Universität Augsburg
Fakultät für Angewandte
Informatik

Auf dem Weg zu einer flexiblen Produktion: Automatische und kollisionsfreie Bahnplanung für kooperierende Industrieroboter

Dissertation
zur Erlangung des akademischen Grades eines
Doktors der Ingenieurwissenschaften
der Fakultät für Angewandte Informatik
der Universität Augsburg

eingereicht von

Lars Larsen

19. November 2019

Erstgutachter: Prof. Dr. phil. Dr. habil. Jonghwa Kim
Zweitgutachter: Prof. Dr.-Ing. Michael Kupke
Weiterer Gutachter: Prof. Dr. Wolfgang Reif
Tag der mündlichen Prüfung: 04.12.2018

Kurzfassung

Auf dem Weg zu einer flexiblen Produktion: Automatische und kollisionsfreie Bahnplanung für kooperierende Industrieroboter.

Der Einsatz von Carbonfaserverstärktem Kunststoff in der Luftfahrtindustrie hat in den letzten Jahren enorm zugenommen. So bestehen etwa 20 Prozent des Strukturgewichts bei einem Airbus A380 und 50 Prozent bei einem A350 aus diesem Werkstoff. Die Herstellung großer Faserverbundbauteile für die Luftfahrt beinhaltet allerdings noch immer viele manuelle Prozessschritte, die zu geringer Reproduzierbarkeit, hohem Prüfaufwand, notwendiger Nacharbeit bis hin zur verzögerter Bauteilauslieferung führen können. Für eine genaue und schonende Verarbeitung der Kohlenstofffasern-Zuschnitte ist eine automatisierte Produktion notwendig. Für die Herstellung großer Strukturen wie dem Rumpf oder Flügeln sind große Roboteranlagen notwendig, die hohe Investitionskosten mit sich bringen. Sollen sich diese Investitionen lohnen, müssen zukünftige Anlagen wandlungsfähig und schnell für andere Bauteile umkonfigurierbar sein. Ein entscheidender Baustein hierfür ist die automatische Bahnplanung der Roboter anhand der Produktionsdaten aus den Konstruktionsdaten, die in dieser Arbeit entwickelt und an einem zwei Meter langen Rumpfbauteil eines Airbus A320 untersucht wurde. Um diesen Demonstrator zu fertigen, müssen 196 Zuschnitte mit kooperierenden Robotern in einer Werkzeugform positioniert werden. Bei Aufnahme der Forschung hat sich gezeigt, dass die aktuell einzige Möglichkeit darin besteht, diesen Prozess mit dem manuellen und zeitaufwendigen Teach-in-Verfahren zu programmieren. Bei diesem muss jeder Punkt, der Bestandteil des Pfades sein soll, manuell durch den Bediener angefahren und programmiert werden. Die Dauer hierfür beträgt im Schnitt etwa einen halben Tag pro Zuschnitt. Zwar bieten Roboterhersteller Pakete wie KUKA RoboTeam, jedoch ist dieses lediglich zur Synchronisation und nicht zur Programmierung der Roboterpfade hilfreich. Die Generierung der eigentlichen Pfade ist damit nach wie vor eine Herausforderung. In dieser Arbeit wurde ein System entwickelt, welches eine kollisionsfreie Pfadplanung für mehrere Roboter in einem flexiblen Produktionsszenario übernimmt. Das System wurde in C# entwickelt. Es berechnet die Pfade zunächst offline in einer Simulationsumgebung. In einem nächsten Schritt werden die Pfade auf eine Roboterzelle mit zwei KUKA QUANTEC KR210 R3100 ultra Robotern auf einer gemeinsamen Linearachse ausgeführt. Die Berechnung eines Pfades für einen Zuschnitt dauert mit dem System lediglich eine Minute, was eine immense Verbesserung gegenüber dem manuellen Programmieren oder einer Offline-Programmierung bedeutet.

Schlagwörter: Pfadplanung, Kooperierende Industrieroboter, Sampling-basierte Algorithmen, Evolutionärer Algorithmus, Geschlossene kinematische Kette, Produktionstechnik, Leichtbau

Abstract

Towards a flexible production: Automatic and collision-free path planning for cooperating industrial robots.

The use of carbon fibre reinforced plastic in the aviation industry has increased enormously in recent years. Approximately 20 percent of the structural weight of an Airbus A380 and 50 percent of an A350 are made of this material. However, the production of large fiber composite components for the aerospace industry still involves many manual process steps, which can lead to low reproducibility, high inspection effort, necessary rework and even delayed component delivery. For an automated production it is necessary to process the carbon fibre textile very precise and gentle. For the production of large structures such as the fuselage or wings, large robot systems are necessary, which entail high investment costs. If these investments should be profitable, future systems must be adaptable and quickly reconfigurable for other components. A major component for such systems is the automatic path planning of the robots based on the production data from the computer-aided design, which was developed in this work and examined on a two-meter long fuselage component of an Airbus A320. To produce this demonstrator, 196 blanks must be positioned in a tool mould with cooperating robots. Research has shown that the only current possibility is to program this process with the manual and time-consuming teach-in method. In this case, each point that should be part of the path must be manually approached and programmed by the operator. This takes about half a day on average per cut. Although robot manufacturers offer packages like KUKA RoboTeam, this is only helpful for synchronizing the robots and not for programming the robot paths. The generation of the actual paths is therefore still a challenge. In this work a system was developed, which executes a collision-free path planning for several robots in a flexible production scenario. The system was developed in C#. It first calculates the paths offline in a simulation environment. In the next step, the paths are executed on a robot cell with two KUKA QUANTEC KR210 R3100 ultra robots on a common linear axis. The calculation of a path for a cut takes only one minute with the system, which is an immense improvement compared to manual programming or off-line programming.

Key words: Path Planning, Cooperating Industrial Robots, Sampling-based Algorithms, Evolutionary Algorithm, Closed Kinematic Chain, Manufacturing Engineering, Lightweight Construction

Danksagung

Die hier vorliegende Dissertation ist im Zeitraum von 2013 bis 2018 entstanden, während meiner Tätigkeit als wissenschaftlicher Mitarbeiter und Gruppenleiter am Zentrum für Leichtbauproduktionstechnologie (ZLP) des Deutschen Zentrums für Luft- und Raumfahrt (DLR) in Augsburg. Meine Forschungsarbeit haben viele Menschen begleitet, denen ich an dieser Stelle meinen Dank aussprechen möchte.

Mein besonderer Dank gilt meinem Erstgutachter und Doktorvater, Prof. Dr. Dr. Jonghwa Kim. Er stand mir stets zur Seite, auch nach seiner Rückkehr nach Korea, und ließ mich von seiner wissenschaftlichen Erfahrung profitieren. Unsere Diskussionen haben meine Forschung befruchtet und mich persönlich bereichert.

Für die Erstellung des Zweitgutachtens bedanke ich mich bei meinem Abteilungsleiter, Prof. Dr.-Ing. Michael Kupke.

Prof. Dr. Wolfgang Reif hat ein weiteres Gutachten erstellt. Dafür bin ich ihm ausgesprochen dankbar.

Für die Übernahme des Vorsitzes der Prüfungskommission bedanke ich mich bei Prof. Dr. Bernhard Bauer.

Zudem fühle ich mich meinem Mentor und ehemaligen Abteilungsleiter, Dr.-Ing. Wolfgang Dudenhausen, zu Dank verpflichtet. Er hat stets an meine Idee geglaubt und mich in ihrer Realisierung bestärkt.

Ebenso gilt mein Dank meinen KollegInnen am ZLP des DLR in Augsburg für den inspirierenden Austausch und das großartige Arbeitsklima. Besonders bedanke ich mich bei Georg Braun für die herausragende Atmosphäre in unserem Büro und bei Dr.-Ing. Alfons Schuster für unzählige gemeinsame Stunden an den Robotern.

Ferner danke ich meinen Freunden und Kollegen, Dr.-Ing. Frederic Fischer und Clemens Schmidt-Eisenlohr, für die stete Unterstützung, insbesondere bei der Vorbereitung auf meine Disputation.

Mein größter und herzlichster Dank gebührt meiner Familie, allen voran meiner wunderbaren Partnerin, Carolin Säugling. Carolin, ich danke dir aus tiefstem Herzen, dass du den Weg der Promotion in jeder Hinsicht gemeinsam mit mir gemeistert hast.

Lars Larsen

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Der Begriff Industrie 4.0 ist heutzutage in aller Munde und wurde in Anlehnung an die industriellen Revolutionsstufen vorgestellt. Diese beschreiben den Übergang von der Agrar- zur Industriegesellschaft, der in der zweiten Hälfte des 18. Jahrhunderts begann. In der ersten Stufe wurden mechanische Anlagen mithilfe von Wasser- und Dampfkraft eingeführt. Die zweite Stufe beschreibt die Einführung von Massenfertigung mithilfe elektrischer Energie. Die dritte Stufe ist gekennzeichnet durch den Einsatz von Elektronik und Informationstechnologie (IT) zur Automation der Produktion. Die vierte Stufe, Industrie 4.0 genannt, beschreibt die Ablösung klassischer Produktionssysteme, die auf starren Grenzen basieren. In der Industrie 4.0 sollen stattdessen individuelle Produktionsanlagen anhand von Sensordaten autonom und selbst steuernd agieren. Ein Ziel besteht bspw. darin, individuelle Produkte in einer Serienproduktion in einer sehr geringen Losgröße automatisch zu fertigen. Laut [1, S.13] nimmt z. B. aufgrund der Veränderung der Produktion im Automobilbau in Richtung Elektromobilität und Leichtbau die Produktvielfalt enorm zu. Gleichzeitig reduziert sich die Stückzahl pro Modell und Variante, was dazu führt, dass die Komplexität der Produktion steigt. Dies zwingt die Hersteller der Produktionsanlagen dazu, diese für geringe Losgrößen und eine hohe Sortenvielfalt auszulegen. Wo früher zehn Anlagen für zehn Produkte notwendig waren, soll heute im Idealfall eine Anlage für 1000 Produkte stehen. Die Unternehmen müssen daher an ihrer Flexibilität und Wandlungsfähigkeit arbeiten und in Richtung einer smarten Fabrik agieren. Für die großen Unternehmen in Deutschland ist Industrie 4.0 schon jetzt ein wichtiges Schlagwort, bei dem es gilt, nicht den Anschluss zu verlieren. Harald Krüger, der Vorstandsvorsitzende der BMW Group, sah 2014, damals noch Produktionsvorstand, drei gute Argumente für Industrie 4.0: demografischer Wandel, Zeitgewinn und Qualität [2]. Auch Thomas Enders, der Vorstandsvorsitzende der Airbus Group, hat sich 2015 zu dem Thema geäußert [3]: „Die Möglichkeiten der digitalen Revolution müssen genutzt werden. Dazu gehört, dass die Konzeption, Entwicklung und Herstellung unserer Produkte wesentlich effizienter und schneller wird. In der Luft- und Raumfahrt erleben wir derzeit dabei eine neue Konkurrenz, die wir so bisher nicht gekannt haben.“.

Auch die Forschungsunion Wirtschaft – Wissenschaft greift in dem 2012 erschienenen Abschlussbericht des Arbeitskreises Industrie 4.0, der den Titel „Deutschlands Zukunft als Produktionsstandort sichern: Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0“ trägt, diese Punkte auf [4]. Darin heißt es, die klassische Erzeugung von Funktionsabläufen in Produktionssystemen durch manuelle und iterative Programmierzyklen müsse überdacht werden. Es wird vorgeschlagen, durch die Anwendung und Adaption kognitiver Basistechnologien aus der Welt der Künstlichen Intelligenz (KI) erhebliche Potentiale für die softwareintensive Inbetriebnahme und den Betrieb im Maschinen- und Anlagenbau zu erschließen. Eine weitere zentrale Forderung des Arbeitskreises Industrie 4.0 ist die übergreifende Betrachtung von Planung und Betrieb. Hierbei sollen Methoden entwickelt werden, die eine kontinuierliche Echtzeitsimulation und Visualisierung ermöglichen. Bisher wurden Roboter-Anlagen, wie sie z. B. in der Automobilindustrie zum Einsatz kommen, einmal in Betrieb genommen, um über Jahre denselben Prozess auszuführen. Eine Neukonfiguration der Anlage für eine neue Variante oder ein neues Produkt ist mit einem hohen Ressourcenaufwand verbunden. In der Luftfahrtindustrie ist man von diesem hohen Grad der Automatisierung noch weit entfernt. Dies liegt zum einen an den viel geringeren Stückzahlen verglichen mit der Automobilindustrie, zum anderen an der Größe und Komplexität der Bauteile. Abb. 1.1 zeigt sehr anschaulich den Größenvergleich zwischen einem Industrieroboter, wie er häufig in der Automobilindustrie eingesetzt wird, und einem Flugzeug.

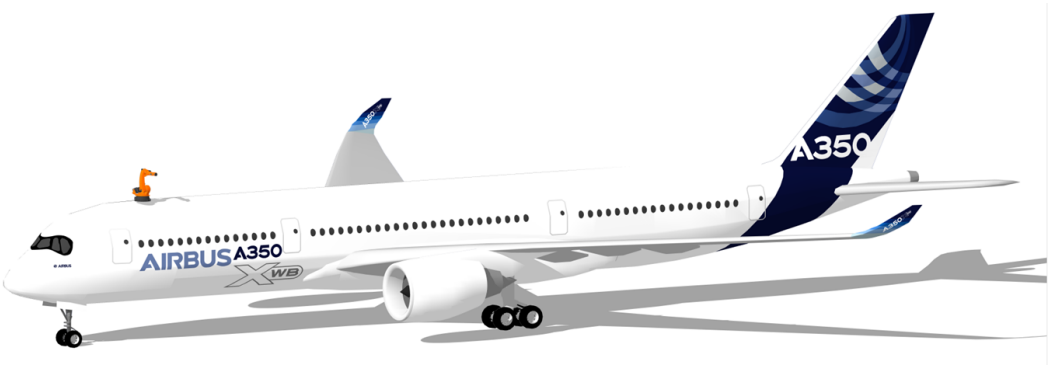
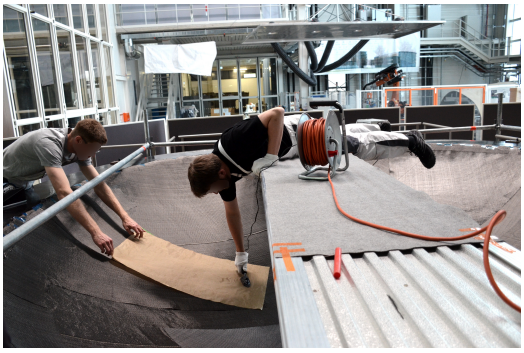


Abbildung 1.1: Größenvergleich zwischen einem Airbus A350-900 XWB und einem Industrieroboter KUKA KR200 (Modelle von 3dwarehouse [5]).

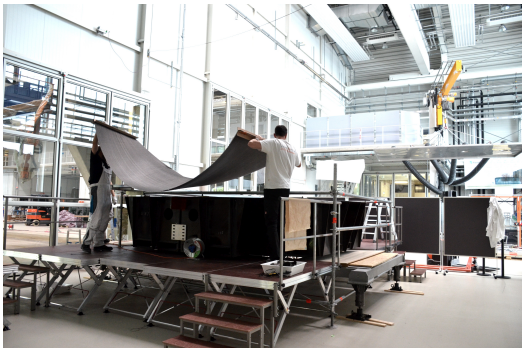
Betrachtet man die Produktion von Bauteilen aus Carbonfaserverstärktem Kunststoff (CFK), findet man häufig Handarbeit vor, was hauptsächlich zwei Ursachen hat. Zum einen sind die Bauteile sehr groß, wodurch Roboter in herkömmlichen Anlagen nicht ausreichend Reichweite haben, zum anderen werden die Bauteile aus hunderten bis tausenden einzelnen Zuschnitten gefertigt, was eine Programmierung sehr zeitaufwendig und somit kompliziert macht.

Abbildung 1.2 zeigt die Herstellung eines Strukturbauteils aus CFK. In mehreren Schichten werden von Werkern die zahlreichen Zuschnitte aus CFK-Multiaxialgewebe in eine Werkzeugform gelegt und am Ende mit Harz infiltriert. Da die Form nicht betreten werden darf, wird über diese eine Brücke gebaut, auf der die Werker auf dem Bauch liegend, Arbeiten durchführen können. Dies ist ergonomisch eine sehr ungünstige Position. Solange die Zuschnitte sehr klein sind, kann ein Werker diese ohne Hilfe positionieren. Bei längeren Zuschnitten müssen mehrere Personen zusammen arbeiten, um einen unbeschadeten Transport des sehr empfindlichen Materials zu garantieren.

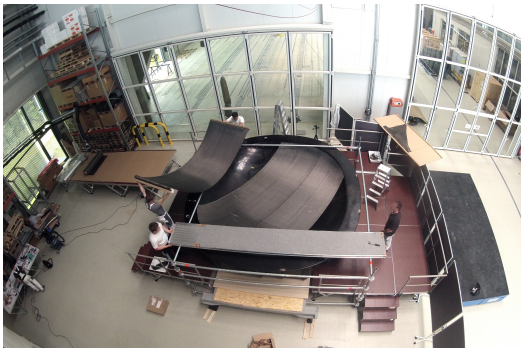
(a) Ablage eines kleinen Zuschnittes



(b) Fixierung eines Zuschnittes



(c) Transport eines großen Zuschnittes



(d) Transport eines großen Zuschnittes

Abbildung 1.2: Manuelle Fertigung eines Strukturbauteils beim DLR.

Würde man den in Abb. 1.2 gezeigten Prozess automatisieren, könnte man dies bei den großen Zuschnitten mit Robotern machen, die kooperieren und gemeinsam das Material transportieren und ablegen.

1.2 Problemstellung

Heute sind Roboter aus Produktionslagen nicht mehr wegzudenken. Aufgrund ihrer relativ hohen Anschaffungskosten, lohnt sich ein Einsatz häufig nur bei großen Stückzahlen, da diese eine höhere Auslastung zur Folge haben, wodurch die Kosten pro zu bearbeitendem Stück geringer ausfallen [6, S.24]. Daher werden Roboter heute bei mehr als 50 Prozent der Betriebe in Deutschland für die Großserienfertigung eingesetzt und lediglich bei 20 Prozent für eine Einzelserienfertigung [6, S.25]. Ein weiterer Grund neben Beschaffungskosten sind die Kosten für die Inbetriebnahme, bei der die Bahnprogrammierung der Roboter einen großen Teil ausmacht. Spätere Anpassungen des Bauteils können hierbei nochmals einen hohen Umfang für die Bahnprogrammierung mit sich bringen. Um die Produktion von großen Flugzeugbauteilen aus CFK zu automatisieren, wurde in dieser Arbeit ein System entwickelt, welches die Pfade für Industrieroboter automatisch berechnet und es somit erlaubt, in sehr schneller Zeit eine Roboteranlage für ein Bauteil einzurichten. Die besondere Herausforderung des entwickelten Systems ist die Unterstützung von kooperierenden Robotern. Aufgrund der Komplexität dieses Prozesses dauert dessen Inbetriebnahme verglichen

mit einem Prozess mit nur einem Roboter bedeutend länger. Im Folgenden sollen das Bauteil und die Roboteranlage, an dem das in dieser Arbeit entwickelte Kollisionsfreie-Kooperation (KoKo) System validiert wurde, vorgestellt werden.

Aufbau des Demo-Panel-Testbauteils

Bei dem in den Experimenten verwendeten Bauteil handelt es sich um die untere Hälfte des Rumpfes eines Flugzeuges vom Typ Airbus A320. Die Zielgeometrie ähnelt sehr stark einem Zylinder mit einem Durchmesser von 1977 mm und einer Tiefe von 2000 mm. Abbildung 8.1 zeigt die bemaßte Zeichnung der Werkzeugform. Die Enddicke des Laminats des infiltrierten und ausgehärteten Bauteils liegt zwischen 2.4 mm und 4.8 mm. Das Panel wird aus Trockenfaserhalbzeugen aufgebaut, die im nachfolgenden Prozessschritt mit Harz infiltriert werden. Der komplette Lagenaufbau besteht aus 208 rechteckigen Zuschnitten, die sich wie folgt zusammensetzen: acht Hautlagen mit jeweils sieben Zuschnitten (siehe Abb. 1.3a), acht Spantaufdickungslagen ebenfalls mit jeweils sieben Zuschnitten (siehe Abb. 1.3b) sowie acht Stringerbaselagen mit jeweils zwölf Zuschnitten (siehe Abb. 1.3c).

Die mittleren fünf Zuschnitte einer Hautlage haben eine Breite von 1034 mm und eine Länge von 1989 mm. Die äußeren beiden Zuschnitte sind jeweils so bemaßt, dass sie den Ablageumfang von 6204 mm auffüllen. Die Zuschnitte der Verstärkungslage unterscheiden sich lediglich in der Länge von den Hautlagen, die zwischen 784 mm und 706 mm liegt. Die Stringerbaselagen bestehen jeweils aus sechs langen Zuschnitten mit einer Länge von 933 mm bis 958 mm und sechs kurzen Zuschnitten mit einer Länge von 268 mm bis 318 mm. Die Breite der Stringerbasezuschnitte wird mit steigender Lagenanzahl immer geringer, um eine Treppenform im Lagenaufbau zu erhalten, und bewegt sich zwischen 184 mm und 139 mm.

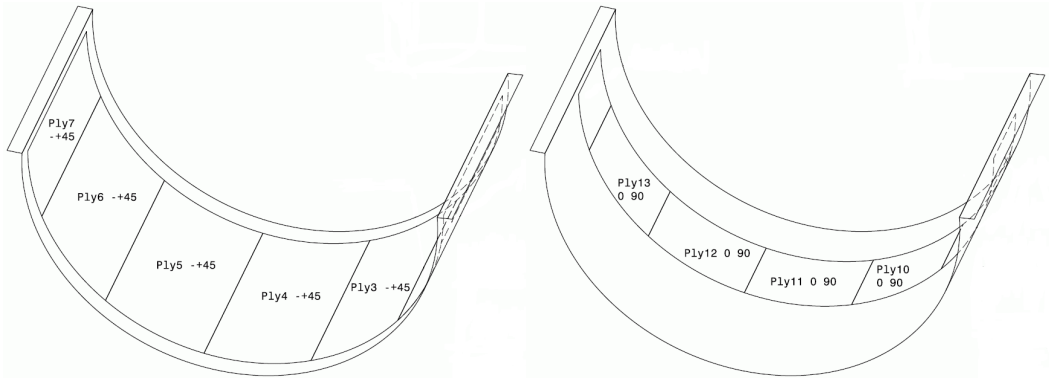
Der Aufbau ist lagenweise so versetzt, dass die Kanten der Zuschnitte nicht übereinander liegen. Da weder die äußeren beiden Zuschnitte der Haut- bzw. Verstärkungslage noch die Zuschnitte der Stringerbaselage mit nur einem Roboter gegriffen werden können, verbleiben 80 Zuschnitte für die Aufnahme mittels kooperierender Roboter.

Als Material für den Lagenaufbau kommt ein CFK-Gelege der Firma SAERTEX zum Einsatz. Dieses hat die Faserorientierungen 0°/90°, +45°/–45° und –45°/+45° mit einem Flächengewicht von 250 g/m^2. Der Lagenaufbau ist so gestaltet, dass im unteren Bereich des Zylinders hauptsächlich 0°/90°-Zuschnitte angeordnet sind, welche später im fertigen Bauteil Zug- und Druckkräfte aufnehmen. Im seitlichen Bereich sind hingegen hauptsächlich die Gelege mit den 45°-Orientierungen verteilt, um Schub und Torsionskräfte aufzunehmen.

Der komplette Fertigungsprozess des Bauteils setzt sich aus den folgenden Schritten zusammen: Preformaufbau, Zwischenkompaktierung, Stringerablage, Vakuumaufbau, Infiltration, Aushärtung, Entformen und Nachbearbeitung. Die in dieser Arbeit durchgeführten Experimente beziehen sich lediglich auf den Prozessschritt des automatisierten Preformaufbaus.

Roboteranlage

Die Validierungsversuche des KoKo-Systems an dem gerade vorgestellten Demo-Panel-Bauteil wurden in der Technologie-Erprobungszelle (TEZ) beim Deutsches Zentrum für Luft- und Raumfahrt (DLR) in Augsburg durchgeführt. Diese besteht aus zwei Industrierobotern vom Typ KUKA QUANTEC KR210 R3100 ultra, die gemeinsam eine 8 m lange Linearachse nutzen. Abbildung 1.4



(a) Hautlage – Sequenz 1 (b) Verstärkungslage – Sequenz 2

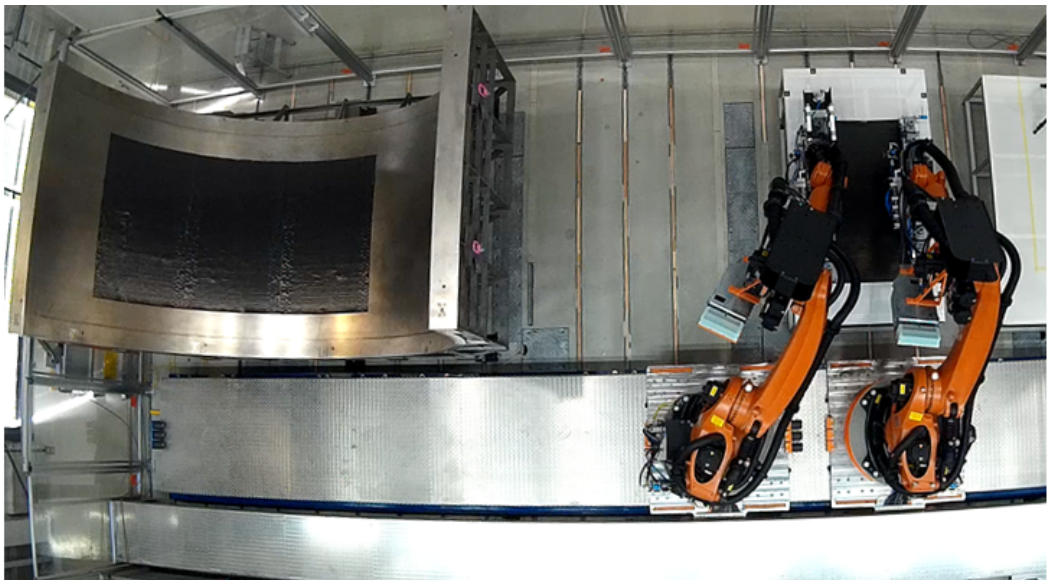
(c) Stringerbaselage – Sequenz 16

Abbildung 1.3: Demo-Panel-Zuschnittstypen.

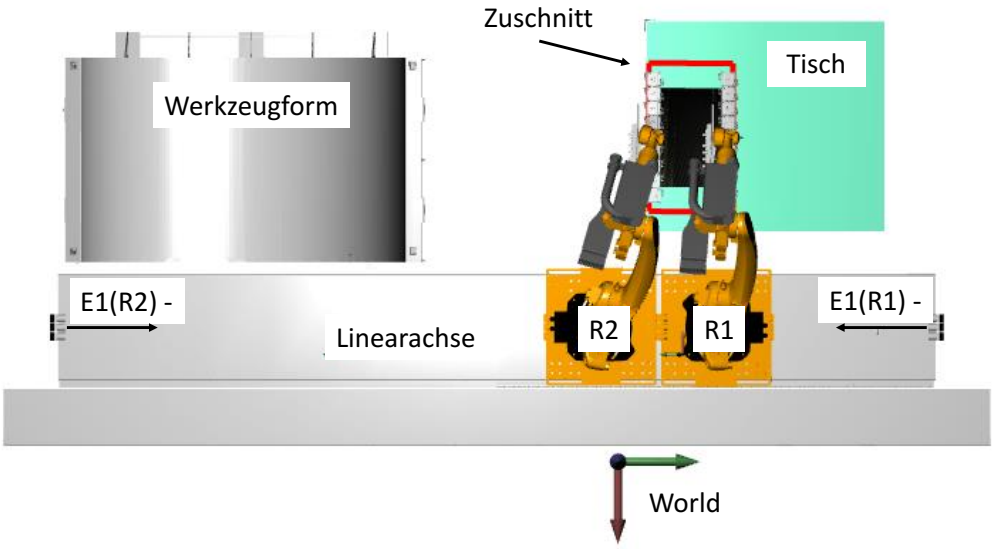
zeigt eine Ansicht von oben auf den Aufbau der Roboterzelle sowohl in der reellen Anlage als auch in der KoKo-Simulationsumgebung.

Im unteren Bereich in horizontaler Ausrichtung ist die Linearachse zu sehen. Für den Roboter R1 - Master-Roboter in der TEZ (R1) befindet sich der Ursprung dieser auf der rechten und für Roboter R2 - Slave-Roboter in der TEZ (R2) auf der linken Seite. Roboter R1 fährt bei einem kleiner werdenden Wert von E1 nach links, R2 nach rechts. In der rechten oberen Ecke des Bildes Abb. 1.4b in Grün dargestellt befindet sich der Aufnahmetisch auf dem ein Zuschnitt positioniert wurde. In der linken oberen Ecke befindet sich die Werkzeugform, in der die Zuschnitte abgelegt werden müssen. Für beide Roboter existiert ein gemeinsames Weltkoordinatensystem (x=rot, y=grün, z=blau), welches unterhalb der Linearachse liegt.

Beide Roboter sind mit einem Greifwerkzeug ausgestattet, welches es erlaubt, die Trockenfaserzuschnitte sicher und schonend zu greifen und zu transportieren. Die Gesamtlänge des Greifers beträgt 1550 mm, wobei dieser aus einzelnen Modulen besteht. Ein Modul ist 160 mm breit und 170 mm lang. Zum Ansaugen des Materials befinden sich auf der Unterseite jedes Moduls acht Silikon-Faltenbalge mit einer Höhe von 55 mm und einem Durchmesser von 31 mm. Durch die Faltenbalge ist es möglich, dass die Greifer beim Ablegen in einem Bereich von 10 mm einfedern können. Da das zu transportierende CFK-Material sehr luftdurchlässig ist, kann es nicht mit Vakuumtechnik gehalten werden. Vielmehr arbeitet der Greifer wie ein Staubsauger, der einen



(a) Foto des Versuchsaufbaus in der reellen Roboteranlage.



(b) Abbildung in KoKo-Simulation.

Abbildung 1.4: Versuchsaufbau in der Technologie-Erprobungszelle (TEZ) beim DLR Augsburg.

hohen Volumenstrom erzeugt. Die Auslegung des Saugsystems wurde in Larsen u. a. [7] entwickelt. Die Haltekraft eines jeden Moduls wird mit einem Förderejektor erzeugt, der aus Druckluft einen hohen Volumenstrom und daraus resultierend einen Druckunterschied erzeugt. Zusätzlich befindet sich an jedem Modul ein linearer Heizstößel, der per Druckluft ausgefahren werden kann. Die Stößel sind beheizt und dienen dazu, das CFK-Material, auf dem ein thermoplastischer Binder aufgebracht ist, an der Ablageposition zu fixieren und somit gegen Verrutschen zu sichern. Um möglichst große Zuschnitte zu transportieren, kommen bei dem Szenario kooperierende Roboter zum Einsatz, die gemeinsam einen Zuschnitt transportieren.

1.3 Struktur der Arbeit

Die vorliegende Arbeit gliedert sich wie folgt: Kapitel 1 startet mit der Motivation für das in dieser behandelte Thema und stellt einen typischen Produktionsprozess eines Strukturbauteils in der Flugzeugfertigung, bei dem Carbonfaserverstärkter Kunststoff (CFK) zum Einsatz kommt, vor. Des Weiteren wird ein Flugzeugrumpf-Testbauteil vorgestellt, das im Laufe der Arbeit als Validierungsszenario dient.

In Kapitel 2 werden die theoretischen Grundlagen für den Bereich der Robotik thematisiert. Diese umfassen z. B. eine Taxonomie von Industrierobotern, die Pose eines starren Körpers, die Kinematik, Bewegungsarten und State of the Art der Roboterprogrammierung. Ferner werden Grundlagen zur Kollisionserkennung und zur Herstellung von Faserverbundbauteilen erläutert.

Kapitel 3 beschäftigt sich mit den Grundlagen der Pfadplanung. Dies beinhaltet zunächst eine Taxonomie verschiedener Verfahren. Sodann werden Sampling-basierte und Natur-inspirierte Verfahren näher erläutert. Abschließend wird noch der Stand der Technik in diesen Bereichen diskutiert.

In Kapitel 4 wird die Entwicklung des im Rahmen dieser Arbeit entwickelten Simulations- und Planungsframeworks vorgestellt. In einem ersten Schritt wird auf die Gesamtarchitektur eingegangen; darauf folgend werden die einzelnen Teile Visualisierung, Planung und Ausführung auf der Hardware beschrieben.

In Kapitel 5 werden Pfadplanungsergebnisse anhand von Simulationsszenarien sowohl für einen Roboter als auch für kooperierende Roboter vorgestellt. Es werden verschiedene Algorithmen getestet und die Ergebnisse im Anschluss verglichen sowie diskutiert.

Der Einsatz des Planungsframeworks für einen CFK-Herstellungsprozess in der Flugzeugproduktion sowie dessen Validierung werden in Kapitel 6 vorgestellt. Bei dem Prozess kommen kooperierende Roboter auf einer Linearachse zum Einsatz, die gemeinsam große Zuschnitte transportieren.

Kapitel 7 fasst die Ergebnisse der Arbeit zusammen und gibt einen Ausblick für weitere Anknüpfungspunkte an diese.

Kapitel 2

Grundlagen

Fortfolgend werden einige wichtige Grundlagen der Robotik sowie der Produktion von CFK erläutert. Diese dienen als Hintergrund für das spätere Planungssystem.

2.1 Einteilung von Industrierobotern

Das englische Wort „robot“ hat eine lange Geschichte und wurde von dem tschechischen Wort „robota“ abgeleitet. Sein erstes Vorkommen findet sich in dem 1921 erschienenen Roman „Rossums Universal Robots“ von Karel Capek. Der Urheber des Wortes, das so viel heißt wie „unermüdliche Arbeit“, war sein Bruder Josef Capek. In Karels Roman sind die Hauptdarsteller unermüdliche Arbeitsmaschinen, die ähnlich wie Menschen aussehen, jedoch fortgeschrittenere Fähigkeiten haben.

Im DUDEN wird „Roboter„ folgendermaßen definiert: „a. (der menschlichen Gestalt nachgebildete) Apparatur, die bestimmte Funktionen eines Menschen ausführen kann; Maschinenmensch - b. (Technik) (mit Greifarmen ausgerüsteter) Automat, der ferngesteuert oder nach Sensorsignalen bzw. einprogrammierten Befehlsfolgen anstelle eines Menschen bestimmte mechanische Tätigkeiten verrichtet.“ [8]. Die Unterscheidung im DUDEN deutet bereits an, dass in der Robotik hauptsächlich zwei Arten von Robotern unterschieden werden: zum einen Roboter, die dem Menschen nachempfunden sind, und zum anderen solche, die einzig dazu dienen, bestimmte Befehlsfolgen in wiederholenden Schleifen auzuführen.

Der offizielle Erfinder des Industrieroboters war 1954 George Devol (1912–2011), der ein Patent für einen programmierbaren Manipulator anmeldete. Offiziell datiert die Erfindung der Industrieroboters auf 1954. In diesem Jahr hat George Devol ein Patent für einen programmierbaren Manipulator angemeldet. Devol gründete anschließend zusammen mit Joseph F. Engelberger die erste Robotikfirma Unimation, die 1961 den Industrieroboter Unimate entwickelte. Dieser wurde erstmals bei General Motors für das Handling von Gussteilen eingesetzt. In der Automobilindustrie kamen ab 1970 die ersten Industrieroboter bei Mercedes-Benz zum Einsatz. Diese waren damals noch mit hydraulischen Zylindern als Antrieb ausgestattet. Mitte der siebziger Jahre des letzten

Jahrhunderts wurden die Industrieroboter mit elektrischen Stellantrieben und einer Mikroprozessorsteuerung ausgestattet; diese kommen noch bei heutigen Modellen zum Einsatz. KUKA stellte seinen ersten Industrieroboter mit dem Namen FAMULUS im Jahre 1973 vor [9]. Dieser hatte sechs elektrisch angetriebene Achsen und wurde hauptsächlich für die Automobilindustrie gebaut. Die Firma Asea Brown Boveri (ABB) präsentierte ihren ersten Roboter nur ein Jahr später, damals noch unter dem Firmennamen Allmänna Svenska Elektriska Aktiebolaget (ASEA) [10].

In [11, S.10] werden die Arten der Roboter noch etwas genauer, in die Gruppe der festen und der mobilen Roboter, unterteilt (siehe Abb. 2.1). Laut der Autoren ist heute die Gruppe der festen (fixed) Roboter, zu der auch die Industrieroboter gehören, zahlenmäßig die größte. Die mobilen Roboter setzen sich aus Robotern auf Rädern, mit Ketten, auf Beinen sowie aus kriechenden Robotern zusammen. Laut den Autoren ist es sehr wahrscheinlich, dass sich die Gruppe der mobilen Roboter mit der Entwicklung von neuen Technologien signifikant vergrößert.

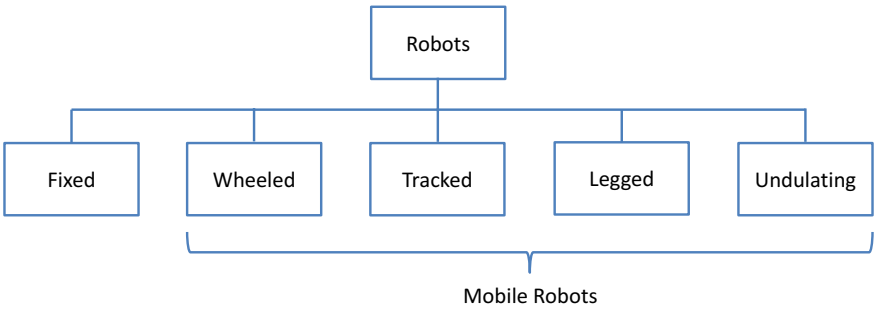
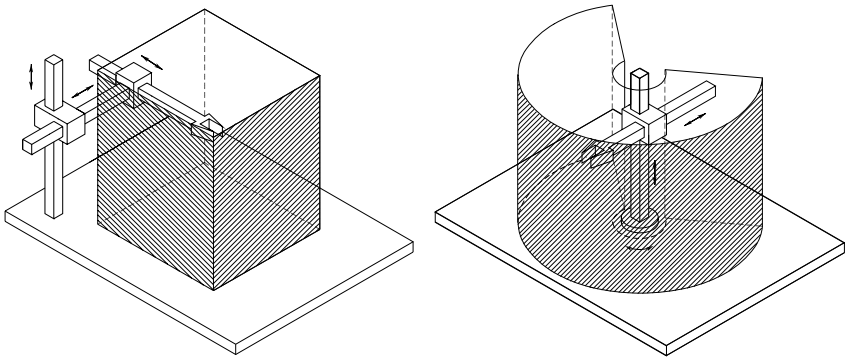


Abbildung 2.1: Übersicht über die Arten von Robotern [11].

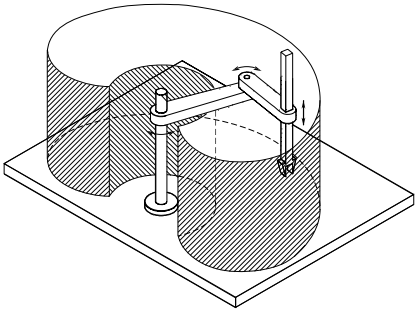
Diese Arbeit legt das Augenmerk ausschließlich auf die Kategorie der festen Roboter bzw. der Industrieroboter. In [11, S. 5-8] werden verschiedene Arten von Industrierobotern vorgestellt (siehe Abb. 2.2). Ein Industrieroboter besteht aus einem Arm, der auf einer Plattform montiert ist und über einen Endeffektor (z. B. Greifer) verfügt. In der Industrie kommen Roboter in den verschiedensten Gebieten zum Einsatz, wie z. B. Schweißen, Palettieren, Lackieren, Handhaben, Klebstoffauftrag, Fräsen, Polieren, Packen usw. Je nachdem welche Bewegungen in dem entsprechenden Anwendungsgebiet gefordert sind, bieten sich unterschiedliche Robotertypen an. Hauptsächlich wird dabei unterschieden, wie die Gelenke untereinander bewegt werden. Die beiden primären Arten sind Translation und Rotation. Abbildung 2.2 zeigt eine Übersicht von Industrierobotern, bei denen die Arten der Gelenke verschieden kombiniert wurden. Portalroboter haben meistens eine sehr massive Bauweise, weshalb sie gerne für schwere Lasten eingesetzt werden. Zudem ist diese Bauform sehr verwindungssteif. Scara-Roboter werden sehr häufig für Pick&Place-Anwendungen verwendet, da die Roboter diese sehr schnell erledigen können. Der in Abb. 2.2d gezeigte Roboter gehört zur Familie der Drehgelenkroboter, welcher 2005 mit 59 Prozent der meistgenutzte Typ war [11, S.9]. Der Vorteil dieser Roboter ist der hohe Freiheitsgrad in ihrem Arbeitsraum. Portalroboter haben bspw. nur drei Freiheitsgrade (X, Y, Z), um eine Position anzufahren. Drehgelenkroboter hingegen haben zusätzlich noch die Möglichkeit, das Werkzeug an jeder Position umzuorientieren (A, B, C). Dies ist bei der Arbeit an komplexen Objekten, wie z. B. Autokarosserien, von Nöten.

Für die Experimente in dieser Arbeit wurden Roboter des Typs QUANTEC KR210 R3100 ultra der Firma KUKA eingesetzt. Diese besitzen im ausgestreckten Zustand einen drei Meter langen

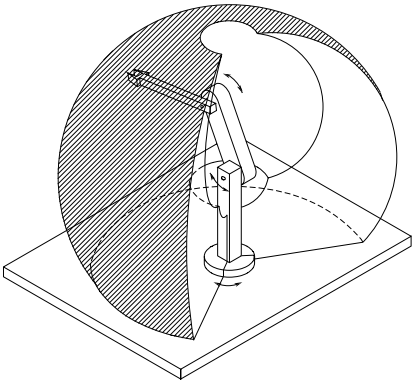


(a) Portal / Kartesisch (TTT)

(b) Zylindrisch (RTT)



(c) SCARA (RRT)



(d) Anthropomorph (RRR)

Abbildung 2.2: Unterschiedliche Industrieroboterarten und deren Arbeitsraum [11, S. 5-8]

.

Arm und einen halbkugelförmigen Arbeitsraum. Die genauen Daten sind in Abschnitt 8.2 und in Abb. 8.2 zu finden.

Bedient werden KUKA-Roboter der QUANTEC-Baureihe, die eine KUKA-Robot-Control (KRC) Steuerung der vierten Generation haben, mit einem smartPAD. Hierbei handelt es sich um ein Bediengerät, welches Knöpfe, ein Touchpad und eine 6D-Maus besitzt (siehe Abb. 2.3). Zum Verfahren des Roboters wird dabei eine der acht Bedientasten auf der rechten Seite oder die 6D-Maus, die ebenfalls auf der rechten Seite, positioniert ist, benutzt. Sonstige Eingaben können allesamt über das eingebaute Touch-Display vorgenommen werden.

2.2 Koordinatensysteme

Jeder starre Körper kann durch seine Position und Orientierung im dreidimensionalen Raum beschrieben werden. Ebenso kann die Position eines Endeffektors, der am Roboter montiert ist, dadurch definiert werden. In der Industrierobotik gibt es verschiedene Koordinatensysteme um die Position des Roboters und des Werkstückes zu beschreiben. Abbildung 2.4 zeigt einen

Abbildung 2.3: SmartPAD-Bediengerät eines KUKA-Roboters.

Industrieroboter mit dessen Robroot-, Welt-, Base- und Tool-Koordinatensystem, die im Folgenden genauer erläutert werden.

Weltkoordinatensystem

Das Weltkoordinatensystem (World) ist ein festes System im Raum, welches das Referenzsystem für den Roboter und die Peripherie in der Roboterzelle ist. Besteht eine Roboterzelle aus mehreren Robotern, dient dieses Koordinatensystem als gemeinsame Basis.

Robroot-Koordinatensystem

Das Robroot-Koordinatensystem befindet sich im Fuß des Roboters. Es liegt bei einem 6-Achs-KUKA-Industrieroboter im Schnittpunkt zwischen der Drehachse der Achse A1 des Roboters und der unteren Fläche des Fußes des Roboters. Die X-Achse zeigt aus Robotersicht nach vorne, die Y-Achse nach links und die Z-Achse nach oben. Das Robroot-Koordinatensystem bildet die Referenz für die mechanische Konstruktion und die Kinematik des Roboters. Im Auslieferungszustand eines Roboters sind das Weltkoordinatensystem und das Robroot identisch.

Werkzeugkoordinatensystem

Das Werkzeugkoordinatensystem (Tool) oder auch Tool Center Point (TCP) sitzt an einem Punkt des Werkzeuges. Standardmäßig befindet sich dieser Punkt in der Mitte der Flangeplatte an Achse 6 des Roboters. Ein typisches Werkzeugkoordinatensystem ist z. B. ein Punkt an der Schweißzange. Somit lässt sich sehr intuitiv ein Bahnpunkt programmieren, da die Bewegung des Roboters

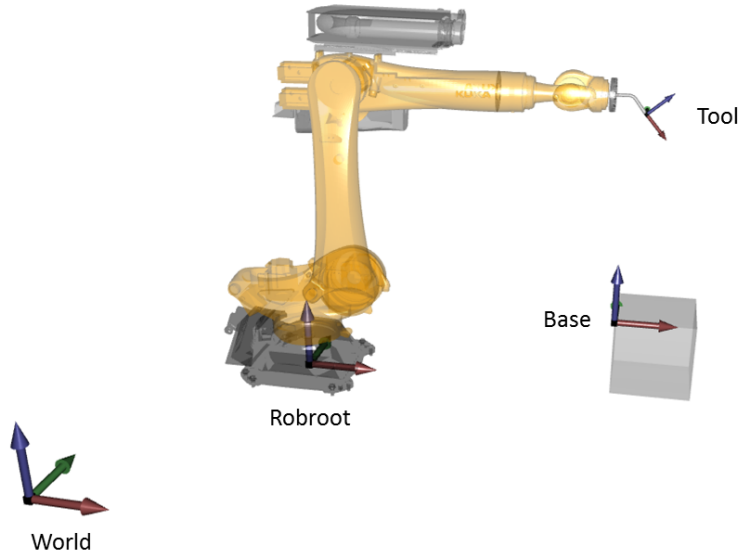


Abbildung 2.4: Kartesische Koordinatensysteme für Roboter (inspiriert von [12, S.54] – Screenshot aus KoKo).

immer bezogen auf diesen Werkzeugpunkt geschieht. Gerade bei Orientierungsänderungen langer Werkzeuge kann dies sehr hilfreich sein. Ändert sich das Werkzeug, da es z. B. verbogen wurde, kann das ursprüngliche Programm weiterhin verwendet werden, indem das Tool neu vermessen wird.

Basiskoordinatensystem

Das Basiskoordinatensystem (Base) wird auf einem Werkstück oder dessen Aufnahme definiert und bildet somit dessen Referenzsystem für ein Werkstück. Alle Punkte eines Roboterprogramms werden normalerweise in Bezug auf dieses Koordinatensystem programmiert. Wird das Werkstück verschoben, kann das Basiskoordinatensystem neu eingemessen werden und alle programmierten Punkte verschieben sich mit. Des Weiteren ist es möglich, identische Werkstücke an verschiedenen Positionen zu bearbeiten.

Ersysroot-Koordinatensystem

Das Ersysroot-Koordinatensystem befindet sich auf dem Ursprungspunkt einer linearen Achse (siehe Abb. 2.5). Normalerweise entspricht dieser Punkt dem Nullpunkt der Achse. Ersysroot erlaubt es, eine Beziehung zwischen dem Ursprung der externen Achse und dem Weltkoordinatensystem herzustellen.

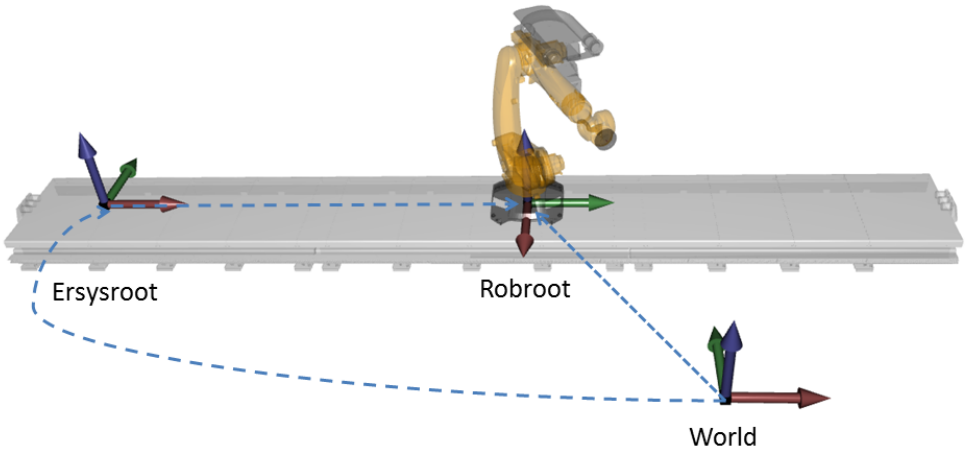


Abbildung 2.5: Kinematische Kette des Robroot auf einer linearen Achse (inspiriert von [13, S.40] – Screenshot aus KoKo)

2.3 Pose eines starren Körpers

Die Pose eines starren Körpers und somit eines Endeffektors lässt sich durch dessen Position und Orientierung bezogen auf ein Referenzkoordinatensystem ausdrücken. Wie in Abschnitt 2.2 beschrieben, existiert in Roboteranlagen dazu im Regelfall ein Weltkoordinatensystem, in dem die Position ausgedrückt werden kann.

Die Position des Punktes O' eines starren Körpers in Bezug auf das Referenzkoordinatensystem O-xyz wird im Allgemeinen durch einen Vektor $\vec{v} \in \mathbb{R}$ ausgedrückt

$$\vec{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

Die Position von O' kann kompakt durch einen (3 x 1)-Vektor ausgedrückt werden:

$$\vec{O'} = \vec{O} + x \cdot \vec{e}_x + y \cdot \vec{e}_y + z \cdot \vec{e}_z$$

wobei $\vec{e}_x$, $\vec{e}_y$ und $\vec{e}_z$ die Einheitsvektoren des Koordinatensystems O sind.

Die einfachste Möglichkeit, eine 3D-Rotation darzustellen, ist, für jede Achse den Drehwinkel anzugeben. Jede beliebige Rotation kann durch diese Schreibweise abgebildet werden. Im Folgenden wird um die X-Achse mit dem Winkel Φ, um die Y-Achse mit Θ und um die Z-Achse mit Ψ rotiert. Die entsprechenden Rotationsmatrizen ergeben sich wie folgt:

$$R_Z(\Psi) = \begin{bmatrix} \cos\Psi & -\sin\Psi & 0 \\ \sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

$$R_Y(\Theta) = \begin{bmatrix} \cos\Theta & 0 & \sin\Theta \\ 0 & 1 & 0 \\ -\sin\Theta & 0 & \cos\Theta \end{bmatrix} \tag{2.2}$$

$$R_X(\Phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\Phi & -\sin\Phi \\ 0 & \sin\Phi & \cos\Phi \end{bmatrix} \tag{2.3}$$

Die Rotationsmatrix ergibt sich durch die Multiplikation der einzelnen Matrizen miteinander:

$$\begin{aligned} R_{Z,Y',X''} &= R_Z(\Psi) \cdot R_Y(\Theta) \cdot R_X(\Phi) \\ &= \begin{bmatrix} \cos\Theta\cos\Psi & \cos\Theta\sin\Psi & -\sin\Theta \\ \sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi & \sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi & \sin\Phi\cos\Theta \\ \cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi & \cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi & \cos\Phi\cos\Theta \end{bmatrix} \end{aligned} \tag{2.4}$$

Der Nachteil an der Rotationsmatrix ist, dass diese neun Einträge hat, um die Orientierung eines Körpers zu beschreiben, obwohl diese mit nur drei Werten definierbar wäre. Daher werden Orientierungen in der Robotik häufiger durch Euler-Winkel (1707–1783) als durch Rotationsmatrizen angegeben. Mit diesen, welche nach dem Mathematiker Leonhard Euler benannt wurden, kann die Orientierung eines starren Körpers im dreidimensionalen Raum, bezogen auf ein Referenzkoordinatensystem, ausgedrückt werden. Die Orientierung wird dabei durch die Abfolge von drei Drehungen hintereinander angegeben. Die erste Drehachse ist dabei raumfest, die nächsten beiden sind mitdrehende Achsen. Je nachdem, in welcher Reihenfolge um die jeweiligen Drehachsen gedreht wird, resultieren unterschiedliche Ergebnisse. Bei den typischen Euler-Winkeln müssen jeweils die erste und die letzte Drehachse übereinstimmen.

- Euler-Winkel (z-x-z, x-y-x, y-z-y, z-y-z, x-z-x, y-x-y)
- Tait–Bryan-Winkel (x-y-z, y-z-x, z-x-y, x-z-y, z-y-x, y-x-z)

Die in der Fahrzeugtechnik, Luftfahrt und Schifffahrt angewendeten Drehfolgen gehören in die Gruppe der Tait-Bryan-Drehungen. In den Normen sind die Formelzeichen Ψ, Θ und Φ (Psi, Theta, Phi) und die Namen Gierwinkel (yaw), Nickwinkel (pitch) und Rollwinkel (roll) für die Euler-Winkel vorgeschrieben.

In dieser Arbeit wird für die Angabe der Orientierung die von KUKA verwendete und in der Literatur Roll-Pitch-Yaw (RPY) genannte Konvention verwendet. Diese wird durch drei aufeinanderfolgende Drehungen um ein raumfestes Bezugskoordinatensystem beschrieben. Die Rotationen werden in folgender Reihenfolge ausgeführt:

- Rotation um die X-Achse des Referenzkoordinatensystems mit Winkel C
- Rotation um die Y-Achse des Referenzkoordinatensystems mit Winkel B
- Rotation um die Z-Achse des Referenzkoordinatensystems mit Winkel A

Dieselbe Rotation kann durch drei aufeinanderfolgende Rotationen um mitdrehende Achsen erreicht werden. In diesem Fall wird bei aufeinanderfolgenden Rotationen immer um die neue Achse rotiert.

- Rotation um die Z-Achse mit dem Winkel A

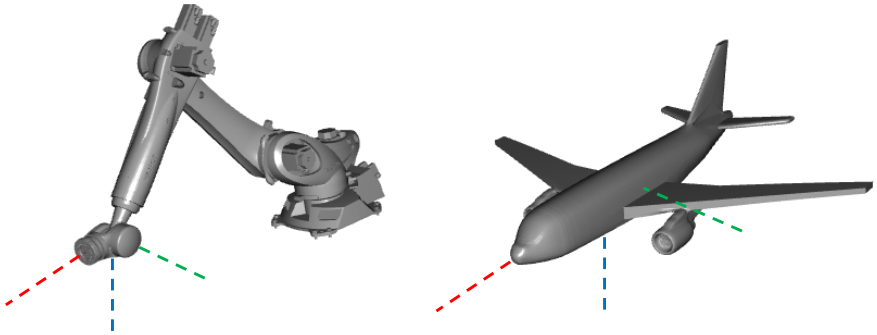


Abbildung 2.6: Roll(rot)-Pitch(grün)-Yaw(blau)-Beschreibung einer Drehung (Screenshot aus KoKo)

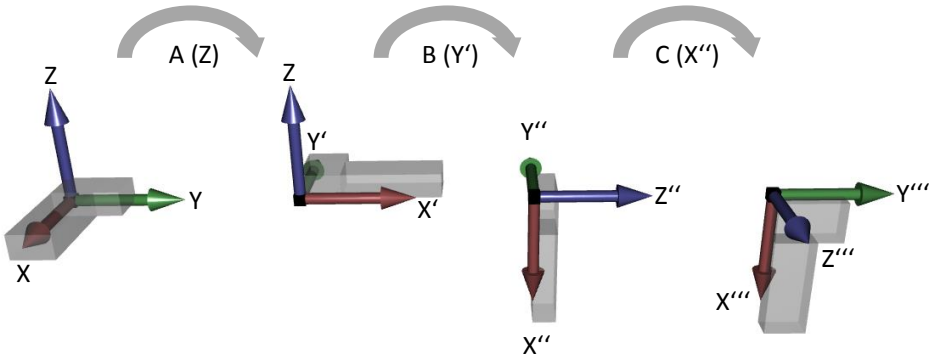


Abbildung 2.7: Beschreibung der schrittweisen Rotation eines Körpers nach Roll-Pitch-Yaw mit mitdrehenden Achsen (inspiriert von [12, S.53] – Screenshot aus KoKo).

- Rotation um die neue Y-Achse mit Winkel B
- Rotation um die neue X-Achse mit Winkel C

In Verbindung mit den Euler-Winkeln können die Position und die Orientierung eines Körpers mit sechs Werten definiert werden. Generell ist es möglich, aus der Rotationsmatrix die dazugehörigen Euler-Winkel eindeutig zu berechnen. Dies ist allerdings nur dann der Fall, wenn der Nickwinkel (pitch) ungleich null ist. Dann stellen Rollwinkel und Gierwinkel dieselbe Drehachse dar und es existieren unendlich viele Lösungen. Diese Stellung wir häufig als Gimbal Lock in der Luftfahrt oder als Singularität in der Robotik bezeichnet.

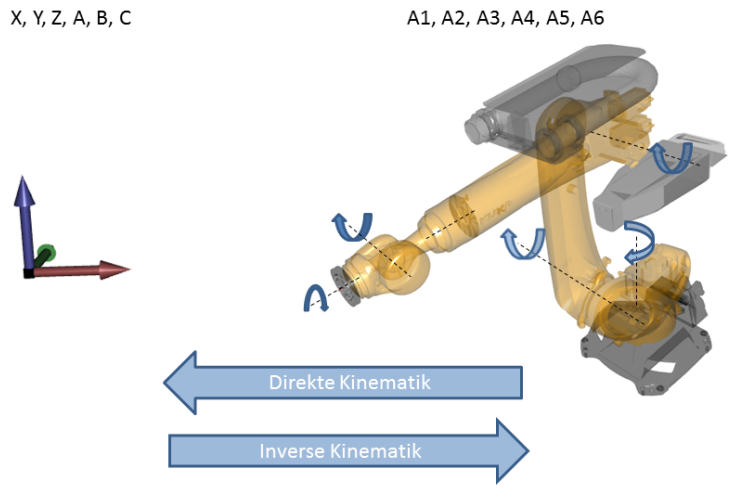


Abbildung 2.8: Direkte- und Inverse- Kinematik eines Roboters (inspiriert von [12, S.52] – Screenshot aus KoKo).

2.4 Kinematik

Eine Stellung des Roboters und somit die Position des TCP lässt sich auf zwei unterschiedliche Arten beschreiben: entweder über die Winkel der einzelnen Achsen (achsspezifisch) oder über die Koordinaten und die Orientierung des TCP (kartesisch) bezogen auf ein Referenzkoordinatensystem. Die Vorwärtskinematik bestimmt anhand der aktuellen Stellung der Roboterachsen die TCP-Position. Die Inverse Kinematik (IK) berechnet bei gegebenem TCP die dazu nötige Stellung der einzelnen Achsen.

Definition 1. Achsspezifisches System: In diesem System wird die Position des Roboters durch die Stellung der einzelnen Achsen beschrieben. Diese sind eindeutig und lassen sich leicht am Roboter ablesen. Um die Position und Orientierung des TCP zu ermitteln, muss die direkte Kinematik berechnet werden. Diese bestimmt aus den Achswinkeln eindeutig die dazugehörigen TCP Werte.

Definition 2. Kartesisches System: Bei Verwendung des Kartesischen Systems wird die TCP-Konfiguration (Position und Orientierung) angegeben. Bei der Erzeugung einer Bewegung bzw. eines Programmes werden alle Positionen durch den TCP ausgedrückt. Um die Achswerte für eine gegebene TCP-Position zu bekommen, muss die Inverse Kinematik berechnet werden. Die Lösung der Inversen Kinematik ist allerdings meistens nicht eindeutig. Es existieren häufig acht Lösungen, von denen die passendste ausgewählt werden muss.

Direkte Kinematik

Die direkte Kinematik für einen seriellen Manipulator berechnet die Position und die Orientierung des Endeffektors relativ zum Basiskoordinatensystem. Die Denavit-und-Hartenberg (DH) Notation ist dabei der De-facto-Standard zur Beschreibung serieller Manipulationen geworden. Die Grundidee besteht darin, dass jeder Roboter als kinematische Kette beschrieben werden kann, die aus prismatischen (sliding) und rotatorischen Verbindungen besteht. Die DH-Parameter werden aus den Abmessungen und dem Versatz der einzelnen Achsen zueinander bestimmt. Dabei wird in den Ursprung jeder Achse ein Koordinatensystem (KS) gelegt, dessen Ursprung und Rotation in Abhängigkeit der vorherigen Achse ausgedrückt wird [14, S.23-24]. Die erforderlichen Parameter a_i, d_i, α_i und Θ_i können anhand der folgenden Schritte ermittelt werden [15, S.119f.]:

- Alle Koordinatensysteme KS_i sind Orthogonalbasen, da die Achsen dieser senkrecht zueinander stehen.
- Die Z-Achse des ersten Koordinatensystems KS_0 liegt in der Drehachse, die Y-Achse ergänzt die Ebene der Orthogonalbasis und die X-Achse steht senkrecht dazu.
- Für alle nachfolgenden Koordinatensysteme $\mathrm{KS}_{i<0}$ gilt:
  1. Z liegt immer auf der Drehachse.
  2. Zwischen den Z-Achsen der aufeinanderfolgenden Achsen KS_i und KS_{i-1} wird die Normale gebildet.
  3. Die X-Achse bildet den Schnittpunkt der Normalen mit der Z-Achse von KS_i.
  4. d_i bezeichnet den Abstand der X-Achse KS_i entlang der Z-Achse KS_{i-1} zu der neuen Normalen. Bei parallelen Gelenken kann dieser Parameter frei gewählt werden.
  5. Θ_i beschreibt die Rotation der alten X-Achse entlang der Z-Achse KS_{i-1}, um in die dieselbe Richtung wie die neue X-Achse zu zeigen.
  6. a_i stellt die Länge der Normalen dar.
  7. α_i ist die Rotation des alten Z um die neue X-Achse, so dass sich die neue Z-Achse ergibt.

Die nach diesem Verfahren bestimmten DH-Parameter für den Roboter KUKA QUANTEC KR 210 R3100 ultra sind in der Tabelle 2.1 zu sehen.

i	a_i	d_i	α_i
1	350	-675	90°
2	1350	0	0°
3	41	0	$-90°$
4	0	-1400	90°
5	0	0	$-90°$
6	0	-240	180°

Tabelle 2.1: DH-Parameter eines Roboters vom Typ KUKA QUANTEC KR 210 R3100 ultra.

Für die Bestimmung des TCP werden die nach dem gerade erläuterten Verfahren gebildeten Rotationsmatrizen nacheinander multipliziert:

$$KS_{TCP}^{Basis} = KS_6^0 = KS_1^0 KS_2^1 KS_3^2 KS_4^3 KS_5^4 KS_6^5 \tag{2.5}$$

Dabei werden die einzelnen Matrizen nach dem in Gleichung (2.7) gezeigten Prinzip erzeugt.

$$KS_i^{i-1} = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

$$KS_{TCP}^{Basis} = KS_6^0 = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

Aus der Rotationsmatrix KS_6^0 können die Koordinaten X, Y und Z sowie die Rotationswinkel A, B und C des TCP bezogen auf die Basis berechnet werden. Für die von KUKA verwendete Roll-Pitch-Yaw-Notation (ZY'X") sieht die Berechnung der Winkel wie folgt aus (siehe [11, S.52]):

Die Lösung für Θ im Bereich $(-\pi/2, \pi/2)$ ist

$$\begin{aligned} \Psi &= \text{atan2}(r_{21}, r_{11}) \\ \Theta &= \text{atan2}\left(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}\right) \\ \Phi &= \text{atan2}(r_{32}, r_{33}) \end{aligned}$$

Die zweite äquivalente Lösung für Θ im Bereich $(-\pi/2, \pi/2)$ ist

$$\begin{aligned} \Psi &= \text{atan2}(-r_{21}, -r_{11}) \\ \Theta &= \text{atan2}\left(-r_{31}, -\sqrt{r_{32}^2 + r_{33}^2}\right) \\ \Phi &= \text{atan2}(-r_{32}, -r_{33}) \end{aligned}$$

Inverse Kinematik (IK)

Die Inverse Kinematik (IK) berechnet aus der Pose (Position und Orientierung) des Roboters die nötigen Achswinkel, um diese zu erreichen. Sie ist somit die Umkehrung der direkten Kinematik. Je nach Robotertyp gibt es verschiedene Verfahren, die IK zu berechnen ([14, S.27-29]). In dieser Arbeit wurde ein analytisches Verfahren aus Brandstötter u. a. [16] implementiert, welches im Folgenden beschrieben wird:

Position:

$$
\begin{aligned}
[c_{x0}\ c_{y0}\ c_{z0}]^T &= [u_{x0}\ u_{y0}\ u_{z0}] - c_4\mathbf{R}_e^0\,[0\ 0\ 1]^T \\
\Theta_{1;i} &= \operatorname{atan2}(c_{y0}, c_{x0}) - \operatorname{atan2}(b, n_{x1} + a_1) \\
\Theta_{1;ii} &= \operatorname{atan2}(c_{y0}, c_{x0}) + \operatorname{atan2}(b, n_{x1} + a_1) - \pi \\
\Theta_{2;i;ii} &= \mp \operatorname{acos}(\frac{s_1^2 + c_2^2 - k^2}{2s_1c_2}) + \operatorname{atan2}(n_{x_1}, c_{z_0} - c_1) \\
\Theta_{2;iii;iv} &= \mp \operatorname{acos}(\frac{s_2^2 + c_2^2 - k^2}{2s_2c_2}) - \operatorname{atan2}(n_{x_1} + 2a_1, c_{z_0} - c_1) \\
\Theta_{3;i;ii} &= \pm \operatorname{acos}(\frac{s_1^2 + c_2^2 - k^2}{2c_2k}) - \operatorname{atan2}(a_2, c_3) \\
\Theta_{3,iii,iv} &= \pm \operatorname{acos}(\frac{s_2^2 + c_2^2 - k^2}{2c_2k}) - \operatorname{atan2}(a_2, c_3)
\end{aligned}
$$

wobei:

$$
\begin{aligned}
n_{x_1} &= \sqrt{c_{x_0}^2 + c_{y_0}^2 - b^2 - a_1} \\
s_1^2 &= n_{x1}^2 + (c_{z_0} - c_1)^2 \\
s_2^2 &= (n_{x_1} + 2a_1)^2 + (c_{z_0 - c_1})^2 \\
k^2 &= a_2^2 + c_3^2
\end{aligned}
$$

Die Konstanten entsprechen den DH-Parametern:

$$
\begin{aligned}
a1 = a_1(DH) = 350, a2 = a_3(DH) = 41, b = 0, c1 = d_1(DH) = 675 \\
c2 = a_2(DH) = 1350, c3 = d_4(DH) = 1400, c4 = d_6(DH) = 240
\end{aligned}
$$

Orientierung:

$$
\begin{aligned}
\Theta_{4;q} = \operatorname{atan2}(&e_{2,3}\cos(\Theta_{1;p}) - e_{1,3})\sin(\Theta_{1;p}),\\
&e_{1,3}\cos(\Theta_{2;p}+\Theta_{3;p})\sin(\Theta_{1;p})+\\
&e_{2,3}\cos(\Theta_{2;p}+\Theta_{3;p})\sin(\Theta_{1;p})-\\
&e_{3,1}\cos(\Theta_{2;p}+\Theta_{3;p}))\\
\Theta_{5;p} = \operatorname{atan2}(&\sqrt{(1-m_p^2)}, m_p)\\
\Theta_{5;q} = &-\Theta_{5;p}\\
\Theta_{6;p} = \operatorname{atan2}(&e_{1,2}\sin(\Theta_{2;p}+\Theta_{3;p})\cos(\Theta_{1;p})+\\
&e_{2,2}\sin(\Theta_{2;p}+\Theta_{3;p})\sin(\Theta_{1;p})+\\
&e_{3,2}\cos(\Theta_{2;p}+\Theta_{3;p}),\\
&-e_{1,1}\sin(\Theta_{2;p}+\Theta_{3;p})\cos(\Theta_{1;p})-\\
&e_{2,1}\sin(\Theta_{2;p}+\Theta_{3;p})\sin(\Theta_{1;p})-\\
&e_{3,1}\cos(\Theta_{2;p}+\Theta_{3;p}))\\
\Theta_{6;q} = &\Theta_{6;p} - \pi
\end{aligned}
$$

wobei:

$$
\begin{aligned}
mp &= e_{1,3}\sin()\cos() + e_{2,3}\sin()\sin() + e_{3,3}\cos()\\
p &= i, ii, iii, iv \; q = v, vi, vii, viii
\end{aligned}
$$

2.5 Bewegungsarten

Punkt zu Punkt (PTP)

Bei Punkt zu Punkt (PTP) Bewegungen im kartesischen Raum muss nur für den Zielpunkt die IK berechnet werden, um die entsprechenden Achsstellungen des Roboters an diesem zu kennen. Dazwischen wird für jede Achse des Roboters ein Bewegungsprofil bestehend aus den folgenden drei Phasen erzeugt: 1. konstante Beschleunigung, 2. konstante Geschwindigkeit und 3. konstante Verzögerung (siehe Abb. 2.10). Bei PTP-Bewegungen ist die Bewegung des TCP zwischen dem Start- und dem Zielpunkt nicht vorhersehbar. Daher ist diese Art der Bewegung nicht für Bahnprozesse, wie z. B. Schweißen, geeignet. Ein großer Vorteil ist allerdings, dass keine Singularitäten auftreten können, da ausgenommen für den Zielpunkt keine IK berechnet werden muss.

Der Vorteil von PTP-Bewegungen ist, dass diese die schnellste Möglichkeit darstellen, um den TCP von der aktuellen zur gewünschten Position zu bewegen. Zudem sind sie schonender für den Roboter und energieeffizienter. Um die Bewegung auszuführen, werden vom Controller die entsprechenden Differenzen der Achsstellungen berechnet.

Gleichzeitige PTP-Bewegungen von mehreren Achsen können folgendermaßen eingeteilt werden:

Abbildung 2.9: Acht verschiedene Lösungen einer Inversen Kinematik eines Roboters vom Typ KUKA QUANTEC KR 210 R3100 ultra für die Position X=1990, Y=0, Z=1984, A=0, B=90, C=0 (Screenshot aus KoKo).

- Lösung 1: A1: 0,00 A2: $-90,00$ A3: 90,00 A4: 0,00 A5: 0,00 A6: 0,00
- Lösung 2: A1: 0,00 A2: $-90,00$ A3: 90,00 A4: $-180,00$ A5: 0,00 A6: 180,00
- Lösung 3: A1: 0,00 A2: 3,85 A3: $-93,35$ A4: 0,00 A5: 89,51 A6: 0,00
- Lösung 4: A1: 0,00 A2: 3,85 A3: $-93,35$ A4: $-180,00$ A5: $-89,51$ A6: 180,00
- Lösung 5: A1: 180,00 A2: $-174,47$ A3: 50,11 A4: $-180,00$ A5: 55,64 A6: 0,00
- Lösung 6: A1: 180,00 A2: $-174,47$ A3: 50,11 A4: $-360,00$ A5: $-55,64$ A6: 180,00
- Lösung 7: A1: 180,00 A2: $-121,66$ A3: $-53,46$ A4: $-180,00$ A5: 4,88 A6: 0,00
- Lösung 8: A1: 180,00 A2: $-121,66$ A3: $-53,46$ A4: $-360,00$ A5: $-4,88$ A6: 180,00

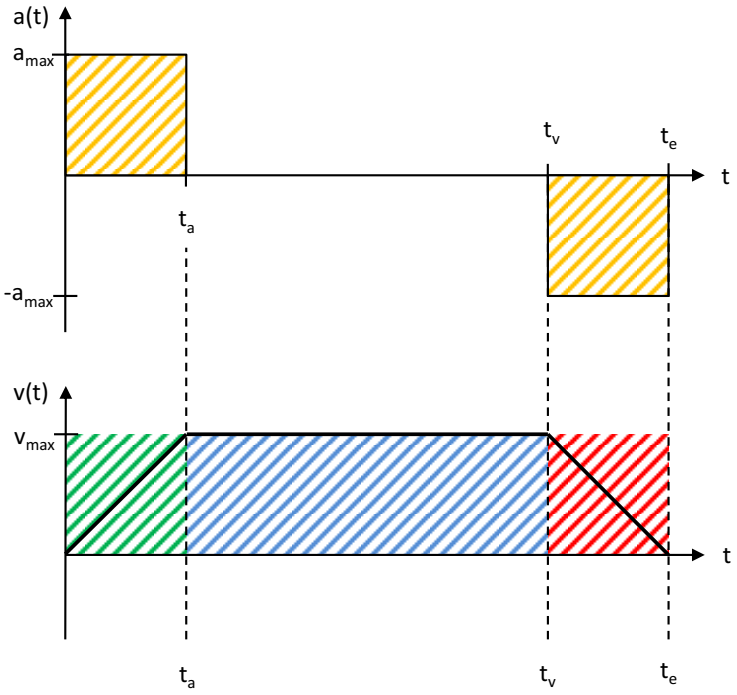


Abbildung 2.10: Beschleunigung einer Roboterachse (inspiriert von [17, S.2]). grün=Beschleunigungsphase, blau=konstante Phase, rot=Verzögerungsphase.

- Asynchron: Bei dieser Bewegungsart ist lediglich der Start der Bewegung der einzelnen Achsen synchronisiert. Während der Bewegung wird keine Synchronisierung vorgenommen. Alle Achsen nutzen ihre maximale Beschleunigung und Geschwindigkeit. Die Achsen erreichen somit zu unterschiedlichen Zeiten ihren Zielpunkt, da alle Achsen unterschiedliche Strecken zurückzulegen haben und verschieden schnell sind. Die Achse, die die längste Zeit benötigt, gibt die Gesamtzeit vor und wird führende Achse genannt.
- Synchron: Bei dieser Bewegungsart ist der Start und das Ende der Bewegung synchronisiert. Die Geschwindigkeit aller Achsen wird auf die der langsamsten und führenden Achse abgestimmt. Die Bahndauer wird im Vergleich zum asynchronen PTP nicht erhöht, die mechanische Beanspruchung jedoch vermindert. Durch die Synchronisierung der Achsgeschwindigkeiten auf eine gemeinsames Ende wird der Energieverbrauch und der Verschleiß des Roboters reduziert, da die Achsen nicht immer mit voller Geschwindigkeit fahren.
- Voll-synchron: Der Verschleiß der Achsen kann noch weiter reduziert werden, wenn nicht nur die reine Dauer der Bewegung, sondern zudem die Dauer der Beschleunigungs- und Bremsphase synchronisiert wird. Die Art der Bewegung wird dann voll-synchron genannt. Um diese Art der Bewegung zu erreichen, müssen die Geschwindigkeiten und die Beschleunigungen aller Achsen angepasst werden.

Bahnbewegungen–kartesische Bahnsteuerung (CP)

Die Linear (LIN) Bewegung ist die einfachste kartesische Bewegungsart. Mithilfe der Linearinterpolation werden zwischen dem Start- und Zielpunkt in einem vorher definierten Abstand Punkte erzeugt, die nach und nach abgefahren werden. Die Zielposition kann dabei ausschließlich in kartesischen Koordinaten angegeben werden, wobei die Position jedes Punktes in einem Werkzeug und Basiskoordinatensystem ausgedrückt werden kann. Für jede Linearbewegung wird eine Geschwindigkeit, Beschleunigung und Überschleifdistanz angegeben. Alle diese Werte beziehen sich immer auf den TCP des Roboters. Diese Art der Bewegung wird z. B. bei Schweißprozessen verwendet, bei denen es wichtig ist, dass der Roboter exakt einer Schweißnaht folgen kann.

Ein Nachteil dieser Bewegungsart sind sogenannte Singularitäten, die auftreten können, wenn es keine eindeutige Lösung für die IK gibt. Somit kann es an diesen Punkten passieren, dass manche Achsen sich bei nur sehr kleinen TCP-Änderungen sehr schnell bewegen. Häufig ist die Bewegung allerdings so schnell, dass die Maximalgeschwindigkeit der Achse überschritten wird und der Roboter in einen STOP geht. Die KUKA-Steuerung unterscheidet drei Positionen, in denen Singularitäten auftreten können [12]: Die Überkopf-Singularität, die auftritt, wenn der Handwurzelpunkt direkt über der Drehachse von A1 steht; die Streck-Singularität die auftritt, wenn Achse A2 und A3 gestreckt sind und eine Linie mit dem Handwurzelpunkt bilden; die Handwurzel-Singularität, die auftritt, wenn die Achsen A4 und A6 der Handwurzel parallel sind. Somit gibt es unendlich viele Möglichkeiten, A4 und A6 zueinander positionieren.

Anstelle einer geraden kann auch eine Circular (CIRC) Bewegung verwendet werden. Um einen Kreis eindeutig zu bestimmen, benötigt man mindestens drei voneinander unabhängige Punkte, die auf der Bahn des Kreises liegen. Den Startpunkt des Kreises stellt dabei die aktuelle Position dar. Zusätzlich muss ein Hilfspunkt auf dem Kreis und ein Zielpunkt angegeben werden.

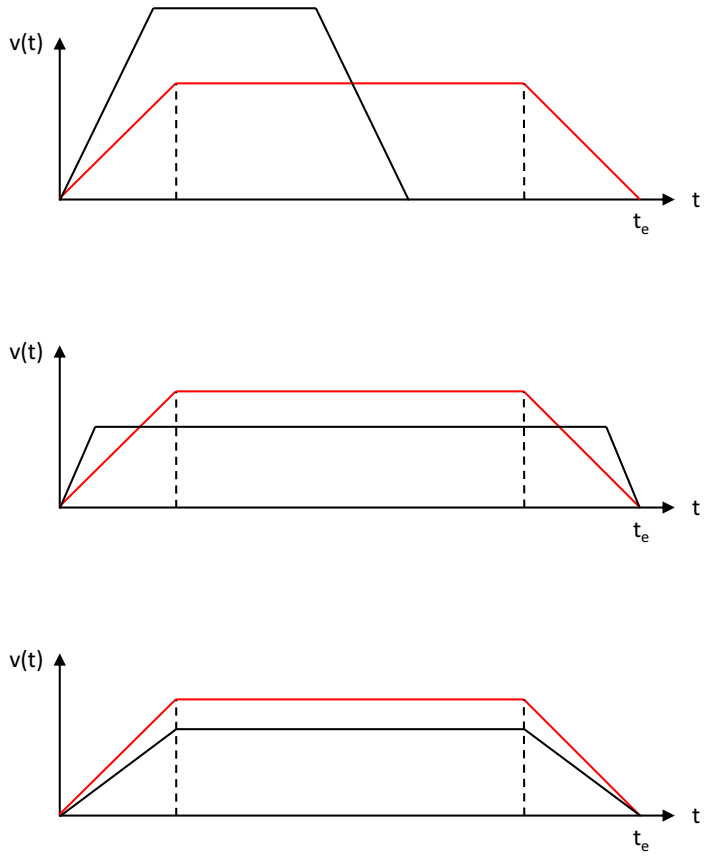


Abbildung 2.11: Beschleunigungstypen bei PTP-Bewegung – asynchron, synchron und voll-synchron. In Rot die Leitachse (inspiriert von [17, S.3]).

Überschleifen

Oft dienen einzelne Punkte einer Bewegung nur als Zwischenpunkte und lediglich der Start- sowie der Endpunkt sollen genau angefahren werden. Die einzelnen Zwischenpositionen einer Bahn müssen nicht exakt erreicht werden. Normalerweise würde ein Roboter bei einer Reihe von Punkten an jedem Punkt anhalten, danach neu beschleunigen und beim nächsten Punkt wieder abbremsen. Um dies zu vermeiden, kann bei einem Industrieroboter durch Parameter angegeben werden, wie stark dieser von einem Zwischenpunkt abweichen darf. Zusätzlich wird der Roboter an diesen Positionen nicht abgebremst. Neben dem zeitlichen Vorteil werden der Verschleiß und Energieverbrauch des Roboters reduziert. Abbildung 2.12 stellt schematisch die Zeitersparnis des Überschleifens dar. Bei der oberen Bewegung wurde nicht überschliffen. Der Roboter muss an jedem Punkt bremsen und neu beschleunigen. Bei der unteren Bewegung hingegen wird einmal

beschleunigt und am Ende der Bewegung wieder gebremst. Man kann sehen, dass dadurch der Ablauf der Bewegung etwa acht Sekunden weniger Zeit beansprucht.

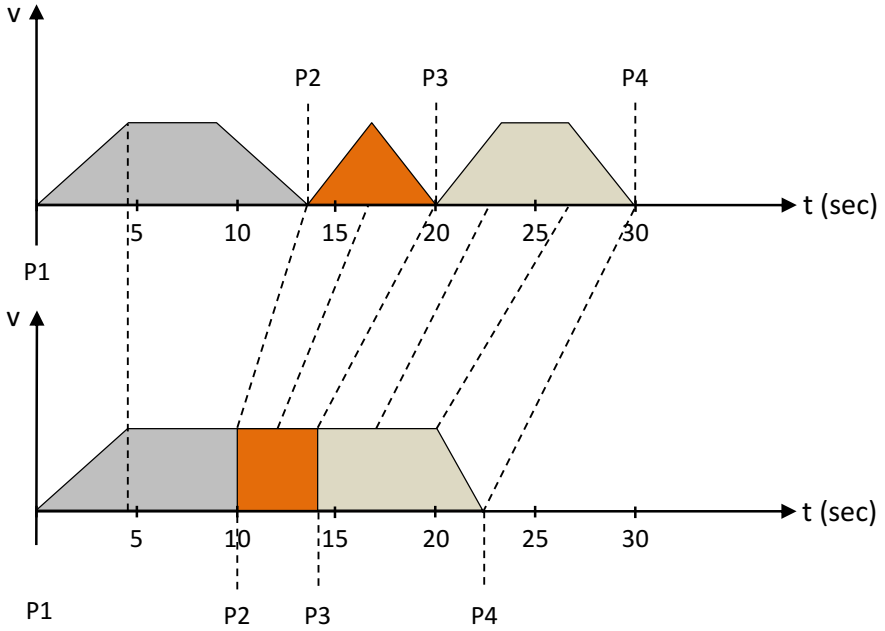


Abbildung 2.12: Zeitersparnis durch Überschleifen. Das obere Bild zeigt eine Bewegung ohne Überschleifen, das untere Bild eine Bewegung mit Überschleifen (inspiriert von [12]).

2.6 Roboter-Programmierung

Es gibt verschiedene Arten, einen Industrieroboter zu programmieren, wobei alle Möglichkeiten entsprechende Vor- und Nachteile haben, die im Folgenden aufgezeigt werden.

Teach-in

Zum Teach-in wird das vom jeweiligen Roboterhersteller zur Verfügung gestellte Bediengerät verwendet. Mit diesem kann der Roboter entweder achsweise oder kartesisch verfahren werden. Je nach Hersteller erfolgt dies über Knöpfe oder über eine 6D-Maus. Wurde der gewünschte Punkt erreicht, kann dieser über vorgefertigte Programmmodule abgespeichert werden. Zusätzlich kann für jeden Punkt beim Anlegen angegeben werden, in welcher Bewegungsart (PTP, LIN, CIRC), Geschwindigkeit, Beschleunigung und Überschleifdistanz dieser angefahren werden soll.

Vorteile:

- Es handelt sich um ein sehr intuitiv und einfach anzuwendendes Verfahren.
- Nur geringe Programmierkenntnisse sind erforderlich.

- Abhängig von der Erfahrung des Programmierers kann eine weitgehend optimale Trajektorie entstehen.
- Kollisionen und Singularitäten können direkt beim Programmieren erkannt werden.
- Die Programme können sofort am echten Roboter getestet werden.

Nachteile:

- Während der Programmierung ist die Roboteranlage belegt.
- Die Erstellung komplexer Programme kann sehr zeitaufwendig sein.

Textuelle Programmierung

Bei der textuellen Programmierung werden die Befehle in einen Editor eingegeben. KUKA-Roboter werden mit der Sprache KUKA Robot Language (KRL) programmiert. Es handelt sich hierbei um eine imperative Sprache, die zur Laufzeit interpretiert wird. Die Sprache unterstützt neben Fahrbefehlen auch gängige Features, wie *IF*-Bedingungen, *FOR*-Schleifen und weitere. Quellcode 2.1 zeigt ein exemplarisches Programm mit einer Auswahl an möglichen Befehlen.

Vorteile:

- Das Programm lässt sich einfach verstehen bzw. nachvollziehen.
- Es können leicht Änderungen am Programm vorgenommen werden.
- Die Erstellung kann ohne Roboter in einem Editor durchgeführt werden.

Nachteile:

- Jeder Hersteller hat seine eigene Programmiersprache, die erlernt werden muss.
- Der Inbetriebnehmer muss Programmiererfahrung haben.

Quellcode 2.1: KUKA Robot Language (KRL) Code-Beispiel

```
1 ; Variablendeklaration
2 DECL FRAME testposition
3 DECL AXIS testaxis
4 DECL INT i
5
6 ; Geschwindigkeit der Achsen setzen
7 FOR i=1 TO 6
8    $VEL_AXIS[i] = 60
9 ENDFOR
10
11 ; PTP Bewegung
12 testaxis = {AXIS: A1 0, A2 -90, A3 90, A4 0, A5 0, A6 0}
13 PTP testaxis
14
15 ; Linearbewegung
16 testposition = {POS: X 100, Y 200, Z 300, A 10, B 20, C 30}
17 LIN testposition
18
19 ; zwei Sekunden warten
20 WAIT SEC 2
21
22 ; Wenn Eingang 10 true dann die Position testaxis anfahren sonst eine Linearbewegung
       ausfuehren
23 if $IN[10] == TRUE THEN
24    PTP testaxis
25 ELSE
26    LIN {X 540,Y 630,Z 1500,A 0,B 90,C 0}
27 ENDIF
28
29 ; eine PTP Bewegung ausfuehren
30 PTP {A1 0,A2 -90,A3 90,A4 0,A5 0,A6 0}
```

Play-Back

Bei dieser Art der Bewegung erfolgt die Programmierung durch Führen des Roboters an die entsprechende Position. Für die Durchführung ist entweder ein spezieller Roboter, wie z. B. der KUKA-Leichtbauroboter (LBR), oder ein speziell ausgerüsteter Industrieroboter notwendig. Der LBR hat in jeder seiner Achsen einen Kraft-Momenten-Sensor eingebaut, der es ermöglicht, die Kraft, mit der auf ihn gewirkt wird, zu erkennen. Bewegen lässt sich der Roboter dann in einem speziellen Modus, in dem er sich ähnlich wie eine Feder verhält. Bei einem Standard-Industrieroboter müssen weitere Vorkehrungen getroffen werden, um diese Art der Programmierung durchführen zu können. Zunächst muss dieser z. B. zwischen Werkzeug und Flansch mit einem

Kraft-Momenten-Sensor ausgestattet werden, der erkennt, wenn auf den End-Effektor eine Kraft ausgeübt wird. Diese Kraft muss von einer externen Steuerung verarbeitet werden, die wiederum Fahrbefehle für den Roboter erzeugt. Diese können dann über eine externe Schnittstelle, wie z. B. das Remote Sensor Interface (RSI) bei KUKA, an die Steuerung übertragen werden. Typische Anwendungen für diese Art der Programmierung sind Spritzlackieren oder Sandstrahlen.

Vorteile:

- Die Bedienung ist sehr intuitiv und schnell, da der Roboter direkt an den entsprechenden Punkt bewegt wird.
- Es sind fast keine Programmierkenntnisse erforderlich.

Nachteile:

- Die Erreichung bestimmter Koordinaten für einen Punkt ist sehr ungenau.
- Eine Änderung bzw. Optimierung einzelner Bahnabschnitte ist schwierig, da das Programm aus einer Liste aus Punkten besteht.
- Die Methode lässt sich nur bei kleinen und für den Bediener gut zugänglichen Arbeitsräumen einsetzen.

Offline-Programmierung (OLP)

Bei dieser Art der Programmierung wird das Roboterprogramm an einem Personal Computer (PC) auf Basis von Konstruktionszeichnungen und einer Simulationsumgebung erstellt. Die Roboterhardware wird während des Programmiervorganges nicht benötigt [18, S.353]. Nach Fertigstellung des Programmes wird dieses per Datenträger auf den Roboter überspielt und auf diesem getestet. Bekannte kommerzielle Beispiele sind: RobCAD, DELMIA und Process Simulate. Abbildung 2.13 zeigt einen Screenshot des Tools DELMIA, welches auf der Konstruktionssoftware CATIA basiert.

Vorteile:

- Bereits vor dem Aufbau der Roboterzelle kann die Programmierung.
- Fehler in der Planung der Roboterzelle können frühzeitig erkannt und beseitigt werden.
- Die kollisionsfreie Erreichbarkeit kann für jeden Punkt ausgiebig getestet werden, ohne die echte Hardware zu beschädigen.
- Es ist nicht nötig, dass sich der Programmierer in der gefährlichen Roboterzelle aufhält.
- Die OLP-Software kann für verschiedene Roboter verwendet werden.
- Beliebig oft können die Programme in der Simulation getestet und optimiert werden.

Nachteile:

- Häufig stimmt das Computer-aided Design (CAD)-Modell nicht exakt mit der Realität überein, da Elemente wie Säulen, Pfeiler usw. fehlen.
- Flexible Leitungen (Energiezuführung) können nicht exakt nachgebildet werden.
- Die Bedienung einer OLP Software ist wesentlich schwieriger als das Teach-in .
- OLP-Software ist in den meisten Fällen sehr teuer.

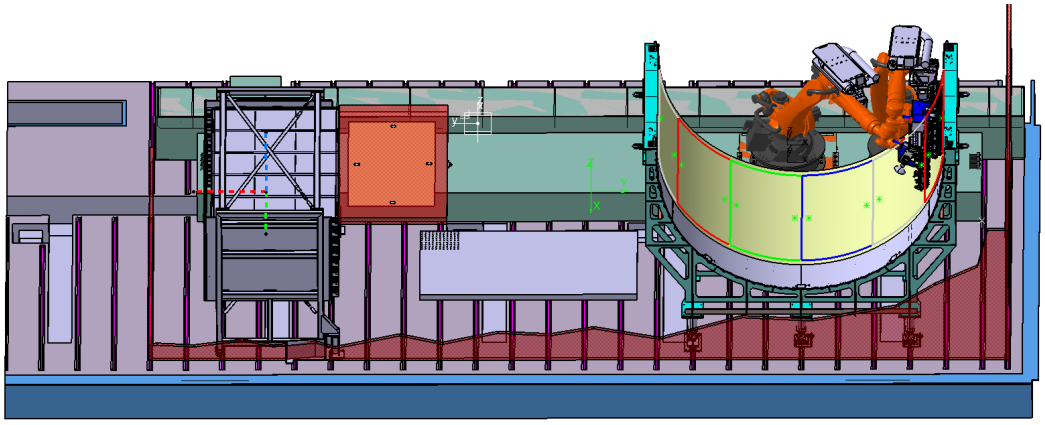


Abbildung 2.13: Abbildung einer Roboteranlage in der OLP-Software DELMIA.

2.7 Arten der Roboterzusammenarbeit

Es gibt verschiedene Arten und Begrifflichkeiten für die Zusammenarbeit von Robotern, sowohl untereinander als auch mit Menschen. Da der Begriff kooperierend häufig im Zusammenhang mit kollaborierend und koordinierend verwendet wird, sollen diese Begriffe zunächst voneinander abgegrenzt werden.

Kollaborierende Roboter

Laut der Definition in der Literatur bedeutet kollaborieren (assistieren), „mit jemandem zusammenzuarbeiten, um etwas zu produzieren“ – es geht also um Assistenz. Kollaborierende Roboter dürfen mit Personen in Kontakt kommen, um sie bei manuellen Tätigkeiten zu unterstützten. Laut der Norm ISO 10218 „Industrieroboter; Sicherheitsanforderungen“ Teil 1 und Teil 2 gibt es verschiedene Arten der Kollaboration, wie z. B. die Handführung, Geschwindigkeitsüberwachung, Kraftüberwachung oder sichere Raumüberwachung.

Kooperierende Roboter

Der Begriff kooperierende Roboter wird für verschiedene Arten der Kooperation verwendet; Diese wird in diesem Abschnitt erläutert werden sollen. In der Literatur gibt es ebenfalls keine einheitliche Definition dieses Begriffes, der das Spektrum von einer Roboterhand mit mehreren Fingern bis hin zu einer Anwendung mit vielen separaten Robotern abdeckt. Ein Vorschlag einer Unterteilung von Koordination und Kooperation zeigt Tabelle 2.2.

Unkoordiniert

In der Industrie sind einzeln arbeitende Roboter die am häufigsten eingesetzte Form. Ein sehr häufiges Einsatzgebiet ist z. B. die Automobilindustrie. Hierbei wird das Werkstück, in dem Fall das Auto, von Station zu Station transportiert und danach von mehreren Robotern gleichzeitig

Unkoordiniert	Koordiniert	
	Ziel-koordiniert	Bimanual
Ein Roboter palettiert, während der andere eine Schweißnaht setzt.	Beide Roboter palettieren auf derselben Palette.	Beide Roboter heben eine Palette zusammen an.

Tabelle 2.2: Hierarchie einer Zwei-Arm-Manipulation (inspiriert von [19])

bearbeitet. Die einzelnen Roboter wissen bei dieser Variante nichts von der Existenz der anderen. Sollen zwei Roboter an demselben Werkstück arbeiten, müssen die Roboter gegebenenfalls aufeinander warten, damit es nicht zu Kollisionen kommt. Roboter 1 transportiert z. B. das Werkstück an eine vordefinierte Position, danach bearbeitet Roboter 2 das Werkstück und nach dessen Fertigstellung holt Roboter 1 das fertige Werkstück wieder ab. In der Automatisierungstechnik wird diese Art des Locking sehr häufig eingesetzt, wenn mehrere Arbeitsschritte nacheinander ausgeführt werden sollen. Ein Problem bei dieser Art der Produktion ist eine relativ lange Totzeit der Roboter während des Wartens, die laut [20] in der Massenproduktion bei einer Minute etwa 10 bis 15 Sekunden ausmacht.

Ziel-koordiniert

Bei dieser Art der Kooperation arbeiten mehrere Roboter in einem großen gemeinsamen Arbeitsraum. Hierbei kann es z. B. vorkommen, dass Roboter an derselben Stelle eines Werkstückes etwas bearbeiten sollen. Um Kollisionen untereinander zu vermeiden, wird der Arbeitsraum in Teil-Arbeitsräume aufgeteilt. Zum Befahren der Teil-Räume müssen die Roboter auf spezielle Freigabekommandos warten. Erst nachdem ein Roboter seinen Raum freigegeben hat, kann der nächste in diesem seine Arbeit fortsetzten.

Bimanual

Erfolg diese Art der Kooperation, hält und transportiert ein Roboter das Werkstück, und ein zweiter bearbeitet dieses zur selben Zeit. Eine weitere Möglichkeit wäre der Transport eines schweren oder sehr großen Werkstückes durch zwei oder mehrere Roboter gleichzeitig. Diese Art der Kooperation ist steuerungstechnisch die schwierigste, da die Position der Roboter zueinander in hoher Geschwindigkeit und sehr genau geregelt werden muss, um eine Kollision zu vermeiden. Weitere typische Anwendungsbereiche wären z. B. eine Prüfung mit Ultraschall, bei der ein Roboter den Sender und der zweite Roboter den Empfänger bewegt. Ebenso stellt die Handhabung großer Bauteile wie z. B. Stringer im Flugzeugbau, die 30 Meter lang sein können und sehr labil sind, oder die Handhabung eines extrem schweren Bauteils durch mehrere Roboter gleichzeitig ein Anwendungsbeispiel dar. Aufgrund der genannten Anwendungsfälle wird diese Art der Kooperation auch Load-Sharing genannt.

Die verschiedenen Roboterhersteller haben jeweils eigene Technologiepakete, die die Kooperation von Robotern ermöglichen. Bei KUKA heißt dieses RoboTeam und unterstützt bis zu 15 einzelne Robotersteuerungen, die über Ethernet kommunizieren. ABB nennt die Technologie MultiMove,

mit dem vier Roboter oder 36 Achsen gesteuert werden können, Fanuc Multi-Arm und Motoman Multi-Robot. KUKA RoboTeam unterstützt die folgenden Funktionalitäten, die den zuvor erläuterten Arten der Kooperation entsprechen:

- Geolink: Dieser bietet eine geometrische Koppelung der beiden Roboter. Sobald der Geolink aktiv ist, folgt der Slave jederzeit den Bewegungen des Master-Roboters. Bewegt man z. B. den Master kartesisch in eine Richtung, folgt der Slave dieser Richtung. Die Geolink-Funktionalität kann sehr hilfreich sein, wenn ein Objekt transportiert wird und währenddessen der Abstand zwischen den beiden Robotern immer gleich sein soll. Eine typische Anwendung hierfür kann der Transport langer oder schwerer Bauteile sein.
- ProgSync: Der ProgSync erlaubt, Marker im Programm zu setzen, an denen beide Roboter gleichzeitig beginnen, eine Aktion auszuführen. Die Roboter starten zur gleichen Zeit mit einer Bewegung, enden allerdings mitunter zu einem unterschiedlichen Zeitpunkt, je nachdem wie lang die Bewegung der einzelnen Roboter dauert. Wenn zwei Roboter eine gleichlange Bewegung mit der gleichen Geschwindigkeit ausführen, kann man den ProgSync auch als Alternative zum GeoLink verwenden.
- MotionSync: Beim MotionSync starten und stoppen die Roboter zur exakt gleichen Zeit. Die Geschwindigkeit der Bewegungen wird von RoboTeam so angepasst, dass unterschiedliche Bewegungen gleich lang dauern.
- RemoteCommand: Erlaubt es, Kommandos zwischen den Robotern auszutauschen. Somit kann z. B. der Master Schaltbefehle an den Slave senden.

2.8 Kollisionserkennung

Um die äußeren Ausmaße eines Objektes durch einen Hüllkörper zu beschreiben, gibt es mehrere Möglichkeiten. Diese reichen von einer einfachen Kugel, die um die äußeren Ecken eines Objektes gelegt wird, bis zu einer beinahe exakten konvexen Hülle. Als Zwischenstufen gibt es noch die Axis-Aligned Bounding Boxes (AABB) und die Oriented Bounding Boxes (OBB). Je einfacher der Hüllkörper ist, desto einfacher ist zum einen die Bestimmung und zum anderen die Berechnung von Kollisionen der Objekte untereinander. Abbildung 2.14 zeigt verschiedene Varianten von Hüllkörpern.

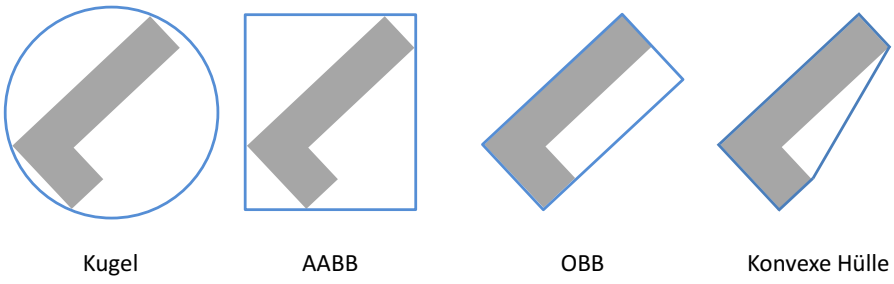


Abbildung 2.14: Verschiedene Varianten von Hüllkörpern. Von links nach rechts werden die Objekte genauer beschreiben. Von rechts nach links erfolgt eine schnellere Berechnung der möglichen Kollisionen.

Die meisten Algorithmen zur Kollisionserkennung unterscheiden eine ferne (broad) und eine nahe (near) Phase [21]. In der fernen Phase werden die Objekte identifiziert, die in der nahen Phase näher untersucht werden sollen. Die gängigsten Verfahren zur Prüfung der fernen Phase sind *exhaustive search*, *sweep and prune* [22, 23] und *hierachical hash tables* [24]. In der nahen Phase werden die Algorithmen zur Berechnung einer Kollision meistens in vier Gruppen kategorisiert: *Feature-basiert*, *Simplex-basiert*, *Image-Space* und *Spatial Data Structures* [24].
Feature-basierte Algorithmen arbeiten direkt mit der Grundform des Objektes. Bekannte Beispiele sind *Polygonal Intersection* [25], *Lin-Canny* [26], *V-Clip* [27] und *SWIFT* [28]. *Simplex-basierte* Algorithmen beruhen auf einem geometrischen Ansatz und beschreiben ein n-dimensionales Polytop, welches die konvexe Hülle von $n + 1$ Ecken ist. Mit zunehmender Dimension ergibt sich somit die folgende Reihe: Punkt, Strecke, Dreieck, Tetraeder. Der Gilbert-Johnson-Keerthi (GJK) Entfernungsalgorithmus ist der bekannteste Vertreter dieser Sorte [29, 30, 31, 32, 33].

Die Gruppe der Image-Space-Algorithmen (ISB) bestimmt Kollisionen durch die Berechnung von Verdeckungsabfragen. Bei der Erkennung von Kollisionen in dynamischen Simulationen sind ISB sehr effizient, da die Berechnung der Verdeckungen sehr gut parallelisiert und damit auf die Grafikkarte (Graphical Processing Unit (GPU)) ausgelagert werden kann. Bekannte Beispiele sind [34, 35, 36, 37, 38, 39].

Zum Bereich *Spatial Data Structures* gehören die Bounding Volume Hierarchies (BVH). In [40] wird eine Übersicht über die verschiedenen Typen gegeben. Vertreter sind u. a. *Bounding Spheres* (Kugeln) [41], die sich sehr leicht berechnen lassen. Eine weitere Gruppe sind Bounding Boxes (BB) (Quader oder Würfel), die Objekte genauer umschreiben als Kugeln und daher in Anwendungen, wie z. B. Raytracing, von Vorteil sind. Spezielle Vertreter von BB sind BVH. Beliebig orientierte Quader werden auch als OBB, an den Achsen ausgerichtete Quader als AABB bezeichnet. AABB werden üblicherweise durch zwei Punkte definiert, die die Position der Ecken auf beiden Seiten einer Quaderdiagonalen angeben. Außerdem können Objekte durch Polyeder, auch k-Discrete Oriented Polytopes (k-DOP) genannt, beschrieben werden. Im Gegensatz zu OBB erlauben Polyeder mehrere Beschränkungsflächen, wodurch sie Objekte besser, d. h. enger einschließen können. Die komplexeste Art der Beschreibung eines Objektes ist die durch ein Dreiecksgitter, welches mit einer vorher definierten Auflösung entlang der äußeren Oberfläche gelegt wird. Da moderne Messgeräte, wie ein Laserscanner oder eine Tiefenkamera, meistens Punktewolken als Repräsentation der Umgebung liefern, ist eine weitere Möglichkeit der Kollisionserkennung das direkte Berechnen auf Basis von Punktewolken [42, 43, 44].

Abbildung 2.15 zeigt die Darstellung der OBBs und der konvexen Hülle eines Industrieroboters. Sowohl bei Bounding Boxes als auch bei konvexen Hüllen tritt eine Kollision auf, wenn diese sich überschneiden. Wie man an der Abbildung sehr gut erkennen kann, wird die Kollisionserkennung durch Bounding Boxen sehr ungenau, hat jedoch den Vorteil, dass sie wesentlich schneller ist. In engen Passagen kann dies allerdings schnell dazu führen, dass kein Pfad gefunden wird, obwohl theoretisch ein möglicher existiert.

2.9 Grundlagen der Faserverbundherstellung

Fasern und Faserverstärkte Kunststoffe (FVK)

Bei Faserverstärkten Kunstoffen handelt es sich um einen Verbundwerkstoff, bei dem Verstärkungsfasern in Form von Kurzfasern (L = 0,1 mm bis 1 mm), Langfasern (L = 1 mm bis 50 mm) oder

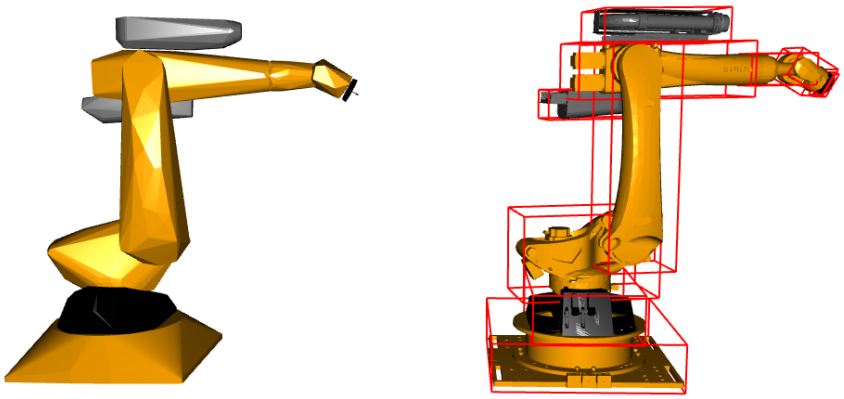


Abbildung 2.15: Repräsentation des Roboters und seiner Anbauteile durch eine konvexe Hülle auf der linken Seite und durch OBB auf der rechten Seite (Screenshot aus KoKo).

Endlosfasern ($L \geq 50\,mm$), gerichtet oder ungerichtet, in eine Kunststoffmatrix eingebettet werden. Kurz- und Langfasern werden gerne bei Spritzgussverfahren eingesetzt, um die Eigenschaften eines Kunststoffes mit schlechten mechanischen Werten zu verbessern. Die größte Bedeutung bei den Verstärkungsfasern haben heute Glas, Aramid und Kohlenstoff [45, S.1]. Letztere haben die höchste Steifig- und Festigkeit. Aramidfasern weisen eine vergleichsweise hohe Bruchdehnung und bessere Dämpfungseigenschaften auf, während die mechanischen Eigenschaften von Glasfasern schlechter sind als die von Aramid- und Kohlenstofffasern. Aufgrund des vergleichsweise niedrigeren Preises werden diese jedoch häufig eingesetzt [46, S.3]. Das Hauptaugenmerk in dieser Arbeit liegt auf carbon- oder kohlenstofffaserverstärkten Kunststoffen (CFK). Die Kohlenstofffasern werden unter hohem Energieaufwand in einem mehrstufigen Prozess entweder aus Polyacrylnitril (PAN) oder Pech hergestellt. In der letzten Stufe der Carbonisierung hat eine einzelne Kohlenstofffaser, auch Filament genannt, einen Durchmesser von etwa $5\,\mu m$. Von diesen werden etwa 1000 bis 24.000 parallel zueinander angeordnet und zu einem Roving zusammengefasst, der dann z. B. zu einem Halbzeug verarbeitet werden kann. Die gängigsten Faserhalbzeuge sind Gewebe, Geflechte und Gelege. Eine umfangreiche Übersicht über die verschiedenen Herstellungsverfahren kann in [47, S.311], [48] und [45] gefunden werden. Die Kohlenstofffasern selbst sind in Längsrichtung sehr zugstabil, allerdings sehr anfällig gegen Druck- und Scherbelastung. Erst durch die kraftschlüssige Verbindung der Fasern durch eine Matrix entsteht ein leichter und stabiler Verbundwerkstoff. Die eingesetzte Matrix wird je nach eingesetztem Kunststoff grob in Duromer und Thermoplast unterteilt. Um den Werkstoff optimal zu nutzen, muss ein lastgerechtes Design erfolgen, durch welches eine maximale Gewichtseinsparung erreicht werden kann.

Wie bereits erwähnt kommen bei FVK duromere und thermoplastische Kunststoffe als Matrix zum Einsatz. Bei einer duromeren Matrix werden die Zuschnitte entweder trocken abgelegt und in einem zweiten Schritt wird die Matrix infiltriert, oder die Zuschnitte sind bereits vorher mit Harz imprägniert (sogenannte Prepregs). Trockene Preforms werden lagenweise aufeinander in einer Werkzeugform abgelegt. Um ein Verrutschen zu verhindern, gibt es Zuschnitte, auf denen ein Bindermaterial aufgebracht ist, welches durch Erwärmen aufschmilzt. Nachdem der

Lagenaufbau beendet ist, wird das Harz eingebracht. Dies kann entweder durch Unterdruck mit einem Vakuumaufbau eingesaugt oder mit Überdruck hineingepresst werden. Je nach Harzsystem wird das Bauteil bei Raumtemperatur oder in einem Ofen ausgehärtet. Duromere Prepregs sind bereits mit Harz getränkt und müssen bei etwa $-18\,°C$ gelagert werden, um eine Reaktion des Harzssystems zu verhindern. Nachdem die Lagen abgelegt wurden, erfolgt die Weiterverarbeitung in einem Autoklaven, in dem das Bauteil unter Druck und Temperatur ausgehärtet wird. Halbzeuge, die mit einer thermoplastischen Matrix imprägniert wurden, sind bei Raumtemperatur lagerfähig, da die Matrix bei sehr hohen Temperaturen aufgebracht wird. Ein sehr großer Vorteil des thermoplastischen Matrixsystems ist die mehrmalige Möglichkeit der Erwärmung und Umformung. So können zunächst die einzelnen Lagen bspw. durch ein Ultraschallschweißgerät miteinander verbunden werden, indem die Matrix aufgeschmolzen wird. Man spricht bei diesen Lagenpaketen auch von Organoblechen. Danach werden die Zuschnitte in einem Vakuumaufbau im Ofen oder in einer Heißpresse konsolidiert, in der sie auch in eine andere Form gebracht werden können.

Automatisierte Herstellung eines Bauteils

Die Auslegung eines Bauteils wird normalerweise in einer Konstruktionssoftware, wie CATIA, durchgeführt. Diese bietet die Composite-Design (CPD) Erweiterung, in der der Bauteildesigner den Lagenaufbau definieren kann. Dazu werden Größe, Position, Material und Faserorientierung der einzelnen Zuschnitte genau definiert. Des Weiteren enthält das Erweiterungsmodul eine Funktionalität, die es erlaubt, die Umformung, auch Drapierung genannt, eines zweidimensionalen Zuschnittes, der auf eine dreidimensionale Werkzeugform gelegt wird, zu berechnen. Nach Fertigstellung des Bauteils kann dessen Aufbau in eine Datei in der Extensible Markup Language (XML), wie in Quellcode 2.2 zu sehen, exportiert werden. Die in dieser Datei enthaltenen geometrischen Informationen zu den Außenkonturen der Zuschnitte sind allerdings nicht ausreichend für ein Roboter-basiertes Produktionssystem. Für dieses müssen weitere Informationen wie die Greif- und Ablagepunkte bekannt sein. Das Job Definition File (JDF), ein Dateiformat welches diese nötigen Informationen enthält, wurde von Schuster [49] entwickelt.

Quellcode 2.2: Auszug aus einer aus dem CATIA-CPD-Modul exportierten Datei.

```
<CATIA_V5_CPD_Composites_Data_Model VERSION="1.0">
<Stacking NAME="Stapelung" TYPE="Engineering">
 <Plies_Group NAME="Plies Group.1" ROSETTE="Achsensystem.1" SURFACE="Fläche mit
     Mehrfachschnitten.1" STRUCTURAL="No" DRAPING_DIR="POSITIVE">
  <Sequence NAME="Sequence.1">
   <Ply NAME="Ply.1">
    <PlyAttributes AREA="0.630299" CG_X="4022.24" CG_Y="-1169.5" CG_Z="-284.676" COST="
        0" WEIGHT="0.000291135" REF_SURF="Fläche mit Mehrfachschnitten.1" ROSSETTE="
        Achsensystem.1" PERIMETER="4611.78" DRAPING_DIR="POSITIVE" MATERIAL_ID="
        Bidiagonal Carbon Fabric -+45" ORIENTATION="0" />
     <Contour_3D NAME="Kontur.39">
      <OuterContour POINTS_COUNT="22">
       <Pt>X=4030.46  Y=-175  Z=-158.257</Pt>
       <Pt>X=4029.69 Y=-175 Z=-189.93</Pt>
      ...
    </OuterContour>
   </Contour_3D>
   <Contour_2D POINTS_COUNT="22">
    <Pt>2056 88.38 873.43</Pt>
    <Pt>2056 120.10 873.43</Pt>
    ...
   </Contour_2D>
  </Ply>
   <Ply NAME="Ply.2">
    <PlyAttributes AREA="2.05654" CG_X="3836.74" CG_Y="-1169.5" CG_Z="-926.664" COST="0
        " WEIGHT="0.000949915" REF_SURF="Fläche mit Mehrfachschnitten.1" ROSSETTE="
        Achsensystem.1" PERIMETER="6045.91" DRAPING_DIR="POSITIVE" MATERIAL_ID="
        Bidiagonal Carbon Fabric -+45" ORIENTATION="0" />
     <Contour_3D NAME="Kontur.40">
      <OuterContour POINTS_COUNT="66">
       <Pt>X=3999.91 Y=-175 Z=-473.956</Pt>
       ...
      </OuterContour>
     </Contour_3D>
     ...
  </Sequence>
  ...
 </Plies_Group>
 ...
</Stacking>
</CATIA_V5_CPD_Composites_Data_Model>
```

Kapitel 3

Pfadplanung

Im folgenden Kapitel werden zunächst die wesentlichen Grundlagen der Pfadplanung erläutert. Im Anschluss daran wird der Stand der Technik in diesem Bereich anhand existierender Verfahren präsentiert und in Bezug zu der in dieser Arbeit behandelten Anwendung gesetzt. Die ersten Veröffentlichungen datieren auf die 1960er Jahre mit [50], der ein Neuronales Netz (NN) verwendete, um den Pfad für einen kamerabasierten mobilen Roboter zu planen. Mit der Einführung des Konfigurationsraumes durch Lozano-Pérez [51] wurde das Thema immer populärer und nahm weiter Fahrt auf. Latombe [52] veröffentlichte 1991 eine sehr umfangreiche Zusammenfassung der Arbeiten der 1990er Jahre.

3.1 Grundlagen der Pfadplanung

Bevor der Stand der Technik im Bereich der Pfadplanung behandelt wird, müssen zunächst einige grundlegende Begrifflichkeiten zum besseren Verständnis eingeführt und erläutert werden.

Pfad- und Trajektorienplanung

Unter Pfadplanung versteht man die Berechnung der Punkte des Pfades eines Roboters von einem Start- zu einem Zielpunkt im Konfigurations- oder im Arbeitsraum. Ein Pfad ist somit lediglich eine geometrische Beschreibung einer Bewegung [11, S.161]. Da Industrieroboter große bewegte Massen haben, kann eine Kollision mit einer Werkzeugform oder gar einem Menschen einen hohen Schaden anrichten. Daher ist es zwingend notwendig, eine kollisionsfreie Bahn zu planen. Wird die Pfadplanung um eine Kostenfunktion erweitert, die bei der Planung minimiert werden muss, z. B. anhand der Pfadlänge oder des Energieeinsatzes, spricht man von optimaler Pfadplanung [53]. Bei der Trajektorienplanung wird die Pfadplanung um die Dimension der Zeit erweitert [54]. Während bei der Pfadplanung nur bestimmt wird, wie ein Roboter sich zwischen einem Start- und einem Endpunkt bewegt, wird bei der Trajektorienplanung zusätzlich bestimmt, mit welcher Geschwindigkeit und Beschleunigung er das tut. Vereinfacht gesagt ist

der Input für die Trajektorienplanung die Pfadplanung plus Einflussgrößen, wie z. B. Schwerkraft und dynamische Größen des Manipulators [11, S.161]. Das Ergebnis ist die Trajektorie des Endeffektors in Bezug auf die Zeit, Geschwindigkeit und Beschleunigung. In der Literatur wird häufig der Begriff *Motion Planning* bzw. *Bewegungsplanung* als Synonym für Pfadplanung und Trajektorienplanung verwendet [55]. Diese Arbeit setzt sich ausschließlich mit dem Begriff der Pfadplanung auseinander.

Freiheitsgrade

Der Freiheitsgrad (Degrees of Freedom (DOF)) beschreibt, in wie viele Richtungen sich ein Körper bzw. ein Roboter bewegen kann. Ein starrer Körper im dreidimensionalen Raum kann dies beispielsweise in sechs Richtungen und hat daher den Freiheitsgrad sechs. Diese Freiheitsgrade setzten sich aus den drei translatorischen Richtungen DOF_{trans} (X, Y und Z) sowie den rotatorischen DOF_{rot} (A, B und C) zusammen. Für einen starren Körper der Dimension N lässt sich der Freiheitsgrad folgendermaßen berechnen [56, S.74] und [57, S.26]:

$$
\begin{aligned}
DOF_{trans} &= N \\
DOF_{rot} &= \frac{N \cdot (N-1)}{2} \\
DOF = DOF_{trans} + DOF_{rot} &= \frac{N \cdot (N+1)}{2}
\end{aligned}
$$

Kinematische Beschränkungen

Die gerade definierten DOF werden aufgrund der Dimension des Arbeitsraumes berechnet. Ein Roboter, der sich im 3D-Raum bewegt, muss allerdings nicht zwangsweise genauso viele ansteuerbare Freiheitsgrade haben. Die kontrollierbaren DOF eines Roboters nennt man Differential Degrees of Freedom (DDOF). Die DOF des Arbeitsraumes bilden das Maximum für die DDOF und es gilt: $DDOF \leq DOF$ [56, S.75].

Der Konfigurationsraum

Häufig wird für die Pfadplanung von Robotern nicht der kartesische Arbeitsraum, sondern ein spezielles mathematisches Konstrukt, der Konfigurationsraum ($\mathcal{C}$-Space) verwendet. Die ursprüngliche Idee dazu kam von Udupa [58]. Die ersten grundlegenden Arbeiten hat Lozano-Pérez [59, 51] und [60] veröffentlicht. Wie der Name sagt, beschreibt der Konfigurationsraum eine Menge möglicher Konfigurationen bzw. Transformationen, die an einem Roboter angewendet werden. Will man bspw. eine Konfiguration eines Roboterarms mit k Freiheitsgraden beschreiben, braucht man für jede Konfiguration k reelle Werte $q_1, ..., q_k$. Die Beschreibung einer Konfiguration eines sechs-achsigen KUKA-Industrieroboters im Achsraum benötigt sechs reelle Werte: $\theta_1, ..., \theta_6$, wobei sich diese im Bereich von 0 bis 360° bewegen. Die beschriebenen k-Werte können ebenso als Punkt P in einem k-dimensionalen Raum, Konfigurationsraum genannt, angesehen werden. Mittels dieser Beschreibung kann die komplexe Geometrie eines Roboters durch einen k-dimensionalen Punkt ausgedrückt werden. Abbildung 3.1 zeigt ein Beispiel eines Konfigurationsraumes, der in kollisionsfreie $\mathcal{C}_{free}$ und kollionsbehaftete $\mathcal{C}_{col}$ Zustände aufgeteilt ist. Häufig wird in der Literatur anstatt $\mathcal{C}_{col}$ auch $\mathcal{C}_{obs}$ (obstacle) verwendet.

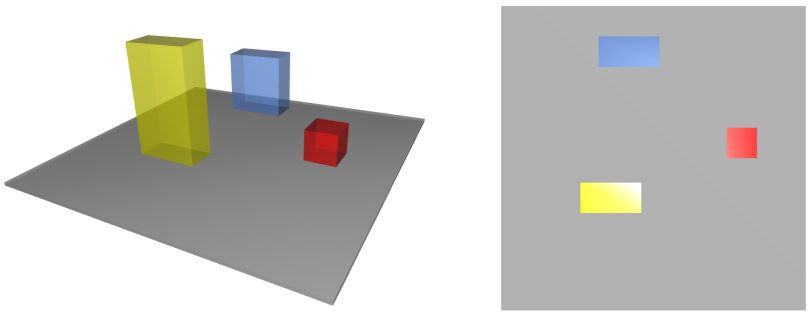


Abbildung 3.1: Beispiel eines Konfigurationsraumes. Die linke Seite zeigt den Arbeitsraum und die rechte Seite den Konfigurationsraum eines Roboters, dessen Größe ein Punkt ist. Die graue Fläche entspricht $\mathcal{C}_{free}$ und die farbigen Flächen entsprechen $\mathcal{C}_{obs}$ (inspiriert von [61] – Screenshot aus KoKo).

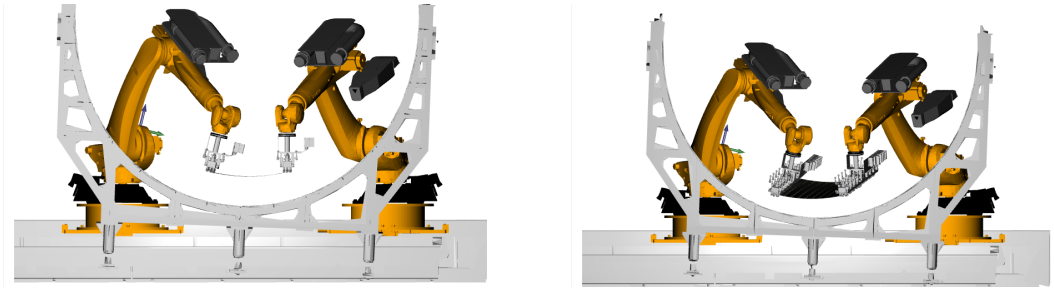
Bei der Planung besteht nun die Herausforderung, $\mathcal{C}_{free}$ und $\mathcal{C}_{col}$ möglichst genau zu bestimmen. Da eine vollständige Bestimmung des Konfigurationsraumes aufgrund der langen Berechnungszeit nicht möglich ist, wird der Konfigurationsraum lediglich an einer begrenzten Anzahl von Positionen untersucht. Daher wurden auch sogenannte Sampling-basierte Verfahren, die in Abschnitt 3.4 beschrieben werden, entwickelt.

Holonomie und Kinodynamik

Generell kann bei Pfadplanungsproblemen zwischen holonomen, nicht-holonomen und kinodynamischen unterschieden werden [62, 63]. Von einem holonomen Pfadplanungsproblem spricht man, wenn alle Freiheitsgrade des Roboters kontrollierbar sind. Dies vereinfacht das Pfadplanungsproblem des Roboter ungemein, da dieser zu jedem Zeitpunkt in alle Richtungen bewegt werden kann [62] [56, S.75]. Unter einem nicht-holonomen Roboter wird z. B. die Gruppe der mobilen Roboter geführt, die normalerweise nur vor- bzw. rückwärts sowie nach links und rechts fahren können. Ein einfaches Beispiel hierfür ist ein Auto. Dieses kann nicht seitlich fahren, was das Einparken in eine Lücke parallel zur Fahrbahn schwierig macht [14, S.119] [64]. Bei kinodynamischen Pfadplanungsproblemen kommen neben Bedingungen wie Kollisionsfreiheit noch weitere, wie bspw. Geschwindigkeits- und Beschleunigungsbedingungen hinzu. Es handelt sich dabei um Systeme, die Differenzialgleichungen zweiter Ordnung (oder höher) beinhalten [65].

SE(3) und SO(3)

In Abschnitt 2.3 wurde bereits die Pose eines starren Körpers eingeführt. Wird lediglich von den drei Drehungen A, B und C eines Körpers gesprochen, wird dies als $SO(3)$-Raum bezeichnet (Special Orthogonal Group). Kommt zu der Rotation noch die Translation X, Y und Z eines Körpers hinzu, wird vom $SE(3)$-Raum gesprochen, wobei $SE(3) = \mathbb{R}^3 \times SO(3)$ [54, S.148].



(a) Planung mit Constraints – die Z-Koordinaten beider Roboter sind gleich und die Orientierungen sind eingefroren.

(b) Planung ohne Constraints – die Z-Koordinaten und die Orientierungen beider Roboter sind unterschiedlich.

Abbildung 3.2: Planung mit und ohne Constraints (Screenshot aus KoKo).

Constraints

Bei der Arbeit mit einem Industrieroboter kann es nötig sein, bei der Planung bestimmte Einschränkungen an die Bewegungsfreiheit zu setzen. Dabei wird zwischen globalen und lokalen Constraints unterschieden. Globale Constraints in $\mathcal{C}$ können neben Hindernissen auch Achsanschläge oder Beschränkungen der Orientierung sein. Letztere können je nach Anforderung an einen Prozess auftreten. Beispiel hierfür ist das Auftragen einer Klebstoffraupe, bei der der Endeffektor in einem bestimmten Winkel zum Bauteil gehalten werden muss. Aber auch der in dieser Arbeit behandelte Prozess des Transportes eines CFK-Zuschnittes kann die Berücksichtigung von Constraints nötig machen. Aufgrund von physikalischen Eigenschaften des Materials soll der Zuschnitt während des Prozesses in einer Seilkurve transportiert werden, wobei die Höhe der Greifer gleich ist. Lokale Constraints werden mit differentiellen Gleichungen beschrieben und daher auch Differential- oder Zeit-Constraints genannt. Diese beschränken z. B. die maximale Geschwindigkeit oder die Beschleunigung [14, S.118]. Durch Constraints wird der mögliche Zustandsraum auf einen Unterraum, den Constraint Manifold eingeschränkt [14, S.109]. Diese Art der Constraints zu berücksichtigen ist wesentlich schwieriger als nur Positionen zu berücksichtigen, die kollisionsfrei sind, da die Dimensionalität des Constraint Manifold kleiner als die Dimension des Zustandsraumes ist. Geometrische Constraints werden im SE(3)-Raum, der sechs Dimensionen hat, angegeben. Während es bei der Kollisionsprüfung möglich ist, zunächst einen Zustand zu generieren und im Falle einer Kollision eine neue Konfiguration zu erzeugen, schlägt dieses Vorgehen im Falle von geometrischen Constraints fehl. Dies liegt daran, dass die Wahrscheinlichkeit, dass ein zufällig gesampelter Zustand einen bestimmten Constraint erfüllt, sehr klein bzw. null ist [66, 67]. Dieses Verhalten kann man sich auch einfach vorstellen. Generiert man zufällig einen Wert für den Achswinkel der sechs Achsen eines Industrieroboters, ist die Wahrscheinlichkeit sehr gering, genau die gewünschte Position für den TCP zu erreichen.

In der Literatur werden verschiedene Möglichkeiten aufgezeigt, mit Constraints umzugehen. Eine Möglichkeit wäre z. B., nur Zustände zu sampeln, die die geforderten Constraints erfüllen. Plant man im SE(3)-Raum, ist dies problemlos möglich, da einfach nur Positionen mit bspw. den richtigen Orientierungen generiert werden. Wird die Planung hingegen im Achsraum durchgeführt, ist ein zufälliges Sampling wie bereits erwähnt nicht so einfach. In diesem Fall müssen Samples im SE(3)-Raum erzeugt werden und im zweiten Schritt über die IK in den Achsraum zurückgerechnet und dem Planer übergeben werden. Leider geht bei dieser Art des Samplings der Vorteil des Konfigurationsraumes verloren, dass keine IK benötigt wird. Zusätzlich muss beim Einsatz einer

Linearachse eine Heuristik gefunden werden, die Position dieser für eine bestimmte TCP-Position zu bestimmen.

Piano Mover's Problem

Im Zusammenhang mit Pfadplanung taucht häufig der Begriff *Piano Mover's Problem* auf. Dieses beschreibt, wie ein Flügel durch ein Haus transportiert werden kann, ohne zu kollidieren. Auf den ersten Blick ist der Vergleich mit der Bahnplanung eines Roboters etwas weit hergeholt. Auf den zweiten Blick jedoch sieht man, dass ein Klavier ebenfalls mit sechs Freiheitsgraden bewegt werden kann. Betrachtet man nun den Spezialfall eines Roboters, der aus nur einem Körper besteht, entspricht dieser dem Klavier.

Vollständigkeit und probabilistische Vollständigkeit

Es ist wichtig, die Qualität von Pfadplanungsalgorithmen zu vergleichen. Typischerweise wird man dabei die Zeit, die der Algorithmus zum Lösen benötigt, als Maß nehmen. In der Informatik werden Algorithmen anhand ihrer Laufzeit bewertet, wobei es drei Klassen gibt: P, NP und NP-Vollständig. Man spricht von P, wenn ein Problem in Polynomialzeit von einer deterministischen Turingmaschine gelöst werden kann. Gerne werden P-Probleme daher auch „praktisch lösbare" Probleme genannt. Die Klasse der NP-Probleme ist ebenfalls in polynomieller Zeit lösbar. In diesem Fall wird allerdings eine nicht-deterministische Turingmaschine eingesetzt. Mit NP-Vollständigkeit bezeichnet man die Klasse der Probleme, die sich nichtdeterministisch in Polynomialzeit lösen lassen.

Im Bereich der Pfadplanung spricht man von einem vollständigen Planungsalgorithmus, wenn garantiert ist, dass dieser immer eine Lösung findet, sofern eine existiert. Falls keine Lösung existiert, soll der Algorithmus diese Information als Rückgabe liefern. Vollständigkeit garantiert, dass ein Algorithmus erwartungsgemäß abläuft. Dies ist im Bereich der Industrierobotik besonders wichtig, da Fehler oder Verzögerungen zu hohen Schäden führen können [68].

Reif [69] hat als erster gezeigt, dass das Piano Mover's Problem NP-hard ist. Dies bedeutet: Wenn man zufällig einen Pfad generiert, lässt sich in polynomieller Zeit prüfen, ob dieser Pfad eine korrekte Lösung ist. Canny [70] hat bewiesen, dass Pfadplanung NP-Vollständig ist. Dies bedeutet, dass das Planungsproblem polynomiell mit der Anzahl der Hindernisse und exponentiell mit der Anzahl der DOF eines Roboters steigt. Daraus lässt sich schlussfolgern, dass die Planung für einen Roboter mit vielen Freiheitsgraden und vielen Hindernissen sehr komplex ist [71].

Da es, wie gerade erläutert, sehr aufwendig ist, in einem komplexen Szenario einen Pfad zu finden, hat sich der Begriff der „probabilistischen Vollständigkeit" entwickelt. Diese Art der Algorithmen garantieren, in einer begrenzten Zeit einen Pfad zu finden, falls ein solcher existiert. In Abschnitt 3.4 werden einige dieser Verfahren vorgestellt und näher erläutert.

Geschlossene kinematische Kette

Ein besonderes Problem der Pfadplanung stellen die sogenannten geschlossenen kinematischen Ketten dar, die z. B. auftreten, wenn zwei Manipulatoren gemeinsam ein Objekt halten. Humanoide Roboter bilden eine geschlossene kinematische Kette, wenn diese mit beiden Beinen auf dem Boden stehen [14, S.121]. Auch bei dem in dieser Arbeit behandelten Szenario des Handlings von CFK-Material mit zwei Industrierobotern kommt dieses Problem zum Tragen. Da die beiden Roboter

aneinander hängen, ist es nicht möglich, unabhängig für die beiden eine Konfiguration zu generieren. Sobald für einen Roboter eine Konfiguration bestimmt wurde, kann der zweite nur noch bestimmte Konfigurationen einnehmen. Diese Abhängigkeit wird als Closure Constraint bezeichnet. Das Problem dieser Ketten besteht darin, dass in diesem Fall der Konfigurationsraum kein Manifold mehr darstellt, sondern eine Varietät. Die Wahrscheinlichkeit, zufällig eine Konfiguration zu finden, die die Bedingung der geschlossenen kinematischen Kette erfüllt, ist nahe null [72]. Es gibt zwar erste Ansätze, mit dieser Form der Ketten umzugehen, wie bspw. den Random Loop Generator (RLG) [73] oder wie in [72], die Kette aufbrechen und für beide Roboter getrennt zu planen. Dennoch gibt es aktuell keine allgemeingültige Lösung für das Problem der geschlossenen kinematischen Ketten. Die existierenden Verfahren müssen jedes Mal auf spezielle Robotertypen abgestimmt werden [73, S.27].

On-line und Off-line

Off-line-Planung geht von einer bekannten und festen Umwelt aus, die sich während der Planung nicht verändert. Bei der On-line-Planung werden auch aktuelle Umweltinformationen, wie z. B. die Veränderung von Hindernissen, mit berücksichtigt.

3.2 Taxonomie verschiedener Pfadplanungsalgorithmen

Da es mittlerweile eine Vielzahl an verschiedenen Pfadplanungsalgorithmen gibt, werden diese in Gruppen eingeteilt. Auf der obersten Stufe können Planungsalgorithmen danach unterschieden werden, ob sie für holonome, nicht holonome oder kinodynamische Probleme geeignet sind [62] (siehe Abschnitt 3.1). Das in dieser Arbeit vorgestellte Bahnplanungsproblem ist ein holonomes Problem, weshalb im Folgenden der Fokus auf existierende Algorithmen in diesem Bereich gelegt wird.

Es handelt sich bei den hier vorgestellten Planern um globale Techniken, da den Robotern die komplette Umgebung bekannt ist. Bei lokalen Planern ist dies nicht der Fall, hier muss erst eine Karte erstellt werden. Die folgenden Veröffentlichungen [74, 62, 75, 53] nehmen alle eine Gruppierung verschiedener Planungsalgorithmen vor und geben eine Übersicht. Abbildung 3.3 zeigt einen Ausschnitt der Einteilung der Algorithmen nach [53], die als Grundlage für die Taxonomie in dieser Arbeit dient. Bei der Auseinandersetzung mit der Literatur im Bereich kooperierende Roboter werden die in Abschnitt 2.7 eingeführten bimanual arbeitenden betrachtet.

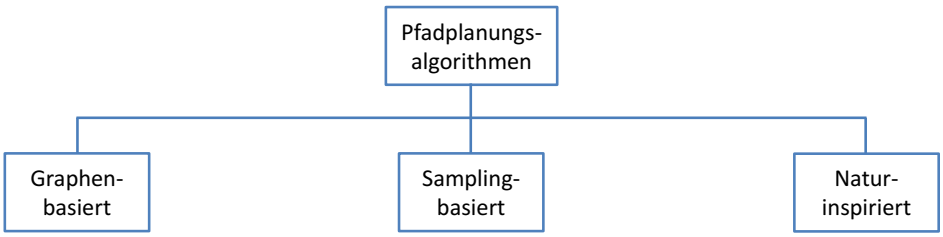


Abbildung 3.3: Planungsalgorithmen-Taxonomie (inspiriert von [53]).

3.3 Graphen-basierte Methoden

Diese Algorithmen dieser Gruppe berechnen einen optimalen Pfad aus einer Menge von Knoten. Dazu muss vorher das Netz der Knoten mit den nötigen Informationen, wie z. B. den Entfernungen, generiert worden sein. Typische Vertreter von Graphen-basierten Algorithmen sind Dijkstra, A*, Lifelong, Theta* und D*.

Der Dijkstra-Algorithmus findet in einem gerichteten Graphen $\mathcal{G}(V, E)$ mit nicht negativ gewichteten Kanten, ausgehend von einem Startknoten, den kürzesten Pfad [76, S.595]. Ein weiterer sehr bekannter Algorithmus dieser Gattung ist der A*. Er wurde 1968 von Peter E. Hart, Nils Nilson und Betram Raphael entwickelt. Die Autoren haben den Dijkstra-Algorithmus um eine Heuristik erweitert, die den Graphen schneller Richtung Ziel lenkt. Dadurch ist A* im Allgemeinen deutlich schneller als Dijkstra [62].

3.4 Sampling-basierte Methoden

Die Idee Sampling-basierter Algorithmen besteht darin, die explizite Konstruktion des $\mathcal{C}_{col}$ zu vermeiden und stattdessen den Konfigurationsraum mit einem Sampling-Schema zu erkunden. Das Ausprobieren in dem unbekannten Raum wird von einem Kollisions-Erkennungsmodul unterstützt, welches allerdings für die Algorithmen nur als eine Black-Box angesehen wird [54, S.185]. Dies hat den großen Vorteil, dass die eigentliche Planung völlig unabhängig von der Kollisionserkennung ist. Der Nachteil dieser Art von Algorithmen ist, dass sie nicht vollständig sind. Wie in Abschnitt 3.1 bereits beschrieben, liefern vollständige Algorithmen in einer endlichen Zeit eine Lösung, falls diese existiert. Ein deterministischer Algorithmus, der sehr dichte Samples erzeugt, wird in einer endlichen Zeit eine Lösung finden. Falls es allerdings keine Lösung gibt, wird er unendlich lange suchen. Für Sampling-basierte Algorithmen gibt es eine abgeschwächte Form der Vollständigkeit. Würden diese unendlich viele Iterationen durchlaufen und währenddessen zufällig Samples erzeugen, würden sie irgendwann eine Lösung finden [54, S.185].

3.4.1 Single-Query-Algorithmen

Single-Query Algorithmen bauen, ausgehend von einem Startpunkt q_{init}, einen Baum auf, der sich so lange durch eine zufällige Heuristik ausbreitet, bis der Zielzustand q_{goal} erreicht wird. Wurde dieser gefunden, brechen die Single-Query-Algorithmen ab und es kann ein Pfad vom Ziel- zum Startzustand gebildet werden. Der Baum muss einmal für ein spezielles Planungsproblem berechnet werden und kann danach nicht für ein weiteres, abweichendes Planungsproblem zur Anwendung kommen. Dies bedeutet bspw., dass eine minimale Veränderung des des Startpunktes zur Folge hat, dass der Baum neu berechnet werden muss. Die Varianten der Single-Query-Algorithmen unterscheiden sich in der Art, wie der Baum aufgebaut wird. Einige der Planer bauen z. B. zwei Bäume gleichzeitig auf, einen vom Start- und einen vom Zielpunkt. Die folgende Liste zeigt einen Auszug verschiedener Single-Query-Algorithmen:

- Rapidly Exploring Random Trees (RRT): Diese waren einer der ersten Baum-basierten Single-Query-Planer. Sie sind einfach zu verstehen und es gibt mittlerweile zahlreiche Abwandlungen [77].

- Expansive Space Trees (EST)): Diese Planer wurden in etwa zur gleichen Zeit wie RRT veröffentlicht. Sie sind ebenfalls Baum-basiert, versuchen jedoch, indem sie die Besiedelung der Bereiche messen, schneller in die freien Gegenden vorzustoßen [78].
- Kinematic Planning by Interior-Exterior Cell Exploration (KPIECE): Auch hierbei handelt es sich um einen Baum-basierten Planer. Dieser verwendet Diskretierungsmethoden, um die Expandierung zu steuern [79].
- Search Tree with Resolution Independent Density Estimation (STRIDE): Diese Planer wurden von EST inspiriert, unterscheiden sich jedoch bei der Bestimmung der wenig besiedelten Gegenden, da eine spezielle Datenstruktur verwendet wird, um den nächsten Nachbarn zu bestimmen [80].
- Path-Directed Subdivision Trees (PDST): Diese Art der Algorithmen breiten sich in große Gegenden, die bisher keine Pfade enthalten, aus; im Gegensatz zu den vorherigen Planern jedoch nicht ausgehend von einem zufällig gewählten Endpunkt des aktuellen Pfades, sondern von einem zufällig gewählten Punkt entlang des Pfades [81].

Die meisten Sampling-basierten Single-Query-Planer gehen nach dem folgendem Schema vor, wobei sich lediglich die Schritte 2 und 3 unterscheiden. Die Implementierungen der anderen Schritte können sich ebenfalls verändern, allerdings kommt diesen keine herausragende Bedeutung zu. Schritt 2 wird häufig Vertex Selection Method (VSM) und Schritt 3 Local Planning Method (LPM) genannt [54, S.217]:

1. Initialisierung: $\mathcal{G}(V, E)$ sei ein ungerichteter Graph, der keine Kanten V und mindestens einen Knoten E, typischerweise vom Typ q_{init} oder q_{goal}, enthält. Im Allgemeinen sind noch weitere Punkte aus $\mathcal{C}_{free}$ enthalten.
2. Selektion einer Ecke (VSM): Es wird eine Ecke $q_{cur} \in V$ zur Expansion gewählt.
3. Lokale Planungsmethode (LPM): Für ein $q_{new} \in \mathcal{C}_{free}$, welches eine Ecke aus V sein kann, wird ein neuer Pfad $\tau_s : [0, 1] \to \mathcal{C}_{free}$ konstruiert, so dass $\tau(0) = q_{cur}$ und $\tau(1) = q_{new}$ ist. Zudem muss für τ_s geprüft werden, ob er kollsionsfrei ist. Ist dies nicht der Fall, wird Schritt 2 wiederholt.
4. Ein einer Knoten wird zum Graphen $\mathcal{G}(V, E)$ hinzugefügt.
5. Prüfung des Vorhandenseins einer Lösung: Es wird geprüft, ob $\mathcal{G}$ einen gültigen Pfad repräsentiert. Falls es nur einen gültigen Baum gibt, ist diese Operation einfach. Gibt es mehrere Bäume, kann dieser Schritt sehr aufwendig und teuer werden.
6. Wechsel zu Schritt 2: Schritte 2 bis 6 werden so lange wiederholt, bis eine Lösung gefunden wurde oder ein Eliminierbarkeitskriterium erreicht ist.

Rapidly Exploring Random Trees (RRT)

Rapidly Exploring Random Trees (RRT) sind ein probabilistischer Ansatz zum Lösen von holonomen, nicht-holonomen und kinodynamischen Planungsproblemen [77], deren Grundidee darin besteht, die Entwicklung in bisher nicht erkundete Bereiche vorzustoßen. Sie erweitern den A*-Algorithmus um zwei wesentliche Merkmale: zum einen den Voronoi Bias, der den Problemraum in gleichmäßige Bereiche aufteilt; zum anderen die Möglichkeit, an einer beliebigen Stelle einen neuen Knoten zu generieren. Die Idee des RRT ist, von einem Startzustand eine zufällige Baumstruktur aufzubauen, bis der Zielzustand erreicht wurde. Ist dies der Fall, bricht der Algorithmus ab und es kann vom gefundenen Zielzustand rückwärts zum Start gegangen

werden. Der Algorithmus läuft nach dem Schema in Algorithmus 1 ab, wobei q_{init} den Startpunkt darstellt, τ den RRT und κ die Anzahl der Knoten. Zunächst wird mit einem leeren Baum τ gestartet, der den Startpunkt q_{init} als Wurzel hat (Zeile 2). Im nächsten Schritt werden so lange Knoten zum Baum hinzugefügt, bis ein Knoten einen Zielzustand erreicht (Zeile 3-6). Ist nach k Durchläufen noch kein Knoten gefunden, bricht der Algorithmus ab.

Ein neuer Knoten wird mit der Operation $RANDOM\_STATE$ (Zeile 4) zufällig aus dem Konfigurationsraum, mit einer gleichmäßigen Verteilung, erzeugt. Bei manchen Implementierungen wird zudem geprüft, ob der neue Knoten in $\mathcal{C}_{free}$, also kollisionsfrei, ist. Falls nicht, wird ein neuer Zustand gesampelt. Des Weiteren gibt es Implementierungen, die einen Goal Bias benutzen, um den Baum schneller in Richtung Ziel zu lenken.

Die $EXTEND$-Operation (siehe Algorithmus 2) erweitert den bestehenden Baum um den neuen Knoten q_{rand}. Dazu wird zunächst in der Operation $NEAREST\_NEIGHBOUR$ (Zeile 2) der nächstgelegene Knoten x_{near} aus den bereits bestehenden zu dem neu generierten Knoten x gesucht. Im nächsten Schritt wird mit der Operation NEW_STATE (Zeile 3) versucht, den Baum in Richtung des gefundenen Knotens x ausgehend vom Knoten x_{near} mit dem Abstand ϵ auszubreiten und diese auf Kollisionen zu prüfen. Hierbei werden drei Fälle unterschieden:

- Reached: Der neu generierte Zustand ist identisch mit dem Sample x.
- Advanced: Ein neuer Knoten $x_{new} \neq x$ wird eingefügt.
- Trapped: Die Operation NEW_STATE konnte keinen Zustand, der in $\mathcal{C}_{free}$ liegt, generieren und wird daher verworfen.

Den Namen *rapidly exploring* hat der Algorithmus aufgrund der Eigenschaft, immer schnell in unbekannte Gebiete zu wachsen. Dies kann sehr anschaulich an einem Voronoi-Graphen des RRT verdeutlicht werden (siehe Abb. 3.5). Es ist gut zu erkennen, dass die Voronoi-Regionen in den äußeren Bereichen wesentlich größer sind. Die Wahrscheinlichkeit zur Auswahl eines Knoten ist direkt proportional zur Größe der Voronoi-Region. Somit werden in den anfänglichen Schritten Knoten ausgewählt, die in den äußeren Regionen liegen. Im späteren Verlauf befinden sich die größten Voronoi-Regionen in den Zwischenräumen der großen Äste, wodurch der Baum sich in diese Richtung entwickelt.

Algorithm 1 Erstellung eines RRT [63]

```
1: procedure BUILD_RRT(q_init, κ)
2:     τ.init(q_init);
3:     for k = 1 to κ do
4:         q_rand ← RANDOM_STATE();
5:         EXTEND(τ, q_rand);
6:     end for
7:     return τ;
8: end procedure
```

RRT-connect

Im Laufe der Zeit wurden Abwandlungen des RRT entwickelt, wie z. B. der RRT-connect [83]; dieser hat zwei wesentliche Unterschiede zum RRT, aufgrund derer er in den meisten Fällen

Algorithm 2 Die $EXTEND$-Operation eines RRT [63]

```
1:  procedure EXTEND(τ, q_rand, q_new)          ▷ q_new ist ein Ausgabeparameter
2:      q_near ← NEAREST_NEIGHBOR(q_rand, τ);    ▷ Nächster Knoten zu q_rand
3:      if NEW_STATE(q_rand, q_near, q_new) then  ▷ q_new ist ein Ausgabeparameter
4:          τ.add_vertex(q_new);
5:          τ.add_edge(q_near, q_new);
6:          if q_new = q_rand then return Reached;
7:          else return Advanced;
8:          end if
9:      end ifreturn Trapped;
10: end procedure
```

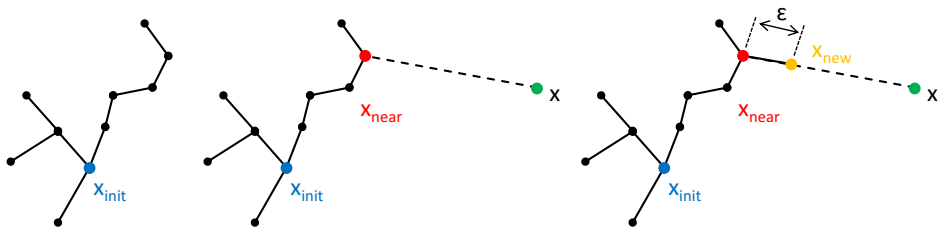


Abbildung 3.4: Die $EXTEND$-Operation des RRT

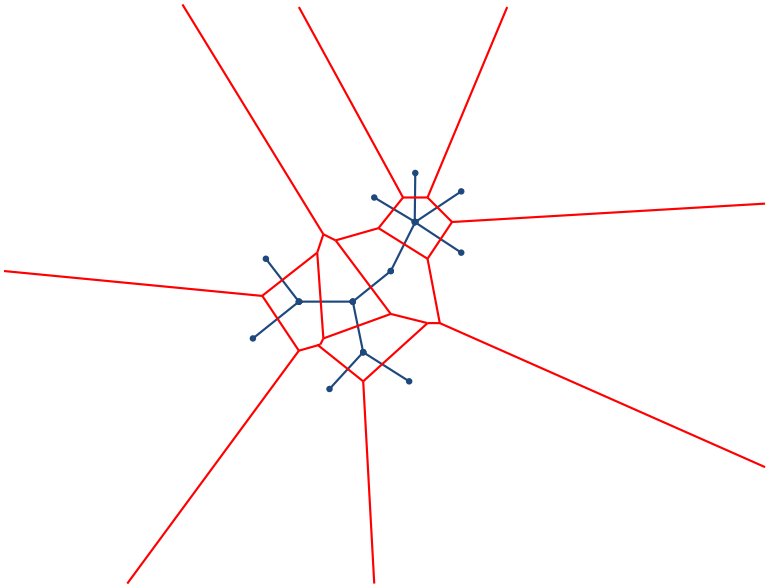


Abbildung 3.5: Die Entwicklung des RRT in große Voronoi-Gebiete. Die roten Linien zeigen die Voronoi-Zellen und die blauen Linien den RRT.

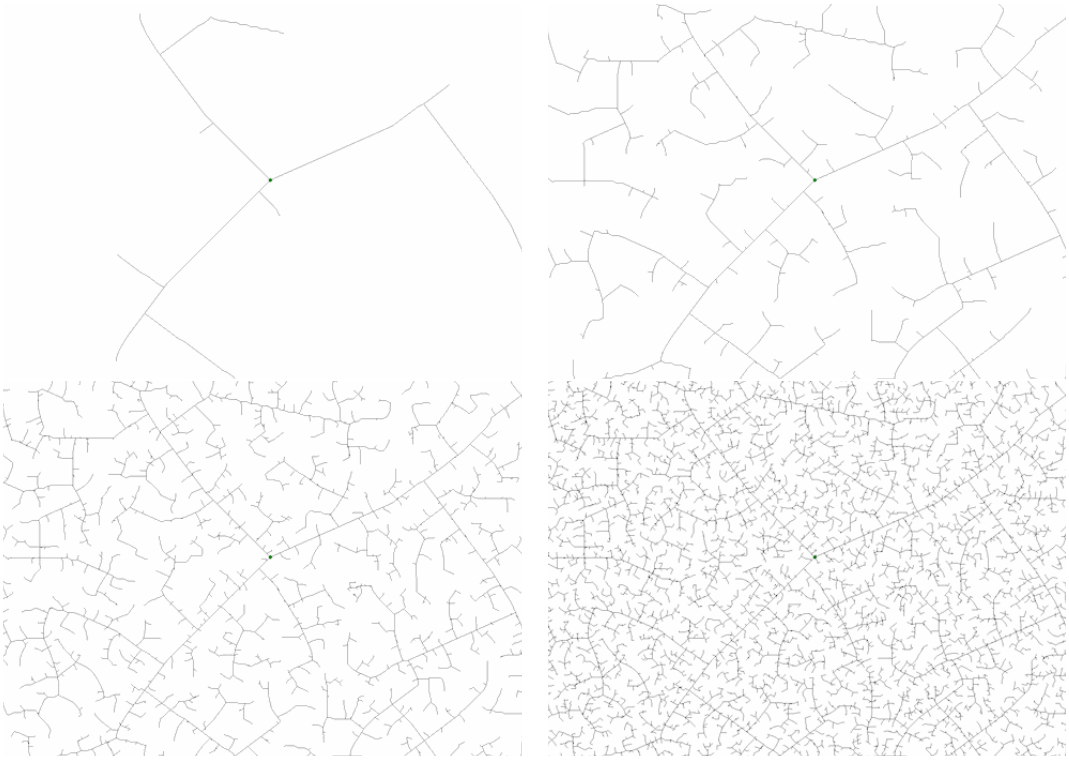


Abbildung 3.6: Ablauf eines RRT nach verschiedener Anzahl an Iterationen. Der Startpunkt befindet sich in der Mitte des Bildes. Zu Beginn erreicht der Algorithmus sehr schnell unerforschte Bereiche [82].

eine wesentlich bessere Performance aufweist. Der erste Unterschied ist, dass der Algorithmus gleichzeitig zwei RRT Bäume, τ_α vom Start und τ_β vom Ziel, aufbaut. Beide Bäume erkunden ihre Umgebung und bewegen sich aufeinander zu. Dies macht ihn wesentlich robuster gegenüber lokalen Minima. Der zweite Unterschied besteht darin, dass die $CONNECT$-Heuristik (siehe Algorithmus 4) versucht, über eine wesentlich größere Strecke zu gehen. Im Gegensatz zur $EXTEND$-Operation des RRT, die den Baum um den Wert ϵ erweitert, führt die $CONNECT$-Operation den $EXTEND$-Schritt aus, bis q oder ein Hindernis erreicht wurde. Diese Art der Ausbreitung erlaubt eine sehr schnelle Konvergenz. Die Algorithmen 3 und 4 zeigen den Ablauf des RRT-connect. In diesem werden die beiden Bäume mit τ_α und τ_β bezeichnet.

Algorithm 3 Der RRT-connect-Algorithmus [63, 83]

```
1: procedure BUILD_RRT_CONNECT(q_init, q_goal)
2:     τ_α.init(q_init); τ_β.init(q_goal);
3:     for k = 1 to κ do
4:         q_rand ← RANDOM_STATE();
5:         if not (EXTEND(τ_α, q_rand, q_new) = Trapped) then
6:             if (CONNECT(τ_β, q_new) = REACHED) then return PATH(τ_α, τ_β);
7:             end if
8:         end if
9:         SWAP(τ_α, τ_β);
10:    end forreturn Failure;
11: end procedure
```

Algorithm 4 Die Connect-Operation des RRT-connect-Algorithmus [63, 83]

```
1: procedure CONNECT(τ, q_new)
2:     repeat
3:         S ← EXTEND(τ, q_new);
4:     until not (S = Advanced) return S;
5: end procedure
```

RRT*

Der RRT* gehört zu der Gruppe der optimierenden Planer, welche häufig wesentlich bessere Pfade, bezogen auf die Pfadlänge, liefern. Grundsätzlich funktioniert der Algorithmus analog zum RRT, lediglich das Hinzufügen einer Kante unterscheidet die beiden. Hier wird nicht einfach der am nächsten gelegene Knoten gewählt, sondern es werden alle Knoten in einer bestimmten Distanz untersucht, und nur die Kante mit den geringsten Kosten, z. B. mit der geringsten Pfadlänge, wird dem Graphen hinzugefügt. Ferner gibt es eine Rewire-Operation, die nach dem Einfügen eines neuen Knoten prüft, ob bereits eingefügte Knoten dadurch kostengünstiger erreicht werden können. Ist dies der Fall, werden entsprechend neue Kanten eingefügt und bestehende, teurere gelöscht, so dass der Graph azyklisch bleibt. Karaman u. a. [84] haben gezeigt, dass sich die Komplexität des RRT* innerhalb eines konstanten Faktors des probabilistisch vollständigen, aber asymptotisch nicht vollständigen RRT befindet.

3.4.2 Multi-Query-Algorithmen

Die Multi-Query-Planer unterschieden sich von den Single-Query-Planern insofern, dass diese zunächst eine *Preprocessing-Phase* haben, in der der Konfigurationsraum nach kollisionsfreien Zuständen durchsucht wird. Während der Durchsuchung legen die Planer eine Datenstruktur an, die in der *Query-Phase* für weitere Anfragen genutzt werden kann. Dies hat den Vorteil, dass neue Anfragen wesentlich schneller erfolgen können, als dies bei Single-Query-Planern der Fall sein würde. In einem statischen Szenario, in dem viele verschiedene Pfade mit unterschiedlichen Start- und Zielpunkten geplant werden sollen, können die Multi-Query-Planer daher durchaus von vorteilhaft sein.

Grundsätzlich lassen sich die Multi-Query-Planer folgendermaßen definieren: Sei $\mathcal{G}(V, E)$ ein ungerichteter Graph mit den Knoten V und den Kanten E in $\mathcal{C}_{free}$, wobei die Algorithmen ihren Namen aufgrund der folgenden beiden Phasen haben [85]:

1. Preprocessing-Phase: Die sogenannte Roadmap wird erstellt und als Graph $\mathcal{G}$ gespeichert. Durch die Knoten des Graphen werden kollisionsfreie Konfigurationen und durch die Kanten kollisionsfreie Bewegungen in $\mathcal{C}_{free}$ dargestellt.
2. Query-Phase: Zunächst werden Start- und Zielpunkt zu $\mathcal{C}_{free}$ hinzugefügt. Danach wird ein Suchalgorithmus, wie z. B. Dijkstra, verwendet, um den kürzesten Weg vom Start zum Ziel zu suchen.

Die folgende Liste zeigt bekannte Vertreter von Multi-Query-Planern:

- Probabilistic Roadmap Method (PRM): Diese Art Planer baut eine Landkarte mit Meilensteinen auf, die kollisionsfreie Zustände im State Space (siehe Abschnitt 4.3.1) repräsentieren. Bekannte Verfahren zum Erstellen dieser sind der Visibility Graph und das Voronoi-Diagramm. Nachdem die Karte aufgebaut wurde, wird z. B. mit A* der kürzeste Pfad gesucht [85].
- Lazy PRM: Der Planer arbeitet ähnlich wie PRM, jedoch reduziert er die Anzahl der Kollisionsprüfungen reduziert. Dazu wird zunächst für alle Zustände davon ausgegangen, dass diese kollisonsfrei sind. Erst wenn in der Roadmap der kürzeste Pfad gesucht wird, werden die Punkte nach Kollisionen durchsucht [86].
- PRM*: Im Gegensatz zum PRM legt PRM* die Anzahl der zu untersuchenden Nachbarn vorher nicht fest, sondern berechnet automatisch je nach Abdeckung eine optimale Lösung [84].
- Sparse Roadmap Spanner Algorithm (SPARS): Dieser Algorithmus arbeitet ähnlich wie PRM, hat allerdings gegenüber diesem einige Vorteile, wie beinahe asymptotische Optimalität und ein aussagekräftiges Abbruchkriterium [87].

Probabilistic Roadmap Method (PRM)

Der wohl bekannteste Multi-Query-Planer ist der PRM, der 1996 von Kavraki u. a. [85] veröffentlicht wurde. Zunächst wird in der *Preprocessing-Phase* die Roadmap erstellt (siehe Abb. 3.7) und als Graph abgespeichert. Der Algorithmus der Preprocessing-Phase ist in Algorithmus 5 dargestellt, wobei $\alpha(i)$ den zufällig generierten i-ten Knoten bezeichnet [54, S.237]. Es wird wiederholt zufällig ein Knoten $\alpha(i)$ erzeugt und geprüft, ob $\alpha(i) \in \mathcal{C}_{free}$. Falls dies zutrifft, wird $\alpha(i)$ zu der Menge V der Knoten des Graphen $\mathcal{G}$ hinzugefügt, was Zeile 4 und 5 zeigen. Falls $\alpha(i) \in \mathcal{C}_{obs}$, wird ein neues Sample generiert. Im nächsten Schritt wird versucht, den neuen

Knoten mit den nächstliegenden Knoten $q \in \mathcal{C}_{free}$ zu verbinden. Jede Verbindung wird durch die $CONNECT$-Operation, bei der es sich um eine LPM handelt, durchgeführt. In den meisten Implementierungen erfolgt hier lediglich die Suche nach dem kürzesten Weg zwischen $\alpha(i)$ und q. Sobald ein kollisionsfreier Pfad gefunden wurde, liefert $CONNECT$ das Ergebnis TRUE.

In der *Query-Phase* wird zunächst davon ausgegangen, dass $\mathcal{G}$ ausreichend groß ist, um die meisten Abfragen zu beantworten. Zunächst werden zwei weitere Iterationen der Query-Phase durchgeführt und der Startpunkt q_0 und der Zielpunkt q_G zum Graphen hinzugefügt. Danach wird durch einen Suchalgorithmus ein Pfad von q_0 zu q_G in $\mathcal{G}$, der automatisch in $\mathcal{C}_{free}$ liegt, gesucht.

Algorithm 5 Erstellung einer Roadmap in der Preprocessing-Phase [54, S. 238]

```
1:  procedure BUILD_ROADMAP
2:      G.init(); i ← 0;
3:      while i < N do
4:          if α(i) ∈ C_free then
5:              G.add_vertex(α(i)); i ← i + 1;
6:              for each q ∈ NEIGHBORHOOD(α(i), G) do
7:                  if (not G.same_component(α(i), q)) ∧ CONNECT(α(i), q)) then
8:                      G.add_edge(α(i), q);
9:                  end if
10:             end for
11:         end if
12:     end while
13: end procedure
```

3.5 Natur-inspirierte Methoden

Evolutionäre Algorithmen (EA) sind populationsbasierte metaheuristische Optimierungsverfahren, die Verhalten der biologischen Evolution imitieren, um sonst unlösbare mathematische Probleme zu lösen [89, S.253]. EA beruhen im Wesentlichen auf der von Charles Darwin (1809–1882) entwickelten Theorie der biologischen Evolution, die er 1859 in seinem Buch *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life* [90] vorgestellt hat. Darin befasst sich Darwin mit der natürlichen Selektion, welche die Grundlage der Evolution bildet. Die Theorie von Darwin kann in einigen wenigen Sätzen zusammengefasst werden: In einer Umgebung mit begrenzten Ressourcen überleben mit großer Wahrscheinlichkeit die Individuen mit den „besten“ Eigenschaften. Diese werden sich vermehren und die guten Eigenschaften an die Nachfahren übertragen, wodurch diese dominant in zukünftigen Populationen sind. Ein weiterer Aspekt seiner Theorie basiert darin, dass zufällige Veränderungen bei der Entstehung eines Kindes wiederum neue Eigenschaften hervorrufen. Sind diese eine Verbesserung für das Lebewesen, werden sie sich in neuen Generationen weiter vermehren [91, S.127].

Bevor die Funktionsweise der EA genauer erläutert wird, müssen zunächst einige biologische Begrifflichkeiten eingeführt werden. Alle Lebewesen bestehen aus Zellen, welche Chromosomen in ihrem Zellkern enthalten. Diese sind Fäden aus Chromatin, einer Kombination von

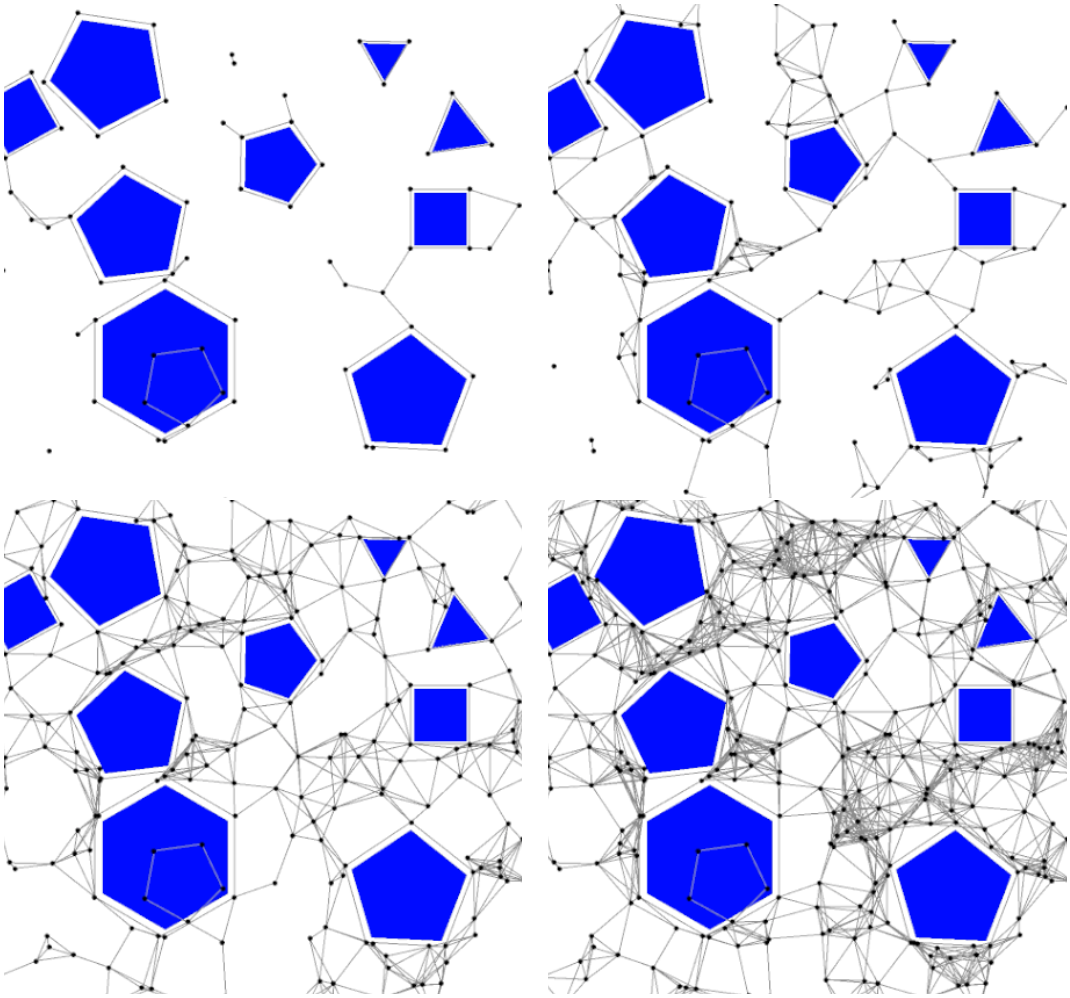


Abbildung 3.7: Beispiel der Preprocessing-Phase des PRM-Algorithmus, der gültige Pfade um eine Reihe von Hindernissen sucht [88].

Desoxyribonukleinsäure (DNA) und Proteinen, und enthalten den größten Teil der Erbinformationen. Die Anzahl der Chromosomen hängt vom Organismus ab; in den meisten Fällen treten sie paarweise auf. Der Mensch hat z. B. 23 verschiedene Chromosomenpaare. Die DNA wiederum besteht aus Genen, welche die kleinste Einheit, die eine erbliche Eigenschaft darstellt, repräsentieren. Der Begriff leitet sich von dem 1889 eingeführten Begriff „Pangen“ ab, der 1909 von Wilhelm Johannsen verkürzt wurde. Die genetische Ausstattung eines Organismus wird als Genotyp bezeichnet [92, S.5]. In den Genen sind die verschiedenen Eigenschaften wie bspw. Haarfarbe, Größe usw. gespeichert. Diese Eigenschaften werden auch Allele genannt. Viele Organismen haben eine Vielzahl an Chromosomen in ihren Zellen. Alle Chromosomen zusammen ergeben die gesamte Erbinformation eines Organismus und werden als Genome bezeichnet. Während der Fortpflanzung von Organismen treten Rekombination oder Crossover auf. Dabei werden einzelne Gene der Chromosomen der Eltern ausgetauscht, wodurch neue Chromosomen aus den Eigenschaften der Eltern entstehen. Zusätzlich können Mutationen auftreten, wenn entweder

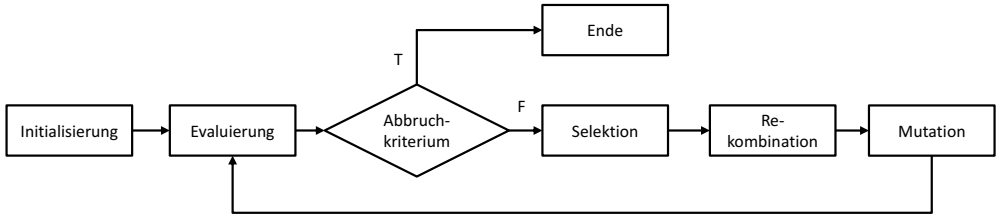


Abbildung 3.8: Ablauf eines EA.

fehlerhaft kopiert wird oder durch Umwelteinflüsse, wie radioaktive Strahlung, Chromosomen verändert werden [93, S.5].

In EA wird der Begriff Chromosom typischerweise für einen Kandidaten einer Problemlösung verwendet. Die Darstellung der Chromosomen kann entweder binär mit einer Liste an Werten erfolgen oder durch eine Baumstruktur. Da für diese Arbeit die Darstellung einer Liste verwendet wurde, wird im Folgenden nur auf diese Darstellung eingegangenen. Ein Gen stellt 1 bis n Blöcke dar. Bei einer binären Darstellung des Gens ist ein Allel eine Ausprägung einer Eigenschaft als 0 oder 1.

In Laufe der Zeit haben sich unterschiedliche Ausprägungen und Namen für EA ausgebildet, die sich hauptsächlich in der Repräsentation der Lösungskandidaten unterscheiden. Werden diese als Bitstrings codiert, bezeichnet man diese Variante als Genetischen Algorithmus (GA). Werden direkte Problemrepräsentationen, wie z. B. reelle Zahlen, verwendet, spricht man von Evolutionsstrategien (ES). Bei der Genetischen Programmierung (GP) werden Strukturen erzeugt, die bestimmte Eingaben in festgelegte Ausgaben verwandeln. Beispiele hierfür sind Computerprogramme, Schaltkreise und mathematische Funktionen. Die Lösungskandidaten werden durch Bäume repräsentiert. Evolutionäre Programmierung (EP) wird dazu verwendet, Strukturen wie bspw. Computerprogramme zu finden, die allerdings nicht durch Bäume, sondern durch endliche Automaten dargestellt werden [94].

Der prinzipielle Ablauf eines EA ist in Abb. 3.8 zu sehen. Zunächst wird in der ersten Iteration $t = 1$ die Population *pop* initialisiert. Je nach Implementierung wird diese Population mit Individuen initialisiert, die zufällig generiert wurden, oder mit welchen, von denen bekannt ist, dass sie bereits gute Kandidaten für die Problemstellung sind. Danach wird für alle Chromosomen von *pop* die Fitness mit der *Objective Function* evaluiert, die ausdrückt, wie gut die Eigenschaften eines Individuums sind. Im nächsten Schritt werden nacheinander die Operationen Selektion, Crossover, Rekombination und Mutation durchgeführt, bis ein Abbruchkriterium erreicht ist. Dieses kann z. B. eine feste Anzahl an Durchläufen oder keine Verbesserung der Fitness in den letzten n Durchläufen sein. Die genaue Funktion der genannten Operationen soll im Folgenden erläutert werden [89, S.254].

Initialisierung

Besteht die Population aus zu wenigen Individuen, hat der EA zu wenig Möglichkeiten, den Crossover-Operator auszuführen, und es wird ein zu kleiner Bereich des Suchraums erkundet.

Wird die Population hingegen zu groß, erhöht sich die Laufzeit des Algorithmus, da wesentlich mehr Berechnungen durchgeführt werden müssen. Gerade im Falle der Pfadplanung sind dies viele teure Kollisionsprüfungen. Zudem gibt es Untersuchungen, die belegen, dass es ab einem bestimmten Schwellwert nicht mehr sinnvoll ist, die Population weiter zu vergrößern [95]. Eine gute Faustregel ist es, die Größe der Population etwa zehnmal so groß wie die Dimension der Problemstellung zu wählen. Ein Industrieroboter hat sechs Freiheitsgrade, wodurch die Population etwa 60 Chromosomen enthalten sollte [96].

Evaluierung

Die Evaluierung erfolgt durch die Fitness-Funktion, die einen entscheidenden Einfluss auf den Erfolg des EA hat. Sie dient dazu, die Qualität von Individuen zu differenzieren und ist daher entscheidend für eine erfolgreiche Implementierung. Die genaue Umsetzung der Funktion für die Pfadplanung von Industrierobotern wird in Abschnitt 4.3.4 beschrieben.

Selektion

In der Natur sorgt die Selektion dafür, dass Lebewesen, die sich ihrer Umwelt sehr gut angepasst haben, eine höhere Überlebenswahrscheinlichkeit haben. Dies hat zusätzlich zur Folge, dass sich diese Individuen mit einer höheren Wahrscheinlichkeit fortpflanzen und somit ihre Eigenschaften in eine neue Generation weitergeben. Im Zusammenhang mit Selektion wird häufig der Begriff Selektionsdruck verwendet. Ist dieser sehr hoch, führen selbst geringe Unterschiede in der Fitness zu erheblichen Unterschieden der Fortpflanzungswahrscheinlichkeit der Individuen. Ein geringer Selektionsdruck hingegen sagt aus, dass die Fortpflanzungswahrscheinlichkeit nur geringfügig von den Fitnessunterschieden abhängt. Für eine sinnvolle Anwendung von EA ist ein gewisser Selektionsdruck nötig, denn ohne diesen wäre die Suche lediglich ein zufälliges Ausprobieren ohne große Erfolgschancen. Es ist wichtig, den Selektionsdruck zu steuern, um ein ausgeglichenes Verhältnis zwischen Erforschung (Exploration) des Suchraums und Ausnutzung guter Individuen zu erreichen. Wird der Selektionsdruck zu groß gewählt, liegt der Fokus von Anfang an auf Individuen, die zufällig in der anfänglichen Population eine sehr gute Fitness hatten. Dies kann dazu führen, dass Gegenden des Suchraums, die in der anfänglichen Population nicht oder nur sehr gering enthalten waren, nie erforscht werden und somit in ihnen liegende Lösungen nie gefunden werden. Am besten ist es daher, den Selektionsdruck zeitabhängig zu gestalten. In frühen Generationen sollte der Druck gering gehalten werden, um eine möglichst hohe Exploration zu erreichen. In späteren Iterationen sollte der Druck erhöht werden, um die besten lokalen Varianten der Individuen zu finden [97, S.197]. In der Praxis wird der Selektionsdruck durch Anpassung der Fitnessfunktion über die Laufzeit oder durch die Wahl einer Selektionsmethode beeinflusst. Fortfolgend werden gängige Methoden vorgestellt:

- Elite-Selektion (Elitist selection): Bei dieser Methode wird das beste Individuum der aktuellen Generation auf jeden Fall in die nächste übernommen. Vorgeschlagen wurde dieser Weg von De Jong [98]; er kann auf die besten B Individuen ausgedehnt werden, wobei gilt $1 \leq B \leq PopulationSize$. Die Elite-Selektion stellt sicher, dass die besten Ergebnisse einer Iteration nicht mehr verloren gehen. Nachteil dieser Methode ist, dass sie schnell Gefahr läuft, eine dauerhafte Stagnation zu verursachen [99, S.71].
- Fitness-proportionale Selektion (Roulette-wheel selection): Bei dieser Art der Selektion werden die Chromosomen anhand ihrer Fitness auf einem virtuellen Glücksrad verteilt. Je größer die Fitness, desto größer ist der Sektor des Chromosoms auf dem Rad. Wird das

Glücksrad gedreht, ist die Wahrscheinlichkeit, dass die Markierung bei einem Individuum mit einer großen Fitness stehen bleibt, deutlich größer als bei einer geringen Fitness. Ein Nachteil dieser Selektionsmethode ist, dass Individuen mit einer großen Fitness die Auswahl sehr stark dominieren können. Das Individuum mit einer großen Fitness hat eine sehr hohe Chance, selektiert zu werden, wohingegen die anderen Individuen eine sehr geringe Chance haben, in die neue Generation zu kommen. Die Dominanz kann von Generation zu Generation noch größer werden. Das Ergebnis ist, dass an manchen Stellen des Suchraumes sehr viele Individuen existieren, während andere Stellen sehr dünn besiedelt sind. Diese Ansammlung wird auch Überbevölkerung genannt. Der Nachteil ist, dass durch die Überbevölkerung eine sehr schnelle Konvergenz in einem lokalen Optimum auftreten kann.

- Stochastisches universelles Stichprobenziehen (Stochastic universal sampling): Diese Art der Selektion ist eine Abwandlung der Glücksradmethode. Während bei Letzterer nur eine Markierung auf dem Rad zur Auswahl genommen wird, werden bei einem universellen Stichprobenziehen n Individuen in jedem Durchgang selektiert. Dazu werden gleichmäßig um das Glücksrad herum Markierungen verteilt. Individuen mit einer großen Fitness werden bei dieser Methode meistens mindestens einmal oder mehrmals ausgewählt, während Individuen mit einer geringen Fitness einmal oder gar nicht ausgewählt werden. Im Allgemeinen liefert diese Art der Selektionsmethode sehr gute Ergebnisse und ist einfach zu implementieren [97, S.202].
- Rangbasierte Selektion (Rank selection): Bei dieser Form der Selektion werden alle Individuen einer Population nach ihrer Fitness sortiert und diesen somit ein Rang zugeordnet. Jedem Rang wiederum wird eine Wahrscheinlichkeit zugewiesen. Je höher der Rang der Individuen, umso höher ist der Wert der Wahrscheinlichkeit. Die Auswahl der Individuen erfolgt wie bei der Roulette-Selektion mit einer Glücksradauswahl. Der Vorteil Rangbasierter Selektion ist allerdings die Entkoppelung des Fitness-Wertes und der Auswahlwahrscheinlichkeit, die in der Roulette-Auswahl proportional zueinander sind. Somit kann die Dominanz der sehr guten Individuen unterdrückt werden, indem man die deren Rangwahrscheinlichkeit nur etwas größer als die der schlechten Individuen wählt. Ein weiterer Vorteil dieses Vorgehens ist, dass der Selektionsdruck mit dem Fortschritt des EA nicht abnimmt, da die Streuung der Rangwahrscheinlichkeiten nicht von der Streuung der Fitness abhängt. Die Entwicklung des Selektionsdrucks lässt sich durch die Anpassung der Rangwahrscheinlichkeiten steuern, indem man nach und nach Wahrscheinlichkeiten von den niedrigen Rängen auf die höheren überträgt. Dadurch wird der EA in den anfänglichen Iterationen eher die Umgebung erforschen und erst später ausnutzen. Ein Nachteil dieser Selektionsmethode besteht darin, dass die Individuen jedes Mal nach der Fitness sortiert werden müssen, was den Aufwand $O(|pop| \cdot log_2|pop|)$ hat [97, S.203].
- Turnier-Selektion (Tournament selection): Hier werden in jedem Durchgang k Individuen zufällig ausgewählt, um an einem Turnier teilzunehmen. Die Auswahl erfolgt mit derselben Wahrscheinlichkeit für alle Kandidaten, völlig unabhängig von deren Fitness. Gewonnen wird das Turnier durch das Individuum mit der größten Fitness. Bei einem Gleichstand, d. h. wenn mehrere Individuen dieselbe Fitness haben, wird zufällig der Gewinner aus diesen gezogen. Nach Beendigung des Turniers werden alle Individuen in die aktuelle Population zurückgegeben. Insgesamt müssen für die Population $|pop|$ Turniere durchgeführt werden, da jedes Mal nur ein Individuum bestimmt wird. Der Selektionsdruck kann bei diese Methode über die Größe des Turnieres $k \in \{2, 3, .., |pop|\}$ gesteuert werden. Je größer das Turnier, umso größer ist der Selektionsdruck [97, S.203].

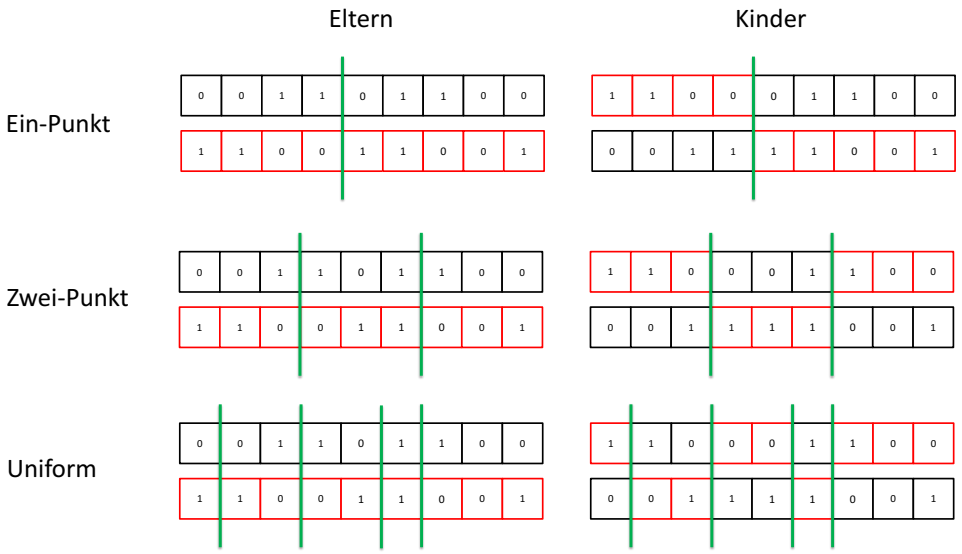


Abbildung 3.9: Die verschiedenen Crossover-Operatoren.

Mutation

Mutation und Rekombination sind die Suchoperatoren evolutionärer Algorithmen. Durch die Mutation werden Allele eines Gens verändert. Dadurch, dass die Veränderungen eher gering ausfallen, bleiben die mutierten Chromosomen sehr ähnlich zu denen ihrer Eltern. Mutation kann an einer oder auch an mehreren Stellen eines Chromosoms vorgenommen werden. Die Mutation wird mit einer vorher definierten Wahrscheinlichkeit durchgeführt. Wird zu stark mutiert, können gute Individuen zu stark verändert werden. Wird allerdings zu wenig mutiert, wird zu wenig im Suchraum exploriert. Um Letzteres zu umgehen, kann mit einer hohen Mutationsrate begonnen werden, die mit der Zeit immer schwächer wird.

Rekombination

Bei der Rekombination entstehen neue Individuen aus der Kombination von Merkmalen bestehender Individuen. Zur Ausführung gibt es den Crossover-Operator, der in verschiedenen Ausprägungen (siehe Abb. 3.9) existiert. Je mehr lokale Optima es in einer Population gibt, desto wahrscheinlicher ist es, dass bei der Rekombination von zwei Individuen ein Nachfahre erzeugt wird, der die Lücke zwischen den beiden Optima füllt [100, S.101 ff.]. Beim einfachsten Typen, dem Ein-Punkt-Crossover, wird zufällig ein Kreuzungspunkt gewählt. An diesem werden die Informationen von zwei Chromosomen ausgetauscht. Die einzelnen Operatoren unterscheiden sich in der Anzahl der Kreuzungspunkte. Folglich hat der Zwei-Punkt-Operator zwei Kreuzungspunkte und der Uniform eine zufällige Anzahl. Wie oft beim Durchlaufen des EA rekombiniert werden soll, wird durch eine Crossover-Wahrscheinlichkeit definiert.

Abbruchkriterium

Es gibt verschiedene Möglichkeiten, einen EA terminieren zu lassen. Beispiele für Terminierungsfunktionen sind in Folge aufgelistet [91, S.140], [101] und [89, S.105]:

- Anzahl der Iterationen bzw. Laufzeit: Der Algorithmus stoppt nach einer maximalen Anzahl an Iterationen. Alternativ kann nach einer fest definierten Zeit abgebrochen werden.
- Erreichen eines Schwellwertes: Erreicht die Fitness einen vorher festgelegten Schwellwert, wird abgebrochen.
- Keine signifikante Veränderung der Fitness: Unterschreitet die Differenz zwischen der aktuellen besten Fitness und dem Mittelwert der n letzten besten Werte einen bestimmten Threshold ϵ, wird abgebrochen.
- Beste-Schlechteste: Ist die Differenz zwischen der besten und der schlechtesten Fitness kleiner ϵ, wird abgebrochen.

3.6 Stand der Technik

Der Stand der Technik gliedert sich in die, in der Taxonomie (siehe Abb. 3.3) eingeführten Gruppen *Graphen-basierte Methoden*, *Sampling-basierte Methoden* und *Natur-inspirierte* Verfahren. Zudem sind die Publikationen in den jeweiligen Bereichen nach Veröffentlichungen, die sich mit der Planung für einen Roboter und für mehrere beschäftigten, unterteilt.

3.6.1 Graphen-basierte Methoden

Diese Art der Pfadplanung wird sehr häufig für mobile Roboter in 2D verwendet und kommt eher selten für Industrieroboter zum Einsatz. Dennoch gibt es einige Veröffentlichungen, die allerdings schon etwas älter sind. [102] stellen einen Graphen-basierten Algorithmus vor, der im Konfigurationsraum arbeitet und eine lineare Laufzeit bezogen auf die Anzahl der Knoten hat. [103] stellt einen Algorithmus für die Planung eines 6-DOF-Industrieroboters vor. Das Verfahren beruht auf einem *best first search* im Konfigurationsraum. Dazu werden freie und kollisionsbehaftete Zustände gespeichert und anschließend anhand einer Kostenfunktion ein Pfad gesucht. In [104] wird der Arbeitsraum zunächst in ein Raster aufgeteilt, in dem die Knoten als frei bzw. kollisionsbehaftet markiert werden; in einem zweiten Schritt wird eine Tiefensuche in dem vorher erstellten Graphen durchgeführt, um einen Pfad zu finden.

3.6.2 Sampling-basierte Methoden

Sehr prominente Vertreter im Bereich der Pfadplanung sind die Sampling-basierten Verfahren. Sie können auch in sehr komplexen Umgebungen mit hohen Freiheitsgraden, gute Lösungen finden.

Single-Roboter-Systeme

Oh u. a. [105] stellen den Retrieval-RRT-Strategy (RRS) Algorithmus vor; dieser kann Änderungen in der Umgebung eingehen kann. Er verwendet den RRT und besteht aus zwei Stufen. Im ersten

Schritt wird eine Support Vector Machine (SVM) verwendet, die aus vorher gespeicherten RRT-Lösungen eine Lösung für das aktuelle Problem heraussucht. Wird keine Lösung gefunden, wird eine solche mit dem RRT berechnet und danach ebenfalls gespeichert. Ausgeführt wird der Algorithmus lediglich in einer Matlab-Simulink-Simulationsumgebung.

Multi-Roboter-Systeme

Koga u. a. [106] haben die erste Veröffentlichung zum Finden eines Pfades für eine geschlossene kinematische Kette bestehend aus mehreren Manipulatoren, die ein festes Objekt transportieren, vorgelegt. Es wird ein Verfahren vorgestellt, das zunächst einen Pfad für das zu bewegende Objekt mit dem Randomized-Path-Planner (RPP) Algorithmus berechnet. Im zweiten Schritt wird für jeden dieser Zustände sowie alle beteiligten Manipulatoren mithilfe der IK geprüft, ob es eine valide Konfiguration gibt. [107] erweitert den PRM Algorithmus um Lösungen für geschlossene Ketten, bei denen sich die Verbindung zwischen den Ketten nicht ändert. Die Grundidee ist, Optimierungstechniken zu verwenden, um lokale Pfade für die geschlossene Kette zu finden. Der Algorithmus wird in der Arbeit anhand einer 2D-Umgebung validiert. Han u. a. [72] stellt ein System vor, bei dem zwei PUMA Roboter auf einer mobilen Plattform in einer Simulationsumgebung gemeinsam eine feste Stange transportieren. Das Verfahren beruht ebenfalls auf dem PRM-Algorithmus. Die geschlossene Kette wird in eine Kette für den aktiven Manipulator und mehrere Unterketten für die passiven Manipulatoren aufgeteilt. Für den Aktive wird mit dem PRM eine Lösung gefunden, und für die Passiven wird über die IK versucht, eine valide Stellung zu finden. Der Nachteil dieses Verfahrens ist, dass durch die Bestimmung des aktiven Manipulators schon Lösungen verloren gehen können, da für den passiven Manipulator nur nach einer Lösung gesucht wird, wenn der aktive eine Lösung im Arbeitsraum hat. Cortés [73] und Cortés u. a. [108] stellten den Random Loop Generator (RLG) vor, der den Ansatz von Han u. a. [72] soweit erweitert, dass bei jedem generierten Sample sowohl für den aktiven Manipulator als auch für den passiven eine Lösung gefunden wird.

3.6.3 Natur-inspirierte Methoden

EA sind in den letzten Jahren sehr populär für die Pfadplanung geworden. Dies ist darin begründet, dass für die Algorithmen theoretisch und empirisch bewiesen wurde, dass sie auch in sehr komplexen Umfeldern in der Lage sind, ein globales Optimum zu finden [109]. Generell kann festgehalten werden, dass EA domänenunabhängig sind und für sehr viele Probleme eingesetzt werden können. Es sei bereits jetzt vorweggenommen, dass es zwar eine Vielzahl an Publikationen zum Thema Pfadplanung gibt, sich allerdings nur sehr wenige dieser mit der konkreten Anwendung im industriellen Umfeld beschäftigen. Die meisten Publikationen beschränken sich auf mobile Robotik oder Manipulatoren mit wenigen Freiheitsgraden im zweidimensionalen Raum.

Single-Roboter-Systeme

Der Einsatz von EA zur Pfadplanung ist nicht neu, allerdings beschränkt sich deren Einsatz häufig auf 2D-Applikationen. Bei diesen kommen diese sowohl für mobile Roboter als auch für 2D-Manipulatoren zum Einsatz.

Die folgenden Arbeiten [110, 111, 112, 113, 114, 115, 116] beschäftigen sich mit der Pfadplanung für mobile Roboter. Dabei bestehen die Chromosomen in den meisten Fällen aus einer Liste XY-Koordinaten von Pfadpunkten. Der EA wird dazu verwendet, eine Lösung zu berechnen, deren

Pfadlänge möglichst klein ist. In manchen Implementierungen wird die Fitness-Funktion noch um weitere Merkmale, wie z. B. die Glätte des Pfades erweitert. Um eine Kollisionsfreiheit des Pfades zu garantieren, bestrafen alle Autoren bei einer auftretenden Kollision das entsprechende Chromosom mit einem sehr kleinen Fitness-Wert.

Es gibt eine Vielzahl an Publikationen zum Thema Pfadplanung für Manipulatoren mit zwei bis zehn Gelenken im zweidimensionalen Raum [117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131]. In diesem Fall bestehen die Chromsomen ebenfalls aus einer Liste von Pfadpunkten, die entweder Punkte im kartesischen Raum oder die entsprechenden Achsstellungen des Roboters an einem Punkt repräsentieren. Die Werte werden entweder als reelle Werte oder als Binärwerte dargestellt. Zur Berechnung der Fitness verwenden die meisten Publikationen die Pfadlänge, Achsbewegung, Entfernung des TCP zur Zielposition sowie die auftretende Kollisionen. Manchmal werden EA auch nicht zur Pfadplanung sondern zur Optimierung einer Kostenfunktion bei der Pfadplanung verwendet [71].

Die EA wurden ebenfalls verwendet, um Pfadplanung im 3D-Raum zu realisieren. In diesem Fall repräsentieren die Chromosomen eine Liste aus Pfadpunkten. Häufig werden die euklidische Distanz oder die Manhattan-Distanz verwendet, um die Fitness zu bewerten. Hauptsächlich unterscheiden sich die folgenden Arbeiten in weiteren Kriterien, die zur Bestimmung der Fitness verwendet werden, und in der Art und Weise, wie die Kollisionserkennung implementiert wurde. Manche Veröffentlichungen nutzen auch den Konfigurationsraum, um die Berechnung der IK zu umgehen.

[132] stellt einen EA vor, der für einen 6-DOF-Roboter des Typs PUMA-560 in einer 3D Simulationsumgebung versucht, den kürzesten Pfad zu finden. [133, 134] nutzen ebenfalls einen PUMA-560-Roboter. Der vorgestellte Algorithmus wird verwendet, um in einem sehr einfachen Szenario, bei dem der Roboterarm kollisionsfrei zwischen zwei Kugeln durchfahren soll, einen Pfad zu berechnen. Die Roboter werden bei der Kollisionserkennung durch Zylinder angenähert. [135] nutzen einen EA in Kombination mit einem Hill-Climbing-Algorithmus, um Konfigurationen für einen 6-DOF-Roboter für die Bearbeitung eines Werkstückes zu erzeugen. Der Pfad wird anhand der Fahrtzeit, der Kollisionsfreiheit und der Vermeidung von singulären Konfigurationen bewertet. Die Kollisionen werden über eine Punktewolken-Darstellung des Roboters realisiert, was eine sehr gute Annäherung an die reale Geometrie liefert. Die Planung beschränkt sich auf einen relativ kleinen Raum eines Werkstückes, welches bearbeitet werden soll. In [109] wird ein Pfadplanungsalgorithmus für einen 5-DOF- und einen 7-DOF-Roboter in 2D präsentiert und auf einem PUMA-ähnlichen 7-DOF-Manipulator in 3D angewendet, der auf einem genetischen Algorithmus beruht. Die Berechnung wird im Kartesischen Raum durchgeführt. Um Kollisionsfreiheit zu erreichen, werden die Achsen immer in einem Sicherheitsabstand zu Objekten gehalten. Es wird nur die Vorwärtskinematik des Roboters benötigt, wobei die Chromosomen binär codiert werden. Die Roboter werden in 3D lediglich als Linien dargestellt. Der Algorithmus wird in verschiedenen Szenarien ausprobiert und anschließend dessen Worst-case-Laufzeit abgeschätzt.

Multi-Roboter-Systeme

Wie im Falle der probabilistischen Algorithmen gibt es auch im Bereich der EA kaum Veröffentlichungen, die sich mit dem Problem der Pfadplanung für Multi-Robotersysteme in industriellen Anwendungen beschäftigen. [136] stellt einen Co-evolutionären Algorithmus vor, der für 2-DOF-Roboter in einer 2D-Umgebung, die sich denselben Arbeitsraum teilen, einen Pfad plant. Der Algorithmus wird mit zwei Populationen initialisiert, wobei jeweils eine Population einen Roboter repräsentiert. Als Fitness werden die Anzahl der Kollisionen der beiden Roboter, die Pfadlänge,

das Bewegungsprofil und die absolute Bewegung der Achsen herangezogen. [137] stellen eine adaptiven Multi-Chromosomen-EA vor, der in der Lage ist, Aufgaben für modulare Systeme mit n Robotern im 3D, die sich im selben Arbeitsraum bewegen, zu planen. Ein Chromosom besteht aus n Listen, die Roboterposen repräsentieren. Curkovic u. a. [138] stellen einen Pareto-basierten Co-evolutionären Algorithmus vor, der den Pfad für zwei 6-DOF-Roboter des Typs FANUC LrMate 200iC plant. Der Fokus liegt allerdings bei der Planung nicht auf der Kooperation der Roboter. Vielmehr geht es darum, die beiden Roboter so zu koordinieren, dass diese nicht kollidieren.

Kapitel 4

Entwicklung eines Frameworks zur Pfadplanung von Industrierobotern

Kapitel 4 gibt einen Überblick über die Architektur des Kollisionsfreie-Kooperation (KoKo) Planungs-Frameworks gegeben. Zusätzlich wird deteilliert auf die Implementierung der einzelnen Module eingegangen.

4.1 Frameworkarchitektur

Im Laufe dieser Arbeit wurde eine Simulationsumgebung entwickelt, die es ermöglicht, Bahnplanungsalgorithmen an Industrierobotern zu validieren. Die Software wurde hauptsächlich in C# mit dem Grafik-Framework Windows Presentation Foundation (WPF) umgesetzt. Zusätzlich wurden auch Planungsalgorithmen in C++ integriert. Grundsätzlich ist die KoKo-Umgebung in drei Hauptmodule *Visual*, *Planning* und *Excecution* unterteilt. Abbildung 4.1 zeigt eine Übersicht über die Haupt- und Untermodule.

Visual stellt Funktionalitäten zur Verfügung, die für die Planung benötigt werden. Dazu gehören die Verwaltung der kompletten 3D-Szene durch den RenderManager, die Kinematik- und Bewegungsberechnung, Koordinatentransformationen und die Kollisionserkennung. Diese Funktionalitäten wurden im Rahmen der dieser Arbeit zugrunde liegenden Forschung umgesetzt und durch existierende Bibliotheken ergänzt.

Im *Planning* Modul sind die verschiedenen Bahnplaner implementiert. Es wurden sowohl Sampling-basierte Verfahren aus Open Motion Planning Library verwendet als auch Planer aus dem Bereich der Computational Intelligence implementiert.

Der *Execution*-Teil stellt die Schnittstelle zur Roboter-Hardware dar, um die geplanten Pfade im realen Prozess auszuführen.

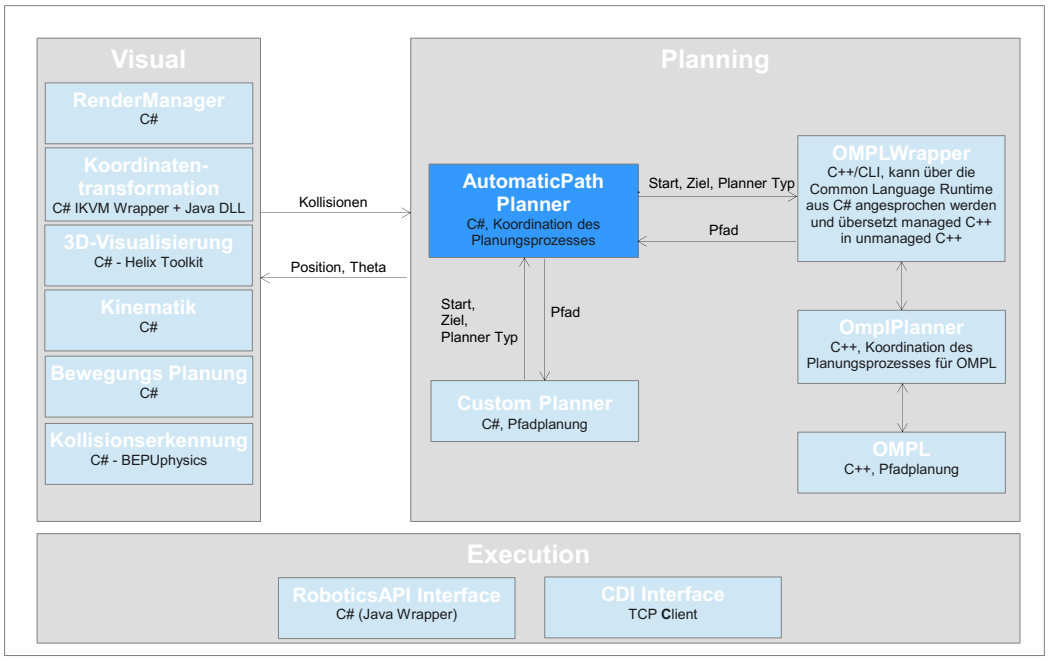


Abbildung 4.1: Architekturübersicht des KoKo-Planungsframeworks.

4.2 Visual

Unter dem Begriff *Visual* werden alle Funktionen zusammengefasst, die zur Darstellung und Kollisionserkennung nötig sind. Abbildung 4.2 zeigt einen Screenshot der KoKo-Umgebung. In dem Fenster auf der linken Seite wird die 3D-Szene dargestellt. Aktuell wurden ein Roboter, an dem ein zusätzlicher Endeffektor befestigt wurde, und eine Werkzeugform geladen. Zudem ist zu sehen, dass eine Kollision zwischen Werkzeugform und Endeffektor von der BEPU-Physik-Engine [139] erkannt und in der Visualisierung rot dargestellt wurde. Die rechte Seite zeigt die Bedienelemente. Unter dem Reiter *Elements* werden in einer Baumstruktur alle aktuell visualisierten Objekte dargestellt. Elemente können entweder programmatisch oder durch Rechtsklick auf die Baumknoten im rechten Fenster hinzugefügt werden. Durch einen Doppelklick auf ein Element kann dieses sichtbar bzw. unsichtbar gemacht werden. Unter dem Reiter *Robot* sind Funktionen hinterlegt, um einen zuvor im Übersichtsbaum selektierten Roboter zu verfahren. Dies kann sowohl kartesisch als auch achsweise sein. Unter *Movements* können Objekte, wie z. B. die Werkzeugform oder Koordinatensysteme, verschoben werden. Die Funktionen unter *Robot* und *Movements* können für Testzwecke, wenn also etwas schnell ausprobiert werden soll, sehr hilfreich sein. Unter dem Reiter *Automatic Path Planning* sind die wichtigsten Bedienfunktionen für die Pfadplanung zusammengefasst.

4.2.1 RenderManager

Der *RenderManager* ist das Herzstück der Visualisierung. In diesem werden alle an der Szene beteiligten Objekte verwaltet. Die zu visualisierenden Objekte leiten sich alle von der abstrakten Klasse *RenderObject* ab; diese stellt die Grundeigenschaften wie Name, Position (X, Y, Z),

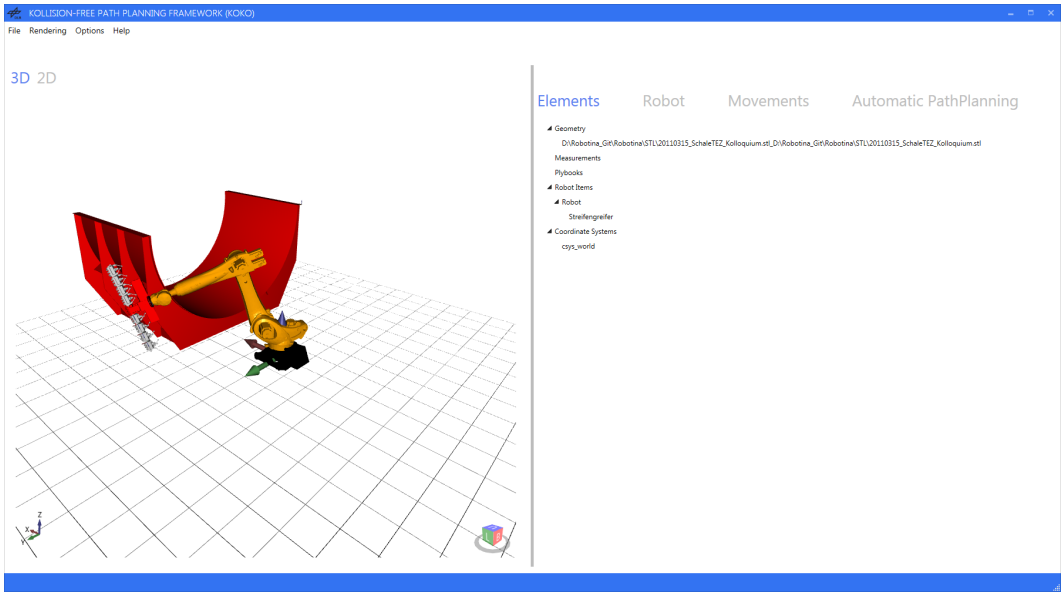


Abbildung 4.2: Screenshot der GUI des KoKo-Frameworks.

Orientierung (A, B, C), Skalierung, Sichtbarkeit, Farbe usw. zur Verfügung. Die wichtigsten abgeleiteten Objekte sind das *CollisionObject*, *CoordinateSystemObject*, *PathObject*, *RobotObject*, *ToolObject* und das *PlybookObject*. Abbildung 4.3 zeigt die Abhängigkeit der Objekte anhand eines Klassendiagramms.

CollisionObject

Auch das *CollisionObject* ist eine abstrakte Klasse, von der das *STLObject* und das *CatenaryObject* erben. Beide können von der Kollisionsberechnung berücksichtigt werden. CAD-Modelle, die in die Visualisierung geladen werden sollen, verwaltet das *STLObject*. Im *CatenaryObject* wird aus den Parametern der Kettenlinie ein Mesh inklusive CFK-Textur berechnet. Bei Änderung von

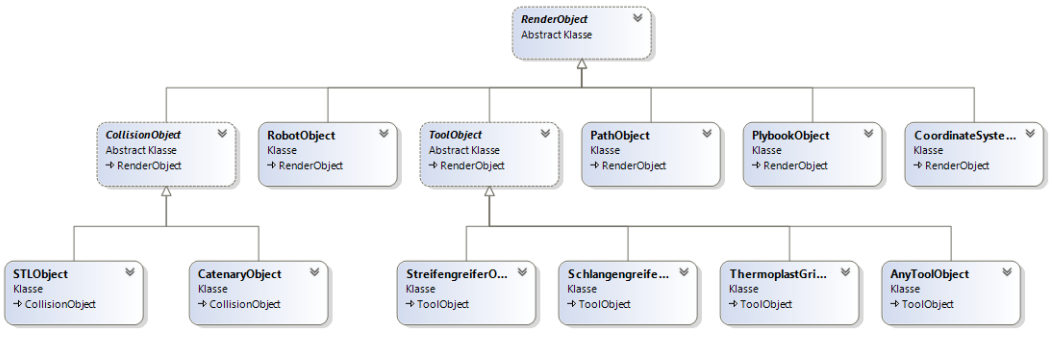
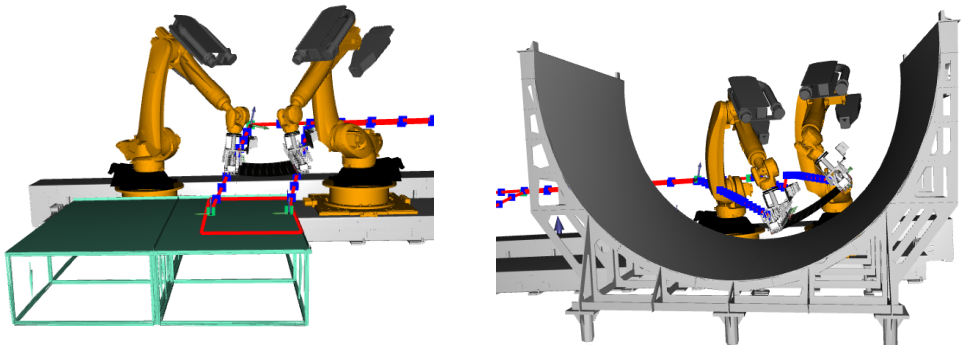


Abbildung 4.3: Übersicht über die vom RenderObject abgeleiteten Objekte.

thetaA, der *GripDistance* oder *S* wird die Kettenlinie (siehe Abschnitt 4.2.7) neu berechnet und das entsprechende *CollisionObject* aktualisiert.



(a) Aufnahme (b) Ablage

Abbildung 4.4: Visualisierung der Kettenlinie bei Aufnahme und Ablage (Screenshot aus KoKo).

CoordinateSystemObject

Das *CoordinateSystemObject* repräsentiert ein Koordinatensystem; dieses kann sehr hilfreich zur Markierung von z. B. TCP und zu Debugging-Zwecken sein.

Koordinatensysteme können bei der späteren Pfadplanung auch dazu verwendet werden, Stützpunkte, die bei der Pfadplanung als Zwischenpunkte berücksichtigt werden sollen, zu definieren. Die Pfadplanung verläuft vom Startpunkt über die Stützpunkte zum Zielpunkt. Ein Koordinatensystem bzw. Stützpunkt kann in dem GUI unter dem Reiter *Elements* mit einem Rechtsklick auf den Punkt *Coordinate System* und anschließenden Klick auf *Add new coordinate system* hinzugefügt werden. In einem Auswahldialog (siehe Abb. 4.5) müssen danach in einer Zeichenfolge, separiert durch ein „;“, die X-, Y-, Z-, A-, B-, C-Werte und der Skalierungsfaktor angegeben werden (z. B. 1000;1500;200;10;20;35). Soll das Koordinatensystem als Stützpunkt fungieren, muss zudem noch „;SP“ an die Zeichenfolge angehängt werden. Bei Aufruf eines Planungsalgorithmus mittels der Funktion *CalculatePath* werden die Stützpunkte in der Datei *supportPoints.txt* gespeichert und beim erneuten Laden der Szene wieder geladen. Stützpunkte bieten den Vorteil, dass bei guter Wahl der Punkte die Planungszeit enorm verkürzt und ein Sicherheitsabstand zu bestimmten Objekten definiert werden kann.

PathObject

PathObject wird verwendet, um einen berechneten Pfad zu visualisieren. Ein Pfad welcher durch Punkte und verbindende Linien dargestellt wird, besteht aus einer Liste von mindestens zwei Punkten (Start und Ziel). Das *PathObject* ist z. B. in Abb. 4.4 durch blaue Punkte, die durch rote Linien verbunden sind, dargestellt.

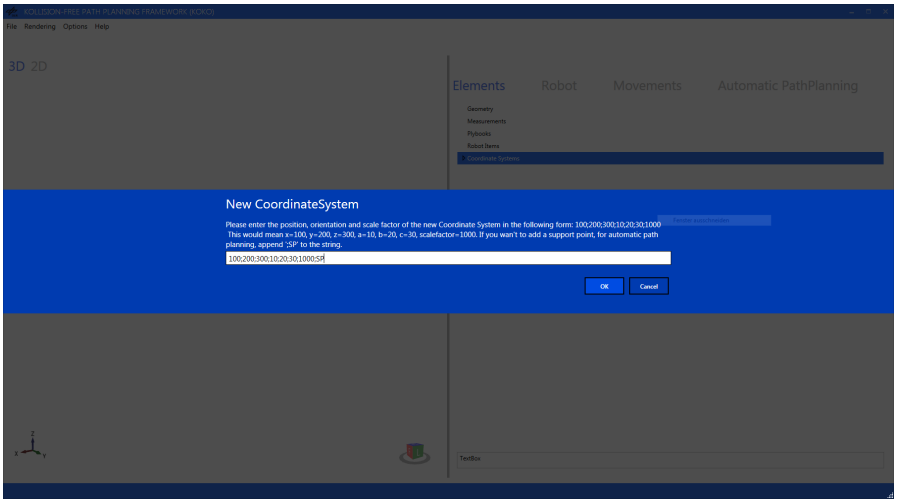


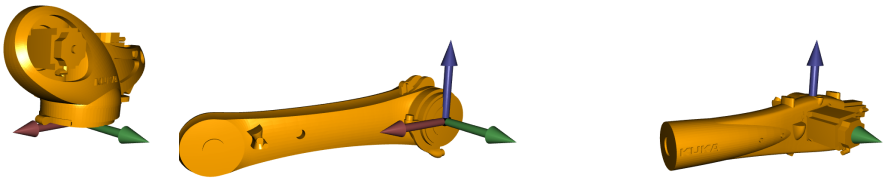
Abbildung 4.5: Einfügen eines Koordinatensystems in KoKo.

RobotObject

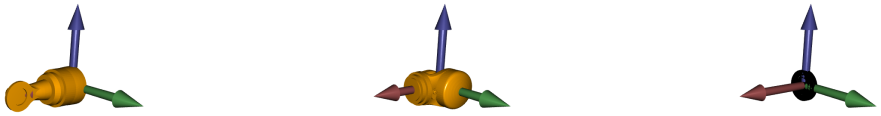
RobotObject ist das komplexeste *RenderObject* und dient zur Repräsentation eines Sechs-Achs-Industrieroboters. Es besteht aus einzelnen *STLObjects*, die über die Vorwärtskinematik, die durch die RoboticsAPI [140] berechnet wird, zusammengesetzt werden. Zusätzlich unterstützt das *RobotObject* eine externe Achse. Daher werden intern folgende Transformationen durchgeführt:

1. Verschiebung des Fußpunktes (Robroot) des Roboters: Für die Verschiebung wurden bei der Implementierung die Werte aus der Roboter Steuerung, die der DH-Notation entsprechen, übernommen. Zunächst wird eine Verschiebung des Frames *ERSYSROOT* berechnet; dieses beschreibt den Offset zwischen Ursprung der Linearachse und des Weltkoordinatensystems. Danach wird das Koordinatensystem durch das Frame *ET1_TA1KR* so gedreht, dass die Z-Achse in Fahrtrichtung der externen Achse zeigt. Durch das Frame *ET1_TFLA3* wird das Koordinatensystem in das Robroot transformiert. Will man in KoKo einen Roboter ohne Linearachse verwenden, müssen die beiden Frames *ET1_TA1KR* und *ET1_TFLA3* für alle Werte null enthalten. Dies entspricht lediglich einer Verschiebung des Robroot bzgl. des Weltkoordinatensystems.
2. Achstransformationen und Roboter-TCP: Der Roboter besteht aus sechs einzelnen Achsen, siehe Abb. 4.6. Für deren richtige Positionierung in der Visualisierung müssen die Achsen als Stereolithography (STL) Modell vorliegen. Des Weiteren muss, wie in Abb. 4.6 abgebildet, bei jeder Achse der Ursprung des Modells in der Drehachse liegen. Zudem entspricht die Rotationsachse der DH-Notation. Da mit den reinen DH-Parametern aus Tabelle 2.1 die Positionierung der CAD-Modelle der einzelnen Achsen für die Visualisierung nicht möglich ist, müssen noch die Werte $d_1 = -675$ in 225,6 und 449,4 und $d_4 = -1400$ in 962 und 438 aufgeteilt werden. Somit kann intern im *RobotObject* der Drehpunkt der CAD-Modelle an die richtige Stelle positioniert werden. Zusätzlich ergibt sich die TCP-Position des Roboters anhand der aufeinanderfolgenden Transformationen.
3. Standardmäßig ist der TCP des Roboters deckungsgleich mit dessen *FLANGE* Koordinatensystem des Roboters. Kommt ein Endeffektor zum Einsatz, können in der Simulation

die entsprechen Tooldaten berücksichtig werden. Hierzu wird intern im *RobotObject* eine weitere Transformation durchgeführt.



(a) Achse A1 (b) Achse A2 (c) Achse A3



(d) Achse A4 (e) Achse A5 (f) Achse A6

Abbildung 4.6: Darstellung der einzelnen Achsen des *RobotObject* inklusive Markierung des Ursprungs der Drehpunkte der Roboterachsen durch ein Koordinatensystem.

Um einen weiteren 6-Achs-Industrieroboter in die Simulation einzubinden, müssen sowohl die DH-Parameter als auch die Verschiebungsparameter für die Visualisierung bekannt sein. Des Weiteren müssen die CAD-Modelle der einzelnen Achsen, wie in Abb. 4.6 beschrieben, im STL-Format vorliegen.

Zusätzlich verwaltet das *RobotObject* noch ein *CarryObject*, welches am TCP des Roboters befestigt und von diesem transportiert werden kann. Dieses ist nötig, um z. B. bei der Aufnahme den CFK-Zuschnitt am Streifengreifer zu fixieren und bei der Ablage zu lösen. Im *RobotObject* wird das *CarryObject* über die Methode *SetCarryObject* hinzugefügt. Abbildung 4.7 zeigt die Beziehung zwischen *Render-, Carry-* und *RobotObject*.

ToolObject

ToolObject dient dem Laden der Modelle von Endeffektoren und deren Befestigung am TCP des Roboters. Da die in dieser Arbeit verwendeten Endeffektoren selbst teilweise eine eigene Kinematik haben, wurden von der Klasse *ToolObject* weitere Klassen *AnyToolObject*, *StreifengreiferObject*, *SchlangengreiferObject* und *ThermoplasticGripperObject* abgeleitet (siehe Abb. 4.8). Intern setzten sich das *ToolObject* und die abgeleiteten Klassen aus *STLObjects* zusammen. Die Klasse *AnyToolObject* kann verwendet werden, wenn lediglich ein einfaches CAD-Modell als Endeffektor an dem Roboter befestigt werden soll. Nachteil ist, dass der Endeffektor in diesem Fall in der Kollisionserkennung nur durch eine konvexe Hülle repräsentiert wird. Soll dieser genauer

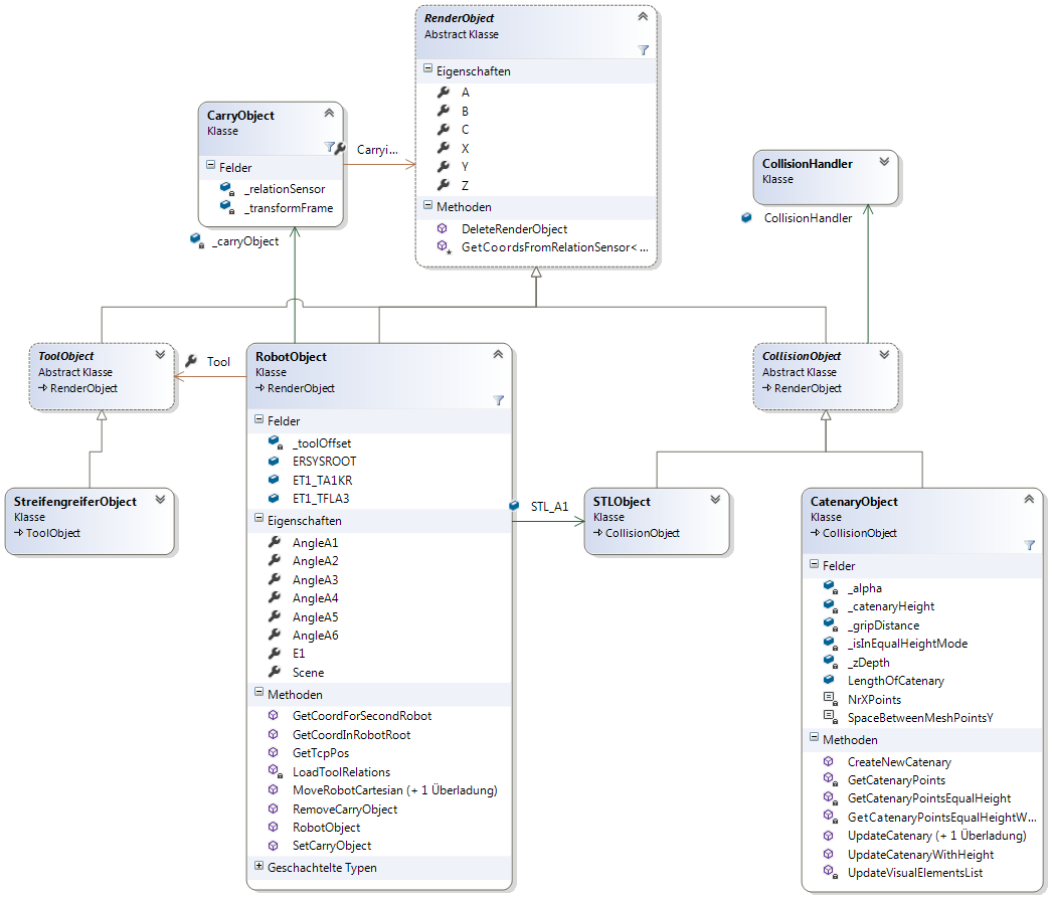


Abbildung 4.7: Klassendiagramm des *RobotObject* und des *CarryObject*.

abgebildet werden, empfiehlt es sich, KoKo um eine eigene Klasse, die von *ToolObject* erbt, zu erweitern.

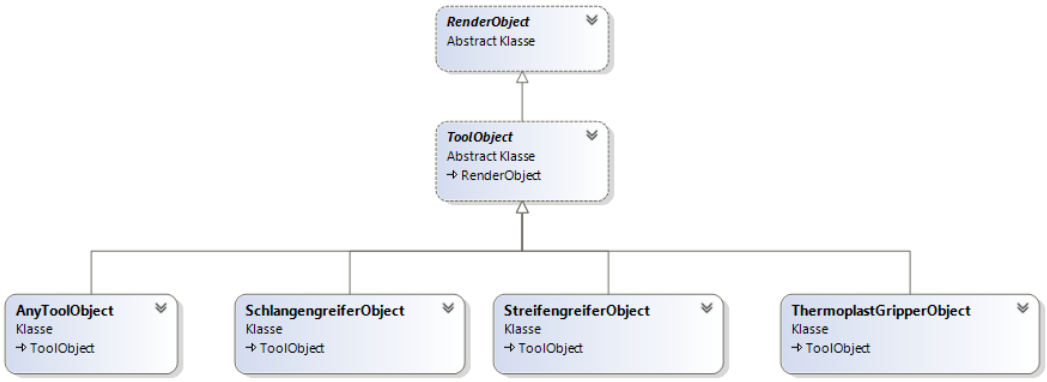


Abbildung 4.8: Klassendiagramm des *ToolObject*.

PlybookObject

PlybookObject ist ein Objekt zur Verwaltung des in Abschnitt 2.9 erläuterten Plybooks. Um das Plybook zu laden, wurde in KoKo zudem noch ein Importer implementiert; dieser liest die entsprechenden Daten aus der im XML Format vorliegenden Datei aus. Die Struktur des geladenen Plybooks kann ebenfalls im Objektbaum angesehen werden. Dort können aber auch das gesamte Plybook oder Teile davon unsichtbar gemacht werden. Zudem können per Tooltip weitere Informationen zu den einzelnen Plys, wie z. B. Material, Drapier-Richtung usw., angezeigt werden.

4.2.2 Koordinatentransformation

Um die Positionierung der einzelnen 3D-Objekte zur Visualisierung und die Beziehungen der Basis- bzw. Werkzeug-Koordinatensysteme zu berechnen, sind zahlreiche Matrixoperationen notwendig. Da bei deren Implementierung schnell Fehler auftreten können, wurde auf die RoboticsAPI der Universität Augsburg [140, 141] zurückgegriffen. Die Bibliothek ist in Java implementiert und bietet zusätzlich einen Wrapper für C#.

Die folgenden Begriffe sind hilfreich bei der Arbeit mit der RoboticAPI:

- Frame: Ein Frame ist ein Punkt inklusive Orientierung im kartesischen Raum. Es ist vergleichbar mit einem Koordinatensystem, dessen exakte Position und Orientierung jedoch erst in Relation zum Weltkoordinatensystem definiert wird (*WorldOrigin*).
- Transformation: Durch eine Transformation lässt sich eine geometrische Verschiebung im Raum beschreiben. Diese besteht aus einem Vektor und einer Rotation.
- Relation/Connection: Eine Relation beschreibt die Beziehung von zwei Frames. Eine Connection hat dieselbe Funktion, ist jedoch dauerhaft, während eine Relation hinzugefügt und gelöscht werden kann.

- Static Connection: Eine Static Connection ist eine Connection mit konstanter Transformation. Diese wird z. B. für die konstante Verschiebung des RobRoot bezogen auf das Weltkoordinatensystem verwendet.
- Dynamic Connection: Bei einer Dynamic Connection ist die Beziehung zwischen zwei Frames veränderbar und wird z. B. für die Verbindung von zwei Roboterachsen im *Visual* verwendet. Variiert die Drehung einer Achse, hat dies zur Folge, dass das Frame am Ende der Achse seine Position ändert.
- Relation Sensor: Der Relation Sensor misst den geometrischen Offset (Translation, Rotation) zwischen zwei Frames. Sie werden verwendet, um die Beziehung der Frames der einzelnen Achsen des Roboters abzufragen. Sobald der Roboter bewegt wird, ändern sich diese.

4.2.3 3D-Visualisierung

Helix Toolkit [142] ist eine Open-Source-3D-Bibliothek, die auf .NET basiert und aktuell auf die WPF Plattform fokussiert. Das Ziel dieser Bibliothek ist, 3D-Programmierung für WPF zu erleichtern und weitere Features zu bieten, die in den Standard-WPF-3D-Funktionen nicht enthalten sind. Im *Visual* dient Helix Toolkit für die Visualisierung sämtlicher 3D-Objekte. Das Visualisierungsfenster ist in der GUI auf der linken Seite zu sehen (siehe Abb. 4.2).

4.2.4 Kinematik

Die direkte Kinematik wurde mithilfe der RoboticAPI, wie unter *RobotObject* beschrieben, implementiert. Die IK wurde mit einem analytischen Verfahren, welches in [16] vorgestellt wurde, umgesetzt und ermöglicht den Einsatz vieler serieller Manipulatoren mit orthoparalleler Basis und einem sphärischem Handgelenk. Die genaue Umsetzung thematisiert Abschnitt 2.4.

4.2.5 Bewegungsplanung

Um die geplanten Pfade vor der Ausführung auf der Roboter-Hardware realistisch zu simulieren, sind in KoKo die Bewegungsprofile Linear, Asynchron-, Synchron- und Voll-Synchron-PTP implementiert. Detailliert beschreibt diese Profile Abschnitt 2.5. Des Weiteren kann eine Geschwindigkeit für die Bewegung angegeben werden.

4.2.6 Kollisionsüberprüfung

Der *RenderManager* verwaltet zusätzlich zu den visualisierten Objekten einen BEPU Physic Space, in dem alle Kollisionsobjekte, sogenannte *Entities*, enthalten sind. Werden Objekte in der Visualisierung bewegt, sorgt der *RenderManager* zusätzlich dafür, dass der BEPU Space aktualisiert wird. Es wird von BEPU lediglich die Funktionalität der Kollisionsprüfung verwendet. Die übrigen typischen Funktionen der Physik Engine, wie Impulsverhalten, werden nicht genutzt. Zur Repräsentation von Kollisionsobjekten nutzt BEPU verschiedene Arten, die unterschiedlich detailliert sind und dementsprechend unterschiedliche Anforderungen an die Rechenzeit haben.

- Konvexe Hülle: Wie in Abschnitt 2.8 beschrieben sind konvexe Hüllen ein gutes Mittel, um 3D-Objekte zu beschreiben. In KoKo werden diese für die meisten darzustellenden CAD-Modelle, wie z. B. Roboterachsen, Aufnahmetisch, Endeffektoren usw., verwendet.

- Static Mesh: Durch ein statisches Mesh kann ein 3D Modell wesentlich genauer und detaillierter beschrieben werden. Zudem ist es anwendbar, wenn der Einsatz einer konvexen Hülle nicht möglich ist. Ein Beispiel hierfür ist die Werkzeugform der Rumpfhalbschale. Würde man bei dieser eine konvexe Hülle einsetzen, wäre der komplette Innenraum des halben Zylinders Teil des Kollisionsobjektes.
- Dynamisches Mesh: Dieses Mesh entspricht in seiner Funktion dem *Static Mesh*, kann jedoch bewegt werden. Dynamische Meshs sind sehr rechenaufwendig und sollten nur sehr beschränkt eingesetzt werden. Besser wäre es, komplexe Objekte in viele kleine Teilobjekte aus konvexen Hüllen aufzuteilen. Der CFK-Zuschnitt den die Roboter transportieren, wird wird in der Kollisionserkennung durch ein dynamisches Mesh repräsentiert.
- Collision Group: Eine *Collision Group* definiert, welche Objekte mit anderen Objekten auf Kollisionen überprüft werden. Objekte, die sich in derselben Kollisionsgruppe befinden, können untereinander nicht kollidieren. So können z. B. alle statischen Elemente in derselben Gruppe sein, um unnötige Kollisionsberechnungen zu unterbinden.

Für die Kollisionsüberprüfung ist die Klasse *CollisionHandler* zuständig. Diese verwaltet für jedes *CollisionsObject* aus dem *RenderManager* die dazugehörigen Kollisions-Entities.

4.2.7 Kettenlinie

Die Kettenlinie, auch Catenary oder Seilkurve genannt, beschreibt die Kurve, die ein frei bewegliches Seil, das an jeweils einem Punkt am Ende befestigt ist, bzw. eine Kette unter dem Einfluss der Schwerkraft bildet. Abbildung 4.9 zeigt verschiedene Kettenlinien-Konfigurationen.

Abbildung 4.9: Beispiele für verschiedene Konfigurationen von Kettenlinien.

Ursprünglich dachte man, die Kettenlinie könne durch eine Parabel beschrieben werden. Joachim Jungius (1587-1657) wies allerdings 1639 nach, dass dies nicht der Fall ist. Gottfried Leibniz (1646-1716), Christian Huygens (1629-1695) und Johann Bernoulli (1667-1748) fanden 1690/91 heraus, dass es sich bei der mathematischen Funktion um Cosinus Hyperbolicus handelt. Korrekterweise wird die Kurve durch folgende Formel ausgedrückt [143, S. 119-124] [144, S. 12-14]:

$$y = \frac{a}{2} \cdot \left(e^{\frac{x-x_0}{a}} + e^{\frac{-(x-x_0)}{a}} \right) + y_0 = a \cdot cosh(\frac{x-x_0}{a}) + y_0 \tag{4.1}$$

Dabei bezeichnet die Variable a den Krümmungsradius bzw. die Öffnung der Kurve; sie hat somit direkten Einfluss auf den Abstand des Scheitels der Kurve von der Y-Achse. Der Ursprung der X-

Koordinate befindet sich immer auf dem tiefsten Punkt der Kettenlinie. Durch die Kettenlinie kann eine sehr gute Annäherung des Verhaltens eines Zuschnitts im Zustand des Hängens berechnet werden. Dies ist für zwei Dinge hilfreich: Zum einen kann damit sichergestellt werden, dass die Fasern beim Transport nicht beschädigt werden; zum anderen kann somit eine Annäherung des Durchhangs für die Berücksichtigung bei der Kollisionsberechnung bestimmt werden.

Um während der Bahnplanung das Verhalten des CFK-Zuschnittes zu berechnen, wird die Kettenlinie, die in Abschnitt 4.2.7 erläutert wurde, verwendet. Während der Planung muss die Kettenliniengleichung für drei unterschiedliche Fälle gelöst werden [145]:

1. Spezialfall gespannte Kettenlinie: Die Kettenlinie kann numerisch nicht gelöst werden und muss daher extra berücksichtigt werden.
2. Planung mit Constraints: Hier befinden sich die beiden Greifer immer auf der gleichen Z-Höhe. Der Winkel *thetaA* ist für beide Roboter, außer bei Aufnahme und Ablage, immer gleich.
3. Planung ohne Constraints: Die Länge des Zuschnittes sind ebenso bekannt wie der Abstand der beiden Greifer. Der Planer übergibt eine Koordinate für die beiden Greifer; daraus kann der Winkel *thetaA* berechnet werden.

Die Berechnung der unbekannten Parameter der Kettenlinie wurde von Kaspar [145] implementiert und in KoKo integriert. Als Lösung wurde für beide Fälle jeweils ein Gleichungssystem, mit und ohne Constraints, aufgestellt. Dieses wird numerisch mithilfe der Bibliothek *ALGLIB* [146] gelöst wird. Die Beschreibung der Berechnung wurde aus [145] übernommen und angepasst.

1. Spezialfall gespannte Kettenlinie

Da die Kettenlinie im Falle einer gespannten Kette numerisch nicht gelöst werden kann, wird dieser Fall abgefangen. Dazu wird die Prüfung $S - gripDistance < Tolerance$ durchgeführt, wobei $Tolerance = 5\,mm$ ist. Trifft die Abfrage zu, wird anstelle der Kettenlinie eine Geradengleichung berechnet.

$$
\begin{aligned}
\alpha &= 90 - thetaA \\
x_2 &= sin(\alpha) \cdot gripDistance \\
y_2 &= cos(\alpha) \cdot gripDistance \\
x_0 &= (x_2 + x_1)/2 \\
y_0 &= (y_2 + y_1)/2
\end{aligned}
$$

2. Planung mit Constraints

Der zweite Fall, der bei der Kettenlinienberechnung berücksichtigt wird, ist die Planung mit Constraints. Hierbei bewegen sich die Roboter nur translatorisch, die Orientierung wird nicht verändert. Da der Abstand der beiden Greifer und die Länge des Zuschnittes bekannt sind, interessiert in diesem Fall, mit welchem Winkel *thetaA* die Endeffektoren orientiert werden müssen, um den Zuschnitt zu transportieren. Um dies zu berechnen, gelten folgende Zusammenhänge:

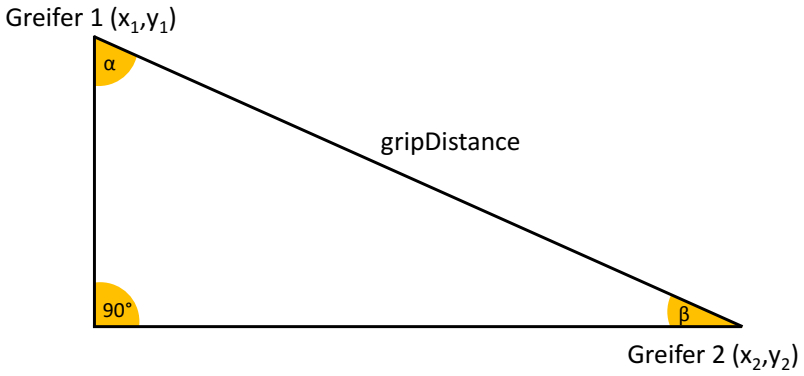


Abbildung 4.10: Berechnung des Winkels *thetaA* mittels Trigonometrie bei einer gespannten Kette.

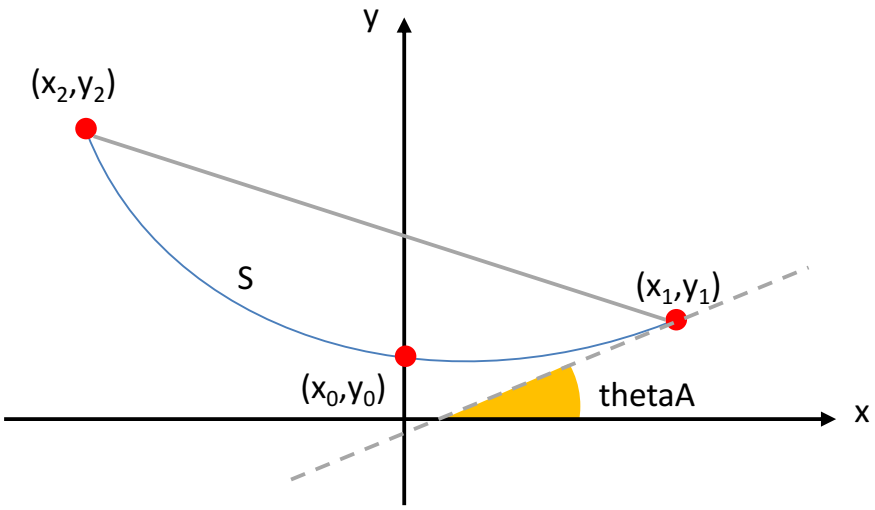


Abbildung 4.11: Kettenlinie mit (x_0, y_0) als tiefstem Punkt. Die Aufhänge- bzw. Greifpunkte sind mit (x_1, y_1) und (x_2, y_2) gekennzeichnet. S beschreibt die Länge der Kette.

$$x_2 = x_1 + gripDistance$$
$$y_2 = y_1$$

Dadurch kann ein alternatives Gleichungssystem aufgestellt und gelöst werden:

$$\begin{aligned} a \cdot cosh(\frac{x_1 - x_0}{a}) + y_0 &= y_1 \\ a \cdot cosh(\frac{x_1 + gripDistance - x_0}{a}) + y_0 &= y_1 \\ a \cdot \left(sinh(\frac{x_1 + gripDistance - x_0}{a}) - sinh(\frac{x_1 - x_0}{a}) \right) &= S \end{aligned}$$

3. Planung ohne Constraints

Die Informationen zu den Punkten und der Kettenlinie können mit der folgenden Gleichung zusammengefasst werden. Gleichung (4.3) stellt dabei die Berechnung der Länge einer Kurve dar. Details können z. B. unter [147] gefunden werden.

$$f'(x_1) = tan(thetaA) \tag{4.2}$$
$$f(x_1) = y_1$$
$$f(x_2) = y_2$$
$$\int_{x_2}^{x_1} \sqrt{1 + (f'(x))^2}\,\mathrm{d}x = S \tag{4.3}$$
$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} = gripDistance$$

Durch Ableitung der Kettenlinien-Funktion in Gleichung (4.2) kann nun folgende Gleichung aufgestellt werden:

$$\begin{aligned} \frac{d}{dx} a \cdot cosh(\frac{x - x_0}{a}) + y_0 &= \\ a \cdot sinh(\frac{x - x_0}{a}) \cdot \frac{1}{a} &= \\ sinh(\frac{x - x_0}{a}) & \end{aligned}$$

Das Integral aus Gleichung (4.3) löst sich wie folgt auf:

$$\begin{aligned} \int_{x_2}^{x_1} \sqrt{1 + (f'(x))^2}\,\mathrm{d}x &= \\ \int_{x_2}^{x_1} \sqrt{1 + (sinh(\frac{x - x_0}{a}))^2}\,\mathrm{d}x &= \\ \int_{x_2}^{x_1} cosh(\frac{x - x_0}{a})\,\mathrm{d}x &= \\ \left[a * sinh(\frac{x - x_0}{a}) \right]_{x_1}^{x_2} &= \\ a * \left(sinh(\frac{x_2 - x_0}{a}) - sinh(\frac{x_1 - x_0}{a}) \right) & \end{aligned}$$

Anschließend kann das unten stehende, nichtlineare Gleichungssystem aufgestellt werden:

$$sinh(\frac{x - x_0}{a}) = tan(thetaA)$$
$$a \cdot cosh(\frac{x_1 - x_0}{a}) + y_0 = y_1$$
$$a \cdot cosh(\frac{x_2 - x_0}{a}) + y_0 = y_2$$
$$a \cdot \left(sinh(\frac{x_2 - x_0}{a}) - sinh(\frac{x_1 - x_0}{a}) \right) = S$$
$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} = gripDistance$$

Der Rotationswinkel ($thetaB$) am Greifer des zweiten Roboters R2, kann schließlich mit folgender Formel berechnet werden:

$$tan(thetaB) = sinh(\frac{x_2 - x_0}{a})$$
$$thetaB = atan(sinh(\frac{x_2 - x_0}{a}))$$

Das Gleichungssystem kann mittels *ALGLIB* gelöst werden; da a, x_0, y_0, x_2 und y_2 bekannt sind, kann die Kettenlinie berechnet werden. Der R2-TCP ist dadurch klar definiert.

4.3 Planning

Wie Abb. 4.1 zeigt, werden die relevanten Funktionen zur Pfadplanung in KoKo unter *Planning* zusammengefasst; dessen Hauptmodul ist der *AutomaticPathPlanner*. Abbildung 4.1 gibt eine Übersicht über die dazugehörigen Klassen.

Die Hauptaufgabe der *AutomaticPathPlanner*-Klasse besteht darin, sowohl die Planungsprozesse zu koordinieren als auch die Verbindung zu den Planern herzustellen. Hierzu gibt es das *IAutomaticPathPlanner*-Interface. Sollen eigene Planer hinzugefügt werden, müssen diese die folgenden Methoden implementieren: *GetAxisFromDoubleList*, *GetListWithAllPoints*, *InitPlanner*, *PlanPath*, *SampleConstrainedState* und *SetStartAndGoal*.

In den Klassen *AutomaticPathPlannerGeneticSingleRobot* und *AutomaticPathPlanningCoopRobot* sind die evolutionären Planungsalgorithmen für eine Szene mit einem Roboter sowie mit kooperierenden Robotern implementiert. Die Klasse *AutomaticPathPlannerOmpl* stellt die Verbindung zum *OmplWrapper* her.

Abbildung 4.13 zeigt die graphische Oberfläche des *Automatic PathPlanning* in KoKo, der unter dem gleichnamigen Reiter auf der rechten Seite der GUI aufgerufen werden kann.

GUI-Einstellungen für Open Motion Planning Library (OMPL) Planungsalgorithmen

- Planner: Hier kann der gewünschte Planungsalgorithmus ausgewählt werden. Zur Auswahl stehen: RRT, RRT-connect, RRT*, PRM, STRIDE, KPIECE, LazyPRM, Automatic, PRM*, Transition-based Rapidly-exploring Random Trees (TRRT), SPARS, SPArse Roadmap Spanner Version 2.0 (SPARStwo), EST, PDST, Fast Marching Tree Algorithm (FMT), Genetic.

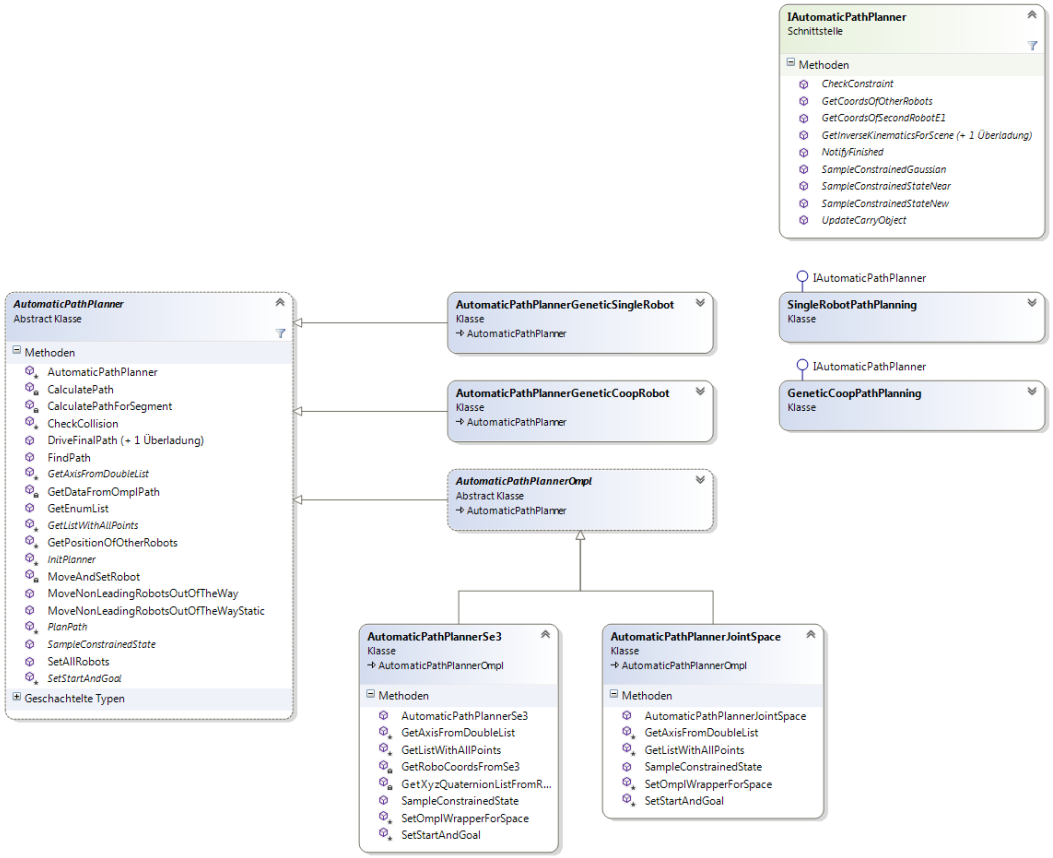


Abbildung 4.12: Klassendiagramm des *AutomaticPathPlanner* und seiner Abhängigkeiten.

- Repair-Planner: Falls mit *Experience* geplant werden soll, kann hier ein Planer zur Reparatur ausgewählt werden. Zur Auswahl stehen die gleichen Planer wie zuvor bei der Auswahl des Planungsalgorithmus.
- Collision Check Resolution: Hiermit lässt sich die Auflösung, in der geplant wird, beeinflussen. Der Wert gibt die Sampling-Dichte des Planers als Maximalmaß des Konfigurationsraumes an. Je kleiner die Zahl wird, umso genauer wird der Raum auf Kollisionen überprüft und umso länger wird die Planungszeit.
- Plan with Experience: Es erfolgt die Festlegung, ob mit Erfahrung geplant werden soll. Die Methoden Thunder und Lightning stehen zur Verfügung.
- Plan with Constraints: Die Planung erfolgt mit oder ohne geometrische Constraints.
- Plan in Joint Space: Geplant wird im Joint- oder im SE(3)-Raum.
- Optimize Path Length: Der Pfad wird nach der Pfadlänge optimiert. Diese Option hat nicht bei allen Planern eine Auswirkung. Bei optimierenden Planern wie dem RRT*, wird unabhängig davon optimiert. Der RRT kann nicht optimieren, daher ist das Setzen

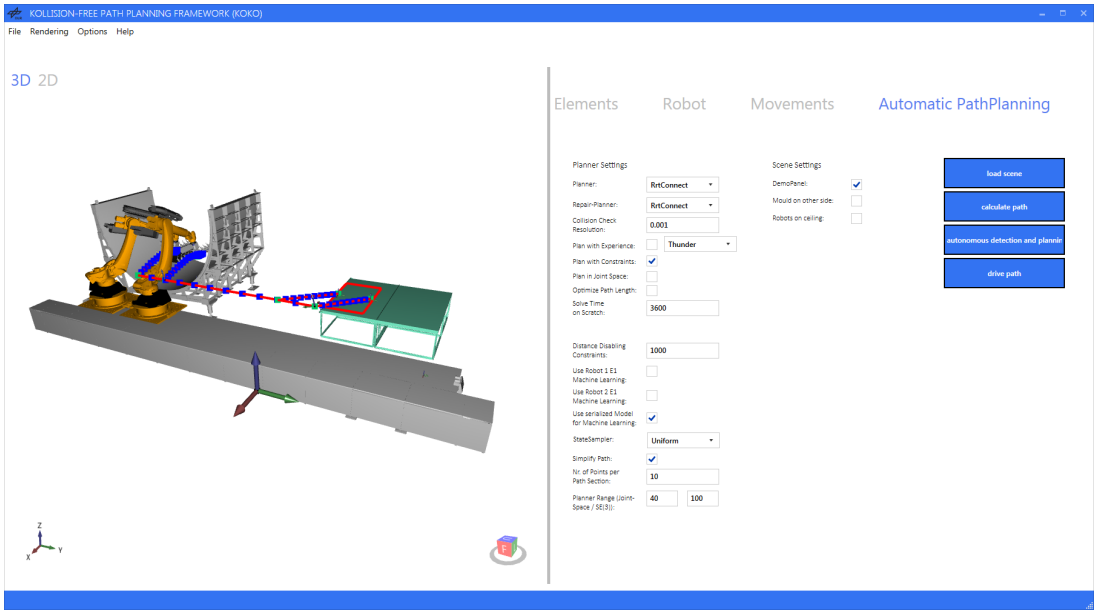


Abbildung 4.13: Die GUI-Einstellungen der verschiedenen Planungsmöglichkeiten von KoKo.

der Option in diesem Fall nutzlos. Beim PRM hingegen wird durch das Setzen die volle Planungszeit verwendet, um den Pfad zu optimieren.

- Solve time on Scratch: Hier wird die maximale Zeit, die dem Planer zur Verfügung steht, festgesetzt. Optimierende Planer nutzen diese Zeit komplett aus. Nicht optimierende Planer schließen ab, sobald ein Pfad gefunden wurde.
- Distance Disabling Constraints: Wird mit Constraints geplant, gibt dieser Wert an, ab welcher Entfernung zur Ablage die Constraints aufgehoben werden dürfen.
- Use Robot 1 E1 Machine Learning: Maschinelles Lernen wird, für das Setzen der externen Achse für Roboter R1 aktiviert.
- Use Robot 2 E1 Machine Learning: Maschinelles Lernen wird, für das Setzen der externen Achse für Roboter R2 aktiviert.
- Use serialized Model for Machine Learning: Falls bereits ein Modell für das Machine Learning hinterlegt ist, wird dieses verwendet; falls nicht, wird ein Neues erzeugt.
- StateSampler: Wird ohne Constraints geplant, kann hier der State-Sampler ausgewählt werden. Zur Auswahl stehen: Uniform, Obstacle und Gaussian.
- Simplify Path: Hiermit kann die *PathSimplification* der OMPL, die auf einen gefundenen Pfad angewendet wird, aktiviert werden.
- Nr. of Points per Path Section: Hier wird die Anzahl der Interpolationspunkte pro Pfad-Segment angegeben.
- Planner Range (Joint Space/SE(3)): Diese Einstellung beeinflusst die Laufzeit des Algorithmus. Dadurch wird festgelegt, wie lang die maximale Bewegung dauern darf, die in den Bewegungsbaum der Algorithmen eingefügt wird. Wenn der Wert zu groß ist, kann der

Pfad nicht erkannte Kollisionen enthalten. Gute Werte sind 40 für den Joint Space und 100 für SE(3).

- DemoPanel: Bei Aktivierung dieser Checkbox werden das Demo-Panel-Szenario und die entsprechenden Planer für das kooperierende Szenario geladen. Andernfalls wird das Single-Robot-Szenario gestartet.
- Mould on other side: Hiermit kann das Demo-Panel-Szenario variiert werden. Die Werkzeugform wird auf der gegenüberliegenden Seite der Linearachse positioniert.
- Robots on ceiling: Die Roboter werden im Demo-Panel-Szenario an der Decke positioniert.
- load scene (Button): Durch Betätigen dieses Button wird die Szene geladen.
- calculate path (Button): Nachdem die Szene geladen wurde, wird dieser Button aktiv und startet die Pfadplanung.
- autonomous detection and planning (Button): Das Betätigen dieses Buttons lädt die Szene und startet die Kommunikation mit dem Manufacturing Execution System (MES).
- drive path (Button): Hiermit kann der gefundene Pfad in der Simulation abgefahren werden.

4.3.1 Open Motion Planning Library

Die Open Motion Planning Library (OMPL) ist eine Open-Source-Bibliothek für Sampling-basierte Pfadplanung, die das Kavraki Lab an der Rise Universität entwickelt hat. Die Bibliothek ist in C++ programmiert und stellt zudem Python Bindings zur Verfügung. In ihr sind eine Vielzahl an Standder-Technik-Planungsmethoden integriert, wie z. B. RRT, PRM und KPIECE [148]. Alle Planer verwenden dieselben Kernfunktionalitäten, wodurch auf einfache Weise neue Planer integriert und miteinander verglichen werden können. Die Bibliothek unterteilt sich in den Core, der die Planungsmethoden zur Verfügung stellt, und die OMPL.app zur Visualisierung. OMPL hat kaum Abhängigkeiten, um kompiliert und ausgeführt zu werden. Lediglich die Boost-Bibliothek ist als Code von Drittanbietern nötig, wodurch es möglich ist, OMPL auf den meisten Betriebssystemen zu betreiben. Zur Zeit werden Linux und Mac OS X unterstützt; die Installation auf einem Windows-Betriebssystem ist ebenfalls möglich.

Die OMPL.app, die in Python und PyQt entwickelt wurde, stellt eine graphische Oberfläche für den OMPL Core zur Verfügung. In ihr ist es möglich, Planungsalgorithmen sowohl in 2D als auch in 3D für eine Vielzahl an einfachen kinodynamischen Systemen, wie bspw. einem Einrad, Flugzeug, Auto usw., zu visualisieren [149].

Der OMPL-Kern, der zur Planung eingesetzt wird, ist sehr flexibel gehalten und hat keine expliziten Funktionalitäten zur geometrischen Repräsentation von Robotern und deren Arbeitsraum. Um die Bibliothek zu verwenden, muss der Benutzer eine geometrische Repräsentation des Roboters und eine entsprechende Kollisionsprüfungsmethode zur Verfügung stellen. Abbildung 4.14 zeigt eine Übersicht über die Architektur von OMPL.

Da OMPL in C++ entwickelt wurde und KoKo in C#, wurde von Kaspar [145] ein Managed Common Language Runtime (CLR) Wrapper implementiert, der es erlaubt, nativen C++-Code in C# zu verwenden. Normalerweise ist C++ ein unmanaged Code, da es keinen Garbage Collector gibt und der Code beim Kompilieren direkt in Maschinencode übersetzt wird. Es gibt von Microsoft jedoch die C++-Erweiterung Common Language Interface (CLI), die managed Code zur Verfügung stellt. Dies ist möglich, indem der Code beim Kompilieren in eine Zwischensprache,

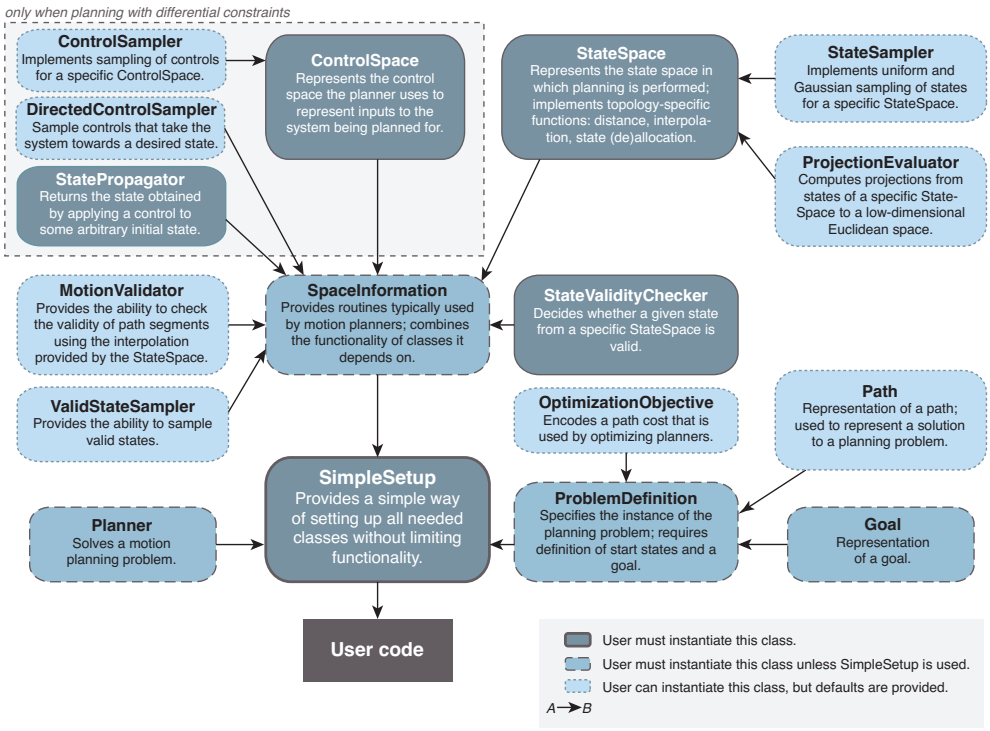


Abbildung 4.14: Ein Überblick über die OMPL API [149, S.15]. Die dunkelblauen Kästen müssen vom Benutzer instanziiert werden, um Pfadplanungsprobleme zu lösen. Die mittelblauen Kästen mit gestrichelten Linien müssen vom Benutzer instanziiert werden, wenn das *SimpleSetup* nicht verwendet wird. Die hellblauen Kästen mit gepunkteten Linien können instanziiert werden, haben allerdings eine Default-Implementierung, wenn der Benutzer diese nicht einbindet.

Common Intermediate Language (CIL), übersetzt wird. Anschließend kann der Code in der CLR ausgeführt werden.

Im Projekt *OmplPlanner* wurden die folgenden Elemente aus Abb. 4.14 umgesetzt:

- SimpleSetup: Hiermit wird die Szene in KoKo konfiguriert.
- State Space: Dieser beschreibt den Parameter-Raum, in dem der Roboter arbeitet. Es werden alle möglichen Konfigurationen des Roboters im Arbeitsraum repräsentiert. Ein einzelner Punkt im State Space ist ein *state*. In dem hier vorgestellten System wird entweder ein i-dimensionaler *RealVectorStateSpace* für den Achsraum oder ein 6-dimensionaler *SE3StateSpace* für den SE(3)-Raum verwendet.
- State Validity Checker: Dieser überprüft über einen Callback zu KoKo, ob ein Zustand kollisionsfrei ist.
- StateSampler/ValidStateSampler: Von dieser Klasse werden Punkte im SE(3)-oder im Konfigurationsraum generiert und über den *State Validity Checker* überprüft, ob der

Zustand kollisionsfrei ist. Findet die Planung unter Berücksichtigung von Constraints statt, wird die Aufgabe komplett an KoKo übergeben, da nur hier die nötigen Informationen, wie z. B. die Kettenlinie des Materials, vorliegen.

4.3.2 Redundanzauflösung der externen Achse für Sampling-basierte Verfahren

Will man für eine beliebige TCP-Position wissen, wie die Achsen des Roboters gestellt werden müssen, um diese zu erreichen, kann dies über die IK berechnet werden. Für den in dieser Arbeit verwendeten Roboter liefert die IK meistens acht Lösungen, wie Abschnitt 2.4 zeigt. Befindet sich der Roboter auf einer Linearachse, gibt es nicht acht mögliche Lösungen, sondern unendlich viele, da die Linearachse beliebig bezogen auf den TCP positioniert werden kann. Solange die Planer im Joint Space arbeiten, stellt dies kein Problem dar. Wird mit OMPL im SE(3) geplant, ist zunächst nur eine 6D-Koordinate bekannt. Um den Roboter positionieren zu können, muss für diesen noch die Position der externen Achse berechnet werden. In KoKo wurden für diesen Zweck verschiedene Verfahren integriert, die im Folgenden vorgestellt werden.

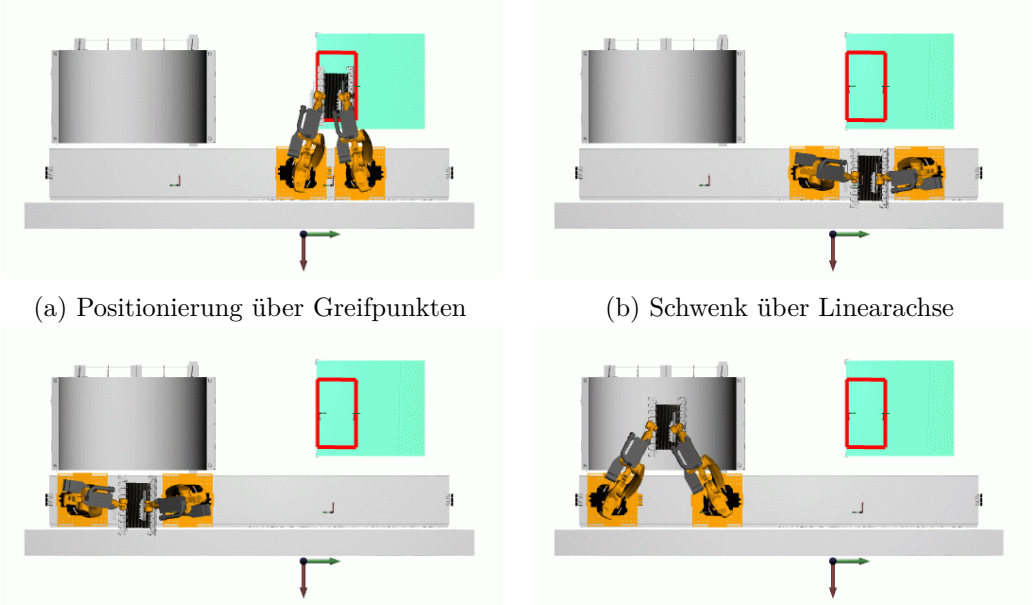
Statisches Setzen

Ein sehr simpler von [150] vorgestellter Ansatz, der allerdings dennoch gute Ergebnisse für das in Abschnitt 1.2 thematisierte Szenario mit kooperierenden Robotern darstellt, ist das statische Setzen der Position der Linearachse für die beiden Roboter. Die Idee bei diesem Verfahren ist relativ einfach. Während der Aufnahme des Zuschnitts vom Tisch können die Schlitten der Linearachse der beiden Roboter sehr nah zusammen stehen (siehe Abb. 4.15a), während sie bei der Bewegung der Greifer Richtung Linearachse auseinanderfahren müssen.

Die Positionierung funktioniert nach folgendem Prinzip: Zunächst werden die Endeffektoren der Roboter über den jeweiligen Greifpunkten positioniert. Dabei wird die externe Achse jeweils um einen festen Offset (in Abb. 4.15a um 150 mm) weg vom Zentrum des Zuschnittes gesetzt, während der TCP des Roboters unverändert bleibt. In der Draufsicht in Abb. 4.15 wird der rechte Roboter R1 demnach mit der Achse nach rechts gesetzt und der linke Roboter R2 nach links. Die externen Achsen der Roboter werden so weit auseinander gesetzt, dass im nächsten Schritt mit dem Zuschnitt über die Linearachse geschwenkt werden kann (siehe Abb. 4.15b). Mit demselben Abstand zwischen den Robotern wird der Mittelpunkt zwischen den beiden Achsen der Roboter auf den Mittelpunkt in Y-Richtung der Werkzeugform gesetzt (siehe Abb. 4.15c). Anschließend werden die externen Achsen, während die Roboter in die Werkzeugform schwenken, näher zusammen gefahren. Der Versatz bezogen auf den TCP wird dabei von 150 mm auf 75 mm verringert (siehe Abb. 4.15d). Beim statischen Setzen gibt es lediglich die folgenden beiden Zustände: Greifer über dem Tisch oder Greifer über der Linearachse. Um das Verhalten in den Zuständen dazwischen vorhersehbarer zu machen, wurde eine Heuristik implementiert, die im Folge beschrieben wird.

Setzen mittels Heuristik

Da das statische Setzen der Linearachse bisher nur zuverlässig funktioniert, wenn der Zuschnitt in der Mitte der Werkzeugform abgelegt werden soll, wurde diese Methode durch eine Heuristik erweitert. Die Grundidee dieser Methode wurde in [145] entwickelt und während der durchgeführten



(a) Positionierung über Greifpunkten

(b) Schwenk über Linearachse

(c) Mittige Positionierung vor Werkzeugform

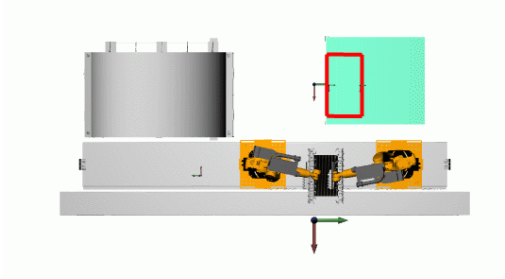
(d) Einschwenken in Werkzeugform

Abbildung 4.15: Positionen der externen Achse bei der statischen Setz-Methode. In Grün auf der rechten Seite der Aufnahmetisch und in Grau auf der linken Seite die Werkzeugform.

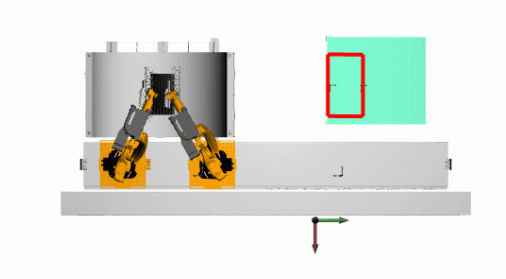
Versuche am realen Bauteil optimiert. Im Folgenden wird die Funktionsweise der Methode für den Master (R1) und Slave (R2) erläutert.

Zunächst wurde eine lineare Funktion aufgestellt; diese ermittelt die Position der Linearachse in Abhängigkeit von der X-Position des TCP. Die entsprechenden Punkte wurden durch manuelles Anfahren verschiedener Punkte auf der Roboteranlage abgeleitet.

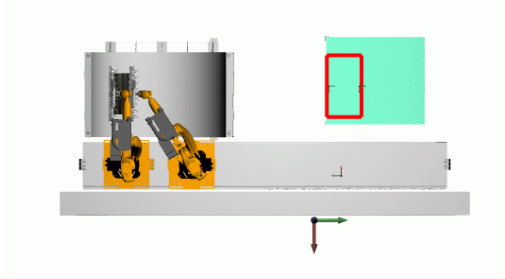
- $x_1 = -3961$ und $E_1 = 525$: Die TCPs der Roboter befinden sich in X-Richtung, in etwa mittig über der Werkzeugform bzw. des dem Tische (siehe Abb. 4.16b).
- $x_2 = -1180$ und $E_2 = 1290$: Die Roboter sind in X-Richtung mit dem TCP auf Höhe der Linearachse positioniert (siehe Abb. 4.16a).



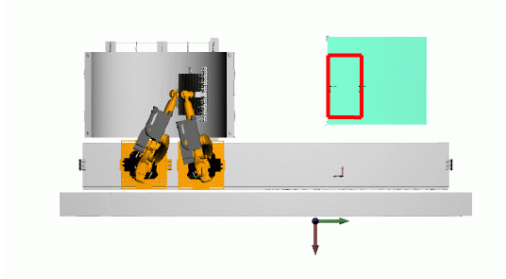
(a) Ablage im rechten Bereich der Werkzeugform.



(b) Ablage im mittleren Bereich der Werkzeugform.



(c) Ablage im linken Bereich der Werkzeugform.



(d) Ablage im rechten Bereich der Werkzeugform.

Abbildung 4.16: Positionierung der externen Achse mittels Heuristik.

Durch Einsetzen dieser Werte in eine Geradengleichung erhält man folgende Funktion:

$$f(x) = mx \tag{4.4}$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

$$b = y_2 - mx_2$$

$$m = \frac{1290 - 525}{-1180 - -3661}$$

$$b = 1290 - \frac{255 \cdot -1180}{827}$$

$$f(x) = \frac{255 \cdot x}{887} + \frac{1475730}{887} \tag{4.5}$$

(4.6)

Solange sich die Roboter bei der Ablage nicht im linken oder rechten Teil der Werkzeugform befinden, wird die Position mit der Geradengleichung ermittelt. Die aktuelle Position für die externe Achse ergibt sich durch Einsetzen der aktuellen X-Position des TCP. Für die Ablage im rechten bzw. linken Teil der Werkzeugform ergeben sich Spezialfälle, die im Folgenden beschrieben werden.

Heuristik des Masters (R1):

1. Z-Position: Die Z-Position wird für die Heuristik in diesem Szenario nicht berücksichtigt, da die Hindernisse nur in X- und Y-Richtung relevant sind.
2. Aufnahme: Greift der Roboter sehr weit in die negative X-Richtung der Roboteranlage, wird das Zentrum des Roboterfußes etwa 500 mm nach **rechts** versetzt sein, um Kollisionen mit R2 zu vermeiden.
3. Transport: Befindet sich der TCP in X-Richtung oberhalb der Linearachse, wird die externe Achse noch weiter in positiver Y-Richtung (in Abb. 4.16a nach **rechts**) bewegt werden.
4. Ablage: Wird ein Zuschnitt in der Mitte der Werkzeugform abgelegt werden, wird das Verhalten analog zu Punkt 1 sein (siehe Abb. 4.16b).
5. Ablage: Werden Zuschnitte im **rechten** Bereich der Werzeugform abgelegt werden, wird die E1 des Roboters R1 in der gleichen Y-Position wie die TCP-Position positioniert. Die E1 des **linken** Roboters R2 wird in **negativer** Y-Richtung positioniert, um Platz für R1 zu machen (siehe Abb. 4.16d).
6. Ablage: Werden Zuschnitte im **linken** Bereich der Form abgelegt werden, geschieht dies analog zu Punkt 5, allerdings mit vertauschtem Verhalten von R1 und R2 (siehe Abb. 4.16c).

Algorithm 6 Heuristik zum Setzen der externen Achse für R1.

procedure R1E1HEURISTIC(RoboCoords TCP)

$R1E1 = 255.0/887.0 \cdot TCP.X + 1475730.0/887.0$

if $TCP.Y < -2550 \wedge TCP.Y > -3950 \wedge TCP.X < -2500$ **then**
// Ablage in der rechten Seite der Werkzeugform
R1E1 −= 400
else if $TCP.Y < -5100 \wedge TCP.Y > -6500 \wedge TCP.X < -2500$ **then**
// Ablage in der linken Seite der Werkzeugform
R1E1 += 1200
else
// Aufnahme, Transport und Ablage in der Mitte der Werkzeugform.
R1E1 += 400
end if
return $R1E1$;
end procedure

Heuristik des Slave (R2):

1. Z-Position: Die Z-Position wird für die Heuristik in diesem Szenario nicht berücksichtigt, da die Hindernisse nur in X- und Y-Richtung relevant sind.
2. Aufnahme: Greift der Roboter sehr weit in die negative X-Richtung der Roboteranlage, wird das Zentrum des Roboterfußes etwa 500 mm nach **links** versetzt sein, um Kollisionen mit R2 zu vermeiden.
3. Transport; Befindet sich der TCP in X-Richtung oberhalb der Linearachse, wird die externe Achse noch weiter in positiver Y-Richtung (in Abb. 4.16a nach **links**) bewegt werden.
4. Ablage: Wird ein Zuschnitt in der Mitte der Werkzeugform abgelegt werden, soll das Verhalten analog zu Punkt 1 sein (siehe Abb. 4.16b).
5. Ablage: Werden Zuschnitte im **linken** Bereich der Werkzeugform abgelegt werden, wird die E1 des R2 in der gleichen Y-Position wie die TCP-Position positioniert. Die E1 des **rechten** Roboters R1 wird in **positiver** Y-Richtung positioniert um Platz für R2 zu machen (siehe Abb. 4.16c).
6. Ablage: Werden Zuschnitte im **rechten** Bereich der Form abgelegt werden, geschieht dies analog zu Punkt 5, allerdings mit vertauschtem Verhalten von R1 und R2 (siehe Abb. 4.16d).

Algorithm 7 Heuristik zum Setzen der externen Achse für R2.

procedure R2E1HEURISTIC(RoboCoords TCP)

$R2E1 = 255.0/887.0 \cdot TCP.X + 1475730.0/887.0$

if $TCP.Y < -2550 \wedge TCP.Y > -3950 \wedge TCP.X < -2500$ **then**
// Ablage in der rechten Seite der Werkzeugform
R2E1 –= 400
else if $TCP.Y < -5100 \wedge TCP.Y > -6500 \wedge TCP.X < -2500$ **then**
// Ablage in der linken Seite der Werkzeugform
R2E1 += 1200
else
// Aufnahme, Transport und Ablage in der Mitte der Werkzeugform.
R2E1 += 400
end if
if $R2E1 > EndanschlagR2E1$ **then**
// Falls der Wert für die externe Achse für R2 außerhalb des Endanschlags ist
$R2E1 = 320$;
end if
return $R2E1$;
end procedure

Setzen mittels Maschinellen Lernens

Bei diesem Verfahren werden die Werte durch ein maschinelles Lernverfahren, welches vorher mit validen Lösungen trainiert wurde, ermittelt. Um keinen zusätzlichen Trainingsschritt durchlaufen zu müssen, werden die Trainingsdaten für das Modell parallel zur Planung nach dem folgenden Verfahren generiert. Zu jedem erzeugten Sample (TCP-Position) werden mit einer vorher festgelegten Anzahl an *nrIterationen* und einer Schrittweite *stepSize* potentielle E1-Positionen links und rechts des TCP nach dem Prinzip in Algorithmus 8 generiert. Als Wert für *stepSize*, welches die Schrittweite der Rasterung angibt, wurde *400* gewählt. Für die Gesamtanzahl der zu testenden Positionen *nrIterations* wurde *14* festgelegt.

Algorithm 8 Eine Position des R1 sampeln (stepSize = 400, nrIterations = 14)

1: **procedure** GETR1E1FROMOFFSETINDEX(RoboCoords TCP, double stepSize, int nrIterations)
2: **for** $offsetIndex = 1$; $offsetIndex <= nrIterations$; $offsetIndex{+}{+}$ **do**
3: yPos = TCP.Y + (nrIterations/2) · stepSize – offsetIndex · stepSize;
4: **end forreturn** $GetLinearPositionForR1(yPos)$;
5: **end procedure**

Für jede der erzeugten Konfigurationen aus TCP und E1-Stellung wird nach dem Prinzip in Abb. 4.17 geprüft, ob es einen kollisionsfreien Zustand gibt, der gespeichert wird. In Abb. 4.17 gab es kollisionsfreie Zustände für 2, 3, 4, 6, 7, 8, 9, 10, 11 und 12. Die Zustände werden zu Intervallen zusammengefasst; von diesen wird das größte ausgewählt. Der Mittelpunkt dieses Intervalls wird als E1-Position festgelegt. In diesem Beispiel ist das die Index-Position 8. Dies

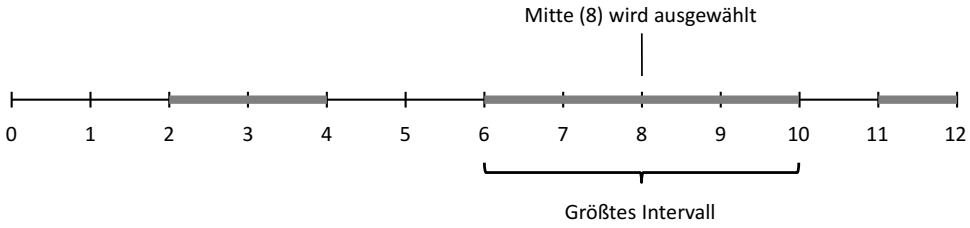


Abbildung 4.17: Schematische Darstellung der Rasterung der Linearachse. In Grau valide (kollisionsfreie) Intervalle. Als Lösung wird die Mitte des größten Intervalls zwischen sechs und zehn ausgewählt.

ergibt eine Bewegung der linearen Achse ausgehend von der TCP-Position um 2800 mm sowohl in negative als auch in positive Richtung. Die Erzeugung von Samples für den R2 läuft analog zu R1, jedoch werden nur Samples in die linke Richtung, weg von R1, erzeugt, um eine Kollision zwischen den Linearachsen zu vermeiden.

Nachdem ein Sample erzeugt wurde, wird dies in einer Datenbank gespeichert. Zu finden ist diese im KoKo-Projektordner unter `Debug/machineLearning/DataSets`. Für R1 wird ein Sample zunächst vorgeschlagen; erst wenn eine valide Postion zu R2 gefunden wurde, werden beide Samples gespeichert. Für den R1 besteht ein Sample aus der Menge *(X,Y,Z), RasterIndex*, wobei *X, Y* und *Z* die kartesische Position des TCPs und *RasterIndex* den dazugehörigen Index der Linearachse beschreiben. Ein Sample von R2, *(X,Y,Z), E1R1, RasterIndex* enthält zusätzlich noch die Position der externen Achse von R1, um diese mit zu berücksichtigen. Da je nach gewähltem Pfadplanungsalgorithmus die Erforschung der Umgebung mit verschiedenen Verfahren geschieht und unterschiedlich schnell eine Lösung gefunden wird, ist auch die Anzahl der generierten Samples für jeden Planer und bei jedem Durchlauf verschieden.

Zunächst erfolgte eine Validierung diverser Klassifikatoren in Matlab mit der Classification Learner Toolbox [151], um einen geeigneten für diese Anwendung zu finden. Hierzu wurden sowohl für R1 als auch R2 Samples erzeugt. Mit den RRT waren es für R1=14556 und für R2=14592 Samples. Mit dem RRTconnect wurden für R1=3655 Samples und für R2=3787 Samples generiert. Die gewonnen Trainingsdaten konnten nun mit einem maschinellen Lernverfahren getestet werden. Dazu wurden die generierten Samples mit einer 5-fold Cross-validation in Test- und Trainingsdaten aufgeteilt. Tabelle 4.1 zeigt eine Übersicht der Ergebnisse. Die Ergebnisse der Klassifikatoren Complex Tree, SVM, k-Nearest Neighbour (KNN) und Ensemble Tree wurden verglichen. Es hat sich gezeigt das sowohl die Bäume als auch der KNN sehr gute Ergebnisse liefern. Ebenso ist die benötigte Zeit für eine Klassifikation bei den beiden Algorithmen sehr gering, was einen Einsatz in KoKo extrem begünstigt, da sehr oft eine Position vorhergesagt werden muss. Da bezüglich der Klassifikationsrate und der Geschwindigkeit nicht wirklich ein Favorit zwischen den beiden Kandidaten bestimmt werden konnte, fiel die Wahl aufgrund einer guten Implementierung des Verfahrens in ALGLIB [146] auf die Bäume. Weitere Ergebnisse des Verfahrens beim Einsatz während der Pfadplanung werden in Abschnitt 5.2 vorgestellt und diskutiert.

[1] Maximum Number of Splits: 100, Split Criterion: Ginis Diveristy Index, Surrogate Decision Splits: off, Maximum Surrogates per Node: 10

[2] Kernel Function: Linear, Box Constraint Level: 1, Kernel Scale Mode: Auto, Multiclass Method: One-vs-One, Stardadize Data: true

[3] number of neighbour: 1, Distance Metrix: Euclidean, Distance Weight: Equal, Standarize Data: true

Klassifikator	Planer	R1	R2
Comlex Tree[1]	RRT	96,6 %	78,6 %
	RRT-connect	98,9 %	98,9 %
SVM[2]	RRT	80,6 %	66,3 %
	RRT-connect	83,6 %	98,7 %
KNN[3]	RRT	98,8 %	97,1 %
	RRT-connect	99,0 %	99,2 %
Ensemble Tree[4]	RRT	70,5 %	53,5 %
	RRTconnect	69,5 %	86,3 %

Tabelle 4.1: Klassifikationsergebnisse für die Positionsermittlung der externen Achse mit den maschinellen Lernverfahren. Die Ergebnisse wurden mit der Classification Learner Toolbox [151] von Matlab berechnet. R1 (RRT)=14556 Samples, R2 (RRT)=14592 Samples, R1 (RRT-connect)=3655 Samples, R2 (RRT-connect)=3787 Samples.

4.3.3 Setzen der Position des Slave-Roboters

Wie in Abb. 4.11 zu sehen ist, beschreibt die Kettenlinie, wie sich eine Kette bzw. ein Seil, das an zwei Punkten aufgehängt wird, verhält. Diese Beziehung wurde in KoKo als Annäherung verwendet, um das Verhalten des Zuschnitts zu berechnen, der von zwei Endeffektoren gehalten wird. Bei jedem Setzen der Position des Masters wird die Kettenlinie neu berechnet. Der Ursprung des Punkts $P(X_1, Y_1)$ liegt dabei im TCP des R1 und der Punkt $P(X_2, Y_2)$ im TCP von R2. Der Winkel α ist der Tangentenwinkel der Kettenlinie in diesen Punkten. Betrachtet man die beiden Greifer aus der X-Richtung der TCP lässt sich die Berechnung auf ein zweidimensionales Problem reduzieren, wodurch die TCP-Position des R2, wie im Folgenden beschrieben, berechnet werden kann.

Während der Aufnahme des Zuschnittes vom Tisch und des Transportes in Richtung der Werkzeugform befinden sich die TCP der beiden Roboter auf derselben Höhe. Die Berechnung des TCP von R2 erfolgt dabei nach Prinzip, welches in Abb. 4.18a zu sehen ist.

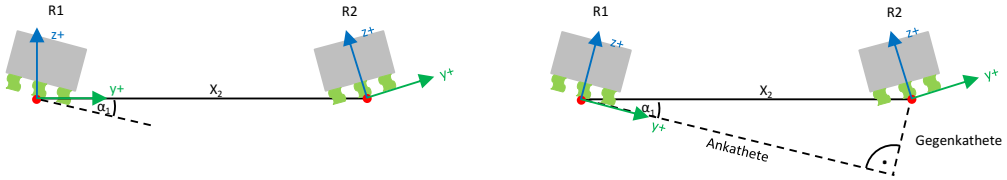
- Ermittlung der TCP-Position von R2: Verschiebe ausgehend vom TCP des R1 in der Y-Richtung des Toolkoordinatensystems von R1 um den Wert X_2 aus der Kettenliniengleichung; hierdurch erhält man die TCP-Position von R2.
- Herstellen der richtigen Orientierung: Drehe zweimal um den Winkel α um die X-Achse des Toolkoordinatensystems in positiver Richtung.

Sobald sich die Roboter in die Ablageposition bewegen und die beiden Greifer nicht mehr auf gleicher Höhe sind, wird die Berechnung nach dem Verfahren in Abb. 4.18b durchgeführt. Hierbei wird zunächst die An- und Gegenkathete mit den folgenden zwei trigonometrischen Funktionen berechnet. Danach werden die Punkte 1–3 ausgeführt um die TCP Position des R2 zu berechnen.

[4]Ensemble Method: AdaBoost, Learner Type: Decision Tree, Number of Learners: 200, Learning Rate: 0,1, Subspace Dimension: 1

$$Ankathete = \cos(\alpha_{radians}) \cdot X_2$$
$$Gegenkathete = \sin(\alpha_{radians}) \cdot X_2$$

1. Verschiebung um Ankathete: Verschiebe ausgehend von R1 in positiver Y-Richtung im Toolkoordinatensystem um die Länge der Ankathete.
2. Verschiebung um Gegenkathete: Verschiebe ausgehend von R1 in positiver Z-Richtung im Toolkoordinatensystem um die Länge der Gegenkathete.
3. Rotation um Winkel der Kettenlinie: Rotiere zweimal um den Winkel α in positiver Richtung.



(a) Aufnahme und Transport (b) Ablage

Abbildung 4.18: Berechnung der TCP-Position von R2 ausgehend von der Position des TCP von R1.

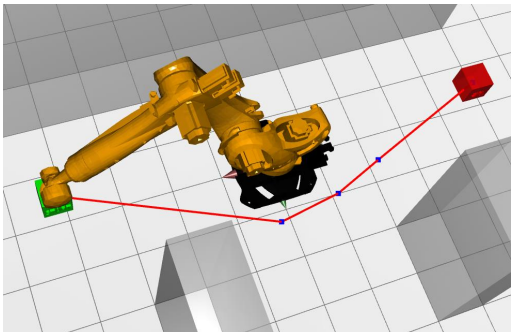
Nachdem die TCP-Position des zweiten Roboters durch die Beziehung der Kettenlinie berechnet wurde, kann die Position der Linearachse für die Pfadplanung mit OMPL entweder durch die Heuristik oder durch das maschinelle Lernverfahren bestimmt werden. Ist die E1-Position bekannt, kann im nächsten Schritt über die IK die Stellung der Achsen des Roboters ermittelt werden.

4.3.4 Evolutionäre Algorithmen

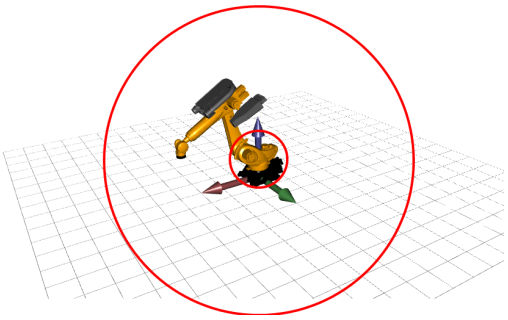
Single-Robot

Die generelle Funktionsweise von EA wurde bereits in Abschnitt 3.5 erklärt. Im Folgenden wird die Integration und Implementierung in KoKo beschrieben. Für diese wurden die Genetic-Funktionen aus dem AForge [152] Framework verwendet; dieses sellt Klassen, die eine unkomplizierte Implementierung eines EA erlauben, zur Verfügung. Zu den verwendeten Klassen zählen:

- ChromosomeBase: Diese Klasse stellt die Basis-Implementierung der IChromosome-Methode und Eigenschaften zur Verfügung, die für alle Arten von Chromosomen gleich sind. Das in KoKo implementierte Chromosom erbt von dieser Klasse.
- IFitnessFunction: Um die Funktion eines Chromosoms zu bestimmen, sollte dieses Interface entsprechend implementiert werden.
- Population: Diese Klasse repräsentiert die Population. Sie arbeitet mit jeder Art von Chromosomen, die mit dem IChromosome-Interface implementiert wurden. Des Weiteren kann jede beliebige Fitness-Funktion, die von der IFitnessFunction abgeleitet wurde, verwendet werden.



(a) In Grün der Start- und in Rot der Zielpunkt. Dazwischen in Blau die ViaPunkte.



(b) Arbeitsraum eines Roboters im Bereich zwischen dem äußeren und dem inneren Kreis.

Abbildung 4.19: Repräsentation eines Chromosoms.

Die Implementierung des genetischen Planers für einen Roboter wurde im *Planning*-Teil von KoKo, in der in Abb. 4.12 beschriebenen Klasse *AutomaticPathPlannerGeneticSingleRobot* realisiert. Die Grundkonzepte des EA wurden folgendermaßen umgesetzt:

- Chromosomen: Jedes Chromosom besteht aus einer Liste von Punkten mit der folgenden Struktur $StartPunkt$, $ViaPunkt_1$, $ViaPunkt_2$, .., $ViaPunkt_n$, $ZielPunkt$ (siehe Abb. 4.19a). Da die Operationen des EA anhand reeller Werte wesentlich leichter abgelesen werden kann, wurde keine Bit-Darstellung gewählt. Die einzelnen Punkte werden als 6D-Koordinate X, Y, Z, A, B und C gespeichert.
- Initialisierung: Bei der Initialisierung eines Chromosoms werden zunächst nach dem Zufallsprinzip Punkte im Arbeitsraum des Roboters generiert. Nachdem ein Punkt erzeugt wurde, wird geprüft, ob dieser vom Roboter erreicht werden kann, indem die IK für den Pfadpunkt berechnet wird. Liefert die IK keine Lösung, wird so lange ein neuer Punkt erzeugt, bis eine valide Lösung gefunden wurde. In Abb. 4.19b ist der Arbeitsbereich schematisch zwischen dem äußeren und dem inneren Kreis dargestellt.
- Selektion: Bei der Selektion werden Individuen für die Reproduktion ausgewählt. AForge stellt die Elite- und Rank-Selektion zur Verfügung. Zusätzlich wurden die Tournament- und Roulette-Wheel-Selektion entwickelt.
- Rekombination: Für die Rekombination wurde ein One-Point Crossover, wie in Abschnitt 3.5 beschrieben, implementiert. Zunächst wird per Zufall ein Seedpoint bestimmt, an dem die Informationen der Pfadpunkte von zwei Chromosomen nach dem Prinzip in Abb. 3.9 ausgetauscht werden.
- Mutation: Für die Mutation wird nach dem Zufallsprinzip ein Punkt aus der Liste der $ViaPoints$ ausgewählt; dessen X-, Y- und Z-Koordinaten werden in einem maximalen Bereich von 500 mm mutiert. Analog zur Erzeugung eines Chromosoms wird die Mutation so lange wiederholt, bis die IK eine Lösung liefert.
- Evaluierung: Algorithmus 9 zeigt den Pseudocode zur Berechnung der Gesamtfitness, die sich aus der Kollisions-, Pfadlängen- und Achsbewegungsfitness zusammensetzt. Der Wert der Kollisionsfreiheit wird in zwei Schritten berechnet. Zunächst wird für jedes Segment des Pfades geprüft, ob dieses eine Rayintersektion mit einem Hindernis bzw. mit

dem Roboter hat. Die Prüfung dieser Überschneidungen benötigt sehr wenig Rechenzeit und kann rechenintensive Überprüfungen von Kollisionen der konvexen Hüllen einsparen. Treten mehr als 25 dieser Überschneidungen auf, ist die Kollisionsüberprüfung direkt beendet und die Kollisionsfitness bekommt einen sehr schlechten Wert $0,001$ (Z.15). Treten maximal 25 Überschneidungen auf, wird der Ray-Kollisionswert normiert und von $1,0$ subtrahiert. Dies hat zur Folge, dass viele Überschneidungen eine sehr kleine und damit schlechte Kollisionsfitness ergeben. Erst wenn gar keine Ray-Kollisionen mehr auftreten, wird die rechenintensive Überprüfung der konvexen Hüllen durch die BEPU Physik-Engine durchgeführt (Z.23). Werden mehr als 40 Kollisionen von BEPU gezählt, bekommt die Kollisionsfitness ebenfalls einen schlechten Wert (Z.26). Werden weniger gezählt, wird der Wert normiert und von $1,0$ subtrahiert. Um die Kollisionen der konvexen Hüllen zu überprüfen, werden die Pfadsegmente mit einem Abstand von 200 mm interpoliert. Danach wird der TCP des Roboters an jeden dieser Punkte gesetzt und über die Physik-Engine wird geprüft, ob eine Kollision mit dem Roboter und den Hindernissen auftritt. Die Anzahl der Kollisionen summiert sich für den Pfad. Um die Berechnung zu beschleunigen, werden bereits geprüfte Punkte in einer Datenstruktur, die eine erneute Abfrage wesentlich performanter zulässt, gespeichert. Da die Chromosomen nur wenig durch Mutation geändert werden, bringt das Speichern eine enorme Zeitersparnis. Dennoch ist die Erkennung der Kollisionen für die Fitness der rechenintensivste Teil des EA. Vor allem bei Chromosomen mit vielen Pfadpunkten oder sehr großen Populationen kann die Berechnung viel Zeit beanspruchen.

In jedem Chromosom sind neben den Start- und Zielpunkten auch die Pfadpunkte in einer Liste gespeichert. Zur Berechnung der Pfadlänge wird zunächst mit der Funktion *CalculatePathLength* die euklidische Distanz zwischen diesen Punkten berechnet (Z.33). Die direkte Pfadlänge zwischen Start- und Zielpunkt beträgt 4185 mm. In vorhergehenden Versuchen mit dem EA konnte festgestellt werden, dass selbst bei einer sehr schlechten Initialisierung eines Chromosoms der Pfad selten länger als die fünffache Länge wird; deshalb wurde die Pfadfitness darauf normalisiert (Z.36). Ist der Pfad dennoch länger, wird eine sehr schlechte Pfadfitness vergeben (Z.40). Falls der Pfad nicht dir fünfache Länge überschreitet, wird der errechnete Pfadfitnesswert von 2,0 subtrahiert, was zur Folge hat, dass bei einem langen Pfad ein sehr kleiner und somit schlechter Pfadfitness-Wert das Ergebnis ist.

$$Pfadlänge = \sum_{i=1}^{\#Punkte} EukilidischeDistanz(PointList[i], PointList[i-1])$$

Neben einer Liste an Punkten ist in jedem Chromosom außerdem noch die Stellung der Achsen des Roboters an jedem Punkt des Pfades abgespeichert. Für die Achsbewegung wird die Bewegung der einzelnen Achsen zwischen den aufeinanderfolgenden Punkten nach der folgenden Formel summiert. Der resultierende Wert wird mit 3000 normalisiert und von 1,0 subtrahiert. Dies hat wie bei den vorherigen Komponenten der Fitness zur Folge, dass ein hoher Achsbewegungswert eine kleine Achsfitness als Ergebnis hat.

$$Achsbewegung = \sum_{i=1}^{\#Punkte} \sum_{j=1}^{6} AxisList[i]_{A_j} + AxisList[i-1]_{A_j}$$

Im letzten Schritt (Z.50) werden die einzelnen Werte zu einer Gesamtfitness summiert. Dabei erfolgt eine Addition der Faktoren mit einem Gewichtungsfaktor von jeweils 0,2 für Pfadlänge und Achsbewegung sowie 0,6 für die Kollision addiert. Je größer der Gesamtfitnesswert ist, umso besser ist ein Chromosom.

- Abbruchkriterium: Der Algorithmus bricht ab, sobald ein kollisionsfreier Pfad gefunden oder eine maximale Zeit erreicht wurde.

Algorithm 9 Pseudocode zur Berechnung der Fitness.

```
1:  procedure CalculateFitness(IChromosome chromosome)
2:
3:      convexCollisions = 0;
4:      rayCollisions = 0;
5:      collisonFitness = 0.001;
6:      pathFitness = 0.001;
7:      axisFitness = 0.001;
8:      fitness = 0.0;
9:
10:     // Berechne die Anzahl der Ray Kollisionen
11:     rayCollisions = CalculateRayCollisions(chromosome)
12:
13:     if rayCollisions > 25 then
14:         // Bei mehr als 25 schlechte Kollisionsfitness
15:         collisionFitness = 0.001;
16:     else if rayCollisions > 0 ∧ rayCollisions ≤ 25 then
17:         // Bei mehr als null und maximal 25 wird nur die Ray Berechnung
18:         // für den Kollisionsfitness Wert genommen
19:         collisionFitness = (1.0 − (rayCollisions/25.0)) · 0.01;
20:     else
21:         // Wenn es gar keine Ray Kollisionen gibt werden die Convex Hull
22:         Kollisionen berechnet
23:         convexCollisions = CalculateConvexCollisions(chromsome);
24:         if convexCollisions ≥ 40 then
25:             // Bei mehr als 40 schlechte Kollisionsfitness
26:             CollisionFitness = 0.001;
27:         else
28:             CollisionFitness = 1.0 − (convexCollisions/40.0);
29:         end if
30:     end if
31:
32:     // Berechne die euklidische Pfadlänge
33:     pathLength = CalculatePathLength(chromosome);
34:     // 4184 ist die direkte Pfadlänge zwischen Start und Ziel
35:     // Normiere auf maximal fünffache Länge
36:     pathFitness = pathFitness/(4184 · 5);
37:     // Umso länger der Pfad umso größer ist zu diesem Zeitpunkt pathFitness
38:     // Wenn diese größer als zwei ist, ist die pfadFitness sehr schlecht
39:     if pathFitness ≥ 2.0 then
40:         pathFitness = 0.001;
```

```
41:     else
42:         pathFitness = 2.0 − pathFitness;
43:     end if
44:
45:     axisFitness = CalculateAxisMovement(chromosome);
46:     // Normiere den Achsbewegungswert.
47:     axisFitness = 1.0 − axisFitness/3000.0;
48:
49:     // Berechne Gesamtfitness.
50:     fitness = 0.2 · pathFitness + 0.2 · axisFitness + 0.6 · collisionFitness;
        return fitness
51: end procedure
```

Kooperierende Roboter

Die Implementierung der EA Pfadplanung für kooperierende Roboter unterscheidet sich von der Planung im Single-Robot-Szenario nur an einzelnen Stellen. Im Folgenden sind daher lediglich die Unterschiede aufgelistet:

- Chromosom-Repräsentation: Jedes Chromosom besteht in diesem Fall, anders als beim Single-Robot-Szenario, aus zwei Listen von Pfadpunkten mit der folgenden Struktur $StartPunkt$, $ViaPunkt_1$, $ViaPunkt_2$, .., $ViaPunkt_n$, $ZielPunkt$ (siehe Abb. 4.19a). Die einzelnen Punkte werden als 6D-Koordinate X, Y, Z, Y, B und C mit dem dazugehörigen Wert $E1$ der externen Achse gespeichert.
- Initialisierung: Algorithmus 10 zeigt den Pseudocode zur Erstellung von Konfigurationen der einzelnen Punkte eines Chromosoms. Zunächst wird zufällig ein Punkt im Arbeitsraum, der in Abb. 4.20 durch einen blauen Rahmen gekennzeichnet ist, für R1 generiert. Anschließend wird die externe Achse des R1 auf dieselbe Höhe der Y-Position des TCP gesetzt (siehe Abb. 4.20a). In einem nächsten Schritt wird die Linearachse des R1 zufällig im Bereich [–200, 1500] nach links oder rechts bewegt, bis eine Position innerhalb der Achsanschläge des R1 gefunden wurde (Abb. 4.20b). Über die IK wird danach geprüft, ob eine valide Konfiguration für den R1 generiert wurde. War die Prüfung erfolgreich, wird ausgehend von der R1-Position anhand, der Kettenlinie die TCP-Postion für R2 bestimmt. Nachdem diese berechnet wurde, wird für R2 ebenfalls eine E1-Konfiguration nach demselben Verfahren wie für R1 generiert (Abb. 4.20c) und über die IK bestimmt, ob es sich um eine valide Roboterkonfiguration handelt. In manchen Fällen kann es vorkommen, dass für R1 eine Konfiguration bestimmt wurde, die es unmöglich macht, für R2 in Abhängigkeit davon ebenfalls eine valide zu finden. Dies kann z. B. der Fall sein, wenn der generierte TCP von R1 unmittelbar in der Nähe des Endanschlages der externen Achse von R2 liegt. Daher wird, wenn nach 30 Versuchen keine Achskonfiguration für R2 gefunden wurde, für R1 ebenfalls eine neue Achskonfiguration erzeugt.
- Mutation: Für die Mutation wird zufällig ein Punkt aus der Liste der Pfadpunkte von R1 eines Chromosoms ausgewählt. Dieser Punkt wird in einem maximalen Bereich von 500 mm verändert. Danach wird für R1 analog zu dem Verfahren der Initialisierung zunächst ein E1-Wert für R1 bestimmt und im zweiten Schritt über die Gleichung der Kettenlinie zunächst die TCP-Position für R2 in einem zweiten Schritt die E1-Stellung.
- Evaluierung: Die Berechnung der Fitness unterscheidet sich kaum von der im Single-Robot-Szenario. Die Berechnung des Kollisionswertes erfolgt nach demselben Prinzip wie in

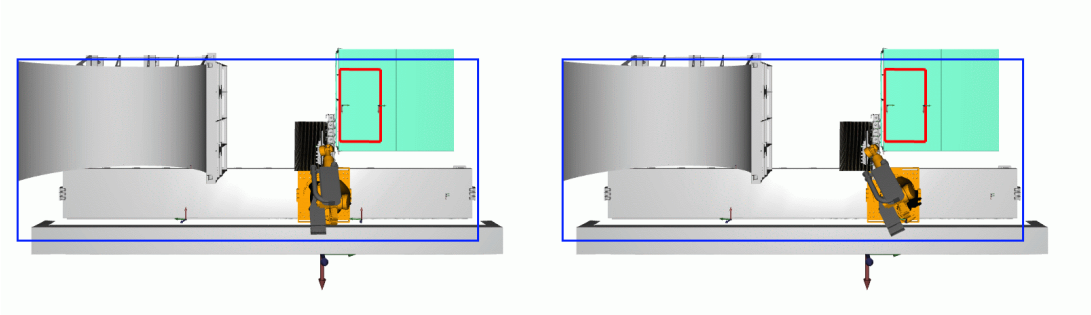
Algorithmus 9 beschrieben. Lediglich in der Funktion *CalculateConvexCollision()* in Zeile 19 werden beide Roboter an die in dem Chromosom gespeicherte Position gesetzt. Für die Berechnung der Pfadlängenfitness wird der Pfad des R1 herangezogen, da sich die Länge der beiden Roboter nicht unterscheidet. Die Achsbewegung beinhaltet zudem noch die E1-Komponente und es werden die Bewegungen beider Roboter berücksichtigt.

$$Achsbewegung_{R1} = \sum_{i=1}^{\#Punkte} \sum_{j=1}^{6} AxisList_{R1}[i]_{A_j} + AxisList_{R1}[i-1]_{A_j} + E1[i]$$

$$Achsbewegung_{R2} = \sum_{i=1}^{\#Punkte} \sum_{j=1}^{6} AxisList_{R2}[i]_{A_j} + AxisList_{R2}[i-1]_{A_j} + E1[i]$$

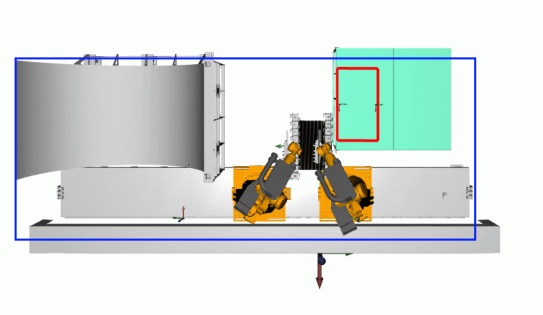
$$Achsbewegung_{gesamt} = Achsbewegung_{R1} + Achsbewegung_{R2}$$

- Abbruchkriterium: Der Algorithmus bricht ab, sobald ein kollisionsfreier Pfad gefunden oder eine maximale Zeit erreicht wurde.



(a) Generierung einer zufälligen TCP-Position für R1.

(b) Bestimmung einer validen E1-Position für R1.



(c) Bestimmung einer Konfiguration für R2.

Abbildung 4.20: Generierung von Chromosomen-Konfigurationen für kooperierende Roboter. Der blaue Rahmen zeigt den möglichen Arbeitsraum, in dem zufällig TCP-Positionen generiert werden.

Algorithm 10 Pseudocode zur Erzeugung einer Konfiguration eines Chromosoms für kooperierende Roboter.

```
1: procedure PathPlanningChromosome(RandomSource rnd)
2:
3:     RoboAxis RA1, RA2
4:     RoboCoords RC1, RC2
5:     float actE1, E1
6:     int trialCounter
7:
8:     for i ← 1, AnzahlViaPunkte do
9:         // Wiederhole bis eine valide Achskonfiguration für R1 und R2
10:        // gefunden wurde
11:        while RA1 == null ∨ RA2 == null do
12:
13:            // Konfiguration für R1 finden
14:            while RA1 == null do
15:                RC1 = GenerateRandomPosInWorkspace()
16:                // berechne E1 Position mit gleicher Höhe wie aktuelle
17:                // Y-Position des TCP
18:                actE1 = GetLinearPositionForR1(rc1.Y);
19:                // finde eine valide E1 Position für R1 für die aktuelle TCP-Position
20:                while true do
21:                    E1 = rnd.NextDouble(−200, 1500) + actE1
22:                    // 330 und –7000 beschreiben Achsanschläge der E1 Achse
23:                    // wenn nach mehr als 30 Versuchen keine Konfiguration gefunden
24:                    // wurde, wird eine neue TCP-Position generiert
25:                    if E1 < 330 ∧ E1 > −7000 ∨ trialCounter > 30 then
26:                        break;
27:                    end if
28:                    trialCounter + +;
29:                end while
30:                RA01 = GetInverseKinematic(RC1, E1);
31:            end while
32:
33:            // Konfiguration für R2 finden
34:            while RA2 == null do
35:                RobotCoord rc2 = GetCoordForSecondRobot(rc1, E1);
36:                // finde eine valide E1 Position für R2 für die aktuelle TCP-Position
37:                while true do
38:                    E1 = rnd.NextDouble(−200, 2000) + actE1
39:                    // 330 und –7000 beschreiben Achsanschläge der E1 Achse
40:                    // wenn nach mehr als 30 Versuchen keine Konfiguration gefunden
41:                    // wurde, wird eine neue TCP-Position generiert
42:                    if E1 < 330 ∧ E1 > −8000 ∨ trialCounter > 30 then
43:                        break;
44:                    end if
45:                end while
46:            end while
47:
48:            // Wurde für R2 keine passende Konfiguration gefunden,
49:            // wird auch für R1 eine neue generiert.
```

```
50:             if RA2 == null then
51:                 RA1 = null;
52:             end if
53:         end while
54:         // Speichere die Punkte im Chromosom
55:         AxisListR1.Add(RA01);
56:         AxisListR2.Add(RA02);
57:     end for
58: end procedure
```

4.4 Execution

Um die geplanten Pfade auf einer realen Roboter-Hardware ausführen zu können, verfügt KoKo zusätzlich über ein *Execution*-Modul. Dieses bietet zwei unterschiedliche Technologien, um Fahrbefehle mit den Robotern austauschen zu können. Der Ablauf der beiden Varianten ist in Abb. 4.21 dargestellt. Die einzelnen Komponenten in den beiden Bildern haben folgende Funktionen:

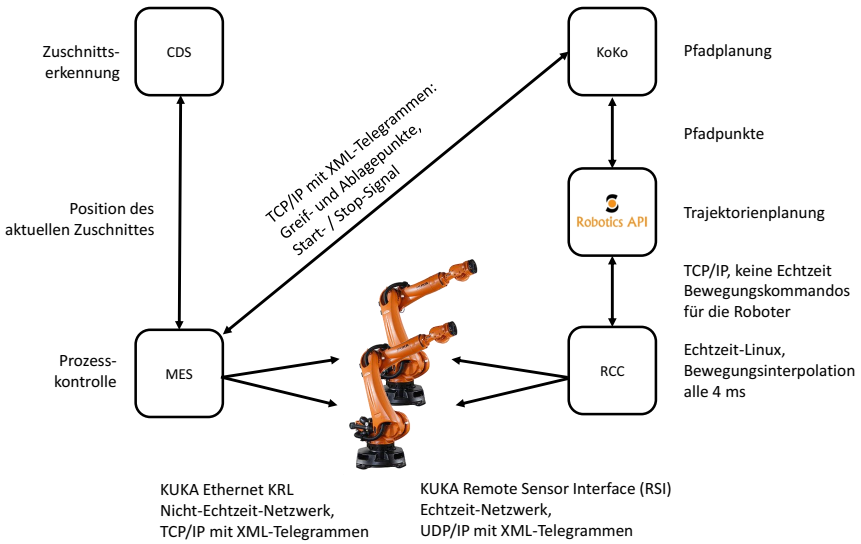
- Cutpiece Detection System (CDS): Am Greifer des Roboters R1 befindet sich eine Kamera; mit dieser wird die Position des nächsten Zuschnittes auf dem Aufnahmetisch detektiert wird.
- Manufacturing Execution System (MES): Im Deutschen oft Prozessleitsystem genannt. Dies ist eine Ebene eines Fertigungsmanagementsystems, die sehr prozessnah arbeitet und für die komplette Ablaufsteuerung zuständig ist. Es sendet Kommandos an die Untersysteme und verarbeitet deren Ergebnisse. Das hier verwendete MES kommuniziert zudem über Ethernet KRL mit den Robotern und kann Fahrbefehle ausführen. Entwickelt wurde es von Schuster [49] entwickelt. Ethernet KRL ist ein zusätzliches Softwarepaket von KUKA, welches es ermöglicht, XML-Diagramme über eine TCP/Internetprotokoll (IP)-Verbindung mit dem Roboter auszutauschen.
- KoKo: Dieses System plant einen kollisionsfreien Pfad. Dazu bekommt es vom MES die Information zu Greif- und Ablagepunkten und liefert als Ergebnis eine Liste von Pfadpunkten zurück; die werden entweder an das MES oder an die RoboticsAPI gesendet.
- RoboticsAPI: Mit dieser lassen sich komplexe, echtzeitkritische Aufgaben von Robotern in Java programmieren. In KoKo kommt ein C# Wrapper zum Einsatz. Ein komfortables und flexibles Programmiermodell ermöglicht eine einfache und schnelle Entwicklung anspruchsvoller Anwendungen. Eine innovative Software-Architektur gewährleistet eine zuverlässige Ausführung von Roboteranwendungen und garantiert das exakte Timing kritischer Operationen.
- Robot Control Core (RCC): Dieser führt die von der RoboticsAPI generierten Befehle mit einer Frequenz von bis zu 1 kHz aus. In jedem Takt werden die entsprechenden Achswinkel für jeden Roboter berechnet und, wenn nötig, die Schaltsignale für das Werkzeug verarbeitet. Der RCC steuert die KUKA-Roboter über die RSI-Echtzeitschnittstelle, die es erlaubt, diese fernzusteuern. Die Kommunikation zwischen dem RCC und der KRC4-Steuerung des Roboters erfolgt über XML-Diagramme, die in ein User Datagram Protocol (UDP)/IP eingebettet sind. Die KUKA-Steuerung sendet alle 4 ms ein Diagramm, das die aktuelle Position und ihre Achswerte enthält; darauf wird eine Antwort innerhalb von weiteren 4 ms, die die Stellung für die Achsen und gegebenenfalls weitere Signale liefert, erwartet.

Erfolgt dies nicht in dieser Zeit, löst die Steuerung einen Notstop aus. Ebenso muss der RCC sicherstellen, dass die gesendeten Punkte innerhalb eines Zyklus erreichbar sind; dies setzt voraus, dass die richtigen Beschleunigungen und Ruck-Werte berechnet werden. Für weitere Details des RCC sei auf Vistein [153] verwiesen.

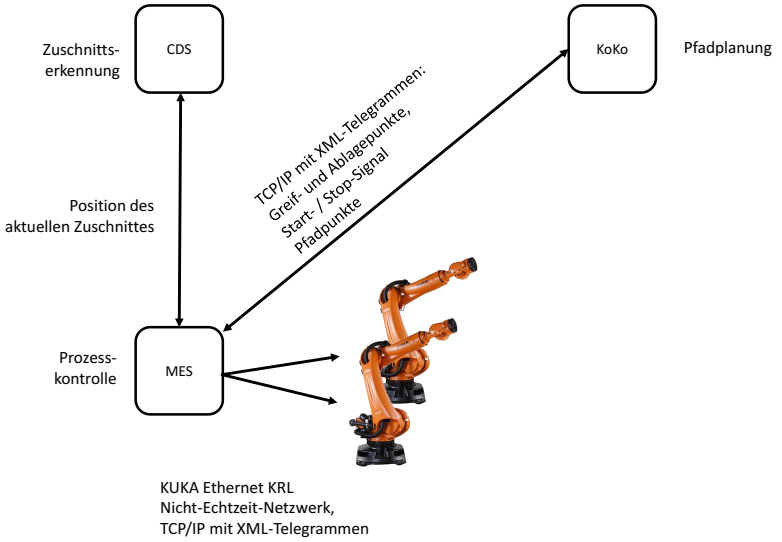
Die beiden Ausführungssysteme Abb. 4.21a und Abb. 4.21b unterscheiden sich lediglich bei der Ansteuerung der Roboter. In Abb. 4.21a werden direkt aus KoKo über den Wrapper der RoboticsAPI Fahrbefehle an den RCC gesendet. Nachdem der Pfad erfolgreich abgefahren wurde, liefert KoKo einen Rückgabewert an das MES, welches den nächsten Prozessschritt startet. Bei Abb. 4.21b liefert KoKo nach erfolgreicher Berechnung die Pfadpunkte an das MES, das die Roboter direkt per EthernetKRL ansteuert. Der Vorteil bei dieser Variante ist, dass die Kontrolle aller am Prozess beteiligten Komponenten zu jeder Zeit beim MES liegt und somit ein Fehlerhandling für unvorhergesehene Ereignisse einfach umgesetzt werden kann.

Abbildung 4.22 zeigt alle Schritte, die das System aus Abb. 4.21b von der Aufnahme bis zur Ablage eines Zuschnittes durchläuft. Der Prozess startet im orangefarbenen Rechteck oben links. Zunächst wird der Roboter R1, an dem die Kamera zur Erkennung der Zuschnitte befestigt ist, in die *Pickpos* gefahren. Aus dieser wird in eine fest einprogrammierte Position mittig über dem Tisch, die *Detektion 1*, angestoßen, um eine ungefähre Position des Zuschnittes zu ermitteln. Danach wird der Roboter mittig über den Zuschnitt positioniert, um die *Detektion 2*, mit einer höheren Genauigkeit, auszuführen.Im Anschluss daran werden R1 und R2 in eine Vorposition (*Prepos_grip*) knapp oberhalb der Greifpunkte des Zuschnitt gefahren. Danach werden die Greifpunkte (*Grippoint*) mit einer kleinen Überdrückung der Saugmodule angefahren und die Ejektoren aktiviert. In einem nächsten Schritt werden die für die Pfadplanung relevanten Informationen mit einer zyklischen TCP/IP-Kommunikation zwischen dem MES und KoKo ausgetauscht.

Das MES sendet die Informationen nach einer vordefinierten Reihenfolge; KoKo antwortet jeweils mit einer Empfangsbestätigung. Zunächst wird eine Testnachricht (*KOKO_TEST*) ausgetauscht, um die Kommunikation zu prüfen, danach wird die Anzahl der Roboter (*SET_NUMBER_OF_ROBOT*), für die ein Pfad geplant werden soll, definiert. Darauf folgend sendet das MES für jeden Roboter die Startposition der externen Achse (*SET_TRANSFER_GRIPPPOINT_EXTENDED*), den Greifpunkt (*SET_TRANSFER_GRIPPOINT*), den Ablagepunkt (*SET_TRANSFER_DROPPOINT*) und den Werkzeug-TCP (*SET_TRANSFER_TCP*). Nachdem sämtliche Informationen ausgetauscht sind, startet das MES die Pfadplanung gestartet (*RENDER_TRANSFER*). Nach erfolgreicher Planung liefert KoKo die Anzahl der Pfadpunkte, die zyklisch für die TCP-Koordinate (*QUERY_TRANSFER_POINT*) und die dazugehörige externe Achsenposition (*QUERY_TRANSFER_EXT*) vom MES abgefragt werden, zurück.



(a) Ablauf der Kommunikation bei der Ausführung der Pfade durch KoKo mit der RoboticsAPI und dem RCC.



(b) Ablauf der Kommunikation bei der Ausführung der Pfade durch das MES mit KUKA EthernetKRL.

Abbildung 4.21: Ablauf der Kommunikation zwischen KoKo, CDS, MES, Robotics-API und RCC.

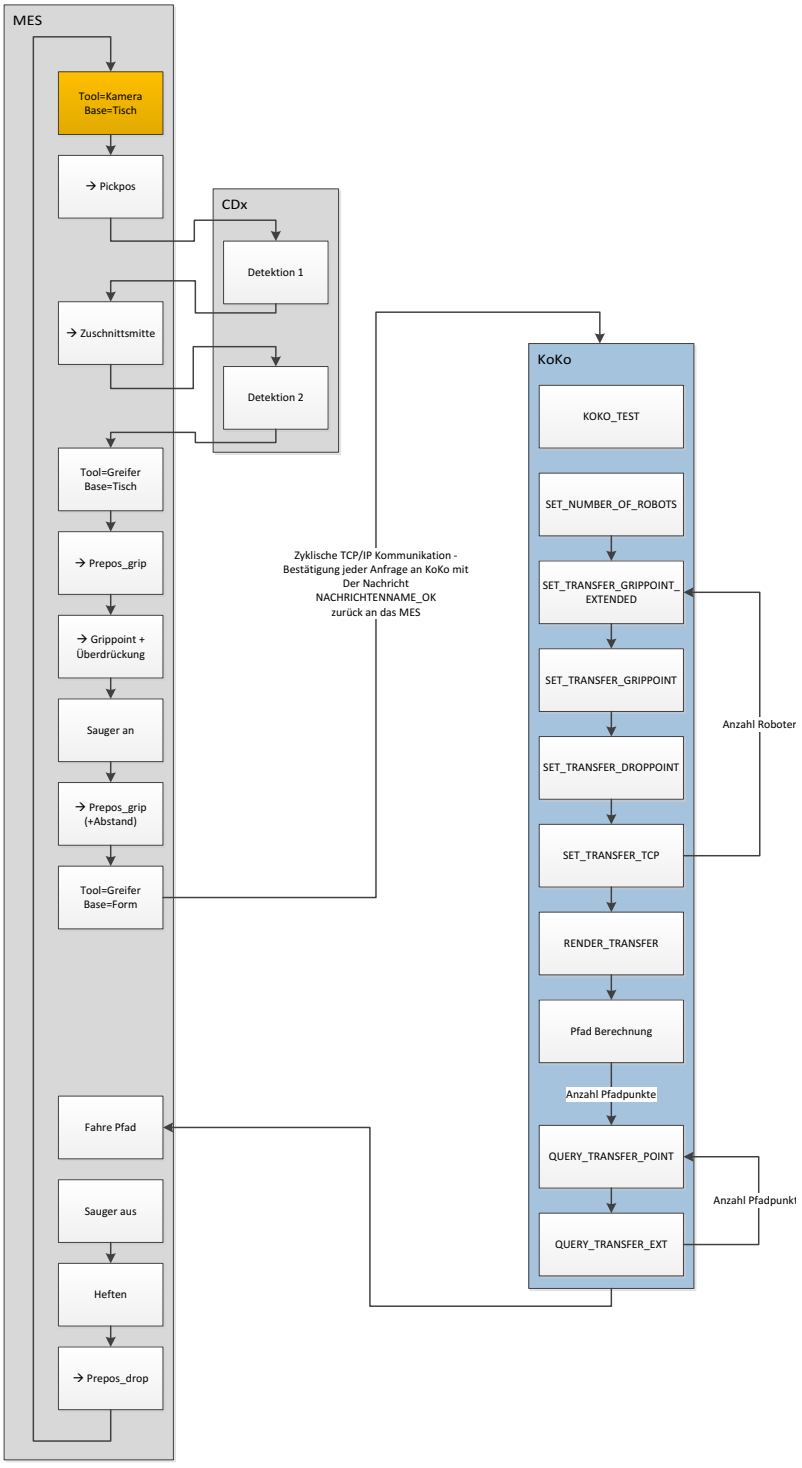


Abbildung 4.22: Komplettablauf und Kommunikation zwischen den einzelnen Komponenten des automatischen Preforming-Systems. Die Ansteuerung der Roboter erfolgt in diesem Fall durch das MES über EthernetKRL.

Kapitel 5

Validierung der Pfadplanungsalgorithmen in der Simulation

Kapitel 5 vergleicht verschiedene Pfadplanungsalgorithmen, sowohl die aus OMPL als auch die in dieser Arbeit entwickelten EA im KoKo-Framework. Zunächst werden die Planungsalgorithmen in einem vereinfachten Szenario mit nur einem Roboter validiert und deren beste Einstellungen ermittelt. Im Anschluss daran werden die besten Planer in einem realistischen Produktionsszenario mit kooperierenden Robotern getestet. Sowohl für die Sampling-basierten Planer als auch für die EA wurde jeder Parametersatz zehnmal durchlaufen, um den Median der Ergebnisse zu erhalten. Für die Planung im SE(3) wird immer die erste Lösung der IK verwendet, da diese im Arbeitsraum vor dem Roboter die plausibelste Lösung ergibt. Sollten Achsanschläge erreicht werden, wird die nächste valide Lösung verwendet. In Single-Robot-Szenario kommt der blanke Roboter ohne Anbauten für die Energieversorgung zum Einsatz. Im Produktionsszenario sind die Roboter zusätzlich mit den Störkonturen der Energieversorgung versehen, um ein möglichst gute Repräsentation der Roboterzelle in der Simulation zu erreichen. Alle Experimente wurden auf einem Intel Core i7-4810MQ CPU mit 2.80 GHz und 32 GB Arbeitsspeicher durchgeführt. Als Betriebssystem kam ein 64-Bit Windows 10 zum Einsatz. Das .NET Framework wurde in der Version 4.5.2 verwendet.

Als Bewertungskriterium dienten die Erfolgsrate, die Pfadlänge, die Berechnungszeit und die Achsbewegung entlang des Pfads. Die Ergebnisse der Durchläufe werden jeweils durch Boxplots, eine Tabelle mit den Werten und Screenshots dargestellt. Boxplots wurden 1977 von Tukey [154, S.47] vorgestellt und sind ein Mittel, um statistische Daten visuell darzustellen.

Der rote Strich im Boxplot zeigt das 0,5-Quantil bzw. den Mittelwert. Der Strich direkt darunter bzw. darüber zeigt das 0,25-Quantil (unteres Quartil) und das 0,75-Quantil (oberes Quartil). Zwischen oberem und unterem Quartil liegt die Hälfte der Stichprobe, unterhalb des unteren Quartils und oberhalb des oberen Quartils jeweils ein Viertel der Stichprobe. Auf Basis der

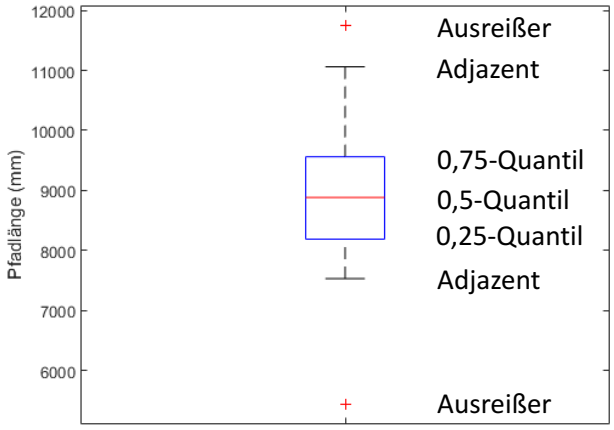


Abbildung 5.1: Beispiel eines Boxplots.

Quartile wird der Interquartilsabstand, ein Streuungsmaß, definiert. Die Nadelkristalle zeigen die minimalen und maximalen Datenwerte. Durch die roten Kreuze werden die Ausreißer dargestellt. Als Ausreißer gelten Werte, die mehr als das 1,5-fache des Quantilenabstandes vom Median entfernt sind. Je höher die blaue Box ist, umso weiter liegen die Werte auseinander. Besteht eine Box aus nur einer roten Linie, deutet dies an, dass es nur ein Sample in der Beobachtung gibt.

5.1 Planung für einen Roboter

Die im vorherigen Kapitel in KoKo integrierten Planungsalgorithmen wurden an einem Testszenario mit einem Roboter evaluiert. Da die Rechenzeit in einem reduzierten Szenario wesentlich geringer ist als für den späteren Demo-Panel-Anwendungsfall mit kooperierenden Robotern, ließen sich so wesentlich mehr Kombinationen der Einstellungen validieren. Abbildung 5.2 zeigt das Single-Robot-Szenario. Dieses besteht aus einer Reihe an Hindernissen in Blau, die die Bewegungsfreiheit des Roboters sehr stark einschränken. Es wurde so gewählt, um die Performance der verschiedenen Planer zu ermitteln. Der rote Quader auf der linken Seite markiert den Startpunkt des Pfades, der grüne auf der rechten Seite den Zielpunkt. Der graue Fußboden, die gelbe Decke und die ebenfalls gelbe Wand stellen weitere Hindernisse dar. Letztere sind so nah an den Roboter gerutscht, dass dieser bei der Bewegung vom Start- zum Zielpunkt gezwungen ist, den TCP unterhalb des rechten blauen Hindernisses zu bewegen, um eine Kollision mit der Achse 3 zu verhindern.

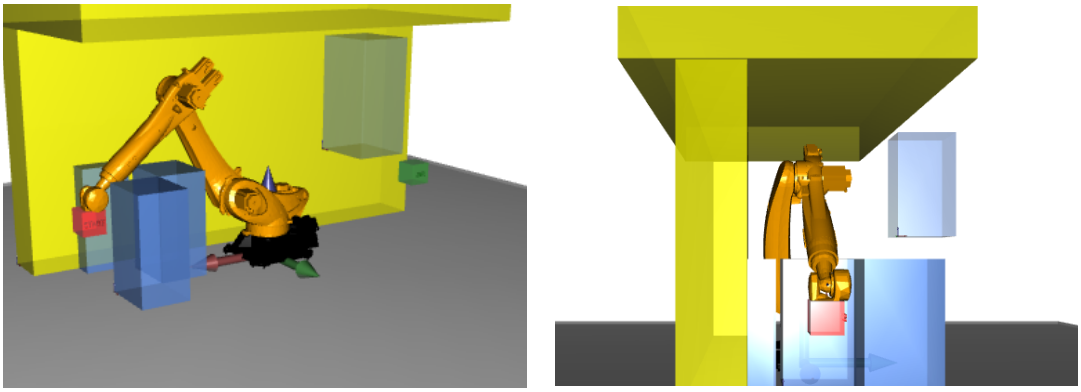


Abbildung 5.2: Screenshots des Single-Robot-Szenarios.

5.1.1 Sampling-basierte Algorithmen

In Abschnitt 4.3 wurden verschiedene Parameter vorgestellt, die für die Planung mit OMPL angepasst werden können. Das Benchmarking erfolgte anhand der folgenden Parametern:

- Collision Check Resolution: 0,001
- Plan with Experience: nein
- Plan with Constraints: ja
- Plan in Joint Space: nein
- Optimize Path Length: nein
- Solve Time on Scratch: 20 min
- Distance Disabling Constraints: für dieses Szenario nicht relevant
- Use Robot 1 E1 Machine Learning: für dieses Szenario nicht relevant
- Use Robot 2 E1 Machine Learning: für dieses Szenario nicht relevant
- Use serialized Model for Machine Learning: für dieses Szenario nicht relevant
- State Sampler: Uniform
- Simplify Path: ja und nein
- Nr. of points per Path Section: 10
- Planner Range (Joint Space/SE(3)): 40 von 100

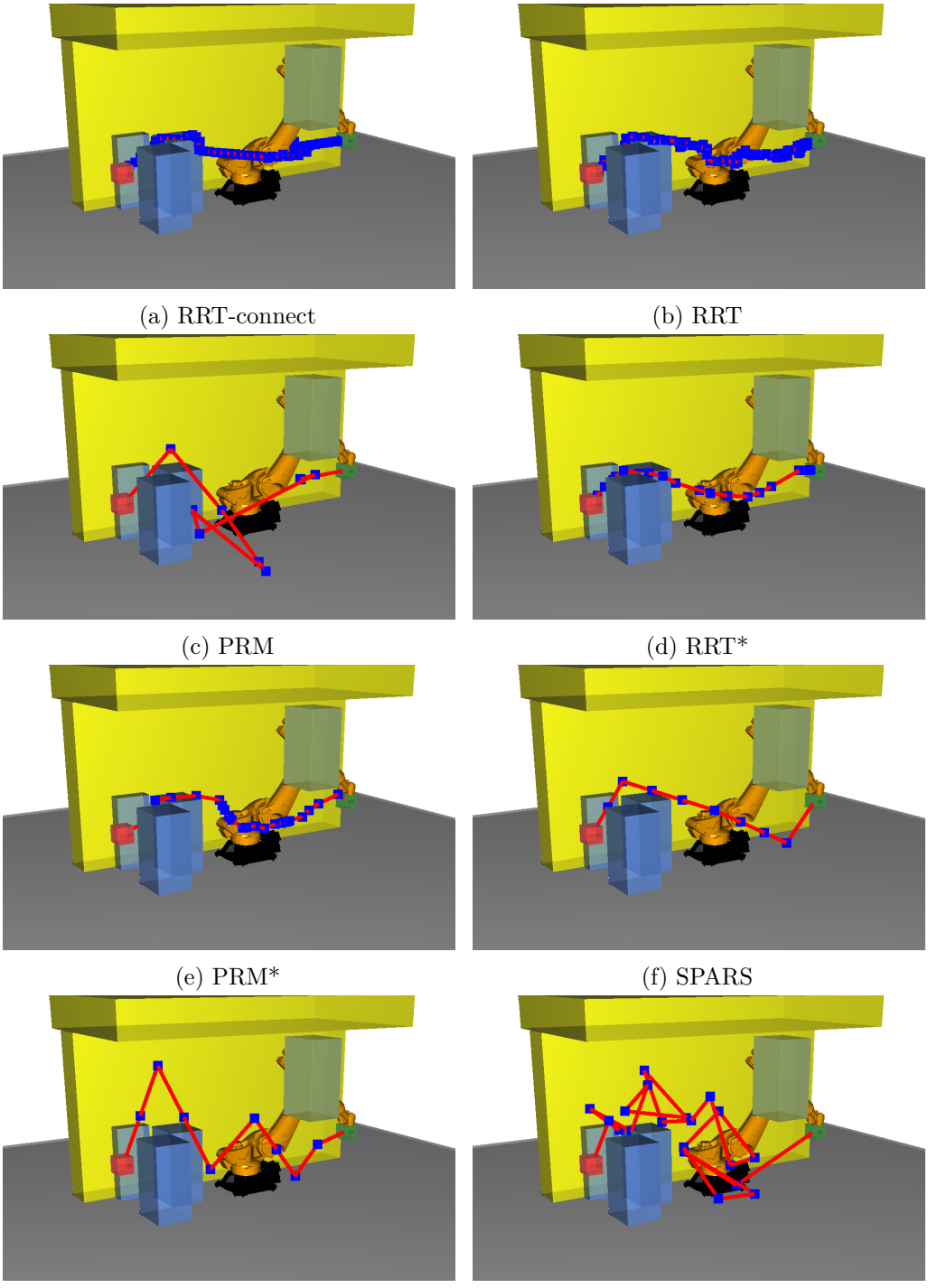
Für das zukünftige Demo-Panel ist eine Planung mit Constraints relevant, da die Orientierung des Materials für die Seilkurve gehalten werden muss. Für den Joint Space hat sich in Vorversuchen gezeigt, dass die Sampling-basierten Planer in diesem engen Szenario Probleme haben, einen Pfad zu finden. Da zudem die EA im SE(3)-Raum arbeiten, wurde die Planung im Joint Space für OMPL nicht weiter verfolgt. OMPL bietet die Möglichkeit berechnete Pfade zusätzlich durch eine Pfadoptimierung zu vereinfachen. Da die EA-Planer diesen Vereinfachungsschritt nicht beinhalten, wurden die Sampling-basierten Algorithmen zunächst mit und in einem zweiten Schritt ohne

Pfadoptimierung durchgeführt. Die maximale Planungszeit für die optimierenden Planer betrug 20 min.

Bei der Pfadvereinfachung (PV) (Simplify Path) in OMPL werden nacheinander folgende Schritte durchgeführt: Zunächst werden Ausreißer auf dem Pfad entfernt. Dies ist ein iterativer Prozess, der versucht den Pfad zu kürzen, ohne ihn ungültig zu machen. In einem nächsten Schritt werden sehr eng aneinanderliegende Pfadpunkte entfernt. Danach wird erneut versucht, den Pfad zu kürzen. Dazu werden zunächst zufällig Punkte auf diesem erzeugt; diese werden im nächsten Schritt miteinander verbunden. Anschließend wird geprüft, ob die neu generierten Pfadsegmente kollisionsfrei sind. Ist dies der Fall, wird der Pfad gekürzt, indem die dazwischenliegenden Punkte eliminiert werden. Zuletzt wird der Pfad noch mit B-Splines geglättet, was dazu führen kann, dass erneut Punkte auf dem Pfad eingefügt werden. Eine genauere Beschreibung der PV bzw. der Quellcode kann unter *Open Motion Planning Library: A Primer* [149] eingesehen werden.

In Abb. 5.3 sind die Screenshots der Durchläufe ohne PV zu sehen. Die Bilder zeigen jeweils das Ergebnis mit dem kürzesten Pfad aus insgesamt zehn Durchläufen. Man kann sehr deutlich den Unterschied der Qualität der verschiedenen Pfade erkennen. RRT-connect, RRT, RRT*, PRM* und SPARS finden auch ohne Vereinfachung sehr vernünftige Pfade, während die Ergebnisse von PRM, STRIDE und KPIECE für eine industrielle Anwendung nicht nutzbar wären. Abbildung 5.4 bildet die Boxplots der Ergebnisse der zehn Durchläufe ab. Hier kann man deutlich sehen, dass die optimierenden Planer RRT*, PRM* und SPARS unter Ausnutzung der maximalen Zeit von 20 min die kürzesten Pfade generieren. RRT-connect und RRT liefern ebenfalls gute Pfadlängen. Lediglich die Ergebnisse von PRM, STRIDE und KPIECE sind nicht überzeugend. Die besten Zeiten konnten RRT-connect und RRT mit im Median 2,0 s bzw. 3,9 s erreichen. PRM, KPIECE und STRIDE liegen zwischen 14,4 s und 26,3 s. RRT*, PRM* und SPARS nutzen die maximale Zeit. Tabelle 5.1 enthält die Ergebnisse der Simulationen in Tabellenform. Es sind jeweils die Mediane, das Minimum und das Maximum für Pfadlänge, Berechnungszeit und Achsbewegung der zehn Durchläufe für jeden Planer aufgeführt.

Im Vergleich sind in Abb. 5.5 die Ergebnisse mit PV abgebildet. Die Ergebnisse der verschiedenen Planer sind in diesem Fall sehr ähnlich, mit dem bloßen Auge sind kaum Unterschiede zu erkennen. Der Bereich der Mediane für die Pfadlänge bewegt sich zwischen 4811 mm und 5467 mm, während er sich ohne Vereinfachung zwischen 4916 mm und 29 637 mm bewegt hat. Auf die Planungszeit hat die Vereinfachung keinen Einfluss. Die Achsbewegung ist aufgrund der geglätteten Pfade wesentlich geringer. Ohne PV hat sich diese im Bereich von 1363 mm für RRT* und 3525 mm für KPIECE bewegt, sowie ebenfalls ohne Vereinfachung in einem wesentlich kleineren Bereich von 1326 mm für RRT-connect und 1410 mm für KPIECE.



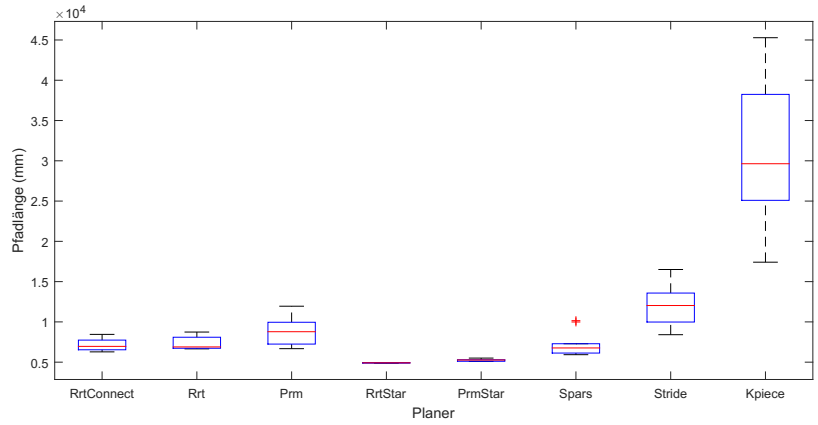
(a) RRT-connect (b) RRT

(c) PRM (d) RRT*

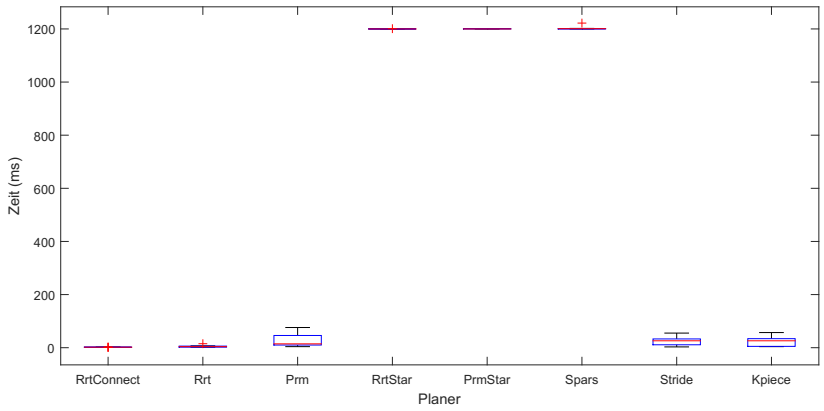
(e) PRM* (f) SPARS

(g) STRIDE (h) KPIECE

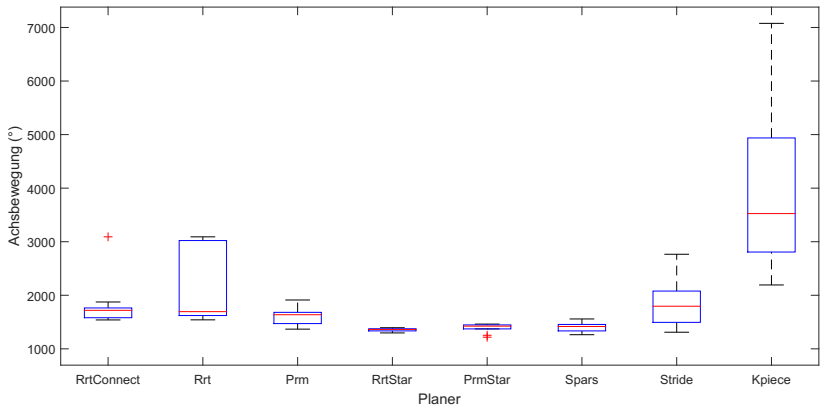
Abbildung 5.3: Screenshots der Ergebnisse der Bahnplanung Sampling-basierter Methoden mit Constraints und ohne PV im Single-Robot-Szenario. Die Bilder zeigen jeweils das Ergebnis mit der kürzesten Pfadlänge aus den zehn Durchläufen. Die Qualität der Pfade unterscheidet sich deutlich zwischen den einzelnen Planern.



(a) Pfadlänge.



(b) Berechnungszeit. RRT*, PRM* und SPARS sind optimierende Planer und nutzen die maximale verfügbare Zeit von 1200 s.

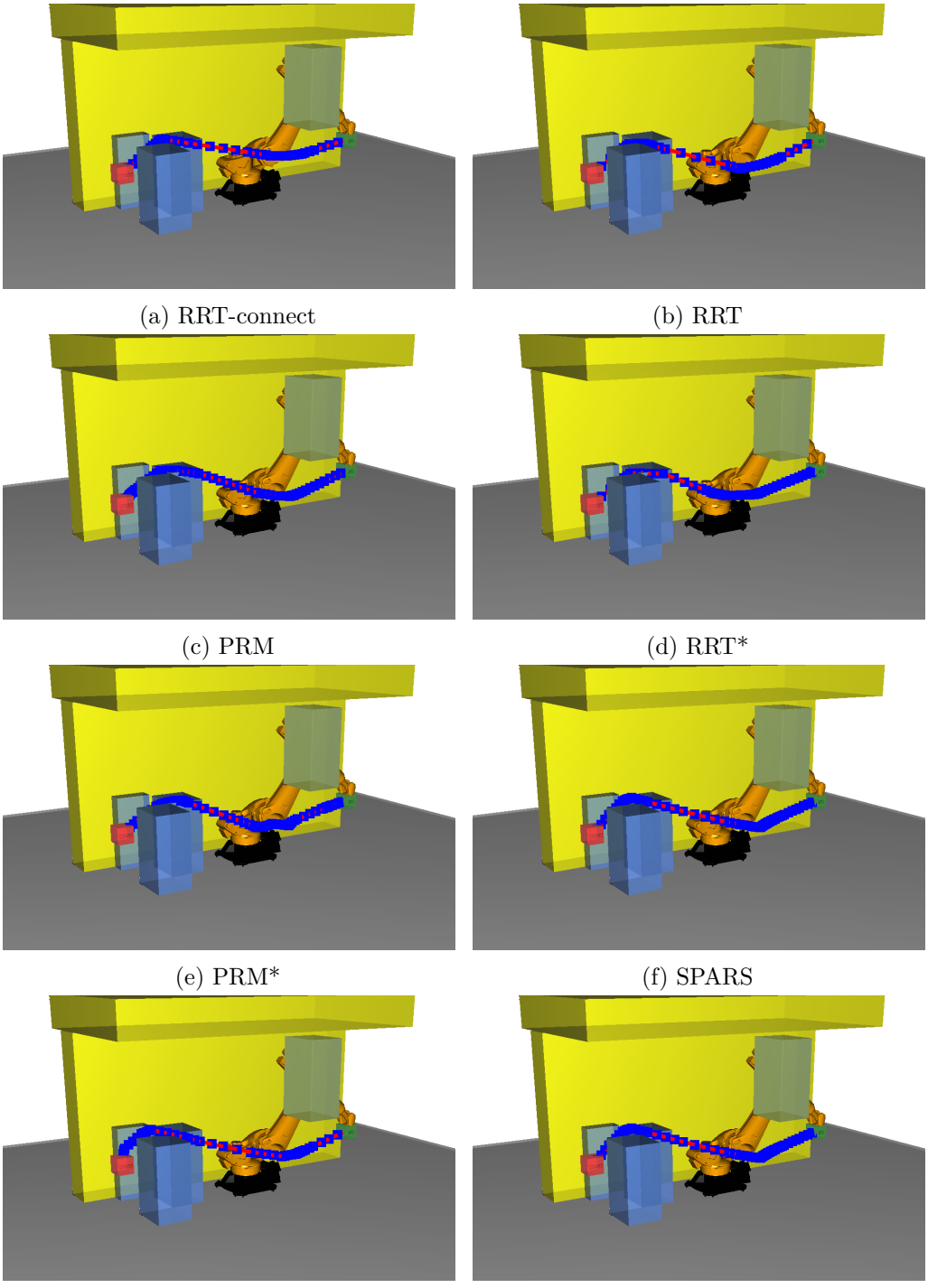


(c) Summierten Achsbewegung.

Abbildung 5.4: Vergleich von Pfadlänge, Berechnungszeit und summierter Achsbewegung Sampling-basierter Methoden mit Constraints und ohne PV im Single-Robot-Szenario.

Planer	Erfolg (%)	Pfadlänge (mm) MED / MIN / MAX	Zeit (s) MED / MIN / MAX	Achsbewegung (°) MED / MIN / MAX
RRT-connect	100	6960 / 6279 / 8450	2,0 / 1,4 / 3,3	1719 / 1540 / 3091
RRT	100	6894 / 6653 / 8739	3,9 / 1,8 / 15,3	1694 / 1541 / 3091
PRM	100	8782 / 6677 / 11947	14,4 / 4,0 / 76,2	1637 / 1368 / 19112
RRT*	100	4916 / 4870 / 4952	1200 / 1200 / 1200	1363 / 1297 / 1395
PRM*	100	5268 / 5093 / 5507	1200 / 1200 / 1200	1424 / 1217 / 1462
SPARS	100	6766 / 5929 / 10153	1200 / 1200 / 1200	1418 / 1264 / 1558
SPARStwo	0	- / - / -	1200 / 1200 / 1200	- / - / -
STRIDE	0	12034 / 8411 / 16505	26,3 / 2,7 / 55,1	1796 / 1310 / 2766
KPIECE	100	29637 / 17413 / 45282	25,6 / 3,5 / 57,0	3525 / 2193 / 7077

Tabelle 5.1: Ergebnisse der Bahnplanung Sampling-basierter Methoden mit Constraints und ohne PV im Single-Robot-Szenario. Die Tabelle zeigt die Erfolgsrate, den Median, minimale und maximale Pfadlänge, Berechnungszeit und Achsenbewegung. SPARStwo hat kein Ergebnis geliefert und wurde daher aus den Plots in Abb. 5.4 entfernt.



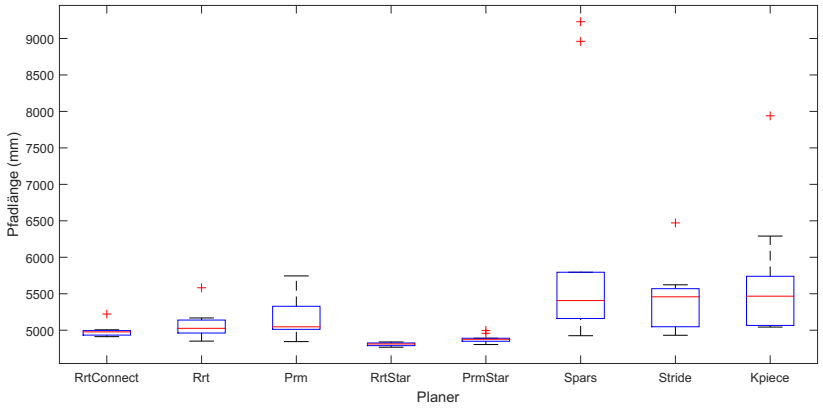
(a) RRT-connect (b) RRT

(c) PRM (d) RRT*

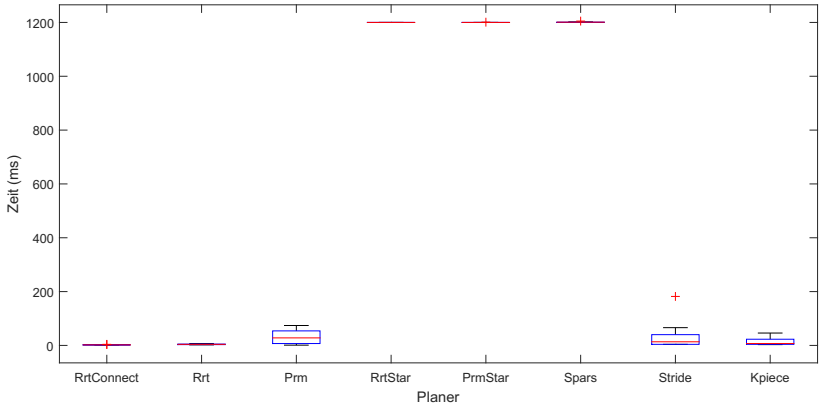
(e) PRM* (f) SPARS

(g) STRIDE (h) KPIECE

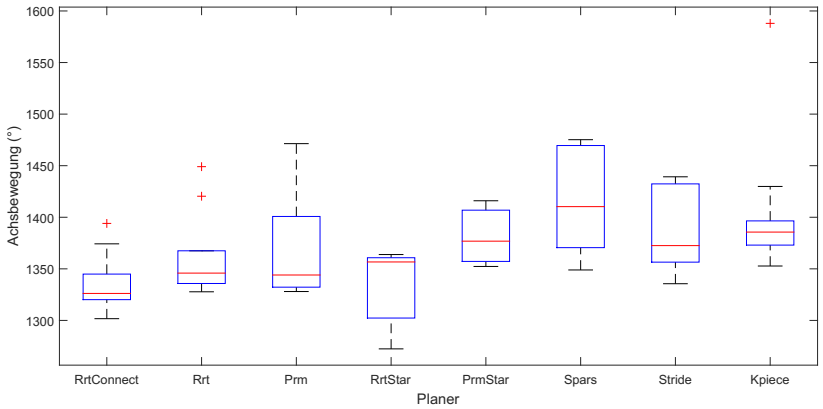
Abbildung 5.5: Ergebnisse der Bahnplanung Sampling-basierter Methoden mit Constraints und PV im Single-Robot-Szenario. Die Bilder zeigen jeweils das Ergebnis mit der kürzesten Pfadlänge aus den zehn Durchläufen. Die Pfade sind alle sehr ähnlich, und mit dem bloßen Auge ist kaum ein Unterschied erkennbar.



(a) Pfadlänge.



(b) Berechnungszeit. RRT*, PRM* und SPARS sind optimierende Planer und nutzen die maximale verfügbare Zeit von 1200 s.



(c) Summierten Achsbewegung.

Abbildung 5.6: Vergleich von Pfadlänge, Berechnungszeit und summierter Achsbewegung Sampling-basierter Methoden mit Constraints und PV im Single-Robot-Szenario.

Planer	Erfolg (%)	Pfadlänge (mm) MED / MIN / MAX	Zeit (s) MED / MIN / MAX	Achsbewegung (°) MED / MIN / MAX
RRT-connect	100	4977 / 4911 / 5221	2,0 / 1,7 / 4,0	1326 / 1302 / 1394
RRT	100	5026 / 4850 / 5582	4,0 / 2,4 / 7,1	1346 / 1328 / 1449
PRM	100	5047 / 4844 / 5744	28,0 / 1,4 / 74,5	1344 / 1328 / 1471
RRT*	100	4811 / 4767 / 4839	1200 / 1200 / 1200	1357 / 1272 / 1364
PRM*	100	4876 / 4804 / 4997	1200 / 1200 / 1200	1377 / 1352 / 1416
SPARS	100	5407 / 4925 / 9230	1200 / 1200 / 1200	1410 / 1349 / 1475
SPARStwo	0	- / - / -	1200 / 1200 / 1200	- / - / -
STRIDE	100	5458 / 4931 / 6471	13,4 / 3,8 / 181,7	1373 / 1336 / 1439
KPIECE	100	5467 / 5043 / 7940	7,0 / 2,6 / 45,6	1386 / 1353 / 1588

Tabelle 5.2: Ergebnisse der Bahnplanung Sampling-basierter Methoden mit Constraints und PV im Single-Robot-Szenario. Die Tabelle zeigt die Erfolgsrate, den Median, minimale und maximale Pfadlänge, Berechnungszeit und Achsenbewegung. SPARStwo hat kein Ergebnis geliefert und wurde daher aus den Plots in Abb. 5.6 entfernt.

5.1.2 Evolutionäre Algorithmen

EA bieten eine Vielzahl an Parametern, die verändert werden können und einen entscheidenden Einfluss auf den Erfolg des Algorithmus haben. Eine Beschreibung des Einflusses der Parameter auf die Pfadplanungsergebnisse kann in Abschnitt 3.5 nachgelesen werden. Ziel war es, anhand von Experimenten im Single-Robot-Szenario die theoretischen Vorüberlegungen zu den Einflüssen zu validieren und einen guten Parametersatz zu identifizieren. Hierzu wurden zunächst die Default-Werte aus AForge für die Parameter, die im nächsten Schritt iterativ geändert wurden, gesetzt. Für jeden Parametersatz gab es zehn Durchläufe, um den Median der Ergebnisse zu bestimmen. Terminiert wurde der Algorithmus, sobald ein kollisionsfreier Pfad ermittelt werden konnte, spätestens nach 20 min. Folgende Parameter wurden validiert:

- Via Punkte (VP): 1 bis 10 (Default: 5)
- Mutationsrate (MR): 0,1 bis 0,9 in 0,1-Schritten (Default: 0,3)
- Crossoverrrate (CR): 0,1 bis 0,9 in 0,1-Schritten (Default: 0,75)
- Random Portion Selection (RPS): 0,0 bis 0,8 in 0,1-Schritten (Default: 0,1)
- Individual Count (IC): 60, 100, 150, 200, 250, 300, 400, 500, 600, 700, 800 (Default: 100)
- Selektionsmethode (SM): Elite Selection, Tournament Selection, Rank Selection und Roulette Wheel Selection (Default: Elite Selection)

In den Grafiken Abb. 5.9 bis Abb. 5.12 sind die Ergebnisse der Durchläufe in Form von Boxplots dargestellt. Als Bewertungskriterien dienten die Erfolgsrate, die Pfadlänge, die Berechnungszeit und die aufsummierte Achsbewegung. Beim Vergleich der Ergebnisse wurde jeweils der Median in den Boxplots betrachtet. Die Screenshots der EA-Verfahren zeigen jeweils den gefundenen Pfad in Rot und die restlichen Lösungskandidaten der Population in Schwarz.

Für die Anzahl der Pfadpunkte ist die Auswertung sehr eindeutig. Die Pfadlänge steigt kontinuierlich mit steigender Anzahl an VP. Bei der Berechnungszeit zeigt sich ein ähnliches Verhalten. Lediglich fünf und sieben VP unterbrechen die Reihenfolge etwas. Für zwei VP wurde bei allen Bewertungskriterien der beste Wert erreicht, für drei VP jeweils der zweitkleinste. Da sich während der Implementierung und der Vorversuche mit den EA herausgestellt hat, dass mit zu wenig Pfadpunkten häufig kein Pfad gefunden wird und die Erfolgsrate für zwei Punkte bei 90 %, für drei bei 100 % liegt, wurden drei Punkte als bester Wert identifiziert.

MR 0,5 ergibt die kürzeste Pfadlänge, ist allerdings bei der Berechnungszeit und der Achsbewegung in der Hälfe der schlechteren Ergebnisse angesiedelt. Die Rate 0,3 ergibt den zweitkürzesten Pfad und die zweitkleinste Berechnungszeit. Lediglich bei der Achsbewegung befindet sich das Ergebnis auf den hinteren Rängen. Die Erfolgsrate für 0,5 liegt bei 80 %, für 0,3 bei 90 %. Somit wird in den folgenden Versuchen für die MR 0,3 als Wert gewählt, was auch dem Default Setting von AForge entspricht.

Den kürzesten Pfad lieferte der Planer mit der CR 0,1. Die Berechnungszeit und die Achsbewegung lagen in diesem Fall jedoch auf dem vorletzten Platz lag; die Erfolgsrate bei nur 80 %. Die Werte 0,7 und 0,6 liefern im Median den zweit- und drittkürzesten Pfad, bei einer Erfolgsrate von 80 % für 0,7 und 100 % für 0,6. Die Berechnungszeit für 0,6 liegt auf Platz 4, für 0,7 jedoch in der Hälfe der schlechteren Ergebnisse. Die kürzeste Achsbewegung kann mit der Rate 0,6 bzw. 0,8 erreicht werden; die sind bei den Ergebnissen gleichauf. Die Rate 0,7 liegt auf dem vierten Platz. Somit konnte 0,6 als gute Rate identifiziert werden. Der Default-Wert vom AForge liegt hier bei 0,75.

Mit einer RPS von 0,2 bzw. 0,6 konnten der kürzeste und der zweitkürzeste Pfad gefunden werden. Die Berechnungszeit für diese Werte liegt jedoch auf den letzten Plätzen. Die zweitkürzeste Achsbewegung konnte für den Wert 0,6 erreicht werden, wobei für diesen Wert die längste Berechnungszeit nötig war. Mit einem Wert von 0,0 konnte der drittkürzeste Pfad gefunden werden und auch die Achsbewegung sowie die Berechnungszeit waren am kürzesten. Jedoch lag die Erfolgsrate in diesem Fall bei nur 70 %. Die höchste Erfolgsrate von 100 % konnte mit 0,1 erreicht werden. Da neben der hohen Erfolgsrate für eine RPS von 0,1 auch die Streuung bei den Werten der Pfadlänge wesentlich geringer als z. B. bei 0,5–0,8 ist, wurde für RPS 0,1 als Default-Wert gesetzt.

Wie in Abb. 5.12b gut zu sehen ist, erhöht sich mit steigender Anzahl der Individuen (IC) die Berechnungszeit; dies liegt daran, dass mehr Kollisionsüberprüfungen durchgeführt werden müssen. Der kürzeste Pfad konnte für eine Anzahl von 250 Individuen gefunden werden. Die Berechnungszeit bewegt sich für diesen Wert ebenfalls in der Hälfte der schnelleren Ergebnisse. Die Achsbewegung belegt allerdings den letzten Platz. Für 100 Individuen wurde der zweitkürzeste Pfad gefunden. Die Berechnungszeit und die Achsbewegung liegen ebenfalls auf dem zweiten bzw. dritten Platz. Die Erfolgsrate beträgt in dem Fall 80 %. Wie schon in Abschnitt 3.5 erwähnt, wird in der Literatur empfohlen, die Größe der Population etwa zehnmal so groß wie den Freiheitsgrad des Problems zu wählen; in diesem Szenario wären dies 60 Individuen. Aufgrund der Simulationsergebnisse wurden 100 als Default-Wert gewählt.

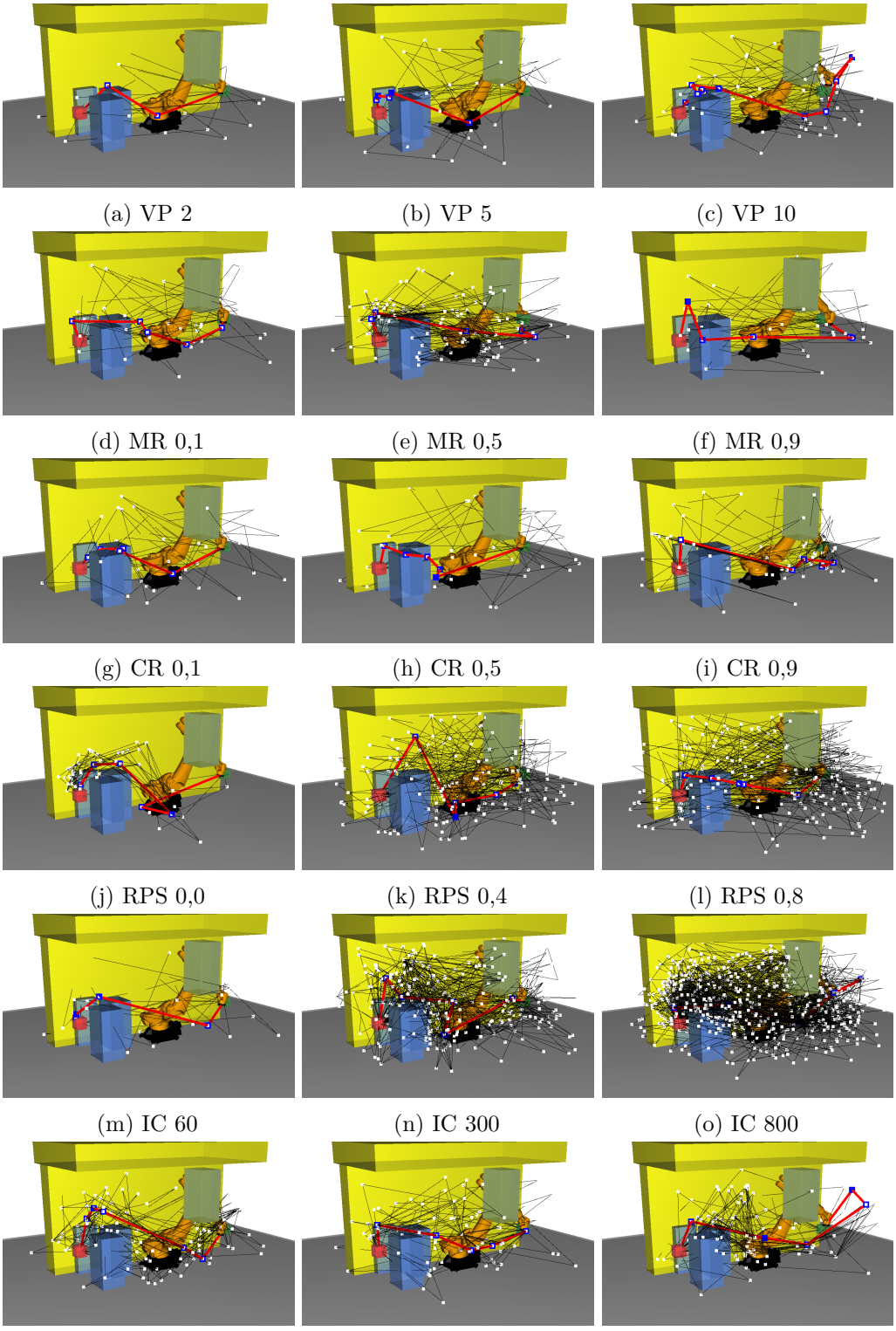
Für die SM Rangbasierte Selektion konnte der kürzeste Pfad gefunden werden, dicht gefolgt von der Elite Selektion. Die kürzeste Berechnungszeit und Achsbewegung erreichte die Elite Selektion. Erfolgsraten lagen bei der Elite-, Turnier- und Rangbasierten Selektion jeweils bei 90 %. Als Default wurde die Elite Selektion gewählt.

Zusammenfassend wurden die folgenden Parameter für den EA identifiziert:

- Via Punkte (VP): 3
- Mutationsrate (MR): 0,3
- Crossoverrrate (CR): 0,6
- Random Portion Selection (RPS): 0,1
- Individual Count (IC): 100
- Selektionsmethode (SM): Elite Selection

Abbildung 5.7 gibt einen Überblick über verschiedene Planungsergebnisse mit unterschiedlichen Parametern. Alle Screenshots zeigen Ergebnisse mit den oben aufgelisteten Default-Werten. Lediglich der in der Bildunterschrift angegebene Parameter wurde verändert. Der Screenshot zeigt jeweils das Ergebnis mit der kürzesten Pfadlänge aus den insgesamt zehn Durchläufen. Die rote Linie zeigt den gefundenen Pfad, die schwarzen Linien zeigen die restlichen Lösungskandidaten der Population. Enthält der Pfad eines Individuums mehr Zwischenpunkte, ist dieses aufwendiger zu optimieren; zum einen, weil wesentlich mehr Kollisionsprüfungen notwendig sind, zum anderen, weil die Punkte der initialen Pfade durcheinander angeordnet sein können, so dass der Pfad eine Art verworrene Schnur bildet. Das Geradeziehen der Schnur wird z. B. durch die Mutation erreicht. Da bei einer steigenden Anzahl an Pfadpunkten die einzelnen Punkte wesentlich seltener verändert werden dauert der Vorgang länger. Der Unterschied der Ergebnisse zwischen wenigen und vielen Pfadpunkten kann sehr gut in den Screenshots gesehen werden. Bei einer höheren MR werden wesentlich öfter einzelne Pfadpunkte mutiert; diese machen sich in Haken in dem Pfad bemerkbar. Dies kann z. B. in Abb. 5.7f direkt nach dem Startpunkt gesehen werden. Die geänderte CR ist in den Screenshots optisch nicht ersichtlich, die Änderung der RPS hingegen

sehr eindeutig. Diese gibt an, wie viele Chromosomen bei jeder Selektion zusätzlich zufällig neu gebildet werden. Da ohne RPS durch Selektion nur die Individuen mit der besten Fitness in die nächste Generation weiterkommen, ähneln sich die Chromosomen bei fortschreitender Iteration immer mehr. Ähnlich heißt in diesem Fall: kollisionsfrei sowie mit möglichst kurzem Pfad. Im Screenshot für RPS 0,0 schlängeln sich die verschiedenen Pfade um die Hindernisse auf der linken Seite. Bei einer höheren RPS werden mit einem immer höheren Prozentsatz neue Individuen gebildet, was zur Folge hat, dass auch nach der Terminierung sehr viele Individuen mit einer schlechten Fitness in der Population sind. Optisch kann man dies sehr gut für den Wert 0,9 sehen. Die Ergebnisse für die unterschiedliche Anzahl an Individuen einer Population ist ebenfalls sehr deutlich zu erkennen. Da bei einer großen Population prozentual auch mehr schlechte Individuen dabei sind, werden durch die Selektion diese schlechten mit in die nächste Generation vererbt; dies hat zur Folge, dass es sehr lange dauert, bis nur noch gute Individuen übrig bleiben. Da bei den Versuchen als Abbruchkriterium das Finden einer Lösung gewählt wurde, gibt es bei 800 Individuen immer noch sehr viele Kandidaten mit schlechten Pfaden. Die unterschiedlichen Selektionsmethoden lassen sich optisch ebenfalls nicht anhand der Screenshots unterscheiden.



(a) VP 2 (b) VP 5 (c) VP 10

(d) MR 0,1 (e) MR 0,5 (f) MR 0,9

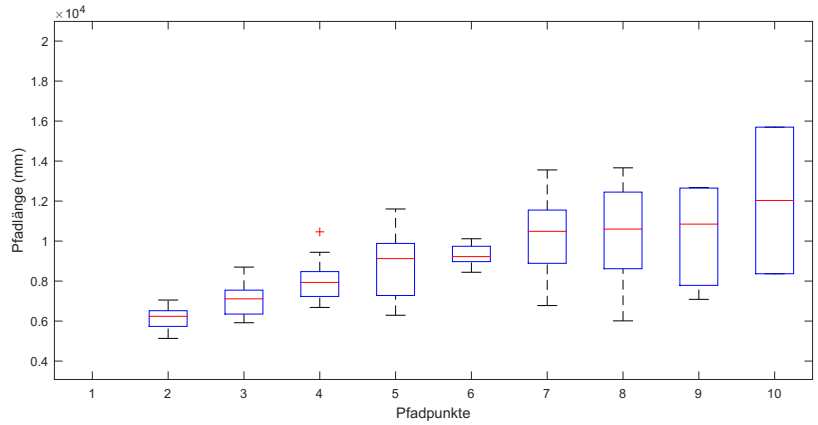
(g) CR 0,1 (h) CR 0,5 (i) CR 0,9

(j) RPS 0,0 (k) RPS 0,4 (l) RPS 0,8

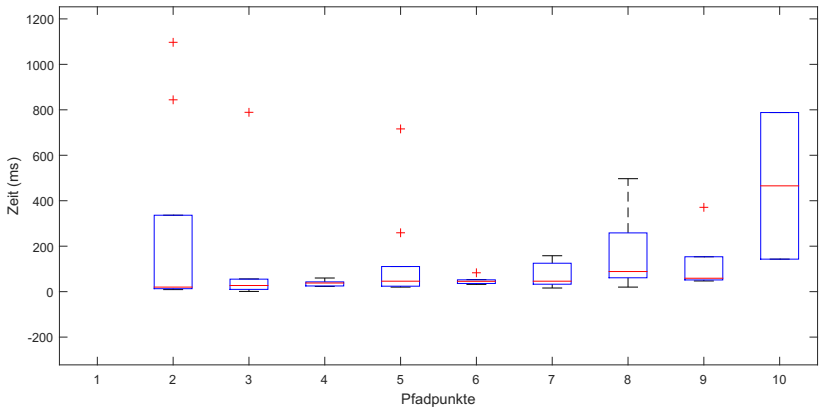
(m) IC 60 (n) IC 300 (o) IC 800

(p) SM Elite (q) SM Rank (r) SM Tournament

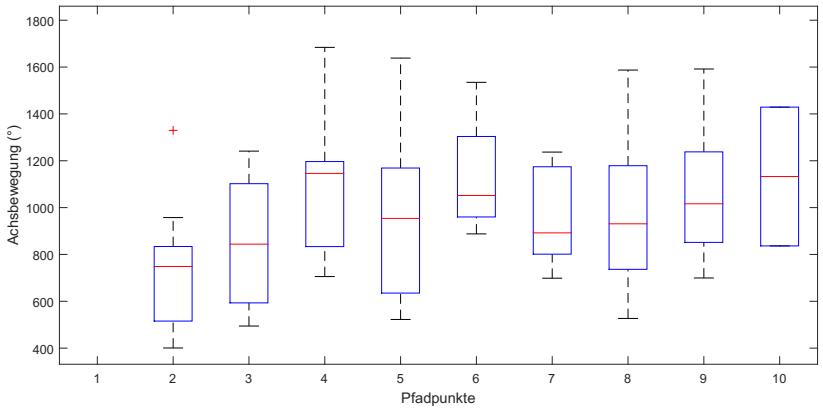
Abbildung 5.7: Screenshots der Ergebnisse der EA im Single-Robot-Szenario mit verschiedenen Parametern.



(a) Pfadlänge.

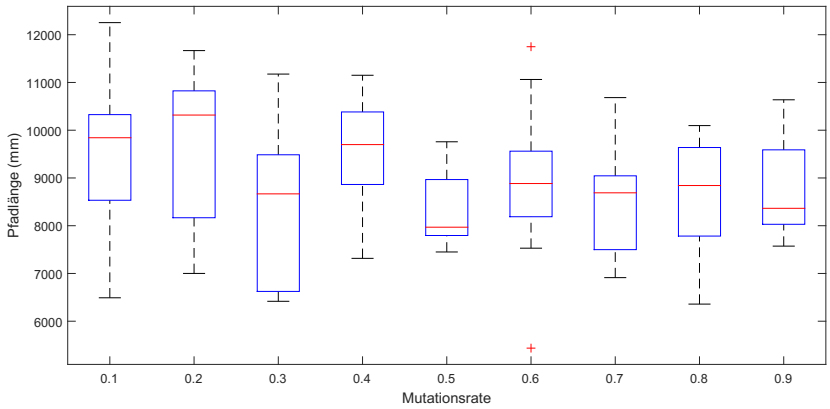


(b) Berechnungszeit.

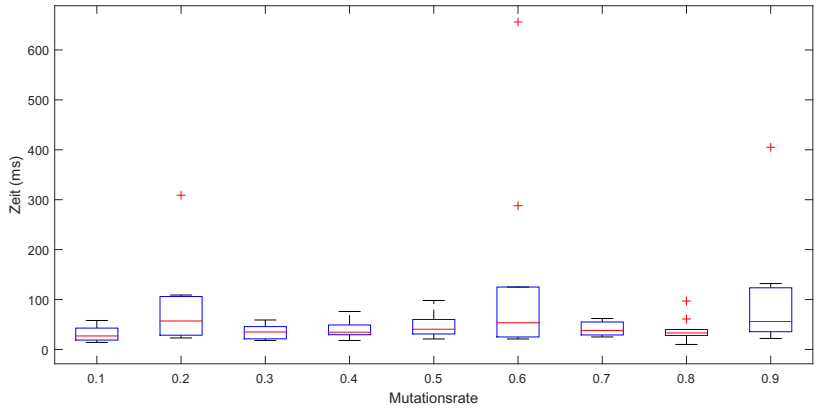


(c) Summierte Achsbewegung.

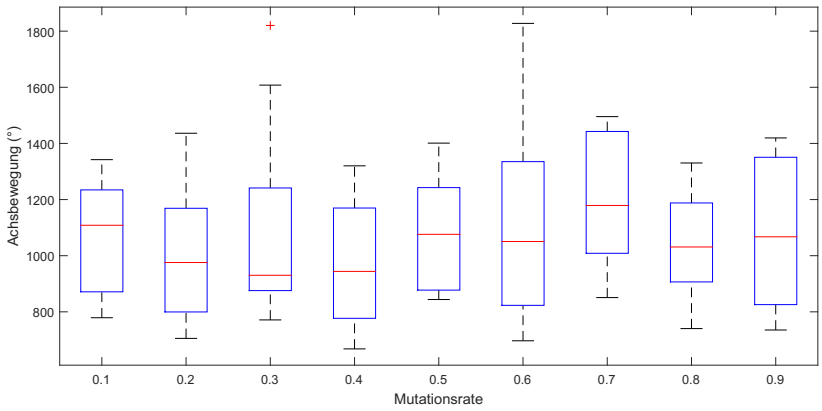
Abbildung 5.8: Vergleich von Pfadlänge, Berechnungszeit und summierter Achsbewegung des EA in Abhängigkeit der Anzahl der VP im Single-Robot-Szenario. Für einen Pfadpunkt wurde keine Lösung gefunden.



(a) Pfadlänge.

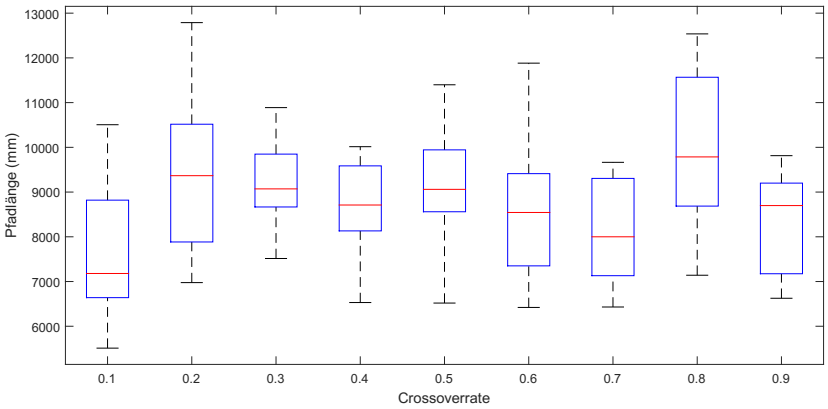


(b) Berechnungszeit.

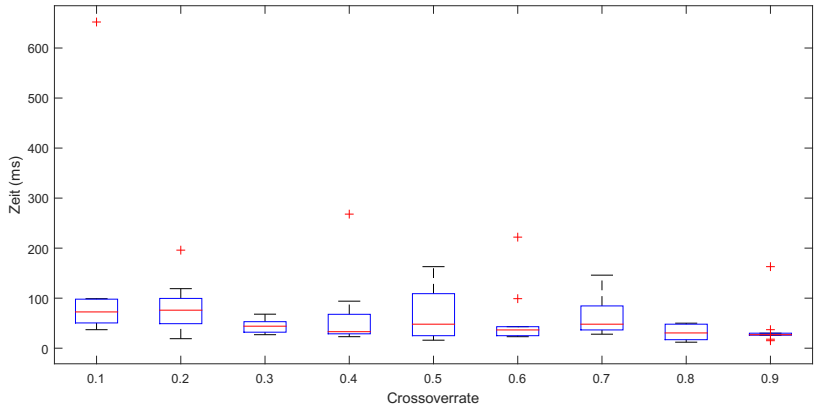


(c) Summierte Achsbewegung.

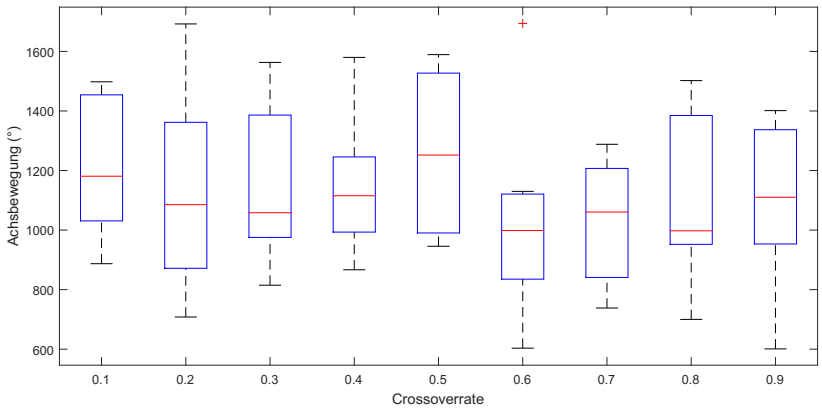
Abbildung 5.9: Vergleich von Pfadlänge, Berechnungszeit und summierter Achsbewegung des EA in Abhängigkeit der MR im Single-Robot-Szenario.



(a) Pfadlänge.

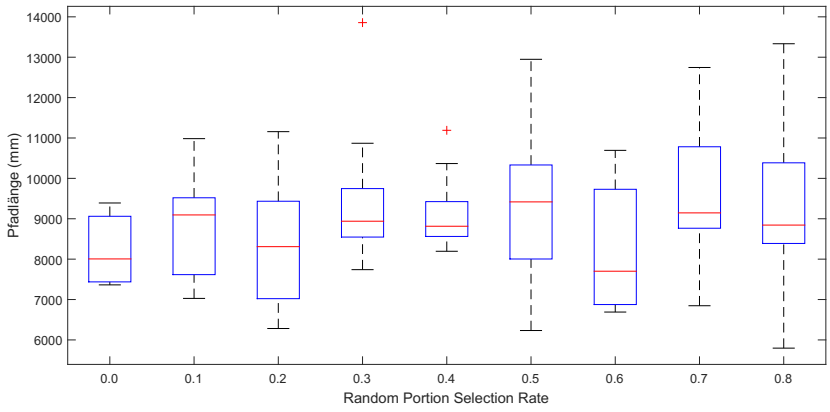


(b) Berechnungszeit.

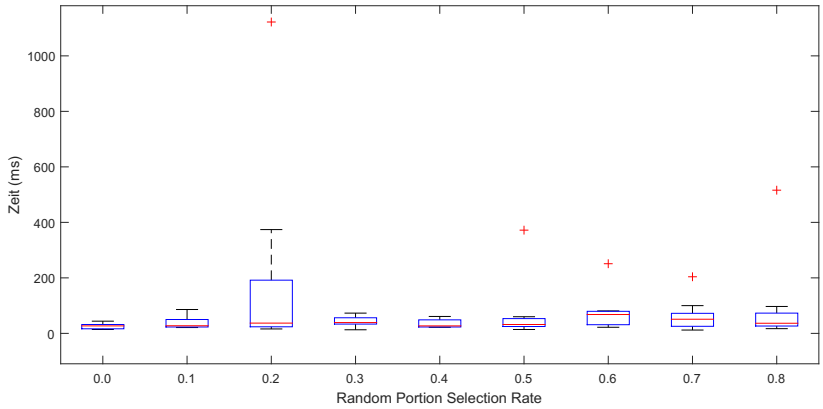


(c) Summierte Achsbewegung.

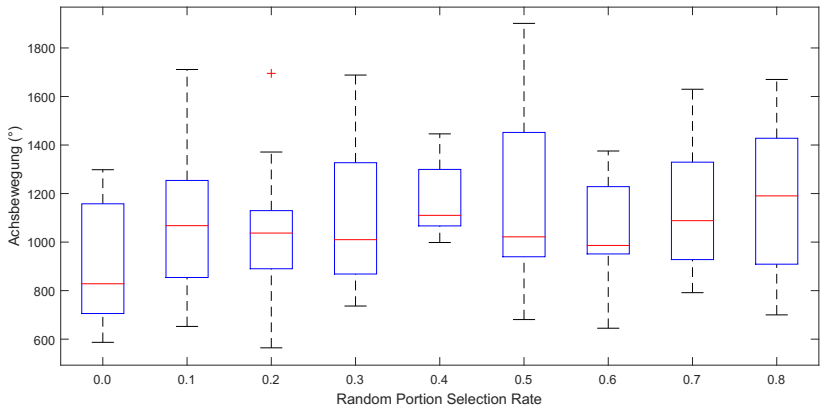
Abbildung 5.10: Vergleich von Pfadlänge, Berechnungszeit und summierter Achsbewegung des EA in Abhängigkeit der CR im Single-Robot-Szenario.



(a) Pfadlänge.

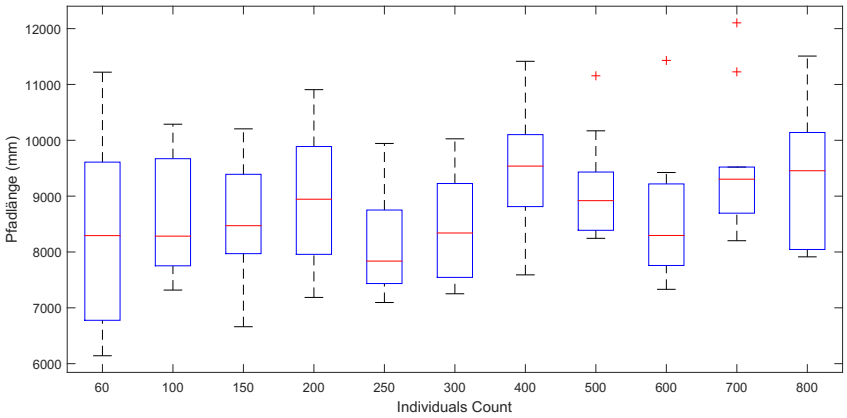


(b) Berechnungszeit.

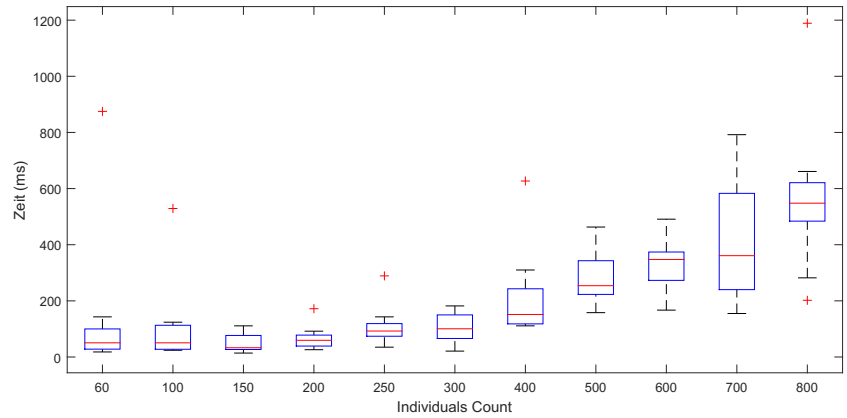


(c) Summierte Achsbewegung.

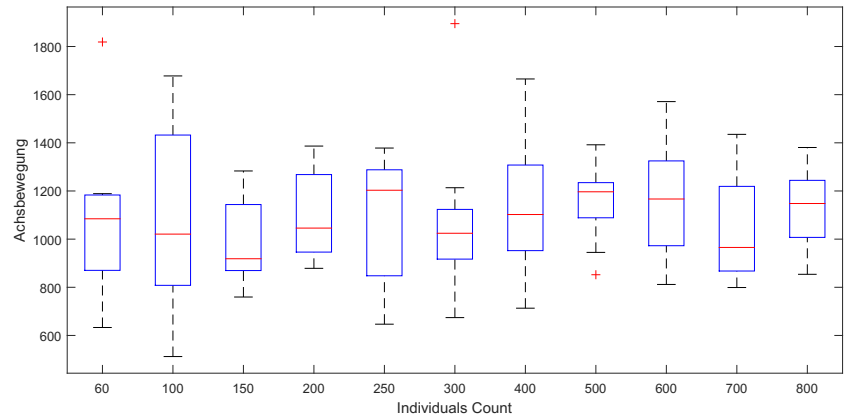
Abbildung 5.11: Vergleich von Pfadlänge, Berechnungszeit und summierter Achsbewegung des EA in Abhängigkeit der RPS im Single-Robot-Szenario.



(a) Pfadlänge.



(b) Berechnungszeit.



(c) Summierte Achsbewegung.

Abbildung 5.12: Vergleich von Pfadlänge, Berechnungszeit und summierter Achsbewegung des EA in Abhängigkeit des IC im Single-Robot-Szenario.

Parameter	Werte	Erfolg (%)	Pfadlänge (mm) MED / MIN / MAX	Zeit (s) MED / MIN / MAX	Achsbewegung (°) MED / MIN / MAX
VP	1	0	- / - / -	- / - / -	- / - / -
	2	90	6239 / 5134 / 7052	20,0 / 9,0 / 1097,0	749 / 401 / 1330
	3	100	7112 / 5917 / 8700	27,0 / 1,0 / 789,0	844 / 494 / 1241
	4	100	7932 / 6683 / 10464	38,0 / 23,0 / 60,0	1146 / 706 / 1684
	5	90	9125 / 6291 / 11606	46,0 / 20,0 / 716,0	954 / 522 / 1638
	6	80	9225 / 8441 / 10117	45,5 / 32,0 / 83,0	1052 / 888 / 1535
	7	70	10487 / 6777 / 13558	46,0 / 16,0 / 158,0	892 / 699 / 1237
	8	80	10600 / 6012 / 13663	88,5 / 20,0 / 497,0	931 / 527 / 1587
	9	50	10848 / 7087 / 12676	59,0 / 47,0 / 371,0	1016 / 700 / 1592
	10	20	12031 / 8366 / 15697	465,5 / 143,0 / 788,0	1133 / 836 / 1429
MR	0,1	90	9842 / 6491 / 12253	27,0 / 14 / 58	1108 / 779 / 1342
	0,2	90	10317 / 7001 / 11667	57,0 / 23,0 / 309,0	999 / 705 / 1436
	0,3	90	8667 / 6417 / 11175	35,0 / 18,0 / 59,0	930 / 771 / 1820
	0,4	80	9699 / 7316 / 11151	34,5 / 18.0 / 76.0	944 / 668 / 1320
	0,5	80	7969 / 7450 / 9758	40,5 / 21,0 / 98,0	1076 / 844 / 1401
	0,6	100	8883 / 5436 / 11749	53,5 / 21,0 / 656,0	1051 / 697 / 1828
	0,7	80	8689 / 6913 / 10683	38,0 / 25,0 / 62,0	1179 / 851 / 1495
	0,8	100	8842 / 6351 / 10097	33,0 / 10,0 / 97,0	1031 / 740 / 1330
	0,9	80	8400 / 7573 / 10637	39,0 / 22,0 / 405,0	1013 / 735 / 1420
CR	0,1	80	7180 / 5511 / 10506	72,5 / 37,0 / 652,0	1181 / 887 / 1498
	0,2	90	9366 / 6976 / 12788	76,0 / 19,0 / 196,0	1085 / 708 / 1692
	0,3	100	9071 / 7515 / 10888	44,0 / 27,0 / 68,0	1158 / 815 / 1563
	0,4	90	8711 / 6530 / 10015	33,0 / 23,0 / 268,0	1115 / 867 / 1580
	0,5	80	9061 / 6518 / 11399	48,0 / 16,0 / 163,0	1252 / 946 / 1589
	0,6	100	8544 / 6421 / 11822	36,5 / 23,0 / 222,0	998 / 603 / 1694
	0,7	80	8000 / 6430 / 9665	48,0 / 28,0 / 146,0	1061 / 738 / 1288
	0,8	100	9786 / 7141 / 12537	30,5 / 12,0 / 50,0	998 / 700 / 1502
	0,9	100	8698 / 6626 / 9814	27,0 / 15,0 / 163,0	1110 / 601 / 1401

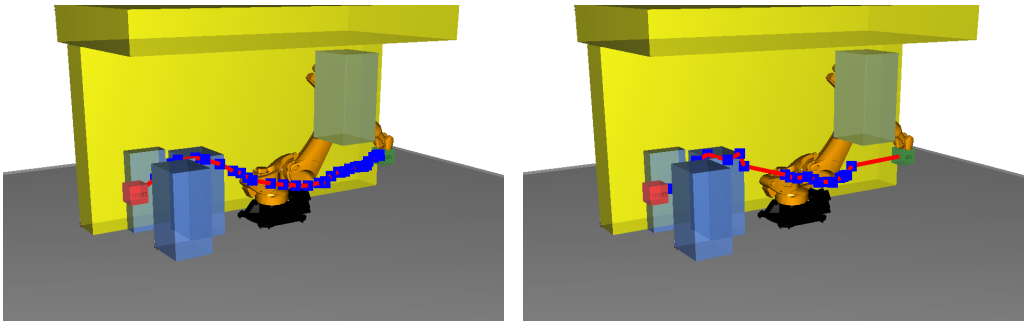
Tabelle 5.3: Ergebnisse der Bahnplanung des EA im Single-Robot-Szenario aufgeteilt in: Via Punkte (VP), Mutationsrate (MR) und Crossoverrrate (CR). Zu sehen sind jeweils der Median, das Minimum und das Maximum aus zehn Durchläufen.

Parameter	Werte	Erfolg (%)	Pfadlänge (mm) MED / MIN / MAX	Zeit (s) MED / MIN / MAX	Achsbewegung (°) MED / MIN / MAX
RPS	0,0	70	8005 / 7363 / 9391	27,0 / 14,0 / 44,0	828 / 587 / 1298
	0,1	100	9095 / 7028 / 10984	27,5 / 21,0 / 86,0	1068 / 652 / 1711
	0,2	90	7280 / 6283 / 9452	39,0 / 16,0 / 1122,0	1037 / 564 / 1695
	0,3	90	8939 / 7740 / 13857	39,0 / 13,0 / 73,0	1010 / 737 / 1688
	0,4	90	8815 / 8195 / 11190	27,0 / 22,0 / 61,0	1110 / 998 / 1446
	0,5	90	9418 / 6233 / 12948	32,0 / 14,0 / 372,0	1022 / 681 / 1901
	0,6	90	7701 / 6690 / 10694	68,0 / 22,0 / 251,0	986 / 645 / 1375
	0,7	90	9146 / 6846 / 12746	51,0 / 12,0 / 204,0	1088 / 792 / 1630
	0,8	80	8843 / 5797 / 13334	36,5 / 17.0 / 516.0	1190 / 700 / 1670
IC	60	80	8296 / 6142 / 11220	50,5 / 18,0 / 875,0	1085 / 633 / 1819
	100	80	8284 / 7318 / 10288	50,5 / 24,0 / 529,0	1021 / 512 / 1678
	150	90	8471 / 662 / 10205	33,0 / 26,0 / 172,0	919 / 879 / 1387
	200	100	8945 / 7186 / 10907	59,5 / 26,0 / 172,0	1046 / 879 / 1387
	250	100	7837 / 7095 / 9940	92,5 / 35,0 / 289,0	1203 / 647 / 1378
	300	100	8340 / 7252 / 10026	100,5 / 21,0 / 182,0	1024 / 674 / 1895
	400	100	9538 / 7590 / 11415	151,5 / 111,0 / 627,0	1102 / 713 / 1665
	500	100	8919 / 8244 / 11156	254,0 / 158,0 / 463,0	1197 / 852 / 1392
	600	100	8296 / 7331 / 11156	347,5 / 167,0 / 491,0	1167 / 812 / 1571
	700	100	9304 / 8202 / 12103	361,0 / 155,0 / 792,0	965 / 799 / 1435
	800	100	9455 / 7913 / 11509	548,0 / 202,0 / 1189,0	1148 / 854 / 1381
SM	Elite	90	8667 / 6417 / 11175	35,0 / 18,0 / 59,0	930 / 771 / 1820
	Tournament	90	10226 / 7863 / 12043	52,0 / 17,0 / 187,0	1165 / 520 / 1913
	Rank	90	7595 / 5797 / 11009	364,0 / 34,0 / 786,0	1147 / 795 / 1416
	Roulette	60	9988 / 7805 / 10826	399,5 / 20,0 / 1213,0	1617 / 1403 / 1923

Tabelle 5.4: Ergebnisse der Bahnplanung des EA im Single-Robot-Szenario aufgeteilt in: Random Portion Selection (RPS), Individual Count (IC) und Selektionsmethode (SM). Zu sehen sind jeweils der Median, das Minimum und das Maximum aus zehn Durchläufen.

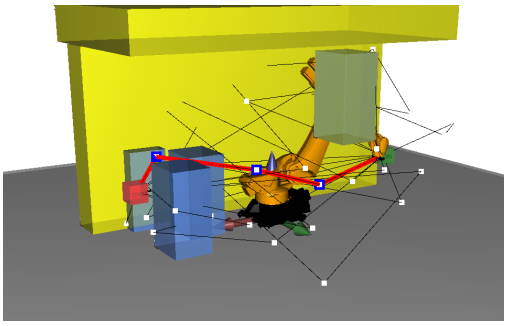
5.1.3 Diskussion der Ergebnisse

Abschließend werden noch einmal die Ergebnisse für das Single-Robot-Szenario zusammengefasst. Um die Planungsqualität zwischen den Sampling-basierten Verfahren und den EA zu vergleichen, werden nur Versuche ohne PV herangezogen. Um eine vergleichbare Bewertung zwischen nicht- und optimierenden Planern zu bekommen, wurden die EA mit den optimierten Parametern (VP=2, MR=0,3, CR=0,75, RPS=0,1, IC=100 und SM=Elite Selection) aus Abschnitt 5.1.2 mit zwei verschiedenen Abbruchkriterien durchgeführt. Die optimierenden Planer aus OMPL wurden mit dem Abbruchkriterium *maximale Zeit (MaximalTime)* verglichen, wobei der EA ebenfalls 1200 s Zeit bekommt, um den Pfad anhand der Fitness zu optimieren. Beim Vergleich mit nicht optimierenden Planern wurde das Abbruchkriterium *Pfad gefunden (SolutionFound)*, bei dem der EA direkt nach dem Finden eines kollisionsfreien Pfades abbricht, verwendet. Tabelle 5.5 gibt einen Überblick über alle Ergebnisse. Der kürzeste Pfad mit 4870 mm wurde von dem optimierenden Planer RRT* gefunden (siehe Abb. 5.13a). Der Pfad mit der kleinsten Bewegung der Achsen wurde von dem EA mit dem Abbruchkriterium *SolutionFound* ermittelt (siehe Abb. 5.14d). Die kürzeste Pfadlänge des EA liegt bei 5134 mm, was lediglich 264 mm mehr sind als bei dem optimierten Ergebnis von RRT* sowie um 1519 mm deutlich kürzer ist als das Ergebnis des nicht optimierenden Planers RRT. Die Berechnungszeit beträgt zwar im Median für RRT-connect 2 s, für RRT 4 s und für den EA 22 s, dennoch kann der EA als Konkurrent gesehen werden, zumal die Summe der Achsbewegung aufgrund der geringeren Zahl an Achspunkten wesentlich geringer ausfällt.

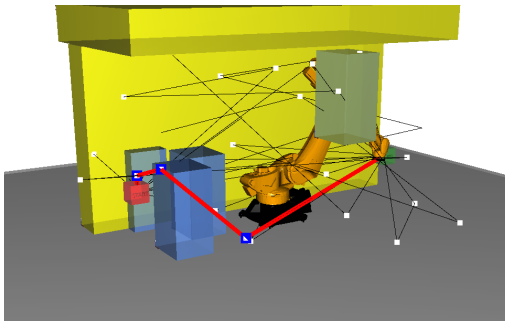


(a) RRT* – Pfadlänge 4870 mm

(b) PRM* – Achsbewegung 1217°



(c) EA 1200 s – Pfadlänge 5158 mm

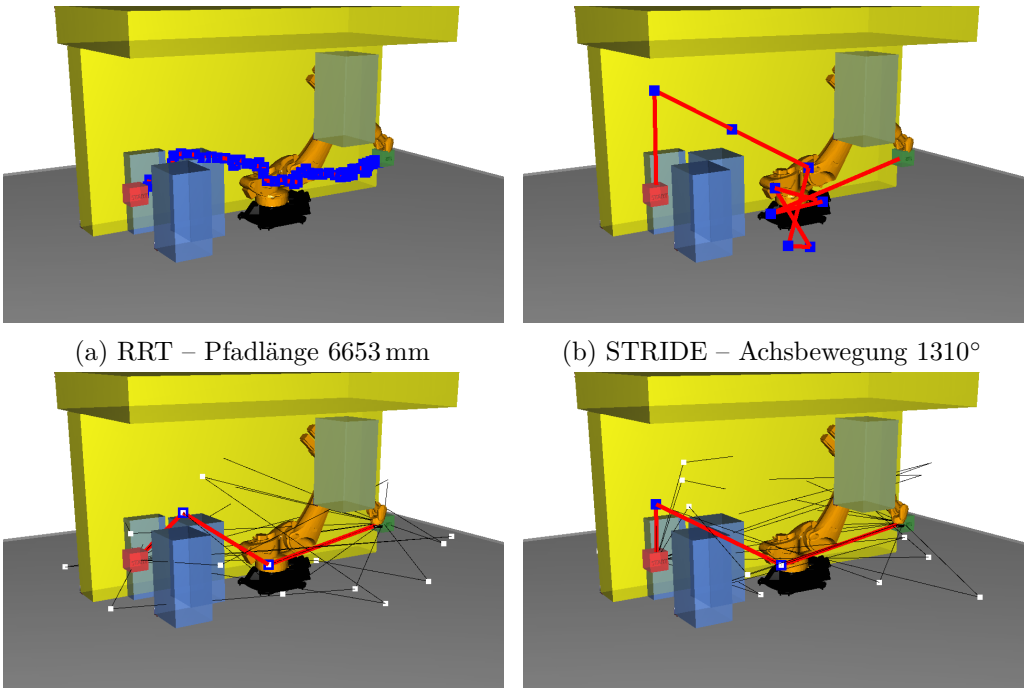


(d) EA 1200 s – Achsbewegung 447°

Abbildung 5.13: Screenshots der besten Ergebnisse alle Versuche im Single-Robot-Szenario; Sampling-basierte Planer mit PV und EA.

Planer	Pfadlänge	Zeit	Achsbewegung
RRT-connect	liefert zuverlässig kurze Pfadlängen	konstant der schnellste Planer	etwas schlechter als PRM und RRT; gleichauf mit STRIDE
RRT	liefert zuverlässig kurze Pfadlängen	zweitschnellster Planer, Zeiten nicht so konstant wie bei RRT-connect	etwas schlechter als SPARS, gleichauf mit PRM
PRM	etwas längere Pfade als RRT-connect und RRT	etwa dreimal langsamer als PRM, Zeiten streuen sehr stark	etwas schlechter als SPARS, gleichauf mit RRT
RRT*	liefert die kürzesten Pfade	maximale Zeit, da optimierend	liefert die geringste Achsbewegung von den OMPL-Verfahren
PRM*	nach RRT* die kürzesten Pfade	maximale Zeit, da optimierend	minimal schlechter als bei RRT*
SPARS	liefert nach RRT* und PRM* die kürzesten Pfade	maximale Zeit, da optimierend	minimal schlechter als bei RRT*
SPARStwo	kein Ergebnis	kein Ergebnis	kein Ergebnis
STRIDE	die zweitlängsten Pfade	etwa doppelte Zeit von PRM	etwas schlechter als PRM und RRT gleichauf mit RRTconnect
KPIECE	die längsten Pfade	etwa doppelte Zeit von PRM	mit Abstand die höchste Achsbewegung, da die Pfade sehr lang sind
EA (Abbruch: Solution Found)	Pfadlänge zwischen RRT-connect und PRM – somit in der besseren Hälfte der nicht optimierenden Planer	zwischen PRM und KPIECE – somit in der schlechteren Hälfte der nicht optimierenden Planer, im Median etwa 20 s langsamer als RRT-connect	benötigt die geringste Achsbewegung
EA (Abbruch: 1200 s)	Pfadlänge minimal schlechter als bei EA mit Abbruchkriterium SolutionFound	maximale Zeit, da optimierend	geringere Bewegung als bei EA mit Abbruchkriterium SolutionFound

Tabelle 5.5: Zusammenfassung der Planungsergebnisse für das Single-Robot-Szenario.



(a) RRT – Pfadlänge 6653 mm

(b) STRIDE – Achsbewegung 1310°

(c) EA – Pfadlänge 5134 mm

(d) EA – Achsbewegung 401°

Abbildung 5.14: Screenshots der besten Ergebnisse alle Versuche im Single-Robot-Szenario; Sampling-basierte Planer ohne PV und EA.

5.2 Planung für kooperierende Roboter

Das Ziel bei der Entwicklung des KoKo-Planungstools bestand seit jeher in dessen Einsatz für eine Produktionsanwendung mit kooperierenden Robotern. Eingeführt wurde dieses Szenario und die dafür verwendete Roboteranlage bereits in Abschnitt 1.2 eingeführt. Vor einem Einsatz im realen Demo-Panel-Szenario, erfolgte zunächst ein Vergleich verschiedener Verfahren in der Simulation. Bei dieser musste Zuschnitt 3 vom Aufnahmetisch in die untere Position der Werkzeugform transportiert werden. Wie schon im Single-Robot-Szenario wurde in enem ersten Schritt für jede Konfiguration zehnmal ein Pfad berechnet; anschließend wurden die Planer anhand der Erfolgsrate, des Medians der Pfadlänge, der Berechnungszeit und der Achsbewegung verglichen.

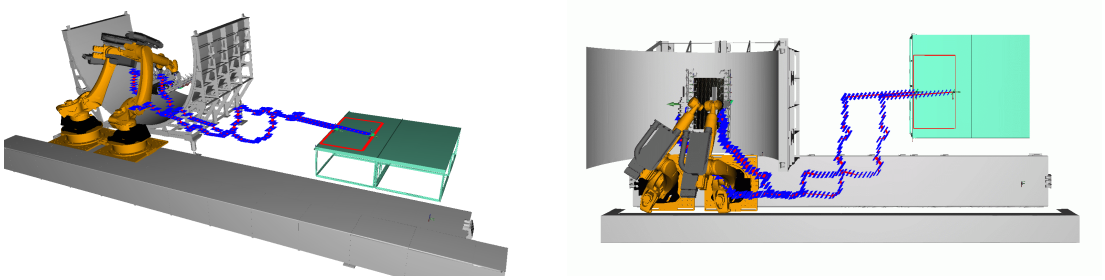
5.2.1 Sampling-basierte Algorithmen

Im Hinblick auf den Einsatz in einem industriellen Umfeld, wo die Taktzeit entscheidende eine Rolle spielt, wurden für dieses Szenario nur diejenigen Planer ausgewählt, die eine sehr schnelle Berechnungszeit haben. Die Kandidaten der Sampling-basierten Verfahren sind hier RRT und RRT-connect. Zudem lieferten diese beiden Planer zuverlässig kurze Pfadlängen.

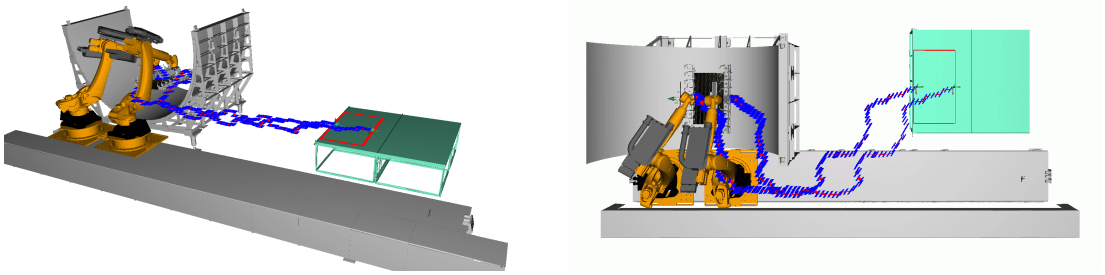
Die Einstellungen wurden analog wie in Abschnitt 5.1 gewählt:

- Collision Check Resolution: 0,001
- Plan with Experience: nein
- Plan with Constraints: ja
- Plan in Joint Space: nein
- Optimize Path Length: nein
- Solve Time on Scratch: 20 min
- Distance Disabling Constraints: für dieses Szenario nicht relevant
- Use Robot 1 E1 Machine Learning: für dieses Szenario nicht relevant
- Use Robot 2 E1 Machine Learning: für dieses Szenario nicht relevant
- Use serialized Model for Machine Learning: für dieses Szenario nicht relevant
- State Sampler: Uniform
- Simplify Path: ja und nein
- Nr. of points per Path Section: 10
- Planner Range (Joint Space/SE(3)): 40 von 100

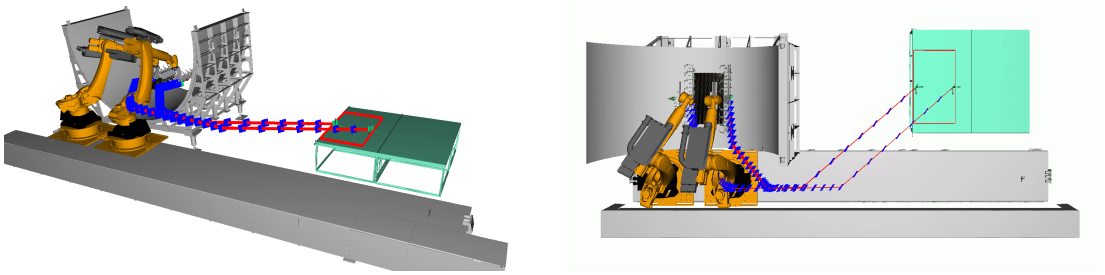
Da für die EA-Verfahren kein Extra-PV Schritt implementiert worden war, wurden die Simulationen der Sampling-basierten Algorithmen für eine bessere Vergleichbarkeit ohne und mit PV durchgeführt. Wie bereits in Abschnitt 5.1 dargestellt, sind die Pfade ohne PV wesentlich länger. Der kürzeste Pfad wurde von RRT mit einer Länge von 10 493 mm gefunden. Mit PV wurde der kürzeste Pfad von RRT mit 7697 mm gefunden. Abbildung 5.15 zeigt die Screenshots der Ergebnisse mit den kürzesten Pfaden. Ohne PV sind die Pfade sowohl bei RRT als auch bei RRT-connect sehr wellig. Mit PV ähneln diese sich bei beiden Algorithmen ab der Aufnahme bis zur Werkzeugform enorm.



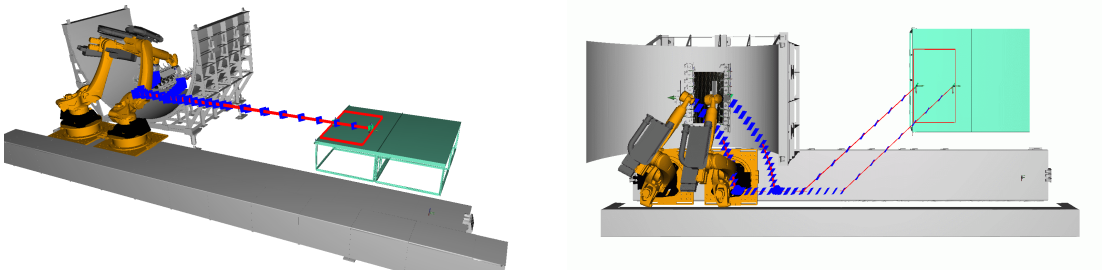
(a) RRT-connect ohne PV – Pfadlänge 11 514 mm, Achsbewegung 1682°, E1-Bewegung 8834°.



(b) RRT ohne PV – Pfadlänge 10 493 mm, Achsbewegung 2339°, E1-Bewegung 7961°.

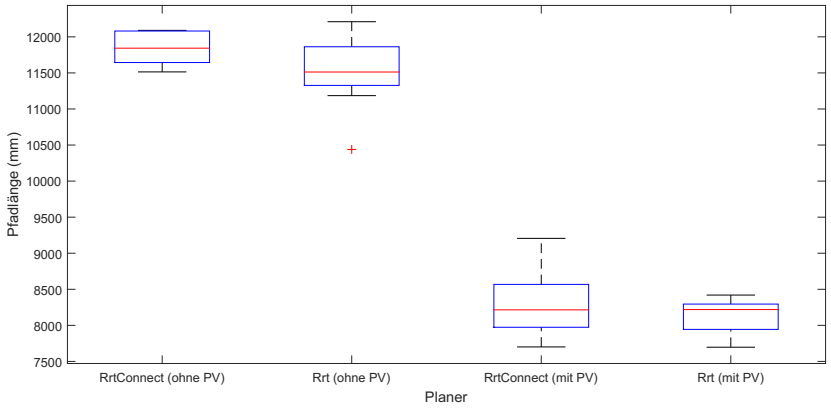


(c) RRT-connect mit PV – Pfadlänge 7701 mm, Achsbewegung 1369°, E1-Bewegung 6476°.

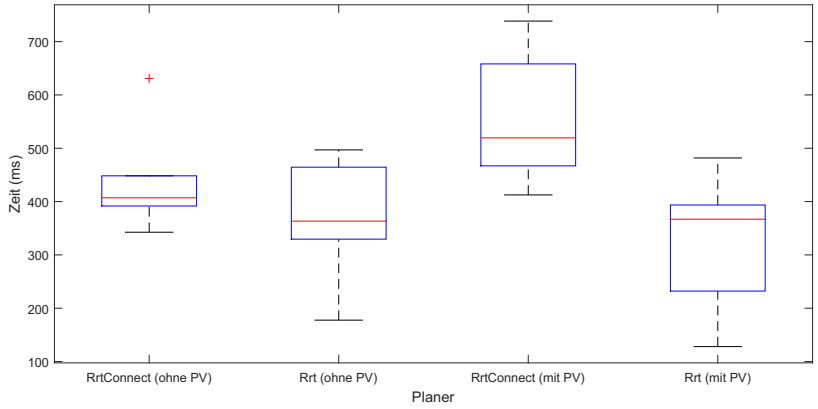


(d) RRT mit PV – Pfadlänge 7697 mm, Achsbewegung 1384°, E1-Bewegung 6474°

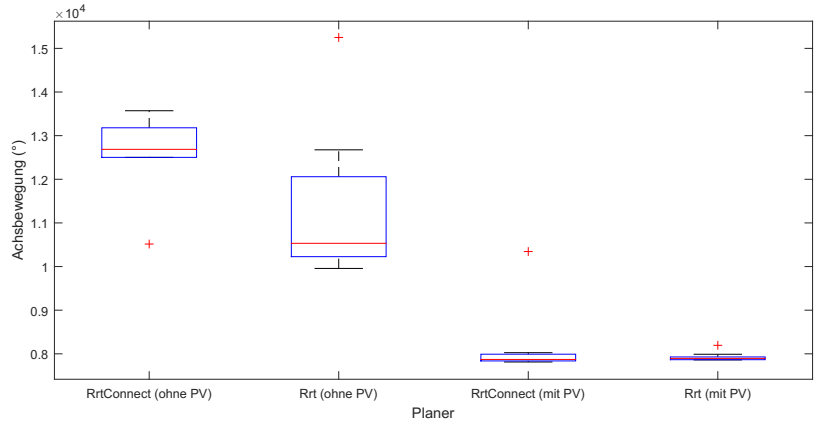
Abbildung 5.15: Screenshots der Ergebnisse Sampling-basierter Verfahren im Demo-Panel Szenario.



(a) Pfadlänge.



(b) Berechnungszeit.



(c) Summierten Achsbewegung.

Abbildung 5.16: Vergleich von Pfadlänge, Berechnungszeit und summierter Achsbewegung (Roboterachsen und externen Achse) Sampling-basierter Methoden mit Constraints, mit und ohne PV im Demo-Panel Szenario.

Planer	Erfolg (%)	Pfadlänge (mm) MED / MIN / MAX	Zeit (s) MED / MIN / MAX	Achsbewegung (°) MED / MIN / MAX	E1-Bewegung (°) MED / MIN / MAX
RRT-connect keine PV	60	11843 / 11514 / 12090	407 / 342,5 / 631	1925 / 1682 / 1978	10758 / 8834 / 11662
RRT keine PV	80	11513 / 10439 / 12210	363,3 / 177,6 / 496,9	2082 / 1655 / 3137	8431 / 7961 / 12997
RRT-connect	90	8215 / 7701 / 9206	519,2 / 412,5 / 738,5	1395 / 1337 / 1689	6476 / 6476 / 8655
RRT	90	8219 / 7697 / 8420	366,9 / 128 / 481,8	1398 / 1315 / 1431	6477 / 6476 / 6880

Tabelle 5.6: Ergebnisse der Bahnplanung Sampling-basierter Methoden mit Constraints im Demo-Panel Szenario. Die ersten beiden Ergebnisse sind ohne PV und die restlichen mit PV. Die Tabelle zeigt Erfolgsrate, Median, minimale und maximale Pfadlänge, Berechnungszeit und Achsenbewegung.

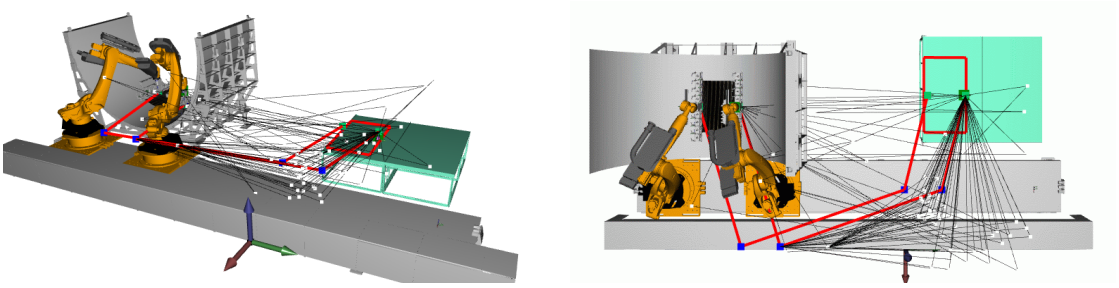
5.2.2 Evolutionäre Algorithmen

In einem ersten Durchlauf wurden exakt die Parameter aus dem Single-Robot-Szenario von Abschnitt 5.1.2 übernommen und eine Simulation für 2, 4 und 6 VP durchgeführt. Wie in Abschnitt 3.5 bereits beschrieben, sollte der IC mindestens zehnmal so groß sein wie die Anzahl der Freiheitsgrade. Die beiden Roboter haben jeweils sechs Achsen und können sich zudem auf einer Linearachse bewegen; die ergibt in Summe 14 Freiheitsgrade. Daher wurde in einem zweiten Durchlauf die Anzahl der IC auf 200 erhöht und ebenfalls eine Simulation für 2, 4 und 6 VP durchgeführt.

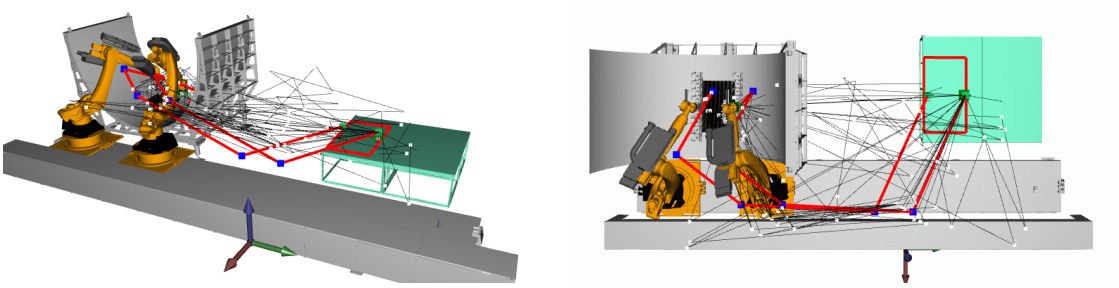
- Via Punkte (VP): 2, 4 und 6
- Mutationsrate (MR): 0,3
- Crossoverrrate (CR): 0,6
- Random Portion Selection (RPS): 0,1
- Individual Count (IC): 100 bzw. 200
- Selektionsmethode (SM): Elite Selection

Der kürzeste Pfad von allen Durchläufen der EA für kooperierende Roboter im Demo-Panel-Szenario konnte für IC=200 und VP=2 mit einer Länge von 9108 mm gefunden werden. Zugleich benötigte diese Berechnung von allen Durchläufen die kürzeste Berechnungszeit von 167 s. Die Erfolgsrate betrug dabei 90 %. Die kürzeste Pfadlänge für IC=100 von 9652 mm konnte für VP=2 erreicht werden. Die Berechnung nahm in diesem Fall 233 s bei einer Erfolgsrate von 60 % in Anspruch. Tabelle 5.7 beinhaltet eine Übersicht der Ergebnisse in Tabellenform. Abbildung 5.17 und Abb. 5.18 zeigen die Screenshots der Planungsergebnisse mit den kürzesten Pfadlängen aus allen Durchläufen; Abb. 5.19 stellt die Boxplots der Ergebnisse dar. Sowohl für IC=100 als auch für IC=200 verteilen sich die Pfadpunkte jeweils in der Höhe des Tisches und der Werkzeugform, was zu einer sehr direkten Bewegung der Roboter führt. Werden mehr Zwischenpunkte zur Berechnung des Pfades herangezogen, scheinen die Pfade auf den ersten Blick weniger kantig zu sein; dies führt zu einer weicheren Roboterbewegung. Auf den zweiten Blick jedoch ist zu erkennen, dass dies zu Zick-Zack-Bewegungen führen kann, wie z. B. in Abb. 5.17c beim Einfahren in die Werkzeugform oder in Abb. 5.18c nach der Aufnahme über dem Tisch. Zusammengefasst kann festgehalten werden, dass mit den EA in relativ kurzer Zeit überraschend gute Pfade gefunden werden können. Während der Versuche hat sich zudem gezeigt, dass es nicht nötig ist, die Fitness-Funktion sehr komplex zu gestalten.

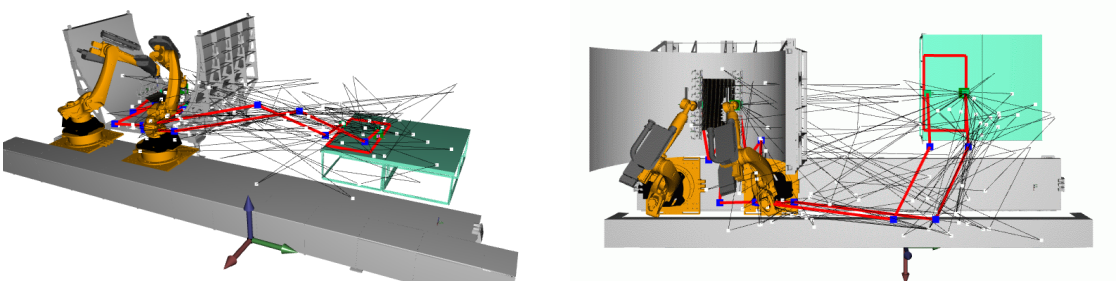
.



(a) IC=100, VP=2 – Pfadlänge 9652 mm, Achsbewegung 1093°, E1-Bewegung 5542°.

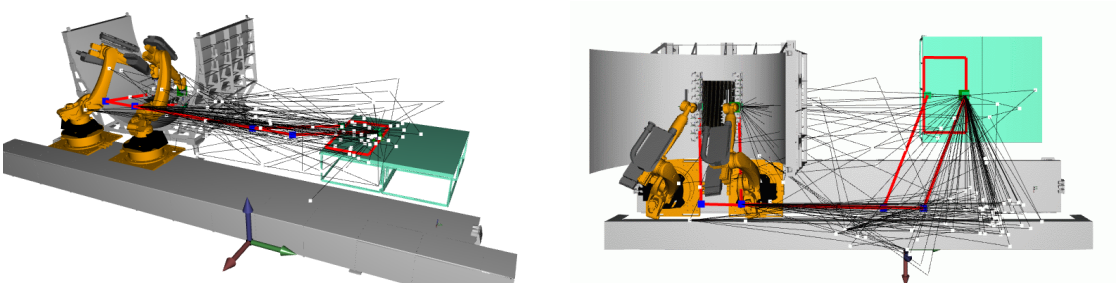


(b) IC=100, VP=4 – Pfadlänge 10 081 mm, Achsbewegung 2294°, E1-Bewegung 5815°.

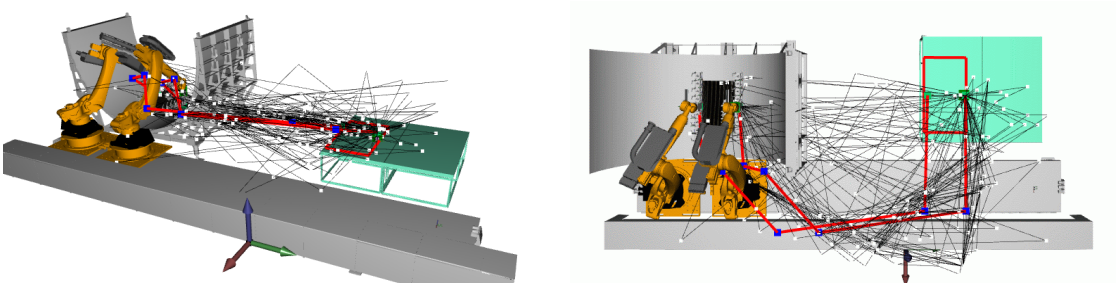


(c) IC=100, VP=6 – Pfadlänge 10 725 mm, Achsbewegung 1382°, E1-Bewegung 6079°.

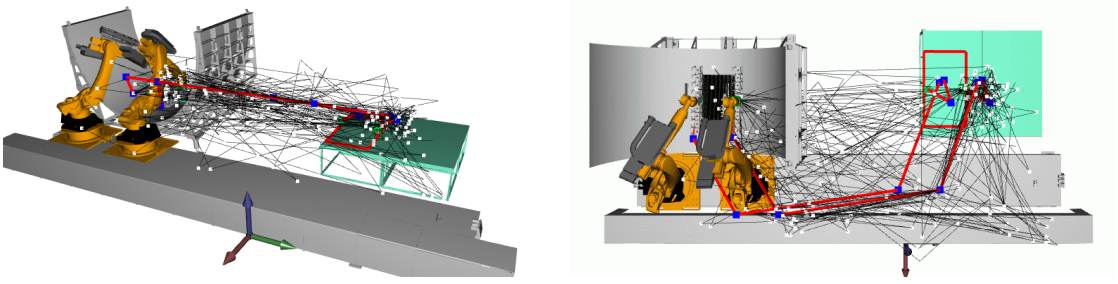
Abbildung 5.17: Screenshots der Ergebnisse des EA mit IC=100 im Demo-Panel Szenario.



(a) IC=200, VP=2 – Pfadlänge 9108 mm, Achsbewegung 1970°, E1-Bewegung 6047°.

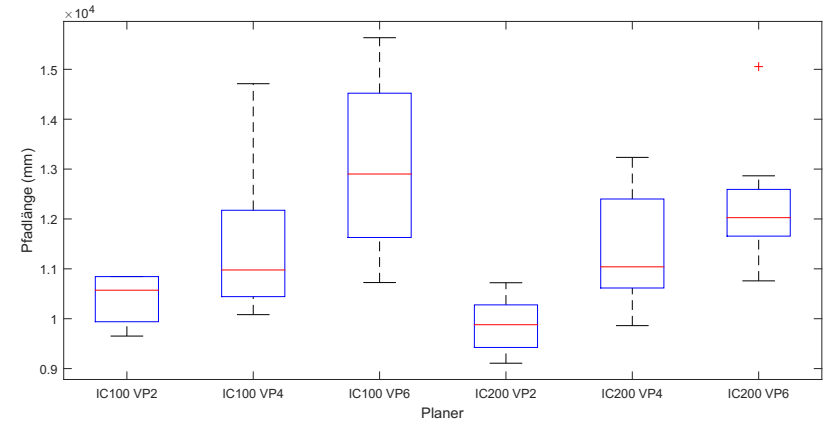


(b) IC=200, VP=4 – Pfadlänge 9861 mm, Achsbewegung 1195°, E1-Bewegung 7364°.

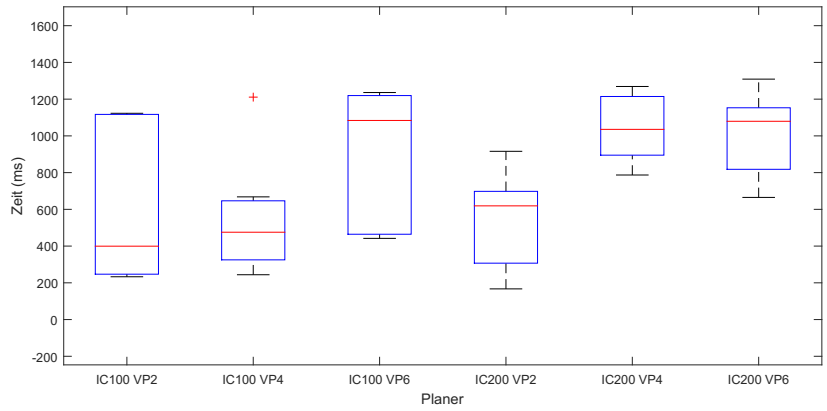


(c) IC=200, VP=6 – Pfadlänge 10 759 mm, Achsbewegung 1452°, E1-Bewegung 6697°.

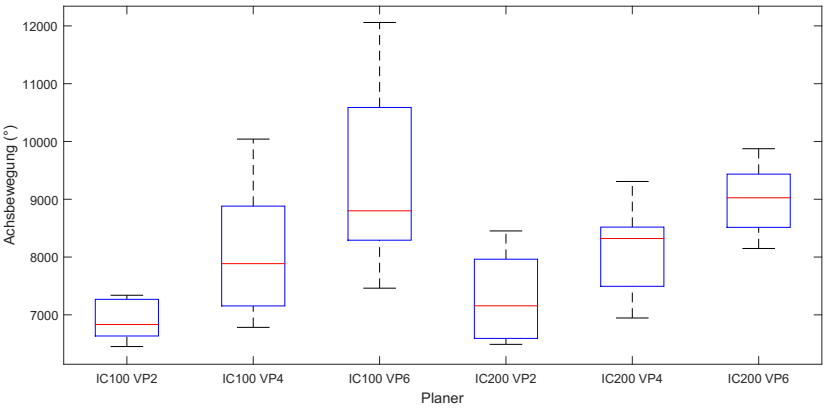
Abbildung 5.18: Screenshots der Ergebnisse des EA mit IC=200 im Demo-Panel Szenario.



(a) Pfadlänge.



(b) Berechnungszeit.



(c) Summierten Achsbewegung.

Abbildung 5.19: Vergleich von Pfadlänge, Berechnungszeit und summierter Achsbewegung (Roboterachsen und der externen Achse) des EA im Demo-Panel Szenario.

Planer	Erfolg (%)	Pfadlänge (mm) MED / MIN / MAX	Zeit (s) MED / MIN / MAX	Achsbewegung (°) MED / MIN / MAX	E1-Bewegung (°) MED / MIN / MAX
EA IC100 VP2	60	10977 / 10081 / 14712	475,5 / 244 / 1211	1252 / 1083 / 2294	6142 / 5602 / 8791
EA IC100 VP4	80	11513 / 10439 / 12210	363,3 / 177,6 / 496,9	2082 / 1655 / 3137	8431 / 7961 / 12997
EA IC100 VP6	50	12901 / 10725 / 15634	1084 / 442 / 1236	1382 / 1314 / 3217	7255 / 3079 / 8842
EA IC200 VP2	90	9880 / 9108 / 10721	619 / 167 / 916	1098 / 994 / 1970	6001 / 5391 / 7351
EA IC200 VP4	80	11040 / 9861 / 13235	1035,5 / 787 / 1269	1256 / 1088 / 2604	6828 / 5683 / 7364
EA IC200 VP6	80	12026 / 10759 / 15054	1079,5 / 665 / 1309	1514 / 1273 / 2558	7291 / 6652 / 8317

Tabelle 5.7: Ergebnisse der Bahnplanung des EA im Demo-Panel Szenario. Die Tabelle zeigt Erfolgsrate, Median, minimale und maximale Pfadlänge, Berechnungszeit und Achsenbewegung.

5.2.3 Diskussion der Ergebnisse

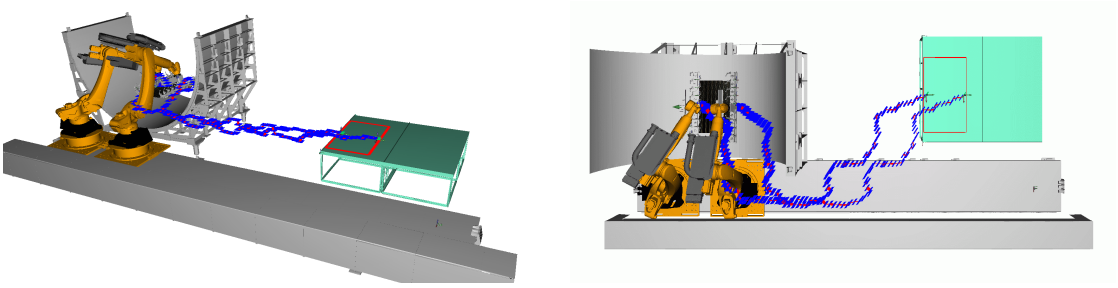
Tabelle 5.8 gibt eine tabellarische Übersicht der Planungsergebnisse von RRT, RRT-connect (sowohl mit als auch ohne PV) und dem EA (IC=200, VP=2) im Demo-Panel-Szenario. Abbildung 5.20 zeigt zudem die Screenshots der besten Ergebnisse im direkten Vergleich. RRT ohne PV liefert sehr wellige Pfade; diese sind für einen Einsatz in der industriellen Produktion kaum geeignet, da zu viele unnötige und zeitraubende Bewegungen erfolgen (siehe Abb. 5.20a). RRT mit PV liefert einen überaus optimierten Pfad, der unmittelbar an der Werkzeugform vorbeiführt. Bezogen auf die Effizienz der Bewegung bietet sich daher dieser Algorithmus in erster Linie für den industriellen Einsatz an. Dennoch könnte es zu unvorhergesehenen Kollisionen kommen, wenn nicht alle Anbauteile des Roboters sowie die Werkzeugform exakt vermessen und simuliert werden. In diesem Einsatzbereich wäre es daher ratsam, den Margin-Wert auf 0 zu setzen. Dieser Wert beschreibt, bei welchem Abstand zwischen zwei Objekten eine Kollision erkannt wird. Alternativ kann die Anzahl der Stützpunkte erhöht werden.

Die Erfolgsquote von RRT (mit PV), RRT-connect (mit PV) und EA liegt jeweils bei 90 %. Die kürzesten Pfade bei der geringsten Berechnungszeit lieferte RRT (mit PV). Die Pfadlängen des EA sind länger als die der Sampling-basierten Algorithmen mit PV und kürzer als ohne PV. Der EA kann mit einer wesentlich geringeren Roboter- und externen Achsbewegung punkten. Ohne PV fällt die Erfolgsrate bei den Sampling-basierten Verfahren auf 60 % für RRT-connect und 80 % für RRT. Zudem nehmen die Pfadlänge und die Achsbewegung drastisch zu.

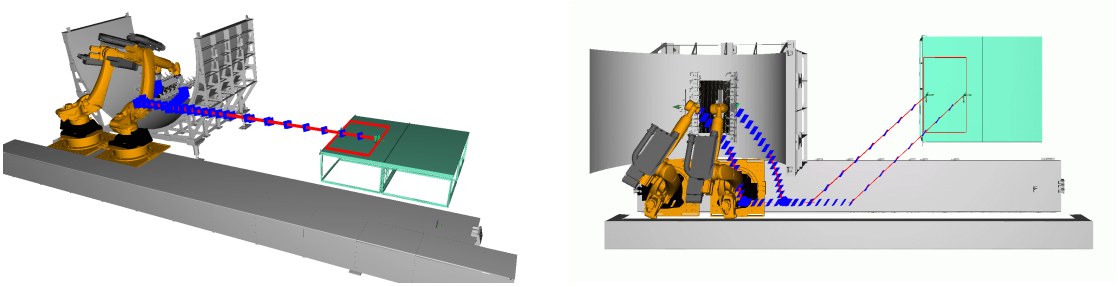
Vergleich der Redundanzauflösung der Linearachse

Zur Redundanzauflösung der externen Achse bei Sampling-basierten Verfahren stehen eine Heuristik und ein maschinelles Lernverfahren in KoKo zur Verfügung. Die EA ermitteln valide Stellungen der externen Achse während des Planungsprozesses. Zwar liefert die Heuristik stabile Ergebnisse, hat allerdings den Nachteil, dass sie für das jeweilige Szenario angepasst werden muss. Das maschinelle Lernverfahren ist theoretisch für verschiedene Szenarien anwendbar, da zunächst eine Trainingsphase erfolgt. Während der Simulationsversuche hat sich jedoch herausgestellt, dass dieses Verfahren nicht besonders stabil funktioniert. So konnte in Bezug auf RRT und RRT-connect in Kombination mit dem maschinellem Lernverfahren für die Ablage von Zuschnitt 3 keine Lösung gefunden werden. Erst durch das Hinzufügen von Stützpunkten (siehe 6.2) war es möglich, einen Pfad zu berechnen.

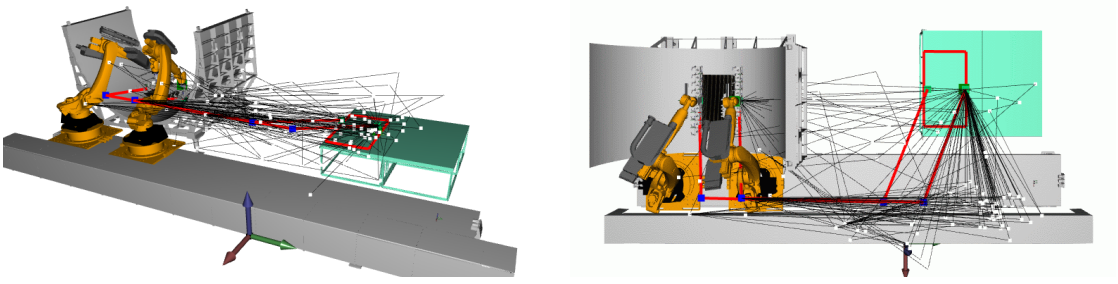
Abbildung 5.21 zeigt einen Vergleich der Bewegung der externen Achse mit den verschiedenen Verfahren. Beim maschinellen Lernverfahren ist die Lösung mit Stützpunkten zu sehen (siehe Abb. 5.21b). Die Bewegung mit dem Heuristik-Verfahren liefert ein sehr gutes Ergebnis (siehe Abb. 5.21a). Lediglich für R1 gibt es in der zweiten Hälfte der Bewegung einen kleinen Knick in der Bewegung. Das maschinelle Verfahren weist zahlreiche unnötige Bewegungen der Achsen zu sehen. Für einen industriellen Einsatz ist dieses Verfahren daher aktuell noch ungeeignet. Da sich der TCP in gleicher Geschwindigkeit weiterbewegen kann, wird die Taktzeit zwar nicht beeinflusst, aber der Energieverbrauch ist deutlich höher. Der EA liefert ein sehr gutes Ergebnis (siehe Abb. 5.21c). Die Achsen beider Roboter bewegen sich durchweg synchron; zudem es sind keinerlei unnötige Bewegungen zu erkennen.



(a) RRT ohne PV – Pfadlänge 10 493 mm, Achsbewegung 2339°, E1-Bewegung 7961°.



(b) RRT mit PV – Pfadlänge 7697 mm, Achsbewegung 1384°, E1-Bewegung 6474°.

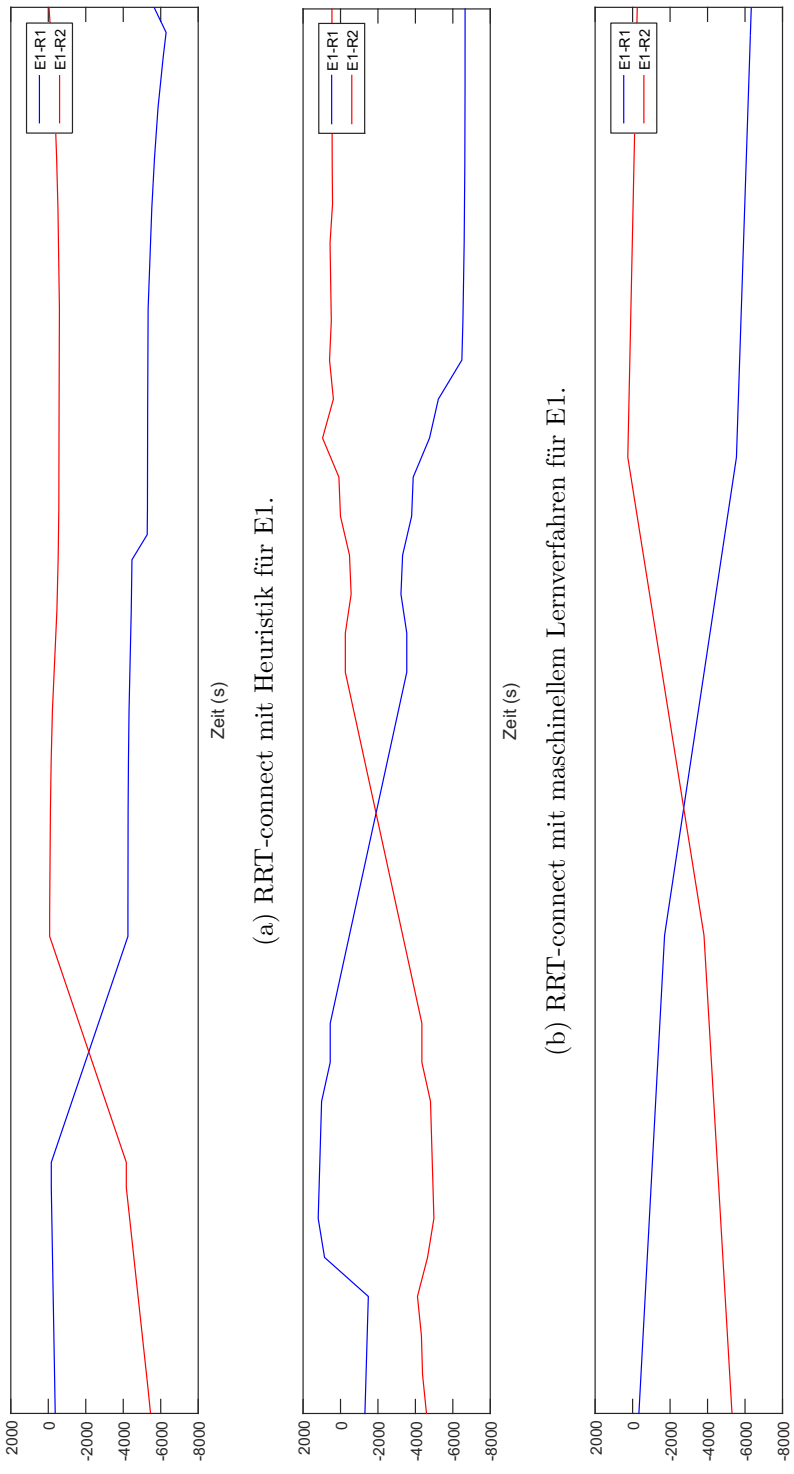


(c) IC=200, VP=2 – Pfadlänge 9108 mm, Achsbewegung 1970°, E1-Bewegung 6047°.

Abbildung 5.20: Screenshots der besten Ergebnisse alle Versuche im Demo-Panel-Szenario; Sampling-basierte Planer und EA.

Planer	Pfadlänge	Zeit	Achsbewegung
RRT ohne PV	deutlich längere Pfade als EA	mit und ohne PV die kürzesten Zeiten	deutlich höhere Achsbewegung sowohl für Roboterachsen als auch bei der externen Achse im Vergleich zu EA
RRT-connect ohne PV	die längsten Pfade	mit und ohne PV um etwa 30% längere Zeiten als bei RRT	siehe RRT ohne PV
RRT mit PV	neben RRT-connect die kürzesten Pfade	mit und ohne PV die kürzesten Zeiten	deutlich höhere Achsbewegung sowohl für Roboterachsen als auch externe Achse im Vergleich zu EA; Bewegung der Roboterachsen bei RRT und RRT-connect sehr ähnlich; Bewegung der externen Achse bei RRT und RRT-connect identisch aufgrund der Verwendung derselben Heuristik
RRT-connect mit PV	neben RRT die kürzesten Pfade	mit und ohne PV um etwa 30 % längere Zeiten als bei RRT	siehe RRT mit PV
EA IC=200, VP=2, Abbruch: 1200 s	längere Pfade als bei RRT und RRT-connect mit PV; jedoch kürzere Pfade im Vergleich zu ohne PV	etwa doppelt so lange Zeiten wie bei RRT	sowohl bei den Roboterachsen als auch bei der externen Achse die geringste Bewegung

Tabelle 5.8: Zusammenfassung der Planungsergebnisse für das Demo-Panel-Szenario.



(a) RRT-connect mit Heuristik für E1.

(b) RRT-connect mit maschinellem Lernverfahren für E1.

(c) Evolutionäre Algorithmen (EA).

Abbildung 5.21: Vergleich der Bewegung der externen Achse für die verschiedenen Verfahren zur Bestimmung einer validen Position.

Kapitel 6

Validierung des Systems anhand eines Flugzeugbauteils

In diesem Kapitel wird das in dieser Arbeit entwickelte KoKo-System an einem realen Bauteil validiert. Bei dem Anwendungsfall handelt es sich um das in Abschnitt 1.2 vorgestellte Demo-Panel Bauteil. Zunächst wird gezeigt, wie die Programmierung der kooperierenden Roboter für die Ablage durch Stand-der-Technik Verfahren erzeugt werden kann. Im zweiten Schritt werden die Bahnen, für die erste Sequence des Plybooks, automatisiert mit KoKo generiert. Zusätzlich wird die erreichte Ablage-Genauigkeit mit einem Lasertracker überprüft.

6.1 Programmierung durch Teach-in

Um eine generelle Machbarkeit des Preformings mit kooperierenden Robotern zu untersuchen, wurden die Roboter im ersten Schritt durch das Teach-in Verfahren programmiert. Dabei kam das Technologiepaket RoboTeam [155] von KUKA, welches die Programmierung synchroner Bewegungen von Roboter-Teams erlaubt, zum Einsatz; dargestellt wurde dies bereits in Abschnitt 2.7. Das System arbeitet nach dem Master/Slave-Prinzip, wobei ein Roboter als Master, die Bewegungen vorgibt und der oder die Slaves diesen folgen.

Um den Zuschnitt an einer definierten Stelle aufnehmen zu können, befinden sich am Aufnahmetisch Anschlagkanten, an die der Zuschnitt in der unteren rechten Ecke angelegt werden muss. Um die genaue Aufnahmeposition der Greifer für den Zuschnitt zu ermitteln, wurden zunächst die Breite $B_1 = 160\,\text{mm}$ und die Länge $L_1 = 1580\,\text{mm}$ der Greifer gemessen. Zudem war die Größe des Testzuschnittes mit der Breite $B_2 = 1220\,\text{mm}$ und der Länge $L_2 = 2000\,\text{mm}$ bekannt. Die Aufnahmepositionen wurde so definiert, dass der TCP des rechten Greifers auf $\text{TCP}_{R_1} = \left(\frac{B_2}{2} + \frac{L_1}{2}, \frac{B_1}{2}\right)$ und der des linken Greifers auf $\text{TCP}_{R_2} = \left(\frac{B_2}{2} + \frac{L_1}{2}, \frac{L_2}{2} - \frac{B_1}{2}\right)$ positioniert wird; dies zeigt Abb. 6.1, wobei die Aufnahmepunkte in der Skizze durch rote Punkte visualisiert sind.

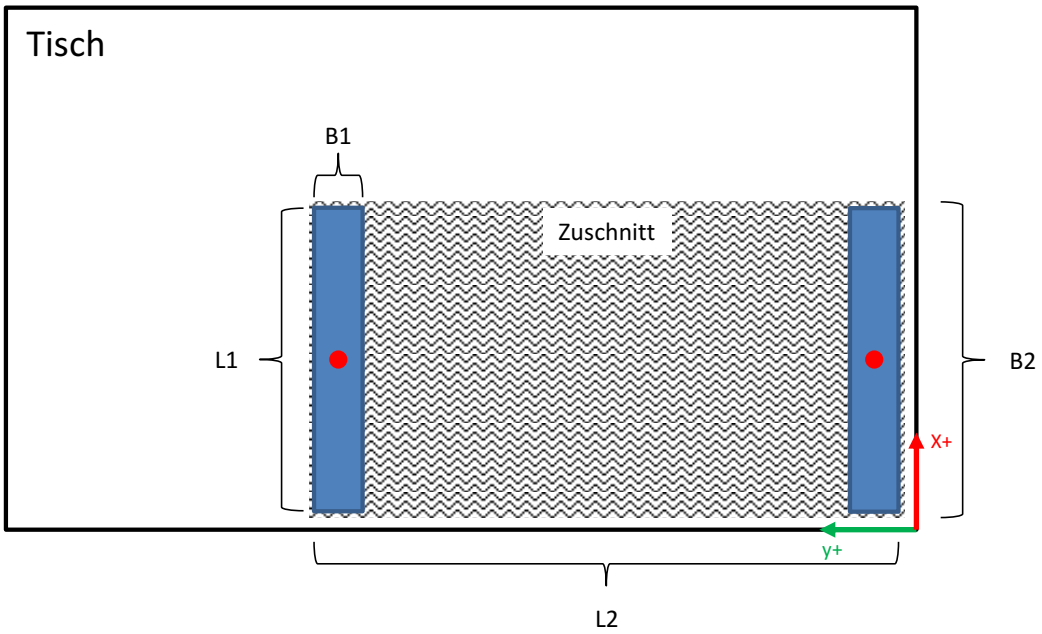


Abbildung 6.1: Definition der Aufnahmeposition des Zuschnittes auf dem Aufnahmetisch. In Blau die Streifengreifer, die roten Punkte definieren den TCP.

Die Aufnahmebewegung wurde durch ein parametrisches Programm generiert mit dem Ziel, eine Kettenlinie, wie in Abschnitt 4.2.7 behandelt, nachzubilden. Dazu wurde eine zweistufige Bewegung programmiert, deren Parameter variabel einstellbar sind. Die Bewegung setzt sich jeweils aus einem Winkel α um die X-Achse, einer Höhe Z und einem Versatz A zusammen. Abbildung 6.1 stellt die relevanten Parameter dar. Durch experimentelle Versuche konnte herausgefunden werden, dass für $\alpha_1 = 15, \alpha_2 = 20, Z_1 = 200, Z_2 = 100, A_1 = 20$ und $A_2 = 100$ sehr gut eine Kettenlinie angenähert werden kann. Quellcode 6.1 zeigt den Programmcode für das Aufnahmeprogramm. Um die Bewegungen der Roboter während der Aufnahme synchron zu halten, wurde der RoboTeam ProgSync verwendet. Dieser sorgt dafür, dass die Bewegungen von Master und Slave gleichzeitig starten. Da die Bewegungen beider Roboter absolut symmetrisch ablaufen und die Geschwindigkeit identisch ist, resultiert der ProgSync in einer synchronen Bewegung beider Roboter.

Nachdem die parametrisierte Aufnahmebewegung beendet ist, wird der RoboTeam-Geolink gestartet. Dieser bewirkt, dass der Slave stets den Bewegungen des Masters folgt. Da der Zuschnitt in der Kettenlinie gehalten werden soll, bietet sich der Geolink für den Transport optimal an. Mit diesem werden die Greifer über die Linearachse gefahren, danach zur Werkzeugform bewegt und schließlich in diese geschwenkt. Nach Erreichen der Mitte der Werkzeugform wurden die beiden Greifer im GeoLink über die Ablageposition bewegt und auf die Ablageposition herabgefahren. Zum Freifahren der Roboter wurde der GeoLink gelöst; die Roboter wurden nacheinander aus der Werkzeugform und danach zur Aufnahmeposition gefahren. Um während des Teach-in in der Werkzeugform die Ablageposition zu treffen, kam ein Laserprojektor zum Einsatz, dieser stellt heute in einer manuellen Produktion den Stand der Technik dar.

Durch die Programmierung des Prozesses mit dem Teach-in-Verfahren konnte erstmals gezeigt werden, dass der Transport von biegeschlaffen Halbzeugen mit kooperierenden Robotern möglich ist.

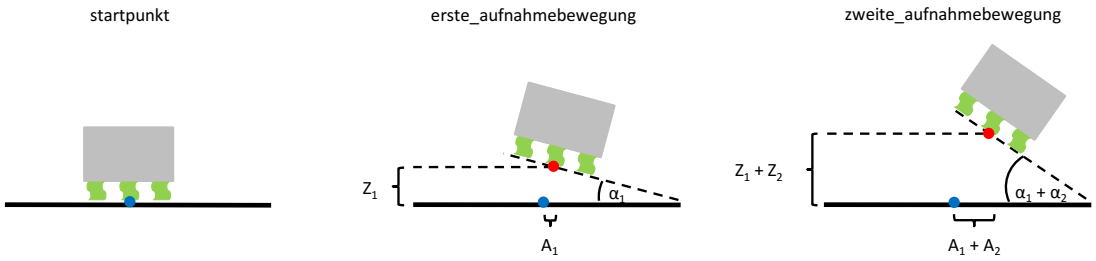


Abbildung 6.2: Von links nach rechts die Abfolge der Aufnahmebewegung. Blick auf die Längsrichtung des Greifers. Der rote Punkt skizziert den TCP, der blaue Punkt den Aufnahmepunkt.

Quellcode 6.1: KRL-Pseudocode des parametrischen Aufnahmeprogramms zur Nachbildung der Kettenlinie.

```
1  ; Variablendeklaration
2  DECL E6POS erste_aufnahmebewegung
3  DECL E6POS zweite_aufnahmebewegung
4
5  ; Winkel um den bei der Aufnahme gedreht wird
6  DECL REAL alpha_1 = 15
7  DECL REAL alpha_2 = 20
8
9  ; Strecke um die bei der Aufnahme Richtung Mittelpunkt gefahren werden soll
10 DECL REAL A_1 = 20
11 DECL REAL A_2 = 100
12
13 ; Strecke um die bei Aufnahme hochgefahren werden soll
14 DECL REAL Z_1 = 200
15 DECL REAL Z_2 = 100
16
17 ; 1. Aufnahmebewegung
18 erste_aufnahmebewegung = startpunkt
19 erste_aufnahmebewegung.z += Z_1
20 erste_aufnahmebewegung.y += -A_1
21 erste_aufnahmebewegung.b += -alpha_1
22 LIN erste_aufnahmebewegung
23 #ProgSync,"erste_aufnahmebewegung"
24
25 ; 2. Aufnahmebewegung
26 zweite_aufnahmebewegung = erste_aufnahmebewegung
27 zweite_aufnahmebewegung.z += Z_2
28 zweite_aufnahmebewegung.y += -A_2
29 zweite_aufnahmebewegung.b += -alpha_2
30 LIN zweite_aufnahmebewegung
31 #ProgSync,"zweite_aufnahmebewegung"
```

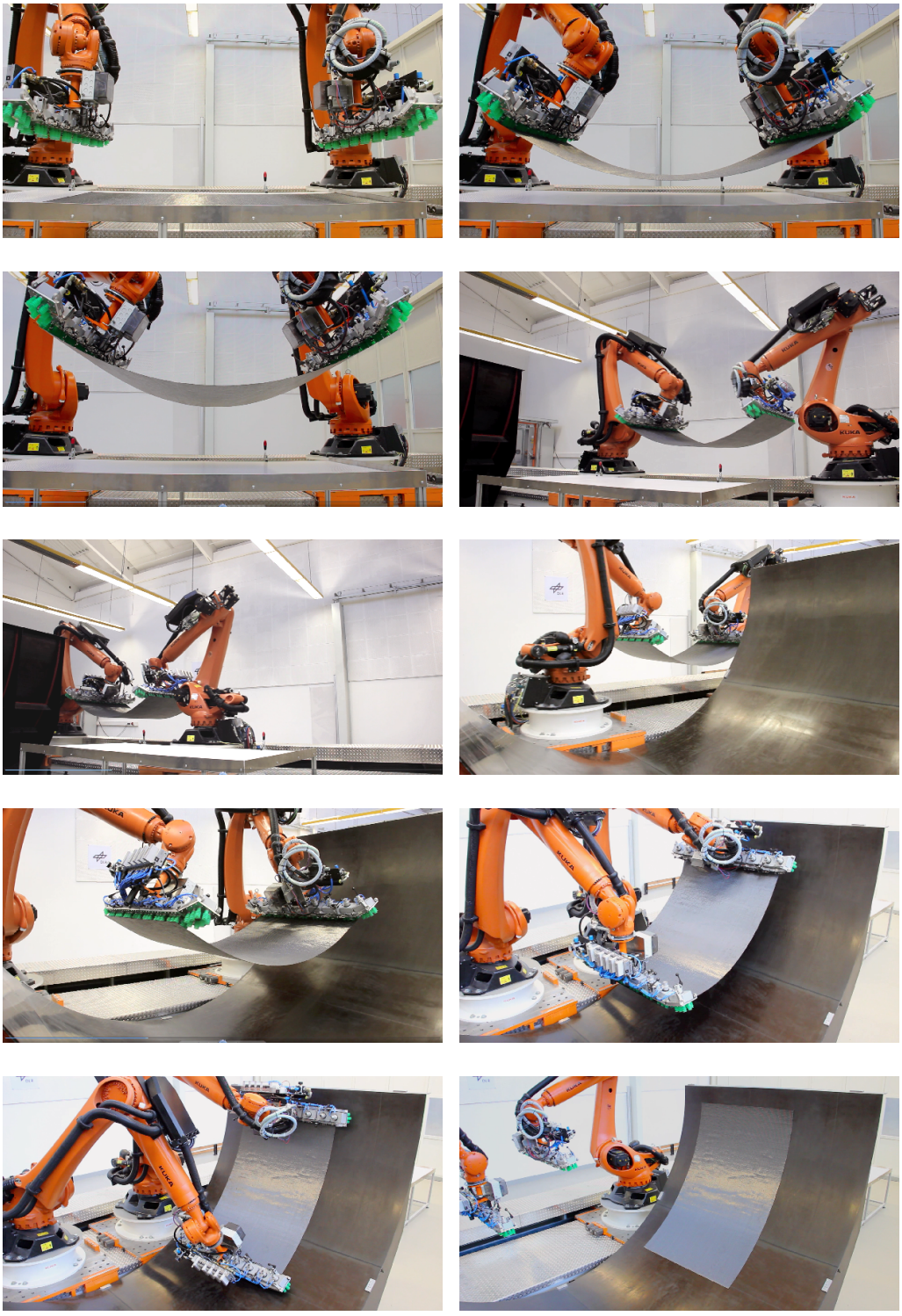


Abbildung 6.3: Fotosequenz des durch Teach-in in Betrieb genommenen Prozesses.

6.2 Programmierung durch automatisches Planungssystem

Nachdem die Planungsalgorithmen ausgiebig in der Simulation validiert wurden, galt es das KoKo-Planungssystem in einem realen Produktionsszenario zu testen. Als Planungsalgorithmus fiel die Wahl auf den RRT-connect, da dieser zuverlässig kurze Pfade liefert und zudem neben RRT die kürzesten Planungszeiten vorzuweisen hat. Dies ist in einem industriellen Kontext, bei dem Taktzeiten entscheidend sind, ein klarer Vorteil. Des Weiteren wurden Stützpunkte für die Planung vorgegeben; diese beschleunigen den Prozess erheblich. Der erste Punkt liegt in Höhe des Aufnahmetisches, der zweite in Höhe der Werkzeugform. Im Schnitt dauert die Berechnung mit dem RRT-connect und dem Heuristik-Verfahren zur Bestimmung der Position der externe Achse eine Minute.

- Stützpunkt 1 (SP1): $X = -1300, Y = 790, Z = 1600, A = 180, B = 0, C = -13,5$
- Stützpunkt 2 (SP2): $X = -1300, Y = -3300, Z = 1600, A = 180, B = 0, C = -13,5$

Zur Aufnahme und Ablage wurde jeweils ein BASE-Koordinatensystem für Tisch und Werkzeugform eingemessen. Die Planung und Ausführung des Transportes mit KoKo erfolgt im Weltkoordinatensystem. In Abschnitt 4.4 wurden bereits die am Preforming-System beteiligten Module und deren Kommunikation untereinander beschrieben. Obwohl das System sehr komplex ist, kann es nach dem Einrichten durch eine Person bedient werden. Das MES, CDS und KoKo wurden unabhängig voneinander entwickelt und müssen im aktuell bestehenden Entwicklungsstatus separat gestartet werden. Dazu muss zunächst auf beiden Robotern ein Programm ausgewählt werden, welches EthernetKRL auf diesen startet. In einem nächsten Schritt wird dann die Anwendung KoKo ausgeführt. Diese lädt automatisch die entsprechende Szene und wartet auf Anweisungen durch das MES. Im nächsten Schritt wird die GUI des Zuschnitt-Erkennungsprogramms, die aus mehreren Fenstern besteht, gestartet; dies zeigt Abb. 6.4. Das obere linke Fenster stellt die Buttons für den Ablageprozess zu Verfügung. Das Fenster darunter dient für Einstellungen der Zuschnitterkennung wie z. B. den Kamera-Trigger. Darunter wird das Template des aktuell erkannten Zuschnitts mit den aus dem JDF ausgelesenen Grippoints in Grün angezeigt. Das große Fenster auf der rechten Seite zeigt das Foto des aktuellen Zuschnitts; die detektierte Kontur in Grün und das Zentrum des aktuellen Zuschnitts in Form eines blauen Kreises. Im Fenster ganz unten sind Statusmeldungen zu sehen. Nachdem die GUI des CDS gestartet wurde, kann über den Button *Connect KRC* die Verbindung zu den Robotern und KoKo gestartet werden. Danach wird über den Button *Read Plybook* das JDF eingelesen. Über *Select Ply* wird der nächste zu legende Zuschnitt ausgewählt. DETPOS bewegt den Roboter R2 in eine sichere Position und R1 bewegt sich in die Position über dem Tisch, um zu fotografieren. Danach wird durch *DETECT* die Position des Zuschnitts auf dem Tisch erkannt. *GRIP* bewegt die beiden Roboter zunächst in eine Vorposition über den Zuschnitt und greift diesen anschließend an der detektierten Position. Durch den *TRANSFER* wird Pfadplanungsanfrage an KoKo gesendet. Nachdem ein Ergebnis zurückgeliefert wurde, startet automatisch der Transport des Zuschnitts in eine Vorposition knapp oberhalb der Ablageposition. *DROP* bewirkt dann das Ablegen des Zuschnittes. Durch erneutes Drücken von *TRANSFER* bewegen sich die Roboter von der Ablageposition zurück in die Aufnahmeposition. Soll das System im Automatik-Modus betrieben werden, müssen die Systeme nur einmal verbunden werden und die Schritte von *DETPOS* bis zum zweiten *TRANSFER* entfallen.

Abb. 6.7 stelt eine Fotosequenz der Ablage von Zuschnitt 5. Im ersten Bild ist die Startposition, in der beide Greifer oberhalb des Tisches positioniert sind, zu sehen. Zunächst bewegt sich R2 in eine sichere Position und R1 nimmt die Detektionsposition in etwa 1 m Abstand zum Tisch ein. Diese beiden Positionen sind die einzigen, die für die Ablage manuell einprogrammiert wurden.

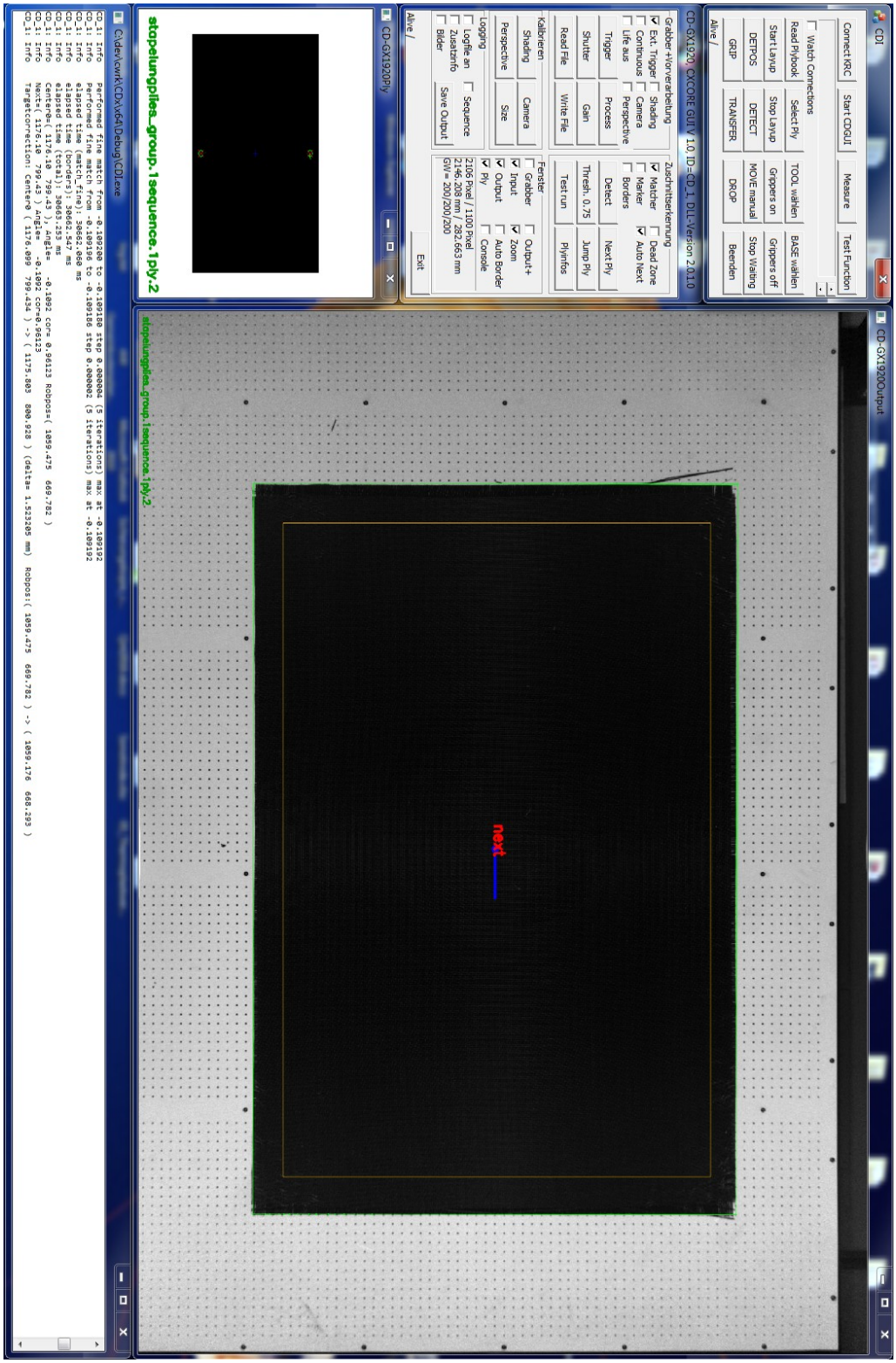


Abbildung 6.4: Screenshot der GUI des CDS.

Sämtliche weitere wurden von KoKo berechnet. In Abb. 6.6 sind die Pfadplanungsergebnisse der kompletten ersten Sequenz in KoKo dargestellt; von oben startend nach unten Zuschnitt 1 bis 5. Durch die komplett automatische Ablage der ersten Sequenz, die Abb. 6.5 darstellt, konnte gezeigt werden, dass das KoKo-System erfolgreich in Produktionsprozessen eingesetzt werden kann. Es wurden für alle fünf Zuschnitte kollisionsfreie Pfade gefunden und mit einer ausreichenden Genauigkeit abgelegt. Mit der in Abschnitt 6.1 beschriebenen manuellen Programmierung hätte ein Einrichten des Prozesses mehrere Tage pro Zuschnitt gedauert. Mit KoKo konnten diese in weniger als zwei Stunden abgelegt werden. Dabei dauerte die Berechnung pro Zuschnitt etwa eine Minute und die Aufnahme und der Transport etwa zwei Minuten. Der Rest der Zeit war nötig für vor- und nachbereitende Aufgaben, wie bspw. das Legen des Zuschnittes auf den Tisch sowie die Video-Dokumentation. Dabei wurde der komplette Versuch und somit auch die Bedienung von CDS und KoKo-System von nur einer Person durchgeführt.

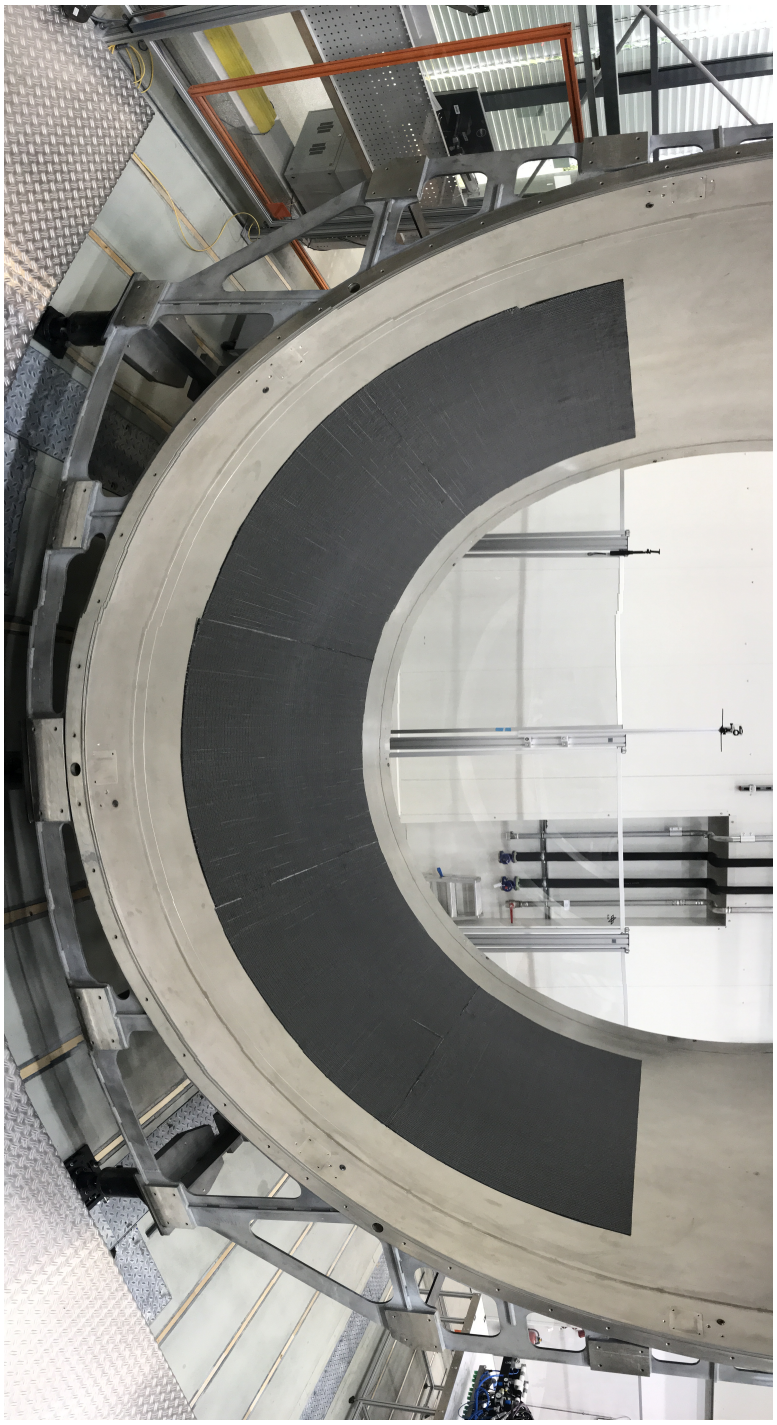


Abbildung 6.5: Ergebnis der Ablage der ersten Sequenz des Demo-Panel-Testbauteils.

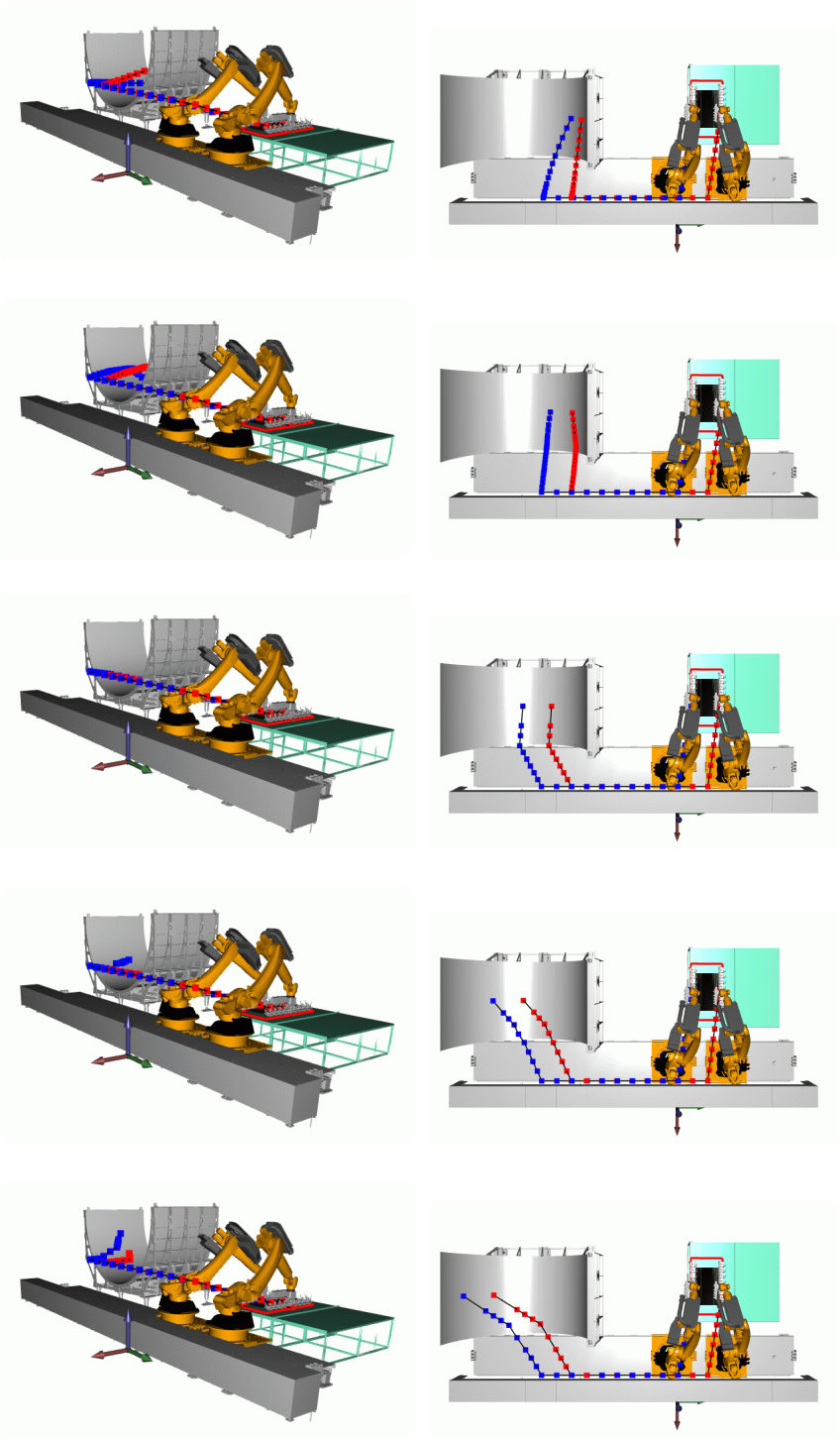


Abbildung 6.6: Planungsergebnisse mit dem RRT-connect Algorithmus. Von oben nach unten Zuschnitte 1 bis 5. Einstellungen: Auflösung: 0,001 mit Stützpunkten und mit Constraints.

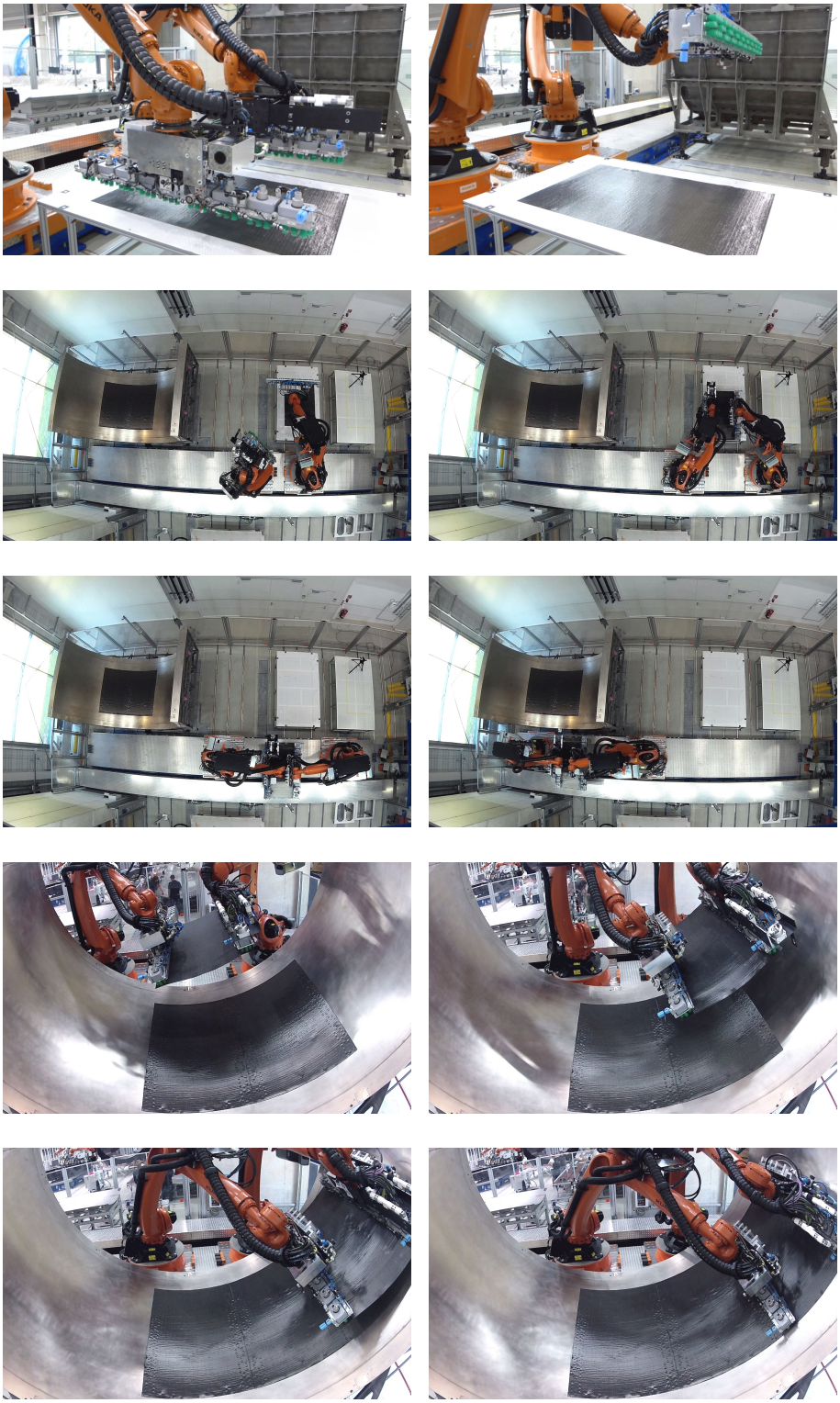


Abbildung 6.7: Fotosequenz der Aufnahme, des Transportes und der Ablage von Zuschnitt 5.

6.3 Überprüfung der Genauigkeit des Systems

6.3.1 Vermessung der Positionierung der Roboter zueinander

Um die Genauigkeit der Positionierung der Roboter zueinander zu bestimmen, wurde diese mithilfe von zwei Lasertrackern ebenfalls in der TEZ vermessen [150]. Jeder Roboter wurde in der Nähe des TCP mit einer Leica T-Mac ausgestattet; dies zeigt Abb. 6.8. In Kombination mit einem Lasertracker konnte damit die 6-DOF-Position der Roboter bei einer maximalen Fehlergrenze von 10 µm für die Entfernung und 15 µm ± 6 µm für die Orientierung bestimmt werden. Abbildung 6.9 zeigt den Versuchsaufbau in der TEZ. R1 und R2 deuten die Roboter auf der Linearachse an. AT1 und AT2 stellen die Position der Lasertracker dar. Die Controller der beiden Tracker liefern die aktuelle Position über einen EtherCAT-Feldbus mit einer Messrate von 1 kHz. Die Messdaten werden von einem Rechner gesammelt und in einem gemeinsamen Datensatz gespeichert. Um sicherzustellen, dass die Messdaten der beiden Tracker zeitlich zusammenpassen, wurden die Uhren der beiden Controller über die Distributed Clocks von EtherCAT abgeglichen, die eine Synchronisierung von ≤ 1 µs [156, S.20] ermöglichen.

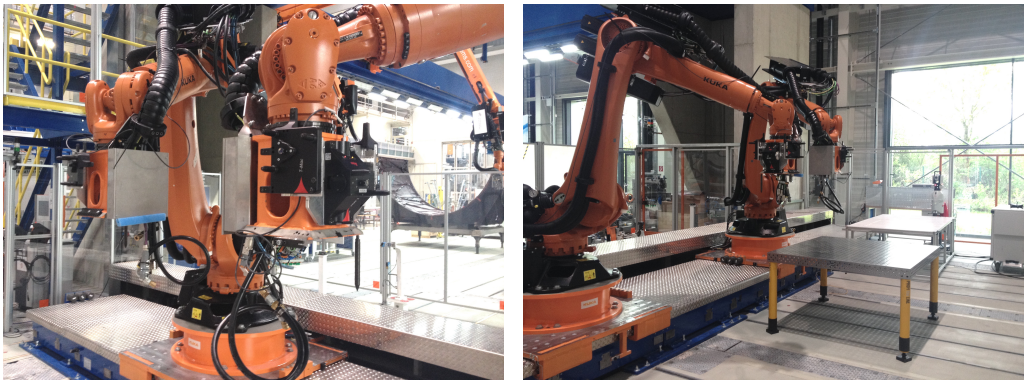


Abbildung 6.8: T-Mac ist am Roboter montiert; Lasertracker befindet sich im Hintergrund.

Zunächst wurde ein Satz an Bewegungen für R1 definiert und abgefahren. R2 wurde so konfiguriert, dass er permanent den Bewegungen von R1 folgte. Diese Art der Koppelung ist vergleichbar mit dem Geolink der KUKA-Roboter. Für den Test hat R1 die folgenden Bewegungen, jeweils bezogen auf die Mess-Basis, abgefahren. Die Linearachse der Roboter wurde während der Versuche nicht bewegt.

1. Linearbewegung 500 mm in Z-Richtung.
2. Linearbewegung 1000 mm in positive Y-Richtung und danach 1000 mm in negative Y-Richtung.
3. Linearbewegung 500 mm in positive X-Richtung und danach 500 mm in negative X-Richtung.

Abbildungen 6.10a bis 6.10c zeigen die gemessenen Positionen während der Messung für die X-, Y- und Z-Koordinate. Alle Bewegungen wurden mit einer Geschwindigkeit von $0.5\,\mathrm{m\,s^{-1}}$ und einer Beschleunigung von $3\,\mathrm{m/s^2}$ durchgeführt. Die Robotics API benutzt ein „Doppel-S“-Geschwindigkeits-Profil für lineare Bewegungen, wie in [157, Sektion. 3.4] erläutert, Diese

verhindern den sogenannten Ruck, im englischen Jerk, der bei plötzlichem Beschleunigen oder Bremsen auftreten kann.

Da während der kompletten Messung die Position beider Roboter im Messkoordinatensystem gemessen wurde, kann im Nachhinein die Abweichung der vorher definierten Distanz zwischen beiden Robotern ermittelt werden. Abbildung 6.10 stellt diese aufgeteilt in die X-, Y- und Z-Komponente dar. Die Abweichung wurde bestimmt, indem die absolute Position der beiden Roboter voneinander subtrahiert und mit der initialen Entfernung verglichen wurde. Man kann sehen, dass sich die Abweichung meistens im Rahmen von $\pm 1\,\mathrm{mm}$ befindet. Zudem gibt es einige Spitzen von $2\,\mathrm{mm}$. Etwas erhöhte Abweichungen treten hauptsächlich an den Stellen auf, an denen die Geschwindigkeit oder die Beschleunigung der Roboter geändert wurde. Die Kommunikation des RCC ist über die RSI-Schnittstelle der KUKA-Steuerung realisiert, Fahrbefehle können maximal alle $4\,\mathrm{ms}$ abgerufen und verarbeitet werden. Da es nicht möglich ist, den Empfang der Befehle über dieses Interface auf beiden Robotern zu synchronisieren, können sich allein durch die minimalen zeitlichen Unterschiede bei der Verarbeitung je nach Geschwindigkeit Ungenauigkeiten im Millimeter-Bereich ergeben. Das zu transportierende CFK-Material ist ähnlich flexibel wie ein Textil; daher stellt diese minimale Abweichung der beiden Roboter für die Ablage keinerlei Problem dar.

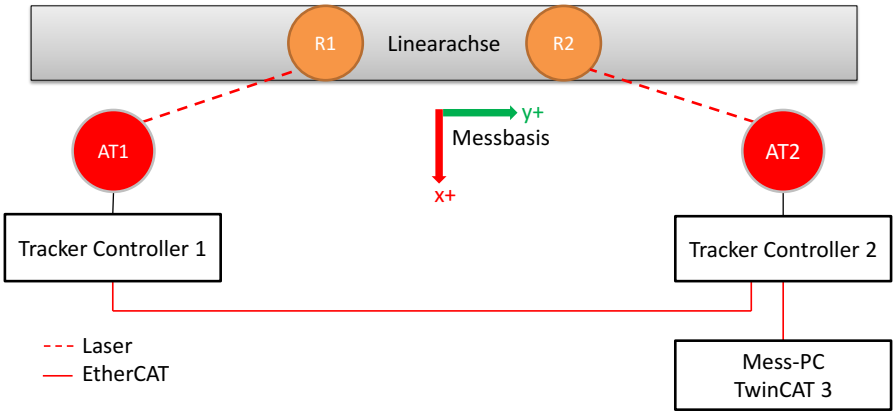
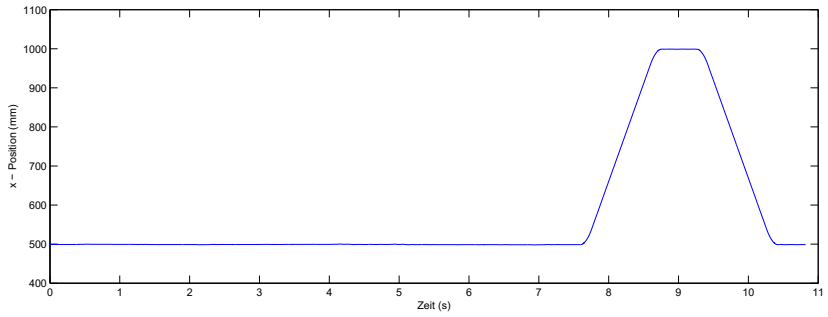
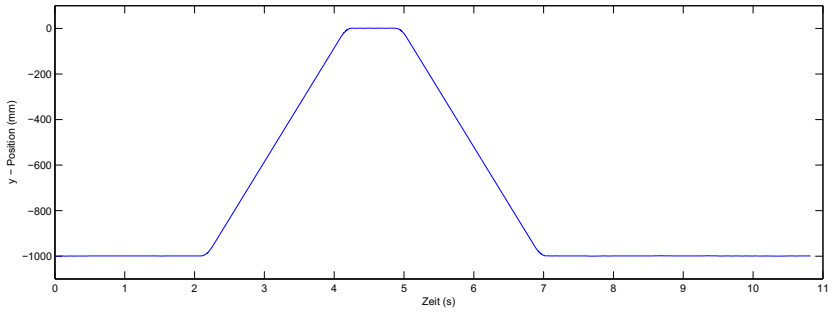


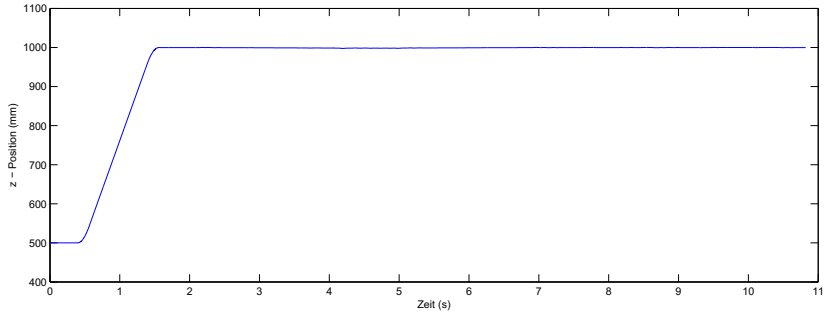
Abbildung 6.9: Versuchsaufbau zur Vermessung der relativen Genauigkeit zwischen Roboter R1 und Roboter R2. AT1 und AT2 bezeichnen die Position der Lasertracker.



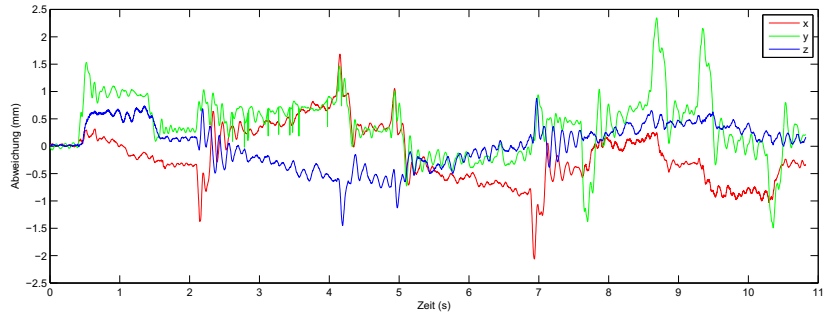
(a) Aufzeichnung der X-Koordinate von R1 in der Mess-Basis.



(b) Aufzeichnung der Y-Koordinate von R1 in der Mess-Basis.



(c) Aufzeichnung der Z-Koordinate von R1 in der Mess-Basis.



(d) Abweichung der Roboter Flange von ihrer vorgegebener Entfernung.

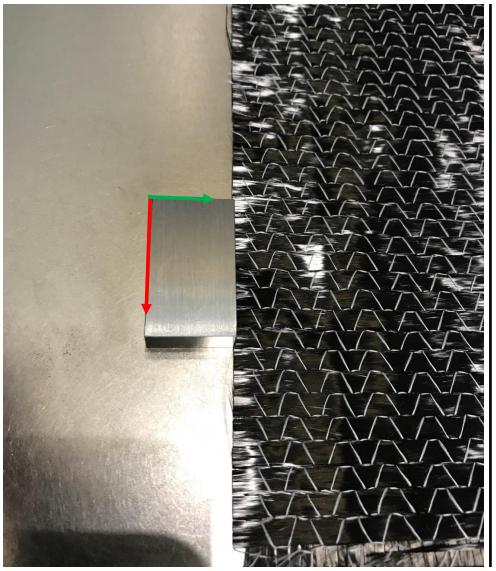
Abbildung 6.10: Messung der Positionierung kooperierender Roboter zueinander.

6.3.2 Vermessung der Ablagegenauigkeit für das Demo-Panel

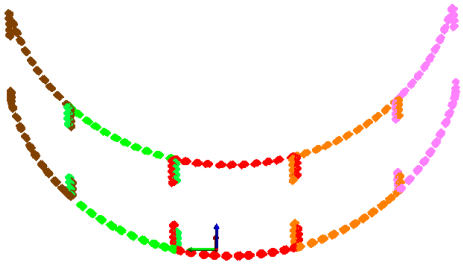
Um eine Aussage über die Genauigkeit des Preforming-Systems zu erhalten, wurden die Ablagepositionen der Zuschnitte der ersten Sequence des Plybooks vom Demo-Panel Bauteil, vermessen. Als Messsystem wurde ebenfalls der Lasertracker Leica AT901 in Verbindung mit einer T-Probe verwendet. Bei der T-Probe handelt es sich um ein Handgerät, mit dem sehr einfach einzelne Punkte vermessen werden können. An der T-Probe befindet sich an einem kleinen Stab eine 3 mm große Messkugel, die in Abb. 6.11a dargestellt ist. Mit dieser wurde der zu vermessende Punkt angetastet; durch einen Knopf kann dann die Messung ausgelöst werden. Da es schwierig ist, über diese Kugel die sehr weiche Kante des trockenen Material definiert anzutasten, wurde ein Messstein als Hilfsmittel eingesetzt. Dessen Größe war genau bekannt und konnte bei der Auswertung des Soll-/Ist-Vergleiches berücksichtigt werden. Der Stein, wie in Abb. 6.11b zu sehen, wurde im Bereich der Ecken der Zuschnitte viermal mit einem Abstand von 10 cm exakt an der Kante des Materials positioniert. Da dieser magnetisch war, hatte er auf der Invar-Werkzeugform sehr gute Haftung und konnte mit der T-Probe vermessen werden. Als Messsoftware wurde SpatialAnalyzer [158] der Firma New River Kinematics verwendet. Um beim Soll-/Ist-Vergleich die Position der Kante zu ermitteln, musste noch die Breite des Messteins abgezogen werden.

Dies war möglich, indem in der Messsoftware ein lokales Koordinatensystem konstruiert wurde, bei dem eine Achse exakt entlang der Breite des Steins gezeigt hat (siehe grüne Achse in Abb. 6.11a). Das Koordinatensystem bildete den Schnittpunkt der oberen, äußeren und seitlichen Fläche des quaderförmigen Messsteins. Die Flächen ließen sich gut mit der T-Probe abtasten; in einem zweiten Schritt konnte jeweils eine Ebene durch die Punkte konstruiert werden. Abbildung 6.11d zeigt das Ergebnis im Spatial Analyzer. In Gelb die Messpunkte und in Rot, Blau und Grün die drei Ebenen. Mithilfe des konstruierten Koordinatensystems konnten die Messpunkte, die sich auf der parallel zur Zuschnittskante liegenden Messsteinkante befinden, um den Offset 20 mm verschoben werden. Das vorher konstruierte Koordinatensystem konnte zudem zur Bestimmung der Abweichung von Kanten- zu der Sollposition genutzt werden. Hierzu wurde ein Lot auf den Sollkantenverlauf gelegt und in derselben Koordinatenachse, mit der der Offset des Messteins abgezogen wurde, die Differenz aus Soll- und Istposition bestimmt. In Abb. 6.12 sind die insgesamt 192 Messpunkte in Form von Vektorpfeilen eingezeichnet. Die Länge der Pfeile sind auf $[-1, 1]$ normiert, in Rot eine negative und in Blau eine positive Abweichung. Es ist deutlich zu erkennen, dass die Zuschnitte 3, 4 und 5 um etwa 1 mm in negativer Y-Richtung in Richtung Zuschnitt 3 verschoben sind. Ebenso sind die Zuschnitte 1 und 2 in positiver Y-Richtung in Richtung Zuschnitt 3 verschoben. Des Weiteren sind alle drei Zuschnitte um etwa 0,5–1 mm verdreht. Die Beobachtung der Verschiebung der Zuschnitte zueinander macht sich zudem beim Abstand zwischen diesen bemerkbar. Es ist eine durchschnittliche Überlappung von 1 mm festzustellen. In Abb. 6.13 ist der Abstand zwischen den Zuschnitten eingezeichnet. Diese minimale Überlappung war allerdings nur nach der Ablage wirklich wahrnehmbar. Sobald die Zuschnitte kompaktiert wurden, war mit dem bloßen Auge kein Überlapp mehr erkennbar. Laut Datenblatt [159, S.102] haben die verwendeten Industrieroboter eine Wiederholgenauigkeit von ±0,06 mm. Die Absolutgenauigkeit liegt eher im Millimeter-Bereich und kann durch den Einsatz einer Linearachse noch größer ausfallen. Da die Linearachse etwa 7 m lang ist, können minimale Winkelfehler beim Einmessen bereits großen Einfluss haben. Daher wurde, um diese Fehler zu kompensieren und die Genauigkeit des Gesamtsystems zu erhöhen, sowohl der Aufnahmetisch als auch die Ablageform als BASE eingemessen. Lediglich der Transport des Zuschnittes zwischen Tisch und Werkzeugform erfolgte im Weltkoordinatensystem. In Summe ist die Gesamtgenauigkeit sehr gut und kann als ausreichend für diesen Fertigungsprozess bewertet werden.

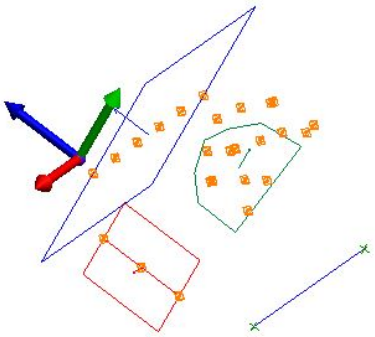
(a) T-Probe.



(b) Messstein.



(c) Darstellung der Zuschnitte 1 (links) bis 5 (rechts).



(d) Konstruierte Ebenen.

Abbildung 6.11: T-Probe, Messstein mit Koordinatensystem, Visualisierung der Messpunkte in Spatial Analyzer und konstruierte Ebenen sowie Koordinatensystem.

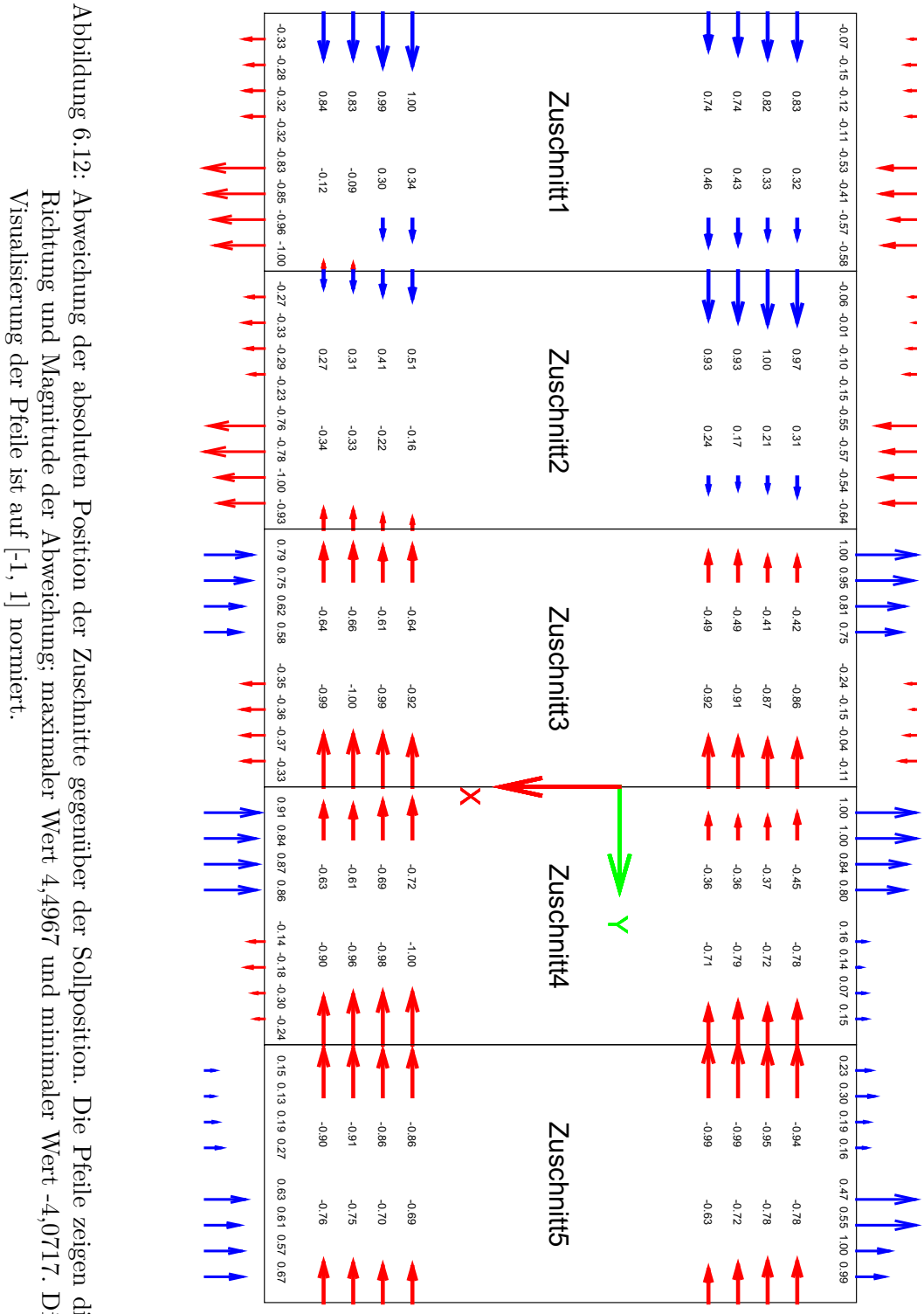


Abbildung 6.12: Abweichung der absoluten Position der Zuschnitte gegenüber der Sollposition. Die Pfeile zeigen die Richtung und Magnitude der Abweichung; maximaler Wert 4,4967 und minimaler Wert -4,0717. Die Visualisierung der Pfeile ist auf [-1, 1] normiert.

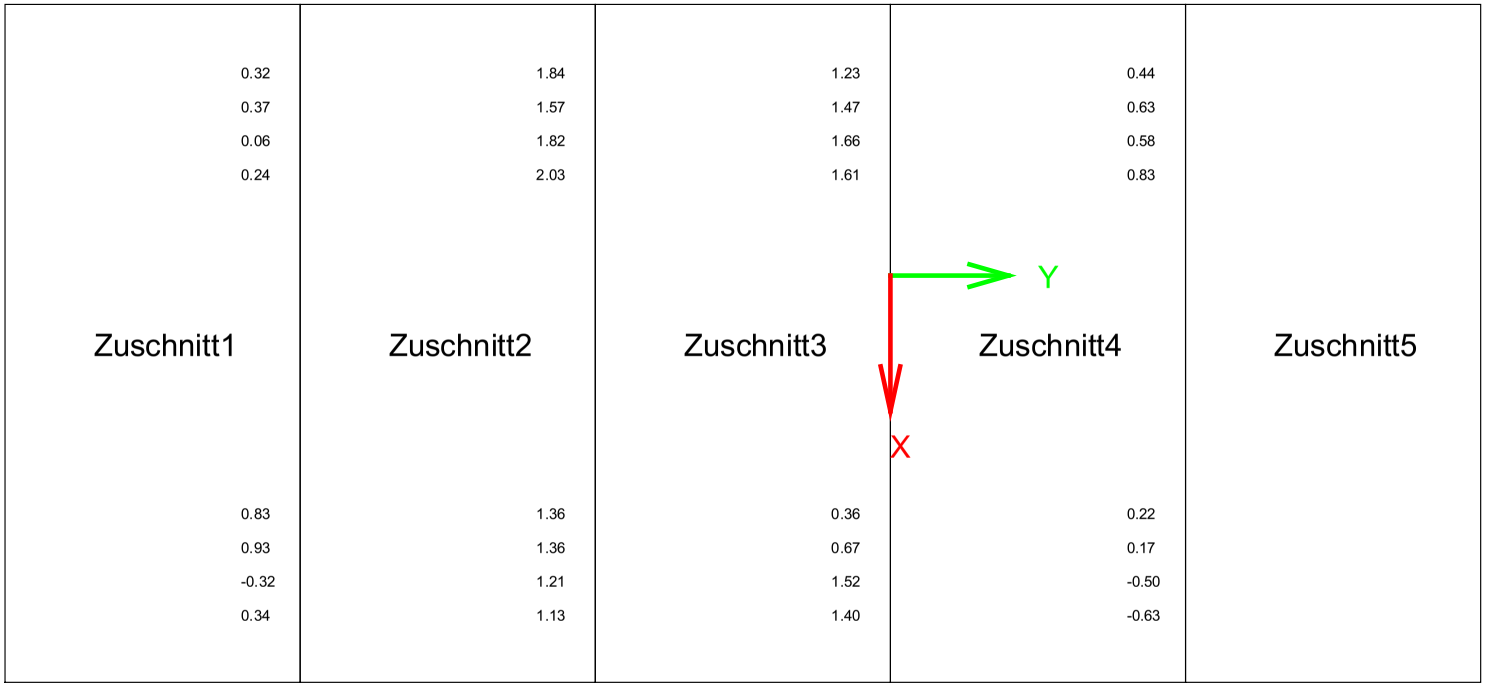


Abbildung 6.13: Abstand bzw. Überlapp zwischen den Zuschnitten; maximaler Wert 2,0307 mm und minimaler Wert -0,6320 mm.

Kapitel 7

Zusammenfassung und Ausblick

Gegenwärtig werden Industrieroboter eingesetzt, um einzelne Aufgaben einer Fertigung, wie z. B. Schweißen, Palettieren usw., zu automatisieren. Nach wie vor müssen sie jedoch manuell programmiert werden; dies beschränkt ihren Einsatz sehr stark auf einzelne Aufgaben. Die Hoffnung für die Zukunft besteht darin, dass Industrieroboter wesentlich flexibler für deutlich ein breiteres Aufgabenspektrum verwendet werden können. Bewegungen von Robotern werden heute allerdings noch bis auf wenige Millimeter genau entweder durch teach-in oder durch ein OLP-System programmiert. In dem in dieser Arbeit vorgestellten Flugzeugrumpf-Produktionsszenario hat sich gezeigt, dass dies im Hinblick auf eine wirtschaftliche Automatisierung nicht zielführend ist.

Als Lösung für dieses Problem wurde das KoKo-System entwickelt und erfolgreich getestet. Dieses erlaubt es, die Bahnplanung von Industrierobotern in Fertigungsprozessen zu automatisieren. Um das KoKo-System einsetzen zu können, muss zunächst wie bei einer herkömmlichen Off-line Programmierung die Roboteranlage in der Simulation modelliert werden. Des Weiteren ist eine Beschreibung des Prozesses, in der die Start- und Zielpunkte hinterlegt sind, erforderlich. Anhand dieser Informationen berechnet das KoKo-System automatisch kollisionsfreie Pfade, die je nach Roboterhersteller entweder direkt an den Roboter gesendet oder in ein entsprechendes Format exportiert werden können. Zudem wurde im Rahmen dieser Arbeit gezeigt, dass durch den Einsatz des automatischen Planungssystems eine enorme zeitliche Einsparung gegenüber einer herkömmlichen Programmierung erreicht werden kann. Ein weiterer Vorteil des KoKo-Systems besteht darin, dass dynamische Prozesse ebenfalls abgedeckt werden können. Szenarien, die anhand von Sensordaten, wie z. B. einer Kamera, für jedes Bauteil unterschiedliche Start- und Zielpositionen liefern und KoKo berechnet entsprechend kollisionsfreie Bahnen, sind denkbar.

Ferner wurden im DLR Augsburg bis heute bereits weitere Anwendungen für den Einsatz von KoKo entwickelt. Ein Beispiel ist ein System, welches automatisiert kleine Thermoplast-Zuschnitte zu einer Preform legt. Deren Anlieferung in die Roboterzelle erfolgt mit einem Schubladenspeicher, der acht Schubladen umfasst. Aus diesem sollen die Zuschnitte gegriffen werden, nachdem eine Kamera ihre Position bestimmt hat. Liegen die Zuschnitte zu nah an den Rändern der Schubladen, kann es vorkommen, dass der Endeffektor kollidiert. Um dies zu verhindern, gibt es bei der

aktuellen Umsetzung des Prozesses eine genaue Vorgabe, wo und mit welcher Orientierung die Zuschnitte in die Schubladen gelegt werden müssen. Dieser Prozess ist allerdings sehr aufwendig; Fehler sind vorprogrammiert. Zusätzlich werden sehr viele Zwischenpunkte definiert, um die komplexen Bewegungsabläufe durch vorgefertigte Bewegungsmodule zu realisieren. Hier kann die Einbindung von KoKo einen enormen Mehrwert generieren, wie in Abb. 7.1 verdeutlicht. Die beiden Bilder stellen bspw. den Transport eines Zuschnittes aus den Schubladen drei und fünf dar.

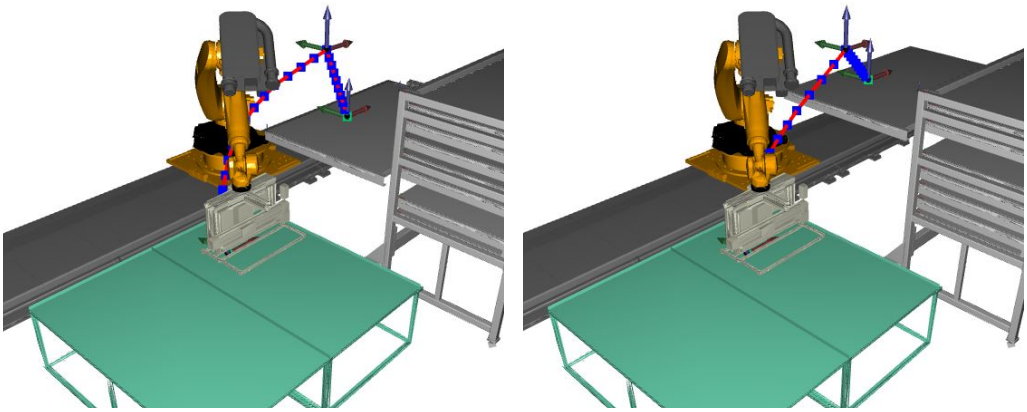


Abbildung 7.1: Weiteres Einsatzszenario für KoKo in der CFK-Produktion. Zuschnitte, die beliebig in verschiedenen Schubladen liegen, werden automatisch gegriffen und zu einem Ablagetisch transportiert (Screenshot aus KoKo).

Wie bereits erwähnt, muss für die Benutzung des KoKo-Systems zunächst die Umgebung in der Roboterzelle modelliert werden. Um zukünftig den Einrichtprozess zu beschleunigen, ist es denkbar, mit dem KoKo-System ein automatisches Vermessungssystem zu entwickeln. Dazu wird eine Tiefenkamera wie z. B. die Microsoft Kinect an einem Endeffektor des Roboters angebracht. Ausgehend von einem kollisionsfreien Startpunkt kann der Roboter mithilfe der Sensorinformationen selbstständig explorieren. Dazu vermisst der Roboter durch Rotationsbewegungen zunächst an seiner Startposition die Umgebung. Die Kinect erlaubt hier eine Messreichweite von circa 2 m. Nach und nach kann ausgehend von der Startposition in kollisionsfreie Gegenden exploriert werden; in diesen wird dann der Vermessungsschritt wiederholt. Bewegungen zu neuen Messpunkten werden jedes Mal vom Planungsmodul aus KoKo realisiert. Am Ende der Exploration können aus den Punktewolken automatisch 3D-Objekte, die bei der Pfadplanung berücksichtigt werden, generiert werden. Hierzu wurden bereits erste vielversprechende Versuche durchgeführt. Abbildung 7.2 zeigt die Vermessung der Umgebung in der TEZ mit einem Kinect-Sensor. Die Messpunkte sind in Rot dargestellt. Zur Verdeutlichung wurde die Kontur des Modells der Werkzeugform in Grau mit eingeblendet. Es lassen sich eindeutig sowohl die Linearachse als auch der zweite Roboter erkennen. Die Kinect ist ein sehr kostengünstiger Sensor und könnte durch einen besser auflösenden industriellen ersetzt werden.

Für den Einsatz von KoKo in einem vollautomatischen industriellen Prozess könnten noch weitere Verbesserungen vorgenommen werden. Aktuell ist die Kollisionserkennung auf lediglich einen Prozessorkern beschränkt. Sobald diese parallelisiert werden würde, kann auch die Bahnplanung

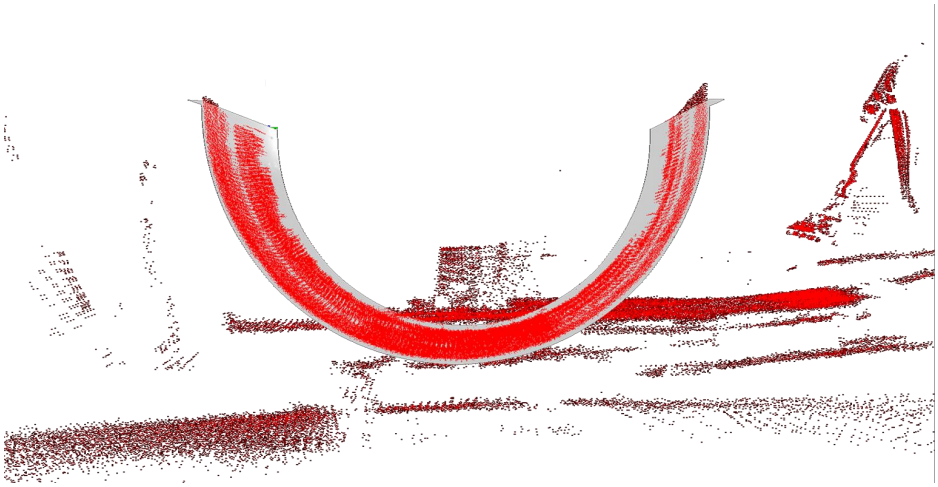


Abbildung 7.2: Scannen der Umgebung mit dem Kinect-Sensor.

deutlich beschleunigt werden; bei Prozessen, bei denen die Taktzeit relevant ist, kann dies von entscheidender Bedeutung sein.

Auf dem Weg zur Industrie 4.0 soll das KoKo-System zukünftig beim DLR Augsburg als Dienst in der Produktionshalle angeboten werden. Dazu werden alle Roboteranlagen mit OLE for Process Control (OPC) Unified Architecture (UA), einem industriellen Machine-to-Machine-Protokoll, vernetzt. KoKo soll in diesem System als eigenständiger Service, dienen, der von allen Robotern abgefragt werden kann. Sobald sich ein Roboter bewegen soll, werden Start- und Zielpunkt übertragen und KoKo berechnet den Pfad. Zusätzlich könnte das System Augmented-Reality-Anwendung verbunden werden, in der ein Bediener in einer virtuellen Welt einen Punkt vorgibt und KoKo den kürzesten Weg dorthin berechnet, weiterentwickelt werden. Dadurch kann selbst ein Teach-in-Prozess enorm beschleunigt werden, da es entfällt, den Roboter manuell entweder kartesisch oder achsweise an die gewünschten Positionen zu fahren. Somit kann KoKo auf dem Weg in Richtung umfassender Industrie 4.0 abschließend eine nicht zu unterschätzende Bedeutung zugesprochen werden.

Abkürzungsverzeichnis

AABB Axis-Aligned Bounding Boxes

ABB Asea Brown Boveri

API Application Programming Interface

ASEA Allmänna Svenska Elektriska Aktiebolaget

BB Bounding Boxes

BVH Bounding Volume Hierarchies

CAD Computer-aided Design

CDS Cutpiece Detection System

CFK Carbonfaserverstärkter Kunststoff

CIL Common Intermediate Language

CIRC Circular

CLI Common Language Interface

CLR Common Language Runtime

CPD Composite-Design

CR Crossoverrate

CR Crossoverrrate

DDOF Differential Degrees of Freedom

DH Denavit-und-Hartenberg

DLR Deutsches Zentrum für Luft- und Raumfahrt

DLR Deutschen Zentrums für Luft- und Raumfahrt

DNA Desoxyribonukleinsäure

DOF Degrees of Freedom

EA Evolutionäre Algorithmen

EP Evolutionäre Programmierung

ES Evolutionsstrategien
EST Expansive Space Trees
FMT Fast Marching Tree Algorithm
FVK Faserverstärkte Kunststoffe
GJK Gilbert-Johnson-Keerthi
GPU Graphical Processing Unit
GUI Graphical User Interface
IC Anzahl der Individuen
IC Individual Count
IK Inverse Kinematik
IP Internetprotokoll
ISB Image-Space-Algorithmen
IT Informationstechnologie
JDF Job Definition File
k-DOP k-Discrete Oriented Polytopes
KI Künstlichen Intelligenz
KNN k-Nearest Neighbour
KoKo Kollisionsfreie-Kooperation
KPIECE Kinematic Planning by Interior-Exterior Cell Exploration
KRC KUKA-Robot-Control
KRL KUKA Robot Language
KS Koordinatensystem
LBR Leichtbauroboter
LIN Linear
LPM Local Planning Method
MES Manufacturing Execution System
MR Mutationsrate
NN Neuronales Netz
OBB Oriented Bounding Boxes
OLP Offline-Programmierung
OMPL Open Motion Planning Library
OPC OLE for Process Control
PAN Polyacrylnitril
PC Personal Computer
PDST Path-Directed Subdivision Trees

PRM Probabilistic Roadmap Method

PTP Punkt zu Punkt

PV Pfadvereinfachung

R1 R1 - Master-Roboter in der TEZ

R2 R2 - Slave-Roboter in der TEZ

RCC Robot Control Core

RLG Random Loop Generator

RPP Randomized-Path-Planner

RPS Random Portion Selection

RPY Roll-Pitch-Yaw

RRS Retrieval-RRT-Strategy

RRT Rapidly Exploring Random Trees

RSI Remote Sensor Interface

SM Selection Method

SM Selektionsmethode

SPARS Sparse Roadmap Spanner Algorithm

SPARStwo SPArse Roadmap Spanner Version 2.0

STL Stereolithography

STRIDE Search Tree with Resolution Independent Density Estimation

SVM Support Vector Machine

TCP Tool Center Point

TEZ Technologie-Erprobungszelle

TRRT Transition-based Rapidly-exploring Random Trees

UA Unified Architecture

UDP User Datagram Protocol

VP Via Punkte

VSM Vertex Selection Method

WPF Windows Presentation Foundation

XML Extensible Markup Language

XWB eXtra Wide Body

ZLP Zentrum für Leichtbauproduktionstechnologie

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodes

Liste der Algorithmen

Literatur

[1] T. Bauernhansl, M. ten Hompel und B. Vogel-Heuser. *Industrie 4.0 in Produktion, Automatisierung und Logistik*. Springer, 2014. DOI: 10.1017/CBO9781107415324.004 (siehe S. 3).

[2] RolandBerger. „Think Act“. In: (2014) (siehe S. 3).

[3] RolandBerger. „Die Digitale Transformation der Industrie“. In: (2015) (siehe S. 3).

[4] H. Kagermann, W. Wahlster und J. Helbig. „Deutschlands Zukunft als Produktionsstandort sichern Umsetzungsempfehlungen für das Zukunftstprojekt Industrie 4.0“. In: (2013) (siehe S. 4).

[5] 3dwarehouse. 2017. URL: https://3dwarehouse.sketchup.com/ (besucht am 20. 02. 2017) (siehe S. 4).

[6] B. Beckert, D. Buschak, B. Graf, M. Hägele, A. Jäger, C. Moll, U. Schmoch und S. Wydra. *Automatisierung und Robotik-Systeme*. 2016 (siehe S. 5).

[7] L. Larsen und B. Georg. *Entwurf und Test eines Saugsystems für das automatisierte Handling von Kohlefasergelegen*. Techn. Ber. 2011, S. 1–23 (siehe S. 9).

[8] Duden. 2016. URL: www.duden.de (besucht am 03. 01. 2016) (siehe S. 11).

[9] KUKA. *Der erste KUKA-Roboter*. 2015. URL: http://www.kuka-robotics.com/germany/de/company/group/milestones/1973.htm (siehe S. 12).

[10] ABB. „ABB-Technologien, die die Welt verändern“. In: (2010), S. 48 (siehe S. 12).

[11] B. Siciliano, L. Sciavicco, L. Villani und G. Oriolo. *Robotics - Modelling, Planning and Control*. 2010, S. 392. DOI: 10.1007/b135806 (siehe S. 12, 13, 21, 39, 40).

[12] KUKA. „KR C2 / KR C3 - Expert Programming - KUKA System Software“. In: (2003), S. 1–183 (siehe S. 15, 18, 19, 26, 28).

[13] KUKA. *External Axes - Expert Documentation*. 2008 (siehe S. 16).

[14] B. Siciliano und O. Khatib. *Springer Handbook of Robotics*. Hrsg. von B. Siciliano und O. Khatib. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008. DOI: 10.1007/978-3-540-30301-5 (siehe S. 20, 21, 41–43).

[15] S. Cubero. *Industrial Robotics.- Theory, Modelling and Control*. Bd. 66. 2007. DOI: 10.1017/CBO9780511712173.002 (siehe S. 20).

[16] M. Brandstötter, A. Angerer und M. Hofbaur. „An Analytical Solution of the Inverse Kinematics Problem of Industrial Serial Manipulators with an Orthoparallel Basis and a Spherical Wrist“. In: *Proceedings of the Austrian Robotics Workshop*. 2014, S. 7–11 (siehe S. 21, 71).

[17] KUKA. *Arm Tutorial 1*. 2009. URL: http://www.kuka-robotics.com/NR/rdonlyres/B32875EB-0F2C-4622-924D-846129E84D6F/0/Armtutorial1.pdf (siehe S. 25, 27).

[18] S. Y. Nof. *Handbook Of Industrial Robotics*. John Wiley & Sons, Inc., 1999, S. 1378 (siehe S. 31).

[19] C. Smith, Y. Karayiannidis, L. Nalpantidis, X. Gratal, P. Qi, D. V. Dimarogonas und D. Kragic. „Dual arm manipulation - A survey“. In: *Robotics and Autonomous Systems* 60 (2012), S. 1340–1353. DOI: 10.1016/j.robot.2012.07.005 (siehe S. 33).

[20] S. Kämpfer. *Kooperierende Roboter: Mehr Wertschöpfung in der Fertigung*. 2008. URL: https://industrieanzeiger.industrie.de/technik/automatisierung/roboter-arbeiten-zu-wenig/ (besucht am 16. 02. 2016) (siehe S. 33).

[21] S. Kockara, T. Halic, K. Iqbal, C. Bayrak und R. Rowe. „Collision detection: A survey“. In: 2007, S. 4046–4051. DOI: 10.1109/ICSMC.2007.4414258 (siehe S. 35).

[22] D. Baraff. *Physically Based Modeling: Rigid Body Simulation*. Techn. Ber. 2001 (siehe S. 35).

[23] M. C. Lin und D. Manocha. *Efficient Contact Determination Between Geometric Models*. Techn. Ber. International Journal of Computational Geometry und Applications (siehe S. 35).

[24] B. V. Mirtich. „Impulse-based Dynamic Simulation of Rigid Body Systems“. AAI9723116. Diss. 1996 (siehe S. 35).

[25] M. Moore und J. Wilhelms. „Collision Detection and Response for Computer Animation“. In: *Computer Graphics*. 1988, S. 289–298 (siehe S. 35).

[26] M. C. Lin und J. F. Canny. „A fast algorithm for incremental distance calculation“. In: *Proceedings. 1991 IEEE International Conference on Robotics and Automation*. 1991, 1008–1014 vol.2. DOI: 10.1109/ROBOT.1991.131723 (siehe S. 35).

[27] B. V. Mirtich. „V-Clip: fast and robust polyhedral collision detection“. In: *ACM Trans. Graph.* 17.3 (1998), S. 177–208. DOI: 10.1145/285857.285860 (siehe S. 35).

[28] S. A. Ehmann und M. Lin. *SWIFT: Accelerated Proximity Queries Using Multi-Level Voronoi Marching*. 2001 (siehe S. 35).

[29] E. G. Gilbert, D. W. Johnson und S. S. Keerthi. „A fast procedure for computing the distance between complex objects in three-dimensional space“. In: *IEEE Journal on Robotics and Automation* 4.2 (1988), S. 193–203. DOI: 10.1109/56.2083 (siehe S. 35).

[30] S. Cameron. „Enhancing GJK: Computing Minimum and Penetration Distances between Convex Polyhedra“. In: *Proceedings of International Conference on Robotics and Automation*. 1997, S. 3112–3117 (siehe S. 35).

[31] G. van den Bergen. „A Fast and Robust GJK Implementation for Collision Detection of Convex Objects“. In: *J. Graph. Tools* 4.2 (1999), S. 7–25. DOI: 10.1080/10867651.1999.10487502 (siehe S. 35).

[32] G. van den Bergen. „Proximity Queries and Penetration Depth Computation on 3D Game Objects.“ In: *Proceedings of Game Developers Conference*. San Jose, CA, 2001 (siehe S. 35).

[33] E. G. Gilbert und C. P. Foo. „Computing the Distance Between General Convex Objects in Three-Dimensional Space“. In: *IEEE Transactions on Robotics and Automation* 6.1 (1990), S. 53–61. DOI: 10.1109/70.88117 (siehe S. 35).

[34] H. Jang, T. Jeong und J. Han. „GPU-based Image-space Approach to Collision Detection among Closed Objects“. In: *Journal of the HCI Society of Korea* 1 (Jan. 2008). DOI: 10.17210/jhsk.2006.05.1.1.45 (siehe S. 35).

[35] N. T. Stewart. *An Image-Space Algorithm for Hardware-based Rendering of Constructive Solid Geometry*. 2008 (siehe S. 35).

[36] B. Heidelberger, M. Teschner und M. Gross. „Detection of Collisions and Self-collisions Using Image-space Techniques“. In: *Journal of WSCG*. 2004, S. 145–152 (siehe S. 35).

[37] H.-Y. Jang, T. Jeong und J. Han. „Image-space Collision Detection Through Alternate Surface Peeling“. In: *Proceedings of the 3rd international conference on Advances in visual computing - Volume Part I*. ISVC'07. Lake Tahoe, NV, USA: Springer-Verlag, 2007, S. 66–75. DOI: 10.1007/978-3-540-76858-6_7 (siehe S. 35).

[38] K. Myszkowski, O. G. Okunev und T. L. Kunii. „Fast Collision Detection Between Complex Solids Using Rasterizing Graphics Hardware.“ In: *The Visual Computer* 11.9 (1995), S. 497–511. URL: http://dblp.uni-trier.de/db/journals/vc/vc11.html#MyszkowskiOK95 (siehe S. 35).

[39] B. Heidelberger, M. Teschner und M. Gross. „Volumetric Collision Detection for Deformable Objects“. In: (2003) (siehe S. 35).

[40] M. Figueiredo, J. Oliveira, B. Araujo und J. Pereira. *An Efficient Collision Detection Algorithm for Point Cloud Models*. Techn. Ber. 2010 (siehe S. 35).

[41] G. Bradshaw und C. O'Sullivan. „Adaptive Medial-Axis Approximation for Sphere-Tree Construction“. In: *ACM Transactions on Graphics* 23.1 (2004), S. 1–26. DOI: 10.1145/966131.966132 (siehe S. 35).

[42] J. Klein und G. Zachmann. „Point Cloud Collision Detection“. In: *Computer Graphics forum (Proc. EUROGRAPHICS)*. Hrsg. von M.-P. Cani und M. Slater. Bd. 23. 3. Grenoble, France, 2004, S. 567–576. URL: http://www.gabrielzachmann.org/ (siehe S. 35).

[43] J. Pan, S. Chitta und D. Manocha. „Probabilistic Collision Detection Between Noisy Point Clouds Using Robust Classification“. In: *International Symposium on Robotics Research*. Flagstaff, Arizona, Aug. 2011 (siehe S. 35).

[44] I. D. Yakut. *Collision Detection of Multiple Moving Objects on GPU*. Techn. Ber. 2010 (siehe S. 35).

[45] M. Flemming, G. Ziegmann und S. Roth. *Faserverbundbauweisen - Halbzeuge und Bauweisen*. Springer, 1996 (siehe S. 36).

[46] B. Fiedler. „Mikromechanische Betrachtung der Lasteinleitung und Lastübertragung in Faserverstärkten Polymeren“. Diss. TU Hamburg-Harburg, 1998 (siehe S. 36).

[47] AVK. *Handbuch Faserverbundkunststoffe*. Hrsg. von V. Mathes. Vieweg+Teubner, 2010. DOI: 10.1007/978-3-658-02755-1 (siehe S. 36).

[48] S.-J. Park. *Carbon Fibers*. Hrsg. von R. Hull, C. Jagadish, R. M. Osgoo, J. Parisi, T.-Y. Seong, U. Shin-ichi und Z. M. Wang. Bd. 21. Springer, 2015, S. 35–40. DOI: 10.1361/asmhba0003354 (siehe S. 36).

[49] A. Schuster. „Autonomes System zur Herstellung von Organoblechen und Preforms“. Diss. Universität Augsburg, 2015 (siehe S. 37, 96).

[50] N. Nilsson. „A mobile automation: an application of artificial intelligence techniques“. In: *1st International Joint Conference on Artificial Intelligence* 1969.May (1969), S. 509–520. URL: http://www.sri.com/sites/default/files/uploads/publications/pdf/1302.pdf (siehe S. 39).

[51] T. Lozano-Pérez. „Spatial Planning : A Configuration Space Approach“. In: *IEEE Transactions on Computers* C-32 (1983), S. 108–120 (siehe S. 39, 40).

[52] J.-C. Latombe. *Robot motion planning*. 1991, S. 667. DOI: 10.1016/1049-9660(91)90042-N (siehe S. 39).

[53] L. Yang, J. Qi, J. Xiao und X. Yong. „A literature review of UAV 3D path planning“. In: *Proceedings of the World Congress on Intelligent Control and Automation (WCICA)* 2015-March.March (2015), S. 2376–2381. DOI: 10.1109/WCICA.2014.7053093 (siehe S. 39, 44).

[54] S. M. LaValle. „Planning Algorithms“. In: (2006). DOI: 10.1017/CBO9780511546877 (siehe S. 39, 41, 45, 46, 51, 52).

[55] M. W. Otte. „A Survey of Machine Learning Approaches to Robotic Path-Planning“. In: (2015) (siehe S. 40).

[56] R. Siegwart und I. R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bd. 23. 2004, S. 47–82. DOI: 10.1109/ROBOT.2010.5509725. arXiv: arXiv:1402.5188v1 (siehe S. 40, 41).

[57] M. Schmidt. „Evaluierung gitterbasierter Pfadplanungs-Algorithmen für die Hardwarebeschleunigung mit FPGAs“. Diss. Friedrich-Alexander-Universität Erlangen-Nürnberg, 2013 (siehe S. 40).

[58] S. M. Udupa. „Collision Detection and Avoidance in Computer Controlles Manipulators“. In: *Proceedings of the 5th International Joint Conference on Artificial Intelligence*. San Francisco: Morgan Kaufmann Publishers Inc., 1977, S. 737–748 (siehe S. 40).

[59] T. Lozano-Pérez. „Automatic Planning of Manipulator Transfer Movements“. In: *IEEE Transactions on Systems, Man and Cybernetics* 11.10 (1981), S. 681–698. DOI: 10.1109/TSMC.1981.4308589 (siehe S. 40).

[60] T. Lozano-Pérez und M. A. Wesley. „An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles“. In: *Communications of the ACM* 22.10 (1979), S. 560–570 (siehe S. 40).

[61] Wikipedia. *Motion Planning*. 2017. URL: https://en.wikipedia.org/wiki/Motion_planning (besucht am 04. 04. 2017) (siehe S. 41).

[62] O. Souissi, R. Benatitallah, D. Duvivier, A. Artiba, N. Belanger und P. Feyzeau. „Path Planning : A 2013 Survey“. In: *International Conference on Industrial Engineering and System Management*. Bd. 2013. October. Rabat, 2013, S. 1–8 (siehe S. 41, 44, 45).

[63] S. M. LaValle und J. J. Kuffner. „Rapidly-exploring random trees: Progress and prospects“. In: *Algorithmic and Computational Robotics: New Directions*. 2000, S. 293–308 (siehe S. 41, 47, 48, 50).

[64] Z. Li und J. Canny. *Nonholomic Motion Planning*. Springer, 1993, S. 448. DOI: 10.1007/978-1-4615-3176-0 (siehe S. 41).

[65] B. Donald, P. Xavier, J. Canny und J. Reif. „Kinodynamic Motion Planning“. In: *IEEE Robotics & Automation Magazine* 17.1 (1993), S. 1048–1066. DOI: 10.1109/MRA.2010.935794 (siehe S. 41).

[66] M. Stilman. „Task Constrained Motion Planning in Robot Joint Space“. In: *International Conference on Intelligent Robots and Systems (IROS)*. San Diego, 2007, S. 3074–3081. DOI: 10.1109/IROS.2007.4399305 (siehe S. 42).

[67] F. Lamiraux und L. E. Kavraki. „Planning Paths for Elastic Objects under Manipulation Constraints“. In: *The International Journal of Robotics Research* 20.3 (2001), S. 188–208. DOI: doi.org/10.1177/02783640122067354 (siehe S. 42).

[68] K. Goldberg. „Completeness in robot Motion Planning“. In: *WAFR Proceedings of the workshop on Algorithmic foundations of robotics*. San Francisco, 1995, S. 419–429. DOI: 10.1109/ICRA.2012.6224742 (siehe S. 43).

[69] J. H. Reif. „Complexity of the Mover's Problem and Generalizations“. In: *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*. August. 1979, S. 421–427. DOI: 10.1109/SFCS.1979.10 (siehe S. 43).

[70] J. F. Canny. *Complexity of Robot Motion Planning*. The MIT Press, 1987 (siehe S. 43).

[71] E. Mazer, J. M. Ahuactzin und P. Bessiere. „The Ariadne ' s Clew Algorithm“. In: *J. Artif. Int. Res.* 9 (1998), S. 295–316 (siehe S. 43, 60).

[72] L. Han und N. M. Amato. *A Kinematic-Based Probabilistic Roadmap Method for Closed-Chain Systems*. 2000 (siehe S. 44, 59).

[73] J. Cortés. „Motion Planning Algorithms for General Closed-Chain Mechanisms“. Diss. Université de Toulouse, 2003 (siehe S. 44, 59).

[74] E. Masehian und D. Sedighizadeh. „Classic and Heuristic Approaches in Robot Motion Planning – A Chronological Review“. In: *World Academy of Science, Engineering and Technology* 29.5 (2007), S. 101–106. DOI: 10.1.1.97.1723 (siehe S. 44).

[75] A. Atyabi und D. M. W. Powers. „Review of classical and heuristic-based navigation and path planning approaches“. In: 5.International Journal of Advancements in Computing Technology (2013), S. 1–14 (siehe S. 44).

[76] T. H. Cormen, C. E. Leiserson, R. L. Rivest und C. Stein. *Introduction to algorithms*. Bd. 53. 9. 2001, S. 1689–1699. DOI: 10.1017/CBO9781107415324.004 (siehe S. 45).

[77] S. M. LaValle. „Rapidly-Exploring Random Trees: A New Tool for Path Planning“. In: *In* 129 (1998), S. 98–11. DOI: 10.1.1.35.1853 (siehe S. 45, 46).

[78] D. Hsu, J.-C. Latombe und R. Motwani. „Path Planning in Expansive Configuration Spaces“. In: *International Journal of Computational Geometry & Aplplications* 9 (1999), S. 495–512. DOI: doi.org/10.1142/S0218195999000285 (siehe S. 46).

[79] I. A. Şucan und L. E. Kavraki. *Kinodynamic motion planning by interior-exterior cell exploration*. Bd. 57. 2008, S. 449–464. DOI: doi.org/10.1007/978-3-642-00312-7_28 (siehe S. 46).

[80] B. Gipson, M. Moll und L. E. Kavraki. „Resolution Independent Density Estimation for motion planning in high-dimensional spaces“. In: *Proceedings - IEEE International Conference on Robotics and Automation* (2013), S. 2437–2443. DOI: 10.1109/ICRA.2013.6630908 (siehe S. 46).

[81] A. M. Ladd und L. E. Kavraki. „Motion Planning in the Presence of Drift, Underactuation and Discrete System Changes“. In: *Robotics: Science and Systems {I}* (2005), S. 233–241. DOI: 10.15607/RSS.2005.I.031 (siehe S. 46).

[82] J. Hossain. *Rapidly-exploring Random Tree (RRT)*. 2012. URL: https://upload.wikimedia.org/wikipedia/commons/6/62/Rapidly-exploring_Random_Tree_%28RRT%29_500x373.gif (besucht am 28. 04. 2017). Lizenz: Creative Commons BY-SA 3.0 (siehe S. 49).

[83] J. Kuffner und S. LaValle. „RRT-connect: An efficient approach to single-query path planning“. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)* 2.Icra (2000), S. 995–1001. DOI: 10.1109/ROBOT.2000.844730 (siehe S. 47, 50).

[84] S. Karaman und E. Frazzoli. „Sampling-based algorithms for optimal motion planning“. In: *International Journal of Robotics Research* 30.7 (2011), S. 846–894. DOI: 10.1177/0278364911406761 (siehe S. 50, 51).

[85] L. E. Kavraki, P. Švestka, J. C. Latombe und M. H. Overmars. „Probabilistic roadmaps for Path Planning in High-Dimensional Configuration Spaces“. In: *IEEE Transactions on Robotics and Automation*. Bd. 12. 4. 1996, S. 566–580. DOI: 10.1109/70.508439 (siehe S. 51).

[86] R. Bohlin und L. E. Kavraki. „Path Planning using lazy PRM“. In: *IEEE International Conference on Robotics and Automation (ICRA)*. Bd. 1. April. 2000, S. 521–528. DOI: 10.1109/ROBOT.2000.844107 (siehe S. 51).

[87] A. Dobson, A. Krontiris und K. E. Bekris. „Sparse Roadmap Spanners“. In: (2012), S. 1–16 (siehe S. 51).

[88] E. O. Scott. *PRM with Ob-maps*. 201. URL: https://upload.wikimedia.org/wikipedia/commons/9/99/PRM_with_Ob-maps.gif (besucht am 28. 04. 2017). Lizenz: Creative Commons BY-SA 4.0 (siehe S. 53).

[89] T. Weise. *Global Optimization Algorithm - Theory and Application*. 2011 (siehe S. 52, 54, 58).

[90] C. Darwin. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*. London: John Murray, 1859 (siehe S. 52).

[91] A. P. Engelbrecht. *Computational Intelligence - An Introduction*. John Wiley & Sons, Ltd., 2007 (siehe S. 52, 58).

[92] M. Henderson. *Genetik - 50 Schlüsselideen*. Springer Spektrum, 2010, S. 208. DOI: 10.1007/978-3-8274-2381-8 (siehe S. 53).

[93] M. Mitchell. *An Introduction to Genetic Algorithms*. Massachusetts Institute of Technology All, 1999, S. 162 (siehe S. 54).

[94] A. E. Eiben und J. E. Smith. *Introduction to Evolutionary Computing*. Springer Berlin Heidelberg, 2003, S. 300. DOI: 10.1007/978-3-662-05094-1 (siehe S. 54).

[95] S. Chen, J. Montgomery und A. Bolufé-Röhler. „Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution“. In: *Applied Intelligence* 42 (2015), S. 514–526. DOI: 10.1007/s10489-014-0613-2 (siehe S. 55).

[96] Y.-A. Zhang, Q. Ma, M. Sakamoto und H. Furutani. „Effects of population size on the performance of genetic algorithms and the role of crossover“. In: *Artificial Life and Robotics* 15.2 (2010), S. 239–243. DOI: 10.1007/s10015-010-0836-1 (siehe S. 55).

[97] R. Kruse, C. Borgelt, C. Braune, F. Klawonn, C. Moewes und M. Steinbrecher. *Computational Intelligence - Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze*. 2012. DOI: 10.1007/978-3-658-10904-2 (siehe S. 55, 56).

[98] K. A. De Jong. „Analysis of the Behaviour of a Class of Genetic Adaptive Systems“. In: *Algorithmic Foundations of Robotics*. X. Springer Berlin Heidelberg, 1975, S. 279–296. DOI: doi.org/10.1007/978-3-642-36279-8_17 (siehe S. 55).

[99] V. Nissen. *Einführung in Evolutionäre Algorithmen - Optmierung nach dem Vorbild der Evolution*. Vieweg+Teubner Verlag, 1997. DOI: 10.1007/978-3-322-93861-9 (siehe S. 55).

[100] W. M. Spears. *Evolutionary Algorithms - The Role of Mutation and Recombination*. Bd. 1. Springer Berlin Heidelberg, 2000. DOI: 10.1007/978-3-662-04199-4 (siehe S. 57).

[101] B. J. Jain, H. Pohlheim und J. Wegener. „On Termination Criteria of Evolutionary Algorithms“. In: *3rd Annual Conference on Genetic and Evolutionary Computation*. Morgan Kaufmann Publishers Inc., 2001, S. 768–768 (siehe S. 58).

[102] B. Fink und H.-D. Wend. „A Fast Path-Planning Algorithm for Industrial Robots“. In: *IEEE International Conference on Control and Applications*. 1989, S. 369–374. DOI: 10.1109/ICCON.1989.770536 (siehe S. 58).

[103] C. H. Wurll, D. O. Henrich und H. E. Wörn. „Multi-Goal Path Planning for Industrial Robots“. In: *International Conference on Robotics and Application (RA´99)*. Santa Barbara, 1999, S. 1–7 (siehe S. 58).

[104] Y. Ting, W. Lei und H. Jar. „A Path Planning Algorithm for Industrial Robots“. In: *Computers & Industrial Engineering* 42.2-4 (2002), S. 299–308. DOI: 10.1016/S0360-8352(02)00013-X (siehe S. 58).

[105] K. Oh, J. P. Hwang, E. Kim und H. Lee. „Path Planning of a Robot Manipulator using Retrieval RRT Strategy“. In: *International Journal of Electrical, Robotics, Electronics and Communications Engineering Vol:1* 1 (2007), S. 25–28. DOI: 10.5391/IJFIS.2007.7.2.138 (siehe S. 58).

[106] Y. Koga und J.-C. Latombe. „On multi-arm manipulation planning“. In: *Proceedings of the 1994 IEEE International Conference on Robotics and Automation* (1994), S. 945–952. DOI: 10.1109/ROBOT.1994.351231 (siehe S. 59).

[107] S. M. LaValle, J. H. Yakey und L. E. Kavraki. „A Probabilistic Roadmap Approach for System with Closed Kinematic Chains“. In: *Internation Conference on Robotic and Automation (ICRA)*. 1999, S. 1671–1676. DOI: 10.1109/ROBOT.1999.770349 (siehe S. 59).

[108] J. Cortés und T. Siméon. „Sampling-Based Motion Planning under Kinematic Loop-Closure Constraints“. In: *Algorithmic Foundations of Robotics*. Bd. VI. Springer Berlin Heidelberg, 2005, S. 75–90. DOI: doi.org/10.1007/10991541_7 (siehe S. 59).

[109] A. C. Nearchou und N. a. Aspragathos. „A genetic path planning algorithm for redundant articulated robots“. In: *Robotica* 15.2 (1997), S. 213–224. DOI: 10.1017/S0263574797000234 (siehe S. 59, 60).

[110] M. F. Ibrahim, A. Zaira, A. Bakar und A. Hussain. „Genetic Algorithm-based Robot Path Planning“. In: *International Conference on Intelligent Computation Technology and Automation (ICICTA)*. 2008, S. 56–59. DOI: 10.1109/ICICTA.2008.216 (siehe S. 59).

[111] F. K. Purian, F. Farokhi und R. S. Nadooshan. „Comparing the Performance of Genetic Algorithm and Ant Colony Optimization Algorithm for Mobile Robot Path Planning in the Dynamic Environments with Different Complexities“. In: *Journal of Academic and Applied Studies* 3.February (2013), S. 29–44 (siehe S. 59).

[112] H. Burchardt und R. Salomon. „Implementation of Path Planning using Genetic Algorithms on Mobile Robots“. In: Vancouver: IEEE, 2006, S. 6. DOI: 10.1109/CEC.2006.1688529 (siehe S. 59).

[113] A. Elshamli, H. A. Abdullah und S. Areibi. „Genetic Algorithm for Dynamic Path Planning“. In: Bd. 2. 2004, S. 677–680. DOI: 10.1109/CCECE.2004.1345203 (siehe S. 59).

[114] X. Zou, B. Ge und P. Sun. „Improved Genetic Algorithm for Dynamic Path Planning“. In: *International Journal of Information and Computer Science* 1.2 (2012), S. 16–20 (siehe S. 59).

[115] C. Liu. „A Path Planning Method Based on Adaptive Genetic Algorithm for Mobile Robot“. In: (2011), S. 808–814 (siehe S. 59).

[116] H. Miao. „Project Title : Robot Path Planning in Dynamic Environments using a Simulated Annealing Based Approach“. Diss. Queensland University of Technology Project, 2009 (siehe S. 59).

[117] P. L. Sivakumar, A. Prof und M. Eng. „Path Planning of Construction Manipulators Using Genetic Algorithms“. In: *Automation and Robotics in Construction* (1999), S. 555–560 (siehe S. 60).

[118] M.-H. Lavoie und R. Boudreau. „Obstacle Avoidance for Redundant Manipulators Using a Genetic Algorithm“. In: *IEEE Proceedings of the SOUTHEASTCON*. 1991, S. 317–320. DOI: 10.1109/SECON.1991.147764 (siehe S. 60).

[119] G. S. Sharma. „Optimization of Energy in Robotic arm using Genetic Algorithm“. In: *InternatIonal Journal of Computer SCIenCe and teChnology* 2.2 (2011), S. 315–317 (siehe S. 60).

[120] C. C. Lin. „Hierarchical path planning for manipulators based on genetic algorithm with non-random initial population“. In: *International Journal of Innovative Computing, Information and Control* 8 (2012), S. 1773–1786 (siehe S. 60).

[121] P. Bezák. *Using Motion Planning and genetic Algorithms in Movement Optimization of industrial Robots*. Hrsg. von P. Husar und K. Resetova. 5. Aufl. Ilmenau: Universitätsverlag Ilmenau, 2012, S. 93 (siehe S. 60).

[122] J. Ahuactzin, E.-G. Talbi, P. Bessiere und E. Mazer. *Using genetic algorithm for robot motion planning*. Heidelberg, 1992. DOI: 10.1007/3-540-57132-9_6 (siehe S. 60).

[123] L. Eduardo, P. Nunes, V. Orlando und G. Rosado. „Robotic Manipulator Path Planning by Genetic Algorithms“. In: *19th International Congress of Mechanical Engineering*. 2007 (siehe S. 60).

[124] S. Yue und D. Henrich. „Point-to-Point Trajectory Planning of Flexible Redundant Robot Manipulators Using Genetic Algorithms“. In: *Robotica* 20 (2002). DOI: 10.1017/S0263574701003861 (siehe S. 60).

[125] F. Y. C. Albert, S. P. Koh, C. P. Chen, S. K. Tiong, S. Y. S. Edwin, U. T. Nasional und J. Kajang-puchong. „Optimizing Joint Angles of Robotic Manipulator using genetic Algorithm“. In: *International Conference on Computer Engineering and Applications*. Bd. 2. Singapore, 2011, S. 134–139 (siehe S. 60).

[126] B. I. Kazem, A. I. Mahdi und A. T. Oudah. „Motion Planning for a Robot Arm by Using Genetic Algorithm“. In: *Jordan Journal of Mechanical and Industrial Engineering* 2.3 (2008), S. 131–136 (siehe S. 60).

[127] V. D. Cueva und F. Ramos. „Adapting the Messy Genetic Algorithm for Path Planning in Redundant and Non-redundant Manipulators“. In: *Mexican International Conference on Artificial Intelligence*. 1. Springer Berlin Heidelberg, 2002, S. 21–30. DOI: doi.org/10.1007/3-540-46016-0_3 (siehe S. 60).

[128] A. A. Ata und T. R. Myo. „Collision-free trajectory planning for manipulators using generalized pattern search“. In: *International Journal of Simulation Modelling* 5.4 (2006), S. 145–154. DOI: 10.2507/IJSIMM05(4)2.074 (siehe S. 60).

[129] C. A. C. Coello, A. de Albornoz, L. E. Sucar und O. C. Battistutti. „Advances in Artificial Intelligence“. In: *Second Mexican International Conference on Artificial Intelligence*. Springer, 2002 (siehe S. 60).

[130] B. K. Gouda. „Optimal Robot Trajectory Planning using Evolutionary Algorithms“. Diss. 2003 (siehe S. 60).

[131] M. A. C. Gill und A. Y. Zomaya. „A Parallel Collision-Avoidance Algorithm for Robot Manipulators“. In: *IEEE Concurrency* 6.1 (1998), S. 68–87. DOI: 10.1109/4434.656781 (siehe S. 60).

[132] F. J. Abu-Dakka. „Trajectory Planning for Industrial Robots Using Genetic Algorithms“. Diss. 2011 (siehe S. 60).

[133] Z. S. Abo-Hammour, O. M. Alsmadi, S. I. Bataineh, M. A. Al-Omari und N. Affach. „Continuous Genetic Algorithms for Collision-Free Cartesian Path Planning of Robot Manipulators“. In: *International Journal of Advanced Robotic Systems* (2011), S. 14–36. DOI: 10.5772/50902 (siehe S. 60).

[134] O. M. K. Alsmadi und Z. S. Abo-Hammour. „Collision-Avoidance Problem Solution of Robot Manipulators Via CGA“. In: *7th International Conference on Electrical and Electronics Engineering*. Bursa, 2011 (siehe S. 60).

[135] S. Mitsi, K.-D. Bouzakis, D. Sagris und G. Mansour. „Robot Path Planning Optimization, Free of Collisions, Using A Hybrid Algorithm“. In: *3rd International Conference on Manufacturing Engineering (ICMEN)*. Hrsg. von K.-D. Bouzakis. October. Chalkidiki, 2008, S. 1–3 (siehe S. 60).

[136] Ć. Petar und B. Jerbi. „Dual-Arm Robot Motion Planning Based on Cooperative Coevolution“. In: *Doctoral Conference on Computing, Electrical and Industrial Systems*. 2010, S. 169–178. DOI: doi.org/10.1007/978-3-642-11628-5_18 (siehe S. 60).

[137] O. Chocron und P. Bidaud. „Evolutionary Algorithms in Kinematic Design of Robotic Systems“. In: *International Conference on Intelligent Robot and Systems (IROS)*. 1997, S. 1111–1117. DOI: 10.1109/IROS.1997.655148 (siehe S. 61).

[138] P. Curkovic, B. Jerbic und T. Stipancic. „Co-Evolutionary Algorithm for Motion Planning of Two Industrial Robots with Overlapping Workspaces“. In: *International Journal of Advanced Robotic Systems* (2013), S. 1. DOI: 10.5772/54991 (siehe S. 61).

[139] BEPU physics. 2018. URL: https://github.com/bepu/bepuphysics1 (besucht am 11. 02. 2018) (siehe S. 64).

[140] A. Angerer, A. Hoffmann, A. Schierl, M. Vistein und W. Reif. „The robotics API: An object-oriented framework for modeling industrial robotics applications“. In: *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings* (2010), S. 4036–4041. DOI: 10.1109/IROS.2010.5649098 (siehe S. 67, 70).

[141] A. Angerer. „Object-oriented software for Industrial Robots“. Diss. 2014 (siehe S. 70).

[142] HelixToolkit. 2017. URL: https://github.com/helix-toolkit (besucht am 03. 01. 2017) (siehe S. 71).

[143] E. H. Lockwood. *A book of curves*. 1962. DOI: 10.1016/0016-0032(62)90661-0 (siehe S. 72).

[144] R. C. Yates. *Curves and their Properties*. 1952 (siehe S. 72).

[145] M. Kaspar. *Automatisierte Bahnplanung kollaborierender 7-Achs-Industrieroboter in der CFK-Fertigung*. 2016 (siehe S. 73, 79, 81, 193).

[146] ALGLIB. 2017. URL: http://www.alglib.net/ (besucht am 22. 02. 2017) (siehe S. 73, 87).

[147] Wolfram. *Arc Length*. 2017. URL: http://mathworld.wolfram.com/ArcLength.html (besucht am 11. 08. 2017) (siehe S. 75).

[148] I. A. Şucan, M. Moll und L. Kavraki. „The open motion planning library“. In: *IEEE Robotics and Automation Magazine* 19.4 (2012), S. 72–82. DOI: 10.1109/MRA.2012.2205651 (siehe S. 79).

[149] *Open Motion Planning Library: A Primer*. 2017. URL: http://ompl.kavrakilab.org/OMPL%7B%5C_%7DPrimer.pdf (siehe S. 79, 80, 104).

[150] A. Angerer, A. Hoffmann, L. Larsen, M. Vistein, J. Kim, M. Kupke und W. Reif. „Planning and Execution of Collision-free Multi-robot Trajectories in Industrial Applications“. In: *47th International Symposium on Robotics*. Munich, 2016 (siehe S. 81, 149).

[151] C. Learner. 2017. URL: %5Curl%7B%22https://de.mathworks.com/help/stats/classificationlearner-app.html%7D%22 (besucht am 08. 08. 2017) (siehe S. 87, 88).

[152] AForge. 2017. URL: http://www.aforgenet.com/framework/ (besucht am 11. 08. 2017) (siehe S. 89).

[153] M. Vistein. „Embedding Real-Time Critical Robotics Applications in an Object-Oriented Language“. Diss. 2015 (siehe S. 97).

[154] J. W. Tukey. *Exploratory data analysis*. 1977. DOI: 10.1007/978-1-4419-7976-6 (siehe S. 101).

[155] KUKA. *RoboTeam 1.0*. 2011 (siehe S. 139).

[156] EtherCAT Technology Group. „EtherCAT Slave Implementation Guide“. In: (2012) (siehe S. 149).

[157] L. Biagiotti und C. Melchiorri. *Trajectory Planning for Automatic Machines and Robots*. 1st. Springer Publishing Company, 2008 (siehe S. 149).

[158] SpatialAnalyzer. 2017. URL: http://www.kinematics.com/spatialanalyzer/ (besucht am 20. 07. 2017) (siehe S. 152).

[159] KUKA. *KR Quantec ultra*. 2016 (siehe S. 152).

[160] KUKA. *Industrial robotics_high payloads*. 2016 (siehe S. 192).

Veröffentlichungen

Wissenschaftliche Veröffentlichungen, die in Bezug zu dieser Arbeit stehen

[1] L. Larsen, J. Kim und M. Kupke. „Intelligent Path Panning Towards Collision-free Cooperating Industrial Robots“. In: *International Conference on Informatics in Control*. Vienna: SciTePress, 2014, S. 39–47.

[2] L. Larsen, V. L. Pham, J. Kim und M. Kupke. „Collision-free Path Planning of Industrial Cooperating Robots for Aircraft Fuselage Production“. In: *IEEE International Conference on Robotics and Automation*. Seattle, 2015. DOI: 10.1109/ICRA.2015.7139466.

[3] A. Angerer, A. Hoffmann, L. Larsen, M. Vistein, J. Kim, M. Kupke und W. Reif. „Planning and Execution of Collision-free Multi-robot Trajectories in Industrial Applications“. In: *47th International Symposium on Robotics*. Munich, 2016.

[4] L.-C. Larsen und A. Schuster. *Intelligente Bahnplanung für kooperierende Industrieroboter*. Augsburg, 2016.

[5] A. Schuster, M. Kühnel und L. Larsen. „Wandlungsfähiges Produktionssystem für Faserverbundstrukturen“. In: *5. Augsburger Technologietransfer-Kongress*. Augsburg, 2016.

[6] F. Krebs, L. Larsen, G. Braun und W. Dudenhausen. „Design of a multifunctional cell for aerospace CFRP production“. In: *Advances in Sustainable and Competitive Manufacturing Systems* (2013), S. 515–524. DOI: 10.1007/978-3-319-00557-7_42.

[7] L. Larsen, M. Kaspar, A. Schuster, M. Vistein, J. Kim und M. Kupke. „Full automatic path planning of cooperating robots in industrial applications“. In: *13th IEEE Conference on Automation Science and Engineering (CASE)*. Xian: IEEE, 2017, S. 523–530. DOI: 10.1109/COASE.2017.8256157.

[8] A. Schuster, M. Kupke und L. Larsen. „Autonomous Manufacturing of Composite Parts by a Multi-Robot System“. In: *Procedia Manufacturing*. Bd. 11. 2017, S. 249–255. DOI: 10.1016/j.promfg.2017.07.238.

[9] T. Schmidt und L. Larsen. *Cooperating robots programming concept with FAST SURF – accuracy on relative positioning*. 2012.

[10] L. Larsen, J. Kim, M. Kupke und A. Schuster. „Automatic Path Planning of Industrial Robots Comparing Sampling-based and Computational Intelligence Methods“. In: *Procedia Manufacturing* 11 (2017), S. 241–248. DOI: 10.1016/j.promfg.2017.07.237.

[11] L. Larsen, A. Schuster, J. Kim und M. Kupke. „Path Planning of Cooperating Industrial Robots Using Evolutionary Algorithms“. In: *Procedia Manufacturing* 17 (2018), S. 286–293. DOI: https://doi.org/10.1016/j.promfg.2018.10.048.

[12] A. Schuster, R. Glück, F. Fischer, M. Kühnel, L. Larsen und M. Kupke. „Smart Manufacturing of Thermoplastic CFRP Skins“. In: *Procedia Manufacturing* 17 (2018), S. 935–946. DOI: doi.org/10.1016/j.promfg.2018.10.147.

Betreute Abschlussarbeiten

[1] J. Robl. *Kooperative Online Robotersteuerung zur Ablage von Kohlenstofffaser-Zuschnitten*. 2015.

[2] M. Kaspar. *Automatisierte Bahnplanung kollaborierender 7-Achs-Industrieroboter in der CFK-Fertigung*. 2016.

[3] J. Brestrich. *Kollisionsberechnung auf der GPU für textile CFK-Werkstoffe*. 2015.

[4] H. V.-L. Pham. *Entwicklung von kollisionsfreien Bahnplanungsstrategien und Algorithmen für einen CFK-Ablageprozess mit kooperierenden Robotern*. 2014.

[5] F. F. B. Herzberg. *Erstellung einer affektiven Musikdatenbank und automatisierte Klassifikation*. 2015.

[6] V. Benks. *Entwicklung eines automatisierten Verfahrens zur Vermessung eines robotergestützten Laser-Messsystems*. 2013.

[7] A. Diefenbach. *Path Planning for Industrial Robots Using Evolutionary Algorithms*. 2017.

Kapitel 8

Anhang

8.1 Technische Daten der Werkzeugform

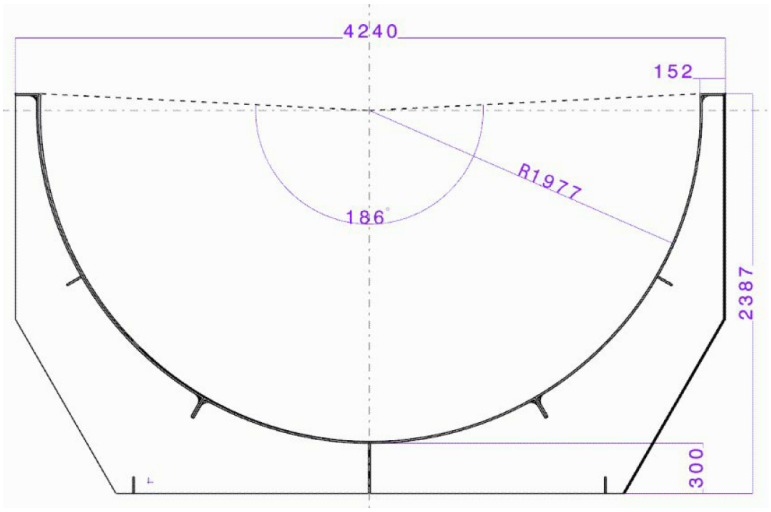
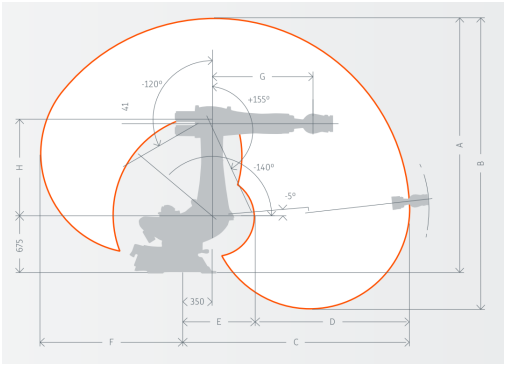
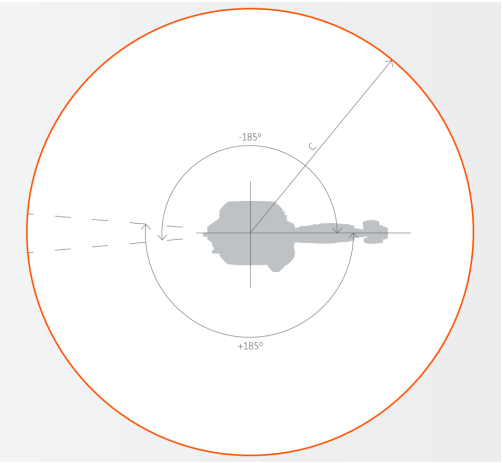


Abbildung 8.1: Zeichnung der Werkzeugform.

8.2 Maschinendaten der Roboter



(a) Seitenansicht



(b) Draufsicht

A	B	C	D	E	F	G	H
3426	4034	3095	2187	908	2085	1400	1350

(c) Längen der Achsen (in mm)

Abbildung 8.2: Abmessungen und DH-Parameter des KUKA-Roboters QUANTEC KR 210 R3100 ultra [160, S.13].

8.3 Open Motion Planning Library kompilieren

Die Integration von OMPL in KoKo wurde von Kaspar [145] durchgeführt. Die folgende Anleitung stammt, bis auf kleine Anpassungen, aus Kaspar [145], soll der Vollständigkeit halber dieser Arbeit angehängt sein.

- Falls *Boost* neu kompiliert werden soll, folgen Sie den Instruktionen weiter unten. Ansonsten können die kompilierten Dateien in `C:\boost_`[**VERSION**] kopiert werden.
- Visual Studio Tools Developer Eingabeaufforderung öffnen und folgende Kommandos ausführen
  - `cd` [**KOKOFOLDER**]`\ompl\`
  - (Falls nicht vorhanden):
    `mkdir build | cd build | mkdir Release | cd Release`
  - `cmake ../.. -T v140 -DCMAKE_CXX_FLAGS_DEBUG= -D_ITERATOR_DEBUG_LEVEL=2`
    `-D_DEBUGDCMAKE_BUILD_TYPE=Debug`
    `-DCMAKE_INSTALL_PREFIX=C:/ompl`
    `-DBOOST_INCLUDEDIR=C:/boost_`**`[VERSION]`**`/boost`
    `-DBOOST_LIBRARYDIR=C:/boost_`**`[VERSION]`**`/boost/stage/lib`
    `-DBOOST_USE_STATIC_LIBS=ON`
    `-DBOOST_USE_MULTITHREADED=ON`
    `-DOMPL_BUILD_DEMOS=ON`
    `-DOMPL_BUILD_PYBINDINGS=OFF`
    `-DOMPL_BUILD_PYTESTS=OFF`
    `-DOMPL_REGISTRATION=OFF`
    `-DBOOST_ROOT=C:\boost_`**`[VERSION]`**
  - **Wichtig:** Backslash bei `C:\boost_`**`[VERSION]`**. Wird `-DCMAKE_CXX_FLAGS_DEBUG` weggelassen und `-DCMAKE_BUILD_TYPE=Release` gesetzt, wird für den Release-Modus kompiliert.
- Der Flag `-T v140` gibt an, dass die Build Tools von Visual Studio 2015 verwendet werden. Für Visual Studio 2017 muss hier `v141` und für Visual Studio 2019 `v142` gewählt werden.
- In den Einstellungen des OMPL Projektes unter: C++ Präprozessor - Präprozessordefinitionen muss folgender Eintrag gemacht werden:
  - `WIN32`
    `_WINDOWS`
    `_ITERATOR_DEBUG_LEVEL=0`
    `_ITERATOR_DEBUG_LEVEL=2`
    `_DEBUG`
    `BOOST_ALL_NO_LIB`
    `BOOST_PROGRAM_OPTIONS_DYN_LINK`
    `BOOST_TEST_DYN_LINK`
    `CMAKE_INTDIR="Debug"`
- Das erste Builden in Visual Studio schlägt normalerweise mit der Meldung, dass eine Abhängigkeit von *OmplWrapper* nicht gefunden wurde, fehl. Dann einfach noch einmal Builden (nicht „Neu erstellen“, sondern „Erstellen“).

- Im Code der Dateien `Thunder.cpp`, `Thunder.h`, `SPARSdb.cpp` wurden ein paar Kleinigkeiten geändert, da hier Bugs gefunden wurden. Diese Änderungen sollten jedoch in zukünftigen OMPL-Versionen integriert sein
  (Siehe auch unten: `depth_first_search.hpp`).
- In `PathGeometric.cpp (Zeile 235)` wurde ebenfalls ein Bug entdeckt, der in kommenden OMPL-Versionen behoben sein sollte.
- `PathSimplifier.cpp (Zeile 447 - 452)` wurden auskommentiert.
  Die Funktion *checkAndRepair()* soll Fehler im Pfad finden und reparieren. In unseren Tests führte sie jedoch häufig zu deutlich schlechteren Pfaden. Falls die Funktion weiter verwendet werden soll, muss in *SampleNear* (in KoKo) darauf geachtet werden, dass passende Distanzwerte gewählt werden. Dies ist jedoch nicht immer leicht zu erreichen.
- Im Release Modus gab es einen Bug in den Dateien `LBTRRT.h (Zeile 227)` und `DynamicSS-SP.h (Zeile 267)`. In beiden Zeilen musste ein const an das Ende der Funktionssignatur hinzugefügt werden. Dies sollte jedoch ebenfalls in kommenden OMPL-Versionen gefixt sein.

Boost neu kompilieren

- Die aktuelle *Boost*-Version unter `http://www.boost.org/users/download` herunterladen (momentan 1.61.0 verwendet).
- *Boost* in `C:\` entpacken, so dass die Ordnerstruktur
  `C:\boost_`[**VERSION**]`\` ist (kein weiterer *Boost*-Unterordner).
- Visual Studio Tools Developer Eingabeaufforderung öffnen und folgende Kommandos ausführen
  - `cd C:\boost_`[**VERSION**]`\`
  - `bootstrap`
  - `.\b2 link=shared --build-type=complete`
- Die DLLs aus `C:\boost_`[**VERSION**] in
  [**KOKOFOLDER**]`\Debug\` kopieren. Evtl. macht Visual Studio dies beim kompilieren auch automatisch.
- Im Projekt *OmplPlanner* und *OmplPlannerCppNativeTester* ggf. bei C++ / Zusätzliche Includeverzeichnisse die *Boost*-Version anpassen.
- Im Projekt *OmplWrapper* und *OmplPlannerCppNativeTester* bei Linker / Eingabe ggf. die *Boost*-Version anpassen (Achtung auch ganz hinten im Dateinamen).
- Um das Thunder-Framework mit CLR zum Laufen zu bringen, mussten in
  `depth_first_search.hpp` in *Boost* (Zeile 81) einige Änderungen durchgeführt werden und die Zeile `...call_finish_edge(vis, e, g)` wurde auskommentiert.