben den entsprechenden Bausteinen zur Erfüllung der Komponentenanforderungen wurden zehn Gelenk- und vier Batteriebausteine erzeugt um die notwendigen Ressourcen bereit zu stellen.

Baustein Instanzen

- *10x id100:* KAIRO 3 Gelenk Baustein
- *4x id103:* KAIRO 3 Batterie Baustein groß
- *1x id107:* KAIRO 3 KaRoLa 3D Laserscanner Baustein
- *1x id109:* KAIRO 3 Tiefenbild Kamera Baustein
- *3x id112:* KAIRO 3 Kamera Baustein HD
- *3x id113:* KAIRO 3 Manipulator Arm Baustein

Schritt 3: Die im dritten Schritt durch die Regelinferenz abgeleiteten Bedingungen sind im Folgenden aufgeführt. Regeln für gleiche Bausteintypen wurden aus Gründen der Übersichtlichkeit zusammengefasst zählen jedoch als eigenständige Bedingungen.

Baustein-spezifische Regeln

- *Sichtkegel 180°:* KAIRO 3 KaRoLa 3D Laserscanner Baustein
- *Sichtkegel 150°:* KAIRO 3 Tiefenbild Kamera Baustein
- *Sichtkegel 170°:* KAIRO 3 Kamera Baustein HD (3x)
- *Sichtkegel 170°:* KAIRO 3 Manipulator Arm Baustein (3x)
- *Absolute Orientierung* KAIRO 3 Batterie Baustein groß (4x)
- *Absolute Orientierung* KAIRO 3 KaRoLa 3D Laserscanner Baustein
- *(...)*

Schritt 4: Die evolutionäre Optimierung in Testszenario 2 erreicht nach 47 Iterationen und einer Laufzeit von 44,7 s das globale Optimum bei einer Fitness von $0,332782$. Die beste Konfiguration bestehend aus den in Abbildung 7.13 dargestellten drei Robotern. Es ist deutlich zu sehen, dass die 22 Bausteine möglichst gleichmäßig auf die drei Roboter aufgeteilt wurden. Diese Konfiguration eignet sich auf Grund der Bausteinverteilung besonders gut zum parallelen Inspizieren bzw. Durchführen von Wartungsarbeiten in sehr großen Einsatzgebieten.

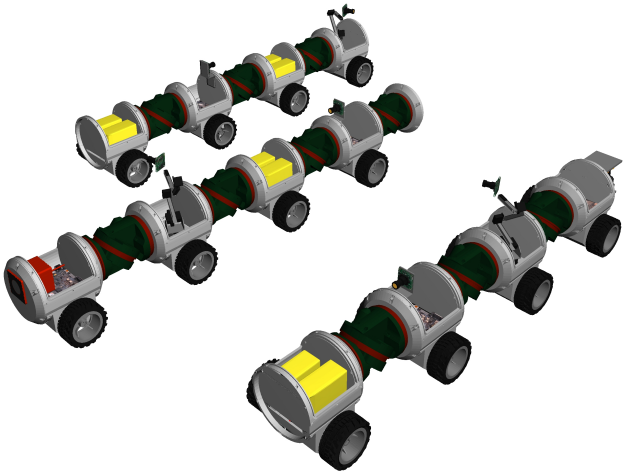


Abb. 7.13: Ergebnis des Syntheseprozess für die Inspektion und Wartung in strukturieren Umgebungen am Beispiel des RMRS KAIRO 3.

Testszenario 3: Suche nach Personen in unwegsamem Gelände

Testszenario 3 verwendet die in Tabelle 7.7 gelisteten Parameter als Eingabe. Im **ersten Schritt** wurden mit Hilfe der spezifizierten Rahmenbedingungen der Aufgabe die im folgenden aufgelisteten globalen Regeln festgelegt:

Globale Regeln

- *Cluster:* 1
- *Anzahl Verbindungszahl:* maximal
- *Massenschwerpunkt:* zentral
- *Konfigurationsraum:* $[50, 50, 1]$ (Flach)
- *Einsatzgebietes:* 160 m^2
- *Einsatzdauer:* 60 min

Zusätzlich wurden basierend auf den Missionsparametern die folgenden Anforderungen an Ressourcen, Komponenten oder speziellen Bausteintypen ausgewählt:

Komponentenanforderungen

- *1x id8:* Laserscanner Hokuyo UTM-30LX
- *1x id9:* Infrarot Kamera (Wärmebild)
- *1x id14:* Kamera HD

Bausteintyp Anforderungen

- *2x Kreuzbaustein:* KAIRO 3 Kreuz Baustein

Schritt 2: Basierend auf den Anforderungen des ersten Syntheseschrittes wurden insgesamt 17 Bausteine automatisch ausgewählt. Einen spezieller Baustein mit vier Schnittstellen (*KAIRO 3 Kreuz Baustein*) ermöglicht es komplexere Konfigurationen zu erzeugen, indem die Morphologie von einem schlangenartigen zu einem salamanderartigen Robotersystem geändert wird.

Baustein Instanzen

- *8x id100:* KAIRO 3 Gelenk Baustein
- *2x id101:* KAIRO 3 Batterie Baustein klein
- *2x id103:* KAIRO 3 Batterie Baustein groß
- *1x id107:* KAIRO 3 KaRoLa 3D Laserscanner Baustein
- *1x id108:* KAIRO 3 Infrarot Kamera Baustein
- *1x id112:* KAIRO 3 Kamera Baustein HD
- *2x id114:* KAIRO 3 Kreuz Baustein

Schritt 3: Aus den in Schritt 2 selektierten Bausteinen und den globalen Bedingungen wurden, wie in Szenario 2, Regeln zur Anordnung der Bausteine abgeleitet.

Schritt 4: Abbildung 7.14 zeigt die für besonders raues und unwegsames Gelände optimierte Konfiguration. Aufgrund der beiden *KAIRO 3 Kreuz Bausteine* weist diese Konfiguration eine Morphologie auf, die einem Salamander oder eine Eidechse ähnlich ist. In dieser Konfiguration ermöglicht das Anheben der Antriebsmodule entlang der Mittelachse die Realisierung einer laufenden oder kriechenden Fortbewegung mit KAIRO 3. Werden dagegen die vier äußeren Antriebsmodule angehoben, kann wie bisher eine radgetriebene Fortbewegung durchgeführt werden.

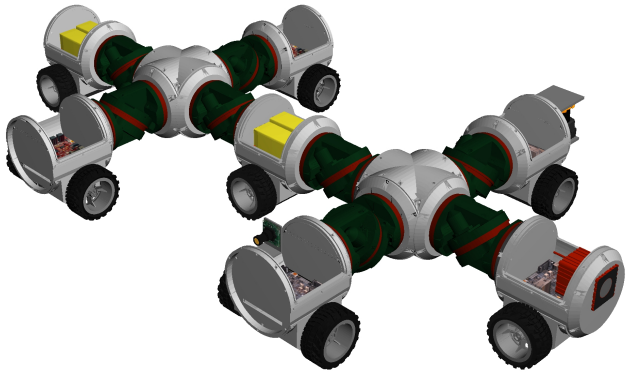


Abb. 7.14: Ergebnis des Syntheseprozess für eine Such- und Rettungsmission in einem sehr unwegsamen Gelände mit Geröll und Schutt.

Die in Kapitel 3.4 präsentierte Schnittstelle erlaubt es KAIRO 3 grundsätzlich, sich selbst zu rekonfigurieren. Auf diese Weise kann die Konfiguration während einer Mission angepasst werden, z. B. wenn unvorhergesehene Situationen oder Umweltbedingungen auftreten. Besteht eine Konfiguration aus mehreren eigenständigen Robotern, wie in Testszenario 2, können sich die einzelnen Roboter untereinander verbinden, um ein größeres Hindernis zu überwinden (siehe Kapitel 6).

Testszenario 4: Generierung und Optimierung modularer Satelliten

Wie bereits in Kapitel 4.4 gezeigt, kann die Synthese für rekonfigurierbare Robotersysteme auch auf andere Domänen und Anwendungsfelder übertragen werden. Durch den Austausch des Komponenten- und Bausteinkatalogs können Konfigurationen für eine Vielzahl an modularen Systemen generiert und optimiert werden. Zur Demonstration der allgemeinen Anwendbarkeit der in dieser Arbeit vorgestellten Synthese und der implementierten Verfahren, wird in diesem Testszenario die Generierung und Optimierung modularer Satelliten evaluiert. Zu diesem Zweck wurde der in Anhang A.1.2 gezeigte Katalog für modulare Satellitenbausteinen und deren Komponenten entwickelt [73]. Auf der Grundlage dieses Katalogs und mit Hilfe des Computer-Aided-Satellite-Design (CASD) Werkzeug werden im Folgenden die Zwischenergebnisse der einzelnen Syntheseschritte diskutiert.

Für diesen Test wurden die globalen Regeln, Komponenten sowie spezifischen Bausteinanforderungen für eine Satellitenmission im geostationären Orbit (GEO) vom Anwender wie folgt spezifiziert:

Globale Regeln

- *Cluster:* 1
- *Anzahl Verbindungszahl:* maximal
- *Massenschwerpunkt:* zentral
- *Konfigurationsraum:* $[100, 100, 100]$

Komponentenanforderungen

- *1x id20:* Reaktionsrad (CMG)
- *1x id29:* Drallrad
- *5x id52:* Kaltgasantriebssystem
- *1x id63:* Trägheitsnavigationssystem (IMU)
- *1x id79:* Magnetorquer
- *1x id101:* Recheneinheit (CPU)
- *1x id126:* zentrale Steuerungseinheit (SMU)

- *2x id151:* Transponder

Bausteinanforderungen (spezifisch für die Mission)

- *1x id246:* Kernstruktur Baustein (2x2x2)
- *2x id422:* Parabolantenne (lang)
- *2x id432:* Antenne Small GEO

Schritt 2: Die Bausteinauswahl konnte aus den gestellten Anforderungen insgesamt 21 Satellitenbausteine instanziieren. Um die ausgewählten Bausteine mit Energie zu versorgen wurde an Hand der Ressourcenanforderungen der einzelnen Bausteine, automatisch ein Batteriebaustein hinzugefügt. Die Batteriebausteine eines Satelliten besitzen im Vergleich zu den Batteriebausteinen von KAIRO 3, die zusätzliche Ressource *Ladung*. Mit Hilfe dieser zusätzlichen Ressource ist es möglich entsprechend des Energiebedarfs des Satelliten und der Anzahl an Batteriebausteinen ebenfalls automatisch entsprechende Solargeneratoren auszuwählen. Neue Ressourcen können bei der Modellierung des Katalogs ohne Veränderung der Algorithmen definiert werden. Im diesem Testszenario wurden dementsprechend zwei zusätzliche Solargenerator Bausteine instanziiert.

Baustein Instanzen

- *1x id162:* Batterie Baustein
- *1x id170:* Reaktionsrad Baustein
- *1x id178:* Drallrad Baustein
- *5x id185:* Kaltgasantrieb Baustein
- *1x id216:* Trägheitsnavigationssystem (sehr genau)
- *1x id246:* Kernstruktur Baustein (2x2x2)
- *1x id252:* Magnetorquer Baustein
- *1x id268:* zentraler Steuerungsbaustein
- *1x id275:* Rechnerbaustein
- *2x id295:* Solargenerator Baustein
- *2x id338:* TM&TC Baustein (S-Band; Transponder+SSPA)
- *2x id422:* Parabolantennen Baustein
- *2x id432:* Antennenbaustein Small GEO

Schritt 3: Bei der Inferenz der Bedingungen zur Anordnung der Bausteine werden zunächst die zwei Baustein-relative Regeln betrachtet. Solargenerator Bausteine besitzen zur Vermeidung von Verschattungen eine Regel zur relativen Orientierung. Parabolantennen senden elektromagnetische Strahlung aus und sind gleichzeitig empfindlich gegen Strahlungen anderer Antennen. Daher wurde gemäß der Modellierung eine Bedingung zur Einhaltung eines minimalen relativen Abstandes zwischen den beiden Parabolantennen-Bausteine abgeleitet.

Baustein-relative Regeln

- *Relative Orientierung (Verschattung):*
  Solargenerator Baustein 1 <-> Solargenerator Baustein 2

- *Relative min. Entfernung (Elektromagnetische Strahlung):*
  Parabolantennen Baustein 1 <-> Parabolantennen Baustein 2

Neben den Baustein-relativen Regeln wurden zusätzlich 21 Baustein-spezifische Regeln inferiert. Diese setzten sich aus 14 Sichtfeld Regeln und 7 absoluten Orientierungsregeln verschiedener Bausteine wie den Kaltgasantrieben, Antennen, Transpondern, usw. zusammen. Auf Grund der Vielzahl dieser Regeln wird auf eine ausführliche Auflistung verzichtet.

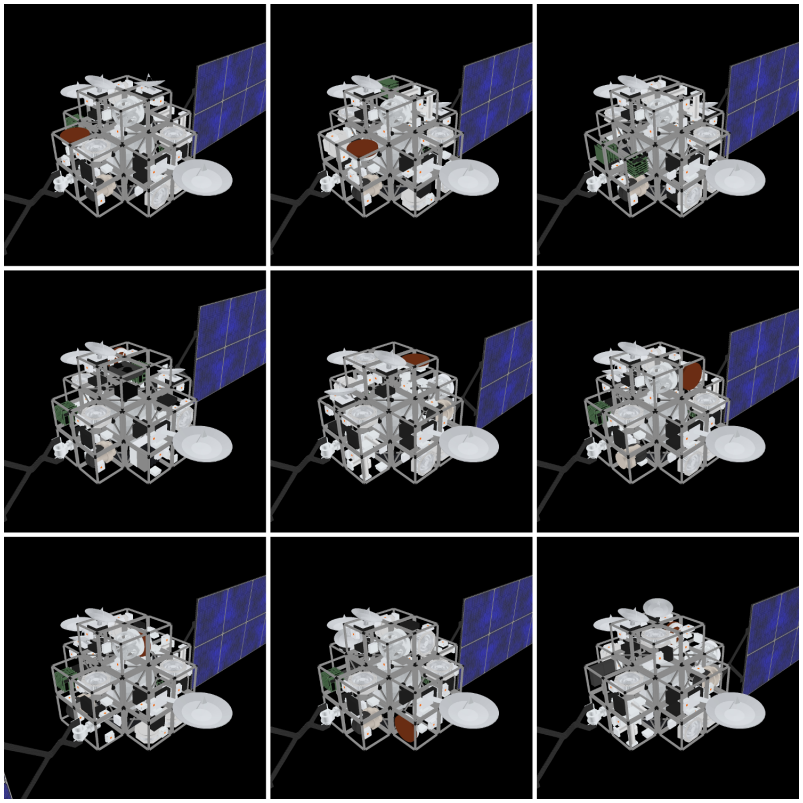


Abb. 7.15: Darstellung der am Besten bewerteten Individuen (Baustein Konfigurationen) von links oben nach rechts unten.

Schritt 4: Die evolutionäre Optimierung findet nach 58 Iterationen und einer Laufzeit von 6,1 s eine optimale Lösung mit einer Fitness von $0,946391$. Die besten 9 Konfigurationen sind in Abbildung 7.15 dargestellt. Der grundlegende Aufbau des GEO Satelliten ist bei allen Konfigurationen gleich, lediglich die Anordnung einzelner Antennen und die Verteilung von Bausteinen innerhalb des Satelliten variieren. Ebenfalls die Position der Solargenerator Bausteinen ist bis auf eine minimale Variation der Positionen gleich.

7.2.5 Zusammenfassung

Zusammenfassend lässt sich sagen, dass die Ausführung der Bausteinauswahl sowie der Regelinferenz als unabhängige Schritte den Vorteil einer festen Anzahl an Bausteinen während des Syntheseprozess bietet. Dies führt im Vergleich zu einer variablen Bausteinanzahl zu einer erheblichen Reduzierung des Suchraums und damit zu einer effizienteren Ausführung. Die Auswertung der drei Verfahren zur Auswahl von Bausteinen (siehe Kapitel 7.2.1) hat gezeigt, dass die Formulierung als ganzzahliges lineares Programm bei der Suche nach der optimalen Lösung am effizientesten ist. Für den zweiten Syntheseschritt, der Regelinferenz, wurde der problemspezifische Reasoner aufgrund der etwas schnelleren Laufzeit und der einfacheren Modellierung bevorzugt. Durch die feste Anzahl an Bausteine und die Diskretisierung des Suchraums zur Optimierung der Bausteinanordnung konnte ein Laufzeitvorteil gegenüber bereits bestehender Verfahren erreicht werden. Die bei der Generierung und Optimierung von Roboterkonfigurationen erzielten Ausführungszeiten ermöglichen deren Einsatz während des Betriebs des Robotersystems. Darüber hinaus ermöglicht der in dieser Arbeit vorgestellte Ansatz für die Synthese modularer Roboterkonfigurationen die Generierung von Systemen, die aus mehreren unabhängigen Teilsystemen bestehen, wie z. B. rekonfigurierbare Multi-Roboter-Systeme (RMRS).

7.3 Konfigurationsabhängige Adaption der Lokomotion

7.3.1 Biologisch inspirierte Fortbewegungsarten

Zur Evaluierung der für KAIRO 3 implementierten biologisch-inspirierten Fortbewegungsarten *Omega-Lokomotion* und *Raupenartige Lokomotion*, wurden mehrere Tests sowohl in der Simulation als auch mit KAIRO 3 durchgeführt. Zusätzlich zu den Tests unter Laborbedingungen wurde die Fortbewegung von KAIRO 3 ebenfalls im Feldeinsatz untersucht. Die durchgeführten Tests und die daraus abgeleiteten Resultate werden in den folgenden Abschnitten diskutiert.

Simulation zur Evaluation der Skalierbarkeit

Mit Hilfe der Simulationsumgebung von KAIRO 3, konnte zunächst die grundlegende Funktionalität der implementierten Lokomotionsarten verifiziert werden. Darüber hinaus wurde in der Simulation ein Roboter bestehend aus insgesamt siebzehn Modulen (neun Antriebsmodule und acht Gelenkmodule) getestet, um die Skalierbarkeit der implementierten Methoden nachzuweisen.

Abbildung 7.16 zeigt eine Bildsequenz der einzelnen Schritte der *Omega-Lokomotion* aufgenommen in der MCA2 3-D Simulationsumgebung. Im ersten

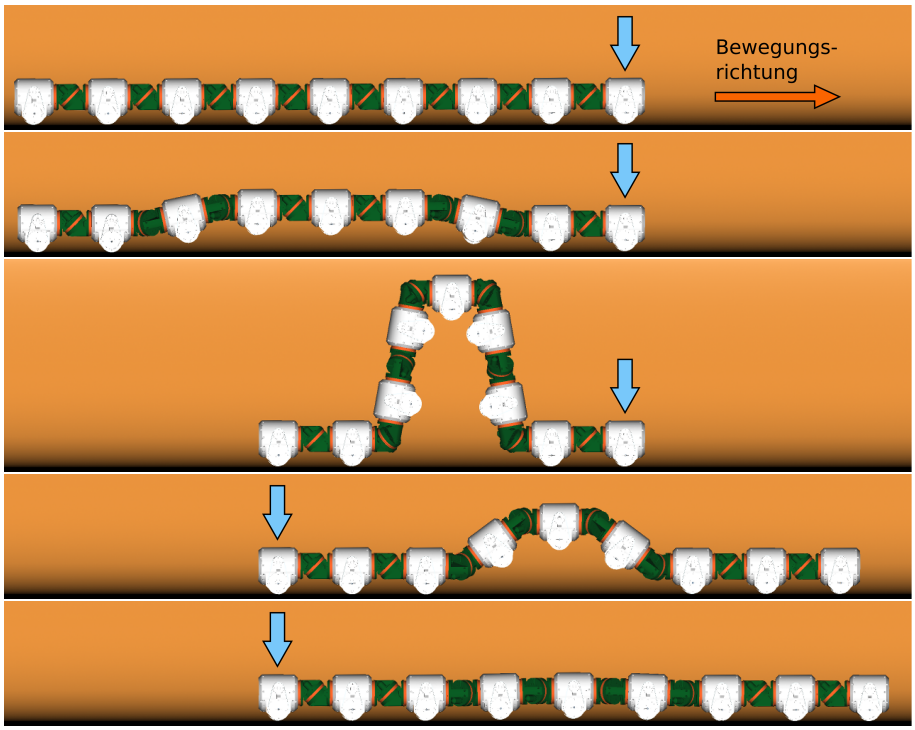


Abb. 7.16: Simulation der Omega-Lokomotion mit 17 Modulen [87].

Bild ist die Ausgangsposition des Roboters dargestellt, bei der das erste Modul

(Kopfmodul) am Boden befestigt ist. Das zweite Bild zeigt den Beginn der Omega-Bewegung durch Anheben der mittleren Module. Nachdem das mittlere Modul vollständig angehoben und das letzte Modul nachgezogen wurde, wird die Fixierung des Kopfmoduls auf das letzte Modul geändert (siehe dritte Abbildung). Das vierte Bild zeigt, wie die mittleren Module abgesenkt und das erste Modul vorwärts bewegt werden. Der Omega-Bewegungszyklus ist abgeschlossen, sobald das mittlere Modul den Boden erreicht hat und der Bewegungszyklus erneut beginnt. Das letzte Bild verdeutlicht, dass der Roboter in einem Bewegungszyklus um ein Drittel seiner Gesamtlänge vorwärts bewegt wurde.

Neben der Omega Fortbewegung wurde ebenfalls die raupenartige Lokomotion und insbesondere die geometrische und nummerische Berechnung der Ausgleichsbewegung (Kapitel 5.2.1) zunächst in der Simulation überprüft. Die Bildsequenz in Abbildung 7.17 zeigt den Ablauf der raupenartigen Fortbewegung beginnend mit der Ausgangsposition im ersten Bild. Die Bewegungsrichtung des

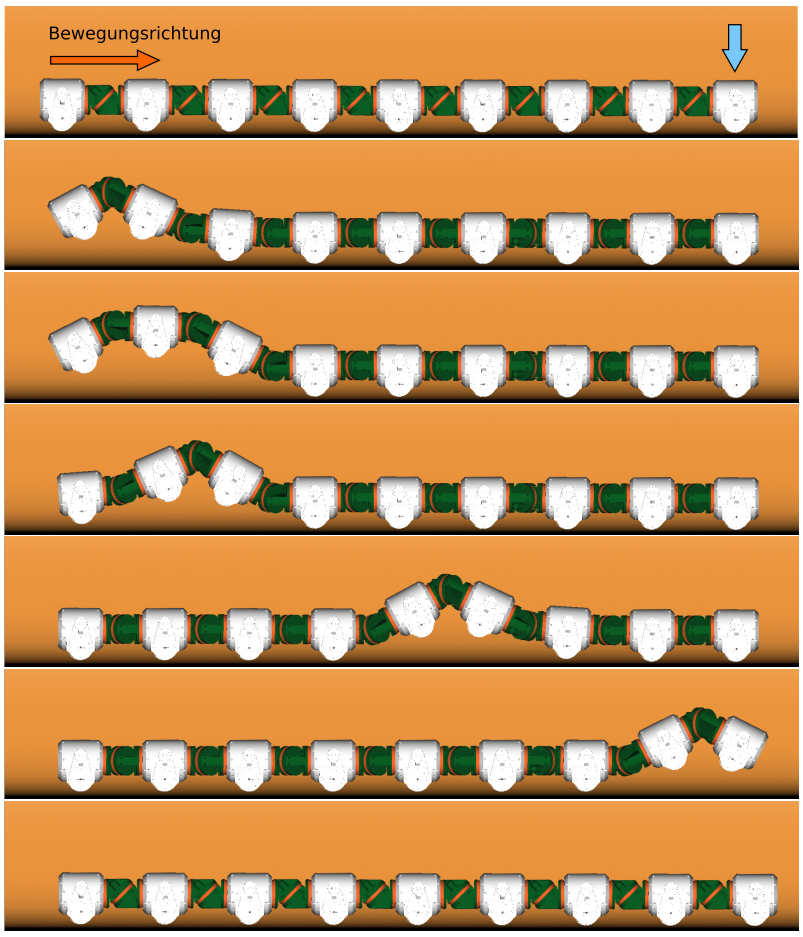


Abb. 7.17: Simulation der raupenartigen Fortbewegung mit 17 Modulen [87].

Roboters verläuft von links nach rechts. Zu Beginn der Bewegung wird entsprechend dem in Kapitel 5.2 definierten Zustandsautomaten das letzte Modul angehoben. Die weiteren Bilder zeigen abwechselnd die beiden Schritte *Nächstes*

Anheben, *Vorheriges Absenken*. Aufgrund der vielen Einzelschritte, wurde die Bildsequenz in der Mitte abgekürzt. Nachdem sich die Bewegung von Anheben und Absenken in Wellenform durch den Roboter zum ersten Modul ausgebreitet hat, wird am Ende das erste Modul abgesenkt. Beim Vergleich des ersten und letzten Bildes ist zu erkennen, dass sich der Roboter um eine kleine Strecke nach vorne bewegt hat. Der abgeschlossene raupenartige Bewegungszyklus kann nun von neuem beginnen.

Mit Hilfe der oben genannten Simulation beider Bewegungsarten konnte gezeigt werden, dass die Implementierung auf Basis der vorhandenen Kinematik und virtuellen Schiene es dem Roboter ermöglichen, sich wie eine Raupe zu bewegen. Insbesondere die geometrische und numerische Berechnung der Ausgleichsbewegung der raupenartigen Fortbewegung wurde durch Erfassen und Vergleichen der Gelenkwinkel sowie der Höhen der einzelnen Module überprüft. Darüber hinaus wurden Tests mit unterschiedlichen Längen der simulierten Roboter durchgeführt und es wurde nachgewiesen, dass die jeweilige Implementierung mit der Roboterkonfiguration skaliert [87].

Bewertung der Omega-Lokomotion mit KAIRO 3

Zur Bewertung der *Omega-Lokomotion*, wurden verschiedene Tests mit dem Roboter KAIRO 3 in einer Laborumgebung durchgeführt (siehe Abbildung 7.18). Zur Messung der real zurückgelegten Entfernung und benötigten Zeit pro Be-

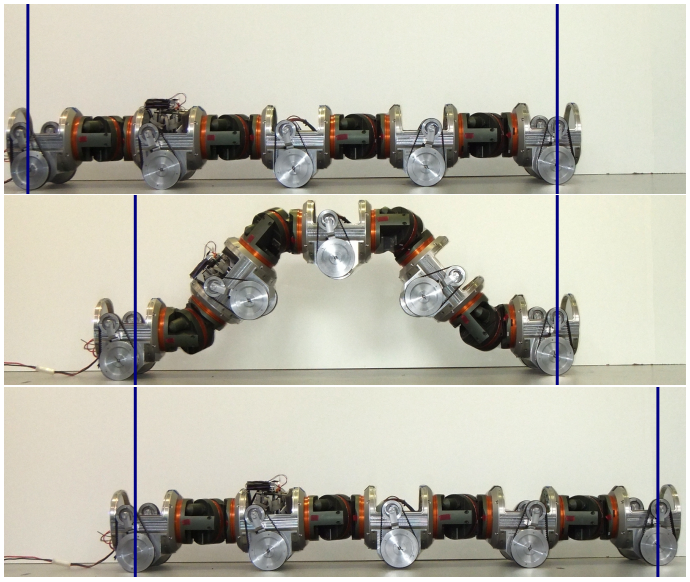


Abb. 7.18: Auswertung der Omega-Fortbewegung [87].

wegungszyklus, wurden in einem speziellen Versuchsaufbau, jeweils die Start- und Endpositionen markiert. Abbildung 7.18 verdeutlicht den Versuchsaufbau und das durchgeführte Experiment. Ausgehend von der Grundstellung des Roboters wurden insgesamt fünf Versuche durchgeführt. Bei jedem Versuch wurde

ein kompletter Bewegungszyklus mit den zwei Zuständen *Zusammenziehen* und *Ausstrecken* ausgeführt.

Zum Vergleich der gemessenen mit der theoretisch maximal zurückgelegten Entfernung, wurde die in Kapitel 5.1 vorgestellte Gleichung 5.1 zur Berechnung der Entfernung mmd pro Bewegungszyklus eingesetzt. Die mmd Funktion hängt von den Parametern n (Anzahl der Antriebsmodule von KAIRO 3), s (Abstand zwischen den Mittelpunkten zweier Gelenkmodule) und h_{max} (Maximale Höhe des mittleren Antriebsmodul) ab. Der Abstand zwischen den Gelenkmodulen s ist durch die Konstruktion festgelegt auf 330 mm. Die maximale Höhe des mittleren Moduls h_{max} wurde aus Sicherheitsgründen auf 270 mm beschränkt. Tabelle 7.8 vergleicht die Messwerte von fünf realen Messungen und deren Durchschnitt mit der theoretisch maximal zurückgelegten Entfernung. Die Messwerte schwan-

Messung	1	2	3	4	5	Ø	berechnet
Entfernung [mm]	275	275	285	285	280	281, 0	280, 5

Tabelle 7.8: Messwerte der jeweils zurückgelegten Entfernungen bei fünf aufeinander folgenden Bewegungszyklen der *Omega-Fortbewegung*.

ken auf Grund von Messfehlern und Ungenauigkeiten zwischen 275 mm und 285 mm. Insgesamt lässt sich jedoch erkennen, dass die tatsächlich zurückgelegte Entfernung und insbesondere der durchschnittliche Entfernungswert von 281 mm mit dem berechneten Wert von 280,5 mm übereinstimmt. Insgesamt hat sich der Roboter in fünf Omega Bewegungszyklen um 1405 mm nach vorne bewegt. Resultierend lässt sich sagen, das sich die *Omega-Lokomotion* insbesondere durch die Geschwindigkeit, besonders für den Einsatz in unebenem Gelände ausgezeichnet.

Bewertung der raupenartigen Lokomotion mit KAIRO 3

Zur Bewertung der *raupenartigen Lokomotion*, wurden ebenfalls verschiedene Tests mit KAIRO 3 in einer Laborumgebung durchgeführt. Wie bei der *Omega-Lokomtion*, wurden die real zurückgelegte Entfernung und die benötigte Zeit pro Bewegungszyklus in einem speziellen Versuchsaufbau gemessen. Abbildung 7.19 zeigt den Ablauf eines Bewegungszyklus, bei dem jeweils die Start- und Endpositionen markiert wurden und verdeutlicht die durchgeführten Experimente. Ein kompletter Bewegungszyklus besteht aus der wiederholten Ausführung der Zustände *Nächstes Anheben* und *Vorheriges Absenken*, wie im Zustandsautomat in Kapitel 5.3 dargestellt. Zur Initialisierung der Bewegung wurde der Zustand *Letztes Anheben* und am Ende der Bewegung *Letztes Absenken* ausgeführt. Insgesamt wurden sieben komplette Durchläufe ausgeführt und jeweils die zurückgelegte Strecke und die benötigte Zeit messen. Tabelle 7.9 zeigt die Entfernungsmesswerte, den Durchschnittswert 67,1 mm und vergleicht diese mit dem theoretischen Maximalwert mmd von 72,1 mm. Der mmd Wert kann mit

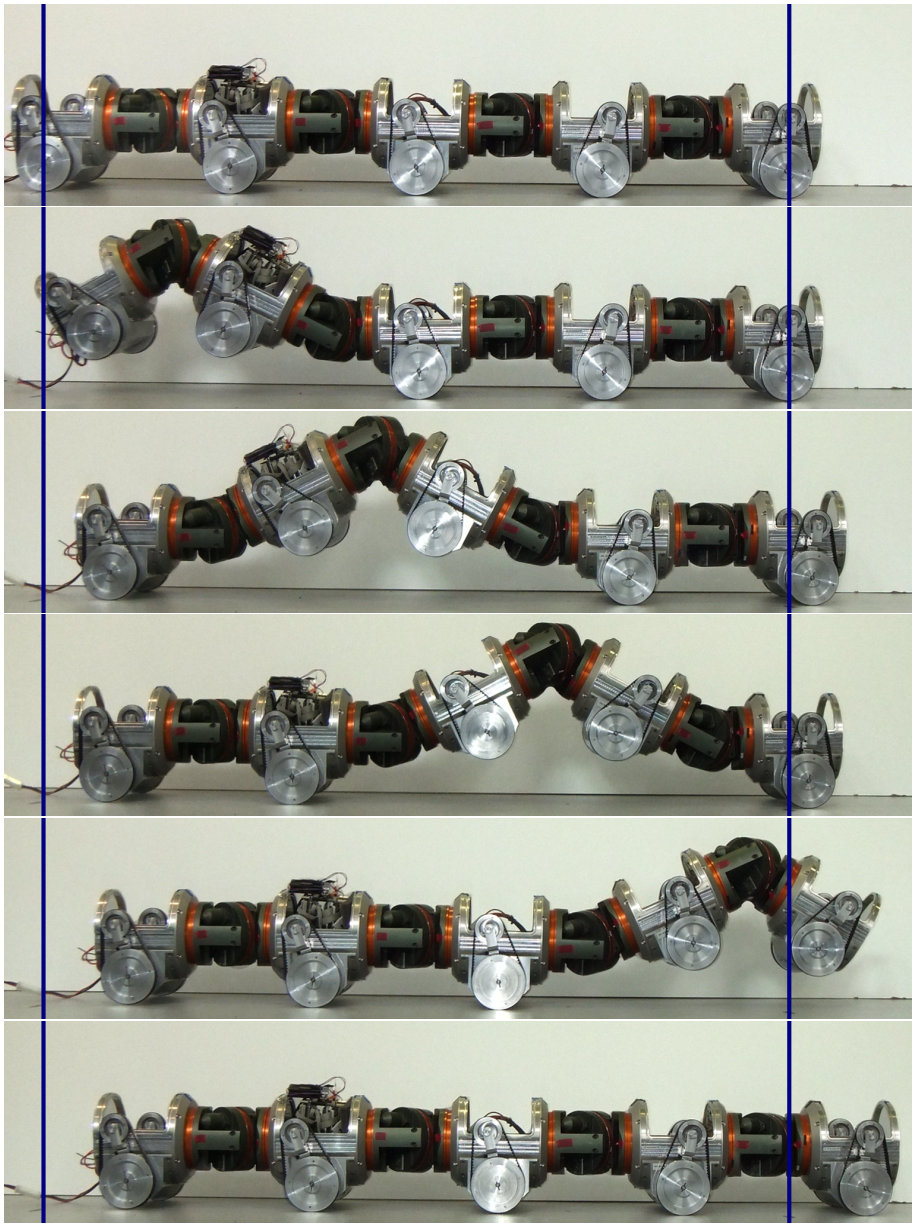


Abb. 7.19: Auswertung der raupenartigen Bewegung [87].

Messung	1	2	3	4	5	6	7	Ø	berechnet
Entfernung [mm]	70	65	65	70	75	65	60	67, 1	72, 1

Tabelle 7.9: Messwerte der jeweils zurückgelegten Entfernungen bei sieben aufeinander folgenden Bewegungszyklen der *raupenartigen Fortbewegung* [87].

der selben Gleichung 5.1 wie für die *Omega-Lokomotion* berechnet werden, jedoch mit den Parametern $n = 5$ (fünf Antriebsmodule) und $h_{max} = 150$ mm (maximale Aufrichthöhe eines Gelenkmoduls). Insgesamt bewegte sich KAIRO 3 in den sieben aufgezeichneten Bewegungszyklen der raupenartigen Fortbewegung um 480 mm vorwärts.

Die Differenz zwischen der berechneten maximalen Entfernung und der tatsächlich zurückgelegten Entfernung wird hauptsächlich durch die geringe Anzahl von Modulen verursacht[5]. Während des raupenartigen Bewegungszyklus entsteht ein kurzer Moment, in dem insgesamt drei Antriebsmodule keinen Kontakt zum Boden haben. Bei einer Konfiguration mit nur fünf Antriebsmodulen befinden sich also mehr Antriebsmodule in der Luft als am Boden, so dass KAIRO 3 kurzzeitig nach hinten kippt. Durch diese Kippbewegung ist die tatsächlich zurückgelegte Strecke kürzer als die maximal berechnete Strecke. Dieses Problem tritt bei einer Roboterkonfiguration mit mehreren Modulen nicht auf.

Vergleicht man die Implementierungen der *Omega-Lokomtion* und der *raupenartigen Lokomotion* auf KAIRO 3 mit den Bewegungsmodellen der biologischen Vorbilder, so lassen sich viele parallelen erkennen. Insbesondere die raupenartige Fortbewegung sieht auf Grund der implementierten Ausgleichsbewegung benachbarter Gelenkmodule, sehr natürlich und gleichmäßig aus. Aber es sind auch kleinere Unterschiede zu erkennen. Um die statische Stabilität des Roboters zu gewährleisten, wurde die *Omega-Fortbewegung* dadurch vereinfacht, indem der im biologischen Modell vorhandene dritte Zustand *Suchen* weggelassen wurde[6]. Ebenso wurde bei der raupenartigen Fortbewegung das Zusammenziehen und Ausdehnen durch Anheben und Abknicken der Gelenkmodule realisiert, da KAIRO 3 keine linearen Aktuatoren besitzt.

Feldversuche zur Evaluierung der Fortbewegungsarten mit KAIRO 3

Neben den Tests in der Simulation und Auswertung unter Laborbedingungen wurden zusätzlich Feldversuche mit KAIRO 3 durchgeführt. Die neu implementierten Fortbewegungsarten wurden auf verschiedenen Geländearten getestet, wie z. B. auf einer geteerten Straße, einem Sandweg und einer Grünfläche. Abbildung 7.20 zeigt beispielhaft, wie KAIRO 3 die *Omega-Fortbewegung* auf einer Grasfläche ausführt.

Bei der Auswertung wurde zunächst festgestellt, dass KAIRO 3 nicht über die unebene Grasfläche oder den sandigen Untergrund fahren kann. Auf Grund der geringen Bodenfreiheit rutschen einzelne Räder durch und der Roboter bleibt stecken. Die beiden neuen Bewegungsarten *Omega-Fortbewegung* und *raupenartige Fortbewegung* hingegen ermöglichen es KAIRO 3, sich über die Grasfläche und

[5] Zum Zeitpunkt der Auswertung konnte nur ein Roboter aus neun Modulen getestet werden, da nur vier der fünf gemeinsamen Module zur Verfügung standen.

[6] Im Zustand *Suchen* hebt eine Raupe ihren Kopf an, wobei sie sich lediglich an den hinteren Körpersegmenten am Boden festkrallt und die Umgebung tastend nach Hindernissen absucht

Abb. 7.20: Feldtest mit KAIRO 3 auf einer unstrukturierten Grasfläche [87].

über den sandigen Untergrund zu bewegen. Mit der *raupenartige Fortbewegung* konnte sich der Roboter sogar selbst befreien, nachdem der sich mit den Radantrieben festgefahren war.

Zur Bewertung der einzelnen Fortbewegungsmöglichkeiten wurden qualitative Analysen zum direkten Vergleich der *Omega*, der *raupenartigen* und der *radgetriebenen Fortbewegung* durchgeführt. Tabelle 5.1 aus Kapitel 5.4 zeigt eine Übersicht dieser Bewertung. Aufgrund der höheren Geschwindigkeit und der schlechten Performance auf unwegsamem Gelände eignet sich die radgetriebene Fortbewegung besonders für ebene Flächen wie Fußböden in Gebäuden oder asphaltierte Straßen. Bewegt sich der Roboter hingegen in unwegsamem Gelände, ist eine andere Art der Fortbewegung vorzuziehen. Im Gegensatz zur radgetriebenen Fortbewegung benötigt die raupenförmige Lokomotive keine Räder und ist daher für besonders unwegsames Gelände mit steinigem oder sandigem Boden geeignet. Der größte Nachteil der Raupenbewegung ist die langsame translatorische Geschwindigkeit.Alternativ kann die schnellere Omega Lokomotion verwendet werden, die sich für teilweise unstrukturiertes Gelände wie z. B. Grasflächen oder flache Steine eignet. Die Omega Lokomotion stellt somit einen Mittelweg zwischen radgetriebener und raupenmobiler Lokomotive dar.

Die vorliegenden Resultate zeigen, dass sowohl die Omega- als auch die Raupenbewegung eine gute Ergänzung zur radgetriebenen Fortbewegung darstellen, so dass sich der Roboter ebenfalls in unwegsamem und unstrukturiertem Gelände bewegen kann.

7.3.2 Unebenheitsmaß

Für die Evaluierung und Optimierung des in Kapitel 5.3.1 eingeführten *Unebenheitsmaß* R wurden verschiedene Tests bei unterschiedlichen Bodenbeschaffen-

heiten durchgeführt. Abbildung 7.21 zeigt vier Oberflächen, die in Labortests näher untersucht wurden.

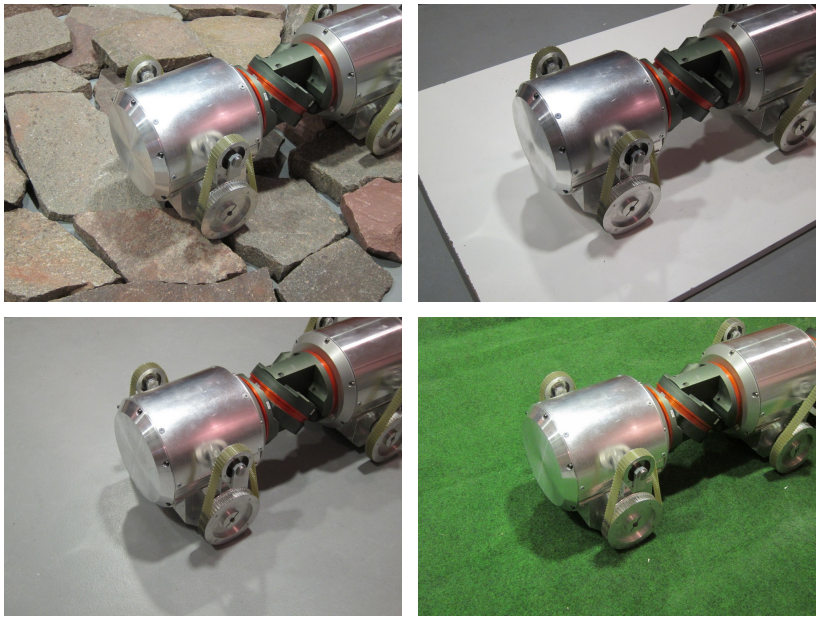


Abb. 7.21: Labortests wurden mit den vier Oberflächen flache Steine, glattes Brett, Industrieboden und Teppich durchgeführt.

Bei den Experimenten wurde der Roboter KAIRO 3 jeweils eine Strecke von mindestens 500 mm über die jeweilige Oberfläche bewegt. Während dessen wurden kontinuierlich alle Motorstromwerte und die berechneten Unebenheitswerte R aufgezeichnet. Für alle Oberflächen wurden mehrere Versuche mit Variationen der Fensterlänge (Δt)[7] durchgeführt. Durch die Variation der Zeitfensterlänge, kann die pro Zeitschritt zu berücksichtigende Anzahl an Messwerten (Motorstromwerte) beeinflusst werden. Eine kleinere Fensterlänge führt dabei zu einer sich schneller ändernden Unebenheit, ein längeres Fenster hingegen er gibt einen glatteren Verlauf. Gute Ergebnisse konnten in diesem Fall mit einer Fensterlänge $\Delta t = 300$ ms erzielt werden. Dies entspricht bei einer Messrate der Motorströme von 30 ms einer Anzahl von $n = 10$ Messungen in jedem Zeitfenster.

Abbildung 7.22 zeigt beispielhaft die Werte des *Unebenheitsmaß* R für die vier Oberflächen flache Steine, glattes Brett, Industrieboden und Teppich. Es ist deutlich zu sehen, dass die Werte für die weniger rauen Oberflächen (Holzbrett, Industrieboden und Teppich) sehr ähnlich und gleichmäßig niedrig ausfallen. Im Vergleich dazu sind die Messwerte beim Überfahren der flachen Steinen im Mittel deutlich höher, vor allem zeigen sich sehr ausgeprägte Amplituden. Dies lässt sich durch abrupte Änderungen der Motorströme erklären, die beim Auftreffen einzelner Räder an den Kanten der Steinen entstehen. Insgesamt lässt sich somit durch das *Unebenheitsmaß* R eine deutliche Unterscheidung zwischen glatten und rauen Oberflächen treffen.

[7] Die Fensterlängen 50 ms, 100 ms, 200 ms, 300 ms, 400 ms und 500 ms wurden untersucht.

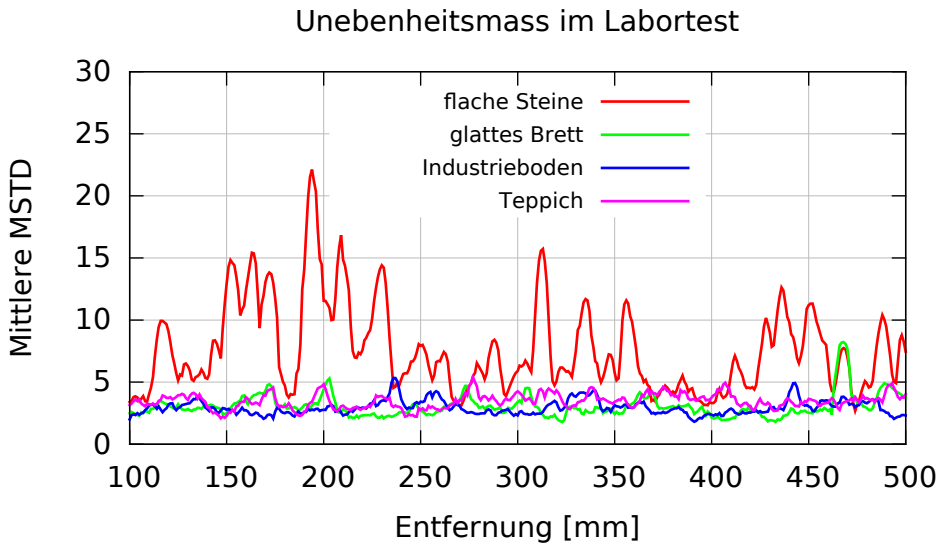


Abb. 7.22: Kontinuierliche Berechnung des *Unebenheitsmaß* R für die durchgeführten Labortest mit einer Fensterlänge von $\Delta t = 300$ ms.

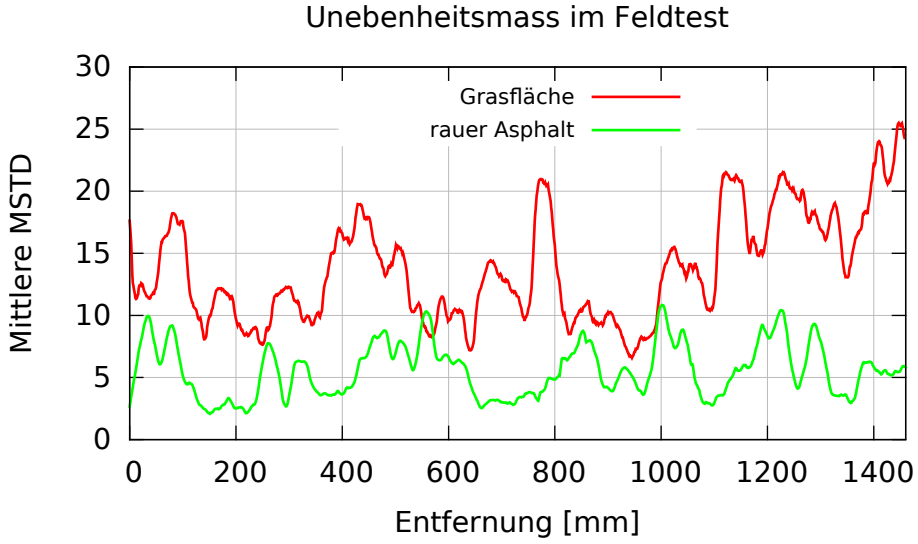


Abb. 7.23: Kontinuierliche Berechnung des *Unebenheitsmaß* R für die Feldtests auf einer Grasfläche und rauem Asphalt bei $\Delta t = 300$ ms.

Neben den Versuchen und Messungen in der Laborumgebung wurden ebenfalls Feldtests mit KAIRO 3 realisiert. Es wurden Experimente auf zwei verschiedenen Oberflächen, einer Grasfläche und rauem Asphalt, ausgeführt. Die aus den Motorströmen berechneten Werte des *Unebenheitsmaß* R, werden in Abbildung 7.23 für die jeweiligen Oberflächen gezeigt. Bei der Auswertung des *Unebenheitsmaß* R bei einer Fensterlänge von $\Delta t = 300$ ms, ist deutlich zu erkennen, dass sich im Vergleich zu glatten Oberflächen, höhere Werte und größere Schwankungen im Unebenheitsmaß ergeben. Für eine Grasfläche lassen sich im Durchschnitt um einen Faktor drei bis vier höhere Werte feststellen. Die hohen Werte für den rauen Asphalt lassen sich durch die grobe Körnung und Unebenheit des Asphalts erklären.

7.3.3 Festigkeitsmaß

Für die Auswertung und Optimierung des in Kapitel 5.3.2 eingeführten *Festigkeitsmaß* S wurde ein kleines Testfeld mit sandigem Untergrund aufgebaut. Zum Vergleich zwischen weichem und festem Untergrund wurden verschiedene Tests auf Sand und festem Industrieboden durchgeführt. Mit Hilfe des in Kapitel 5.3.2 beschriebenen Manövers konnte mit der Methode der kleinsten Fehlerquadrate das *Festigkeitsmaß* S bestimmt werden. Eine große Steigung der Regressionsgerade impliziert einen festen Untergrund und eine kleinere Steigung lässt auf weicheren Untergrund schließen. Entsprechend wurden für beide Untergrundarten die Motorströme in jeweils drei Testdurchläufen aufgezeichnet und verglichen.

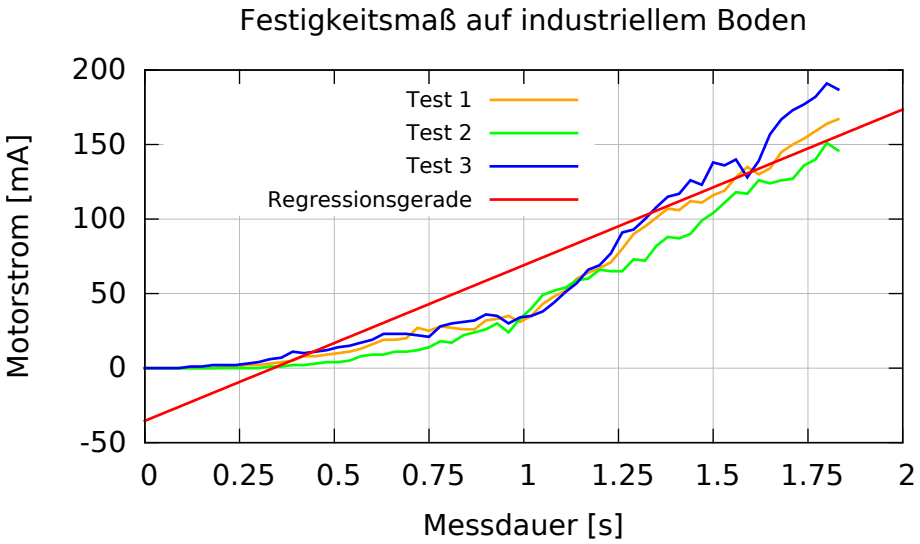


Abb. 7.24: Verlauf der Motorstromwerte während der drei Testläufe auf industriellem Boden. Aus der Steigung der Regressionsgerade (für Test 3 eingezeichnet) ergibt sich das *Festigkeitsmaß* S.

Abbildung 7.24 zeigt die aufgezeichneten Messwerte für den Industrieboden

sowie eine zugehörige Regressionsgerade. Aus Gründen der Übersichtlichkeit, wurde nur die Regressionsgerade für Test 3 eingezeichnet.

Bei der Ausführung des Manövers zur Messung der Bodenfestigkeit, ist bei festem Untergrund zu erkennen, dass der Widerstand der Räder beim Auftreffen auf die Oberfläche sich sehr stark erhöht. Kurz nach dem Auftreffen bleiben die Räder stehen und der Motorstrom steigt sehr rapide bis zur maximalen Schwelle an. Der maximale Motorstrom Schwellwert sorgt für eine Notabschaltung der Motoren, um diese nicht zu überlasten oder zu beschädigen.

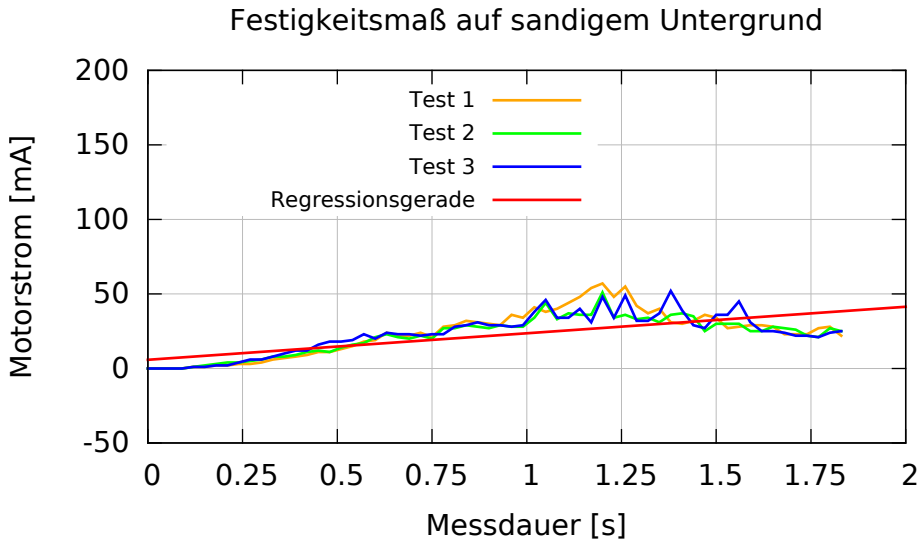


Abb. 7.25: Verlauf der Motorstromwerte während der drei Testläufe auf sandigem Boden. Aus der Steigung der Regressionsgerade (für Test 3 eingezeichnet) ergibt sich das *Festigkeitsmaß* S.

Im Gegensatz zum festen Untergrund, lässt sich auf dem weichen Untergrund beobachten, dass die Räder des Roboters beim Absenken des Kopfes zunächst einen kleinen Widerstand haben. Die Räder drehen sich kontinuierlich weiter während sie sich in den Untergrund eingraben. Ist die maximale Tiefe des Absenkens erreicht, drehen sich die Räder weiter und der Widerstand des Sandes wird geringer (Die Räder entfernen den lockeren Sand). Für die Motorströme in Abbildung 7.25 bedeutet das, dass im Falle des weichen Untergrundes nur ein sehr geringer Anstieg zu verzeichnen ist und bei erreichen der maximalen Tiefe, diese sogar fallen. Die resultierende Regressionsgerade für Test 3 hat eine deutlich flachere Steigung als die Regressionsgerade für den Industrieboden. Der Wert des *Festigkeitsmaß* S ist somit für weiche Oberflächen deutlich kleiner als auf festeren Oberflächen.

7.3.4 Selektion und Adaption der Fortbewegung

Als Erweiterung zu den Tests der verschiedenen Lokomotionsarten und der Bestimmung der Bodenbeschaffenheit, wurden Gesamttests zur Evaluation der Selektion der Fortbewegung sowohl in der Laborumgebung als auch im Feld durchgeführt. Zunächst wurden die in Tabelle 5.2 gezeigten Schwellenwerte bei den Laborexperimenten empirisch bestimmt.

Insbesondere zur Evaluierung der automatischen Umschaltvorgänge zwischen den drei Fortbewegungsarten *Radgetrieben*, *Omega-Lokomotion* und *Raupenartige Lokomotion* wurde ein Feldexperiment aufgebaut. Abbildung 7.26 zeigt den Tes-

Abb. 7.26: Aufbau des Testszenario für die Feldversuche. Auf einer Grasfläche wurden flache und kantige Steine platziert (rechts im Bild).

taufbau der eine Grasfläche und unstrukturierte Steine kombiniert. Ziel des Experiment ist, dass der Roboter KAIRO 3, selbstständig sowohl die Grasfläche als auch die Steinfläche[8] komplett überwindet.

Dazu wurde KAIRO 3 auf der linken Seite der Grasfläche in einer Entfernung von circa 3 m vor der Steinfläche positioniert, und in radgetriebener Geradeausfahrt gestartet. Zunächst wurden mehrere Tests lediglich im Radantriebsmodus durchgeführt. Wie erwartet, war keiner der durchgeführten Testläufe erfolgreich, da sich KAIRO 3 entweder mit den Rädern zwischen den Steinen verklemmte oder auf Grund von Bodenunebenheiten der Grasfläche seine Orientierung verlor. Dies zeigt deutlich die Notwendigkeit zur Anpassung der Fortbewegungsart. In den weiteren Experimenten wurde die automatische Selektion der Lokomotion aktiviert und der gleiche Test fünf mal wiederholt. Der detaillierte Ablauf eines solchen Experiments ist in der Bildsequenz in Abbildung 7.27 dargestellt. KAIRO 3 wurde erneut mit der radgetriebenen Fortbewegung gestartet. Auf Grund von Bodenunebenheiten auf der Grasfläche, schaltete KAIRO 3 selbstständig auf die *Omega-Lokomotion* um. Im weiteren Verlauf wechselte der Roboter

[8]Die Experimente wurden ohne zusätzliche optische Sensorik zur Umwelterkennung durchführt und dienen dem Test des automatischen Wechsel der Fortbewegungsart basierend auf den gemessenen Bodenbeschaffenheiten.

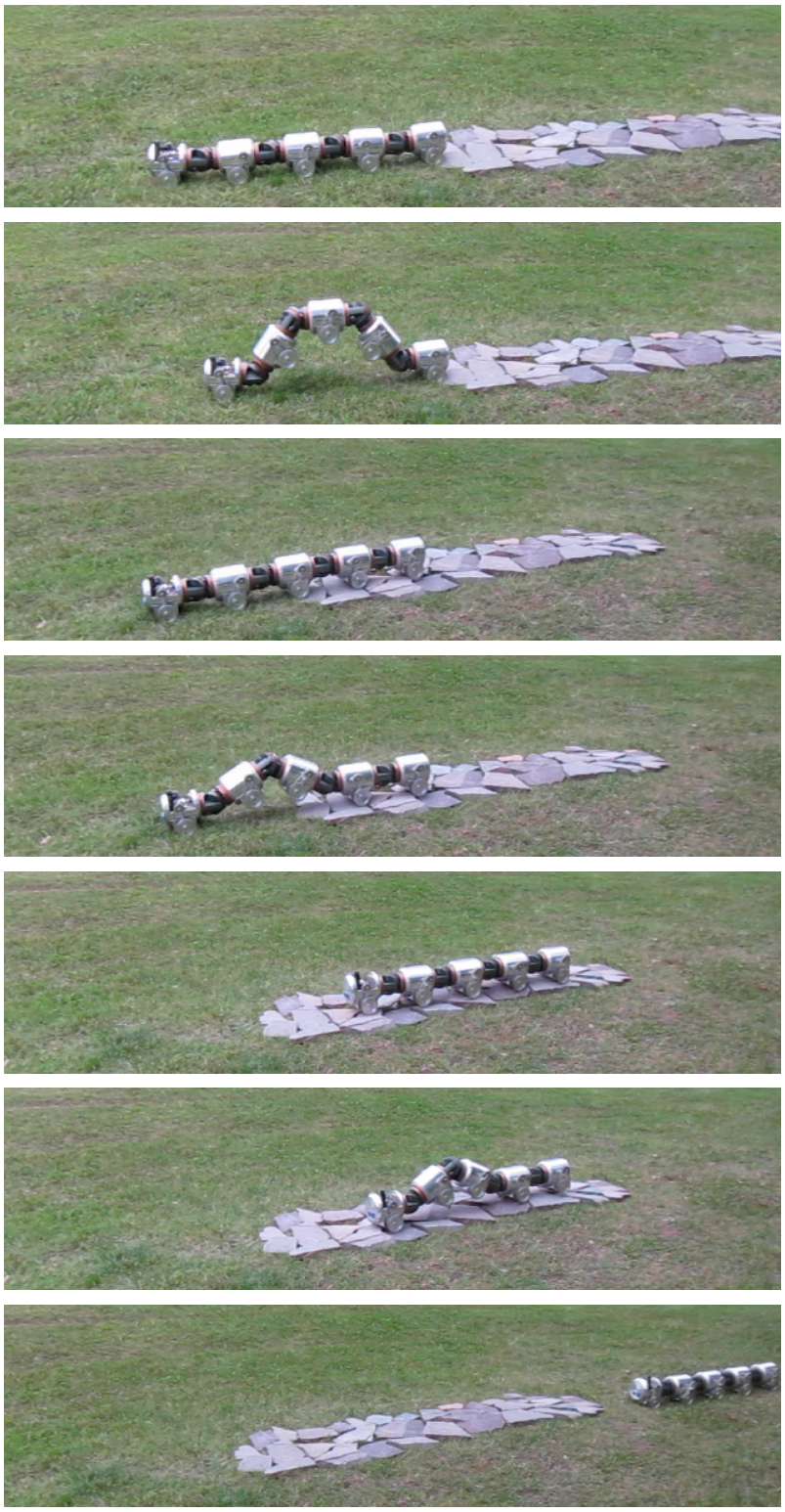


Abb. 7.27: Bildsequenz zum Ablauf eines Experiments zur Evaluierung der automatischen Umschaltung der Fortbewegungsart.

auf Grund einer höheren gemessenen Unebenheit zur raupenartigen Fortbewegung. Der erhöhte Wert des Unebenheitsmaß entstand infolge von blockierten und zwischen den Steinen festhängenden Rädern. Nach dem Durchlauf eines Zyklus der raupenartigen Fortbewegung, konnte KAIRO 3 die blockierten Räder befreien und auf den Radantriebsmodus zurück schalten. Während des gesamten Testlaufs wechselte die Steuerung mehrmals zwischen raupenartiger und radgetriebener Fortbewegung. Das Experiment wurde beendet sobald die Steinfläche komplett überwunden war. Insgesamt wurden fünf Wiederholungen durchgeführt, von denen KAIRO 3 vier Mal die Steinfläche erfolgreich überwinden konnte.

7.4 Konfigurationsabhängige Navigations- und Bewegungsplanung

Zur Evaluierung der Navigations- und Bewegungsplanung wurden vier verschiedene Szenarien entwickelt, die eine umfassende Überprüfung der unterschiedlichen Anforderungen ermöglichten. Alle im Folgenden beschriebenen Tests, wurden mit dem modularen Robotersystem KAIRO 3 (Kapitel 2.5) durchgeführt.

7.4.1 Szenario 1 : Rollwinkel Ausgleich

Das erste Szenario zielt darauf ab, die Gelenkwinkelplanung zu evaluieren. Insbesondere wird dabei die Rollwinkelanpassung untersucht, die dazu dient, unterschiedliche Bodenhöhen zwischen dem linken und rechten Rad eines Antriebsmoduls auszugleichen. Solche Höhenunterschiede treten besonders bei sehr unstrukturiertem Gelände auf. Für dieses Szenario wurden, wie in Abbildung 7.28 (oben links) gezeigt, grobe Steine zufällig innerhalb eines Quadrats mit einer Kantenlänge von 2 m angeordnet.

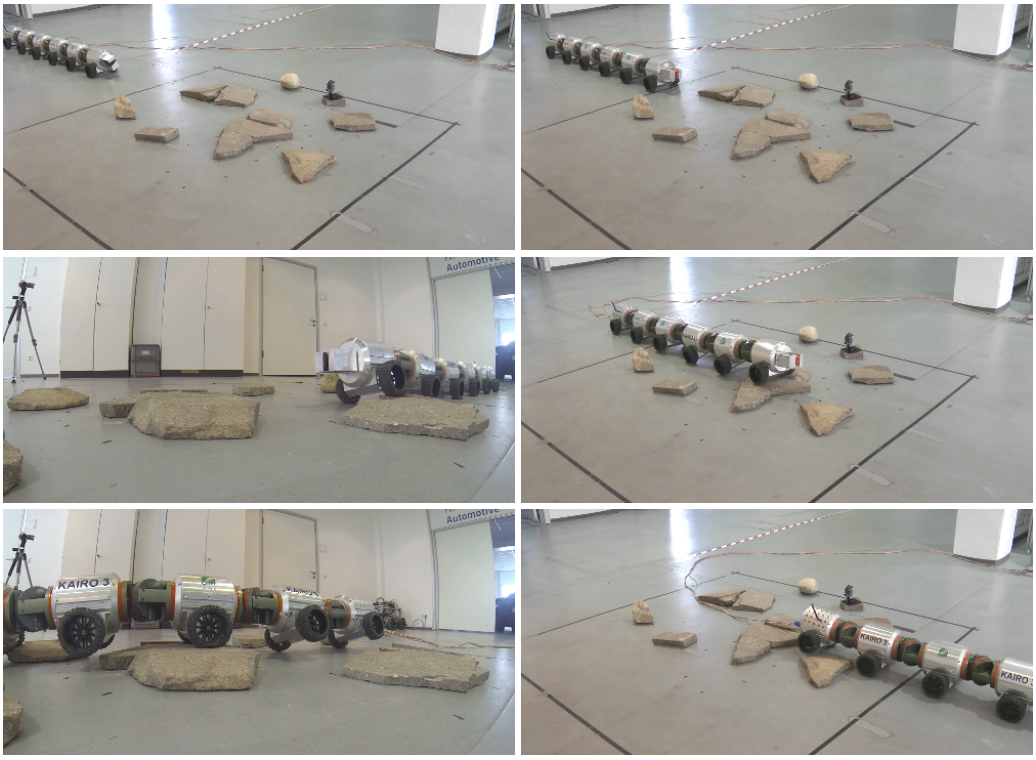


Abb. 7.28: Bildsequenz der automatischen Rollwinkelanpassung beim Durchqueren des steinigen Geländes.

In diesem Szenario wurden mehrere Experimente durchgeführt in denen KAIRO 3 wie im ersten Bild gezeigt außerhalb des Testgebietes platziert wurde.

Der Zielpunkt wurde in einem Abstand von 4 m auf der gegenüberliegenden Seite des Testbereichs gewählt, so dass der Roboter den markierten Bereich vollständig durchfährt. Nachdem die Zielposition festgelegt wurde, begann KAIRO 3 automatisch mit der Aufnahme von 3-D Punktwolken der Umgebung aus verschiedenen Posen (Mapping-Manöver in Kapitel 6.1.1). Die einzelnen Punktewolken wurden zu einer *PlexMap* kombiniert und die mehrstufige Planung erstellte einen Gelenkwinkelpfad zur Zielpose. Dieser Pfad enthält die 6-D Posen für das virtuelle Gelenkmodul sowie die Gelenkwinkel zum Ausgleich des Rollwinkels.

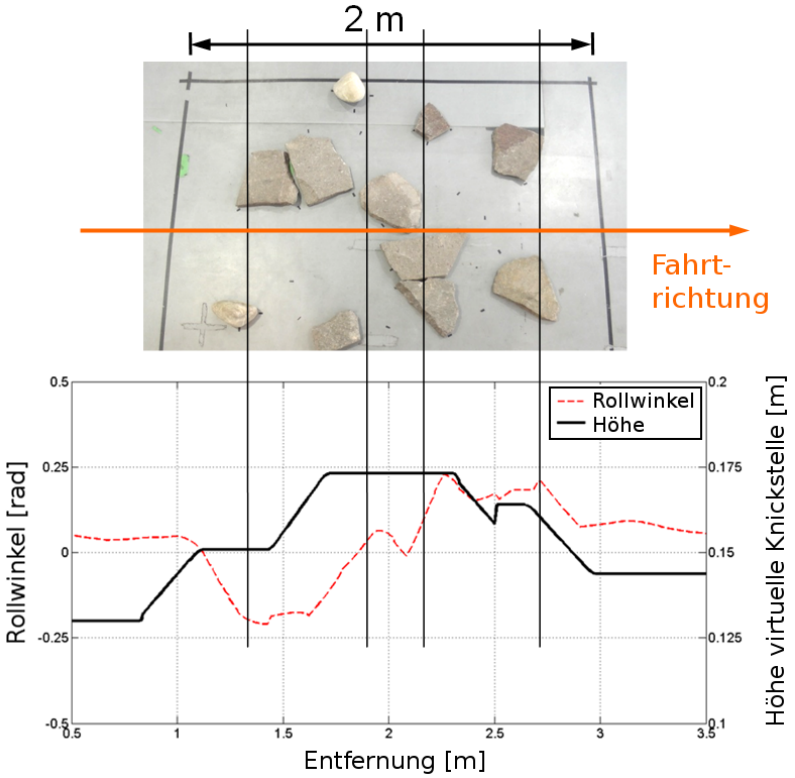


Abb. 7.29: Analyse der Rollwinkel- und Höhenanpassung bei unebenem Gelände.

Abbildung 7.29 zeigt das präparierte Gelände von oben (Draufsicht) sowie die Bewegungsrichtung des Roboters. Zur Überprüfung der erzeugten Trajektorie, wurden die Rollwinkelwerte des ersten Antriebsmoduls und die Höhenwerte des virtuellen Gelenkmodul[9] während der Ausführung des Pfades gemessen und analysiert.

Vergleicht man den Verlauf der Rollwinkel mit den abgebildeten Gelände, erkennt man, dass die Wendepunkte mit dem Anfang und Ende der Hindernisse übereinstimmen. Dies bedeutet, dass KAIRO 3 zu Beginn sein linkes Rad anhebt

[9] Der Ursprung des virtuellen Gelenkmodul befindet sich 130 mm über den Radaufstandspunkten.

(Rollwinkel wird negativ) um den Bodenkontakt mit beiden Rändern beizubehalten, während das linke Rad über die Steine fährt. Im weiteren Verlauf wird zusätzlich das rechte Rad angehoben, was zu einer nahezu Neutralisierung des Rollwinkels führt, da sich nun beide Räder auf einer ähnlichen Höhe befinden. Nach dem dritten Messpunkt senkt sich das linke Rad langsam ab und der Rollwinkel geht in den positiven Bereich über. Die Schwankungen zwischen dem dritten und vierten Messpunkt ergeben sich aus dem kurzzeitigen absenken und erneutem anheben des rechten Rades auf Grund eines weiteren Steins auf der rechten Seite. Nach dem vierten Messpunkt geht der Rollwinkel nahezu auf Null zurück.

Betrachtet man die Höhenkurve des virtuellen Gelenkmodul, können ebenfalls dem Gelände entsprechende Änderungen verzeichnet werden. Der erste Anstieg der Höhe lässt sich auf ein Anheben des virtuellen Gelenkmoduls zurück führen, um entsprechend das linke Rad anheben zu können. Der zweite Anstieg ergibt sich aus dem zusätzlichen Anheben des rechten Rades. Danach werden beide Räder nacheinander abgesenkt wobei auch in der Höhe die gleiche Schwankung wie bei den Rollwinkeln zu beobachten ist. Bei genauerer Betrachtung beider Kurven, ist eine leichte Verschiebung der Höhenkurve im Vergleich zur Rollwinkelkurve zu erkennen. Dies wurde durch das Abgreifen der Messwerte zum gleichen Zeitpunkt an unterschiedlichen Positionen (Rollwinkel im ersten Antriebsmodul und Höhenwerte im virtuellen Gelenkmodul) verursacht. Der Versatz entspricht dem Abstand zwischen Mittelpunkten des virtuellen Gelenkmoduls und des Antriebsmodul [91].

7.4.2 Szenario 2 : Überwinden von Stufen

Das zweite Testszenario, *Überwinden von Stufen*, basiert auf der zweistufigen Treppe, die in Abbildung 7.30 dargestellt ist. Grundsätzlich wurden die Fähigkeiten der implementierten Navigations- und Bewegungsplanung zur Überwindung komplexer Hindernisse untersucht. Insbesondere wurde evaluiert, ob die Pfadplanung es ermöglicht, Hindernisse mit unterschiedlichen Höhenabstufungen

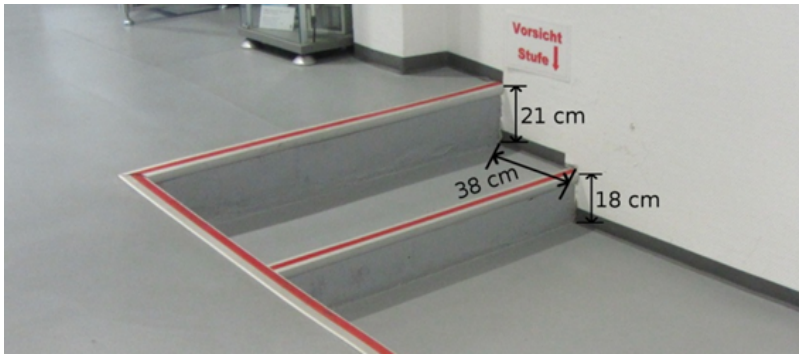


Abb. 7.30: Szenario 2 nutzt eine Treppe mit unterschiedlichen Stufenhöhen von 18 cm und 21 cm sowie einer Stufentiefe von 38 cm [91].

wie z. B. eine Treppe zu erklimmen. Besonderes Augenmerk wurde auf die Auswertung der erzeugten Gelenkwinkelvorgaben und deren Interpolation während der Ausführung gelegt.

Bei diesem Experiment wurde der Roboter, wie in Abbildung 7.31 oben links dargestellt, 1 m vor den Stufen platziert und eine Zielpose in 4 m Entfernung vor dem Roboter vorgegeben. Die darauf folgenden Schritte wurden alle autonom

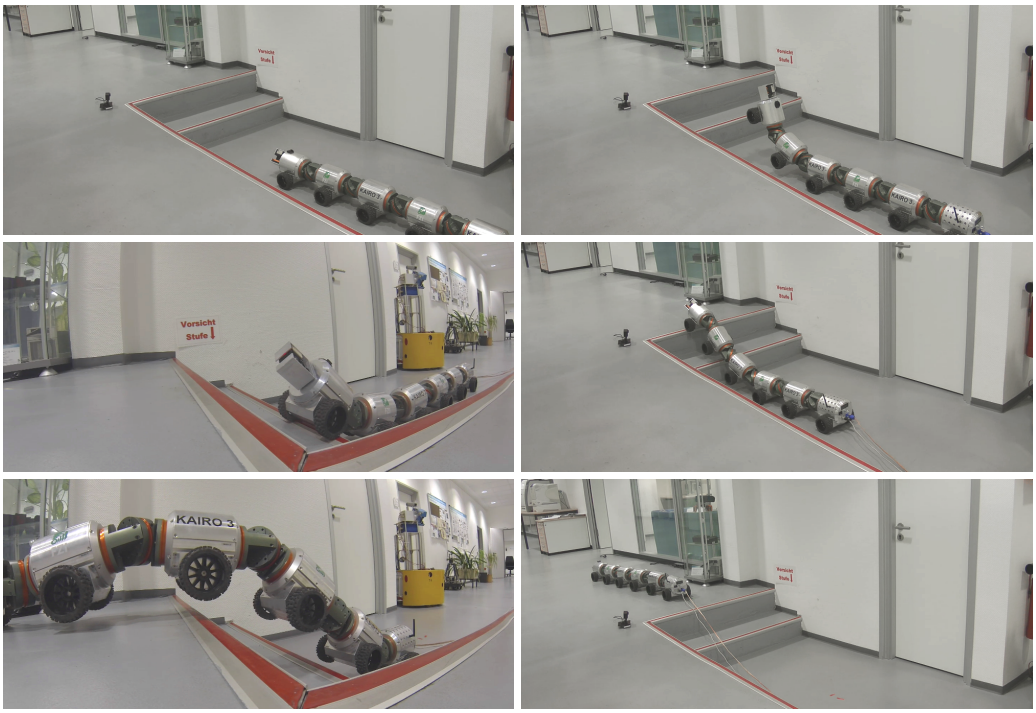


Abb. 7.31: Bildsequenz des Treppensteigens mit KAIRO 3 in Szenario 2 [91].

von KAIRO 3 geplant und durchgeführt. Zunächst wurde an der Startposition eine Punktewolke mit dem 3-D Laserscanner KaRoLa aufgenommen. Mit Hilfe des Mapping-Manövers aus Kapitel 6.1.1 wurde eine weitere 3-D Punktewolke von einer erhöhten Position (Abbildung 7.31 oben rechts) aufgenommen und mit der ersten Punktewolke registriert. Während der Aktualisierung der *PlexMap* kehrte KAIRO 3 in die ursprüngliche Position zurück. Die so erhaltene Karte wurde zur Planung eines Pfades zur Zielpose genutzt. Wie in Kapitel 6.3 erläutert, wurde zunächst eine Grob- und Feinplanung durchgeführt, gefolgt von der Planung und Erzeugung einer Gelenkwinkel-Trajektorie. Abbildung 7.32 stellt die registrierten Punktewolken, die erzeugte Karte und die geplanten Pfade dar. Die resultierende Gelenkwinkel-Trajektorie wurde von der *Bewegungsausführung* stückweise in die *Virtuelle Schiene* überführt und von der *Adaptive Steuerung* ausgeführt. Wie in Abbildung 7.31 gezeigt, erreicht KAIRO 3 die gewünschte Zielpose autonom, indem es der virtuellen Schiene folgt.

Um die Wiederholbarkeit zu bewerten, wurden in diesem Szenario insgesamt fünf Testläufe mit dem realen Roboter durchgeführt. In vier dieser fünf Durchläufe erreichte KAIRO 3 den vorgegebenen Zielpunkt ohne fremde Einmischung.

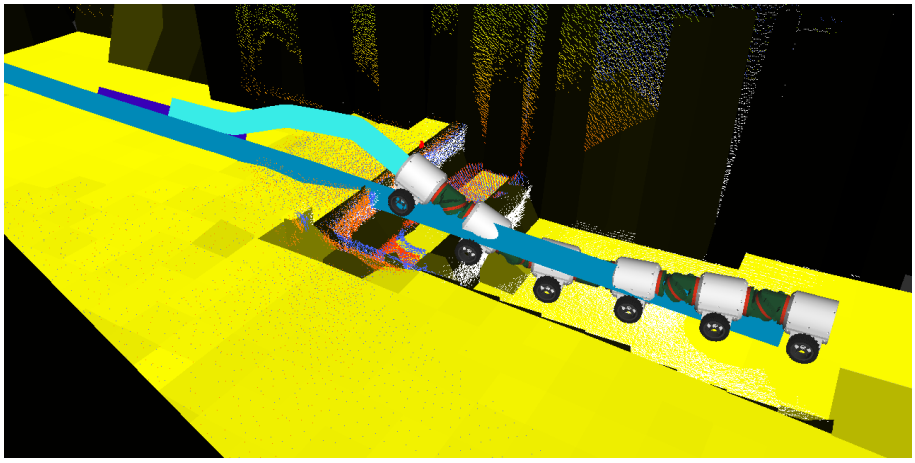


Abb. 7.32: Erzeugte 3-D Punktewolken, PlexMap Karte, geplanter grober und feiner Pfad sowie resultierende Gelenkwinkel-Trajektorie in Szenario 2 [91].

Allerdings musste einer der Testläufe aufgrund einer zu großen Lokalisierungsabweichung vorzeitig abgebrochen werden. Ausschlaggebend dafür war ein Fehler in der Roboter-Odometrie, der durch Schlupf und das Abrutschen des Roboters verursacht wurde. Die meisten dieser Abweichungen können durch die Aufnahme einer zusätzlichen 3-D Punktwolke zur Bestimmung der aktuellen Roboterposition korrigiert werden. Größere Ungenauigkeiten der Odometrie, die zwischen zwei Korrekturpunkten auftreten, können jedoch dazu führen, dass der geplante Pfad ungültig wird. Eine Erweiterung der Lokalisierung durch ein zusätzliches Sensorsystem zur Bestimmung der Roboterpose, wie beispielsweise eine Trägheitsmesseinheit (IMU), kann die Lokalisierung zwischen zwei Korrekturpunkten verbessern. Gegebenenfalls kann bei großen Abweichungen oder Ungenauigkeiten eine Anpassung oder Neuerzeugung des Bewegungspfades angestoßen werden.

Die Ausführungszeiten der einzelnen Planungsstufen wurden über die fünf durchgeführten Experimente gemittelt und sind in Tabelle 7.10 aufgeführt. Die Planungszeiten wurden auf einen i5-Prozessor mit 3,3 GHz und 4 GB RAM gemessen [91].

Planungsschritt	Gemittelte Rechenzeit
Grobplanung	5,5 s
Feinplanung	4,5 s
Gelenkwinkelplanung	2,0 s
Summe	12,0 s

Tabelle 7.10: Durchschnittliche Rechenzeit der einzelnen Planungsschritte zur Navigations- und Bewegungsplanung [91].

Zur Auswertung und Analyse der Gelenkwinkelplanung, wurden während der

Ausführung aller Experimente die Winkel α, β und γ der entsprechenden Gelenke des virtuellen Gelenkmoduls[10] und die Höhe z der virtuellen Knickstelle aufgezeichnet. Abbildung 7.33 zeigt den Verlauf der Messwerte beim Überwinden der Stufen für einen ausgewählten Testlauf. Die Aufzeichnung startet nach

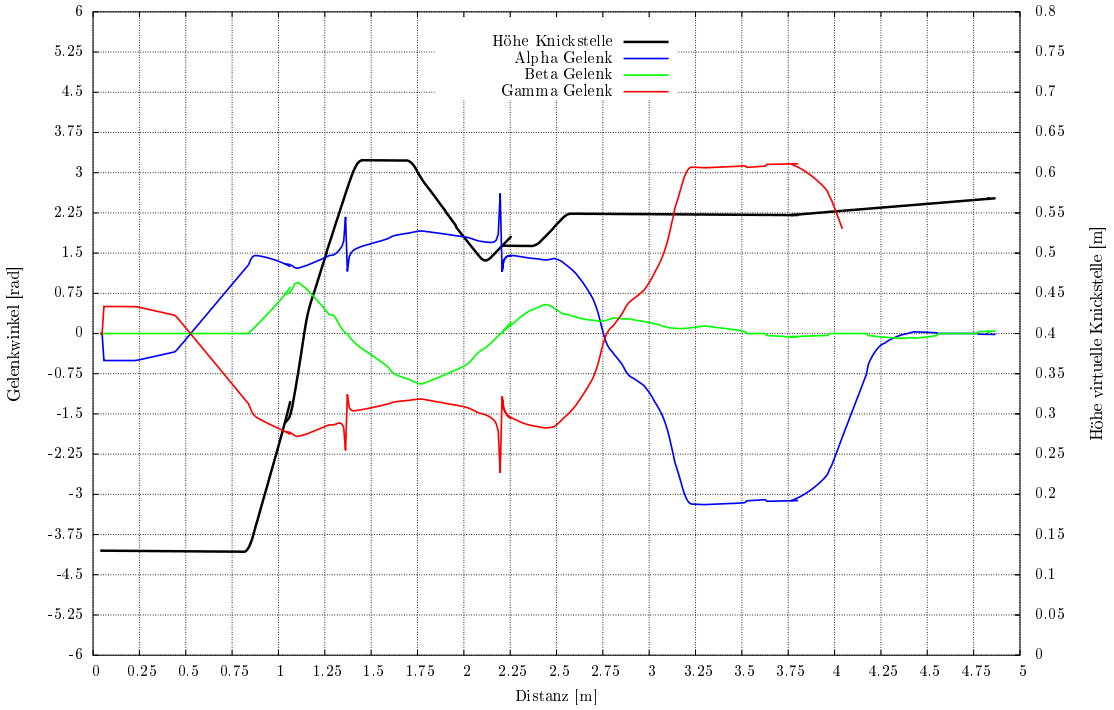


Abb. 7.33: Auswertung der Gelenkwinkel und des Höhenprofils des ersten Moduls.

dem der Roboter das *Kartierungsmanöver* durchgeführt hat. Zu Beginn lässt sich erkennen, dass die α und γ Gelenke sich drehen, obwohl der Roboter keine Kurvenfahrt (β Konstant) macht oder seine Höhe ändert. Dieses Vordrehen der Gelenke bereitet den Roboter auf das Anheben der Module vor. Auf Grund der speziellen Kinematik eines Gelenkmoduls, müssen α und γ jeweils um 90° gedreht sein, bevor über das β Gelenk ein Anheben des davor liegenden Moduls erreicht werden kann. Der Gelenkwinkelplaner erzeugt automatisch diese vorausschauende Bewegung der Gelenke bei großen Höhenänderungen. Im weiteren Verlauf ist zu erkennen, dass sich mit Änderung des β Winkels, die Höhe z der virtuellen Knickstelle rapide ansteigt. Der β Winkel fällt entsprechend ab, wenn die maximale Steigrate erreicht ist und sich das Modul zurück in waagerechte Position bewegt. Erreicht β den Nullpunkt, steigt die Höhe nicht weiter an. Der Verlauf der Höhe z steigt zeitweise über die finale Höhe. Dieses Überschwingen der Knickpunkte der Gelenkmodule wird ebenfalls vom Gelenkwinkelplaner erzeugt und sorgt dafür, dass die Räder der Antriebsmodule sich an der Kante eines Hindernisses entlang bewegen. Die kurzen Schwankungen der α und γ Gelenke

[10] Das virtuelle Gelenkmodul befindet sich vor dem ersten Antriebsmodul und dient der Richtungsvorgabe, da sich lediglich die Mittelpunkte der Gelenkmodule, die sogenannten Knickstellen, auf der virtuellen Schiene befinden.

an Stellen von Nulldurchgängen der β Gelenke lässt sich auf numerische Berechnungsungenauigkeiten der inversen Kinematik zurück führen, machen sich bei der Ausführung des Pfades jedoch nicht negativ bemerkbar. Drastischer sind die großen Winkeländerungen (Vorzeichentausch) der α und γ Winkelwerte bei einer Distanz von 0,5 m und 2,75 m. Diese Winkeländerungen verlangsamen die Ausführung durch Wartezeiten bis die Gelenke sich auf die gewünschte Position gedreht haben. Diese Problematik ergibt sich aus der unabhängigen Berechnung der Gelenkwinkel und der anschließenden Interpolation zwischen den einzelnen Posen. Ein zusätzlicher Schritt zur Synchronisierung der Winkel bei der Gelenkwinkelplanung könnte dies verbessern [91].

7.4.3 Szenario 3 : Überwinden einer Kiste

Im dritten Szenario wird neben dem Hinaufsteigen bzw. Erklimmen von Stufen und Hindernissen, die Fähigkeiten untersucht größere Hindernisse komplett zu überwinden. Dazu wurde die in Abbildung 7.34 oben links dargestellte Kiste mit einer Höhe von 0,42 m und einer Tiefe von 0,56 m als Hindernis präpariert. Um zu vermeiden, dass die Pfadplanung einen Weg um die Kiste herum erzeugt, wurden noch höhere Hindernisse links und rechts neben der Kiste platziert. KAIRO 3 wurde ähnlich wie bei den anderen Experimenten, 1 m vor der Kiste platziert. In diesem Experiment wurde die Zielpose allerdings in einem benachbarten Raum in einer Entfernung von $x = 10$ m und $y =$ -6 m festgelegt. Die Zielpose ist vom Startpunkt des Roboters aus nicht direkt sichtbar. Damit wurde zusätzlich getestet, wie gut die *Grobplanung* mit unbekannten Bereichen umgehen kann und ob das Verfahren sich aus lokalen Minima befreien kann. Die resultierenden Pfade der *Grobplanung*, *Feinplanung* und *Gelenkwinkelplanung* sowie die *PlexMap* sind in Abbildung 7.35 dargestellt. Abbildung 7.34 zeigt den Ablauf eines der durchgeführten Experimente, die von KAIRO 3 inklusive der Planung eigenständig ausgeführt wurde.

Bei den durchgeführten Experimenten war es KAIRO 3 immer möglich sich auf die Kiste zu bewegen, wie in Abbildung 7.34 links in der Mitte zu sehen. Jedoch gelang es nur bei einem von drei Versuchen, auf der anderen Seite der Kiste den Boden ohne Umkippen des Roboters zu erreichen. Dies lässt sich auf zwei verschiedene Ursachen zurückführen:

1. Abweichungen der Odometrie durch Wegrutschen der Räder auf der glatten Kunststoffoberfläche der Kiste.

2. Ungenaue Karte auf der Rückseite der Kiste durch Verdeckung bei der Datenaufzeichnung: Der Boden hinter der Kiste wurde als flache Rampe anstelle einer Stufe erkannt.

Die erste Ursache kann durch die in Szenario 1 erwähnte kontinuierliche Odometriekorrektur mit Hilfe eines zusätzlichen Lokalisierungsverfahrens verbessert werden. Um die Problematik des zweiten Punktes zu lösen, ist es notwendig, die

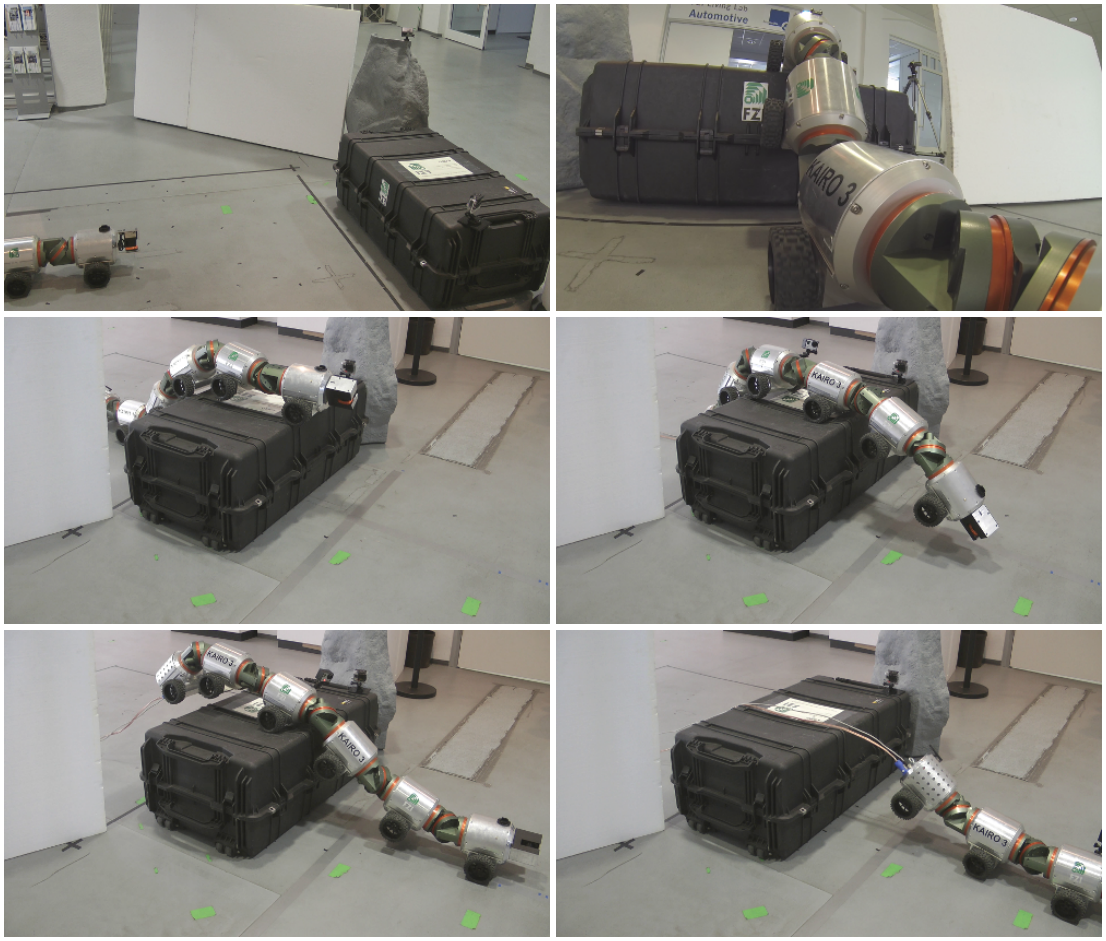


Abb. 7.34: Die Bildsequenz zeigt KAIRO 3 beim Überwinden einer große Kiste als Hindernis [93].

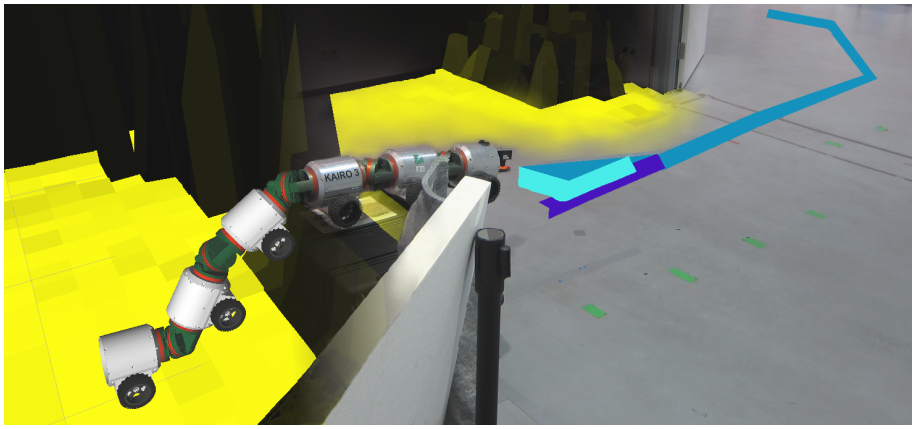


Abb. 7.35: Übergang aus der virtuellen Umgebung in die reale Welt: KAIRO 3 überwindet eine große Kiste [93].

Genauigkeit für relevante Teile der Karte zu erhöhen. Besonders bei Hindernissen und Verdeckungen ist dies nicht einfach, da der entsprechende Kartenausschnitt oft erst kurz vor Erreichen in das Sichtfeld des Sensors gelangt. Durch Ausnutzen zeitlicher oder räumlicher Impulse der *Missionssteuerung* lässt sich die Aufnahme einer neuen Punktewolke und Aktualisierung der Karte auslösen. Solche räumlichen Impulse könnte z. B. durch unzureichende oder unbekannte Karteninformationen, wie z. B. zu große Kartenzellen oder zu wenige Messpunkte in den relevanten Kartenausschnitten, erzeugt werden [93].

7.4.4 Szenario 4 : Unterschiedliche Roboter Konfigurationen

Ziel von Szenario 4 ist die Evaluierung der Planungsfähigkeiten bezüglich unterschiedlicher Roboterkonfigurationen. In Bezug auf KAIRO 3 bedeutet dies, dass die Länge des Roboters bzw. die Anzahl der Module des Zentralkörpers z. B. durch Rekonfiguration verändert wird, wie in Kapitel 3 erläutert.

Abbildung 7.36 oben links zeigt den Testaufbau bestehend aus einer 6 m langen Rampe und einer präparierten Stufe mit einer Höhe von 0,54 m. Die Rampe fällt vom oberen Niveau der Stufe langsam ab und geht in die untere Bodenebene über. Damit hat der Roboter die Möglichkeiten, entweder über die Stufe zu klettern oder einem Umweg über Rampe in Kauf zu nehmen, um einen Zielpunkt auf dem höheren Niveau zu erreichen.

Für alle Experimente wurde KAIRO 3 parallel zur Rampe und in einem Abstand von 1,5 m vor der Stufe platziert. Der vorgegebene Zielpunkt liegt oberhalb der Stufe in 5 m Entfernung vor dem Roboter. Es wurden verschiedene Experimente mit zwei unterschiedlichen Roboterkonfigurationen durchgeführt: Eine lange Konfiguration, bestehend aus insgesamt elf Modulen (6 Antriebs- und 5 Gelenkmodule) und eine kurze Konfiguration, bestehend aus nur fünf Modulen (3 Antriebs- und 2 Gelenkmodule).

Abbildung 7.37 zeigt die erzeugten Punktewolken, die Karte und die Pfade für einen Roboter mit der langen Konfiguration aus elf Modulen. Die beiden aufgenommen Punktewolken sind jeweils durch weiße und blaue Punkte dargestellt und die daraus generierte *PlexMap* ist, je nach Inklination der Zellen, in verschiedenen Gelbstufen dargestellt[11]. Das eingezeichnete Koordinatensystem zeigt den Startpunkt vor der Durchführung des *Kartierungsmanöver*. Die drei geplanten Pfade sind ebenfalls farblich kodiert: Der grobe Pfad ist in dunklem türkis, der feine Pfad in violett und der Gelenkwinkelpfad in hellem türkis dargestellt. Am Beispiel des groben Pfades, der durch die Stufe hindurch geht, ist deutlich zu sehen, dass dieser nur eine grobe Approximation darstellt und die Konturen und Oberflächen von Hindernissen im Pfad selbst nicht berücksichtigt. An Hand der Kostenfunktion wird jedoch berücksichtigt, ob die aktuelle Roboterkonfiguration in der Lage ist dieses Hindernis zu überwinden. Die Feinanpassung des Pfades an

[11] Die hellgelbe Farbe entspricht einer horizontalen Ausrichtung und die dunkelgelbe Farbe einer vertikalen Ausrichtung.

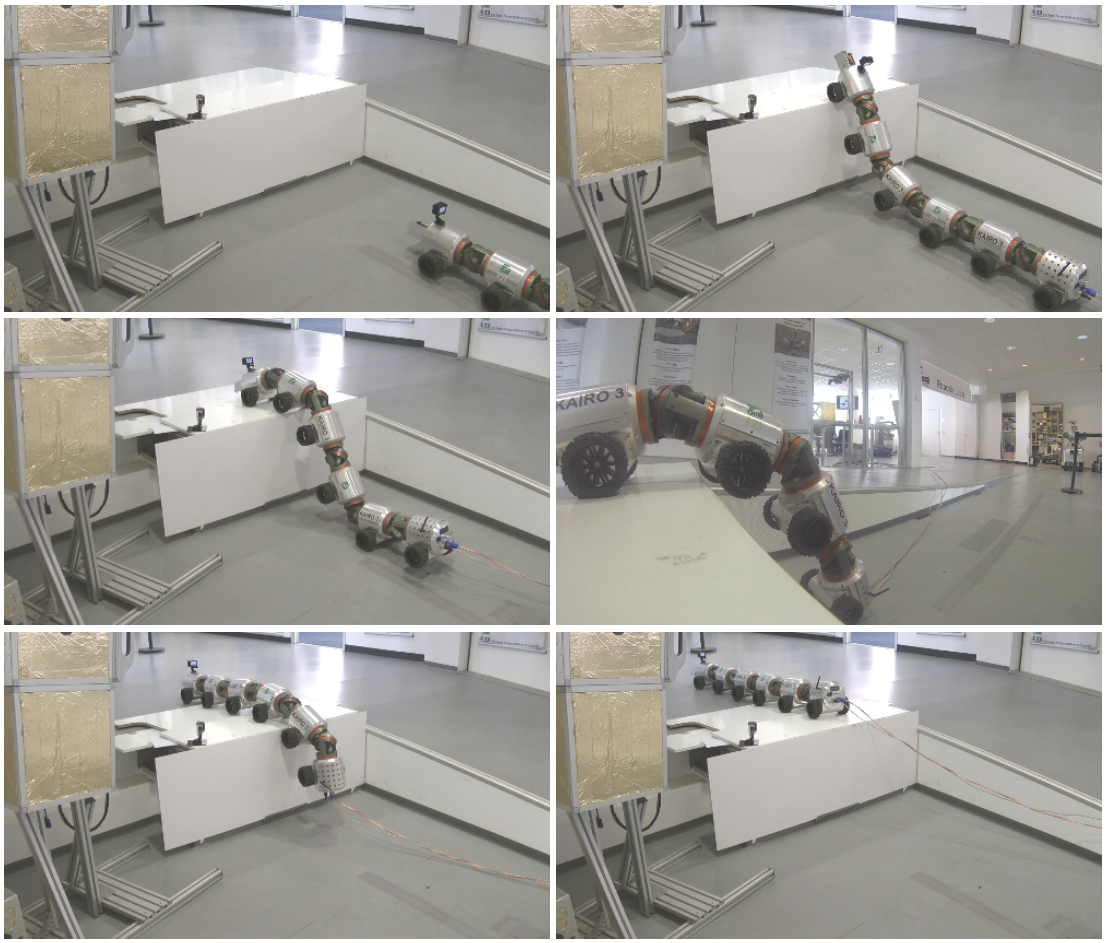


Abb. 7.36: Die Bildsequenz zeigt wie KAIRO 3 mit einer Konfiguration aus elf Modulen selbstständig eine 0,54 m hohe Stufe erklimmt.

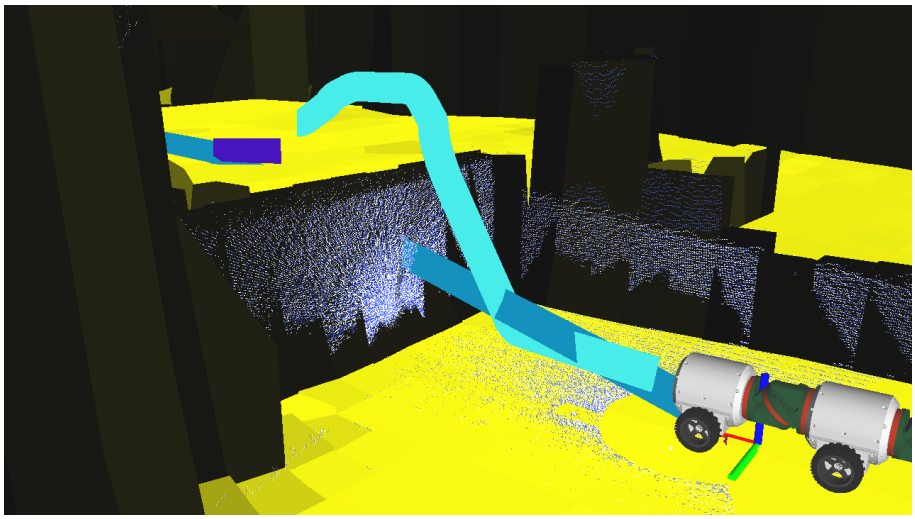


Abb. 7.37: Virtuelle Umgebung mit 3-D Punktewolken, *PlexMap* und der geplanten Pfade für KAIRO 3 mit einer Konfiguration aus elf Modulen.

die Oberfläche von Hindernissen erfolgt schließlich in der Gelenkwinkelplanung. Betrachtet man den resultierenden Gelenkwinkelpfad, so ist das Überschwingen der Knickstellenhöhe auf der Oberseite der Stufe deutlich zu erkennen. Damit wird erreicht, dass sich die Antriebsräder stets entlang der Kante bewegen.

Die Bilderserie in Abbildung 7.36 stellt den erfolgreichen Ablauf der Ausführung dar, bei dem KAIRO 3 die 0,54 m hohe Stufe selbständig überwindet und die Zielpose hinter der Stufe erreicht. Bei einer kürzeren Konfiguration von KAIRO 3 mit lediglich fünf Modulen konnte die Navigations- und Bewegungsplanung ebenfalls erfolgreich einen Pfad zum Zielpunkt finden. Abbildung 7.38 (links) zeigt den globalen Pfad, der entsprechend der Roboterkonfiguration über die Rampe verläuft, um den Höhenunterschied zum Zielpunkt zu überwinden. Zunächst wurde eine Kehrtwende geplant, damit der Roboter aus dem lokalen Minimum vor der Stufe entkommen kann, welches nun ein Hindernis darstellt. An einer ebenen Stelle führt der Weg auf die Rampe und anschließend zum Zielpunkt.

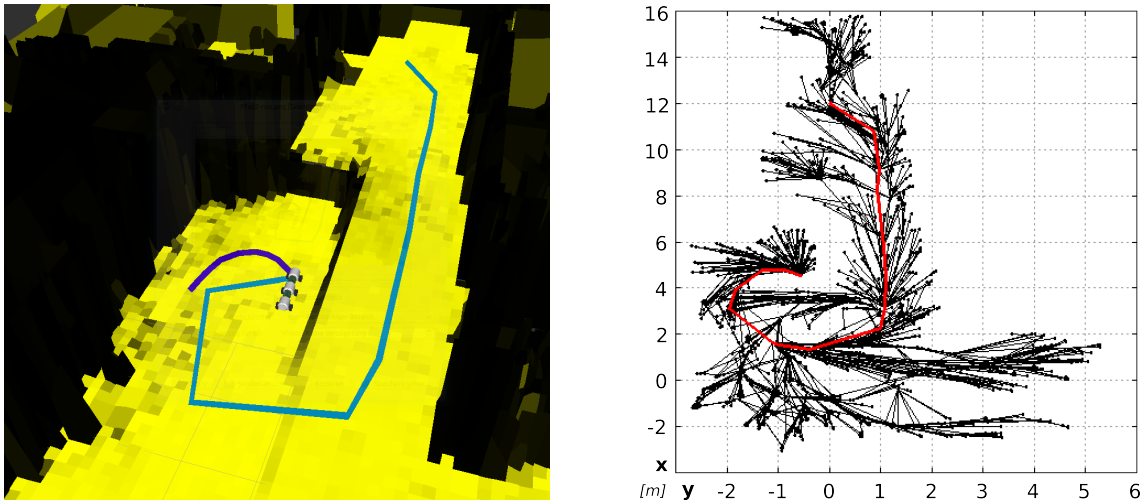


Abb. 7.38: Geplante Pfade für KAIRO 3 mit einer kurzen Konfiguration (links) mit zugehörigem RRT* Baum der *Grobplanung* (rechts).

Abbildung 7.38 (rechts) stellt den zugehörigen durch die *Grobplanung* generierten RRT* Baum dar. Anhand der erzeugten Konten und Kanten ist zu erkennen, dass die kinematischen Einschränkungen für einen nicht-holonomen Roboter bei der Planung berücksichtigt werden.

7.5 Zusammenfassung der Ergebnisse

Das Konzept einer verteilten und flexiblen Steuerungsarchitektur für RMRS und dessen Umsetzung mit dem ROS 2 Framework konnte mit Hilfe von drei verschiedenen Testszenarien verifiziert werden. Die entwickelte Hardwareschnittstelle zum Koppeln und Entkoppeln der Bausteine eines RMRS wurde mittels eines 3-D gedrucktem Prototypen getestet. Die Implementierung der

elektrischen und Datenschnittstelle, der integrierten Sensorik und Schnittstellensteuerung sowie der mechanischen Schnittstelle und des Kopplungsvorgangs wurden bewertet und diskutiert. Ein Belastungstest mit KAIRO 3 verdeutlichte, dass der Schnittstellenprototyp bereits die gestellten Anforderungen bezüglich Stabilität und Robustheit übertrifft. Das Visual Servoing basierte Verfahren zur Lokalisierung und Kopplung zweier KAIRO 3 Roboter wurde in der Physiksimulation Gazebo mit einer virtuellen Kamera untersucht. Die Ergebnisse für die drei verschiedenen Startpositionen legen die grundsätzliche Funktionsweise sowie Möglichkeiten zur Verbesserung der Winkelerkennung dar.

Im Rahmen der Bestimmung optimaler Roboterkonfigurationen wurden verschiedene Teilaspekte der jeweiligen Synthese Schritte untersucht. Beim Vergleich der drei implementierten Algorithmen zur Bausteinauswahl zeigte die *ganzzahlige lineare Optimierung* gegenüber der *Brute-Force* und *Greedy* Implementierung ein besseres Laufzeitverhalten und eine höhere Qualität der Ergebnisse. Die Inferenz von Regeln mit einem individuellen problemspezifischen Reasoning ergab hinsichtlich der Modellierung und der Ausführungszeiten bessere Resultate als eine Ontologie-basierte Umsetzung mittels *FaCT++*. Der evolutionäre Algorithmus zur Bausteinanordnung wurde in mehreren Teilbereichen untersucht. Es konnte gezeigt werden, dass die zielgerichtete Initialisierung und Erzeugung neuer Individuen eine Steigerung der Qualität bei reduzierter Ausführungszeit ermöglicht. Bei der Betrachtung der Laufzeit wurde durch Parallelisierung eine Reduktion der Rechenzeit von mehr als 50 % festgestellt. Zur Bewertung des interaktiven CARD Werkzeug und des Syntheseprozess als Ganzes, wurden vier Testszenarien mit jeweils verschiedenen Missions-Anforderungen miteinander verglichen. Im vierten Szenario wurde die Übertragbarkeit der implementierten Verfahren auch auf andere Anwendungsbereiche, wie der Optimierung modularer Satelliten, evaluiert.

Die Auswertung der biologisch motivierten Fortbewegungsarten erfolgte sowohl durch Labortests als auch durch Feldversuche mit dem Roboter KAIRO 3. Die Ergebnisse belegen die Funktionalität und die Simulation von Roboterkonfigurationen mit einer großen Anzahl von Modulen zeigt die Skalierbarkeit des Ansatzes. Die Bestimmung der Bodenbeschaffenheit mittels Unebenheits- und Festigkeitsmaß sowie die automatische Auswahl der Lokomotion konnte ebenfalls durch Versuche im Feld erfolgreich verifiziert werden.

Die Evaluierung der konfigurationsabhängigen Navigations- und Bewegungsplanung wurde in verschiedenen Experimenten mit dem Roboter KAIRO 3 durchgeführt. Die Resultate des ersten Testszenarios bestätigen die Funktionsweise der Rollwinkelanpassung bei unebenem Gelände. Mit Hilfe des zweiten und dritten Szenarios konnte die Planung von Gelenk-Winkel-Trajektorien und damit das Überwinden komplexer Hindernisse demonstriert werden. Das vierte Testszenario zeigte, dass bei der Navigations- und Bewegungsplanung unterschiedliche Konfigurationen berücksichtigt werden. Insbesondere erzeugte die Planung in Abhängigkeit von der Roboterkonfiguration (kurzer oder langer Roboter) unterschiedliche Wege.

8 Schlusswort

In dieser Arbeit wurden Methoden zur aufgabenbasierten Rekonfiguration und Adaption von modularen Servicerobotern untersucht und entwickelt mit dem Ziel, die heutigen Robotersysteme anpassungsfähiger und autonomer zu gestalten. Darüber hinaus wurde eine Verbesserung der Robustheit von Hardware- und Softwaresystemen für den Feldeinsatz bei Such- und Rettungseinsätzen sowie Inspektions- und Wartungsarbeiten angestrebt. Teile der vorgestellten Algorithmen wurden im Rahmen des iBOSS-Projekts [58] entwickelt oder erweitert. Dieses Kapitel fasst den Inhalt der vorliegenden Arbeit zusammen und gibt einen Ausblick auf zukünftige Arbeiten.

8.1 Zusammenfassung der Arbeit

Die Eigenschaften aktueller selbst-rekonfigurierbarer Robotersysteme wurden nach dem Stand der Technik analysiert und eine Klassifizierung der Systeme hinsichtlich der verwendeten Hard- und Softwarekonzepte beschrieben. Darauf aufbauend wurden aktuelle Verfahren zur Repräsentation sowie Generierung und Optimierung von Hardwarekonfigurationen vorgestellt. Darüber hinaus wurden neueste Ansätze zur Fortbewegung von Robotern und deren Anpassung an die Umgebung sowie Algorithmen zur Navigations- und Bewegungsplanung von rekonfigurierbaren Robotersystemen dargestellt. Zudem wurden die Grundlagen und der Aufbau des modularen schlangenartigen Roboters KAIRO 3 erläutert, der als Versuchsträger für diese Arbeit dient.

Das Kapitel *Entwurf rekonfigurierbar Multi-Roboter-Systeme (RMRS)* gibt einen Einblick in die zur Realisierung eines RMRS erforderlichen Hardware- und Softwarekomponenten. Zunächst wurde die Klassifizierung von modularen, rekonfigurierbaren Systemen verfeinert, indem die Geometrie und Funktionalität von Bausteinen auf verschiedenen Ebenen betrachtet wurde. Speziell für RMRS wurde eine flexible und verteilte Steuerungsarchitektur entwickelt, die sich durch eine dezentrale Struktur an unterschiedliche Roboterkonfiguration anpasst. Neben den Steuerungskomponenten wurden ebenfalls verschiedene Konzepte zur Kommunikation und Datenübertragung diskutiert. Das Design einer Hardwareschnittstelle bestehend aus elektrischer, mechanischer und Datenschnittstelle wurde als 3-D gedruckter Prototyp realisiert und ermöglicht das automatische An- und Abkoppeln von Modulen. Zur Erkennung und Zusammenführung zweier unabhängiger Roboter wurde ein Visual Servoing basierter Ansatz implementiert und in einer Simulationsumgebung getestet.

Die Generierung und Optimierung von Roboterkonfigurationen beschreibt einen mehrstufigen Syntheseprozess zur aufgabenbezogenen Konfigurationserzeugung. Neben der Modellierung eines Baustein- und Regelkatalogs wurden verschiedene Algorithmen und Verfahren für die drei Schritte Bausteinauswahl, Regelinferenz und Optimierung der Bausteinanordnung diskutiert und deren Umsetzung dargelegt. Die Realisierung einer interaktiven Benutzerschnittstelle führt den Anwender durch den mehrstufigen Syntheseprozess. Um die Allgemeingültigkeit der implementierten Methoden nachzuweisen, erfolgte die Übertragung auf andere Anwendungsbereiche am Beispiel der Erzeugung und Optimierung modularer Satellitenkonfigurationen.

Darüber hinaus wurden biologisch inspirierte Methoden zur Fortbewegung schlangenartiger Robotersysteme und deren Anpassung an die Umweltbedingungen untersucht. Im Speziellen wurden die beiden verschiedenen Lokomotionsarten der Nachtfalter- und Spannerraupe analysiert und beschrieben. Die Implementierung einer Ausgleichsbewegung ermöglichte es, beide Arten der Raupenbewegung mit den Roboter KAIRO 3 zu realisieren. Zur Ermittlung der Bodenbeschaffenheit wurden der Unebenheits- und Festigkeitswert basierend auf den Sensordaten der Antriebselemente eingeführt. Mit Hilfe der beiden Messgrößen konnte die Anpassung der Fortbewegungsart im Feldversuch durch Auswahl der für das Gelände am besten geeigneten Fortbewegung erfolgreich erprobt werden.

Die Navigations- und Bewegungsplanung für RMRS ist ein Verfahren, das unter Berücksichtigung der aktuellen Roboterkonfiguration einen geeigneten Pfad zur Fortbewegung findet. Basierend auf dem PlexNav-Framework und einem modifizierten RRT*-Algorithmus wurde ein mehrstufiger Planungsansatz realisiert. Mit dem schlangenartigen Roboter KAIRO 3 konnte gezeigt werden, dass in Abhängigkeit von der Umgebung und der aktuellen Roboterkonfiguration Wege über Hindernisse geplant werden können. Neben der Umweltkartierung und -modellierung wurde ein besonderes Augenmerk auf die Optimierung der Kostenfunktion und die Erweiterung des RRT* um einen Sekundären Nächsten Nachbarn Raum (SNN) gelegt, um die Pfadplanung nicht-holonomer Roboter zu verbessern. Eine Erweiterung des Konzepts der virtuellen Schiene ermöglicht die Ausführung der geplanten Pfade mit KAIRO 3.

Die Auswertung ergab, dass sich mit Hilfe der entwickelten Hardwareschnittstelle einzelne oder mehrere Module bzw. Bausteine zu einen bereits bestehenden Roboter hinzugefügt werden können, um dessen Leistungsfähigkeit generativ zu steigern. Beispielsweise ermöglicht die Kopplung mehrerer KAIRO 3 Robotersysteme die Überwindung von Hindernissen, die von den einzelnen Teilsystemen nicht bewältigt werden konnten. Zudem können einzelne Bausteine angedockt werden, die neue Ressourcen wie z. B. Energie oder ergänzende Sensoren in das Gesamtsystem einbringen. Durch das Abkoppeln oder Austauschen defekter Bausteine kann sich ein Roboter selbst aus einer Notsituation befreien. Wurden beispielsweise Module eines Such- und Rettungsroboter von herabfallenden Gegenständen verschüttet worden, können die betroffenen Module abge-

koppelt werden. Der zusätzliche Einsatz von redundanten Modulen steigert die Ausfallsicherheit im Vergleich zu monolithischen Systemen. Dank der redundant ausgelegten Antriebsmodule ist KAIRO 3 beispielsweise in der Lage, einen Defekt in den Abtrieben zu kompensieren. Das defekte Modul kann bei blockierten Rädern angehoben oder bei einem schwerwiegenden Defekt abgekoppelt werden. Der entwickelte Syntheseprozess zur Generierung und Optimierung von Roboterkonfigurationen in Verbindung mit dem interaktiven CARD-Werkzeug ermöglicht die gezielte Vorbereitung von Missionen und Einsätzen mit modularen und rekonfigurierbaren Robotersystemen. Darüber hinaus konnte durch eine Verbesserung der Ausführungszeit eine Online-Nutzung des Syntheseprozesses erreicht werden. Dies erlaubt die Bestimmung einer möglichen Rekonfiguration bei unvorhersehbaren Ereignissen oder Änderungen der Aufgabe während einer Mission. Die mit KAIRO 3 durchgeführten Feldversuche zeigten die Anpassungsfähigkeit der Fortbewegung an sich ändernde Umweltbedingungen und die Flexibilität der Navigationsplanung unter Berücksichtigung unterschiedlicher Roboterkonfigurationen.

8.2 Ausblick

Die Auswertung hat gezeigt, dass sowohl die Synthese von Roboterkonfigurationen als auch die konfigurationsabhängige Navigationsplanung für sich genommen gute Ergebnisse liefern. Durch eine Kombination beider Verfahren könnte die Navigationsplanung unter Berücksichtigung der Rekonfiguration während der Planung die Effizienz der erzeugten Pfade steigern. Auf diese Weise könnte die Anpassungsfähigkeit von modularen Robotern in Zukunft weiter verbessert und der Grad der Autonomie von RMRS erhöht werden.

Eine kontinuierliche Erweiterung des Bausteinkatalogs und der damit verbundenen Fähigkeiten von KAIRO 3 ermöglicht es, in Zukunft weitere Anwendungsfelder zu erschließen. Durch die Erstellung neuer Baustein- und Regelkataloge können die entwickelten Werkzeuge zur Generierung und Optimierung einer Vielzahl von modularen und rekonfigurierbaren Systemen eingesetzt werden, wie das CASD-Tool zur Erstellung modularer Satelliten zeigt.

Darüber hinaus spielen die Flexibilität und die Robustheit der Hard- und Softwarestruktur von RMRS eine wichtige Rolle. Nach der Umsetzung und Validierung der wesentlichen Komponenten eines RMRS in dieser Arbeit ist ein weiterer Meilenstein die automatische Rekonfiguration in Echtzeit. Darüber hinaus können die Randbedingungen für spezielle Missionen und Szenarien erweitert werden, wie z.B. der Einsatz als Unterwasserroboter, bei dem die Dichtheit der Schnittstellen und Komponenten entscheidend ist. Auch bei der Auslegung von Bauteilen und Bausteinen können für den Einsatz im Weltraum besondere Randbedingungen wie Strahlungsabschirmung und extreme Temperaturänderungen berücksichtigt werden.

Basierend auf den in dieser Arbeit entwickelten Methoden und Werkzeugen können in Zukunft flexiblere und intelligentere Robotersysteme für die Suche und Rettung von Menschen, für die Entschärfung von Sprengstoffen sowie die Analyse und Untersuchung von Gefahrstoffen realisiert werden. Darüber hinaus eröffnen diese Systeme neue Anwendungsgebiete für die Robotik, wie z. B. den Einsatz modularer rekonfigurierbarer Satelliten in der Luft- und Raumfahrtindustrie.

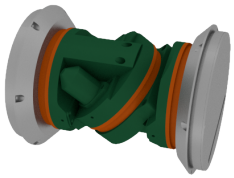
Anhang

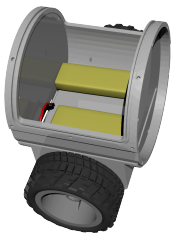
A Anhang

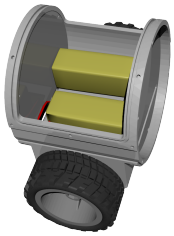
A.1 Modellierte Bausteine und Komponenten für die Konfigurationsgenerierung

A.1.1 Bausteinkatalog KAIRO 3

Zwecks Entwicklung und Evaluierung der in dieser Arbeit vorgestellten Methoden für die Synthese von Roboterkonfigurationen wurde der im Folgenden dargestellte Bausteinkatalog für den Roboter KAIRO 3 modelliert. Der Katalog enthält neben den Bausteine und dessen Eigenschaften ebenfalls die verwendeten Komponenten.

KAIRO 3 Gelenk Modul [id100]		
	Schnittstellen	2
	Abmessungen	0,17 x 0,17 x 0,17 m
	Gewicht	2,50 kg
	Leistungsaufnahme	18,00 W
	Komponenten	
	Keine	

KAIRO 3 Batterie Modul klein [id101]		
	Schnittstellen	2
	Abmessungen	0,17 x 0,17 x 0,17 m
	Gewicht	1,70 kg
	Leistungsaufnahme	20,00 W
	Komponenten	
	1x	Batterie klein

KAIRO 3 Batterie Modul mittel [id102]		
	Schnittstellen	2
	Abmessungen	0,17 x 0,17 x 0,17 m
	Gewicht	2,20 kg
	Leistungsaufnahme	20,00 W
	Komponenten	
	1x	Batterie mittel

KAIRO 3 Batterie Modul groß [id103]		
	Schnittstellen	2
	Abmessungen	0,17 x 0,17 x 0,17 m
	Gewicht	3,00 kg
	Leistungsaufnahme	20,00 W
	Komponenten	
	1x	Batterie groß

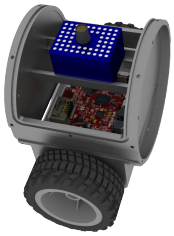
KAIRO 3 Rechner Modul klein [id104]		
	Schnittstellen	2
	Abmessungen	0,17 x 0,17 x 0,17 m
	Gewicht	1,70 kg
	Leistungsaufnahme	20,00 W
	Komponenten	
	1x	Beagleboard

KAIRO 3 Rechner Modul mittel [id105]		
	Schnittstellen	2
	Abmessungen	0,17 x 0,17 x 0,17 m
	Gewicht	1,70 kg
	Leistungsaufnahme	30,00 W
	Komponenten	
	1x	Pandaboard

KAIRO 3 Rechner Modul groß [id106]		
	Schnittstellen	2
	Abmessungen	0,17 x 0,17 x 0,17 m
	Gewicht	1,70 kg
	Leistungsaufnahme	35,00 W
	Komponenten	
	1x	RaspberryPi 3

KAIRO 3 KaRoLa Modul [id107]		
	Schnittstellen	1
	Abmessungen	0,17 x 0,17 x 0,17 m
	Gewicht	3,20 kg
	Leistungsaufnahme	42,00 W
	Komponenten	
	1x	Pandaboard
	1x	Hokuyo UTM-30LX

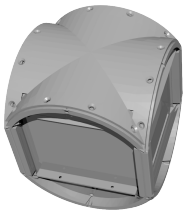
KAIRO 3 Infrarot Kamera Modul [id108]		
	Schnittstellen	1
	Abmessungen	0,17 x 0,17 x 0,17 m
	Gewicht	2,10 kg
	Leistungsaufnahme	32,00 W
	Komponenten	
	1x	Infrarot Kamera

KAIRO 3 TOF Modul Swissranger [id109]		
	Schnittstellen	2
	Abmessungen	0,17 x 0,17 x 0,17 m
	Gewicht	2,20 kg
	Leistungsaufnahme	28,00 W
	Komponenten	
	1x	SwissRanger 300

KAIRO 3 RaspberryPi Kamera Modul [id111]		
	Schnittstellen	1
	Abmessungen	0,17 x 0,17 x 0,17 m
	Gewicht	1,80 kg
	Leistungsaufnahme	20,00 W
	Komponenten	
	1x	Raspberry Pi Kamera

KAIRO 3 Kamera Modul HD [id112]		
	Schnittstellen	2
	Abmessungen	0,17 x 0,17 x 0,17 m
	Gewicht	1,70 kg
	Leistungsaufnahme	20,00 W
	Komponenten	
	1x	Kamera HD

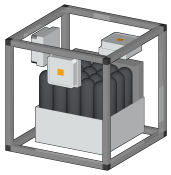
KAIRO 3 Manipulator Modul [id113]		
	Schnittstellen	2
	Abmessungen	0,17 x 0,17 x 0,17 m
	Gewicht	2,00 kg
	Leistungsaufnahme	30,00 W
	Komponenten	
	1x	Manipulator Arm

KAIRO 3 Kreuz Modul [id114]		
	Schnittstellen	4
	Abmessungen	0,17 x 0,17 x 0,17 m
	Gewicht	1,70 kg
	Leistungsaufnahme	20,00 W
	Komponenten	
	Keine	

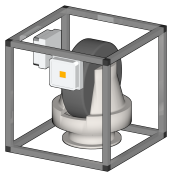
KAIRO 3 Kamera Modul Explosion [id115]		
	Schnittstellen	2
	Abmessungen	0,17 x 0,17 x 0,17 m
	Gewicht	2,30 kg
	Leistungsaufnahme	35,00 W
	Komponenten	
	1x	Beagleboard
	1x	Kamera Explosionssicher

A.1.2 Bausteinkatalog iBOSS

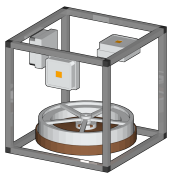
Der folgende Abschnitt zeigt einen Auszug aus dem iBOSS Bausteinkatalog der als Grundlage für die Erzeugung modularer Satelliten dient. Die Modellierung der Bausteine erfolge im Rahmen des iBOSS Projekts.

Batteriebaustein [id162]		
	Schnittstellen	6
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	32,47 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	20,00 W

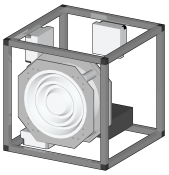
Komponenten	
3x	Hardware-Stack 1
3x	Sonnensensor 1
20x	Li-Ion-Batterie
1x	Bausteinstruktur1x1x1
48x	Versorgungsleitungen
1x	Batteriekontrolle

Control Momentum Gyroscope Baustein [id170]		
	Schnittstellen	6
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	52,05 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	45,00 W

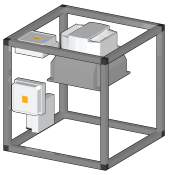
Komponenten	
3x	Hardware-Stack 1
3x	Sonnensensor 1
2x	CMG
2x	Bausteinstruktur1x1x1
80x	Versorgungsleitungen
1x	CMG-Electronics

Drallradbaustein [id178]		
	Schnittstellen	6
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	26,77 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	320,00 W

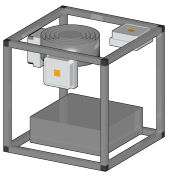
Komponenten	
3x	Hardware-Stack 1
3x	Sonnensensor 1
1x	Bausteinstruktur1x1x1
48x	Versorgungsleitungen
1x	Drallrad (inkl.Elektro.)

elektrischer Antriebs-Baustein [id185]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	104,66 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	10020,00 W

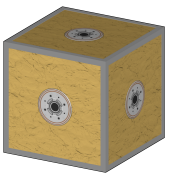
Komponenten	
3x	Hardware-Stack 1
1x	FCU
2x	Sonnensensor 1
1x	Hall Effekt Triebwerk
3x	Bausteinstruktur1x1x1
104x	Versorgungsleitungen
1x	Tank 1
1x	PSCU

Erdsensor-Baustein [id195]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	12,85 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	28,00 W

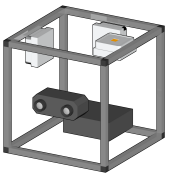
Komponenten	
3x	Hardware-Stack 1
3x	Sonnensensor 1
1x	Bausteinstruktur1x1x1
40x	Versorgungsleitungen
1x	Erdsensor 2

GPS-Baustein (GPS und GLONASS) [id202]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	9,35 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	20,00 W

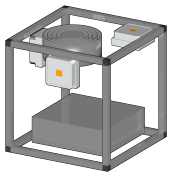
Komponenten	
3x	Hardware-Stack 1
3x	Sonnensensor 1
1x	Bausteinstruktur1x1x1
40x	Versorgungsleitungen

HIL-Baustein (Basis) [id203]		
	Schnittstellen	6
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	9,35 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	20,00 W

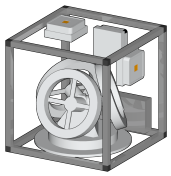
Komponenten	
3x	Hardware-Stack 1
1x	Bausteinstruktur1x1x1
40x	Versorgungsleitungen

HIL-Baustein (PiCam) [id204]		
	Schnittstellen	6
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	9,35 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	20,00 W

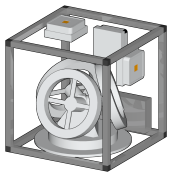
Komponenten	
3x	Hardware-Stack 1
1x	Bausteinstruktur1x1x1
40x	Versorgungsleitungen

GPS-Baustein (Standard) [id208]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	14,00 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	30,00 W

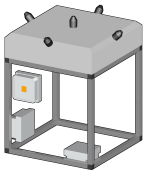
Komponenten	
3x	Hardware-Stack 1
3x	Sonnensensor 1
1x	Bausteinstruktur1x1x1
40x	Versorgungsleitungen
1x	GNSS-Receiver 1
1x	GPS-Antenne

Inertial Measurement Unit Baustein (sehr genau) [id216]		
	Schnittstellen	6
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	27,95 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	44,00 W

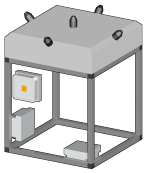
Komponenten	
3x	Hardware-Stack 1
3x	Sonnensensor 1
2x	Bausteinstruktur1x1x1
80x	Versorgungsleitungen
1x	IMU (sehr genau)

Inertial Measurement Unit Baustein (Standard) [id223]		
	Schnittstellen	6
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	14,47 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	35,00 W

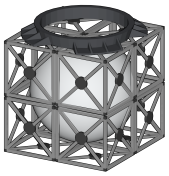
Komponenten	
3x	Hardware-Stack 1
3x	Sonnensensor 1
1x	IMU (Standard)
1x	Bausteinstruktur1x1x1
48x	Versorgungsleitungen

Kaltgasantriebs-Baustein 1 [id230]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	75,98 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	75,00 W

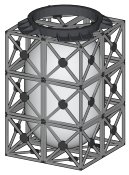
Komponenten	
3x	Hardware-Stack 1
2x	Sonnensensor 1
2x	Bausteinstruktur1x1x1
72x	Versorgungsleitungen
1x	Kaltgasantriebssyst.
1x	Tank 1

Kaltgasantriebs-Baustein 2 [id238]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	76,68 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	75,20 W

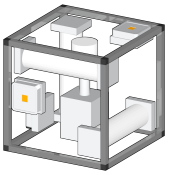
Komponenten	
3x	Hardware-Stack 1
2x	Sonnensensor 2
2x	Bausteinstruktur1x1x1
72x	Versorgungsleitungen
1x	Kaltgasantriebssyst.
1x	Tank 1

Kernstruktur2x2x2 [id246]		
	Schnittstellen	20
	Abmessungen	0,82 x 0,82 x 0,82 m
	Gewicht	72,00 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	0,00 W

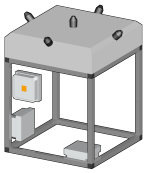
Komponenten	
1x	Tank 3 (Hydrazin)
8x	Bausteinstruktur1x1x1

Kernstruktur2x2x3 [id249]		
	Schnittstellen	32
	Abmessungen	0,82 x 0,82 x 1,23 m
	Gewicht	98,00 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	0,00 W

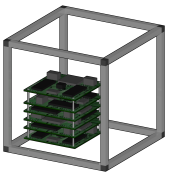
Komponenten	
1x	Tank 3 (Hydrazin)
12x	Bausteinstruktur1x1x1

Magnetorquer-Baustein [id252]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	21,49 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	35,30 W

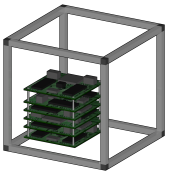
Komponenten	
1x	Magnetometer
3x	Hardware-Stack 1
3x	Magnetorquer
3x	Sonnensensor 1
1x	Bausteinstruktur1x1x1
40x	Versorgungsleitungen

Monopropellantantriebs-Baustein (Hydrazin bzw. HPGP) [id260]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	83,98 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	120,00 W

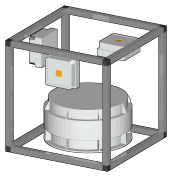
Komponenten	
3x	Hardware-Stack 1
2x	Sonnensensor 1
2x	Bausteinstruktur1x1x1
72x	Versorgungsleitungen
1x	Tank 2
1x	Monopropell.System

OBC/SMU-Baustein [id268]		
	Schnittstellen	6
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	19,77 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	45,00 W

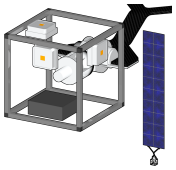
Komponenten	
3x	Hardware-Stack 1
3x	Sonnensensor 1
1x	Bausteinstruktur1x1x1
48x	Versorgungsleitungen
1x	SMU

PCU-Baustein [id275]		
	Schnittstellen	6
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	35,27 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	20,00 W

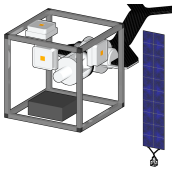
Komponenten	
3x	Hardware-Stack 1
3x	Sonnensensor 1
1x	PCU
1x	Bausteinstruktur1x1x1
48x	Versorgungsleitungen

Reaktionsradbaustein [id282]		
	Schnittstellen	6
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	26,77 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	320,00 W

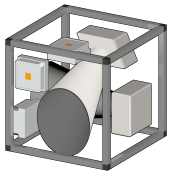
Komponenten	
3x	Hardware-Stack 1
3x	Sonnensensor 1
1x	Reaktionsrad
1x	Bausteinstruktur1x1x1
48x	Versorgungsleitungen

Solargeneratorbaustein (entfaltbar) [id289]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	5,42 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	370,20 W

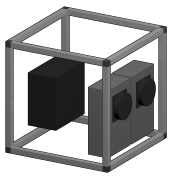
Komponenten	
3x	Hardware-Stack 1
1x	Sonnensensor 2
1x	zentrale Solarfläche
8x	Versorgungsleitungen

Solargeneratorbaustein (entfaltbar, nachführbar) [id295]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	60,82 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	1070,00 W

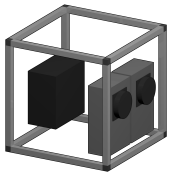
Komponenten	
1x	SADA
3x	Hardware-Stack 1
2x	Sonnensensor 1
5x	entfalt. Solarflächen 2
1x	Bausteinstruktur1x1x1
48x	Versorgungsleitungen
1x	entfalt. Elektronik

Sternsensorbaustein [id304]		
	Schnittstellen	4
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	9,51 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	30,00 W

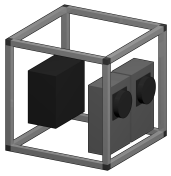
Komponenten	
3x	Hardware-Stack 1
3x	Sonnensensor 1
1x	Sternsensor
1x	Bausteinstruktur1x1x1
24x	Versorgungsleitungen

TM&TC-Baustein (S-Band; Transceiver+SSPA) [id319]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	12,68 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	81,00 W

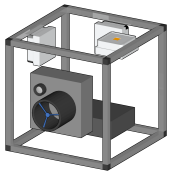
Komponenten	
1x	Antenne
3x	Sonnensensor 1
1x	Transceiver
3x	Hardware-Stack 1
1x	SSPA
1x	Bausteinstruktur1x1x1
40x	Versorgungsleitungen

TM&TC-Baustein (S-Band; Transmitter+Receiver+SSPA) [id328]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	14,26 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	79,00 W

Komponenten	
2x	Antenne
3x	Sonnensensor 1
3x	Hardware-Stack 1
1x	SSPA
1x	Bausteinstruktur1x1x1
40x	Versorgungsleitungen
1x	Receiver
1x	Transmitter

TM&TC-Baustein (S-Band; Transponder+SSPA) [id338]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	14,46 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	96,00 W

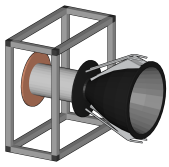
Komponenten	
2x	Antenne
1x	Transponder
3x	Sonnensensor 1
3x	Hardware-Stack 1
1x	SSPA
1x	Bausteinstruktur1x1x1
40x	Versorgungsleitungen

Nutzlast optische Linse [id410]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	0,00 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	25,00 W

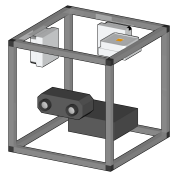
Komponenten	
	Keine

Antenne Small GEO [id432]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	21,50 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	25,00 W

Komponenten	
	Keine

Apogäum Düse [id435]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	0,00 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	25,00 W

Komponenten	
	Keine

Nutzlast Camera [id416]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	20,00 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	7,00 W

Komponenten
Keine

Nutzlast Parabolantenne [id422]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	50,00 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	10,00 W

Komponenten
Keine

Nutzlast Parabolantenne Lang [id42201]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	50,00 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	10,00 W

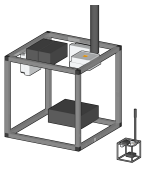
Komponenten
Keine

Nutzlast Parabolantenne Schräg Rechts [id42202]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	50,00 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	10,00 W

Komponenten
Keine

Nutzlast Parabolantenne Schräg Links [id42203]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	50,00 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	10,00 W

Komponenten
Keine

Nutzlast Stabantenne [id428]		
	Schnittstellen	5
	Abmessungen	0,41 x 0,41 x 0,41 m
	Gewicht	0,00 kg
	Wärmekapazität	10,00 J/K
	Leistungsaufnahme	10,00 W

Komponenten
Keine

Tabelle A.1: Definition von Regelklassen

ID	Name	Wirkt auf	Bezüglich KS	Beschreibung
RK1	Global	Alle Bausteine	Global	Globale Regeln für die Bausteine
RK2	Baustein-relativ	Koordinaten-system	Relativer Baustein	Regeln zwischen zwei Bausteinen
RK3	Baustein-spezifisch	Baustein	Roboter Koordinaten-system	Regeln die speziell für einen Baustein gelten

A.2 Implementierte Regeln für die Konfigurationsgenerierung

Tabelle A.2: Definition von Regeltypen

ID	Name	RK	Wirkt auf	Bezug	Messung	Bewertung
RT1	Zusammenhängender Verbund	RK1	Ausgewählte BS	Ausgewählte BS	Anzahl zusammenhängender BS	Algorithmus: Floodfill
RT2	Schwerpunkt	RK1	Verteilung aller BS	Alle BS	Abstand Massen- u. Geometr. Schwerpunkt	Formel: Schwerpunkte
RT3	Flussregeln	RK1	Verteilung aller BS	Ausgewählte BS	Durchleitekapazitäten der BS	Algorithmus: Maximum-Flow-Problem
RT4	Anzahl Verbindungen	RK1	Verteilung aller BS	Alle BS	Anzahl gültiger Verbindungen	Formel: Anzahl
RT5	BS relativer Abstand (Mindest-/ Maximal)	RK2	1 BS	1 BS	Metrische Distanz zwischen BS	Formel: Abstand (Manhattan)
RT6	BS relative Orientierung	RK2	1 BS	1 BS	Rotation in Grad zwischen BS	Formel: Orientierungsoffset
RT7	BS relative Position	RK2	1 BS	1 BS	Translation zwischen BS	Formel: Manhattan Abstand
RT8	Sichtfeld frei (FOV)	RK3	1 BS	Übrige BS	Sichtkegel enthält keinen BS	Algorithmus: Schnitt BS-Hülle / Sichtkegel
RT9	BS spez. Orientierung	RK3	1 BS	System KS	BS Orientierung in Bezug auf System KS	Formel: Orientierungsoffset
RT10	BS spez. Lage im System	RK3	1 BS	System KS	BS Lage in Bezug auf System KS	Formel: Positionsoffset

Tabelle A.3: Definition von Ursachen

ID	Ursache	Überträger	Wechselwirkung
U1	Anfällig für Magnetfelder	Magnetfeld	Beeinflusster
U2	Sensor Fokus	-	Beeinflusster
U3	Sendet Strahlung aus	Strahlung	Verursacher
U4	Sichtfeld (FOV)	Sichtkegel	Beeinflusster
U5	Wärme-abfuhr	Wärme	Beeinflusster
U6	Anfällig für Strahlung	Strahlung	Beeinflusster
U7	hohe Datenrate	Daten	Verursacher

Tabelle A.4: Beispiele konkreter Regeln

ID	Name	Typ	Wirkt auf	Bezug	Ursache	Bewertung
R1	Zusammenhängend	RK1 RT1	Gesamtsystem	Gesamtsystem	keine	Anzahl Cluster = 1
R2	Struktur-BS zusammenhängend	RK1 RT1	Struktur BS	Struktur BS	keine	Anzahl Struktur BS Cluster = 1
R3	Massenschwerpunkt bei geom. Schwerpunkt	RK1 RT2	Gesamtsystem	Gesamtsystem	keine	Proportional Abstand Schwerpunkte
R4	Stromfluss	RK1 RT3	Gesamtsystem	Gesamtsystem	keine	Stromfluss > Schwellenwert
R5	Datenfluss	RK1 RT3	Gesamtsystem	Gesamtsystem	keine	Datenfluss > Schwellenwert
R6	Kompaktes System	RK1 RT4	Gesamtsystem	Gesamtsystem	keine	Gültige Verbind. > Schwellenwert
R7	Gleiche Kamera Orientierung	RK2 RT6	BS1: Kamera links	BS2: Kamera rechts	U2	Offset Orient. = 0
R8	Keine Blendung: Licht-BS zu Kamera-BS	RK2 RT6	BS1: Beleuchtung	BS2: Kamera	U3	Offset Orient. nicht gegengleich
R9	Maximalabstand: Sensor-BS zu Rechner-BS	RK2 RT5	BS1: hohe Datenrate Sensor-BS	BS2: Rechner-BS	U7	Abstand < Schwellenwert des BS
R10	Kamera BS (freies Sichtfeld)	RK3 RT8	BS	Gesamtsystem	U4	Wahr / Falsch
R11	BS im Innern	RK3 RT10	BS	Gesamtsystem	U6	Position im System

A.3 Initialisierung von Individuen zur Konfigurationsgenerierung

Algorithm 1 Erzeugen eines Individuums der initialen Population.

```
    procedure GENERATECOMPOSED(blocks, config_space)
2:      ▷ Initialize empty individual and list of available positions
        Allocate new_individual
4:      pos_avail.add(config_space / 2)
        while blocks.size() > 0 do
6:          ▷ Choose random block index and use first available position
            bb_index ← rand() mod blocks.size()
8:          pos_index ← 0
            pos ← pos_avail[pos_index]
10:         ▷ Choose rotation and position
            calculateRotation(blocks[bb_index])
12:         found ← calulatePosition(blocks[bb_index], pos_avail, pos)
            ▷ Create and add new block
14:         if found then
                new_block ← createNewBlock(blocks[bb_index], pos, orient_in-
    dex)
16:             addNewBlock(new_individual, new_block, pos_avail
                blocks.remove(bb_index)
        return new_individual
```

Algorithm 2 Funktion zur Bestimmung der Position.

```
    function CALCULATEPOSITION(block, pos_avail, pos)
2:      div_index ← 0
        div_pos ← block.getPosDivergences()
4:      while not found and pos_index < pos_avail.size() do
            if gridIsFree(block) and isInsideConfigSpace(block) then
6:              found ← true
            else
8:              ▷ Try all position offsets of bb by moving the block
                if div_index < div_pos.size() then
10:                 div_index++
                else
12:                 pos_index++
                    div_index ← 0
14:             ▷ Set new position
                pos ← pos_avail[pos_index] - div_pos[div_index]
        return found
```

Algorithm 3 Funktion zur Bestimmung der Rotation.

```
    function CALCULATEROTATION(block)
2:      ▷ Initialize list of orientations
        orient_avail ← getCubeOrientations()
4:      ▷ Choose orientation index
        if block.hasConstrOrient() then
6:          orient_index ← block.getConstrOrientIndex()
        else
8:          orient_index ← rand() mod orient_avail.size()
        ▷ Rotate the block to fit inside the config space
10:     while not isInsideConfigSpace(block) and orient_avail.size() > 0 do
            orient_avail.remove(orient_index)
12:         orient_index ← rand() mod orient_avail.size()
```

Algorithm 4 Funktion zum Erzeugen eines neuen Blocks.

```
    function ADDNEWBLOCK(new_individual, new_block, pos_avail)
2:      new_individual.add(new_block)
        addToGrid(new_block)
4:      ▷ Update available positions
        for all neighbor_pos of new_block do
6:          if isGridPosFree(neighbor_pos) and isInsideConfigSpace(new_block)
    then
                pos_avail.add(neighbor_pos)
8:      pos_avail.remove(pos_index)
```

A.4 Implementierung der Mutationsoperationen zur Konfigurationsgenerierung

Algorithm 5 Implementierung der *Move*-Operation.

```
    procedure MOVE(index)
2:      save_position ← individual[index].getPosition()
        ▷ Save fitness of best fitting position
4:      best_pos_fitness ← -individual[index].getNeighborCount()
        best_position ← Vector3Zero()
6:      ▷ Iterate through all blocks of individuum to find best fitting position
        for all block ∈ individual and block ! = individual[index] do
8:          ▷ Iterate through all neighbors
            for all neighbor ∈ block.getNeighborCount() do
10:             move_pos ← neighbor.getPosition()
                individual[index].setPosition(move_pos)
12:             ▷ Check if neighbor grid cell is free and inside the config space
                if gridIsFree(block) and isInsideConfigSpace(block) then
14:                 ▷ Search for the best fitting position to move
                    pos_fitness ← getPosFitness(block)
16:                 if pos_fitness > best_pos_fitness then
                        best_pos_fitness ← pos_fitness
18:                     best_position ← move_pos
        ▷ Set block position and rotation to best
20:     individual[index].setPosition(best_position)
        if best_pos_fitness < -individual[index].getNeighborCount() then
22:         ▷ Set block to its old position
            individual[index].setPosition(save_position)
24:         return false
        return true
```

Algorithm 6 Implementierung der *Rotate*-Operation.

```
    procedure ROTATE(index)
2:      block ← individual[index]
        prev_rot_index ← block.getRotationIndex()
4:      removeFromGrid(block)
        max_trials ← 10
6:      while try_index < max_trials do
            rot_index ← 0
8:          rot_count ← block.getAbsPossibleRotationCount()
            if rot_count > 0 then
10:             ▷ Choose randomly from possible rotations
                abs_rot_index ← rand() mod rot_count
12:             rot_index ← block.getAbsPossibleRotationIndex(abs_rot_index)
            else
14:             ▷ Create random rotation along random axis
                rot_index ← rand() mod getMaxRotationCount()
16:         block.setRotationIndex(rot_index)
            ▷ Apply rotation, if valid
18:         if gridIsFree(block) and isInsideConfigSpace(block) then
                addToGrid(block) return true
20:         try_index++
        ▷ Reset block to previous rotation
22:     block.setRotationIndex(prev_rot_index)
        block.addToGrid(block) return false
```

Algorithm 7 Implementierung der *Swap*-Operation.

```
    procedure SWAP(individual, bb_index)
2:      ▷ Randomly choose swapping block
        swap_index ← rand() mod individual.size()
4:      ▷ Create and init buckets of store source and target block ids
        bucket_src.add(bb_index)
6:      bucket_tgt.add(swap_index)
        ▷ Store positions of main swap blocks as zero positions
8:      zero_pos_src ← individual[bucket_src[0]].getPosition()
        zero_pos_tgt ← individual[bucket_tgt[0]].getPosition()
10:     ▷ Collect all blocks for swapping in the source and target buckets
        getSetOfSwappingBlocks(bucket_src, bucket_tgt, zero_pos_src, zero_-
    pos_tgt)
12:     ▷ Remove all blocks of source bucket from the grid
        for all block ∈ bucket_src do
14:         removeFromGrid(block)
        ▷ Remove all blocks of target bucket from the grid
16:     for all block ∈ bucket_tgt do
            removeFromGrid(block)
18:     ▷ Move all source blocks to their new position
        for block ∈ bucket_src do
20:         pos_offset ← blocks.getPosition() - zero_pos_src
            new_pos ← zero_pos_tgt + pos_offset
22:         block.setPosition(new_pos)
            addToGrid(block)
24:     ▷ Move all target blocks to their new position
        for block ∈ bucket_tgt do
26:         pos_offset ← blocks.getPosition() - zero_pos_tgt
            new_pos ← zero_pos_src + pos_offset
28:         block.setPosition(new_pos)
            addToGrid(block)
```

Algorithm 8 Suche nach Quell- und Zielmenge für den Bausteintausch.

```
function GETSETOFSWAPPINGBLOCKS(bucket_src, bucket_tgt, zero_pos_src,
zero_pos_tgt)
    ▷ Fill buckets with source and target block ids for swapping
    src_index, tgt_index ← 0
    while src_index < bucket_src.size() or tgt_index < bucket_tgt.size() do
        block_src ← individual[bucket_src[src_index]]
        block_tgt ← individual[bucket_tgt[tgt_index]]
        pos_src ← zero_pos_tgt + block_tgt.getPosition() - zero_pos_src
        pos_tgt ← zero_pos_src + block_src.getPosition() - zero_pos_tgt
        found_src ← findSwapCand(pos_src, block_tgt, bucket_src, bucket_-
tgt)
        found_tgt ← findSwapCand(pos_tgt, block_src, bucket_tgt, bucket_-
src)
        if not (found_src & found_tgt) then return false
        ▷ Increase source and target indices to check next blocks in the buckets
        if src_index < bucket_src.size() then
            src_index++
        if tgt_index < bucket_tgt.size() then
            tgt_index++
```

Algorithm 9 Hilfsfunktion zum Finden zweier Tauschkandidaten.

```
function FINDSWAPCAND(pos, block, bucket_src, bucket_tgt)
    ▷ Check all grid positions in case of multi-dimensional blocks
    for all pos_offset ∈ block.getPositionOffset() do
        if isGridPosFree(pos + pos_offset) then
            cur_index ← getBlockIndexFromGrid(pos + pos_offset)
            found ← false
            for all bb_index ∈ bucket_src do
                if cur_index = bb_index then found ← true
            ▷ Do not allow the same block in both swap buckets
            for all bb_index ∈ bucket_tgt do
                if cur_index = bb_index then return false
            ▷ Add block index as swap candidate to source bucket
            if not found then
                bucket_src.add(cur_id)
    return true
```

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] M. Yim, W. Shen, B. Salemi, D. Rus, M. Moll, H. Lipson, E. Klavins, and G. Chirikjian, "Modular self-reconfigurable robot systems [grand challenges of robotics]," *Robotics & Automation Magazine, IEEE*, vol. 14, no. 1, pp. 43–52, 2007.

[2] Österreichisches Rotes Kreuz (Landesverband Salzburg), "EU-Taranis 2013 - Int. Katastrophenschutzübung." Online verfügbar unter `http://www.taranis2013.eu/de/`, 2013. [Zugriff am 21.05.2018].

[3] M. Yim, P. J. White, M. Park, and J. Sastra, "Modular Self-Reconfigurable Robots," in *Encyclopedia of Complexity and Systems Science*, pp. 5618–5631, 2009.

[4] S. Murata and H. Kurokawa, "Self-reconfigurable robots [shape-changing cellular robots can exceed conventional robot flexibility]," *Robotics & Automation Magazine, IEEE*, vol. 14, no. 1, pp. 71–78, 2007.

[5] S. S. R. Chennareddy, A. Agrawal, and A. Karuppiah, "Modular Self-Reconfigurable Robotic Systems: A Survey on Hardware Architectures," *Journal of Robotics*, p. 19, 2017.

[6] J. G. Gómez, *Modular robotics and locomotion: application to limbless robots*. PhD thesis, Pdd. Universidad Autonoma de Madrid. Madrid, 2008.

[7] G. Müller, "Entwicklung einer mechanischen Schnittstelle für rekonfigurierbare Robotersysteme anhand von KAIRO-II," Bachelorthesis, FZI Forschungszentrum Informatik, 2012.

[8] M. Gerdes, "Development and Implementation of a Docking Interface for the Reconfigurable Robot KAIRO 3," Bachelorthesis, FZI Forschungszentrum Informatik, 2017.

[9] M. Göller, L. Pfotzer, J. Oberländer, K. Uhl, T. Büttner, A. Roennau, and R. Dillmann, "Rekonfigurierbare Robotersysteme: von der Erde in den Orbit," in *Deutscher Luft- und Raumfahrtkongress (DLRK)*, 2012.

[10] G. Yang, I. Chen, *et al.*, "Task-based optimization of modular robot configurations: minimized degree-of-freedom approach," *Mechanism and machine theory*, vol. 35, no. 4, pp. 517–540, 2000.

[11] Q.-X. Wu, G.-Y. Cao, and Y.-Q. Fei, "Described model of a modular self-reconfigurable robot," in *Proceedings of 2005 International Conference on Machine Learning and Cybernetics*, vol. 1, pp. 188–193, IEEE, 2005.

[12] B. Dong and Y. Li, "Multi-objective-based configuration generation and optimization for reconfigurable modular robot," in *International Conference on Information Science and Technology (ICIST)*, pp. 1006–1010, IEEE, 2011.

[13] L. Winkler, A. Kettler, M. Szymanski, and H. Worn, "The Robot Formation Language - A formal description of formations for collective robots," in *IEEE Symposium on Swarm Intelligence (SIS)*, pp. 1–8, IEEE, 2011.

[14] L. Kunze, T. Roehm, and M. Beetz, "Towards semantic robot description languages," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 5589–5595, May 2011.

[15] C. Leger, *Automated synthesis and optimization of robot configurations: an evolutionary approach*. PhD thesis, Carnegie Mellon University, 1999.

[16] L. Winkler, H. Worn, and A. Friebel, "A distance and diversity measure for improving the evolutionary process of modular robot organisms," in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 2102–2107, IEEE, 2011.

[17] K. Stoy, W. Shen, and P. Will, "Using role-based control to produce locomotion in chain-type self-reconfigurable robots," *Mechatronics, IEEE/ASME Transactions on*, vol. 7, no. 4, pp. 410–417, 2002.

[18] D. Tsakiris, M. Sfakiotakis, A. Menciassi, G. la Spina, and P. Dario, "Polychaete-like Undulatory Robotic Locomotion," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 3018–3023, April 2005.

[19] Z. Bayraktaroglu, A. Kilicarslan, and A. Kuzucu, "Design and control of biologically inspired wheel-less snake-like robot," in *Biomedical Robotics and Biomechatronics, 2006. BioRob 2006. The First IEEE/RAS-EMBS International Conference on*, pp. 1001–1006, Feb 2006.

[20] J. Conradt and P. Varshavskaya, "Distributed central pattern generator control for a serpentine robot," in *International Conference on Artificial Neural Networks (ICANN 2003)*, 2003.

[21] A. M. Bloch, P. S. Krishnaprasad, J. E. Marsden, and R. M. Murray, "Nonholonomic mechanical systems with symmetry," *ARCH. RATIONAL MECH. ANAL*, vol. 136, pp. 21–99, 1996.

[22] G. Chirikjian and J. Burdick, "An obstacle avoidance algorithm for hyper-redundant manipulators," in *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*, pp. 625–631 vol.1, May 1990.

[23] J. Gonzalez-Gomez, H. Zhang, E. Boemo, and J. Zhang, "Locomotion capabilities of a modular robot with eight pitch-yaw-connecting modules," in *9th international conference on climbing and walking robots*, 2006.

[24] I.-M. Chen, S. H. Yeo, and Y. Gao, "Locomotive gait generation for inchworm-like robots using finite state approach," in *Robotica*, 2001.

[25] E. A. Avila, A. M. Melendez, and M. R. Falfan, "An Inchworm-Like Robot Prototype for Robust Exploration," in *Electronics, Robotics and Automotive Mechanics Conference*, CERMA, 2006.

[26] Y. Lü, S. Ma, B. Li, and L. Chen, "Ground Condition Sensing of a Snake-like Robot," in *Proceedings of the 2003 IEEE, Intemational Conference on Robotics,Intelligent Systems and Signal Processing, Changsha, China*, 2003.

[27] Y. N. Khan, P. Komma, K. Bohlmann, and A. Zell, "Grid-based visual terrain classification for outdoor robots using local features," in *IEEE Symposium on Computational Intelligence in Vehicles and Transportation Systems (CIVTS)*, pp. 16–22, 2011.

[28] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, vol. 2, pp. 500–505, 1985.

[29] J. Ziegler and C. Stiller, "Spatiotemporal state lattices for fast trajectory planning in dynamic on-road driving scenarios," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2009.

[30] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* with Provable Bounds on Sub-Optimality," in *In advances in Neural Information Processing Systems (NIPS)*, MIT Press, 2003.

[31] S. M. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," 1998.

[32] J. J. Kuffner and S. M. LaValle, "RRT-Connect: An Efficient Approach to Single-query Path Planning," in *IEEE Int. Conf. on Robotics and Automation (ICRA)*, vol. 2, pp. 995–1001, 2000.

[33] S. Karaman and E. Frazzoli, "Incremental Sampling-based Algorithms for Optimal Motion Planning," in *Proc. of Robotics: Science and Systems II*, (Philadelphia, PA, USA), 2006.

[34] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," *The Int. Journal of Robotics Research*, vol. 30, pp. 846–894, June 2011.

[35] J.-C. Latombe, *Robot Motion Planning*. Boston, MA: Kluwer, 1991.

[36] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006. Available at http://planning.cs.uiuc.edu/.

[37] P. Sanz, "Robotics: Modeling, Planning, and Control (Siciliano, B. et al; 2009) [On the Shelf]," *Robotics Automation Magazine, IEEE*, vol. 16, pp. 101–101, December 2009.

[38] S. A. M. Coenen, "Motion Planning for Mobile Robots - A Guide," 2012.

[39] S. Hirose and M. Mori, "Biologically Inspired Snake-like Robots," in *Robotics and Biomimetics, 2004. ROBIO 2004. IEEE International Conference on*, pp. 1–7, Aug 2004.

[40] G. Chirikjian and J. Burdick, "The kinematics of hyper-redundant robot locomotion," *Robotics and Automation, IEEE Transactions on*, vol. 11, pp. 781–793, Dec 1995.

[41] J. B. Grzegorz Granosik, Malik G. Hansen, "The OmniTread serpentine robot for industrial inspection and surveillance," *Industrial Robot: An International Journal, Vol. 32 Iss: 2*, 2005.

[42] G. Granosik, K. Mianowski, and M. Pytasz, "Wheeeler – Hypermobile Robot," in *Research and Education in Robotics — EUROBOT 2008*, vol. 33, pp. 68–83, Springer Berlin Heidelberg, 2009.

[43] T. Kamegawa, T. Yamasaki, and F. Matsuno, "Evaluation of snake-like rescue robot "KOHGA" for usability of remote control," in *Safety, Security and Rescue Robotics, Workshop, 2005 IEEE International*, pp. 25–30, June 2005.

[44] L. Pfotzer, S. Ruehl, G. Heppner, A. Roennau, and R. Dillmann, "KAIRO 3: A Modular Reconfigurable Robot for Search and Rescue Field Missions," in *Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on*, Dec 2014.

[45] S. Yamada, H. ; Hirose, "Development of Practical 3-Dimensional Active Cord Mechanism ACM-R4," 2006.

[46] K.-U. Scholl, *Konzeption und Realisierung einer Steuerung für vielsegmentige, autonome Kanalroboter*. Verl. für Wissenschaft und Kultur $WiKu - Verl.$ Stein, Berlin, 2003.

[47] G. Granosik and M. Pytasz, "Control Methods for Wheeeler – The Hypermobile Robot," in *Robot Motion and Control 2011* (K. Kozłowski, ed.), vol. 422 of *Lecture Notes in Control and Information Sciences*, pp. 27–37, Springer London, 2012.

[48] B. Murugendran, A. Transeth, and S. Fjerdingen, "Modeling and path-following for a snake robot with active wheels," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pp. 3643–3650, Oct 2009.

[49] C. Altafini, "Some Properties of the General n-Trailer," *International Journal of Control*, vol. 74, pp. 409–424, March 2001.

[50] A. Roennau, G. Heppner, M. Nowicki, and R. Dillmann, "LAURON V: A versatile six-legged walking robot with advanced maneuverability," in *IEEE/ASME Int. Conf. on Advanced Intelligent Mechatronics (AIM)*, pp. 82–87, July 2014.

[51] T. Kano and A. Ishiguro, "Obstacles are beneficial to me! Scaffold-based locomotion of a snake-like robot using decentralized control," in *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 3273–3278, Nov 2013.

[52] T. Kamegawa, R. Kuroki, M. Travers, and H. Choset, "Proposal of EARLI for the snake robot's obstacle aided locomotion," in *IEEE Int. Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–6, Nov 2012.

[53] P. Liljeback, K. Y. Pettersen, Ø. Stavdahl, and J. T. Gravdahl, "Experimental Investigation of Obstacle-Aided Locomotion With a Snake Robot," *IEEE Trans. on Robotics*, vol. 27, pp. 792–800, Aug 2011.

[54] C. Birkenhofer, *Adaptive Steuerung eines mehrsegmentigen Inspektionsroboters*. PhD thesis, FZI Karlsruhe, 2010.

[55] J. Y. D. M. Wembe, "Entwicklung alternativer Antriebsmöglichkeiten für modulare Robotersyteme," Bachelorthesis, FZI Forschungszentrum Informatik, 2012.

[56] M. Ziegenmeyer, *Entwicklung einer semantischen Missionssteuerung für autonome Inspektionsroboter*. PhD thesis, Karlsruhe Institute of Technology, 2011.

[57] K.-U. Scholl, *Konzeption und Realisierung einer Steuerung für vielsegmentige, autonome Kanalroboter*. PhD thesis, FZI Karlsruhe, 2003.

[58] M. Kortmann, S. Rühl, J. Weise, J. Kreisel, T. A. Schervan, H. Schmidt, and A. Dafnis, "BUILDING BLOCK – BASED "iBOSS" APPROACH: FULLY MODULAR SYSTEMS WITH STANDARD INTERFACE TO ENHANCE FUTURE SATELLITES," in *66rd International Astronautical Congress, Jerusalem, Israel.*, 2015.

[59] Quigley, Morgan and Conley, Ken and Gerkey, Brian P. and Faust, Josh and Foote, Tully and Leibs, Jeremy and Wheeler, Rob and Ng, Andrew Y., "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[60] ROS Core developerteam, "ROS 2 Design documentation," 2015.

[61] K. Uhl and M. Ziegenmeyer, "MCA2 – An Extensible Modular Framework for Robot Control Applications," in *Advances in Climbing and Walking Robots: Proceedings of the 10th International Conference on Climbing and Walking Robots*, pp. 680–689, World Scientific, 2007.

[62] Manso, Luis and Bachiller, Pilar and Bustos, Pablo and Núñez, Pedro and Cintas, Ramón and Calderita, Luis, *Simulation, Modeling, and Programming for Autonomous Robots: Second International Conference, SIMPAR 2010, Darmstadt, Germany, November 15-18, 2010. Proceedings*, ch. RoboComp: A Tool-Based Robotics Framework, pp. 251–262. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010.

[63] C. Schlegel and R. Worz, "The software framework SMARTSOFT for implementing sensorimotor systems," in *Intelligent Robots and Systems, 1999. IROS '99. Proceedings. 1999 IEEE/RSJ International Conference on*, vol. 3, pp. 1610–1616 vol.3, 1999.

[64] L. Nachtigall, "Entwicklung einer flexiblen Steuerung für modulare selbstrekonfigurierbare Roboter," Bachelorthesis, FZI Forschungszentrum Informatik, 2016.

[65] Object Management Group, "Data Distribution Service (DDS) - Specifications," 2015.

[66] Maruyama, Yuya and Kato, Shinpei and Azumi, Takuya, "Exploring the Performance of ROS2," in *Proceedings of the 13th International Conference on Embedded Software*, EMSOFT '16, (New York, NY, USA), pp. 5:1–5:10, ACM, 2016.

[67] K. Regenstein, *Modulare, verteilte Hardware-Software-Architektur für humanoide Roboter*. PhD thesis, FZI Karlsruhe, 2010.

[68] K. Ibrahimov, "Visual Servoing Based Docking for the Reconfigurable Multi-Robot System KAIRO 3," Bachelorthesis, FZI Forschungszentrum Informatik, 2015.

[69] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, (Los Alamitos, CA, USA), pp. 511–518, IEEE, Apr. 2001.

[70] R. Lienhart and J. Maydt, "An Extended Set of Haar-Like Features for Rapid Object Detection," in *IEEE ICIP 2002*, pp. 900–903, 2002.

[71] E. Marchand, F. Spindler, and F. Chaumette, "ViSP for visual servoing: a generic software platform with a wide class of robot control skills," *IEEE Robotics and Automation Magazine*, vol. 12, pp. 40–52, December 2005.

[72] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen, "From {SHIQ} and {RDF} to OWL: the making of a Web Ontology Language," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 1, no. 1, pp. 7 – 26, 2003.

[73] M. Göller, L. Pfotzer, J. Oberländer, K. Uhl, T. Büttner, A. Roennau, and R. Dillmann, "Modellierung und Konfigurationsgenerierung für bausteinbasierte Satellitensysteme," in *Deutscher Luft- und Raumfahrtkongress (DLRK)*, 2011.

[74] M. Hoppen, M. Rast, M. Schluse, J. Rossmann, L. Pfotzer, and T. Meschede, "XML-Based Modeling Languages and Data Binding for Collaborative Design in Multidisciplinary Teams," in *ECEC 2015 - 21st European Concurrent Engineering Conference* (S. Goncalves, ed.), (Lisbon), pp. 15–22, EUROSEUR-ETI, Ghent, Belgium, April 27-29 2015.

[75] CODE SYNTHESIS TOOLS CC, "XSD: XML Data Binding for C++." Online verfügbar unter `http://www.codesynthesis.com/products/xsd/`. [Zugriff am 16.02.2016].

[76] Jan Oberländer and Nils Berg and Michael Göller and Lars Pfotzer and Klaus Uhl and Arne Roennau and Rüdiger Dillmann, "Softwareunterstützter Designprozess für modulare Satelliten: Auswahl geeigneter Bausteine," in *Deutscher Luft- und Raumfahrtkongress (DLRK)*, 2012.

[77] S. Nickel, O. Stein, and K.-H. Waldmann, *Operations Research*. Springer Gabler.

[78] "GNU Linear Programming Kit." Online verfügbar unter `https://www.gnu.org/software/glpk/`, 2016. [Zugriff am 22.02.2016].

[79] T. Achterberg, T. Berthold, T. Koch, and K. Wolter, "Constraint Integer Programming: A New Approach to Integrate CP and MIP," in *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 5th International Conference, CPAIOR 2008* (L. Perron and M. Trick, eds.), vol. 5015, pp. 6 – 20, 2008.

[80] "IBM ILOG CPLEX Optimizer." Online verfügbar unter `https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/index.html`, 2016. [Zugriff am 22.02.2016].

[81] D. Tsarkov and I. Horrocks, "FaCT++ Description Logic Reasoner: System Description," in *Automated Reasoning* (U. Furbach and N. Shankar, eds.), (Berlin, Heidelberg), pp. 292–297, Springer Berlin Heidelberg, 2006.

[82] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1995.

[83] W. W. Cohen, R. Greiner, Dale, and D. Schuurmans, "Probabilistic Hill-Climbing," in *Proceedings of Computational Learning Theory and 'Natural' Learning Systems*, pp. 171–181, MIT Press, 1991.

[84] I. Wegener, *Automata, Languages and Programming: 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005. Proceedings*, ch. Simulated Annealing Beats Metropolis in Combinatorial Optimization, pp. 589–601. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005.

[85] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *Computational Science & Engineering, IEEE*, vol. 5, no. 1, pp. 46–55, 1998.

[86] J.-S. Koh and K.-J. Cho, "Omegabot : Biomimetic inchworm robot using SMA coil actuator and smart composite microstructures (SCM)," in *IEEE International Conference on Robotics and Biomimetics*, 2009.

[87] L. Pfotzer, S. Bohn, C. Birkenhofer, and R. Dillmann, "Biologically-Inspired Locomotion with the Multi-Segmented Inspection Robot KAIRO-II," in *Proceedings of CLAWAR2011, 14th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, p. 181, 9 2011.

[88] W. Wang, Y. Wang, K. Wang, H. Zhang, and J. Zhang, "Analysis of the kinematics of module climbing caterpillar robots," in *2008 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, pp. 84–89, July 2008.

[89] L. Pfotzer, S. Bohn, G. Heppner, A. Roennau, and R. Dillmann, "Environment-Dependent Selection of Locomotion Patterns with KAIRO-II," in *Proceedings of CLAWAR2012, 15th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, 2012.

[90] J. Oberlander, S. Klemm, G. Heppner, A. Roennau, and R. Dillmann, "A multi-resolution 3-D environment model for autonomous planetary exploration," in *Automation Science and Engineering (CASE), 2014 IEEE International Conference on*, pp. 229–235, Aug 2014.

[91] L. Pfotzer, S. Klemm, A. Roennau, J. Zöllner, and R. Dillmann, "Autonomous navigation for reconfigurable snake-like robots in challenging, unknown environments," *Robotics and Autonomous Systems*, vol. 89, pp. 123 – 135, 2017.

[92] L. Pfotzer, J. Oberlaender, A. Roennau, and R. Dillmann, "Development and calibration of KaRoLa, a compact, high-resolution 3D laser scanner," in *Safety, Security, and Rescue Robotics (SSRR), 2014 IEEE International Symposium on*, pp. 1–6, Oct 2014.

[93] L. Pfotzer, M. Stähler, A. Hermann, A. Roennau, and R. Dillmann, "KAIRO 3: Moving over stairs & unknown obstacles with reconfigurable snake-like robots," in *Mobile Robots (ECMR), 2015 European Conference on*, pp. 1–6, IEEE, 2015.

[94] S. M. LaValle and J. J. Kuffner, "Randomized Kinodynamic Planning," *International Journal of Robotics Research*, vol. 20, pp. 378–400, May 2001.

[95] A. Qureshi, S. Mumtaz, K. Iqbal, B. Ali, Y. Ayaz, F. Ahmed, M. Muhammad, O. Hasan, W. Y. Kim, and M. Ra, "Adaptive Potential guided directional-RRT," in *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1887–1892, Dec 2013.

[96] M. Stähler, "Umweltbasierte Bewegungsplanung zur Fortbewegung schlangenartiger Roboter in unebenem Gelände," Masterthesis, FZI Forschungszentrum Informatik, 2014.

[97] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1982.

[98] B. H. Siciliano, ed., *Springer handbook of robotics : with 84 tables*. Berlin: Springer, 2008.

[99] G. Sánchez and J.-C. Latombe, "A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking," in *Robotics Research* (R. A. Jarvis and A. Zelinsky, eds.), vol. 6 of *Springer Tracts in Advanced Robotics*, pp. 403–417, Springer Berlin Heidelberg, 2003.

[100] J. A. Reeds and L. A. Shepp, "Optimal paths for a car that goes both forwards and backwards.," *Pacific Journal of Mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[101] K. Yang, "An efficient Spline-based RRT path planner for non-holonomic robots in cluttered environments," in *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, pp. 288–297, May 2013.

[102] J. Cortés, S. Martínez, J. P. Ostrowski, and K. A. Mcisaac, "Optimal gaits for dynamic robotic locomotion," *International Journal of Robotic Research*, pp. 707–728, 2001.

[103] F. Matsuno and K. Mogi, "Redundancy controllable system and control of snake robots based on kinematic model," in *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, vol. 5, pp. 4791–4796 vol.5, 2000.

[104] K.-U. Scholl, J. Albiez, and B. Gassmann, "MCA – An Expandable Modular Controller Architecture," in *3rd Real-time Linux Workshop, Milan*, 2001.

[105] L. Pfotzer, S. Bohn, J. Oberlaender, G. Heppner, A. Roennau, and R. Dillmann, "Cooperative Multi-Robot Localization with Radio-Based Sensors," in *Proceedings of CLAWAR2013, 16th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines*, 2013.