Online-Adaption und synchronisierte Ausführung einmalig demonstrierter Roboterverhalten

Von der Universität Bayreuth
zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
genehmigte Abhandlung

von

Christian Groth
aus Kulmbach

1. Gutachter: Prof. Dr. Dominik Henrich
2. Gutachter: Prof. Dr. Ulrike Thomas

Tag der Einreichung: 03.11.2015
Tag des Kolloquiums: 08.07.2016

Zusammenfassung

Roboter sind im Bereich der vollautomatisierten Großindustrie weit verbreitet. In kleinen und mittelständischen Unternehmen sowie im Haushalt sind insbesondere Industrieroboter jedoch kaum anzutreffen. Einer der Hauptgründe hierfür liegt in der komplexen und langwierigen Programmierung der Roboter, die nur von Experten durchgeführt werden kann. Um die Akzeptanz und Verbreitung von Robotern zu fördern, besteht daher Bedarf an intuitiven und transparenten Programmiermethoden für Roboter.
In dieser Arbeit werden zwei Ziele verfolgt. Zum einen werden Verfahren entwickelt, die eine intuitive Programmierung eines Industrieroboters erlauben, so dass dies auch für Nicht-Experten möglich ist. Zum anderen soll die Häufigkeit der Neuprogrammierung von Aufgaben gesenkt werden, um die zeitliche Belastung für den Programmierer zu reduzieren und die Wirtschaftlichkeit für kleine und mittelständische Unternehmen zu erhöhen.
Um den Aspekt der Programmierung von Robotern intuitiver zu gestalten, werden zwei Methoden vorgestellt. Bei beiden stellt der Benutzer eine einzige Demonstration zur Verfügung (Single-Shot Learning), indem er den Roboter durch die Aufgabe führt. Diese Demonstration wird gespeichert und kann an geänderte Randbedingungen, wie beispielsweise veränderte Positionen und Orientierungen der Objekte, adaptiert werden, so dass die Aufgabe vom Roboter auch in neuen, sensorerfassten Situationen erfolgreich ausgeführt werden kann. Dazu betrachtet die erste Methode die Trajektorie des Roboters als Kette von Partikeln, die mit der Umwelt verbunden sind. Die Erstellung dieser Verbindungen erfolgt durch unterschiedliche Ansätze. Verändern sich die Randbedingungen, befinden sich die Partikel der Trajektorie nicht mehr im Gleichgewicht und es entstehen Kräfte und Momente, welche die Partikel in Bewegung versetzen. Aufgrund dieser Bewegungen verändern sich wiederum die Kräfte und Momente. Das Ergebnis dieser Wechselwirkungen wird mit Hilfe einer Partikelsimulation numerisch ermittelt und ermöglicht die Online-Reproduktion der Aufgabe unter geänderten Randbedingungen. Die zweite Methode stellt eine analytische Näherung dessen dar, indem sie die Trajektorie in werkstückbezogene Segmente und Transfersegmente klassifiziert. Bei Veränderung der Randbedingungen werden die objektbezogenen Segmente der Trajektorie entsprechend der Änderung ihres Bezugssystems transformiert. Anschließend werden die Transferbewegungen hinsichtlich der veränderten Objektpositionen adaptiert und an den Schnittstellen angeglichen, um eine Glatte Bewegung zu erzeugen.
Zur Reduzierung der Programmierhäufigkeit wird eine verhaltensbasierte Architektur für Industrieroboter vorgestellt. Die vom Benutzer demonstrierten Aufgaben werden als Verhalten gespeichert und können erneut ausgeführt werden, sobald alle notwendigen Ressourcen für die Aufgabe verfügbar sind. Die Ermittlung dieser Ressourcen geschieht dabei automatisch und dient auch der Synchronisation der Verhalten untereinander. Aufgrund der entwickelten prozessbasierten Ausführung ist es möglich, mehrere Aufgaben mit dem Roboter quasiparallel auszuführen, sie zu unterbrechen und konsistent wieder aufzunehmen.
Sowohl die intuitive Programmierung als auch die verhaltensbasierte Ausführung von Aufgaben werden in verschiedenen Experimenten untersucht. Mit Hilfe eines Prototypen werden virtuelle, sowie Realwelt-Experimente an einem Roboterarm durchgeführt und die Verfahren qualitativ und quantitativ evaluiert. Die Ergebnisse zeigen eine hohe Ähnlichkeit zwischen den vom Benutzer demonstrierten Trajektorien und den durch die Adaption erzeugten Trajektorien. Zudem zeigt sich, dass obwohl lediglich eine einzige Demonstration der Aufgabe ohne zusätzliche Informationen bereitgestellt wird, dies dennoch zur intuitiven Programmierung vieler Aufgaben aus den anvisierten Domänen ausreicht. In Kombination mit der verhaltens-

basierten Reproduktion kann der Roboter differenziert auf verschiedene Situationen reagieren und auch mehrerer Aufgaben ohne Neuprogrammierung ausführen, indem er online zwischen diesen wechselt, was den Aufwand für den Benutzer minimiert.

Abstract

The use of robots is well established in fully automated assembly lines of big companies. But robots are rarely used in small and medium enterprises or in the household. This is mainly due to their tedious and complicated programming, which is only feasible for experts. To faciliate the usage of robots in new areas, there needs to be a simple and intuitive way to program a robot.
The contribution of this work is twofold. On the one hand, intuitive methods to program a robot by demonstration are developed, which allow even non-experts to program new abilities to the robot. On the other hand, a method is developed, which reduces the workload for the user by lowering the frequency of required programmings of the robot. Both aspects will help to increase the efficiency of robots in small and medium enterprises.
In this work, two approaches are proposed in order to simplify the programming of robots. Both rely on just a single demonstration of the task, which is provided by guiding the robot through the task (single-shot learning). This demonstration can be adapted online to altered constraints, like modified object positions and orientations, which allows the robot to fullfill a task in an unknown situation. In the first method, the trajectory is considered as a chain of particles, which are linked to each other and to objects in various ways. Due to these links, the particles of the trajectory will experience forces and torques, if an object changes its position or orientation. These forces and torques result in a movement of the trajectory particles, which in turn changes the resulting forces and torques. The outcome of these interactions can be calculated in a particle simulation, which enables the fulfilment of a task under changed constraints.
The second method is an analytical approximation of the first approach. The trajectory is divided into segments, which can be classified as object-related movements or transfer movements between objects. Under changed constraints, the object-related segments will we transformed accordingly to the modification of the object poses. Afterwards the transfer movements are constructed correspondingly to the object poses. A blending of the segments is applied to generate smooth trajectories.
In order to reduce the frequency of programming by the user, a behavior-based architecture is proposed. Every demonstration of a task is stored as a behavior, which can be adapted and executed, if all required resources can be provided. These resources are identified by the system automatically and are also used to synchronize the execution of the behaviors. By utilizing a process-based execution model, several tasks can be executed by the robot in parallel and each task can be consistently interrupted and resumed.
Both intuitive programming approaches and the behavior-based execution of tasks are investigated in various experiments. A prototype was developed, which is capable of executing simulated and real-world experiments in order to receive a qualitative and quantitative evaluation. Both programming approaches allow the teaching of new tasks to the robot, which

can be executed under changed constraints, even with just a single demonstration. Besides this, the adapted trajectories show high similarity with the user-demonstrated trajectories. The combination of the intuitive programming approaches and the behavior-based execution of tasks minimizes the workload for the user, since the robot can fulfil several task without reprogramming by just switching between them online.

Danksagung

Meinem Doktorvater Prof. Dr. Dominik Henrich danke ich für die Bereitstellung dieses hochinteressanten Themas und die vielen ideenreichen Gespräche.
Meinen Kollegen danke ich für die angenehme Arbeitsatmosphäre und die vielen fachlichen und nicht-fachlichen Diskussionen.
Meiner Familie, meinen Eltern und meinen Schwiegereltern danke ich besonders - nicht nur für ihre uneingeschränkte und vielseitige Unterstützung, sondern auch für den oft so notwendigen Ausgleich. Ohne Sie wäre die Entstehung dieser Arbeit nicht möglich gewesen.

Inhaltsverzeichnis

1 Einführung

1.1 Motivation

Schon lange werden Roboter erfolgreich in einem breiten Spektrum von Anwendungen eingesetzt. Zwischen 1,3 und 1,6 Millionen Roboter verrichten derzeit weltweit, vor allem im Bereich der Automobilindustrie, der Elektroindustrie, der chemischen Industrie und im Maschinenbau verschiedenste Arbeiten [Robo14]. Dort sorgen sie für eine anhaltend hohe Qualität der Erzeugnisse bei gleichzeitiger Verringerung der Taktzeit und Senkung der Produktionskosten. Die Verwendung beschränkt sich im Wesentlichen auf Industriezweige mit Großserienfertigung und hohen Losgrößen.
Obwohl der potentielle Nutzen von Robotern sowohl für kleine und mittelständischen Unternehmen (KMU) als auch für den Haushalt bereits erkannt wurde [Armb06], sind sie in diesen Bereichen nur vereinzelt anzutreffen. Es existieren zwar Roboterlösungen, die bereits nützliche Arbeit im Haushalt erledigen, allerdings handelt es sich hierbei um Speziallösungen für bestimmte Aufgaben, wie beispielsweise Staubsaugen, Rasenmähen oder Bodenwischen [Edma15]. Die verfügbaren Modelle sind in Ihrer Handlungsweise sehr eingeschränkt und verfügen im Gegensatz zu herkömmlichen Industrierobotern nicht über die wünschenswerte bzw. notwendige Flexibilität. An dieser Stelle existiert Forschungsbedarf, da sich die Mehrheit der Bundesbürger vorstellen könnte, einen Roboter im Haushaltsbereich einzusetzen [Shah14]. Für den Einsatz in KMUs existieren lediglich wenige prototypische Lösungen, wie der Rethink Robotics Baxter [Guiz12] und der ABB YuMi [ABB15]. Beide sind speziell auf die Abläufe im produzierenden Gewerbe ausgelegt und daher für Pick-and-Place-Aufgaben optimiert.
Die Hauptgründe für die geringe Verbreitung von Robotern in Haushalt und KMU sind der hohe Preis, die Komplexität der Programmierung und das hohe Verletzungsrisiko durch die Roboterbewegung [Brem12]. Die traditionelle Programmierung von Robotern ist auf eine hohe Losgröße ausgelegt und lässt sich nicht einfach auf die Anforderungen in einem KMU übertragen. Nach [Hage02] lohnt sich eine Automatisierung ab einer Losgröße von etwa 10000 Stück pro Jahr. Da sowohl in Unternehmen als auch im Haushalt mit häufig wechselnden Prozessen und kleinen Losgrößen gerechnet werden muss, treibt eine komplexe und langwierige Programmierung von Robotern und Sensoranwendungen den Anpassungsaufwand in unangemessene Höhen [Denk05]. Zudem erlaubt eine kleine Betriebsgröße oft nicht die Beschäftigung eines oder mehrerer Robotik-Spezialisten, welche für die Projektierung und Umsetzung der entsprechenden roboterbasierten Lösungen nötig wären.
Dass die Programmierung von Robotern so kompliziert und zeitintensiv ist, liegt in erster Linie an der Art der Programmierung. In der Regel wird von einem Experten an einem Computer ein Programm geschrieben und anschließend werden die im Programm genutzten Positionen direkt am Roboter bestätigt bzw. korrigiert.
Bei dieser Vorgehensweise lassen sich zwei große Nachteile feststellen. Zum einen benötigt der Programmierer wesentliche Kompetenzen im Bereich der Roboterprogrammierung, was beispielsweise die Kenntnis einer geeigneten Programmiersprache und Wissen über den kinematischen Aufbau des Robotersystem beinhaltet. Zum anderen ist diese Art der Programmierung

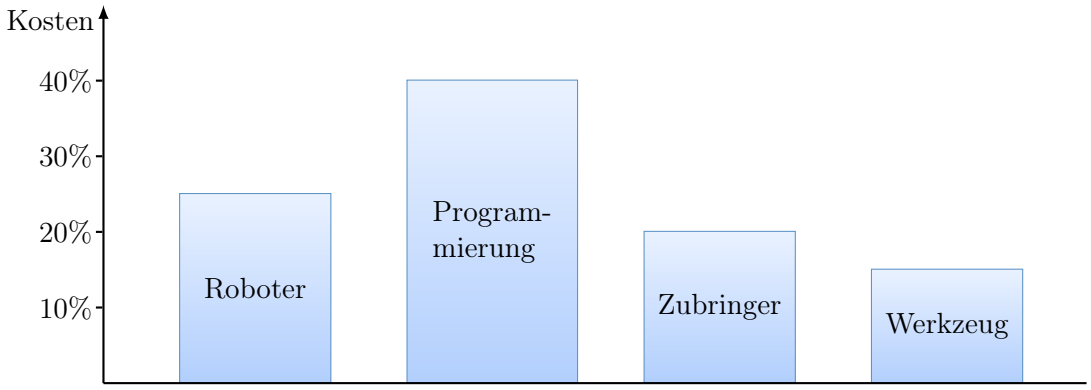


Abbildung 1.1: Die Aufteilung der Gesamtkosten produzierender Robotersysteme nach [Blac10] zeigt deutlich den hohen Anteil der Anwendungsprogrammierung. Diese liegt mit ca. 40 % vor dem Roboter selbst, den Zubringern wie beispielsweise Förderanlagen und den Roboterwerkzeugen, wie etwa Greifern oder Schweißgeräten.

selbst für Experten zeitaufwändig und erlaubt kein einfaches Wechseln zwischen Aufgabenstellung, was sich vor allem im Bereich der Kleinserienfertigung negativ auswirkt.
Des Weiteren werden Roboter üblicherweise aufgrund des hohen Verletzungsrisikos in speziellen abgeschlossenen Bereichen betrieben. Diese Bereiche sind für den Menschen nur zugänglich, sofern der Roboter still steht. Diese Art des Betriebs ist für kleine Unternehmen oft schwierig umzusetzen und schließt auch eine mögliche Mensch-Roboter Kooperation zur Entlastung von Arbeitern aus. Darüber hinaus ist sie für den Einsatz im Haushalt schlicht ungeeignet. Ein Betrieb ohne Sicherheitsvorkehrungen wäre jedoch aufgrund des Verletzungsrisikos ebenfalls nicht tragbar.

1.2 Allgemeine Forderungen

Um die Verbreitung und effiziente Nutzung von Robotern weiter zu stärken, lassen sich also drei Forderungen formulieren:

F1 Gewährleistung eines sicheren Arbeitsumfeldes

F2 Minimierung der Anschaffungskosten

F3 Minimierung des Programmieraufwandes

Der erste Punkt ist die Gewährleistung der Sicherheit von Personen im Umfeld eines Robotersystems. Industrieroboter sind von Haus aus nicht in der Lage mit Menschen auf engem Raum zu koexistieren. Die aufgrund der Bewegung erzeugten Kräfte in Kombination mit einer fehlenden Sensorik können ein hohes Verletzungsrisiko erzeugen [Hadd07]. Die Erhöhung der Sicherheit bei der Mensch-Roboter-Kooperation ist ein eigenständiger Forschungs- und Industriezweig. Ziel ist es, die Trennung der Bereiche des Roboters und des Menschen aufzuheben und eine sichere Koexistenz zu ermöglichen. Es existieren Lösungen mit externer Überwachungssensorik [Ober14, Kuhn12, Baer92], spezielle Regelungen für Robotersysteme [Albu07, Guiz12] oder auch künstliche Häute [Wink09, Kerp03] für Industrieroboter.

Ein weiterer Punkt ist die Senkung der Kosten. Diese liegen für einen Industrieroboter samt aller notwendigen Komponenten im Durchschnitt bei etwa $140000 und sind damit zwar sehr hoch, jedoch in den letzten Jahren deutlich gefallen [Robo14]. Zusätzlich drängen neue Hersteller mit günstigen Modellen auf den Markt, die vor allem für die Mensch-Roboter-Kooperation konzipiert sind und somit theoretisch auch für den Heimgebrauch tauglich sind [GomT15, Kino15, Robo15]. Alternativ existieren Bestrebungen zur günstigen Selbstproduktion von Robotern [Lang15].
Letztendlich können auf den Anschaffungspreis des Roboters nur die Roboterhersteller selbst direkt Einfluss nehmen. Allerdings machen etwa 40 Prozent der Kosten eines produzierenden Robotersystems die Programmierung aus (Abbildung 1.1), welche durch effiziente und intuitive Programmiermethoden reduziert werden können.
Dies führt zum letzten Punkt, der Reduzierung des Programmieraufwandes. Eine einfache, intuitive Methode zur Roboterprogrammierung muss das Ziel verfolgen, selbst nicht speziell geschultem Fachpersonal eine effiziente und fristgerechte Einrichtung des Robotersystems zu ermöglichen. Die klassischen Programmiermethoden werden bereits seit längerer Zeit ergänzt durch intuitivere Programmiermethoden, wie grafische Programmierung, modellbasierte Programmierung, Augmented Reality oder maschinellem Lernen. Allerdings ist die Entwicklung intuitiver Programmiermethoden noch immer ein eigener Forschungszweig mit hohem Einsatzpotential [Blac10, Dill94].

1.3 Ziele und Abgrenzung

Die Forderungen F1 und F2 werden im Rahmen dieser Arbeit nicht betrachtet. Wie bereits erwähnt, kann das Problem der Kosten prinzipiell nur durch die Roboterhersteller angegangen werden und bei der Gewährleistung der Sicherheit handelt es sich um einen eigenes komplexes Themengebiet, dessen Untersuchung hier nicht vorgenommen werden soll.
Im Rahmen dieser Arbeit wird ausschließlich die Forderung F3 (Minimierung des Programmieraufwandes) untersucht. Wie bereits erwähnt, resultiert der hohe Aufwand der Programmierung aus der Komplexität der Aufgabe und dem grundsätzlichen zeitlichen Bedarf in der Durchführung der Programmierung und Einrichtung. Die Lösung des Problems kann daher durch das Erreichen zweier unterschiedlicher Ziele erfolgen:

Z1 Reduzierung der Häufigkeit der Programmierung

Z2 Reduzierung der Komplexität der Programmierung einer Aufgabe

Ziel Z1 lässt sich realisieren, indem man sicherstellt, dass bereits einmal programmierte Aufgaben nicht erneut programmiert werden müssen. Die Möglichkeit online, d.h. ohne erneute Programmierung und Einrichtung auf verschiedene Situationen reagieren zu können, ist eine klassische Anwendungsdomäne für verhaltensbasierte Systeme. Diese können mehrere Verhalten (Programme) besitzen, die unterschiedliche Ziele verfolgen. Je nach Situation wird das passendste Verhalten online ausgewählt und ausgeführt, so dass eine erneute Programmierung oder Einrichtung des Roboters entfällt.
Ziel Z2 lässt sich realisieren, indem eine Methode zur intuitiven Programmierung des Roboters entwickelt wird, in welcher der Programmierer den Roboter im Idealfall einmalig durch die Aufgabe führt. Das System extrahiert aus dieser Demonstration die relevanten Merkmale und kann die Aufgabe auch unter geänderten Randbedingungen, wie beispielsweise veränderten

Objektpositionen oder einer veränderten Roboterkonfiguration ausführen. Der Aufwand für den Programmierer wird dadurch so gering wie nötig gehalten.

1.4 Kapitelübersicht

In Kapitel 2 und Kapitel 3 wird zunächst das Ziel Z1 betrachtet. Es werden Anforderungen an eine verhaltensbasierte Architektur ermittelt, die es ermöglicht mehrere Programme auf einem Industrieroboter quasiparallel zu betreiben. Dafür wird der aktuelle Stand der Forschung untersucht und hinsichtlich definierter Anforderungen bewertet. Anschließend werden Aspekte einer geeigneten Modellierung diskutiert und die Funktionsweise der parallelen Ausführung dargelegt.
In Kapitel 4 wird auf das Ziel Z2 eingegangen. Auch hier werden die Bedingungen für ein System ermittelt, das es dem Nutzer auf einfache Art und Weise ermöglichen soll, eine Aufgabe zu programmieren. Nach einem Überblick über den Stand der Forschung werden zwei Umsetzungsmöglichkeiten präsentiert.
In Kapitel 5 werden die Ansätze für Z1 und Z2 experimentell evaluiert. Dazu werden sowohl individuelle Experimente zu beiden Ansätzen durchgeführt, als auch das Zusammenspiel zwischen Programmierung und Ausführung betrachtet.
In Kapitel 6 werden die Ergebnisse der Arbeit zusammengefasst und es wird ein Ausblick auf weitere Forschungsarbeiten zur Optimierung und Erweiterung der entwickelten Methoden gegeben.

2 Modellierung von Verhalten

Verhaltensbasierte Systeme werden vor allem dort eingesetzt, wo verschiedene Module eines Systems gleichzeitig agieren müssen. Diese Fähigkeit qualifiziert sie direkt zur Umsetzung des Ziels Z1 aus Kapitel 1.3. Für das weitere Vorgehen muss jedoch geklärt werden, wie ein solches verhaltensbasiertes System definiert ist.
Mataric [Mata08] unterscheidet vier Arten der Robotersteuerungen: Deliberativ, reaktiv, hybrid und verhaltensbasiert. Die Einteilung ist nicht überschneidungsfrei und eine Zuordnung ist oft nicht eindeutig. Jedoch gibt sie einen guten Überblick über die unterschiedlichen Ansätze. Bei der *deliberativen* Steuerung verwendet der Roboter sowohl internes Wissen als auch äußere Sensorinformationen, um seine nächste Aktion zu planen. Die Verarbeitung folgt in der Regel einer streng linearen Ordnung (Abbildung 2.1 (a)) der Form „Erfassen - Planen - Ausführen“ und benötigt ein symbolisches Weltmodell [Albu91, Albu89, Fial87]. Die Bereitstellung einer Aktion kann aufgrund der umfangreichen Berechnungen viel Zeit in Anspruch nehmen, ist aber theoretisch sehr flexibel. [Mata08]

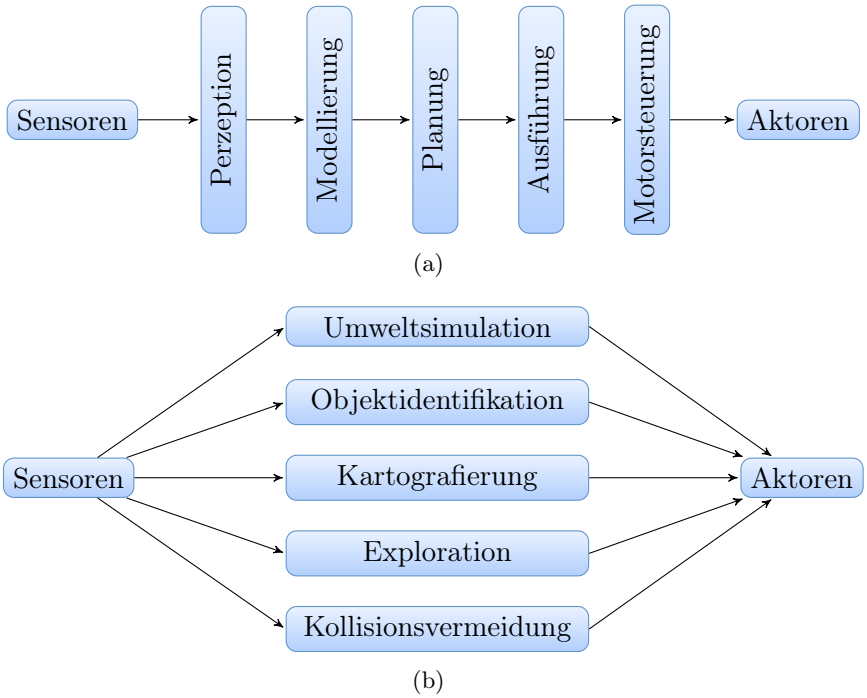


Abbildung 2.1: Architekturenvergleich nach [Broo86]: Im Gegensatz zur deliberativen Steuerung (a) haben bei der verhaltensbasierten Steuerung mehrere Unabhängige Komponenten Zugriff auf die Sensoren und Aktoren.

Im Gegensatz dazu besitzen *reaktive* Systeme eine enge Kopplung zwischen Sensoren und Aktoren nach einem Stimulus-Response Muster [Broo86]. Solche Systeme kommen vor allem in

unstrukturierten und dynamischen Umgebungen zum Einsatz, da sie eine schnelle, wenn auch unflexible, Reaktion auf äußere Reize ermöglichen. [Mata08]
Die *hybride* Steuerung versucht die Vorteile beider Methoden zu kombinieren, indem sich eine eine reaktive Komponente um die kurzfristigen Probleme, wie z.B. Kollisionsvermeidung kümmert und eine deliberative Komponente die längerfristigen Ziele, wie das Erfüllung von Aufgaben [Fier01, Qure04, Arki94a] übernimmt. Problematisch bei dieser Architektur ist die dazwischenliegende Synchronisationsschicht, denn auf deliberativer Ebene liegen die Informationen meist in symbolischer Form vor, während sie auf reaktiver Ebene in subsymbolischer Form vorliegen. [Mata08]
Ein *verhaltensbasiertes* System dagegen verwendet eine Menge an Komponenten, sogenannte *Verhalten*, die miteinander kommunizieren und interagieren können. Das Zusammenspiel all dieser Komponenten erzeugt das *Systemverhalten*, welches sich nicht unbedingt aus den individuellen Verhalten ablesen lässt [Arki98a, Mata97]. Wie bei reaktiven Steuerungen besitzt auch bei verhaltensbasierten Steuerungen jedes Verhalten gleichermaßen Zugriff auf alle Sensoren und Aktoren (Abbildung 2.1 (b)). Die Abgrenzung zu reaktiven Systemen ist fließend. Allerdings besitzen reaktive Systeme im Gegensatz zu verhaltensbasierten Systemen kurze wenn-dann-Anweisungen ohne inneren Zustand oder Repräsentation der Umwelt, so dass eine Aktion a nur von externen Stimulationen s abhängt $a = f(s)$. In verhaltensbasierten Systemen kann die Aktion der Verhalten auch vom inneren Zustand z abhängen $a = f(s, z)$, so dass prinzipiell auch eine Planung möglich ist. [Mata08]
Eine weitere Möglichkeit der Definition ist die Betrachtung von verhaltensbasierten Systemen als Multi-Agenten-Systeme. Hierbei stellt jedes Verhalten einen Agenten dar, der nach [Russ03] definiert wird als

> *„anything that can be viewed as perceiving its environment through sensors and acting upon that environment through effectors“.*

Nach dieser Definition muss ein Agent zwingend mit der Umwelt interagieren, was jedoch sogenannte versteckte und blinde Verhalten ausschließt (siehe Kapitel 2.4). Dagegen fallen alle Subsumption-basierten Systeme und deren Derivate nach der Definition von Mataric [Mata08] aus der Gruppe der verhaltensbasierten Systeme heraus, da sie keinen inneren Zustand besitzen. Allerdings werden gerade diese Systeme in der Literatur oft als Paradebeispiel eines verhaltensbasierten Systems genannt. Daher soll der Begriff in dieser Arbeit weiter gefasst werden.

Definition 1 (Verhaltensbasiertes System) Ein System wird als *verhaltensbasiert* bezeichnet, wenn es die folgenden drei Anforderungen $A1$ bis $A3$ erfüllt.

$A1$: Das System verfügt über mehrere Komponenten, die individuelle Ziele verfolgen können und Zugriff auf Sensoren und Aktoren des Systems besitzen.

$A2$: Das System ist in der Lage aufgrund von Sensorinformationen und/oder internem Zustand Einfluss auf die Umwelt und/oder seinen inneren Zustand zu nehmen.

$A3$: Das System ist durch die Wahl einer Untermenge aller vorhandenen Verhalten in der Lage, das beobachtbare Systemverhalten zu ändern, ohne dass eine Neuprogrammierung des Systems notwendig ist.

Mit dieser Definition wird ein breiter Bereich an Ansätzen abgedeckt und es fallen auch einige der reaktiven Systeme, wie [Broo86] in den Bereich der verhaltensbasierten Systeme. Die

Forderung $A1$ schränkt hierbei nicht ein, dass eine Komponente ihr Ziel nicht alleine erreichen kann und auch nicht, dass mehrere Verhalten gleiche Ziele besitzen können.

2.1 Aufgabenstellung

Mit dem langfristigen Ziel der Erhöhung des Einsatzes von Robotersystemen und der Reduzierung des Einrichtungsaufwandes soll in diesem Kapitel zunächst Ziel Z2 umgesetzt werden, indem die Häufigkeit der Programmierung gesenkt wird. Wie bereits erwähnt, soll dafür eine verhaltensbasierte Architektur zum Einsatz kommen. Für eine solche Architektur lassen sich die folgenden notwendigen Bedingungen identifizieren:

Flexibilität: Um sich den ständig ändernden Anforderungen anzupassen, muss das System in der Lage sein, verschiedene Aufgaben zu bewältigen. Dazu muss es möglich sein, den Zweck des Systems zu ändern, ohne eine komplette Neugestaltung dessen vorzunehmen. So besteht ein System zur Navigation zwar meist aus mehreren Verhalten mit unterschiedlichem Zweck, allerdings ist es in seinem Einsatz doch beschränkt auf einen Aufgabentyp.
Es muss also möglich sein, ohne großen Aufwand zwischen verschiedenen Aufgaben zu wechseln, wie beispielsweise dem Auftragen von Klebstoff auf ein Bauteil und dem Sortieren von Objekten.

Skalierung: Das System muss fähig sein, eine variable Anzahl von Verhalten zu verwalten. Da es möglich sein soll, dem System neue Aufgaben zu übertragen, muss sich die Anzahl auch während der Laufzeit dynamisch ändern können. Auch die maximale Anzahl der Verhalten sollte nicht beschränkt sein.

Unterbrechbarkeit: Um rechtzeitig auf ein sich änderndes Umfeld reagieren zu können und relevantere Aufgaben priorisieren zu können, müssen Aufgaben zugunsten von anderen hinreichend schnell unterbrochen werden können.

Konsistenz: Neben der Unterbrechung muss es möglich sein, die Ausführung eines Verhaltens an der Stelle der Unterbrechung weiterzuführen ohne zu gefährden, dass sich relevante Teile der Aufgabe geändert haben, so dass die Aufgabe nicht mehr erfüllbar ist. Es muss dem Roboter also nicht nur möglich sein, eine Aufgabe zugunsten einer anderen zu unterbrechen, sondern es muss auch möglich sein, den Zustand eines Verhaltens wiederherzustellen und damit die Aufgabe zu einem späteren Zeitpunkt wieder aufzunehmen und zu beenden.

Manipulation: Das System muss fähig sein, Manipulationen an der Umwelt durchführen zu können. Das bedeutet, die Verhalten sollen nicht nur den internen Zustand des Systems verändern können, sondern auch die Umwelt. Andernfalls würden sich die lösbaren Aufgaben auf eine reine Navigation und Inspektion der Umwelt beschränken.

Jeder dieser Bedingungen ist notwendig zur Umsetzung einer flexiblen Architektur, die es einem Roboter ermöglicht, nützliche Arbeit in einem Umfeld mit ständig wechselnden Aufgaben zu verrichten.
Es soll daher im Folgenden ein Überblick über den Stand der Forschung im Bereich verhaltensbasierter Systeme gegeben werden, der anschließend anhand dieser Bedingungen bewertet wird.

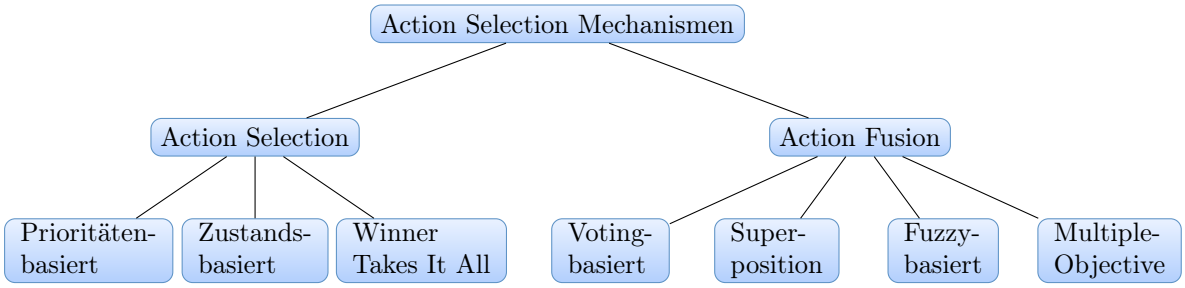


Abbildung 2.2: Umsetzungsmöglichkeiten einer Verhaltenskoordination nach [Pirj99a]

2.2 Stand der Forschung

In verhaltensbasierten Systemen erfolgt die Steuerung des Roboters durch mehrere zweckgerichtete Einheiten (Verhalten). Jedes Verhalten generiert Reaktionen auf seine sensorische Wahrnehmung der Umwelt, um ein bestimmtes Ziel zu erreichen. Da die Ziele einzelner Verhalten nicht gleich oder sogar unvereinbar sind, muss ein entsprechender Mechanismus zur Koordination der Verhalten existieren. Die Auswahl der nächsten Aktion und die Kombination mehrerer Aktionen ist bekannt als *Action Selection Problem* [Maes89]. Ansätze zur Lösungen dieses Problems lassen sich in den verschiedenen *Action Selection Mechanismen* (ASM) finden. Nach [MacK95] und [Pirj99a] lassen sich diese unterteilen in *Action Fusion* und *Action Selection* (Abbildung 2.2). Zusätzlich existieren hybride Ansätze, welche das dynamische Wechseln zwischen Action Fusion und Action Selection erlauben, allerdings stellen diese ausschließlich theoretische Arbeiten dar [Sche04].
Bei der Action Selection, auch *Action Arbitration* genannt, ist zu einem Zeitpunkt immer nur ein Verhalten aktiv. Nach einer bestimmten Zeitspanne wird ein anderes Verhalten aktiviert, welches die alleinige Kontrolle über das System übernimmt. In der Regel lassen sich komplette Aufgaben mit einem einzigen Verhalten durchführen. Vorteile der Action Selection Technik liegen in einer einfachen Modellierung und dem Wechseln von Zielen. Dies hat eine hohe Skalierbarkeit und Flexibilität zur Folge.
Bei der Action Fusion, auch *Command Fusion* genannt, tragen permanent mehrere bzw. alle Verhalten zur Kontrolle des Roboters bei. Die Ausgaben der Verhalten werden kombiniert und anschließend wird eine Aktion gewählt, die vom Roboter ausgeführt wird [Pirj99a]. Der Vorteil der Action Fusion Mechanismen liegt potentiell in der gleichzeitigen Einhaltung mehrerer Ziele und in der Regel in der Schnelligkeit der Reaktion. Ein Umschalten zwischen Verhalten entfällt, da die Ausgabe kontinuierlich erfolgt. Die Skalierbarkeit ist eingeschränkt, da eine sinnvolle Kombination bei einer immer größer werdenden Anzahl Verhalten immer schwieriger wird. Auch die explizite Angabe von Zielen ist meist erschwert, da das Systemverhalten immer aus der kontinuierlichen Ausgabe mehrerer Verhalten resultiert.
Aufbauend auf der Einteilung der Action Selection Mechanismen nach [Pirj99a] soll der Stand der Forschung untersucht werden.

2.2.1 Action Selection

Der Bereich der Action Selection lässt sich in drei Klassen unterteilen: die *prioritätenbasierte Action Selection*, die *zustandsbasierte Action Selection* und die *Winner-Takes-It-All*-Strategie. Bei der Prioritäten-basierten Action Selection sind die Prioritäten der Verhalten statisch und

a priori festgelegt. Im Gegensatz dazu sind bei der *zustandsbasierten Action Selection* die Prioriäten dynamisch und verändern sich je nach aktuellem Systemzustand. Bei der *Winner-Takes-It-All* Strategie gibt es in der Regel keine Ordnung unter den Verhalten. Alle Verhalten sind zunächst gleichberechtigt. Die Prioritäten ändern sich nicht aufgrund eines inneren Zustands, sondern lediglich situationsbedingt durch äußerer Einflüsse.

Prioritäten-basierte Action Selection

Bei der Prioritäten-basierten Action Selection besitzen die Verhalten eine feste Rangordnung, die sich während der Laufzeit nicht ändert.

Subsumption Architektur Der bekannteste Vertreter dieser Gruppe ist die 1986 von Rodney Brooks vorgestellte Subsumption Architektur [Broo86]. Die Verhalten werden durch spezielle endliche Zustandsautomaten (AFSM) modelliert, die in einem hierarchischem Netzwerk organisiert sind (sog. Kompetenzebenen). Verhalten einer höheren Ebene können die Ausgabe von Verhalten einer niedrigeren kurzzeitig zurückhalten oder vollständig überschreiben. Die Struktur des Netzwerks muss vom Systemarchitekten erstellt werden und kann sich zur Laufzeit nicht verändern.
Die Subsumption Architektur ist im mobilen Bereich sehr verbreitet [Broo97, Edma15] und wird dort auch als Erweiterung zu fuzzy-basierten Systemen verwendet [Wica09]. Die Kompetenzebenen sind meist sehr ähnlich gewählt (Abbildung 2.1). Neben der Navigation findet die Subsumption Architektur auch Einsatz bei der Gang-Synthese von Hexapods [Ferr95] oder Humanoiden [Luks08].
Es lassen sich auch komplexe Systeme damit realisieren, wie z.B. eine kraftgeregelte Hand mit taktiler Sensorik[Park06] oder den Humanoiden Domo. Dieser ist in der Lage verschiedene Objekte mit beiden Händen ineinander zu stecken [Edsi08a] oder Bücher in einem Regal zu platzieren [Edsi08b]. Die Vorteile der Architektur liegen in der Robustheit und dem einfachen Funktionsprinzip, das die Erstellung komplex wirkender Systeme durch wenige Verhalten erlaubt.
Die Nachteile dieser Architektur liegen in der mangelnden Flexibilität. Die Mechanismen zur Unterdrückung eines Verhaltens durch ein anderes können nicht dynamisch geändert werden. Soll sich der Einsatzzweck des Systems ändern, muss es komplett neu gestaltet werden. Außerdem lässt sich zeigen, dass die Anzahl der resultierenden Verhalten rasant steigen kann, wenn sich die Anzahl der Kompetenzebenen erhöht, was die Vorhersehbarkeit des Systems erschwert [Ande90, Tsot95, Naka98].

Circuit Architektur Die Circuit Architektur [Bona91, Kael86, Kael91] erweitert die Subsumption Architektur um die Darstellung der Verhalten durch eine Prädikatenlogik und die Möglichkeit mehrere Verhalten zu Gruppen zusammenzufassen, so dass die Action Selection in jeder Gruppe erfolgen kann. Durch diesen Ansatz wird die Modularität des Systems unterstützt, da Gruppen leichter entfernt oder hinzugefügt werden können. Gleichzeitig bietet er dennoch eine enge Kopplung zwischen Sensoren und Aktoren [Wool09, Arki98a].

Colony Architektur Bei der Colony Architektur [Arki98a] handelt es sich um eine Abwandlung der Subsumption Architektur. Während bei der Subsumption Architektur eine strikt

hierarchische Ordnung aller Verhalten existiert, können bei der Colony Architektur mehrere Verhalten die gleiche Priorität besitzen, was zu einer baumartigen Struktur führen kann [Arki98a]. Eine Umsetzung findet man z.B. im mobilen Manipulator Herbert, der Getränkedosen am Massachusetts Institute of Technology sammelt [Conn89b, Conn89a].

Weitere Ansätze Neben den bekannten Vertretern existieren unzählige weitere Arbeiten von denen nur einige exemplarisch genannt werden. So kann in der Arbeit von Doroftei [Doro09] ein autonomes Fahrzeug die Verhalten zur Navigation zu Gunsten einer Mineninspektion unterbrechen. Ein ähnliches Prinzip wird in [Jung96] zur Implementierung eines Wall Followers verwendet. Während bei den meisten Arbeiten die Prioritäten von Hand vergeben werden, beschäftigt sich [Eskr07] mit der automatischen Ermittlung der Prioritäten über maschinelle Lernverfahren.

Zustandsbasierte Verhaltensauswahl

Bei dieser Gruppe findet die Auswahl des aktiven Verhaltens aufgrund eines internen Systemzustands statt, so dass auch bei gleichen Umwelteinflüssen, unterschiedliche Reaktionen des Systems beobachtbar sind.

Systeme mit endlichen Zustandsautomaten Bei den *Discrete Event Systems* (*DES*) wird das Gesamtsystem durch einen endlichen Zustandsautomaten modelliert, in welchem die Zustände Verhalten darstellen. Damit spiegelt der aktuelle Zustand das gerade aktive Verhalten wider. Die Action Selection findet durch Zustandsübergänge statt, die aufgrund verschiedener Ereignisse ausgelöst werden. Oft ist ein zweiter Automat vorhanden, der die korrekte Ausführung der Ziele der Verhalten durch eine Closed-Loop Regelung sichert [Pirj99a].
Die DES finden ihre Anwendung vor allem in der Navigation mobiler Roboter [Kose93]. Es existieren Erweiterungen, die durch Fuzzifizierung des aktuellen Zustandes die Kombination mehrerer Verhalten zu einem Zeitpunkt ermöglichen [Huq06] und somit eine Vermischung aus Action Fusion und Action Arbitration darstellen. Eine Erweiterung um nachgeschaltete Neuronale Netze ermöglicht eine differenziertere Reaktion basierend auf dem internem Zustand und der Vielzahl an äußeren Reizen [Gada03].
Einen ähnlichen Ansatz wie die DES bildet das *Temporal Sequencing* bzw. *Perceptual Sequencing* zurückgehend auf Ronald Arkin [Arki94b, Arki97]. Die Zustände repräsentieren sogenannte Motor Schemas [Arki89] (siehe Kapitel 2.2.2) und die Übergänge entsprechen bestimmten Bedingungen, die erfüllt sein müssen, um den Zustand zu wechseln. Auch das Temporal Sequencing wird hauptsächlich für mobile Roboter zur Navigation [Arki90, MacK95] oder für Box-Pushing Tasks [Emer01] verwendet.

Hybride Automaten Als eine Erweiterung lassen sich die Ansätze mittels hybrider Automaten sehen, durch die eine diskret-kontinuierliche Regelung umgesetzt wird. Verhalten stellen dabei kontinuierliche Regelungen mit verschiedener Führungsgröße dar. Durch Wechsel in einen anderen Zustand des Automaten wird eine andere Regelstrategie aktiviert [Eger99, Eger00].
Einen ähnlichen Ansatz verfolgt das Deutsche Zentrum für Luft- und Raumfahrt, bei dem das System je nach Zustand jedoch nicht nur die Parameter der Regelstrategien wechseln, sondern

das komplette Regelverhalten des Roboters [Paru11]. So wird beispielsweise bei Kontakt mit einem Hindernis der Roboter in einen sehr weichen Impedanzmodus geschaltet, um Schäden zu verhindern.

Prädikatenlogik Eine weitere Untergruppe bilden die Systeme, die nicht auf einen endlichen Automaten zurückgreifen, sondern auf die Koordination mittels Prädikatenlogik. Verhalten besitzen dabei Vorbedingungen, die zur Ausführung notwendig sind und Nachbedingungen, die nach deren Beendigung gültig werden. Durch die Menge der Vor-und Nachbedingungen lassen sich Abhängigkeitsgraphen erstellen, aus denen ersichtlich wird, welche Verhalten unter welchen Bedingungen ausführbar sind [Nico02, Nico03].

Winner-Takes-It-All

Bei den Winner-Takes-It-All Architekturen werden zu keiner Zeit bestimmte Verhalten durch eine organisatorische Schicht von ihrer Ausführung abgehalten, wie dies z.B. bei den zustandsbasierten Ansätzen der Fall ist. Vielmehr konkurrieren alle Verhalten gleichermaßen darum ausgeführt zu werden, bis sich schließlich eines durchsetzt und eine Aktion ausführt.

Activation Networks Bei den *Activation Networks* [Maes90, Maes93] sind die Verhalten als Tupel modelliert, die in einem Netzwerk angeordnet sind. Diese Tupel beinhalten eine Menge von Vorbedingungen zur Aktivierung des Verhaltens, eine Menge von Aktionen und eine Aktivierungsenergie. Durch die Anordnung im Netzwerk können Aktionen als Vorbedingungen für weitere Verhalten dienen und ein Verhalten kann einen Teil seiner Aktivierungsenergie auf die mit ihm verknüpften Verhalten übertragen. Überschreitet die Energie eines Verhaltens eine bestimmte Schranke und sind alle Vorbedingungen erfüllt, wird das Verhalten aktiv und führt eine Aktion aus.
Die ursprüngliche Architektur von Maes leidet bei mehreren Zielen unter Entscheidungsschwierigkeiten. Es folgten daher Erweiterungen um eine hierarchische Anordnung der Verhalten [Decu98] und um eine Lernkomponente, die versucht die Konsequenzen einer Aktion abzuschätzen [Maes91]. Damit sollte eine differenziertere Auswahl des Verhaltens unter Berücksichtigung der Folgen ermöglicht werden.

Sonstige Ansätze Eine weitere Möglichkeit zur Auswahl des aktiven Verhaltens besteht darin, das Verhalten zu wählen, welches einen bestimmten Ausgabewert maximiert (*Reward*). Dieser Wert kann z.B. die maximal erzeugbare Geschwindigkeit in einem potentialfeldbasierten Navigationssystem eines mobilen Roboters sein [Laue05] oder das maximale Gebot für eine Aktion bei Verwendung des Auktions-Schemas [Towl14]. Auch die Einbeziehung des internen Zustands des Roboters kann eine tragende Rolle bei der Auswahl des Verhaltens spielen, indem emotionale Komponenten modelliert werden und das System durch Wahl der Verhalten die Frustration minimiert [Sche02].
Einen Schritt weiter gehen die Ansätze, die Optimierungsverfahren einsetzen um den geschätzten Reward einer Aktion zu maximieren, indem die Eintrittswahrscheinlichkeit bestimmter Ereignisse durch die erzeugte Aktion berücksichtigt wird [Pine02].

2.2.2 Action Fusion

Bei der Action Fusion lassen sich vier Ansätze zur Generierung eines Systemverhaltens unterscheiden. Bei den Voting-basierten Ansätzen gibt es eine begrenzte Menge an möglichen Aktionen. Die Verhalten können für eine oder mehrere dieser Aktionen abstimmen und die Aktion mit der größten Zustimmung wird anschließend ausgeführt. Bei den Superpositions-basierten Ansätzen gibt es keine diskrete Menge an Aktionen. Stattdessen werden die kontinuierlichen Ausgaben der einzelnen Verhalten aufsummiert und dieses Ergebnis ausgeführt. Bei den Fuzzy-basierten Ansätzen werden die Ausgaben nicht einfach aufsummiert, sondern durch den Umweg über eine unscharfe Darstellung und eine anschließende Schlussfolgerung (approximatives Schließen) systematisch kombiniert. Noch einen Schritt weiter gehen die Multigoal-basierten Ansätze. Hier werden die Vorschläge aller Verhalten berücksichtigt und die resultierende Aktion durch ein Optimierungsverfahren gewählt.

Voting-basierte Ansätze

Ein bekannter Vertreter dieser Kategorie ist z.B. die *Distributed Architecture for Mobile Navigation* (*DAMN*). Alle Verhalten geben Stimmen zu den beeinflussbaren Größen des Systems ab, wie beispielsweise Richtung und Geschwindigkeit. Anschließend werden die Stimmen normalisiert und die so resultierende Aktion ausgeführt [Rose97, Rose95, Kwon12].
Eine Erweiterung um sogenannte *Utility Functions* lässt die Verhalten nicht direkt die Aktionen wählen. Stattdessen wird eine Hilfsmetrik verwendet, welche den Nutzen zukünftiger Weltzustände angibt [Rose00]. Anschließend werden die Aktionen ausgeführt, welche die Wahrscheinlichkeit für den von allen Verhalten favorisierten Weltzustand maximieren.
Die *SAMBA* Architektur baut auf DAMN auf [Riek97, Kuni94, Riek95]. Sie besitzt sowohl eine reaktive Schicht, welche die Sensoren und Aktoren direkt miteinander verbindet, als auch eine zusätzliche Schicht, in der Aufgaben auf höherer Ebene mittels sogenannter *Action Maps* koordiniert werden. Diese geben zu jeder möglichen Aktion an, wie nützlich sie für jedes Verhalten ist. Zum Schluss wird die Aktion ausgeführt, welche von der Mehrheit der Verhalten als nützlich angesehen wird.
Bei der Umsetzungen mit Neuronalen Netzen sind die Verhalten durch ein solches mit einer Ausgabeschicht verbunden, welche die möglichen Aktionen des Systems darstellt. Die Neuronen in der Ausgabeschicht besitzen einen festen Schwellwert, der überwunden werden muss, um die Aktion zu legitimieren [Hoff95, Fagg94]. Die Verwendung neuronaler Netze erfolgt beispielsweise zur Positionierung mobiler Roboter [Hoff04] oder auch für Pick-and-Place-Aufgaben [Pirj98b].

Superposition

Bei der Superposition werden die Ausgaben der einzelnen Verhalten überlagert. In der Regel geschieht dies durch komponentenweise Addition der Ausgabevektoren der einzelnen Verhalten.
Viele der Ansätze basieren auf der Idee, die Umwelt durch anziehende und abstoßende Potentialfelder für Verhalten zu modellieren [Conn92, Lato91, Khat85]. Durch Superposition der Potentialfelder lässt sich anschließend mit Hilfe eines Gradientenabstiegs eine resultierende Aktion ermitteln.
Die Potentialfeldmethode wurde durch die Gruppe um R.C. Arkin um sogenannte *Motor*

Schemas [Arki97] erweitert, die eine taskspezifische Parametrisierung der Potentialfelder ermöglichen, um so die Ziele der Verhalten zu modellieren [Pirj99a]. Potentialfeldmethoden und Motor Schemas sind in der mobilen Robotik verbreitet, wo sie zur Navigation und Kollisionsvermeidung [Arki89, Olen05, Balc93], zur mobilen Manipulation [Came93] und zum Formationsfahren bei Multirobotersystemen [Balc00, Arki98b, Balc98, Balc95] eingesetzt werden.
Eine andere Möglichkeit der Superposition besteht darin, jeweils einem Verhalten die Kontrolle über einen Freiheitsgrad des Systems zu überlassen, wie etwa einem Gelenk eines Manipulators [Pluz12].
Eine weitere Kategorie bilden die *dynamischen Systeme* [Scho92]. Die Verhaltensmodellierung erfolgt als Differenzialgleichungssystem, das den Zusammenhang zwischen einer Verhaltensvariable ϕ und deren zeitlicher Änderung darstellt $\dot{\phi} = f(\phi)$. Bei ϕ kann es sich z.B. um die Geschwindigkeit oder die Ausrichtung des Roboters handeln [Mata98, Stei00b, Stei98b, Stei98a, Menz00, Waar03a, Waar03b, Grim10]. Die Verhalten geben also nicht konkrete Ziele an, sondern die gewünschte Änderung einer Zustandsvariablen des Roboters. Es existieren Erweiterungen um gewichtete Verhalten [Larg99], da die ursprüngliche Fassung oft Entscheidungsprobleme besitzt. Um zeitliche Abfolgen von Verhalten besser koordinieren zu können, gibt es Erweiterungen des Ansatzes um eine sogenannte „Unterdrückungs- und Anforderungsmatrix“ [Stei00a] oder um einen zusätzlichen Zustandsautomaten zur Sicherstellung sequentieller Abläufe [Blum01].

Fuzzy Action Selection

Bei der *Fuzzy Action Selection* sind die Verhalten als wenn-dann-Regeln modelliert. Die Kombination erfolgt durch approximatives Schließen aus den Zugehörigkeitsfunktionen und den Aktivierungsgraden. Anschließend erfolgt beispielsweise über die Schwerpunktmethode eine Defuzzifizierung, womit eine konkrete Aktion erzeugt wird.
Diese Methode wird vor allem für mobile Roboter zur Navigation in unbekanntem Gelände eingesetzt [Agui00, Bona06, Saff97b, Park07, Kova04, Saff97a, Lian11, Dong11, Inno07, Thon00] oder zum Durchlaufen von Labyrinthen [Rusu03].
Die Regelmengen können eine zusätzliche dynamische Gewichtung erfahren [Sera02] oder für Mehrrobotersysteme ausgelegt werden [Vada07]. Alternativ kann die Auswahl der relevanten Regeln durch die Kombination mit Neuronalen Netzen [Li97], durch Verwendung von Optimierungen [Jaaf07] oder durch Kombination mit der Subsumption Architektur [Wica09] erweitert werden.
Die Anwendbarkeit von Fuzzy-Regeln wurde auch bei Manipulatoren untersucht [Dass01]. Wobei dort in der Regel zusätzliche Vorkehrungen zur Sequentialisierung von Regelmengen getroffen werden, wie die Anordnung in Zustandsautomaten [Wasi02, Wasi03] damit die Aufgaben zielgerichtet erfüllt werden können.

Multiple-Objective

Bei der Multiple-Objective-Koordination wird versucht aus einer Menge von alternativen Aktionen eine Untermenge auszuwählen, die den Zielen aller Verhalten am besten entspricht. In der Regel geschieht dies, indem der Wert einer gewählten Funktion (Reward) durch die geeignete Wahl einer Aktionsmenge maximiert wird. Grundsätzlich lassen sich hier die Ansätze anhand ihrer Vorgehensweisen in konservativ und nicht-konservativ unterteilen.

Konservative Ansätze Die Gruppe der konservativen Ansätze versucht die Ziele der Verhalten zu vereinen, ohne dabei die individuelle Aktion eines der Verhalten einzuschränken. Ziele sind in diesem Fall nur vereinbar, solange sie nicht gegensätzlich sind. Ist dies nicht möglich, wird die Priorität eines der Verhalten situationsabhängig erhöht. Beispielsweise werden bei der Maximierung des gesamten Rewards Untergruppen gesucht, in denen ausschließlich nicht konfliktbehaftete Verhalten kombiniert werden [Lens02].
Eine ähnliche Möglichkeit besteht in sogenannten *Nullspace*-basierten Ansätzen. Ein Verhalten A wird in den Nullspace [Chia97] eines weiteren Verhaltens B abgebildet, d.h. Verhalten A kann nur die Freiheitsgrade nutzen, die von dem bereits aktiven Verhalten B nicht blockiert werden. Somit sind bei N Freiheitsgraden maximal N Verhalten gleichzeitig ausführbar. Anwendung findet diese Technik sowohl bei einzelnen Robotern [Anto07, Chia91] als auch bei Robotergruppen [Yang03]. Nach dem gleichen Prinzip arbeitet auch der Constrain-Based-Behavior-Fusion-Mechanism (CBFM) [Schu08, Huan08, De S88].
Eine ähnliche Technik verwendet nicht die verfügbaren Freiheitsgrade eines Systems, sondern dessen verfügbare Ressourcen. Die Verhalten werden parallel ausgeführt, sofern sie nicht die selben Ressourcen verwenden [Taip09, Fuji03].

Nicht-Konservative Ansätze Nicht-konservative Ansätze versuchen die bestmöglichen Aktionen basierend auf allen Zielen der Verhalten umzusetzen. Hier können auch die Ausgaben von Verhalten mit gegensätzlichen Zielen kombiniert werden, wodurch sich deren individuelle Ausgabe verändern oder sogar aufheben kann. Lösungen entspringen der Multiple-Objective-Entscheidungstheorie und basieren in der Regel auf dem Finden einer Lösung, die bestimmte Mindesteigenschaften erfüllt [Simo60, Spra04] oder auf dem Pareto-Optimum aller Ausgaben [Pirj98b, Pirj99b, Pirj98a, Pirj00]. Der Ansatz kann auch um eine Fuzzy-Komponente [Dria01] [Pirj99b, Noji09] sowie auf mehrere Roboter erweitert werden [Kubo03]. Auch das Betrachten einer zeitlich versetzten Ausführung [Good92] kann die Möglichkeiten einer Optimierung erweitern.

2.2.3 Bewertung

Ausgehend von den in Kapitel 2.1 festgelegten Merkmalen, die ein System erfüllen muss, um die Ziele aus Kapitel 1.3 zu erreichen, sollen die bisher gesammelten Ansätze bewertet werden (Tabelle 2.1 und Tabelle 2.2).

Flexibilität: Viele der Systeme sind für einen bestimmten Zweck entworfen. Obwohl mehrere Verhalten parallel agieren, sind sie trotzdem so aufeinander abgestimmt, dass das System nur sehr wenige Aufgaben tatsächlich erfüllen kann. Darunter fallen alle Systeme, die der reinen Navigation dienen. Auch solche mit einer festen Struktur, wie beispielsweise die Subsumption Architektur, müssten komplett neu entworfen werden, um andere Aufgaben durchzuführen. Die Systeme, die prinzipiell einen flexiblen Einsatz in neuen Aufgabenbereichen ermöglichen stammen aus den Bereichen Winner-Takes-It-All, Fuzzy Action Fusion und Multiple-Objective Action Fusion.

Action Arbitration		Flexibilität	Skalierung	Unterbrechbarkeit	Manipulation	Konsistenz
Priority Based	[Broo86, Broo97, Ferr95, Luks08, Doro09, Jung96, Eskr07]	−	−	+	−	−
	[Park06, Edsi08a, Edsi08b, Conn89a, Conn89b]	−	−	+	+	−
	[Kael86, Kael91, Wool09]	−	o	+	−	−
	[Bona91]	−	o	+	o	−
	[Naka98]	−	−	+	o	−
Zustandsbasiert	[Kose93]	−	o	o	−	−
	[Huq06, Gada03, Arki90, MacK95]	−	−	+	−	−
	[Arki94b, Arki97, Arki89, Paru11]	−	o	+	−	−
	[Emer01]	−	o	o	o	−
	[Eger99, Eger00]	−	o	+	−	−
	[Nico02, Nico03]	o	o	−	o	−
Winner takes all	[Maes90, Maes93, Sche02]	o	−	+	o	−
	[Laue05, Towl14]	−	+	+	−	−
	[Decu98, Maes91]	o	o	o	o	−
	[Pine02]	+	+	−	o	−

Tabelle 2.1: Bewertung des Forschungsstandes im Bereich Action Arbitration nach den in Kapitel 2.1 festgelegten Merkmalen. Hierbei bezeichnet + eine hohe Unterstützung des Merkmals, o eine prinzipiell mögliche Unterstützung und − eine geringe Unterstützung.

Skalierung: Die meisten Systeme aus dem Bereich Action Fusion besitzen Probleme mit der Skalierung, also der Verwaltung einer größeren Anzahl von Verhalten, da die Kombinationsmöglichkeiten sehr groß werden. Bei den Multiple-Objective Systemen wächst der Aufwand zur Optimierung des Rewards bzw. dem Finden konfliktfreier Untergruppen. Ebenso steigt das Konfliktpotential bei einer deutlich wachsenden Menge an Regeln für die Fuzzy Systeme. Zwar lassen sich die Ausgaben bei der Superposition auch bei einer großen Zahl von Teilnehmern einfach überlagern, jedoch verliert die so resultierende Ausgabe immer mehr an Sinn. Durch zusätzliche Widersprüche in den Verhalten kann dies sogar zur Lähmung des Systems führen. Ähnlich ist es bei den meisten Voting-basierten Verfahren. Bei der Subsumption Architektur konnte bereits gezeigt werden, dass eine hohe Anzahl an Verhalten die Menge der resultierenden Systemverhalten exponentiell ansteigen lässt. Hierdurch steigt die Wahrscheinlichkeit, das der Roboter ein nicht erwünschtes Systemverhalten aufzeigt.

Unterbrechbarkeit: Unterbrechbarkeit findet man hauptsächlich bei den mobilen Systemen. Sofern die Umwelt nicht manipuliert wird, können Verhalten jederzeit problemlos unterbrochen und wieder aufgenommen werden. Dies wird durch die kontinuierliche Ausgabe

Action Fusion		Flexibilität	Skalierung	Unterbrechbarkeit	Manipulation	Konsistenz
Voting	[Rose97, Rose95, Kwon12, Rose00, Riek97, Riek95]	−	o	+	−	−
	[Kuni94]	−	−	o	−	−
	[Pirj98b]	−	o	+	o	−
	[Hoff95, Fagg94, Hoff04]	−	−	+	−	−
Superposition	[Conn92, Lato91, Khat85, Arki97, Arki89, Olen05, Balc93, Balc00, Arki89, Balc98, Balc95, Stei00b, Stei98b, Stei98a, Menz00, Grim10, Larg99, Stei00a]	−	−	+	−	−
	[Came93]	−	−	−	o	−
	[Waar03b, Waar03a, Pluz12]	o	−	+	+	−
	[Blum01, Pluz12]	−	o	+	−	−
Fuzzy	[Agui00, Bona06, Saff97b, Park07, Kova04, Saff97a, Lian11, Dong11, Inno07, Thon00, Rusu03, Sera02, Li97, Jaaf07, Wica09, Vada07]	o	−	+	−	−
	[Dass01, Wasi02, Wasi03]	o	−	+	+	−
Multiple-Objective	[Simo60, Spra04, Taip09]	o	o	−	−	−
	[Lens02, Schu08, Huan08, De S88]	−	+	+	o	−
	[Pirj98b, Pirj98a, Pirj00, Pirj99b]	o	o	−	+	−
	[Dria01]	−	−	+	−	−
	[Noji09, Kubo03, Good92]	o	o	+	−	−
	[Chia91]	−	o	+	o	−
	[Anto07]	o	o	+	o	−
	[Yang03]	−	o	+	−	−
	[Fuji03]	o	+	+	−	−
Dieser Ansatz	[Grot13]	+	+	+	+	+

Tabelle 2.2: Bewertung des Forschungsstandes im Bereich Action Fusion nach den in Kapitel 2.1 festgelegten Merkmalen. Hierbei bezeichnet + eine hohe Unterstützung des Merkmals, o eine prinzipiell mögliche Unterstützung und − eine geringe Unterstützung.

der Verhalten im Bereich der Action Fusion noch begünstigt. Bei den wenigen Manipulatoren sind die Verhalten so zugeschnitten, dass das System nur sehr wenige resultierende Systemverhalten besitzt und damit nur eine sehr kleine Menge von Aufgaben lösen kann. Eine komplette Manipulationsaufgabe kann in keinem der o.g. Ansätze zu Gunsten einer weiteren Manipulation unterbrochen und konsistent wieder aufgenommen werden.

Manipulation: Die meisten verhaltensbasierten Ansätze werden immer noch zur Navigation verwendet. Die wenigen verbleibenden System sind in der Regel nicht in der Lage, ein Verhalten zu unterbrechen. Wenn doch, sind sie nicht in der Lage eine größere Menge von verschiedenen Manipulationen durchzuführen, sondern meist genau eine als Resultat ihrer Verhalten.

Aus der Bewertung des Forschungsstandes wird deutlich, dass das Haupteinsatzgebiet für verhaltensbasierte Systeme nach wie vor im Bereich der mobilen Robotik liegt. Dort werden sie vor allem bei hochdynamischen und unbekannten Umgebungen eingesetzt. Außerdem besitzen verhaltensbasierte Systeme vor allem im Bereich der Action Selection Mechanismen hervorragende Skalierungspotentiale.
Die Bewertung zeigt allerdings deutlich, dass kein Ansatz existiert, der alle Bedingungen aus Kapitel 2.1 erfüllt. Keines der betrachteten Systeme ist in der Lage mehrere verschiedene Manipulationsaufgaben durchzuführen, die auch online geändert werden können, und das in der Lage ist, eine Manipulation zugunsten einer anderen zu unterbrechen und anschließend konsistent wiederaufzunehmen. Die Skalierbarkeit ist vor allem für Systeme im Bereich Action Fusion ein Problem. Dabei ist sie zentraler Bestandteil eines lernenden Systems. Wie in Kapitel 2.1 gefordert, soll dem System das Erlernen neuer Aufgaben ermöglicht werden, indem die Lösung einmal demonstriert wird, also indem neue Verhalten hinzugefügt werden.
Da keiner der Ansätze die Anforderungen erfüllt oder Potential zur Erweiterung auf alle Merkmale bietet, besteht Forschungsbedarf für die Entwicklung einer Lösung, die allen Anforderungen gerecht wird.

2.2.4 Abgrenzung

Die Ausrichtung des hier entwickelten verhaltensbasierten Systems richtet sich explizit an die Verwendung von stationären Roboter-Armen und bietet in dieser Form keine direkte Verwendbarkeit für mobile Roboter.
Wie bereits in Kapitel 2.1 angesprochen, wird auch der Sicherheitsaspekt hier nicht betrachtet. Das heißt, dass unter anderem keine Kollisionserkennung zwischen Roboter und Umwelt durchgeführt wird und auch nicht auf Selbstkollision geprüft wird. Diese wären mit bekannten Methoden einfach integrierbar.
Bei den durchführbaren Aufgaben handelt es sich um statische Handhabungsaufgaben ohne Berücksichtigung der Dynamik. D.h. nur die Dauer, jedoch nicht das Ergebnis einer Aufgabe hängt von der Geschwindigkeit der Ausführung ab. Zusätzlich wird erwartet, dass durchgeführte Aufgaben erfolgreich sind. Es existiert keine zusätzliche Regelung oder Fehlererkennung, die die Robustheit des Systems garantiert. Allerdings wird eine mögliche Erweiterung im Ausblick angesprochen.
Laut [Sche02] ist es von Vorteil wenn die einzelnen Verhalten direkt Einfluss auf den Verhaltensauswahlmechanismus haben, um diesen beispielsweise durch Lernverfahren zu verbessern. Es handelt sich zwar insgesamt um ein lernendes System, allerdings nur im Bezug auf die Menge der Verhalten. Die Auswahl des nächsten Verhaltens soll als fester Mechanismus ohne

Lernkomponente umgesetzt werden.

2.3 Begriffsklärung

Im bisherigen Verlauf der Arbeit wurde der Begriff Verhalten gleichsam für den beobachtbaren Vorgang, also die erzeugten Aktionen des Systems verwendet und auch für die programmatische Darstellung als Handlungsanweisung bzw. Datenstruktur. Eine Unterscheidung wird auch in der Literatur in der Regel nicht getroffen. Weiterhin wird auch meist nicht zwischen dem gesamten beobachtbaren Systemverhalten und der Ausgabe einzelner Komponenten unterschieden. Da jedoch im weiteren Verlauf dieser Arbeit explizit zwischen diesen Aspekten differenziert werden soll, werden zunächst die folgenden Begriffe definiert.

Definition 2 (Verhaltensmodul) Ein *Verhaltensmodul* M ist eine abgeschlossene Menge von Regeln, Anweisungen oder Randbedingungen, die notwendig sind, um durch deren Einhaltung oder Ausführung ein bestimmtes Ziel zu erreichen.

Definition 3 (Menge aller Verhaltensmodule) Als $\mathcal{M} = \bigcup_{i \in \mathbb{N}_0} M_i$ wird die *Menge aller Verhaltensmodule* bezeichnet, die dem System bekannt sind.

Definition 4 (Verhalten) Als *Verhalten* B wird ein in der Ausführung befindliches Verhaltensmodul bezeichnet.

Definition 5 (Menge aller Verhalten) Als $\mathcal{B} = \bigcup_{i \in \mathbb{N}_0} B_i$ wird die *Menge aller Verhalten* bezeichnet, die sich in der Ausführung befinden.

Definition 6 (Systemverhalten) Das *Systemverhalten* ist die beobachtbare Überlagerung aller Verhalten eines Systems zu einem Zeitpunkt.

Während ein Verhaltensmodul also einen passiven Informationsträger darstellt, entspricht ein Verhalten einer wahrnehmbaren Aktivität, die den Weisungen eines Verhaltensmoduls entspricht. Die einzelnen Verhalten müssen dabei nicht notwendigerweise durch einen externen Beobachter identifizierbar sein. Diese Definitionen sind angelehnt an die psychologische Nomenklatur, bei der Verhalten wahrnehmbare Vorgänge eines Individuums darstellen, die aufgrund verschiedener Strukturen im Gehirn entstehen [Robb10, Fodo83].

2.4 Klassifikation von Verhalten

Für die Formalisierung eines Verhaltensmoduls ist zunächst zu klären, welche Art von Verhalten umgesetzt werden soll. Auf Basis der Definition eines Verhaltens aus Kapitel 2.3, sowie der funktionalen Darstellung $f(input) \rightarrow output$, lassen sich Verhalten nach [Kasp99] wie in Tabelle 2.3 klassifizieren. Als Eingabe kommen sowohl äußere Einflüsse s durch Sensoren in Frage als auch der innere Zustand z des Verhaltens. Die Ausgabe kann aus einer wahrnehmbaren Aktion a und der Änderung des inneren Zustands z bestehen.
Die Verhalten aus Tabelle 2.3 lassen sich nun im Hinblick darauf untersuchen, ob sie für die Verwendung in einem System geeignet sind, das die Anforderungen aus der Aufgabenstellung von Kapitel 2.1 erfüllt. Da dem System alle Verhalten durch die Programmierung eines Laien

Funktion	**Typ**	**Sichtbarkeit**
$f(s) \rightarrow (a)$	reaktive Verhalten	offen
$f(s) \rightarrow (z)$		verdeckt
$f(s) \rightarrow (a, z)$		offen
$f(z) \rightarrow (a)$	blinde Verhalten	offen
$f(z) \rightarrow (z)$		verdeckt
$f(z) \rightarrow (a, z)$		offen
$f(s, z) \rightarrow (a)$	zustandsabhängige Verhalten	offen
$f(s, z) \rightarrow (z)$		verdeckt
$f(s, z) \rightarrow (a, z)$		offen

Tabelle 2.3: Klassifikation der Verhalten nach [Kasp99] basierend auf deren Zustand z, sensorieller Wahrnehmung s und Aktion a.

beigebracht werden sollen, ist eine uniforme Darstellung aller Verhalten erstrebenswert. Der Programmierer müsste sich sonst mit der zusätzlichen Frage nach dem Verhaltenstyp auseinandersetzen und Kenntnisse über das System besitzen.
Zunächst werden alle Verhalten $f(\ldots) \rightarrow (z)$ betrachtet, die eine verdeckte Ausgabe besitzen. Diese sind für die Verwendung im Rahmen dieser Arbeit ungeeignet, da diese nicht in der Lage sind, die geforderten Manipulationen durchzuführen.
Die Gruppe der blinden Verhalten $f(z) \rightarrow (\ldots)$ ist ebenfalls nicht geeignet, da diese die Ausgabe unabhängig von der aktuellen Situation erzeugen, wohingegen das angestrebte System differenziert auf unterschiedliche Situation reagieren soll.
Betrachtet man die Gruppe der rein reaktiven Verhalten $f(s) \rightarrow (\ldots)$, so lassen diese Manipulationen der Umwelt zu und sind prinzipiell auch gut geeignet zur Umsetzung eines lernenden Systems [Arga09]. Allerdings lassen sich damit ausschließlich einfache Zusammenhänge zwischen Sensoreingabe und Aktion modellieren. Die Abbildung komplexer zeitlicher Abfolgen ist damit nicht möglich. Dies muss allerdings, wie in Kapitel 2.2.3 erörtert, durch ein Verhalten möglich sein, da sonst die Skalierung und Flexibilität eingeschränkt ist. Bezüglich der Anforderungen aus Kapitel 2.1 verletzen sie zusätzlich die Forderung einer konsistenten Unterbrechung und Wiederaufnahme, da ihnen ein Zustand fehlt, der wiederaufgenommen werden könnte. Die Menge der rein reaktiven Verhalten ist daher für das angestrebte System ebenfalls nicht geeignet.
Betrachtet man die Gruppe der zustandsbasierten, offenen Verhalten $f(s, z) \rightarrow (a)|(a, z)$ erfüllen sie als einzige alle Anforderungen, die gestellt werden. Die resultierende Aktion a ist sowohl abhängig vom internen Zustand des Verhaltens z, als auch von der Umwelt s. Um die maximale Flexibilität bei der Wahl zu behalten, soll es den Verhalten auch möglich sein, ihren inneren Zustand z zu verändern. Daher sollen die Verhalten die Form $f(s, z) \rightarrow (a, z)$ besitzen. Im folgenden Kapitel soll nun eine geeignete Repräsentation in Form eines Modells für diese Kategorie von Verhalten gefunden werden.

2.5 Verhaltensmodell

In diesem Kapitel sollen die Verhaltensmodule modelliert werden. Dazu werden zunächst einige Standardmodelle in Kapitel 2.5.1 untersucht. Anschließend wird die generelle Umsetzung vorgestellt (Kapitel 2.5.2) und abschließend einige Einschränkungen des Modells vorgenommen

(Kapitel 2.5.3).

2.5.1 Modellvergleiche

Wie bereits erwähnt, gelten die folgenden Anforderungen an das Modell. Es muss einen Zustand halten können und abhängig von diesem unterschiedliche Aktionen erlauben. Weiterhin muss es möglich sein, auf Stimulation durch die Umwelt zu reagieren und auch koordinierte Manipulationen an der Umwelt durchzuführen.
Die Umsetzung einer verhaltensbasierten Architektur kann prinzipiell auf zwei Arten geschehen. Die Umsetzung kann durch ein Modell geschehen, das die Beschreibung von parallelen Abläufen explizit ermöglicht, wie beispielsweise Petri-Netze. Alternativ lässt sich das Problem in zwei Komponenten zerlegen. Man nutzt zunächst ein Modell zur Darstellung eines Verhaltensmoduls. Dieses hält die Informationen und Anweisungen zur Durchführung einer bestimmten Aufgabe. Zusätzlich nutzt man einen Mechanismus zur parallelen Ausführung, der die Kombination bzw. Koordination der Ausgaben der Verhalten untereinander übernimmt.
Eine Übersicht über mögliche Umsetzungen des Systems mit Standardmodellen ist in Tabelle 2.4 zu finden. Die dort gewählten Merkmale zur Bewertung entsprechen den eben erläuterten und wurden auch bereits im Hinblick auf die Forderung gewählt, das neue Verhalten ohne großen Aufwand dem System hinzugefügt werden können. Idealerweise kann dies durch einmaliges Demonstrieren der Aufgabe geschehen.

	Stimulus-Response	Neuronale Netze	Endliche Zustands-automaten	Hidden Markov Modelle	Petrinetze
Zustandsbehaftet	−	o	+	o	+
Programmierbar (One-Shot)	o	−	+	−	o
Trainierbar (Mulit-Shot)	−	+	o	+	o
Modellierung eines Verhaltens	+	o	+	o	−
Gleichzeitige Modellierung und Koordination	−	o	o	o	+

Tabelle 2.4: Vergleich möglicher Modelle zur Umsetzung eines Verhaltensmoduls.

Stimulus-Response bildet die einfachste Verbindung von Sensorik und Aktorik. Durch die fehlende Zustandsmodellierung erhält man immer die gleiche Ausgabe zu einer Eingabe. Durch das Fehlen eines Zustands sind keine längeren und komplexen Aufgaben möglich. Die Verwendung findet vor allem in reaktiven mobilen Systemen statt.
Feedback Neuronale Netze besitzen genau wie Hidden Markov Modelle zwar einen Zustand, stehen aber im Gegensatz zur Forderung der Parametrisierung durch nur ein einzige Demonstration. Dies macht sie zumindest für die Modellierung eines Verhaltensmoduls ungeeignet. Prinzipiell könnte man mit Neuronalen Netzen das gesamte System abbilden [Lewi98, Fier98], allerdings würde die Koordination bei einer hohen Anzahl von Verhalten eine extrem hohe Anzahl an Trainingsbeispielen benötigen. Erschwerend kommt hinzu, dass der interne Zustand bei neuronalen Netzen nur schwierig zu bestimmen ist beziehungsweise bei Hidden Markov Modellen nicht direkt beobachtbar ist.
Endliche Zustandsautomaten bieten sich gut für die Modellierung eines Verhaltens an, da

sie zu jeder Zeit einen diskreten, beobachtbaren internen Zustand besitzen und klare Übergangsregeln, die einfach modelliert werden können. Es ist prinzipiell möglich, die Koordination mehrerer Verhalten damit zu organisieren, wie bei den Discrete Event Systems oder dem Temporal Sequencing, allerdings nur unter der Einschränkung, dass die Menge der Verhalten im Vorfeld bekannt ist und sich nicht ändert.
Die Modellierung mit Petrinetzen scheint zunächst brauchbar, da sie häufig zur Modellierung nebenläufiger Vorgänge verwendet werden. Als einziges der Modelle wäre eine einheitliche Modellierung des gesamten Systems samt aller Verhalten möglich. Allerdings müsste auch hier die Anzahl der Verhaltensmodule von vornherein bekannt sein und dürfte sich nicht ändern, was jedoch der Forderung der Skalierbarkeit und Erweiterbarkeit widerspricht.
Gerade dieser Gedanke der Skalierbarkeit lässt nur die getrennte Modellierung von Verhalten und deren Koordination zu. Zwar wäre auch die integrierte Variante möglich, jedoch ist der Aufwand für das Erweitern um ein Verhalten deutlich höher, da im schlimmsten Fall das gesamte Modell angepasst werden muss, um das zusätzliche Verhalten zu integrieren und die Koordinaten zu aktualisieren. Ist der Action Selection Mechanismus von dem Modell eines Verhaltens getrennt, kann er selbst nach bestimmten Anforderungen modelliert werden und ist dadurch unabhängig von den Verhalten selbst. Es existieren daher keine Einschränkungen im Bezug auf die Anzahl der Verhalten. Dadurch erfüllt sich die Forderung nach einer Skalierbarkeit wesentlich einfacher als durch ein komplettes Modell, das sowohl Verhalten als auch die Koordination abbildet.
Die Entscheidung zur Modellierung eines Verhaltens fällt somit zu Gunsten der endlichen Zustandsautomaten aus, da sie die notwendigen Eigenschaften erfüllen und gleichzeitig ein einfach parametrierbares Modell bieten. Im Folgenden wird zunächst auf die Umsetzung eines Verhaltensmoduls eingegangen und anschließend auf die Modellierung der Koordination.

2.5.2 Umsetzung

Die Umsetzung der Verhaltensmodule erfolgt, wie in Kapitel 2.4 festgelegt, mittels endlicher Zustandsautomaten. Genauer gesagt wird ein Verhaltensmodul $M \in \mathcal{M}$ modelliert als Mealy-Maschine

$$M = [S, C, A, \delta, \omega, s_0, S_F] \tag{2.1}$$

mit einer Menge von Zuständen S, dem Initialzustand $s_0 \in S$, einer Menge von Finalzuständen S_F, einer Menge von Aktionen $A = \{A_0, \ldots, A_n\}$ mit $A_i = \{a_{i0}, \ldots, a_{im}\}$ und Bedingungen $C = \{C_0, \ldots, C_n\}$ mit $C_i = \{c_{i0}, \ldots, c_{ik}\}$ mit $m, n, k \in \mathbb{N}^+,\ c_{ij} \in \mathcal{C}, \mathcal{C} \subseteq \Gamma,\ a_{ij} \in \mathcal{A}$.

Definition 7 (Aktionsmenge $\mathcal{A}$) Als *Aktionsmenge* $\mathcal{A}$ wird die Menge aller vom Robotersystem durchführbaren Aktionen bezeichnet.

Definition 8 (Menge aller Stimuli Γ) Als Γ wird die *Menge aller Stimuli* der Umwelt bezeichnet, die vom Robotersystem wahrgenommen werden können.

Die Übergangsfunktion δ ist definiert durch

$$\delta : S \times C \to S \tag{2.2}$$

und die Ausgabefunktion ω ist definiert durch

$$\omega : S \times C \to A\ . \tag{2.3}$$

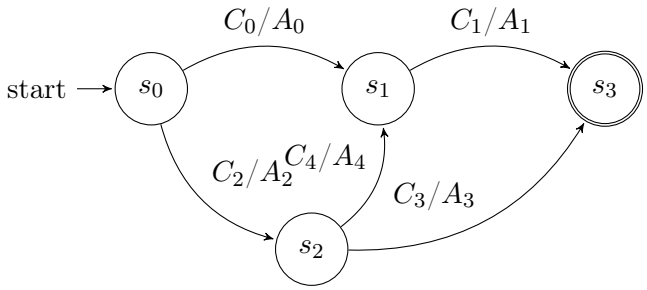


Abbildung 2.3: Automatenmodell mit Bedingungen C und Aktionen A der Transitionen

Damit ein Verhalten vom Zustand s_i in den Zustand s_j wechseln kann, müssen die Bedingungen C der Transition $T(s_i, s_j)$ durch Stimuli $\gamma \in \Gamma$ der Umwelt erfüllt werden. Dies wird durch eine Matching Funktion $match(c, \gamma)$ sichergestellt. Mögliche Umsetzungen für das Matching werden in Kapitel 2.8 vorgestellt.
Bei den Stimuli handelt es sich prinzipiell um alle durch das System wahrnehmbaren Reize und Objekte der Umwelt, den Roboter selbst mit eingeschlossen. Es kann sich also z.B. um eine eingenommene Pose des Roboters handeln, um einen Sprachbefehl oder ein Objekt im Arbeitsraum. Diese Arbeit zielt jedoch nicht auf ein sozial interaktives oder autonomes System ab, sondern auf ein System, das Handhabungsaufgaben in der Industrie und im Haushalt verrichtet. Daher beschränken sich die Stimuli und damit gekoppelt die Bedingungen auf vorhandene Ressourcen $\mathcal{R} \subseteq \Gamma$, die nötig sind um bestimmte Aufgaben zu erfüllen. Die Modellierung der Ressourcen wird in Kapitel 2.6 diskutiert. Die Modellierung der Bedingungen erfolgt in Kapitel 2.7.
Alle Bedingungsmengen C_j der Transitionen $T(s_0, s_i)$, die vom Initialzustand s_0 ausgehen, werden als *Primärbedingung* bezeichnet und die passenden Stimuli $\Gamma_j \subseteq \Gamma$ als *Primärstimuli*. Bei Erfüllung der Matching Funktion wird eine Menge von Aktionen $A_j \subseteq \mathcal{A}$ ausgeführt. Bei den emittierten Aktionen handelt es sich beispielsweise um Trajektorien des Endeffektors mit Zusatzinformationen wie etwa Werkzeugbefehlen und einem Referenzkoordinatensystem. Prinzipiell ist die Menge der Aktionen $\mathcal{A}$ abhängig vom verwendeten Robotersystem und dessen Fähigkeiten. Handelt es sich beispielsweise um ein Multi-Arm System kann A_j auch aus mehreren Trajektorien und zugehörigen werkzeugspezifischen Operationen bestehen.

Billard [Bill08] unterscheidet bei Aktionen solche relativ zur aktuellen Roboterposition, relativ zu einem Objekt und absoluten Bewegungen. Generell lässt sich sagen, dass alle Bewegungen damit Relativbewegungen innerhalb eines bestimmten Referenzsystems darstellen. In dieser Arbeit wird ein Referenzsystem durch Ressourcen definiert (Kapitel 2.6). Die Generierung der Aktionen mit Hilfe dieser Referenzsysteme wird in Kapitel 4 vorgestellt.
Zusammenfassend lässt sich festhalten, dass im Rahmen dieser Arbeit ein Verhaltensmodul durch einen endlichen Zustandsautomaten modelliert wird. Ein Verhalten entspricht der Ausführung eines solchen Verhaltensmoduls und wechselt den Zustand, wenn sich die Ressourcenanforderung für die zu emittierenden Aktionen ändert. Sobald die Bedingungen für den Übergang erfüllt sind, können die Aktionen ausgeführt werden.

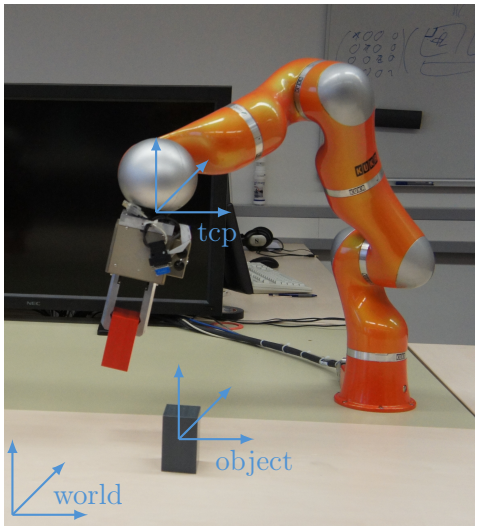


Abbildung 2.4: Beispiele für mögliche Koordinatensysteme in einem Robotersystem

2.5.3 Modelleinschränkungen

Prinzipiell ist es möglich ein Verhaltensmodul als beliebigen nichtdeterministischem endlichen Automaten umzusetzen. Im Hinblick auf eine spätere Programmierung lohnt es sich jedoch gewisse Einschränkungen vorzugeben. Ein allgemeiner endlicher Zustandsautomat lässt sich durch bestimmte Einschränkungen zu einem gerichteten azyklischen Graphen oder einer Sequenz degenerieren. Im Laufe dieses Kapitels soll gezeigt werden, das auch solche Sequenzen ausreichend sind um mit dem hier vorstellten Ansatz Schleifen und Verzweigungen zu realisieren.
Betrachtet man nur den Aspekt der verhaltensbasierten Ausführung von Aufgaben ist theoretisch jeder Zustandsautomat mit dem in dieser Arbeit vorgestellten Ansatz ausführbar. Allerdings soll das Modell im Anschluss noch durch eine intuitive Programmierung parametrisiert werden. Die Programmierung von Zyklen und Verzweigungen werden in einer parallelen Arbeit betrachtet [Soll09, Bart12]. In dieser kann das System zwar nur eine Aufgabe durchführen, dafür sind aber auch Entscheidungen und Schleifen innerhalb des Automaten möglich, die durch mehrere Demonstrationen parametrisiert werden. Der Fokus dieser Arbeit liegt allerdings auf der Durchführung und Programmierung mehrerer unterschiedlicher Aufgaben, zwischen denen auch gewechselt werden soll. Das heißt, es gibt mehrere Automaten mit unterschiedlichem Zweck und jeder dieser Automaten soll mit nur einer Demonstration erzeugt werden. Mit n Demonstrationen ist es möglich im naiven Fall einen Baum mit n Ästen zu erzeugen, indem jeweils ein Ast von der Wurzel für jede Demonstration erzeugt wird. Erzeugt man mit jeder neuen Demonstration einen neuen Ast in allen bereits vorhandenen Ästen ist es bestenfalls möglich einen Baum mit 2^{n-1} Blättern zu erzeugen. Übertragen bedeutet dies, es existieren 2^{n-1} Wege von der Wurzel bis zu einem Blatt und damit 2^{n-1} Möglichkeiten der Ausführung. Mit $n = 1$ ist damit genau ein Ast erzeugbar, was für lineare Aufgaben ausreichend ist. Der Automat degeneriert in diesem Fall zu einer Sequenz.
Durch die Einschränkung der Verhaltensmodule auf eine Sequenz scheint es zunächst nicht mehr möglich zu sein, Alternativen (Verzweigungen) oder Wiederholung von Aufgabenteilen (Schleifen) zu modellieren. Mit gewissen Einschränkungen ist dies dennoch möglich. Denn

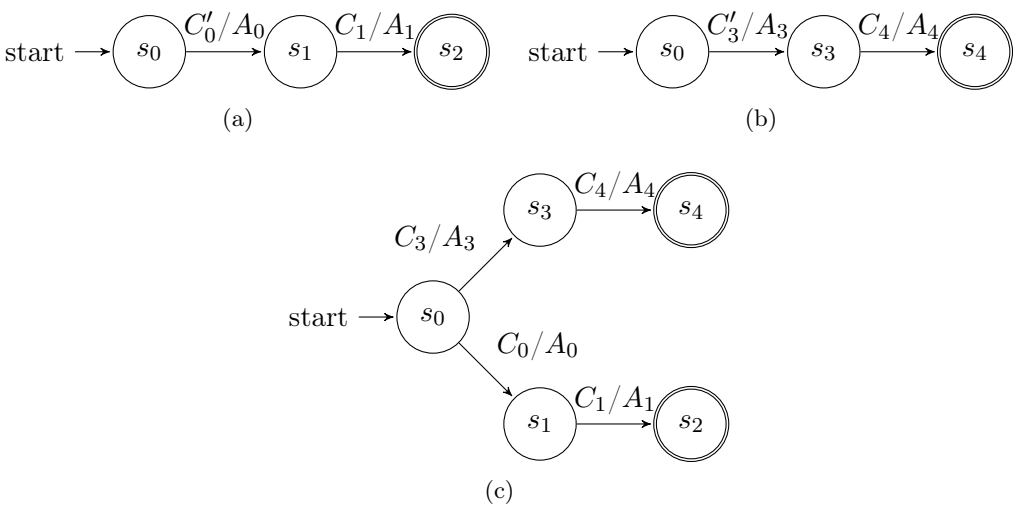


Abbildung 2.5: Modellierung von alternativen Ausführungen durch zwei sequentielle Automaten (a,b) oder durch einen Automaten mit Verzweigung (c)

durch das Vorhandensein mehrerer Sequenzen verlagert sich diese Funktionalität aus dem Automaten in die Wechselwirkung zwischen den Automaten.
Die Entscheidung, welcher Teilbaum innerhalb eines Automaten ausgeführt wird, verlagert sich in die Entscheidung welcher Automat ausgeführt wird. Auch Wiederholungen in Form von Schleifen sind prinzipiell möglich, indem das Verhalten eines Verhaltensmoduls mehrfach ausgeführt wird.

Verzweigungen Die Darstellung eines Automaten mit Verzweigung durch mehrere Sequenzen ist trivial, sofern der Automat eine Baumstruktur besitzt, bei der der Wurzelknoten bereits dem Entscheidungspunkt entspricht (siehe Abbildung 2.5). Hierbei beinhalten die Primärbedingungen C'_0 und C'_3 die Bedingung, dass die jeweils andere Sequenz noch nicht ausgeführt wurde. Die Umsetzung dieses Konzeptes wird in Kapitel 2.7 erläutert.
Besitzt der Automat die Form eines gerichteten azyklischen Graphen ist die Darstellung als Sequenzen nicht eindeutig. Betrachtet man Abbildung 2.6 (a) wird klar, dass bei einer Aufspaltung nicht pauschal entschieden werden kann, was mit den Zuständen $S_\mathrm{H} = \{s_0, s_1, s_2\}$ und $S_\mathrm{T} = \{s_7, s_8\}$ passieren soll. Es gibt die Möglichkeit, die Zustände an beide Sequenzen anzuhängen (c), an nur einen der beiden Automaten anzuhängen (b) oder teilweise als eigene Sequenz abzukoppeln (d).
Es ist nicht bekannt, welche Aktionen (Manipulationen) hier durchgeführt werden. Werden sie nur an eine der Sequenzen angehängt (b), können wichtige Schritte im Voraus und im Nachhinein bei einem der Verhalten fehlen. Da sich nicht entscheiden lässt, welcher Sequenz S_H und S_T zuzuordnen ist, fällt diese Möglichkeit aus.
Werden S_H und S_T an beide Sequenzen angehängt (c) scheint dies zunächst plausibel zu sein. Doch wird Sequenz $K1$ bis zum Zustand s_2 ausgeführt und nimmt man an, dass nun die Bedingungen C_2 nicht erfüllt werden kann, allerdings die Bedingung C_8 in Sequenz $K2$ sehr wohl, blockiert $K1$ an dieser Position und hat möglicherweise schon relevante Änderungen an der Umwelt vorgenommen um auch $K2$ im Zustand s_1 zu blockieren. Somit kann auch diese Umsetzung fehlschlagen.

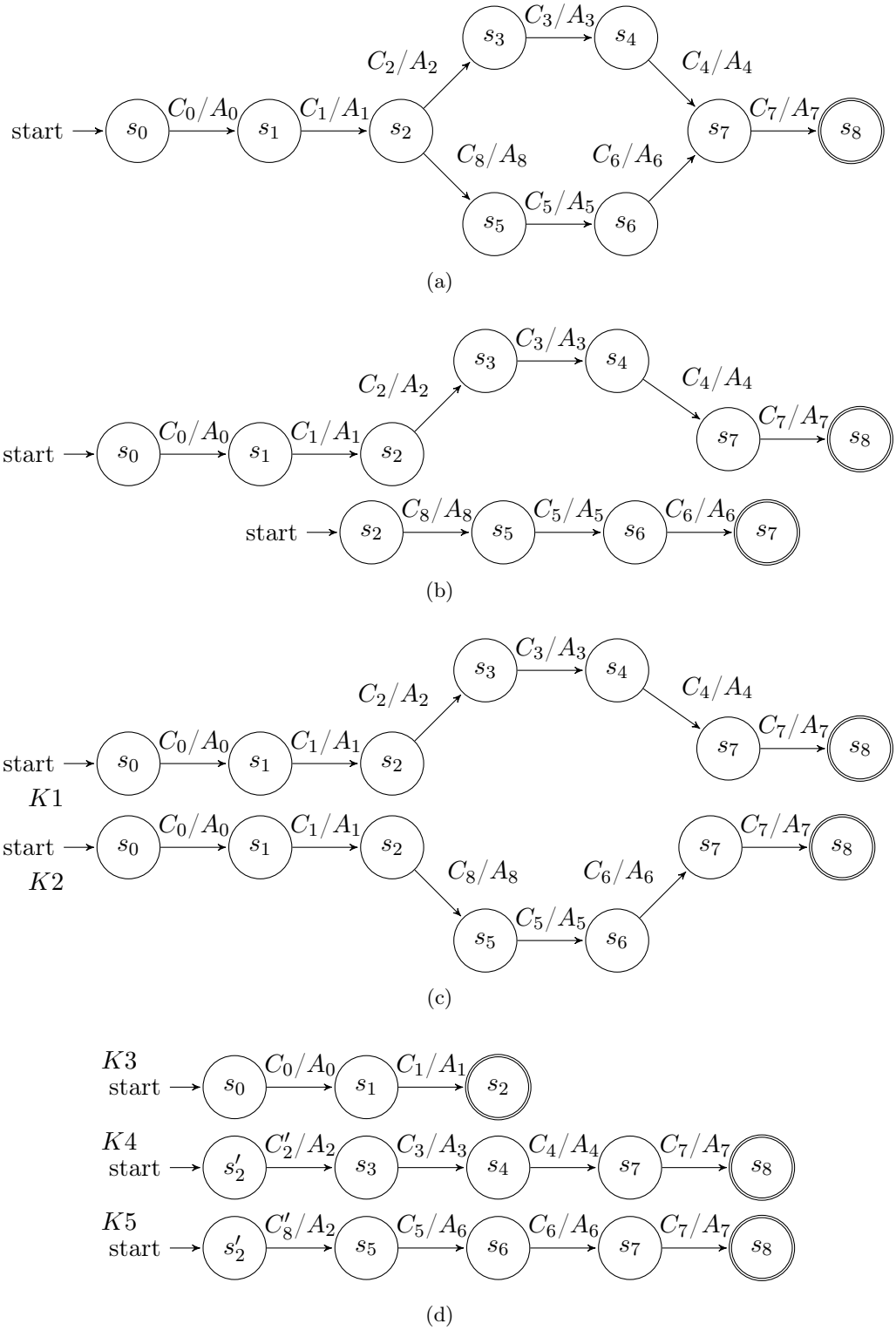


Abbildung 2.6: Prinzipielle Darstellungsmöglichkeiten eines Zustandsautomaten (a) mittels verschiedener Sequenzen (b) - (d)

Die plausible Variante ist die Zerteilung des Baumes nach (d). Die vollständige Ausführung von Verhalten $K3$ muss als Bedingung für beide Teile $K4$ und $K5$ gelten. Dies ist durch C_2' bzw. C_8' gewährleistet. Die Zustände von S_T können in diesem Fall nicht in eine eigene Sequenz abgekoppelt werden, da sie als Startbedingung beinhalten müssten, dass $K4$ oder $K5$ ausgeführt wurde. Eine solche „oder"-Bedingung ist hier jedoch nicht umsetzbar, wie in Kapitel 2.7 zu sehen ist. Diese Zerteilung muss allerdings für einen gerichteten azyklischen Graphen für jede Verzweigung iterativ durchgeführt werden, wodurch die Menge an Sequenzen schnell ansteigen kann. Besitzt der Automat zusätzlich noch Schleifen und Querverweise über die Teilbäume hinweg, ist eine solche Aufteilung in Sequenzen nicht mehr möglich.

Schleifen Auch bei der Umsetzung von Schleifen als Sequenzen gibt es Einschränkungen. So sind zwar Wiederholungen eines Verhaltens möglich, sofern das matching-Verfahren die Bedingungen noch als erfüllt ansieht, allerdings ist es nicht möglich explizite Abbruchbedingungen für die Schleifen anzugeben. Das System reagiert lediglich auf seine Umwelt und wendet die Verhalten so lange an, bis die Bedingungen nicht mehr erfüllt sind. Allerdings müssen diese Verhalten weder am Stück ausgeführt werden noch in einer bestimmten Reihenfolge.
Noch unsicherer wird die Ausführung wenn eine Schleife eine oder mehrere Verzweigungen beinhaltet. Durch Aufspalten der Automaten in Sequenzen, kann die sichere und wiederholte Einhaltung der Reihenfolge, wie sie in einem integrierten Automaten der Fall wäre nicht garantiert werden. Die explizite intuitive Programmierung dieser beiden Konzepte wird in den Arbeiten [Soll09, Bart12] behandelt. Die Einschränkung der Mächtigkeit bezüglich dieser beiden Konzepte scheint zunächst ein Nachteil zu sein. Allerdings bietet es auf der anderen Seite ein sehr einfach parametrierbares Modell einer Zustandssequenz und der Mächtigkeit durch einen äußeren Mechanismus quasi beliebig zwischen Verhalten hin und her zu wechseln. Im Hinblick auf eine spätere intuitive Programmierung und auf mögliche Erweiterungen ist das Modell für alle Forderungen ausreichend.

2.6 Ressourcenmodell

Wie in Kapitel 2.5.2 erwähnt, handelt es sich bei Ressourcen um Elemente des Roboters oder der Umwelt, die notwendig sind, damit die programmierten Aufgaben ausgeführt werden können. Diese Ressourcen $\mathcal{R}$ lassen sich in aktive Ressourcen $\mathcal{R}_\mathrm{a}$, semi-aktive Ressourcen $\mathcal{R}_\mathrm{sa}$ und passive Resourcen $\mathcal{R}_\mathrm{na}$ einteilen. Bei den aktiven Ressourcen handelt es sich um direkt steuerbare Komponenten, die eine Aktion $a \in \mathcal{A}$ erzeugen können, und so direkt ihre Umwelt oder den Systemzustand verändern können. Damit definieren sie die möglichen Aktionen $\mathcal{A}$ eines Systems. Typische Beispiele sind der Roboter und seine montierten Werkzeuge. Semi-aktive und passive Ressourcen stellen die Umwelt der aktiven Komponenten dar. Bei semi-aktiven Ressourcen handelt es sich um solche, deren Zustand nicht direkt steuerbar ist, mit denen aber prinzipiell auch die Umwelt verändert werden kann, indem sie als Werkzeug verwendet werden oder mit anderen Ressourcen kombiniert werden. Typischerweise handelt es sich dabei um Objekte im Arbeitsraum. Bei passiven Ressourcen handelt es sich um Komponenten, die lediglich beeinflusst werden können, aber selbst keinen direkten oder indirekten Einfluss nehmen können. Typischerweise handelt es sich hier um den Raum selbst.
Eine Ressource r ist definiert als

$$r = \{T_r, f_r\} \quad \text{mit} \quad T_r \in \big\{\text{aktiv},\ \text{semiaktiv},\ \text{passiv}\big\}$$

und einer zusätzlichen Merkmalsmenge f_r, die Informationen über die Ressource speichert. Diese Menge f_r ist abhängig von T_r und erlaubt die Anwendung unterschiedlicher Matching-Verfahren (Kapitel 2.8).

2.6.1 Aktive Ressourcen

Bei aktiven Ressourcen $\mathcal{R}_\mathrm{a}$ handelt es sich vornehmlich um Komponenten des ausführenden Robotersystems samt aller Werkzeuge und Apparaturen. Die Menge dieser Ressourcen ist üblicherweise von vornherein bekannt und ändert sich im Betriebsverlauf nicht. In der Regel existiert auch ein vollständiges Modell dieser Ressourcen. Die kleinste Darstellungsmöglichkeit einer Ressource ist eine funktional abgeschlossene Einheit, die eine sinnvolle Aktion a ermöglicht.
Es ließe sich beispielsweise auch jedes Segment eines Roboterarms als Ressource darstellen. Allerdings ist es wenig sinnvoll, wenn man für eine Aktion wie dem Bewegen des Roboters stets jede dieser Ressourcen benötigt. Statt der Flexibilität würde dies nur die Komplexität und die Fehleranfälligkeit des Systems erhöhen. Sinnvoller gewählt sind also z.B. der Roboterarm, der Greifer, ein Werkzeug oder ein Zubringer.
Jede Ressource $r \in \mathcal{R}_\mathrm{a}$ wird durch die Menge f_r weiter definiert. In dieser befinden sich Elemente, welche die Ressource mit absteigender Eindeutigkeit beschreiben:

$$f_r = \{ID, Typ, Klasse, LLMerkmale\} \tag{2.4}$$

So stellt ID einen eindeutigen Bezeichner innerhalb des Systems dar. Den Typ teilen sich alle baugleichen Elemente, wie z.B. „Schunk PG70“. Mit $Klasse$ erhält man eine Beschreibung der Hauptfunktionalität, wie z.B. „Greifer“. Die $LLMerkmale$ beschreiben Low-Level-Merkmale der Ressource, wie z.B. geometrische Ausprägungen. Während die ersten drei Merkmale symbolischer Natur sind, können die $LLMerkmale$ auch subsymbolischer Natur sein. Diese Ausprägungen stellen damit die Abstraktionsebene dar, auf der eine Ressource beschrieben wird. Jede dieser Ausprägungen kann theoretisch unbestimmt sein. Jedoch muss insgesamt mindestens eine dieser Ausprägungen vorhanden sein, um ein Matching zu erlauben. Bei den aktiven Ressourcen wird davon ausgegangen, dass alle symbolischen Elemente bekannt sind und der Zustand, wie etwa die Konfiguration des Roboters, jederzeit bestimmbar ist.

2.6.2 Semi-aktive Ressourcen

Die Gruppe der semi-aktiven Ressourcen $\mathcal{R}_\mathrm{sa}$ bilden die Objekte im Arbeitsraum des Robotersystems. Während bei den aktiven Ressourcen der Zustand zu jeder Zeit bekannt ist, kann bei Objekten nicht automatisch davon ausgegangen werden. Es ist zwar denkbar, dass vor allem in produzierenden Unternehmen digitale geometrische Modelle vorhanden sind, in der Regel müssen die Objekte aber sensorisch erfasst, analysiert und klassifiziert werden.
Die Darstellung entspricht auch hier der von aktiven Ressourcen. Jedes Objekt $r \in \mathcal{R}_\mathrm{sa}$ wird durch zusätzliche Merkmale

$$f_r = \{ID, Typ, Klasse, LLMerkmale\} \tag{2.5}$$

beschrieben. Auch hier folgt die Reihenfolge einer absteigenden Eindeutigkeit mit dem eindeutigen Bezeichner ID, einem gemeinsamen Typ aller nicht unterscheidbaren Objekte (z.B. „ACME Löffel 0815“) und einer Funktionellen Klasse $Klasse$ (z.B. "Kochlöffel"). Bei $LLMerkmale$

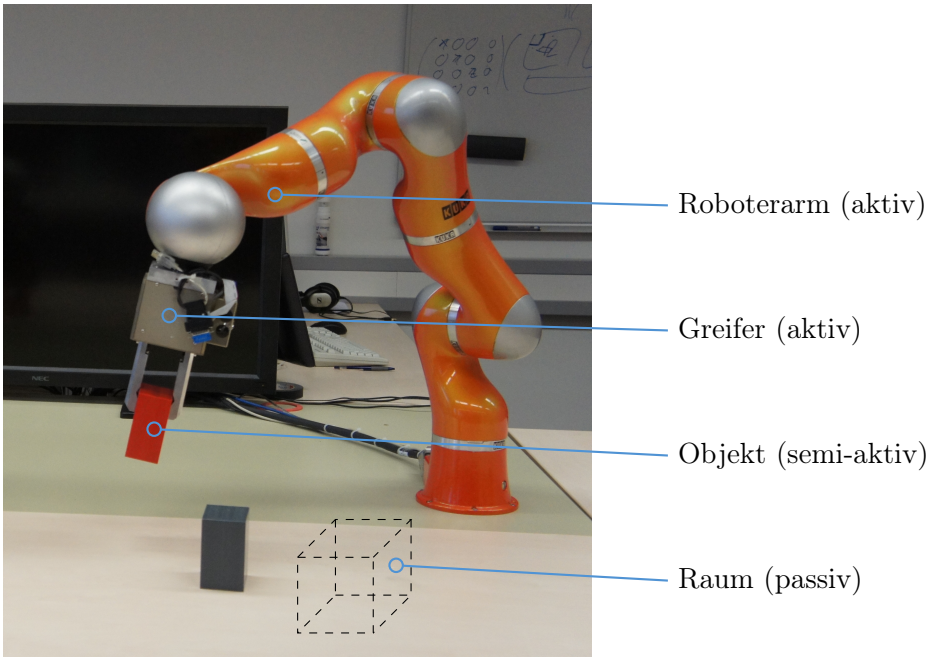


Abbildung 2.7: Mögliche Ressourcen an einem Roboter Arbeitsplatz

handelt es sich beispielsweise um geometrisch oder Farb-basiert Merkmale [Khot90, Bay06, Rusu10]. Welche Elemente gefüllt werden, hängt ab von der Art der Objektdetektion. Je nachdem auf welcher Abstraktionsebene die Erkennung und Repräsentation der Umwelt stattfindet, können mehr oder weniger Merkmale gegeben werden.
In der einfachsten Variante enthält f_r nur eine Identifikationsnummer. Alternativ kann ein Typ durch eine Objekterkennung [Ehre00, Rusu10] erkannt werden, die beispielsweise auf $LLMerkmale$ arbeitet. Aus dem Typ kann je nach Mächtigkeit des Systems auch die $Klasse$ geschlossen werden. Im Gegensatz zu den aktiven Ressourcen ist deren Anzahl im Vorhinein nicht bekannt, da sie sich während der Ausführung ändern kann.

2.6.3 Passive Ressourcen

Die Menge der passiven Ressourcen $\mathcal{R}_{\text{na}}$ entspricht dem Arbeitsraum. Eine Möglichkeit Raum diskret und identifizierbar darzustellen bieten Voxel. Man betrachtet den Arbeitsraum dazu als $n \cdot m \cdot l$ diskrete Voxel mit $n, m, l \in \mathbb{N}^+$. Jedes dieser Voxel entspricht einer Ressource. Da es sich um passive Ressourcen handelt, deren Anzahl und Position bekannt und konstant sind, besitzen sie lediglich das Merkmalstupel $f_r = (i, j, k)$ mit den Koordinaten $i, j, k \in \mathbb{N}^+$ welche die Ressource eindeutig identifizieren.
Jede semi-aktive und aktive Ressource im Arbeitsraum belegt diese passiven Ressourcen indirekt durch seine Anwesenheit und baut damit eine Abhängigkeit auf (siehe Kapitel 3.5.2). Die Bestimmung, welche passiven Ressourcen benötigt werden, kann auf zwei Arten geschehen. Dies ist abhängig davon, ob die semi-aktiven und aktiven Ressourcen bekannt sind oder nicht. Sind die Ressourcen bekannt, wird durch eine Abbildungsfunktion

$$occupy(r_a) \rightarrow R_{na}, \quad r_a \in \left(\mathcal{R}_{\text{a}} \cup \mathcal{R}_{\text{sa}}\right), R_{na} \subseteq \mathcal{R}_{\text{na}} \tag{2.6}$$

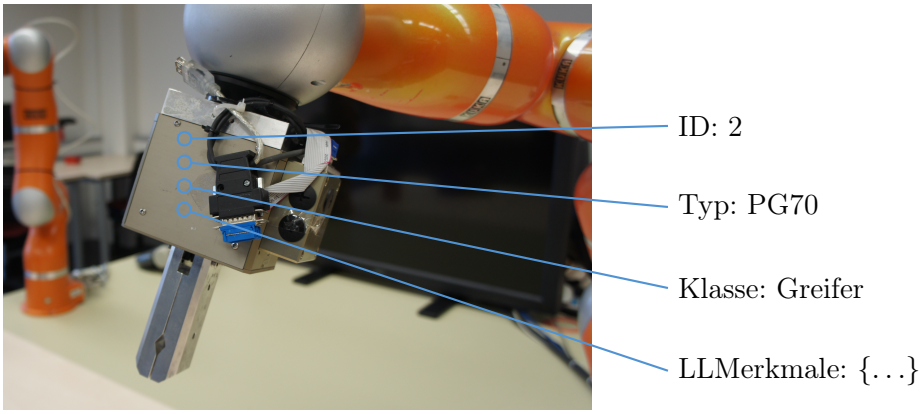


Abbildung 2.8: Beispiel zur Repräsentation einer Ressource auf verschiedenen Abstraktionsebenen.

bestimmt, welche passiven Ressourcen R_{na} für eine aktuelle oder zukünftige Aktion benötigt werden.
Existieren keine Modelle der Ressourcen, sind diese also nicht bekannt, muss hierfür beispielsweise ein Rekonstruktionsalgorithmus verwendet werden [Kuhn12, Wern14], der die belegten Voxel dem Objekt zuordnet. Dies wird in dieser Arbeit jedoch nicht näher betrachtet.

2.6.4 Objektgedächtnis

Um eine zeitliche Koordination von Verhalten zu ermöglichen, muss das System ein Gedächtnis besitzen. Das System muss wissen, welche Verhalten bereits auf eine Ressource angewendet wurden und wie oft dies geschehen ist. Dies verhindert, dass ein Verhalten fälschlicherweise wiederholt auf eine Ressource angewendet wird oder Verhalten in unerwünschter Reihenfolge ausgeführt werden.
In Kapitel 2.5.3 wurde bereits angesprochen, dass eine Bedingung ebenfalls fordern kann, dass ein Verhalten bereits auf ein Objekt ausgeführt wurde. Dies wird in Kapitel 2.7 näher erläutert. Da dieses Gedächtnis an bereits ausgeführten Verhalten als Merkmal einer Ressource gehandhabt werden soll, wird hier das Merkmalstupel der semi-aktiven Ressourcen f_r um ein Gedächtnis mem zu $f_r \oplus mem = \{f_r, mem\}$ erweitert. Hierbei speichert mem für jedes Verhaltensmodul $M_i \in \mathcal{M}$ die Häufigkeit der Ausführung e_i mit dieser Ressource als

$$mem = \big\{\{M_1, e_1\}, \ldots, \{M_n, e_n\}\big\}, \quad e_i \in \mathbb{N}_0, \quad n = |\mathcal{M}|. \tag{2.7}$$

Der binäre Zustand (belegt/frei) der passiven Ressourcen ist in der Regel problemlos und vollständig wiederherstellbar und ändert sich auch nicht durch häufige Verwendung. Daher wird für sie keine Historie gespeichert.
Auch bei den aktiven Ressourcen werden die Merkmale nicht um ein Gedächtnis erweitert, da sie sich als aktiver Teil des Systems nicht irreversibel ändern sollen. Es gibt zwar unerwünschte Änderungen, wie beispielsweise Verschleiß am Werkzeug, allerdings wird dies in dieser Arbeit nicht berücksichtigt, da es ebenfalls in den Bereich Robustheit fällt. Dieser wird im Ausblick kurz angesprochen.

2.7 Bedingungen

Die Bedingungen $\mathcal{C}$ sind eng an das Ressourcenmodell gekoppelt und orientieren sich an der Einteilung der Ressourcen in aktive, semi-aktive und passive Bedingungen. Die Darstellung einer Bedingung $c \in \mathcal{C}$ erfolgt durch $c = \{T_c, f_c\}$. Die Variable T_c besitzt die gleichen Ausprägungen wie eine Ressource, also $T_c \in \{\text{aktiv, semiaktiv, passiv}\}$. Die Merkmalsmenge f_c besitzt die gleichen Ausprägungen wie f_r und wird durch *hist* ergänzt, welches das Analogon zu *mem* darstellt.

$\mathbf{T_c}$	$\mathbf{f_c}$
aktiv	$\{id, Typ, Klasse, LLMerkmale\}$
semiaktiv	$\{id, Typ, Klasse, LLMerkmale\} \oplus hist$
passiv	$\{i, j, k\}$

Beim Typ passiv müssen, wie bei den Ressourcen, alle Elemente von f_c gefüllt sein, da nur so eine eindeutige Zuordnung möglich ist. Bei den aktiven und semi-aktiven Ressourcen darf dagegen nur eines der Elemente ID, Typ, $Klasse$ und $LLMerkmale$ von f_c gefüllt sein.
Objekte besitzen ein Gedächtnis, das speichert, welche Verhalten bereits auf das Objekt angewendet wurden. Für eine koordinierte Ausführung mehrerer Verhalten kann es nützlich sein, als Erweiterung der Bedingung zu fordern, dass andere Verhalten bereits mit diesem Objekt durchgeführt wurden. Z.B. macht es in einer Küchenanwendung durchaus Sinn den Topf erst nach dem Einfüllen aller Zutaten zu rühren und nicht davor. Zu diesem Zweck wird in *hist* zu jedem im System vorhandenen Verhaltensmodul $M \in \mathcal{M}$ die minimal und maximal geforderte Anzahl $e_{\min}$ und $e_{\max}$ an Ausführungen mit dieser Ressource gespeichert als $(M, e_{\min}, e_{\max})$. Somit ergibt sich *hist* zu

$$hist = \big\{\{M_1, e_{1,min}, e_{1,max}\}, \ldots, \{M_n, e_{n,min}, e_{n,max}\}\big\}, \quad e_{i,\min}, e_{i,\max} \in \mathbb{N}_0, \quad n = |\mathcal{M}| \tag{2.8}$$

Damit werden implizit die Aktionen definiert, die bereits mit dieser Ressource durchgeführt worden sein sollen. Durch Setzen von $e_{\max} = 0$ für ein bestimmtes Verhaltensmodul kann gefordert werden, dass das zugehörige Verhalten noch nicht auf die Ressource angewendet wurde.

2.8 Matching Verfahren

Eine Matching Funktion *match* gibt an, ob eine bestimmte Ressource $r \in \mathcal{R}$ einer bestimmten Bedingung $c \in \mathcal{C}$ genügt. Ein Matching kann allerdings nur zwischen gleichen Typen von Ressourcen und Bedingungen angewendet werden ($f_r = f_c$). Sowohl Ressourcen als auch Bedingungen können auf unterschiedlicher Abstraktionsebene dargestellt werden, die mehr oder weniger eindeutig sind. Betrachtet man das Matching zunächst ohne Objektgedächtnis, ist ein Vergleich auf gleicher Abstraktionsebene trivial, sofern diese auf symbolischer Ebene vorliegen.

$$match_{\text{symb}}(c, r) = \begin{cases} \text{wahr}, & \text{falls } \forall f_{c,i} \in f_c \text{ mit } f_{c,i} \neq \emptyset,\ f_{c,i} = f_{r,i} \\ & \text{und } \ T_c = T_r \\ \text{falsch}, & \text{sonst.} \end{cases} \tag{2.9}$$

Allerdings ist auch denkbar, dass beide auf unterschiedlicher Abstraktionsebene vorliegen. In diesem Fall gibt die Bedingung die Abstraktionsebene vor (siehe Tabelle 2.5). Fordert eine

f_c	f_r
ID	ID
Typ	ID, Typ
Klasse	ID, Typ, Klasse

Tabelle 2.5: Mögliche Ressourcendarstellungen f_r bei Vorgabe der Bedingung f_c. Aus ID lässt sich Typ ableiten und aus Typ lässt sich $Klasse$ ableiten.

Bedingung zur Ausführung einer Aktion z.B. „Kuka LBR“ so kann nicht davon ausgegangen werden, dass sie sich mit einem anderen Roboter erfüllen lässt. Wird allerdings nur „Roboterarm“ gefordert, könnte ein beliebiger anderer Roboterarm die Bedingung erfüllen.
Im Rahmen dieser Arbeit wird davon ausgegangen, dass sich die symbolischen Merkmale der niedrigeren Abstraktionsebene einer Ressource aus den höheren Abstraktionsebenen ableiten lassen. D.h. aus ID lässt sich beispielsweise Typ und $Klasse$ ableiten. Anschließend wird ein Matching auf der Ebene von f_c durchgeführt. Die Ableitung kann durch eine Ontologie realisiert werden [Bill10], durch ein analysierendes System oder vom Einrichter des Systems hinterlegt werden.
Liegen die Informationen f_c und f_r jeweils als subsymbolische Eigenschaften $LLMerkmale$ vor, muss zunächst deren Ähnlichkeit berechnet werden. Dazu wird der Abstand beider Vektoren durch eine Funktion d auf eine normalisierte Ausgabe zwischen 0 und 1 abgebildet.

$$d(f_c, f_r) \rightarrow [0, 1] \tag{2.10}$$

Anschließend kann geprüft werden, ob die Ähnlichkeit ausreichend ist, um die Bedingung zu erfüllen. Dafür wird ein Schwellwert θ festgelegt, der prüft ob die Distanz der beiden Merkmalsvektoren klein genug ist:

$$match_{\text{subsymb}}(c, r) = \begin{cases} \text{wahr}, & \text{falls } d(f_c, f_r) \leq \theta \\ \text{falsch}, & \text{sonst.} \end{cases} \tag{2.11}$$

Konkret würde dies beispielsweise bedeuten, dass die Abweichung eines Objektes in der Farbe gering genug ist gegenüber der gewünschten Farbe. Die Berechnung des Abstandes der beiden Merkmalsmengen f_c und f_r hängt von der verwendeten Metrik ab. Möglich sind hier Ausprägungen der in der Klassifikation häufig genutzten Minkowski Metrik auf einem n-dimensionalen Merkmalsraum

$$d_k(f_c, f_r) = \left(\sum_{i=0}^{n} |f_{c,i} - f_{r,i}|^k \right)^{\frac{1}{k}} \tag{2.12}$$

wie die euklidische Distanz für $k = 2$ oder die Chebyshev Distanz für $p = \infty$ als

$$d_{\text{Chebyshev}}(f_c, f_r) := \max_i(|f_{c,i} - f_{r,i}|), \tag{2.13}$$

die starke Abweichungen in einer Dimension besonders berücksichtigt. Natürlich sind auch weitere individuell definierte Abstandsmaße denkbar.
Eine gemischte Darstellung von symbolischen Bedingungen und subsymbolischen Ressourcen wird hier ausgeschlossen. Der Übergang von subsymbolischer zur symbolischer Darstellung ist ein eigenes Problemgebiet und benötigen oft zusätzliches Domänenwissen [Gat98, Zoll04,

Rick09, Span14, Pais13].
Das Matching der Historie *hist* mit dem Objektgedächtnis erfolgt komponentenweise zu:

$$match_{\text{hist}}(mem, hist) = \begin{cases} \text{wahr}, & \text{falls } \forall\{M_i, e_i\} \in mem\text{: } \exists\{M_j, e_{j,min}, e_{j,max}\} \in hist \\ & \text{mit } M_i = M_j \text{ und } e_{j,min} \leq e_i \leq e_{j,max} \\ \text{falsch}, & \text{sonst.} \end{cases} \tag{2.14}$$

Es wird für jede *semiaktive* Ressource geprüft, ob die bisherige Verwendung durch bestimmte Verhalten innerhalb der erlaubten Toleranz liegt. Sollte ein Matching aufgrund der Merkmale positiv ausfallen, aber aufgrund der Historie negativ, so erfüllt die Ressource die Bedingung letztendlich nicht.
Die Kategorie *passiv* nimmt aufgrund der Darstellung des Raums eine Sonderrolle ein. Denn Raum wird in dieser Arbeit nicht explizit bedingt. Die Forderung nach Raum entsteht vielmehr durch aktive und semi-aktive Ressourcen und ändert sich stark mit dem Szenenaufbau und der durchgeführten Aufgabe. Der benötigte Raum wird u.a. durch Abbildung der nächsten Roboterposition auf die Raumelemente erreicht. Raum kann eine wichtige Rolle spielen, falls es mehr als nur einen ausführenden Roboterarm gibt, um Kollisionen zu verhindern. Auch die Kollision mit der Umgebung lassen sich vermeiden, indem man die belegten Raumelemente schlicht nicht als Ressourcen zur Verfügung stellt.
Generell ist ein Matching-Verfahren $match(c, r)$ eindeutig, wenn die Menge der passenden Ressourcen $R_M \subseteq \mathcal{R}$ mit

$$R_M = \{r \in \mathcal{R} \mid match(c, r) = wahr\} \tag{2.15}$$

für eine Bedingung c eine Mächtigkeit von $|R_M| = 1$ besitzt. Gilt $|R_M| > 1$ gibt es mehrere Möglichkeiten die Bedingung einer Transition zu erfüllen. Eine mögliche Lösung wäre die Entscheidung für die erste passende Ressource $r \in R_M$.

2.9 Zusammenfassung

In diesem Kapitel wurde die Modellierung von Verhalten diskutiert. Dazu wurde zunächst ein Überblick über den Stand der Forschung gegeben und mögliche Verhalten anhand ihrer Mächtigkeit klassifiziert. Anschließend wurde die Modellierung eines Verhaltensmoduls und die Mechanismen zur Ausführung eines Verhaltens präsentiert. Daraufhin wurde die Modellierung der Ressourcen und deren Abgleich (Matching) mit den Bedingungen der Verhalten vorgestellt.

3 Ausführung von Verhalten

Die in Kapitel 2 vorgestellten Modelle und Methoden sind notwendig, um ein einzelnes Verhalten isoliert auszuführen. Bei Erfüllung der Bedingungen $C \subseteq \mathcal{C}$ durch die passende Ressourcen $R \subseteq \mathcal{R}$ werden die Aktionen $A \subseteq \mathcal{A}$ eines Zustandsübergangs T ausgeführt. In diesem Kapitel soll die (quasi-)parallele Ausführung mehrerer Verhalten betrachtet werden und der dazu notwendige Mechanismus vorgestellt werden.

3.1 Prozessbasierte Ausführung

Wie in Kapitel 2.1 festgelegt, soll das System in der Lage sein, mehrere zweckgerichtete Verhalten auszuführen, die jeweils eigene Ziele verfolgen. Betrachtet man diese Vorgabe aus Sicht der Multiagentensysteme (Tabelle 3.1) würde das System als konkurrierend eingestuft werden (*individuelle Konkurrenz*), da viele Agenten (Verhalten) um die gleichen, begrenzten Ressourcen konkurrieren, jedoch nicht notwendigerweise ein einheitliches oder gemeinsames Ziel verfolgen. Betrachtet man die Ausgangssituation genauer, so existiert eine sich dynamisch än-

Ziele der Agenten verträglich	Zur Verfügung stehende Ressourcen ausreichend	Fähigkeit der Agenten zur individuellen Zielerreichung	Art der Simulation	Klasse der Situation
+	+	+	Unabhängigkeit	Indifferenz
+	+	−	Einfache Kollaboration	Kooperation
+	−	+	Behinderung	
+	−	−	Koordinierte Kollaboration	
−	+	+	Rein indiv. Konkurrenz	Antagonismus
−	+	−	Rein kollektive Konkurrenz	
−	−	+	Individueller Ressourcenkonflikt	
−	−	−	Kollektiver Ressourcenkonflikt	

Tabelle 3.1: Mögliche Einteilung von Agenten nach [Ferb99]. Hierbei entspricht + einem „ja“ und − einem „nein“.

dernde Anzahl von Verhaltensmodulen $\mathcal{M}$, die parallel und evtl. auch mehrfach ausgeführt werden sollen. Zu deren Bearbeitung stehen aktive und semi-aktive Ressourcen $R \subseteq \mathcal{R}$ zur

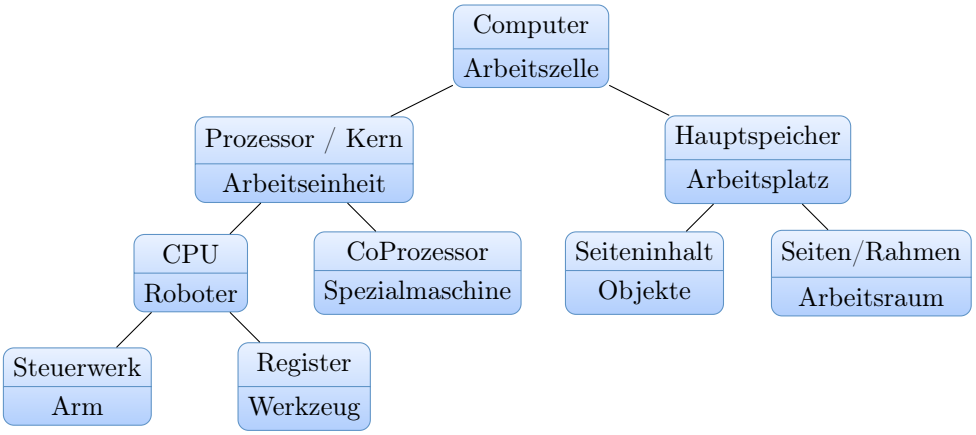


Abbildung 3.1: Analogien eines Computers (oberer Zellenteil) aus Betriebssystemsicht und eines Robotersystems (unterer Zellenteil). Dabei entsprechen die Blätter des Baumes den in dieser Arbeit verwendeten Ressourcen.

Verfügung, die von den Verhaltensmodulen an unterschiedlichen Stellen benötigt werden.
Gewünscht ist nun ein Mechanismus, der die aktive Ausführung und Koordination dieser Verhaltensmodule erfüllt und für eine faire Verteilung der Ressourcen $\mathcal{R}$ sorgt, so dass mehr als ein Verhalten ausgeführt werden kann. Dies ist eine Minimalanforderung an einen solchen Mechanismus. Weitere Forderungen können beispielsweise die Maximierung der Anzahl ausgeführter Verhalten innerhalb einer bestimmten Zeit sein oder die Minimierung der Reaktionszeit der Verhalten.
Eine solche Problemstellung mit ebenso diesen Forderungen findet man im Bereich moderner Betriebssysteme [Tane07]. Zur Übertragung auf Robotersysteme, betrachtet man jedes Verhaltensmodul $M \in \mathcal{M}$ als Programm. Die Ausführung eines Verhaltensmoduls M als Verhalten $B \in \mathcal{B}$ entspricht damit einem Prozess. Prozesse fordern während ihrer Ausführung verschiedene Ressourcen an. Diese können aktiv, semi-aktiv oder passiv sein. Abbildung 3.1 zeigt die Analogien zwischen Betriebssystemen und dem angestrebten verhaltensbasierten Robotersystem. Hierbei stellen die Blätter des Baumes die in dieser Arbeit verwendeten Ressourcen dar.
Die Zuteilung der Ressourcen erfolgt dabei durch ein Scheduling-Verfahren, das verschiedene Ziele umsetzt, wie beispielsweise den maximalen Durchsatz von Verhalten. Beide Domänen besitzen starke Ähnlichkeit in ihrem Aufbau, ihrem Zweck und der Vorgehensweise zur Erreichung von Zielen. Es erlauben auch beide eine Skalierung um weitere aktive Elemente (Multi-Core bzw. Mehrrobotersystem).
Es werden daher im Folgenden die Methoden zur Synchronisation und Koordination von Prozessen in einem Betriebssystem adaptiert, so dass sie zur Koordination von Verhalten auf einem Robotersystem verwendet werden können.

3.2 Ressourcen-basierte Synchronisation von Verhalten

Ressourcen werden von jedem Verhalten während seiner Ausführung benötigt. Da im System üblicherweise weniger Ressourcen zur Verfügung stehen, als benötigt werden, müssen diese effizient verteilt werden. In Anlehnung an ein Betriebssystem soll im Folgenden der Zustand

eines Verhaltens festgelegt werden, allerding angepasst an das Ressourcenmodell für Roboter. Zunächst müssen die Ressourcen dafür entsprechend kategorisiert werden.
Die Ressourcen $\mathcal{R}$ beinhalten alle Ressourcen der Umwelt und des Roboters. Diese lassen sich in zwei Untermengen $\mathcal{R}_{\text{np}}$ und $\mathcal{R}_{\text{p}}$ unterteilen, mit

$$\mathcal{R} = \mathcal{R}_{\text{a}} \cup \mathcal{R}_{\text{sa}} \cup \mathcal{R}_{\text{na}} = \mathcal{R}_{\text{np}} \cup \mathcal{R}_{\text{p}}. \tag{3.1}$$

Dabei stellt $\mathcal{R}_{\text{p}}$ die Menge der präemptiven Ressourcen und $\mathcal{R}_{\text{np}}$ die Menge der nicht-präemptiven Ressourcen dar.

Definition 9 (Preemptiv) Eine Ressource $r \in \mathcal{R}$ ist *preemptiv*, wenn sie einem Verhalten B_i entzogen und einem Verhalten $B_j \neq B_i$ zugewiesen werden kann.

Jede Ressource $r \notin \mathcal{R}_{\text{p}}$ ist nicht-preemptiv. Nicht-präemptive Ressourcen können nicht vorzeitig entzogen werden und erhalten eine Sperre, solange sie von einem Verhalten benötigt werden. Präemptive Ressourcen sind üblicherweise alle aktiven Komponenten, wie z.B. Roboterarm oder Greifer und alle nicht-manipulierten Objekte, da ihr Zustand unverändert oder leicht wiederherstellbar ist. Die Ressourcen $\mathcal{R}_{\text{np}}$ sind z.B. manipulierte Objekte im Arbeitsraum, deren früherer Zustand nicht direkt wiederherstellbar ist. Ansätze zur Klassifikation finden sich in Kapitel 3.6.

Definition 10 (Gehaltene Ressourcen H_i) Als $H_i \subseteq \mathcal{R}$ wird die Menge, der von Verhalten B_i *gehaltenen Ressourcen* bezeichnet. Diese wurden von B_i zur Erfüllung von Bedingungen akquiriert.

Definition 11 (Freie Ressourcen V) Die Menge der Ressourcen, die von keinem Verhalten gehalten wird, sei die Menge der *freien Ressourcen* $V = \{r \in \mathcal{R} | r \notin \bigcup H_i\}$

Im folgenden sei T_i die aktuelle Transition des Verhaltens B_i und $C_i = \{c_{i,1}, \ldots, c_{i,m}\}$, $m \in \mathbb{N}^+$ die Menge der Bedingungen dieser Transition T_i. Es wird nun geprüft, unter welchen Umständen die Möglichkeit zur Erfüllung der Bedingungen C_i der aktuellen Transition gegeben ist, so dass die zugehörigen Aktionen A_i ausgeführt werden können.
Es wird dazu für jede dieser Bedingungen $c_{i,k} \in C_i$ eine Menge Ressourcen $\hat{H}_{i,k} \subseteq H_i$ definiert, welche die Bedingung $c_{i,k}$ erfüllen und die bereits von B_i gehalten werden.

$$\hat{H}_{i,k} = \{r \in H_i | match(c_{i,k}, r) = true\} \quad mit\ c_{i,k} \in C_i. \tag{3.2}$$

Weiterhin lässt sich analog die Menge $\hat{V}_k \subseteq V$ aller freien Ressourcen definieren, die die Bedingung $c_{i,k}$ erfüllen:

$$\hat{V}_k = \{r \in V | match(c_{i,k}, r) = true\} \quad mit\ c_{i,k} \in C_i. \tag{3.3}$$

Mit Hilfe dieser Definitionen lässt sich nun der Zustand eines Verhaltens B_i bestimmen. Ein Verhalten kann sofort in den Zustand `aktiv` wechseln, sofern alle Bedingungen C_i der aktuellen Transition T_i des Verhaltens B_i durch bereits gehaltene Ressourcen H_i oder freie Ressourcen V erfüllt werden können. Weiterhin kann ein Verhalten in den Zustand `bereit` wechseln, sofern alle Bedingungen C_i der aktuellen Transition nicht ausschließlich durch gehaltene oder freie Ressourcen, aber durch Entzug zusätzlicher präemptiver Ressourcen $R \subseteq \mathcal{R}_{\text{p}}$ eines weiteren Verhaltens ermöglicht werden. Ein Verhalten B_i gilt als `terminiert`, falls der aktuelle Zustand s_c einem der Finalzustände entspricht $s_c \in F$.

Ein Verhalten gilt als `blockiert`, falls er nicht `terminiert` ist und weder die Bedingung für `aktiv` noch für `bereit` erfüllt sind. Das bedeutet, es konnte zur Ausführung der nächsten Aktion mindestens eine der benötigten Ressourcen nicht akquiriert werden.
Während der Zustand `terminiert` eindeutig identifizierbar ist, können die Zustände `aktiv` und `bereit` abhängig von der Abstraktionsebene der Bedingungen und Ressourcen unterschiedlich bestimmt werden. Entspricht der Zustand keinem dieser genannten, so ist das Verhalten `blockiert`.
Es folgt daher die Bestimmung des Prozesszustandes für das Vorliegen der Bedingungen $\mathcal{C}$ und Ressourcen $\mathcal{R}$ als ID, als Typ, als $Klasse$ und als $LLMerkmale$.

ID-basiert In diesem Fall besitzen sowohl alle Bedingungen $\mathcal{C}$ als auch alle Ressourcen $\mathcal{R}$ eine eindeutige ID. Daher gilt $\forall k, |\hat{H}_{i,k}| \leq 1$ und $\forall k, |\hat{V}_k| \leq 1$. Es gelten alle Bedingungen

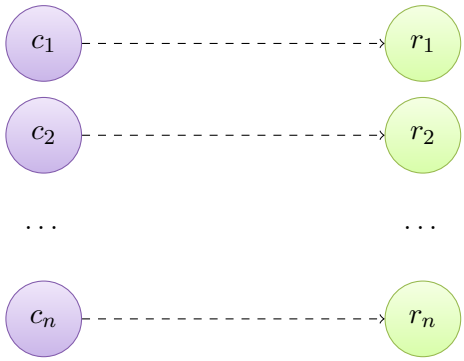


Abbildung 3.2: Erfolgt ein Matching der Bedingungen $\mathcal{C}$ und Ressourcen $\mathcal{R}$ auf der Abstraktionsebene ID, ist die Zuordnungen eindeutig.

$c_{i,k} \in C_i$ als erfüllt, so dass das Verhalten B_i in den Zustand `aktiv` wechseln kann, wenn

$$|C_i| \leq |\hat{H}_{i,k}| + |\hat{V}_k| \tag{3.4}$$

erfüllt ist. Dies gilt, da die mögliche Zuordnungen zwischen Bedingungen und Ressourcen bijektiv ist und sich eine passende Ressource nur exklusiv in einer der beiden Mengen H oder V befinden kann (Abbildung 3.2). Weiterhin kann B_i in den Zustand `bereit` wechseln, falls gilt:

$$|C_i| \leq |\hat{H}_{i,k}| + |\hat{V}_k| + |\bigcup_{h \neq i} \hat{H}_{h,k} \cap \mathcal{R}_\mathrm{p}|. \tag{3.5}$$

Das heißt, dass das Verhalten die benötigten Ressourcen entweder bereits hält, sie noch frei verfügbar sind oder einem anderen Verhalten $B_h \neq B_i$ entzogen werden können, sofern sie preemptiv sind.

Typen-basiert Betrachtet man die Zuordnung der Ressourcen $\mathcal{R}$ zu den Bedingungen $\mathcal{C}$ auf der Abstraktionsebene Typ, existiert keine eindeutige Zuordnung mehr zwischen einer Ressource $r \in \mathcal{R}$ und einer Bedingung $c \in \mathcal{C}$. Allerdings ist das Matching der Ressourcen immer noch stark eingeschränkt, was sich in einem k-partiten Zuordnungsgraphen widerspiegelt (Abbildung 3.3). Denn alle Ressourcen R, die zu einer Bedingung $c_{i,k} \in C_i$ passen (matchen),

sind nicht mehr eindeutig, da sie vom gleichen Typ sind. Zusätzlich können natürlich auch mehrere Bedingungen vom gleichen Typ existieren. Dies ist beispielsweise der Fall, wenn zwei

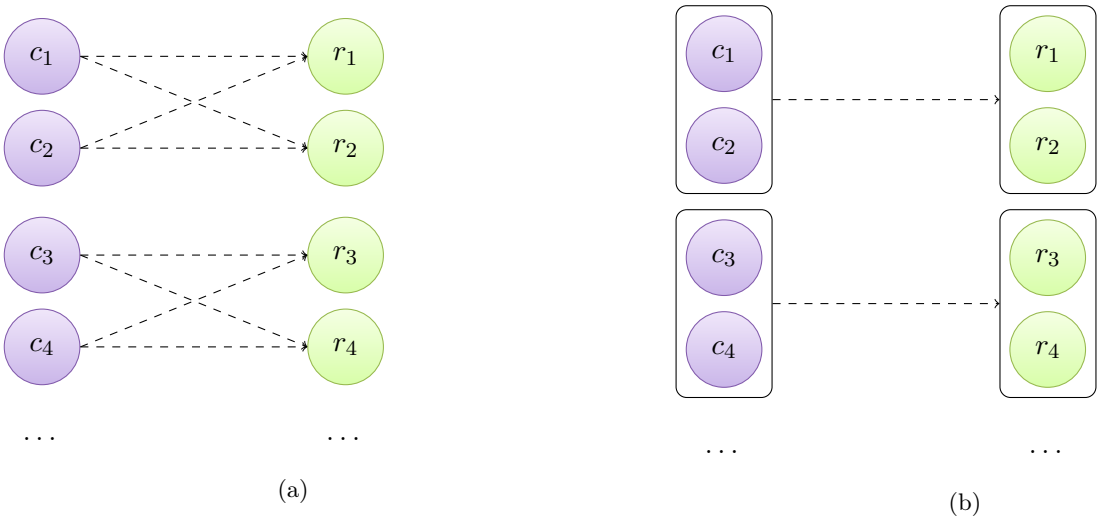


Abbildung 3.3: Aufgrund einer *Typ*-basierten *match*-Funktion entstehen mehrere Mögliche Zuordnungen zwischen Bedingungen und Ressourcen (a). Durch Zusammenfassen der Bedingungen und Ressourcen nach ihrem *Typ*, ist die Zuordnung zwischen diesen Gruppen jedoch eindeutig (b).

gleiche Bauteile für eine Montage benötigt werden. Die Bedingungen und Ressourcen lassen sich daher gruppieren indem man die Menge C_i in Teilmengen G_g aufteilt mit

$$\bigcup_{g \in \mathbb{N}^+} G_g = C_i. \tag{3.6}$$

Alle Elemente einer Gruppe G_g erfüllen die Bedingung

$$\forall c_i \in G_g, \forall c_j \in G_g,\ \forall r \in \mathcal{R},\ match(c_i, r) = match(c_j, r), \tag{3.7}$$

d.h. sie stellen die gleichen Anforderungen an eine Ressource r. Gleichzeitig gilt

$$\forall c_i \in G_g,\ \forall c_j \notin G_g,\ \forall r \in \mathcal{R},\ match(c_i, r) \neq match(c_j, r). \tag{3.8}$$

Dies bedeutet, dass eine Ressource immer nur den Anforderungen der Bedingungen einer Gruppe exklusiv genügen kann. Durch diese Gruppierung lassen sich die Verhaltenszustände effizient bestimmen. Das Verhalten ist `aktiv` wenn

$$|G_g| \leq |\hat{H}_{i,g}| + |\hat{V}_g| \tag{3.9}$$

für jede Menge $G_g \subseteq C_i$ erfüllt ist. Hierbei entspricht $\hat{H}_{i,g} \subseteq H_i$ der Menge, die eine Bedingung $c_k \in G_g$ erfüllt

$$\hat{H}_{i,g} = \{r \in H_i | match(c_k, r) = true\} \quad \text{mit } c_k \in G_g. \tag{3.10}$$

Dies gilt analog für $\hat{V}_g$. Gleichung 3.9 bedeutet, dass für jede Gruppe von Bedingungen G_g genügend Ressourcen des passenden Typs direkt verfügbar sind.

Auch hier kann B_i analog zum ID-basierten Fall in den Zustand `bereit` wechseln, wenn die Bedingung

$$|G_g| \leq |\hat{H}_{i,g}| + |\hat{V}_g| + |\bigcup_{h \neq i} \hat{H}_{h,g} \cap \mathcal{R}_\mathrm{p}| \tag{3.11}$$

für jede Gruppe G_g erfüllt ist.

Klassen-basiert Betrachtet man die Erfüllung der Bedingungen durch Ressourcen, wenn beide als Klassen dargestellt sind, gibt es die Möglichkeit zuzulassen, dass eine Ressource unterschiedliche Klassen von Bedingungen erfüllen kann oder nicht. Lässt man dies nicht zu, verhält sich die Klassen-basierte Zuordnung wie die Typ-basierte. Lässt man zu, dass eine Ressource verschiedene Klassen von Bedingungen erfüllt, erhält man ein Zuordnungsproblem auf einem bipartiten Graphen, bei dem eine möglichst hohe Anzahl an Zuordnungen dieser Bedingungen zu Ressourcen erreicht werden muss [Diet13]. Dies wird klar, wenn man sich ein kurzes Gegenbeispiel überlegt. Angenommen es existieren genau zwei Ressourcen r_1 und r_2 und es soll der Zustand eines Verhaltens ermittelt werden, dessen aktuelle Transition die Bedingung $C = \{c_1, c_2\}$ besitzt. Nimmt man weiter an, dass r_1 sowohl für c_1 als auch c_2 passend ist, r_2 jedoch nur für c_1, dann ist sofort ersichtlich, dass ein naiver iterativer Abgleich zunächst nur c_1 durch r_1 erfüllt. Da r_2 jedoch die Bedingung c_2 nicht erfüllt, könnte der Prozess nicht in den `bereit`-Zustand, obwohl dies bei geschickter Zuordnung sehr wohl möglich wäre. Daher muss eine möglichst hohe Anzahl Zuordnungen von Ressourcen zu Bedingungen erfolgen. Idealerweise entspricht die Anzahl der Zuordnungen der Anzahl der Bedingungen der Transition.
Man betrachtet zur Lösung den bipartiten Graph $G = \{U, K\}$. Bei $U = C_i \cup W$ handelt es sich um die Vereinigung der Bedingungen C_i einer Transition und einer Menge von Ressourcen $W \subseteq \mathcal{R}$, die zunächst noch nicht weiter definiert werden soll. Die Menge der Kanten entspricht $K = C_i \times W$, d.h. Kanten verlaufen von C_i nach W. Weiterhin gebe es eine $n \times m$ große Zuordnungs-Matrix O, deren Einträge $o_{kl} = match(c_k, w_l)$ beinhalten. Gesucht ist nun eine Zuordnung Z im bipartiten Graphen $G = \{C_i \cup W, C_i \times W\}$ so dass

$$\sum_{k,l \in O} o_{kl} = \max \tag{3.12}$$

Im Fall des Klassen-basierten Matchings gilt $o_{kl} \in \{0, 1\}$, d.h. eine Ressource genügt einer Bedingung oder nicht. Durch Lösen der Maximierungsungleichung in Kombination mit dem binären Wertebereich von o erhält man ein kardinalitätsmaximales Matching, das eine Menge $Z \subseteq K$ erzeugt. Sofern es eine Lösung zur Erfüllung aller Bedingungen C_i eines Transition T_i gibt, entspricht sie dem kardinalitätsmaximalen Matching.
Die Lösung der Maximierungsungleichung und damit die Berechnung von Z kann über einen Auktions-Algorithmus [Drex13, Elli13, Bert91, Doms07] erfolgen (Algorithmus 1), der jeder Ressource $r \in \mathcal{R}$ einen Besitzer $c \in C_i$ zuordnet. Der Algorithmus endet nach endlich vielen Schritten aufgrund der δ Addition und besitzt eine asymptotische Laufzeit von $O(Dn^3)$ mit $D = \max o_{kl}$. Die Menge Z enthält nach Beendigung alle Paare (k, l) mit zulässiger Zuordnung $match(c_k, r_l)$. Diese Menge kann nun zur Feststellung des Verhaltenszustandes genutzt werden. Ein Verhalten B_i ist `aktiv`, wenn der Auktions-Algorithmus eine Lösung Z findet mit

$$|Z| = |C_i| \tag{3.13}$$

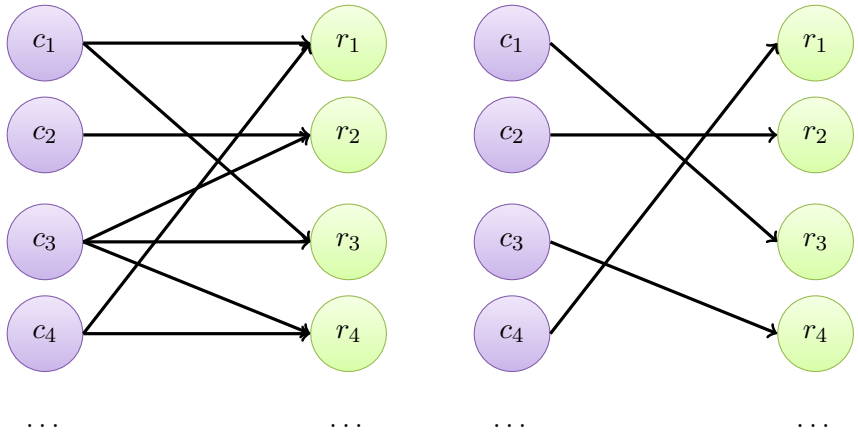


Abbildung 3.4: Durch das Klassen-basierte Matching entstehen mehrere Zuordnungsmöglichkeiten zwischen Ressourcen und Bedingungen (a). Existiert für jede Bedingung c_i einer Transition eine Ressource r_j, kann dies durch eine optimale Zuordnung gelöst werden (b).

und sich die Ressourcen W zusammensetzen als

$$W = H_i \cup V, \tag{3.14}$$

da zu jeder Bedingung c_k eine passende Ressource r_l ermittelt wurde. Analog ist ein Verhalten B_i `bereit`, wenn der Auktions-Algorithmus eine Lösung Z findet mit

$$|Z| = |C_i| \tag{3.15}$$

und sich die Ressourcen W aber zusammensetzen aus

$$W = H_i \cup V \cup \big(\bigcup_{h \neq i} H_h \cap \mathcal{R}_\mathrm{p} \big). \tag{3.16}$$

Subsymbolische Darstellung Betrachtet man die Zuordnung der Ressourcen zu den Bedingungen auf Ebene der LLMerkmale, erweitert sich das zuvor gezeigte Zuordnungsproblem um eine Gewichtung der Kanten. Denn während bei der klassenbasierten Zuordnung eine Ressource eine Bedingung entweder erfüllt oder nicht, existiert bei der subsymbolischen Darstellung lediglich ein bestimmter Grad der Übereinstimmung durch die match-Funktion (Abbildung 3.5). Der zuvor genutzte Auktions-Algorithmus kann jedoch nur mit ganzzahligen Kantengewichten umgehen. Es gibt nun verschiedene Möglichkeiten auf diese Einschränkung zu reagieren.
Um den Auktions-Algorithmus trotzdem zu verwenden, ist es möglich das Matching auf einen Ganzzahlwert abzubilden, wie in Gleichung 2.11 auf Seite 41, falls eine Schranke θ überschritten wird

$$o_{kl} = \begin{cases} 1 & \text{falls} \quad match(c_k, w_l) \geq \theta \\ 0 & \text{sonst.} \end{cases} \tag{3.17}$$

Initialisierung
$r \in W\ \forall r : p_r \leftarrow 0$;
$\forall r : Besitzer(r) \leftarrow NIL$;
Füge alle Bieter $c_i \in C_i$ in Wartemenge Q ein;
Setze $\delta := 1/(|C_i| + 1)$;
Bieten
while $Q \neq \emptyset$ **do**
    Entnehme Eintrag k aus Q;
    Bestimme ein r, das $o_{kr} - p_r$ maximiert (Bei Gleichheit: beliebig);
    if $o_{kr} - p_r > 0$ **then**
        Füge (aktuellen) Besitzer(r) in Q ein;
        Füge ((aktuellen) Besitzer(r), k) in Z ein ;
        $Besitzer(r) \leftarrow k$;
        $p_r \leftarrow p_r + \delta$;
    else
        k wird inaktiv;
    end
end
Ausgabe: Menge Z aller Paare (Besitzer(l), k) mit $Besitzer(l) \neq NIL$

Algorithm 1: Auktions-Matching zur Lösen des Zuordnungsproblems von Bedingungen und Ressourcen

Damit ginge allerdings die Güte einer Übereinstimmung verloren und es würde keine optimale Zuordnung mehr stattfinden, da eine schlecht passende Ressource das gleiche Gewicht wie eine perfekt passende besitzen würde.
Auch eine Transformation aller Werte der Zuordnungsmatrix in $\mathbb{N}_0$ wäre denkbar. Allerdings geht das größte Gewicht in die Laufzeit mit ein [Diet13], so dass diese Option aus Performance-Gründen wenig Sinn ergibt.
Eine geschicktere Möglichkeit ist die Nutzung eines Lösungsalgorithmus der direkt mit Gewichten $o \in \mathbb{R}$ umgehen kann, wie beispielsweise die Ungarische Methode [Burk12, Munk57]. Die Ungarische Methode löst das Problem einer kardinalitätsmaximalen Zuordnung, indem sie eine perfekte Zuordnung $Z \subseteq K$ für zwei Knotenmengen C_i und W findet mit

$$\sum_{(c,w)\in O} match(c, w) = \max, \qquad c \in C_i, w \in W \tag{3.18}$$

und

$$match(c, w) \to \mathbb{R} \tag{3.19}$$

Allerdings fordert die Methode, dass $|C_i| = |W|$ gilt. Um dies zu erreichen wird die kleinere Menge üblicherweise künstlich vergrößert. Sollte $|C_i|$ kleiner als $|W|$ sein, wird C_i mit $|W|-|C_i|$ Platzhalter-Werten ergänzt und die so zusätzlich entstandenen Kanten mit $o = 0$ gewichtet (keine Übereinstimmung). Dadurch kann der Algorithmus für die Zuordnung verwendet werden. Sollte dagegen $|C_i| > |W|$ sein, tritt der Trivialfall ein, dass nicht genügend Ressourcen für alle Bedingungen vorhanden sind. In diesem Fall kann der Verhaltenszustand direkt als `blockiert` klassifiziert werden.
Eine effiziente Implementierung der ungarischen Methode erhält man mit dem Kuhn-Munkres

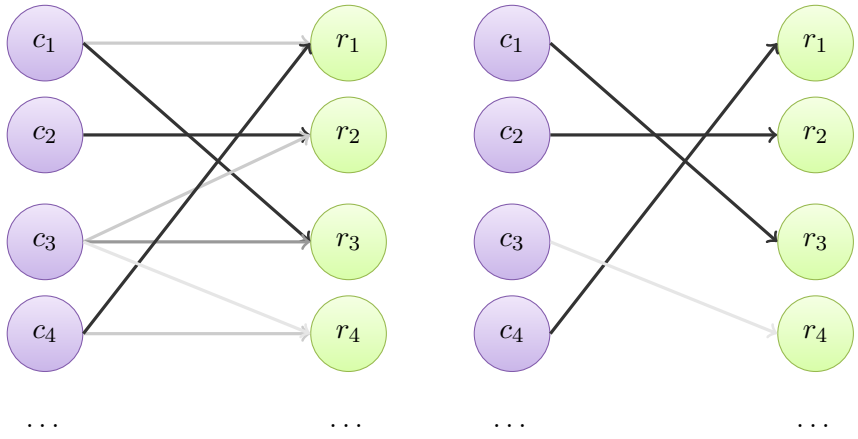


Abbildung 3.5: (a) Mögliche Zuordnungen zwischen Ressourcen und Bedingungen bei Verwendung der $LLMerkmale$, wobei die Intensität des Grauwerts der Verbindungslinie der Güte des Matchings entspricht. (b) Optimale Zuordnung, so dass alle c_i erfüllt werden.

Algorithmus [Kuhn10], der hier jedoch aus Gründen des Umfangs nicht näher betrachtet wird. Die Bestimmung des Verhaltenszustandes erfolgt analog zur Lösung mit dem Auktions-Algorithmus. Ein Verhalten B_i ist `aktiv`, wenn der Kuhn-Munkres-Algorithmus eine Lösung Z findet so dass

$$|Z| = |C_i| \qquad \text{mit} \quad W = H_i \cup V \tag{3.20}$$

und ist `bereit`, wenn der Algorithmus eine Lösung Z findet mit

$$|Z| = |C_i| \qquad \text{mit} \quad W = H_i \cup V \cup \Big(\bigcup_{h \neq i} H_h \cap \mathcal{R}_\mathrm{p} \Big) \tag{3.21}$$

3.3 Instantiierung von Verhalten

Ebenso wie bei einem Betriebssystem macht es bei manchen Programmen Sinn, dass sie mehrfach parallel ausführbar sind, wogegen manche Prozesse nur einmal im System vorhanden sein sollten. Übertragen auf ein Robotersystem bedeutet dies, dass mehrere Manipulationsverhalten gleichzeitig existieren können, die das selbe Verhaltensmodul realisieren, die aber auf unterschiedliche Objekte angewendet werden. Dagegen macht es wenig Sinn, eine randomisierte Bewegung zur Suche nach Objekten mehrfach parallel auszuführen. Daher werden zwei Klassen von Verhalten definiert, nämlich regulär und singleton.
Während mehrere Verhalten eines *regulären* Verhaltensmoduls existieren können, kann von einem *singleton* zu einem Zeitpunkt immer nur eine Instanz existieren. Um dies zu realisieren, hält das System eine Liste aller bekannter Verhaltensmodule $\mathcal{M}$. Sie ist unterteilt in reguläre Verhalten $\mathcal{M}_r$ und singleton Verhalten $\mathcal{M}_s$. Das System instanziert zunächst ein Verhalten jedes Verhaltensmoduls. Diese Verhalten sind in der Regel sofort blockiert, da die Primärbedingungen nicht erfüllt sind. Sobald ein reguläres Verhalten jedoch von `blockiert` nach `bereit` oder `aktiv` wechselt, wird ein neues Verhalten des Verhaltensmoduls erstellt. Somit

kann das System weiterhin auf ähnliche oder gleiche Stimuli aus der Umwelt reagieren. Die Menge der regulären Verhalten erfüllen somit folgende Bedingung

$$\forall M \in \mathcal{M}_r, \exists B_M \in \mathcal{B}_M \text{ mit } status(B_M) = \texttt{blockiert}. \tag{3.22}$$

Hierbei stellt B_M eine Instanz eines Verhaltens dar, welches das Verhaltensmodul M repräsentiert und $\mathcal{B}_M$ die Menge aller nicht-terminierten Verhalten, die M repräsentieren.
Dagegen werden singleton Verhalten jeweils nur durch eine Instanz repräsentiert. Sie erfüllen also die Bedingung

$$\forall M \in \mathcal{M}_s, \exists B_M \in \mathcal{B}_M \text{ mit } s_c \notin S_F. \tag{3.23}$$

Wie in Kapitel 2.5.2 definiert, handelt es sich bei s_c um den aktuellen Zustand des Verhaltens und S_F um die Menge der Finalzustände. Ein neues Verhalten wird nur erzeugt, wenn alle bereits existierenden Verhalten $\mathcal{B}_M$ des Verhaltensmoduls den Zustand `terminiert` besitzen. Durch diese Strategie existiert ein Verhalten für jede Aufgabe, die das System ausführen soll. Durch das Vorliegen all dieser Verhalten, können entsprechende Scheduling-Algorithmen entworfen werden, welche eine Koordination ermöglichen.
Die Menge der Verhaltensmodule kann sich dynamisch ändern. Kommen neue Verhaltensmodule hinzu, werden auch von ihnen Verhalten basierend auf den o.g. Regeln instantiiert. Sollen Verhalten entfernt werden, wird das Verhaltensmodul aus $\mathcal{M}$ entfernt, so dass keine Instantiierung mehr stattfindet. Bereits ausgeführte Verhalten werden allerdings nicht abgebrochen um Inkonsistenzen zu vermeiden.

3.4 Koordination von Verhalten

Sind mehrere Verhalten im System instantiiert, müssen diese auch koordiniert werden. Die Verhaltenskoordination entspricht dem Problem der Action Selection, also welches Verhalten als nächstes eine Aktion emittieren darf.
Bei der Umsetzung gibt es zwei Möglichkeiten, die davon abhängen, ob Wissen über die eingesetzte Domäne vorhanden ist oder nicht. Zwar wäre es möglich durch Vorgabe einer Einsatzdomäne sehr effiziente Koordinationsmechanismen zu erstellen, allerdings würde es den Einsatzbereich unter Umständen stark einschränken. Mit dem Ziel des Einsatzes sowohl im Haushalt als auch in der Industrie, existiert kein Wissen über die später konkret programmierten Aufgaben. Daher muss ein allgemeiner Mechanismus zur Koordination der Verhalten verwendet werden.
Diese Anforderung gilt ebenso im Bereich der Betriebssysteme. Dort ist dem Betriebssystem die Aufgabe der Prozesse nicht bekannt. Zur Koordination dieser Prozesse werden Scheduling-Verfahren eingesetzt. Während sich bei Betriebssystemen das Scheduling v.a. auf die Zuteilung der CPU-Zeit bezieht, bezieht es sich in dem hier entwickelten System auf Zeitslots, in dem ein Verhalten das Anrecht auf Ausführung einer Aktion erhält, die generell alle Ressourcen beinhalten kann. Zwar nimmt der Roboter in einem Robotersystem eine zentrale Rolle ein, allerdings können davon auch mehrere vorhanden sein und es können weitere Spezialmaschinen als aktive Komponenten existieren.
Da das System kein Wissen über den Einsatz und Zweck der Verhalten hat, können verschiedene Verhalten prinzipiell nur über statistische Daten koordiniert werden. Bei der Umsetzung wird zunächst geprüft welche Verhalten in den `bereit` Zustand wechseln können. Diese Liste wird durch einen Scheduling Algorithmus sortiert. Das Verhalten am Kopf der Liste wird `aktiv`. Anschließend kann optional innerhalb des Zeitslots geprüft werden, ob ein weiteres

Verhalten ebenfalls direkt in den Zustand `aktiv` wechseln kann. Durch diese optionale Möglichkeit fällt der hier entwickelte Ansatz in die Kategorie Multiple-Objective Action Fusion. Wird diese Option nicht wahrgenommen, ist der Ansatz im Bereich Action Selection einzuordnen. Die Unterkategorie bestimmt sich aufgrund der Wahl des Scheduling-Verfahrens (Kapitel 3.7).
Wie bei Betriebssystemen muss auch hier bei Wechsel des Verhaltens am Listenkopf ein Verhaltenswechsel durchgeführt werden, so dass die Ausgabe von Verhalten $B_i \in \mathcal{B}$ unterbrochen wird und Verhalten $B_j \neq B_i$ mit der Ausführung beginnen oder diese fortsetzen kann (Kapitel 3.5). Dafür ist es notwendig, den Kontext von Verhalten B_i zu sichern und den Kontext von Verhalten B_j wiederherzustellen.

3.5 Wechsel des aktiven Verhaltens

In diesem Kapitel wird der Wechsel des aktiven Verhaltens näher betrachtet. Dazu wird zunächst beschrieben, wie ein Kontextwechsel durchgeführt wird (Kapitel 3.5.1) und anschließend wird auf mögliche Abhängigkeiten zwischen den Ressourcen eingegangen (Kapitel 3.5.2).

3.5.1 Kontextwechsel

Bei modernen Betriebssystemen wird bei einem Prozesswechsel der Kontext eines Prozesses B_i gespeichert und der eines weiteren Prozesses B_j wiederhergestellt. Zunächst ist zu klären, was im Bereich eines Robotersystems alles als Kontext definiert ist. Anschließend ist zu klären, welche Rolle er beim Verhaltenswechsel in einem Ressourcen-basierten System spielt.
Unter Kontext versteht man alle nötigen Elemente zur Aufnahme einer Tätigkeit, in diesem Fall zum Ausführen oder Fortsetzen eines Verhaltens. Bei Verhalten sind in einer Transition die Bedingungen C gespeichert, die von Ressourcen R erfüllt werden müssen, um die Aktionen A dieser Transition durchführen zu können. Das heißt, der Kontext wird im Rahmen dieser Arbeit wie folgt definiert:

Definition 12 (Kontext) Als *Kontext* eines Verhaltens B_i gelten alle Ressourcen $R_T \subseteq H_i$, die zur Erfüllung der Bedingungen der aktuellen Transition T_i von B_i benötigt werden.

Das Sichern des Kontextes beruht also auf der Speicherung der Information, welche Ressourcen die Bedingungen der aktuellen Transition erfüllt haben. Und, falls es sich um aktive Ressourcen handelt, die Sicherung derer Zustände zum Zeitpunkt des Wechsels. Betrachtet man beispielsweise den Roboter als aktive Ressource, so besteht die Sicherung des Kontextes aus dem Speichern der Pose des Roboters. Bei den semi-aktiven Ressourcen, wie etwa einem Werkstück, wird dieses auf einer definierten Stelle abgelegt. Passive Ressourcen, also Raumelemente werden als nicht belegt markiert.
Beim Wiederherstellen des Kontextes durch ein Verhalten, werden die entsprechenden Ressourcen erneut akquiriert und der Zustand Ressourcen wiederhergestellt. D.h. die Pose des Roboters wird wieder eingenommen, evtl. abgelegte Objekte wieder gegriffen und Raumelemente als belegt markiert.

3.5.2 Abhängigkeiten zwischen Ressourcen

Anhand der passiven Ressourcen kann einfach aufgezeigt werden, dass Abhängigkeiten zwischen verschiedenen Ressourcen existieren können. Als Beispiel sei die Nutzung des Arbeitsraumes genannt, welcher entweder frei oder durch Ressourcen (z.B. Objekte) belegt sein kann. Diese Abhängigkeiten sind insbesondere beim Kontextwechsel zu beachten. Es lassen sich zwischen den einzelnen Ressourcen Abhängigkeiten identifizieren, aus denen sich ein Abhängigkeitsgraph generieren lässt.
Bei einem Abhängigkeitsgraphen $DG = \{R, E_{\text{DG}}\}$ handelt es sich um einen gerichteten Graphen mit Knoten $R \subseteq \mathcal{R}$, Kanten $E_{\text{DG}} \subseteq R \times R$ und einer Wurzel *root*. Eine Kante $e \in E_{\text{DG}}$ verläuft von r_i nach r_j, wenn die Ressource r_i von r_j abhängt. Verbindungen können dauerhaft existieren, wie etwa zwischen einem Roboterarm und einem Greifer oder dynamisch hinzugefügt oder verändert werden, wenn z.B. ein Objekt vom Roboter gegriffen wird. Ebenfalls enthalten sind die Abhängigkeiten einer Ressource vom benötigten Raum $V \subseteq \mathcal{R}_{\text{p}}$.
Möchte ein Verhalten B_i eine Ressource r akquirieren, die bereits von einem Verhalten B_j gehalten wird, müssen zunächst die Abhängigkeiten geprüft werden. Bei der Akquise werden alle Ressourcen R_d betrachtet, die von r abhängen und durch eine spezifische Preemptierungsfunktion gelöst. Da r selbst von Ressourcen R_c abhängen kann, müssen rekursiv alle R_c betrachtet werden und gegebenenfalls ebenso kurzzeitig akquiriert werden um die Preemption von r durchführen zu können.
Möchte B_i beispielsweise den Greifer akquirieren, der bereits ein Objekt o_2 gegriffen hat (Abbildung 3.6) muss diese Abhängigkeit gelöst werden indem das Objekt abgelegt wird. Dies ist zulässig, da es sich hierbei um eine preemptive Ressource handelt. Es muss zusätzlich der Roboterarm akquiriert werden, um das Objekt abzulegen, das heißt auch dessen Zustand muss gesichert werden, damit er von B_j später wieder verwendet werden kann. Zusätzlich muss geprüft werden, ob genügend Raum akquiriert werden kann, um das Objekt abzulegen. Anschließend kann ein evtl. gespeicherter Zustand von B_i wiederhergestellt werden indem die Abhängigkeiten wieder aufgebaut werden und die Zustände dieser Ressourcen wieder hergestellt werden. Das bedeutet, dass beispielsweise ein zuvor abgelegtes Objekt aufgegriffen wird und der Roboter anschließend die gesicherte Pose erneut einnimmt.

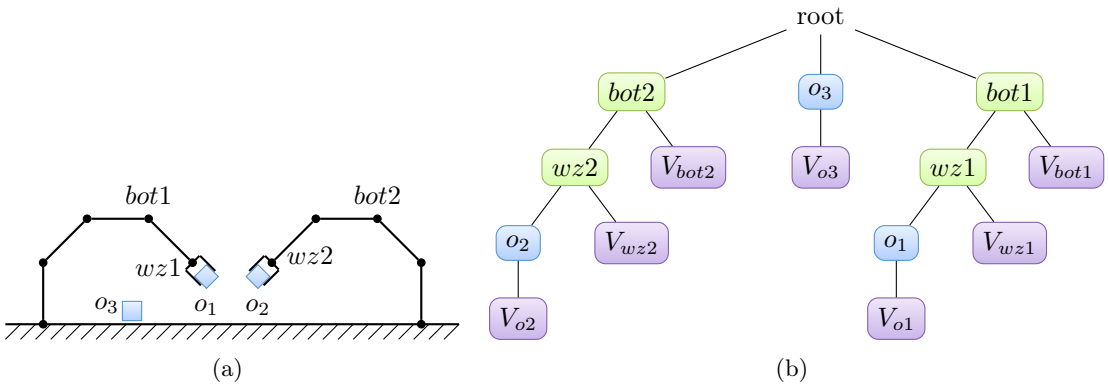


Abbildung 3.6: Unter (a) ist ein Zwei-Arm Roboterarbeitsplatz dargestellt, der aus Robotern (*bot*), Werkzeugen (*wz*) und Objekten (*o*) besteht. Unter (b) ist der zugehörige Abhängigkeitsgraph *DG* mit allen aktiven Ressourcen (grün), semi-aktiven Ressourcen (blau) und passiven Ressourcen (lila) zu sehen.

3.6 Klassifikation von Ressourcen

Die Menge der Ressourcen ist in zwei Klassen geteilt, nämlich in präemptive $\mathcal{R}_{\text{p}}$, wenn sie ihrem Besitzer entzogen werden können und nicht preemptiv $\mathcal{R}_{\text{np}}$, wenn dies nicht möglich ist.
Nun stellt sich die Frage, wie bestimmt werden kann, ob eine Ressource preemptiv ist oder nicht. Dies könnte manuell durch den Programmierer festgelegt werden, da dieser Wissen über besondere Ressourcen oder das zu implementierende Roboterprogramm besitzt. Behält man das Ziel der intuitiven Programmierung im Auge, sollte dies jedoch entweder automatisch geschehen oder durch eine simple Vorgabe des Programmierers während des Programmiervorgangs.
Kritisch ist hier der Verhaltenswechsel, denn dort können einem Verhalten Ressourcen entzogen werden. Ein Problem tritt immer dann auf, wenn ein Verhalten eine Ressource irreversibel verändert hat oder die Wiederherstellung des Ausgangszustandes der Ressource nur mit unverhältnismäßig großem Aufwand möglich ist. Es ist beispielsweise kein Problem, wenn ein Verhalten den Roboter benötigt und dessen Pose verändert. Die Wiederherstellung des vorherigen Zustandes ist problemlos möglich. Wird dagegen der Inhalt einer Tasse verschüttet ist das Wiederherstellen der vorherigen Zustandes schwierig bis unmöglich.
Betrachtet man die Einteilung der Ressourcen, tritt das Problem nur bei den semi-aktiven Ressourcen, also den Werkstücken auf. Deren Zustand kann sich durch direkte Manipulation oder durch Einwirkung eines Werkzeugs verändern. Werden die Ressourcen auf einer subsymbolischen Ebene repräsentiert oder ist das System mächtig genug, solche Änderungen symbolisch zu erkennen, ist es unkritisch. Denn in diesem Fall erfüllt die Ressource möglicherweise nach einem Verhaltenswechsel schlicht die zugehörige Bedingung nicht mehr und das Verhalten muss entsprechend reagieren, indem es eine andere passende Ressource findet. Ist dies nicht möglich, müssen solche veränderten Ressourcen bis zur Terminierung des Verhaltens gesperrt werden, d.h. so lange in die Menge $\mathcal{R}_{\text{np}}$ eingefügt werden.
Eine konservative Lösung wäre die Klassifizierung aller semi-aktiven Ressourcen als nicht preemptiv, sobald sie von einem Verhalten akquiriert wurden, bis hin zu dessen Terminierung. Dieses Vorgehen könnte allerdings zu einer starken Sequenzierung der Verhalten führen, was die Reaktivität und Mächtigkeit des Systems einschränkt.
Eine weniger restriktive Lösung wäre alle potentiell manipulierten Objekte zu erkennen, indem man Kenntnis über das Werkzeug besitzt. Verwendet das Robotersystem ein direkt manipulierendes Werkzeug, wie etwa eine Schweißspitze oder einen Bohrer, müssen alle Objekte als manipuliert angesehen werden, die mit dem montierten Werkzeug in Kontakt kommen. Besitzt der Roboter ein Haltewerkzeug, wie etwa einen Greifer, werden damit in der Regel Pick-and-Place Operationen durchgeführt oder Handhabungsaufgaben bei denen das gegriffene Objekt r_t zunächst als manipulierendes Werkzeug eingesetzt wird. In diesem Fall muss angenommen werden, dass alle Objekte, die mit r_t direkten Kontakt hatten manipuliert wurden.
Die Ermittlung von Kontaktzuständen werden in verschiedenen Arbeiten bereits betrachtet [Aleo06b, Hira92, Kond08] und sollen hier nicht näher ausgeführt werden. Absolute Sicherheit gibt es auch bei dieser Methode nicht. Um sicher zu gehen, müsste das System mächtig genug sein, Veränderungen der Ressourcen zu erkennen. Da eine solche Erkennung jedoch nicht Teil dieser Arbeit ist, werden für die später durchgeführten Experimente alle Ressourcen als preemptiv klassifiziert.

3.7 Klassische Schedulingverfahren

Aus dem Bereich der Betriebssysteme sind verschiedene präemptive und nicht präemptive Schedulingverfahren bekannt. Im Gegensatz zu den präemptiven werden bei den nicht präemptiven Verfahren einmal zugeteilte Verhalten nicht unterbrochen sondern bleiben aktiv bis sie selbst den aktiven Zustand verlassen. Im Folgenden werden einige dieser Verfahren nach [Tane07] hinsichtlich der Verwendbarkeit für Roboter untersucht.

3.7.1 Nicht Preemptiv

Die Verwendung eines nicht-präemptiven Schedulings (Batch Betrieb) scheint zunächst unvereinbar mit dem Ziel der Unterbrechbarkeit aus Kapitel 2.1. Dennoch können diese Verfahren nützlich sein, um im Bereich der Industrie kleinere Losgrößen im Stapelbetrieb abzuarbeiten, denn diese Gruppe maximiert den Durchsatz des Systems bei gleichzeitiger Minimierung der durchschnittlichen Bearbeitungszeit [Tane07].

First Come First Serve

Ein mögliches Verfahren ist *First Come First Serve* (*FCFS*) bei dem neue Prozesse an das Ende einer Liste angehängt und der Reihe nach abgearbeitet werden. Das Scheduling ist fair, allerdings können sehr lange Prozesse kurze verzögern, was zu langen Wartezeiten führen kann. Die Adaption zur Verwendung in einem Robotersystem ist trivial. Das Einhängen eines Verhaltens in die Liste erfolgt sobald ein Verhalten von `blockiert` nach `bereit` wechselt.

Shortest Job First

Ein weiteres Verfahren ist *Shortest Job First* (*SJF*). Die Prozesse werden nach ihrer Bearbeitungszeit sortiert und nacheinander abgearbeitet. Hierbei gibt es zwei Probleme. Zum einen ist die Bearbeitungszeit der Prozesse nicht immer bekannt und kann dann nur geschätzt werden. Zum anderen ist SJF nur dann optimal, wenn alle Jobs gleichzeitig vorliegen. Ist dies nicht der Fall, erhöht sich die Wartezeit [Tane07].
Die Probleme bei der Umsetzung auf einem Robotersystem entsprechen denen bei der Umsetzung in einem Betriebssystem. Die Länge der Verhaltens ist in der Regel nicht bekannt und muss geschätzt werden. Eine Möglichkeit wäre es, Statistiken über bereits ausgeführte Verhalten eines Verhaltensmoduls anzulegen und damit eine mittlere Bearbeitungszeit oder eine Worst-Case Abschätzung über die Dauer durchzuführen. Alternativ können auch direkt beobachtbare Eigenschaften der zugehörigen Verhaltensmodule zum Vergleich herangezogen werden, wie z.B. die Länge der Trajektorie, die Häufigkeit der Ressourcenwechsel oder die Anzahl der Werkzeugoperationen.

3.7.2 Preemptiv

Wie bereits erwähnt, kann beim preemptiven Scheduling ein Prozess zu Gunsten eines anderen Prozesses unterbrochen werden. Damit ist es prinzipiell möglich, schneller auf Stimuli der Umwelt zu reagieren.

Shortest Remaining Time Next

Eine präemptive Abwandlung von SJF ist das Verfahren *Shortest Remaining Time Next* (*SRN*). Sobald ein neuer Prozess hinzukommt, wird geprüft ob die restliche Bearbeitungszeit des aktuellen Prozesses größer ist als die des neu hinzukommenden. Ist dies der Fall wird der neue Prozess zuerst ausgeführt.
Dieses Scheduling ist prinzipiell geeignet für die Roboteranwendung, allerdings besteht auch hier das Problem, dass die restliche Bearbeitungszeit bekannt sein muss. Mögliche Lösungen entsprechen denen von Shortest Job First.

Round Robin

Ein weit verbreitetes Verfahren ist *Round Robin* (*RR*). Jedem Prozess wird ein Zeitintervall (Quantum) zugeordnet, währenddessen er aktiv sein kann. Falls nach Ende des Quantums der Prozess noch immer aktiv ist, wird er unterbrochen und ein anderer Prozess ausgeführt. Beendet oder blockiert der Prozess vor Ende des Quantums, erfolgt der Prozesswechsel schon zu diesem Zeitpunkt. Problematisch an RR ist die Wahl des Quantums. Ist es zu klein, verschwendet das System zu viel Zeit mit dem häufigen Wechsel von Prozessen. Ist das Quantum zu groß, erhöht sich die Reaktionszeit und wird bei vielen Prozessen unangemessen hoch.
In Betriebssystemen wählt man typischerweise ein Quantum mit etwa 20-50 Millisekunden. Bei Roboteranwendungen muss diese Grenze deutlich höher gewählt werden, da hier nicht nur Informationen übertragen werden, sondern reale physische Objekte mit begrenzter Geschwindigkeit bewegt werden. Verschiedene Größen für das Quantum werden in Kapitel 5 untersucht.

Priority Scheduling

Beim *Priority Scheduling* (*PS*) werden den Prozessen verschiedene Prioritäten zugeordnet. Es existiert eine nach den Prioritäten sortierte Liste und der Prozess mit der höchsten Priorität wird ausgeführt. Falls ein Prozess mit höherer Priorität hinzukommt, wird der aktuelle Prozess preemptiert und der neue Prozess ausgeführt.
Auch hier ist die Übertragung auf ein Robotersystem trivial. Mit der Forderung, dass die Verhalten durch den Benutzer demonstriert werden, entsteht nur das Problem der Prioritätenzuweisung. Einer automatischen Vergabe der Prioritäten fehlt die Grundlage. Mögliche Lösungen wären das Setzen der Prioritäten durch den Benutzer im Anschluss an eine Demonstration oder das Eingreifen des Benutzers bei der Ausführung. Dieser könnte die ausgeführten Verhalten abbrechen bis das gewünschte Verhalten ausgeführt wird. Das System müsste die Prioritäten anschließend entsprechend angleichen.

3.7.3 Bewertung

Bei Betrachtung von Tabelle 3.2 zeigt sich, dass sowohl SRN als auch PS gute Kandidaten für den universellen Einsatz in Industrie und Haushalt sind, sofern man davon ausgeht, dass es dem Benutzer möglich ist die Verhalten in irgendeiner Form zu priorisieren. Beide besitzen ihre Berechtigung auch im Hinblick auf „menschliches Scheduling“. Bei der Bearbeitung langer Aufgaben werden oft kürzere eingeschoben um die Antwortzeit zu senken. Auf der anderen Seite werden Aufgaben auch nach ihrer Dringlichkeit erledigt. Die experimentelle Untersuchung aller Scheduling-Verfahren erfolgt in Kapitel 5.

Algorithmus	Eignung KMU	Eignung Haushalt	Vorteile	Nachteile	Herausforderungen bei Umsetzung
FCFS	o	–	Fair, einfach	Evtl. hohe Antwortzeit	–
SJN	o	–	Min. mittlere. Bearbeitungszeit	Verhungern langer Verhalten	Ermittlung Taskgröße
SRN	+	+	Min. mittlere. Bearbeitungszeit	Verhungern langer Verhalten	Ermittlung Taskgröße
RR	o	+	Fair	Verhungern hochpriorer Verhalten	Quantumsgröße
PS	+	+	Priorisierung möglich	Verhungern möglich	Prioritätenvergabe

Tabelle 3.2: Bewertung der Schedulingverfahren zum Einsatz in einem Robotersystem

3.8 Verhaltensbasiertes Scheduling

Zwar sind die vom Benutzer programmierten Aufgaben nicht a priori bekannt, allerdings lässt sich der verhaltensbasierte Ansatz zur Optimierung der Aufgabenbearbeitung durch den Roboter nutzen. Gerade im Bereich der KMU kommt es häufig vor, dass ein Stapelverarbeitungsbetrieb mit geringer Losgröße vorliegt. Bei bestimmten Aufgabentypen lassen sich hier Optimierungen vornehmen, indem man Verhalten des gleichen Verhaltensmoduls gruppiert, kombiniert und redundante Schritte eliminiert.
Dazu betrachtet man zunächst eine allgemeine Task-Struktur für Manipulationsaufgaben. In [Frie96] wird diese als sogenannte *Object Group (OG)* bezeichnet (Abbildung 3.7). Diese besteht im Prinzip aus dem Aufgreifen, mehrfachen Verwenden und anschließendem Ablegen eines Objektes.

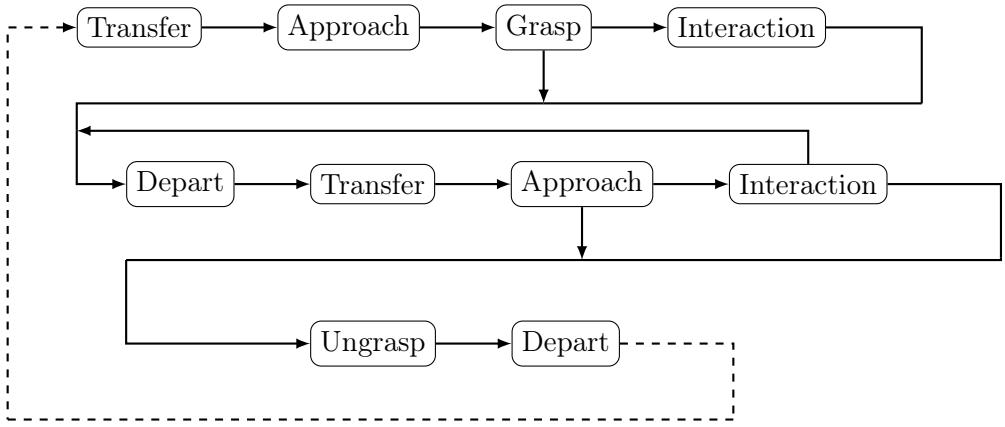


Abbildung 3.7: Allgemeine Task Struktur einer *Object Group* für Manipulationsaufgaben nach [Frie96].

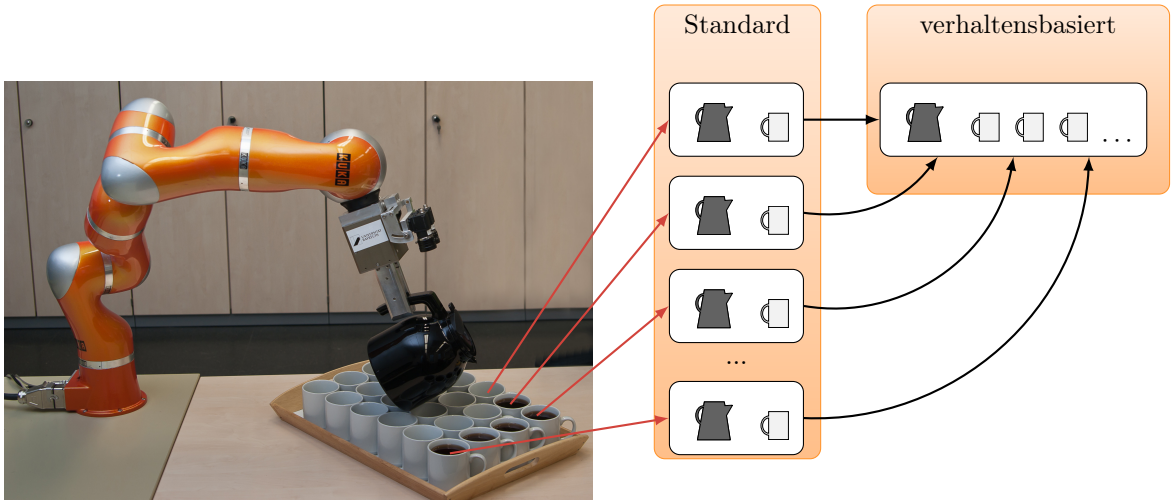


Abbildung 3.8: Zusammenfassung von Verhalten und Elimination von redundanten Operationen. Anstatt die Kaffeekanne n Mal aufzugreifen um n Tassen einzeln zu füllen, wird die Kanne ein Mal gegriffen und anschließend n Tassen gefüllt.

Damit kann beispielsweise die Umsetzung des Einschenkens aller Tassen mit jeweils einem Verhalten (Abbildung 3.8) zusammengefasst werden zu einem. Voraussetzung dafür ist, dass in der Vorbereitungsphase ein Werkzeug aufgenommen wird, dessen Funktion (Operator) mehrmals auf die zu manipulierenden Objekte angewendet werden kann (Operanden). Dies ist für eine Vielzahl von Anwendungen im Haushalt und Industrie der Fall. Ist diese Voraussetzung erfüllt, kann die komplette Gruppe als ein einziges Meta-Verhalten betrachtet werden und nahtlos in ein bestehendes Scheduling-Verfahren integriert werden. Experimente dazu werden in Kapitel 5 durchgeführt.
Zur Bestimmung der Einsparungshöhe betrachtet man ein Verhalten dessen Struktur durch die Aktionsfolge $PI-MA-PL$ darstellbar ist. Hierbei steht PI für ein Pick, MA für eine Manipulation und PL für ein Place. Werden n dieser Verhalten durch diese Art des Schedulings ausgeführt, beträgt die Einsparung

$$(n-1)\cdot(PI+PL)-O \tag{3.24}$$

Dabei beschreibt O den Overhead der entsteht, um den Transfer von einer Manipulation zur nächsten durchzuführen, der normalerweise in PI und PL integriert ist.
Betrachtet man Abbildung 3.8 könnte man annehmen, dass ein Skalierungsproblem auftreten kann, wenn eine große Menge Verhalten zusammengefasst wird. Im Beispiel des Einschenk-Verhaltens kann die Kanne nach dem Einschenken einiger Tassen leer sein. Allerdings wäre dies auch bei n Einzelverhalten der Fall. Zur Lösung müsste das System diese Abweichungen erkennen können, was jedoch nicht Fokus dieser Arbeit ist. Auf diesen Punkt wird jedoch im Ausblick kurz eingegangen.

3.9 Deadlocks

Es können wie im Bereich der Betriebssysteme Deadlocks entstehen, da alle Bedingungen nach [Coff71] erfüllt sind. Deadlocks können entstehen, wenn Verhalten u.a. Ressourcen wechselsei-

tig blockieren und anfordern. Die Erkennung lässt sich einfach über einen Betriebsmittelgraphen realisieren, da die Ressourcen im System alle explizit modelliert sind. Durch die Abbildung auf ein prozessbasiertes System lassen sich die Standardtechniken moderner Betriebssysteme anwenden. Die Situation lässt sich einfach lösen durch das Festlegen einer geordneten linearen Reihenfolge für Ressourcen bei der Akquirierung. Da die gesamte Ressourcenanforderung implizit bekannt ist, wäre auch die Verwendung des Bankier-Algorithmus denkbar oder das Erzwingen einer kompletten Betriebsmittelanforderung. Die Umsetzung solcher Mechanismen wird hier nicht näher betrachtet. In den Experimenten dieser Arbeit wird das Problem ausgeschlossen indem, wie bereits erwähnt, alle Ressourcen als preemptiv klassifiziert werden.

3.10 Zusammenfassung

In diesem Abschnitt wurde die quasi-parallele Ausführung mehrerer Verhalten für einen Roboterarm vorgestellt. Diese Verhalten sind unterbrechbar und können konsistent wieder aufgenommen werden. Die Bestimmung der ausführbaren Verhalten erfolgt durch eine Ressourcenbasierte Zustandsbestimmung. Diese Verhalten werden wiederum durch Scheduling-Verfahren untereinander koordiniert, so dass mehrere unterschiedliche Ziele realisiert werden können. Durch die Trennung der Modellierung der Verhalten und des Ausführungssystems (ASM) konnte eine Architektur entwickelt werden, die um neue Verhalten erweitert werden kann und die flexibel für verschiedene Aufgaben eingesetzt werden kann.

4 Programmierung von Verhalten

In Kapitel 2 und Kapitel 3 wurde ein Ansatz zur quasi-parallelen Bearbeitung verschiedener Aufgaben mit einem Roboterarm vorgestellt. In diesem Kapitel soll nun das Ziel Z2 betrachtet werden, indem eine intuitive Programmierung dieser Aufgaben ermöglicht wird. Dafür werden notwendige Merkmale an eine Umsetzung bestimmt, anhand derer der aktuelle Forschungsstand kategorisiert und bewertet wird (Kapitel 4.2). Daraufhin erfolgt eine Abgrenzung der Arbeit in Kapitel 4.3 und eine formale Darstellung der Aufgabe in Kapitel 4.4. Für diese werden zwei mögliche Ansätze mittels intuitiver Programmierung in Kapitel 4.5 und Kapitel 4.6 beschrieben. Die Integration dieser Ansätze in Z1 wird in Kapitel 4.7 beschrieben.

4.1 Aufgabenstellung

Um die Akzeptanz von Robotern in kleinen und mittelständischen Unternehmen sowie im Haushalt zu erhöhen, muss der Nutzen für den Anwender maximiert und der Aufwand minimiert werden. Der Nutzen entsteht beispielsweise dann, wenn der Roboter dem Menschen lästige Aufgaben abnehmen kann, ohne dass eine komplizierte und langwierige Vorbereitung oder Spezialwissen über den Roboter notwendig ist (Siehe Kapitel 1.1).
Es muss also einem Benutzer, der keine Kenntnisse in der Roboterprogrammierung besitzt, möglich sein, eine Aufgabe zu programmieren, die der Roboter dann selbstständig ausführt. Im Folgenden wird daher der Begriff Benutzer und Programmierer gleichwertig benutzt. Idealerweise geschieht die Programmierung, indem der Benutzer die Aufgabe einmal vormacht, also eine Demonstration der Aufgabe bereitstellt. Das System sollte mächtig genug sein, auf Änderungen der Umwelt zwischen der Demonstration und der Ausführung reagieren zu können, wie beispielsweise eine veränderte Position eines Werkstücks, indem es eine entsprechende Adaption durchführt. Außerdem sollte sich die Reproduktion nahtlos in den Ansatz für Z1 integrieren, so dass die Ausführung nicht manuell ausgelöst werden muss und Aufgaben nicht vor jeder Aufgabe wiederholt programmiert werden müssen.
Zur Eindeutigen Verwendung bestimmter Begriffe im weiteren Verlauf, sollen diese zunächst definiert werden.

Definition 13 (Randbedingungen) Als *Randbedingungen* einer Aufgabe, die mit dem Roboter auszuführen ist, werden die Positionen und Orientierungen aller Objekte und die Startkonfiguration des Roboters bezeichnet.

Definition 14 (Demonstrationsphase) Als *Demonstrationsphase* wird der Zeitraum bezeichnet, in dem der Programmierer dem Roboter eine Aufgabe demonstriert und die Sensorik des Systems diese Demonstration und die zugehörigen Randbedingungen erfasst und speichert.

Definition 15 (Reproduktionsphase) Als *Reproduktionsphase* wird der Zeitraum bezeichnet, in dem der Roboter eine in der Demonstrationsphase programmierte Aufgabe unter gegebenenfalls geänderten Randbedingungen ausführt.

Mit Hilfe dieser Definitionen lassen sich die folgenden vier notwendigen Merkmale an eine Lösung zur intuitiven Roboterprogrammierung stellen:

Intuitiv: Eine Programmierung muss durch Laien möglich sein, ohne dass diese Vorwissen über die dahinterstehende Technologie besitzen. Idealweise geschieht die Programmierung des Roboters durch einfaches Vormachen der Aufgabe.

Minimal: Der Aufwand der Programmierung und sonstiger Anweisungen durch den Benutzer muss so gering wie möglich gehalten werden. Dies beinhaltet auch, dass möglichst keine weiteren Hilfsmittel neben dem Roboter verwendet werden.

Adaptiv: Die programmierte Aufgabe soll auch unter geänderten Randbedingungen ausgeführt werden können, um flexibel auf die Umwelt reagieren zu können.

Integriert: Der hier entwickelte Ansatz zur Programmierung soll sich so in den Ansatz für Z1 integrieren, dass die Reproduktion einer Aufgabe verhaltensbasiert erfolgen kann und damit mehrere demonstrierte Aufgaben quasi-parallel abgearbeitet werden können.

4.2 Stand der Forschung und Technik

In diesem Kapitel wird ein Überblick über den Stand der Forschung und Technik im Bezug auf die Roboterprogrammierung gegeben. Die verschiedenen Ansätze werden anhand der in Kapitel 4.1 aufgestellten Kriterien bewertet. Dazu werden zunächst kurz die traditionellen Programmiermethoden, wie die Offline- und Onlineprogrammierung in Kapitel 4.2.1 und Kapitel 4.2.2 betrachtet. Anschließend erfolgt die Betrachtung und Bewertung von Ansätzen mit künstlicher Intelligenz in Kapitel 4.2.3.

4.2.1 Offline Programmierung

Offline-Programmiermethoden sind häufig in Betrieben mit hohen Losgrößen anzutreffen. Hier wird das Programm an einem Computer erstellt und anschließend auf den Roboter übertragen. Die Programmierung kann entweder textuell oder graphisch erfolgen.
Bei der textbasierten Programmierung wird manuell ein Programm in einer formalen Programmiersprache für den Roboter geschrieben, das anschließend ausgeführt wird. Die Implementierung erfolgt meist in einer Hersteller-spezifischen Sprache [KUKA, Stau15a]. Allgemeine Hochsprachen kommen fast ausschließlich im Bereich der Forschung zum Einsatz. [Bigg03]
Bei der graphischen Programmierung von Industrierobotern existiert ein Modell des Roboters und dessen Umwelt in einer Simulationsumgebung. Die Bewegung des Roboters erfolgt in dieser Simulationsumgebung (Abbildung 4.1) und wird später auf den realen Roboter übertragen. Graphische Programmierumgebungen sind mittlerweile nicht mehr nur auf Industrieroboter ausgelegt. Es existieren auch Lösungen für den Consumer-Bereich zur Programmierung von Fischer-Technik [Fish] und Lego Robotern [Grou15], die jedoch sehr eingeschränkt in ihren Möglichkeiten sind.
Beide Varianten der Offline-Programmierung für Roboterarme benötigen speziell geschultes Personal, zusätzliche Ausrüstung und entsprechend viel Zeit um einen Roboter zu programmieren. Sie sind weder intuitiv, noch minimal und können aufgrund der proprietären Formate nur schwer integriert werden. Daher sind diese Verfahren nicht für das angestrebte Verfahren verwendbar.

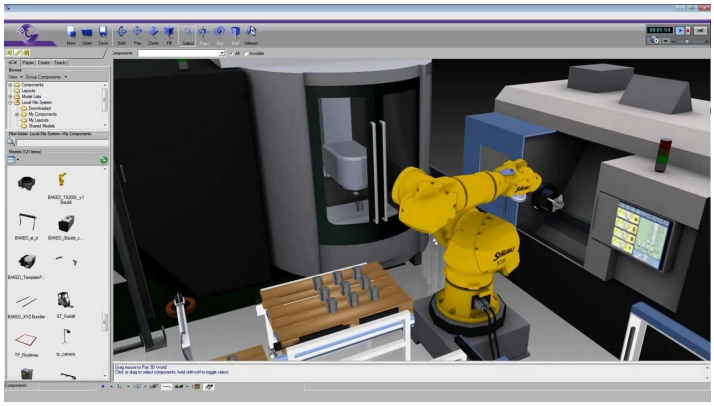


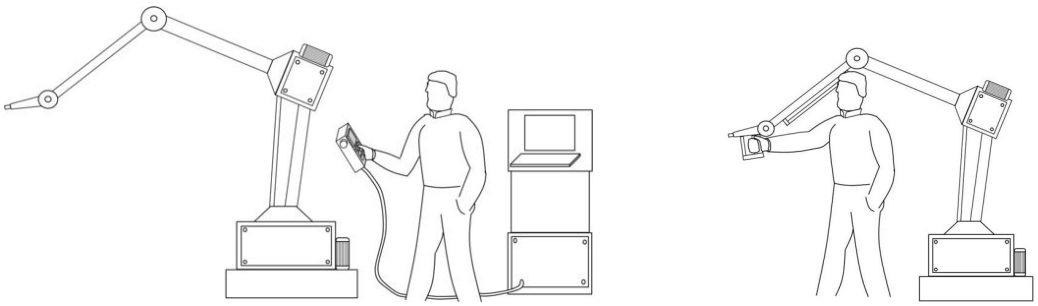
Abbildung 4.1: Grafische Offline-Programmierung eines Stäubli Roboters mit vollständig simulierter Umgebung. Quelle [Stau15b]

4.2.2 Online Programmierung

Bei der Online Programmierung erfolgt die Programmierung direkt am Roboter (Abbildung 4.2). Man unterscheidet hier die sogenannte Teach-In Programmierung und die Playback-Programmierung.
Bei der *Teach-In* Programmierung wird der Roboter mit Hilfe eines Handgerätes sequentiell an verschiedene Punkte im Raum verfahren. Diese Punkte werden vom Bediener abgespeichert und später in der Reproduktionsphase vom Roboter erneut ausgeführt.
Bei der *Playback*-Programmierung erfolgt die Speicherung der Punkte nicht mehr explizit durch den Benutzer. Stattdessen hält der Nutzer den Roboter nahe des Werkzeugs und führt die Aufgabe mit ihm durch während die Bremsen des Roboters gelöst sind. Die Pose des Roboters wird in zeitdiskreten Abständen gespeichert und in der Reproduktionsphase wieder genauso abgefahren. Eine Abwandlung bei zu großen oder schweren Robotern ist die *Master/Slave* Programmierung. Hier wird statt des Roboters selbst eine Ersatzkinematik verwendet, die einfacher zu handhaben ist. Die Bewegung wird anschließend auf den Roboter übertragen. [Bend]
Die Programmierung der Aufgabe ist hier deutlich intuitiver, da der Benutzer den Roboter direkt bewegt. Jedoch ist bei der Teach-In Methode der Aufwand nicht minimal und es findet in beiden Fällen keine Adaption statt. Der Sensoreinsatz beschränkt sich auf die Gelenkwinkel des Roboters und Abweichungen der Objektpositionen werden nicht berücksichtigt. Daher eignen sich beide Methoden nicht für das angestrebte Verfahren.

4.2.3 Programmieren durch Vormachen

Beim Programmieren durch Vormachen wird der Roboter, wie bei der Playback Programmierung, durch die Aufgabe geführt. Der Unterschied liegt darin, dass bei der Ausführung die Demonstration nicht starr wiederholt wird, sondern das System versucht, die Aufgaben an geänderte Randbedingungen anzupassen.
Bei der Art der Programmierung lassen sich generell die beiden Konzepte Programmieren durch Demonstrieren und Programmieren durch Imitieren [Arga09] unterscheiden (Abbildung 4.3). Beim *Programmieren durch Demonstrieren* wird die Aufgabe direkt am Roboter de-



(a) Teach-In Programmierung, Quelle [Bend]

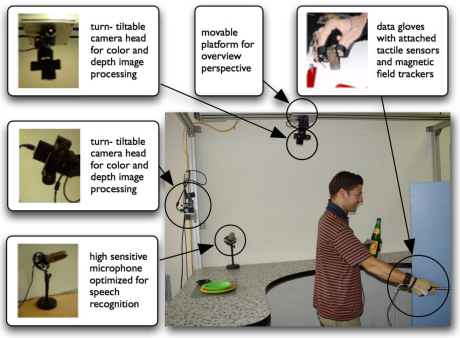
(b) Playback Programmierung, Quelle [Bend]

Abbildung 4.2: Bei der Teach-In Programmierung (a) werden einzelne Positionen mit dem Roboter angefahren und gezielt gespeichert. Bei der Playback-Programmierung (b) wird die komplette vom Programmierer abgefahrene Bahn gespeichert.

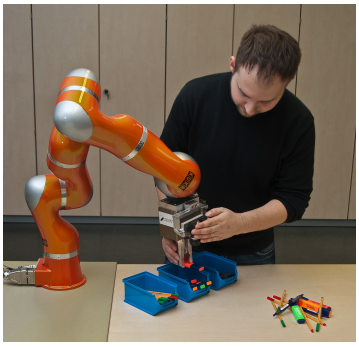
	Single-Shot	Multi-Shot
Symbolisch	Erkennung einer Sequenz von Elementaroperationen	Konstruktion hierarchischer Aufgaben aus Elementaroperationen
Subsymbolisch	Adaption einzelner Trajektorien	Generalisierung durch maschinelle Lernenverfahren

Tabelle 4.1: Möglichkeiten des Roboter Programmieren durch Vormachen basierend auf der Repräsentation der Aufgabe und der Anzahl der Demonstrationen.

monstriert, der diese später auch ausführt. Dies geschieht indem der Roboter entweder direkt bewegt wird oder durch eine Teleoperation gesteuert wird.
Beim *Programmieren durch Imitieren* wird die Aufgabe durch den Menschen selbst vorgeführt. Bei der Ausführung muss zusätzlich eine Abbildung durchgeführt werden, welche die menschliche Kinematik auf die des Roboters überträgt [Scha03]. Des weiteren existiert bei dieser Variante ein erhöhter Aufwand bezüglich der Sensorik zur Überwachung der Bewegung und Interaktion des Programmierers mit der Umwelt [Ekva07, Bill04, Ehre01, De R10]. Aufgrund des erhöhten Sensorbedarfs und der zusätzlich notwendigen Abbildung der menschlichen Handlungen auf den Roboter konzentriert sich diese Arbeit auf ein Programmieren durch Demonstrieren.
Dort lassen sich die Ansätze weiter unterteilen nach der Anzahl der verfügbaren Demonstrationen der Aufgabe und deren Repräsentation (Tabelle 4.1). Der Benutzer kann mehrere Demonstrationen der Aufgabe bereitstellen (*Multi-Shot*) oder nur eine einzige (*Single-Shot*). Des weiteren kann eine Demonstration auf *symbolischer* oder *subsymbolischer* Ebene repräsentiert werden. Während die Programmierung auf subsymbolischer Ebene z.B. als Trajektorie des Roboters oder als Folge von Gelenkwinkeln erfolgt, existieren bei der symbolischen Programmierung verschiedene Bausteine zur Darstellung einer Aufgabe. Diese werden in der Regel im Vorhinein definiert und werden bei der Programmierung wiedererkannt und gegebenenfalls parametrisiert. Die Vorteile einer symbolischen Programmierung liegen darin, dass auch komplexe Aufgaben auf einer abstrakten Ebene erlernt werden können, was auf subsymbolischer



(a) Programmieren durch Imitieren; Quelle [Pard07c]



(b) Programmieren durch Vormachen

Abbildung 4.3: Beim Programmieren durch Imitieren (a) wird die Aufgabe vom Menschen selbst vorgeführt und durch externe Sensorik aufgezeichnet. Beim Programmieren durch Demonstrieren wird die Aufgabe direkt mit dem Roboter durchgeführt.

Ebene nur schwierig möglich ist. Der Nachteil der symbolischen Darstellung liegt darin, dass die Elemente im Vorhinein definiert werden müssen und die Aktionsmenge somit begrenzt ist. Auf subsymbolischer Ebene können dafür prinzipiell beliebige Aktionen programmiert werden. [Bill08]

Nationale Projekte und Forschungsgruppen

Mit dem Paradigma des Roboter Programmieren durch Vormachen beschäftigen sich verschiedene nationale Institute und Forschungsgruppen, die hier kurz genannt werden sollen.
Der Lehrstuhl für Regelungssystemtechnik der TU Dortmund [Rueg15] beschäftigt sich beispielsweise mit der Hybriden Regelung von Bildinformation und Kraftinformation um ein Werkstück-basiertes Playback zu ermöglichen. Somit sind Trajektorien, bei denen das Werkzeug des Roboters auf der Oberfläche des Werkstücks entlangfährt möglich. Bei der Programmierung helfen diverse Möglichkeiten der Nachbearbeitung am Computer. Der Fokus liegt hier klar auf kleinen und mittelständischen Unternehmen und nicht auf dem Haushalt, da Oberflächenbearbeitungen eines einzelnen Objektes im Mittelpunkt stehen.
Das Institut für Werkzeugmaschinen und Fabrikbetrieb der TU Berlin [Temp14] beschäftigt sich mit dem Programmieren durch Imitieren, wobei die Programmierung aufgabenzentriert erfolgt, indem z.B. eine explizite Definition der Aufgabe vorgegeben wird. Der Fokus liegt hier mehr auf dem Aspekt der Imitation als der Adaption selbst. So wird beispielsweise untersucht, welche Hilfsmittel bei der Programmierung eines Roboter hilfreich sind.
Hervorzuheben ist das SME Robotics Projekt [GPS15], das eine intuitive Programmierung für kleine und mittelständische Unternehmen ermöglichen sollte. Auch hier erfolgt die Programmierung zum Teil durch Programmieren durch Vormachen. Jedoch liegt der Schwerpunkt auf der Verwendung zusätzlicher Werkzeuge, wie Spracherkennung, automatischer Programmierung oder intelligenten Eingabegeräten und der Entwicklung von Lösungen für verschiedene typische Industrieanwendungen, die vorzugsweise in der Holzbearbeitung angesiedelt sind. Diese beinhalten spezielle Programmiermethoden, wie beispielsweise dem Definieren von Sägebahnen auf einem Tablet-Computer. Die Übertragung auf neue Handhabungsaufgaben ist

nur beschränkt möglich und eine Übertragung in den Haushalt ist nicht vorgesehen.
Auch am Fraunhofer Institut für Produktionstechnik und Automatisierung (IPA) wird an einem ähnlichen Verfahren für Schweiß- und Klebeanwendungen geforscht [Meye07]. Auch hier führt der Benutzer den Roboter mit der Hand durch die Aufgabe. Zusätzlich können Informationen mit einem tragbarem Computer oder per Spracheingabe zur Verfügung gestellt werden. Anschließend kann die Trajektorie am PC in einer 3D-Umgebung auf einfache Weise angepasst werden.
Die Verwendung weiterer Hilfsmittel bzw. die manuelle Nachbearbeitung von Trajektorien sind essentielle Bestandteile all dieser Ansätze. Da weitere Hilfsmittel in dieser Arbeit jedoch ausgeschlossen werden sollen und auch eine Verwendung im Haushalt angestrebt wird bei der der Roboter automatisch auf Veränderungen reagieren soll, eignen sie sich nicht für das angestrebte Verfahren. Im Folgenden wird daher der Stand der Forschung anhand der Klassifikation von Tabelle 4.1 untersucht.

Single-Shot Ansätze

Single-shot Ansätze besitzen den Vorteil, dass kein langwieriges Training mit großen Datenreihen notwendig ist. Die Methoden erlauben eine Programmierung eines Roboters durch lediglich eine einzige Demonstration. Allerdings besitzen sie eine geringere Verbreitung im Bereich Roboter Programmieren durch Vormachen als durch Imitieren. Hier sollen die existierenden Ansätze, getrennt nach symbolischer und subsymbolischer Darstellung, untersucht werden.

Symbolische Repräsentation der Demonstration Symbolische Ansätze sind im Bereich des Single-Shot-Lernens häufiger anzutreffen. Es existiert eine endliche Menge an sogenannten Elementaroperationen, die aus einer Demonstration erkannt, parametrisiert und zur Ausführung adaptiert werden [Mata98].
Vor allem die Gruppe um Rüdiger Dillmann am Karlsruhe Institut für Technologie setzt auf dieses Verfahren. Die Elementaroperationen werden durch eine Reihe von Klassifikatoren unter hohem sensorischen Aufwand erkannt. Es existieren Ansätze für Ein-Arm-Roboter [Dill10, Kais96, Dill00, Ehre01, Ehre02, Pard07a] und auch für Zwei-Arm-Roboter [Zoll04].
Die Verwendung sogenannter *Dynamic Movement Primitives (DMP)* [Scha03] ermöglicht es, elementare Operationen durch eine Differentialgleichung zweiten Grades darzustellen. Jedes DMP besitzt eine Start- und Zielposition. Die Adaption beschränkt sich auf das Ändern dieser Zielposition. Das Erlernen der DMPs erfolgt zwar auf Trajektorienebene, allerdings sind damit keine komplexen Aufgaben möglich, bei der beispielsweise ein Zwischenziel existiert wie bei Pick-and-Place-Aufgaben. Die DMPs müssen anschließend vielmehr durch einen Planer [Past09, Scha05] oder weitere Lernverfahren wie Reinforcement Learning [Pete08] oder eine zweite Demonstrationsstufe [Zoll04, Zoll05] koordiniert werden und dienen somit als Elementaroperatoren.
Die Notwendigkeit eines solchen zweistufigen Verfahrens bzw. die Vorgabe von Bausteinen ist für das angestrebte Verfahren nicht geeignet, da es die Bedingung nach Minimalität verletzt und wie bereits erwähnt, die Adaptivität einschränkt (siehe Kapitel 4.2.4).

Subsymbolische Repräsentation der Demonstration Es existieren nur wenige subsymbolische Ansätze im Bereich des Single-Shot Lernens. Da hier keine Elementaroperatoren zur Verfügung stehen, ist die demonstrierte Bewegung prinzipiell erst einmal bedeutungslos für

den Roboter. Es existieren zwar Ansätze, die weitere Information auf multimodalem Weg ergründen [Rybs07], jedoch handelt es sich dabei um reine Navigationsaufgaben, die nicht zur Verwendung mit einem Roboterarm vorgesehen sind. Die einzige Möglichkeit besteht also darin, die Trajektorie aufgrund bestimmter Merkmale an neue Situationen anzupassen.
Der Ansatz aus [Maye07] verwendet dazu Methoden aus der Fluiddynamik. Der Raum wird als Flüssigkeit betrachtet, in dem durch eine Demonstration eine bestimmte Strömung erzeugt wird. Die Adaption erfolgt, indem sich die Strömung an die geänderten Randbedingungen anpasst. Die Reproduktion erfolgt, indem die Bahn eines virtuellen Partikels in diesem Strom extrahiert wird. Diese Bahn kann anschließend vom Roboter ausgeführt werden. Allerdings kann das Verfahren nur mit zweidimensionalen Problemen umgehen, benötigt manuelle Eingriffe und besitzt eine extrem hohe Berechnungszeit.
Der Ansatz aus [Wu10] findet mit Hilfe einer Stereokamera geometrische Merkmale, die sowohl während der Demonstration als auch der Reproduktion vorliegen. Zur Adaption wird die gesamte Umwelt samt Trajektorie an die geänderten Merkmalspunkte mittels Thin-plate-Splines adaptiert. Allerdings ist auch dieses Verfahren berechnungsaufwändig ($O(n^3)$), wobei n die Anzahl der Samples darstellt. Zudem ermöglicht es keine Adaption der Orientierung des Roboters.
Ganz ähnlich funktioniert auch der Ansatz aus [Schu13]. Es werden sogenannte non-rigid Registrierungen verwendet um die Trajektorie während der Reproduktion an die Randbedingungen anzupassen. Allerdings ist auch dieses Verfahren berechnungsaufwändig, beruht nach eigenen Aussagen der Autoren auf starken Annahmen und würde in komplizierteren Situationen fehlschlagen [Rybs07].
Prinzipiell sind all diese Verfahren geeignet für das angestrebte System, da keinerlei vorherige Definition von Aktionen stattfinden muss und dennoch eine einzige Demonstration ausreichend ist. Allerdings besitzen die vorhandenen Ansätze alle Schwachstellen, die sie als Ungeeignet erscheinen lassen, wie beispielsweise der hohe Berechnungsaufwand oder die mangelnde Adaption der Orientierung.

Multi-Shot Ansätze

Multi-Shot Ansätze ermöglichen prinzipiell eine Form der Generalisierung. D.h. es werden mehrere Demonstrationen miteinander kombiniert um relevante und irrelevante Merkmale voneinander zu trennen. Anschließend wird die Aufgabe auf einer abstrakten Ebene dargestellt, auf der nur die relevanten Aufgabeninformationen gespeichert sind. In der Reproduktionsphase wird diese Darstellung wieder an die konkret vorliegende Situation adaptiert.
Die Ansätze lassen sich, wie bereits die Single-Shot Ansätze, anhand der Darstellung der Demonstrationen weiter in symbolische und subsymbolische Ansätze unterteilen.

Symbolische Repräsentation der Demonstration Im Bereich der symbolischen Darstellung existieren nur wenige Ansätze, die mehrere Demonstrationen verwenden. Die Stärke der symbolischen Ansätze liegt in der Möglichkeit, Aufgaben durch eine einzige Demonstration zu erlernen, indem das System die Symbole erkennt und die Sequenz anschließend parametrisiert und reproduziert.
Ähnlich dem maschinellen Lernen, erlauben mehrere Demonstrationen in diesem Bereich zum einen eine Generalisierung bestimmter Aufgabenmerkmale [Niek12] [Ekva08]. Zum anderen können auch alternative Ausführungssequenzen generiert werden, indem mehrere Sequenzen zu einer verschmolzen werden [Nico03, Zoll05, Koni11]. Des weiteren ermöglichen mehrere

Demonstrationen auch intuitiv Schleifen für wiederkehrende Aufgabenteile zu beschrieben [Pard07b].
Da die Symbole auch hier im Vorhinein definiert werden müssen, eignen sich diese Ansätze ebenfalls nicht für die Verwendung im angestrebten Verfahren.

Subsymbolisch Repräsentation der Demonstration Betrachtet man die Ansätze auf Trajektorienebene, die mehrere Demonstrationen verwenden, lassen sich verschiedene Algorithmen aus dem Bereich des maschinellen Lernens wiederfinden. Eine Übersicht findet sich in [Arga09]. Eines der am häufigsten eingesetzten Verfahren ist die Darstellung als Gaußmixturen. Durch die Bereitstellung mehrerer Demonstrationen die sich abschnittsweise mehr oder weniger ähneln, ist es möglich relevante von irrelevanten Abschnitten zu unterscheiden. Die Darstellung mehrerer Trajektorien erfolgt durch mehrere, unterschiedlich parametrisierte Gaußmixturen. Die Reproduktion der Trajektorie erfolgt durch statistische Regressionstechniken [Cali08, Cali07, Cali10, Brow04, Bren07, Cakm11]. Der Ansatz wird vor allem durch die Gruppe um Aude Billard am Learning Algorithms and Systems Laboratory (LASA) der École polytechnique fédérale de Lausanne (EPFL) und Darwin Caldwell am Italian Institue of Technology vorangetrieben. Aktuellere Ansätze erweitern das Verfahren um die Erkennung sogenannter „Subtasks“ [Grol10], um die Kombination mit dynamischen Bewegungsprimitiven [Cali12], um ein inkrementelles online Lernen der Gaußmixturen [Cede10], um die Einbeziehung von Redundanzen des Roboters [Wred13], um die Integration von Kräften durch virtuelle Federn [Rozo13] oder um eine Kollisionsvermeidung [Ye11].
Neben Gaußmixturen ist auch die Verwendung von Hidden Markov Modellen stark verbreitet. Hier werden die Trajektorien verwendet, um ein Markov Modell zu trainieren, das anschließend zur Reproduktion der Aufgabe unter geänderten Randbedingungen dient [Grib08, Hovl96, Niek13, Vaka12, Schn10]. Hidden Markov Modelle werden allerdings nicht nur zum Training ganzer Aufgaben genutzt, sondern auch zur Identifikation bzw. Klassifikation einzelner Segmente einer Demonstration verwendet, wie beispielsweise der Erkennung eines erfolgreichen Greifens [Niek13, Aleo06a].
Alle Ansätze in diesem Bereich stellen zwei Anforderungen an den Benutzer. Zum einen muss eine ausreichend hohe Anzahl an Demonstrationen zur Verfügung gestellt werden. Zum anderen müssen irrelevanten Merkmale in jeder Demonstration möglichst unterschiedlich ausgeprägt sein, damit das System die relevanten Bereiche erkennen kann. Konkret heißt das, dass beispielsweise Objekte im Arbeitsraum umpositioniert werden müssen oder sich die Trajektorie stark ändern muss. Ist die Anzahl an Demonstrationen zu gering oder sind sich diese zu ähnlich, ist eine Generalisierung nicht möglich. Diese notwendige Vorgehensweise steht entgegen der Forderung, den Aufwand für den Benutzer minimal zu halten. Auch die Intuitivität ist nur eingeschränkt gegeben, da der Benutzer prinzipiell die Funktionsweise verstehen sollte um gute Generalisierungen zu ermöglichen.

4.2.4 Bewertung

Wie bereits erwähnt, eignet sich die offline Programmierung nicht für die Verwendung durch Laien. Bei der Online-Programmierung bietet die Teach-In Programmierung zwar eine intuitivere Herangehensweise, ist aber tendenziell aufwändiger. Die Playback-Variante reduziert den Aufwand, bietet aber genau wie die Teach-in-Programmierung keine Möglichkeit der Adaption. Programmieren durch Vormachen bietet als einziges Paradigma die Möglichkeit einer Adaption bei gleichzeitiger Intuitivität der Programmierung.

Dort sind am häufigsten symbolische Single-Shot Verfahren und subsymbolische Multi-Shot-Verfahren anzutreffen. Multi-Shot-Verfahren besitzen ihre Stärke in der möglichen Generalisierung, die aufgrund der vielfältigen maschinellen Lernmethoden möglich ist. Es müssen kaum Annahmen getroffen werden und auch eine Merkmalsauswahl [Guyo03] erfolgt implizit, sofern eine ausreichend hohe Anzahl von Demonstrationen zur Verfügung steht, welche eine entsprechend hohe Varianz bei irrelevanten Aspekten aufweisen. Dagegen existieren vor allem im symbolischen Bereich Single-Shot Verfahren. Der Vorteil liegt ganz klar im niedrigen Aufwand für den Programmierer. Der Nachteil hierbei liegt in der vorherigen Definition der Elementaroperatoren und deren sensorischer Erkennung.
Nach [Frie99] schränkt es die Interaktionsmöglichkeit des Benutzers ein, sofern eine Generalisierung von Beobachtungen in eine Menge von intrinsischen Parametern erfolgt, wie es bei den Elementaroperatoren der Fall ist. Nach [Akgu12] ist eine Programmierung auf trajektorienebene flexibler und für den Programmierer meist schneller umzusetzen als eine symbolische Zwischendarstellung. Des Weiteren stellt [Zoll04] fest, dass mehrere Demonstrationen der gleichen Aufgabe für den Benutzer nicht zumutbar sind. Aus diesen Gründen soll die hier angestrebte Programmierung im Bereich des subsymbolischen Single-Shot Programmieren durch Demonstrieren liegen.

4.3 Abgrenzung

Im Rahmen dieses Arbeit soll ein Verfahren entwickelt werden, das die Reproduktion einer Aufgabe unter geänderten Randbedingungen ermöglicht, die einmalig an einem Roboter demonstriert wurde. Es existiert keine Wissensbasis oder sonstiges kontextuelles Wissen über die Aufgabe, wie in [Dill96]. Stattdessen soll untersucht werden, wie lediglich mit den Informationen einer einzigen Demonstration die Aufgabe unter geänderten Randbedingungen durchgeführt werden kann. Nach Dautenhahn [Daut01] lassen sich im Kontext des Programmieren durch Vormachen die 5W des Programmieren untersuchen: *Wann, Wer, Wo, Was, Wie*. Die Frage *Wo* spielt vor allem bei mobilen Systemen eine Rolle. In dieser Arbeit soll ein Verfahren für einen industriellen Manipulator entwickelt werden, so dass der Ort nicht variabel ist und damit die Frage nicht relevant ist.
Die Frage nach dem *Wer* stellt sich üblicherweise im Bereich des Programmieren durch imitieren. Denn hier können auch mehrere potentielle Lehrer vorhanden sein, so dass das System eine Auswahl treffen muss, wessen Anweisungen relevant sind. Durch die Festlegung auf ein Programmieren durch Demonstrieren ist klar, dass die Bewegung des Roboters selbst entscheidend ist. Dies hat zwei Vorteile. Zum einen wird das Abbildungs-Problem ausgeschlossen und zum anderen erhält der Benutzer ein sofortiges Feedback, ob die Aufgabe mit der Kinematik des vorliegenden Roboters durchführbar ist.
Die Frage nach dem *Wann* kann hier auf zwei Arten interpretiert werden. Zum einen, wann das System mit dem Lernen beginnen soll und zum anderen, wann die Ausführung erfolgen soll. Beides ist vor allem für autonome Systeme relevant. In dieser Arbeit ist es akzeptabel den Lernvorgang explizit durch den Programmierer zu initiieren und zu beenden. Der Beginn der Reproduktion soll über das in Kapitel 2 und 3 entwickelte Verfahren automatisch erfolgen.
Der Fokus dieser Arbeit liegt also darauf, *was* zu lernen ist und *wie* eine Adaption in einer geänderten Situation erfolgen soll. Es wird davon ausgegangen, dass die vom Programmierer gegebene Demonstration die Aufgabe erfüllt und von ihm als ausreichend erachtet wird. Der Umgang mit suboptimalen Demonstrationen wird in eigenen Arbeiten betrachtet [Chen03].

Desweiteren werden nur statische Handhabungsaufgaben betrachtet [Span14]. Die Integration dynamischer Aspekte wird kurz im Ausblick diskutiert.

4.4 Formalisierung der Aufgabenstellung

Die Demonstration der Aufgabe erfolgt kontinuierlich und nicht Keyframe-basiert. Keyframe-Verfahren erweisen sich als sehr effektiv [Akgu12], da der Benutzer lediglich die in seinen Augen relevanten Positionen festlegt und somit keine unerwünschte Bewegungen aufgezeichnet werden. Jedoch schränken sie die möglichen Aufgabenklassen ein und erhöhen den Aufwand für den Benutzer, da er explizit und manuell festlegen muss, welche Positionen relevant sind.
Innerhalb dieses Abschnitts wird davon ausgegangen, dass alle Objekte im Arbeitsraum bekannt sind und über eine eindeutige ID erkannt werden können [Ehre00]. Außerdem sind alle Objekte, die während der Demonstrationsphase im Arbeitsraum existieren, relevant für die Aufgabe. Diese Forderung ist legitim, da irrelevante Objekte bei maschinellen Lernverfahren für jede Demonstration entweder umpositioniert werden müssen oder der Programmierer darauf achten muss, dass die Bewegung des Roboters relativ zu diesen Objekten möglichst variiert. Der Aufwand unwichtige Objekte für die Demonstration zu entfernen oder nur relevanten Objekte zu markieren kann bedeutend geringer sein.

Gegeben ist also eine Demonstration $D = \{T, G, R, f_{\text{tool}}\}$ mit einer Trajektorie T, einer Menge an Werkzeugoperationen G und einer Menge Ressourcen R. Die Trajektorie

$$T = (t_1, \ldots, t_n), n \in \mathbb{N}^+ \tag{4.1}$$

beschreibt die Bahn des Tool Center Point (TCP) bestehend aus einzelnen kartesischen Posen $t = (t_{\text{pos}}, t_{\text{rot}})$ mit Position $t_{\text{pos}} \in \mathbb{R}^3$ und Orientierung $t_{\text{rot}} \in [0°, 360°[^3$. Zusätzlich existiert eine Menge an Werkzeugoperationen

$$G = \{g_1, \ldots, g_w\}, w \in \mathbb{N}^+ \tag{4.2}$$

und eine Funktion

$$f_{\text{tool}}(t) = (g, R_t), g \in G, R_t \in \mathcal{R}_{\text{a}}, t \in T \tag{4.3}$$

durch die einem Element t der Trajektorie eine entsprechende Werkzeugoperation g und die dafür notwendigen aktiven Ressourcen R_t zugeordnet werden können. Dies ist z.B. das Öffnen oder Schließen des Greifers an einer bestimmten Stelle t der Trajektore.
Die Menge R setzt sich zusammen aus den aktiven Ressourcen $\mathcal{R}_{\text{G}} \subseteq \mathcal{R}_{\text{a}}$, wie dem Roboter selbst und den für Werkzeugoperationen G notwendigen Ressourcen sowie den semi-aktiven Ressourcen $\mathcal{R}_{\text{O}} \subseteq \mathcal{R}_{\text{sa}}$, die durch Objekte definiert werden.

$$R = \mathcal{R}_{\text{G}} \cup \mathcal{R}_{\text{O}} = \{r_1, \ldots, r_m\}, m \in \mathbb{N}^+. \tag{4.4}$$

Jedes dieser Objekte besitzt eine Pose $k = (k_{\text{pos}}, k_{\text{rot}})$ mit Position $k_{\text{pos}} \in \mathbb{R}^3$ und Orientierung $k_{\text{rot}} \in [0°, 360°[^3$ (*Frame*). Die Menge aller Frames ergibt sich damit zu $K = \bigcup_{i \in \mathbb{N}_0} k_i$.

Gesucht ist nun eine Adaption f_{adapt} von T nach T', welche die demonstrierte Aufgabe unter den geänderten Randbedingungen R' löst.

$$T' = f_{\text{adapt}}(T, R, R') \tag{4.5}$$

Diese Trajektorie T' soll während der Reproduktionsphase ausgeführt werden, um die Aufgabe in einer geänderten Situation durchzuführen. Die Zuordnung der Werkzeugoperationen kann über die Funktion $f_{\text{tool}}(t)$ erfolgen, da die Form der Trajektorie zwar adaptiert wird, sich aber nicht die Anzahl der Samples ändert.
Des weiteren soll die Trajektorie keine Änderung erfahren, sofern sich die Randbedingungen zwischen Demonstration und Reproduktion nicht verändern ($R = R'$):

$$T' = f_{\text{adapt}}(T, R, R) = T \tag{4.6}$$

Zusätzlich soll sich die Lösung noch in den bestehenden Ansatz für Z1 integrieren. Gesucht ist also weiterhin eine Funktion f_{behavior} die ein Verhaltensmodul $B \in \mathcal{B}$ erzeugt, das die gegebene Aufgabe repräsentiert, so dass sie durch den in Z1 entwickelten Ansatz ausgeführt werden kann

$$B = f_{\text{behavior}}(T, G, R), B \in \mathcal{B} \tag{4.7}$$

Zwei mögliche Ansätze zur Umsetzung von f_{adapt} werden in Kapitel 4.5 und Kapitel 4.6 diskutiert. Auf die Umsetzung von f_{behavior} wird in Kapitel 4.7 eingegangen.

4.5 Adaption mittels orientierter Partikel

Der hier vorgestellte Ansatz zur Verwirklichung von f_{adapt} verwendet Methoden einer Partikelsimulation um eine Adaption an geänderte Randbedingungen zu ermöglichen. Grundsätzlich erfolgt die Adaption der Trajektorie auf die folgende Weise. Der Benutzer stellt eine Demonstration $D = \{T, G, R, f_{\text{tool}}\}$ zur Verfügung. Die vom Benutzer demonstrierte Trajektorie T wird als eine Menge von Partikeln betrachtet, die miteinander verbunden sind (Trajektorienpartikel). Durch die Funktion f_{tool} sind den Partikeln bestimmte Werkzeugoperationen G zugeordnet. Weiterhin werden auch alle Objekte $\mathcal{R}_{\text{O}}$ im Arbeitsraum als Partikel betrachtet (Objektpartikel). Zusätzlich existieren Verbindungen zwischen den Trajektorienpartikeln und den Objektpartikeln.
Alle Partikel einer Demonstration befinden sich zunächst im Gleichgewichtszustand. Verändern sich die Positionen der Objekte während der Reproduktion zu R' , ändern sich auch deren Objektpartikel. Aufgrund dieser Änderung erfahren die mit den Objektpartikeln verbundenen Trajektorienpartikel Kräfte, die sie wieder in die ursprüngliche Position relativ zu den Objektpartikel bewegen. Gleichzeitig wirken die Kräfte der Trajektorienpartikel untereinander, die danach streben die Form der Trajektorie zu wahren. Aufgrund dieser Kräfte bewegen sich die Partikel, was wiederum zu veränderten Kräften führt. Diese Wechselwirkungen halten so lange an, bis sich erneut ein Gleichgewichtszustand einstellt (Abbildung 4.4). Dieser erneute Gleichgewichtszustand stellt die Adaption an die geänderten Randbedingungen dar und kann vom Roboter ausgeführt werden.
Die Modellierung der Partikel und deren Wechselwirkung wird in Kapitel 4.5.1 vorgestellt. Darauf folgt die Berechnung der Kräfte und Momente in Kapitel 4.5.2 und die Erstellung der Partikelverbindungen in Kapitel 4.5.3. In Kapitel 4.5.4 wird schließlich die numerische Integration der Adaption beschrieben, die notwendig ist um einen erneuten Gleichgewichtszustand herzustellen.

4.5.1 Modellierung

Eine Demonstration D besteht unter anderem aus der Trajektorie T und den Ressourcen $\mathcal{R}_{\text{O}}$, welche zur Darstellung der Wechselwirkung mit der Umwelt benötigt werden. Sowohl die Tra-

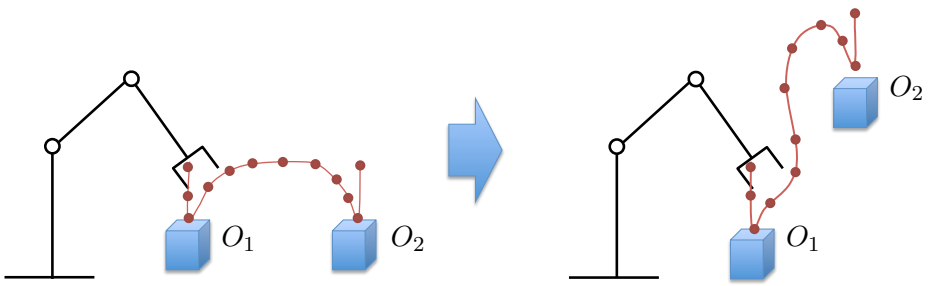


Abbildung 4.4: Symbolische Darstellung der Adaption einer Trajektorie (rot) aufgrund der Verschiebung des Objektes O_2 (blau).

jektorie selbst, als auch jedes Objekt sollen durch Partikel P modelliert werden.
Die Trajektorie T wird durch eine Menge von Partikel modelliert, indem die Bahn aufgezeichnet wird, während der Benutzer den Roboter durch die Aufgabe führt. Diese Aufzeichnung kann entweder in zeit- oder ortsdiskreten Abständen geschehen. Die Entscheidung fällt hier zu Gunsten der zeitdiskreten Variante, da auf diese Weise Bewegungen modelliert werden können, die einen Stillstand des Roboters für bestimmte Zeit erfordern und feine Bewegungen, die durch den Programmierer langsamer gezeigt werden durch eine höhere Anzahl an Partikeln repräsentiert werden (siehe Abbildung 4.5).

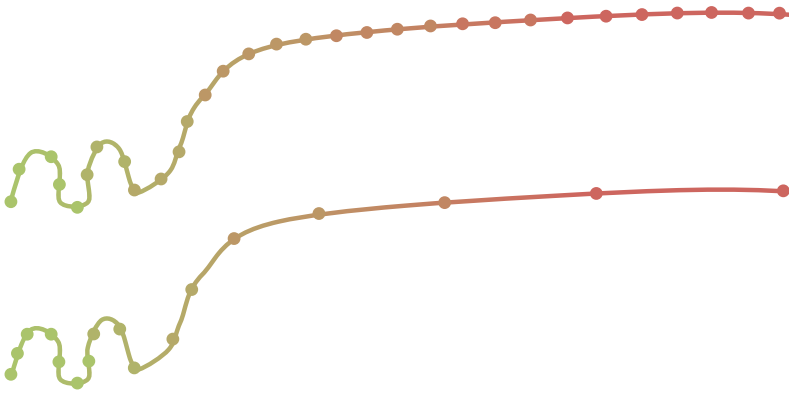


Abbildung 4.5: Vergleich des ortsdiskreten (oben) und des zeitdiskreten Samplings (unten). Hierbei entspricht grün einer niedrigen und rot einer hohen Geschwindigkeit des Roboters bei der Demonstration. Der Vorteil des zeitbasierten Samplings liegt darin, dass feine, vom Programmierer langsam demonstriert Bewegungen durch mehr Samples dargestellt werden als schnelle Bewegungen und somit eine höhere Informationsdichte besitzen.

Um nicht nur die Position sondern auch die Orientierung des Roboters und der Objekte zu kodieren erfolgt die Modellierung als sogenannte orientierte Partikel [Mull11].

Definition 16 (Partikel P) Ein *Partikel* P ist eine Menge $P = \{p, q\}$ mit einer Position $p \in \mathbb{R}^3$ und einer Orientierung $q \in \mathbb{R}^3$.

Definition 17 (Menge aller Partikel $\mathcal{P}$) Die Menge $\mathcal{P} = \bigcup_{i \in \mathbb{N}^+} P_i$ ist die *Menge aller Partikel*.

Jedes gespeicherte Sample t der Trajektorie T wird nun durch ein Partikel P dargestellt, welches die gleiche Position t_{pos} und Orientierung t_{rot} wie das ursprüngliche Sample $t \in T$ besitzt.

$$\forall t \in T, \exists P = \{p, q\} \in \mathcal{P}_{\text{T}} : p = t_{\text{pos}} \wedge q = t_{\text{rot}}. \tag{4.8}$$

Definition 18 (Trajektorienpartikel $\mathcal{P}_{\text{T}}$) Die Menge $\mathcal{P}_{\text{T}} = \{P_1, \ldots, P_n\}, n \in \mathbb{N}^+, P \in \mathcal{P}$ beschreibt jedes Element der gespeicherten Trajektorie durch einen Partikel und heißt Menge der *Trajektorienpartikel*.

Neben der Trajektorie müssen auch alle Objekte $\mathcal{R}_{\text{O}}$ der Demonstration D als Partikel modelliert werden.

Definition 19 (Objektpartikel $\mathcal{P}_{\text{R}}$) Die Menge $\mathcal{P}_{\text{R}} \subset \mathcal{P}$ beschreibt jedes Objekt einer Demonstration durch einen Partikel und heiß *Menge der Objektpartikel* $\mathcal{P}_{\text{R}}$.

Dies beinhaltet auch ein virtuelles Objekt, das die Startposition des Roboters darstellt, um während der Reproduktion auf eine geänderte Startkonfiguration reagieren zu können.
Die Umsetzung erfolgt, indem die Frames K dieser Ressourcen $\mathcal{R}_{\text{O}}$ als Partikel modelliert werden

$$\forall k \in K, \exists P = \{p, q\} \in \mathcal{P}_{\text{R}} : p = k_{\text{pos}} \wedge q = k_{\text{rot}}. \tag{4.9}$$

Hierbei entspricht k_{pos} wieder der Position und t_{rot} der Orientierung des Frames k. Neben diesen relevanten Objekten können sich während der Reproduktionsphase noch weitere Objekte im Arbeitsraum des Roboters befinden. Diese Objekte stellen sogenannte Hindernisse dar.

Definition 20 (Hindernispartikel $\mathcal{P}_{\text{H}}$) Die Menge $\mathcal{P}_{\text{H}} \subset \mathcal{P}$ beschreibt jedes Hindernis während der Reproduktionsphase durch einen Partikel und heißt Menge der Hindernispartikel $\mathcal{P}_{\text{H}}$.

Zur Adaption der Trajektorie einer Demonstration D an eine neue Situation werden die Partikel $\mathcal{P}_{\text{R}}$ entsprechend der sensorisch erfassten Positionen der Objekte aktualisiert. Alle Trajektorienpartikel, die mit diesen Objekpartikeln verbundenen sind, passen sich nun entsprechend der geänderten Randbedingungen an.
Um dies zu erreichen, erfahren die Partikel $P_i \in \mathcal{P}_{\text{T}}$ Kräfte F_i und Momente τ_i aufgrund der veränderten Positionen p und Orientierungen q der verbundenen Partikel. Mit diesen Kräften und Momenten lassen sich für jedes der Partikel P_i die newtonschen Bewegungsgleichungen anwenden.

$$a_i = \tfrac{F_i}{m_i} \quad \alpha_i = I_i^{-1}\tau_i$$

$$\dot{v}_i = a_i \qquad \dot{\omega}_i = \alpha_i$$

$$\dot{p}_i = v_i \qquad \dot{q}_i = \omega_i$$

Hierbei stellt m_i die Partikelmasse, I_i das Trägheitsmoment, a_i und v_i die Beschleunigung und Geschwindigkeit und α_i und ω_i die Winkelbeschleunigung und Winkelgeschwindigkeit dar. Aufgrund der Kräfte erfahren die Partikel eine Beschleunigung. Die Beschleunigung verändert die Geschwindigkeit der Partikel und damit auch deren Position. Analog gilt die auch für die Orientierung q_i aufgrund eines Momentes τ_i.
Da nur die Trajektorie adaptiert werden soll und nicht die Pose der Objekte, werden diese

Kräfte und Momente nur auf die Partikel der Trajektorie $\mathcal{P}_\mathrm{T}$ angewendet. Die Masse jedes Partikels wird dabei zu $m_i = 1$ gewählt.
Die Kraft F_i und das Drehmoment τ_i auf ein Partikel P_i ergeben sich zu

$$F_i = s_\mathrm{s} \cdot \sum_{\substack{P_j \in \mathcal{P}_\mathrm{T} \\ j \neq i}} F_{\mathrm{s},i}(P_j) + s_\mathrm{r} \cdot \sum_{P_r \in \mathcal{P}_\mathrm{R}} F_{\mathrm{r},i}(P_r) + \sum_{P_h \in \mathcal{P}_\mathrm{H}} F_{\mathrm{c},i}(P_h) - F_\mathrm{VD} \tag{4.10}$$

$$\tau_i = s_\mathrm{s} \cdot \sum_{\substack{P_j \in \mathcal{P}_\mathrm{T} \\ j \neq i}} \tau_{\mathrm{s},i}(P_j) + s_\mathrm{r} \cdot \sum_{P_r \in \mathcal{P}_\mathrm{R}} \tau_{\mathrm{r},i}(P_r) - \tau_\mathrm{VD}, \tag{4.11}$$

wobei $F_{\mathrm{r},i}$, $F_{\mathrm{c},i}$ und $\tau_{\mathrm{r},i}$ durch die veränderten Randbedingungen während der Reproduktion entstehen, wie beispielsweise geänderte Position und Orientierung der Objekte $\mathcal{P}_\mathrm{R}$ und Hindernisse $\mathcal{P}_\mathrm{H}$. Dagegen entstehen $F_{\mathrm{s},i}$ und $\tau_{\mathrm{s},i}$ durch die Veränderung der Form der Trajektorie selbst. Zusätzlich existieren noch Dämpfungsterme F_VD und τ_VD zur Stabilisierung der Adaption. Bei s_s und s_r handelt sich um Gewichtungen mit $s_\mathrm{s} + s_\mathrm{r} = 1$ um die Adaption an geänderte Randbedingungen oder die Beibehaltung der Trajektorienform zu priorisieren.

4.5.2 Berechnung der Kräfte und Momente

Die Kräfte F und Momente τ werden aufgrund verschiedener Potentiale ϕ berechnet. Jedes Partikel der Trajektorie unterliegt einem Potential ϕ_s, das dazu dient, die Form der Trajektorie zu erhalten. Dies geschieht indem das Potential ϕ_s eine Kraft $F_{\mathrm{s},i}$ erzeugt, die jedes Partikel in die gleiche Pose relativ zu seinen Nachbarpartikeln drängt, die es in der Demonstration besaß. Dazu werden die Verbindungen des Partikels P_i zu allen Nachbarn P_j analysiert. Aufgrund der relativen Position des Partikels P_i^D zu P_j^D während der Demonstration, wird während der Adaption der Partikel P_i projiziert als P_i^j. Diese Projektion P_i^j entspricht der von Partikel P_j erwarteten Position des Partikels P_i zum Zeitpunkt der Reproduktion, ausgehend von der demonstrierten Position innerhalb der Trajektorie. Die Berechnung von P_i^j erfolgt bei Darstellung in homogenen Koordinaten zu:

$$P_i^j = P_j \cdot \left(P_j^\mathrm{D}\right)^{-1} P_i^\mathrm{D} \tag{4.12}$$

Im Weiteren bezeichnet p die Position und q die Orientierung von P. Es lässt sich nun die resultierende Kraft $F_{\mathrm{s},i}$ auf die Position p_i des Partikels P_i berechnen zu

$$F_{\mathrm{s},i}(p_j) = -\nabla_p \phi_s = |p_i^j - p_i|^2 \cdot (p_i^j - p_i). \tag{4.13}$$

Das Moment τ_i berechnet sich analog aus der Abweichung der relativen Orientierung der Partikel zu

$$\tau_{\mathrm{s},i}(q_j) = -\nabla_q \phi_\mathrm{s} = |q_i^j \cdot q_i^{-1}|^2 \cdot (q_i^j \cdot q_i^{-1}). \tag{4.14}$$

Diese Kraft und dieses Moment bewegen das Partikel P_i zurück auf seine ursprüngliche Position und Orientierung relativ zu P_j. Sowohl die Kraft als auch das Moment nehmen hier quadratisch mit der Abweichung von Soll- und Ist-Position zu (Abbildung 4.7), um starke Abweichungen schneller zu kompensieren. Prinzipiell kann dies beispielsweise auch kubisch oder linear erfolgen, jedoch ist eine quadratische Zunahme üblich in Partikelsimulationen.
Wird diese Kraft und dieses Moment auf alle Partikel der Trajektorie angewandt, bewirkt dies, dass die Trajektorie danach strebt ihre ursprüngliche Gestalt aus der Demonstration aufrecht

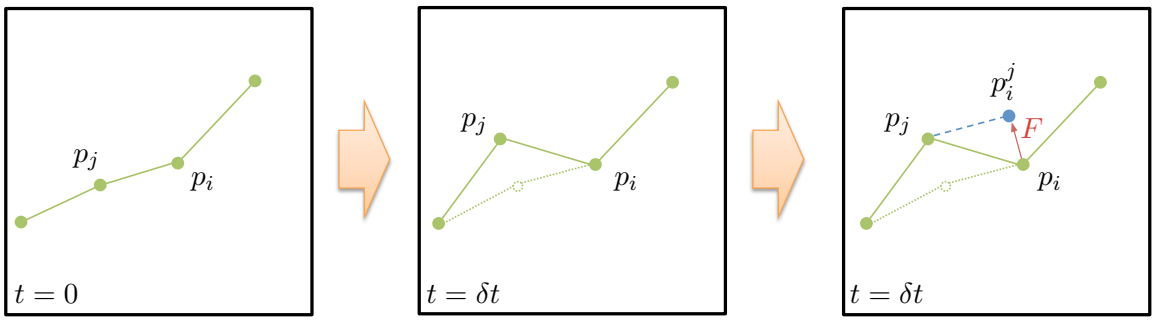


Abbildung 4.6: Entwicklung einer Trajektorie zum Zeitpunkt $t = 0$ und $t = \delta t$. Die Kraft $F_{\mathrm{s},i}(p_j)$, hier als F bezeichnet entsteht aufgrund der Abweichung zwischen der tatsächlichen Position p_i und der Soll-Position p_i^j ausgehend von der demonstrierten Lage zu p_j.

zu erhalten. Dies ist insofern wichtig, als dass das System nicht die Bedeutung einzelner Abschnitte kennt und somit bestrebt sein soll, eine möglichst ähnliche bzw. gleiche Trajektorie wie in der Demonstration zu generieren.
Um die Trajektorie an die aktuellen Randbedingungen, also die geänderte Umwelt während der Reproduktion anzupassen, wird ein Potential ϕ_r eingeführt. Die Funktion ist ähnlich, wie die von ϕ_s. Durch ϕ_r soll die Trajektorie im Bezug auf jedes Objekt $r \in \mathcal{R}_\mathrm{O}$ reproduziert werden. Dafür wird von jedem Objekt r, genauer gesagt, dessen Repräsentation als Partikel $P_r \in \mathcal{P}_\mathrm{R}$ die Kraft $F_{\mathrm{r},i}$ und das Moment $\tau_{\mathrm{r},i}$ auf alle mit P_r verbundenen Partikel P_i der Trajektorie angewendet. Auch hier wird die erwartete relative Position P_i^r von P_i im Bezug auf P_r berechnet. Bei Verwendung von homogenen Koordinaten ergibt sich P_i^r zu

$$P_i^r = P_r \cdot \left(P_r^\mathrm{D}\right)^{-1} \cdot P_i^\mathrm{D}, \tag{4.15}$$

wobei p_r der Position von P_r entspricht. Damit lässt sich die durch P_r auf P_i wirkende Kraft $F_{\mathrm{r},i}(p_r)$ berechnen zu

$$F_{\mathrm{r},i}(p_r) = -\nabla_p \phi_\mathrm{r} = |p_i^r - p_i|^2 \cdot (p_i^r - p_i) \cdot \psi(p_i^\mathrm{D}, p_r^\mathrm{D}). \tag{4.16}$$

Analog lässt sich das auf Partikel P_i wirkende Moment $\tau_{\mathrm{r},i}$ berechnen zu:

$$\tau_{\mathrm{r},i}(q_r) = -\nabla_q \phi_r = |q_i^r \cdot q_i^{-1}|^2 \cdot (q_i^r \cdot q_i^{-1}) \cdot \psi(p_i^\mathrm{D}, p_r^\mathrm{D}) \tag{4.17}$$

Auch hier entsprechen p und q der Position und Orientierung von P. Bei ψ handelt es sich um eine Gewichtung, die sich zu

$$\psi(p_i^\mathrm{D}, p_j^\mathrm{D}) = \frac{d_\mathrm{max} - |p_i^\mathrm{D} - p_j^\mathrm{D}|}{d_\mathrm{max}} \tag{4.18}$$

berechnet, wobei d_max den maximalen Abstand zweier verbundener Partikel $P_i, P_j \in \mathcal{P}_\mathrm{T} \cup \mathcal{P}_\mathrm{R}$ während der Demonstration darstellt. Es findet somit eine Skalierung aller Kräfte und Momente anhand des Abstandes während der Demonstration statt. Partikel mit kleinerer Distanz wirken stärker aufeinander ein, als Partikel mit großer Distanz.

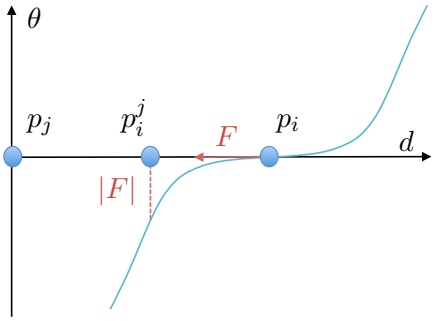


Abbildung 4.7: Symbolische Darstellung der Potentialfelder ϕ_r und ϕ_s als ϕ mit projizierter Partikelposition p_i als p_i^j. Je höher die Abweichung zwischen Soll- und Ist-Position des Partikels ist, desto höher ist die resultierende Kraft auf den Partikel.

Diese Gewichtung ist notwendig, denn es muss sichergestellt werden, dass Bewegungen, die nahe einem Objekt geschehen, genauer reproduziert werden, also solche die weiter entfernt sind. So sollten Kontaktzustände beispielsweise identisch reproduziert werden, während Annäherungsbewegungen eine höhere Abweichung bei der Reproduktion aufweisen dürfen.
Die Integration einer Kollisionsvermeidung kann ohne großen Aufwand in den Ansatz integriert werden, indem ein abstoßendes Potential ϕ_c zu allen Hindernissen hinzugefügt wird. Da hier lediglich eine Kollision vermieden werden soll, muss nur die Position betrachtet werden, nicht jedoch die Orientierung. Die abstoßende Kraft $F_{\mathrm{c},i}$ auf ein Partikel $P_i \in \mathcal{P}_\mathrm{T}$ durch ein Hindernis $P_h \in \mathcal{P}_\mathrm{H}$ berechnet sich zu

$$F_{\mathrm{c},i} = -\nabla_p \phi_\mathrm{c} = \frac{1}{|p_i - p_h|^2} \cdot (p_i - p_h). \tag{4.19}$$

Um ein Aufschaukeln des Systems und zu hohe Geschwindigkeiten zu vermeiden, werden Dämpfungsterme eingeführt. Die Partikel erhalten somit einen Reibungswiderstand, der Abhängig von der aktuellen Geschwindigkeit eine entgegenwirkende Kraft ausübt. Die Dämpfungskraft F_VD und das Dämpfungsmoment τ_VD berechnen sich zu

$$F_\mathrm{VD} = -v \cdot s_\mathrm{VD} \tag{4.20}$$

$$\tau_\mathrm{VD} = -\omega \cdot s_\mathrm{VD} \tag{4.21}$$

mit dem skalaren Dämpfungsfaktor $0 < s_\mathrm{VD} < 1$. Je höher s_VD gewählt wird, desto stabiler erfolgt die Adaption. Allerdings benötigt sie im Gegenzug mehr Zeit, da die erreichten Geschwindigkeiten der Partikel niedriger sind.

4.5.3 Partikelverbindungen

Im vorherigen Kapitel wurde die Modellierung der Trajektorie und der Objekte als Partikel diskutiert und die Kräfte und Momente eingeführt, die zwischen diesen Partikeln auftreten können. Damit die Kräfte zwischen zwei Partikeln P_i und P_j wirken können, muss eine Verbindung zwischen diesen existieren. Verbindungen zwischen Partikeln sorgen also dafür, dass

diese miteinander interagieren können. Es ist nun zu klären, welche Partikel miteinander verbunden werden müssen, um eine Adaption an geänderte Randbedingungen zu ermöglichen.

Definition 21 (Verbindung) Eine *Verbindung* $l \in \mathcal{P} \times \mathcal{P}$ ist ein ungeordnetes Paar $l = \{P_i, P_o\}$ mit $P_i \neq P_o$, das eine Kraft- und Momentenwirkung zwischen P_i und P_o ermöglicht.

Definition 22 (Menge aller Verbindungen $\mathcal{L}$) Die Menge $\mathcal{L} = \bigcup_{i \in \mathbb{N}_0} l_i$ heißt die *Menge aller Verbindungen*.

Ein Standardvorgehen bei physikalischen Partikelsimulationen ist es, jedem Partikel Einfluss auf alle anderen Partikel zu ermöglichen, d.h. es existiert eine vollständige Verknüpfung aller Partikel. Dazu werden in der Regel die Abstände zwischen allen Partikeln berechnet und die resultierenden Kräfte entsprechend skaliert und angewendet. Dies führt bei n Partikeln zu einer Komplexität von $O(n^2)$. In diesem Ansatz können jedoch Einschränkungen bei den Verbindungen vorgenommen werden, um die Komplexität der Berechnung zu reduzieren. Die Partikel der Trajektorie besitzen eine strikte temporale Ordnung, so dass nicht alle Kombinationen betrachtet werden müssen. Auch die Menge aller Verbindungen zu Objekten kann durch entsprechende *Segmentierungsverfahren* reduziert werden. Dazu wird hier davon ausgegangen, dass sich bestimmte Abschnitte einer Trajektorie auf ganz bestimmte Objekte beziehen und nicht jedes Objekt Einfluss auf die gesamte Trajektorie besitzt.

Definition 23 (Trajektorienverbindungen $\mathcal{L}_\mathrm{T}$) Die Menge $\mathcal{L}_\mathrm{T} \subseteq \mathcal{L}$ aller existierenden Verbindungen für die gilt $\mathcal{L}_\mathrm{T} = \{\{P_i, P_j\} \mid P_i \in \mathcal{P}_\mathrm{T} \wedge P_j \in \mathcal{P}_\mathrm{T}\}$ heißt Menge der *Trajektorienverbindungen*, da beide Partikel Trajektorienpartikel sind.

Definition 24 (Objektverbindungen $\mathcal{L}_\mathrm{R}$) Die Menge $\mathcal{L}_\mathrm{R} \subset \mathcal{L}$ aller existierenden Verbindungen für die gilt $\mathcal{L}_\mathrm{R} = \{\{P_i, P_j\} \mid P_i \in \mathcal{P}_\mathrm{T} \wedge P_j \in \mathcal{P}_\mathrm{R}\}$ heißt *Objektverbindungen*, da eines der Partikel ein Objekt repräsentiert.

Prinzipiell gibt es zwei Möglichkeiten die Trajektorienverbindungen $\mathcal{L}_\mathrm{T}$ zu generieren, nämlich ortsbasiert und zeitbasiert. Bei der ortsbasierten Generierung von $\mathcal{L}_\mathrm{T}$ werden nur räumliche Informationen verwendet. Die Menge $\mathcal{L}_\mathrm{T}$ bestimmt sich damit zu

$$\mathcal{L}_\mathrm{T} = \{\{P_i, P_j\} \mid P_i \in \mathcal{P}_\mathrm{T} \wedge P_j \in \mathcal{P}_\mathrm{T} \wedge d(P_i, P_j) \leq \delta_d\}. \tag{4.22}$$

Es werden also alle Partikel deren Distanz d eine gewählte Schranke δ_d nicht überschreitet miteinander verbunden.
Im Gegensatz dazu geht man bei der zeitbasierten Verbindung der Partikel davon aus, dass ein Partikel nur von seinen n Vorgängern abhängt. Die Variable n gibt den Grad der Abhängigkeit an, d.h. bei $n = 1$ findet die Verbindung nur mit dem direkten Vorgänger statt. Daher bestimmt sich die Menge in diesem Fall zu

$$\mathcal{L}_\mathrm{T} = \{\{P_i, P_j\} \mid P_i \in \mathcal{P}_\mathrm{T} \wedge P_j \in \mathcal{P}_\mathrm{T} \wedge (i - j) \leq n\}. \tag{4.23}$$

Einen Vergleich beider Ansätze findet man in Abbildung 4.8.
Da der Benutzer keine graphische Rückmeldung über die im Raum verteilten Partikel während der Programmierung besitzt, kann die ortsbasierte Variante auch zu unerwünschten Verbindungen führen. So können Verbindungen zwischen Trajektoriepartikeln entstehen, die zeitlich

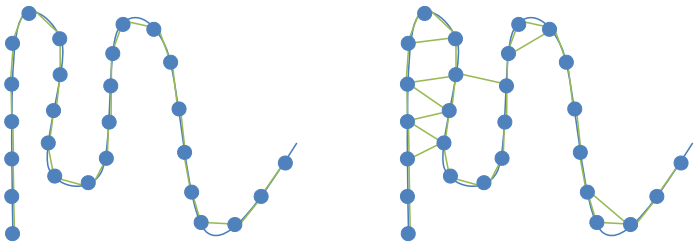


Abbildung 4.8: Bei der zeitbasierten Verbindung der Trajektorienpartikel mit $n = 1$ wird jedes Partikel mit seinem zeitlichen Nachfolger verbunden (links). Bei der ortsbasierten Verbindung der Trajektorienpartikel können auch nicht zeitlich aufeinander folgende Partikel verbunden werden (rechts).

weit auseinander liegen und semantisch nichts verbindet. Im weiteren Verlauf wird daher die zeitbasierte Variante mit $n = 1$ verwendet. Die Verbindungen $\mathcal{L}_\mathrm{T}$ der Trajektorienpartikel $\mathcal{P}_\mathrm{T}$ in Kombination mit den zuvor genannten Kräften und Momenten führt dazu, dass das System versucht eine deformierte Trajektorie wieder in die ursprüngliche Form zu bringen. Je höher n gewählt wird, desto stärker ist der Effekt und desto unflexibler gegenüber Änderungen der Umwelt verhält sich die Trajektorie.
Zusätzlich müssen noch Trajektorienpartikel mit den Objektpartikeln $\mathcal{P}_\mathrm{R}$ im Arbeitsraum verbunden werden, um auf geänderte Randbedingungen zu reagieren. Die Wahl dieser Verbindungen $\mathcal{L}_\mathrm{R}$ entspricht dem „was“ gelernt wird (Kapitel 4.3), also welche Abschnitte einer Demonstration sich auf welche Objekte beziehen. Dieses Problem wird durch die Ansätze im maschinellen Lernen üblicherweise implizit gelöst. Der Ansatz von Grollman [Grol10] beschäftigt sich explizit damit, ist jedoch nicht für Single-Shot-Verfahren anwendbar.
Im folgenden werden verschiedene Möglichkeiten der Segmentierung von $\mathcal{P}_\mathrm{T}$ und der anschließenden Zuordnung in Form von Verbindungen $\mathcal{L}_\mathrm{R}$ vorgestellt. In Kapitel 5 werden diese Verfahren experimentell untersucht und verglichen.

Abstandsbasierte Zuordnung

Die einfachste Möglichkeit eine Segmentierung der Trajektorie zu erreichen, besteht darin, jedes Partikel einzeln zu betrachten und ein festgelegtes Abstandsmaß zu allen Objekten zu berechnen. Ausgehend von diesem berechneten Abstand erfolgt eine Segmentierung der Trajektorienpartikel $\mathcal{P}_\mathrm{T}$ in Teilmengen $\mathcal{P}_i$.
Es gilt hier zwei Möglichkeiten der Umsetzung zu betrachten. Die Segmentierung der Trajektorie kann vollständig oder Schranken-basiert erfolgen. Die vollständige Segmentierung der Trajektorie entspricht einer Voronoi-Zerlegung der Trajektorie $\mathcal{P}_\mathrm{T}$ mit den Objekten $\mathcal{P}_\mathrm{R}$ als Zellenzentren. Ein Segment $\mathcal{P}_i$ wird in diesem Fall definiert durch

$$\mathcal{P}_i = \{P_k \in \mathcal{P}_\mathrm{T} \mid \forall P_r \in \mathcal{P}_\mathrm{R} : d(P_k, P_r) = \min\} \tag{4.24}$$

mit $\forall i, \forall j, \mathcal{P}_i \cap \mathcal{P}_j = \emptyset$. Die Bestimmung der Verbindungen $\mathcal{L}_i \subseteq \mathcal{L}_\mathrm{R}$ ergibt sich damit zu

$$\mathcal{L}_i = \{\{P_k, P_r\} \mid P_k \in \mathcal{P}_i \wedge P_r \in \mathcal{P}_\mathrm{R} \wedge d(P_k, P_r) = \min\} \tag{4.25}$$

Jeder Trajektorienpartikel ist somit mit dem Objekt verbunden, das den geringsten Abstand zu diesem Partikel besitzt.

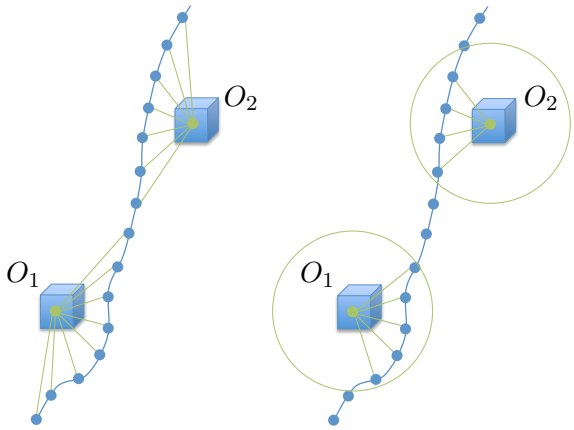


Abbildung 4.9: Bei der vollständigen Segmentierung (links) ist im Gegensatz zu Schranken-basierten Segmentierung (rechts) jedes Trajektorienpartikel mit einem Objekt verbunden.

Geht man dagegen von einer Schranken-basierten Segmentierung mit einer oberen Schranke d_s aus, lassen sich die Verbindungsmengen $\mathcal{L}_i$ definieren durch

$$\mathcal{L}_i = \{\{P_k, P_r\} \mid P_k \in \mathcal{P}_\mathrm{T} \wedge P_r \in \mathcal{P}_\mathrm{R} \wedge d(P_k, P_r) \leq d_\mathrm{s}\}. \tag{4.26}$$

Hierbei entsteht der Effekt, dass ein Partikel sowohl mehreren als auch keinem Objekten zugeordnet sein kann (Abbildung 4.9).
Die Schwierigkeit dieser Methode besteht in der Bestimmung von d_s. Dieser Parameter kann entweder empirisch ermittelt werden oder aus der Trajektorie, der Geometrie der Objekte und deren Konstellation zueinander berechnet werden. Eine Möglichkeit wäre z.B. $d_\mathrm{s} = 2 \cdot r_\mathrm{max}$ zu wählen, wobei r_max dem Durchmesser der größten Bounding-Geometrie der Objekte entspricht.
Beide Verfahren besitzen den Vorteil, dass die Segmentierung einfach und intuitiv ist. Der Nachteil liegt darin, dass die Form der Trajektorie nur indirekt über den Abstand berücksichtigt wird, d.h. Nachbarschaften von Partikeln werden nicht betrachtet.

Clustering

Eine weitere Möglichkeit der Segmentierung besteht darin, nicht nur den Abstand zu den Objekten, sondern auch die Partikel der Trajektorie selbst als Informationsquelle zu verwenden. Dies ist beispielsweise bei der Verwendung eines Clusteringverfahrens der Fall.
Dazu werden ähnliche Partikel der Trajektorie zu Gruppen (Clustern $\mathcal{D}$) zusammengefasst, die wiederum den Objekten $\mathcal{P}_\mathrm{R}$ zugeordnet werden. Dies geschieht indem Verbindungen $\mathcal{L}_i$ zwischen dem entsprechenden Objekt $P_R \in \mathcal{P}_\mathrm{R}$ und allen Partikeln des entsprechenden Clusters $\mathcal{D}_i$ erstellt werden. Zur Durchführung des Clusterings müssen nach [Duda98] zwei Aspekte geklärt werden.

- Bestimmung eines Ähnlichkeitsmaß der Partikel untereinander
- Evaluierung der Güte eines Clusters

Zur Bestimmung der Ähnlichkeit zweier Partikel untereinander wird ein Ähnlichkeitsmaß s_S definiert

$$s_\mathrm{S} : \mathcal{P}_\mathrm{T} \times \mathcal{P}_\mathrm{T} \to \mathbb{R}, \tag{4.27}$$

das auf den Merkmalen $X(P)$

$$X(P) = \{x_1, \ldots, x_k\}, k \in \mathbb{N}^+, P \in \mathcal{P}_\mathrm{T}. \tag{4.28}$$

basiert. Elemente dieses Merkmalsvektors $X(P)$ können sich entweder nur auf die Trajektorienpartikel selbst beziehen und damit Informationen wie die Position, Orientierung, Geschwindigkeit, etc. beinhalten oder auch auf die Relation der Trajektorienpartikel $\mathcal{P}_\mathrm{T}$ zu den Objektepartikeln $\mathcal{P}_\mathrm{R}$ eingehen und beispielsweise den Abstand und die Orientierung zu jedem Objekt während der Demonstration beinhalten. Um möglichst zusammenhängende und kompakte Cluster zu generieren ist es vorteilhaft auch das Wissen über die zeitliche Abhängigkeit der Partikel der Trajektorie zu verwenden, indem auch die Position des Partikels P innerhalb der Trajektorie $\mathcal{P}_\mathrm{T}$ verwendet wird.
Jedes dieser Merkmale x muss normiert werden um einen Wertebereich zwischen Null und Eins zu besitzen. Die kann bei einigen Merkmalen statistisch durch Ermittlung des Mittelwertes m_x und der Varianz μ_x geschehen

$$\bar{x} = \frac{x - m_x}{\mu_x}. \tag{4.29}$$

Auf diese Weise erhält man den normierten Merkmalsvektor $\bar{X}(P)$. Bei einigen der Merkmale kann auch semantische Information genutzt werden, um die Normalisierung durchzuführen, wie beispielsweise die maximalen Werte für die räumliche Position die durch den Arbeitsraum des Roboters gegeben sind oder den Abstand der Greiferbacken.
Als nächstes muss die Güte aller c Cluster $\mathcal{D}_i$ mit Hilfe eines Kriteriums J_e evaluiert werden. Da hier kompakte abgetrennte Cluster gewünscht sind, eignet sich das Mean-Varianz Kriterium

$$J_e = \frac{1}{2} \sum_{i=1}^{c} n_i \bar{s}_i \tag{4.30}$$

mit

$$\bar{s}_i = \frac{1}{n^2} \sum_{X \in \mathcal{D}_i} \sum_{X' \in \mathcal{D}_i} s_\mathrm{S}(\bar{X}, \bar{X}'). \tag{4.31}$$

Hierbei bezeichnet n_i die Anzahl der Partikel im Cluster $\mathcal{D}_i$ und s entspricht der Ähnlichkeitsfunktion der normalisierten Merkmalsvektoren $\bar{X}$ und $\bar{X}'$. Die Durchführung des Clusterings erfolgt, indem das Gütekriterium J_e maximiert wird. Das bedeutet, dass die Partikel innerhalb eines Clusters maximal unähnlich zu den Partikeln anderer Cluster sind. Die Umsetzung selbst erfolgt iterativ, z.B. über einen K-Means Clustering Algorithmus [LLoy82], der hier nicht näher aufgeführt wird.
Da die Anzahl der Cluster unbekannt ist, kann auch ein hierarchischer Ansatz gewählt werden, bei dem jeder Partikel P_i zunächst einen Cluster $\mathcal{D}_i$ darstellt. Die zwei ähnlichsten Cluster werden iterativ so lange verschmolzen bis ein Sprung beim Verlauf von Schritt t nach $t+1$ von J_e erfolgt. In diesem Fall ist der Zustand im Schritt t die natürlichste Ausprägung der Cluster. [Duda98]
Anschließend müssen die Cluster noch den Objekten zugeordnet werden (Abbildung 4.10). Die erste Möglichkeit dies umzusetzen besteht darin, das Objekt dem Cluster zuzuordnen, das

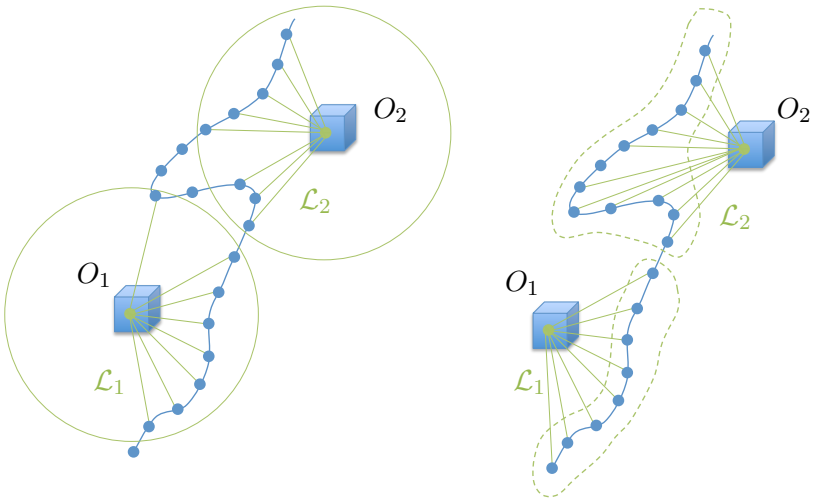


Abbildung 4.10: Generierung der Verbindungen $\mathcal{L}_R$ durch ein abstandsbasiertes Verfahren mit fester Schranke d_s (links) und durch ein Clustering-Verfahren (rechts). Während beim Abstands-basierten Verfahren Objekt O_2 Ausreißer zugeordnet werden, entstehen beim Clustering kompakte Segmente.

den geringsten mittleren Abstand $\bar{d}$ zu allen Partikeln besitzt. Die zweite Möglichkeit besteht darin, das Objekt dem Cluster zuzuordnen das den kleinsten Abstand zu einem beliebigen Partikel im Cluster besitzt.
Die Verbindungen $\mathcal{L}_i$ ergeben sich im ersten Fall zu

$$\mathcal{L}_i = \left\{ \{P_k, P_r\} \mid P_k \in \mathcal{D}_i \wedge P_r \in \mathcal{P}_\mathrm{R} \wedge \left(\bar{d} = \frac{\sum_{P_k \in \mathcal{D}_i} d(P_k, P_r)}{n_i} = \min \right) \right\}, 1 \leq i \leq c. \quad (4.32)$$

Für den zweiten Fall ergeben sich die Verbindungen $\mathcal{L}_i$ zu

$$\mathcal{L}_i = \{ \{P_k, P_r\} \mid P_k \in \mathcal{D}_i \wedge P_r \in \mathcal{P}_\mathrm{R} \wedge \exists P_m \in \mathcal{D}_i : d(P_m, P_r) = \min \}, 1 \leq i \leq c. \quad (4.33)$$

Die Erstellung der Verbindungen $\mathcal{L}_R = \bigcup \mathcal{L}_i$ über ein Clustering bietet prinzipiell eine flexiblere Möglichkeit der Zuordnung als eine feste Abstandschranke. Allerdings erfolgt die Zuordnung auch hier letztendlich Abstands-basiert und mit Hilfe von Heuristiken.

Flutungs-Algorithmus

Eine Möglichkeit dieses Verfahren robuster zu gestalten liegt darin, sichere Zuordnungspunkte zwischen Trajektorienpartikeln und Objektpartikeln zu bestimmen und diese allein oder als Startpunkte einer Segmentierung zu verwenden (siehe Abbildung 4.11).
Eine Verbindung zwischen Trajektorienpartikel $P_i \in \mathcal{P}_\mathrm{T}$ und Objektpartikel $P_r \in \mathcal{P}_\mathrm{R}$ kann als sicher angesehen werden, wenn ein direkter Kontakt zustande kommt. In diesem Fall ist der Bezug zu diesem Objekt eindeutig. Dies kann sowohl direkt zwischen dem Roboter und dem Objekt der Fall sein, wie beispielsweise beim Aufgreifen eines Objektes, als auch indirekt zwischen einem vom Roboter gehaltenen Objekt (Werkzeug) und einem weiteren Objekt (Abbildung 4.11). Die Erkennung des Kontaktzustandes [Abeg00] erfordert hohen Sensoreinsatz

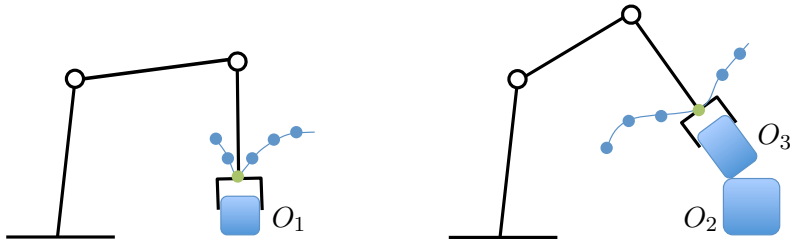


Abbildung 4.11: Die grünen Partikel einer Trajektorie (blau) lassen sich als sichere direkte (links) und sichere indirekte Punkte (rechts) identifizieren. Da ein Kontaktzustand zu dem Objekt besteht, bezieht sich dieser Teil der Trajektorie mit hoher Sicherheit auf das Objekt.

und wird in [Paul95, Ikeu92, Taka99] näher behandelt.
Werden ausschließlich diese sicheren Verbindungen betrachtet, sind damit prinzipiell Handhabungsaufgaben und Pick-and-Place Anwendungen durchführbar. Um die Adaption zu beschleunigen und weitere Teile der Trajektorie den Objekten zuzuweisen, wie beispielsweise das Annähern oder Entfernen von Objekten können diese Verbindungen als Startpunkte für eine Ausweitung verwendet werden. Dies ist zu einem gewissen Grad sogar notwendig, denn die Roboterposition und Orientierung sollte nicht erst zum Kontaktzustand angepasst werden, sondern schon während der Annäherungsphase an ein Objekt, da die Geometrie dessen nicht notwendigerweise bekannt sein muss.

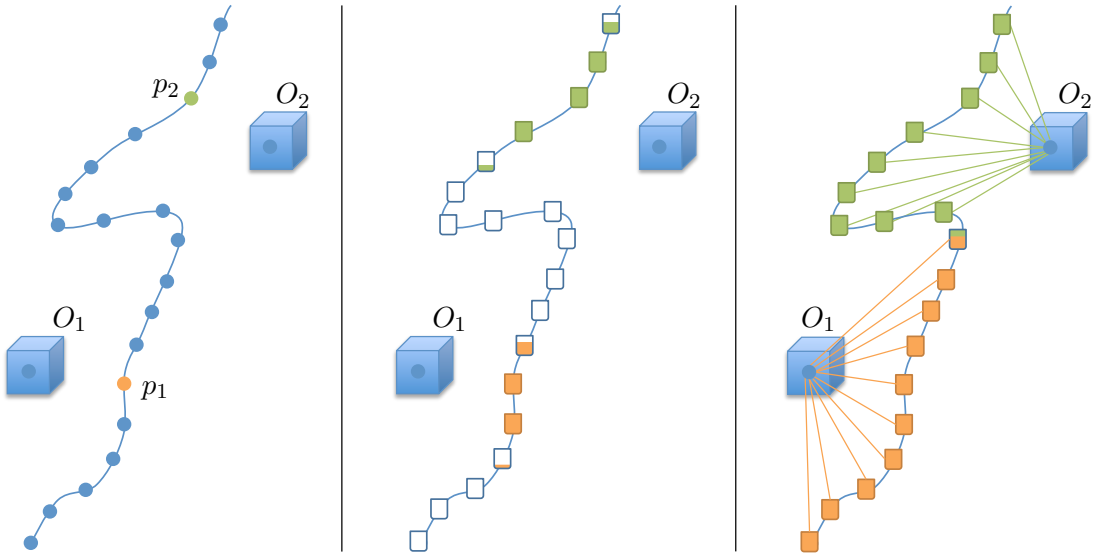


Abbildung 4.12: Vorgehensweise des Flutungsalgorithmus zur Generierung der $\mathcal{L}_i$. Jeder Partikel wird als Behälter betrachtet. Ausgehend von den sicheren Punkten (orange, grün) werden sequentiell weitere Behälter geflutet, bis eine vollständige Segmentierung erfolgt ist. Behälter mit gemischtem Inhalt, werden dem Segment zugeordnet, das den höheren Anteil besitzt.

Eine einfache Möglichkeit wäre nun ausgehend von diesen Partikeln die Vorgänger- und Nachfolger-Partikel den entsprechenden Objekten zuzuordnen. Dies würde zunächst einmal die zeitliche Nachbarschaft der Partikel betrachten. Zudem kann es günstig sein, auch deren Merkmalsvektor $X(P)$ mit einzubeziehen, so dass zusammenhängende Bewegungen, wie etwa eine Kreisbewegung, nicht auseinandergerissen werden.
Um dies zu berücksichtigen, werden die Partikel differenziert betrachtet und ein Flutungs-Algorithmus von den sicheren Punkten aus durchgeführt (Abbildung 4.12) der nach und nach alle Partikel einem Objekt zuweist. Dazu wird jedes Partikel P der Trajektorie als Behälter mit einem bestimmten Volumen V betrachtet. Dieses Volumen kann abhängig sein vom Abstand zum Nachbarpartikel, der Geschwindigkeit, der Änderung der Geschwindigkeit, dem Gradienten der Bewegungsrichtung etc. Wird nun schrittweise immer eine bestimmte Menge Flüssigkeit δV bei den sicheren Punkten eingefügt, laufen diese Behälter von dort aus der Reihe nach voll und werden somit auch dem entsprechenden Objekt zugeordnet.

```
1  Initialisierung
2  ∀P_i ∈ P_T : V(i) ← f(P_i, P_{i−1}, P_{i+1});
3  ∀P_i ∈ P_T : bucket(i) ← 0;
4  L_R ← NIL;
5  Bestimme L_R = (L_1, ..., L_n) durch Kontaktzustände (sichere Punkte);
6  Flutung
7  while ∃ bucket(i) ≠ V(i) do
8      forall the  L_k ∈ L_R do
9          sortiere L_k nach der Position der Partikel in der Trajektorie;
10         P_r ← Partikel des Objektes der Verbindungen L_R ;
11         P_f ← erster Partikel von L_k ;
12         P_u ← letzter Partikel von L_k;
13         f ← δV/2 ;
14         while f > 0 do
15             if V(f) − bucket(f) < f then
16                 bucket(f) ← bucket(f) + f;
17             else
18                 f ← f − (V(f) − bucket(f));
19                 f ← f − 1;
20                 Füge l = (P_f, P_r) in L_k ein;
21             end
22         end
23         f ← δV/2 ;
24         while f > 0 do
25             if V(u) − bucket(u) < f then
26                 bucket(u) ← bucket(u) + f;
27             else
28                 f ← f − (V(u) − bucket(u));
29                 u ← u + 1;
30                 Füge l = (P_u, P_r) in L_k ein;
31             end
32         end
33     end
34 end
35 Ausgabe: Menge aller Verbindungen L_R = ⋃ L_k
```

Algorithm 2: Flutungsalgorithmus zur Bestimmung der Verbindungen $\mathcal{L}_R$

Dieses Verfahren lässt sich mit Hilfe einer dynamischen Programmierung einfach umsetzen (Algorithmus 2). Hierbei ist zu beachten, dass Algorithmus 2 aus Gründen der Übersichtlichkeit lediglich die prinzipielle Vorgehensweise zeigt. Für eine fehlerfreie Umsetzung müssen die Partikel am Kopf und Ende der Trajektorie gesondert betrachtet werden.

Hindernisse

Damit ein Hindernisse eine abstoßende Kraft auf die Trajektorie ausüben kann, müssen auch hier Verbindungen erstellt werden. Da prinzipiell neue Hindernisse während der Ausführungsphase hinzukommen können und auch wieder aus dem Arbeitsraum verschwinden können, müssen diese Verbindungen dynamisch generiert werden.

Definition 25 (Hindernisverbindungen $\mathcal{L}_H$) Die Menge $\mathcal{L}_H \subset \mathcal{L}$ aller existierenden Verbindungen für die gilt

$$\mathcal{L}_H = \{\{P_i, P_j\} \mid P_i \in \mathcal{P}_\mathrm{T} \wedge P_j \in \mathcal{P}_H \wedge d(P_i, P_j) < \delta_\mathrm{H} \wedge \{P_i, P_j\} \notin \mathcal{L}_\mathrm{R}\}$$

heißt Menge der *Hindernisverbindungen*, da eines der Partikel ein Hindernis repräsentiert.

Diese Verbindungen werden erstellt, wenn ein bestimmter Mindestabstand δ_H zwischen den Partikeln unterschritten wurde und keine Verbindung zwischen diesen Partikeln innerhalb $\mathcal{L}_\mathrm{R}$ existiert. Dieser Abstand kann global festgelegt werden oder abhängig von den Objekten bzw. deren Bounding-Geometrie sein. Anschließend können die in Kapitel 4.5.2 definierten Kräfte $F_{\mathrm{c},i}$ angewendet werden.
Durch die in diesem Abschnitt definierten Verbindungen innerhalb der Trajektorienpartikel und zu den Objektpartikeln reduziert sich der Aufwand der Kräfteberechnungen von $O(n \cdot n + r \cdot n + h \cdot n) \subseteq O(n^2)$ auf $O(2n + rn + hn) \subseteq O(n)$, wobei n der Anzahl der Trajektorienpartikel, r der Anzahl der Objektpartikel und h der Anzahl der Hindernispartikel entspricht.

Bewertung

Die vorgestellten Methoden arbeiten entweder lokal indem der Abstand der Trajektorienpartikel zu den Objektpartikeln betrachtet wird, global durch das Optimieren von Clustern oder mit Hilfe von sicheren Ausgangspunkten.
Die vorgestellten Verfahren versuchen das Problem zu lösen, welche Teile einer Demonstration welchem Objekt der Umwelt zugeordnet werden können. Dieses Problem wird von maschinellen Lernverfahren üblicherweise dadurch gelöst, dass mehrere Demonstrationen zur Verfügung stehen. Existieren Trajektorienstücke, die sich relativ zu einem Objekt in vielen Demonstrationen ähneln, kann erwartet werden, dass sich dieser Teil der Trajektorie auf das entsprechende Objekt bezieht.
Steht nur eine Demonstration zur Verfügung, kann diese Entscheidung nicht eindeutig getroffen werden. Die vorgestellten Verfahren stellen Heuristiken dar, die eine bestmögliche Entscheidung auf den zur Verfügung stehenden Informationen treffen. Die Güte der Verbindungserstellung kann erhöht werden, wenn beispielsweise weitere Informationen zur Verfügung stehen. Dies wird im Ausblick näher erläutert. In dieser Arbeit soll jedoch auf weitere Informationen oder Vorwissen verzichtet werden, daher werden nur die zuvor diskutierten Methoden in Kapitel 5 experimentell untersucht und verglichen.

4.5.4 Adaptionsmechanismus

Die Adaption an eine geänderte Umwelt erfolgt, indem die Partikel $\mathcal{P}_\mathrm{R}$ der Objekte $\mathcal{R}_\mathrm{O}$ entsprechend der sensorisch ermittelten Positionen der Objekte aktualisiert werden. Weichen Partikel von der demonstrierten Position ab, befindet sich das System nicht mehr im Gleichgewicht und es treten die in Kapitel 4.5.2 definierten Kräfte und Momente auf. Durch diese erfahren die Partikel eine Beschleunigung bzw. ein Moment, was zu einer Bewegung bzw. Drehung führt. Aufgrund der Verbindungen und der damit mehrfachen Kräfteeinwirkung auf die Partikel muss die Position und Orientierung neu berechnet werden, indem eine numerische Integration durchgeführt wird.

Numerische Integration

Klassischerweise wird für die numerische Integration der Newton-Bewegungsgleichungen ein Euler-Verfahren verwendet [Hock88]. Dabei erfolgt die Berechnung der Position zum Zeitpunkt $t + \Delta t$ durch

$$p(t + \Delta t) = p(t) + \Delta t \cdot v(t). \tag{4.34}$$

Dieses Verfahren leidet jedoch an numerischer Instabilität und Drift, bei der sich die numerische Lösung immer weiter von der tatsächlichen entfernt [Tesc15]. Dieser Effekt tritt vor allem bei großen Schrittweiten Δt auf. Daher wird stattdessen ein Runge - Kutta Verfahren verwendet [Joy07, Goul13], das auch bei größeren Schrittweiten stabil arbeitet und den Drift minimiert.
Mit diesem berechnet sich die Position $p(t + \Delta t)$ zu

$$p(t + \Delta t) = p(t) + \frac{1}{6} \cdot (k_1^\mathrm{p} + 2k_2^\mathrm{p} + 2k_3^\mathrm{p} + k_4^\mathrm{p}) \tag{4.35}$$

mit

$$\begin{aligned}
k_1^\mathrm{p} &= \Delta t \cdot v(t) && \text{mit} && v(t) = \frac{\Delta t}{m} \cdot F\left(p(t), v(t - \Delta t)\right) \\
k_2^\mathrm{p} &= \Delta t \cdot v_1(t + \Delta t) && \text{mit} && v_1(t + \Delta t) = \frac{\Delta t}{m} \cdot F\left(p(t) + \frac{1}{2}k_1^\mathrm{p}, v(t)\right) \\
k_3^\mathrm{p} &= \Delta t \cdot v_2(t + \Delta t) && \text{mit} && v_2(t + \Delta t) = \frac{\Delta t}{m} \cdot F\left(p(t) + \frac{1}{2}k_2^\mathrm{p}, v_1(t + \Delta t)\right) \\
k_4^\mathrm{p} &= \Delta t \cdot v_3(t + \Delta t) && \text{mit} && v_3(t + \Delta t) = \frac{\Delta t}{m} \cdot F\left(p(t) + \frac{1}{2}k_3^\mathrm{p}, v_2(t + \Delta t)\right).
\end{aligned}$$

Hierbei stellt k_i^p die berechnete Positionsänderung aufgrund der berechneten Geschwindigkeit v_{i-1} dar. Die Position des Partikels wird also iterativ für unterschiedliche Zeitpunkte evaluiert und anschließend eine gewichtete Mittlung berechnet. In Abbildung 4.13 ist eine schematische Darstellung der Berechnung nach [Joy07] zu sehen. Die Orientierung $q(t + \Delta t)$ berechnet sich analog zu

$$q(t + \Delta t) = q(t) + \frac{1}{6} \cdot (k_1^\mathrm{q} + 2k_2^\mathrm{q} + 2k_3^\mathrm{q} + k_4^\mathrm{q}) \tag{4.36}$$

mit

$$
\begin{aligned}
k_1^{\mathrm{q}} &= \Delta t \cdot \omega(t) && \text{mit} && \omega(t) = I^{-1} \cdot \tau\left(q(t), \omega(t - \Delta t)\right) \cdot \Delta t \\
k_2^{\mathrm{q}} &= \Delta t \cdot \omega_1(t + \Delta t) && \text{mit} && \omega_1(t + \Delta t) = I^{-1} \cdot \tau\left(q(t) + \frac{1}{2} k_1^{\mathrm{q}}, \omega(t)\right) \cdot \Delta t \\
k_3^{\mathrm{q}} &= \Delta t \cdot \omega_2(t + \Delta t) && \text{mit} && \omega_2(t + \Delta t) = I^{-1} \cdot \tau\left(q(t) + \frac{1}{2} k_2^{\mathrm{q}}, \omega_1(t + \Delta t)\right) \cdot \Delta t \\
k_4^{\mathrm{q}} &= \Delta t \cdot \omega_3(t + \Delta t) && \text{mit} && \omega_3(t + \Delta t) = I^{-1} \cdot \tau\left(q(t) + \frac{1}{2} k_3^{\mathrm{q}}, \omega_2(t + \Delta t)\right) \cdot \Delta t.
\end{aligned}
$$

Die Verwendung des Runge - Kutta Verfahrens lässt größere Schrittweiten zu als das Euler Verfahren und sorgt damit für eine schnellere Adaption der Trajektorie bei gleichzeitiger Erhaltung der Stabilität.

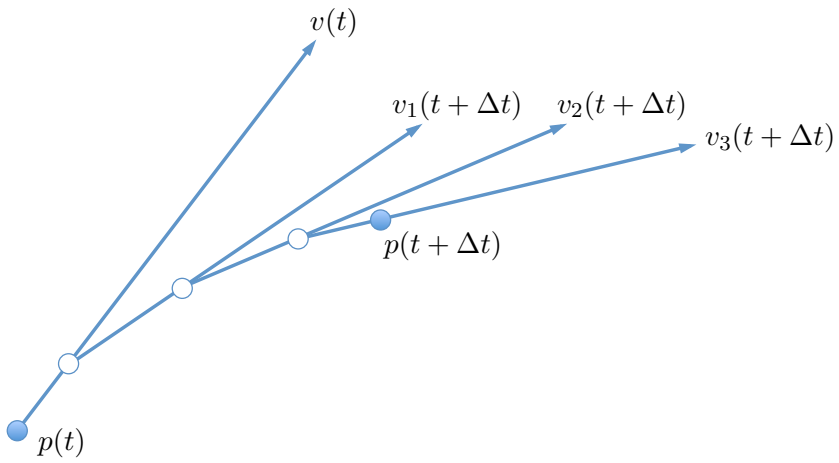


Abbildung 4.13: Veranschaulichung des Runge Kutta Verfahren zur iterativen Berechnung von Partikelbewegungen nach [Joy07].

Abschluss der Adaption

Durch die Änderung der Partikelpositionen und -orientierungen entstehen die in Kapitel 4.5.2 definierten Kräfte und und Momente aufgrund derer das System beginnt die Trajektorie $\mathcal{P}_{\mathrm{T}}$ entsprechend der soeben vorgestellten Integrationsmchanismen zu adaptieren. Die Geschwindigkeit der Partikel ist zunächst hoch, da aufgrund der starken Abweichung zwischen Soll- und Ist-Positionen hohe Kräfte und Momente auftreten. Die Adaptionsgeschwindigkeit nimmt jedoch mit Annäherung an einen Gleichgewichtszustand ab, bis schließlich keine Bewegung der Partikel mehr wahrnehmbar ist.
Die Adaption an eine geänderte Situation erfolgt umso schneller, je mehr Verbindungen zwischen den Objektpartikeln $\mathcal{P}_{\mathrm{R}}$ und den Trajektorienpartikeln $\mathcal{P}_{\mathrm{T}}$ bestehen. Zur Erhöhung der Adaptionsgeschwindigkeit ist auch ein Downsampling der Trajektorienpartikel $\mathcal{P}_{\mathrm{T}}$ denkbar (siehe Ausblick Kapitel 6.2).
Um die Ausführung mit dem Roboter zu ermöglichen, muss ein Zeitpunkt bestimmt werden, an dem die Adaptionsgüte hoch genug ist, um die Adaption zu beenden. Hierfür lässt sich die

kinetische Energie E_{kin} des Systems verwenden, die ein direktes Maß für die aktuelle Bewegung der Partikel ist. Sie setzt sich zusammen aus der Energie der linearen Bewegung $E_{\text{kin,t}}$ und der Energie der Drehung $E_{\text{kin,r}}$.

$$E_{\text{kin}} = E_{\text{kin,t}} + E_{\text{kin,r}} = \sum_i \frac{1}{2} m v^2 + \sum_i \frac{1}{2} \omega^t I \omega \tag{4.37}$$

Sobald eine Sättigung eintritt, findet nahezu keine Bewegung mehr statt. Beide Komponenten werden differenziert betrachten, daher wird jeweils eine Schwelle ϵ_{t} bzw. ϵ_{r} festgelegt. Sobald

$$E_{\text{kin,t}} < \epsilon_{\text{t}} \tag{4.38}$$

und

$$E_{\text{kin,r}} < \epsilon_{\text{r}} \tag{4.39}$$

gilt, kann die nun adaptierte Trajektorie T', die den adaptierten Trajektorienpartikeln $\mathcal{P}_{\text{T}}$ entspricht, vom Roboter ausgeführt werden. Die experimentelle Ermittlung von ϵ_{t} und ϵ_{r} erfolgt in Kapitel 5.3.5.

4.5.5 Zusammenfassung

In diesem Abschnitt wurde ein Ansatz zur Adaption einer Demonstration an eine neue Situation vorgestellt, die durch veränderte Randbedingungen entsteht. Zu diesem Zweck wird eine einmal durch den Benutzer demonstrierte Trajektorie des Roboters als Kette von Partikeln modelliert. Diese sind sowohl untereinander als auch stellenweise mit den Objekten der Umwelt verbunden. Aufgrund von Veränderungen der Randbedingungen entstehen Kräfte und Momente unter den verbundenen Partikeln, die zu Wechselwirkungen führen bis sich wieder ein Gleichgewichtszustand eingestellt hat. Die numerische Berechnung dieses Gleichgewichtszustandes erfolgt mit Hilfe eines Runge - Kutta Verfahrens. Der ermittelte Zustand entspricht der an die neue Situation adaptierten Trajektorie, die vom Roboter ausgeführt werden kann.

4.6 Adaption mittels Motion Segments

Die Zuvor vorgestellte Methode erlaubt die Adaption einer Demonstration D an geänderte Randbedingungen, die aufgrund einer geänderten Situation entstehen. Die Vorteile dieser Adaption liegen in der flexiblen Integration weiterer Potentiale ϕ und damit der universellen Lösung des Problems. Als nachteilig kann sich die Adaption über die numerische Integration herausstellen, wenn z.B. harte Echtzeitanforderungen gestellt werden. Daher soll hier ein weiterer analytischer Ansatz vorgestellt werden, der unter bestimmten Annahmen möglich ist.

4.6.1 Annahmen

Der in Kapitel 4.5 gezeigte Ansatz erlaubt es, dass sich Partikel einer Demonstration auf beliebig viele Objekte der Umwelt beziehen, d.h. mit diesen verbunden sind. Es existieren jedoch Aufgaben, bei denen dies nicht notwendig ist. Stattdessen lässt sich klar feststellen, dass sich bestimmte Teile der Trajektorie jeweils nur auf ein Objekt beziehen, wie es z.B. bei Pick-and-Place Aufgaben der Fall ist. Die Verbindungen $\mathcal{L}_\mathrm{R}$ der Trajektorienpartikel entspricht dann einer Segmentierung der Trajektorie wie in Kapitel 4.5.3 ohne Überschneidung.

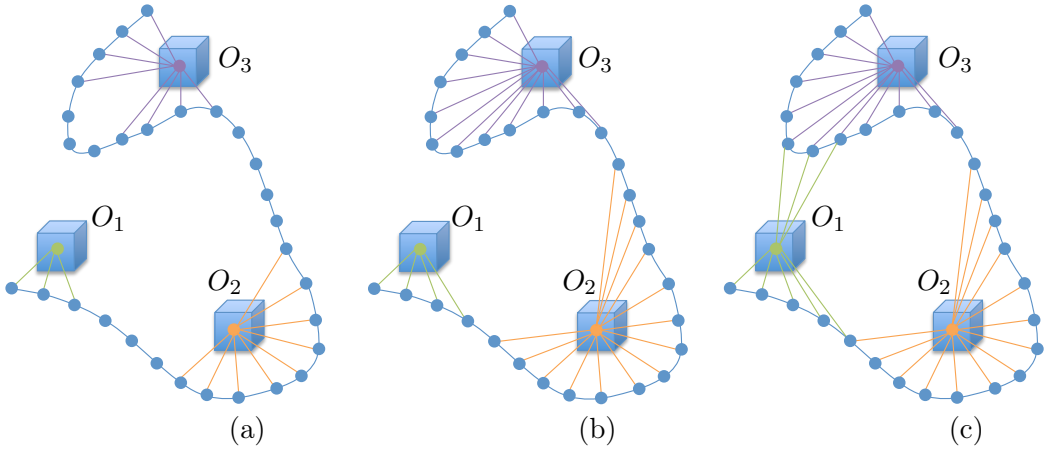


Abbildung 4.14: Übersicht über verschiedene Verbindungsmöglichkeiten; (a) unvollständige Zuordnung zwischen Partikeln und Objekten aufgrund einer niedrigen Schranke d_s; (b) Vollständige Zuordnung durch Flutung; (c) Überlappende Zuordnung; Eine Adaption mittels Motion Segments ist in den Fällen (a) und (b) möglich, nicht jedoch (c), wenn ein Partikel $P \in \mathcal{P}_\mathrm{T}$ Verbindungen mit mehreren Objekten besitzt.

Betrachtet man die Voronoi-Segmentierung aus Kapitel 4.5.3, lässt sich jedes Partikel P der Trajektorie $\mathcal{P}_\mathrm{T}$ genau einem Objekt $P_R \in \mathcal{P}_\mathrm{R}$ zuweisen. D.h. bei gegebenen Verbindungen $\mathcal{L}$ gilt:

$$\forall P \in \mathcal{P}_\mathrm{T} \nexists l_1 = (P, P_a), l_2 = (P, P_b) \in \mathcal{L}_R : P_a \neq P_b \text{ mit } P_a, P_b \in \mathcal{P}_\mathrm{R} \tag{4.40}$$

Ist die Segmentierung nicht vollständig, d.h. existieren Partikel ohne Verbindung zu Objekten, lässt sich annehmen, dass es sich hierbei um Transferbewegungen zwischen den Objekten handelt (Abbildung 4.14). Dies ist insbesondere der Fall, wenn Objekte eine hohe Distanz untereinander besitzen. Unter der Annahme, dass sich jedes Partikel $P \in \mathcal{P}_\mathrm{T}$ entweder exklusiv einem Objekt zuordnen lässt oder Teil einer Transferbewegung ist, kann eine analytische

Lösung zur Adaption einer Trajektorie an geänderte Randbedingungen umgesetzt werden, die im Folgenden näher beschrieben wird.

4.6.2 Ansatz

Die analytische Adaption an veränderte Randbedingungen erfolgt in mehreren Schritten. Für jedes Partikel $P \in \mathcal{P}_\mathrm{T}$ wird ein zugehöriges Koordinatensystem (Frame) k ermittelt, das beispielsweise durch ein Objekt definiert ist. Dann wird das Partikel in dieses lokale Koordinatensystem k transformiert. Bei der Reproduktion wird dieses Koordinatensystem k mit seinen Änderungen, z.B. durch sensorisch erfasste Objekte erneut bestimmt. Durch die Zugehörigkeit und lokale Darstellung des Partikels P in k, wird dieses Partikel direkt an die neue Situation adaptiert. Anschließend erfolgt eine Transformation zurück in Weltkoordinaten, damit die adaptierte Trajektorie vom Roboter ausgeführt werden kann.
Diese notwendigen Schritte werden in den folgenden Kapiteln beschrieben. Zunächst werden die für ein Partikel relevanten Frames in Kapitel 4.6.3 bestimmt, anschließend wird auf die Koordinatensysteme für Transfer-Bewegungen in Kapitel 4.6.4 eingegangen. Mit diesen wird die Transformation eines Samples zwischen Demonstration und Reproduktion in Kapitel 4.6.5 beschrieben. Anschließend werden noch Optimierungen in Kapitel 4.6.6 behandelt und alternative Adaptionsmöglichkeiten in Kapitel 4.6.7 diskutiert.

4.6.3 Frame-Zuordnung

Gegeben ist zunächst die Trajektorie $\mathcal{P}_\mathrm{T} = (P_1, \ldots, P_n)$ mit $n \in \mathbb{N}^+$ und die Frames $K_\mathrm{O} = \{k_1, \ldots, k_m\}$ mit $m \in \mathbb{N}^+$ aller Objekte. Weiterhin ist eine überschneidungsfreie Segmentierung in Form einer Segmentierungmatrix U gegeben. Die Anzahl der Spalten entspricht der Anzahl der Partikel der Trajektorie, die Anzahl der Zeilen entspricht der Anzahl der Frames und damit der Objektpartikel $\mathcal{P}_\mathrm{R}$. Ein Eintrag $u_{i,j}$ wird berechnet zu

$$u_{i,j} = \begin{cases} 1 & \text{wenn } \exists l = (P_j, P_i) \in \mathcal{L} : P_i \in \mathcal{P}_\mathrm{R} \wedge P_j \in \mathcal{P}_\mathrm{T} \\ 0 & \textit{sonst} \end{cases} \tag{4.41}$$

und entspricht damit der Zuordnung eines Partikels P_j zu einem Objekt-Frame k_i (siehe Kapitel 4.5.3). Handelt es sich dagegen um eine vollständige Segmentierung, existiert für jedes $P \in \mathcal{P}_\mathrm{T}$ ein entsprechendes $k \in K_\mathrm{O}$ und damit

$$\forall j \leq n \wedge j > 0 : \sum_{i=1}^{m} u_{ij} = 1 \tag{4.42}$$

D.h. es existiert kein Partikel, ohne Zuordnung zu einem Objekt-Frame. Handelt es sich um eine unvollständige Segmentierung gilt:

$$\exists j \in \{1, \ldots, n\} : \sum_{i=1}^{m} u_{ij} = 0 \tag{4.43}$$

In diesem Fall existiert kein k für P. Es handelt sich bei diesen Partikeln um Transferbewegungen, die keinem Frame $k \in K_\mathrm{O}$ zugewiesen sind. Mit diesen kann auf verschiedene Weise verfahren werden. So wäre es möglich, jedes Partikel exklusiv einem der beiden Objekt-Frames zwischen denen der Transfer stattfindet zuzuordnen, anteilig beiden zuzuordnen oder einem

dedizierten Frame zuzuordnen. Die Option der Zuordnung zu einem der Objekt-Frames entfällt, denn wäre diese Zuordnung legitim, wäre es bereits bei der Erstellung der Verbindungen geschehen. Die Zuordnung zu beiden führt zu Problemen, die in Kapitel 4.6.7 aufgezeigt werden. Es stellt also lediglich die dritte Variante eine vertretbare Lösung dar. Das bedeutet, die bisherige Menge Frames K_O muss erweitert werden um die Menge der Transfer-Frames K_T.

Definition 26 (Frames K) Sei K die Menge aller *Frames*, die aus den *Objekt-Frames* K_O und den *Transfer-frames* K_T besteht ($K = K_\mathrm{O} \cup K_\mathrm{T}$).

Da prinzipiell zwischen allen Frames K_O Transferbewegungen auftreten können, muss für jede mögliche Kombination $\{k_i, k_j\} \in K_\mathrm{O}^2$ mit $k_i \neq k_j$ ein Transfer-Frame definiert werden. Mit $m = |K_\mathrm{O}|$ lässt sich die Anzahl an Transfer-Frames zu

$$|K_\mathrm{T}| = \frac{m(m-1)}{2} \tag{4.44}$$

bestimmen. Hier wird angenommen, dass es keine Rolle spielt ob der Transfer von k_i nach k_j oder umgekehrt erfolgt.

Definition 27 (Transfer Frame) Sei $k_{i \rightleftarrows j} \in K_\mathrm{T}$ der *Transfer-Frame* zwischen Frame $k_i \in K_\mathrm{O}$ und Frame $k_j \in K_\mathrm{O}$.

Die genaue Konstruktion dieser Frames K_T wird im nächsten Kapitel beschrieben. Die Zuordnung der Partikel zu den Frames K lässt sich nun durch eine $|K| \times |n|$ große Gewichtungsmatrix W durchführen, bei der ein Eintrag $w_{i,j}$ die Gewichtung eines Frames für ein Partikel repräsentiert. Dieser Eintrag wird später für die Adaption an eine neue Situation verwendet.

$$W = \overbrace{\begin{bmatrix} w_{1,1} & \dots & w_{1,n} \\ \vdots & \ddots & \\ w_{m,1} & & w_{m,n} \\ w_{m+1,1} & \dots & w_{m+1,n} \\ \vdots & \ddots & \\ w_{|K|,1} & & w_{|K|,n} \end{bmatrix}}^{P_\mathrm{T}} \begin{matrix} \left.\vphantom{\begin{matrix} a \\ a \\ a \end{matrix}}\right\} K_O \\ \left.\vphantom{\begin{matrix} a \\ a \\ a \end{matrix}}\right\} K_T \end{matrix}$$

Zunächst werden alle $w_{i,j} = 1$ gesetzt, wenn $u_{i,j} = 1$ gilt, da dies den Zuordnungen der Objekt-Frames entspricht. Anschließend müssen noch die Zuordnungen zu den Transfer-Frames K_T berücksichtigt werden. Dazu werden alle Spalten j betrachtet, für welche

$$\sum_{i=1}^{m} u_{i,j} = 0 \tag{4.45}$$

gilt. Dies sind Partikel, die noch keinem Frame k zugeordnet wurden. Für jede dieser Spalten j wird nun die Spalte p und s bestimmt, so dass gilt

$$\exists p \in \mathbb{N} : \sum_{i=1}^{m} u_{i,p} = 1 \wedge |j - p| = \min \wedge p < j \tag{4.46}$$

und

$$\exists s \in \mathbb{N} : \sum_{i=1}^{m} u_{i,s} = 1 \wedge |s - j| = min \wedge s > j. \tag{4.47}$$

Diese spalten p und s entsprechen den Partikeln P_p und P_s, welche als letztes bzw. als nächstes einem Objekt-Frame $k \in K_\mathrm{O}$ zugeordnet werden. In diesen beiden Spalten s und p von U existiert jeweils genau ein Eintrag mit $u_{a,p} = 1$ bzw. $u_{b,s} = 1$. Hierbei entsprechen k_a und k_b den Frames, denen P_p und P_s zugeordnet sind. Sei nun $k_t = k_{a \leftrightarrows b}$ der Transfer-Frame zwischen den Objekt-Frames k_a und k_b und e die Zeile in W, die k_t repräsentiert, dann wird $w_{e,j} = 1$ gesetzt. Damit wird der Partikel P_j dem Transfer-Frame $k_t \in K_\mathrm{T}$ zugeordnet, der durch die Objekt-Frames k_a und k_b definiert ist.
Zusätzlich muss der Fall betrachtet werden, dass ein Partikel P_z der Trajektorie einem Frame k zugeordnet wird, darauf einige Partikel $P_{z+1}, \ldots, P_{z+x}$ mit $x \in \mathbb{N}^+$ ohne Zuordnung zu einem Frame folgen und anschließend der Partikel P_{z+x+1} wieder dem gleichen Frame k zugeordnet wird. Diese Partikel $P_{z+1}, \ldots, P_{z+x}$ ohne Zuordnung würden nun der Definition nach einem Transfer-Frame zugeordnet werden. Da bei diesem Transfer-Frame jedoch $k_a = k_b$ gilt, entspricht dies einer Zuordnung zum Frame $k_{a \rightleftarrows a} = k_a$, also dem Objekt-Frame k_a selbst. Entsprechend werden auch hier die Einträge $w_{a,z+1}$ bis $w_{a,z+x}$ gleich 1 gesetzt.
Ein weiterer Spezialfall tritt auf, wenn Partikel am Ende der Trajektorie existieren, die keinem Objekt-Frame zugeordnet worden sind. Prinzipiell wäre es möglich, diese Partikel dem Weltkoordinatensystem zuzuordnen, dem TCP-Frame des Roboters oder dem Objekt, das als letztes referenziert wurde. Die Zuordnung zum Weltkoordinatensystem ist wenig sinnvoll, da dieses nicht an geänderte Randbedingungen adaptiert wird. Auch die Zuordnung zum TCP-Frame macht wenig Sinn, da sich die Adaption nur aufgrund der Startposition des Roboters ändern würde. Die einzig logische Konsequenz ist es, die Partikel dem zuletzt referenzierten Objekt-Frame zuzuordnen. Es ist plausibel, dass es sich hierbei beispielsweise um eine „detach"-Bewegungen handelt [Frie96]. Dazu wird in allen $j = 1, \ldots, n$ Spalten, diejenige Spalte l in U

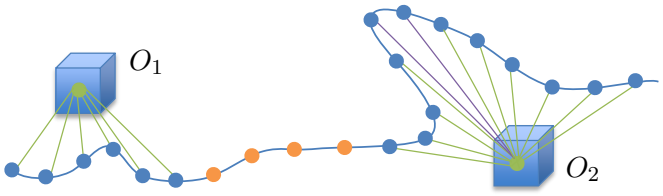


Abbildung 4.15: Zuordnung der Partikel zu einem Objekt-Frame (grün), zu einem Transfer-Frame (orange) und einem Objekt-Frame $k_{a \leftrightarrows a} = k_a$ (blau).

gesucht, für die gilt

$$\sum_{i=1}^{m} u_{i,l} = 1, l = \max_{j} \tag{4.48}$$

Dies ist die letzte Spalte in der die Summe den Wert 1 besitzt. Der Eintrag $w_{k,l} = 1$ bezeichnet die Zuordnung des Partikels P_l zum Objekt-Frame k_k. Diese Zuordnung wird nun auch für die zeitlich folgenden Partikel durchgeführt, d.h.

$$\forall j > l \wedge j \leq n : w_{k,j} = 1 \tag{4.49}$$

Somit gilt für jede Spalte von W

$$\forall j \in \{1, \ldots, n\} : \sum_{i=1}^{|K|} w_{i,j} = 1 \tag{4.50}$$

Es wird also die Bedingung erfüllt, dass jedes Partikel P einem Frame k zugeordnet ist, was durch die Matrix W repräsentiert wird. Für die Adaption muss nun noch geklärt werden, wie die Transfer-Frames konkret definiert sind.

4.6.4 Transfer-Frames

Jeder Partikel $P \in \mathcal{P}_\mathrm{T}$ wird durch die Matrix W exakt einem Frame $k \in K$ zugeordnet. Dabei kann k aus der Menge der Objektframes K_O stammen oder aus der Menge der Transferframes K_T. In dieser Arbeit wird angenommen, dass der Objekt-Frame eines Objekts eindeutig ist. Die Transfer-Frames müssen nun noch anhand der Objekt-Frames definiert werden.
Gegeben seien zwei Objektframes k_i und k_j. Der Transfer-Frame $k_{i \leftrightarrows j}$ ist so zu definieren, dass er nur von diesen beiden Frames abhängt und Trajektorienstücke zwischen den Objekten entsprechend angepasst werden.

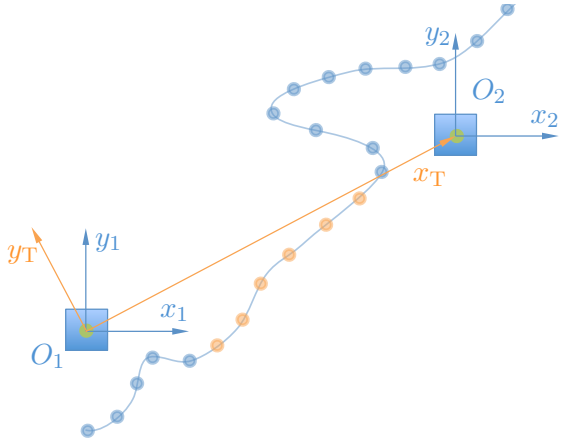


Abbildung 4.16: Definition eines Transfer-Frames im zweidimensionalen. Die x_T -Achse wird aufgrund der Objektpositionen von O_1 und O_2 ermittelt. Die y_T Achse wird so gewählt, dass ein rechtwinkliges, rechtshändiges Koordinatensystem entsteht.

Zur vollständigen Definition des rechtshändigen, rechtwinkligen Koordinatensystems wird ein Ursprung u_T und die Achsen $x_\mathrm{T}, y_\mathrm{T}, z_\mathrm{T}$ benötigt. Der Ursprung u_T kann prinzipiell beliebig als Ursprung des Frames k_i oder k_j gewählt werden. Es wird hier $u_\mathrm{T} = (0,0,0)_{k_i}$ gewählt, also der Ursprung des Frames, der dem Objekt mit der niedrigeren ID zuzuordnen ist. Ausgehend von diesem wird folgendermaßen ein Koordinatensystem $x_\mathrm{T}, y_\mathrm{T}, z_\mathrm{T}$ definiert mit

$$\begin{aligned} x_\mathrm{T} &= (0,0,0)_{k_j} - (0,0,0)_{k_i} \\ x_\mathrm{T} \perp y_\mathrm{T}, x_\mathrm{T} &\perp z_\mathrm{T}, y_\mathrm{T} \perp z_\mathrm{T} \\ \angle(z_\mathrm{T}, -g) &= \min, \end{aligned} \tag{4.51}$$

wobei g den Gravitationsvektor darstellt. Die Achse x_T zeigt in Richtung des zweiten Objektes und entspricht der Transferrichtung. Die Komponenten y_T und z_T stehen dazu orthogonal, wobei z_T entgegen der Gravitation zeigt um den zusätzlichen Freiheitsgrad zu eliminieren. Diese Konvention ist hilfreich, da evtl. Behälter vom Roboter transportiert werden und diese eine aufrechte Orientierung während der Reproduktion beibehalten sollen.
Diese Transfer-Frames werden sowohl während der Demonstration als auch zum Zeitpunkt der Reproduktion konstruiert um eine Adaption zu ermöglichen, wie sie im Folgekapitel beschrieben wird.

4.6.5 Adaptionsmechanismus

Gegeben ist, wie in Kapitel 4.4 beschrieben, eine Trajektorie $\mathcal{P}_\mathrm{T}$, Frames $K = K_\mathrm{O} \cup K_\mathrm{T}$ aus der Demonstration und $K' = K'_\mathrm{O} \cup K'_\mathrm{T}$ aus der Reproduktion. Jedes Partikel $P \in \mathcal{P}_\mathrm{T}$ ist durch W einem Referenzkoordinatensystem $k \in K$ zugeordnet.
Die Adaption erfolgt in vier Schritten. Zunächst werden die Partikel P in die lokalen Koordinaten der Objekt- und Transfer-Frames K transformiert. Diese werden an die geänderten Randbedingungen K' der Reproduktion adaptiert, wodurch sich die Partikel gleichsam mit adaptieren (Abbildung 4.17). Nachdem die Partikel zurück in globale Koordinaten transformiert wurden erhält die adaptierte Trajektorie P' stellenweise eine Glättung bevor sie vom Roboter ausgeführt werden kann. Diese Schritte werden im Folgenden näher beschrieben.
Zunächst werden die Partikel in die lokalen Koordinatensysteme ihrer zugeordneten Frames transformiert. Aufgrund der Zuordnung der Partikel zu jeweils genau einem Frame, kann die Trajektorie $\mathcal{P}_\mathrm{T}$ als Sequenz $\mathcal{P}_S$ von Segmenten $\langle \mathcal{P}_k \rangle_\mathrm{frame}$ betrachtet werden, wobei frame das Koordinatensystem bezeichnet in dem sich alle Partikel des Segmentes befinden. Die Trajektorie ergibt sich somit zu

$$\mathcal{P}_S = (\langle \mathcal{P}_1 \rangle_\mathrm{world}, \langle \mathcal{P}_1 \rangle_\mathrm{world}, \ldots, \langle \mathcal{P}_t \rangle_\mathrm{world}),\ t \in \mathbb{N}^+. \tag{4.52}$$

Zunächst liegen alle Partikel in Weltkoordinaten vor. Da sich alle Partikel innerhalb eines Segments auf den gleichen durch W spezifizierten Frame beziehen, können alle Partikel in das lokale Koordinatensystem $k \in K$ des zugehörigen Objektes transformiert werden. Dazu wird die Transformation ${}^{k_i}M_\mathrm{world}$ für jeden Frame k_i angewendet. Die transformierten Segmente $\mathcal{P}_s$ berechnen sich zu

$$\langle \mathcal{P}_s \rangle_{k_i} = {}^{k_i}M^{-1}_\mathrm{world} \langle \mathcal{P}_s \rangle_\mathrm{world}. \tag{4.53}$$

Die Trajektorie befindet sich nun in einer lokalen Zwischendarstellung, die sich an veränderte Randbedingungen anpassen lässt. Dazu werden im nächsten Schritt während der Reproduktionsphase die Koordinatensysteme K'_O der Objekte sensorisch bestimmt. Basierend auf K'_O werden auch die Transferkoordinatensystem K'_T berechnet. Es werden die Transformationen ${}^{k'}M_\mathrm{world}$ für alle Frames $k' \in K'$ analog berechnet.
Das Ergebnis der adaptierten Trajektorie $\mathcal{P}'_\mathrm{T}$ erhält man, indem die Segmente $\mathcal{P}_s$ als $\mathcal{P}'_s$ zurück in das Weltkoordinatensystem transformiert werden, mit

$$\langle \mathcal{P}'_s \rangle_\mathrm{world} = {}^{k'_i}M_\mathrm{world} \langle \mathcal{P}_s \rangle_{k_i}. \tag{4.54}$$

Die Trajektorie, die durch dieses Verfahren generiert wird, kann Sprünge beinhalten, wenn sich die Pose der Objekte stark gegenüber der Demonstrationsphase verändert hat. Um dies zu kompensieren, wird hier eine Überblendung von Trajektoriensegmenten an den Grenzen der Segmente vorgestellt (Abbildung 4.18). Dadurch werden Bereiche an den Rändern der

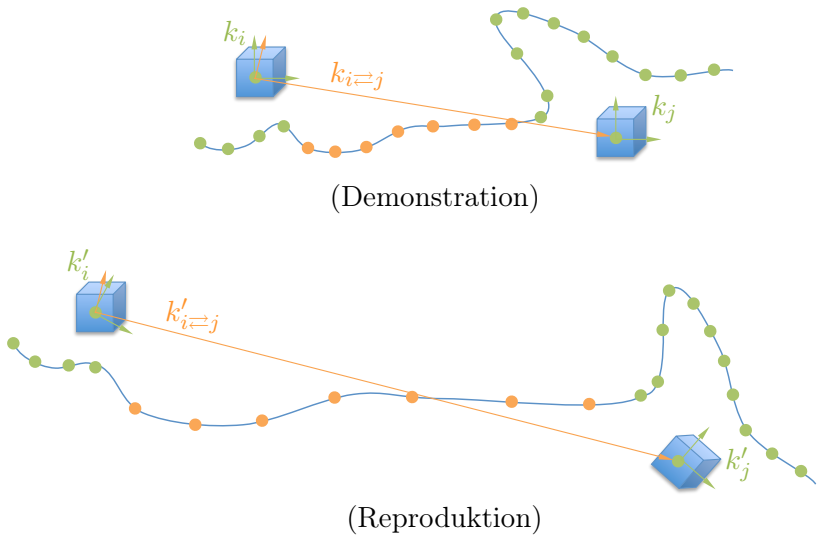


Abbildung 4.17: Darstellung der Funktionsweise einer Adaption mittels Motion Segments. Die Objekt-Frames k_i, k_j (grün) der Demonstration werden in der Reproduktion ebenfalls durch Sensorik bestimmt zu k'_i, k'_j und darauf aufbauen der Transferframe (orange) generiert. Durch die Adaption dieser Frames werden die Partikel entsprechend transformiert.

Transfer-Trajektorien so verändert, dass ein glatter Übergang entsteht.
Hierfür betrachtet man die ersten bzw. letzten n_b Partikel eines Segmentes $\langle \mathcal{P}_s \rangle$, das eine Transferbewegung repräsentiert, also dessen Frame $k_T \in K_\mathrm{T}$ ist. Diese Partikel werden so verändert, dass ihre Position P'_j bei Darstellung in homogenen Koordinaten neu berechnet wird zu P''_j mit

$$P''_j = blend(d_{norm}) \cdot {}^{k'_T}M_\mathrm{world} \cdot {}^{k}M_\mathrm{world}{}^{-1} \cdot P_j + \\ (1 - blend(d_{norm})) \cdot {}^{k'_o}M_\mathrm{world} \cdot {}^{k_o}M_\mathrm{world}{}^{-1} \cdot P_j \quad (4.55)$$

wobei k_T, k'_T die Transfer-Frames der Demonstrations- und Reproduktionsphase darstellen und k_O, k'_O die Objekt-Frames der Objekte in beiden Phasen. Beim ersten Term handelt es sich um die Position des Partikels im Transfer-Frame. Der zweite Term entspricht einer Reproduktion, bei welcher der Partikel dem Objekt-Frame zugeordnet wäre statt dem Transfer-Frame.
Die normalisierte Distanz d_norm berechnet sich zu

$$d_\mathrm{norm} = \begin{cases} \frac{|n_\mathrm{b} - i_\mathrm{s}|}{n_\mathrm{b}} & \text{wenn } 1 \leq i_\mathrm{s} \leq n_\mathrm{b}, \\ \frac{|n_\mathrm{s} - i_\mathrm{s}|}{n_\mathrm{b}} & \text{wenn } (n_\mathrm{s} - n_\mathrm{b}) \leq i_\mathrm{s} \leq n_\mathrm{s}, \\ 1 & \text{sonst} \end{cases} \quad (4.56)$$

wobei n_s die Menge der Partikel im Segment $\langle \mathcal{P} \rangle_s$ darstellt und i_s die Position des Partikels in $\langle \mathcal{P} \rangle_s$. Um einen glatten Übergang zu erhalten, wird die Sigmoid Funktion

$$blend(d_\mathrm{norm}) = \frac{1}{1 + e^{-16 \cdot (d_\mathrm{norm} - 0.5)}} \quad (4.57)$$

mit empirische bestimmten Parametern und einer Wertemenge $blend(d_\mathrm{norm}) \rightarrow [0; 1]$ verwendet. Zusätzlich zu diesen Punkten ist eine Interpolation zwischen den reproduzierten Partikeln

notwendig, da diese evtl. nicht im Interpolationstakt des Roboters reproduzierbar sind. Zu diesem Zweck kann eine stückweise lineare Interpolation oder eine (kubisch) Spline-Interpolation genutzt werden.

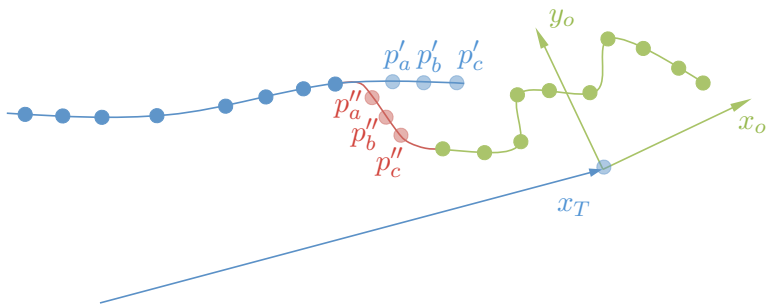


Abbildung 4.18: Zweidimensionales Beispiel einer Überblendung zweier Segmente. Die transparent blauen Partikel p'_a,p'_b und p'_c werden durch Formel 4.55 korrigiert, so dass ein weicher Übergang durch p''_a,p''_b und p''_c entsteht.

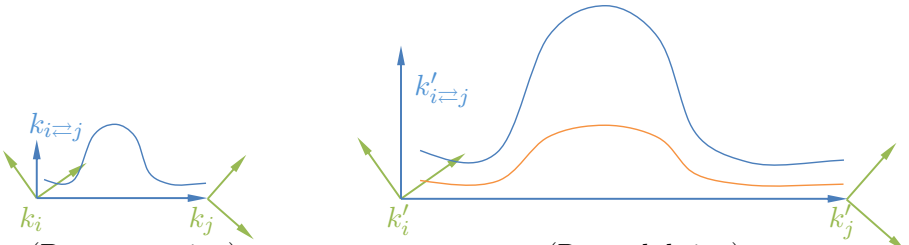
Zusammenfassend lässt sich die Adaption einer Trajektorie an eine neue Situation also folgendermaßen beschreiben. Zunächst wird die Trajektorie in Segmente unterteilt, von denen sich jeweils alle Partikel eines Segmentes auf ein Objekt oder den Transfer zwischen zwei Objekten beziehen. Danach werden die Partikel in die lokalen Koordinaten dieser Frames transformiert. In der Reproduktion werden die Partikel aufgrund der geänderten Koordinaten der Frames und der anschließenden Transformation in Weltkoordinaten auf die Situation angepasst. Eine Überblendung kompensiert Sprünge in der Trajektorie.

4.6.6 Optimierungen

Die vorgestellte Methode erlaubt die analytische Adaption einer Demonstration an geänderte Randbedingungen, jedoch können dabei unerwünschte und vermeidbare Effekte auftreten.

Normierung der Auslenkungs-Achsen

Zunächst soll die Adaption einer Demonstration wie in Abbildung 4.19 betrachtet werden. Gegeben ist beispielsweise die zweidimensionale Demonstration, wie sie auf der linken Seite abgebildet ist. Erhöht sich der Abstand zwischen den Objekten in der Reproduktionsphase, so wird das Transfer-Koordinatensystem entsprechend skaliert. Bei doppeltem Abstand verdoppelt sich sowohl x_T als auch y_T. Dies hat zur Folge, dass die demonstrierte Trajektorie nicht nur in die Länge gezogen wird, sondern auch in y_T skaliert. Die Abweichung in y_T kann so groß werden, dass sie außerhalb des Arbeitsraums des Roboters verläuft. Ein weiteres Problem entsteht dadurch, dass der Programmierer unter Umständen nicht damit rechnet, dass sich die Auslenkungen eine Transferbewegung, wie beispielsweise eines Bogens verstärken. Denn auch wenn der Programmierer bei der Demonstration eine geradlinige Bewegung anstrebt, kann er dies unter Umständen nicht bewerkstelligen. Dies hängt unter anderem davon ab, in welcher Pose sich der Roboter zu Beginn befindet, wie gut die Regelung zur Entlastung des Benutzers vom Robotergewicht funktioniert und wie erfahren der Benutzer im Umgang mit dem Roboter ist.



(Demonstration) (Reproduktion)

Abbildung 4.19: Adaption ohne Normierung der y_T- und z_T- Achse und mit Normierung (orange). Bei Erweiterung der Abstände skaliert y_T und z_T entsprechend x_T mit und somit können extreme Abweichungen in der adaptierten Trajektorie (blau) entstehen.

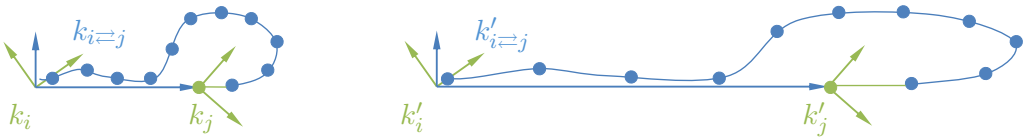
Aus diesem Grund wird eine Normierung von y_T und z_T im dreidimensionalen vorgenommen. Der Transfer-Frame ist nun definiert durch

$$\begin{aligned} x_\mathrm{T} &= (0,0,0)_{k_j} - (0,0,0)_{k_i} \\ x_\mathrm{T} \perp y_\mathrm{T}, x_\mathrm{T} &\perp z_\mathrm{T}, y_\mathrm{T} \perp z_\mathrm{T} \\ \|y_\mathrm{T}\| = 1, &\|z_\mathrm{T}\| = 1 \\ \angle(z_\mathrm{T}, -g) &= \min . \end{aligned} \tag{4.58}$$

Die Adaption nimmt damit die Form wie in Abbildung 4.19 (orange) an und entspricht mehr einer Stauchung und Streckung der Trajektorie in x_T-Richtung bei Transferbewegungen.

Normierung der x_T-Achse

Aufgrund der Maßnahme des letzten Abschnitts sind die Abweichungen entlang der y_T und z_T Achse korrigiert worden. Ein weiteres Problem entsteht, wenn eine Demonstration wie in Abbildung 4.20 an die dort abgebildete Situation adaptiert werden soll. Der Abstand des Tra-



(Demonstration) (Reproduktion)

Abbildung 4.20: Adaption einer Trajektorie bei der die Segmentierung abstandsbasiert durchgeführt wurde. Durch die Verdoppelung des Abstandes (rechts) entsteht ein deutlich verlängerter Bogen der Transferbewegung auf der „Rückseite“ des Objektes mit dem Frame k_j.

jektorie zum Objekt (k_j) wird nicht stetig geringer, sondern vergrößert sich erneut, da das Objekt von der „Rückseite“ manipuliert wird. Zwar ist durch die Segmentierung sichergestellt, dass die Objekt-basierten Bewegungen nahe des Objektes korrekt an die geänderte Pose des

Objektes angepasst werden, jedoch kann trotzdem der dort abgebildete Effekt eintreten. Aufgrund der Skalierung der x_T Achse wird dieser Bogen auf der Rückseite des Objektes ebenfalls in die Länge gezogen. Dies ist jedoch nicht zielführend, da die Transferbewegung dadurch unnötig verlängert wird.
Um die Trajektorie, wie in Abbildung 4.21 zu korrigieren, wird die Adaption der Transfer-Bewegungen folgendermaßen angepasst. Nachdem die Segmentierung erfolgt ist, werden nur die Partikel in x_T skaliert, die sich real zwischen den beiden Objekten befinden, d.h. mit einem x_T Wert von $0 \leq x_\mathrm{T} \leq 1$. Zur Umsetzung wird zusätzlich die x_T Achse normiert, so dass der Frame $k_{i \leftrightarrows j}$ folgendermaßen durch die beiden Objektframes k_i und k_j definiert ist:

$$x_\mathrm{T} \perp y_\mathrm{T}, x_\mathrm{T} \perp z_\mathrm{T}, y_\mathrm{T} \perp z_\mathrm{T}$$

$$\|x_\mathrm{T}\| = 1, \|y_\mathrm{T}\| = 1, \|z_\mathrm{T}\| = 1$$

$$\angle\left(z_\mathrm{T}, -g\right) = \min . \tag{4.59}$$

Man betrachtet nun alle Partikel P eines Segments $\mathcal{P}_s$, das eine Transferbewegung darstellt und adaptiert sie wie folgt in das lokale Koordinatensystem.

$$\left\langle \mathcal{P}_s \right\rangle_{k_{i \leftrightarrows j}} = {}^{k_{i \leftrightarrows j}} M_\mathrm{world} \left\langle \mathcal{P}_s \right\rangle_\mathrm{world} . \tag{4.60}$$

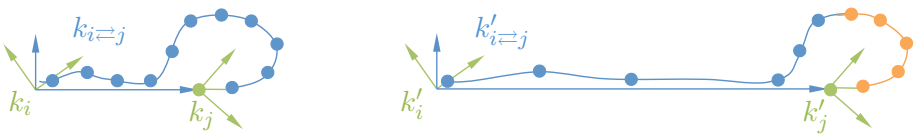
Desweiteren sei d_o der euklidische Abstand der beiden Objektframes k_i und k_j während der Demonstrationsphase und d'_o während der Reproduktionsphase. Nun erfolgt eine Anpassung aller Partikel P zu P', indem x_P, also der Wert in Richtung der x_T-Achse, berechnet wird zu x'_P durch

$$x'_P = \begin{cases} x_P - d_o + d'_o & \text{falls } x_p > d_o \\ x_P + d_o - d'_o & \text{falls } x_p < 0 \\ x_P \cdot \frac{d'_o}{d_o} & \text{sonst} \end{cases} \tag{4.61}$$

Das so entstandene modifizierte Trajektorienstück $\mathcal{P}'_s$ aller modifizierten Partikel P' kann nun in Weltkoordinaten transformiert werden

$$\left\langle \mathcal{P}'_s \right\rangle_\mathrm{world} = {}^{k'_{i \leftrightarrows j}} M^{-1}_\mathrm{world} \left\langle \mathcal{P}'_s \right\rangle_{k_{i \leftrightarrows j}} . \tag{4.62}$$

Die auf diese Weise durchgeführte Adaption erzeugt die in Abbildung 4.21 gezeigte Reproduktion, bei der Bewegungen auf der Rückseite nicht mehr skaliert werden.

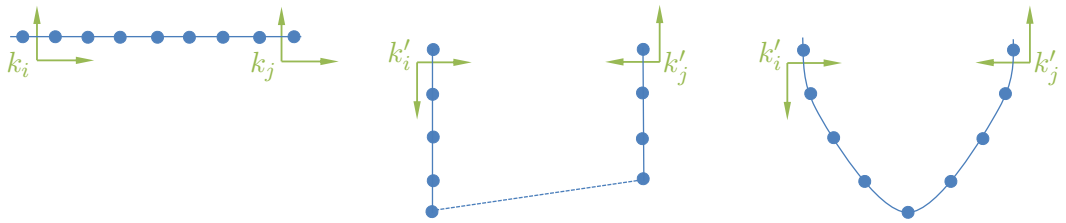


(Demonstration) (Reproduktion)

Abbildung 4.21: Durch die optimierte Zuordnung zu Transfer-Frames wird der Trajektorienabschnitt auf der Rückseite des Objektes nicht skaliert.

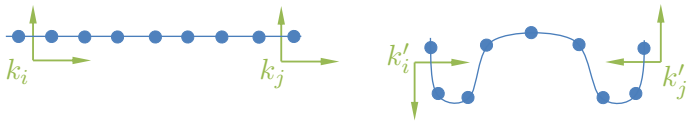
4.6.7 Alternative Adaptionen der Transferbewegungen

In diesem Kapitel soll diskutiert werden, in wieweit die Transfer-Frames für die Adaption einer Trajektorie notwendig sind. Eine alternative Adaption ohne Transfer-Frames würde eine vollständigen Segmentierung der Trajektorie voraussetzen, so dass jedes Partikel einem Objekt-Frame $k \in K_O$ zugeordnet ist. Bei der Adaption würde die Trajektorie entsprechend der geänderten Posen der Objekte angepasst werden. Betrachtet man Abbildung 4.22 (Mitte),



(Demonstration) (Reproduktion a) (Reproduktion b)

Abbildung 4.22: Links: Vom Programmierer demonstrierte Trajektorie (blau) und Objekt-Frames (grün); Mitte: Adaption der Trajektorie ohne Verwendung von Transfer-Frames; Rechts: Adaption der Trajektorie ohne Transfer-Frames, jedoch mit Überblendung;



(Demonstration) (Reproduktion)

Abbildung 4.23: Links: Vom Programmierer demonstrierte Trajektorie (blau) und Objekt-Frames (grün); Rechts: Adaption der Trajektorie bei Verwendung von Transfer Frames mit Überblendung.

in der die Objekte der Demonstrationsphase in der Reproduktionsphase lediglich eine veränderte Orientierung besitzen, wird deutlich, dass dies extreme Verformungen der Trajektorie hervorrufen kann. Da jedes Partikel einem der Objekte zugeordnet ist, werden diese entsprechend mit-transformiert. Somit entsteht nicht nur ein Sprung in der Trajektorie, sondern auch eine starke Ausprägung zur Seite, so dass die in Kapitel 4.6.6 genannten Probleme auftreten können. Selbst eine starke Überblendung der beiden Trajektorienteile wäre in diesem Fall nur bedingt hilfreich (Abbildung 4.22, Rechts).
Erst bei Verwendung der Transfer-Frames, wie in Abbildung 4.23 gezeigt, wird die Auslenkung zur Seite minimiert und besitzt zwischen den Objekten wieder Ähnlichkeit mit der ursprünglichen Trajektorie bei gleichzeitiger Anpassung an die geänderten Objektorientierungen. Der Effekt wird stärker, je größer der Abstand der Objekte ist.
Der Nachteil der Verwendung von Transfer-Frames liegt darin, dass Trajektorienstücke, die nach menschlichem Ermessen einem Objekt zugeordnet werden müssten, irrtümlicherweise als Transferbewegung klassifiziert werden können. Um dies zu verhindern sind weitere Untersuchungen bei der Erstellung der Verbindungen $\mathcal{L}_R$ notwendig (siehe Ausblick Kapitel 6.2).

4.6.8 Zusammenfassung

In diesem Abschnitt wurde eine analytische Näherung des Ansatzes aus Kapitel 4.5 vorgestellt, der es erlaubt eine Demonstration an veränderte Randbedingungen zu adaptieren. Um dies zu ermöglichen, wird die vom Benutzer demonstrierte Trajektorie segmentiert, so dass jedes Partikel der Trajektorie entweder einem Objekt oder einer Transferbewegung zugeordnet wird. Durch die Zugehörigkeit zu genau einem Koordinatensystem (Frame) können die Änderungen dieses Frames zwischen Demonstration und Reproduktion direkt auf die zugehörigen Partikel übertragen werden. Durch verschiedene Maßnahmen, wie die Normierung von Achsen oder der Glättung der Segmenträder erzeugt die Adaption schließlich das gewünschte Ergebnis, welches die Aufgabe unter geänderten Randbedingungen erfüllt.

4.7 Verhaltensgenerierung

Bisher wurde die konkrete Adaption einer Trajektorie an geänderte Randbedingungen diskutiert. Nun soll diese Adaptionsmöglichkeit in den in Kapitel 2 und 3 vorgestellten Ansatz integriert werden [Grot14c]. Dazu wird die Generierung eines Verhaltensmoduls $M \in \mathcal{M}$ beschrieben.
Aus einer Demonstration D lässt sich ein Verhaltensmodul $M = [S, C, A, \delta, \omega, s_0, S_F]$ erzeugen, indem die Trajektorie entsprechend der generierten Verbindungen $\mathcal{L}_\mathrm{R} = \{\mathcal{L}_1, \ldots, \mathcal{L}_l\}$ segmentiert wird (siehe Kapitel 4.5.3) zu $\mathcal{P}_\mathrm{T}$ mit

$$\mathcal{P}_\mathrm{T} = (\mathcal{P}_1, \ldots, \mathcal{P}_n). \tag{4.63}$$

Da die Verbindungen nicht notwendigerweise eine zeitliche Ordnung der Trajektorienpartikel berücksichtigen, muss immer dann ein neues Segment erstellt werden, wenn ein Partikel P_{i+1} nicht die gleichen Verbindungen zu Objekten wie Partikel P_i besitzt oder sich die Werkzeugoperationen G_{i+1} auf andere Werkzeuge beziehen als G_i. Ein Segment entspricht damit einem Abschnitt der Trajektorie, bei dem sich die Anforderungen an Ressourcen nicht ändern. Erst mit Beginn des nächsten Segmentes ändern sich die benötigten Ressourcen. Für jedes der n Segmentstücke $\mathcal{P}_i$ wird nun ein Zustand s_i in S eingefügt. Zusätzlich wird der zu $\mathcal{P}_n$ korrespondierende Zustand s_n als einziger Finalzustand $S_F = \{s_n\}$ gekennzeichnet. Die Übergangsfunktion $\delta : S \times C \to S$ bestimmt sich zu

$$\delta : (s_i, C_i) \to s_{i+1}, \quad i = 0, \ldots, n-1 \tag{4.64}$$

und $\omega : S \times C \to A$ zu

$$\omega : (s_i, C_i) \to A_i, \quad i = 0, \ldots, n-1 \tag{4.65}$$

wodurch der Automat die Form einer Sequenz annimmt. In dieser entspricht die Aktion A_i beim Übergang von s_i nach s_{i+1} dem Trajektorienstück $\mathcal{P}_i$ und den zu diesen Partikeln gehörigen Werkzeugoperationen G_i. Entsprechend ergibt sich A zu $A = \bigcup_{i \in \mathbb{N}^+} A_i \cup \bigcup_{j \in \mathbb{N}_0} G_j$.
Die Bedingungen C_i entsprechen allen Ressourcen, die zur Ausführung von A_i notwendig sind. Je nach Abstraktionsebene (Kapitel 2.6) besteht C_i aus dem Roboterarm, den für G_i notwendigen Werkzeugen $R_g \subseteq \mathcal{R}_\mathrm{G}$ und den mit diesem Teilstück $\mathcal{P}_i$ verknüpften Objekten $R_o \in \mathcal{R}_\mathrm{O}$. Das bedeutet, mit jedem neuen Zustand s_i ändern sich die notwendigen Ressourcen für die Aktionen A_i.
Es ist nicht notwendig, die Ressourcen einer Bedingung C_i auch in alle Übergangsbedingungen

C_j mit $j > i$ einzufügen. Denn auf diese Weise werden Ressourcen angefordert, die überhaupt nicht benötigt werden und sofern die Ressourcen preemptiv sind, können diese sowieso zu einem späteren Zeitpunkt erneut akquiriert werden. Falls sie nicht preemptiv sind, weil sie z.B. verändert wurden, ist es ohnehin nicht möglich, dass sie vor Prozessende von einem anderen Prozess akquiriert werden.
In den Bedingungen C_i sind u.a. die ursprünglichen Positionen und Orientierungen der Objekte gespeichert. Durch den Abgleich mit den passenden Ressourcen durch eine *match*-Funktion stehen alle Informationen zur Verfügung um ein Trajektoriensegment $\mathcal{P}_i$ an die aktuellen Objektpositionen zu adaptieren. Die Adaption geschieht online zur Laufzeit.
Alternativ zu dieser Erzeugung des Verhaltensmoduls könnte für jedes einzelne Partikel $P_i \in \mathcal{P}$ ein Zustand $s_i \in S$ erzeugt werden, der jeweils nur eine einzige Aktion beinhaltet. Allerdings sprechen hier zwei Gründe dagegen. Zum einen würde der Aufwand zur Bestimmung des Prozesszustandes (Kapitel 3.2) drastisch steigen, da die Prüfung auf vorhandene Ressourcen nicht mehr nur bei einer Änderung der Ressourcenanforderung stattfinden würde, sondern vor jeder Aktion. Zudem würde dies keinerlei Mehrwert bieten. Zum anderen liefert die zuvor genannte Methode eine bessere Übersichtlichkeit, da die somit generierten Sequenzen deutlich kürzer sind. Damit wäre eine u.U. gewünschte visuelle Rückmeldung wesentlich effizienter.
Eine weitere Alternative bestünde in der Repräsentation der gesamten Trajektorie $\mathcal{P}_\mathrm{T}$ durch einen einzigen Zustand. Dies entsprecht einer vollständigen Betriebsmittelanforderung, was den Aufwand für die Ressourcenakquise minimiert. Durch die in Kapitel 3.2 gezeigten Verfahren kann dies auch bei einer hohen Anzahl von Ressourcen effizient durchgeführt werden. Der Nachteil dieser Darstellung liegt in den unter Umständen langen Wartezeiten, da die Ausführung erst möglich ist, sobald alle Ressourcen verfügbar sind.

4.8 Zusammenfassung

In diesem Kapitel wurde das Ziel Z2 durch die intuitive Programmierung eines Roboters betrachtet. Dazu wurde zunächst ein Überblick über den Stand der Forschung gegeben und die Aufgabenstellung formal definiert. Anschließend wurden zwei Ansätze vorgestellt, welche die Adaption einer durch einen Benutzer demonstrierten Trajektorie an geänderte Randbedingungen erlauben. Beim ersten Ansatz handelt es sich um ein numerisches Verfahren, dass die Trajektorie als Partikel betrachtet, die untereinander und mit der Umwelt verbunden sind. Durch die Änderung der Umwelt entstehen Wechselwirkungen zwischen den Partikeln, die so lange anhalten bis sich erneut ein stabiles Gleichgewicht einstellt. Dies entspricht der Adaption an die neue Situation. Der zweite Ansatz stellt eine analytische Näherung dar, bei der die Partikel entweder einem Objekt oder einer Transferbewegung zugeordnet werden. Durch die Transformation der Objekte zwischen Demonstration und Reproduktion kann die Adaption der Partikel segmentweise berechnet werden. Beide Möglichkeiten können in den in Kapitel 2 und Kapitel 3 vorgestellten Ansatz zur Umsetzung von Z1 integriert werden.

5 Experimentelle Untersuchungen

In diesem Kapitel sollen die Verfahren, die zur Erreichung von Z1 und Z2 vorgestellt wurden experimentell evaluiert werden. Dazu wird in Kapitel 5.1 das verwendete System beschrieben, das für die Untersuchung der verhaltensbasierten Ausführung (Kapitel 5.2), der Programmierung mittels orientierter Partikel (Kapitel 5.3) und der Programmierung mittels Motion Segments (Kapitel 5.4) verwendet wird.
Der Aufbau der Experimente teilt sich dabei jeweils in zwei Bereiche. Zum einen wird in einem funktionalen Teil die Wirkungsweise der einzelnen Komponenten betrachtet. Die dafür notwendigen Experimente werden in der Simulation oder mit einfachen geometrischen Objekten wie beispielsweise Holzbausteinen durchgeführt, da sie nicht zwangsläufig einen bestimmten Kontext aufweisen müssen. Zum anderen werden Realwelt-Experimente durchgeführt, die möglichen Anwendungen entsprechen, um die Effektivität und Nützlichkeit des Systems aufzuzeigen. Die Anwendungsbereiche für diese Experimente sind der EU-Agenda Horizon 2020 entnommen [Bisc09]. Diese listet die folgenden Domänen, welche durch die Verwendung von Robotern stark profitieren können:

- Zivil (Gebäudebau, Suche & Rettung)
- Gewinnung (Bergbau & Mineralien)
- Logistik & Transport (Waren & Personen)
- Haushalt (Haushaltshilfe, Betreutes Leben)
- Landwirtschaft (Ackerbau, Försterei)
- Gesundheit (Chirurgie, Rehabilitation)
- Herstellung (Produktion, Lebensmittel)

In dieser Arbeit werden Anwendungen gewählt, die der Produktion, dem Haushalt und mit Einschränkung auch der Logistik zugeordnet werden können. In diesen Bereichen kann eine intuitive Programmierung eines Roboters einen Mehrwert bringen, auch wenn dieser nicht mobil, also stationär ist. Insbesondere lassen sich in all diesen Bereichen bei Vorgabe eines stationären Roboters und unter Verzicht auf die Berücksichtigung von Dynamiken die folgenden verschiedenen statischen Handhabungsaufgaben identifizieren:

- Direkte Manipulation
- Greifen und Ablegen (Pick-and-Place)
- Greifen und Manipulieren (Pick-and-Manipulate)

Pick-and-Place Aufgaben entsprechen einem klassischen Anwendungsgebiet eines Roboters, wie beispielsweise bei der Montage oder Kommisionierung von Werkstücken. Als *Direkte Manipulation* wird die Art der Manipulation betrachtet, bei der am Roboter ein Spezialwerkzeug

montiert ist, das während der Aufgabe nicht entfernt wird und der Objektveränderung dient. Dies ist beispielsweise beim Fräsen der Fall. Diese Art von Aufgaben ist ebenfalls mit dem Ansatz aus Kapitel 2 und 3 umsetzbar, wird jedoch hier nicht explizit betrachtet, da es als Sonderfall von *Pick-and-Manipulate* betrachtet werden kann. Dort muss das Objekt zunächst aufgegriffen werden und kann anschließen auf ein weiteres Objekt angewendet werden. Hierein fallen prinzipiell alle Aufgaben mit Werkzeugen oder beispielsweise das Einschenken von Flüssigkeiten. Die im folgenden verwendeten Aufgaben entstammen also den Pick-and-Place und der Pick-and-Manipulate Klassen aus den zuvor genannten Domänen.

5.1 Prototypische Umsetzung

Alle Experimente wurden mit einem KuKa LBR 4 durchgeführt, der auf einem Arbeitstisch montiert ist (siehe Abbildung 5.1). Als Werkzeug ist ein Schunk PG-070 Backengreifer montiert. Zur Erkennung der Objekte auf dem Arbeitsplatz dient eine an der Decke montierte Logitec c930 HD-Webcam und eine Microsoft Kinect.

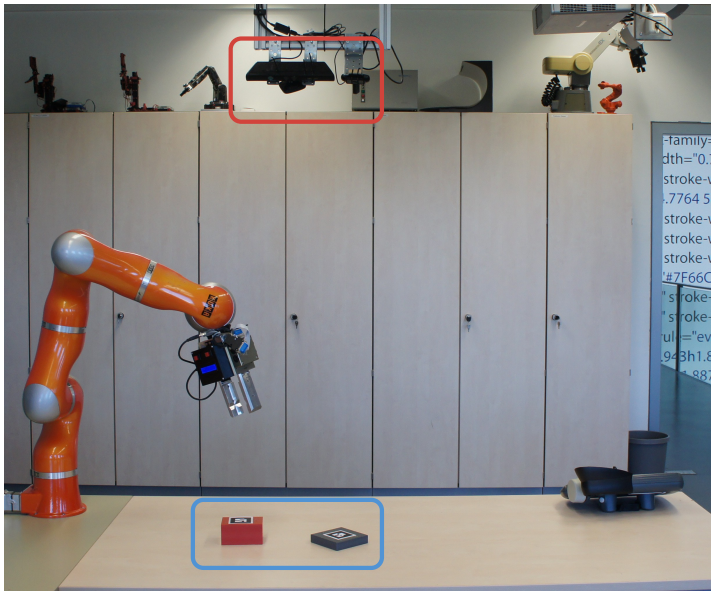


Abbildung 5.1: Das für die Experimente verwendete Robotersystem besteht aus einem fest montierten Kuka LBR IV und einem System (rot) zur Erkennung von Objekten (blau).

Der Roboter kann in einen geregelten Impedanzmodus versetzt werden (Gravitationskompensation), damit er vom Benutzer in der Demonstrationsphase frei bewegt werden kann. Während der Bewegung werden Position und Gelenkwinkel im Takt von 12ms aufgezeichnet. Die Aufzeichnung der Trajektorie beginnt, sobald der Roboter die Startkonfiguration verlässt, da die Startposition als virtuelles Objekt für die Adaption benötigt wird. Würde die Startposition ebenfalls als Trajektorienpartikel gespeichert werden, käme es zu Problemen bei der numerischen Berechnung von $F_{\mathrm{r},i}$ und $\tau_{\mathrm{r},i}$.
Der Greifer lässt sich über eine direkt angebrachte Steuereinheit bedienen, die sowohl ein kraft- als auch positionsbasiertes Greifen ermöglicht (Abbildung 5.2). Auch hier wird die Position und die anliegende Kraft aufgezeichnet und zusammen mit der entsprechenden Position des

Roboters abgespeichert. Die entwickelte Software wird auf einem Intel Xeon W3565 mit vier Kernen und 16GB Arbeitsspeicher ausgeführt.
Zur Erkennung der Objekte wird eine modifizierte Version der Alvar-Tracking Library [VTT 14] verwendet, die es ermöglicht die Position und Orientierung von Markern im Arbeitsraum zu detektieren. Die Marker werden an den Objekten angebracht, so dass deren Positions- und Orientierungsänderung zwischen Demonstrations- und Reproduktionsphase erkannt werden kann. Alternativ könnten hier auch Feature-basierte Verfahren wie etwa VFH [Rusu10] oder SIFT eingesetzt werden. Die Kommunikation aller Module erfolgt mittels ROS [Smit15].

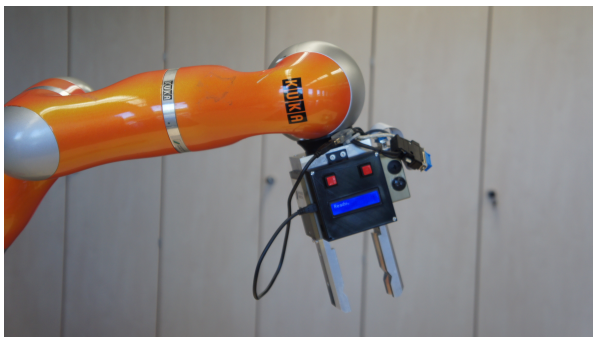


Abbildung 5.2: Am Roboter selbst befindet sich eine Einheit zur kraftbasierten Steuerung des Greifers.

Für die Durchführung aller Experimente erfolgt die Detektion der Marker bzw. Objekte durch eine Kamera, die an der Decke angebracht ist. Alternativ dazu kann die Kamera auch am Roboter selbst montiert werden, genauer gesagt am Werkzeugflansch des Roboters. Beide Varianten besitzen ihre Vor- und Nachteile. Die Montage an der Decke bietet eine bessere Übersicht, muss jedoch auf die Position des Roboters kalibriert werden und der Roboter selbst kann für Verdeckungen sorgen. Da es sich um einen stationären Roboter handelt, kann dies im Vorhinein geschehen oder durch einen dedizierten Marker am Roboter berechnet werden. Bei einer Montage der Kamera direkt am Roboter ist die Kalibrierung theoretisch weniger aufwendig, da vorhandene Software verfügbar ist [Stro15]. Dafür schränkt die angebrachte Kamera den Arbeitsraum des Roboters ein und der ohnehin kleinere Sichtbereich kann durch gegriffene Objekte weiter eingeschränkt werden. Aus praktischen Gründen, wurde hier die Variante der Deckenmontage gewählt.

5.2 Ausführung von Verhalten

In diesem Abschnitt soll die in Kapitel 2 und Kapitel 3 vorgestellte verhaltensbasierte Architektur experimentell evaluiert werden. Dazu werden in Kapitel 5.2.1 die Aspekte der prozessbasierten Ausführung von Verhalten gezeigt. Anschließend erfolgt eine quantitative Evaluierung der in Kapitel 3.7 und Kapitel 3.8 vorgestellten Scheduling-Verfahren.

5.2.1 Prozessbasierte Ausführung

Zunächst soll eine qualitative Evaluierung der Architektur an zwei Beispielen gezeigt werden. Das erste Beispiel orientiert sich dabei an kleinen und mittelständischen Unternehmen.

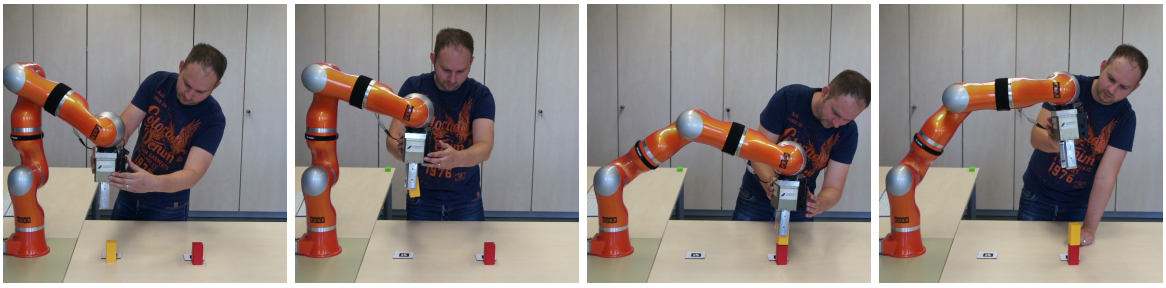


Abbildung 5.3: v.l.n.r.: Demonstration der Montageoperation, die in den Experimenten adaptiert und ausgeführt wird.

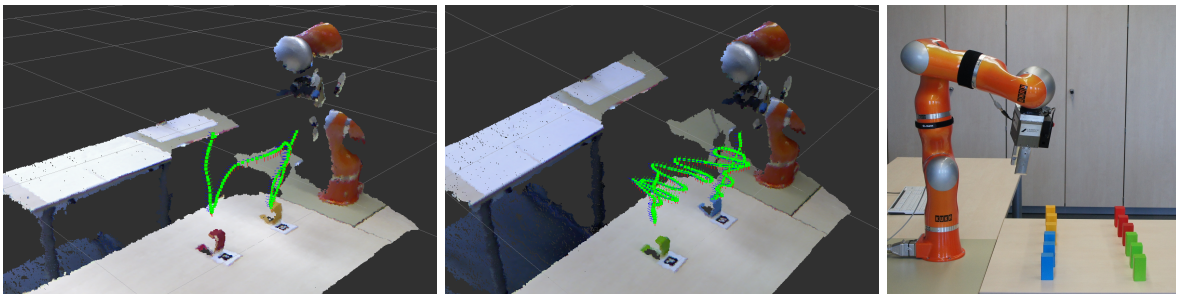


Abbildung 5.4: Die demonstrierten Trajektorien von Montage-gelb-rot (links) und Montage-blau-grün (mitte) werden auf eine Menge von Bauteilen angewendet (rechts).

An einem Arbeitsplatz werden zu verschiedenen Zeitpunkten Bauteile zur Montage bereitgestellt. Im Experiment werden zur Umsetzung jeweils blaue auf grüne Holzbausteine und gelbe auf rote Holzbausteine gestapelt (siehe Abbildung 5.3 und 5.4). Diese Bausteine repräsentieren die Ressourcen, die neben dem Roboter selbst zur Durchführung notwendig sind. Diese beiden Arbeiten stehen symbolisch für Arbeiten in der Industrie, die unregelmäßig aber Chargen-weise anfallen. Das System kennt die beiden Verhalten „Montage-blau-grün“ (A) und „Montage-gelb-rot“ (B). Verhalten A ist künstlich verlängert, indem zusätzliche Bewegungen bei der Transferbewegung anfallen, damit die Verhalten eine unterschiedliche Länge besitzen (Abbildung 5.4).
Als Scheduling wird FCFS verwendet, da es nach der Bewertung aus Kapitel 3.7 gut für industrielle Arbeiten geeignet ist. Jedoch sind auch andere denkbar, da hier zunächst nur die Funktionsweise der prozessbasierten Ausführung untersucht werden soll. Die Adaption erfolgt analytisch durch Motion Segments, um die Zeiten nicht zu verfälschen. In Abbildung 5.5 ist das entsprechende Gantt- Diagramm zu sehen. Aufgrund der Länge der Prozesse ist gut zu erkennen, ob es sich um Typ A (lang) oder B (kurz) handelt. Die letzten Zeilen entsprechen dem Leerlaufprozess (`idle`) und dem Wechsel für Prozesse (`switch`), der in diesem Fall direkt und ohne Verzögerung stattfindet.
Für die Durchführung werden verschiedene Bauteile auf den Arbeitsplatz positioniert. Eine Objekterkennung informiert das System über eine erste Charge an Objekten ($t = 2s$). Das System erstellt nun für jedes mögliche Paar von Ressourcen einen Prozess, der direkt bereit ist und führt die Prozesse nacheinander in der Reihenfolge aus, in der die notwendigen Ressourcen erkannt werden. Nachdem die Charge abgearbeitet ist ($t_1 = 127s$) wechselt das System in den Leerlauf bis neue Bauteile erkannt werden ($t2 = 173s$). Hier beginnt der Roboter erneut mit

der Montage, da neue Prozesse für die Montage bereit sind.

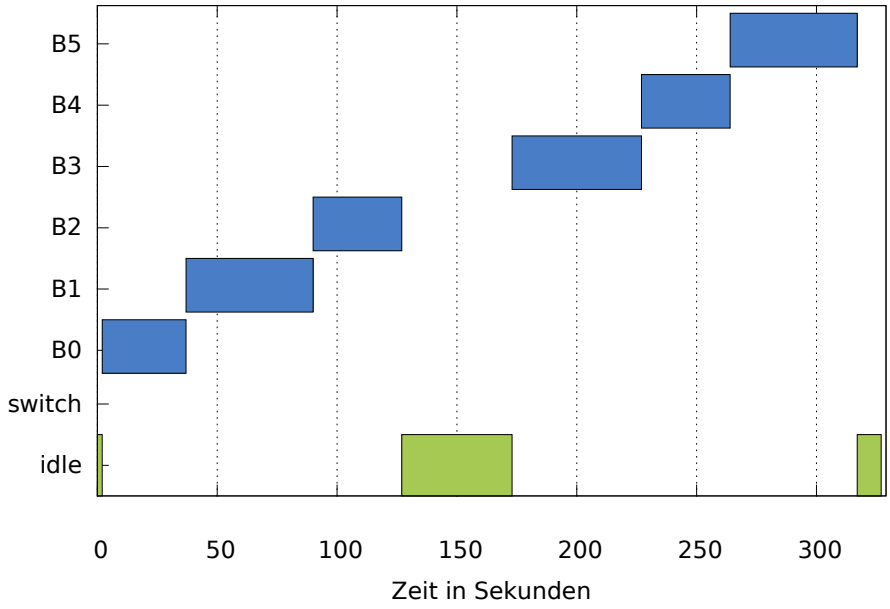


Abbildung 5.5: Gantt Diagramm einer Chargen-weise Bearbeitung von Bauteilen.

Es ist in diesem Fall einfach möglich gewesen durch lediglich zwei Demonstration (Verhalten A und B) ein nützliches System zu programmieren, das selbstständig die notwendigen Aufgaben übernimmt, sobald diese anfallen. Durch das FCFS Scheduling werden die Aufgaben der Reihe nach erledigt. Dies ist natürlich abhängig davon, wann die Objekte erkannt werden. Im Gegensatz dazu könnte die Verwendung des PS Schedulings bestimmte Montageoperationen priorisieren.
Das zweite Beispiel orientiert sich an der Domäne des Haushalts - in diesem Fall einem Küchen-Szenario. Der Roboter soll als Hilfe bei der Zubereitung von Mahlzeiten fungieren. Dazu kennt er die folgenden Verhalten, die über das prioritätenbasierte Scheduling (PS) koordiniert werden.

- `stirr` (hohe Priorität): Der Roboter greift ein Objekt vom Typ „Löffel“ auf und verrührt damit den Inhalt des Objektes vom Typ „Schüssel“.
- `pour` (niedrige Priorität): Der Roboter schüttet den Inhalt des Objektes „Tasse“ in das Objekt „Schüssel“.

Das System startet im Leerlauf-Modus (`idle`). Sobald der Löffel erkannt wird, wird ein `stirr`-Prozess bereit und aktiv (Abbildung 5.6 und 5.7). Nachdem der Löffel aufgegriffen wurde, ist das System wieder `idle`, da die Ressource Schüssel fehlt und der Prozess somit blockiert. Bei Hinzufügen der Schüssel wird der Prozess `stirr` wieder aufgenommen und beginnt zu rühren. Sobald die beiden Tassen in die Szene gebracht werden, wird das Rühren unterbrochen, da `pour` eine höhere Priorität besitzt. Der Löffel wird beiseite gelegt und die Tassen nacheinander geleert. Da `pour` nicht als Singleton angelegt ist, können verschiedene Zutaten hinzugefügt werden. Anschließend erfolgt ein erneuter Kontextwechsel zu `stirr`. Dazu wird der Löffel wieder gegriffen und der Roboter rührt weiter in der Schüssel.

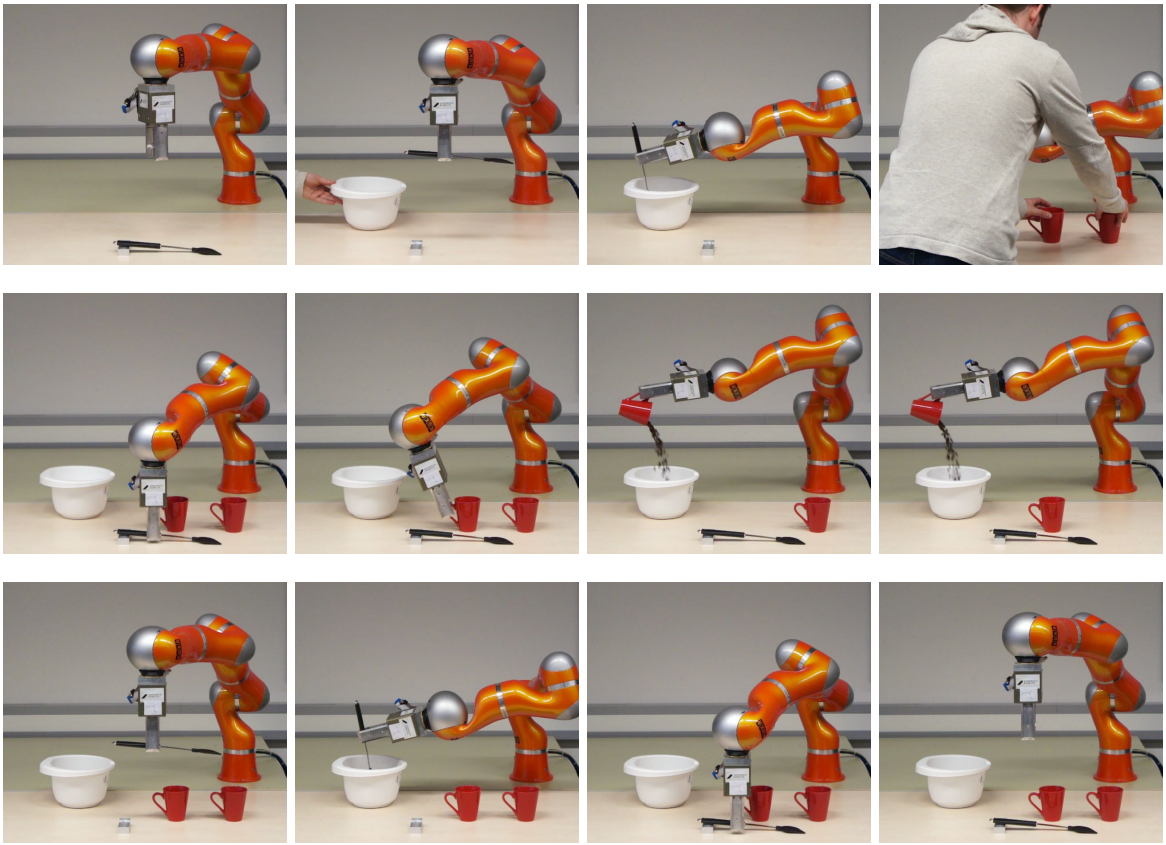


Abbildung 5.6: Veranschaulichung von Beispiel 2: Das System befindet sich zunächst im Leerlauf (`idle`). Sobald der Löffel erkannt wird, greift ihn der Roboter aufgrund des `stirr`-Verhaltens auf und blockiert so lange, bis die Schüssel erkannt wurde. Sobald die Tassen erkannt werden, erfolgt ein Kontextwechsel zu `pour` mit Ablegen des Löffels. Nachdem beide Tassen durch `pour`-Verhalten in die Schüssel geschüttet wurden, wird der Löffel wieder aufgenommen und das `stirr`-Verhalten fortgesetzt.

Der Wechsel zwischen den `pour` Prozessen geht ohne Verzögerung vonstatten, da der Greifer leer ist und somit nichts abgelegt und gegriffen werden muss. Der Wechsel von `stirr` zu `pour` dauert jedoch deutlich länger, da hier die Ressource Greifer gebraucht wird und damit Abhängigkeiten unter den Ressourcen aufgelöst werden müssen. Auch der Wechsel zurück zu `stirr` dauert entsprechend lange, da der Löffel wieder aufgegriffen werden muss. In dieser Anwendung wäre es möglich, statt dem Leerlaufprozess ein Singleton-Verhalten zur Suche zu aktivieren, um z.B. den Sichtbereich für die Objekterkennung zu maximieren. Sofern es sich dabei um Zufallsbewegungen ohne Greiferfunktionalität handelt, würde es vom Ablauf keinen Unterschied zur Variante mit `idle` darstellen.
Die Anzahl der maximalen Anwendungen eines Verhaltens auf eine Ressource ist mit Ausnahme der Schüssel jeweils auf $e_{\max} = 1$ begrenzt, damit auch mehrere Tassen hineingekippt werden können. Es wäre auch denkbar, die maximale Anwendbarkeit von `stirr` zu erhöhen, um das Rühren durch wiederholtes Ausführen zu verlängern.

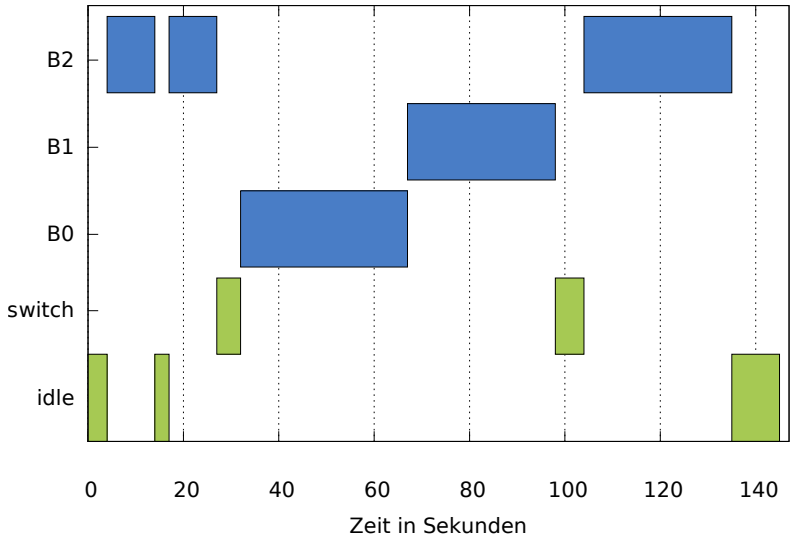


Abbildung 5.7: Im Gantt-Diagramm des Koch-Szenarios ist zu sehen, dass Prozesswechsel ohne gegriffenes Objekt (B0 nach B1) deutlich schneller durchgeführt werden, als solche mit gegriffenem Objekt (B2 nach B0, B1 nach B2).

Die Beispiele zeigen, dass sowohl das Scheduling, als auch das konsistente Unterbrechen und sichere Wiederaufnehmen von Prozessen bei Manipulationsaufgaben problemlos funktioniert. Die experimentelle Validierung für die Voraussetzung eines bereits durchgeführten Verhaltens als Bedingung für ein weiteres entfällt aus Gründen des Umfangs.

5.2.2 Quantitative Evaluierung der Scheduling-Verfahren

Neben einer qualitativen soll auch eine quantitative Evaluierung des Systems erfolgen. Dazu werden die Schedulingverfahren getestet und hinsichtlich der folgenden Kennzahlen ausgewertet, die Rückschlüsse auf die Anwendbarkeit geben sollen:

Mittlere Bearbeitungszeit $\bar{t}_{\text{flow}}$**:** Mittlere Zeit zwischen dem Zeitpunkt, zu dem der Prozess initial `bereit` ist (Bereitzeitpunkt) und dem Zeitpunkt der Terminierung

Mittlere Wartezeit $\bar{t}_{\text{wait}}$**:** Mittlere Zeit zwischen dem Bereitzeitpunkt und dem Zeitpunkt, zu dem der Prozess das erste Mal `aktiv` wird.

Gesamtzeit t_{total}**:** Der Zeitraum zwischen dem der erste Prozess `bereit` ist und der letzte Prozess `terminiert`.

Anzahl Prozesswechsel n_{sw}**:** Anzahl der durchgeführten Prozesswechsel

Als Verhalten dienen zwei Pick-and-Place Verhalten unterschiedlicher Länge. Die Trajektorie von Verhalten A besitzt 2113 Datenpunkte und die von Verhalten B 548 Datenpunkten (Abbildung 5.8). Beide Verhalten wurden durch Vormachen programmiert und werden, um die Zeitmessung nicht zu beeinflussen, mittels Motion Segments adaptiert. Die erzeugten Prozesse erfahren zusätzlich eine gewisse Varianz in der Länge der Ausführung durch die Adaption. Es werden verschiedene Szenarien $S1$ bis $S4$ erzeugt. Der Unterschied liegt darin, wie viele

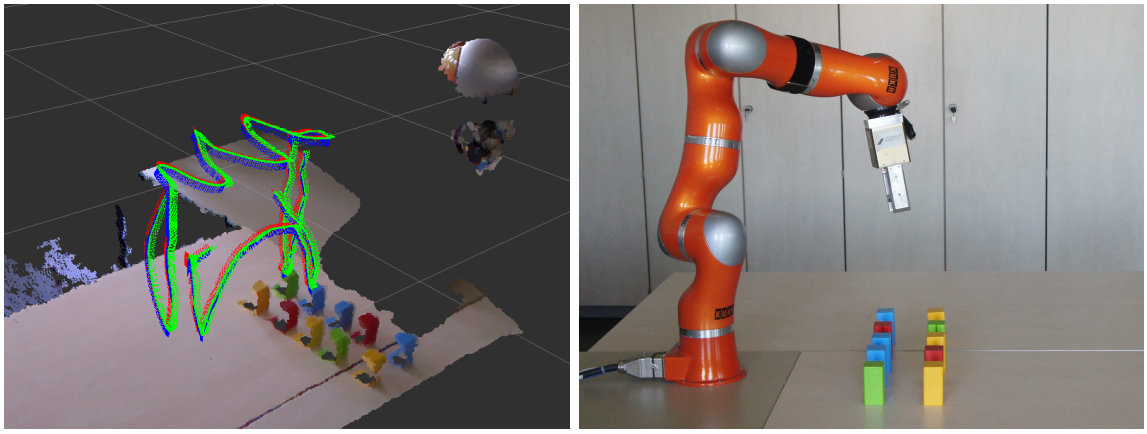


Abbildung 5.8: Darstellung der Trajektorien der Pick-and-Place Verhalten A und B. Die Objekte aus der ersten Reihe werden durch eine kurze, direkte Bewegung versetzt und die Objekte aus der zweiten Reihe durch eine künstlich verlängerte Bewegung.

unterschiedliche Verhalten existieren und ob alle Ressourcen von Anfang an präsent sind oder erst nach und nach durch die Objekterkennung sichtbar geschaltet werden.

	Alle Ressourcen zum Zeitpunkt $t = 0$ präsent	Verzögerte Bereitstellung der Ressourcen
Verhalten A und B	S1	S2
Nur Verhalten B	S3	S4

Für Round-Robin(RR) werden Zeitscheiben von 4, 8 und 16 Sekunden getestet. Eine Zeitscheibe wird auch dann preemptiert, wenn der aktive Prozess vor dem Ende der Scheibe terminiert. Das Priority-Scheduling (PS) priorisiert hier die langen Prozesse A. Als Länge eines Prozesses für Shortest-Job-Next (SJN) und Shortest-Remaining-Time-Next-Scheduling (SRN) wird die Anzahl der Samples der Trajektorie verwendet. Alternativ könnte man auch die Länge der Trajektorie über ein Integral berechnen. Allerdings erfolgt die Adaption hier online, so dass diese Information nicht a priori zur Verfügung steht.
Die Ergebnisse der Scheduling Versuche sind in Tabelle 5.1 zu sehen. Obwohl alle Zeitmessungen durch die Variationen der Generalisierung, den Zeitpunkten der Ressourcenfreigaben und die Menge der Verhalten variieren, lassen sich einige Gemeinsamkeiten erkennen. So lässt sich feststellen, dass Round-Robin die längste Gesamtzeit und die höchste mittlere Bearbeitungszeit besitzt. Dies ist nicht weiter verwunderlich, da in Robotikanwendungen Kontextwechsel deutlich teurer sind als bei Betriebssystemen und bei Round-Robin sind diese am häufigsten. Die Tasks werden von einem physisch bewegten Roboter ausgeführt, der gewissen Geschwindigkeits- und Beschleunigungsschranken unterliegt. Zusätzlich müssen bei einigen Kontextwechseln Objekte gegriffen und abgelegt werden, was einen Kontextwechsel noch teurer macht. Ab einem gewissen Punkt besteht ein Großteil der Gesamtzeit nur noch aus Kontextwechseln (Abbildung 5.9). Mit größeren Zeitscheiben, reduziert sich die mittlere Bearbeitungszeit $\bar{t}_{\text{flow}}$, bis zu dem Punkt an dem ein Prozess innerhalb einer Zeitscheibe ausgeführt werden kann. Der Nachteil langer Zeitscheiben und auch nicht-präemptiver Verfahren ist die vergleichsweise hohe Reaktionszeit auf neue Objekte was durch eine höhere Wartezeit $\bar{t}_{\text{wait}}$ bezeugt wird. Die Zeiten variieren zwar über die Szenarien S1 bis S4, besitzen jedoch alle

S	Scheduling	$\bar{t}_{\text{flow}}$(s)	$\bar{t}_{\text{wait}}$(s)	t_{total}(s)	n_{sw}
1	FCFS	56.10	36.29	560.99	9
1	PS	49.58	29.81	495.83	9
1	RR (4s)	152.95	7.83	1529.46	106
1	RR (8s)	86.88	15.85	868.79	35
1	RR (16s)	60.97	25.33	609.74	15
1	SJF	45.91	25.62	459.06	9
1	SRN	45.63	25.48	456.32	9
2	FCFS	29.04	9.04	290.39	9
2	PS	40.62	10.33	406.18	13
2	RR (4s)	115.37	7.96	1153.67	85
2	RR (8s)	61.26	10.57	612.56	28
2	RR (16s)	42.37	11.43	423.71	15
2	SJF	35.60	8.47	356.02	13
2	SRN	36.22	8.81	362.21	13
3	FCFS	43.88	28.88	438.78	9
3	PS	37.60	22.63	376.04	9
3	RR (4s)	130.68	9.61	1306.82	86
3	RR (8s)	70.08	14.46	700.81	31
3	RR (16s)	52.16	24.65	521.63	14
3	SJF	44.09	29.02	440.88	9
3	SRN	43.87	28.88	438.72	9
4	FCFS	31.29	16.24	312.93	9
4	PS	39.61	15.31	396.09	14
4	RR (4s)	116.42	8.20	1164.20	85
4	RR (8s)	65.15	13.75	651.52	29
4	RR (16s)	39.38	12.22	393.77	16
4	SJF	31.05	16.06	310.53	9
4	SRN	31.44	16.36	314.40	9

Tabelle 5.1: Ergebnisse der Scheduling-Versuche mit verschiedenen Algorithmen in den Experimenten S1-S4.

die gleiche Tendenz. Dies kann auch beim Priority-Scheduling beobachtet werden, wenn lange Prozesse mit hoher Priorität vorhanden sind. Betrachtet man $\bar{t}_{\text{flow}}$ und $\bar{t}_{\text{wait}}$, ist Shortest-Remaining-Time-Next durch die Preemption generell eine gute Wahl. Die Anzahl der Unterbrechungen ist vergleichsweise gering, da nur bei neuen kürzeren Prozessen gewechselt wird. Gleichzeitig minimiert es die mittlere Bearbeitungszeit $\bar{t}_{\text{flow}}$ bei geringer Gesamtzeit $\bar{t}_{\text{total}}$ und moderater Wartezeit $\bar{t}_{\text{wait}}$.
Round-Robin lohnt sich im Haushalt bzw. der Industrie nur, wenn die Aufgaben einen messbaren, bleibenden Fortschritt nach einer Zeitscheibe aufweisen oder wenn lange Wartezeiten verhindert werden sollen. Selbst dann muss die Zeitscheibengröße mit Bedacht gewählt werden, um einen guten Kompromiss aus Reaktionszeit und Bearbeitungszeit zu gewährleisten. Das Priority-Scheduling lohnt sich, wie bereits in Kapitel 3.7 erwähnt, wenn einige Arbeiten gegenüber anderen zu bevorzugen sind. Auch im Haushalt kann Priority-Scheduling sinnvoll

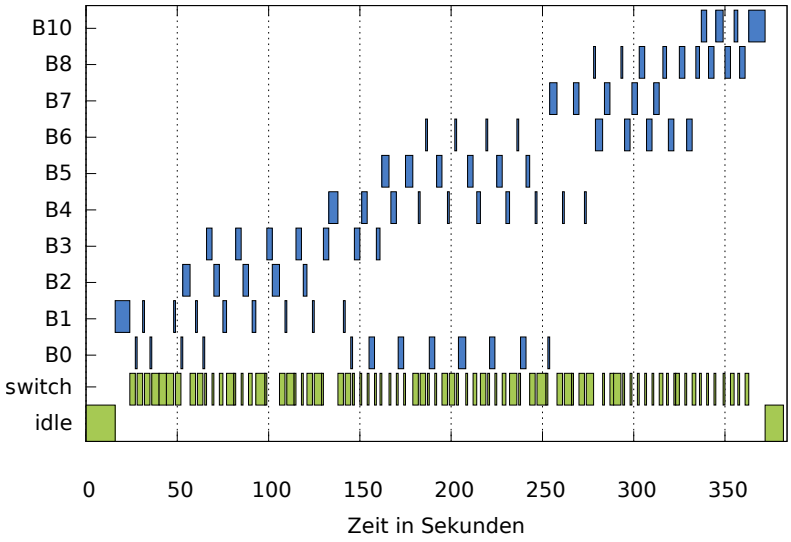


Abbildung 5.9: Gantt Diagramm eines Round-Robin Schedulings mit einer Zeitscheibe von vier Sekunden im Szenario S4.

eingesetzt werden, wenn Leerlaufaktivitäten mit niedriger Priorität existieren, wie beispielsweise ein tägliches Reinigen des Arbeitsplatzes oder das Einräumen des Geschirrspülers. Das Ergebnis dieser Arbeit darf jedoch nicht zeitkritisch sein, so dass wichtige Arbeiten mit höherer Priorität eingeschoben werden können.

5.2.3 Quantitative Evaluierung des verhaltensbasierten Schedulings

Nachdem die klassischen Scheduling-Verfahren untersucht wurden, soll in diesem Kapitel das verhaltensbasierte Scheduling betrachtet werden. Wie in Kapitel 3.8 beschrieben, können unter bestimmten Umständen mehrere Prozesse des gleichen Verhaltens zu einem zusammengefasst werden, um die Anzahl der Aufgreif- und Ablege-Operationen zu reduzieren.
Zur Evaluierung werden die Aufgaben mit dem FCFS-Scheduling und mit dem verhaltensbasierten Scheduling durchgeführt und anschließend die Laufzeiten verglichen. In Abbildung

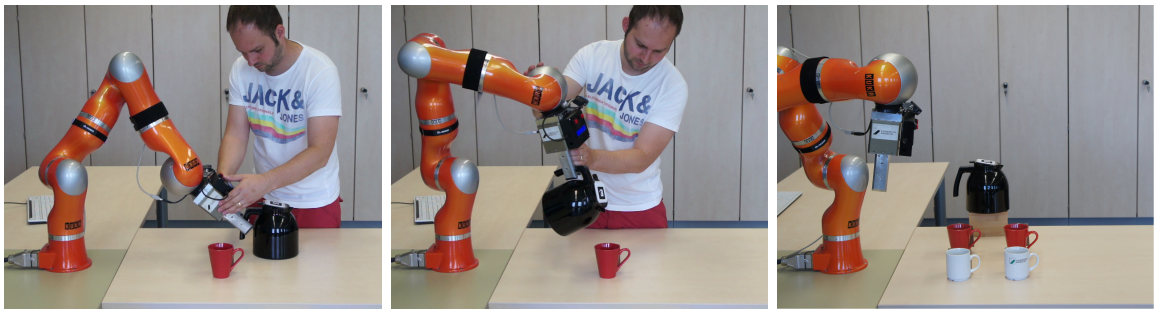


Abbildung 5.10: Demonstration der Aufgabe „Kaffee einschenken“ (links, mitte) und Ausgangssituation zur Durchführung des verhaltensbasierten Schedulings (rechts).

5.10 ist das Szenario zu sehen, bei dem. die demonstrierte Aufgabe des Einschenkens für alle vier Tassen durchgeführt werden soll. Dies könnte eine typische Aufgabe für einen Roboter am Frühstückstisch einer Familie sein oder bei einem teilautomatisierten Catering-Service. In Abbildung 5.11 sind die Gantt-Diagramme der beiden Scheduling Verfahren abgebildet. Es ist deutlich zu erkennen, wie kurz die eigentliche Manipulation (B1, B2) im Vergleich mit

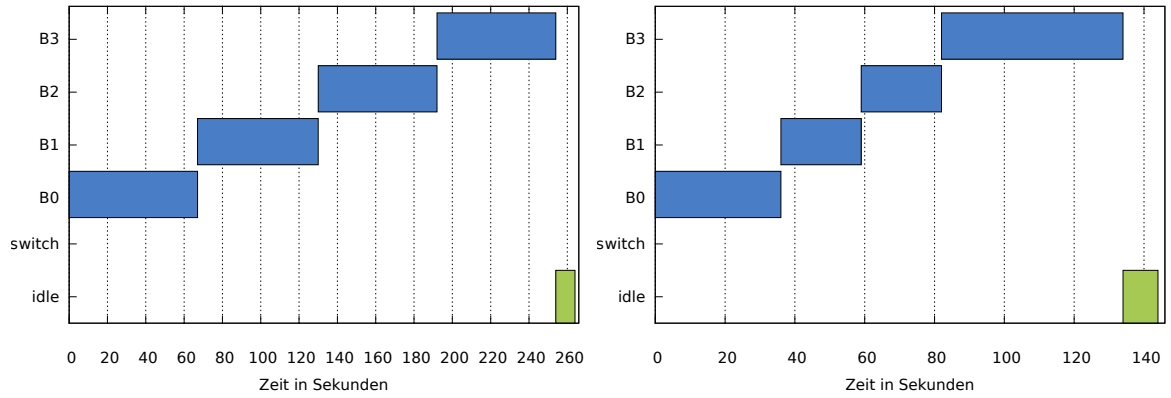


Abbildung 5.11: Gantt Diagramm der Aufgabe „Kaffee einschenken“ mit FCFS Scheduling (links) und mit dem verhaltensbasierten Scheduling (rechts).

den kompletten Prozessen ist. Zwar variieren die Laufzeiten etwas aufgrund der Adaption der Trajektorien, jedoch verringerte sich in diesem Beispiel die Gesamtlaufzeit von 443 Sekunden auf 208 Sekunden und die mittlere Bearbeitungszeit von 111s auf 52s. Beide Werte konnten in diesem Beispiel also mehr als halbiert werden.
Die Durchführung eines Szenarios mit n gleichartigen Prozessen ermöglicht eine Einsparungen von $(n-1)\cdot(T(\text{pick})+T(\text{place}))$. Die Einsparung hängt stark von der Dauer des Aufgreifens und Ablegens und damit von der Segmentierung der Trajektorie ab. In diesem Fall wurde die Adaption mittels Motions Segments gewählt und eine Zuordnung basierend auf einer Voronoi-Segmentierung. D.h. es wurde die Bewegung die sich ausschließlich auf die Tasse bezieht, als Manipulation angesehen. Prinzipiell könnte bei einer Schranken-basierten Segmentierung auch die Transferbewegung eingespart werden und lediglich die Manipulationsbewegung mehrfach ausgeführt werden, jedoch besteht die Gefahr, relevante Bewegungen zu eliminieren. Bei einer konservativen Kombination der Verhalten ist die Zeitersparnis geringer, da weniger eingespart wird. Jedoch ist die Wahrscheinlichkeit höher, dass alle Aufgaben korrekt ausgeführt werden.

5.3 Intuitive Programmierung mittels orientierter Partikel

In diesem Kapitel wird die Adaption mittels orientierter Partikel experimentell evaluiert. Dazu wird zunächst die prinzipielle Funktionsweisen des Ansatzes in Kapitel 5.3.1 untersucht. Anschließend werden die Auswirkungen der Verbindungen $\mathcal{L}_{\text{R}}$ auf die Adaption in Kapitel 5.3.2 diskutiert. Schließlich erfolgt eine qualitative und quantitative Evaluation des Ansatzes in Kapitel 5.3.3.

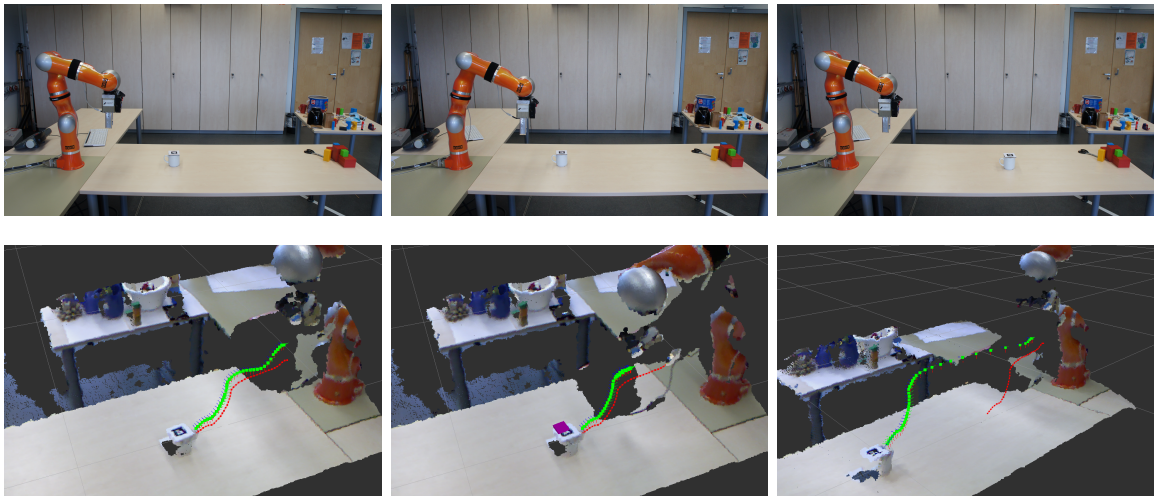


Abbildung 5.12: Adaption (grün) der demonstrierten Trajektorie (rot) Serie A: Durch Veränderung des Objektabstands wird die demonstrierte Trajektorie gestreckt bzw. gestaucht. In der unteren Reihe sind die adaptierten Trajektorien der jeweils darüber abgebildeten Situation visualisiert.

5.3.1 Wirkungsweise der Adaption

Nach [Frie96] lassen sich viele Aufgaben durch eine Folge von Operationen der Art Aufgreifen-Manipulieren-Ablegen beschreiben. Für die Untersuchung der prinzipiellen Funktionsweise soll daher zunächst eine einfache Pick-Aufgabe gewählt werden. Das aufzugreifende Objekt ist hierbei nicht symmetrisch, d.h. dessen Orientierung ist relevant. Die Demonstration und Adaption an die geänderten Randbedingungen ist in Abbildung 5.12 zu sehen. Entfernt sich der Roboter vom Objekt, wird die Trajektorie gestreckt, kommt er näher wird sie gestaucht. Auch die Änderung der relativen Orientierung zueinander wird kompensiert (Abbildung 5.13). Des Weiteren ist zu beachten, dass wenn sich die Orientierung des Objektes ändert, sich die Trajektorie, entsprechend anpasst. Je höher der Abstand der Partikel vom Objekt ist, desto geringer gleicht sich deren Position an, um eine glatte Bahn zu erzeugen. Gleichsam adaptiert sich natürlich die Orientierung der Partikel, und damit letztendlich des Roboters, so dass die Bewegung im Bezug zum Objekt korrekt ausgeführt wird (Abbildung 5.14).
Als zweites soll die online-Fähigkeit demonstriert werden (Abbildung 5.15). Es wird eine Pick-and-Place Aufgabe demonstriert, bei der die Position und Orientierung der Objekte verändert wird, so dass eine Adaption stattfindet. Auch während der Roboter die reproduzierte Trajektorie abfährt, kann die Adaption fortgesetzt werden, so dass Abschnitte, die noch nicht ausgeführt wurden noch adaptiert werden. Um dies zu zeigen, wird das Ablage-Ziel bewegt nachdem die Ausführung bereits begonnen hat. Die Adaption ist sogar noch bis kurz vor Ende der Aufgabe möglich, wobei es dem Roboter trotzdem möglich ist, das Objekt korrekt abzulegen[1].
Um die Kollisionsvermeidung und den Umgang mit Hindernissen zu zeigen, wird eine einfache Transferbewegung zwischen zwei Objekten demonstriert. Während der Reproduktion sind die Objekte verschoben (Abbildung 5.16) und ein Hindernis wird so positioniert, dass es mit

[1] Ein Video hierzu findet sich unter ai3.uni-bayreuth.de/projects/introp.

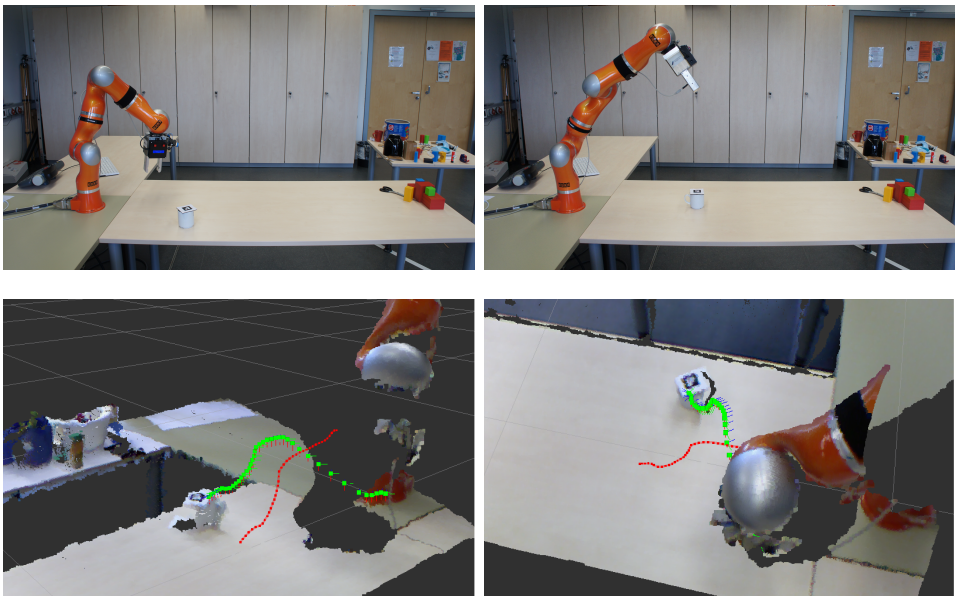


Abbildung 5.13: Adaption (grün) der demonstrierten Trajektorie (rot) Serie B: Die Adaption ermöglicht nicht nur eine Positions- sondern auch eine Orientierungsänderungen des Objektes und eine veränderte Startkonfiguration des Roboters.

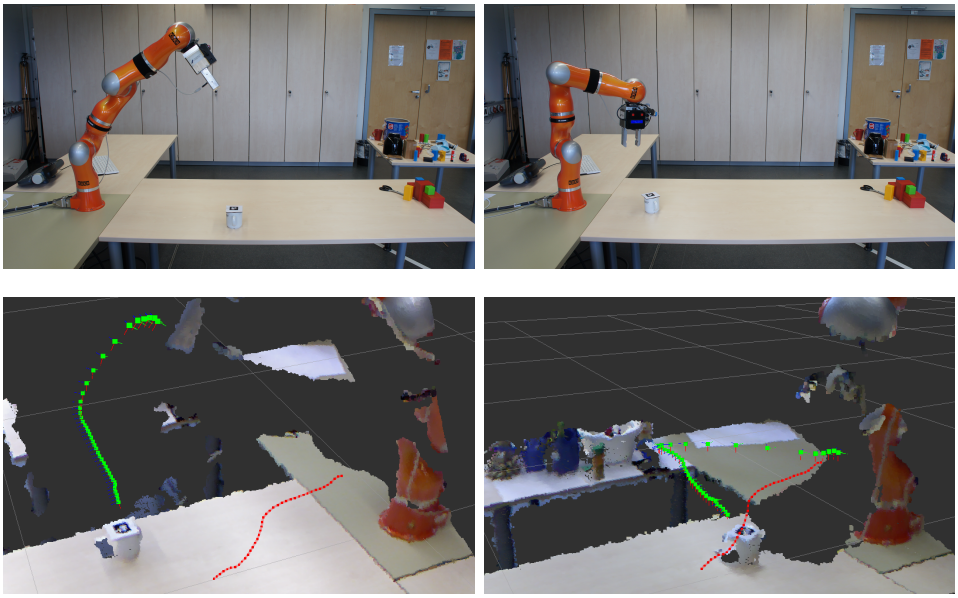


Abbildung 5.14: Adaption (grün) der demonstrierten Trajektorie (rot) Serie C: Die Adaption erfolgt auch bei starken Änderungen zwischen Demonstrationsphase und Adaptionsphase.

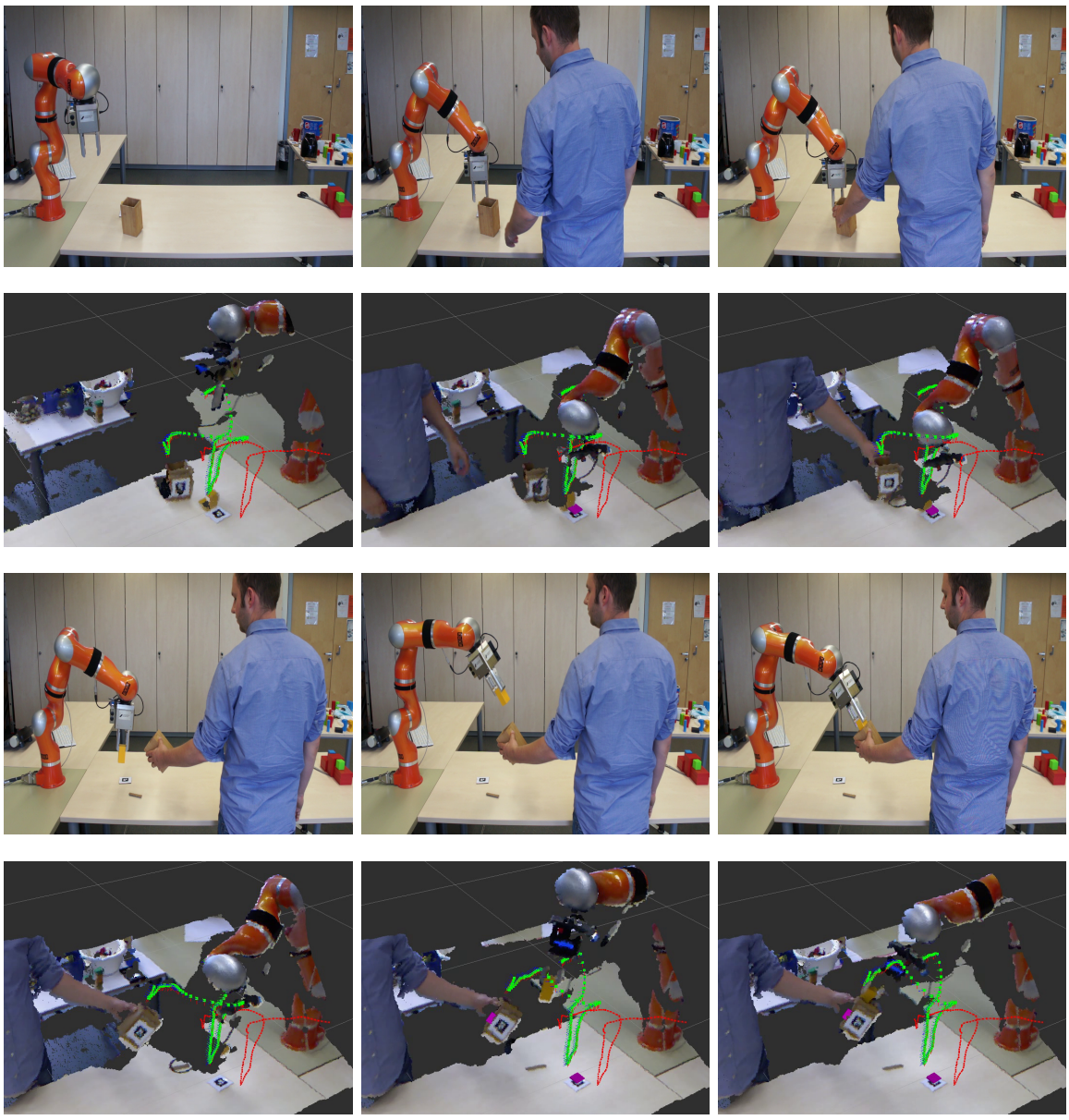


Abbildung 5.15: Durch die Online-Adaption können trotz bereits gestarteter Reproduktion noch Anpassungen an veränderte Objektpositionen durchgeführt werden. Die Trajektorie (grün) passt sich hier auch nach gestarteter Ausführung noch an ein verändertes Ablageziel an.

der Trajektorie kollidiert, worauf diese einen Bogen um das Hindernis bildet. Da das ganze onlinefähig ist, restauriert sich die Trajektorie, sobald sich das Hindernis entfernt. Die Kollisionsvermeidung ist abhängig von der Distanz ab welcher die Kraft F_c wirkt. Wird eine geringe Distanz gewählt, ab der die Kraft wirkt, entsteht eine kurze halbkreisförmige Ausweichbewegung. Je höher man die Distanz wählt, desto glatter wird die Bahn und desto besser sind die ursprünglichen Bewegungsmuster noch zu erkennen.

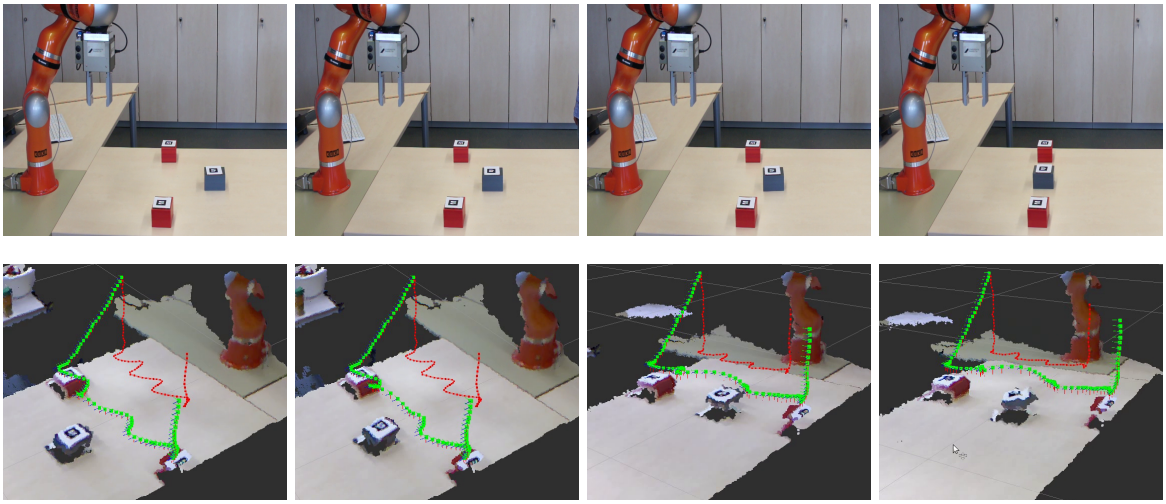
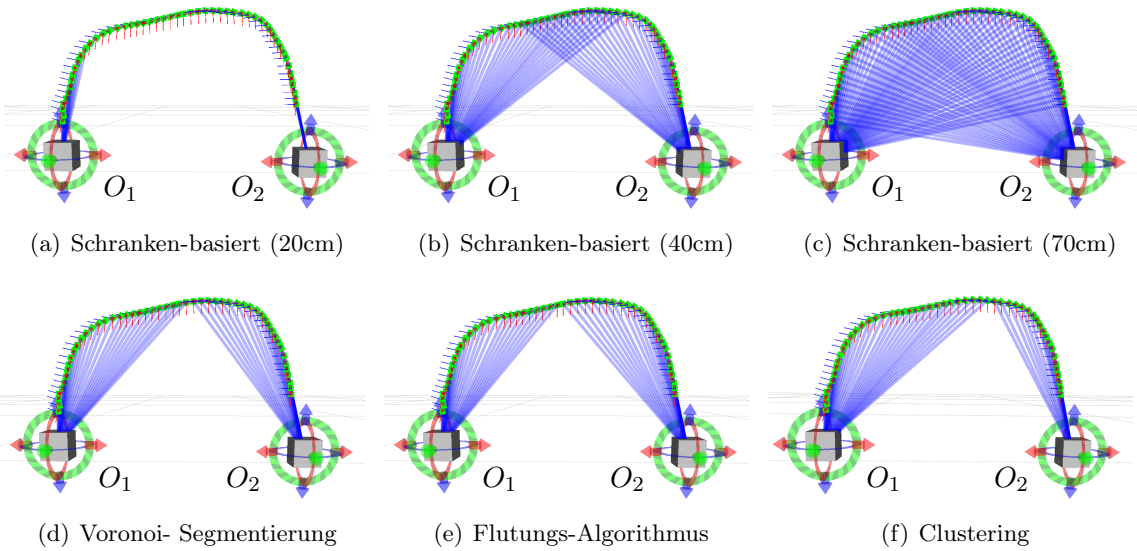


Abbildung 5.16: Die Kollisionsvermeidung wird dadurch erreicht, dass Hindernisse ein abstoßendes Potential auf die Trajektorienpartikel ausüben. Im Beispiel ist zu sehen, wie sich ein graues Hindernis der adaptierten Trajektorie (grün) nähert und diese daraufhin eine Ausweichbewegung vollführt.

5.3.2 Auswirkung der Verbindungen $\mathcal{L}_R$ auf die Adaption

Nachdem die generelle Funktionsweise gezeigt wurde, soll der Einfluss der verschiedenen Verbindungen gezeigt werden. Dazu werden die Verbindungen durch die in Kapitel 4.5.3 eingeführten Methoden bei verschieden Demonstrationen erzeugt und visualisiert. Anschließend wird eine qualitative Evaluierung der Adaptionsergebnisse durchgeführt.
In diesen Experimenten wird ein K-Means Clustering verwendet, das als Merkmalsvektor X die Position der Partikel, der Abstand der Greiferbacken und die Position der Partikel innerhalb der Trajektorie verwendet. Die Anzahl der zu erzeugenden Cluster entspricht der Anzahl der Objekte, sofern es nicht explizit erwähnt wird. Der Flutungs-Algorithmus arbeitet in diesen Experimenten mit einer einheitlichen Behälter-Größe.
In Abbildung 5.17 sind die Verbindungen $\mathcal{L}_R$ zwischen den Objekten und der Trajektorie zu sehen, wobei die Verbindungen als blaue Linien und die Objekte als graue Quader dargestellt sind. Die Trajektorie (grün) entspricht der Bahn der Werkzeugspitze des Roboters und wird mit verringerter Auflösung dargestellt, so dass nicht alle Partikel sichtbar sind. Dies dient der Übersichtlichkeit, da sonst die einzelnen Verbindungen und auch Streckungseffekte nicht mehr wahrzunehmen sind. Der größte Unterschied liegt hier zwischen der oberen und der unteren Ergebnisreihe. Während bei den Schranken-basierten Verbindung eine Zuordnung eines Partikels zu mehreren Objekten möglich ist, existiert in der unteren Reihe eine eindeutige Zuordnung.



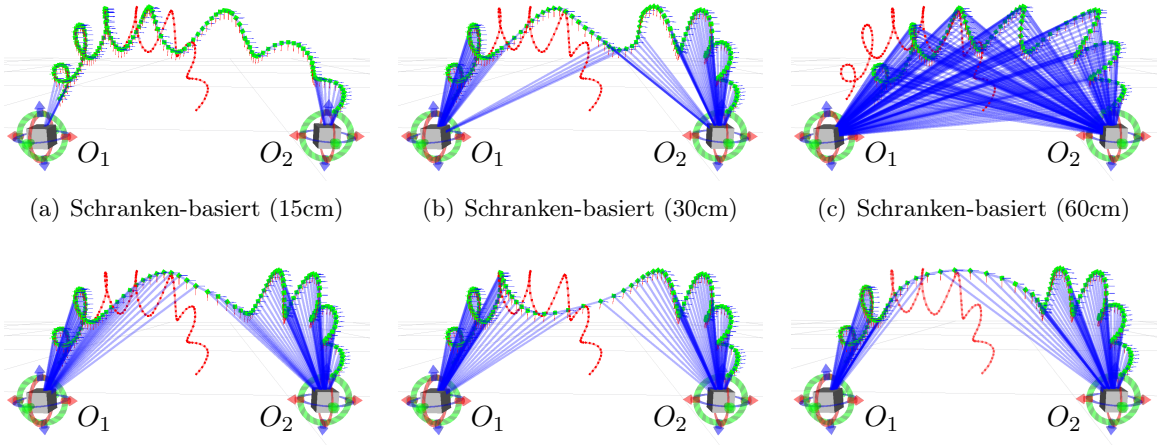
(a) Schranken-basiert (20cm) (b) Schranken-basiert (40cm) (c) Schranken-basiert (70cm)

(d) Voronoi- Segmentierung (e) Flutungs-Algorithmus (f) Clustering

Abbildung 5.17: Vergleich der unterschiedlichen Verbindungen zwischen Trajektorie und Objekten.

Bei einer Schranke von 70cm (Abbildung 5.17 (c)) tritt der Fall ein, dass jedes Objekte mit jedem Partikel verbunden ist. Die Unterschiede in der unteren Reihe sind gering. Die Trajektorie dieses ersten Beispiels ist relativ symmetrisch, daher erzeugt sowohl der Flutungs-Algorithmus als auch das Clustering ein ähnliches Ergebnis wie die Voronoi-Segmentierung.
Als nächstes soll die Verbindungsmenge $\mathcal{L}_R$ und Adaption einer Trajektorie wie in Abbildung 5.18 betrachtet werden. Die Trajektorie besitzt in diesem Experiment eine kompliziertere Form, die einer spiralförmigen Bewegung zwischen zwei Objekten entspricht. Die Demonstration rot und die Adaption grün dargestellt. Für die Reproduktion wird das Objekt O_2 nach rechts bewegt[2]. Aufgrund der relativ symmetrischen Verbindung wird bei den unteren Varianten (Flutung, Voronoi, Clustering) die Trajektorie vor allem in der Mitte gestreckt, wobei es zu den Objekten hin schwächer wird. Der Unterschied der Verfahren liegt hauptsächlich darin, an welcher Stelle die Streckung stattfindet. Werden beim Clustering vier statt zwei Cluster gebildet kann ein Ergebnis wie in Abbildung 5.19 entstehen. Durch die abwechselnde Zuordnung der Cluster zu den Objekten wird nicht die typische Schraubenbewegung reproduziert sondern entspricht mehr einem Kreis. Diese Gefahr besteht aufgrund der hohen Clusteranzahl. Es können dadurch vermehrt kleinere Cluster entstehen, die aufgrund ihrer Nähe dem falschen Objekt zugeordnet werden. Da genau dies durch das Clustering verhindert werden soll, sollte die korrekte Bestimmung der Cluster-Anzahl Bestandteil zukünftiger Untersuchungen sein.
Bei der abstandsbasierten Variante mit einer 30cm Schranke (b) wird ein ähnliches Bild erzeugt, wie bei den zuvor genannten Verfahren, da die Schranke etwa dem halben Objektabstand entspricht. Bei der Schranken-basierten Verbindung mit 70cm (c) ist diese so hoch, dass jedes Partikel mit beiden Objekten verbunden ist. Beim auseinanderziehen der Objekte wird die Bahn komplett gestreckt. Auf den ersten Blick sieht dies nach einer optimalen Anpassung an die neuen Randbedingungn aus, da die Trajektorie sehr gleichmäßig gestreckt wird. Problematisch hierbei ist, dass die Partikel die einen sehr geringen Abstand zu den Objekten

[2]Da hier eine qualitative Untersuchung durchgeführt wird, werden keine exakten Positionen angegeben.



(a) Schranken-basiert (15cm) (b) Schranken-basiert (30cm) (c) Schranken-basiert (60cm)

(d) Voronoi-Segmentierung (e) Flutungs-Algorithmus (f) Clustering

Abbildung 5.18: Vergleich der Adaptionen (grün) einer Demonstration (rot) ausgehend von unterschiedlichen Verbindungen.

haben, verschoben werden und damit Objektmanipulationen evtl. nicht mehr korrekt sind. Die Schranken-basierte Variante mit 15cm (a) wurde nach 100000 Adaptionsschritten abge-

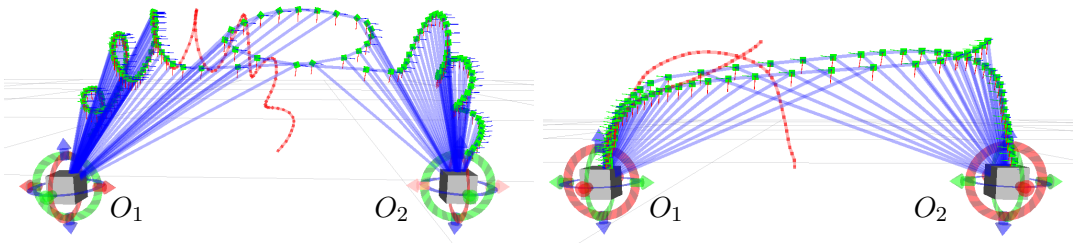
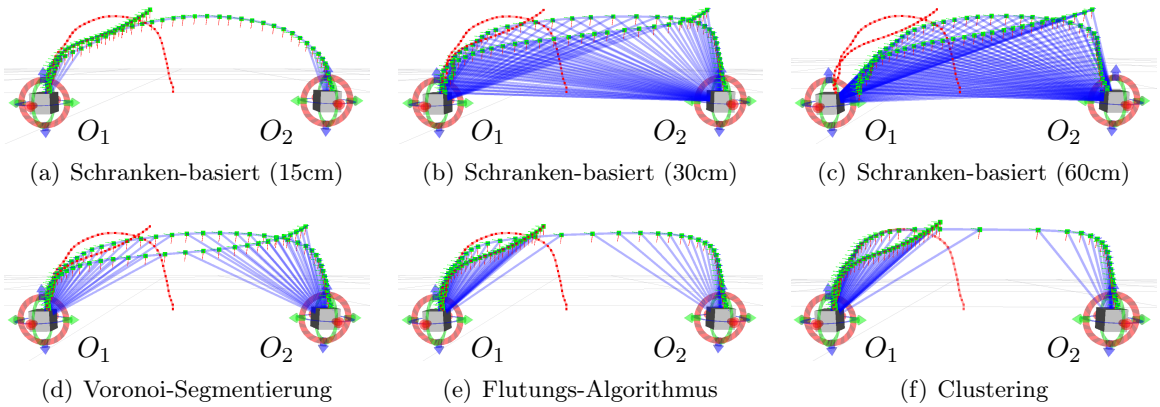


Abbildung 5.19: Adaptionen (grün) zweier Demonstrationen (rot) bei Erzeugung von vier statt zwei Clustern.

brochen, da aufgrund der wenigen Verbindungen die Adaption extrem langsam von statten geht. Das Bild vermittelt den Eindruck, nur die rechte Seite sei gestreckt. Tatsächlich homogenisiert sich die Streckung der Trajektorie mit der Zeit und dehnt sich auf die linke Seite aus. An diesem Beispiel ist sehr gut zu erkennen, dass zu wenige Verbindungen die Adaption schlicht unpraktikabel machen.
Die Methode der Voronoi-Segmentierung (d) erweist sich als sehr effektiv, da sie ähnlich gute Ergebnisse wie eine Flutung und ein Clustering liefert, jedoch mit minimalen Informationen auskommt, einfach umzusetzen ist und für den Benutzer gut nachzuvollziehen.
In diesem Szenario besitzt der Roboter eine gute Startposition für die Demonstration. Im folgenden Szenario soll untersucht werden, welche Verbindungen entstehen, wenn dies nicht mehr der Fall ist. In Abbildung 5.20 ist zu sehen, wie der Benutzer den Roboter zunächst am rechten Objekt vorbeiführt um dann das linke Objekt zu greifen und auf dem rechten abzulegen. In der Reproduktionsphase ist erneut das Objekt O_2 nach rechts verschoben. Die demonstrierte Bahn ist in rot dargestellt. Sowohl bei der Voronoi-Segmentierung (d), als auch



(a) Schranken-basiert (15cm) (b) Schranken-basiert (30cm) (c) Schranken-basiert (60cm)

(d) Voronoi-Segmentierung (e) Flutungs-Algorithmus (f) Clustering

Abbildung 5.20: Vergleich der Adaption (grün) einer Demonstration (rot) ausgehend von unterschiedlichen Verbindungen.

bei zwei der Schranken-basierten Methoden (b, c) wird das Annähern an O_1 tatsächlich O_2 zugeordnet. Bei (a) geschieht dies nicht, da die Schranke sehr klein gewählt ist. Hier besteht jedoch erneut das Problem, dass die Adaption sehr viel Zeit benötigt. Bei den Varianten mit Flutung (e) und Clustering (f) funktioniert die Zuordnung wie gewünscht - selbst bei der sehr einfachen Implementierung, bei der alle Behälter gleich groß sind. Da die Implementierung des Clustering Zufallskomponenten beinhaltet, kann dieser sowohl günstige als auch ungünstige Ergebnisse wie in Abbildung 5.19 erzeugen, die der schranken-basierten Zuordnungen ähnelt. Insgesamt kann festgestellt werden, dass die einfache Voronoi-Segmentierung oft eine sehr gute Lösung liefert. Ist eine höhere Anzahl an Objekten präsent oder ist die Demonstration nicht optimal, kann die Flutung eine zufriedenstellende Lösung liefern. Das Clustering kann gute Ergebnisse erzielen, sofern die Anzahl der Cluster korrekt gewählt wird, jedoch können aufgrund der Zufallskomponente auch schlechte Ergebnisse entstehen. Daher ist das Clustering in dieser Form nur wenig geeignet. Das Problem bei allen Verfahren ist, dass dem System nicht mehr Informationen zur Verfügung stehen, als auf den Abbildungen zu sehen ist. Die Entscheidung welcher Teile einer Trajektorie sich welchem Objekt zuordnen lassen, kann nicht sicher getroffen werden. Selbst für einen Menschen ist zunächst nicht klar, welche der Adaptionen aus Abbildung 5.20 nun eine korrekt Lösung liefert. Erst durch weitere Informationen, wie beispielsweise dem Kontext der Aufgabe, kann er dies sicher entscheiden.
Die Heuristiken zeigen jedoch für die gegebenen Aufgaben gute Ergebnisse. Zur Erhöhung der Qualität müssen entweder klare Vorgaben an den Programmierer während der Demonstrationsphase gemacht werden oder er muss zusätzliche Informationen bereitstellen. Dies wird im Ausblick (Kapitel 6) weiter erläutert.

5.3.3 Adaptionsergebnisse

In diesem Kapitel sollen die Adaptionsergebnisse für eine größere Anzahl an Demonstrationen qualitativ evaluiert werden. Dazu werden zunächst einige rein virtuelle Adaptionen durchgeführt, bei denen eine Demonstration mit einem Objekt an verschiedene Objektpositionen und Orientierungen in der Simulation adaptiert wird. Anschließend werden typische Aufgaben aus den zu Beginn dieses Kapitels genannten Domänen demonstriert und adaptiert.

Virtuelle Experimente

In diesem Kapitel werden einige Demonstrationen an geänderte Randbedingungen adaptiert und anschließend qualitativ evaluiert. Bei der ersten Demonstration handelt es sich um eine Bogenbewegung zwischen zwei Objekten, wobei mit dem TCP ein Pfeil auf dem zweiten Objekt gezeichnet wird. Nun wird die Position und Orientierung des Zielobjektes willkürlich verändert, so dass das System die Trajektorie an die geänderten Randbedingungen adaptiert. Die Verbindungen entstanden hier durch die Voronoi-Zerlegung. Die Ergebnisse sind in Abbildung 5.21 zu sehen. Es ist zu erkennen, dass der Bogen entsprechend adaptiert wird um die Positionsänderung zu kompensieren, die Bewegung auf dem Objekt jedoch relativ zum Objekt bestehen bleibt. Die Trajektorie an sich behält auch ihre charakteristische Form, sofern die Änderungen nicht zu stark sind und kann vom Roboter ausgeführt werden, sofern die Inverse Kinematik eine Lösung findet.

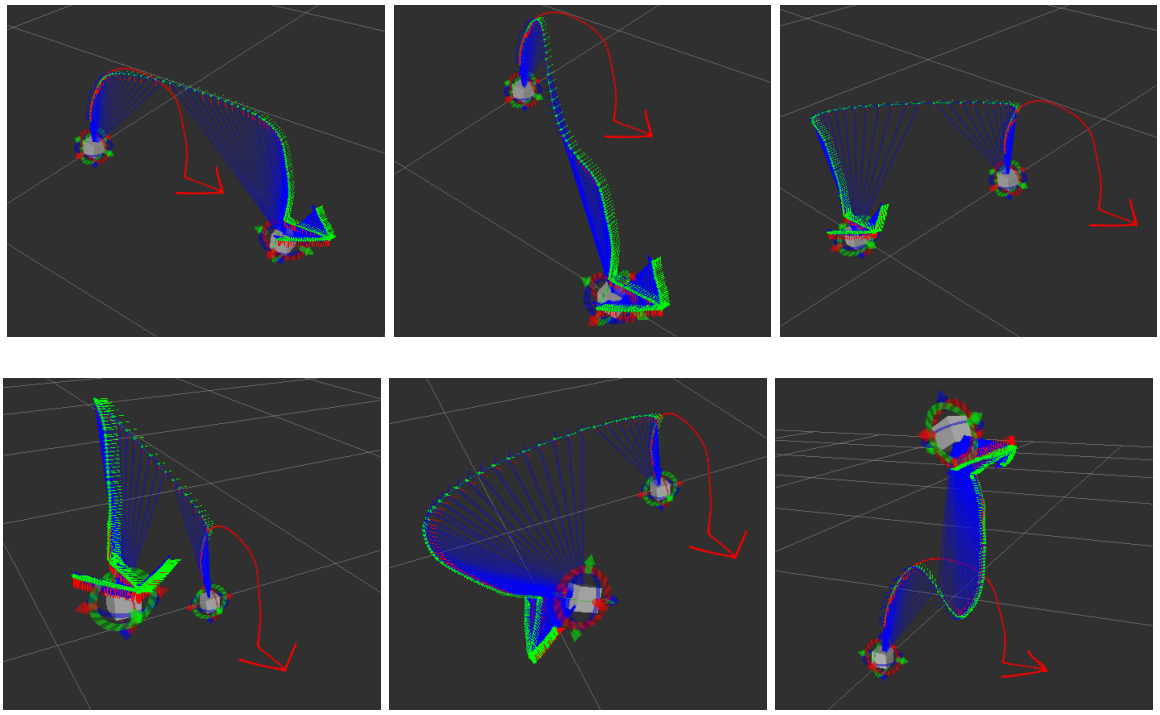


Abbildung 5.21: Adaption (grün) einer Demonstration (rot) an unterschiedliche Posen des Zielobjektes. Während die Bogenbewegung adaptiert wird, bleibt die Pfeilbewegung auf dem Objekt intakt.

Die zweite Demonstration besteht aus zwei aufeinanderfolgenden Pick-and-Place Anwendungen, die in einer einzigen Demonstration gezeigt werden. Hier soll die Fähigkeit ermittelt werden, auch bei mehreren Objekten zu adaptieren. In Abbildung 5.22 ist zu sehen, wie einige Objekte nur verschoben sind (b) bzw. zusätzlich die Orientierung verändert wurde (c). Die Verbindungen entstanden hier durch den Flutungs-Algorithmus. Dies ist deutlich am langen Anfangsstück zu erkennen, das nur dem ersten Objekt zugeordnet wird. Bei einer Voronoi-Segmentierung kommt es hier zu fehlerhaften Zuordnungen. Auch hier wird die Pick-and-Place Bewegungen entsprechend den veränderten Objektpositionen angepasst, so dass sie erfolgreich

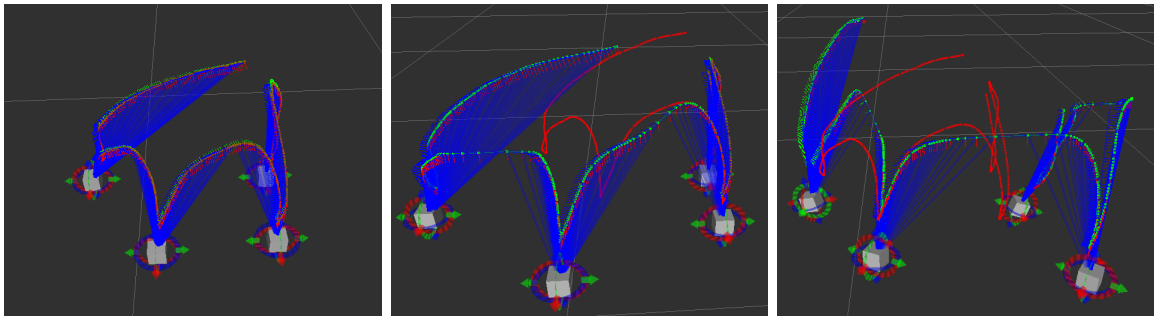


Abbildung 5.22: Adaption (grün) einer Demonstration (rot) zweier aufeinanderfolgender Pick-and-Place Operationen. Auch bei Änderung mehrerer Objektpositionen und Orientierungen ist das System in der Lage die Demonstration erfolgreich zu adaptieren.

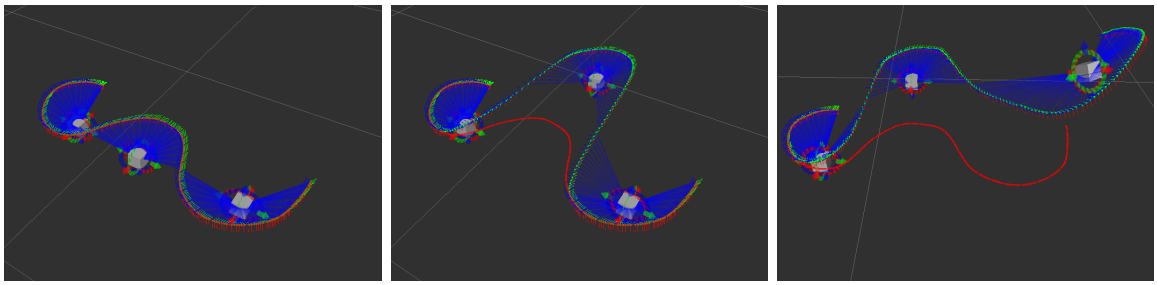


Abbildung 5.23: Adaption (grün) einer Slalom-Demonstration (rot). Auch bei starker Änderung der Objektposen wird die Trajektorie so adaptiert, dass die Slalombewegung noch zu erkennen ist.

ausgeführt werden kann, jedoch ohne die charakteristische Form der Demonstration zu verlieren.
Im dritten Beispiel wird eine Slalom-artige Bewegung um die Objekte demonstriert (Abbildung 5.23). Bei Veränderung des mittleren Objektes wird die Bewegung entsprechend weit gestreckt. Auch bei willkürlich veränderter Position und Orientierung adaptiert sich die Trajektorie entsprechend mit, so dass die Slalombewegung stets zu erkennen ist. Die Verbindungen erfolgten hier ebenfalls durch die Voronoi-Segmentierung.
Das System ist in der Lage die Demonstration erfolgreich an geänderte Randbedingungen anzupassen. Kernpunkt ist wie immer die Verbindung der Trajektorie zu den Objekten. Betrachtet der Nutzer ein Objekt als relevant, dass jedoch weiter entfernt ist als ein anderes Objekt, kann dies vom System nur in begrenztem Rahmen erkannt werden (Flutungs-Algorithmus). Die Voronoi-Variante eignet sich bei den meisten, eher einfachen Aufgaben mit wenigen Objekten. Sind viele Objekte im Arbeitsraum präsent, steigt die Wahrscheinlichkeit einer Falschzuordnung. Zu diesem Zweck sollte der Nutzer nach der Programmierung immer eine Rückmeldung erhalten, wie die Trajektorie mit den Objekten verbunden wurde. Hierfür könnten Augmented Reality Methoden eingesetzt werden, die dem Benutzer erlauben in einer Simulation die Objekte zu bewegen und die Verbindung und Adaption somit online zu überprüfen. Ist das Ergebnis nicht zufriedenstellen, muss der Aufbau der Demonstration geändert werden.

Domänenspezifische Experimente

In diesem Kapitel soll die Adaption einiger Beispielszenarien aus dem Haushalt gezeigt werden. Im ersten Beispiel soll dem Roboter beigebracht werden Kleidungsstücke zu bügeln. Aus technischen Gründen sind Marker auf dem Bügeleisen und auf dem Bügelbrett statt auf dem Kleidungsstück selbst angebracht (Abbildung 5.24). Über diese werden die Positionen und Orientierungen erkannt. Für eine Erhöhung des Technologie-Reifegrades müsste stattdessen das Kleidungsstück selbst erkannt werden.
Im ersten Teilbild von Abbildung 5.25 ist die Adaption der Trajektorie dargestellt. Sowohl das Bügelbrett als auch das Bügeleisen haben die Position und Orientierung geändert. In den weiteren Teilbildern ist die Ausführung der Adaption zu erkennen.

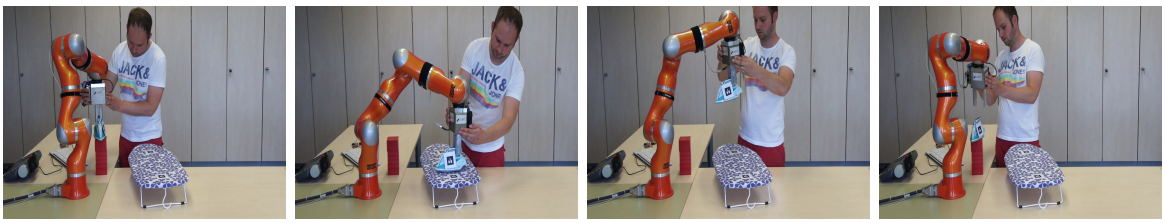


Abbildung 5.24: Zur Demonstration der Aufgabe „Bügeln“ wird zunächst ein Bügeleisen gegriffen, anschließend damit mehrfach auf dem Bügelbrett hin und her bewegt und letztendlich wieder abgelegt.

Abbildung 5.25: Adaption und Ausführung der Aufgabe „Bügeln“. Sowohl Bügelbrett als auch Bügeleisen haben die Position und Orientierung geändert. Das System adaptiert die Demonstration und Reproduziert die Aufgabe korrekt mit dem Roboter. Ein Video ist auf der Seite ai3.uni-bayreuth.de/projects/introp verfügbar.

Die Adaption und Reproduktion funktioniert auch hier sehr gut. Die Verbindungen wurden durch den Flutungs-Algorithmus erzeugt. Die geänderten Positionen und Orientierungen werden berücksichtigt und auch die typische demonstrierte Bügelbewegung beibehalten. Natürlich wäre es in diesem Beispiel sinnvoll, die Ausführung um eine kraft-basiert Komponente zu erweitern. Die Integration von Kräften ist prinzipiell möglich und wird kurz im Ausblick angesprochen. Auch die bereits angesprochene Erkennung des Kleidungsstückes selbst wäre hilfreich. In diesem Zusammenhang müsste für jeden Typ von Kleidungsstück jeweils eine Demonstration gegeben werden. Die weitergehende Adaption an veränderte Objekte wird im Ausblick diskutiert.
Im zweiten Beispiel soll der Roboter als Frühstückshilfe fungieren (Abbildung 5.26).

Abbildung 5.26: Zur Demonstration der Aufgabe wird dem Roboter gezeigt einen Toast zu greifen und anschließend auf einem Teller anzulegen.

Dem Roboter wird beigebracht, einen Toast zu greifen und auf einem Teller abzulegen. Bei der Reproduktion sind die Positionen und Orientierungen der Objekte verändert. Das System reagiert darauf entsprechend und adaptiert die Demonstration (Abbildung 5.27). Auch hier wurden die Verbindungen mittels Flutungs-Algorithmus generiert.
Der am Toast platzierte Marker kann erkannt werden, sobald die Scheibe Toast ausgeworfen wird. In diesem Moment beginnt die Adaption, welche in diesem Beispiel unmittelbar ausgeführt wird. Zusätzlich wird das zweite Objekt (Teller) hier während der Reproduktion bewegt. Der noch nicht ausgeführte Teil der Trajektorie wird entsprechend adaptiert, so dass der Toast dennoch korrekt abgelegt werden kann.
Die Experimente zeigen die Anwendbarkeit des Ansatzes auch bei komplexeren Aufgaben im Haushalt. In beiden Experimenten lagen die Schwierigkeiten vor allem auf der technischen Seite. Die Erkennung der Objekte ist für eine Adaption unerlässlich und die realisierte Lösung mit Hilfe von Markern ist zwar effizient, jedoch nicht immer fehlerfrei und für den Einsatz im Haushalt nur schwer umzusetzen. Zur Verbesserung des Systems, wäre es sinnvoll die Objekte selbst zu erkennen, indem während der Demonstrationsphase ein Modell der relevanten Objekte erstellt wird, das in der Reproduktionsphase erkannt werden kann.

5.3.4 Vergleich mit menschlicher Intuition

Neben der qualitativen soll in diesem Kapitel auch eine quantitative Evaluation durchgeführt werden. Die Experimente dazu orientieren sich an [Wu10] und basieren auf dem folgenden Prinzip. Mehrere Aufgaben werden unter bestimmten Randbedingungen demonstriert. Anschließend wird jeweils eine Demonstration A an die Randbedingungen einer Demonstration B adaptiert. Nun wird gemessen, wie ähnlich diese adaptierte Demonstration jener ist, die der

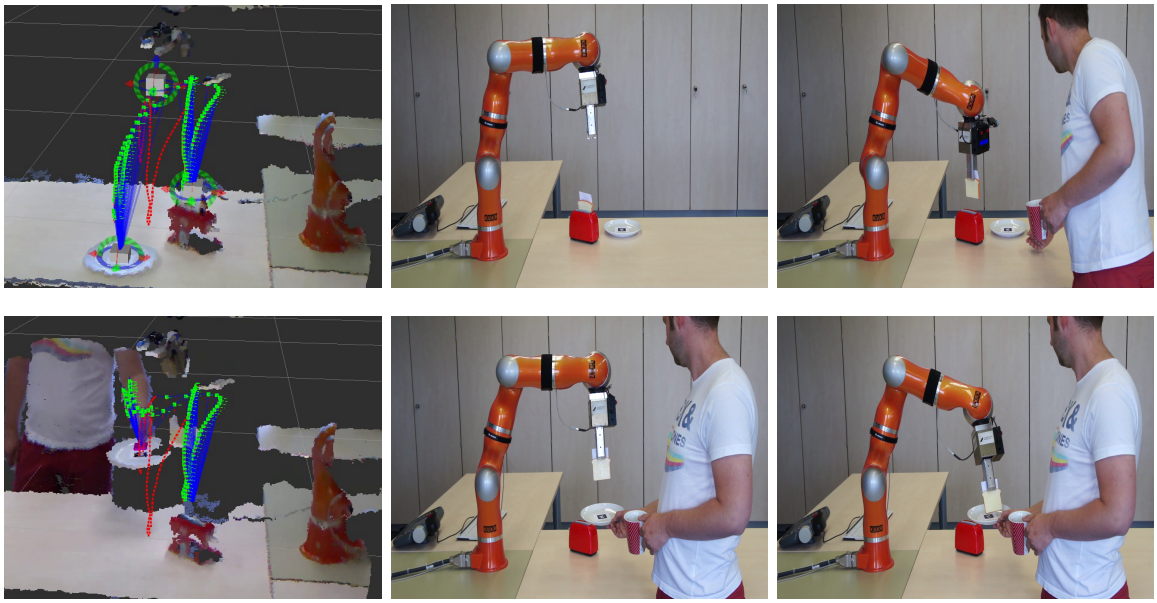


Abbildung 5.27: Die Adaption der Aufgabe wird getriggert, sobald eine Scheibe Toast ausgeworfen wird, da in diesem Moment alle Ressourcen präsent sind. Die Adaption der Trajektorie wird auch während der Reproduktion fortgeführt, so dass der Benutzer die Position des Tellers noch verändern kann.

Benutzer für diese Randbedingungen demonstriert hatte. Man kann davon ausgehen, dass je höher die Ähnlichkeit ist, es umso mehr der Intuition des Benutzers entspricht.
Für die Durchführung werden verschiedene Abstandsmaße benötigt. Analog zu [Wu10] wird die mittlere quadratische Abweichung MSE berechnet zu

$$MSE = \frac{1}{N}\sum_{i=1}^{N} ||p_i^r - p_i|\,|, \tag{5.1}$$

wobei p_i^r die Position des adaptierten Partikels einer Demonstration A darstellt, der an die Randbedingungen einer Demonstration B adaptiert wurde und p_i die Position des demonstrierten Partikels der Demonstration B. Die Berechnung dieses Wertes gibt Auskunft darüber, wie hoch die Abweichung zwischen der adaptierten und der demonstrierten Trajektorie ist.
Ebenso wird der Korrelationskoeffizient R^2 berechnet zu

$$R^2 = \frac{\sum_{i=1}^{N}(p_i - \bar{p}) \cdot (p_i^r - \bar{p}^r)}{\sqrt{(\sum_{i=1}^{N}(p_i - \bar{p})^2) \cdot (\sum_{i=1}^{N}(p_i^r - \bar{p}^r)^2)}} \tag{5.2}$$

wobei $\bar{p}$ und $\bar{p}^r$ die arithmetischen Mittel von p_i und p_i^r darstellen. Der Korrelationskoeffizient beschreibt wie ähnlich die adaptierte Trajektorie der vom Benutzer demonstrierten ist. Beide Metriken setzen voraus, dass die Eingabevektoren die gleiche Länge N besitzen. Um dies zu gewährleisten, wird die kürzere Trajektorie durch Hermite Splines interpoliert und neu abgetastet.
Es existieren fünf Serien S1 - S5, die schematisch in Abbildung 5.28 dargestellt sind. Jede Serie entspricht einem gesonderten Szenario und wird 20 mal demonstriert, so dass insgesamt 100 Demonstrationen entstehen.

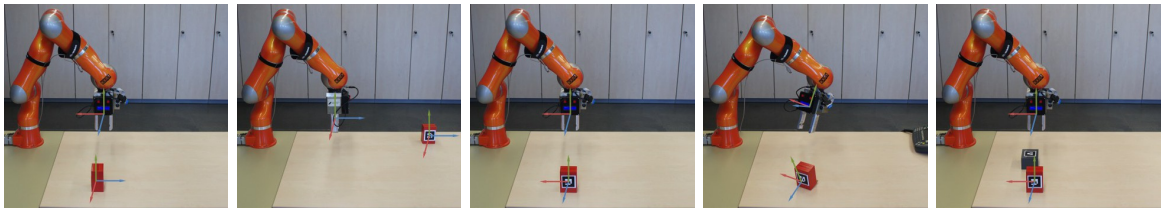


Abbildung 5.28: Schematischer Aufbau der Experimentreihen S1 - S5

- Serie S1 ist eine geradlinige Greifbewegung.
- Serie S2 basiert auf S1, jedoch ist der gesamte Aufbau um 90° in der x-y-Ebene gedreht.
- Serie S3 basiert auf S1, jedoch ist das Ziel um 90° in der x-y-Ebene gedreht.
- Serie S4 ist eine Greifbewegung mit willkürlich gewählten Positionen und Orientierungen des Roboters und des Ziels.
- Serie S5 basiert auf S1, jedoch befindet sich ein Hindernis auf dem Weg zum Ziel.

Es werden jeweils paarweise Vergleiche zwischen einer Demonstration und jeder möglichen Adaption gemacht. Hierfür werden der Zustand der Ressourcen $\mathcal{R}_{\text{input}}$, wie Objekte und Startkonfiguration des Roboters einer Demonstration *input* verwendet und eine weitere Demonstration an diese adaptiert (*output*). Dies wird erreicht, indem die Objekte $\mathcal{R}_{\text{input}}$ von *input* als Objekte $\mathcal{R}'$ für die Reproduktion von *output* verwendet werden

$$\mathcal{R}'_{\text{output}} = \mathcal{R}_{\text{input}}. \tag{5.3}$$

Danach wird die reproduzierte Trajektorie mit der durch den Benutzer demonstrierten verglichen, indem die mittlere quadratische Abweichung MSE und der Korrelationskoeffizient R^2 berechnet werden.
Bei dieser Art der Evaluation existieren 100 Fälle, bei denen eine Demonstration sowohl als *input* als auch als *output* dient. Bei diesen sind also die Randbedingungen für Demonstration und Reproduktion identisch. Hier betragen alle $MSE = 0$ und $R^2 = 1$. Das System verhält sich in diesem Fall wie im Playback Modus. Dies entspricht exakt der Forderung aus Kapitel 4.4, da es genau die vom Benutzer gewünschten Trajektorie darstellt.
In Tabelle 5.2 und 5.3 sind die arithmetischen Mittel der Evaluierungsmetriken für jede Gruppe von *input-output* Serien abgebildet.
Beide Tabellen zeigen gute Werte für den Ansatz. Die MSE ist niedrig, was bedeutet, dass die Abweichungen der Trajektorien gering sind. Hinzu kommt, dass der minimale Wert für den Korrelationskoeffizient bei $R^2 = 0,808$ liegt und der maximale Wert bei $R^2 = 0,994$, sofern man die Werte auf der Diagonalen nicht beachtet. Weiterhin zeigen 84% aller Mittelwerte einen Korrelationskoeffizienten von über $0,9$. Das bedeutet, dass die gesamte Form der Reproduktionen sehr große Ähnlichkeit mit den Demonstrationen besitzt, was für eine gute Qualität der Adaption spricht. Natürlich sind die Werte für S4 und S5 niedriger als die von S1 bis S3. Die Aufgabe ist in diesen Fällen komplizierter und daher unterscheiden sich bereits die Demonstrationen stärker voneinander und damit auch die Reproduktionen.
Es ist zu erkennen, dass der Algorithmus nicht nur in der Lage ist eine Demonstration an eine

Tabelle 5.2: Arithmetische Mittel der mittleren quadratischen Abweichung MSE für alle *input-output* Kombinationen von S1 bis S5

	input				
output	S1	S2	S3	S4	S5
S1	0,000921	0,000532	0,00293	0,00316	0,0118
S2	0,00064	0,000212	0,00475	0,00322	0,0141
S3	0,00371	0,00495	0,00155	0,00592	0,00784
S4	0,00444	0,00127	0,00655	0,00109	0,016
S5	0,0123	0,00663	0,00555	0,0143	0,00116

Tabelle 5.3: Arithmetische Mittel des Korrelationskoeffizienten R^2 für alle *input-output* Kombinationen von S1 bis S5

	input				
output	S1	S2	S3	S4	S5
S1	0,995	0,988	0,938	0,968	0,926
S2	0,994	0,998	0,911	0,962	0,905
S3	0,955	0,808	0,974	0,918	0,961
S4	0,96	0,967	0,901	0,989	0,892
S5	0,928	0,817	0,956	0,887	0,987

neue Situation zu adaptieren, sondern dabei auch eine hohe Ähnlichkeit zu dem erzeugt, wie es ein Mensch in dieser Situation demonstrieren würde.

5.3.5 Dynamik der Adaption

Wie in Kapitel 4.5.4 erwähnt, kann die kinetische Energie des Systems dazu benutzt werden, das Ende der Adaption automatisch zu ermitteln. In diesem Kapitel sollen diese dynamischen Eigenschaften des Systems näher betrachtet werden. Dazu werden verschiedene Demonstrationen an neue Randbedingungen adaptiert und währenddessen die Entwicklung des Energieniveaus des Systems untersucht.
Zur Untersuchung werden verschiedene Demonstrationen an unterschiedlich stark geänderte Randbedingungen in der Simulation adaptiert. Dazu werden die Demonstrationen geladen und zum Zeitpunkt $t_0 = 0$ erhalten ein oder mehrere Objekte ohne Verzögerung eine neue Pose. Die Messung erfolgt ab dem Zeitpunkt t_0. In Abbildung 5.29 ist der typische Ablauf einer Adaption zu sehen. Aufgrund der stärkeren Gewichtung der Verbindungen $\mathcal{L}_\mathrm{R}$ adaptieren sich zunächst sehr schnell die Partikel, die sich nahe den Objekten befinden. Diese ziehen anschließend weiter entfernte Partikel nach und gleichen Ihre Positionen daraufhin auch noch in gewissem Rahmen an, bis eine glatte Bahn entsteht.
Die durchgeführten Experimente sind wie folgt aufgebaut: E1 entspricht einer geringen Än-

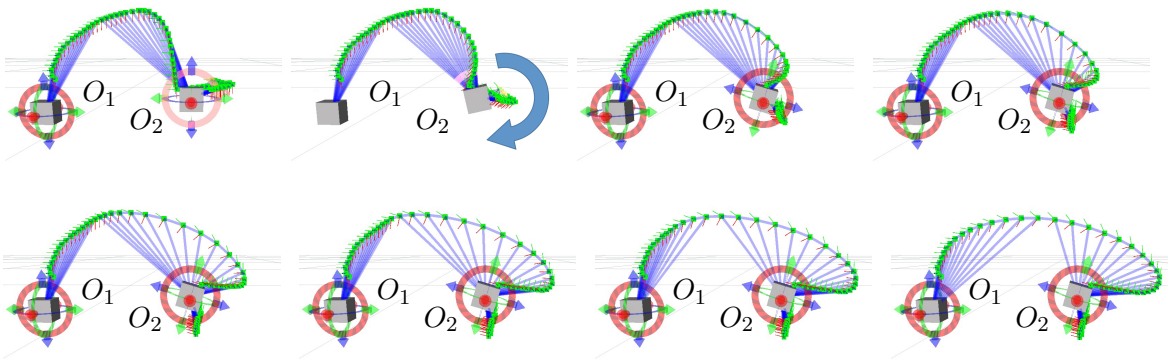


Abbildung 5.29: Eine Trajektorie (erster Bild) wird aufgrund einer Orientierungsänderung des rechten Objektes (zweites Bild) adaptiert. Die Schritte der Adaption zeigen, wie zunächst die Trajektorie nahe der Objekte mitgezogen wird, anschließend erfolgt die Adaption der weiter entfernten Partikel.

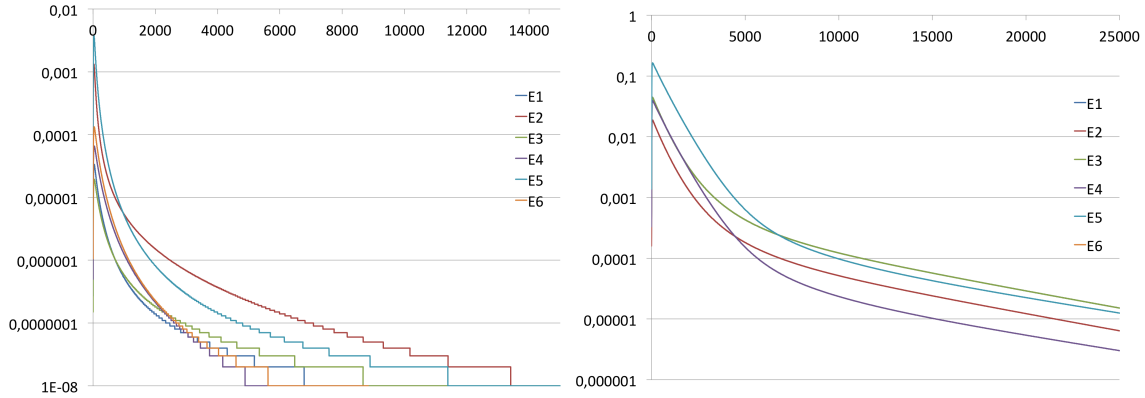


Abbildung 5.30: Links ist der Verlauf der kinetischen Energie aufgrund linearer Bewegung und rechts aufgrund rotatorischer Bewegung aufgetragen. Die horizontale Achse entspricht der Anzahl der durchgeführten Adaptionsschritte und die vertikale Achse (logarithmisch) der gemessenen Energie.

derung von Position und Orientierung zwischen Demonstration und Reproduktion, E2 einer starken Änderung in der Größenordnung des gesamten Arbeitsraumes und E3 lediglich einer linearen Verschiebung eines einzelnen Objektes um ca 30cm, so dass die Trajektorie etwas gestreckt wird. Die Experimente E4 bis E6 entsprechen E1-E3, jedoch mit einer anderen Demonstration. Bei den Demonstrationen handelt es sich um die gleichen wie sie in Abbildung 5.17 und Abbildung 5.20 zu sehen sind.
In Abbildung 5.30 ist die kinetische Energie der Partikel über die Anzahl der bisherigen Adaptionsschritte aufgetragen. Dabei wurde zwischen der translatorischen und rotatorischen Bewegung unterschieden. Beide Darstellungen sind logarithmisch. Es ist zu erkennen, dass die Energieabnahme exponentiell voranschreitet und die kinetische Energie aufgrund der sinkenden Kräfte und Momente stetig fällt, wobei das absolute Maximum natürlich stark von der Anzahl der Partikel und der Veränderung zur Demonstration abhängig ist. Die durchschnittliche Dauer einer Adaption, auch bei komplexen Änderungen, welche die gesamte Trajektorie

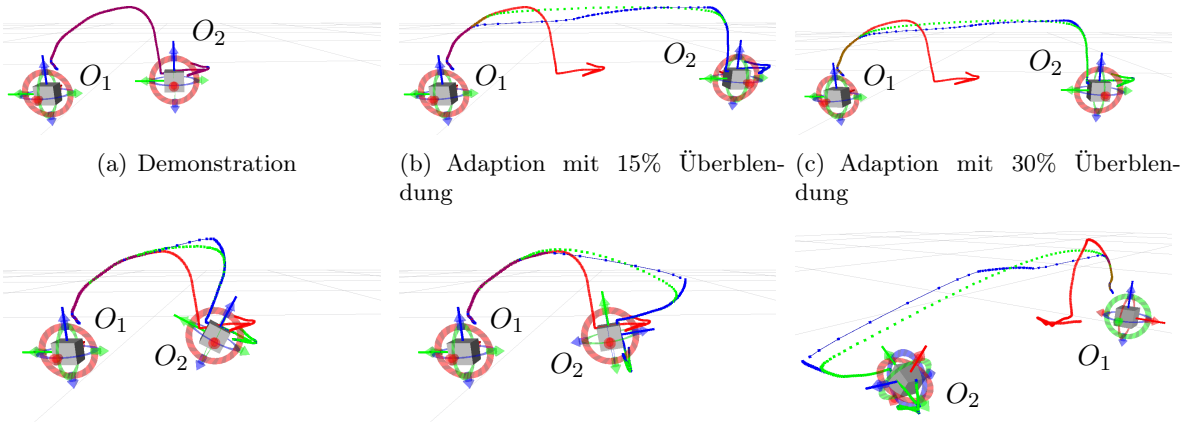
betreffen, dauert nur wenige Sekunden. Dies spiegelt auch die menschliche Wahrnehmung wieder, bei der nach einigen Sekunden nur noch geringe Änderungen wahrnehmbar sind. In Kombination mit der Art der Adaption (Abbildung 5.29) werden im fortgeschrittenen Verlauf der Adaption vor allem Transferbewegungen nachkorrigiert und geglättet. Aufgrund dessen wird auch klar, warum eine online Adaption möglich ist. Da die Energien zunächst extrem hoch sind und rapide fallen, erfolgt ein Großteil der Adaption unmittelbar. Dies ist insbesondere nahe Objekten der Fall. Die zunächst adaptierte Bahn ist nicht energieminimal, besitzt jedoch bereits nach ca. 2000 Schritten die notwendige Genauigkeit für eine Reproduktion.
Die Schranken für eine ausreichende Adaption können bei diesen typischen Applikationen und einem stationären Roboter zu etwa $\epsilon_\mathrm{t} = 10^{-7}$ (Position) und $\epsilon_\mathrm{r} = 10^{-4}$ (Orientierung) gewählt werden, was etwa 10000 Schritten entspricht. Die dafür notwendige Zeit liegt, abhängig von der Anzahl der Verbindungen und der verwendeten Hardware, bei ca. 10 - 15 Sekunden. Das Verfahren selbst eignet sich auch sehr gut für eine Parallelisierung, was jedoch im Rahmen dieser Arbeit nicht umgesetzt oder untersucht wird.

5.4 Intuitive Programmierung mittels Motion Segments

Die Adaption mittels orientierter Partikel liefert gute Ergebnisse bei gleichzeitiger Online-Fähigkeit. Wird aber eine Echtzeitfähigkeit benötigt, kann die in Kapitel 4.6 vorstellte analytische Lösung verwendet werden. In diesem Kapitel soll daher der analytische Ansatz mittels Motion Segments mit den Ergebnissen der Partikelsimulation qualitativ verglichen werden. Anschließend erfolgt ebenfalls eine quantitative Evaluation wie in Kapitel 5.3.4.

5.4.1 Adaptionsergebnisse

Aus Gründen des Umfangs werden die Experimente der Partikelsimulation hier nicht wiederholt. Stattdessen soll hier der Ansatz nur vergleichend untersucht werden. Als Referenz dienen zunächst einfache Demonstrationen, wie sie bereits aus Abbildung 5.17 bekannt sind und anschließend auch komplexere. In Abbildung 5.31 (a) ist zunächst die Ausgangssituation zu sehen. In Abbildung (b) und (c) wird das Objekt O_2 erneut nach rechts verschoben - zunächst mit einem Überblendbereich von 15% anschließend mit 30%. In Abbildung (d) wird das Objekt um einen Winkel von etwa 30° um die x-Achse gedreht und in (e) um ca. 85°. Wie in den Abbildungen deutlich zu erkennen ist, unterscheiden sich die Ergebnisse beider Adaptionsmethoden nur geringfügig voneinander.
Wird der Blendbereich höher gewählt (Abbildung 5.31 (c)), erzeugt das Verfahren eine glattere Trajektorie und ähnelt stärker der Adaption mittels Partikelsimulation, verliert aber zunehmend ihre typischen Feinbewegungen aufgrund der Überblendung der Segmente. Ein deutlicher Unterschied ist auch bei starken Änderungen der Randbedingungen zwischen Demonstration und Reproduktion beobachtbar. Betrachtet man die Situation in Abbildung 5.32 (a), bei der die Objektpositionen gespiegelt sind, jedoch nicht die Orientierungen, fällt auf, dass die Partikelsimulation eine glattere Bewegung erzeugt. Dagegen entsteht bei der Adaption mittels Motion Segments ein deutlich sichtbarer Haken an den Enden der Transferbewegung. Dieser kann abgeschwächt werden, sofern der Blendbereich größer gewählt wird.
Bei der Streckung 5.32 (c) entsteht zwischen den Adaptionen beider Varianten eine Art Phasenverschiebung der Wellenbewegung. Dies liegt daran, dass die Streckung bei der Adaption mittels orientierter Partikel aufgrund der Skalierung der Kräfte stetig abnimmt. Bei der Adaption mittels Motion Segments setzt die Streckung aufgrund der Transfer-Frames sprunghaft



(a) Demonstration

(b) Adaption mit 15% Überblendung

(c) Adaption mit 30% Überblendung

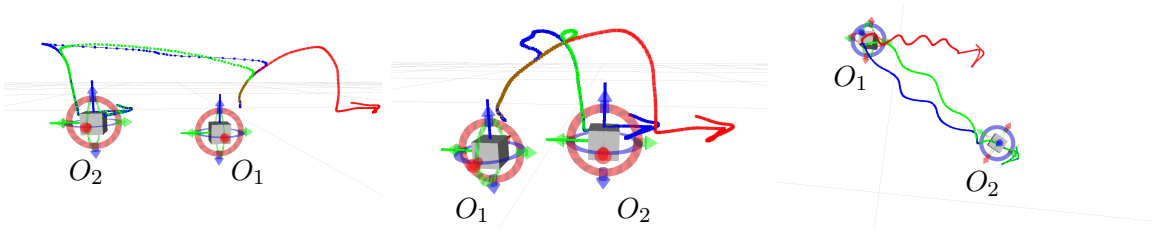
(d) Änderung der Orientierung um 30°

(e) Änderung der Orientierung um 85°

(f) Starke willkürliche Änderung der Objektpose

Abbildung 5.31: Vergleich der Adaptionen mittels orientierter Partikel (grün) und mittels Motion Segments (blau). Die Ergebnisse unterscheiden sich hauptsächlich in der Glattheit der adaptierten Trajektorien.

ein. Noch deutlicher ist der Unterschied bei der Stauchung wahrzunehmen, wie in Abbildung 5.32 (b) zu sehen ist. Während die Adaption mittels Partikelsimulation eine glatte Schleife erzeugt (grün), wird die Trajektorie bei den Motion Segments derart gestaucht, dass zuvor nicht erkennbare Ungleichmäßigkeiten sehr stark hervortreten. Die Bewegungen nahe den Objekten sind jedoch identisch. Es lässt sich feststellen, dass das Verfahren auf geänderte Position der Objekte ebenfalls korrekt reagiert und die Transfer-Frames entsprechend erzeugt werden.
In Abbildung 5.33 sind die Auswirkungen einer fehlenden Normierung der x_T-Achse (Kapitel 4.6.6) zu sehen. Die Demonstration ist in rot, die Adaption durch eine Partikelsimulation in grün und die Adaption mittels Motion Segments in blau aufgetragen. Die Trajektorie beginnt bei Objekt O_1 und nähert sich hier von der linken Seite O_2. Für die Reproduktion wird O_2 nach links verschoben. Bei der Adaption mittels Motion Segments wurde eine Segmentierung mit einer Schranke von $d = 17cm$ vorgenommen, so dass ein Teil der Partikel im Transfer-Frame einen Wert von $x_\mathrm{T} \geq 1$ besitzt. Als Referenz werden die Verbindungen für die Partikelsimulation durch den Flutungs-Algorithmus erstellt, so dass jedes Partikel der Trajektorie einem Objekt zugeordnet ist. Bei der Adaption mittels Partikelsimulation wird der Transferbereich entsprechend gestreckt, die Rückseite des Objektes jedoch bleibt relativ zum Objekt konstant. Dagegen wird bei der Adaption mittels Motion Segments der Bogen entsprechend skaliert, verlängert die Bewegung somit unnötig und verfälscht unter Umständen die demonstrierte Manipulation. Da dies linear geschieht, kann die so skalierte Trajektorie schnell außerhalb des Arbeitsraums verlaufen. Im unteren Bereich von Abbildung 5.33 ist die Adaption mit entsprechender Normierung zu sehen. Hier entspricht das Ergebnis auf der Rückseite des Objektes dem Ergebnis der Partikelsimulation. Insgesamt liefert das Verfahren sehr positive und ähnliche Ergebnisse. Die Bahnen der Partikelsimulation sind an sich glatter, was nachvollziehbar ist, da auch die Kräfte der Trajektorienpartikel untereinander berücksichtigt werden. Der größte Unterschied kann im Transferbereich wahrgenommen werden, denn dort nimmt die Stärke der Verbindungen bei der Partikelsimulation stetig ab, so dass ein fließender Übergang entsteht. Bei der Adaption mittels Motion Segments muss dazu ein entsprechend



(a) Spiegelung der Objektpositionen

(b) Extreme Stauchung

(c) Adaption der Transferbewegung

Abbildung 5.32: Vergleich der Adaptionen einer Demonstration (rot) mittels orientierter Partikel (grün) und mittels Motion Segments (blau).

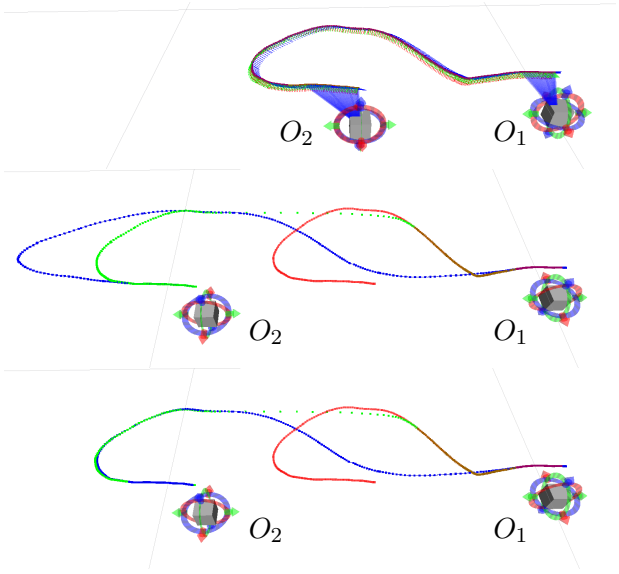


Abbildung 5.33: Vergleich der Adaption einer Demonstration (rot) mittels orientierter Partikel (grün) und Motion Segments (blau). Im ersten Bild ist die Demonstration und die Verbindungen der Partikel für die Adaption mittels Motion Segments zu sehen. Die Partikelsimulation arbeitet als Referenz mit dem Flutungsalgorithmus. Durch die Schranken-basierten Verbindung der Trajektorie ohne die Normierung der x_{T}-Achse entsteht der in der zweiten Abbildung dargestellte Effekt, dass alle Partikel des Transfer-Frames mit $x_{\mathrm{T}} > 1$ in die Länge gezogen werden. Erst durch die Normierung (letztes Bild) wird dies verhindert und die Adaption erfolgt in diesem Bereich wie bei einer Adaption mittels Partikelsimulation.

hoher Blend-Bereich gewählt werden, der jedoch die charakteristische Bewegung verfälschen könnte.

5.4.2 Vergleich mit menschlicher Intuition

Nach der qualitativen Analyse soll in diesem Kapitel die Adaption mittels Motions Segments ebenfalls quantitativ evaluiert werden. Die Experimente sind ähnlich aufgebaut wie in Kapitel 5.3.4, entsprechen jedoch durch abgewandelte Szenarien den Eigenheiten dieses Verfahrens. Es werden dazu zwei verschiedene Aufgaben T1 und T2 betrachtet. Zunächst wird wieder eine einzelne Greif-Aktion T1 untersucht. Da das Verfahren jedoch keine Kollisionsvermeidung beinhaltet, existieren für T1 nur S1 - S4 aus Kapitel 5.3.4.
Die zweite Aufgabe T2 ist eine Pick-and-Place Anwendung, die in drei verschiedenen Serien S5 - S7 demonstriert wird. Eine schematische Abbildung findet sich in Abbildung 5.34. Die Serien zeigen einen etwas komplexeren Task und sind folgendermaßen spezifiziert:

- Serie S5 ist eine Pick-and-Place Aufgabe bei der der Roboter zum Schluss zur Ausgangsposition zurückkehrt.

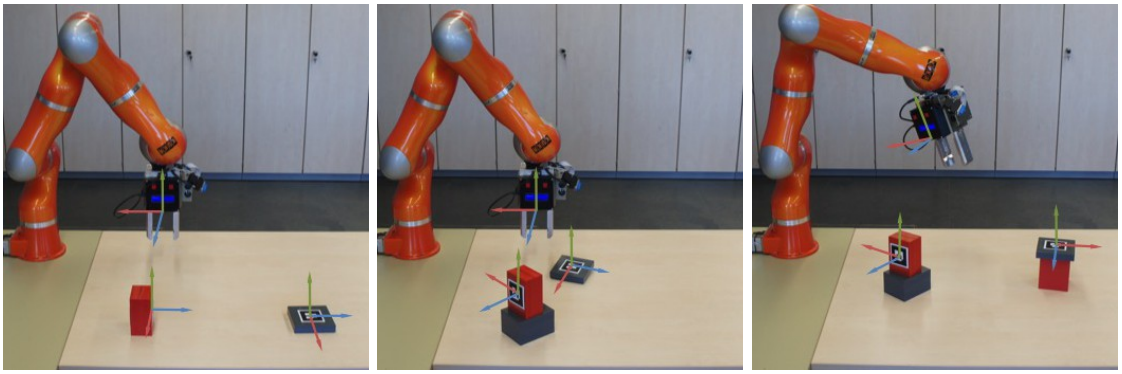


Abbildung 5.34: Schematischer Aufbau der Experimentreihe T2, bei welcher das rote Objekt auf der grauen Platte abgelegt werden soll.

- Serie S6 basiert auf S5, jedoch besitzen die Objekte eine willkürliche Position und sind in der x-y-Ebene verdreht.
- Serie S7 basiert auf S6, jedoch ist zusätzlich die Startposition des Roboters verändert.

Dadurch entstehen insgesamt $80 \cdot 80 + 60 \cdot 60 = 10.000$ paarweise Validierungen. Im Fall der 140 Selbst-Adaptionen sind die Werte wie bei der Partikelsimulation konstant bei $MSD = 0$ und $R^2 = 1$. Dies entspricht auch hier einem Playback, welches exakt so gewünscht ist.
Die Tabellen 5.4 bis 5.7 zeigen die Mittelwerte der Metriken der Gruppen aus *input* und *output*. Auch hier ist eine durchgehend hohe Ähnlichkeit bei den Korrelationskoeffizienten zu erkennen. Die minimale Korrelation liegt bei $R^2 = 0,81$. Insgesamt sind die Werte niedriger als die der Partikelsimulation, jedoch nicht wesentlich. Dies liegt vor allem an den deutlich glatteren Bahnen der Partikelsimulation. In Anbetracht der sofortigen analytischen Lösung, die das System erlaubt, sind die Werte vertretbar.
Bezüglich Task $T2$ sinkt die Korrelation weiter ab. Jedoch zeigen immer noch 78% der Experimente einen Wert von $R^2 \geq 0,7$. Das Problem hier ist die Länge der Aufgabe. Die Demonstrationen unterscheiden sich dadurch deutlich stärker voneinander, als es bei einem einfachen Greifen der Fall ist. Dadurch entstehen sofort niedrigere Werte für R^2 und höhere für MSD. Nichtsdestotrotz ist der Algorithmus in der Lage Demonstrationen mit einem guten Ergebnis an eine neue Situation zu adaptieren. Die Ergebnisse dieses Abschnitts und deren Vergleich mit der Partikelsimulation zeigen, dass die Adaption mittels Partikelsimulation sehr gut für

Tabelle 5.4: Arithmetische Mittel der mittleren quadratischen Abweichung MSD für alle *input-output* Kombinationen von $T1$

	input			
output	S1	S2	S3	S4
S1	0,000238	0,00421	0,00369	0,00324
S2	0,00428	9.89e-05	0,00815	0,00895
S3	0,00405	0,00767	0,000648	0,00385
S4	0,00385	0,00564	0,0045	0,000638

Tabelle 5.5: Arithmetische Mittel des Korrelationskoeffizienten R^2 für alle *input-output* Kombinationen von $T1$

	input			
output	S1	S2	S3	S4
S1	0,996	0,935	0,909	0,904
S2	0,951	0,998	0,811	0,868
S3	0,851	0,923	0,981	0,883
S4	0,892	0,896	0,909	0,974

Tabelle 5.6: Arithmetische Mittel der mittleren quadratischen Abweichung MSD für alle *input-output* Kombinationen von $T2$

	input		
output	S5	S6	S7
S5	0,00202	0,017	0,0277
S6	0,0168	0,00243	0,0102
S7	0,0267	0,0101	0,00172

die Adaption an stark veränderte Randbedingungen geeignet ist. Die Adaption mittels Motions Segments eignet sich vor allem für kleinere Abweichungen. Dies ist beispielsweise in der Industrie nützlich, damit Werkstücke von einem Arbeiter nicht exakt positioniert werden müssen, um so Zeit zu sparen. Dies wird durch die sofortige analytische Lösung noch begünstigt. Eine Evaluierung von T1 nach T2 ist nicht möglich, da die Anzahl der involvierten Objekte unterschiedlich ist und damit eine Adaption durch den Algorithmus nicht möglich ist.

Tabelle 5.7: Arithmetische Mittel des Korrelationskoeffizienten R^2 für alle *input-output* Kombinationen von $T2$

	input		
output	S5	S6	S7
S5	0,965	0,733	0,51
S6	0,789	0,954	0,784
S7	0,588	0,811	0,968

5.5 Einschränkungen

Beide Verfahren stellen lediglich Heuristiken dar und unterliegen einigen Einschränkungen bei der Adaption an geänderte Randbedingungen, die in Kapitel 5.5.1 aufgezeigt werden. Anschließend werden zusätzliche Einschränkungen der Motion Segments in Kapitel 5.5.2 diskutiert.

5.5.1 Einschränkungen der Single-Shot Verfahren

Bei beiden Verfahren besteht das Problem der Zuordnung. Aufgrund der Tatsache, dass nur eine einzige Demonstration gegeben ist, kann ohne weitere Anwendungsinformationen nicht sicher ermittelt werden, welche Teile einer Demonstration sich welchem Objekt zuordnen lassen. Dazu betrachtet man folgendes Gedankenexperiment, das in Abbildung 5.35 dargestellt ist. Eine Trajektorie verläuft zwischen zwei Objekten mit den Frames k_1 und k_2. Bei der Reproduktion sind beide Objekte weiter voneinander entfernt. Es existieren nun vier Möglichkeiten der Zuordnung, sofern nicht bekannt ist, welche Bedeutung die Trajektorie besitzt.

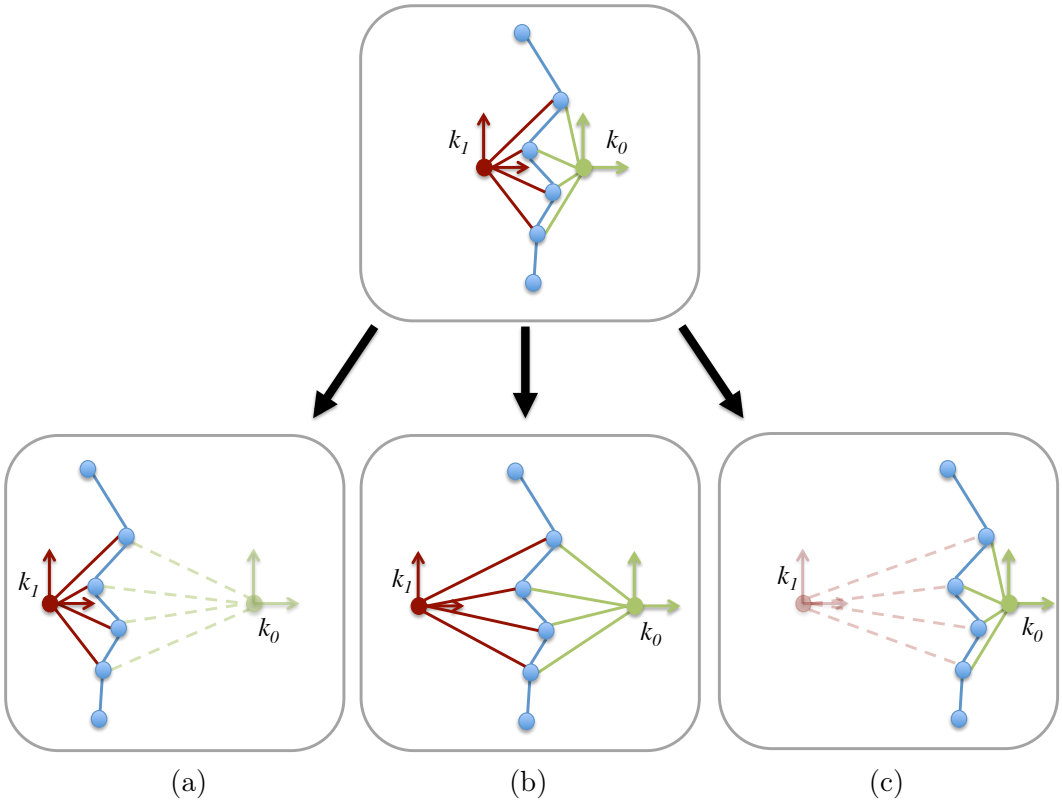


Abbildung 5.35: Je nach Art der Zuordnung der Trajektorie zu den Objekten, können verschiedene Adaptionen (untere Reihe) einer Demonstration (oben) erfolgen.

Abgebildet sind hier nur drei der vier Möglichkeiten, da die vierte mit Abbildung 5.35 (b) übereinstimmt. Bei der Zuordnung zum Weltkoordinatensystem würde nämlich die Trajektorie unabhängig von allen Objekten immer konstant bleiben.
Die abgebildeten Lösungen entsprechen einem Umbiegen der Trajektorie bei dem diese entweder dem linken Objekt zugeordnet werden (a), dem rechten (c) oder beiden (b). Erlaubt man zusätzlich, dass auch Bewegungen hinzugefügt werden dürfen, wäre zudem eine Serialisierung

der Bewegung möglich. In diesem Fall wird zunächst Lösung (a) und anschließend Lösung (c) ausgeführt, so dass für beide Objekte die Bewegung reproduziert wird - nur eben zeitversetzt. Wie auch bereits an anderer Stelle erwähnt, ist eine Zuordnung mit nur einer einzigen Demonstration nicht sicher entscheidbar. Abhilfe kann nur die Integration weiterer Informationen aus zusätzlichen Quellen [Iba05, Stei04, Mcgu02] oder zusätzlichen Demonstrationen [Niek15] schaffen.

5.5.2 Einschränkungen der Motion Segments

Bei der Adaption können sich bestimmte Konstellationen der Objekte als schwierig herausstellen. Insbesondere gilt dies, wenn beispielsweise die Orientierung der Objekte stark verändert wird, aber deren Position gleich bleibt. Als Beispiel dient der in Abbildung 5.36 gezeigte Fall. Die Trajektorie des Transfersegments verläuft nun direkt durch Objekt O_2 hindurch, was bei der Reproduktion zu einer Kollision des Roboters mit dem Objekt führt. Eine Kollision erfolgt

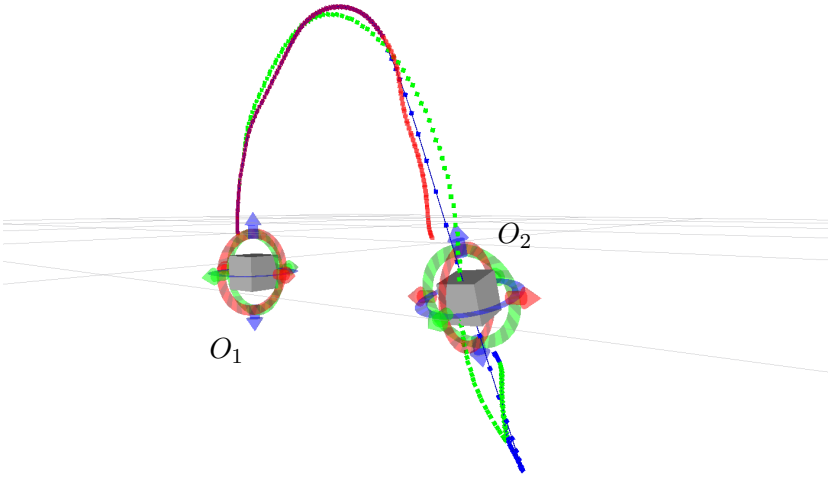


Abbildung 5.36: Kollision der Trajektorie mit einem der Objekte aufgrund der Adaption durch beide Verfahren bei Adapion der Demonstration (rot) durch Motion Segments (blau) und durch eine Partikelsimulation (grün).

in diesem Fall sowohl durch die Adaption mittels Partikelsimulation als auch mittels Motion Segments. Bei einer Adaption mittels Partikelsimulation kann sie jedoch unter Umständen verhindert werden, indem die Partikel noch die abstoßende Kraft F_c aus Kapitel 4.5.2 durch das Objekt O_2 erfahren. Bei der Adaption mittels Motion Segments ist dies nur mit erheblichem Mehraufwand und einer Erweiterung des Ansatzes möglich.
Eine weitere Einschränkung betrifft die Art der Verbindungen. Während bei der Adaption mittels orientierter Partikel sich ein Partikel prinzipiell auf unbegrenzt viele Objekte beziehen kann, ist die Anzahl bei den Motion Segments beschränkt auf eines bzw. zwei im Falle einer Transferbewegung.

5.6 Zusammenfassung

In diesem Kapitel wurden die zur Erreichung von Z1 und Z2 entwickelten Methoden und Ansätze qualitativ und quantitativ evaluiert. Hierfür wurden zunächst die Mechanismen der

Prozess-basierten Ausführung von Verhalten an verschiedenen Beispielen untersucht und anschließend auch eine Untersuchung der Laufzeiten verschiedener Scheduling-Verfahren durchgeführt. Dabei zeigte sich, dass insbesondere das SRN-Scheduling sowohl für den Haushalt als auch die industrielle Anwendung gut geeignet ist.
Im Bereich der intuitiven Programmierung wurde zunächst die Funktionsweise der Adaption mittels orientierter Partikel anhand von virtuellen Experimenten veranschaulicht. Anschließend wurde der Einfluss der Verbindungen demonstriert und eine Abschätzung für ein Abbruchkriterium der Adaption gefunden. Hierbei konnte festgestellt werden, dass die Adaption und Verbindungsgenerierung gute Ergebnisse erzielen. Insbesondere gilt dies, wenn der Benutzer bei der Demonstration darauf achtet, die involvierten Objekte mit ausreichend Abstand zu platzieren. In weiteren Versuchen wurde das Verfahren für Szenarien des Haushalts positiv getestet und es konnte eine hohe Ähnlichkeit zwischen den adaptierten Trajektorien und denen, die vom Benutzer programmiert wurden, festgestellt werden.
Daneben wurde noch die analytische Näherung der Motion Segments mit den Ergebnissen der Partikelsimulation verglichen. Dabei zeigte sich, dass die Ergebnisse der Adaption ähnlich waren wie die der Partikelsimulation, sofern die Voraussetzungen für das Verfahren eingehalten wurden. Auch hier wurde ein Vergleich der adaptierten Trajektorien mit solchen durchgeführt, die vom Benutzer programmiert wurden. Die Werte hierfür waren insgesamt niedriger, jedoch noch auf akzeptablen Niveau.
Zuletzt wurden noch die Einschränkungen der Verfahren aufgezeigt und auf das allgemeine Problem des Single-Shot Programmieren durch Vormachen eingegangen.
Insgesamt erzeugten alle Methoden in vielerlei Hinsicht zufriedenstellende Ergebnisse. Weiterhin zeigte sich, dass die Forderungen, die jeweils zu Beginn von Kapitel 2 und Kapitel 4 gestellt wurden durch die entwickelten Ansätze erfüllt werden.

6 Zusammenfassung und Ausblick

Im Folgenden wird zunächst eine Zusammenfassung der Arbeit gegeben (Kapitel 6.1) und anschließend verschiedene Ansätze zur Erweiterung und Optimierung der darin vorgestellten Verfahren aufgezeigt (Kapitel 6.2).

6.1 Zusammenfassung

Die Verbreitung von Robotern ist im Bereich der vollautomatisierten Großindustrie weit fortgeschritten. Im Bereich kleiner und mittelständischer Unternehmen sowie im Haushalt sind insbesondere Industrieroboter kaum anzutreffen. Einer der Hauptgründe hierfür liegt in der komplexen und langwierigen Programmierung der Roboter, die nur von Experten durchgeführt werden kann. Es besteht daher Forschungsbedarf an intuitiven und transparenten Programmiermethoden für Industrieroboter, um so langfristig deren Akzeptanz und Verbreitung zu steigern.
In dieser Arbeit wurden zwei Ziele verfolgt. Zum einen wurde ein Verfahren entwickelt, dass eine intuitive Programmierung eines Industrieroboters erlaubt, so dass dies auch durch Nicht-Experten möglich ist. Zum anderen wurde ein Verfahren vorgestellt, das die Häufigkeit der Neuprogrammierung senkt. Dies reduziert die zeitliche Belastung für den Programmierer und erhöht die Wirtschaftlichkeit für kleine und mittelständische Unternehmen.
Zur Reduzierung der Programmierhäufigkeit, wurde eine verhaltensbasierte Architektur entwickelt, die jede neu erlernte Aufgabe als Verhalten speichert[Grot13, Grot14c]. Diese Verhalten können immer dann ausgeführt werden, wenn die aktuelle Situation alle oder einen Teil der notwendigen Ressourcen, wie beispielsweise die benötigten Objekte, bereitstellt. Um zu überprüfen, ob die verfügbaren Ressourcen den benötigten genügen, wurden verschiedene Matching-Verfahren vorgestellt. Diese Ressourcen können zudem zur Synchronisation mehrerer Verhalten genutzt werden, indem entsprechende Verfahren entwickelt wurden. Durch die Ausführung der Verhalten als Betriebssystem-ähnliche Prozesse können mehrere Verhalten mit dem Roboter quasi-parallel ausgeführt werden. Zudem können diese jederzeit online gewechselt, unterbrochen und konsistent wiederaufgenommen werden. Um eine größere Menge von Verhalten effizient auszuführen, wurden verschiedene Scheduling-Verfahren für die Verwendung mit Industrierobotern umgesetzt.
Zur intuitiven Programmierung von Aufgaben für Industrieroboter konnten zwei Methoden entwickelt werden [Grot14a, Grot14b]. Beide Verfahren zeichnen sich dadurch aus, dass der Programmierer lediglich eine einzige Demonstration der Aufgabe bereitstellen muss, indem er den Roboter durch die Aufgabe führt. Im Gegensatz zur Playback Programmierung verwenden die Verfahren Sensorinformationen der Umwelt und ermöglichen die erfolgreiche Ausführung der Aufgabe selbst unter geänderten Randbedingungen, wie beispielsweise veränderten Positionen und Orientierungen der verwendeten Objekte oder einer veränderten Startposition des Roboters. Dies wird ermöglicht, indem zunächst ermittelt wird, welche Objekte für die Aufgabe relevant sind und welche Teile der Demonstration welchen Objekten zugeordnet werden können. Mit diesen Informationen ist es möglich, die demonstrierte Trajektorie

an geänderte Randbedingungen zu adaptieren. Dies wurde zum einen über ein numerisches Verfahren erreicht, das die Trajektorie als Kette von Partikeln betrachtet. Diese können mit den Objekten der Umwelt auf unterschiedliche Weise verbunden sein. Die demonstrierte Trajektorie befindet sich zunächst mit der Umwelt im Gleichgewicht. Verändert sich jedoch die Umwelt, indem Objekte beispielsweise eine neue Position annehmen, erfahren die mit den Objekten verbundenen Partikel Kräfte und Momente. Zudem wirken Kräfte und Momente zwischen den Partikeln der Trajektorie, um die Form dieser aufrecht zu erhalten. Die Kombination dieser Einflüsse ruft Positionsänderungen hervor und es entsteht ein Wechselspiel, welches einen Energie-minimalen Zustand des Systems anstrebt. Dieser Zustand kann durch ein Runge-Kutta Verfahren ermittelt werden und entspricht der Adaption der Trajektorie an eine veränderte Umwelt. Ein Vorteil dieser Modellierung ist unter anderem die integrierte Vermeidung von Kollisionen. Die so adaptierte Trajektorie kann anschließend vom Roboter ausgeführt werden, um die Aufgabe in einer neuen Situation zu erfüllen. Zusätzlich konnte ein analytisches Verfahren entwickelt werden, das die typische Task-Struktur nutzt, welche aus Manipulations- und Transferbewegungen besteht. Nachdem die Trajektorie in Transferbewegungen und objektbezogene Bewegungen segmentiert wurde, können die objektbezogenen Segmente an die neue Situation adaptiert werden, indem sie entsprechend ihrem Bezugssystem transformiert werden. Anschließend können analog die Transferbewegungen adaptiert werden. Diese segmentweise adaptierte Trajektorie kann nach einer Glättung an den Schnittbereichen ebenfalls vom Roboter ausgeführt werden, um die Aufgabe unter geänderten Randbedingungen auszuführen.
Beide Ansätze erleichtern die Programmierung von Industrierobotern für den Benutzer, indem sie zum einen die Komplexität einer Programmierung senken und zum anderen die Häufigkeit der Programmierungen reduzieren. Zur Evaluation beider Ansätze wurden verschiedene simulierte und Realwelt-Experimente mit positivem Ergebnis durchgeführt. Die Experimente beinhalteten dabei verschiedene qualitative sowie quantitative Untersuchungen. Bei der Programmierung konnte zusätzlich festgestellt werden, dass eine durch die Adaption erzeugte Trajektorie hohe Ähnlichkeit mit einer durch den Benutzer demonstrierten Trajektorien besitzt, was auf eine hohe Qualität der Adaption hinweist. Des Weiteren zeigten die Experimente, dass trotz der geringen Informationen aus lediglich einer Demonstration, die Adaption für eine Vielzahl von Aufgaben ausreichend gute Ergebnisse liefert.
Insgesamt lässt sich feststellen, dass das One-Shot Programmieren durch Vormachen auf Trajektorienebene eine schnelle und intuitive Möglichkeit bietet, einen Roboterarm auch ohne Expertenkenntnisse zu programmieren. Diese Programmiermethode in Kombination mit einer verhaltensbasierten Reproduktion ermöglicht zudem die differenzierte Reaktion auf unterschiedliche Umweltsituationen und die quasiparallele Bearbeitung mehrerer Aufgaben ohne Neuprogrammierung.

6.2 Ausblick

Die vorgestellten Ansätze zeigten in den Experimenten bereits gute Ergebnisse. Dennoch existiert an unterschiedlichen Stellen Potential zur Verbesserung der Ansätze. Im Folgenden werden zunächst mögliche Optimierungen beschrieben und anschließend Vorschläge für Erweiterungen gegeben.

6.2.1 Fehlererkennung in der Reproduktionsphase

In Kapitel 4.3 wurde angenommen, dass die Reproduktion einer adaptierten Demonstration immer erfolgreich ist. Allerdings existieren diverse Fehlerquellen aufgrund der Sensorik, der Ungenauigkeit des Roboters oder durch unbeabsichtigtes Eingreifen des Menschen, die eine Reproduktion selbst bei unveränderten Randbedingungen scheitern lassen können. Diese möglichen Fehlerquellen wurden in der hier vorliegenden Arbeit jedoch nicht weiter berücksichtigt. Um die Reproduktion weiter zu optimieren, müssten mögliche Fehler während der Ausführung erkannt werden. Dies kann beispielsweise durch die Überwachung der Ergebnisse der Reproduktion geschehen, indem ein Soll-Zustand mit dem tatsächlichen Ist-Zustand verglichen wird. Treten Abweichungen auf, weil beispielsweise ein Objekt nicht vorhanden ist, sollte idealerweise darauf mit dem Roboter reagiert werden.

6.2.2 Optimierung der Verbindungen $\mathcal{L}_{\mathrm{R}}$

Das größte Potential zur Optimierung bieten die Verbindungen $\mathcal{L}_{\mathrm{R}}$ der Partikel mit den Objekten. Hier wird bestimmt, welche Teile einer Trajektorie welchen Objekten zugeordnet werden. Diese Zuordnung geschieht bisher auf der Basis von Heuristiken, die prinzipiell nur die räumliche Lage der Trajektorienpartikel bezüglich der Objekte und untereinander verwenden. Zur Steigerung der Qualität der Zuordnung wäre die Integration weiterer Informationen sinnvoll. Dies kann beispielsweise durch Gesten geschehen, indem der Benutzer durch Zeigen deutlich macht, welche Objekte für die aktuelle Bewegung relevant sind. Auch die Analyse der Blickrichtung des Benutzers kann nützliche Informationen liefern [Brea04, Rich10], da dieser höchstwahrscheinlich die relevanten Objekte im Blick behält. Hierfür wäre jedoch ein erhöhter sensorischer Aufwand notwendig, wie beispielsweise eine Spezialbrille. Eine weitere Möglichkeit wäre die Integration von Sprachanweisungen. Der Benutzer könnte die Aufgabe am Roboter demonstrieren und zusätzliche Informationen bereitstellen, welche Merkmale relevant sind. Dies entspricht der menschlichen Art, einer weiteren Person die Lösung einer Aufgabe zu vermitteln und könnte somit auch sinnvoll im Bereich der Roboterprogrammierung eingesetzt werden.

6.2.3 Erhöhung der Adaptionsgeschwindigkeit

Ein weiterer Punkt der Optimierung betrifft die Geschwindigkeit der Adaption der Partikelsimulation und damit die Zeit bis zur Bereitstellung einer reproduzierbaren Trajektorie. Wie bereits angedeutet, benötigt das System einige Sekunden, um eine Demonstration zu adaptieren. Die Geschwindigkeit ist zwar hoch genug, um in den durchgeführten Experimenten online auf die Änderungen der Umwelt reagieren zu können. Bei Erhöhung der Robotergeschwindigkeit wäre es jedoch sinnvoll, die Adaption zu beschleunigen. Dies könnte durch die Parallelisierung der Adaption geschehen. Da nur Wechselwirkungen zwischen Partikeln stattfinden, die miteinander verbunden sind, könnte die Parallelisierung über die Menge der Verbindungen $\mathcal{L}$ stattfinden. Bei einer solchen Aufteilung unterscheiden sich die Berechnungen nicht voneinander und damit ist prinzipiell eine faire Lastverteilung über mehrere Threads möglich. Aufgrund der Art der Berechnung, die mit wenigen Fallunterscheidungen umsetzbar ist, bietet sich auch die Parallelisierung auf einer Grafikkarte an.
Ein weiterer Ansatz zur Steigerung der Geschwindigkeit wäre es, die Trajektorie einem intelligentem Downsampling zu unterziehen. Dieses könnte beispielsweise gerade Segmente stärker

reduzieren als komplexe Bewegungen und dabei auch die bestehenden Verbindungen in Betracht ziehen. Aufgrund der niedrigeren Anzahl von Partikeln und Verbindungen würde sich der Berechnungsaufwand reduzieren, so dass eine schnellere Adaption möglich ist.

6.2.4 Erweiterung auf unbekannte Objekte

In der aktuellen experimentellen Umsetzung der Verfahren, müssen die Objekte in der Demonstration und Reproduktion identisch sein, da sie jeweils nur durch ein einzelnes Partikel repräsentiert werden. Die Trajektorie wird aufgrund der Verbindungen mit allen Objektpartikeln entsprechend adaptiert. Dieses Prinzip könnte auch auf Objektebene angewendet werden. Dazu müssten Merkmale auf der Objektoberfläche detektiert werden und jedes dieser Merkmale als Partikel modelliert werden. Sollte in der Reproduktionsphase ein verändertes Objekt vorliegen, könnten die Matching-Verfahren aus Kapitel 2.8 verwendet werden um zu prüfen, ob das Objekt für die Aufgabe geeignet ist. Die Oberflächenpartikel werden anschließend entsprechend der Oberflächenmerkmale des passenden Objektes auf der Oberfläche verschoben. Die mit diesen Oberflächenpartikeln verbundenen Partikel der Trajektorie adaptieren sich entsprechend und so kann die Aufgabe idealerweise auch mit unbekannten Objekten durchgeführt werden.

6.2.5 Erweiterung der Partikel-Informationen

Eine weitere Möglichkeit der Erweiterung wäre die Integration weiterer Informationen in den Partikeln. In der aktuellen Umsetzung tragen die Partikel als Zusatzinformation nur die Werkzeugoperation. Man könnte darüber hinaus z. B. noch die Gelenkwinkelstellung der Demonstration für jedes Partikel speichern, um für die adaptierte Trajektorie die Berechnung der inversen Kinematik zu beschleunigen oder auf bestimmte Konfigurationsräume zu limitieren. Eine andere Möglichkeit bestünde in der Integration von Kräften. Jedes Partikel könnte nicht nur seine Position und Orientierung, sondern auch eine aufzubringende Kraft und deren Richtung speichern. Obwohl die Adaption positionsbasiert erfolgt, könnte während der Reproduktion diese Position eines Partikels als Startpunkt für eine durch den Roboter aufzubringende Kraft dienen, deren Orientierung entsprechend mit dem Partikel adaptiert wird. Natürlich müssten hier entsprechende Regelungen zur Vereinbarkeit beider Konzepte eingesetzt werden.

6.2.6 Multi-Arm-Programmierung

Die Ansätze zur Programmierung berücksichtigen derzeit nur die Aufgaben, die mit einem Roboterarm durchgeführt werden können. Jedoch wäre eine relativ einfache Erweiterung auf mehrere Arme möglich. Dazu betrachtet man den TCP eines Roboters als Objektpartikel mit zeitlicher Komponenten für einen zweiten Roboter. Aufgrund der online-fähigen Adaption ist es prinzipiell möglich kooperative Mehrarm-Manipulationen durchzuführen. Es müssten jedoch die Verbindungen entsprechend um eine zeitliche Komponenten erweitert werden, um die sich während der Demonstration bewegenden Objekte zu berücksichtigen. Somit können die Roboter jeweils gegenseitig als Referenzobjekte füreinander dienen und es wären Manipulationsaufgaben mit mehreren Armen durchführbar.

6.2.7 Verhaltensadministration

Zwar ist mit dem gezeigten System die parallele Ausführung von Verhalten und das Erlernen neuer Verhalten möglich, jedoch existiert keine Möglichkeit, diese durch den Benutzer intuitiv zu administrieren. Wünschenswert wäre beispielsweise die Möglichkeit, Einfluss auf die Priorität einzelner Verhalten nehmen zu können oder auch Verhalten löschen zu können. In diesem Zusammenhang sollte eine Schnittstelle entwickelt werden, das es erlaubt, neue Verhalten online zu erlernen und sich widersprechende Verhalten zu erkennen und bestehende zu verwalten.

Literaturverzeichnis

[ABB15] ABB. "ABB YuMi (Offizielle Produktwebseite)". 2015. `http://new.abb.com/products/robotics/de/YuMi` (Letzter Abruf: 18.02.2015).

[Abeg00] Frank Abegg, Axel Remde und Dominik Henrich. "Force-and vision-based detection of contact state transitions". In: *Robot Manipulation of Deformable Objects*, Kap. 3, S. 111–134, Springer, 2000.

[Agui00] Eugenio Aguirre und Antonio Gonz. "Fuzzy behaviors for mobile robot navigation: design, coordination and fusion". *International Journal of Approximate Reasoning*, Bd. 25, S. 255–289, 2000.

[Akgu12] Baris Akgun, Maya Cakmak, Jae Yoo und Andrea Thomaz. "Trajectories and Keyframes for Kinesthetic Teaching: A Human-Robot Interaction Perspective". In: *International Conference on Human-Robot Interaction*, 2012.

[Albu07] A. Albu-Schäffer, S. Haddadin, Ch. Ott, A. Stemmer, T. Wimböck und G. Hirzinger. "The DLR lightweight robot: design and control concepts for robots in human environments". *Industrial Robot: An International Journal*, Bd. 34, Nr. 5, S. 376–385, 2007.

[Albu89] James S. Albus, Harry G. McCain und Ronald Lumia. "NASA/NBS Standard reference model for telerobot control system architecture (NASREM)". Tech. Rep., 1989.

[Albu91] James S. Albus. "Outline for a theory of intelligence". *IEEE Transactions on Systems, Man and Cybernetics*, Bd. 21, Nr. 3, 1991.

[Aleo06a] Jacopo Aleotti. "Robust trajectory learning and approximation for robot programming by demonstration". *Robotics and Autonomous Systems*, Bd. 54, Nr. 5, S. 409–413, 2006.

[Aleo06b] Jacopo Aleotti und Stefano Caselli. "Grasp recognition in virtual reality for robot pregrasp planning by demonstration". In: *IEEE International Conference on Robotics and Automation*, 2006.

[Ande90] Tracy L Anderson und Max Donath. "Animal behavior as a paradigm for developing". *Robotics and Autonomous Systems*, Bd. 6, Nr. 1, S. 145–168, 1990.

[Anto07] Gianluca Antonelli, Filippo Arrichiello und Stefano Chiaverini. "The null-space-based behavioral control for autonomous robotic systems". *Intelligent Service Robotics*, Bd. 1, Nr. 1, S. 27–39, 2007.

[Arga09] Brenna D. Argall, Sonia Chernova, Manuela Veloso und Brett Browning. "A survey of robot learning from demonstration". *Robotics and Autonomous Systems*, Bd. 57, Nr. 5, S. 469–483, 2009.

[Arki89] Ronald C. Arkin. "Motor schema—based mobile robot navigation". *The International journal of robotics research*, Bd. 8, Nr. 4, S. 92–112, 1989.

[Arki90] Ronald C. Arkin. "Integrating behavioral, perceptual, and world knowledge in reactive navigation". *Robotics and Autonomous Systems*, Bd. 6, Nr. 1-2, S. 105–122, 1990.

[Arki94a] Ronald C. Arkin und Douglas C. Mackenzie. "Planning to Behave: A Hybrid Deliberative/Reactive Robot Control Architecture for Mobile Manipulation". *International Symposium on Robotics and Manufacturing, Maui, HI*, S. 5–12, 1994.

[Arki94b] Ronald C. Arkin und Douglas C. Mackenzie. "Temporal coordination of perceptual algorithms for mobile robot navigation". *Robotics and Automation*, Bd. 10, Nr. 3, S. 276–286, 1994.

[Arki97] Ronald C. Arkin und Tucker Balch. "AuRA: Principles and practice in review". *Journal of Experimental & Theoretical Artificial Intelligence*, 1997.

[Arki98a] Ronald Arkin. *Behavior Based Robotics*. MIT Press, Cambridge, Massachusetts, 1998.

[Arki98b] Ronald C. Arkin und Tucker Balch. "Cooperative multiagent robotic systems". In: *Artificial Intelligence and Mobile Robots*, S. 1–16, MIT Press, 1998.

[Armb06] Heidi Armbruster. "Neue Kundengruppen für Industrieroboter". Tech. Rep., No. 38. Fraunhofer Institute for Systems and Innovation Research (ISI), 2006.

[Baer92] A.-J. Baerveldt. "Cooperation between man and robot: Interface and safety". *IEEE International Workshop on Robot and Human Communication*, 1992.

[Balc00] Tucker Balch und Maria Hybinette. "Behavior-based coordination of large-scale robot formations". *IEEE Fourth International Conference on MultiAgent Systems*, S. 363–364, 2000.

[Balc93] Tucker Balch. "Avoiding the past: A simple but effective strategy for reactive navigation". *IEEE International Conference on Robotics and Automation*, S. 678–685, 1993.

[Balc95] Tucker Balch und Ronald C. Arkin. "Motor Schema-Based Formation Control for Multiagent Robot Teams.". *ICMAS*, S. 10–24, 1995.

[Balc98] Tucker Balch und Ronald C. Arkin. "Behavior-based Formation Control for Multirobot Teams". *Robotics and Automation*, Bd. 14, Nr. 6, S. 1–15, 1998.

[Bart12] Katharina Barth und Dominik Henrich. "A GOTO-based concept for intuitive robot programming". *IEEE International Conference on Intelligent Robots and Systems*, S. 2338–2345, 2012.

[Bay06] Herbert Bay, Tinne Tuytelaars und Luc Van Gool. "SURF Speeded Up Robust Features". In: *Computer vision–ECCV*, S. 404–417, Springer Berlin Heidelberg, 2006.

[Bend] Marc Bender. "Verfahren zur Industrieroboterprogrammierung - Onlineprogrammierverfahren". `https://wiki.zimt.uni-siegen.de/fertigungsautomatisierung/index.php/Verfahren{_}zur{_}Industrieroboterprogrammierung{_}-{_}Onlineprogrammierverfahren` ter Abruf: 30.04.2015).

[Bert91] Dimitri P. Bertsekas. *Linear Network Optimization.* MIT Press, 1991.

[Bigg03] Geoffrey Biggs und Bruce Macdonald. "A Survey of Robot Programming Systems". *Proceedings of the Australasian conference on robotics and automation*, S. 1–3, 2003.

[Bill04] Aude Billard, Yann Epars, Sylvain Calinon, Stefan Schaal und Gordon Cheng. "Discovering optimal imitation strategies". *Robotics and Autonomous Systems*, Bd. 47, Nr. 2-3, S. 69–77, 2004.

[Bill08] Aude Billard und Sylvain Calinon. *Handbook of Robotics Chapter 59 : Robot Programming by Demonstration.* Springer-Verlag Berlin Heidelberg, 2008.

[Bill10] Erik A. Billing und Thomas Hellström. "A formalism for learning from demonstration". *Paladyn*, Bd. 1, Nr. 1, S. 1–13, 2010.

[Bisc09] Rainer Bischoff und T. Guhl. "Robotic Visions to 2020 and beyond – The Strategic Research Agenda (SRA) for robotics in Europe". Tech. Rep., 2009.

[Blac10] Colin Blackman und Paul Desruelle. "A Helping Hand for Europe: the Competitive Outlook for the EU Robotics Industry". Tech. Rep., 2010.

[Blum01] Stefan A. Blum. "Towards a component-based system architecture for autonomous mobile robots". *IASTED Int. Conf. Robotics and Applications*, 2001.

[Bona06] Andrea Bonarini, Matteo Matteucci und Marcello Restelli. "Concepts and fuzzy models for behavior-based robotics". In: *International Journal of Approximate Reasoning*, S. 110–127, 2006.

[Bona91] R. Peter Bonasso. "Integrating reaction plans and layered competences through synchronous control". In: *12th International Joint Conference on Artificial Intelligence*, S. 1225–1231, 1991.

[Brea04] Cynthia Breazeal, Guy Hoffman und Andrea Lockerd. "Teaching and working with robots as a collaboration". *Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 2004.

[Brem12] Bremer Institut für Produktion und Logistik - BIBA. "RoboScan12 - Studienergebnisse der Onlinebefragung zum Markt der Robotik-Logistik". Tech. Rep., Bremen, 2012.

[Bren07] Manuela Veloso Brenna Argall, Brett Browning. "Learning by Demonstration with Critique from a Human Teacher". In: *Proceedings of the 2007 Conference on Human-Robot Interaction*, Washington, DC, 2007.

[Broo86] Rodney A. Brooks. "A robust layered control system for a mobile robot". *IEEE Journal on Robotics and Automation*, Bd. 2, Nr. 1, S. 14–23, 1986.

[Broo97] Rodney A. Brooks. "From earwigs to humans". *Robotics and Autonomous Systems*, Bd. 20, Nr. 2-4, S. 291–304, 1997.

[Brow04] Brett Browning. "Skill Acquisition and Use for a Dynamically- Balancing Soccer Robot". *Plays*, 2004.

[Burk12] R.E. Burkard, M. Dell'Amico und S. Martello. *Assignment Problems*. SIAM, 2012.

[Cakm11] Maya Cakmak und Andrea Thomaz. "Active Learning with Mixed Query Types in Learning from Demonstration". In: *ICML Workshop on New Developments in Imitation Learning*, 2011.

[Cali07] Sylvain Calinon, Florent Guenter und Aude Billard. "On learning, representing, and generalizing a task in a humanoid robot.". *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, Bd. 37, Nr. 2, S. 286–98, 2007.

[Cali08] Sylvain Calinon und Aude Billard. "A probabilistic Programming by Demonstration framework handling constraints in joint space and task space". *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, S. 367–372, sep 2008.

[Cali10] Sylvain Calinon, D. Florent, Eric L. Sauser, Darwin G. Caldwell und Aude G. Billard. "Learning and reproduction of gestures by imitation". *Robotics & Automation Magazine*, Bd. 17, Nr. 2, S. 44–54, 2010.

[Cali12] Sylvain Calinon und Zhibin Li. "Statistical dynamical systems for skills acquisition in humanoids". *12th International Conference on Humanoid Robots*, S. 323–329, 2012.

[Came93] J.M. Cameron, D.C. MacKenzie, K.R. Ward, R.C. Arkin und W.J. Book. "Reactive control for mobile manipulation". *IEEE International Conference on Robotics and Automation*, S. 228–235, 1993.

[Cede10] Thomas Cederborg, Adrien Baranes und Pierre-Yves Oudeyer. "Incremental local online Gaussian Mixture Regression for imitation learning of multiple tasks". *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010.

[Chen03] Jason Chen und Alex Zelinsky. "Programing by Demonstration : Coping with Suboptimal Teaching Actions". *The International Journal of Robotics Research*, Bd. 22, Nr. 5, S. 299–319, 2003.

[Chia91] Pasquale Chiacchio, Stefano Chiaverini, L. Sciavicco und Siciliano Bruno. "Closed-Loop Inverse Kinematics Schemes for Constrained Redundant Manipulators with Task Space Augmentation and Task Priority Strategy". 1991.

[Chia97] Stefano Chiaverini. "Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators". *Robotics and Automation*, Bd. 13, Nr. 3, S. 398–410, 1997.

[Coff71] Edward G. Coffman und Melanie J. Elphick. "System Deadlocks". *Computing Surveys*, Bd. 3, Nr. 2, S. 67—-78, 1971.

[Conn89a] Jonathan H. Connell. "A Behavior Based Arm Controller". *Robotics and Automation*, Bd. 5, Nr. 6, S. 784–791, 1989.

[Conn89b] Jonathan H. Connell. "A Colony Architecture for an Artificial Creature". Tech. Rep., DTIC Document, 1989.

[Conn92] C.I. Connolly und R.A. Grupen. "Applications of harmonic functions to robotics". Tech. Rep., Computer and InformationScience Department, University of Massachusetts, Massachusetts, 1992.

[Dass01] Palitha Dassanayake, Keigo Watanabe, Kazuo Kiguchi und Kiyotaka Izumi. "Robot manipulator task control with obstacle avoidance using fuzzy behavior-based strategy". *Journal of Intelligent and Fuzzy Systems*, Bd. 10, Nr. 3, S. 139–158, 2001.

[Daut01] Kerstin Dautenhahn und Chrystopher Nehaniv. "Like me? - Measures of Correspondence and Imitation". *Cybernetics & Systems*, Bd. 32, Nr. 1-2, S. 11–51, 2001.

[De R10] Antoine De Rengervé, Sofiane Boucenna, Pierre Andry und Philippe Gaussier. "Emergent imitative behavior on a robotic arm based on visuo-motor associative memories". In: *International Conference on Intelligent Robots and Systems*, S. 1754–1759, 2010.

[De S88] J. De Schutter und H. Van Brussel. "Compliant Robot Motion II. A Control Approach Based on External Control Loops". *The International Journal of Robotics Research*, Bd. 7, Nr. 4, S. 18–33, 1988.

[Decu98] Vincent Decugis und Jacques Ferber. "Action selection in an autonomous agent with a hierarchical distributed reactive planning architecture". In: *Proceedings of the second international conference on Autonomous agents*, S. 354–361, ACM, 1998.

[Denk05] B. Denkena, H. Wörn, R. Apitz, R. Bischoff, B. Hein, P. Kowalski, D. Mages und H. Schuler. "Roboterprogrammierung in der Fertigung". *Werkstattstechnik online*, Bd. 95, S. 656–660, 2005.

[Diet13] M. Dietzfelbinger. "Matchings in bipartiten Graphen". In: *Skript zur Vorlesung Effiziente Algorithmen II*, TU Ilmenau, Ilmenau, 2013.

[Dill00] Rüdiger Dillmann, Oliver Rogalla, Markus Ehrenmann, R Zollner und Monica Bordegoni. "Learning from Human Demonstration and Advice: the machine learning paradigm". *Robotics Research*, Bd. 9, S. 229–238, 2000.

[Dill10] Rüdiger Dillmann, Tamim Asfour, Martin Do, Rainer Jäkel, Alexander Kasper, Pedram Azad, Aleš Ude, Sven R. Schmidt-Rohr und Martin Lösch. "Advances in Robot Programming by Demonstration". *KI - Künstliche Intelligenz*, S. 1–9, 2010.

[Dill94] Rüdiger Dillmann. "Programmieren durch Vormachen in der Robotik". *18. Jahrestagung für Künstliche Intelligenz*, 1994.

[Dill96] R Dillmann, H Friedrich, M Kaiser und A Ude. "Integration of symbolic and subsymbolic learning to support robot programming by human demonstration". *Robotics Research*, 1996.

[Doms07] Wolfgang Domschke. *Transport: Grundlagen, lineare Transport- und Umladeprobleme*. Walter de Gruyter GmbH & Co KG, 2007.

[Dong11] Wang Dongshu, Zhang Yusheng und Si Wenjie. "Behavior-based hierarchical fuzzy control for mobile robot navigation in dynamic environment". In: *Chinese Control and Decision Conference (CCDC)*, S. 2419–2424, School of Electrical Engineering, Zhengzhou University, Zhengzhou, CO 450001, P.R. China, IEEE, 2011.

[Doro09] Daniela Doroftei, Eric Colon, Yvan Baudoin und H Sahli. "Development of a behaviour-based control and software architecture for a visually guided mine detection robot". *Eur Jour Autom Systems*, S. 295–316, 2009.

[Drex13] Wolfgang Drexl und Andreas Domschke. *Einführung in Operations Research*. Springer Verlag, 2013.

[Dria01] D Driankov und A Saffiotti. *Fuzzy logic techniques for autonomous vehicle navigation*. Physica-Verlag, Heidelberg, 2001.

[Duda98] Richard O. Duda, Peter E. Hart und David G. Stork. "Unsupervised Learning and Clustering". In: *Pattern Classification and Scene Analysis*, Kap. 10, Wiley, New York, 1998.

[Edma15] Alistair Edmands. "IRobot Corporation (Offizielle Webseite)". 2015. `http://www.irobot.de/` (Letzter Abruf: 18.02.2015).

[Edsi08a] Aaron Edsinger. "A Behavior Based Approach to Humanoid Robot Manipulation". *Artificial Intelligence*, 2008.

[Edsi08b] Aaron Edsinger und Charles C Kemp. "Two Arms are Better than One : A Behavior Based Control System for Assistive Bimanual Manipulation". In: *Recent progress in robotics: Viable robotic service to human*, S. 345–355, Springer, 2008.

[Eger00] Magnus Egerstedt. "Behavior-based robotics Using Hybrid Automata". *Hybrid Systems: computation and control*, S. 103–116, 2000.

[Eger99] Magnus Egerstedt und K. Johansson. "Behavior based robotics using regularized hybrid automata". In: *Decision and Control*, 1999.

[Ehre00] M. Ehrennmann, Despina Ambela, P. Steinaus und Rüdiger Dillmann. "A comparison of four fast vision based object recognition methods for programming by demonstration applications". *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, Bd. 2, 2000.

[Ehre01] Markus Ehrenmann, Oliver Rogalla, Raoul Zöllner und Rüdiger Dillmann. "Teaching Service Robots complex Tasks: Programming By Demonstration for Workshop and Household Environments". In: *2001 International Conference on Field and Service Robots (FSR)*, 2001.

[Ehre02] M. Ehrenmann, R. Zöllner, O. Rogalla und R. Dillmann. "Programming service tasks in household environments by human demonstration". In: *Proceedings - IEEE International Workshop on Robot and Human Interactive Communication*, S. 460–467, 2002.

[Ekva07] Staffan Ekvall. *Robot Task Learning from Human Demonstration.* Dissertation, 2007.

[Ekva08] Staffan Ekvall und Danica Kragic. "Robot learning from demonstration: A task-level planning approach". *International Journal of Advanced Robotic Systems*, Bd. 5, S. 223–234, 2008.

[Elli13] Theodor Ellinger, Günter Beuermann und Rainer Leisten. *Operations Research.* Springer Verlag, 2013.

[Emer01] Rosemary Emery und Tucker Balch. "Behavior-based control of a non-holonomic robot in pushing tasks". In: *IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, S. 2381–2388, Ieee, 2001.

[Eskr07] Brent E. Eskridge und Dean F. Hougen. "Using priorities to simplify behavior coordination". In: *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems - AAMAS '07*, S. 1, ACM Press, New York, New York, USA, 2007.

[Fagg94] Andrew H Fagg und David Lotspeich. "Rapid reinforcement learning for reactive control policy design in autonomous robots". In: *World Congress on Neural Networks*, 1994.

[Ferb99] Jacques Ferber. *Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence.* Bd. 222, Addison-Wesley, 1999.

[Ferr95] Cynthia L. Ferrell. "Global behavior via cooperative local control". *Autonomous Robots*, Bd. 2, Nr. 2, S. 105–125, 1995.

[Fial87] John C. Fiala, Ronald Lumia und James S. Albus. "Servo Level Algorithms for the NASREM Telefobot Control Architechture". *International Society of Optical Engineering*, Bd. 851, Nr. Space Station Automation III, 1987.

[Fier01] R. Fierro, A.K. Das, V. Kumar und J.P. Ostrowski. "Hybrid control of formations of robots". In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, 2001.

[Fier98] R Fierro und FL Lewis. "Control of a nonholonomic mobile robot using neural networks". In: *IEEE Transactions on Neural Networks*, 1998.

[Fish] "Fishertechnik ROBO Pro Webpage". `http://www.fischertechnik.de/home/info/Computing/ROBO-Pro-Software.aspx/usetemplate-1{_}column{_}no{_}pano/` (Letzter Abruf: 03.08.2015).

[Fodo83] Jerry A Fodor. "The modularity of mind: An essay on faculty psychology". Tech. Rep., MIT, 1983.

[Frie96] H Friedrich, S Münch und R Dillmann. "Robot programming by demonstration (RPD): Supporting the induction by human interaction". *Machine Learning*, Bd. 28, S. 1–28, 1996.

[Frie99] Holger Friedrich, Rudiger Dillmann und Oliver Rogalla. "Interactive robot programming based on human demonstration and advice". *Sensor Based Intelligent Robots*, 1999.

[Fuji03] Masahiro Fujita, Yoshihiro Kuroki und Tatsuzo Ishida. "Autonomous behavior control architecture of entertainment humanoid robot SDR-4X". In: *Intelligent Robots and Systems*, 2003.

[Gada03] S. C. Gadanho. "Learning behavior-selection by emotions and cognition in a multi-goal robot task". *The journal of machine learning research*, Bd. 4, S. 385–412, 2003.

[Gat98] Erann Gat. "On three-layer architectures". In: *Artificial intelligence and mobile robots*, S. 195–210, 1998.

[GomT15] "GomTec Roberta Website". 2015. `http://www.gomtec.de/leichtbaurobotik.html` (Letzter Abruf: 18.02.2015).

[Good92] Richard Goodwin und R Simmons. "Rational handling of multiple goals for mobile robots". *Artificial intelligence planning systems*, 1992.

[Goul13] Harvey Gould, Jan Tobochnik und Wolfgang Christian. "Simulating Particle Motion". In: *An Introduction to Computer Simulation Methods*, S. 60–97, 2013.

[GPS15] Gesellschaft für Produktionssysteme GmbH GPS. "SME Robotics Project (Offizielle Webseite)". 2015. `http://www.smerobotics.org/` (Letzter Abruf: 26.07.2015).

[Grib08] E Gribovskaya und A Billard. "Combining Dynamical Systems control and programming by demonstration for teaching discrete bimanual coordination tasks to a humanoid robot". In: *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, S. 33–40, 2008.

[Grim10] Matthias Grimm und Ioannis Iossifidis. "Behavioral Organization for Mobile Robotic Systems: An Attractor Dynamics Approach". *Memory*, S. 56–61, 2010.

[Grol10] Daniel H Grollman und Odest Chadwicke Jenkins. "Incremental learning of subtasks from unsegmented demonstration". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, S. 261–266, Ieee, oct 2010.

[Grot13] Christian Groth und Dominik Henrich. "Multi-Tasking of Competing Behaviors on a Robot Manipulator". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2013.

[Grot14a] Christian Groth und Dominik Henrich. "One-shot Robot Programming by Demonstration by Adapting Motion Segments". In: *IEEE Robotics and Biomimetics*, 2014.

[Grot14b] Christian Groth und Dominik Henrich. "One-Shot Robot Programming by Demonstration using an Online Oriented Particles Simulation". In: *IEEE Robotics and Biomimetics*, 2014.

[Grot14c] Christian Groth und Dominik Henrich. "Single-Shot Learning and Scheduled Execution of Behaviors for a Robotic Manipulator". In: *International Symposium on Robotics*, 2014.

[Grou15] The Lego Group. "Lego Mindstorms EV3 Software (Offizielle Webseite)". 2015. `http://www.lego.com/de-de/mindstorms/learn-to-program` (Letzter Abruf: 03.08.2015).

[Guiz12] Erico Guizzo und Evan Ackerman. "The rise of the robot worker". *IEEE Spectrum*, Bd. 49, Nr. 10, S. 34–41, 2012.

[Guyo03] Isabelle Guyon und A Elisseeff. "An introduction to variable and feature selection". *Journal of Machine Learning Research*, Bd. 3, S. 1157–1182, 2003.

[Hadd07] Sami Haddadin, Alin Albu-Schaffer und Gerd Hirzinger. "Safety Evaluation of Physical Human-Robot Interaction via Crash-Testing". *Robotics: Science and Systems Conference (RSS 2007)*, Bd. 3, 2007.

[Hage02] Martin Hägele, Walter Schaaf und Evert Helms. "Robot assistants at manual workplaces - Effective co-operation and safety aspects". *Proceedings of the 33rd International Symposium on Robotics (ISR)*, S. 6, 2002.

[Hira92] Shinichi Hirai und Kazuaki Iwata. "Recognition of contact state based on geometric model". In: *Robotics and Automation*, 1992.

[Hock88] Roger W Hockney und James W Eastwood. *Computer simulation using particles*. CRC Press, 1988.

[Hoff04] Heiko Hoffmann und Ralf Möller. "Action Selection and Mental Transformation Based on a Chain of Forward Models Action Selection and Mental Transformation Based on a Chain of Forward Models". In: *From Animals to Animats 8*, S. 213–222, 2004.

[Hoff95] Joel Hoff und George A Bekey. "An architecture for behaviour coordination learning". In: *IEEE International Conference on Neural Networks*, 1995.

[Hovl96] Geir E. Hovland, Pavan Sikka und Brenan J. McCarragher. "Skill acquisition from human demonstration using a hidden Markov model". *Proceedings of IEEE International Conference on Robotics and Automation*, Bd. 3, S. 2706–2711, 1996.

[Huan08] Shu Huang, Erwin Aertbeliën und Hendrik Van Brussel. "A Constraint-Based Behavior Fusion Mechanism on Mobile Manipulator". In: *2008 ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems (LAB-RS)*, S. 83–88, IEEE, aug 2008.

[Huq06] Rajibul Huq, George Mann und Raymond Gosine. "Behavior-based Robot Control Using Fuzzy Discrete Event System". In: *IEEE International Conference on Fuzzy Systems*, 2006.

[Iba05] Soshi Iba, Christiaan J.J. Paredis und Pradeep Khosla. "Interactive multi-modal robot programming". *The international journal of robotics research*, Bd. 24, Nr. 1, S. 83–104, 2005.

[Ikeu92] Katsushi Ikeuchi und Takashi Suehiro. "Assembly task recognition using face-contact relations". *Proceedings of the IEEE International Conference on Robotics and Automation*, Bd. 3, 1992.

[Inno07] Bianca Innocenti, Beatriz López und Joaquim Salvi. "A multi-agent architecture with cooperative fuzzy control for a mobile robot". *Robotics and Autonomous Systems*, Bd. 55, Nr. 12, S. 881–891, 2007.

[Jaaf07] Jafreezal Jaafar, Eric McKenzie und Alan Smaill. "A fuzzy action selection method for virtual agent navigation in unknown virtual environments". *Fuzzy Systems Conference*, Bd. 2, Nr. 2, S. 144–154, 2007.

[Joy07] K.I. Joy. "Numerical Methods for Particle Tracing in Vector Fields". Tech. Rep., University of California, 2007.

[Jung96] David Jung und Alexander Zelinsky. "Whisker based mobile robot navigation". In: *International Conference on Intelligent Robots and Systems*, 1996.

[Kael86] Leslie Pack Kaelbling. "An architecture for intelligent reactive systems". *Proceedings of the 1986 Workshop: Reasoning about Actions and Plans*, Bd. 30, 1986.

[Kael91] Leslie Pack Kaelbling. "A situated-automata approach to the design of embedded agents". *ACM SIGART Bulletin*, Bd. 2, Nr. 4, S. 85–88, 1991.

[Kais96] M. Kaiser und R. Dillmann. "Building elementary robot skills from human demonstration". *Proceedings of IEEE International Conference on Robotics and Automation*, Bd. 3, Nr. April, S. 2700–2705, 1996.

[Kasp99] M. Kasper, G. Fricke und E. von Puttkamer. "A behavior-based architecture for teaching more than reactive behaviors to mobile robots". *1999 Third European Workshop on Advanced Mobile Robots (Eurobot'99). Proceedings (Cat. No.99EX355)*, S. 203–210, 1999.

[Kerp03] Oliver Kerpa, Karste Weiss und Heinz Wörn. "Development of a flexible tactile sensor system for a humanoid robot". *Proc. of the 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nr. October, 2003.

[Khat85] Oussama Khatib. "Real-time obstacle avoidance for manipulators and mobile robots". *IEEE International Conference on Robotics and Automation*, Bd. 2, 1985.

[Khot90] Alireza Khotanzad und Yaw Hua Hong. "Invariant image recognition by Zernike moments". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Bd. 12, S. 489–497, 1990.

[Kino15] "Kinova Robotics (Offizielle Webseite)". 2015. `http://kinovarobotics.com/` (Letzter Abruf: 18.02.2015).

[Kond08] M. Kondo, J. Ueda und T. Ogasawara. "Recognition of in-hand manipulation using contact state transition for multifingered robot hand control". In: *Robotics and Autonomous Systems*, 2008.

[Koni11] George Konidaris, Scott Kuindersma, Roderic Grupen und Andrew G. Barto. "Robot learning from demonstration by constructing skill trees". *The International Journal of Robotics Research*, Bd. 31, Nr. 3, S. 360–375, dec 2011.

[Kose93] Jana Kosecka und Ruzena Bajcsy. "Cooperation of visually guided behaviors". In: *1993 (4th) International Conference on Computer Vision*, 1993.

[Kova04] Szilveszter Kovács. "A Flexible Fuzzy Behaviour-based Control Structure". In: *2nd Slovakian – Hungarian Joint Symposium on Applied Machine Intelligence*, 2004.

[Kubo03] N. Kubota und M. Mihara. "Multi-objective behavior coordination of multiple robots interacting with a dynamic environment". *The 12th IEEE International Conference on Fuzzy Systems, 2003. FUZZ '03.*, Bd. 1, 2003.

[Kuhn10] Harold W. Kuhn. "The Hungarian method for the assignment problem". In: *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art (2010)*, S. 29–47, Springer Berlin Heidelberg, 2010.

[Kuhn12] Stefan Kuhn. *Wissens- und sensorbasierte geometrische Rekonstruktion*. Dissertation, Universität Bayreuth, 2012.

[KUKA] KUKA Roboter GmbH. "The Power of Control. KR C4. Das Steuerungssystem mir Zukunftssicherheit.". `http://www.kuka-robotics.com/res/sps/a737ee03-5832-4c95-9d91-84e0de80c664{_}KUKA{_}PB{_}STEUERUNGSSYSTEME{_}DE.pdf` (Letzter Abruf: 03.08.2015).

[Kuni94] Yasuo Kuniyoshi, Jukka Rickki, Makoto Ishii und Sebastien Rougeaux. "Vision-based behaviors for multi-robot cooperation". In: *Intelligent Robots and Systems (IROS)*, 1994.

[Kwon12] Tan Chee Kwong, Shamsudin H.M. Amin, Rosbi Mamat und Jozsef K. Tar. "Using voting technique in mobile robot behavior coordination for goal-directed navigation". *Jurnal Teknologi*, Bd. 36, Nr. D, S. 55–69, 2012.

[Lang15] Gael Langevin. "InMoov - open source 3D printed life-size robot". 2015. `http://www.inmoov.fr` (Letzter Abruf: 18.02.2015).

[Larg99] E. W. Large. "Scaling the Dynamic Approach to Path Planning and Control: Competition among Behavioral Constraints". *The International Journal of Robotics Research*, Bd. 18, Nr. 1, S. 37–58, 1999.

[Lato91] Jean-Claude Latombe. "Robot Motion Planning". In: *Wiley Encyclopedia of Computer Science and Engineering*, S. 2439–2446, 1991.

[Laue05] Tim Laue und T Röfer. "A behavior architecture for autonomous mobile robots based on potential fields". *RoboCup 2004: Robot Soccer World Cup VIII*, 2005.

[Lens02] Scott Lenser, James Bruce und Manuela Veloso. "A modular hierarchical behavior-based architecture". *RoboCup 2001: Robot Soccer World Cup V*, 2002.

[Lewi98] F. W. Lewis, S. Jagannathan und A. Yesildirak. "Neural network control of robot manipulators and non-linear systems". 1998.

[Li97] Wei Li, Chenyu Ma und F.M. Wahl. "A neuro-fuzzy system architecture for behavior-based control of a mobile robot in unknown environments". *Fuzzy sets and systems*, Bd. 87, Nr. 2, 1997.

[Lian11] Yuming Liang, Lihong Xu, Ruihua Wei, Bingkun Zhu und Haigen Hu. "Behavior-based fuzzy control for indoor cleaning robot obstacle avoidance under dynamic environment". In: *International Conference on Measuring Technology and Mechatronics Automation*, S. 637–640, 2011.

[LLoy82] Stuart P. LLoyd. "Least Squares Quantization in PCM". *IEEE Transaction on Information Theory*, Bd. 28, Nr. 2, 1982.

[Luks08] Tobias Luksch und Karsten Berns. "Initiating normal walking of dynamic biped with a biologically motivated control". In: *Proceedings of the 11th International Conference on Climbing and Walking Robots (CLAWAR)*, 2008.

[MacK95] Douglas C. MacKenzie, Jonathan M. Cameron und Ronald C. Arkin. "Specification and execution of multiagent missions". In: *Intelligent Robots and Systems (IROS)*, 1995.

[Maes89] Pattie Maes. "How to do the right thing". Tech. Rep., Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1989.

[Maes90] Pattie Maes. "Situated agents can have goals". *Robotics and Autonomous Systems*, Bd. 6, Nr. 1-2, S. 49–70, 1990.

[Maes91] Pattie Maes. "Learning behavior networks from experience". In: Francisco J Varela und Paul Bourgine, Eds., *Toward a practice of autonomous systems*, S. 48–57, The MIT Press, 1991.

[Maes93] Pattie Maes. "Modeling Adaptive Autonomous Agents". *Artificial Life*, Bd. 1, Nr. 1-2, S. 135–162, 1993.

[Mata08] Maja Matarić und François Michaud. "Behavior-Based Systems". In: *Handbook of Robotics*, Kap. 38.1, S. 891–909, Springer, 2008.

[Mata97] M.J. Matarić. "Reinforcement learning in the multi-robot domain". In: *Robot Colonies*, S. 73–83, Springer, 1997.

[Mata98] Maja Matarić. "Behavior-based Primitives for articulated Control". In: *International Conference on Simulation of Adaptive Behavior*, 1998.

[Maye07] H. Mayer, I. Nagy und A. Knoll. "Adaptive control for human-robot skilltransfer: Trajectory planning based on fluid dynamics". In: *Robotics and Automation*, S. 10–14, 2007.

[Mcgu02] P. Mcguire, J. Fritsch, J.J. Steil, F. Röthling, G.A. Fink, S. Wachsmut, G. Sagerer und H. Ritter. "Multi-modal human-machine communication for instructing robot grasping tasks". *International Conference on Intelligent Robots and Systems*, Bd. 2, 2002.

[Menz00] Rainer Menzner, Axel Steinhage und Wolfram Erlhagen. "Generating interactive robot behavior: A mathematical Approach". *From animals to animats*, Nr. Proc. of the Sixth Int. Conf. On Simulation of Adaptive Behavior, 2000.

[Meye07] Christian Meyer, Rebecca Hollmann, Christopher Parlitz und Martin Hägele. "Programmieren durch Vormachen für Assistenzsysteme – Schweiß- und Klebebahnen intuitiv programmieren (Programming by Demonstration for Assistive Systems – Intuitive Programming of Welding and Gluing Trajectories)". *it - Information Technology*, Bd. 49, Nr. 4, S. 238–246, 2007.

[Mull11] Matthias Müller und Nuttapong Chentanez. "Solid simulation with oriented particles". *ACM SIGGRAPH 2011 papers on - SIGGRAPH '11*, Bd. 1, Nr. 212, S. 1, 2011.

[Munk57] James Munkres. "Algorithms for the Assignment and Transportation Problems". *Journal of the Society for Industrial and Applied Mathematics*, Bd. 5, Nr. 1, S. 32–38, 1957.

[Naka98] H. Nakashima und I. Noda. "Dynamic subsumption architecture for programming intelligent agents". *Proceedings International Conference on Multi Agent Systems (Cat. No.98EX160)*, 1998.

[Nico02] Monica Nicolette Nicolescu und Maja Matarić. "A hierarchical architecture for behavior-based robots". In: *Proceedings of the first international joint conference on Autonomous agents and multiagent systems*, ACM, 2002.

[Nico03] Monica Nicolette Nicolescu und Maja Matarić. "Natural methods for robot task learning: Instructive demonstrations, generalization and practice". In: *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, 2003.

[Niek12] Scott Niekum, Sarah Osentoski, George Konidaris und Andrew G. Barto. "Learning and generalization of complex tasks from unstructured demonstrations". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, S. 5239–5246, IEEE, 2012.

[Niek13] Scott Niekum und Sachin Chitta. "Incremental Semantically Grounded Learning from Demonstration". In: *Robotics: Science and Systems Conference*, 2013.

[Niek15] Scott Niekum, Sarah Osentoski, George Konidaris, Sachin Chitta, Bhaskara Marthi und Andrew G. Barto. "Learning grounded finite-state representations from unstructured demonstrations". *The International Journal of Robotics Research*, Bd. 34, Nr. 2, S. 131–157, 2015.

[Noji09] Yusuke Nojima. "Multi-objective behavior coordination based on sensory network for multiple mobile robots". In: *2009 IEEE Workshop on Robotic Intelligence in Informationally Structured Space, RiiSS 2009 - Proceedings*, S. 66–72, 2009.

[Ober14] Antje Ober-Gecks und Maria Haenel. "Fast multi-camera reconstruction and surveillance with human tracking and optimized camera configurations". In: *ISR/Robotik 2014*, 2014.

[Olen05] Adam Olenderski, Monica Nicolescu und Sushil Louis. "Robot Learning by Demonstration using Forward Models of Schema-Based Behaviors". In: *ICINCO*, 2005.

[Pais13] Lucia Pais und Aude Billard. "Extracting task constraints as a middle layer between low level control and high level planning". In: *RSS Workshop on Programming with constraints*, 2013.

[Pard07a] Michael Pardowitz und Rüdiger Dillmann. "Towards life-long learning in household robots: The Piagetian approach". *2007 IEEE 6th International Conference on Development and Learning, ICDL*, S. 88–93, 2007.

[Pard07b] Michael Pardowitz, Bernhard Glaser und Rüdiger Dillmann. "Learning Repititive Robot Programs from Demonstrations Using Version Space Algebra". *Hand*, S. 250, 2007.

[Pard07c] Michael Pardowitz, Steffen Knoop, Ruediger Dillmann und Raoul D Zöllner. "Incremental learning of tasks from user demonstrations, past experiences, and vocal comments.". *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, Bd. 37, Nr. 2, S. 322–32, apr 2007.

[Park06] No-Hoon Park, Yonghwan Oh und Sang-Rok Oh. "Behavior-based control of robotic hand by tactile servoing". *International Journal of Applied Electromagnetics and Mechanics*, Bd. 24, Nr. 3-4, S. 311–321, 2006.

[Park07] Kiwon Park und Nian Zhang. "Behavior-based autonomous robot navigation on challenging terrain: A dual fuzzy logic approach". In: *Proceedings of the 2007 IEEE Symposium on Foundations of Computational Intelligence, FOCI 2007*, S. 239–244, 2007.

[Paru11] Sven Parusel. "Modular state-based behavior control for safe human-robot interaction: A lightweight control architecture for a lightweight robot". In: *Robotics and Automation (ICRA)*, 2011.

[Past09] Peter Pastor, Heiko Hoffmann, Tamim Asfour und Stefan Schaal. "Learning and generalization of motor skills by learning from demonstration". In: *International Conference on Robotics and Automation*, S. 763–768, IEEE, 2009.

[Paul95] George Paul und Katsushi Ikeuchi. "Modelling planar assembly tasks: Representation and recognition". *Human Robot Interaction and Cooperative Robots*, Bd. 1, S. 17–22, 1995.

[Pete08] Jan Peters und Stefan Schaal. "Policy learning for motor skills". *Neural Information Processing*, S. 233–242, 2008.

[Pine02] Joelle Pineau und Sebastian Thrun. "High-level robot behavior control using POMDPs". *AAAI-02 Workshop on Cognitive Robotics*, Bd. 107, Nr. 1, 2002.

[Pirj00] Paolo Pirjanian. "Multiple objective behavior-based control". *Robotics and Autonomous Systems*, Bd. 31, Nr. 1, S. 53–60, 2000.

[Pirj98a] P Pirjanian, HI Christensen und JA Fayman. "Application of voting to fusion of purposive modules: An experimental investigation". In: *Robotics and Autonomous Systems*, 1998.

[Pirj98b] Paolo Pirjanian. *Multiple Objective Action Selection & Behavior Fusion using Voting*. Dissertation, 1998.

[Pirj99a] Paolo Pirjanian. "Behavior coordination mechanisms state-of-the-art". *Institute for Robotics and Intelligent Systems Technical Report IRIS*, Bd. 1, Nr. 213, S. 1–49, 1999.

[Pirj99b] Paolo Pirjanian und Maja Matarić. "A decision-theoretic approach to fuzzy behavior coordination". In: *IEEE International Symposium on Computational Intelligence in Robotics and Automation. CIRA'99 (Cat. No.99EX375)*, 1999.

[Pluz12] Sergey Pluzhnikov und Daniel Schmidt. "Behavior-based arm control for an autonomous bucket excavator". In: *Commercial Vehicle Technology*, S. 251–261, 2012.

[Qure04] Faisal Qureshi, Demetri Terzopoulos und Ross Gillett. "The Cognitive Controller: A hybrid, deliberative/reactive control architecture for autonomous robots". *Innovations in Applied Artificial Intelligence*, Bd. 3029, S. 1102–1111, 2004.

[Rich10] Charles Rich, Brett Ponsler, Aaron Holroyd und Candace L. Sidner. "Recognizing engagement in human-robot interaction". In: *ACM/IEEE International Conference on Human-Robot Interaction*, 2010.

[Rick09] Markus Rickert, Michael Kassecker und Alois Knoll. "Aufgabenbeschreibung mit verhaltensbasierter Robotersteuerung und natürlicher Kommunikation". Tech. Rep., Technische Universität München, 2009.

[Riek95] Jukka Riekki und Yasuo Kuniyoshi. "Architecture for vision-based purposive behaviors". In: *IEEE International Conference on Intelligent Robots and Systems*, 1995.

[Riek97] Jukka Riekki und Juha Roning. "Reactive task execution by combining action maps". In: *IEEE International Conference on Intelligent Robot and Systems*, S. 224–230, 1997.

[Robb10] Philip Robbins. "Modularity of mind". *Stanford Encyclopedia of Philosophy*, 2010.

[Robo14] International Foundation of Robotics. "Executive Summary Industrial Robots". Tech. Rep., IFR Statistical Department, 2014.

[Robo15] Universal Robots. "Universal robots (Offizielle Webseite)". 2015. `http://www.universal-robots.com/de/` (Letzter Abruf: 18.02.2015).

[Rose00] Julio K. Rosenblatt. "Optimal selection of uncertain actions by maximizing expected utility". *Autonomous Robots*, Bd. 9, Nr. 1, S. 17–25, 2000.

[Rose95] Julio K. Rosenblatt und Charles E. Thorpe. "Combining multiple goals in a behavior-based architecture". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, S. 136–141, IEEE Comput. Soc. Press, 1995.

[Rose97] Julio K. Rosenblatt. "DAMN: A distributed architecture for mobile navigation". *Journal of Experimental & Theoretical Artificial Intelligence*, Bd. 9, Nr. 2-3, S. 167–178, 1997.

[Rozo13] Leonel Rozo, Sylvain Calinon, Darwin Caldwell, Pablo Jimenez und Carme Torras. "Learning Collaborative Impedance-based Robot Behaviors". In: *Proceedings of the AAAI Conference on Artificial Intelligence.*, 2013.

[Rueg15] Livia Rueger. "TU Dortmund - Lehrstuhl fuer Regelunstechnik (Offizielle Webseite)". 2015. `http://www.rst.e-technik.tu-dortmund.de/cms/de/Forschung/Schwerpunkte/Robotik/Roboterprogrammierung{_}durch{_}Demonstration/index.html` (Letzter Abruf: 26.07.2015).

[Russ03] Stuart Russell und Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, New Jersey, 2003.

[Rusu03] Petru Rusu, Emil M. Petriu, Thomas E. Whalen, Cornell Aurel und Hans J.W. Spoelder. "Behavior-based neuro-fuzzy controller for mobile robot navigation". *IEEE Transactions on Instrumentation and Measurement*, Bd. 52, Nr. 4, S. 1335–1340, 2003.

[Rusu10] Radu Bogdan Rusu, Gary Bradski, Romain Thibaux und John Hsu. "Fast 3D recognition and pose using the viewpoint feature histogram". In: *International Conference on Intelligent Robots and Systems*, S. 2155–2162, 2010.

[Rybs07] Paul E Rybski, Kevin Yoon, Jeremy Stolarz und Manuela M Veloso. "Interactive Robot Task Training through Dialog and Demonstration". In: *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, S. 49–56, 2007.

[Saff97a] Alessandro Saffiotti. "Fuzzy logic in autonomous robotics: behavior coordination". In: *Proceedings of 6th International Fuzzy Systems Conference*, 1997.

[Saff97b] Alessandro Saffiotti. "The uses of fuzzy logic in autonomous robot navigation". In: *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, S. 180–197, 1997.

[Scha03] Stefan Schaal, Auke Ijspeert und Aude Billard. "Computational approaches to motor learning by imitation.". *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, Bd. 358, S. 537–547, 2003.

[Scha05] Stefan Schaal, Jan Peters und Jun Nakanishi. "Learning movement primitives". In: *The Eleventh International Symposium Robotics Research*, 2005.

[Sche02] Matthias Scheutz. "Affective Action Selection and Behavior Arbitration for autonomous Robots". In: *IC-AI*, S. 334–340, 2002.

[Sche04] Matthias Scheutz und Virgil Andronache. "Architectural mechanisms for dynamic changes of behavior selection strategies in behavior-based systems". *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, Bd. 34, Nr. 6, S. 2377–2395, 2004.

[Schn10] Markus Schneider und Wolfgang Ertel. "Robot Learning by Demonstration with local Gaussian process regression". In: *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, S. 255–260, IEEE, 2010.

[Scho92] Gregor Schöner und Michael Dose. "A dynamical systems approach to task-level system integration used to plan and control autonomous vehicle motion". *Robotics and Autonomous Systems*, Bd. 10, Nr. 4, S. 253–267, 1992.

[Schu08] Joris De Schutter, Tinne De Laet, Johan Rutgeerts, Wilm Decr, Ruben Smits, Erwin Aertbeli, Kasper Claes und Herman Bruyninckx. "Constraint-Based Task Specification and Estimation for Sensor-Based Robot Systems in the Presence of Geometric Uncertainty". *The International Journal of Robotics Research*, Bd. 23, Nr. 3, S. 383–415, 2008.

[Schu13] John Schulman, Jonathan Ho, Cameron Lee und Pieter Abbeel. "Learning from Demonstrations Through the Use of Non-Rigid Registration". In: *International Symposium on Robotics Research*, 2013.

[Sera02] H. Seraji und A. Howard. "Behavior-based robot navigation on challenging terrain: A fuzzy logic approach". *IEEE Transactions on Robotics and Automation*, Bd. 18, Nr. 3, S. 308–321, 2002.

[Shah14] Maurice Shahd und Michael Schidlack. "Grosses Interesse an Haushaltsrobotern". Tech. Rep., Berlin, 2014.

[Simo60] Herbert A. Simon. "The new science of management decision". *The Academy of Management Review*, Bd. 3, Nr. 1, 1960.

[Smit15] Tim Smith. "Robot Operating System". 2015. `www.ros.org` (Letzter Abruf: 01.10.2015).

[Soll09] Katharina Soller und Dominik Henrich. "Intuitive Robot Programming of Spatial Control Loops with Linear Movements". In: *Advances in Robotics Research*, S. 147–158, Springer Berlin Heidelberg, 2009.

[Span14] Michael Spangenberg und Dominik Henrich. "Towards an intuitive interface for instructing robots handling tasks based on verbalized physical effects". In: *IEEE RO-MAN*, 2014.

[Spra04] Nathan Sprague. *Learning to coordinate visual behaviors.* Dissertation, The University of Rochester, 2004.

[Stau15a] Stäubli. "Die VAL3 Programmiersprache". 2015. `http://www.staubli.com/de/robotik/robotersoftware/staeubli-robotics-controls/val3-programmiersprache/` (Letzter Abruf: 03.08.2015).

[Stau15b] Stäubli. "Visual Components, Staubli Add-Ons (Offizielle Webseite)". 2015. `http://www.visualcomponents.com/add-ons-2/staubli-add-on/` (Letzter Abruf: 30.04.2015).

[Stei00a] Axel Steinhage und Thomas Bergener. "Learning by doing: A dynamic architecture for generating adaptive behavioral sequences". In: *Proceedings of the Second International ICSC Symposium on Neural Computation (NC)*, S. 813–820, 2000.

[Stei00b] Axel Steinhage und Werner von Seelen. "Dynamische Systeme zur Verhaltensgenerierung eines antropomorphen Roboters". *AMS*, 2000.

[Stei04] Jochen Steil, Frank Röthling, Robert Haschke und Helge Ritter. "Situated robot learning for multi-modal instruction and imitation of grasping". *Robotics and Autonomous Systems*, Bd. 47, Nr. 2-3, S. 129–141, 2004.

[Stei98a] Axel Steinhage und Thomas Bergener. "Dynamical systems for the behavioral organization of an anthropomorphic mobile robot". *Proceedings of the Fifth International Conference on Simulation of Adaptive Behavior*, Bd. 5, 1998.

[Stei98b] Axel Steinhage und Gregor Schöner. "Dynamical systems for the behavioral organization of autonomous robot navigation". *Photonics East*, 1998.

[Stro15] K.H. Strobl, W. Sepp, S. Fuchs, C. Paredes, M. Misek und K. Arbter. "DLR Calde and DLR Callab". 2015. `http://www.robotic.dlr.de/callab/`.

[Taip09] Tapio Taipalus und Aarne Halme. "An action pool architecture for multi-tasking service robots with interdependent resources". In: *IEEE international conference on Computational intelligence in robotics and automation*, 2009.

[Taka99] Jun Takamatsu, H. Kirnura und Katsushi Ikeuchi. "Classifying contact states for recognizing human assembly task". In: *Multisensor Fusion and Integration for Intelligent Systems*, 1999.

[Tane07] Andrew S. Tanenbaum. *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA, 2007.

[Temp14] Udo Templiner. "TU Berlin - Industrieroboterprogrammierung durch räumliche Interaktion". 2014. `https://www.iat.tu-berlin.de/menue/forschung/projekte/roboterprogrammierung{_}durch{_}raeumliche{_}interaktion/` (Letzter Abruf: 26.07.2015).

[Tesc15] Matthias Teschner. *Simulation in Computer Graphics (Course Notes)*. Universität Freiburg, Freiburg, 2015.

[Thon00] S. Thongchai, S. Suksakulchai, D.M. Wilkes und Sarkar. N. "Sonar behavior-based fuzzy control for a mobile robot". In: *IEEE International Conference on Systems, Man, and Cybernetics*, 2000.

[Towl14] Bradford A. Towle und Monica Nicolescu. "An auction behavior-based robotic architecture for service robotics". *Intelligent Service Robotics*, Bd. 7, Nr. 3, S. 157–174, 2014.

[Tsot95] John K. Tsotsos. "Behaviorist intelligence and the scaling problem". *Artificial Intelligence*, Bd. 75, S. 135–160, 1995.

[Vada07] Prahlad Vadakkepat, Xiao Peng, Boon Kiat Quek und Tong Heng Lee. "Evolution of fuzzy behaviors for multi-robotic system". *Robotics and Autonomous Systems*, Bd. 55, Nr. 2, S. 146–161, feb 2007.

[Vaka12] Aleksandar Vakanski, Iraj Mantegh, Andrew Irish und Farrokh Janabi-sharifi. "Demonstration Using Hidden Markov Model and Dynamic Time Warping". *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, Bd. 42, Nr. 4, S. 1039–1052, 2012.

[VTT 14] "VTT Technical Research Centre of Finland - The ALVAR Tracking Library". 2014. `http://virtual.vtt.fi/virtual/proj2/multimedia/alvar/` (Letzter Abruf: 20.07.2015).

[Waar03a] B.J.W. Waarsing, M. Nuttin und H. Van Brussel. "A software framework for control of multi-sensor, multi-actuator systems". In: *International Conference on Advanced Robotics*, S. 41–46, 2003.

[Waar03b] B.J.W. Waarsing, M. Nuttin und H. Van Brussel. "Behavior-based mobile manipulation inspired by the human example". In: *2003 IEEE International Conference on Robotics and Automation*, S. 268–273, IEEE, 2003.

[Wasi02] Zbigniew Wasik und Alessandro Saffiotti. "A fuzzy behavior-based control system for manipulation". In: *IEEE/RSJ International Conference on Intelligent Robots and System*, S. 1596–1601, IEEE, 2002.

[Wasi03] Zbigniew Wasik und Alessandro Saffiotti. "A Hierarchical Behavior-Based Approach to Manipulation Tasks". In: *Proceedings of 2003 IEEE international conference on robotics and automation*, S. 2780–2785, Taipei, 2003.

[Wern14] Tobias Werner und Dominik Henrich. "Efficient and Precise Multi-Camera Reconstruction". In: *Proceedings of the International Conference on Distributed Smart Cameras - ICDSC '14*, S. 1–6, ACM Press, New York, New York, USA, nov 2014.

[Wica09] Handy Wicaksono, Khairul Anam und Rusdhianto Effendi. "Modified fuzzy behavior coordination for autonomous mobile robot navigation system". In: *IEEE ICCAS-SICE*, S. 2944–2948, 2009.

[Wink09] Alexander Winkler und Jozef Suchý. "Intuitive collision avoidance of robots using charge generated virtual force fields". *Advances in Robotics Research*, S. 77–87, 2009.

[Wool09] Brian G. Woolley und Gilbert L. Peterson. "Unified behavior framework for reactive robot control". *Journal of Intelligent and Robotic Systems: Theory and Applications*, Bd. 55, S. 155–176, 2009.

[Wred13] Arne Wrede, Sebastian Emmerich, Christian Grünberg, Ricarda Nordmann, Agnes Swadzba und Jochen Steil. "A User Study on Kinesthetic Teaching of Redundant Robots in Task and Configuration Space". *Journal of Human-Robot Interaction*, Bd. 2, Nr. 1, S. 56–81, 2013.

[Wu10] Yan Wu und Yiannis Demiris. "Towards One Shot Learning by Imitation for Humanoid Robots". In: *IEEE International Conference on Robotics and Automation*, S. 2889–2894, 2010.

[Yang03] Yuandong Yang, Oliver Brock und Roderic Grupen. "Exploiting redundancy to implement multiobjective behavior". In: *Robotics and Automation*, 2003.

[Ye11] Gu Ye und Ron Alterovitz. "Demonstration-guided motion planning". In: *International Symposium on Robotics Research*, 2011.

[Zoll04] R. Zöllner, T. Asfour und R. Dillmann. "Programming by demonstration: dual-arm manipulation tasks for humanoid robots". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, S. 479–484, Ieee, 2004.

[Zoll05] R. Zöllner, M. Pardowitz, S. Knoop und R. Dillmann. "Towards cognitive robots: Building hierarchical task representations of manipulations from human demonstration". In: *Proceedings - IEEE International Conference on Robotics and Automation*, S. 1535–1540, 2005.