

# Bildschirmfreie Roboterprogrammierung für Nichtexperten mittels Roboterzustandsautomaten

Screenless Robot Programming for Non-Experts  
Using Robot State Automata

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

von  
Lukas Sauer  
aus Stuttgart

1. Gutachter: Prof. Dr. Dominik Henrich  
2. Gutachter: Prof. Dr. Jochen Steil

Tag der Einreichung: 28. August 2023  
Tag des Kolloquiums: 11. April 2024



## Danke!

Ich wäre heute nicht an dem Punkt, diese Dissertation zu publizieren, ohne eine Vielzahl an anderen Menschen. Um einige davon zu nennen, ohne Anspruch auf Vollständigkeit, danke ich:

- meinem Doktorvater Prof. Dr. Dominik Henrich, der mich überhaupt auf diesen Weg gebracht und ihn mir ermöglicht hat
- meinen Eltern, die beide maßgeblichen Einfluss darauf hatten, wie ich meinen Kopf um diese Welt biege
- meinen Kommilitonen und Kollegen am Lehrstuhl, die fachlich, technisch und auch anderweitig immer zum Gespräch bereit standen, sowie Anke und Simon, die den Lehrstuhl im Hintergrund am Laufen halten
- meinen Korrekturlesern, Gabi und Michael, ohne die mehr meiner Fehler es bis in die Arbeit geschafft hätten
- zuletzt: Sophia, meiner Liebe, die mich nach Bayreuth geführt hat – Ich habe vor langer Zeit gesagt, dass Du der Grund für alles bist, und das schreibe ich heute einmal mehr

All diese Menschen haben positiven Einfluss auf die Arbeit genommen; Alle Ecken und Kanten, die verbleiben, sind ausschließlich von mir zu verantworten.



## Zusammenfassung

In kleinen und mittleren Unternehmen (KMU) ist die Automatisierung mit Robotern eine laufende Entwicklung, und es existieren noch ungenutzte Potenziale. Ein Faktor ist dabei der Programmieraufwand: Durch kleinere Losgrößen muss sich eine häufigere Neuprogrammierung gegen den Nutzen aus kürzeren Produktionsphasen amortisieren. Daraus motiviert sich die intuitive Roboterprogrammierung, die generell den Programmiervorgang beschleunigen, aber insbesondere auch die Programmierung durch existierendes Personal (in der Regel Domänenexperten, aber Robotik-Nichtexperten) ermöglichen soll. Existierende Ansätze in diesem Bereich, aus der Forschung wie kommerziell, stützen sich zumeist auf grafische Editoren oder Assistenten. Gleichzeitig finden Leichtbauroboter zunehmend Verbreitung, da auch diese durch geringere Anschaffungskosten und geringeren Aufwand für Sicherheitsmaßnahmen neue Marktsegmente eröffnen. Meist bringen sie die Möglichkeit der kinästhetischen Führung mit sich, d.h. das Bewegen des Roboters durch das direkte Ausüben von Kräften, beispielsweise mit den Händen am Endeffektor. Daher wären Systeme von Interesse, bei denen auch die übrigen Eingaben direkt über den Roboter stattfinden können, und keine Wechsel zu anderen Eingabegeräten nötig sind. Durch den Verzicht auf Bildschirme und Eingabegeräte ließe sich auch der Hardwareaufbau im Arbeitsraum des Roboters verringern. Die vorliegende Arbeit widmet sich deshalb der Frage, inwieweit zur Programmierung von Leichtbaurobotern durch Nichtexperten auf eine grafische Schnittstelle verzichtet werden kann.

Zur Umsetzung eines solchen bildschirmfreien Systems wird in dieser Arbeit auf Zustandsautomaten zurückgegriffen. Diese basieren auf einfachen Grundprinzipien wie der schrittweisen Aneinanderreihung von Zuständen und damit verbundenen Aktionen, oder der Auswahl zwischen Folgeschritten anhand des aktuellen Aufgabenzustands. Damit sind Nutzende zugleich in der Lage, Schleifen und Verzweigungen im Kontrollfluss umzusetzen. Die in der Arbeit entwickelte Automatenvariante der Roboterzustandsautomaten bietet zentrale Fähigkeiten zur Automatisierung einfacher Aufgaben an, etwa die Spezifikation von Bewegungen in absoluten Koordinaten, relativ zur vorherigen Pose oder relativ zu Objekten im Arbeitsraum. Zur Erzeugung von Programmstruktur ohne explizite Eingabe durch die Nutzenden wird ein alternativer Ansatz vorgestellt, der Teile des Automaten auf Basis einer formulierten Ähnlichkeitsheuristik automatisch zusammenfügt. Auch eine Anwendung des Systems auf Szenarien mit mehreren Roboterarmen wird betrachtet, insbesondere der prototypisch implementierte Spezialfall von zwei Robotern. Um die Benutzung des Systems rein über den Roboter zu realisieren, wird auf sogenannte haptische Eingaben zurückgegriffen. Der Begriff beschreibt hier bestimmte Bewegungen, die während der Führung durch den Roboter erkannt werden und Operationen des Programmiersystems auslösen. Haptische Eingaben stellen ein bisher spärlich erforschtes Feld dar, weshalb dem vorgestellten Entwurf eine Vorstudie vorangeht.

Die in mehreren Studien zu verschiedenen Aspekten des Systems erhobenen Daten deuten darauf hin, dass das System auch ohne grafische Schnittstelle Nutzenden, insbesondere Nichtexperten, eine erfolgreiche Erzeugung von Roboterprogrammen ermöglicht.



## Abstract

In small and medium-sized enterprises (SME), automation with robots is an ongoing process, and there is still unused potential. One contributing factor is programming effort: Because of smaller lot sizes, a more frequent reprogramming has to pay off against the benefits from shorter production runs. This motivates the field of intuitive robot programming, which aims at a faster programming process, but also at enabling existing personnel (generally, domain experts, but robotics non-experts) to do the programming themselves.

Existing approaches, both in research and commercial, are mostly based on graphical editors or wizards. Meanwhile, lightweight robots are increasingly employed, since with their lower costs and lower requirements for additional security measures they also open up new market segments. These usually support kinesthetic guiding, that is, moving the robot by direct application of forces e.g. by hand on the endeffector. Because of this development, systems would be interesting where other inputs also happen directly via the robot, and no switching between different input devices is necessary. Doing away with screens and input devices would also reduce the amount of hardware in the robot workspace. Thus, the present work is dedicated to the question to which extent the programming of lightweight robots by non-experts can forgo a graphical interface.

The implementation of such a screenless system proposed in this work makes use of state automata. These are based on simple principles like stepwise sequencing of states and the associated actions, or selection between different succeeding steps by means of the current state of the task. They also enable users to realize loops and branchings in the control flow of their programs. Additionally, Robot State Automata, the variant developed in this work, offer key functionality to automatise simple tasks, like specifying movement in absolute coordinates, relative to the previous pose, or relative to objects in the workspace. In order to generate program structure without explicit specification by the user, an alternative approach is presented which automatically unifies parts of the automaton based on a heuristic for consistency. An application of the system to scenarios with multiple robot arms is considered, including the prototypically implemented specific case of two robots. To realize usage exclusively via the robot, so-called haptic inputs are employed. This term denotes specific movements which are detected while guiding the robot and which trigger operations of the programming system. Haptic inputs have scarcely been investigated previously, so a preliminary study precedes the actual design. Data from multiple user studies covering different aspects of the system indicate that the system, even without a graphical interface, enables users, especially non-experts, to successfully generate robot programs.





---

## Inhaltsverzeichnis

---

<b>1. Einführung</b>	<b>1</b>
1.1. Motivation . . . . .	1
1.1.1. Hintergrund . . . . .	1
1.1.2. Systemaspekte . . . . .	4
1.2. Fragestellung . . . . .	7
1.3. Abgrenzung . . . . .	8
1.4. Überblick . . . . .	9
<b>2. Stand der Forschung</b>	<b>11</b>
2.1. Roboterprogrammierung . . . . .	11
2.1.1. Eingabe von Roboterposen . . . . .	13
2.1.2. Eingabe des Programminhalts . . . . .	16
2.2. Automaten . . . . .	19
2.2.1. Abstraktionsniveau . . . . .	19
2.2.2. Automatenmodelle . . . . .	22
2.3. Fazit . . . . .	29
<b>3. Roboterzustandsautomaten</b>	<b>31</b>
3.1. Einfache Roboterzustandsautomaten . . . . .	31
3.1.1. Grundlagen endlicher Automaten . . . . .	31
3.1.2. Formalismus RSA . . . . .	33
3.1.3. Anwendungsbeispiel . . . . .	34
3.2. Erweiterte Roboterzustandsautomaten . . . . .	36
3.2.1. Formalismus ERSA . . . . .	37
3.2.2. Anwendungsbeispiel . . . . .	39
3.3. Programmierung . . . . .	42
<b>4. Haptische Eingaben</b>	<b>47</b>
4.1. Stand der Forschung . . . . .	48
4.2. Erkennung haptischer Gesten . . . . .	50
4.2.1. Vorüberlegungen . . . . .	50

4.2.2.	Einfache Ruckbewegungen . . . . .	51
4.2.3.	Komplexere Gesten . . . . .	55
4.3.	Vorstudie und Implementierung . . . . .	56
4.3.1.	Studienentwurf . . . . .	57
4.3.2.	Ergebnisse . . . . .	58
4.3.3.	Schlussfolgerungen . . . . .	63
<b>5.</b>	<b>Struktursynthese</b>	<b>67</b>
5.1.	Stand der Forschung . . . . .	68
5.2.	Markierungsalgorithmus . . . . .	69
5.2.1.	Markierungsfunktionen . . . . .	69
5.2.2.	Algorithmus . . . . .	70
5.3.	Unifikation . . . . .	71
5.3.1.	Algorithmus . . . . .	72
5.3.2.	Anwendung im Programmiersystem . . . . .	73
5.4.	Evaluation . . . . .	74
5.4.1.	Experimentelle Verifikation . . . . .	74
5.4.2.	Komplexität . . . . .	75
<b>6.</b>	<b>Mehrarmsysteme</b>	<b>81</b>
6.1.	Stand der Forschung . . . . .	81
6.1.1.	Synchronisationsvarianten . . . . .	82
6.1.2.	Automatenmodelle . . . . .	83
6.2.	Multi-ERSA . . . . .	84
6.2.1.	Definition . . . . .	84
6.2.2.	Ausführung . . . . .	86
6.2.3.	Beispiel . . . . .	86
6.3.	Programmierung im Gesamtsystem . . . . .	88
6.3.1.	Operationen . . . . .	88
6.3.2.	Interaktionen . . . . .	88
<b>7.</b>	<b>Evaluation</b>	<b>91</b>
7.1.	ERSA . . . . .	92
7.1.1.	Studienaufbau . . . . .	92
7.1.2.	Programmieraufgabe . . . . .	92
7.1.3.	Erhobene Daten . . . . .	95
7.1.4.	Ergebnisse . . . . .	96
7.2.	Multi-ERSA . . . . .	98
7.2.1.	Studienaufbau . . . . .	98
7.2.2.	Programmieraufgaben . . . . .	100
7.2.3.	Ergebnisse . . . . .	102
7.3.	Haptische Interaktionen . . . . .	104
7.4.	Vergleichsstudie mit grafischer Schnittstelle . . . . .	106
7.4.1.	Studienaufbau . . . . .	107

7.4.2. Ergebnisse . . . . .	107
7.5. Bemerkungen zu statistischer Signifikanz . . . . .	110
<b>8. Fazit</b>	<b>113</b>
8.1. Zusammenfassung und Diskussion . . . . .	113
8.2. Ausblick . . . . .	117
8.2.1. Technische Verbesserungen am existierenden System . . . . .	117
8.2.2. Breitere Datengrundlage . . . . .	117
8.2.3. Aspekte aus der initialen Abgrenzung . . . . .	118
8.2.4. Neue Stoßrichtungen . . . . .	119
<b>A. Studienergebnisse</b>	<b>121</b>
<b>B. Vorstudie zu haptischen Interaktionen</b>	<b>127</b>
<b>C. MINERIC-Fragebögen</b>	<b>141</b>
<b>Tabellenverzeichnis</b>	<b>147</b>
<b>Abbildungsverzeichnis</b>	<b>149</b>
<b>Literaturverzeichnis</b>	<b>159</b>



# KAPITEL 1

---

## Einführung

---

1.1. Motivation . . . . .	1
1.1.1. Hintergrund . . . . .	1
1.1.2. Systemaspekte . . . . .	4
1.2. Fragestellung . . . . .	7
1.3. Abgrenzung . . . . .	8
1.4. Überblick . . . . .	9

---

## 1.1. Motivation

In diesem Kapitel soll zunächst das Feld der intuitiven Roboterprogrammierung an sich motiviert werden. Dabei wird zunächst insbesondere auf die Situation kleinerer und mittlerer Unternehmen Bezug genommen. Anschließend werden die konkreten Rahmenbedingungen für die vorliegende Arbeit vorgestellt und begründet, nämlich die Erforschung eines führungsbasierten Verfahrens und der Verzicht auf eine grafische Schnittstelle.

### 1.1.1. Hintergrund

Roboter werden seit vielen Jahren erfolgreich in der Industrie eingesetzt, um Aufgaben zu automatisieren. Sie bringen eine Reihe an wesentlichen Vorteilen mit sich, wie etwa eine hohe Wiederholgenauigkeit (und damit verbunden konsistente Qualität), hohe Geschwindigkeit und kurze Taktzeiten, Abwesenheit von Ermüdungseffekten, Dauerlauffähigkeit, Einsatz in für Menschen schädlichen oder gefährlichen Umgebungen oder Unabhängigkeit von ergonomischen Bedenken. In [75] wird der Schluss gezogen, dass Roboter die Arbeit allgemein sicherer und weniger körperlich belastend machen können. Damit spielen sie in der heutigen Produktion eine zentrale Rolle [162]. Am stärksten ist dieser Effekt in der Automobilindustrie, wo z.B. die Karosseriefertigung zu über 90% automatisiert abläuft [149].

Im Einsatz in der traditionellen Industrie können sich die Kosten über Hunderttausende

bis Millionen produzierter Einheiten amortisieren, und auch eine räumliche Isolation (z.B. durch Sicherheitskäfige) ist möglich und wird in Kauf genommen [91]. Neben der Großindustrie ist der Einsatz in kleinen und mittleren Unternehmen (KMU) aber eine jüngere und noch laufende Entwicklung, die durch verschiedene Faktoren aufgehalten wurde und wird. Der Anteil an Unternehmen im verarbeitenden Gewerbe in Deutschland, der Industrieroboter benutzt, ist etwa von 2018 auf 2020 im Segment von 250 und mehr Beschäftigten um 8% von 45% auf 53% gestiegen, im Segment von 50 bis 249 Beschäftigten ebenfalls um 7% auf immerhin 27% gewachsen, hat im Segment von 10 bis 49 Beschäftigten dagegen auf 9% stagniert [147]. Dies ist insofern eine wichtige Beobachtung, als dass KMU nach [123] Stand 2019 bzw. nach [148] 2020 mehr als 99% der Unternehmen in der EU ausmachen und in den vorangegangenen fünf Jahren zwei Drittel der Beschäftigungsverhältnisse im privaten Bereich beinhalteten sowie ca. 85% der neuen Arbeitsplätze generierten. Auch allgemein machen KMU nach [75] in den meisten Wirtschaftskreisen mehr als 90% der Unternehmen aus. Für die US-Wirtschaft beschreibt [91], dass z.B. Fertigungsbetriebe mit weniger als 250 Mitarbeitern 57% der Beschäftigungsverhältnisse in der Fertigung und 46% der Gehälter ausmachen. Entsprechend hoch sind die Bestrebungen, die Hürden zwischen KMU und innovativen Technologien zu verringern. Seitens des Bundesministeriums für Bildung und Forschung gibt es z.B. die Förderinitiative KMU-NetC<sup>1</sup>, worunter unter anderem auch Automatisierungsprojekte fallen<sup>2</sup>.

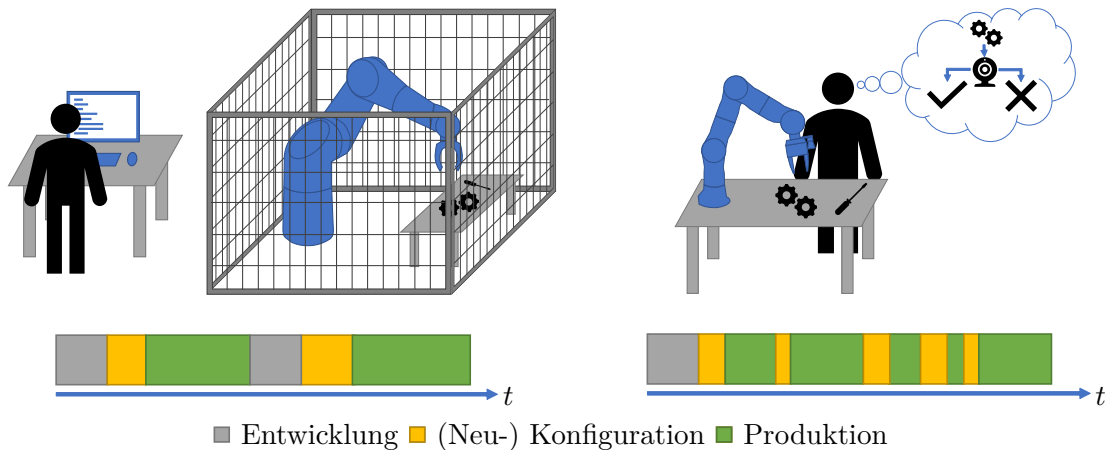
Charakteristische Eigenschaften von KMU sind häufige Produktwechsel, große Variantenvielfalt und kleine Losgrößen [63, 123]. Diese Kombination, die nicht ausschließlich, aber insbesondere auf KMU zutrifft, führt dazu, dass beispielsweise in Deutschland die Mehrheit der Montageaufgaben noch manuell gelöst wird [18]. Für den Einsatz von Robotern ist als Konsequenz eine häufige Neuprogrammierung erforderlich. Die Programmierung traditioneller Roboter erfordert zumeist spezialisierte Experten [162]. Gleichzeitig haben insbesondere kleine Unternehmen selten dedizierte Informatik- und Robotikangestellte, die diese Programmierung übernehmen könnten [91, 123]. Stattdessen besteht das Personal im Wesentlichen aus Domänenexperten, die Fachwissen darüber haben, wie die Aufgaben manuell zu lösen sind. Nur die Einbindung dieser vorhandenen Arbeitskräfte macht eine effiziente Neuprogrammierung und damit einen effizienten Einsatz möglich [149].

Infolgedessen beschäftigt sich ein großer Forschungszweig mit Verfahren zur intuitiven Programmierung von Robotern. Durch solche Verfahren soll eine Programmierung direkt durch die vorhandenen Mitarbeitenden ermöglicht werden, ohne dass diese dafür erst Expertenwissen außerhalb ihrer Domäne erwerben müssen. Gleichzeitig soll der Aufwand für die Neuprogrammierung auch anderweitig reduziert werden, insbesondere die benötigte Zeit (sowohl Arbeitszeit als auch Stillstand der involvierten Anlagen) [91]. In Abbildung 1.1 wird der Kontrast zwischen traditioneller Industrierobotik und der Vision

---

<sup>1</sup>[https://www.bmbf.de/bmbf/shareddocs/bekanntmachungen/de/2016/08/1235\\_bekanntmachung.html](https://www.bmbf.de/bmbf/shareddocs/bekanntmachungen/de/2016/08/1235_bekanntmachung.html), besucht: 11.11.2022

<sup>2</sup><https://www.bmbf.de/bmbf/shareddocs/kurzmeldungen/de/automatisierung-fuer-kmu-kollege-roboter-an-der-hobelbank.html>, besucht: 11.11.2022

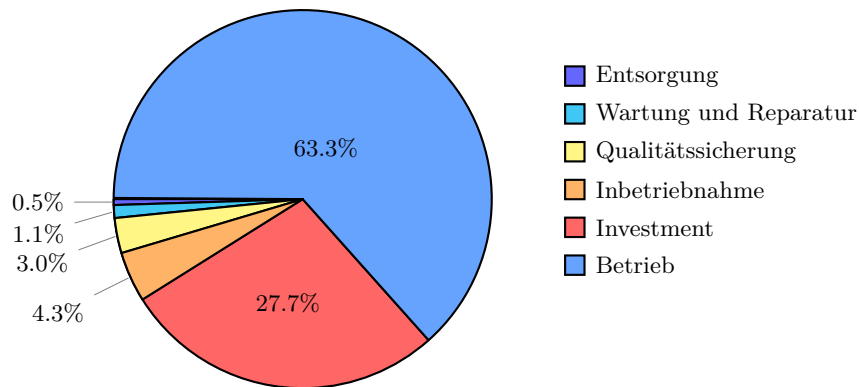


**Abbildung 1.1.:** Konzeptgrafik zu klassischer Industrierobotik (links) gegenüber der Vision von Robotik in kleineren Unternehmen (rechts). Oben: typisches Verwendungsszenario. In der klassischen Industrierobotik ist der Roboter räumlich abgezüngelt, und die vorwiegend textuelle Programmierung erfordert hohe Expertise. In der modernen Anwendung kann der Mensch sich im Arbeitsraum des (Leichtbau-) Roboters aufhalten, und die Programmierung soll auch für Nichtexperten unter Kenntnis der Aufgabe machbar sein.

Unten: Zeitlicher Verlauf von Nutzungsphasen, nach [44]. In der klassischen Industrierobotik liegen zwischen den Phasen der Neukonfiguration oder -programmierung lange Produktionsperioden. In kleineren Unternehmen sind häufigere Aufgabenwechsel erforderlich.

zukünftiger Anwendungen nochmals visualisiert. Solche Verfahren, die die Interaktion mit dem Roboter in geringerer Zeit, bestenfalls auch mit geringerem Fehlerpotenzial, ermöglichen, werden gemeinhin als Schlüsselkomponente für den weiteren Fortschritt der Automatisierung identifiziert [51, 91, 106, 162].

Die intuitive Roboterprogrammierung hat in dieser Hinsicht großes Potenzial, die Automatisierung in KMU voranzutreiben. Wenngleich die genauen Zahlen variieren, werden gemeinhin die Gesamtkosten beim Einsatz von Robotern durch Kosten abseits der initialen Investition in die Hardware dominiert, wie (für eine kooperative Roboterzelle) auch in Abbildung 1.2 dargestellt [123]. Nach [75] liegt der Anteil von Software und Systemintegration bei etwa 60 bis 75 Prozent der Gesamtkosten. Auch in [18] wird der Preis des Roboters als nur etwa ein Drittel der Gesamtkosten eingeschätzt, dasselbe Verhältnis wie in [91]. In [16] ist die Kostenaufteilung Stand 2014 gelistet, wobei ca. 25% auf die reine Roboterhardware anfallen, dagegen ca. 35% für Leistungen wie Programmierung und Installation. Dieser Schwerpunkt in der Kostenverteilung wird potenziell noch verstärkt durch den Trend, dass sich der Einsatz von Robotern weiter verbreitet und in Verbindung damit auch die Geräte kostengünstiger werden. Prognosen gingen 2019 von einer Steigerung des Angebots an Industrierobotern um ca. 13% im Jahr aus [162]. Auch in [149] wurde für das Jahr 2022 ein Zuwachs von ca. 580.000 neu installierten Einheiten vorausgesagt. Der erfasste Bestand von 2011 bis 2021 zeigt dazu passend eine stetige Steigerung [74]. Der Gesamtumsatz mit Industrierobotern lag 2018 bei 13.85 Mrd. US-Dollar, mit ebenfalls steigender Prognose [158]. Daneben sind nach Schätzungen die Preise für



**Abbildung 1.2.:** Kostenverteilung für eine kooperative Roboterzelle, übersetzt und adaptiert aus [123]. Betriebskosten stellen mit Abstand den größten Posten dar.

Industrieroboter etwa von 2014 bis 2017 um ca. 25% gesunken [75]. Die Gesamtkosten des Einsatzes sind für die Verbreitung von Robotern stark relevant, da Unternehmen in der Regel erwarten, dass sich Investitionen über einen gewissen Zeitraum refinanzieren, häufig ein bis wenige Jahre [18]. Aktuell sind neben finanziellen Überlegungen daher auch schwerer quantitativ zu bewertende Nebeneffekte wie weniger Monotonie oder bessere Ergonomie für die menschlichen Mitarbeitenden noch vermehrt Anreize für die Automatisierung [18]. Im Gegenzug ist der Bedarf an Experten und Expertenwissen noch ein Haupthemmnis [80]. Auch in [91] werden die Kosten (initial wie Folgekosten) gemeinsam mit der Lernkurve als häufiges Hindernis beschrieben. Für kleine Unternehmen werden als Haupthindernisse in [63] zu kleine Losgrößen, hoher Kosten- und Zeitaufwand sowie der Bedarf an Expertenwissen genannt. Damit lässt sich sagen, dass intuitive Roboterprogrammierung weithin als eine wichtige Entwicklung erachtet wird. Auf die US-Wirtschaft bezogen, wird intuitiveren Schnittstellen für Robotik eine geschätzte Kostenersparnis von 7 Mrd. US-Dollar bescheinigt [91].

### 1.1.2. Systemaspekte

Eine wichtige Rolle bei der Erschließung von KMU für die Automatisierung (und umgekehrt) kommt der ebenfalls jüngeren Entwicklung von kollaborativen Robotern, kurz *Cobots* (für engl. *collaborative robots*) [162], zu. Diese sind typischerweise leichter und kleiner als herkömmliche Industrieroboter. Dadurch, und durch integrierte Sicherheitsmechanismen beispielsweise auf Basis integrierter Kraft- und Momentensensorik, kann teilweise auf bisher nötige Sicherheitsvorkehrungen im Einsatz verzichtet werden [162].<sup>3</sup> Bei diesen Robotern liegen in der Regel auch geringere maximale Traglasten vor, die aber

<sup>3</sup>Die Begriffe Cobot und Leichtbauroboter werden beide teilweise für Roboter verwendet, die in ihrer Größe menschlichen Armen ähneln, leicht bewegt werden können und sich ohne Sicherheitszaun einen Arbeitsraum mit Menschen teilen können [18]; so auch in dieser Arbeit.



dennoch für viele Aufgaben genügen [18].<sup>4</sup> Die geringere Traglast senkt wie das geringere Eigengewicht das Gefahrenpotenzial [149]. Neben direkten Kosten für Sicherheitsvorkehrungen sinkt so auch der räumliche Mehrbedarf, der ansonsten insbesondere im direkten Vergleich mit einem menschlichen Arbeitsplatz anfällt [123]. Die ähnliche Größe und Traglast zu menschlichen Arbeitenden wird zudem als Vorteil dabei angesehen, um Cobots in existierende Produktionslinien zu integrieren [91]. Der Markt für Cobots ist inzwischen so weit, dass die meisten Roboterhersteller Modelle in diesem Bereich anbieten [162]. Die Preise sind ebenfalls deutlich niedriger als für größere Industrieroboter, beginnend im Bereich von etwa 15.000€. Solche geringeren Kosten und erhöhte Flexibilität werden als zentrale Vorteile von Cobots genannt [80]. Auch [91] kommt zu der Einschätzung, dass Cobots zusammen mit der allgemeinen Preisentwicklung für Roboter KMU zunehmend erlauben, risikoarm in die Automatisierung einzusteigen.

Auch seitens der Unternehmen selbst besteht großes Interesse an dieser Entwicklung. In einer Onlineumfrage im Jahr 2014 gaben 86% der Teilnehmenden an, in Leichtbauroboter investieren zu wollen, und in der späteren Industriebefragung von 2016 wurden 25 Anwendungen vorgestellt, bei denen Leichtbauroboter bereits im Einsatz oder kurz davor waren [18]. Auch in [80] gaben zwei Drittel der befragten Industrieexperten an, dass Cobots in ihren Unternehmen schon eingesetzt würden oder ein Einsatz grob im nächsten Jahr geplant sei.

Die Möglichkeit für Nutzende, mit Cobots im selben Arbeitsraum zu sein und direkt zu interagieren, ist ein wichtiger Faktor, um die Forderung nach schnellen und unkomplizierten Änderungen an Programmen zu ermöglichen [63] und somit eine direkte Motivation für deren Entwicklung, wie am Beispiel des Kuka LWR in [21] genannt. Durch den Wegfall einer räumlichen Trennung können Programmierung und Ausführung im Gegensatz zu traditionellen Industrierobotern in einfachen, kurzen Zyklen vor Ort stattfinden [162]. Solch ein verwobener Prozess in Anwesenheit der Nutzenden bildet einen Kernaspekt des in dieser Arbeit vorgestellten Systems.

Für den Entwurf intuitiver Programmierschnittstellen ist es wichtig, die Interaktionen mit dem Roboter einfach zu gestalten. Sowohl der Lernaufwand sollte minimiert werden, als auch die mentale Kapazität, die während der Benutzung auf die Interaktion mit dem System aufgewendet werden muss. Diese Zwecke erfordern neue Programmierverfahren und Interaktionsmodalitäten. Informationsengpässe und Limitationen klassischer Schnittstellen, wie Tastatur und Maus, Touchscreen oder Handprogrammiergerät, sollen umgangen werden [162]. Neue Interaktionsmöglichkeiten, beispielsweise haptisch oder multimodal, werden auch in [149] in Aussicht gestellt. Der Bedarf an natürlichen und intuitiven Interaktionen wird auch in [91] herausgestellt, um Robotern zu breiterer Anwendung zu verhelfen, insbesondere, ohne textuell Programmcode zu schreiben.

Die Führung von Robotern von Hand durch direkte Krafteinwirkung, als *kinästhetische Führung* bezeichnet, bietet eine Basis vieler neu entwickelter Ansätze, ein Spezialfall des Spektrums an Mensch-Roboter-Kollaboration [162]. Sie senkt die technischen Anforderungen an Anwendende und wurde in einer Studie von Experten unter einer Reihe möglicher

---

<sup>4</sup>Teils werden Leichtbauroboter auch genau über das höhere Verhältnis von maximaler Traglast und Eigenmasse charakterisiert [4, 101].

neuartiger Schnittstellen bevorzugt [80]. In einer weiteren Studie wurde verifiziert, dass kinästhetische Führung tatsächlich einfach zu erlernen ist, nicht signifikant beeinflusst durch Faktoren wie Alter oder Expertise [105]. Cobots sind durch die auch zu Sicherheitszwecken verbaute interne Sensorik dazu meist gut geeignet [162]. Die Handführung stellt sowohl ein sogenanntes *Natural User Interface* (NUI) als auch ein *Tangible User Interface* (TUI) dar, da auf existierende Erfahrungen von Menschen beim Umgang mit anderen Lebewesen zurückgegriffen wird, und Nutzende die ganze Zeit über (greifbares) Feedback über die wahrgenommene Pose des Roboters erhalten (durch dessen reale Pose) [162]. Handführung wird in [63] als intuitive und effiziente Eingabemethode für Roboterposen bewertet, sowie als zentrale Fähigkeit von Robotersystemen für KMU. Darüber hinaus wird bei der physischen Interaktion mit dem Roboter ein positiver Effekt auf die Akzeptanz seitens der Anwendenden vermutet [106]. Auch diese Arbeit beschäftigt sich mit einem Ansatz, der essenziell auf der Handführung des Roboters aufbaut. Zusätzlich können auch andere Ansätze von NUI bzw. TUI z.B. auf Basis von Gesten einfache Kommunikationswege zwischen Mensch und Roboter darstellen [80]. Diese Erkenntnis motiviert die Untersuchung haptischer Eingaben, die im Lauf der Arbeit vorgestellt wird: Bestimmter Gesten oder Bewegungen des Roboters, die während der Führung erkannt werden.

In dieser Arbeit nicht betrachtet werden dagegen andere alternative Eingabemodalitäten. So kann etwa auch die Verwendung von menschlichen Nutzenden geäußerter Anweisungen, in natürlicher Sprache, ein NUI darstellen. Der Einsatz in der realen Anwendung scheitert dabei meist an fehlender Robustheit, sowohl auf Seiten der Spracherkennung (insb. in lauten Produktionsumgebungen) als auch des Sprachverstehens (natürliche Sprache ist notorisch uneindeutig und kontextabhängig) [162]. Trotz aktiver Forschung in diesem Bereich grenzt sich die vorliegende Arbeit daher in diese Richtung ab.

Auch taktile<sup>5</sup> Eingaben, beispielsweise über eine künstliche sensorische Haut am Roboter, stellen eine Alternative zu herkömmlichen Schnittstellen dar [84]. Das Ziel in dieser Arbeit ist jedoch die Verwendung des Roboters selbst zur Eingabe, ohne spezielle Zusatzhardware, aufgrund von Kosten, Platzbedarf im Arbeitsraum, Verfügbarkeit/Verallgemeinerbarkeit und ähnlicher Aspekte. Auch taktile Schnittstellen werden daher explizit außen vor gelassen.

Grafische Schnittstellen zur Erzeugung von Roboterprogrammen stellen eine Verbesserung gegenüber traditioneller Eingabe dar [91]. Andererseits werden Eingabemethoden wie Maus und Tastatur oder Touchscreens auch als Engpässe in der Interaktion mit Robotern bewertet [162]. Darüber hinaus ist auch zusätzliche Belegung des Arbeitsraums ein Hemmnis beim Einsatz von Robotern in KMU, und ein Ziel sollte sein, diese zu minimieren [123]. In [63] wird ebenfalls kompakte Zusatzhardware als Anliegen von Anwendenden aufgeführt. Daher untersucht die vorliegende Arbeit einen Ansatz, der ohne grafische Schnittstelle arbeitet und die Interaktion rein über den Roboter ermöglicht.

---

<sup>5</sup>Zur Verwendung der Begriffe *taktil*, *haptisch* und *kinästhetisch* in dieser Arbeit erfolgt in Kapitel 4 eine genaue Einordnung; Hier ist die Rede von Schnittstellen, die Oberflächenberührung wahrnehmen.

## 1.2. Fragestellung

Auf Basis der im letzten Abschnitt präsentierten Motivation wird in dieser Arbeit ein „bildschirmfreies“ Verfahren zur Programmierung von Leichtbaurobotern für Nichtexperten untersucht, das also nicht auf eine grafische Schnittstelle zurückgreift. Die erste und übergreifende Frage lautet daher:

**F1** Inwieweit sind Robotik-Nichtexperten als Endanwendende in der Lage, Roboter ohne grafische Schnittstelle zu programmieren?

In der Evaluation werden dazu tatsächlich zwei Aspekte betrachtet, nämlich erstens die Anwendbarkeit eines Systems ohne grafische Schnittstelle an sich, und zweitens der Einfluss von Robotik-Vorerfahrung auf die Benutzbarkeit.

Zur Beantwortung dieser Frage (bzw. Fragen) kann die vorliegende Arbeit naturgemäß nur in beschränktem Maß beitragen. Das Ausmaß, in dem Anwendende mit dem vorliegenden System ohne grafische Schnittstelle erfolgreich sind, stellt dabei zumindest eine erste untere Schranke dar, die von zukünftigen Arbeiten weiter verbessert werden kann.

Einige Aspekte des Systems können schon zu Beginn festgelegt werden. Das beschriebene System verwendet die Handführung aufgrund der ihr bescheinigten Intuitivität als zentrale Interaktionsmodalität. Sowohl dadurch als auch durch den geringeren Kostenfaktor sind die Zielplattformen Leichtbauroboter.

In ihrem zugrunde liegenden Formalismus basiert diese Arbeit auf einer einfachen Variante des Modells der endlichen Automaten, welche in Richtung der gewählten Anwendung erweitert wird. In dem Zusammenhang stellt sich die weitere Frage:

**F2** Mit welcher Funktionalität lässt sich das ursprüngliche Modell erweitern? Mit welchen Operationen können Nutzende solche erweiterten Automaten anlegen?

Da wie beschrieben die Interaktion möglichst rein über den Roboter stattfinden soll, beschäftigt sich diese Arbeit auch mit der Interaktionsmodalität haptischer Gesten, also bestimmter mit dem Roboter durchgeführter Bewegungen, die während der Führung detektiert werden. In diesem Feld existieren bisher nur wenige Vorarbeiten. Daher ist ein weiterer Teil der Fragestellung:

**F3** Mit welchen haptischen Gesten lassen sich die nötigen Programmieroperationen umsetzen? Welchen Einfluss haben haptische Eingaben auf die Intuitivität des Systems?

Daneben wird in der Evaluation innerhalb einer Vergleichsstudie auch eine der Annahmen ausgelotet, die die Programmierung über den Roboter selbst überhaupt erst motivieren:

**F4** Welchen Einfluss haben Wechsel der Eingabemodalität und der dadurch anfallende Mehraufwand auf die Benutzbarkeit?

Einige weitere Fragen beschäftigen sich mit Erweiterungen des Grundsystems. Als Vorgriff in den späteren Verlauf der Arbeit hat sich die Programmierung von Schleifenstrukturen für die Anwendenden als eine Schwierigkeit herausgestellt. Daher wird als Fragestellung mit aufgegriffen:

**F5** Wie können sich wiederholende Strukturen in den Automaten erkannt und ergänzt werden?

Des Weiteren gibt es in der Praxis auch Aufgaben, die sich mit einem einzelnen Roboterarm nicht lösen lassen, oder bei denen die Lösung wesentlichen Zusatzaufwand beispielsweise an Haltevorrichtungen oder Spezialwerkzeugen erfordern würde. Schon in reinen Handhabungsaufgaben können insbesondere bei Leichtbaurobotern Probleme dadurch auftreten, dass Objekte zu schwer oder zu sperrig sind, um sie mit einem einzelnen Arm zu greifen. Deshalb wird auch eine Adaption des Ansatzes in dieser Richtung untersucht:

**F6** Wie lässt sich der vorgestellte Ansatz in Automatenmodell und Programmierung auf mehrere Roboter erweitern? Inwieweit sind Nutzende damit in der Lage, mehrere Roboter ohne grafische Schnittstelle zu programmieren?

### 1.3. Abgrenzung

Der in dieser Arbeit untersuchte Ansatz zur automatenbasierten intuitiven Roboterprogrammierung arbeitet auf der Ebene von Grobbewegungen, also solchen im Freiraum, die anhand von Gelenkwinkeln charakterisiert sind. Es gibt darüber hinaus eine Vielzahl an Anwendungen, für die Feinbewegungen nötig oder wünschenswert wären. Dabei handelt es sich dann um Bewegungen im Kontakt mit der Umwelt, was über die Positioniergenauigkeit des Roboters hinausgeht und daher Kraft- und Momenteninformation verwendet. Typischerweise sind solche Fähigkeiten für die Oberflächenbearbeitung oder für Montageschritte wie Fügen notwendig.

Feinbewegungen stellen einen hochgradig relevanten Forschungsbereich dar, der allerdings weitaus genug Raum für eigene Dissertationen bieten würde und damit den Rahmen der vorliegenden Arbeit deutlich sprengt. Infolgedessen ist der dargestellte Ansatz auf Grobbewegungen beschränkt. Es bestehen auch unter dieser Einschränkung ausreichend Anwendungsszenarien; beispielsweise wurden in [18] das Greifen von einem oder mehreren Bauteilen, Pick-and-Place, das Beladen von Maschinen, Packen und Palettieren als auftretende Anwendungen von jeweils zwischen 20 und über 50 Prozent der Teilnehmenden gewählt. Auch in den Daten aus [91] machen Handhabung und Maschinenbestückung über ein Drittel der Anwendungen aus.

Ein vielversprechender Ansatz, Roboter durch Nichtexperten verwendbar zu machen, ist das *Programmieren durch Vormachen* (PdV, auch PbD für engl. *Programming by Demonstration*). Dabei findet keine explizite Programmierung statt, sondern Nutzende demonstrieren (Lösungen der) Aufgaben, etwa durch Führen des Roboters, ebenfalls kinästhetisch, mit einer Masterkinematik, oder direkt durch Motion Tracking. Aus den aufgenommenen Daten werden dann Roboterprogramme bzw. dessen Ansteuerung automatisch generiert. PdV-Ansätze versprechen, eine verallgemeinerbare Lösung zu lernen. Dafür benötigen sie aber häufig mehrfache Demonstrationen der Aufgabe. Es ist auch für Nutzende weniger direkt kontrollierbar und nachvollziehbar, in welcher Weise das Roboterprogramm aus ihren Aktionen entsteht. Für kleinere Aufgaben ist im Gegenzug

eine einmalige, explizite Programmierung angemessener, und auch in Zukunft haben Verfahren zu diesem Zweck ihre Daseinsberechtigung. Genau auf solche Aufgaben zielt die vorliegende Arbeit ab. Verfahren im Bereich PdV sind ein gänzlich eigenes Forschungsfeld, und werden in diesem Rahmen nicht weiter betrachtet.

## 1.4. Überblick

Im weiteren Verlauf dieser Arbeit wird zunächst in Kapitel 2 auf den allgemeinen Stand der Forschung eingegangen. Dabei geht es erstens um verwandte Arbeiten im Feld der Roboterprogrammierung, unter den beiden Gesichtspunkten der Eingabe von Posen und der Eingabe des eigentlichen Programminhalts. Zweitens wird ein Überblick über die Verwendung von endlichen Automaten und verschiedenen Varianten davon im Bereich der Robotik gegeben.

In Kapitel 3 werden der Ansatz und das Automatenmodell entwickelt, welche dem Rest der Arbeit zugrunde liegen, nämlich die *Erweiterten Roboterzustandsautomaten* (engl. *Extended Robot State Automata*, kurz ERSA). Dabei werden zuerst die *Einfachen Roboterzustandsautomaten* aus einem herkömmlichen Formalismus für endliche Automaten entwickelt. Nach der formalen Definition der ERSA folgt dann eine Beschreibung des grundlegenden Programmiersystems.

Als nächstes wird in Kapitel 4 die Eingabemodalität der haptischen Interaktionen beschrieben, also die Erkennung bestimmter Bewegungen während der Führung zum Auslösen von Programmieroperationen. Dazu gibt es eine gesonderte Einführung in den Stand der Forschung und eine kurze Übersicht über Varianten der Definition und Erkennung solcher Gesten. Weiterhin wird eine durchgeführte Vorstudie beschrieben, auf Basis deren Ergebnisse die konkrete Implementierung im ERSA-System gestaltet wurde.

Kapitel 5 beschäftigt sich mit einem Verfahren zur automatischen Erzeugung von Programmstruktur in den erzeugten Automaten. Auch hier werden zunächst verwandte Arbeiten vorgestellt. Anschließend folgen die beiden algorithmischen Komponenten, nämlich der Markierungsalgorithmus, mit dem Übereinstimmung oder Widerspruch auf Zustandspaaren bestimmt wird, und die eigentliche Unifikation für Zustände zu geschlossenen Strukturen. Weiterhin wird die Evaluation aus theoretischer Sicht und auf Beispielaufgaben angesprochen.

Mit Kapitel 6 wird der ERSA-Ansatz zur Anwendung mit mehreren Roboterarmen erweitert. Nach der Einordnung in das Feld der Multiroboterprogrammierung und dort verwendeter Synchronisationsmechanismen wird das Multi-ERSA-Modell beschrieben, in dem sich solche Programme ausdrücken und umsetzen lassen, wiederum aufgegliedert in den Formalismus und das Programmierverfahren.

Kapitel 7 stellt dann die Experimente vor, mit denen die verschiedenen Aspekte der Arbeit evaluiert wurden. Diese bestehen aus einer Studie zur Programmierung von ERSA, einer Studie zur Programmierung von Multi-ERSA, und den im Rahmen der zweiten Studie erfassten Daten bezüglich der haptischen Interaktionen. Zusätzlich wird in einer Vergleichsstudie die Programmierung der ERSA-Aufgabe mittels einem existierenden kommerziellen System, der Franka Emika Desk Schnittstelle, in mehreren Varianten

betrachtet.

Die Arbeit schließt in Kapitel 8 mit einem Rückblick und einer Diskussion der gewonnenen Ergebnisse. Zuletzt wird ein Ausblick über mögliche weitere Forschungsrichtungen gegeben, die an das Präsentierte anschließen könnten.

---

2.1. Roboterprogrammierung . . . . .	11
2.1.1. Eingabe von Roboterposen . . . . .	13
2.1.2. Eingabe des Programminhalts . . . . .	16
2.2. Automaten . . . . .	19
2.2.1. Abstraktionsniveau . . . . .	19
2.2.2. Automatenmodelle . . . . .	22
2.3. Fazit . . . . .	29

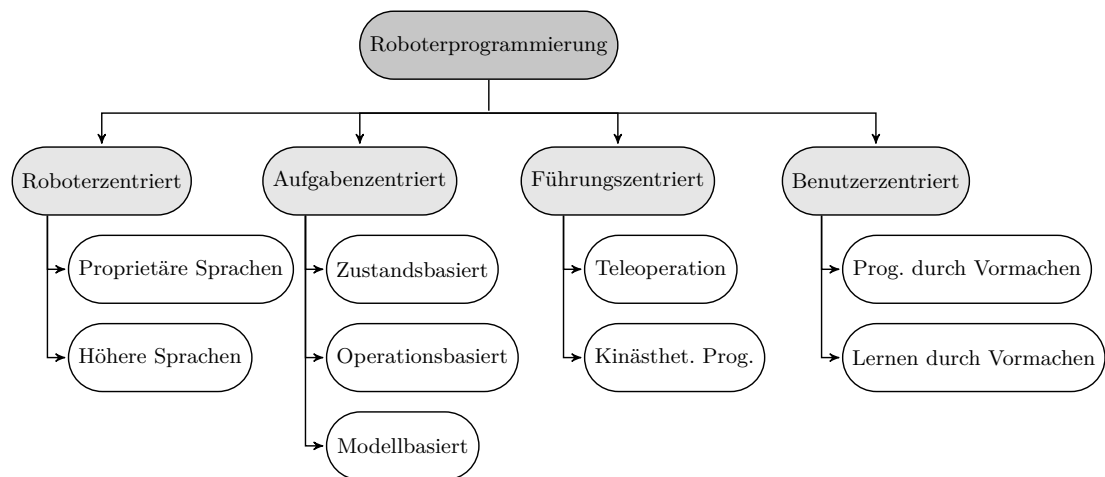
---

In diesem Kapitel wird auf allgemeine verwandte Arbeiten eingegangen, die das Thema der Arbeit generell betreffen. Dabei gliedert sich der Inhalt in einen Überblick zur Roboterprogrammierung in Abschnitt 2.1 und eine anschließende Übersicht über die Verwendung von Automaten in der Roboterprogrammierung in Abschnitt 2.2. Zuletzt werden die gesammelten Erkenntnisse in Abschnitt 2.3 nochmals zusammengefasst.

## 2.1. Roboterprogrammierung

Die Programmierung von Robotern stellt sich schon genauso lange als Problem, wie es Roboter selbst gibt. Entsprechend ist die Fülle an Arbeiten, die sich in irgendeiner Weise mit diesem Thema beschäftigen, mehr als erschlagend. Einen frühen Überblick liefert [93], während [19, 111, 117] aktuellere Übersichtsarbeiten zu diesem Feld darstellen. Ein zweigeteilter Blick aus industrieller Sicht findet sich in [138] (zur Eingabe von Pfaden) und [137] (zur Erzeugung von Programminhalten). Weiterhin beschäftigt sich [162] in Teilen damit, genau wie [64, 83, 124, 173] in [145]. Im Speziellen ist die vorliegende Arbeit auf die Anwendung durch Nichtexperten ausgerichtet. Auch dies ist ein aktives Forschungsfeld, wozu sich etwa in [1] ein Überblick finden lässt.

Im den folgenden Abschnitten sollen die üblichen Ansätze der Roboterprogrammierung kurz vorgestellt und für den Anwendungsfall in dieser Arbeit bewertet werden. Dabei lassen sich die Methoden nach unterschiedlichen Kriterien einordnen. Eine Möglichkeit ist



**Abbildung 2.1.:** Unterteilung der Roboterprogrammierung nach Zentrierungen (Roboter, Aufgabe, Führung, Benutzer) und deren Unterkategorien, nach [111]

die Aufteilung nach verschiedenen *Zentrierungen* [64, 93, 111]. Dabei wird unterschieden, um welchen von mehreren Aspekten sich der Prozess der Programmierung fokussiert. In [111] werden die Ansätze, basierend auf [93], in Roboterzentriert, Aufgabenzentriert, Führungszentriert und Benutzerzentriert aufgeteilt. Die Unterteilung ist grafisch in Abbildung 2.1 dargestellt. In [64] wird ein feinerer Fokus anhand des Roboters angelegt, in die Fälle Produktzentriert, Prozesszentriert, Werkzeugzentriert, Armzentriert und Gelenkzentriert. Beide dieser Einordnungen sind allerdings für die Zwecke dieser Arbeit nicht ideal. Für viele der im Folgenden genannten Ansätze ist die Verwendung in diesem Kontext nicht eindeutig, und letztlich nicht entscheidend. So werden Aspekte der Führungszentrierung, insbesondere die kinästhetische Eingabe, aus praktischen Gründen an vielen Stellen genutzt, auch wenn die Programmerzeugung sich ansonsten eigentlich auf einen anderen Aspekt konzentriert.

Andere Einordnungen folgen grundlegend der Frage, ob die Programmierung unter Nutzung des realen Roboters erfolgt, was als *On-line-Programmierung* betitelt wird, oder nicht, was auch als *Off-line-Programmierung* bezeichnet wird [111, 162]. Off-line-Ansätze haben den Vorteil, dass der reale Roboter während der Programmierzeit noch unabhängig zur Verfügung steht. Damit kann beispielsweise die Produktion bis zum Programmwechsel länger weiterlaufen. Ein Nachteil ist andererseits, dass Off-line erzeugte Programme auf realen Anlagen in der Regel erst validiert und gegebenenfalls angepasst werden müssen. Je weniger Anpassungen vorgenommen werden sollen, desto präziser muss der ursprüngliche Programmentwurf gewesen sein, was je nach Verfahren z.B. detaillierte Simulationsumgebungen erfordert. Die Vorteile von On-line-Verfahren sind symmetrisch dazu: Da die Eingabe unter Nutzung des realen Roboters erfolgt, ist sofort ersichtlich, ob der Roboter das Programm auch korrekt abfahren kann. Dafür kann während der Programmierzeit die Anlage nicht anderweitig genutzt werden.

Nützlich für den Kontext dieser Arbeit ist jedoch vor allem eine Betrachtung des



(mit dem On-line/Off-line-Aspekt zusammenhängenden) Gesichtspunkts der verfügbaren Eingabemodalitäten. Diese können in ihren typischen Eigenschaften beschrieben und u.a. hinsichtlich der Eignung für Nichtexperten bewertet werden. Hierbei lässt sich eine Unterscheidung treffen in die Eingabe von Roboterposen und die Erstellung des eigentlichen Programms, also etwa die Eingabe konkreter Anweisungen und Programmstrukturen.

### 2.1.1. Eingabe von Roboterposen

Die Eingabe von Posen kann direkt numerisch erfolgen, z.B. als Gelenkwinkelstellungen des Roboters oder als Transformation des Endeffektors, aus der über die inverse Kinematik Gelenkwinkel berechnet werden können. Die Spezifikation von roboterspezifischen Angaben (wie Gelenkwinkeln) statt aufgabenspezifischen (wie der Endeffektorpose) erfordert eine mentale Abbildung durch die Nutzenden selbst, was zum Aufwand der Programmierung beiträgt [110].

Unabhängig von der Variante ist eine numerische Eingabe potenziell präzise und kann von Experten sehr schnell durchgeführt werden, auch Off-line. Im Gegenzug ist sie für Nichtexperten als problematisch zu bewerten: Eine genaue Kenntnis des Roboters (etwa seiner Kinematik) wird vorausgesetzt. Ganz allgemein besteht ein großes Potenzial für Fehler, da gegebenenfalls auch Posen angegeben werden können, die Kollisionen des Roboters mit sich selbst oder der Umwelt erzeugen, oder (bei Endeffektortransformationen) für den Roboter gar nicht erreichbar sind. Als teilweise Abhilfe kann ein simulierter Roboter verwendet werden, um die resultierenden Posen zu visualisieren. Das Problem der erforderlichen Vorkenntnisse bleibt dabei dennoch grundsätzlich bestehen. Weiterhin ist eine getreue Simulation mit hohem Aufwand verbunden, insbesondere bei Repräsentation der Umgebung.

Unter Benutzung des realen oder eines simulierten Roboters können Posen auch inkrementell spezifiziert werden, also z.B. dadurch, dass die einzelnen Gelenke nacheinander in ihre entsprechende Endposition verfahren werden. Auch Endeffektorposen können inkrementell angefahren werden, wobei diese wiederum über die inverse Kinematik zu Gelenkwinkelposen aufgelöst werden müssen. Die Eingaben finden dabei zumeist über ein Handbediengerät statt, sei es über einzelne Tasten, einen Touchscreen oder auch eine 6D-Maus. Die Anwendbarkeit auf Nichtexperten ist prinzipiell ähnlich wie bei der numerischen Eingabe mit einem simulierten Roboter zur Visualisierung. Die Eingabe ist hier direkt nachvollziehbar. Andererseits ist der Prozess, die einzelnen Gelenke einzustellen, langwierig. Auch hier ist fehlende Kenntnis des Roboters ein deutlicher Nachteil, etwa wenn für jedes Gelenk erst ausprobiert werden muss, welche Drehrichtung ein positiver Winkel angibt.

Zuletzt besteht On-line auch die Möglichkeit der *kinästhetischen Programmierung* oder *Führung*. Dabei wird der Roboter unter nachgiebigen Gelenken von den Nutzenden direkt mit der Hand bewegt [111]. Üblicherweise (insbesondere für größere Roboter) kompensiert dabei der Roboter sein eigenes Gewicht, sodass er ohne extern ausgeübte Kraft einfach in der jeweiligen Pose verbleibt. Dies wird auch als *Gravitationskompensation* bezeichnet [116].

Die kinästhetische Programmierung hat den großen Vorteil, dass sie keine besonderen

	Nötige Ressourcen	Mögliche Fehler:		
		Ungültige Pose	Selbstkollision	Kollision Umwelt
Gelenkwinkel spezifizieren	–	✓	✓	✓
Endeffektorpose spezifizieren	Geometrie Roboter	✗	✗	✓
Kinästhetische Eingabe der Pose	Dynamik Roboter, realer Roboter	✗	✗	✗
Nötiges Zusatzwissen zur Fehlervermeidung		Anzahl Gelenke, Winkelgrenzen	Geometrie Roboter	Geometrie Umwelt
		Abdeckbar durch Visualisierung Roboter		
		Abdeckbar durch vollständige Simulation oder Durchführung On-line		

**Tabelle 2.1.:** Übersicht über verschiedene Varianten der Poseneingabe, welche Ressourcen sie seitens des Systems voraussetzen, und welche Fehler damit an sich möglich sind. Die mit ✗ markierten Einträge bedeuten, dass Fehler dieser Art bei einer bestimmten Eingabeart automatisch vermieden werden können; So findet die kinästhetische Eingabe etwa immer in der realen Umgebung statt, sodass Posen, die mit der Umwelt kollidieren, auffallen.

Durch zusätzliches Wissen (seitens der Nutzenden oder seitens des Systems) können Fehler erkannt werden. In der untersten Zeile ist aufgeführt, welches Zusatzwissen zur Vermeidung der einzelnen Fehlerarten benötigt wird. Zusätzlich sind zwei häufige Arten markiert, wie den Nutzenden dieses Wissen an die Hand gegeben wird: Mit einer Visualisierung des Roboters, oder einer vollständigen Simulation bzw. der Durchführung On-line, am realen System.

Die kinästhetische Führung benötigt zwar seitens des Systems vergleichsweise viele Ressourcen, erlaubt dafür aber von Haus aus weniger Fehler.

Vorkenntnisse voraussetzt, und auch hier sind die Resultate im realen Arbeitsraum direkt sichtbar. Die Eingabe erfolgt nicht gelenkweise, sondern durch die einwirkenden Kräfte am Roboter werden alle Gelenkstellungen gleichzeitig verändert. In Bezug auf Montageprobleme wurde z.B. in [103] in einer Benutzerstudie festgestellt, dass kinästhetische Führung und eine Variante davon kleinere Bewegungen erforderten als Eingaben über Tastatur und 6D-Maus, zeitlich schneller waren, und bessere Intuitivität und Zufriedenheit der Nutzenden erzielten.

Als Nachteil ist anzugeben, dass die Führung körperlich anstrengender sein kann als eine indirektere Eingabe. Die kinästhetische Eingabe sehr präziser Posen ist je nach Anwendung ebenfalls ein Problem. Auch ist diese Eingabemodalität per Definition an den realen Roboter gebunden, was wie oben erwähnt zu Produktionsstillstand über den Programmierzeitraum führt und nicht in allen Anwendungsfällen machbar ist. Trotzdem kommt dieses Verfahren aufgrund seiner Vorteile häufig zum Einsatz [2, 82, 100, 112, 143]. Auch für den gegebenen Anwendungsfall überwiegt der Vorteil der Verwendbarkeit durch Nichtexperten. Infolgedessen stützt sich die vorliegende Arbeit auf die kinästhetische Programmierung. Tabelle 2.1 gibt nochmals eine graphische Einordnung der genannten Punkte.

Als Variante existiert die Eingabe über eine zweite, leichtere oder kleinere Kinematik, deren Bewegung der eigentliche Roboter repliziert (wobei die beiden dann häufig als *Master-/Slave-Kinematik* bezeichnet werden).

Unter Nutzung der realen Umgebung können Endeffektorposen auch ohne Roboter direkt demonstriert werden. Dabei wird die Pose der Anwendenden oder die eines Eingabegeräts erfasst, beispielsweise über Kamerasysteme oder gezielte Tracking-Hardware mit Markern. Hier fällt der Aufwand weg, dass der Roboter selbst bewegt werden muss. Dafür öffnen sich neue Probleme wie die Genauigkeit der erfassten Posen.

Auch Off-line können Posen in bestimmten Fällen anderweitig spezifiziert werden. So lassen sich etwa Punkte auf der Oberfläche eines CAD-Objekts als Ziele per Mausklick markieren, wiederum noch über die inverse Kinematik umzuwandeln. Dabei ist jedoch eine detaillierte Modellierung unumgänglich.

Ein relativ neuer Trend ist die Verwendung von *Mixed Reality* (MR) in der Roboterprogrammierung, meist unter Verwendung von *Head-mounted Displays* (HMD). MR beschreibt allgemein die Vermischung realer und virtueller Szenenelemente. Die beiden typischen Teilfelder sind *Virtual Reality* (VR), wo eine komplett virtuelle Umgebung dargestellt wird, und *Augmented Reality* (AR), wo die reale Umgebung mit virtuellen Objekten oder Informationen angereichert wird. VR ermöglicht z.B. Methoden aus der Off-line-Programmierung, die sich zur Visualisierung auf eine Simulation stützen, diese den Nutzenden statt zweidimensional auf einem Monitor dreidimensional als virtuelle Szene anzuzeigen. Mittels AR lässt sich die Visualisierung auch direkt in die reale Umgebung einbetten. Eine andere Einsatzmöglichkeit ist es, die Erfassung von Nutzerposen in einer virtuellen Umgebung auszuführen, wodurch der reale Einsatzbereich des Roboters zur anderweitigen Verwendung frei bleibt. MR bietet somit viel Potenzial für zukünftige Anwendungen. Für die aktuelle Arbeit wurde gegen einen Einsatz entschieden: HMD stellen wieder eine zusätzliche Hardwarekomponente dar, was der Motivation entgegen-

läuft, auf solche zu verzichten. Aktuelle Geräte bringen außerdem noch verschiedene eigene Probleme mit sich, etwa ein eingegrenztes Sichtfeld oder eine Kabelverbindung zum Rechner, die die Bewegungsfreiheit einschränkt.

### 2.1.2. Eingabe des Programminhalts

Für die Eingabe des eigentlichen Programms und dessen Struktur gibt es eine ähnliche Abstufung. Anweisungen können textuell eingegeben werden, was allerdings nur für Experten sinnvoll machbar ist. Nicht nur die Kenntnis des Roboters selbst, sondern auch einer entsprechenden Programmiersprache ist dazu notwendig. Die verwendeten Sprachen können roboterspezifisch oder auch allgemeinere Hochsprachen sein. Dadurch bietet diese Methode als Vorteil die größte Ausdrucksstärke für Nutzende. Verschiedene Beispiele für proprietäre Sprachen wie auch für Hochsprachen sind in [111] gegeben.

Für Systeme, die auf Nichtexperten abzielen, werden häufig grafische oder visuelle Programmiermethoden verwendet. Solche Methoden existieren in der allgemeinen (nicht auf Roboter bezogenen) Programmierung schon seit geraumer Zeit und in einiger Fülle<sup>1</sup>, zu großem Teil auch im Bildungssektor. In [46] wird dazu ein ausführlicherer Überblick gegeben, sowie in [15] einige primär auf Kinder ausgerichtete kommerzielle Produkte aufgelistet.<sup>2</sup> Einige bekannte Beispiele aus der Forschung, die in ersterer Arbeit auch genannt werden, sind [33, 38, 53, 94].

Auch in der Roboterprogrammierung wird unter verschiedenen visuellen Ansätzen in der Regel das Programm (und gegebenenfalls auch die einzelnen Anweisungen) in einer einfacher verständlichen Darstellung angezeigt. Einen Überblick liefert hier [39]. Darstellungsvarianten sind etwa Baumstrukturen [161] oder Ketten aus einzelnen als Icons repräsentierten Operationen [52]. In [120] wird ein Überblick über häufig verwendete visuelle Metaphern gegeben, so z.B. Puzzlestücke für Operationen (was suggeriert, dass nur bestimmte Verbindungen möglich sind), oder Zeitleisten oder Pipelines für die Visualisierung eines sequenziellen Ablaufs. Eine visuelle Programmierschnittstelle in irgendeiner Form wird heutzutage von den meisten Roboterherstellern direkt angeboten, so etwa das Desk Interface von Franka Emika<sup>3</sup>, PolyScope von Universal Robots<sup>4</sup>, die Intera-Plattform für Roboter von Rethink Robotics<sup>5</sup>, Wizard Easy Programming von ABB<sup>6</sup> oder TMflow von Techman Robot<sup>7</sup>. Manche Lösungen sind schon auf spezielle Anwendungsfälle zugeschnitten, wie PickMaster und ähnliche Applikationssoftware von

---

<sup>1</sup>Für eindrucksvolle Übersichtssammlungen sh. <https://web.engr.oregonstate.edu/~burnett/vp1.html>, besucht 16.12.2022, oder <http://blog.interfacevision.com/design/design-visual-programming-languages-snapshots/>, besucht: 16.12.2022

<sup>2</sup>Neben der visuellen Programmierung bieten einige der in [15] genannten Produkte auch eine Programmierung über Interaktion mit realen Objekten an, teils auch als *Tangible User Interfaces* bezeichnet. Außerhalb des Bildungs- und Unterhaltungsbereichs finden solche Varianten allerdings nur selten Anwendung [144].

<sup>3</sup><https://www.franka.de/interaction>, besucht: 21.11.2022

<sup>4</sup><https://www.universal-robots.com/products/polyscope/>, besucht: 21.11.2022

<sup>5</sup><https://www.rethinkrobotics.com/de/intera>, besucht: 21.11.2022

<sup>6</sup><https://new.abb.com/products/robotics/de/applikationssoftware/wizard>, besucht: 21.11.2022

<sup>7</sup><https://www.tm-robot.com/de/tm-robot/>, besucht: 21.11.2022

ABB<sup>8</sup>. Alternativ existieren im Markt auch diverse Dritthersteller für entsprechende Software, wie etwa ArtiMinds<sup>9</sup> oder Wandelbots<sup>10</sup>. Auch viele Forschungsarbeiten stützen sich auf visuelle Darstellungen [27, 41, 72, 108, 121, 133, 143, 150, 156]. Im Gegenzug zur Verständlichkeit ist die Funktionalität und Ausdrucksstärke teilweise beschränkt, sodass unter anderem kommerzielle Systeme häufig trotzdem parallel eine (textuelle) Expertendarstellung anbieten.

Wie in Kapitel 1 motiviert, gibt es bei beiden bisher genannten Varianten die Voraussetzung zusätzlicher Eingabehardware. Im Gegensatz dazu stehen Methoden, die nur über den Roboter selbst stattfinden. Dies bringt allerdings verschiedene Einschränkungen mit sich.

Lange etabliert existiert die sogenannte *Playback-Programmierung* (auch: *Walk-Through-Programmierung*), bei der die Bewegungen des Roboters automatisch aufgezeichnet und in einen linearen Programmablauf umgewandelt werden, in welchem die einzelnen Posen sequenziell abgefahren werden [131]. In Spezialanwendungen wie dem Schweißen, Kleben oder Lackieren kommen solche Programmierverfahren oft zur Anwendung, da dort die aufgenommene Trajektorie genau den Aufgabeninhalt beschreibt. Für allgemeinere Verwendung fehlt dagegen Flexibilität in der Programmstruktur. Einige wünschenswerte Verallgemeinerungsmöglichkeiten von einer einzelnen, festen Trajektorie werden in [114] identifiziert: z.B., dass das System mit leicht unterschiedlich platzierten Werkstücken umgehen kann, mit einer anderen Anzahl oder Reihenfolge an Werkstücken, oder mit verschiedenen möglichen Produktvarianten. All dies erfordert Ausdrucksmöglichkeiten im Programm über die reine Trajektorie hinaus.

Ein verwandter Ansatz ist das *Teach-In* (auch: *Lead-Through-Programmierung*), wo nicht die gesamte Trajektorie, sondern nur einzelne Schlüsselposen aufgenommen werden, die zur Ausführung interpoliert werden [131]. Dies ist historisch eine häufige Einsatzart insbesondere bei der Automatisierung für große Losgrößen. Wiederum, aus denselben Gründen wie weiter oben, mangelt es für den Anwendungsfall dieser Arbeit an struktureller Flexibilität.

Ein anderer Ansatz ist es, die Aufgabenlösung durch Anwendende aufzunehmen und daraus die eigentliche Programmlogik zu lernen. Dies wird in Varianten als *Programming by Demonstration* (PbD), *Learning from Demonstration*, *Programmieren durch Vormachen* o.ä. bezeichnet. In diesem Bereich gibt es eine breite Fülle an genauen Modalitäten, z.B. mit unterschiedlichen Eingaben der Demonstration oder unterschiedlicher Sensorik zur Aufnahme. Eine Auswahl an Übersichtsarbeiten [13, 20, 30, 88, 128, 175] kann für weitere Details herangezogen werden. Beispielarbeiten sind etwa [32, 59, 119]. Es stellen sich hier aber klassische Probleme des maschinellen Lernens. Um robuste Ergebnisse (oder auch nur eine vollständige Abdeckung von variantenreicheren Aufgaben) zu erreichen, ist möglichst ein großer Umfang an Trainingsdaten (hier: Demonstrationen) wünschenswert [31]. Nutzende haben im Kontrast dazu wenig Interesse, die zu lösenden Aufgaben mehrmals vorzumachen, geschweige denn viele Male. Es gibt in diesem Forschungsfeld

<sup>8</sup><https://new.abb.com/products/robotics/de/applikationssoftware>, besucht: 21.11.2022

<sup>9</sup><https://www.artiminds.com/robot-applications/robot-programming/>, besucht: 21.11.2022

<sup>10</sup><https://wandelbots.com/de/>, besucht: 21.11.2022

Programm- eingabe	Mögliche Programm- struktur	Nötige Expertise	Eingabe- aufwand	Nötiges Umgebungs- modell
Playback	- sequenziell	+ keine	○ ausführen, einmal	+ keines
Textuell	+ frei	- textuelle Sprache	+ schreiben	+ keines
Visuell	+ frei	○ grafisches Interface	+ grafisch bauen	+ keines
Demonstration	○ nach Modell	+ keine	- ausführen, mehrmals	- Aufgaben- verständnis
Automatische Generierung	○ nach Modell	○ Syntax Planer	+ Problem angeben	- komplett
<b>Ziel dieser Arbeit</b>	+ frei	○ Aktions- befehle	○ ausführen, Struktur	+ keines

**Tabelle 2.2.:** Übersicht über Varianten der Programmeingabe. Die roten, negativ bewerteten Eigenschaften zeigen die jeweiligen Einschränkungen existierender Ansätze auf. Gesucht ist ein Verfahren, das in allen Aspekten zumindest akzeptabel ist. Die visuelle Programmierung, die für die Zielsetzung dieser Arbeit ansonsten am besten zu bewerten ist, hat den oben erwähnten Nachteil zusätzlicher Eingabegeräte und deren Wechsel, was die in dieser Arbeit vorgestellte Methode umgehen möchte. Es geht also um den Entwurf eines Systems, das ähnliche Eigenschaften ohne eine grafische Schnittstelle erzielt.

jedoch auch einzelne Arbeiten, die mit einzelnen (auch: *one-shot* oder *single-shot*) bis einigen wenigen Demonstrationen gute Resultate erzielen [61, 62, 115, 171, 178].

Unabhängig davon stellt sich bei Lernverfahren die Frage der Vorhersagbarkeit bzw. Nachvollziehbarkeit, sowie das Problem, bei strukturierten Aufgaben die Logik z.B. von Verzweigungen automatisch zu extrahieren. Für diese Anforderungen ist nach wie vor die explizite Programmierung vorteilhaft. In der vorliegenden Arbeit geht es genau um ein Verfahren zur expliziten Programmierung, an der also weiterhin Bedarf besteht.

Generell nicht betrachtet werden hier Verfahren, die das Programm rein auf Basis einer Aufgabenbeschreibung automatisiert generieren, etwa durch Planungssysteme. Einige Beispiele dafür werden etwa in [159] gegeben. Deren Einsatz setzt eine große Menge an im System modelliertem Vorwissen voraus, sei es in Bezug auf die Simulation des Roboters, sei es über domänenspezifische Objekte und Aufgaben. Dieses Vorwissen zu erlangen und explizit für das System zur Verfügung zu stellen, ist äußerst ressourcenintensiv. Daher ist ein flächendeckender Einsatz solcher Systeme, insbesondere in kleineren Unternehmen, aktuell nicht absehbar. Stattdessen besteht im Moment weiterhin Bedarf an Methoden, die ohne aufwändig zu modellierendes Vorwissen auskommen, etwa durch die direkte Programmierung der Roboteroperationen.

In Tabelle 2.2 sind die obigen Informationen nochmals visuell zusammengetragen. Es zeichnet sich aus diesem Überblick insgesamt eine Lücke ab: Verfahren, die Nichtexperten das explizite Erzeugen strukturierter und komplex aufgebauter Programme ermöglichen,

ohne zusätzliche Eingabegeräte und aufwändige Modelldaten. Genau in dieser Richtung wird mit der vorliegenden Arbeit erforscht, inwieweit und mit welchem Erfolg sich diese Lücke schließen lässt.

## 2.2. Automaten

Zustandsautomaten bieten eine Möglichkeit, Prozesse – oder das dynamische Verhalten von Systemen – zu formalisieren. Sie werden in diesem Zusammenhang unter anderem als „a priori recht natürliches Medium“ bezeichnet [65]. In der Robotik lassen sich damit insbesondere reaktive Systeme elegant ausdrücken, aber auch ansonsten kann eine übersichtliche Darstellung für die Nutzenden ihren Einsatz motivieren. Im Folgenden werden verschiedene Anwendungen von Automaten vorgestellt, um die Wahl des Automatenmodells in der vorliegenden Arbeit einzuordnen. Sie unterscheiden sich sowohl im konkreten Formalismus, der verwendet wird, als auch in der Abstraktionsebene, auf der etwa Zustände und Transitionen definiert sind.

### 2.2.1. Abstraktionsniveau

Als erstes werden verwandte Arbeiten im Bereich der automatenbasierten Roboterprogramme unter dem Gesichtspunkt eingeordnet, auf welcher Abstraktionsebene die Automaten agieren, bzw. welche Semantik die Zustände dabei tragen.

Die meisten Arbeiten bewegen sich hierbei auf einer Verhaltensebene: die einzelnen Zustände der Automaten stellen jeweils komplexere Verhaltensweisen des Roboters dar [23, 26, 59, 81, 96, 129]. Zustandsübergänge sind damit die Wechsel zwischen einzelnen Verhaltensweisen, und werden häufig nicht als Reaktion auf einen Stimulus, sondern über kontinuierlich zu prüfende Bedingungen herbeigeführt. Die konkreten Details variieren stark zwischen einzelnen Arbeiten.

Ein Beispiel ist etwa [26], wo die Zustände mit LISP-Programmen verknüpft sein können. Jeder der Zustände (dort auch als Module bezeichnet) hat eine klare Semantik, wobei das Gesamtsystem einen großen Automaten bildet, und Zustände auf unterschiedliche Kompetenzebenen abzielen können. Dieses als *Subsumption Architecture* betitelte Konzept wurde vielfach aufgegriffen [83]. Die Gesamtanwendung auf einer mobilen Plattform in der ursprünglichen Arbeit bewegt sich von reiner Kollisionsvermeidung hin zu explorativem Verhalten.

In [59] werden endliche Automaten gelernt, mit einem Ballspielszenario auf einem mobilen Roboter als Ziel. Auch hier entspricht jeder Zustand einer Teilaufgabe oder einem Teilverhalten und enthält entsprechend eine eigene Strategie. Wie weiter oben erwähnt für PbD-Ansätze häufig der Fall, ist auch hier eine einzelne Demonstration nicht ausreichend, sondern die Nutzer können anhand des aktuellen Lernstands des Roboters weitere Demonstrationen vorgeben.

Auch in [96] gibt es Zustandsautomaten über Verhalten, genau genommen sogar auf

mehreren Hierarchieebenen.<sup>11</sup> In der Arbeit geht es um eine Patrouillenaufgabe für mehrere Roboter, und Verhalten sind beispielsweise, weiter dem Grenzverlauf zu folgen, oder anderen Robotern auszuweichen. In den Worten der Autoren selbst, wurden endliche Automaten gewählt als der einfachste Weg, die Entscheidungsprozesse darzustellen.

In [81] stellen die einzelnen Zustände des Automaten ebenfalls vorgegebene Verhaltensweisen (von mobilen Schwarmrobotern) dar, in dem Fall jeweils die konkrete Handlung, die in dem Zustand erfolgt. Es wird dort eine eigene Variante endlicher Automaten vorgestellt, ebenfalls mit dem Argument, dass diese einfacher und verständlicher als andere Alternativen sind. Diese werden mittels genetischer Algorithmen für die Beispielaufgaben optimiert.

Ein ähnlicher Ansatz wird in [129] verfolgt: Auch dort werden die Automaten, deren Zustände verschiedene Operationen repräsentieren, mittels genetischer Algorithmen erzeugt, allerdings wiederum für einen Zweiarmtorso.

Auch in den hierarchischen Automaten in [23] definieren die einzelnen Basiszustände ein auszuführendes Verhalten eines Zweiarmtorsos, und die Übergänge richten sich nach den möglichen Resultaten dieser Ausführung.

Im Gegensatz dazu gibt es auch einzelne Arbeiten, in denen die Zustände der Automaten tatsächlich logische Aufgabenzustände repräsentieren, und in denen das Roboterverhalten entlang der Zustandsübergänge annotiert wird [114, 115]. Für die Transitionen werden dabei zu einzelnen Zeitpunkten gezielt Eingaben ausgewertet, beispielsweise in der Form des aktuellen Weltzustands. In den genannten Arbeiten ist die Aufgabendomäne auch ähnlich zum Ziel der vorliegenden Arbeit, nämlich Pick-and-Place-Szenarien. Aufgaben werden dort gelernt und sind zunächst rein sequenziell, erhalten später aber mit Schleifen und Verzweigungen auch Strukturelemente.

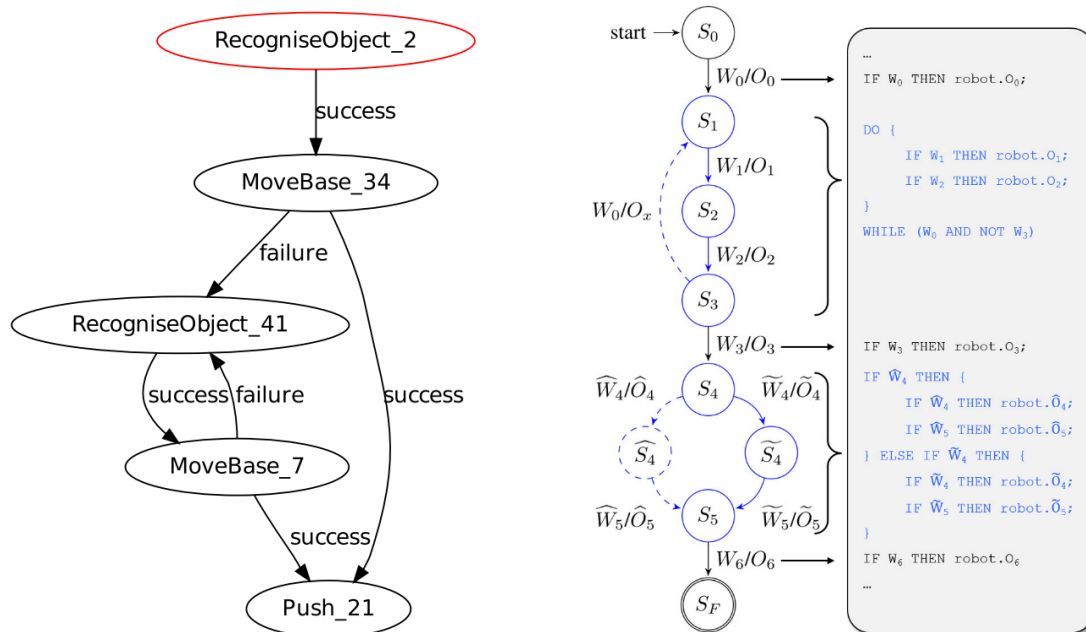
In Abbildung 2.2 sind einige Beispielautomaten aus verwandten Arbeiten dargestellt.

Zusammenfassend lässt sich sagen, dass Zustandsautomaten lange und erfolgreich in der Robotik eingesetzt werden. Sie bieten eine konzeptionell einfache und übersichtliche Modellierungsmethode für Prozesse, wenn diese beispielsweise reaktiv oder diskret ablaufen. In derselben Funktion werden sie auch in dieser Arbeit zum Einsatz kommen. Zum Abstraktionsniveau der Modellierung gibt es grundlegend zwei unterschiedliche Ansätze: In einem tragen die Zustände selbst komplexere Verhaltensweisen, und Zustandsübergänge stellen den Wechsel zwischen solchen Verhalten dar. Im anderen repräsentieren die Automatenzustände direkt den Aufgaben- oder Weltzustand, und das Verhalten des Roboters wird im Automaten selbst, entlang der Transitionen, definiert. Ersterer Ansatz erfordert darunterliegend die Definition der eigentlichen Verhalten in den Zuständen, was andererseits den Vorteil bringt, komplexere Umsetzungsdetails auf dieser Ebene kapseln zu können. Da im Automatenmodell in dieser Arbeit die einzelnen Arbeitsschritte des Roboters aber einerseits explizit programmiert werden, und andererseits einzeln konzeptionell nicht besonders komplex sind, verwendet diese Arbeit den zweiten Ansatz, die Modellierung des Verhaltens in den Transitionen im Automaten.

---

<sup>11</sup>Formal ist die Verwendung dennoch von den später vorgestellten hierarchischen Automaten abzugrenzen.





**Abbildung 2.2.:** Zwei Beispiele endlicher Automaten aus verwandten Arbeiten. Links: Ein in [129] per genetischen Algorithmen erzeugter Automat, um ein Objekt zu verschieben. Wie anhand der Benennung deutlich, repräsentieren die Zustände einzelne Arbeitsschritte des Roboters, und Zustandsübergänge finden jeweils nach Beendigung eines Schritts statt. Rechts: Beispiel aus [114], das die Umsetzung von Kontrollstrukturen illustriert. Automatenzustände stellen hier direkt den Aufgabenzustand dar, und Transitionen definieren einen Zustandsübergang unter in dem Moment wahrgenommenen Bedingungen. In dem Fall sind auch die Ausgaben, d.h. die Roboteraktionen, an den Transitionen annotiert.

### 2.2.2. Automatenmodelle

Ein zweiter Gesichtspunkt, unter dem verwandte Arbeiten eingeordnet werden können, ist das verwendete Automatenmodell selbst.

#### Einfache endliche Automaten

Eine Vielzahl an Arbeiten verwendet als formale Grundlage endliche Automaten etwa nach [70] mit verschiedenen kleineren Erweiterungen [26, 59, 81, 96, 114, 115, 129]. Insoweit die Erweiterungen nicht von der eigentlichen Automatenlogik unabhängig sind, belaufen sie sich meist auf die Verwendung zusätzlicher Variablen. Der Formalismus der endlichen Automaten selbst wird in Kapitel 3 im Detail vorgestellt.

In [26] werden beispielsweise zusätzlich zum eigentlichen Zustandsautomaten einzelne Variablen verwaltet und zwischen Zuständen weiterpropagiert. Diese Datenpfade können zudem unterschiedlichen Charakter haben, wie ein bedingtes Überlagern eines anderen Signals.

Die in [81] verwendeten *Moore Automata for Robotic Behavior* verwenden ebenfalls (Eingabe- und Ausgabe-) Variablen, die sich allerdings konzeptionell im Alphabet bzw. der Ausgabefunktion vollständig abkapseln lassen. Kommandos werden nicht entlang Transitionen generiert, sondern aus dem aktuellen Zustand<sup>12</sup>. Ähnlich verhält es sich mit den Formalismen aus [114, 115].

#### Timed Automata

Mit den sogenannten *Timed Automata* kommt in der Robotik gelegentlich eine Variante endlicher Automaten zum Einsatz, die sich hauptsächlich in der expliziten Modellierung von zeitlichen Variablen abgrenzt [10, 126]. Dabei können in Zuständen Flussbedingungen dieser Variablen angegeben werden. Dadurch, und durch die Zustandsübergänge nach Bedingungen über diesen Variablen, lassen sich komplexe Verhaltensweisen modellieren, die sich etwa durch Differenzialgleichungen beschreiben lassen. Für eine Anwendung durch Nichtexperten ist der Formalismus allerdings deutlich zu diffizil. In [10, 126] wird damit allgemein koordinierte Bewegungsplanung mit Hindernissen betrachtet.

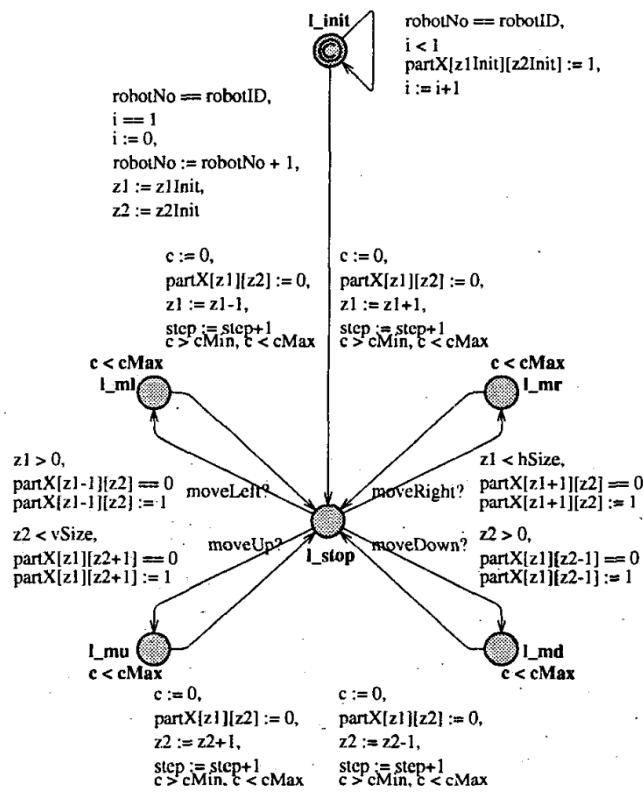
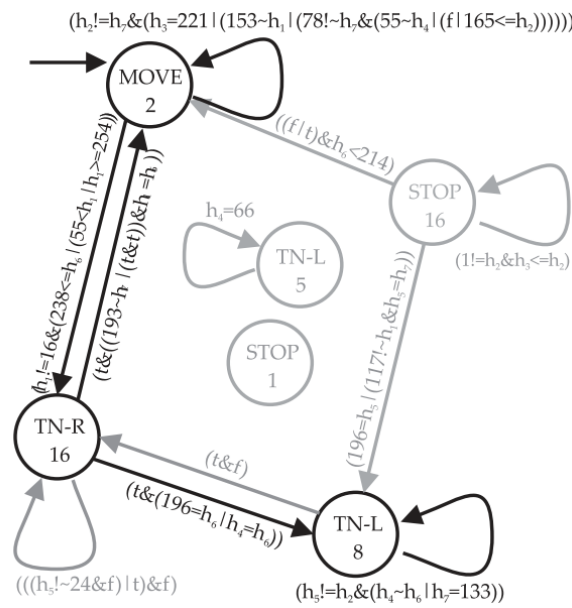
Abbildung 2.3 zeigt zwei Beispiele, eines für eine Erweiterung der endlichen Automaten sowie eines für die Verwendung von Timed Automata.

#### Hierarchische Automaten

Zu erwähnen ist auch die Fülle an Arbeiten, die mit *hierarchischen Automaten* arbeitet. Hier existiert eine Bandbreite an konkreten Formalismen, häufig unter dem Titel *Hierarchical State Machines* (HSM), oder unter dem Begriff *Statecharts* wie in einer der fundamentalen Publikationen vorgeschlagen [65]. Erweiterungen dieser Art finden in der Robotik regelmäßige Anwendung [12, 15, 23, 27, 28, 50, 58, 73, 78, 108, 136, 152, 156, 157, 172].

---

<sup>12</sup>analog zum Unterschied zwischen Moore- und Mealy-Automaten, sh. Kapitel 3



**Abbildung 2.3.:** Oben: Automatenerweiterung aus [81]. An den Transitionen sind komplexe Bedingungen über den zusätzlichen Variablen notiert, die kontinuierlich überwacht werden. Unten: Timed Automaton aus [126]. Die reellwertigen Zeitvariablen laufen während der Ausführung mit, und werden entlang der Transitionen geprüft bzw. zurückgesetzt.

Die genaue Semantik unterscheidet sich zwischen den einzelnen Arbeiten.<sup>13</sup> Grundsätzlich stellen hierarchische Automaten eine konzisere und übersichtlichere Darstellung für gegebene Automaten zur Verfügung, während die formale Ausdrucksstärke (ohne Verwendung zusätzlicher Erweiterungen) gleich bleibt wie bei den endlichen Automaten [174]. Dies wird dadurch ermöglicht, dass die Automaten in unterschiedliche Abstraktionsebenen verfeinert werden können (hierarchische Zustände), und unabhängige Teilprobleme auch in unabhängige Automatenteile zerlegt werden können (parallele Teilautomaten). Infolgedessen fällt allerdings auch die Syntax komplexer aus. Abbildung 2.4 zeigt zwei exemplarische Automaten aus verwandten Arbeiten.

In [156] werden Automaten verschiedener Hierarchieebenen verwendet, die sich stark an UML/P-Statecharts orientieren, um dafür existierende Werkzeuge zur Programmierung beziehungsweise Quellcodegenerierung zu verwenden. Explizit erwähnt wird hier neben der textuellen Programmierung eine grafische Schnittstelle im Eclipse-Framework.

Das grafische Tool RAFCON [27] verwendet ebenfalls eine eigene Variante von hierarchischen Automaten oder Statecharts. Hier liegt ein klarer Fokus auf der visuellen Programmierung und komplexen Szenarien. Obwohl nach eigenen Informationen die Ausdrucksstärke gegenüber allgemeinen Statecharts eingeschränkt wurde, ist das System in Funktionsumfang und zum Editieren nur durch elaborierte Navigationsmöglichkeiten wie Zoom und adaptive Größe von Elementen oder einen frei einstellbaren Anzeigebereich praktisch nutzbar. In einer späteren Arbeit wird das verwendete Modell dann in Richtung automatischer Parallelisierung erweitert [28].

Auch in [58, 108] handelt es sich um auf Experten ausgerichtete Systeme, die sich auf eine visuelle Darstellung stützen. Ebenso erfolgt in [73, 78] die Anwendung durch Experten im Rahmen autonomen Fahrens.

In [12, 152] werden UML-Statecharts auf Multiagentensysteme angewendet, was in Kapitel 6 nochmals genauer betrachtet wird. Dort werden explizite Synchronisationsmechanismen beschrieben, und eine der beschriebenen Anwendungen ist in einer Roboterfußballdomäne angesiedelt. Der Programmiervorgang wird hier nicht näher ausgeführt.

In [136] werden hierarchische Automaten als abstraktere von zwei Programmebenen im Kontext von komplexen Aufgreifstrategien eingesetzt.

Mit [50, 172] liegen weitere Anwendungen von HSM vor, die auf deren Verwendung aber nicht im Detail eingehen.

Der in [23] verwendete Formalismus orientiert sich stark an Statecharts, mit den Möglichkeiten paralleler Ausführung, und der zusätzlichen Nutzung von Datenvariablen auch über Zustandsgrenzen hinweg. In [157] können als Erweiterung Verhalten (d.h. Zustände) selbst als Objekte zwischen anderen Verhalten weitergegeben werden.

Auch verschiedene existierende Softwarelösungen unterstützen die Verwendung von hierarchischen Automaten, wie SMACH unter ROS<sup>14</sup> oder rFSM unter Orocos<sup>15</sup>.

Trotz der ansonsten weiten Verbreitung hierarchischer Automaten muss im Kontext dieser Arbeit abgewägt werden zwischen den potenziell kompakteren Automaten, die durch

---

<sup>13</sup>Ein Überblick über Varianten des Formalismus an sich wird in [17] gegeben.

<sup>14</sup><http://wiki.ros.org/smach>, Zugriff: 16.12.2022

<sup>15</sup><https://orocos.org/stable/documentation/rFSM/index.html>, Zugriff: 16.12.2022

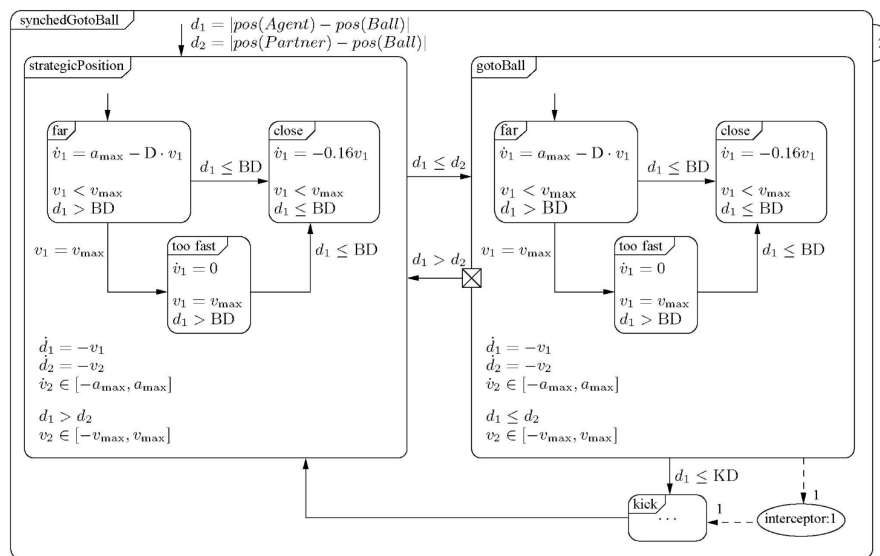
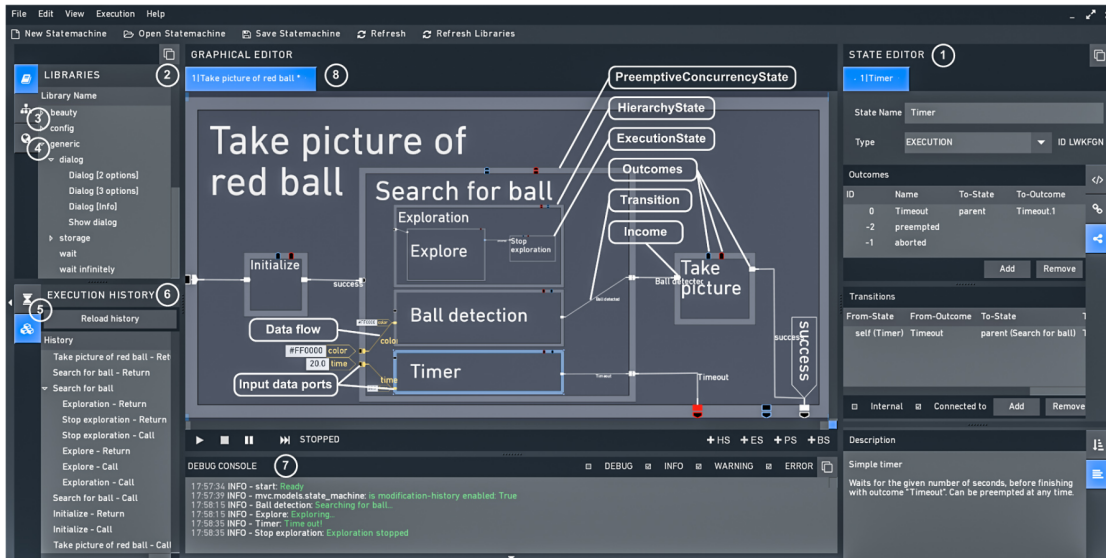


Abbildung 2.4.: Zwei Beispiele für komplexere Programme unter Verwendung von Statecharts (bzw. Varianten davon), beide aus dem Kontext des Roboterfußballs. Oben: Automat im grafischen Editor des RAFCON-Systems aus [27]. Unten: Darstellung aus [102].

die zusätzliche Ausdrucksstärke ermöglicht werden, und der erhöhten Komplexität auf der anderen Seite, insbesondere was Nutzbarkeit durch Nichtexperten ohne eine grafische Schnittstelle anbelangt. Hierbei stellt sich zentral auch die Frage, inwieweit die Ausdrucksstärke für die angepeilten Anwendungsbereiche nützlich oder erforderlich ist. Die Möglichkeit paralleler Teilautomaten findet üblicherweise entweder zur Modellierung mehrerer Agenten (beispielsweise mobiler Roboter) Anwendung, oder es werden darüber parallel laufende Verhaltensweisen eines einzelnen Roboters modelliert. Ersterer Fall ist für diese Arbeit zunächst nicht relevant: Es geht grundlegend um die Programmierung eines einzelnen Leichtbauarms. (Die Erweiterung des Ansatzes auf mehrere Arme wird in Kapitel 6 behandelt.) Zweiterer Fall ist in der Regel motiviert dadurch, in reaktiven Systemen emergentes Verhalten aus dem Zusammenspiel einzelner (einfacherer) Komponenten zu gewinnen. Dieser Ansatz birgt aber die Schwierigkeit, schwer nachvollziehbar zu sein, und unterscheidet sich fundamental von der hier umgesetzten expliziten Eingabe eines einzelnen Programms. Parallelität kommt also als Vorteil in dieser Anwendung nicht zum Tragen.

Hierarchie innerhalb der Automaten hat verschiedene Vorteile. Einer davon liegt in einer übersichtlicheren Darstellung (wo etwa Unterzustände ausgeblendet und erst bei Bedarf angezeigt werden können), was aber beim Verzicht auf eine grafische oder textuelle Darstellung nicht relevant ist. Ein anderer stellt sich konzeptionell in der Modellierung: Teilautomaten können einzelne Aspekte des Programms abdecken, deren konkrete Umsetzung durch Austausch des Innenlebens transparent angepasst werden kann. Insofern Teilautomaten an mehreren Stellen der Hierarchie verwendet werden können, können hier auch tatsächlich konzisere Automaten entstehen. Eine Verwendung dieser Möglichkeiten erfordert seitens des Systems das Editieren und die Navigation über mehrere Hierarchieebenen. Nicht nur steigt dadurch die Gesamtkomplexität; die Information, auf welcher Hierarchieebene man sich gerade befindet, ist auch a priori nicht in der realen Umwelt repräsentiert. Zuletzt bieten die Aufgaben, auf die das System in dieser Arbeit abzielt (relativ kleine Pick-and-Place-Szenarien) nur eingeschränkt Raum für die Art von Komplexität, die von Hierarchie profitieren würde, oder für ausreichend große wiederholt auftretende Programmteile, die eine explizite Kapselung lohnen würden.

Als exotischere (im Sinne von: weniger verbreitete) Variante seien hier noch SyncCharts angeführt [11]. Außerdem macht auch [92] für den dort definierten XABSL-Formalismus Gebrauch von einer Art hierarchischer Automaten, die nicht direkt mit Statecharts oder hybriden Automaten zusammenhängen. Die Anwendung kommt erneut aus dem Roboterfußball.

### **Hybride Automaten**

In Hinblick auf weitere Varianten der Zustandsautomaten stellen *hybride Automaten* konzeptionell eine Annäherung an Prozesse der realen Welt dar. So werden darin innerhalb von Zuständen kontinuierliche Veränderungen der Welt oder Variablen modelliert, während die Transitionen zwischen Zuständen im Gegensatz dazu diskrete Verhaltenswechsel darstellen [9]. Dieses Modell kommt entsprechend ebenfalls in der Robotik zur Anwendung, häufig in Kombination mit hierarchischen Elementen. So beschäftigt sich [102] mit

einer Anwendung im Roboterfußball, wobei hybride Automaten für die Synchronisation mehrerer Agenten benutzt werden. Es wird nicht näher auf die eigentliche Programmierung eingegangen. In [8] werden sie zudem für Gruppen von Robotern benutzt, die gemeinsam ein Zielobjekt aus einem abgeschlossenen Raum bergen sollen. In [24] werden ebenfalls hybride Automaten verwendet, wiederum mithilfe eines visuellen Editors, sowie in [109] in einem auf Experten ausgerichteten, primär textuellen System.

### Petrinetze

Zuletzt gibt es auch Arbeiten, die formal auf *Petrinetzen* basieren [177, 178]. Petrinetze werden auch in anderen Kontexten zur Prozessmodellierung verwendet. Dabei besteht der Grundmechanismus aus Markern, die sich über sogenannte Transitionen<sup>16</sup> zwischen Stellen genannten Knoten bewegen. Hinzu kommen auch hier häufig Bedingungen parallel zu Teilen der Netze. Petrinetze bieten sich insbesondere an, wenn es nicht nur einen klaren Kontrollfluss gibt. Marker können nach einer Transition in mehreren Zielknoten entstehen, sodass parallele Abläufe angestoßen werden können. In den hier motivierten Roboterprogrammen soll für einzelne Roboter dagegen ein nachvollziehbarer Kontrollfluss gegeben sein. Auf die Verwendung von Petrinetzen für Mehrarmsysteme wird in Kapitel 6 genauer eingegangen.

Die Anwendungen aus [177] beschäftigen sich mit kooperativer Erkundung und Suche. In [178] kommt ein Petrinetz-basiertes System auf einem Zweiarmtorso zum Einsatz, um z.B. mit beiden Händen koordiniert Gefäße zu öffnen.

Die vorgestellten Arbeiten mit ihrer Kategorisierung sind in Tabelle 2.3 nochmals zusammengetragen. Als Fazit lässt sich hier ziehen, dass neben den einfachen endlichen Automaten auch in großem Umfang komplexere Automatenmodelle in der Robotik zum Einsatz kommen, meist mit klarem Bezug auf einen entsprechenden Erweiterungsbedarf wie Parallelisierung, hierarchische Zustände oder Synchronisationsmechanismen. Teilweise werden die Automaten dabei automatisch erzeugt. Ansonsten sind die Arbeiten, die auf die Programmierung eingehen, meist der visuellen Programmierung zuzuordnen. Daraus resultiert die Einschätzung, dass eine solche Schnittstelle bei Erweiterungen wie den hybriden oder hierarchischen Automaten für eine sinnvolle Nutzung als erforderlich erachtet werden kann. Da die Zielsetzung der vorliegenden Arbeit eine Anwendbarkeit ohne grafische Darstellung ist, wird hier auf den Einsatz eines der komplexeren Automatenmodelle verzichtet. Weiterhin soll auf eine zusätzliche, darunter gelagerte Abstraktionsebene verzichtet werden, sodass in den Automaten direkt die programmierten Aktionen des Roboters ausgedrückt werden sollen (und nicht etwa in einzelnen Zuständen komplexere Verhaltensweisen versteckt werden). Entsprechend wird für die vorliegende Arbeit weiterhin vom ursprünglichen Modell endlicher Automaten ausgegangen, und dieses in Kapitel 3 mit der Ansteuerung des Roboters verbunden.

<sup>16</sup>Eine der zwei Knotenarten in Petrinetzen; Nicht zu verwechseln mit den Zustandsübergängen in endlichen Automaten, die hier ansonsten mit Transitionen gemeint sind.

**Tabelle 2.3.:** Übersicht über Automatenmodelle in verwandten Arbeiten in der Roboterprogrammierung. Die Spalte „Kategorie“ gibt die Einordnung gemäß dem Fließtext an. In der Spalte „Bezeichnung“ ist der Titel des verwendeten Modells oder des Frameworks angegeben (falls von der Kategoriebezeichnung verschieden). Die Spalte „Programmierung“ gibt ggfs. wieder, auf welche Anwendenden die Arbeiten abzielen, oder andere Details zum Programmiervorgang – Falls nichts genaueres angegeben ist, wird von einer Verwendung durch Robotikexperten ausgegangen. Die Spalte „Ref.“ gibt die Referenzen zu einem Tabelleneintrag an.

Kategorie	Bezeichnung	Programmierung	Ref.	Anmerkungen
Endliche Automaten	MARB	Experten	[26]	Module sind Automaten, kommunizieren über Signale. Anwendung: Autonomer mobiler Roboter.
		PbD	[59]	Anwendung: Roboterfußball.
		–	[96]	Anwendung: Patrouille durch Roboterschwarm.
		Evolutionäre Algorithmen	[81]	Anwendung: Schwarm mobiler Roboter.
		Evolutionäre Algorithmen	[129]	Zustände haben Parameter, gemeinsamen Speicher. Anwendung: Stapeln, Eingießen mit Zweiarmtorso.
		PbD	[114, 115]	Anwendung: Pick-and-Place, Leichtbauroboter.
Timed Automata		–	[10, 126]	Anwendung: Bahnplanung für mehrere mobile Roboter.
Hierarchische Automaten	UML/P Statecharts	Nichtexperten, grafisch	[156]	Experten schreiben Skills. Anwendung: Montageaufgaben.
			[28]	Anwendung: Haushaltsdomäne mit mobilem Manipulator.
	UML Statecharts	–	[12, 152]	Synchronisationszustände. Anwendung: Roboterfußball.
	XRobots	–	[157]	Lokale Variablen. Zustände können selbst als Parameter übergeben werden. Anwendung: Mobile Roboter.
	SMACH	–	[23]	Grafische Schnittstelle zur Laufzeit. Anwendung: Getränke servieren mit mobilem Zweiarmtorso.
		Experten	[15]	Anwendung: Mobile Roboter zur Interaktion mit Kindern.



Tabelle 2.3.: Fortsetzung.

Kategorie	Bezeichnung	Programmierung	Ref.	Anmerkungen
Hierarchische Automaten (Forts.)		Experten, grafisch	[58]	Anwendung: Proben aufnehmen in der Raumfahrt.
		Experten	[73, 78]	Anwendung: Autonomes Fahren.
		Experten	[136]	Anwendung: Greifplanung.
		Experten	[50]	Anwendung: Greifplanung und Falten von Papier.
		Experten	[172]	Inspiziert durch XABSL. Anwendung: Roboterfußball.
		Experten, grafisch	[108]	Anwendung: Haushaltsaufgaben mit mobilem Zweiarmtorso.
Hybride Automaten		–	[8]	Anwendung: Suchen von Objekten mit Schwarmrobotern.
	CHARON	grafisch	[9]	Anwendung: Bahnplanungsprobleme.
		–	[102]	Kombination aus hybriden Automaten und Statecharts. Anwendung: Roboterfußball.
		grafisch	[24]	Anwendung: Peg-in-Maze-Montageoperationen.
		Experten	[109]	Anwendung: Roboterfußball.
Petri netze	Petri Net Plans	–	[177]	Anwendung: Suche/Übergabe von Objekten mit mehreren Robotern.
		PbD	[178]	Anwendung: Behälter öffnen mit Zweiarmtorso.
Andere	SyncCharts	–	[11]	An Statecharts angelehnt. Anwendung: Greifplanung.
	XABSL	–	[92]	Hierarchien endlicher Automaten. Anwendung: Roboterfußball.

## 2.3. Fazit

In Kombination ergibt sich aus den obigen Überlegungen folgende Erkenntnis: Es besteht ein Bedarf an Verfahren zur Roboterprogrammierung durch Nichtexperten. Dabei besteht aktuell eine wissenschaftliche Lücke in Form von führungsbasierten Methoden zur expliziten Programmierung, die ohne eine textuelle oder grafische Eingabe oder Darstellung des Programms auskommen, die aber gleichzeitig das Erstellen strukturierter und komplexer

Programme ermöglichen.

Für diese Zwecke zeichnen sich einige weitere Randbedingungen als vorteilhaft heraus. Die Verwendung endlicher Automaten als zugrunde liegendes Modell bietet sich als einfacher und verbreiteter Formalismus an, der auch ohne genaue Kenntnis verwendet werden kann. Um zusätzliche Abstraktionsebenen unterhalb der Zustände zu vermeiden, modellieren diese den Aufgabenzustand, und die tatsächlichen Aktionen des Roboters werden entlang der Zustandsübergänge definiert.

3.1. Einfache Roboterzustandsautomaten . . . . .	31
3.1.1. Grundlagen endlicher Automaten . . . . .	31
3.1.2. Formalismus RSA . . . . .	33
3.1.3. Anwendungsbeispiel . . . . .	34
3.2. Erweiterte Roboterzustandsautomaten . . . . .	36
3.2.1. Formalismus ERSA . . . . .	37
3.2.2. Anwendungsbeispiel . . . . .	39
3.3. Programmierung . . . . .	42

---

Bei *Roboterzustandsautomaten* handelt es sich um Erweiterungen des Modells (*deterministischer*) *endlicher Automaten*. Im Folgenden wird zunächst dessen zugrunde liegende Formalisierung vorgestellt und, darauf aufbauend, das Modell der *einfachen Roboterzustandsautomaten*. Anschließend wird das für den Rest dieser Arbeit essenzielle Modell der *erweiterten Roboterzustandsautomaten* definiert.

Einfache Roboterzustandsautomaten wurden ursprünglich in [183] (bzw. publiziert in [182]) eingeführt, allerdings noch mit deutlich abweichender Definition. Erweiterte Roboterzustandsautomaten wurden erstmals in [179] vorgestellt, ebenso noch in anderer Formulierung.

## 3.1. Einfache Roboterzustandsautomaten

Zunächst wird das einfache Automatenmodell aus herkömmlichen endlichen Automaten hergeleitet und vorgestellt, in seiner Definition und einem illustrativen Beispiel.

### 3.1.1. Grundlagen endlicher Automaten

Eine übliche Formalisierung für endliche Automaten z.B. nach [70, 142] lautet wie folgt:

**Definition 3.1.1** (Deterministischer endlicher Automat). Ein deterministischer endlicher Automat, kurz DEA, ist definiert als Quintupel  $M = (Q, \Sigma, \delta, q_s, F)$ . Dabei ist

- $Q$  die endliche Zustandsmenge
- $\Sigma$  das endliche Eingabealphabet
- $\delta : Q \times \Sigma \rightarrow Q$  die Transitionsfunktion
- $q_s \in Q$  der Startzustand
- $F \subseteq Q$  die Menge an akzeptierenden Zuständen

DEA verarbeiten Eingabeworte, indem jeweils ein Eingabesymbol  $\sigma \in \Sigma$  gelesen wird, und der Automat aus dem aktuellen Zustand  $q \in Q$  (beginnend in  $q_s$ ) entsprechend der Transitionsfunktion in  $q' = \delta(q, \sigma)$  übergeht. Die Funktion  $\delta$  kann partiell definiert sein. In dem Fall bricht der Automat bei nicht vorgesehenen Transitionen (also bei Eingaben, für die im aktuellen Zustand kein Wert der Transitionsfunktion definiert ist) die Bearbeitung ab, ohne zu akzeptieren. Alternativ würden diese Transitionen bei einer total definierten Transitionsfunktion in einen expliziten Fang- oder Fehlerzustand führen.

Neben der Entscheidung, ob ein Eingabewort in einer Sprache enthalten ist, werden endliche Automaten auch zur Modellierung von Prozessen mit Ein- und Ausgaben verwendet.

**Definition 3.1.2** (Mealy-Automat [70]). Ein Mealy-Automat ist definiert als Sextupel  $M = (Q, \Sigma, \Delta, \delta, \lambda, q_s)$ . Dabei ist

- $Q$  die endliche Zustandsmenge
- $\Sigma$  das endliche Eingabealphabet
- $\Delta$  das endliche Ausgabealphabet
- $\delta : Q \times \Sigma \rightarrow Q$  die Transitionsfunktion
- $\lambda : Q \times \Sigma \rightarrow \Delta$  die Ausgabefunktion
- $q_s \in Q$  der Startzustand

Ein Mealy-Automat beginnt im Startzustand  $q_s \in Q$  und folgt genau wie ein DEA den der Eingabe entsprechenden Transitionen. Zu jedem Zustandsübergang wird dabei jedoch auch ein Zeichen aus  $\Delta$  ausgegeben.<sup>1</sup> Es kann davon ausgegangen werden, dass  $\forall q \in Q, \sigma \in \Sigma : [\lambda(q, \sigma) \text{ definiert} \iff \delta(q, \sigma) \text{ definiert}]$ <sup>2</sup>. Wie in der Definition 3.1.2 ersichtlich, ist für Mealy-Automaten (oder generell Automaten zur Prozessmodellierung) meist keine akzeptierende Zustandsmenge definiert. Stattdessen reagieren diese Automaten nur auf die (beliebig langen) Eingabefolgen.

---

<sup>1</sup>Parallel zu Mealy-Automaten existiert das Modell der Moore-Automaten, worin die Ausgabefunktion nur über den Zuständen definiert ist, und die Ausgabe daher mit dem Betreten eines Zustands assoziiert ist.

<sup>2</sup>„definiert“ wird hier aus Gründen der Übersichtlichkeit verwendet. Formal ließe sich etwa die Aussage „ $\delta(q, \sigma)$  (ist) definiert“ auch via Quantor darstellen als „ $\exists q' \in Q : \delta(q, \sigma) = q'$ “.

### 3.1.2. Formalismus RSA

Ein einfacher Roboterzustandsautomat stellt sich nun ähnlich zum Mealy-Automaten dar.

**Definition 3.1.3** (Einfacher Roboterzustandsautomat). Ein einfacher Roboterzustandsautomat, kurz RSA (für engl. *Robot State Automaton*), ist definiert als Tupel  $M = (Q, \Sigma, P, \delta, u, q_s)$ . Dabei ist

- $Q$  die endliche Zustandsmenge
- $\Sigma$  das endliche Eingabealphabet
- $P$  der Raum der Ausgabefunktion an Posen
- $\delta : Q \times \Sigma_\varepsilon \rightarrow Q$  die Transitionsfunktion
- $u : Q \times \Sigma_\varepsilon \rightarrow P$  die Updatefunktion
- $q_s \in Q$  der Startzustand

Die Benennung der Updatefunktion als  $u$  statt  $\lambda$  ist dabei nur eine Konventionsfrage. Es gilt analog zu oben  $\forall q \in Q, \sigma \in \Sigma_\varepsilon : [u(q, \sigma) \text{ definiert} \iff \delta(q, \sigma) \text{ definiert}]$ . Bei  $P$  handelt es sich um Endeffektorposen (d.h., Position und Orientierung) im Arbeitsraum des Roboters, dargestellt als Transformationen aus der Gruppe der eigentlichen Bewegungen  $E^+(3)$  (auch: *spezielle euklidische Gruppe*  $SE(3)$ ). In der Implementierung eignen sich dazu z.B. affine Transformationsmatrizen. Zum (räumlich begrenzten) Raum  $P$  der möglichen Ausgabeposen stellt sich die Frage, ob dieser als diskret oder kontinuierlich betrachtet wird. In ersterem Fall ist das Modell formal betrachtet komplett analog zum Mealy-Automaten. Tatsächlich existieren innerhalb der ansonsten endlichen Automaten-Definition nur endlich viele Transitionen, an denen ein Update definiert sein kann, und damit wird praktisch stets nur eine endliche Teilmenge aus  $P$  verwendet.<sup>3</sup>

Das Eingabealphabet  $\Sigma$  kann bei RSA allgemein verschiedene, über aus der Umwelt wahrnehmbare Informationen unterscheidbare Fälle umfassen. Praktisch handelt es sich etwa um die Bezeichner verschiedener Objekte, die über eine am Roboter montierte Kamera erkannt werden können, auch z.B. in Verbindung mit einer Teilmenge an Posen, die das Objekt einnehmen darf, um dem speziellen Fall zugeordnet zu werden. Zu  $\Sigma$  wird die Ergänzung  $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$  gebildet. In der Definition einer Transition bedeutet  $\varepsilon$  im Gegensatz zu den  $\sigma \in \Sigma$ , dass diese Transition ausgeführt wird, ohne eine Eingabe zu erfassen. Die Transition wird auch als *spontan* bezeichnet. Da die Automaten deterministisch sein sollen, folgt direkt die Aussage:  $\forall q \in Q : [\delta(q, \varepsilon) \text{ definiert} \implies \nexists \sigma \in \Sigma : \delta(q, \sigma) \text{ definiert}]$ . Zustände mit einer spontanen Transition erfassen nie ein explizites Eingabesymbol (sonst wäre bei Auftreten der entsprechenden Eingabe der zu verfolgende Übergang nicht eindeutig). Auch die Formulierung der Kontraposition ist einleuchtend: Wenn in einem Zustand bereits mindestens eine Transition unter einer bestimmten expliziten Eingabe

<sup>3</sup>Eine ähnliche Überlegung bezogen auf den dort verwendeten Formalismus wurde auch in [183] getroffen.

$\sigma \in \Sigma$  definiert ist, darf von dem Zustand keine spontane Transition mehr ausgehen. Entsprechend lässt sich die Zustandsmenge partitionieren als  $Q = Q_b \uplus Q_v \uplus Q_t$ , wobei:

- $Q_b = \{q \in Q \mid \exists \sigma \in \Sigma : \delta(q, \sigma) \text{ definiert}\}$  sind *Verzweigungszustände* (engl. *branching states*), Zustände mit mindestens einer ausgehenden Transition, die eine explizite Eingabe erfordert. Das System muss entsprechend auch vor dem Übergang eine Eingabe erfassen. Diese Zustände sind die einzigen, die mehr als eine ausgehende Transition haben können, daher der Name.
- $Q_v = \{q \in Q \mid \delta(q, \varepsilon) \text{ definiert}\}$  sind *Transitzustände* (engl. *via states*), solche mit einer ausgehenden  $\varepsilon$ -Transition. In diesen wird der Übergang direkt verfolgt, ohne eine Eingabe zu erfassen.
- $Q_t = Q \setminus \{Q_b \cup Q_v\}$  sind Zustände, für die aktuell keine Transition definiert ist, auch als *Terminalzustände* bezeichnet (nicht zu verwechseln mit den akzeptierenden Zuständen in DEA, die ebenfalls teilweise so bezeichnet werden).

Ein RSA kann in Verbindung mit einem Roboter wie folgt verwendet werden: Der RSA beginnt im Startzustand  $q_s \in Q$ , und folgt wie ein Mealy-Automat auf jede Eingabe mit einem entsprechenden Zustandsübergang, bzw. führt spontane Transitionen automatisch aus. Bei einem Zustandsübergang wird jeweils das entsprechende Update angewendet, d.h. der Roboter wird zur Pose  $u(q, \sigma)$  verfahren.<sup>4</sup> Wenn diese Pose erreicht ist, gilt auch der Folgezustand als erreicht, und die nächste Transition (und das zugehörige Update) kann ausgeführt werden.

### 3.1.3. Anwendungsbeispiel

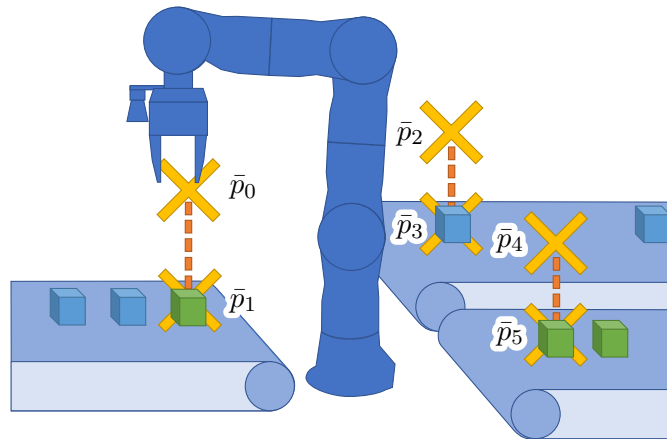
Um die Anwendung des Formalismus zu illustrieren, zeigt Abbildung 3.1 eine kleine Beispielaufgabe. Dabei werden verschiedene Objekttypen unterschieden und in Pick-and-Place-Manier zwischen Förderbändern bewegt. Die Aufgabe enthält nur absolute Bewegungen (sowohl die Ankunftsposition als auch die Zielpositionen sind fest), und abgesehen von der Wiederholung des gesamten Ablaufs gibt es keinen Bedarf an weiteren Schleifen.

Ein zugehöriger RSA ist in Abbildung 3.2 angegeben. Dieser fällt entsprechend simpel aus, verdeutlicht aber das Prinzip des Formalismus.

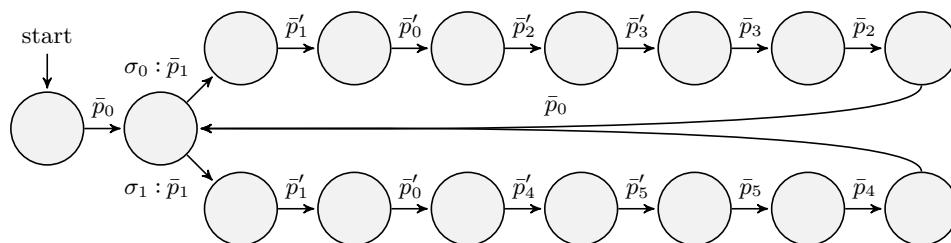
Alle Transitionen mit Ausnahme der beiden Fälle der Verzweigung sind spontan, d.h. definiert unter einer Eingabe von  $\varepsilon$ . Zur besseren Übersicht wird auf die Beschriftung solcher Eingaben in den abgebildeten Automaten in dieser Arbeit i.A. verzichtet. Da die Updatefunktion  $u$  direkt auf die anzufahrenden Posen abbildet, ist das auch genau die

---

<sup>4</sup>In der Implementierung in dieser Arbeit sind alle Bewegungen des Roboters, sofern nicht anders kenntlich gemacht, Point-to-Point-Bewegungen, unter minimalen Gelenkwinkeländerungen. Dies ist eine praktische Entscheidung auf Basis einer einfachen Ansteuerung. Prinzipiell lassen sich ebenso Linearbewegungen verwenden, oder es kann eine komplexere Bewegungsplanung erfolgen (z.B. in Kombination mit zusätzlicher Sensorik oder einem Umweltmodell). Einige Vorteile des Anwendungsfalls in dieser Arbeit sind aber, auf eine Umweltmodellierung zu verzichten und wenig Sensorhardware zu benötigen, was bei aufwändigerer Bahnplanung ggfs. wegfallen würde.

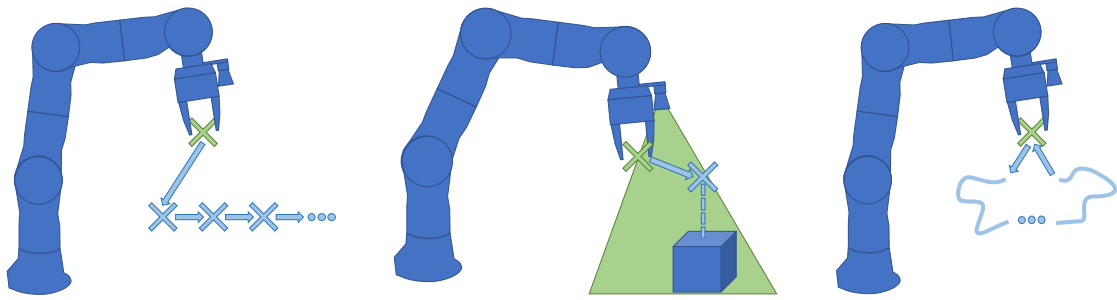


**Abbildung 3.1.:** Eine einfache Beispielaufgabe für die Verwendung von RSA. Ein Roboter verteilt Objekte zwischen Förderbändern um. Links ankommende Objekte werden nach ihrer Farbe auf eines der beiden ausgehenden Bänder rechts gelegt. Es sind sechs Posen  $\bar{p}_0$  bis  $\bar{p}_5$  markiert, die in einer einfachen Umsetzung relevant wären: die Aufgreif- und Ablageorte auf den drei Förderbändern, und jeweils eine Pose darüber, die eine kollisionsfreie Annäherung ermöglicht.



**Abbildung 3.2.:** RSA zur Beispielaufgabe aus Abbildung 3.1. Der Automat teilt sich zu Beginn in zwei Zweige, und am Ende beider Zweige wird durch eine Kante zurück zum Verzweigungszustand die Schleife geschlossen.

An jeder Kante von  $q$  zu  $q'$  (d.h. zu jeder Transition  $\delta(q, \sigma) = q'$ ) ist eine Zielpose gemäß  $u(q, \sigma) = \bar{p}$  notiert (wobei Posen  $\bar{p}'$  einen geschlossenen Greifer bei ansonsten gleicher Pose wie  $\bar{p}$  symbolisieren). Insofern dabei  $\sigma \neq \varepsilon$  ist, ist dieses ebenfalls an der Kante notiert, in der Form „ $\sigma$  :“ vor der entsprechenden Zielpose. Die Benennung der einzelnen Posen ist dabei identisch wie in Abbildung 3.1.



**Abbildung 3.3.:** Defizite des (einfachen) RSA-Modells, von links nach rechts: Ausdruck relativer Bewegungen z.B. für Schleifeninkremente, Ausdruck von relativ zu Objekten definierten Zielposen, z.B. um mit ungenauer Platzierung umzugehen, Speichern und anschließende Verwendung von zur Programmierzeit unbekanntem Posen (zur Laufzeit).

Information, die an den Kanten angegeben wird.

Zu den beiden Verzweigungsfällen:  $\sigma_0$  und  $\sigma_1$  beschreiben genau, dass ein Objekt vom entsprechenden Typ (blauer Würfel bzw. grüner Würfel) erkannt wurde.

### 3.2. Erweiterte Roboterzustandsautomaten

Einfache Roboterzustandsautomaten haben im Gegenzug zu ihrer formalen und konzeptionellen Simplizität verschiedene Defizite. Drei Punkte, die sich im Formalismus beispielsweise nicht ausdrücken lassen, sind etwa wie in Abbildung 3.3 visualisiert:

- Relative Bewegungen. Alle Updates stellen in RSA eine Bewegung zu einer neuen, absoluten Zielpose dar. Es wäre jedoch wünschenswert, an manchen Transitionen eine Bewegung relativ zur aktuellen Pose des Roboters auszuführen. Als Beispiel: Wenn Objekte von einem Tablett unbekannter Breite abgeräumt werden sollen, wäre eine Möglichkeit, von einer Basispose am Rand des Tablett aus immer die jeweils nächste Stelle bzw. immer das jeweils nächste Objekt anzufahren. Für eine unbekannte Tablettbreite und damit Objektanzahl ist das mit absoluten Bewegungen nicht möglich.
- Relativ zu Objekten definierte Zielposen. Für reale Anwendungen ist es naheliegend, von Objekten aus der Kamerawahrnehmung auch die jeweilige Pose zu berechnen. Damit wäre es z.B. möglich, auf eine variierende Platzierung eines zu greifenden Objekts innerhalb des Sichtbereichs zu reagieren (ohne die explizite Unterscheidung verschiedener Orte).
- Speichern und Anfahren zur Laufzeit von zur Programmierzeit unbekanntem Posen. Eine Motivation dafür folgt aus dem obersten Punkt der Liste: Wenn über relative Bewegungen eine Pose erreicht wurde, aus der zwar eine ganz andere Bewegung herausführt, die aber anschließend wieder angefahren werden soll, würde eine Speicherung der Pose in einer Posenvariablen das ermöglichen. In Erweiterung des Tablettbeispiels könnten die Objekte vom Tablett jeweils auf dieselbe Position auf



einem Förderband gestellt werden, aber beim Abräumen soll direkt von der Pose des letzten Objekts fortgefahren werden, ohne unnötige Suchschritte zu machen.

Um diese Defizite zu beseitigen, wird mit den erweiterten Roboterzustandsautomaten ein Modell vorgestellt, dem entsprechende Komponenten hinzugefügt wurden, wiederum gefolgt von einem kleinen Beispiel.

### 3.2.1. Formalismus ERSA

**Definition 3.2.1** (Erweiterter Roboterzustandsautomat). Ein erweiterter Roboterzustandsautomat, kurz ERSA (für engl. *Extended Robot State Automaton*), ist definiert als Tupel  $M = (Q, \Sigma, P, n, \delta, u, q_s)$ . Dabei ist

- $Q$  die endliche Zustandsmenge
- $\Sigma$  das endliche Eingabealphabet
- $P$  der Raum an Posen
- $n$  als die Anzahl der verwendeten Posenvariablen definiert via  $D = P^n$  den Raum des Vektors für Posenvariablen
- $\delta : Q \times \Sigma_\varepsilon \rightarrow Q$  die Transitionsfunktion
- $u : Q \times \Sigma_\varepsilon \rightarrow com$  die Updatefunktion
- $q_s \in Q$  der Startzustand

In der Ausführung befindet sich ein ERSA-System jeweils in einem aktuellen Zustand  $q \in Q$ , hat einen aktuellen Posenvariablenvektor  $d \in D$ , und der Roboter befindet sich in einer aktuellen Pose  $p_{cur} \in P$ . Falls es sich bei  $q$  um einen Verzweigungszustand handelt (nach derselben Unterteilung wie für RSA), wird vor dem Zustandsübergang eine Eingabe  $\sigma \in \Sigma$  erfasst, und falls dabei ein Objekt erkannt wird, wird dessen Pose als  $p_{obj}$  abgelegt. Für die Updatefunktion kann auch hier wieder angenommen werden, dass  $\forall q \in Q, \sigma \in \Sigma_\varepsilon : [u(q, \sigma) \text{ definiert} \iff \delta(q, \sigma) \text{ definiert}]$ . Der Wert  $u(q, \sigma) \in com$  gibt dann das Update entlang dieser Transition an, wobei die Menge  $com$  der zulässigen Kommandos wie folgt definiert ist:

$$com = \left\{ \begin{array}{l} move\_abs[\bar{p}](out : p_{cur}), \\ move\_rel[\bar{p}](inout : p_{cur}), \\ move\_obj[\bar{p}](in : p_{obj}, out : p_{cur}), \\ save\_var[j](in : p_{cur}, inout : d), \\ load\_var[j](in : d, out : p_{cur}) \end{array} \right\}, 1 \leq j \leq n, \bar{p} \in P \quad (3.1)$$

Der allgemeine Aufbau von Kommandos ist wie folgt zu lesen: Nach dem Namen werden in eckigen Klammern die sogenannten *festen* Parameter angegeben (deren Wert zur Programmierzeit festgelegt wird), und in runden Klammern die sogenannten *freien*

Parameter (deren Wert erst zur Laufzeit ermittelt wird). Freie Parameter sind im System definierte Variablen und können hier nur drei Formen haben:  $p_{cur}$ ,  $p_{obj}$  oder  $d$ . Sie sind weiterhin als *in*, *out* oder *inout* markiert, je nachdem, ob sie durch das Update gelesen werden, geschrieben werden oder beides. Dabei kann  $p_{obj}$  nicht überschrieben werden (d.h. nur als *in*-Parameter auftreten), da das System Objektposen nicht direkt verändert, sondern nur über Aufgreifen und anschließendes Bewegen. Diese Kennzeichnung ist inspiriert durch die Verwendung in der Programmiersprache Ada [154].

Die einzelnen Fälle der Updatefunktion korrespondieren (neben dem Basisfall einer absoluten Bewegung) genau mit den zu Beginn des Abschnitts 3.2 motivierten Defiziten. Zur genaueren Erläuterung, auch in Anbetracht der Notation, folgt eine detailliertere Beschreibung.

Der oberste Fall von Kommandos,  $move\_abs[\bar{p}](out : p_{cur})$ , kodiert eine durch den Roboter auszuführende absolute Bewegung, d.h. zu einer absolut definierten Zielpose  $\bar{p} \in P$ . Die absolute Bewegung benötigt keine weiteren Information zur Laufzeit, daher gibt es keine *in*- oder *inout*-Parameter. Die Variable, die durch das Update neu gesetzt wird, ist die aktuelle Roboterpose  $p_{cur}$ , die deshalb als *out*-Parameter gelistet ist. Im Zusammenhang mit dem Schreiben in  $p_{cur}$  wird auch der physische Roboter neu verfahren.

Der nächste Fall ist die relative Bewegung mit  $move\_rel[\bar{p}](inout : p_{cur})$ . Dabei geht neben dem (festen) Offset, als Transformation gegeben, auch die aktuelle Pose des Roboters in das Update mit ein. Zur Laufzeit verfährt der Roboter damit in die Pose  $p' = p_{cur} \cdot \bar{p}$ , wendet also die gegebene Transformation auf die aktuelle Roboterpose an. Diese wird damit sowohl gelesen als auch neu gesetzt und ist entsprechend als *inout* markiert.

Der dritte Fall  $move\_obj[\bar{p}](in : p_{obj}, out : p_{cur})$ , auch als objektrelative Bewegung bezeichnet, beschreibt das Verfahren zu einer Zielpose, die relativ zu einem wahrgenommenen Objekt definiert ist. Der Roboter bewegt sich dabei zu  $p' = p_{obj} \cdot \bar{p}$ . Hier geht die Objektpose  $p_{obj}$  als *in*-Parameter ein,  $p_{cur}$  hingegen nur als *out*-Parameter, da die vorherige Pose nicht relevant ist.

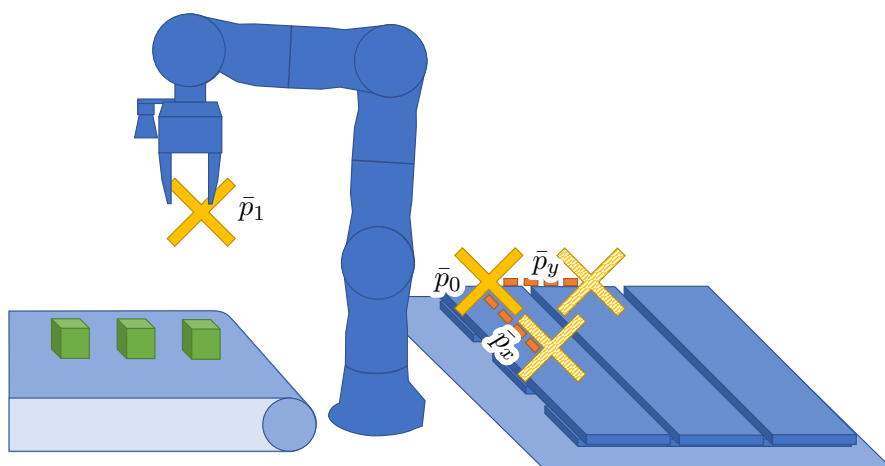
Fall vier beschreibt mit  $save\_var[j](in : p_{cur}, inout : d)$  das Speichern der aktuellen Pose  $p_{cur}$  in der  $j$ -ten Komponente  $d_j$  des Posenvariablenvektors. Hier ist  $j$  ein zur Programmierzeit bekannter fester Parameter (ein Index mit  $1 \leq j \leq n$ ), während die Werte der freien Parameter  $p_{cur}$  (die abzuspeichernde Pose) und  $d$  (der übrige Inhalt des Variablenvektors) erst zur Laufzeit ermittelt werden. Die Markierung von  $d$  als *inout*-Parameter rührt daher, dass die übrigen Komponenten außer  $d_j$  unverändert zurückgeschrieben werden sollen und dazu konzeptionell gelesen werden.

In Fall fünf wird zuletzt eine in einer Komponente des Posenvariablenvektors gespeicherte Pose durch den Roboter angefahren. Formal als  $load\_var[j](in : d, out : p_{cur})$  dargestellt, spezifiziert  $j$  auch hier wieder einen Index innerhalb von  $d$ , und zwar den der Komponente  $d_j$ .

Updates in einem ERSA, entlang eines Zustandsübergangs, entsprechen also stets einem dieser fünf Fälle, wobei die festen Parameter die konkrete Ausprägung spezifizieren.<sup>5</sup>

---

<sup>5</sup>Wie bereits bei den RSA bietet die Definition der Updatefunktion auch hier viel Spielraum für Varianten. Es könnten etwa weitere spezielle Roboteraktionen abgedeckt werden, oder andere Formalismen wie *Dynamic Movement Primitives* (DMP) [140] verwendet werden. Das Modell in dieser Arbeit



**Abbildung 3.4.:** Eine Beispielaufgabe für die Verwendung von ERSA. Ein Roboter palettiert Objekte auf einer Palette unbekannter Größe, die an einer gegebenen Ecke bereitgelegt wird. Die mit Zahlen nummerierten  $\bar{p}_0$  und  $\bar{p}_1$  stehen für feste Posen (Ecke der Palette und Ankunftsort der Objekte), die mit Buchstaben gekennzeichneten  $\bar{p}_x$  und  $\bar{p}_y$  für (relative) Transformationen, nämlich das Inkrement von einer Zeile bzw. Spalte zur nächsten.

Die Ausführung eines ERSA läuft wie folgt ab: Das System befinde sich im aktuellen Zustand  $q \in Q$ . Eine Eingabe  $\sigma \in \Sigma$  sei erfasst worden, falls  $q$  ein Verzweigungszustand ist; Falls  $q$  ein Transitzustand ist, sei dagegen  $\sigma = \varepsilon$ . (Falls es sich bei  $q$  um einen Terminalzustand handelt, ist eine weitere Ausführung an der Stelle aktuell nicht möglich.) Der einzelne Ausführungsschritt besteht nun wiederum darin, dass das System in  $q' = \delta(q, \sigma)$  übergeht (falls definiert).<sup>6</sup> Dabei wird zunächst das Update  $u(q, \sigma)$  zu dieser Transition durch den Roboter ausgeführt, bevor der Zustand auf  $q'$  gesetzt wird.

Als Anmerkung: Für Updates in ERSA ist also zu trennen zwischen der Updatefunktion  $u$ , und einem konkreten Update  $u(q, \sigma)$ , was der Wert der Updatefunktion an einer bestimmten Stelle ist. Einzelne Updates sind selbst Kommandos aus  $com$ , also gewissermaßen auch Funktionen, die dann auf den aktuellen Zustand des Roboters angewendet werden. In der Ausführung wird beim Verfolgen einer Transition zunächst das zugehörige konkrete Update ermittelt (durch Auswertung der Updatefunktion), und dieses dann im zweiten Schritt durchgeführt (oder angewendet).

### 3.2.2. Anwendungsbeispiel

Zur Illustration des ERSA-Formalismus wird wieder ein Anwendungsbeispiel vorgestellt. Das Szenario ist dabei wie in Abbildung 3.4 dargestellt: Objekte kommen auf einem

stellt wieder den Versuch eines möglichst einfachen Funktionsumfangs unter Verwendung geringer Zusatzfähigkeiten z.B. im Bereich der Perzeption dar.

<sup>6</sup>Eine Möglichkeit, wann das neben dem Erreichen eines Terminalzustands (ohne ausgehende Transitionen) nicht gilt, ist, wenn in einem Verzweigungszustand keine Transition für die erfasste Eingabe vorliegt. Falls das eintritt, kann das System später erneut eine Eingabe aufnehmen, in der Erwartung, dass Benutzende oder sonstige äußere Einflüsse die Situation verändert haben.

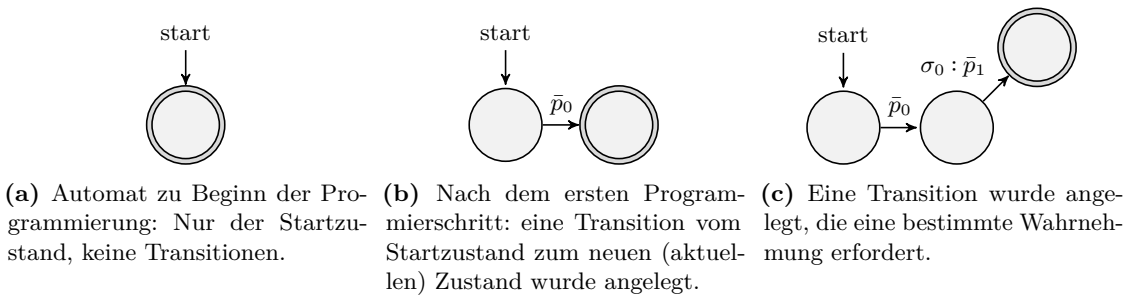


Automaten gemäß  $\delta$  stattfindet, wird dazu der Wert von  $u$  ermittelt, also das konkrete Update, das ausgeführt werden soll (bestehend aus dem Updatetyp,  $move\_rel$ , und dem zugehörigen festen Parameter, der Transformation  $\bar{p}$ ). Dieses Update wird dann auf die tatsächliche Roboterpose angewendet. Im Gegensatz zu absoluten Bewegungen können so auch Posen angefahren werden, die in der Programmierung nicht explizit angelegt wurden, wie im Beispiel die (unbekannt vielen) weiteren Ablageplätze auf einer neuen Palette.

Objektrelative Bewegungen oder Posen werden im Beispiel an der Stelle verwendet, an der ein neues Objekt aufgegriffen wird. Das an dieser Kante in Form von „ $p \leftarrow p_{obj} \cdot \bar{p}_r$ “ kodierte Update drückt aus, dass als neue Pose die wahrgenommene Pose des Objekts mit einem vorgegebenen Versatz angefahren werden soll. In der Updatefunktion wäre das ein Updatetyp von  $move\_obj$  kombiniert mit dem festen Parameter  $\bar{p}_r$ . Letzterer stellt z.B. die Transformation zwischen der Pose des Objekts selbst und einer darüberliegenden Aufgreifpose dar. Die neue Zielpose ist an dieser Stelle entsprechend durch die Objektpose klar vorgegeben und hängt nicht mehr von der vorherigen Roboterpose ab. Das Update wird beim Übergang nun auf die tatsächliche Roboterpose (und Objektpose) angewendet, sodass die bisherige Pose durch die berechnete Pose überschrieben wird und dabei der Roboter auch in diese neue Pose verfährt.

Wenn anschließend weitere relative Bewegungen verfolgt werden, ist dieser ganze Teil des Programms an die Objektpose angepasst. Zu bemerken ist, dass objektrelative Bewegungen nur direkt aus Verzweigungen und nur für Zweige mit einem erkannten Objekt zulässig sind – nur unter diesen Bedingungen sind sie wohldefiniert (da neben dem erkannten Objekt auch dessen Pose aus der Objekterkennung zurückgeliefert wird). Daher können eventuelle weitere davon abhängige Aktionen (z.B. zum Aufgreifen wie in Abbildung 3.2: mit den Greiferbacken um das Objekt verfahren, Greifer schließen, wieder zur darüberliegenden Zwischenpose fahren) auch nur über (herkömmliche) relative Bewegungen definiert werden.

An diesem Beispiel kann, neben der Funktionsweise des Formalismus, auch erläutert werden, dass Posenvariablen Aufgaben häufig beschleunigen können. Hier ist es etwa so, dass der Schritt zur nächsten freien Ablageposition auch für jedes einzelne Objekt immer von der Ecke als Basis neu generiert werden könnte, durch zeilen- und spaltenweise wiederholte Prüfung. Das würde allerdings in der Ausführung einen erheblichen Mehraufwand darstellen. Mit einer einzelnen Posenvariable kann entweder die Basis der aktuellen Spalte gespeichert werden, oder die letzte Ablageposition, von der aus weiter verfahren wird. Beide Varianten sparen aber nur einen Teil der redundanten Prüfung ein (bei der Spaltenbasis muss trotzdem für jedes Objekt innerhalb der Spalte neu geprüft werden, bei der Ablageposition muss zum Schritt auf die nächste Spalte zunächst Position für Position zum Beginn zurückgekehrt werden). Die grundlegende Funktionalität ist mit zwei, einer oder auch gar keiner Posenvariable umsetzbar, und der Aufwand zur Verarbeitung eines einzelnen Objekts steigt mit sinkender Variablenanzahl.



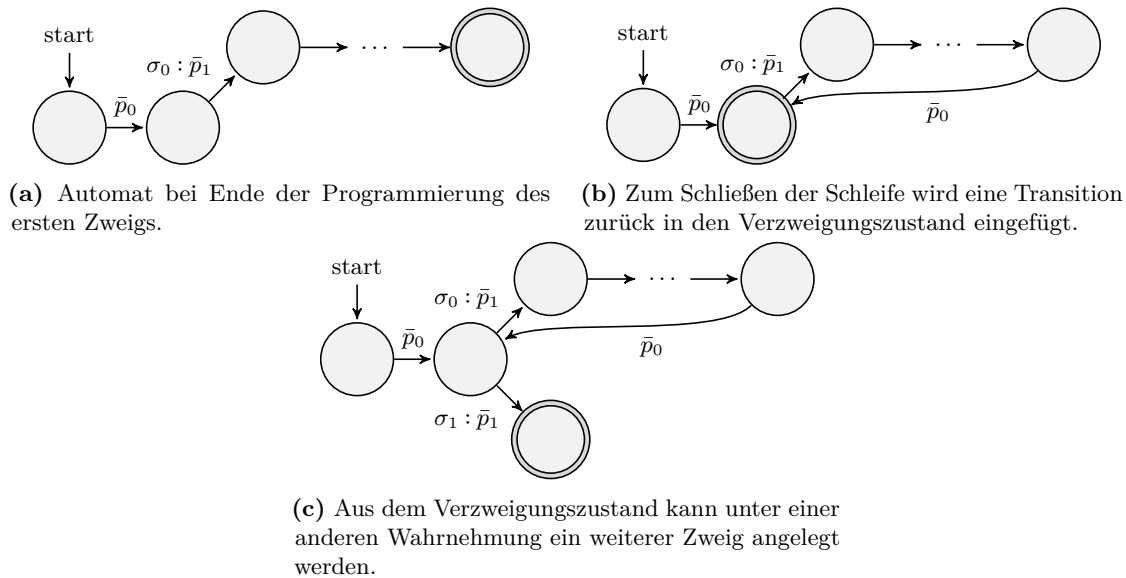
**Abbildung 3.6.:** Beispiel für das grundlegende Programmierkonzept, am Fall eines RSA. Der aktuelle Zustand ist jeweils mit einer doppelten Umrandung kenntlich gemacht. Ausgehend vom Startzustand (3.6a) spezifiziert der Mensch zunächst das erste Update (also eine neue Zielpose). Dazu wird ein neuer Zustand angelegt und durch eine neue Transition vom Startzustand aus angebunden (3.6b). Im zweiten Schritt spezifiziert der Mensch erst eine Wahrnehmung, dann ein zugehöriges Update. Daraus wird im Automaten eine Transition generiert, die nicht spontan ist, sondern zu der die erforderte Wahrnehmung  $\sigma_0$  abgespeichert wird. Wiederum geht das System in den neu angelegten Zielzustand über (3.6c).

### 3.3. Programmierung

Der Prozess zur Programmierung eines Roboterzustandsautomaten, einfach oder erweitert, ist grundlegend stets gleich: Ausgehend von einem einzigen Initialzustand als aktuellem Zustand ( $q_s \in Q$ , wobei  $Q = \{q_s\}$ ), legen die Nutzenden in jedem Schritt das Update entlang einer neuen Transition an. Das System generiert dazu einen neuen Zielzustand  $q'$ , der in  $Q$  eingefügt wird, und geht direkt in diesen als neuen aktuellen Zustand über. Um das Update zu spezifizieren, wird der Roboter kinästhetisch geführt, d.h. unter Kompensation des eigenen Gewichts von den Nutzenden gegriffen und in die Zielpose bewegt.

Dieser Prozess wird für einen einfachen Roboterzustandsautomaten in Abbildung 3.6 verdeutlicht. Dabei handelt es sich um den Beginn des Automaten aus Abbildung 3.2. Der Mensch führt dabei als erstes den Roboter in die erste Zielpose  $\bar{p}_0$  und löst dort die Programmieroperation für eine neue Bewegung aus. Es wird im Automaten ein neuer Zielzustand generiert und über eine neue Transition an den vorher aktuellen Zustand angehängt. Das Update parallel zu dieser Transition beinhaltet dann genau die programmierte Aktion (Bewegung nach  $\bar{p}_0$ ). Im zweiten Schritt nimmt der Mensch zuerst über eine Operation des Systems eine Wahrnehmung auf (also in diesem Fall einen bestimmten Objekttyp). Diese Wahrnehmung  $\sigma_0$  wird intern abgelegt. Als nächstes spezifiziert der Mensch wieder das gewünschte Update durch Führen des Roboters zur Zielpose und anschließendes Auslösen der Programmieroperation. Intern wird dann erneut ein neuer Zustand angelegt und über eine Transition an den vorherigen Zustand angeschlossen. Diese Transition wird nun nicht unter Eingabe  $\varepsilon$  angelegt, sondern unter der vorher spezifizierten Wahrnehmung  $\sigma_0$ .

Bei einem einfachen Roboterzustandsautomaten muss für einen neuen Updateschritt jeweils nur festgelegt werden, ob dieser spontan passieren soll oder eine Eingabe erforder-



**Abbildung 3.7.:** Fortgeführtes Beispiel, hier zum Schließen von Schleifen. Der angefangene Automat aus Abbildung 3.6 wurde weiterprogrammiert und der erste Zweig damit abgeschlossen (3.7a, Aktionen innerhalb des Zweigs ausgelassen). Als nächstes muss der Mensch die Schleife schließen, z.B. durch Zurückführen des Roboters an die ursprüngliche Pose beim Programmieren der Verzweigung und dortiges Auslösen der Verbindeoperation. Eine Transition zurück wird angelegt, und das System geht wieder in den Verzweigungszustand als aktuellen Zustand über (3.7b). Anschließend kann wie in Abbildung 3.6 ein neuer Zweig angelegt werden, durch Spezifikation der erforderlichen Wahrnehmung  $\sigma_1 \neq \sigma_0$  und der eigentlichen Aktion (3.7c).

dern soll (die im letzteren Fall auch abgespeichert werden muss). Weiterhin muss es zum Erzeugen von Schleifenstrukturen möglich sein, eine Transition zu einem bereits existierenden Zustand herzustellen, was auch als *Verbinden* bezeichnet wird. Alternativ kann die Struktur auch aus Überlappung von Teilen des programmierten Automaten synthetisiert werden, falls also z.B. ein weiterer Durchlauf einer Schleife programmiert wurde. Diese Variante wird in Kapitel 5 genauer ausgeführt.

Zum Schließen von Schleifen ist in Abbildung 3.7 das vorherige RSA-Beispiel fortgeführt. Nach vollständiger Programmierung eines Zweigs muss eine Kante zurück in den Verzweigungszustand angelegt werden. Dazu wird der Roboter zu der Pose zurückbewegt, in der der Verzweigungszustand ursprünglich angelegt wurde, und dort die Verbindeoperation ausgelöst.<sup>7</sup> Das System erzeugt daraufhin die Transition und geht direkt in den Zielzustand über, hier also wieder den Verzweigungszustand. Dort können dann weitere Zweige in derselben Weise wie der erste angelegt werden.

Die Eingabe der unterschiedlichen Updatefunktionen in einem erweiterten Roboterzustandsautomaten erfordert hingegen eine größere Anzahl an programmierbaren Fällen. Prinzipiell muss jede der fünf Varianten durch eine eigene Operation abgedeckt werden.

<sup>7</sup>Für die Bewegung entlang dieser Transition wie im Automaten dargestellt muss zusätzlich die Programmieroperation erfolgen.

Für objektrelative Bewegungen kann eine Heuristik auf Basis der Distanz zwischen Greifer (bzw. Tool Center Point) und Objekt verwendet werden: Wenn eine Bewegung zu einem Ziel in unmittelbarer Nähe des wahrgenommenen Objekts spezifiziert wird, dient diese in der Regel zum Aufgreifen und kann so als objektrelativ angelegt werden.<sup>8</sup> Auf die spezifischen Eingabemethoden wird in Kapitel 4 genauer eingegangen.

Ein wichtiges Detail zum Programmierprozess besteht darin, dass Programmierung und Ausführung verwoben stattfinden können, etwa zur Programmierung mehrerer Fälle in einer Verzweigung. Solange sich das System in einem Terminalzustand als aktuellem Zustand befindet, können Nutzende durch Spezifikation eines Updateschritts den Automaten wie oben angegeben erweitern. Neben dem Übergang in einen neu angelegten Zustand ist wie beschrieben auch das Verbinden zu einem bereits existierenden Zustand möglich. Zur Spezifikation des anzuspringenden Zielzustands wird der Roboter hier wiederum kinästhetisch in eine ähnliche Pose bewegt, wie sie zur Programmierung des Zielzustands vorlag. Nach der Programmierung solch einer Transition befindet sich das System in einem Zustand, in dem schon ausgehende Kanten vorhanden sind. Nun können von diesem Zustand aus bereits programmierte Operationen ausgeführt werden, bis das System wieder einen Terminalzustand oder einen Verzweigungszustand erreicht hat, wo in beiden Fällen neue Transitionen erzeugt werden können. Abgesehen davon ist es allgemein in jedem Zustand möglich, durch Spezifikation eines neuen Updateschritts das bisher Programmierte zu überschreiben, in Verzweigungen auch für einen bestimmten Zweig. Neben dem Verbinden kommt der verwobene Programmierprozess auch zur Anwendung, wenn das Programm an irgendeinem Punkt vom Anfangszustand aus neu betreten wird, was ebenfalls möglich ist.

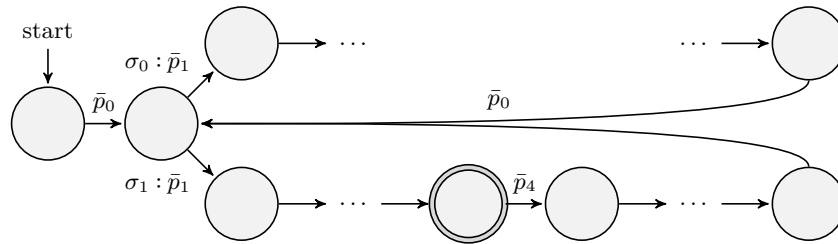
Ein RSA-Beispiel für diesen verwobenen Prozess ist in Abbildung 3.8 gegeben. Der Automat aus den vorherigen Beispielen wurde fertig programmiert und anschließend zunächst weiter ausgeführt. Dann ändert sich ein Teil des Aufbaus, sodass die Ablageposen für den unteren Zweig verändert werden müssen. Dazu wird mit der Ausführung zunächst bis unmittelbar vor der ersten betroffenen Bewegung fortgefahren. Dort wird dann ein neuer Übergang programmiert. Dieser steht im Konflikt mit dem bisher existierenden, sodass letzterer überschrieben wird. Ab hier kann die Programmierung wie zu Beginn weiter erfolgen.

Der Grundgedanke hinter diesem verwobenen Prozess ähnelt dem anderweitig als *sliding autonomy* [59] oder *adjustable autonomy* [42] bezeichneten: Statt einer klaren Trennung in Programmierungs- und Ausführungsphase lassen Nutzende den Roboter auf dem bereits programmierten (oder, in anderen Kontexten, gelernten) Teil der Aufgabe autonom agieren, und ziehen diese Autonomie dann teilweise und nach Bedarf wieder an sich. Der fließende Übergang bringt den Vorteil, genau und erst zu dem Zeitpunkt und an der Stelle in die Ausführung eingreifen zu können, an der Änderungsbedarf besteht (wie etwa in Abbildung 3.8 nach einer Änderung in der Aufgabe selbst, an genau der betroffenen Stelle im Programm). Auch hinter dem in [47] beschriebenen System steht etwa ein ähnlicher Gedanke.

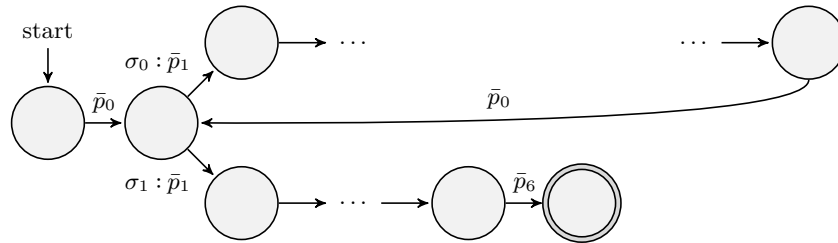
---

<sup>8</sup>Heuristiken dieser und ähnlicher Art werden in anderem Kontext beispielsweise in [5, 7, 60] verwendet (und diskutiert).





(a) Im Automaten sollen die Ablageposen für den unteren Zweig verändert werden.



(b) Nach Eingabe einer anderen Aktion (Bewegung zu einer anderen Zielpose) wurde die alte Transition ersetzt, und es kann neu weiterprogrammiert werden.

**Abbildung 3.8.:** Beispiel für den verwobenen Programmierprozess und das Überschreiben aus der Ausführung. Der Automat aus den vorherigen Beispielen wurde vervollständigt, wie ursprünglich in Abbildung 3.2 dargestellt. Nun soll die Ablageposition für den unteren Zweig verändert werden. Dazu wird zunächst bis direkt vor der entsprechenden Bewegung ausgeführt (3.8a). Anschließend wird eine neue Aktion programmiert, in diesem Fall eine Bewegung zur neuen Zielpose  $\bar{p}_6$ . Da keine Wahrnehmung spezifiziert wurde, wird diese Bewegung wieder an eine  $\varepsilon$ -Transition zu einem neuen Zustand annotiert. Diese überschreibt die bisherige Transition (3.8b). Damit kann von dieser Stelle aus weiterprogrammiert werden. Der dahinter liegenden Teil des Zweigs wird unerreichbar und in der Folge aus dem Automaten entfernt.

Eine naheliegende Variante des Systems wäre es, die Trajektorie während der Programmierung andauernd automatisch aufzunehmen (d.h. entsprechend neue Zustände in hoher Taktrate jeweils mit Bewegungen anzuhängen). Dazu müssten zunächst die Eingaben anders gestaltet werden. Die automatisch generierten Bewegungen sollten standardmäßig relativ zu vorherigen sein, sonst wäre es nur schwer möglich, überhaupt relative Bewegungen anzugeben. Explizite Eingaben müssten demnach für absolute Bewegungen gegeben werden, sowie für die diversen anderen Updates.

Die Variante hätte den Vorteil, dass Nutzende von den generierten Bewegungen zwischen einzelnen Zielposen nicht überrascht werden bzw. diese eben selbst angeben. Dem stehen jedoch einige Nachteile gegenüber: Insbesondere bei unerfahrenen Nutzenden kommen in der reinen Trajektorie während der Programmierung recht häufig Stillstände und kleinere oder größere Bewegungen vor, die nicht zur eigentlichen Aufgabe gehören.<sup>9</sup> Es entstehen so auch deutlich größere Automaten, was das gezielte Verbinden erschwert.

Beobachtungen dieser Art werden auch schon in [2] geäußert, wo u.a. Demonstration durch Trajektorien und durch einzelne Zielposen (dort *Keyframes* genannt) verglichen wird. Im Gegensatz zu den dortigen Aufgaben ist bei den Pick-and-Place-Szenarien, auf die das System in dieser Arbeit abzielt, die Trajektorie nur zur Vermeidung von Hindernissen relevant, nicht als Teil der Aufgabe selbst. Außerdem ist zu erwarten, dass die mentale Anstrengung, um strukturierte Programme zu erzeugen, nochmals höher ist. Insgesamt fiel die Entscheidung im Rahmen der vorliegenden Arbeit daher auf den in diesem Kapitel beschriebenen Teach-In-Ansatz. Der Gedanke einer hybriden Variante (ähnlich wie z.B. ebenfalls in [2] beschrieben) wird aber im Ausblick kurz aufgegriffen.

---

<sup>9</sup>Dieses Argument spricht auch dagegen, einzelne Zielposen bei ausreichend langem Stillstand abzuspeichern wie in [143] vorgeschlagen.

## KAPITEL 4

---

### Haptische Eingaben

---

4.1. Stand der Forschung . . . . .	48
4.2. Erkennung haptischer Gesten . . . . .	50
4.2.1. Vorüberlegungen . . . . .	50
4.2.2. Einfache Ruckbewegungen . . . . .	51
4.2.3. Komplexere Gesten . . . . .	55
4.3. Vorstudie und Implementierung . . . . .	56
4.3.1. Studienentwurf . . . . .	57
4.3.2. Ergebnisse . . . . .	58
4.3.3. Schlussfolgerungen . . . . .	63

---

Wie in Kapitel 1 motiviert, soll ein möglichst großer Teil der Eingaben direkt über den Roboter stattfinden. Dazu sind *haptische Interaktionen* ein plausibles Mittel, da sie den Nutzenden die Interaktion direkt aus der kinästhetischen Führung ermöglichen, wobei sogar beide Hände in der gleichen Haltung am Roboter verbleiben können.

Im Kontext dieser Arbeit wird *haptisch* in der Bedeutung von physischer Interaktion mit dem Roboter benutzt. Der Begriff steht zuweilen neben den angrenzenden Bezeichnungen *taktil* und *kinästhetisch*. Dabei hat sich im Robotikbereich keine einheitliche Verwendung oder Definition durchgesetzt:

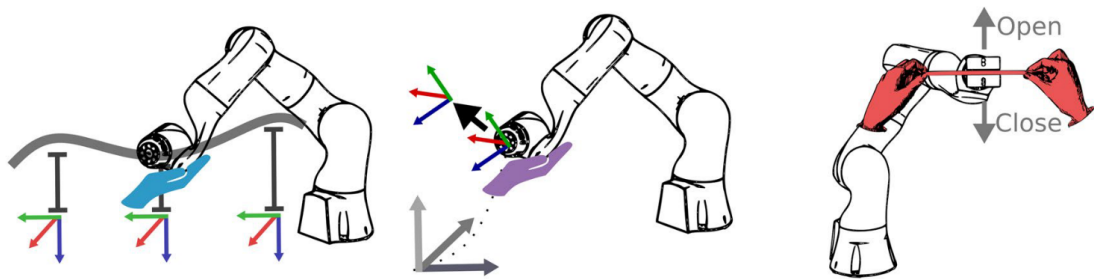
„There is no consensus over the definitions of tactile and haptic interactions.“ [34]

Wie bisher auch wird im Folgenden kinästhetisch entsprechend dem natürlichsprachlichen englischen Begriff *kinesthetic* (oder *kinaesthetic*) benutzt, um etwas auf die Fähigkeit zur Wahrnehmung des eigenen Körpers zu beziehen (wie eben beim kinästhetischen Führen der Roboter seine eigenen Posen wahrnimmt).<sup>1</sup> Mit *taktil* wird, ebenfalls auf Basis der Bedeutung des englischen *tactile*, der Berührungssinn referenziert.<sup>2</sup> Damit bildet der

---

<sup>1</sup><https://dictionary.cambridge.org/dictionary/english/kinaesthetic>, besucht: 11.11.2022

<sup>2</sup><https://dictionary.cambridge.org/dictionary/english/tactile>, besucht: 11.11.2022



**Abbildung 4.1.:** Skizze aus [153] zur dortigen Verwendung haptischer Eingaben. Neben der ursprünglichen Eingabe (links) und späteren Einstellung von Posenparametern (Mitte) wird dort eine Schlüsselgeste zur Übergabe von Objekten verwendet: Ein Auslenken des Roboters nach oben bzw. unten zum Öffnen bzw. Schließen des Greifers.

Begriff *haptisch* gewissermaßen eine Obermenge beider Begriffe – In dieser Verwendung ist eine taktile oder kinästhetische Schnittstelle eigentlich auch eine haptische, da beides die physische Interaktion mit dem Roboter betrifft. Für die Gesten aus der Führung, um die es im weiteren Verlauf dieses Kapitels gehen soll, wird trotzdem der Begriff der haptischen Interaktion (statt der kinästhetischen Interaktion) verwendet, um sie im Text klarer von der kinästhetischen Führung abzugrenzen.

## 4.1. Stand der Forschung

Zunächst werden verwandte Arbeiten betrachtet, in denen ebenfalls haptische Interaktionen zur Anwendung kommen. Davor sei nochmals die Abgrenzung deutlich gemacht: Es geht um bestimmte Bewegungen oder Gesten, die von Nutzenden mit dem Roboter ausgeführt werden, und die einzelne Kommandoingaben im System auslösen. Insbesondere geht es nicht um Gesten, die die Nutzenden mit ihren eigenen Händen oder Körpern ausführen. Wie durch die Verwendung des Wortes haptisch signalisiert, geht es auch nicht um die Erkennung von Kontakt mit oder gezeichneten Gesten auf der Oberfläche des Roboters. Zuletzt geht es explizit darum, mit den haptischen Gesten einzelne Funktionen des Systems auszulösen, und nicht darum, die geführte Trajektorie selbst zu verwenden.

Zur Verwendung solcher bestimmten haptischen Interaktionen während der Führung gibt es kaum Vorarbeiten. Es wurden nur zwei dokumentierte Verwendungen haptischer Gesten in der Forschung gefunden [153, 167]. In [153] kommen diese, als haptische Interaktionsprimitive betitelt, im Kontext eines Systems für Assistenz im Baugewerbe zur Anwendung. Dabei geht es primär um die Parametrierung vorher geplanter Bewegungen. Der Roboter kann in einzelnen Situationen eine nachgiebige Pose einnehmen, in der die Anwender durch bestimmte Bewegungen unterschiedliche Eingaben signalisieren können, beispielsweise ein Öffnen und Schließen des Greifers durch Auslenken nach oben oder unten. In Abbildung 4.1 ist die begleitende Skizze reproduziert. In der Arbeit wird explizit der Unterschied festgestellt, dass die Verwendung der haptischen Gesten außerhalb der eigentlichen Demonstration liegt, was sich genau mit der oben getroffenen Charakterisie-



**Abbildung 4.2.:** Konzeptgrafiken aus [167] zu verschiedenen Varianten für eine Schlüsselgeste zum Öffnen und Schließen des Greifers. Es werden dort kraftbasierte und positionsbasierte Richtungswechsel vorgeschlagen (Mitte), in beliebigen Achsen oder solchen des NSA-Koordinatensystems (links), sowie rotationsbasierte Richtungswechsel (rechts). Letztere Möglichkeit wurde in den Experimenten in der Arbeit verwendet.

rung deckt.

Eine weitere Vorarbeit, die auch als Motivation für die weitere Betrachtung haptischer Gesten diente, liegt in [167] vor. Dort wird ebenfalls auf die Anwendung eingegangen, den Greifer zu öffnen und zu schließen, in diesem Fall direkt während der kinästhetischen Führung. Zu diesem Zweck werden drei Ansätze vorgestellt: mit einem kraftbasierten Richtungswechsel, mit einem positionsbasierten Richtungswechsel, und mit einem rotatorischen Richtungswechsel. Anhand einer Betrachtung a priori wurde für die Experimente in der Arbeit der rotatorische Richtungswechsel gewählt und umgesetzt.

Eine Anwendung haptischer Gesten in einem kommerziellen System besteht im Desk Interface für die Panda-Roboter der Firma Franka Emika. Dort werden Antippgesten als Möglichkeit der Bestätigung angeboten, etwa vor der erneuten Ausführung einer Schleife. In der Programmierung kann festgelegt werden, in welchen Achsen und Richtungen solche Gesten akzeptiert werden [52].

Aus dem Endanwenderbereich ist die Steuerung mittels Gesten eines Eingabegeräts wohl hauptsächlich im Bereich der Unterhaltungssoftware (d.h., Videospiele) bekannt, wobei die Verwendung des Eingabegeräts dort womöglich nur als technischer Lösungsweg zu sehen ist, um die (Hand-) Pose der Anwendenden selbst zu erfassen. Solche Ansätze finden grob seit der Markteinführung zweier Konsolen vermehrt Anwendung, der Playstation 3 von Sony und der Nintendo Wii [125]. Bei diesen und ähnlichen Geräten (bzw. deren verschiedenen Nachfolgern) werden in vielen Anwendungen bestimmte mit dem Controller ausgeführte Gesten erfasst und lösen Aktionen innerhalb des Spiels aus.

Im weiter gefassten Kontext mit der Verwendung taktiler Wahrnehmung für ansonsten auch haptisch umsetzbare Gesten finden sich weitere Arbeiten [76, 169]. Eine sehr einfache Variante wird in [76] beschrieben: Bei der multimodalen Interaktion mit einem Nao-Roboter der Firma SoftBank Robotics als Gesprächspartner wurde auf eine Geste zurückgegriffen, um den Roboter in seinem Redefluss zu unterbrechen. Dazu tippen die Anwender auf den Kopf des Roboters. In [169] geht es wiederum um taktile Gesten während der Führung, die hier ebenfalls über taktile Sensoren realisiert wird. Als bei-

spielhafte Geste wird für einen Wechsel zwischen zwei Führungsmodi zweimaliger kurzer Druck am anzusteuernenden Segment erwähnt. Auch für das Öffnen und Schließen des Greifers wird ein Vorschlag gemacht, bestehend aus einer Sequenz mehrerer Berührungen an den Greiferbacken. Arbeiten wie [84, 104] verwenden schließlich tatsächlich Alphabete taktiler Gesten, was sich aber nicht ohne weiteres auf haptische Gesten übertragen lässt.

Zusammenfassend ist das Feld haptischer Interaktionen noch wenig erforscht. Gleichzeitig wird die Motivation, Eingaben direkt während der Führung über den Roboter zu tätigen, in den wenigen Vorarbeiten deutlich, weshalb dieser Ansatz in der vorliegenden Arbeit zur Anwendung kommt und ebenfalls evaluiert wird.

### 4.2. Erkennung haptischer Gesten

In diesem Abschnitt werden verschiedene Möglichkeiten für haptische Gesten und deren Erkennung identifiziert. Der Suchraum für zunehmend beliebig komplexe Gesten ist unbegrenzt; Die Betrachtung geht daher als Startpunkt umgekehrt von den simpelsten Fällen aus, mit Heuristiken für einfache Ruckbewegungen. Aus praktischer Sicht sind einfache Gesten unter Abwesenheit anderweitiger Nachteile vor komplexen Gesten zu bevorzugen, da die Nutzbarkeit nach wie vor im Kern der Motivation steht. Im Gegenzug bieten komplexere Gesten mehr Spielraum für eine Entsprechung zur Nutzerintuition. Neben dieser Abwägung gibt es das Kriterium der Robustheit, wie in [167] formuliert in beide Richtungen:

- gegen falsch positive Detektion
- gegen falsch negative Detektion

Diese beiden Punkte entsprechen inhaltlich den Kriterien G1 und G3 aus [167]. Dort wird außerdem als Kriterium G2 Unabhängigkeit von gehaltenen Objekten genannt. Sowohl nach der Vision als auch nach den Nutzlasten und Greiferbauarten realer Leichtbauroboter beschränkt sich das System in dieser Arbeit auf mittelgroße, vergleichsweise leichte Objekte (in der Größenordnung 10 cm Breite in allen Dimensionen und 100 g Gewicht). Für solche Objekte ist wenig Einfluss auf die Führung durch die Nutzenden zu erwarten. Der weitere in [167] genannte Punkt von Ungenauigkeiten bei der Bestimmung der Lastdaten kommt hier ebenfalls nicht zum Tragen, da keine Objektgewichte erfasst oder berechnet werden. Daher ist das Kriterium G2 für den vorliegenden Anwendungsfall zunächst vernachlässigbar.

#### 4.2.1. Vorüberlegungen

Unabhängig von den konkreten Gesten gibt es einige allgemeine Fragen, die beim Entwurf betrachtet werden müssen. Darunter fallen etwa die verschiedenen Möglichkeiten, in welchem Kontext die Gesten überhaupt definiert sind. Wenn es um den kartesischen Raum geht, zeichnen die Nutzenden die Gesten gewissermaßen mit dem Endeffektor in die Luft. In dem Fall ist weiter zu entscheiden, ob nur die Position des Endeffektors

betrachtet wird, die volle Pose (also inklusive Orientierung), oder ggfs. auch nur die Orientierung. Es ist aber auch möglich, Posen direkt über Gelenkwinkel zu definieren, beispielsweise bei einfachen Ruck-/Drehbewegungen in bestimmten Gelenken, oder wenn tatsächlich die Stellung des gesamten Roboters betrachtet werden soll.

Bei Gesten im kartesischen Raum stellt sich ggfs. die Frage nach dem Referenzkoordinatensystem (auch: *Frame*), auf das sich die Geste bezieht, falls dieses fest vorgegeben sein soll. Naheliegende Alternativen sind hier Weltkoordinaten oder Endeffektorkoordinaten (häufig als NSA-Koordinaten mit den drei Achsrichtungen *Normal*, *Slide* und *Approach*); Aber auch z.B. die Verwendung von Objekten zur Definition eines Referenzsystems ist denkbar.

Auf technischer Seite ist je nach gewähltem Referenzsystem eine Überlegung, ob sich dieses während der Geste aktualisiert (z.B. mit dem Roboter mitbewegt), oder ob es zu Beginn der Bewegung einmal erfasst wird und dann fest ist.

Ebenfalls ein technischer Aspekt ist die Wahl, welche Daten der Gestenerkennung zugrunde liegen. Direkt über den Roboter sind üblicherweise zumindest die Gelenkstellungen und -geschwindigkeiten auszulesen. Mittels der Vorwärtskinematik kann auch die Endeffektorpose verwendet werden. Handführbare Leichtbauroboter bringen in der Regel auch Informationen über anliegende Kräfte und Momente mit sich, sowohl in den einzelnen Gelenken als auch (rekonstruiert) auf den Endeffektor bezogen.

Da für die Gesten im Allgemeinen wünschenswert ist, dass sie unabhängig von der aktuellen aufgabenbedingten Pose des Roboters ausgeführt und erkannt werden können, ist beispielsweise eine reine Verwendung von Positionen ungünstig, bzw. müssen Invarianzen dann erst erzeugt werden. Stattdessen können auch Geschwindigkeiten betrachtet werden (entweder direkt oder als Differenzen der Position), was bereits Invarianz gegen unterschiedliche Startposen liefert. Der Schritt zur Beschleunigung als zweite Ableitung würde Invarianz gegen einen konstanten Drift während der Geste erzeugen; dies dürfte allerdings deutlich seltener nötig sein. Kräfte stehen wiederum in linearem Zusammenhang zur Beschleunigung.

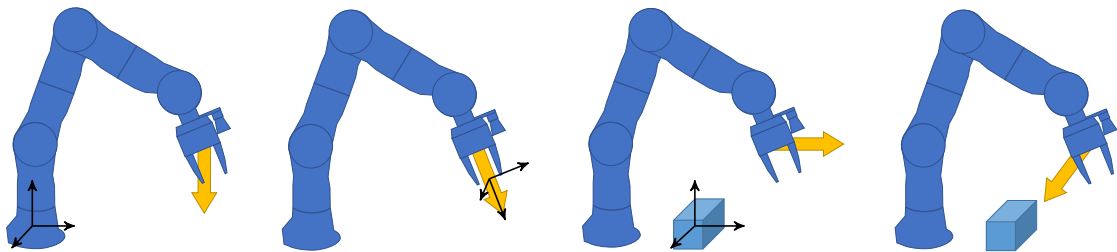
Im Gesamtentwurf ist zu guter Letzt zu beachten, dass verschiedene verwendete Gesten möglichst konfliktfrei definiert sein sollten. Eine Bewegung in Weltkoordinaten nach unten für eine Geste und eine Bewegung in Richtung der A-Achse des NSA-Systems für eine andere Geste sind z.B. nicht mehr sinnvoll unterscheidbar, wenn in der aktuellen Pose die beiden Achsen aufeinander fallen.

#### 4.2.2. Einfache Ruckbewegungen

Einzelne ruckartige Bewegungen stellen einige der einfachsten Möglichkeiten an Gesten dar, die während der Führung erkannt werden können, und bieten gleichzeitig schon eine gewisse Fülle an konkreten Varianten.

Eine Definition für den Zweck dieser Arbeit kann formuliert werden als: eine zu anderen Bewegungen abgegrenzte, deutliche Änderung ausreichender Dauer in einer Richtung oder Achse, während in allen anderen betrachteten Richtungen keine signifikante Änderung stattfindet.

Abgegrenzt heißt in diesem Kontext, dass die Bewegung in Stillstand beginnen und en-



**Abbildung 4.3.:** Varianten für bestimmte Richtungen bei Ruckbewegungen, von links nach rechts: in Weltkoordinaten (Beispiel: nach unten), in NSA-Koordinaten (Beispiel: in Richtung der A-Achse), im Koordinatensystem eines Objekts (Beispiel: in einer Achsrichtung), in Richtung des Objekts selbst. Die Referenzkoordinatensysteme sind jeweils angedeutet (soweit zutreffend).

den soll. Dies ist wünschenswert, damit in der Führung immer noch beliebige Bewegung möglich ist, ohne eine Eingabe auszulösen. Die deutliche Änderung kann mittels eines Schwellwerts definiert sein, wie auch eine Obergrenze für die Bewegung in den anderen Achsen festgelegt werden kann.

Diese Definition zielt auf das oben genannte Kriterium der Robustheit ab, zunächst konkret gegenüber falsch Positiven während der Führung. Keiner der Aspekte ist dazu für sich allein hinreichend: Nach den Erfahrungen früherer Experiment bewegen Nutzende den Roboter insbesondere bei größeren Bewegungen häufig auch in der Führung sehr schnell, und beim Nachdenken zwischen einzelnen Programmierschritten kommt es oftmals zu kurzen Stillständen und kleinen, unbeabsichtigten Bewegungen. Auch die ausreichende Dauer dient dazu, z.B. die kurzen, schwer vermeidbaren Rucke beim Greifen des Roboters auszuschließen. Kompletten Stillstand in anderen Achsen zu erfordern würde wiederum falsch Negative erzeugen; die menschlich geführte Bewegung im Freiraum erfolgt im Allgemeinen nicht exakt entlang einer Geraden.

### Ruckbewegungen des Endeffektors

Eine Ruckbewegung des Endeffektors lässt sich positions- bzw. geschwindigkeitsbasiert umsetzen.<sup>3</sup> Konkret ist dies eine wie oben beschriebene abgegrenzte, deutliche Änderung, in diesem Fall des Endeffektors in einer Achse im kartesischen Raum.

Bezüglich der Bewegungsrichtung stellt sich wie oben erwähnt die Frage, ob diese frei gewählt werden kann oder im Voraus festgelegt wird, sowie in letzterem Fall die konkrete Wahl der Achse und des Bezugssystems. Verschiedene Optionen sind in Abbildung 4.3 skizzenhaft visualisiert. Bei beliebiger Achse muss diese nach Stillstand aus der Bewegung selbst erfasst werden, z.B. als die der ersten Bewegung oder als Gesamtmittel. Bei vorgegebenen Richtungen sind insbesondere die Achsen des entsprechenden Bezugskordinatensystems einfach zu prüfen: Dann muss in jedem Zeitpunkt die entsprechende

<sup>3</sup>Wie in [167] ausgeführt, lässt sich diese Art von Gesten des Endeffektors auch z.B. kraftbasiert detektieren, was mit äquivalenten Anforderungen an den Kraftverlauf passieren kann. Da im vorliegenden Fall während der ganzen Zeit die kinästhetische Führung aktiv sein soll, ist das Resultat aus einem solchen Kraftverlauf aber immer auch entsprechende Änderung der Position.



Komponente in der Bewegungsrichtung einen Mindestwert erreichen, während die übrigen Komponenten einen Höchstwert nicht überschreiten dürfen. Für Welt- oder NSA-Koordinaten sind diese Informationen eventuell direkt verfügbar, für beliebige Frames über eine entsprechende Transformation.

Es wäre auch möglich, Richtungen über Referenzpunkte zu definieren, beispielsweise die Richtung zum Ursprung eines Objektkoordinatensystems. Es fallen wieder Richtungs-berechnungen an, und die Frage, ob die Richtung zum Referenzpunkt hin zu Beginn ermittelt und dann als fest angenommen wird, oder tatsächlich in jedem Schritt neu berechnet wird (was bei ungenauen Bewegungen nahe des Referenzpunktes einen Unterschied machen kann).

Bei der Betrachtung der Endeffektorpose kann neben der Position auch die Orientierung verwendet werden. Damit können ruckhafte Drehungen in derselben Weise definiert werden wie Bewegungen. Es stellt sich auch hier die Frage nach der Achse, insbesondere, ob diese vorgegeben oder beliebig ist. In jedem Fall muss die Orientierungsänderung in den einzelnen Zeitschritten berechnet werden, und dann gegen eine entsprechende Referenz (fest oder aus der Bewegung selbst ermittelt) verglichen werden.

Während theoretisch typische Leichtbauroboter mit dem Endeffektor alle Posen im Arbeitsraum einnehmen können, ist praktisch allerdings eine Orientierungsänderung, die nicht mit einem Gelenk des Roboters zusammenfällt, häufig mit aufwändiger Änderung der Gesamtpose verbunden. Dies gilt umso mehr, wenn die Position dabei beibehalten werden soll. Aus diesem Grund sind ruckhafte Drehbewegungen des Endeffektors für den gegebenen Anwendungsfall unvorteilhaft.

Einen Sonderfall stellt die Drehung um die Approach-Richtung des NSA-Koordinatensystems dar. Diese Achse fällt bei vielen Robotern mit der Achse des letzten Gelenks zusammen, so auch bei den in den Experimenten dieser Arbeit verwendeten. In dem Fall entspricht die Detektion der einer Ruck-/Drehbewegung im letzten Gelenk.

### **Ruck-/Drehbewegungen in Gelenken**

Dasselbe Prinzip wie bei Ruckbewegungen des Endeffektors lässt sich auf Bewegungen in einem Gelenk des Roboters anwenden. Hier handelt es sich dann um einen Stillstand, eine zeitlich begrenzte und deutliche Änderung und einen weiteren Stillstand im entsprechenden Gelenkwinkel. Für die Anwendenden äußert sich der Unterschied darin, dass statt einer Positionsänderung eine Drehung um die Gelenkachse durchzuführen ist, was sich koordinativ ähnlich einfach bis einfacher darstellt.

Da die einzelnen Gelenkstellungen von Haus aus getrennte Werte darstellen, stellt sich das Problem von Bewegungsrichtung und Referenzsystem hier nicht, bzw. reduziert sich auf die Wahl von Gelenk und Drehsinn. Auch hier muss beachtet werden, dass der konzeptionell zu erwartende Stillstand in allen anderen Gelenken in der Praxis nicht präzise eingehalten werden kann.

### Bewegungen in mittleren Segmenten

Ähnlich wie für den Endeffektor ließen sich Ruckbewegungen auch für andere Punkte entlang des Roboterarms definieren. Hier stellen sich mehrere Probleme: Erstens sind die Anwendenden beim Führen tendenziell mit den Händen am Endeffektor, müssten also für eine direkte Bewegung an einem vorgelagerten Segment umgreifen. Eine indirekte Kontrolle über den Endeffektor vermeidet zwar ein Umgreifen, ist aber schwierig durchzuführen und nachzuvollziehen, nicht zuletzt, da die aktuelle Ausrichtung der verschiedenen Gelenke einen starken Einfluss darauf hat, wie sich die Bewegung des Endeffektors auf den Rest des Roboters auswirkt. Das zweite Problem ist, dass für Segmente weiter vorne im Roboter nicht die volle Anzahl an Freiheitsgraden zur Verfügung steht, sodass manche Bewegungen ggfs. nicht möglich sind.

Eine Motivation, warum Gesten mit anderen Teilen des Roboters dennoch sinnvoll sein könnten, ist, bei redundanten Robotern die Pose des Endeffektors durch die Geste nicht zu ändern. Dies schränkt allerdings auch den Spielraum an möglichen Gesten stark ein (auf Nullraumbewegungen). Anbieten würden sich in diesem Fall vor allem solche Dinge wie ein einzelnes, angewinkeltes Gelenk zwischen den beiden benachbarten Segmenten in die eine oder andere Richtung zu drehen.

Die Robustheit der verschiedenen Ruckgesten lässt sich auf Kosten höherer Komplexität beispielsweise dadurch weiter erhöhen, mehrere davon in Sequenz auszuführen. Einfache Ruckbewegungen entstehen in der Führung oft automatisch durch energisches Bewegen des Roboters zur nächsten Zielpose, und sollten keine fehlerhaften Eingaben erzeugen.<sup>4</sup> Naheliegend sind etwa Ruckbewegungen direkt nacheinander in entgegengesetzte Richtungen. Praktisch birgt dies den Vorzug, dass der Roboter damit nach der Geste etwa dieselbe Pose einnimmt wie vor der Geste, und gleichzeitig ist diese Art von Folgebewegung koordinativ relativ einfach, im Vergleich zu beispielsweise einer zweiten Ruckbewegung in identischer Richtung, oder (bei Positionen) senkrecht dazu. Die Richtungsumkehr stellt andererseits praktisch eine zusätzliche Fehlerquelle dar, was bei der Detektion durch eine entsprechende Toleranz berücksichtigt werden muss.

Dieser Ansatz ließe sich offensichtlich auf höhere Anzahlen an Einzelbewegungen fortsetzen, etwa drei Ruckbewegungen in eine Richtung, die entgegengesetzte Richtung, und wieder die ursprüngliche Richtung. Der potenzielle Gewinn an Robustheit steht gesteigertem zeitlichen und mentalen Aufwand, sowie zusätzlichem Fehlerrisiko bei jeder Richtungsumkehrung gegenüber. Insbesondere pflanzen sich Ungenauigkeiten zwischen den Bewegungen fort: Bei der Verwendung der Endeffektorposition wird z.B. die Richtung des dritten Rucks meistens nur grob der Richtung des ersten Rucks entsprechen.

---

<sup>4</sup>In [104] wird ein ähnlicher Gedanke in Bezug auf mehrfaches statt einfaches Antippen geäußert.

### 4.2.3. Komplexere Gesten

In Richtung höherer Komplexität sind wie eingangs erwähnt kaum Grenzen gesetzt. Das beginnt bereits bei der Kombination von Ruckbewegungen. Wie im letzten Abschnitt erwähnt, stellen Sequenzen mehrerer entgegengesetzter Bewegungen eine einfache Kombinationsmöglichkeit dar. Es wäre aber auch ohne weiteres denkbar, andere Bedingungen an die einzelnen Teilbewegungen zu stellen, beispielsweise, dass die Richtung jeweils zu der der vorherigen Teilbewegung senkrecht sein muss. Auch die Kombination mehrerer Ruckbewegungen verschiedener Art bietet sich an, z.B. ein positionsbasierter Ruck gefolgt von einer Drehbewegung.

Von einzelnen ruckartigen Teilbewegungen kann auch abgewichen werden. Es ist etwa möglich, mit dem Endeffektor beliebige zu erkennende Bahnen zu führen. Das beginnt bei Kreisbewegungen oder Polygonen in einer bestimmten Ebene, erstreckt sich aber auch über Spiralen, Sterne oder dreidimensionale Verläufe. Eine allgemeine Bewertung ist dabei kaum möglich. Konkrete Beispiele müssen jeweils einzeln beurteilt werden, inwieweit sie sich zur Anwendung eignen.

Die Erkennung kann in gewissem Maße heuristisch passieren wie bei den obigen Ruckgesten. So ist einer der Vorschläge aus der Studie in Abschnitt 4.3 eine Kreisbewegung mit dem Endeffektor in einer horizontalen Ebene, was im realen System über eine durchgehende Bewegung mit Änderung der Richtung über vier Richtungsquadranten (und ausreichend kleine vertikale Änderung) umgesetzt wurde. Für ein größeres Gestenvokabular sind aber allgemeine Systeme sinnvoll, die sich nicht mehr mit bestimmten einzelnen Gesten beschäftigen.

Es existieren zwar wie weiter oben dargestellt kaum Vorarbeiten zur Verwendung haptischer Gesten des Roboters selbst. Allerdings kann auf die Literatur zur Gestenerkennung im Allgemeinen zurückgegriffen werden, deren grundlegende Prinzipien sich übertragen lassen sollten.

In der Erkennung dynamischer Gesten<sup>5</sup> bei Menschen teilt sich das Problem meist in mehrere Schritte:

- Erfassung der Daten
- Vorverarbeitung
- eigentliche Gestenerkennung

Bei der Erkennung von Gesten am Menschen findet die Datenerfassung häufig über Kameras und ähnliche Sensorik statt, von außen und über Distanz. Bei der Vorverarbeitung geht es daher darum, an Informationen über die tatsächliche Pose zu gelangen, beispielsweise an die Positionen von Gelenken. Dieser Vorverarbeitungsschritt fällt beim Roboter weg: Gelenkwinkel sind direkt verfügbar, und auch die Endeffektorpose über die Vorwärtskinematik. Andere Technologien wie z.B. die Verwendung von Sensorhandschuhen bei Handgesten erlauben auch die direkte Erfassung von Gelenkwinkeln. Eingabe in

<sup>5</sup>Damit sind Bewegungen gemeint, im Kontrast zu statischen Gesten, also z.B. bestimmten Haltungen. Für die vorliegende Arbeit wurden statische Gesten nicht in Betracht gezogen, da einzelne Posen des Roboters zur Verwendung in Aufgaben allgemein verfügbar bleiben sollen.

die eigentliche Gestenerkennung ist in jedem Fall üblicherweise eine Zeitreihe über die tatsächlich relevanten Daten, z.B. eben die Gelenkwinkel.

Für den Schritt der Gestenerkennung selbst kommen viele verschiedene Systeme zum Einsatz. Ein Ansatz ist der Vergleich mit Vorlagen der Gesten, auch als *Template Matching* bezeichnet. Dabei gibt es ein Abstandsmaß zwischen der Pose zu einem gegebenen Zeitpunkt und der Pose zu einem Punkt der Vorlage. Die Geste wird anhand der Vorlage klassifiziert, zu der der Gesamtabstand minimal ist. Da sich die Zeitdauer zwischen verschiedenen Ausführungen einer Geste meist unterscheidet, wird in der Regel der zeitliche Ablauf angepasst, z.B. über *Dynamic Time Warping*. Einige Beispielarbeiten sind [3, 14, 35, 40, 43, 166]

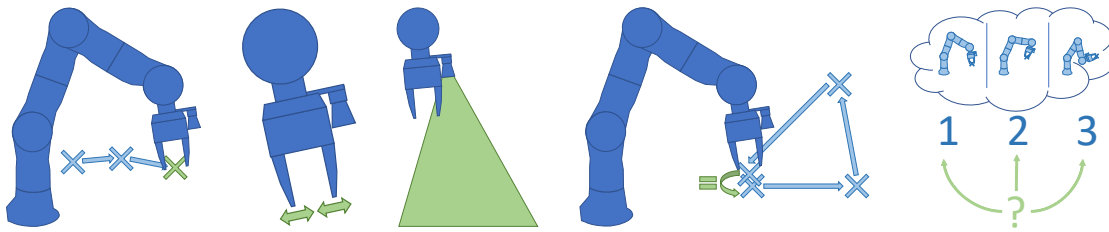
Eine weitere breit vertretene Technologie zur Gestenerkennung sind *Hidden Markov Models*. Für die einzelnen Gesten wird dabei jeweils ein Markov-Modell angenommen, d.h. diese bestehen aus einzelnen Zuständen (etwa Posentypen), zwischen denen es dann eine Matrix an Übergangswahrscheinlichkeiten gibt. Der versteckte Aspekt ist dabei, dass nicht die Zustände direkt erfasst werden, sondern nur nach einer weiteren Matrix emittierte Beobachtungen. Um die realen Daten auf diskrete Beobachtungen abzubilden, kommt beispielsweise *Vector Quantization* zum Einsatz, oder andere Clustering-basierte Methoden. Auch hier wird die Geste anhand des am besten passenden Modells klassifiziert, hier entsprechend das, bei welchem das Auftreten der Beobachtungssequenz am wahrscheinlichsten ist (ggfs. unter vorgegebenen a-priori-Wahrscheinlichkeiten). Eine Übersicht zu Ansätzen dieser Art findet sich in [99], und Beispiele etwa in [22, 36, 48, 79, 95, 107, 127, 134, 141, 166]

Darüber hinaus gibt es eine Reihe anderer Ansätze, die Anwendung finden, beispielsweise auf Basis von neuronalen Netzen [89, 107, 160, 163], endlichen Automaten oder deren Varianten [68, 85, 165] oder Entscheidungsbäumen [97]. In [98] ist eine Übersicht über das Feld gegeben, und [139] vergleicht verschiedene umgesetzte Verfahren am Praxisbeispiel.

Zusammenfassend bietet existierende Forschung zur Gestenerkennung bei Menschen also für haptische Eingaben an Robotern einen breiten Spielraum an möglichen Methoden auch für komplexere Gesten. Im Rahmen der vorliegenden Arbeit wurde die Umsetzung jedoch auf einige einfache Heuristiken beschränkt, was weiter unten genauer beschrieben wird.

### 4.3. Vorstudie und Implementierung

Die Entwicklung der haptischen Schnittstellen im vorliegenden Programmiersystem wurde in zwei Stufen aufgeteilt. Nach einer ersten Konzeptionsphase wurde eine Online-Umfrage durchgeführt, bei der Teilnehmende identifizierte Gestenvarianten in Kombination mit Beschreibungen der zugehörigen Operationen auf wahrgenommene Übereinstimmung bewerten sollten. Weiterhin wurden in der Studie auch Ideen unabhängig der gegebenen Vorschläge abgefragt. Der vollständige Inhalt der Umfrage ist in Anhang B reproduziert.



**Abbildung 4.4.:** In der Onlineumfrage verwendete Skizzen, von links nach rechts: Bewegung, Greiferoperation, Kameraaufnahme, Verbinden und Zahleneingabe

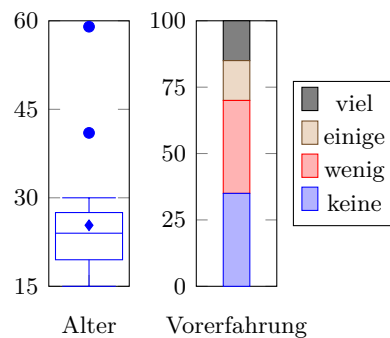
### 4.3.1. Studienentwurf

Die Onlineumfrage gliederte sich in drei Teile. Im ersten Teil („Beschreiben“) wurde für die Operationen des Systems, mit denen sich die Umfrage befasste, jeweils ein initialer Vorschlag der Teilnehmenden für eine haptische Interaktion erfragt. Dabei sollte auch immer eine kurze Assoziation für die jeweilige Bewegung als Begründung oder Merkhilfe angegeben werden. Im zweiten Teil („Bewerten“) wurden für dieselben Operationen jeweils einige im Vorfeld formulierte Vorschläge dargestellt, zu denen einzeln gefragt war, wie passend die Teilnehmenden diese fanden (in vier Stufen von „Perfekt“ bis „Gar nicht“). Neben haptischen Interaktionen wurden zum Vergleich auch einige Gesten vorgeschlagen, die Nutzende mit der Hand vor der Kamera ausführen könnten. Diese Art von Gesten wurde zwar für die vorliegende Arbeit im Vorfeld ausgeschlossen, wurde aber in der Studie im Hinblick auf eventuelle zukünftige Arbeiten mit aufgenommen. Im dritten Teil („Favoriten nennen“) sollten die Teilnehmenden dann schließlich jeweils die Interaktionen angeben, welche sie nach Bearbeitung der Studie am besten für eine Operationen fanden. Diese Trennung der Umfrage in drei Teile erfolgte aus der Motivation heraus, sowohl möglichst unvoreingenommene Ideen der Teilnehmenden zu ermitteln, als auch im zweiten Teil Interaktionen deutlich zu machen, die praktisch gut umsetzbar wären. Aufgrund des resultierenden Umfangs wurde die Umfrage auf fünf Operationen beschränkt: Eingabe einer Bewegung, Auslösen des Greifers, Auslösen der Kamera, Schließen von Schleifen (Verbinden) und Eingabe einer Zahl. Zur Kommunikation der Operationen wurden die in Abbildung 4.4 abgebildeten Grafiken in Kombination mit kurzen Beschreibungstexten (sh. Anhang B) verwendet.

Die Umfrage wurde online über eine existierende, kommerzielle Plattform durchgeführt<sup>6</sup>. Alle Fragen und Grafiken wurden parallel auf Deutsch und Englisch angelegt, um verschiedene Teilnehmerfelder abzudecken. Die Umfrage wurde über mehrere Kanäle verbreitet: vor Ort an der Universität, an interessierte Studierende aus verschiedenen Lehrveranstaltungen, sowie über ein englischsprachiges Forum zu diesem Zweck<sup>7</sup>.

<sup>6</sup><https://www.umfrageonline.com/>, besucht: 21.11.2022

<sup>7</sup><https://www.reddit.com/r/SampleSize/>, besucht: 21.11.2022



**Abbildung 4.5.:** Daten zu Teilnehmenden der Vorstudie. Links: Angegebenes Alter in Jahren. Rechts: Prozentuale Verteilung der angegebenen Vorerfahrung im Umgang mit Robotern auf der Skala von 0 („keine“) bis 3 („viel“).

### 4.3.2. Ergebnisse

Die Onlineumfrage wurde im Juli und August 2020 durchgeführt. Insgesamt wurden 26 Teilnahmen registriert. Davon wurde als Sprache in 11 Fällen Deutsch, in 15 Fällen Englisch gewählt. Die Teilnehmenden identifizierten sich in 14 Fällen als männlich (54%), in 10 Fällen als weiblich (38%), und in den verbleibenden 2 Fällen wurden keine Angaben gemacht. Der Altersdurchschnitt lag bei ca. 25.4 Jahren. Die Vorerfahrung im Umgang mit Robotern wurde in 9 Fällen als „keine“ angegeben (35%), ebenfalls in 9 Fällen als „wenig“ (35%), in 4 Fällen als „einige“ (15%), und wiederum in 4 Fällen mit „viel“, dem höchsten Wert der Skala (15%). Alter und Vorerfahrung sind auch in Abbildung 4.5 visualisiert.

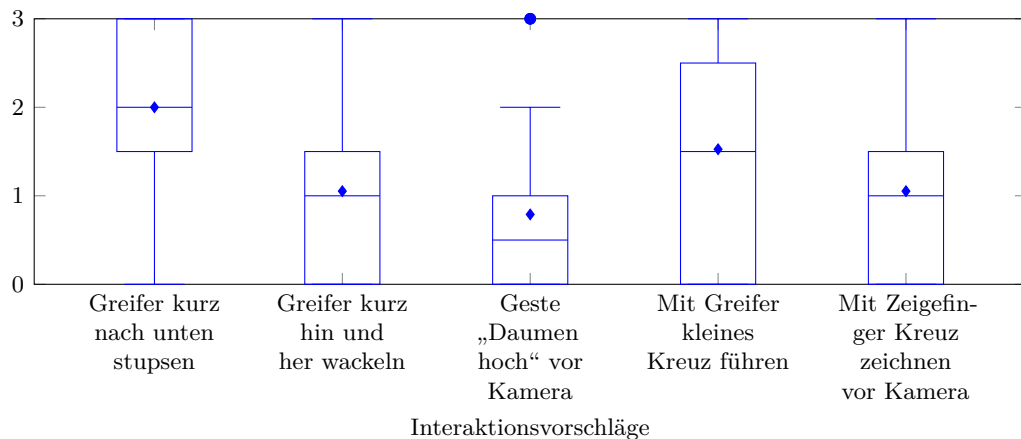
In 8 Fällen wurde die Teilnahme nicht vollständig abgeschlossen; in diesen Fällen wurden die Daten verwendet soweit vorhanden.

Im Folgenden werden die Ergebnisse anhand der einzelnen Operationen vorgestellt.

#### Ziel markieren

Für das Markieren von Zielposen wurde als häufigster Vorschlag in sieben Fällen genannt, den Endeffektor in einer kurzen Ruckbewegung nach unten zu bewegen. Die Details variierten dabei leicht, etwa darin, ob der Roboter anschließend mit einer Bewegung in die entgegengesetzte Richtung zurückgestellt und ob die Ruckbewegung durch Führung oder durch Antippen von oben ausgelöst werden sollte. Als Intuition dahinter wurde das Drücken eines Knopfes oder das Aufsetzen eines Stifts herangezogen.

Als weitere Vorschläge wurden genannt: Drehbewegungen im letzten Gelenk, Nicken im vorletzten Gelenk oder Nicken im Ellenbogen. Einige andere Antworten, die mit dem verfolgten Ansatz nicht vereinbar waren, wurden aus den Ergebnissen entfernt. Darunter fielen z.B. Stillstand in der Führung (was häufig durch die Handführung selbst passiert und so keine zuverlässige Entscheidung erlaubt) oder die Extraktion von End- und Wendepunkten aus der Bewegung selbst (was eine grundlegend andere Herangehensweise als explizite Programmiereingaben bedeutet).



**Abbildung 4.6.:** Bewertungen der Interaktionsvorschläge für die Operation zum Markieren einer Zielpose, von 0 („passt gar nicht“) bis 3 („passt perfekt“).

Im zweiten Teil der Studie, bei der Bewertung der gegebenen Vorschläge, erzielte ebenfalls eine Ruckgeste nach unten das beste Ergebnis. Mit numerischen Werten von 0 für „passt gar nicht“ bis 3 für „passt perfekt“ erreichte diese einen Mittelwert  $\bar{x} = 2$  bei Standardabweichung  $\sigma = 0.9$ . Der Vorschlag mit der zweitbesten Bewertung war, ein kleines Kreuz mit dem Endeffektor zu führen ( $\bar{x} = 1.5$ ,  $\sigma = 1.1$ ). Die übrigen Vorschläge folgten deutlich abgeschlagen (Ruckbewegung horizontal mit  $\bar{x} = 1.1$ ,  $\sigma = 0.9$ ; Daumen hoch vor der Kamera mit  $\bar{x} = 0.8$ ,  $\sigma = 1.0$ ; Kreuz zeichnen mit dem Zeigefinger vor der Kamera mit  $\bar{x} = 1.1$ ,  $\sigma = 1.1$ ). Die Ergebnisse sind in Abbildung 4.6 als Diagramm abgebildet.

Im dritten Teil der Studie, bei der Nennung der insgesamt favorisierten Bewegung, wiederholten sich die Ergebnisse aus dem ersten Teil, mit der Ruckbewegung nach unten als häufigste Antwort mit sieben Nennungen.

### Greifer operieren

Mit neun Nennungen war der häufigste Vorschlag zum Auslösen des Greifers, die Greiferbacken direkt auf- und zuzuziehen. Weitere Varianten waren eine Drehbewegung um das Handgelenk (d.h. letzte Gelenk) des Roboters oder eine Ruckbewegung senkrecht zur Greifrichtung (also senkrecht zur A-Achse des NSA-Koordinatensystems). Auch hier wurden wieder einige Interaktionen vorgeschlagen, die nicht direkt zum vorliegenden Ansatz passen, so etwa eine Geste vor der Kamera (die Hand vor der Kamera zur Faust schließen und wieder öffnen) oder eine taktile Eingabe am Roboter (eine Stelle am Arm drücken, um den Greifer zu schließen, und zum öffnen wieder loslassen). Weiterhin wurde entgegen der Fragestellung in einigen Fällen eine Taste in Hardware genannt.

Die vorgegebenen Vorschläge im zweiten Teil der Umfrage waren teilweise einzelne Interaktionen zum Öffnen und Schließen, aber auch eine Interaktion für den Wechsel zum jeweils anderen Zustand (aus dem englischen entlehnt als Toggle bezeichnet). Die besten

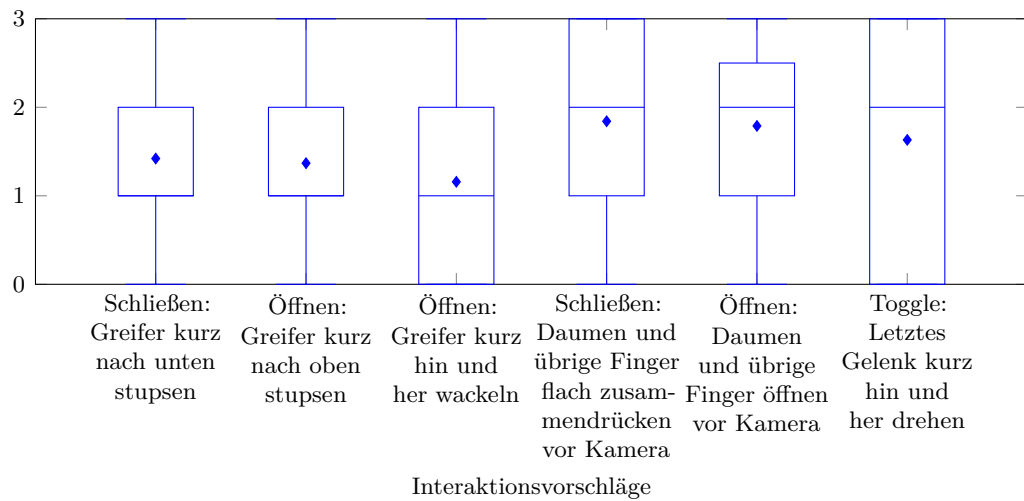


Abbildung 4.7.: Bewertungen der Interaktionsvorschläge zum Bedienen des Greifers.

Bewertungen erzielten dabei das Schließen durch Zusammenführen von Daumen und Zeigefinger vor der Kamera ( $\varnothing = 1.8$ ,  $\sigma = 1.0$ ), das Öffnen durch Spreizen der Hand vor der Kamera ( $\varnothing = 1.8$ ,  $\sigma = 1.0$ ) und der Toggle durch eine Drehbewegung um das letzte Gelenk des Roboters ( $\varnothing = 1.6$ ,  $\sigma = 1.2$ ). Die übrigen, schlechter bewerteten Vorschläge waren Schließen durch einen Ruck nach unten ( $\varnothing = 1.4$ ,  $\sigma = 0.9$ ), Öffnen durch einen Ruck nach oben ( $\varnothing = 1.4$ ,  $\sigma = 0.9$ ) und Öffnen durch horizontale Ruckbewegungen ( $\varnothing = 1.4$ ,  $\sigma = 0.9$ ). Die Ergebnisse der Bewertungen sind in Abbildung 4.7 dargestellt.

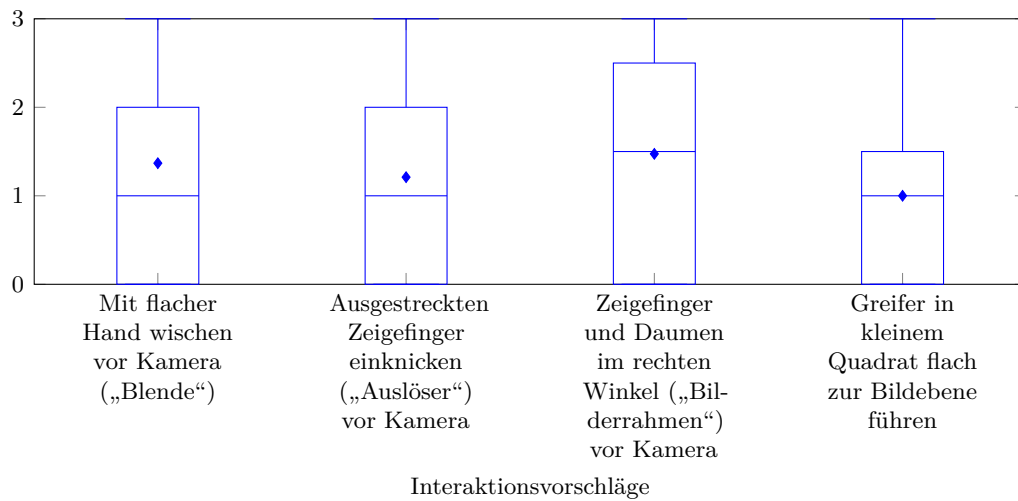
Im dritten Teil zeigte sich, dass unter Miteinbeziehung von Gesten vor der Kamera auch insgesamt am häufigsten eine solche Greifgeste bevorzugt wurde (Hand zur Faust schließen und wieder öffnen, oder Daumen und Zeigefinger), mit insgesamt elf Nennungen. Mit vier Nennungen folgte hier eine Drehbewegungen um das letzte Gelenk des Roboters als beste haptische Interaktion, sowie direkte Manipulation der Greiferbacken mit zwei Nennungen.

### Bild aufnehmen

Zum Auslösen der Kamera, beispielsweise beim Anlegen von Verzweigungen, wurde mit fünf Nennungen am häufigsten eine physische Taste zum Auslösen vorgeschlagen. Es folgte eine Ruckbewegung in Blickrichtung der Kamera mit vier Nennungen, sowie eine Nickbewegung im vorletzten Gelenk des Roboters, ein zweifacher Ruck nach unten, oder ein kurzes Klatschen vor der Kamera. Ein weiterer mehrfach genannter Vorschlag war ein automatisches Auslösen, was für eine reine Objekterkennung möglich wäre, aber bei der Verwendung in Verzweigungen nicht dem Systementwurf entspricht.

Im Bewertungsteil wurde die beste Wertung mit einer Geste aus abgespreiztem Daumen und Zeigefinger vor der Kamera erzielt (in Andeutung eines Bilderrahmens), mit  $\varnothing = 1.5$ ,  $\sigma = 1.2$ . An zweiter Stelle folgte eine Wischgeste vor der Kamera (um eine Blende zu symbolisieren) mit  $\varnothing = 1.4$ ,  $\sigma = 1.0$ . Die weiteren Vorschläge waren eine Knickgeste wie





**Abbildung 4.8.:** Bewertungen der Interaktionsvorschläge zum Auslösen der Kamera.

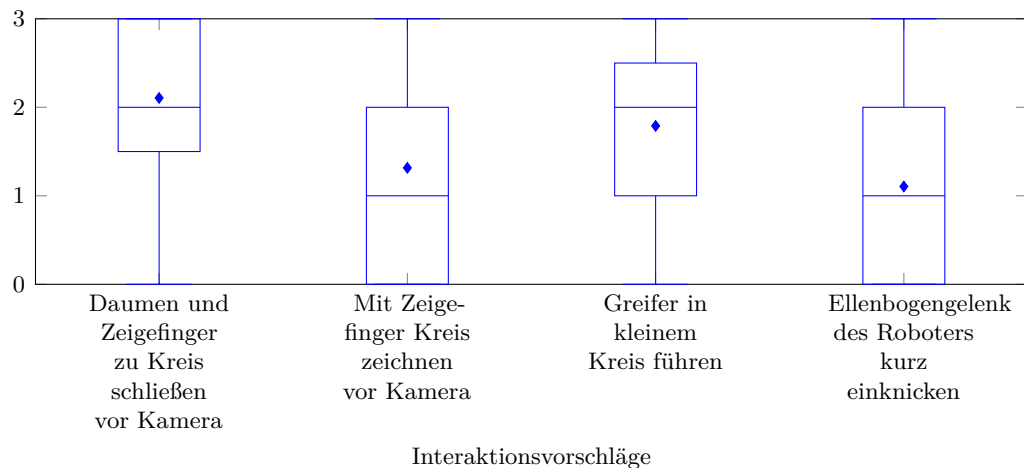
mit einem Auslöser mit dem Zeigefinger vor der Kamera ( $\varnothing = 1.2$ ,  $\sigma = 1.0$ ), und den Greifer parallel zur Bildebene in einem Rechteck zu führen, wiederum in Assoziation mit einem Bilderrahmen ( $\varnothing = 1.0$ ,  $\sigma = 1.0$ ). Die Daten sind in Abbildung 4.8 wieder visualisiert.

Die Geste mit abgespreiztem Zeigefinger und Daumen wurde im dritten Teil sechsmal als Favorit genannt, der höchste Wert für diese Operation. Weitere häufige Nennungen waren die Knipsgeste (den Finger vor der Kamera abknicken) dreimal, die Wischgeste dreimal, und zweimal ein Antippen oder Ruck in Blickrichtung der Kamera.

### Verbinden

Die Operation, zu einem existierenden Zustand zurückzukehren um beispielsweise eine Schleife zu schließen, wurde am häufigsten mit einer Kreisgeste in der Führung assoziiert, mit fünf Nennungen. Ansonsten war die einzige mehrfache Nennung ein automatisches Schließen der Schleife, wenn in die Nähe der ursprünglichen Position zurückgekehrt wird. Einer der Vorschläge warf dabei die Idee auf, den Roboter in der Nähe bereits existierender Zustände steifer zu schalten bzw. zu der genauen Pose driften zu lassen, wo dann eine Ruckbewegung ein Verbinden auslösen würde. Weitere Vorschläge waren etwa ein dreifacher Ruck nach unten, ein Ruck nach oben, oder wiederum eine physische Taste. Unter den vorgegebenen Vorschlägen war die bestbewertete Geste, Daumen und Zeigefinger vor der Kamera zu einem Kreis zu schließen ( $\varnothing = 2.1$ ,  $\sigma = 0.9$ ). Auch eine Kreisgeste in der Führung wurde noch in etwa gut bewertet ( $\varnothing = 1.8$ ,  $\sigma = 1.0$ ). Die beiden übrigen Vorschläge erzielten schlechtere Bewertungen, konkret den Zeigefinger vor der Kamera in einem Kreis zu bewegen ( $\varnothing = 1.3$ ,  $\sigma = 1.0$ ) und eine Ruckbewegung im Ellenbogen des Roboters ( $\varnothing = 1.1$ ,  $\sigma = 1.0$ ). Diagramme für die Bewertungen sind in Abbildung 4.9 gegeben.

Als Gesamtfavorit wurde fünfmal die Kreisgeste mit Daumen und Zeigefinger vor der



**Abbildung 4.9.:** Bewertungen der Interaktionsvorschläge zum Schließen von Schleifen (Verbinden).

Kamera genannt, viermal eine Kreisbewegung (z.B. mit dem Zeigefinger). Ebenfalls vier Nennungen erwähnten eine Kreisbewegung in der Führung.

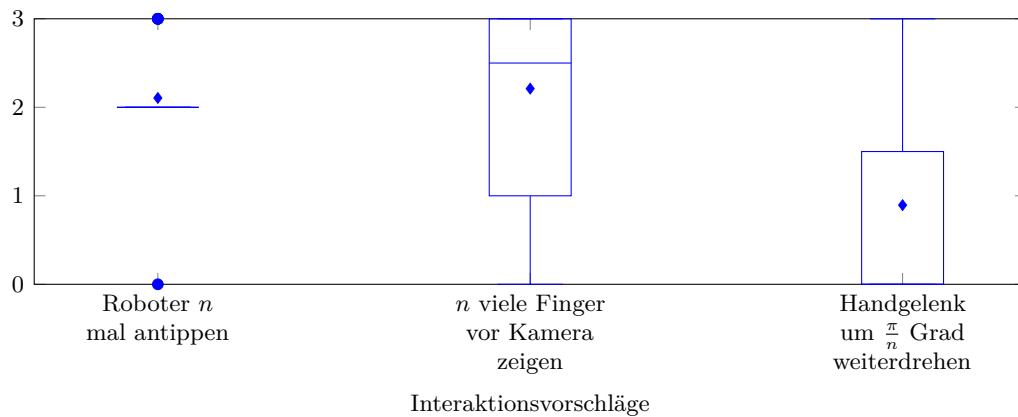
### Zahl eingeben

Die Eingabe einer Zahl wurde z.B. für die Auswahl von Zweigen in Verzweigungszuständen in Erwägung gezogen. Als Initialvorschlag kam hier von den Teilnehmenden dreimal, den Roboter entsprechend oft anzutippen bzw. nach unten zu rucken, ggfs. nach einer vorhergehenden Auslöseinteraktion. Ebenfalls dreimal wurde genannt, die Zahl selbst mit dem Greifer zu schreiben, sowie einmal, entsprechend oft vor der Kamera mit dem Finger einen Strich zu ziehen. Ansonsten variierten die Vorschläge stark: z.B. wurde jeweils einmal genannt, den Roboter nach rechts oder links zu rucken für ein Inkrement bzw. Dekrement, mit anschließender Bestätigung durch einen Ruck nach vorne. Ein weiterer Vorschlag war, den Arm des Roboters in der Art eines Hebels zu benutzen, für ein gedachtes Zählrad wie etwa in einem NumberPicker-Widget unter Android<sup>8</sup>.

In der Bewertung war die beste Geste, die entsprechende Anzahl an Fingern in die Kamera zu halten ( $\varnothing = 2.2$ ,  $\sigma = 1.0$ ). Ein mehrfaches Antippen oder Rucken des Roboters wurde ähnlich gut bewertet ( $\varnothing = 2.1$ ,  $\sigma = 0.6$ ). Eine schlechtere Bewertung erhielt dagegen der Vorschlag, die Zahl mit dem letzten Gelenk des Roboters wie mit einem Drehregler einzustellen ( $\varnothing = 0.9$ ,  $\sigma = 1.1$ ). Die entsprechenden Diagramme liegen in Abbildung 4.10 vor.

Auch in der Nennung der abschließend bevorzugten Interaktion für diese Operation war das Zeigen mit den Fingern vor der Kamera mit zehn Nennungen mit Abstand am häufigsten. Es folgte mit viermaliger Nennung, den Roboter entsprechend der Anzahl oft anzutippen.

<sup>8</sup><https://developer.android.com/reference/android/widget/NumberPicker>, besucht: 21.11.2022



**Abbildung 4.10.:** Bewertungen der Interaktionsvorschläge zur Eingabe einer Variablennummer, d.h. einer Zahl  $n$ .

### Statistische Auswertung

Bei den relativ kleinen Stichprobengrößen sind die beobachteten Effekte in vielen Fällen nicht stark genug für statistische Signifikanz. Auf die Bewertungen der verschiedenen Gestenvorschläge einer einzelnen Operationen lassen sich als nichtparametrisches Verfahren für abhängige Stichproben jeweils Friedman-Tests durchführen. Die in R ermittelten Werte sind in Anhang A in den Tabellen A.7 bis A.10 dargestellt, und werden hier in einem kurzen Überblick beschrieben.

Innerhalb der Vorschläge für Operieren des Greifers und Aufnehmen eines Bildes liegt keine statistische Signifikanz mit  $\alpha = 0.05$  vor. Beim Markieren eines Ziels ist nur der Unterschied zwischen dem besten und schlechtesten Ergebnis statistisch signifikant (Ruckbewegung des Greifers im Vergleich mit Geste „Daumen hoch“ vor der Kamera), genau wie innerhalb der Vorschläge zum Verbinden (Kreis aus Daumen und Zeigefinger vor der Kamera verglichen mit Einknicken des Ellenbogengelenks). Für die Zahleneingabe sind schließlich die Unterschiede zwischen dem letzten und jedem der beiden ersten Vorschläge statistisch signifikant.

Trotz dieser Einschränkungen in der Verallgemeinerbarkeit der erzielten Resultate werden diese für den folgenden ersten Entwurf eines haptisch basierten Systems herangezogen, in Kombination mit weiteren Auswahlkriterien.

### 4.3.3. Schlussfolgerungen

Aus den Resultaten dieser Vorabumfrage wurde eine Zuordnung der haptischen Gesten ermittelt, die für die später folgende eigentliche Evaluation implementiert wurde. Hier ist anzumerken, dass die Implementierung der haptischen Gesten zunächst für allgemeine ERSA erfolgte, aber später für das Multi-ERSA-Modell nochmals erweitert und auch erst in diesem Zusammenhang explizit ausgewertet wurde.

Für den Entwurf der haptischen Interaktionspalette sind mehrere relevante Aspekte

hervorzuheben. Offensichtlich ist ein Ziel, die einzelnen Operationen auf Interaktionen abzubilden, die für Nutzende intuitiv nachvollziehbar sind. Dabei ist die Annahme, dass die theoretischen Bewertungen aus der Vorstudie auf die reale Anwendung übertragbar sind. Dies ist nicht zwingend tatsächlich der Fall, etwa aufgrund von Verständnisproblemen bei den textuellen Beschreibungen, oder weil Gesten in der realen Ausführung koordinativ komplizierter sein können als man sich beim Lesen vorstellt.<sup>9</sup> Nichtsdestotrotz geht diese Arbeit für die Auswahl in einer ersten Implementierung zumindest von einer groben Korrespondenz der Bewertung aus. Als Ausblick werden ausführlichere Auswertungen von möglichen haptischen Interaktionen in Kapitel 8 diskutiert.

Ein weiterer Gesichtspunkt für das Gesamtsystem muss sein, dass die einzelnen Interaktionen sauber voneinander trennbar sind. Von vornherein ist klar, dass nicht mehrere verschiedene Operationen dieselbe Geste verwenden können, auch wenn diese bei beiden Operationen einzeln als am passendsten bewertet wäre. Auch Gesten, die sich unter bestimmten Umständen überschneiden, sind problematisch, wie der bereits zuvor erwähnte Fall einer Ruckbewegung nach unten in Weltkoordinaten und einer Ruckbewegung in Richtung der A-Achse des NSA-Koordinatensystems, die bei entsprechender Orientierung des Endeffektors zusammenfallen können und dann für das System nicht mehr zu unterscheiden sind.

Ein praktisches Kriterium für die in dieser Arbeit getroffene Auswahl waren auch die Einschätzungen von Implementierungsaufwand und Robustheit bzw. das Verhältnis zwischen beidem. Wie weiter oben diskutiert, bieten unter diesem Aspekt einfache Ruckgesten in verschiedenen Varianten naheliegende Heuristiken, während komplexere Gesten zwar schwer als Gesamtheit zu beurteilen sind, deren Heuristiken oder allgemeinere Verfahren aber generell mehr Aufwand darstellen.

Wie bereits angeführt wurden Handgesten vor der Kamera zwar in Teilen von der Vorstudie abgedeckt, für die Umsetzung in der vorliegenden Arbeit aber ausgeschlossen. Weiterhin war zwar das Ziel, einen möglichst großen Teil der Interaktionen haptisch zu lösen, aber ein am Greifer montierter Tastenblock stand als Ausweichoption zur Verfügung. Entsprechend war zusätzlich zur Auswahl an passenden Interaktionen pro Operation die Entscheidung zu treffen, ob einzelne Operationen stattdessen auf Tasten abgebildet würden. Dadurch konnte auch die quantitative Bewertung der Operationen aus der Vorstudie in Betracht gezogen werden (nicht nur die qualitative Rangordnung), sowie Überlegungen, welche Funktionen von haptischen Eingaben unabhängig sind und welche nicht. So ist für Posen im Freiraum im Rahmen grober Bewegungen eine haptische Interaktion zum Auslösen weitgehend unproblematisch. Bei einer Aufgreif- oder Kamerapose dagegen ist es wichtiger, dass Nutzende diese genau einstellen können, was durch eine haptisch ausgelöste Operation erschwert wird (oder wobei eine vorangegangene genaue Einstellung zunichte gemacht wird).

In Kombination ist die Abbildung von Operationen auf Interaktionen für das ERSA-System in der vorliegenden Arbeit:

- **Greifer und Kamera auslösen: Tasten.** Beide Operationen profitieren in ihrer

---

<sup>9</sup>Ähnliche Bedenken werden in [79] festgehalten.

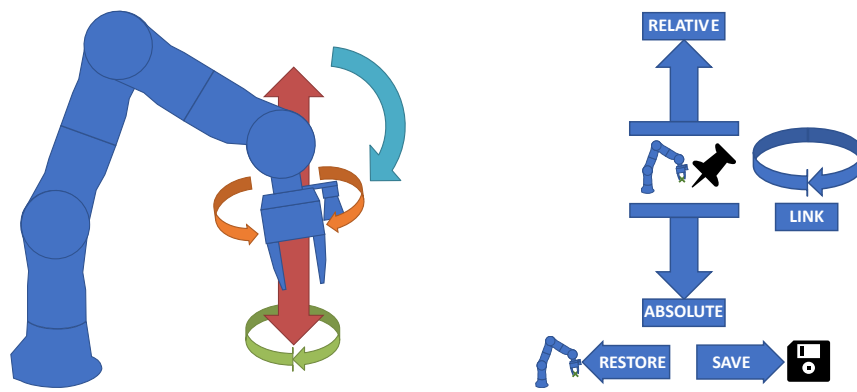
Genauigkeit vom Verzicht auf eine haptische Geste.

- **Ziel markieren: Ruckbewegung nach unten** in Weltkoordinaten. Eine maximal einfache Geste, was für die vermutlich häufigste Operation Sinn ergibt, und Favorit aus der Vorstudie.
- **Verbinden: Kreisbewegung** (horizontal in Weltkoordinaten). Bestbewertete haptische Geste für die Operation.
- **Zahl eingeben: Nicht verwendet.** Letztendlich wurden Verbindungsziele und Zweige anderweitig identifiziert als über abstrakte Zahlen.

Zusätzlich wurden ausgehend von den Ergebnissen der Vorstudie einige weitere Operationen des Systems, die später hinzugefügt wurden oder für die Onlineumfrage nicht einfach genug zu kommunizieren waren, auf haptische Gesten abgebildet:

- **Ziel für relative Bewegung markieren: Ruckbewegung nach oben** in Weltkoordinaten. Aufgrund der Korrespondenz zum Markieren von absoluten Zielen gewählt.
- **Posenvariable speichern: Ruckbewegung im Handgelenk in positive Drehrichtung** (mit Greifer nach unten: Ruck nach rechts). Einfache und erprobte Geste, die anderweitig z.B. für den Greifer verwendet wurde.
- **Posenvariable laden** (d.h. wieder anfahren): **Ruckbewegung im Handgelenk in negative Drehrichtung** (mit Greifer nach unten: Ruck nach links). Aufgrund der Korrespondenz zum Speichern gewählt.

In Abbildung 4.11 sind die gewählten haptischen Interaktionen visualisiert, sowie die als Gedächtnisstütze verwendeten Skizzen, die in den Experimenten am Roboter angebracht wurden.



**Abbildung 4.11.:** Links: Konzeptgrafik der im ERSA-System verwendeten haptischen Interaktionen. Pinngesten für Bewegungen rot, horizontale Kreisgeste zum Verbinden grün, Drehung im letzten Gelenk zum Speichern/Laden orange. (Eine optionale Nickgeste im vorletzten Gelenk zum Auslösen der Kamera, hier türkis, wurde probenhalber implementiert, aber nicht in den Experimenten verwendet.) Rechts: Merkhilfe für die Benutzerstudie. Diese wurde auf den Roboter aufgeklebt (der obere Teil zu Bewegungen und Verbinden auf das letzte Gelenk, der Teil zum Speichern und Laden von Posenvariablen auf den Flansch).

---

5.1. Stand der Forschung . . . . .	68
5.2. Markierungsalgorithmus . . . . .	69
5.2.1. Markierungsfunktionen . . . . .	69
5.2.2. Algorithmus . . . . .	70
5.3. Unifikation . . . . .	71
5.3.1. Algorithmus . . . . .	72
5.3.2. Anwendung im Programmiersystem . . . . .	73
5.4. Evaluation . . . . .	74
5.4.1. Experimentelle Verifikation . . . . .	74
5.4.2. Komplexität . . . . .	75

---

In diesem Kapitel wird ein Verfahren präsentiert, um die explizite Programmierung von Schleifen seitens der Nutzenden durch eine automatisierte Generierung zu ersetzen, hier auch als *Struktursynthese* bezeichnet. Dies motiviert sich durch Rückmeldungen aus der ersten durchgeführten Benutzerstudie mit dem ERSA-System (genaue Beschreibung und Ergebnisse dazu finden sich in Kapitel 7). Es wurde dort als freies Feedback häufiger angemerkt, dass es als mental anstrengend empfunden wurde, sich über die Aufgabenstruktur Gedanken zu machen, sowie an den Enden von Schleifenrumpfen diese Information präsent zu haben und die explizite Verknüpfung anzulegen. Der Grundgedanke ist nun, auf die explizite Programmierung der Struktur zu verzichten, und stattdessen auch für wiederholte Aktionsfolgen in Schleifen einfach immer schrittweise weiterzuprogrammieren. Das System muss dann selbstständig feststellen, an welchem Punkt die Anweisungen sich zu überlappen beginnen, und daraus die zugehörigen Vereinfachungen vornehmen.

Im weiteren Verlauf des Kapitels werden zunächst verwandte Arbeiten zu diesem Ansatz eingeordnet. Es folgt die Beschreibung des zugrunde liegenden Markierungsalgorithmus, der für Zustandspaare bestimmt, ob diese denselben logischen Zustand repräsentieren könnten oder nicht. Anschließend wird die eigentliche Vereinigungsmethode angegeben. Zuletzt wird kurz die Evaluation vorgestellt, die mit dem Ansatz durchgeführt wurde,

einerseits eine Verifikation am Beispiel, andererseits Überlegungen zur resultierenden Komplexität.

Die Inhalte in diesem Kapitel wurden teilweise in ihrer ursprünglichen Form in [180] veröffentlicht.

### 5.1. Stand der Forschung

Das formale Modell erweiterter Roboterzustandsautomaten, das in dieser Arbeit verwendet wird, wurde für diese entwickelt und findet außerhalb davon bisher keine Anwendung. Insofern stellt sich auch für die Struktursynthese vielmehr die Frage, welche existierenden Arbeiten auf Basis anderer Formalismen und Anwendungsszenarien, die Struktur automatisiert erzeugen, dafür adaptiert werden können, und inwieweit bzw. in welchen Aspekten.

Eine Alternative, die zur Struktursynthese gelegentlich benutzt wird, sind sogenannte *Version Spaces* [86, 87, 118]. Diese sind im Bereich des *Concept Learning* angesiedelt, wo es darum geht, ein zu lernendes Konzept aus einem größeren Raum an möglichen Konzepten einzugrenzen. Version Spaces finden z.B. darin Anwendung, aus Demonstrationen Makros für Textverarbeitungsprogramme zu generieren. Damit ist die Anwendung weiter von der Programmierung von Leichtbaurobotern entfernt. Der Ansatz, aus einzelnen ausgeführten Aktionsketten die gemeinsamen Komponenten zu verallgemeinern, ist jedoch derselbe. Auch werden insbesondere Schleifen daraus erstellt, dass in einem einzelnen Ablauf der Schleifenrumpf mehrmals hintereinander ausgeführt wird, ein Gedanke, den das entwickelte Verfahren ebenfalls aufgreift.

Auch *Planungssysteme* bieten Möglichkeiten zum automatisierten Erstellen von Programmstruktur [71, 90]. Dabei liegen zumeist eher aufgabenzentrierte Beschreibungen der Probleme zugrunde, zu denen die Planungsverfahren Lösungen in Form von (ggfs. komplexer strukturierten) Plänen erzeugen. Da jedoch das vorliegende System auf explizite Programmierung statt automatisierte Planung abzielt, lassen sich diese Verfahren kaum auf den ERSA-Anwendungsfall übertragen.

Verfahren aus dem Bereich *Programmieren durch Vormachen* synthetisieren ebenfalls teilweise strukturierte Programme [49, 54, 114]. Dabei werden jedoch in der Regel mehrere vollständige Demonstrationen verwendet. Im Gegensatz dazu ist das Ziel der folgenden Methode, bereits aus einer teilweisen Überlappung programmierter Abläufe die entsprechenden Wiederholungen festzustellen und im Automaten zusammenzuführen.

Zu guter Letzt existieren auch Ansätze, die auf Basis einer Demonstration ein Grundprogramm anlegen, dessen Struktur dann z.B. über visuelle Programmierung verändert und erweitert werden kann [5, 6]. Die Struktur im Grundprogramm selbst ist dabei jedoch nur sequenziell, und die explizite weitere Programmierung fällt nicht in den Fokus dieses Kapitels.



## 5.2. Markierungsalgorithmus

Um zu bestimmen, welche Zustände in einem existierenden ERSA *unifizierbar* (vereinigbar) sind, wird hier eine Markierungsfunktion benutzt. Diese verfolgt denselben Gedanken wie bei der Minimierung endlicher Automaten (sh. etwa [69, 164]). Informell ausgedrückt werden Zustandspaare als *widersprüchlich* markiert, wenn aus ihren direkten Eigenschaften oder aus Wissen über darauf folgende Zustandspaare hervorgeht, dass diese nicht vereinigbar sind.

### 5.2.1. Markierungsfunktionen

Die erste Möglichkeit eines Widerspruchs besteht zwischen zwei Transitionen, die unter derselben Eingabe aus den zwei Zuständen eines Paares ausgehen. Wenn die Updateschritte entlang zweier solcher Transitionen sich qualitativ (unterschiedlicher Aktionstyp, z.B. Bewegung im einen Fall und Speichern einer Posenvariablen im anderen Fall) oder quantitativ (in den Parametern der Aktion, z.B. über einen Schwellwert voneinander abweichende Zielposen) unterscheiden, dann können die Zustände offenbar nicht unifiziert werden, da sich danach in mindestens einem der Fälle die Semantik des Updateschritts verändern würde. Formal wird dies ausgedrückt in einer Funktion

$$M_{trans} : Q \times Q \times \Sigma_\varepsilon \rightarrow \{-1, 0\},$$

$$M_{trans}(q, q', \sigma) = \begin{cases} -1 & \text{falls } u(q, \sigma) \text{ und } u(q', \sigma) \text{ aus unterschiedlichen Fällen} \\ & \text{in (3.1) sind, oder } j \text{ oder } \bar{p} \text{ sich darin unterscheiden,} \\ 0 & \text{sonst} \end{cases}$$

Zur einfacheren Übersicht sei hier nochmal die referenzierte Definition von *com* angeben:

$$com = \left\{ \begin{array}{l} move\_abs[\bar{p}](out : p_{cur}), \\ move\_rel[\bar{p}](inout : p_{cur}), \\ move\_obj[\bar{p}](in : p_{obj}, out : p_{cur}), \\ save\_var[j](in : p_{cur}, inout : d), \\ load\_var[j](in : d, out : p_{cur}) \end{array} \right\}, 1 \leq j \leq n, \bar{p} \in P \quad (3.1 \text{ reproduziert})$$

Diese *Markierungsfunktion auf Transitionen*  $M_{trans}$  wird anschließend in der Definition der *Markierungsfunktion auf Zustandspaaren*  $M$  verwendet.

Zu Zwecken der Übersicht wird die Zustandsmenge wie auch in Kapitel 3 schon als  $Q = Q_b \uplus Q_v \uplus Q_t$  in drei disjunkte Teilmengen partitioniert, wobei:

- $Q_b = \{q \in Q \mid \exists \sigma \in \Sigma : \delta(q, \sigma) \text{ definiert}\}$  sind die *Verzweigungszustände*
- $Q_v = \{q \in Q \mid \delta(q, \varepsilon) \text{ definiert}\}$  sind die *Transitzustände*
- $Q_t = Q \setminus \{Q_b \cup Q_v\}$  sind die *Terminalzustände*

Es ist zu bemerken, dass Zustandspaare aus einem Verzweigungszustand und einem Transitzustand automatisch widersprüchlich sind: bei einer Vereinigung wäre unklar, ob das System die spontane Transition ausführen soll (automatisch, und ohne eine Eingabe aufzunehmen), oder ob tatsächlich eine Eingabe  $\sigma \in \Sigma$  erfasst werden soll für die Transitionen aus dem Verzweigungszustand. Ein vereinigter Zustand mit beiden Arten von ausgehenden Transitionen würde Nichtdeterminismus erzeugen, und stünde im Widerspruch zu obiger Aussage, dass die Mengen  $Q_b$  und  $Q_v$  disjunkt sind.

Wenn bei einem Zustandspaar keiner der Fälle für Widerspruch vorliegt, wird dieses vom System als Indiz für eine mögliche Vereinigung angesehen. Jedes weitere Paar in zwei parallelen Ketten an paarweise widerspruchsfreien Zuständen erhöht daher den Wert der Markierungsfunktion der Vorgängerpaare um eins.

Damit wird die Markierungsfunktion auf Zustandspaaren wie folgt definiert:

$$M : Q \times Q \rightarrow \mathbb{N} \cup \{-1\},$$

$$M(q, q') = \begin{cases} -1 & \text{falls } (q \in Q_b \wedge q' \in Q_v) \vee (q \in Q_v \wedge q' \in Q_b) \\ & \vee (\exists \sigma \in \Sigma_\varepsilon : M(\delta(q, \sigma), \delta(q', \sigma)) = -1) \\ & \vee (\exists \sigma \in \Sigma_\varepsilon : M_{trans}(q, q', \sigma) = -1), \\ \min(\max_{\sigma \in \Sigma_\varepsilon} \{M(\delta(q, \sigma), \delta(q', \sigma)) + 1\}, |Q|) & \text{sonst} \end{cases}$$

Zusammengefasst: zwei Zustände  $q \in Q$  und  $q' \in Q$  werden per  $M(q, q') = -1$  als widersprüchlich markiert gdw. sie widersprüchlichen Typs sind (ein Verzweigungszustand und ein Transitzustand), sie ein widersprüchliches Nachfolgerzustandspaar unter einer Eingabe  $\sigma \in \Sigma_\varepsilon$  haben, oder sie ausgehende Transitionen unter einer Eingabe  $\sigma \in \Sigma_\varepsilon$  haben, die widersprüchlich sind. Ansonsten ist der Wert der Markierungsfunktion gleich der größten Markierung eines Nachfolgerzustandspaares (unter irgendeiner Eingabe  $\sigma \in \Sigma_\varepsilon$ ) plus eins, mit einem minimalen Wert von 0. Solche mit  $M(q, q') = n \in \mathbb{N}$  als nicht widersprüchlich markierten Zustandspaare werden stattdessen als *bestätigend* (zu einer Tiefe von  $n$ ) bezeichnet.<sup>1</sup>

### 5.2.2. Algorithmus

Der Markierungsalgorithmus selbst, mit dem sich die Funktion  $M$  gemäß obiger Definition berechnen lässt, ist in Algorithmus 1 in Pseudocode dargestellt. Dabei ist anzumerken, dass  $M_{trans}$  im Gegensatz zu  $M$  direkt und ohne iterative Berechnung aus der Automatendefinition hergeleitet werden kann.

Die Formulierung der beiden Markierungsfunktionen  $M$  und  $M_{trans}$  über  $Q \times Q$  ist etwas übersichtlicher, relevant sind aber eigentlich die Werte für  $\{\{q, q'\} \mid q \in Q, q' \in Q, q \neq q'\}$ : Aus den Definitionen fällt heraus, dass beide Funktionen symmetrisch (in den Zuständen) sind, d.h. es gilt  $M(q, q') = M(q', q) \forall q, q' \in Q$  (und für  $M_{trans}$  analog). Sie sind außerdem zwar auf  $q = q'$  definiert, aber diese Werte werden an keiner Stelle benutzt. Daher

<sup>1</sup>Der Maximalwert von  $|Q|$  dient dazu, dass der später vorgestellte Algorithmus zur Berechnung von  $M$  auch bei Zyklen sich bestätigender Zustände terminiert und die Bestätigungstiefe nicht unbegrenzt steigt.

---

**Algorithmus 1** Markierungsalgorithmus zur Berechnung der Funktionswerte von  $M$ .

---

**Daten:** Tabelle  $M$  über  $Q \times Q$

```

1: if  $M$  wurde neu generiert then
2:   for  $(q, q') \in Q \times Q$  do
3:     if  $(q \in Q_b \wedge q' \in Q_v) \vee (q \in Q_v \wedge q' \in Q_b)$  then
4:        $M(q, q') \leftarrow -1$ 
5:     end if
6:   end for
7: else if ein neuer Zustand  $q^*$  wurde generiert als Nachfolger von  $q^\dagger$  then
8:   for  $q \in Q$  do
9:      $M(q, q^*) \leftarrow 0$ 
10:    if  $(q \in Q_b \wedge q^\dagger \in Q_v) \vee (q \in Q_v \wedge q^\dagger \in Q_b)$  then
11:       $M(q, q^\dagger) \leftarrow -1$ 
12:    end if
13:  end for
14: end if
15: repeat
16:   for  $(q, q') \in Q \times Q$  do
17:     if  $\exists \sigma \in \Sigma_\varepsilon : M(\delta(q, \sigma), \delta(q', \sigma)) = -1 \vee M_{trans}(q, q', \sigma) = -1$  then
18:        $M(q, q') \leftarrow -1$ 
19:     else
20:        $M(q, q') \leftarrow \min(\max_{\sigma \in \Sigma_\varepsilon} \{M(\delta(q, \sigma), \delta(q', \sigma)) + 1\}, |Q|)$ 
21:     end if
22:   end for
23: until kein Eintrag von  $M$  hat sich in dieser Iteration verändert

```

---

sind auch im Algorithmus entsprechende einfache Optimierungen möglich: Statt allen Paaren  $(q, q') \in Q \times Q$  können auch nur solche mit  $q \neq q'$  betrachtet werden, und wenn die Zustände über numerische Bezeichner  $id: Q \rightarrow \mathbb{N}$  geordnet sind, genügt es, sich auf eine der beiden Richtungen festzulegen, z.B. nur die Paare mit  $id(q) < id(q')$ . Für die Markierungstabelle kann dann konzeptionell z.B. eine obere Diagonalmatrix verwendet werden.

### 5.3. Unifikation

Wenn im Markierungsalgorithmus ein oder mehr Zustandspaare identifiziert werden, die bestätigend mit einer Tiefe größer oder gleich dem Unifikationsschwellwert  $m$  sind, wird anschließend ein Unifikationsschritt eingeleitet. Dies bedeutet informell, Zustandsmengen zu ermitteln, in denen jeweils alle Zustände semantisch gleich sein sollten, und alle Vorkommen von Zuständen aus einer solchen Menge durch einen gemeinsamen *Repräsentanten* zu ersetzen.

### 5.3.1. Algorithmus

---

**Algorithmus 2** Unifikationsalgorithmus, nachdem ausreichend tief bestätigende Zustandspaare gefunden wurden.

---

**Daten:** Union-Find-Datenstruktur  $I$  für Zustände ▷ z.B. Disjoint-Set-Forest [56]

```

1: for  $(q, q') \in Q \times Q \mid M(q, q') \geq m$  do
2:    $I.union(q, q')$ 
3: end for
4: repeat
5:   for  $q \in Q \mid I.find(q) \neq q$  do
6:     for  $\sigma \in \Sigma_\epsilon$  do
7:       if  $I.find(\delta(q, \sigma)) \neq I.find(\delta(I.find(q), \sigma))$  then
8:          $I.union(\delta(q, \sigma), \delta(I.find(q), \sigma))$ 
9:       end if
10:    end for
11:   end for
12: until keine vorher getrennten Zustandsmengen wurden in dieser Iteration vereinigt
13: for  $q \in Q$  do
14:   for  $\sigma \in \Sigma_\epsilon$  do
15:      $\delta(q, \sigma) \leftarrow I.find(\delta(q, \sigma))$ 
16:   end for
17: end for
18: Setze Startzustand und aktuellen Zustand auf ihre korrekten Repräsentanten
19: Entferne Zustände, die nicht mehr vom Startzustand oder aktuellen Zustand aus
    erreichbar sind

```

---

Der Unifikationsalgorithmus ist in Algorithmus 2 in Pseudocode angegeben. Darin wird eine Union-Find-Datenstruktur (z.B. [55, 56, 69], auch Disjoint-Set-Datenstruktur) zur Verwaltung der unifizierten Zustandsmengen benutzt.

Der grobe Ablauf ist folgender: Zunächst werden die Zustandspaare vereinigt, die bereits mit Bestätigungstiefe  $m$  oder größer markiert waren.<sup>2</sup> Dies bedeutet, die Mengen, die ursprünglich die einzelnen Zustände enthalten, zu vereinigen. Die so entstandene Menge hat einen der beiden Zustände als Repräsentanten. Anschließend werden im eigentlichen Unifikationsschritt die ausgehenden Transitionen der zu vereinigenden Zustände betrachtet, und die Ziele von Transitionspaaren (unter einer gegebenen Eingabe  $\sigma \in \Sigma_\epsilon$ ) wiederum selbst zur Unifikation festgehalten. Dieser Schritt kann transitiv weitere Vereinigungen entlang der Zustandskette nach sich ziehen. Wenn alle Vereinigungen durchgeführt wurden (sprich: in einer Iteration der mittleren Schleife in Algorithmus 2 wurden keine neuen Zustandsmengen vereinigt), werden die Ziele aller Transitionen im Automaten auf ihre korrekten Repräsentanten aktualisiert. Zuletzt werden der Startzustand und der aktuelle Zustand ebenfalls durch ihre Repräsentanten ersetzt. Dadurch können nun Zustände

---

<sup>2</sup>Genau wie oben genügt es aus Symmetriegründen, Zustandspaare jeweils nur in einer Reihenfolge zu betrachten.

redundant geworden sein – vereinigte Zustände abseits ihres Repräsentanten, die nicht mehr vom Start- oder aktuellen Zustand aus erreichbar sind. Diese Zustände können durch einfache Erreichbarkeitstests auf dem Graphen ermittelt und entfernt werden.

### 5.3.2. Anwendung im Programmiersystem

Das oben beschriebene Unifikationsverfahren führt im Zusammenhang mit der Demonstration überlappender Schleifeniterationen zur Synthese der Schleifenstruktur: Nachdem die sich wiederholenden Zustände als identisch festgestellt werden, entstehen bei der Vereinigung mehrere Transitionen zum selben Repräsentanten, und damit Kanten zurück zu einem früheren Zustand im Automaten.

Die beiden Prozeduren (Markierung und Unifikation) lassen sich nun wie folgt in den Programmierablauf integrieren: In jedem Schritt wird weiterhin eine Aktion programmiert, die als Updateschritt entlang einer neuen Transition zu einem neuen Zustand notiert wird. Anschließend wird der Markierungsalgorithmus auf dem Automaten ausgeführt. Dabei können zunächst alle Zustandspaare mit dem neu erstellten Zustand mit einer Bestätigungstiefe von 0 markiert werden (der neue Zustand hat noch keine ausgehenden Kanten und kann daher noch keine Widersprüche erzeugen). Des Weiteren kann die Markierungstabelle aus dem letzten Programmierschritt als Ausgangsbasis dienen: Markierungen können immer nur erhöht werden, oder kollabieren in einen Wert von  $-1$ , sobald ein Widerspruch festgestellt wurde. Wenn nun im Markierungsschritt Zustandspaare mit Bestätigungstiefe  $\geq m$  erkannt wurden, wird anschließend die Unifikation durchgeführt.

Der beschriebene Vorgang setzt einen passend gewählten Unifikationsschwellwert  $m$  voraus. Insbesondere sollte dieser eher zu hoch als zu niedrig gesetzt sein, da durch wiederholte Programmierung in jedem Fall die Schwelle erreicht werden kann, da aber andererseits durch fälschlicherweise vereinigte Zustände ein Teil der Programmierung verlorengehen kann.<sup>3</sup> Eine Möglichkeit, damit umzugehen, wäre seitens des Systems das Mitteilen einer potenziellen Vereinigung (mittels Audio, bestimmten Roboterbewegungen o.ä.), und seitens der Nutzenden die Möglichkeit zur Ablehnung, worauf die betroffenen Zustände explizit als widersprüchlich markiert werden könnten. Dies wäre gewissermaßen eine Erweiterung der Widerspruchsdefinition um eine Orakelentscheidung.

Unter Verwendung von Ablehnung (und Bestätigung) könnte der Schwellwert  $m$  auch im Lauf der Programmierung angenähert werden. Häufige Ablehnung der vorgeschlagenen Vereinigungen würde dazu führen, dass  $m$  erhöht wird. In entgegengesetzter Richtung könnte  $m$  mit der Zeit gesenkt werden, solange nicht zu viele Ablehnungen auftreten. Dieser Ansatz ist angelehnt an den Page-Fault-Frequency-Algorithmus zur Seitenverdrängung im Kontext von Betriebssystemen [37]. Er verlagert das Problem von der Identifikation eines Schwellwerts für die Synthese auf die eines Schwellwerts für die Obergrenze der Ablehnungsfrequenz (und ggfs. eines zweiten für die Untergrenze).

Für die folgende Verifikation des Ansatzes wurden feste, für die Aufgabe geeignete Syntheseschwellwerte verwendet.

---

<sup>3</sup>Man bemerke, dass für beliebiges  $m$  pathologische Aufgaben formuliert werden können, sodass unerwünschte Unifikation stattfindet – für praktische Aufgaben werden aber handhabbare Mengen an Überlappung angenommen.

## 5.4. Evaluation

Der beschriebene Ansatz wurde praktisch an verschiedenen Aufgaben erprobt, wovon stellvertretend ein Szenario beschrieben wird, das auch in den Benutzerstudien zur Anwendung kam. Daneben können einige Überlegungen zur Laufzeitkomplexität getroffen werden.

### 5.4.1. Experimentelle Verifikation

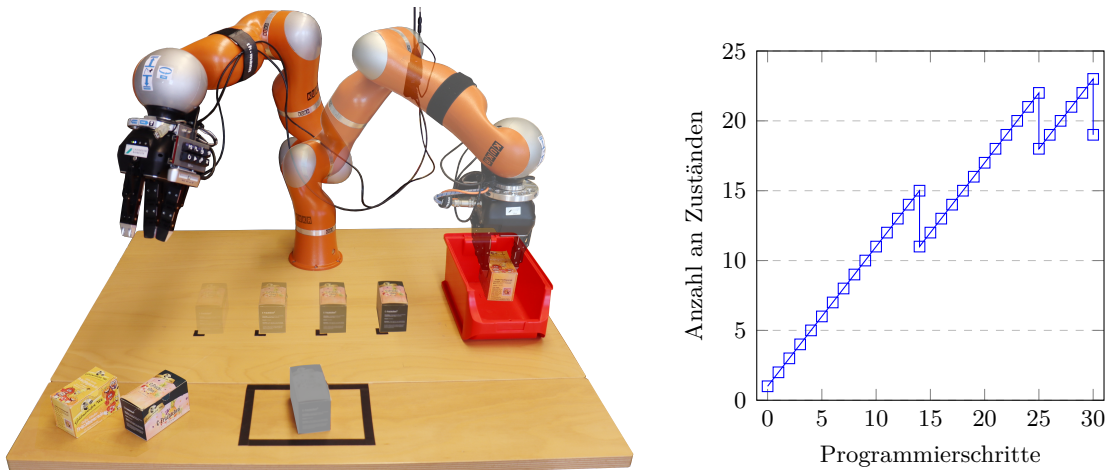
Als Beispiel für die oben beschriebenen Verfahren zur Struktursynthese durch Unifikation von Zuständen wird der Prozess für die Aufgabe aus der Benutzerstudie zur Evaluation des ERSA-Systems an sich dargestellt (mehr dazu in Kapitel 7). Diese besteht darin, wiederholt ankommende Objekte aus einer von zwei Kategorien handzuhaben. Eine der Objektkategorien soll an markierten Ablagepositionen aufgereiht werden. Dabei entstehen an zwei Stellen im Programm Schleifenstrukturen: Erstens die wiederholte Verarbeitung ankommender Objekte in einer äußeren Schleife um das gesamte übrige Programm; Zweitens die Ablage eines einzelnen Objektes vom zweiten Typ, wofür zunächst die Ablageorte überprüft werden müssen, was ebenfalls Wiederholung beinhaltet. (Abbildung 5.3b zeigt einen ERSA zu dieser Aufgabe mit allen geschlossenen Schleifen.) Die weiteren Details der Aufgabe, die dazu dienen, die komplette Automatendefinition abzufragen, sind an dieser Stelle noch nicht von näherem Belang. Eine Konzeptgrafik zur Aufgabe, wie sie am Roboter umzusetzen ist, ist in Abbildung 5.1 links abgebildet.

Bei der Verifikation der Struktursynthese wurde jeweils die komplette Aufgabe demonstriert, an den erforderlichen Stellen mit mehreren sich überlappenden Schleifeniterationen. Der Unifikationsschwellwert  $m$  wurde hierbei auf 3 festgelegt. Dies ist der kleinste für die Aufgabe geeignete Wert: Es tritt eine Überlappung von zwei Zuständen auch dadurch auf, dass in beiden Zweigen der äußeren Schleife über ein erkanntes Objekt verfahren und dieses aufgegriffen wird. Die Zweige dürfen aber explizit nicht vereinigt werden.

Die Gesamtanzahl an Zuständen im Verlauf über die Programmierschritte ist in Abbildung 5.1 rechts zu sehen. Dabei steigt die Anzahl stets linear an, bis in den Schleifeniterationen eine ausreichende Bestätigungstiefe erreicht wurde. An diesen Punkten kollabiert die überlappende Struktur jeweils durch die Unifikation in eine geschlossene Schleife, woraufhin eine Zahl an Zuständen in Abhängigkeit vom Unifikationsschwellwert als unerreichbar entfernt wird. Dabei handelt es sich genau um die redundanten Versionen der überlappenden Zustände. In Abbildung 5.2 ist ein Teilautomat vor und nach der Unifikation des ersten Zweiges in eine geschlossene Schleife abgebildet. Dabei wird ersichtlich, dass bereits eine teilweise Überlappung der Schleifeniterationen für eine Synthese der Schleifenstruktur ausreichend ist.

In Abbildung 5.3 ist die Unifikation der kleineren, inneren Schleife dargestellt, d.h. der Iteration über mögliche Ablageorte. Da diese Schleife (je nach Umsetzung) nur einen einzelnen Zustandsübergang enthält, muss zum Erreichen des Unifikationsschwellwerts dieser Schleifenrumpf mehrmals programmiert werden. In absoluten Zahlen bedeutet das aber nach wie vor, dass nur  $m = 3$  weitere Übergänge angelegt werden müssen.

Die Synthese der äußeren Schleife mit zwei Zweigen erfordert, dass beide Zweige als



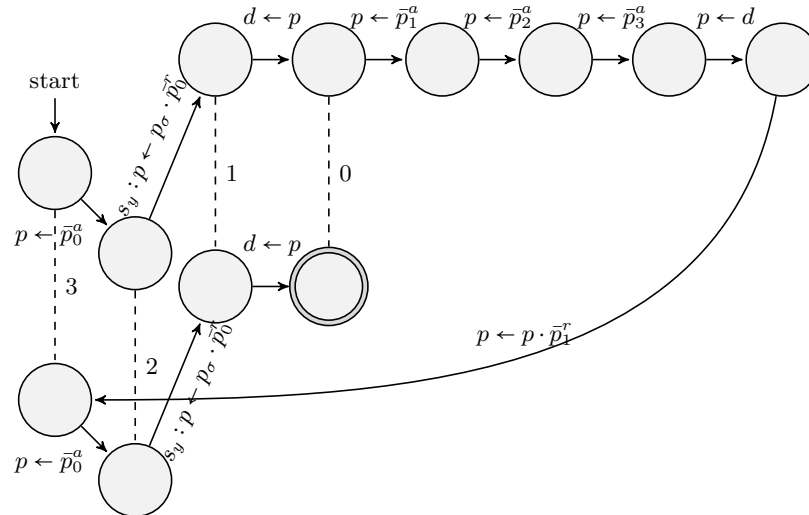
**Abbildung 5.1.:** Links: Versuchsaufbau, wie bei der ERSA-Benutzerstudie. Objekte werden im schwarz markierten Areal angeliefert und müssen entweder in den Behälter eingetaucht und an die genaue Aufnahmeposition zurückgelegt werden, oder an der ersten markierten Stelle platziert werden, die noch frei ist. Rechts: Anzahl an Zuständen über der Anzahl an programmierten Schritten in einem Beispieldurchlauf. In jedem Schritt wird ein neuer Zustand hinzugefügt. In Schritt 14, 25 und 30 wird ein neuer Zustand generiert, aber anschließend werden durch die Unifikation  $m + 1 = 4$  Zustände als unerreichbar entfernt (genau die Zustände entlang der drei überlappenden Transitionen, die redundant zu ihrer ursprünglichen Repräsentation sind).

Wiederholungen identifiziert werden. Je nach Programmierreihenfolge können dabei unterschiedliche Zwischenzustände entstehen, wie in Abbildung 5.4 visualisiert. Ungeachtet dessen entsteht unter ausreichend großer Überlappung die erwünschte Gesamtstruktur. An diesem Beispiel sieht man auch, dass die Anzahl an zwischenzeitlich existierenden Zuständen teilweise von der Reihenfolge der Programmierung und der entsprechenden Synthese abhängt. In Abbildung 5.4a ist etwa der Zustandsübergang mit der ersten absoluten Bewegung dreifach redundant vorhanden, wofür die Vereinfachungen noch ausstehen.

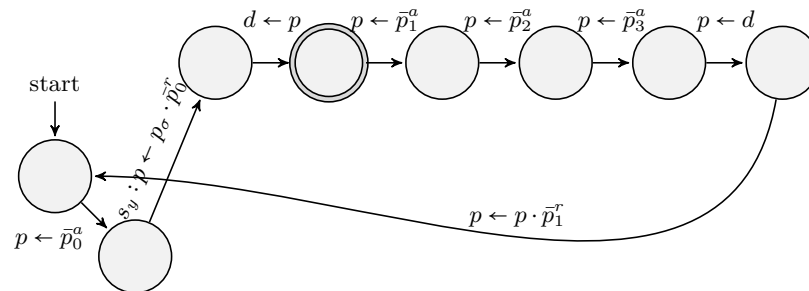
Neben der rein funktionalen Verifikation wurde stichprobenhaft auch die Anwendbarkeit des Verfahrens durch zwei Nichtexperten erprobt. Auch in diesen beiden Fällen konnte die Programmstruktur nach vollständiger Demonstration erfolgreich synthetisiert werden, d.h. auch die Nichtexperten konnten das Konzept hinter der Unifikation aus teilweiser Überlappung verstehen und anwenden.

### 5.4.2. Komplexität

Das früher in diesem Abschnitt vorgestellte Verifikationsszenario stellt grob ein Beispiel dar, auf welche Größe an Aufgaben das System abzielt. Für wesentlich größere und komplexere Anwendungen ist die Anwendbarkeit des Ansatzes ohne grafische Repräsentation vermutlich stärker eingeschränkt; Zumindest der Bedienkomfort bzw. die Nutzbarkeit wird deutlich abnehmen. Daher bewegen sich die zu erwartenden Automaten in der Größen-



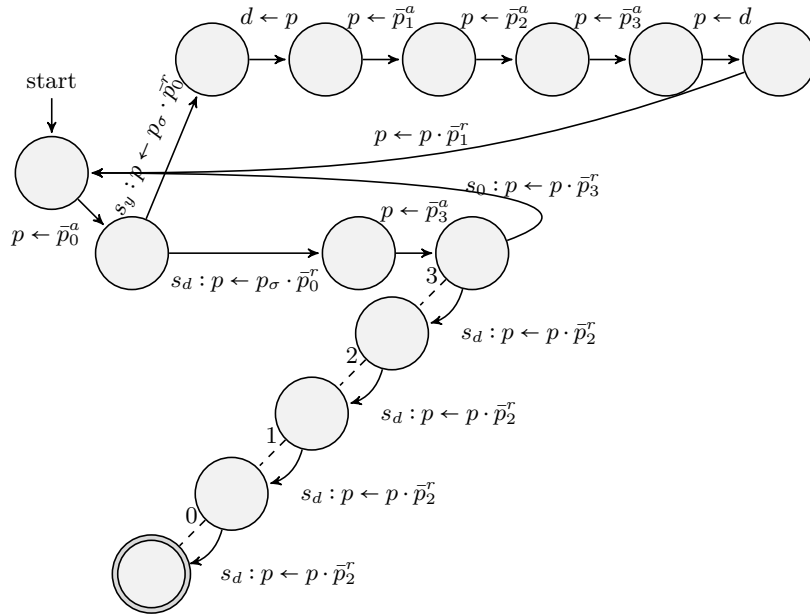
(a) Teilautomat vor Unifikation. Die Bestätigungstiefe der relevanten Zustandspaare ist entlang der gestrichelten Verbindungen notiert.



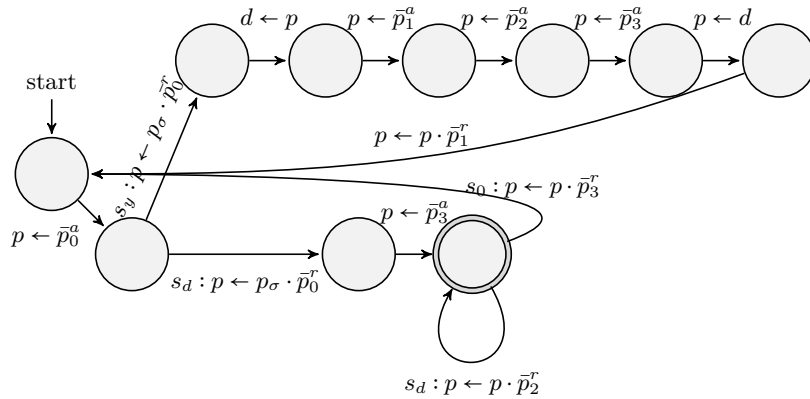
(b) Teilautomat nach Unifikation.

**Abbildung 5.2.:** Teilautomaten vor (5.2a) und nach (5.2b) der ersten Unifikation. Direkt vor der Vereinigung hat das ganz linke Zustandspaar eine Bestätigungstiefe von  $3 \geq m = 3$ . Von diesem Paar ausgehend, werden Zustandspaare vereinigt, was jeweils die Vereinigung der direkten Nachfolgerzustände nach sich zieht. Der aktuelle Zustand nach der Unifikation ist der Repräsentant für den aktuellen Zustand vor der Unifikation (jeweils mit einer doppelten Umrandung markiert).



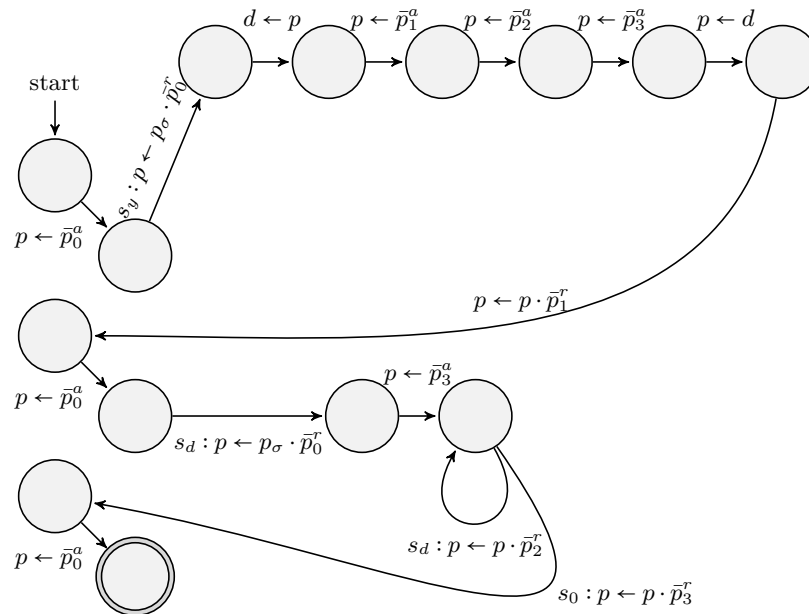


(a) Automat vor Synthese der Schleife über Ablageorte. Es wurde eine Kette nicht widersprüchlicher Zustände und Übergänge für den Schleifenrumpf angelegt. Die Bestätigungstiefe ist erneut entlang der gestrichelten Kanten annotiert.

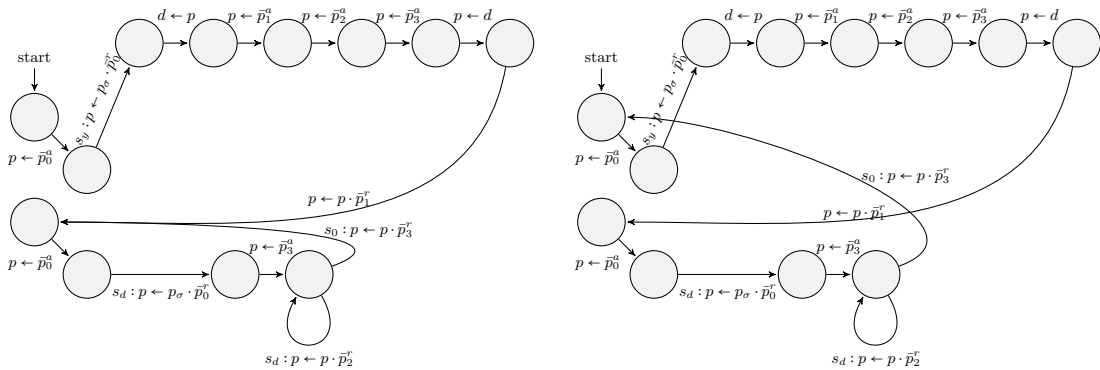


(b) Automat nach der Synthese der Ablageschleife. Die Kette an übereinstimmenden Zuständen und Übergängen ist in die kleinste (und dadurch am stärksten bestätigte) geschlossene Schleife kollabiert.

**Abbildung 5.3.:** Ein weiterer partieller Automat für die gleiche Beispielaufgabe, vor (5.3a) und nach (5.3b) Synthese der Schleife über Ablageorte. Die abgebildete Kette an Zuständen und Übergängen erreicht eine ausreichende Bestätigungstiefe (genauer: die ersten beiden Zustände darin). Die Zustände werden entlang der Kette paarweise vereinigt, was dazu führt, dass am Ende nur ein einzelner Zustand übrig bleibt, mit einer Transition zu sich selbst mit der entsprechenden relativen Bewegung. Durch die Übereinstimmung entlang der Kette sind auch andere Zustands-paare als nicht widersprüchlich markiert, aber deren Bestätigungstiefe führt weniger schnell zu einer Unifikation. Z.B. könnte auch ein Schleifenrumpf aus jeweils zwei der Übergänge gebildet werden.



(a) Automat nach Programmierung beider Zweige in der äußeren Schleife hintereinander. In keinem der beiden Fälle wurde direkt eine zweite Iteration angelegt, sodass das Programm aktuell eine einzelne Sequenz der beiden Teilstrukturen darstellt.



(b) Automat aus 5.4a nach Fortfahren mit einer weiteren Iteration des unteren Zweigs und entsprechender Schleifensynthese. Die Schleife des oberen Zweigs ist nach wie vor offen, und es muss noch der Beginn der nächsten Wiederholung programmiert werden (entweder vom Startzustand oder vom Start des unteren Zweigs aus).

(c) Automat aus 5.4a nach Fortfahren mit einer weiteren Iteration des oberen Zweigs. Die Sequenz aus beiden Zweigen wurde als eine große Schleife interpretiert und entsprechend zusammengelegt. Um die beiden einzelnen Teilstrukturen zu trennen, muss nun in einem der beiden Fälle eine direkt Wiederholung des Zweigs programmiert werden.

**Abbildung 5.4.:** Teilautomaten im Fall, dass die äußere Schleife nicht sofort geschlossen wurde, bzw. in 5.4a genauer genommen nach der Rückkehr in den gemeinsamen Verzweigungszustand. Die Schleifen der beiden Zweige können nun nicht sofort beide geschlossen werden. Je nach Reihenfolge können zwei Fälle entstehen: Bei erneuter Programmierung der ersten Aktionen aus dem unteren Zweig wird die Schleife für diesen geschlossen (5.4b). Bei erneuter Programmierung der ersten Aktionen aus dem oberen Zweig wird stattdessen die Sequenz der beiden Zweige zu einer großen Schleife zusammengelegt (5.4c). In beiden Fällen kann in der Folge durch weitere Überlappung die korrekte Gesamtstruktur erzeugt werden.

ordnung von maximal wenigen hundert, vielmehr vermutlich einigen Dutzend Zuständen und Übergängen. Unter diesem Gesichtspunkt und der Erkenntnis, dass neue Zustände nur explizit auf Nutzereingaben hin generiert werden (was schätzungsweise einmal in der Sekunde und seltener passiert), ist die asymptotische Zeitkomplexität kein besonders kritischer Faktor. Der vorgestellte Ansatz aus Markierungsalgorithmus und Unifikation könnte aber genauso auf Systemen Anwendung finden, die neue Zustände während der Aufnahme einer Trajektorie automatisch und höherfrequent generieren, eher an der Playback-Programmierung orientiert (statt wie hier an einer Teach-In-Programmierung). Die Taktraten der Abtastung würden sich in dem Fall eher im zwei- bis dreistelligen Hertzbereich bewegen. Dadurch können schon bei relativ einfachen Aufgaben Automaten mit Tausenden und Zehntausenden Zuständen entstehen. Daher betrachten wir an dieser Stelle dennoch einige grobe Abschätzungen für die Zeitkomplexität der Verfahren.

Der in Algorithmus 1 angegebene Markierungsalgorithmus besteht aus einer Schleife, deren Rumpf die Komplexität  $\mathcal{O}(|Q|^2 \cdot |\Sigma|)$  hat. Die Schleife wird so lange ausgeführt, bis sich in einer Iteration keiner der Einträge mehr verändert hat. Die Einträge selbst haben Werte von  $-1$  bis  $|Q|$ , und können sich nicht beliebig verändern: Die Bestätigungstiefe entlang zweier Zustandsketten kann nur entweder ansteigen, oder auf  $-1$  kollabieren, falls weiter hinten ein neuer Konflikt entstanden ist. Aus einem Konflikt kann keine andere Bestätigung mehr entstehen. Damit kann sich die Markierung für jedes einzelne Zustandspaar maximal  $\mathcal{O}(|Q|)$  mal verändern. Insgesamt ergibt sich durch Multiplikation mit der Anzahl an Zustandspaares eine obere Schranke von  $\mathcal{O}(|Q|^3)$  für die Anzahl an Wiederholungen der Schleife, und von  $\mathcal{O}(|Q|^5 \cdot |\Sigma|)$  für die Laufzeit des Markierungsalgorithmus insgesamt.

In der Unifikation wird mit Vereinigungsoperationen der Union-Find-Datenstruktur zuerst über die Zustandspare iteriert, dann über die Zustände selbst. Letzteres wiederholt sich so lange, bis sich in einer Iteration keine Vereinigungsmenge verändert hat. Auch dies kann nur endlich lange dauern, da die einzelnen Zustandsmengen nur vereinigt, nicht aber wieder aufgetrennt werden. Es kann maximal für jeden Zustand eine einzelne Vereinigungsoperation stattfinden, und in jeder Operation muss auch mindestens eine Vereinigung passieren. Wenn die beiden Operationen der Union-Find-Datenstruktur (konservativ) mit  $\mathcal{O}(\log(|Q|))$  nach oben abgeschätzt werden (siehe z.B. [146]), ergibt sich eine Gesamtkomplexität von  $\mathcal{O}(|Q|^2 \cdot \log(|Q|) \cdot |\Sigma|)$  für diesen Teil des Algorithmus. Der anschließende Schritt, alle Transitionsziele auf ihre korrekten Repräsentanten zu setzen, läuft in einer Schleife über alle Zustände ab, und wird durch diese Abschätzung bereits dominiert. Das abschließende Entfernen unerreichbarer Zustände kann mit üblichen Suchmethoden auf Graphen in  $\mathcal{O}(|Q|^2)$  passieren.

Insgesamt ist damit die Schleife im Markierungsalgorithmus der aufwändigste Teil beider Algorithmen. Die Komplexität von  $\mathcal{O}(|Q|^5 \cdot |\Sigma|)$  ist immer noch in Polynomialzeit und kann damit zwar aus theoretischer Sicht als handhabbar betrachtet werden, allerdings dürfte für die oben erwähnte mögliche Anwendung auf Playback-Ansätze, mit etwa einigen Tausend Zuständen, der Exponent bereits problematisch sein. Aus praktischer Sicht wurden die in Teach-In-Manier angewendeten Verfahren, wie im letzten Abschnitt dargestellt, auf Beispielaufgaben verifiziert. Weiterhin ließe sich die obere Schranke für Markierung-

gen (die auch über die äußere Schleife direkt in die Komplexität eingeht) vermutlich auf Kosten einer komplexeren Verfolgung von Bestätigungspfaden noch verbessern.

6.1. Stand der Forschung . . . . .	81
6.1.1. Synchronisationsvarianten . . . . .	82
6.1.2. Automatenmodelle . . . . .	83
6.2. Multi-ERSA . . . . .	84
6.2.1. Definition . . . . .	84
6.2.2. Ausführung . . . . .	86
6.2.3. Beispiel . . . . .	86
6.3. Programmierung im Gesamtsystem . . . . .	88
6.3.1. Operationen . . . . .	88
6.3.2. Interaktionen . . . . .	88

---

Dieses Kapitel stellt eine Erweiterung des ERSA-Ansatzes auf mehrere Roboter vor. Dazu wird zunächst der Stand der Forschung in diesem Bereich aufgegriffen, insbesondere unter den Gesichtspunkten der verwendeten Synchronisationsmechanismen und Formalismen. Anschließend wird das Modell der Multi-ERSA beschrieben, das entsprechende Möglichkeiten umsetzt, sowie zuletzt die Programmierung auf dessen Basis. Die Inhalte in diesem Kapitel wurden teilweise in ihrer ursprünglichen Form in [181] veröffentlicht.

### 6.1. Stand der Forschung

In diesem Abschnitt werden verwandte Arbeiten aus der Multi-Roboter-Programmierung betrachtet. Dabei ist der Überblick gegliedert in die beiden Gesichtspunkte der jeweils vorhandenen Synchronisationsmöglichkeiten und der verwendeten Automatenmodelle mit ihren jeweiligen interessanten Eigenschaften.

### 6.1.1. Synchronisationsvarianten

Arbeiten im Feld der Multi-Roboter-Programmierung stellen allgemein zumindest zwei Synchronisationsvarianten neben der unabhängigen Ausführung zur Verfügung [122, 176]. Das ist erstens die Ausführung ab einem gemeinsamen Start- oder Wartezeitpunkt. Tritt diese innerhalb eines längeren Ablaufs auf, müssen jeweils Roboter, die den Synchronisationspunkt zuerst erreichen, auf die anderen warten. Nach Ankunft aller Roboter können diese gleichzeitig, wieder unabhängig voneinander fortfahren. Dieser Synchronisationsmechanismus wird im Folgenden als *Rendezvous* bezeichnet.<sup>1</sup> Zweitens gibt es die *synchrone* (sprich: kontrolliert nebenläufige) Ausführung von Aktionen. Dabei finden die Abläufe mehrerer Roboter über einen Zeitraum hinweg in festem zeitlichen Zusammenhang statt. Die Terminologie variiert; So bezeichnet z.B. [122] die beiden Fälle als *barriers* und *ranges*, während [176] von *loose* und *tight synchronisation* spricht.

Auch im Feld der Multi-Roboter-Programmierung werden häufig grafische Editoren eingesetzt, die diese Synchronisationsvarianten unterstützen [132, 135, 151]. Dabei variiert die Darstellung der Programme und Synchronisationsprimitive; Beispielsweise verwendet erstere Arbeit eine auf Gantt-Diagrammen basierende Schnittstelle. Darin können Intervalle auf einer Zeitleiste als synchron markiert werden. Rendezvous (auch Synchronisationspunkte genannt) lassen sich in dem Formalismus als degenerierte Synchronisationsintervalle mit Länge 0 darstellen.

Teilweise werden synchrone Aktionen (insb. Bewegungen) weiter differenziert. So unterscheidet [57] gekoppelte (engl. *coupled*) und kombinierte (engl. *combined*) synchrone Bewegungen, wobei bei ersterem mit den (beiden) involvierten Robotern identische Bewegungen ausgeführt werden, während bei letzterem ein Roboter Bewegungen relativ zur sich verändernden Pose des anderen Endeffektors ausführt. Auch in [178] wird weiter unterschieden in symmetrische und asymmetrische koordinierte Programmteile, wobei wiederum letzteres unterschiedliche bzw. relative Bewegung des zweiten Roboters bezeichnet.

Zuletzt wird reine *Präzedenz* (im Sinne von Programmteilen mit einer einseitigen vorher-nachher-Beziehung) ebenfalls teilweise als Synchronisationsvariante betrachtet, etwa in [177] (dort unter der Bezeichnung *soft synchronisation*, im Kontrast zu Rendezvous als *hard synchronisation*).

Zusammenfassend lassen sich daraus die drei Synchronisationsvarianten aufstellen, die auch im entwickelten Formalismus modelliert werden:

- Präzedenz als asymmetrische Synchronisationspunkte
- Rendezvous als symmetrische Synchronisationspunkte
- echt synchrone Intervalle oder Aktionen

---

<sup>1</sup>Der Begriff geht auf die Verwendung aus dem Kontext von Prozesssynchronisation zurück, wie etwa in [25, 155] dargestellt. Die reine Verhaltensbeschreibung entspricht eigentlich einer sogenannten *Barrier* [155], aber die später implementierte Form verwendet intern tatsächlich eine synchron übermittelte Nachricht und wurde deshalb nach der genaueren Ähnlichkeit als Rendezvous benannt. Aus Konsistenzgründen wird diese Terminologie auch hier verwendet.

Eine weitere Unterteilung der synchronen Aktionen wird im Formalismus nicht modelliert, aber im Ausblick (Abschnitt 8.2) wird nochmals darauf eingegangen.

### 6.1.2. Automatenmodelle

In puncto Automaten gibt es mehrere der bereits in Kapitel 2 vorgestellten Modelle, die in der Multi-Roboter-Programmierung zur Anwendung kommen. An dieser Stelle werden diese noch einmal kurz umrissen und explizit unter dem Gesichtspunkt der verwendeten Synchronisationsmechanismen betrachtet.

Eines der Modelle sind *Petrinetze* [177,178]. Diese sind durch ihre Grundfunktionalität des Transports von Marken genau für zeitliche Abläufe in Netzwerken geeignet, und werden auch außerhalb der Roboterprogrammierung zu solchen Zwecken verwendet. Petrinetze unterscheiden sich in einigen fundamentalen Aspekten von Roboterzustandsautomaten. So gibt es bei letzteren kein Konzept von Marken, die sich in den einzelnen Stellen (oder Zuständen) befinden können. Gleichwohl muss für die Multi-Roboter-Programmierung auch in Automaten die Bedingung aufgeweicht werden, dass das Gesamtsystem sich in jedem Moment nur in genau einem Zustand befindet. Weiterhin sind Transitionen in Petrinetzen formal eine von zwei Knotenarten in einem bipartiten Graphen, im Gegensatz zu Kanten in Roboterzustandsautomaten. Ungeachtet der Unterschiede lassen sich mehrere interessante Aspekte herausstellen: Einerseits die implizite Synchronisation von Transitionen im Netz durch die Voraussetzung von Marken in den Eingabestellen, andererseits die Möglichkeit von expliziten Bedingungen an Kanten, wodurch sich weitere Einschränkungen modellieren lassen. Zuletzt wird in [177] mit Petrinetzplänen eine Variante an Netzen betrachtet, die gerade dazu gedacht ist, in Teilnetze für die einzelnen Roboter zerlegt zu werden. Auch dieser Aspekt wird im späteren Formalismus in diesem Kapitel aufgegriffen.

Ein anderes verwendetes Automatenmodell sind die sogenannten *Timed Automata* [8,10,126]. Diese beinhalten, wie der Name suggeriert, zeitliche Variablen, die entlang der Kanten gesetzt oder in Bedingungen verwendet werden können. Timed Automata zielen in ihrer Verwendung eher auf die formale Analyse ab als auf intuitive Programmierbarkeit. Auch die zeitlichen Variablen, die tatsächlich mit der fortschreitenden Zeit mitlaufen, kommen im entwickelten Formalismus nicht in der Form zur Anwendung, sondern nur eine ähnliche Verwendung an Transitionen.

Der letzte in der Multi-Roboter-Programmierung verwendete Formalismus, der hier betrachtet wird, sind *Statecharts* (oder auch z.B.: *hierarchische Automaten*) [12,102,152]. Diese stellen ebenfalls eine Erweiterung endlicher Automaten dar, je nach konkreter Variante mit derselben oder erweiterter Mächtigkeit, i.d.R. aber zumindest zur konziseren Darstellung. Die grundlegenden Erweiterungsmöglichkeiten sind dabei die hierarchische Verknüpfung von Automaten (d.h., Zustände in einem höherliegenden Automaten werden auf ganze tieferliegende Automaten abgebildet), und die parallele Ausführung von (Teil-) Automaten. Beide dieser Anwendungen übersteigen das Niveau an Komplexität, auf das der vorliegende, nicht auf eine grafische Repräsentation gestützte Ansatz abzielt. Dennoch lässt sich auch hier die wiederholte Verwendung von Variablen und Kantenbedingungen festhalten, die im Formalismus später aufgegriffen wird.

Die herausgestellten Aspekte dienen als Grundlage für das Konzept von *Multi-ERSA*, welches im Folgenden definiert wird. Dabei wird das zugrunde liegende Modell der Roboterzustandsautomaten erweitert, ohne die Semantik in einer der erwähnten existierenden Richtungen essenziell weiter zu öffnen.

## 6.2. Multi-ERSA

Die folgende Definition erweitert die Roboterautomaten bzw. ERSA in Richtung Multi-Roboter-Programmierung, mit einer Anzahl  $k$  an Systemkomponenten (d.h. Teilautomaten, und korrespondierend dazu Robotern), und einer Anzahl  $m$  an verwendeten Signalen.

### 6.2.1. Definition

Grundlegend ist die formale Definition aufgebaut wie schon bei ERSA. Die zentrale Änderung ist, dass jede Komponente des ERSA-Tupels (also die Zustandsmenge, die Transitionsfunktion etc.) für jede der verschiedenen Systemkomponenten (also jeden einzelnen Roboter) vorkommt, adressiert über den entsprechenden Index. Es gibt also gewissermaßen für den  $i$ -ten Roboter einen Teilautomaten mit den Bestandteilen  $Q_i, \Sigma_i, P_i, n_i, \delta_i, u_i, q_{s,i}$ . Für die volle Multi-ERSA-Definition kommen dann noch die Synchronisationsmechanismen hinzu.

Zur Notation: Im Folgenden wird  $(Q_i)$  benutzt für das Kreuzprodukt mehrerer Komponenten  $Q_1$  bis  $Q_k$ . Dies dient sowohl der Übersicht, als auch als Umgang mit der unbekanntem Zahl  $k$  an involvierten Systemkomponenten. Die Verwendung von  $(\Sigma_i)$ ,  $(P_i)$  usw. erfolgt analog. Mit  $Q = \bigcup_{i=1}^k Q_i$  wird die Vereinigung aller Zustandsmengen der Teilautomaten bezeichnet. Weiterhin wird  $[m]$  verwendet für die Menge an natürlichen Zahlen  $\{1, \dots, m\} \subset \mathbb{N}$ . Es gibt die Erweiterung  $[m]_\varepsilon = [m] \cup \{\varepsilon\}$  wie bisher.

**Definition 6.2.1** (Multi-ERSA). Ein Multi-Roboter-ERSA, kurz Multi-ERSA, ist definiert als Tupel  $M = ((Q_i), (\Sigma_i), (P_i), (n_i), (\delta_i), (u_i), (q_{s,i}), m)$ ,  $1 \leq i \leq k$ . Dabei ist jeweils für die einzelne Systemkomponente  $i$

- $Q_i$  die endliche Zustandsmenge
- $\Sigma_i$  das endliche Eingabealphabet
- $P_i$  der Raum an Posen
- $n_i$  die Länge des Posenvariablenvektors (und entsprechend  $D_i = P_i^{n_i}$  der zugehörige Raum)
- $\delta_i : Q_i \times \Sigma_{i,\varepsilon} \rightarrow Q_i \times [m]_\varepsilon$  die Transitionsfunktion
- $u_i : Q_i \times \Sigma_{i,\varepsilon} \rightarrow \text{com}_i \times [m]_\varepsilon$  die Updatefunktion
- $q_{s,i} \in Q_i$  der Startzustand



sowie:

- $m$  die Anzahl der verwendeten Signale

Im Folgenden wird jede der Komponenten nochmals kurz eingeordnet.

Wie bereits erwähnt bezeichnen die Komponenten  $P_i$  und  $D_i = P_i^{n_i}$  den Raum der Posen und des Posenvektors der  $i$ -ten Systemkomponente (d.h. des  $i$ -ten Roboters).  $S = \mathbb{N}^m$  bezeichne den Raum des Signalvektors. Mit  $s \in S$  bezeichnet  $s_j$  die  $j$ -te Komponente von  $s$ , d.h. das  $j$ -te Signal. Einzelne Signale verhalten sich konzeptionell ähnlich zu Semaphoren [45]: Sie lassen sich atomar inkrementieren, überprüfen und dekrementieren, und ein Semaphorwert von 0 führt dazu, dass vor dekrementierenden Kanten gewartet wird.<sup>2</sup> Die Menge der zu Signalen aus  $S$  zulässigen Indizes ist genau  $[m]$ .

Transitionen sind definiert über die partiellen Funktionen  $\delta_i : Q_i \times \Sigma_{i,\varepsilon} \rightarrow Q_i \times [m]_\varepsilon$ . Dabei geben die ersten beiden Argumente wie bisher einen Zustand und eine passende Eingabe an. Die erste Komponente des Funktionswerts gibt den Folgezustand an. Die zweite Komponente des Funktionswerts spezifiziert zusätzlich dazu optional den Index eines erforderlichen Signals (d.h.  $\varepsilon$ , falls keines). Dieses Signal wird vor Ausführung der entsprechenden Transition geprüft. Die Transition kann nur verfolgt werden (d.h. der Teilautomat in den Folgezustand übergehen), wenn der aktuelle Wert des Signals größer Null ist, in welchem Fall das Signal auf Auslösen der Transition hin dekrementiert wird. Bezüglich Determinismus folgt aus der Darstellung der  $\delta_i$  als Funktionen, dass ein Zustand nicht mehrere ansonsten gleiche ausgehende Kanten unter verschiedenen erforderlichen Signalen haben kann. Dies ist erwünscht, da Signale sich nicht gegenseitig ausschließen.

Die Updatefunktionen sind definiert als  $u_i : Q_i \times \Sigma_i \rightarrow com_i \times [m]_\varepsilon$ . Dabei werden wie bisher nur Updates für Transitionen spezifiziert, die auch nach  $\delta$  definiert sind, d.h.  $\forall 1 \leq i \leq k \forall q \in Q_i, \sigma \in \Sigma_{i,\varepsilon} : [u_i(q, \sigma) \text{ definiert} \iff \delta_i(q, \sigma) \text{ definiert}]$ . Die erste Ausgabe-komponente aus  $com_i$  gibt dabei die Aktion des betroffenen Roboters an, und die zweite Komponente aus  $[m]_\varepsilon$  spezifiziert den Index eines zu inkrementierenden Signals (oder  $\varepsilon$ , falls keines).

Zu jedem Zeitpunkt gibt es im System einen aktuellen Signalvektor  $s \in S$ . Jeder Teilautomat  $i$  ist in einem aktuellen Zustand  $q_{cur,i} \in Q_i$ , hat einen aktuellen Posenvariablenvektor  $d_i \in D_i$ , und der korrespondierende Roboter befindet sich in einer aktuellen Pose  $p_{cur,i} \in P_i$ . Falls durch den Roboter  $i$  eine Eingabe erfasst wird, ist die Pose des erkannten Objekts durch  $p_{obj,i} \in P_i$  angegeben. Damit lassen sich die Mengen der validen Aktionen angeben als:

$$com_i = \left\{ \begin{array}{l} move\_abs[\bar{p}](out : p_{cur,i}), \\ move\_rel[\bar{p}](inout : p_{cur,i}), \\ move\_obj[\bar{p}](in : p_{obj,i}, out : p_{cur,i}), \\ save\_var[j](in : p_{cur,i}, inout : d_i), \\ load\_var[j](in : d_i, out : p_{cur,i}) \end{array} \right\}, 1 \leq i \leq k, \bar{p} \in P_i, 1 \leq j \leq n_i \quad (6.1)$$

<sup>2</sup>Für weitere Details zur Verwendung von Semaphoren siehe z.B. [25, 155].

### 6.2.2. Ausführung

Auch die einzelnen Ausführungsschritte sind eine Erweiterung der im ERSA-Modell beschriebenen. Die folgenden Beschreibungen sind der Übersicht halber auf den Fall von zwei Robotern beschränkt (d.h.  $k = 2$ ). Sie lassen sich aber auf höheres  $k$  verallgemeinern. Sei O.B.d.A. Teilautomat 1 in  $q_{cur,1} = q \in Q_1$  (für Teilautomat 2 analog): Mit Eingabe  $\sigma \in \Sigma_{1,\varepsilon}$ , falls  $\delta_1(q, \sigma) = (q', b) \in Q_1 \times [m]_\varepsilon$  und  $(b \neq \varepsilon \implies s_b > 0)$ , dann folgt der Automat dieser Transition und führt den entsprechenden Updateschritt aus. Dazu wird zunächst  $s_b$  dekrementiert, falls  $b \neq \varepsilon$ . Als nächstes wird die erste Komponente von  $u_1(q, \sigma)$ , auch bezeichnet als  $(u_1(q, \sigma))_1$ , ausgeführt. Anschließend wird, falls für die zweite Komponente  $(u_1(q, \sigma))_2 = b'$  gilt, dass  $b' \neq \varepsilon$ , das Signal  $s_{b'}$  entsprechend inkrementiert. Zuletzt wird  $q_{cur,1}$  auf  $q'$  gesetzt.

Der Aspekt von Multi-ERSA, Aktionen synchron auszuführen, entsteht darüber, dass Zustände in mehreren Teilautomaten enthalten sein können. Für gemeinsame Übergänge der Teilautomaten (von einem geteilten Zustand in einen geteilten Zustand) gilt, dass diese stets spontan sein sollen (um Uneindeutigkeiten zu umgehen, wenn in einem Automaten eine passende Eingabe vorliegt, im anderen nicht), und es sollen weder Signale erfordert noch gesetzt werden (um Schwierigkeiten in der Programmierung zu vermeiden). Formal ausgedrückt:

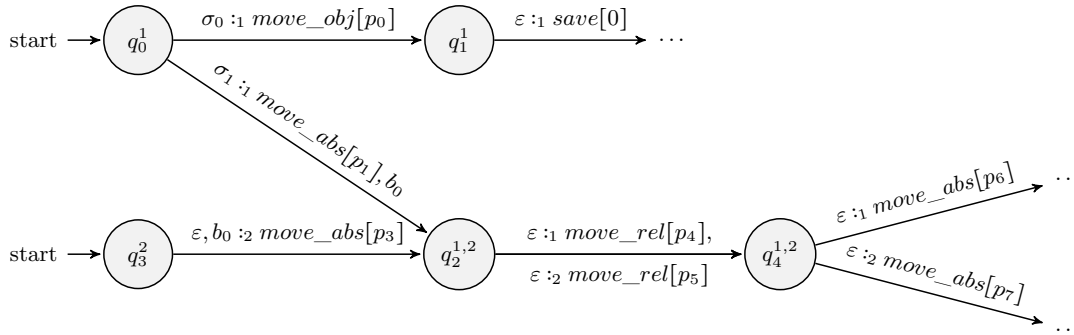
$$\begin{aligned} \forall q, q' \in Q_1 \cap Q_2 : \\ \left[ \exists \sigma \in \Sigma_{1,\varepsilon}, \sigma' \in \Sigma_{2,\varepsilon}, s, s' \in [m]_\varepsilon : \delta_1(q, \sigma) = (q', s) \wedge \delta_2(q, \sigma') = (q', s') \right] \\ \implies \left[ \sigma = \sigma' = \varepsilon \wedge s = s' = \varepsilon \wedge u_1(q, \sigma) \in com_1 \times \{\varepsilon\} \wedge u_2(q, \sigma') \in com_2 \times \{\varepsilon\} \right] \end{aligned} \quad (6.2)$$

Solche Übergänge können nur ausgeführt werden, wenn auch beide Teilautomaten im Ausgangszustand waren. D.h., falls beispielsweise mit  $q \in Q_1 \cap Q_2$  und  $q' \in Q_1 \cap Q_2$  gilt  $\delta_1(q, \varepsilon) = (q', \varepsilon)$  und  $\delta_2(q, \varepsilon) = (q', \varepsilon)$  sowie  $u_1(q, \varepsilon) \in com_i \times \{\varepsilon\}$  und  $u_2(q, \varepsilon) \in com_2 \times \{\varepsilon\}$ , dann erfordert die Transition, dass  $q_{cur,1} = q_{cur,2} = q$ . In dem Fall würden sowohl  $q_{cur,1}$  als auch  $q_{cur,2}$  nach dem Updateschritt auf den Nachfolgerzustand gesetzt (also im Beispiel auf  $q'$ ), und die Updates  $u_1(q, \varepsilon)$  und  $u_2(q, \varepsilon)$  würden synchron ausgeführt werden. Im Gegenzug sollen Zustände in Multi-ERSA auch nur dann geteilt sein, wenn sie Start oder Ziel einer synchronen Kante sind, d.h.:

$$\begin{aligned} \forall q \in Q : q \in Q_1 \cap Q_2 \implies \exists q' \in Q_1 \cap Q_2 : \\ \left[ \delta_1(q, \varepsilon) = (q', \varepsilon) \wedge \delta_2(q, \varepsilon) = (q', \varepsilon) \right] \vee \left[ \delta_1(q', \varepsilon) = (q, \varepsilon) \wedge \delta_2(q', \varepsilon) = (q, \varepsilon) \right] \end{aligned} \quad (6.3)$$

### 6.2.3. Beispiel

In Abbildung 6.1 ist ein Teil eines Multi-ERSA als Beispiel angegeben, konkret mit  $k = 2$ , also für zwei Roboterarme. Der Anwendungsfall wäre hier, dass ein Roboterarm verschiedene ankommende Objekte identifiziert, wovon manche von beiden Armen gemeinsam bewegt werden müssen. Es passieren gemeinsame (synchrone) Operationen, d.h. die beiden Teilautomaten der Systemkomponenten überschneiden sich: Es gilt  $\{q_0, q_1, q_2, q_4\} \subset Q_0$  und  $\{q_2, q_3, q_4\} \subset Q_1$ . Neben der impliziten Synchronisation für die gemeinsame Kante



**Abbildung 6.1.:** Partieller Multi-ERSA am Beispiel für zwei Roboterarme. Transitionen sind wie üblich beschriftet, mit dem Zusatz, dass das Subskript am Doppelpunkt die betroffene Transitionsfunktion angibt (d.h. „:1“ markiert einen Übergang gemäß  $\delta_1$ ).

Links des Doppelpunkts steht die erforderliche Eingabe (ein  $\sigma \in \Sigma$  oder  $\varepsilon$ ) sowie das erforderliche Signal (ein  $b \in [m]$ ) falls zutreffend. Falls kein Signal erforderlich ist, wird der zweite Teil (d.h.  $\varepsilon$ ) weggelassen. Auf der rechten Seite steht das aus der Updatefunktion definierte Kommando, sowie ggfs. das zu inkrementierende Signal (das ansonsten, also im Fall  $\varepsilon$ , wiederum weggelassen wird). Zustände sind per Superskript markiert, zu welchen Teilautomaten sie gehören, d.h. ein Superskript von „1,2“ bedeutet etwa, dass der Zustand sowohl in  $Q_1$  als auch in  $Q_2$  enthalten ist.

von  $q_2$  nach  $q_4$  wird auch ein Signal mit Index  $b_0$  verwendet, das für eine explizite Präzedenz zwischen der Kante von  $q_0$  nach  $q_2$  und der von  $q_3$  nach  $q_2$  sorgt.

Vom Startzustand des ersten Roboters,  $q_0$ , gehen zwei Transitionen unter den Eingaben  $\sigma_0$  und  $\sigma_1$  aus. Beide definieren einen Übergang und ein Update nur für den ersten Roboter, passend dazu, dass mit  $q_0$  und  $q_1$  beide Enden der ersten Kante nicht in  $Q_2$  sind, und nur das Ende  $q_2$  der zweiten Kante, nicht aber deren Start  $q_0$ . Die zweite Kante, nach  $q_2$ , spezifiziert außerdem den zu inkrementierenden Signalindex  $b_0$ . Die einzige Kante aus dem Startzustand des zweiten Roboters,  $q_3$ , spezifiziert im Gegenzug das Signal mit Index  $b_0$  als zu dekrementieren, also besteht zwischen diesen beiden Kanten eine Präzedenzbeziehung. Erst nachdem die Kante des ersten Automaten verfolgt wurde, wird das Signal inkrementiert, und erst daraufhin kann der zweite Automat seine Transition ausführen. Konzeptionell modelliert dies etwa, dass der erste Roboter aus einer Wahrnehmungspose in seine Aufgreifpose verfährt und damit erst den Weg für den zweiten Roboter freimacht.

Die Kante zwischen  $q_2$  und  $q_4$  ist dann mit einem Übergang und Update sowohl für den ersten als auch für den zweiten Roboter beschriftet. Diese werden entsprechend erst ausgeführt, wenn beide Teilautomaten in  $q_2$  sind, und dann gemeinsam. Damit ist das gemeinsame Aufgreifen und Bewegen dargestellt. Aus  $q_4$  spalten sich die Teilautomaten dann wieder auf, mit getrennten Transitionen.

## 6.3. Programmierung im Gesamtsystem

Auch die Programmierung von Multi-ERSA wurde im existierenden System auf den Fall von  $k = 2$  Robotern und Teilautomaten beschränkt. Der Programmiervorgang stellt dabei wiederum eine Erweiterung des allgemeinen Vorgangs für ERSA dar. Die Roboter werden weiterhin kinästhetisch geführt, und einzelne Aktionen durch haptische Gesten angelegt (bei Bewegungen in der jeweiligen Zielpose). Hinzu kommen Gesten für die drei betrachteten, in Abschnitt 6.1 identifizierten Synchronisationsmechanismen: *Präzedenz* (unidirektionale oder asymmetrische Synchronisationspunkte), *Rendezvous* (symmetrische Synchronisationspunkte, auch als Barrier bezeichnet), und echt *synchrone* Transitionen oder Aktionen.

### 6.3.1. Operationen

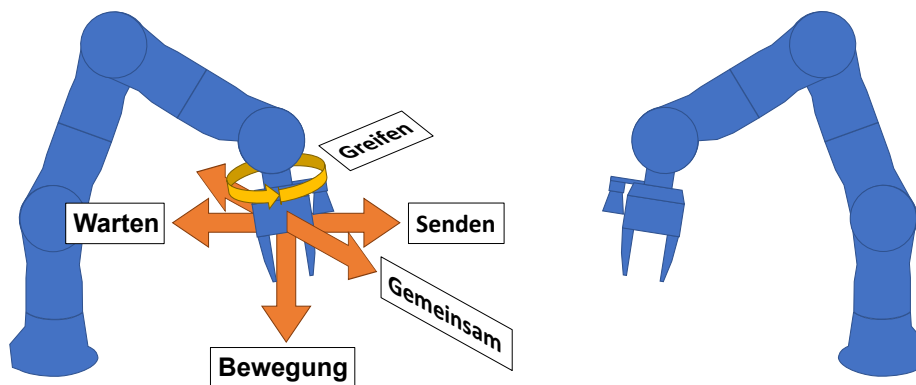
Für die Programmierung einer Präzedenzbeziehung zwischen Punkten in zwei Teilautomaten wurde ein Paar aus Operationen gewählt. Die erste Operation erzeugt dabei den *Sendepunkt* im ersten Teilautomaten, mit der Semantik, dass dieser Punkt in diesem Teilautomaten erreicht werden muss, *bevor* der korrespondierende Punkt im anderen Teilautomaten passiert werden kann. Die zweite Operation erzeugt den entsprechenden *Wartepunkt*, mit symmetrischer Semantik: Dieser Punkt im Teilautomaten darf erst passiert werden, *nachdem* der korrespondierende Sendepunkt im anderen Teilautomaten erreicht wurde. Die Modellierung von Präzedenz in Multi-ERSA geschieht durch Einführung einer neuen Komponente des Signalvektors (d.h. eines neuen Signals). Diese wird am Sendepunkt inkrementiert, und ist für die Transition am Wartepunkt Voraussetzung und wird dort dekrementiert.

Die Programmierung von Rendezvous besteht aus einer einzelnen Operation, die auf beiden Robotern ausgelöst werden muss. Diese erzeugt jeweils den konkreten Synchronisationspunkt, der wiederum erst passiert werden kann, wenn der Synchronisationspunkt im anderen Teilautomaten ebenfalls erreicht wurde. Rendezvous werden in Multi-ERSA entsprechend durch zwei neue Signale modelliert, von denen in jedem Teilautomaten zunächst eines gesetzt (inkrementiert) wird, und dann auf das andere gewartet (und dieses dekrementiert) wird.

Für synchrone Aktionen gibt es zuletzt eine weitere Operation, die ebenfalls auf beiden Robotern ausgelöst werden muss. Diese fasst die Aktionen beider Roboter zu einer synchronen Transition zusammen, die dann in beide Teilautomaten integriert wird. Hier findet die Synchronisation nicht über ein Signal statt, sondern implizit über die in Abschnitt 6.2 erwähnte Anforderung, dass alle im Updateschritt involvierten Komponenten im zugehörigen Vorgängerzustand sein müssen. Nach Ausführung der synchronen Aktion können beide Teilautomaten wieder unabhängig voneinander fortfahren.

### 6.3.2. Interaktionen

Mit dem erklärten Ziel, alle Interaktionen soweit möglich über den Roboter selbst abzudecken, wurden für die Synchronisationsoperationen weitere haptische Gesten gewählt.



**Abbildung 6.2.:** Übersicht über haptische Gesten im Rahmen der Synchronisation, wie in der Benutzerstudie (sh. Abschnitt 7.2) als Merkhilfe verwendet: Weiterhin Anpinngeste vertikal nach unten für (absolute) Bewegung, Anpinngeste in Richtung der anderen Roboterbasis für einen Sendepunkt, von der anderen Roboterbasis weg für einen Wartepunkt (Präzedenz oder Rendezvous), senkrecht zu den beiden anderen Achsen für synchrone Aktionen. (In der Benutzerstudie wurde außerdem der Greifer auf eine Drehbewegung um das letzte Gelenk des Roboters abgebildet.)

Dabei handelt es sich um Anpinngesten analog zu denen für absolute und relative Bewegung, nur in anderen Raumachsen. Entlang der Achse zwischen den Basen der beiden Roboter generiert eine Anpinngeste in Richtung des jeweils anderen Roboters (d.h.: kurze Bewegung parallel zur Achse zwischen den Roboterbasen in Richtung der des anderen Roboters, anschließend kurze Bewegung in entgegengesetzter Richtung) den Sendepunkt einer Präzedenz. Entlang derselben Achse, aber in entgegengesetzter Richtung, wird entweder ein Wartepunkt als andere Hälfte einer Präzedenz, oder ein symmetrischer Synchronisationspunkt als einer von zwei in einem Rendezvous. Zu guter Letzt dient eine Anpinngeste in der letzten verbleibenden Raumachse (d.h. senkrecht sowohl zur Vertikalen als auch zur Achse zwischen den Roboterbasen) zum Auslösen der Operation für synchrone Aktionen. Dabei werden beide Richtungen gleichermaßen akzeptiert.<sup>3</sup> Eine Skizze der verschiedenen Achsen ist in Abbildung 6.2 gegeben.

Haptische Interaktionen haben im Zusammenhang mit der Multi-Roboter-Programmierung den zusätzlichen Vorteil, dass durch die Eingabemodalität direkt klar ist, welchen Roboter eine gerade programmierte Aktion adressiert (eben den, über den die Eingabe stattgefunden hat). Das ist für andere Eingabemodalitäten nicht zwingend automatisch der Fall; Man denke z.B. an eine Eingabe von Posen über ein Tablet oder einen Desktop-Computer unabhängig von den Robotern.

Zu bemerken ist, dass der hier beschriebene Ansatz zur haptischen Eingabe der Synchronisationsoperationen explizit für den Fall zweier Roboterarme entwickelt wurde. Eine naive Verallgemeinerung auf mehr oder andere Roboter wirft die Schwierigkeit auf, dass z.B. keine wohldefinierte einzelne Achse zwischen den Basen von drei oder

<sup>3</sup>Die Bewegungen beider Roboter in entgegengesetzte Richtungen auszuführen, erlaubt eine Programmierung ohne größere Bewegung eines gemeinsam gehandhabten Objekts, sondern gewissermaßen mittels einer Drehung um dessen Zentrum.

mehr Robotern existiert. Es müsste in dem Fall auch schon vorher evaluiert werden, ob die Rendezvous-Operation (für symmetrische Synchronisation zwischen allen Robotern) beibehalten würde, oder etwa Varianten wie paarweise symmetrische Synchronisation wünschenswert wären. All das wäre im Automatenmodell umsetzbar, aber die Programmierung verallgemeinert sich nicht trivial aus dem Fall  $k = 2$ .

---

7.1. ERSA . . . . .	92
7.1.1. Studienaufbau . . . . .	92
7.1.2. Programmieraufgabe . . . . .	92
7.1.3. Erhobene Daten . . . . .	95
7.1.4. Ergebnisse . . . . .	96
7.2. Multi-ERSA . . . . .	98
7.2.1. Studienaufbau . . . . .	98
7.2.2. Programmieraufgaben . . . . .	100
7.2.3. Ergebnisse . . . . .	102
7.3. Haptische Interaktionen . . . . .	104
7.4. Vergleichsstudie mit grafischer Schnittstelle . . . . .	106
7.4.1. Studienaufbau . . . . .	107
7.4.2. Ergebnisse . . . . .	107
7.5. Bemerkungen zu statistischer Signifikanz . . . . .	110

---

In diesem Kapitel werden die verschiedenen Benutzerstudien beschrieben, die zur experimentellen Auswertung der vorgestellten Ansätze und Forschungsfragen durchgeführt wurden. Die erste davon betrifft das ERSA-System an sich, auf einer einzelnen Programmieraufgabe, die das Modell möglichst umfassend abdeckt. Die zweite Studie betrachtet den Fall mehrerer Roboter mit dem Multi-ERSA-System, und die dafür bereitgestellten Synchronisationsmechanismen. Im Rahmen der zweiten Studie wurde auch der Einfluss haptischer Interaktionen auf die Benutzbarkeit evaluiert, was anschließend dargestellt ist. Zuletzt folgen die Ergebnisse einer Vergleichsstudie mit der existierenden grafischen Schnittstelle eines Robotersystems.

Die in dieser Arbeit erfassten Teilnehmerzahlen sind nicht groß genug, um statistische Signifikanz zu erzielen, daher findet die Diskussion zunächst stets nur unter Bezug auf die konkrete Stichprobe statt. Eine kurze Einordnung zur statistischen Signifikanz der Ergebnisse findet sich zu Ende des Kapitels in Abschnitt 7.5.

Die ERSA-Studie (d.h. deren Beschreibung, Ergebnissen und Auswertung) wurde in ihrer ursprünglichen Form in [179] veröffentlicht, die Multi-ERSA-Studie in [181].

### 7.1. ERSA

Die erste Benutzerstudie sollte die Nutzbarkeit des automatenbasierten Ansatzes ohne grafische Schnittstelle an sich auswerten. Neben den Ergebnissen über alle Teilnehmenden ist auch interessant zu betrachten, inwieweit sich die Bewertung durch Experten und Nichtexperten unterscheidet.

#### 7.1.1. Studienaufbau

Mit dem grundlegenden System zur Programmierung auf Basis von erweiterten Roboterzustandsautomaten wurde eine Benutzerstudie durchgeführt, um den Ansatz zu verifizieren. Der verwendete Aufbau war dabei wie in Abbildung 7.1 links abgebildet. Es wurde ein Kuka LBR IV verwendet, mit einem Robotiq 3-Finger-Greifer als Werkzeug und einer Intel RealSense D435 RGB-D-Kamera am Greiferflansch. Außerdem wurde ein Tastenblock in Form einer MAX Falcon 8 Minitastatur am Greifer befestigt, über den weitere Interaktionen abgedeckt wurden (Speichern/Laden des Programms, Ausführen, Sprung auf Anfang, Greifer Operieren, Kamera Auslösen, Bezug zu Objekt). Auf dem Greifer wurden auch Piktogramme zur Erinnerung an die haptischen Interaktionen angebracht, wie in Abbildung 7.1 auf der rechten Seite dargestellt.<sup>1</sup>

Eine Teilnahme an der Benutzerstudie lief dabei wie folgt ab: Zu Beginn wurde eine allgemeine Einführung in das Thema und die Studie gegeben (inklusive Einwilligung nach erfolgter Aufklärung). Anschließend wurden in einem kurzen Tutorial die Programmiermöglichkeiten im System mit den jeweils zugehörigen Interaktionen vorgestellt. Teilnehmende hatten dabei Gelegenheit, die einzelnen Schritte (insbesondere die haptischen Interaktionen) selbst auszuprobieren. Im eigentlichen Programmiereteil wurde dann eine einzelne Aufgabe gestellt, die dazu entworfen war, alle Aspekte des Systems anzuschneiden. Eine einzelne Teilnahme nahm ca. 60 Minuten in Anspruch. In dieser Studie wurde keine Aufwandsentschädigung ausgezahlt.

#### 7.1.2. Programmieraufgabe

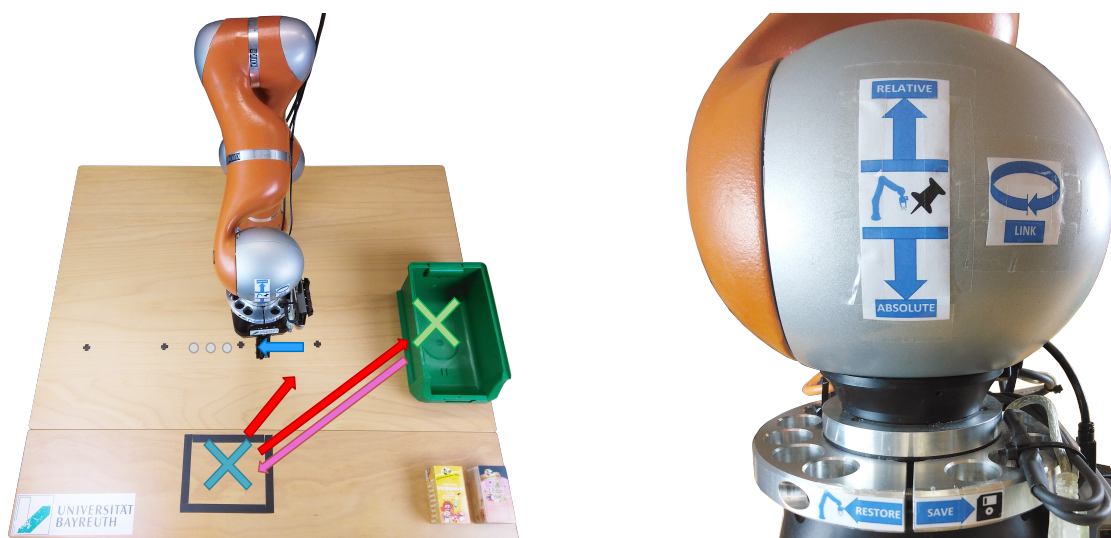
Die gestellte Programmieraufgabe bestand darin, zwei unterschiedliche Arten ankommender Objekte (gelbe und dunkle/fliederfarbene Teeschachteln) zu verarbeiten. Gelbe Teeschachteln sollten in einen Behälter eingetaucht werden (in Anlehnung an z.B. einen Beschichtungsschritt) und anschließend wieder in genau die Position zurückgelegt werden, wo sie aufgenommen wurden. Dunkle Teeschachteln sollten dagegen an der ersten freien Stelle in einer Reihe von markierten Ablagepositionen platziert werden.

Ein möglicher (vereinfachter) Automat, um diese Aufgabenbeschreibung umzusetzen,

---

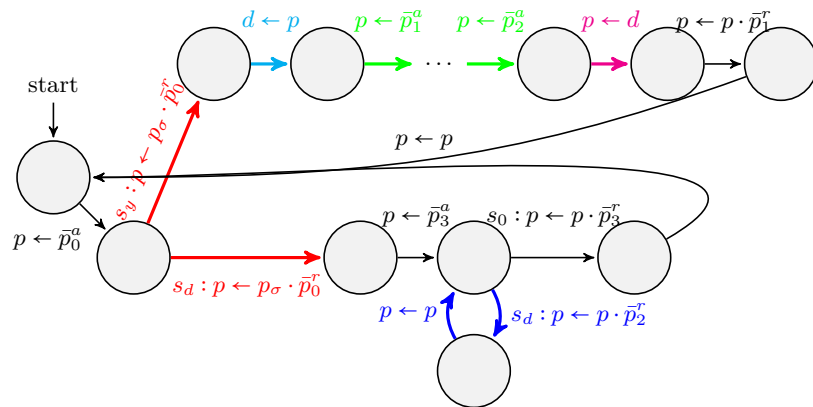
<sup>1</sup>Aufgrund der hohen Unsicherheit im Umgang mit haptischen Interaktionen wurde zusätzlich bei erkannten Gesten ein einfaches Audiosignal ausgegeben.





**Abbildung 7.1.:** Links: experimenteller Aufbau, mit angedeuteten Schritten analog zu Abbildung 7.2 eingefärbt. Vor dem Roboter werden Objekte im schwarz umrandeten Bereich abgelegt. Je nach Objektart werden diese entweder in den Behälter rechts eingetaucht und zurückgelegt, oder an den mit einem schwarzen Punkt markierten Ablageorten aufgereiht.

Rechts: Letzte Segmente des Roboters mit den darauf angebrachten Piktogrammen (sh. Abbildung 4.11) zur Erinnerung an die in Kapitel 4 beschriebenen haptischen Interaktionen (Ruckbewegungen nach unten/oben für absolute/relative Bewegungen, horizontale Kreisbewegung zum Verbinden, Ruckbewegungen nach rechts/links im letzten Gelenk zum Speichern/Laden von Posensvariablen).



**Abbildung 7.2.:** Vereinfachter Automat für die Aufgabe aus der ERSA-Studie. Kanten, die mit „ $\sigma$ “ beschriftet sind, werden nur dann ausgeführt, wenn ein entsprechender Stimulus aus der Objektwahrnehmung auftritt. Für die einzige auftretende Posenvariable wird  $d$  verwendet (statt der entsprechenden Komponente  $d_1$ ). Die jeweiligen Updates sind ähnlich einer Pseudocode-Zuweisung notiert, z.B. „ $d \leftarrow p$ “ für das Abspeichern der aktuellen Roboterpose in der Posenvariable.

Die einzelnen Programmiermöglichkeiten aus der Automatendefinition sind abgedeckt: Verzweigung und Bewegung relativ zur aktuellen Pose des wahrgenommenen Objekts zum Aufnehmen (rot), Speichern der Aufnahmepose für gelbe Teeschachteln (oberer Pfad, hellblau), Eintauchen in den Behälter mit absoluten Bewegungen (grün) und Anfahren der gespeicherten Pose aus der Posenvariablen (magenta), Schleife über Ablagepositionen mit relativen Bewegungen für fliederfarbene Teeschachteln (unterer Pfad, dunkelblau).

ist in Abbildung 7.2 gegeben. Wie oben erwähnt, hatte diese Aufgabe den Zweck, alle Programmiermöglichkeiten im ERSA-System abzufragen:

- Verzweigungen auf Basis der Kamerawahrnehmung
- Das Aufnehmen von Objekten an ihrer jeweils aktuellen Pose durch objektrelative Bewegungen
- Das Speichern und spätere Anfahren einer a priori unbekanntenen Pose mit der Aufgreifpose im Zweig für gelbe Teeschachteln
- Relative Bewegungen in der Schleife innerhalb des Zweigs für dunkle Teeschachteln, um die Bewegung von einer markierten Ablageposition zur nächsten umzusetzen

### 7.1.3. Erhobene Daten

In der Studie wurde zur Auswertung der Effektivität und Intuitivität des Systems auf die Fragebögen aus dem MINERIC-Toolkit [113] zurückgegriffen. Konkret wurden die Teilnehmenden vor der Studie nach ihrer Vorerfahrung im Bereich der Programmierung im Allgemeinen und im Umgang mit Robotern im Speziellen, sowie nach ihrer a-priori-Einschätzung der Komplexität des Systems (COM-E) befragt. Nach der anschließenden Einführung in das Programmiersystem und der Bearbeitung der eigentlichen Aufgabe wurde dann ihre a-posteriori-Einschätzung der Komplexität (COM-C) ermittelt, ihre wahrgenommene subjektive mentale Belastung (SSEE), und es wurde ihnen der QUESTI-Fragebogen vorgelegt, in dem über die Zustimmung zu natürlichsprachlichen Aussagen verschiedene Aspekte der Programmiererfahrung abgefragt werden.<sup>2</sup> Parallel wurde seitens des Versuchsleiters das PAC-U-Formular betreffend des Programmiererfolgs ausgefüllt.

Der Wertebereich für PAC-U geht von 0 bis 5, wobei 0 einer vollständig erfolgreichen Programmierung der Aufgabe ohne Hinweise entspricht, und die Abweichung von erfolgreicher Programmierung mit bis zu 3 Punkten bewertet wird, sowie die Abweichung durch Anforderung von Hinweisen mit bis zu 2 Punkten. Der SSEE-Wert gibt die subjektive mentale Anstrengung bei Benutzenden auf einer Skala von 0 bis 220 an. Der Wert 0 entspricht dabei gar keiner mentalen Anstrengung, der Wert 220 extremer mentaler Belastung. Die Skala ist zur Orientierung natürlichsprachlich annotiert. Die Komplexitätswerte COM reichen von -5 für „sehr kompliziert“ bis +5 für „sehr einfach“. Zuletzt besteht der QUESTI-Fragebogen aus 14 positiven Aussagen zur Benutzbarkeit, die jeweils auf einer Likert-Skala von 1 („trifft gar nicht zu“) bis 5 („trifft völlig zu“) bewertet werden. Diese lassen sich weiterhin in verschiedene Teilmengen gruppieren, die verschiedene Aspekte des Systems abfragen: niedrige mentale Belastung (Fragengruppe W für *Workload*), hohe Erfüllung der eigenen Ziele (G für *Goals*), niedrigen Lernaufwand (L für *Learning*), hohe Vertrautheit mit dem System (F für *Familiarity*) und niedrige Fehlerrate (E für *Error Rate*).

<sup>2</sup>In den Versuchsdaten in Anhang A ist statt COM-E jeweils COM-D gegeben, die Differenz zwischen COM-C und COM-E.

Die verwendeten, deutschsprachigen Fragebögen sind in Anhang C reproduziert. Mehr Details zum Design und der Verwendung des Toolkits finden sich in der Originalpublikation [113] oder werden in [131] ausführlicher diskutiert.

### 7.1.4. Ergebnisse

Die Studie wurde ursprünglich im Oktober 2020 durchgeführt. Durch Einschränkungen aufgrund der zu dem Zeitpunkt grassierenden COVID-19-Pandemie wurde nur eine kleine Menge von sieben Teilnehmenden erfasst. Zwei weitere Datensätze wurden parallel zu einer späteren Studie nachgelagert im Dezember 2021 erhoben. Insgesamt setzten sich die Teilnahmen damit aus vier von Robotikexperten und fünf von Nichtexperten zusammen. In Tabelle A.1 in Anhang A sind die gewonnenen Rohdaten aufgelistet.

Die Ergebnisse der Studie sind als Kastengrafiken (engl. *Box-Whisker-Plots*)<sup>3</sup> in Abbildung 7.3 dargestellt. Anhand der kleinen Teilnehmezahlen wird auf detailliertere statistische Auswertung verzichtet, und es können nur Aussagen zu groben Tendenzen in den beobachteten, beschränkten Stichproben gemacht werden. Für weitere Bemerkungen zur statistischen Signifikanz der Ergebnisse, siehe Abschnitt 7.5. Sofern im Folgenden neben den Diagrammen konkrete Werte angegeben sind, handelt es sich dabei um (empirische) Mittelwerte und Standardabweichungen.

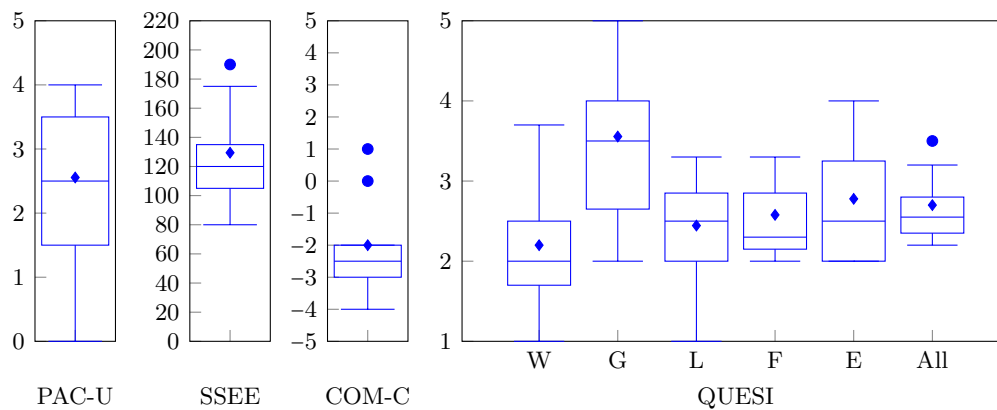
Die Auswertung beschäftigt sich primär mit den zwei oben genannten Aspekten: der Effektivität und der Intuitivität der Programmierung. In puncto Effektivität kann dazu der PAC-U-Wert betrachtet werden. Der erzielte Mittelwert von 2.6 ( $\sigma = 1.4$ ) entspricht dabei überwiegend erfolgreichen Programmierprozessen, aber mit regelmäßigem Rückfragebedarf an den Versuchsleiter. Auch aus den QUESI-Werten lassen sich einzelne Aspekte isolieren, wie etwas die Fragenteilmenge G betreffend der Erfüllung der eigenen Ziele. Der erzielte Wert von 3.6 ( $\sigma = 1.0$ ) zeigt dabei, dass die Teilnehmenden diesen (positiven) Aussagen generell eher zustimmten.

Zur Intuitivität fallen die Ergebnisse weniger positiv aus. Der Mittelwert des COM-C (wahrgenommene Komplexität) ist  $-2$  ( $\sigma = 1.6$ ), unter Null und damit auf der komplexen Seite der gegebenen Skala. Teilnehmende fanden das System also generell eher kompliziert zu benutzen. Der SSEE-Wert von 129.4 ( $\sigma = 36.3$ ) fällt natürlichsprachlich zwischen „ziemlich anstrengend“ und „stark anstrengend“. Weiterhin sind die drei QUESI-Unterwerte zur mentalen Belastung (W), zum Lernaufwand (L) und zur Vertrautheit mit dem System (F) alle unterhalb des neutralen Niveaus von 3. In anderen Worten: Die Teilnehmenden stimmten den positiven Aussagen zu diesen drei Aspekten tendenziell eher nicht zu.

Eine Unterscheidung der Teilnehmenden in Robotikexperten und -nichtexperten hilft, das Bild weiter auszudifferenzieren. Die entsprechenden Diagramme sind in Abbildung 7.4 gegeben. Der PAC-U-Durchschnitt ist dabei fast gleich, mit geringerer Varianz auf Seiten der Experten (Experten:  $\bar{\varnothing} = 2.5$ ,  $\sigma = 1.3$ , Nichtexperten:  $\bar{\varnothing} = 2.6$ ,  $\sigma = 1.7$ ). Die mentale Belastung wurde interessanterweise von Experten im SSEE-Wert höher angesetzt als von Nichtexperten (Experten:  $\bar{\varnothing} = 151.3$ ,  $\sigma = 44.0$ , Nichtexperten:  $\bar{\varnothing} = 112.0$ ,  $\sigma = 17.9$ ). Auch

---

<sup>3</sup>Für Details zu dieser Darstellungsform siehe z.B. [67].



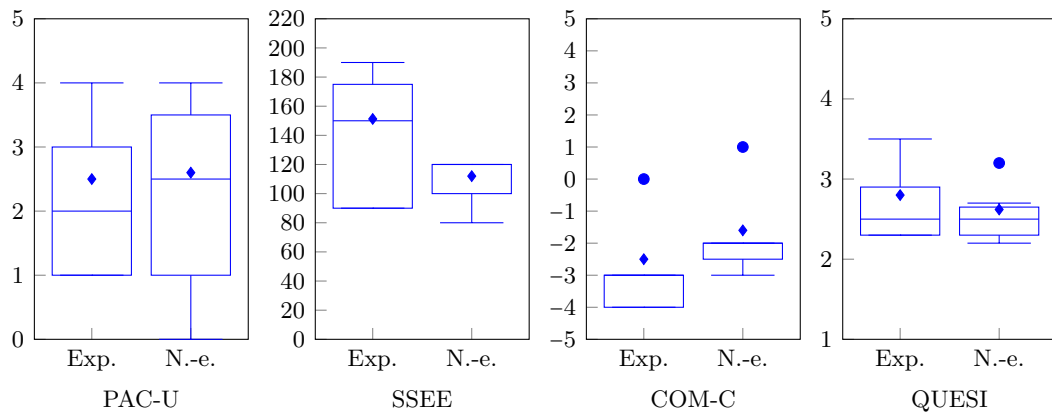
**Abbildung 7.3.:** Ergebnisse aus der Studie zum ERSA-System.

Von links nach rechts: PAC-U-Wert zum Erfolg der Programmierung (Abstand vom Optimum, d.h. niedriger ist besser), SSEE-Wert zur mentalen Belastung (niedriger ist besser), wahrgenommene Komplexität COM-C (positiv bedeutet einfach, negativ bedeutet komplex), und Zustimmung zu den positiven QUESI-Aussagen (höher ist besser).

die wahrgenommene Komplexität COM-C war bei Experten im Durchschnitt kleiner, also komplexer auf der Skala (Experten:  $\bar{x} = -2.5$ ,  $\sigma = 1.7$ , Nichtexperten:  $\bar{x} = -1.6$ ,  $\sigma = 1.5$ ). Die QUESI-Durchschnittswerte sind dagegen wieder ähnlich, tendenziell leicht besser unter der Gruppe der Experten (Experten:  $\bar{x} = 2.8$ ,  $\sigma = 0.5$ , Nichtexperten:  $\bar{x} = 2.6$ ,  $\sigma = 0.4$ ). Experten fanden das System also tendenziell komplexer als Nichtexperten, stuften seine Benutzbarkeit aber etwas besser ein. Dies deutet darauf hin, dass die Bewertungsskalen, etwa für Komplexität, zwischen den verschiedenen Teilnehmenden und Gruppen nicht exakt übereinstimmen.

Ebenfalls interessant sind die verschiedenen Teilgruppen der QUESI-Fragen. Dabei wurde von Experten etwa der Aspekt der Zielerfüllung deutlich besser bewertet als von Nichtexperten (Experten:  $\bar{x} = 4.4$ ,  $\sigma = 0.5$ , Nichtexperten:  $\bar{x} = 2.9$ ,  $\sigma = 0.7$ ). Lernaufwand (Experten:  $\bar{x} = 2.1$ ,  $\sigma = 0.8$ , Nichtexperten:  $\bar{x} = 2.7$ ,  $\sigma = 0.5$ ) und Vertrautheit mit dem System (Experten:  $\bar{x} = 2.4$ ,  $\sigma = 0.6$ , Nichtexperten:  $\bar{x} = 2.7$ ,  $\sigma = 0.4$ ) fielen dagegen unter Nichtexperten leicht besser aus. Bei der Fehlerrate war die Wahrnehmung der Experten wiederum positiver als die der Nichtexperten (Experten:  $\bar{x} = 3.0$ ,  $\sigma = 0.7$ , Nichtexperten:  $\bar{x} = 2.6$ ,  $\sigma = 0.8$ ).

Zusammenfassend lässt sich sagen, dass sowohl Experten als auch Nichtexperten tendenziell in der Lage waren, die gestellte Aufgabe mit dem System zu lösen, allerdings mit recht häufigem Rückfragebedarf. Im Vergleich zur Benutzbarkeit ist die Intuitivität also als noch ausbaufähig zu bewerten. Betrachtet man Experten und Nichtexperten isoliert, zeigt sich, dass Experten ihre eigenen Ziele mit dem System besser erreichen konnten, aber auf Kosten von höherer mentaler Belastung. Auch bewerteten sie das System als komplexer, möglicherweise durch den Konflikt zu existierenden Vorerfahrungen, der bei Nichtexperten nicht gegeben ist.



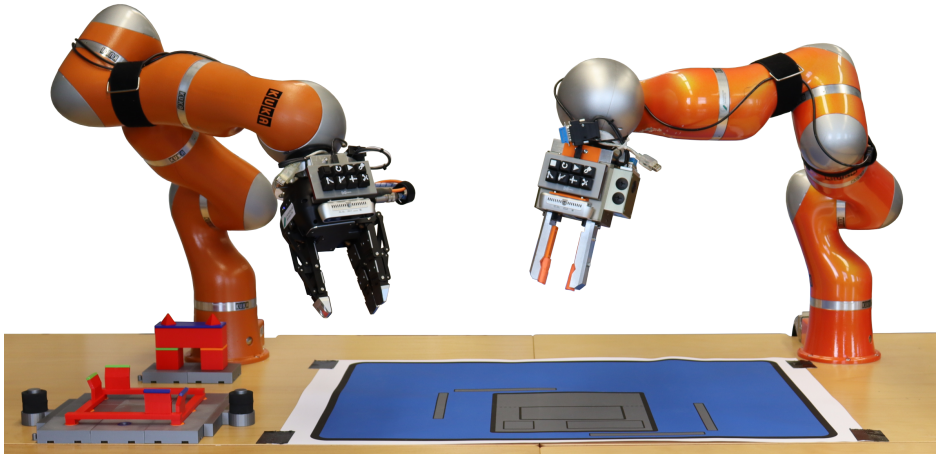
**Abbildung 7.4.:** Ergebnisse mit dem ERSA-System für PAC-U, SSEE, COM-C und QUESI-Durchschnitt, aufgeteilt in Robotikexperten („Exp.“) und Robotiknichtexperten („N.-e.“).

## 7.2. Multi-ERSA

In einer weiteren Benutzerstudie wurde evaluiert, inwieweit die Synchronisationsaktionen für Experten und Nichtexperten nutzbar sind. In derselben Studie wurden auch Daten zur Auswertung der haptischen Eingaben erhoben, wie in Kapitel 4 referenziert.

### 7.2.1. Studienaufbau

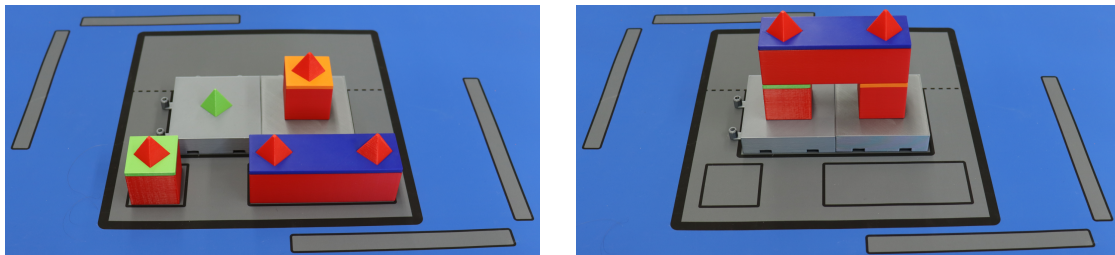
Der Studienaufbau ist in Abbildung 7.5 abgebildet. Dieser bestand aus zwei KUKA-Robotern (LWR4 und LWR4+), jeweils mit einem Greifer (Robotiq adaptiver 3-Finger-Greifer und Schunk PG70), sowie mit einer MAX Falcon 8 Minitastatur ausgestattet. Letztere stellte einerseits Operationen zur Navigation innerhalb des Systems (z.B. Sprung an den Anfang des Programms, schrittweise Ausführung, Speichern und Laden des Programms) zur Verfügung. Andererseits diente diese auch der Kontrollgruppe zur Auswertung der haptischen Interaktionen. Die Synchronisationsoperationen wurden zu diesem Zweck auf Tasten gelegt, deren Symbole in Abbildung 7.6 dargestellt sind. Eine gedruckte Matte mit Umrissen für Aufnehm- und Ablagestellen diente zur Orientierung für die Teilnehmenden. Beide Roboter waren jeweils mit einem korrespondierenden PC verbunden. Der Ablauf der Studie bestand zunächst aus einer initialen Einweisung (inklusive Einwilligung nach erfolgter Aufklärung). Anschließend führte ein kurzes Tutorial durch die einzelnen Operationen und die zugehörigen Gesten, bevor die zwei eigentlichen Programmieraufgaben gestellt wurden. Auch in dieser Studie wurden den Teilnehmenden Fragebögen aus dem MINERIC-Toolkit [113] vorgelegt (beschränkt auf die mit Bezug zur Programmierung, d.h. COM, SSEE und QUESI, während der Versuchsleiter PAC-U-Fragebögen ausfüllte). Die Teilnahme dauerte insgesamt ca. 60 Minuten, und eine Aufwandsentschädigung in Höhe von 10€ wurde ausgezahlt.



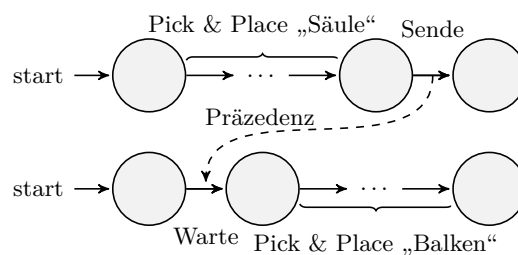
**Abbildung 7.5.:** Experimenteller Aufbau zur Evaluation des Multi-ERSA-Systems, bestehend aus zwei Leichtbaurobotern, einer bedruckten Matte mit den relevanten Ablageorten, und den verwendeten Objekten auf der linken Seite. Die Roboter sind jeweils mit einem Greifer und einem Tastenblock ausgestattet.



**Abbildung 7.6.:** Tastensymbole für Synchronisationsoperationen, von links nach rechts mit zugrunde liegender Assoziation: Sendepunkt (Senden eines Signals), Wartepunkt (Warten auf ein Signal), Rendezvous (Erstellen einer Synchronisationsbarriere), und synchrone Aktion (gemeinsame Ausführung/Kante).



**Abbildung 7.7.:** Ausgangs- und Zielzustand der ersten Programmieraufgabe. Ein Bogen muss mithilfe der beiden neben der Basisplatte verbleibenden Blöcke aufgebaut werden, in korrekter Reihenfolge.

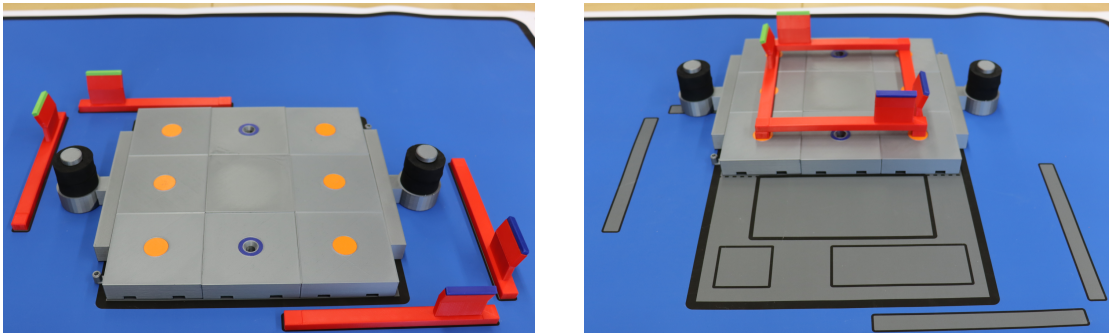


**Abbildung 7.8.:** Vereinfachter Automat für Aufgabe 1. Im oberen Teil des Automaten wird vom ersten Roboter der zweite kleinere Bauklotz (in Abbildung 7.7 grün, vorne links) neben dem bereits vorhandenen abgelegt und anschließend ein Signal gesendet. Im unteren Automatenenteil wartet der zweite Roboter zunächst auf dieses Signal, d.h. bis die zweite „Säule“ platziert wurde und der Arbeitsraum wieder frei ist. Erst danach kann der „Balken“, der größere Bauklotz, auf den beiden kleineren abgelegt werden.

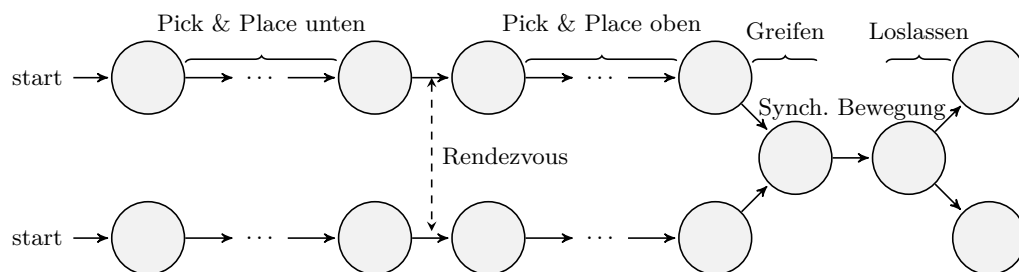
### 7.2.2. Programmieraufgaben

Die erste Programmieraufgabe bestand darin, einen Bogen aus drei kleinen Bauklötzen aufzubauen. Dabei befand sich ein Block bereits an der korrekten Position, während die anderen beiden jeweils von einem der Roboter platziert werden mussten. Für einen korrekten Bauprozess musste gewährleistet werden, dass der zweite kleinere Block (die zweite „Säule“) vor dem größeren oberen Block (dem „Balken“) platziert wurde. Dies erforderte die Kodierung einer Präzedenz im Programm. Abbildung 7.7 zeigt den Ausgangs- und Zielzustand der ersten Aufgabe. In Abbildung 7.8 wird ein vereinfachter Automat für die Aufgabe wiedergegeben, in dem insbesondere die erzeugte Präzedenz dargestellt ist. Die zweite Programmieraufgabe bestand darin, zunächst vier flache Verbindungselemente in einem Quadrat auf der Basisplatte zu platzieren. Auch hier sollte jedes Verbindungselement von einem bestimmten Roboter gehandhabt werden, und eine korrekte Platzierreihenfolge musste sichergestellt werden. Anschließend sollte die Platte an den beiden Knäufen aufgenommen und nach hinten zur entsprechenden Markierung bewegt werden. Um die Platte mit beiden Robotern gemeinsam erfolgreich zu bewegen, war





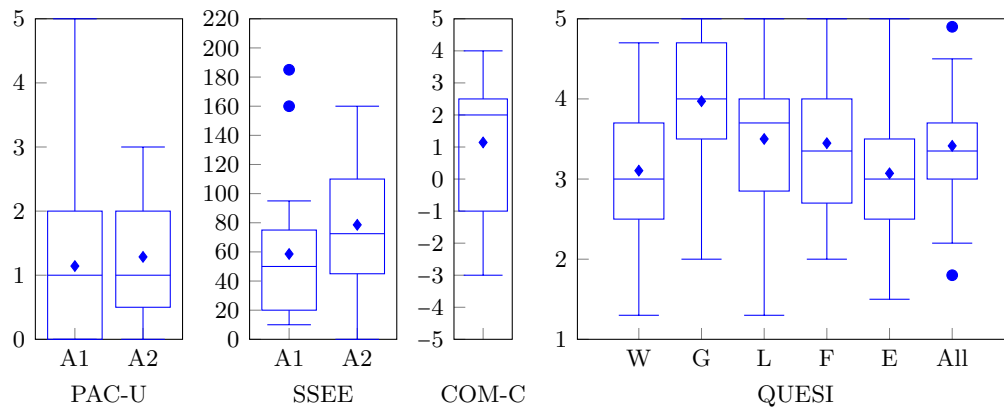
**Abbildung 7.9.:** Ausgangs- und Zielzustand der zweiten Programmieraufgabe. Zunächst muss in korrekter Reihenfolge ein Quadrat aus den vier flachen Verbindungselementen gelegt, anschließend die gesamte Platte von beiden Robotern gemeinsam nach hinten bewegt werden.



**Abbildung 7.10.:** Vereinfachter Automat für Aufgabe 2. Zunächst wird von beiden Robotern unabhängig voneinander jeweils ein Verbindungselement in der unteren Ebene platziert, auf gegenüberliegenden Seiten des Quadrats. Es folgt ein Rendezvous, um sicherzustellen, dass die untere Ebene abgeschlossen ist, bevor Verbindungselemente in der oberen Ebene platziert werden dürfen. Nach dem Rendezvous wird dieser Schritt wieder von beiden Robotern unabhängig voneinander ausgeführt. Zuletzt greifen beide Roboter gemeinsam die Basisplatte, bewegen sie entlang einer gemeinsamen Kante im Automaten durch eine synchrone Bewegung nach hinten und legen sie dort wieder ab.

eine synchrone Aktion notwendig. Die Konstruktion des Quadrats konnte idealerweise mit Rendezvous zwischen den einzelnen Schritten gelöst werden, sodass beide Roboter parallel zunächst die beiden unteren Verbindungselemente platzieren konnten, dann wiederum parallel die beiden oberen Verbindungselemente. Es war jedoch auch möglich, den Vorgang über Präzedenzen rein sequenziell abzusichern. Ausgangs- und Zielzustand der zweiten Aufgabe sind in Abbildung 7.9 dargestellt. Ein Automat mit den entsprechenden Synchronisationsschritten ist in Abbildung 7.10 gegeben.

Die verwendeten Objekte sind aus [130] entnommen, teilweise direkt (Elemente der Basisplatten, Bauklötze), teilweise mit Modifikationen (Verbindungselemente, verlängert und zum Greifen angeschrägt).



**Abbildung 7.11.:** MINERIC-Ergebnisse über alle Teilnehmenden. A1 und A2 bezeichnen die erste und zweite Programmieraufgabe.

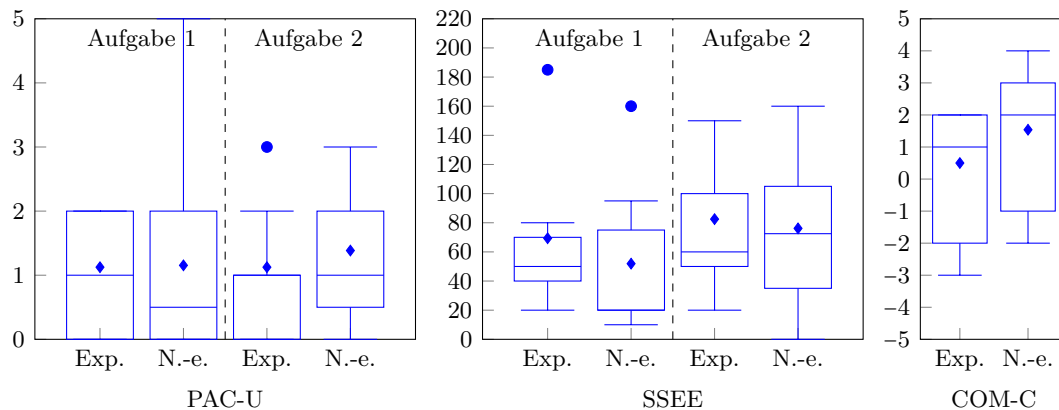
### 7.2.3. Ergebnisse

Die Studie wurde insgesamt mit 21 Teilnehmenden durchgeführt. Diese setzten sich zusammen aus 8 Robotikexperten und 13 Robotiknichtexperten. Darauf werden im Folgenden die allgemeinen Resultate zur Benutzbarkeit evaluiert. Für eine Auswertung unter Gesichtspunkten der haptischen Eingaben, siehe Abschnitt 7.3. Die kompletten erhobenen Rohdaten finden sich in Anhang A in Tabelle A.3.

Abbildung 7.11 zeigt die allgemeinen Ergebnisse der MINERIC-Fragebögen, wiederum als Kastengrafiken. In beiden Aufgaben waren die Nutzenden etwa gleich erfolgreich, mit PAC-U-Durchschnittswerten von 1.1 für Aufgabe 1 ( $\sigma = 1.3$ ) und 1.3 für Aufgabe 2 ( $\sigma = 1.1$ ). Der SSEE-Mittelwert von 58.6 ( $\sigma = 46.1$ ) bei der ersten Aufgabe landet in der natürlichsprachlichen Annotation der Skala zwischen „etwas anstrengend“ und „einigermaßen anstrengend“, während der Mittelwert von 78.6 ( $\sigma = 45.4$ ) bei Aufgabe 2 „einigermaßen anstrengend“ entspricht. Im Durchschnitt bewerteten die Teilnehmenden das System mit einem COM-C-Wert von 1.1 ( $\sigma = 2.2$ ) als eher simpel als kompliziert, wobei die Werte zwischen einzelnen Teilnehmenden stark variierten. Zuletzt ist der QUESTI-Mittelwert über alle Fragen und Teilnehmenden hier 3.4 ( $\sigma = 0.7$ ), also leicht besser als neutral.

Diagramme unter einer Aufteilung der Teilnehmenden in Robotikexperten und Robotiknichtexperten für die PAC-U-, SSEE- und COM-C-Werte sind in Abbildung 7.12 angegeben. Unter den Experten waren die PAC-U-Durchschnittswerte beider Aufgaben höher als unter Nichtexperten (Aufgabe 1: Experten  $\bar{x} = 1.1$ ,  $\sigma = 1.0$ , Nichtexperten  $\bar{x} = 1.2$ ,  $\sigma = 1.5$ ; Aufgabe 2: Experten  $\bar{x} = 1.1$ ,  $\sigma = 1.0$ , Nichtexperten  $\bar{x} = 1.4$ ,  $\sigma = 1.1$ ). D.h., Experten waren im Durchschnitt erfolgreicher als Nichtexperten.

Interessanterweise sind die Werte für SSEE (d.h. mentale Anstrengung) für Nichtexperten etwas niedriger als für Experten, auch hier bei beiden Aufgaben (Aufgabe 1: Experten  $\bar{x} = 69.4$ ,  $\sigma = 50.2$ , Nichtexperten  $\bar{x} = 51.9$ ,  $\sigma = 44.1$ ; Aufgabe 2: Experten  $\bar{x} = 82.5$ ,  $\sigma = 46.0$ , Nichtexperten  $\bar{x} = 76.2$ ,  $\sigma = 46.7$ ). Dies ließe sich möglicherweise dadurch



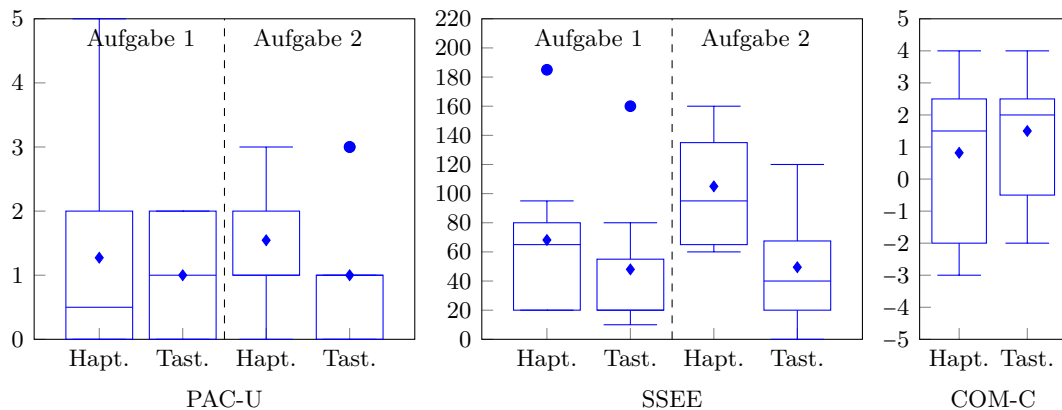
**Abbildung 7.12.:** Ergebnisse für PAC-U, SSEE und COM-C, aufgeteilt in Robotikexperten und Robotiknichtexperten.

erklären, dass Nichtexperten weniger kritisch gegenüber ihrer eigenen Lösung sind. Da die SSEE-Werte erst erhoben wurden, nachdem die Teilnehmenden die Ausführung ihres Programms beobachten konnten und die korrekte Lösung mit dem Versuchsleiter besprechen konnten, hatten trotz des durchschnittlich geringeren Erfolgs bei Nichtexperten beide Gruppen gleichermaßen ein Verständnis dafür, wie die Aufgaben korrekt zu lösen waren.

Der COM-C-Wert unter Experten ( $\varnothing = 0.5$ ,  $\sigma = 2.0$ ) ist niedriger als unter Nichtexperten ( $\varnothing = 1.5$ ,  $\sigma = 2.2$ ), d.h., ähnlich zu den SSEE-Werten wurde das System von Experten im Durchschnitt als komplizierter bewertet als unter Nichtexperten. Auch hier sind möglicherweise unterschiedliche Erwartungshaltungen eine plausible Erklärung. Experten, die bereits in der Lage sind, Roboter auf anderem Wege zu programmieren, nehmen das neue System eventuell als komplizierter als ein ihnen bekanntes wahr, während Nichtexperten tendenziell positiv überrascht sind, die Roboter überhaupt erfolgreich programmieren zu können. Unabhängig von der genauen Erklärung spricht das Auftreten dieser Art von Ergebnis sowohl in den mehreren Studien dieser Arbeit als auch in anderen Arbeiten wie z.B. [77, 170] zumindest stark dafür, dass es sich um einen tatsächlichen Effekt und nicht um ein Artefakt in den Daten handelt.

Der QUESI-Durchschnitt unter Experten ist insgesamt 3.5 ( $\sigma = 0.6$ ) und gleicht damit etwa dem Wert von 3.4 ( $\sigma = 0.8$ ) für Nichtexperten. In den Werten der QUESI-Teilmengen ist der größte Unterschied im Aussagensatz E zur wahrgenommenen Fehlerrate (Experten:  $\varnothing = 3.4$ ,  $\sigma = 0.8$ , Nichtexperten:  $\varnothing = 2.9$ ,  $\sigma = 1.0$ ). Die zweitgrößte Differenz ist im Aussagensatz G zur Erreichung der eigenen Ziele (Experten:  $\varnothing = 4.2$ ,  $\sigma = 1.0$ , Nichtexperten:  $\varnothing = 3.8$ ,  $\sigma = 0.8$ ). In beiden Unterkategorien passt der Unterschied zur allgemein höheren Erfolgsrate unter den Experten.

Zusammenfassend lässt sich sagen, dass sowohl Robotikexperten als auch Robotiknichtexperten in der Lage waren, erfolgreich mit dem System zu programmieren. Das Ausmaß des Erfolgs war dabei bei Experten durchschnittlich höher als bei Nichtexperten, bzw. diese erforderten dazu weniger Unterstützung. Auf der anderen Seite bewerteten Nicht-



**Abbildung 7.13.:** Ergebnisse der Studie zum Mehrarmsystem, aufgeteilt in haptische Eingaben („Hapt.“) und Tasteneingaben („Tast.“).

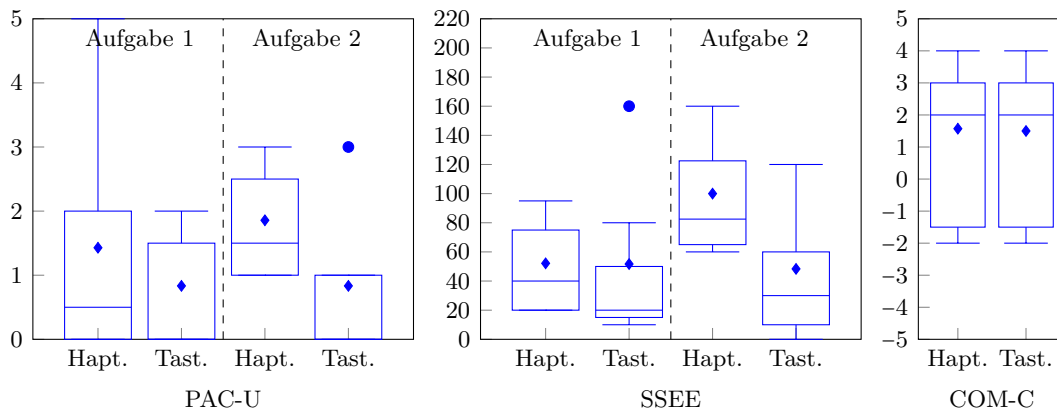
experten das System im Durchschnitt als weniger kompliziert und weniger anstrengend zu benutzen als Experten.

Vergleicht man die Ergebnisse mit denen zum ERSA-System an sich aus Abschnitt 7.1, zeigt sich ein qualitativ ähnliches Bild: In beiden Fällen waren Nutzende zur Programmierung mit dem System in der Lage, aber die Intuitivität wurde als ausbaufähig bewertet. Ein quantitativer Vergleich z.B. der QUESI-Gesamtdurchschnitte von 2.7 ( $\sigma = 0.4$ ) in der ERSA-Studie gegenüber 3.4 ( $\sigma = 0.7$ ) in der Multi-ERSA-Studie ist möglich, aber nur sehr eingeschränkt aussagekräftig. Die verwendete Funktionalität und die Aufgaben sind in den beiden Studien zwangsläufig stark unterschiedlich. Insoweit ein Vergleich erfolgt, könnte die Interpretation getroffen werden, dass die kleineren Aufgaben unter Verwendung der Synchronisationsprimitive die Nutzenden vor geringere Probleme stellten als die eine größere ERSA-Aufgabe mit Sprüngen, Verzweigungen und Posensvariablen. Detaillierte Auswertungen auf weiteren Aufgaben in beiden Bereichen mit unterschiedlichem Umfang könnten hier weitere Erkenntnisse liefern.

### 7.3. Haptische Interaktionen

In der in Abschnitt 7.2 dargestellten Benutzerstudie zu Multi-ERSA und Synchronisation wurde parallel auch die Verwendung haptischer Eingaben evaluiert. Zu diesem Zweck gab es eine Kontrollgruppe, die die Synchronisationsoperationen mittels Tasten auf dem am Greifer montierten Tastenblock statt mittels der beschriebenen Gesten programmierte. Diese setzte sich aus insgesamt 10 Teilnehmenden zusammen, im Vergleich zu 11 Teilnahmen unter Nutzung der haptischen Eingaben

Die Gesamtergebnisse sind in Abbildung 7.13 dargestellt. Es zeichnet sich hier ab, dass haptische Eingaben den im PAC-U-Wert erfassten durchschnittlichen Erfolg der Programmierung negativ beeinflussten (Aufgabe 1: haptisch  $\bar{x} = 1.3$ ,  $\sigma = 1.6$ , Tasten  $\bar{x} = 1.0$ ,  $\sigma = 0.9$ ; Aufgabe 2: haptisch  $\bar{x} = 1.5$ ,  $\sigma = 0.9$ , Tasten  $\bar{x} = 1.0$ ,  $\sigma = 1.2$ ). Dasselbe

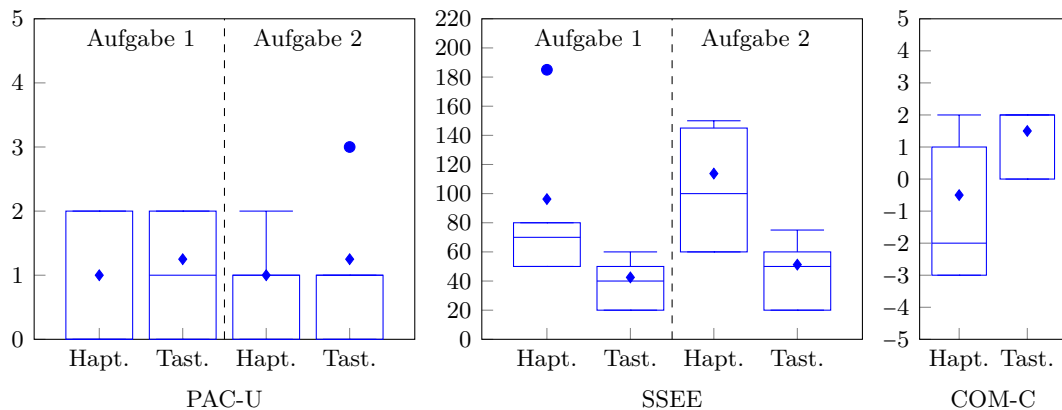


**Abbildung 7.14.:** Ergebnisse der Studie zum Mehrarmsystem innerhalb der Nichtexpertengruppe, aufgeteilt in haptische und Tasteneingaben.

trifft auf die angegebene mentale Belastung SSEE zu (Aufgabe 1: haptisch  $\bar{x} = 68.2$ ,  $\sigma = 46.9$ , Tasten  $\bar{x} = 48.0$ ,  $\sigma = 45.2$ ; Aufgabe 2: haptisch  $\bar{x} = 105.0$ ,  $\sigma = 37.0$ , Tasten  $\bar{x} = 49.5$ ,  $\sigma = 35.5$ ). Dabei ist der Unterschied in PAC-U und SSEE in der zweiten Aufgabe jeweils deutlicher als in der ersten. In den QUESI-Unterwerten zeigen sich die deutlichsten Unterschiede in den Kategorien L zum Lernaufwand (haptisch:  $\bar{x} = 3.2$ ,  $\sigma = 0.8$ ; Tasten:  $\bar{x} = 3.8$ ,  $\sigma = 1.1$ ) und F zur Vertrautheit mit dem System (haptisch:  $\bar{x} = 3.1$ ,  $\sigma = 0.7$ ; Tasten:  $\bar{x} = 3.8$ ,  $\sigma = 1.0$ ). Die erzielten QUESI-Gesamtwerte von 3.3 ( $\sigma = 0.5$ ) für haptische Eingaben und 3.6 ( $\sigma = 0.9$ ) für Tasteneingaben sind entsprechend beide leicht positiv, aber für Tasteneingaben deutlicher als für haptische.

In Abbildung 7.14 sind die Ergebnisse innerhalb der Nichtexpertengruppe für haptische und Tasteneingaben zu sehen. Diese unterscheiden sich nur im Detail von den Ergebnissen über allen Teilnehmenden. Der Erfolg sowie die mentale Belastung in Aufgabe 2 fielen innerhalb dieser Teilgruppe ähnlich gespalten aus. Die mentale Belastung in Aufgabe 1 wurde dagegen unter Nichtexperten für haptische und Tasteneingaben durchschnittlich fast gleich bewertet, nur mit unterschiedlicher Varianz (haptisch:  $\bar{x} = 52.1$ ,  $\sigma = 31.9$ , Tasten:  $\bar{x} = 51.7$ ,  $\sigma = 58.8$ ). Auch auf die Bewertung der Komplexität hatte der Eingabemodus keine deutlichen Auswirkungen (haptisch:  $\bar{x} = 1.6$ ,  $\sigma = 2.2$ ; Tasten:  $\bar{x} = 1.5$ ,  $\sigma = 2.4$ ). Die QUESI-Ergebnisse unterscheiden sich wiederum nur wenig zu den oben genannten für alle Teilnehmer.

Die Ergebnisse innerhalb der Expertengruppe sind in Abbildung 7.15 dargestellt. Auch diese sind qualitativ ähnlich. Interessanterweise fielen innerhalb der Expertengruppe die PAC-U-Werte für den Erfolg der Programmierung unter haptischen Eingaben sogar leicht besser aus als unter Tasteneingaben, sowohl für die erste Aufgabe (haptisch:  $\bar{x} = 1.0$ ,  $\sigma = 1.2$ ; Tasten:  $\bar{x} = 1.3$ ,  $\sigma = 1.0$ ) als auch für die zweite (haptisch:  $\bar{x} = 1.0$ ,  $\sigma = 0.8$ ; Tasten:  $\bar{x} = 1.3$ ,  $\sigma = 1.3$ ). Dafür ist der Unterschied in den SSEE-Werten deutlicher, genau wie in der wahrgenommenen Komplexität. Auch bei den QUESI-Teilwerten zeigen sich noch größere Abstände in den Kategorien L (haptisch:  $\bar{x} = 2.8$ ,  $\sigma = 1.1$ ; Tasten:  $\bar{x} = 3.9$ ,  $\sigma = 1.0$ ) und F (haptisch:  $\bar{x} = 3.0$ ,  $\sigma = 0.5$ ; Tasten:  $\bar{x} = 3.8$ ,  $\sigma = 0.7$ ), sowie in der Kategorie E



**Abbildung 7.15.:** Ergebnisse der Studie zum Mehrarmsystem innerhalb der Expertengruppe, aufgeteilt in haptische und Tasteneingaben.

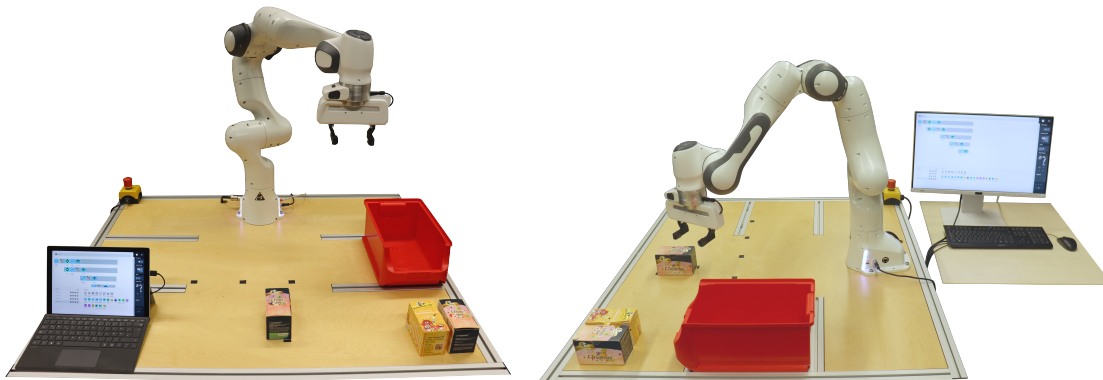
für die Wahrnehmung von Fehlern im Umgang mit dem System (haptisch:  $\varnothing = 3.6$ ,  $\sigma = 0.5$ ; Tasten:  $\varnothing = 3.1$ ,  $\sigma = 1.1$ ). Als mögliche Interpretation ließe sich formulieren, dass Experten vermutlich eine genauere Vorstellung des Programmiervorgangs und ein besseres Verständnis dafür aufweisen. Sie schaffen es dadurch, auch mit haptischen Eingaben erfolgreich zu sein, aber es kostet sie mehr mentale Anstrengung. Im Gegensatz dazu führt die zusätzliche Komplikation haptischer Eingaben bei Nichtexperten scheinbar direkter zu Fehlern in der Programmierung.

In Summe lässt sich sagen, dass die Expertise in der Roboterprogrammierung nur einen begrenzten Einfluss auf die Schwierigkeiten im Umgang mit haptischen Eingaben zu haben scheint. In beiden Teilgruppen erhöht sich die mentale Belastung. Experten waren jedoch tendenziell eher in der Lage, zusätzliche Fehler zu vermeiden.

## 7.4. Vergleichsstudie mit grafischer Schnittstelle

Zu Vergleichszwecken wurde eine weitere Studie mit gleicher Aufgabe wie die ursprüngliche ERSA-Studie aus Abschnitt 7.1 durchgeführt, aber mit einem existierenden Programmiersystem mit grafischer Schnittstelle. Aufgrund vorhandener Hardware handelte es sich dabei um das Desk Interface der Franka Emika Panda Roboter<sup>4</sup>, eines der bereits in Kapitel 2 erwähnten Beispiele. Es wurden außerdem verschiedene mögliche Eingabevarianten für das Programm verglichen: Über einen Desktop-Rechner an einem getrennten Tisch, über denselben Rechner in Kombination mit dem am Roboter vorhandenen sogenannten Pilot-Interface, und über ein Tablet direkt neben dem Roboter (ebenfalls inklusive der Pilot-Tasten am Roboter).

<sup>4</sup><https://www.franka.de/interaction>, besucht: 21.11.2022



**Abbildung 7.16.:** Aufbau in der Vergleichsstudie, links: bei Verwendung des Surface-Tablets (Aufnahme von vorne), rechts: bei Verwendung des Desktop-PCs (Aufnahme von der Seite).

### 7.4.1. Studienaufbau

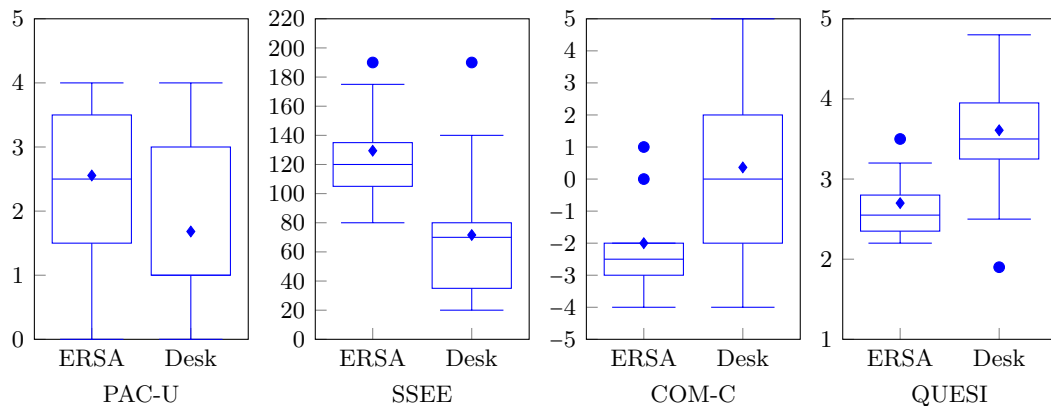
Die Vergleichsstudie wurde auf einem Franka Emika Panda durchgeführt, der wie der Kuka LBR in der Originalstudie auf einem Tisch montiert war. Der restliche Aufbau, bestehend aus einem markierten Ankunftsort, markierten Ablageorten, einem Behälter zum Eintauchen der Objekte und den Objekten (Teeboxen) selbst, war gleich.

In der Studie wurde wie oben erwähnt die Eingabe über einen Desktop-Rechner auf einem getrennten Tisch mit der über ein Tablet direkt auf dem Robotertisch verglichen. In einem Fall kamen dadurch Maus, Tastatur und Bildschirm auf einem zweiten Tisch zum Hardwaresetup dazu, im anderen Fall das Tablet, ein Microsoft Surface, das sowohl per Touchdisplay als auch per Touchpad und Tastatur bedient werden konnte. Beide Varianten des Aufbaus sind in Abbildung 7.16 wiedergegeben. Die Pilot-Schnittstelle, deren Verwendung im ersten Fall ebenfalls als Variante betrachtet wurde, besteht aus Tasten am Panda-Roboter selbst, war also ohnehin vorhanden.

Der übrige Ablauf der Studie, bestehend aus der allgemeinen Einführung, dem Tutorial (in diesem Fall für das Franka Desk Interface) und der Programmieraufgabe selbst, war soweit möglich gleich. Als Einschränkung war die Verwendung von Posenvariablen über die Desk-Schnittstelle nicht möglich, sodass dieser Aspekt (aufgreifen an und zurücklegen zu einem variierenden Ort im Eingangsbereich) wegfiel. Stattdessen sollte davon ausgegangen werden, dass die Objekte in einer festen Eingangspose ankommen, und sie sollten auch genau dorthin zurückgelegt werden. Auch die erfassten Daten, bestehend aus einer Teilmenge des MINERIC-Toolkits, waren identisch.

### 7.4.2. Ergebnisse

An der Benutzerstudie nahmen 22 Personen teil. Zunächst lassen sich die allgemeinen Ergebnisse gegen diejenigen unter Verwendung des ERSA-Systems vergleichen. Die Diagramme sind in Abbildung 7.17 gegenübergestellt. Der Reihe nach: Der Erfolg der Programmierung in Form des PAC-U-Wertes war unter Verwendung der grafischen Schnittstelle im Durchschnitt höher (ERSA:  $\varnothing = 2.6$ ,  $\sigma = 1.4$ , Desk:  $\varnothing = 1.7$ ,  $\sigma = 1.4$ ). Die mentale



**Abbildung 7.17.:** Ergebnisse unter dem ERSA-System und der Franka Desk Schnittstelle für PAC-U, SSEE, COM-C und QUESI-Durchschnitt im Vergleich.

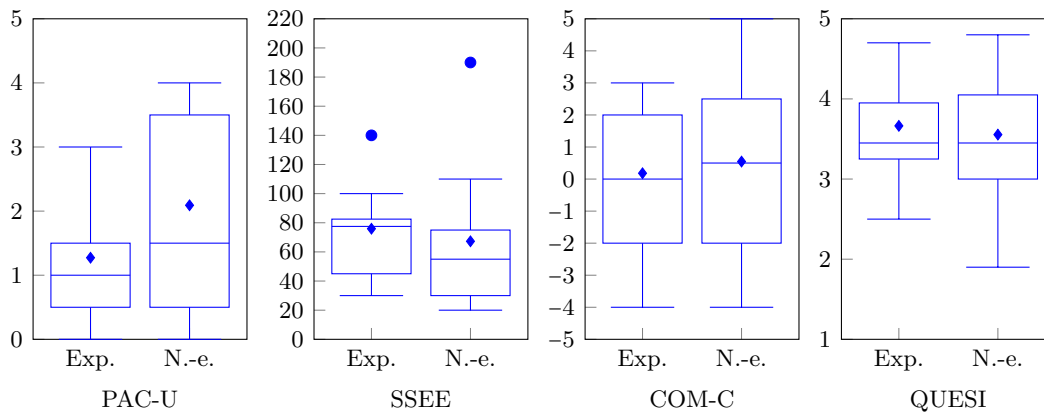
Belastung SSEE wurde unter dem ERSA-System als höher beurteilt (ERSA:  $\bar{x} = 129.4$ ,  $\sigma = 36.3$ , Desk:  $\bar{x} = 71.6$ ,  $\sigma = 39.6$ ). Das System wurde nach Verwendung im COM-C-Wert im Fall der Desk-Schnittstelle als weniger komplex bewertet (ERSA:  $\bar{x} = -2.0$ ,  $\sigma = 1.6$ , Desk:  $\bar{x} = 0.4$ ,  $\sigma = 2.5$ ). Zuletzt war der QUESI-Mittelwert mit der grafischen Schnittstelle ebenfalls höher (ERSA:  $\bar{x} = 2.7$ ,  $\sigma = 0.4$ , Desk:  $\bar{x} = 3.6$ ,  $\sigma = 0.7$ ). Dieser Unterschied verteilt sich weitgehend gleichmäßig auf die Teilkategorien; Ausgenommen ist der Aspekt G zur Erreichung der eigenen Ziele, der in beiden Systemen ähnlich hoch bewertet wurde (ERSA:  $\bar{x} = 3.6$ ,  $\sigma = 1.0$ , Desk:  $\bar{x} = 3.7$ ,  $\sigma = 0.9$ ).

All diese Aspekte kombiniert zeichnen ab, dass der Verzicht auf eine grafische Schnittstelle wohl durchaus einen negativen Einfluss auf die Benutzbarkeit des Systems hat. Es ist allerdings auf der anderen Seite klarzustellen, dass es sich beim ERSA-System um eine prototypische Umsetzung handelt, im Gegensatz zu einem zur Marktreife entwickelten System in der Desk-Schnittstelle. An dieser Stelle nicht repräsentiert ist auch der Vergleich gegen herkömmliche, meist textuelle Verfahren der industriellen Roboterprogrammierung, deren Nutzbarkeit für Nichtexperten erwartungsgemäß noch niedriger liegen würde.

Ein anderer interessanter Aspekt ist wiederum die Aufteilung der Teilnehmenden in jeweils 11 Experten und Nichtexperten. Die entsprechenden Diagramme sind in Abbildung 7.18 gegeben. Ähnlich wie bei den bisherigen Studien zeigt sich, dass Nichtexperten im Durchschnitt weniger erfolgreich waren als Experten (PAC-U für Experten:  $\bar{x} = 1.3$ ,  $\sigma = 1.0$ , für Nichtexperten:  $\bar{x} = 2.1$ ,  $\sigma = 1.6$ ), aber auch die mentale Belastung etwas geringer einstufen (SSEE für Experten:  $\bar{x} = 75.9$ ,  $\sigma = 30.5$ , für Nichtexperten:  $\bar{x} = 67.3$ ,  $\sigma = 48.2$ ). Auch der COM-C-Wert ist unter Nichtexperten leicht höher (Experten:  $\bar{x} = 0.2$ ,  $\sigma = 2.2$ , Nichtexperten:  $\bar{x} = 0.5$ ,  $\sigma = 2.8$ ), während die QUESI-Werte weitgehend übereinstimmen, sowohl im Gesamtdurchschnitt (Experten:  $\bar{x} = 3.7$ ,  $\sigma = 0.6$ , Nichtexperten:  $\bar{x} = 3.6$ ,  $\sigma = 0.8$ ) als auch in den Teilkategorien.

Weiterhin können die verschiedenen Eingabevarianten untereinander verglichen werden. Von den Teilnehmenden benutzten 7 nur den getrennten Desktop-Rechner, 8 weitere die-

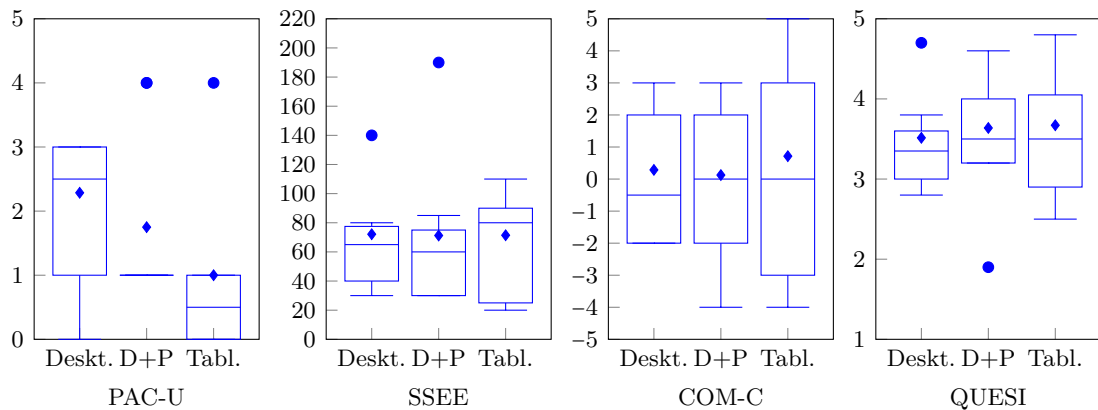




**Abbildung 7.18.:** Ergebnisse unter der Franka Desk Schnittstelle für PAC-U, SSEE, COM-C und QUESI-Durchschnitt, aufgeteilt in Robotikexperten und Robotiknichtexperten.

sen in Kombination mit der Pilot-Schnittstelle, und 7 das Tablet direkt neben dem Roboter. Dazu sind Diagramme in Abbildung 7.19 dargestellt. Die PAC-U-Durchschnittswerte sind am besten (d.h. niedrigsten) für die Eingabe unter Nutzung des Tablets, gefolgt von der Eingabe über Desktop-Rechner und Pilot-Schnittstelle, und zuletzt von der Eingabe rein über den Desktop-Rechner am Nebentisch (Desktop:  $\bar{x} = 2.3$ ,  $\sigma = 1.1$ , Desktop mit Pilot:  $\bar{x} = 1.8$ ,  $\sigma = 1.4$ , Tablet:  $\bar{x} = 1.0$ ,  $\sigma = 1.4$ ). Die mentale Belastung wurde im SSEE-Wert etwa gleich eingestuft (Desktop:  $\bar{x} = 72.1$ ,  $\sigma = 34.4$ , Desktop mit Pilot:  $\bar{x} = 71.3$ ,  $\sigma = 52.0$ , Tablet:  $\bar{x} = 71.4$ ,  $\sigma = 33.9$ ). In der Komplexität COM-C nach Benutzung des Systems zeigt sich eine leichte Tendenz zugunsten des Tablets (Desktop:  $\bar{x} = 0.3$ ,  $\sigma = 2.2$ , Desktop mit Pilot:  $\bar{x} = 0.1$ ,  $\sigma = 2.3$ , Tablet:  $\bar{x} = 0.7$ ,  $\sigma = 3.3$ ). Die QUESI-Durchschnittswerte sind zuletzt wieder nahezu gleich (Desktop:  $\bar{x} = 3.5$ ,  $\sigma = 0.6$ , Desktop mit Pilot:  $\bar{x} = 3.6$ ,  $\sigma = 0.9$ , Tablet:  $\bar{x} = 3.7$ ,  $\sigma = 0.7$ ). In den Teilkategorien ist der einzige deutliche Unterschied in der Erfüllung der eigenen Ziele (Desktop:  $\bar{x} = 3.2$ ,  $\sigma = 0.8$ , Desktop mit Pilot:  $\bar{x} = 3.5$ ,  $\sigma = 1.1$ , Tablet:  $\bar{x} = 4.2$ ,  $\sigma = 0.6$ ), was im Gesamtdurchschnitt in den Schwankungen der anderen Werte verloren geht.

Damit ist der abzulesende Trend etwas überraschenderweise, dass die Wahl der Eingabemodalität wenig Einfluss auf die subjektiven Benutzbarkeitswerte hatte, aber tendenziell den Erfolg der Programmierung beeinflusste. Dies steht im Kontrast dazu, dass die Ausdrucksstärke in allen drei Szenarien identisch ist, sich aber der Aufwand durch Gerätewechsel deutlich unterscheidet. Eine mögliche Erklärung wäre, dass Teilnehmende durch die zusätzlichen Zwischenschritte (z.B. die Bewegung von der Handführung am Roboter zum Rechner) aus ihrer Konzentration auf die Aufgabe fallen und so Fehler entstehen. Grundsätzlich stützt dieses Resultat die Motivation der vorliegenden Arbeit: Eingabemodalitäten, die direkt am oder über den Roboter passieren, können von Vorteil sein.



**Abbildung 7.19.:** Ergebnisse unter der Franka Desk Schnittstelle für PAC-U, SSEE, COM-C und QUESI-Durchschnitt, aufgeteilt in die drei Eingabevarianten: Eingabe nur über den Desktop-Rechner auf dem Nebentisch („Deskt.“), über den Desktop-Rechner unter Zuhilfenahme der Pilot-Schnittstelle auf dem Roboter („D+P“) und über das Tablet direkt neben dem Roboter („Tabl.“).

## 7.5. Bemerkungen zu statistischer Signifikanz

Aufgrund der geringen Stichprobengrößen insbesondere in den einzelnen Teilgruppen beschränken sich die obigen Diskussionen auf Betrachtung der (empirischen) Mittelwerte und Standardabweichungen. Im Folgenden ist der Vollständigkeit halber dennoch eine kurze Einordnung zur statistischen Signifikanz dargestellt.

Da es sich bei den im MINERIC-Toolkit erhobenen Werten um ordinale, aber nicht um metrische Skalen handelt, findet eine Auswertung über nichtparametrische Verfahren statt, die keine Annahmen über die zugrundeliegenden Verteilungen voraussetzen. Die im Folgenden erwähnten (in R ermittelten)  $p$ -Werte sind in Anhang A in den Tabellen A.4 bis A.6 aufgeführt.

Zu den prinzipiell über Tests evaluierbaren Aussagen von oben zählen:

- Die Werteverteilungen von Experten und Nichtexperten unterscheiden sich nicht signifikant: Überprüfung mit Mann-Whitney-U-Tests auf den verschiedenen Datensätzen (PAC-U, SSEE, COM-C, QUESI-Gesamtwert) der drei Studien ergibt, dass die Nullhypothese (Verteilungen unterscheiden sich nicht zwischen Experten und Nichtexperten) in keinem der Fälle mit  $\alpha = 0.05$  abgelehnt werden kann. Anhand der Stichprobe ist die Aussagekraft jedoch gering, d.h. es kann daraus jeweils nicht verlässlich die Annahme getroffen werden, dass die Verteilungen zwischen Experten und Nichtexperten tatsächlich gleich sind.
- Die Werteverteilungen von haptischen und Tasteneingaben aus der Multi-ERSA-Studie unterscheiden sich nicht signifikant: Auch hier zeigen Mann-Whitney-U-Tests überwiegend keine Unterscheidbarkeit mit  $\alpha = 0.05$ <sup>5</sup>. Auch hier ist die Annahme

<sup>5</sup>mit Ausnahme des SSEE-Werts der zweiten Aufgabe ( $p = 0.003$ )

der Nullhypothese aufgrund der Stichprobengröße diskutabel.

- Die Werteverteilungen der drei Eingabemodi in der Vergleichsstudie unterscheiden sich nicht signifikant: Aufgrund der drei möglichen Ausprägungen werden hier Kruskal-Wallis-Tests angewendet. Erneut sind die Nullhypothesen, obwohl keine Unterscheidbarkeit mit  $\alpha = 0.05$  vorliegt, nicht sinnvollerweise automatisch anzunehmen.

Zusammengefasst sollen die aufgeführten Bemerkungen illustrieren, dass für statistisch signifikante Schlussfolgerungen der Umfang der durchgeführten Studien nicht ausreicht. Das bedeutet nicht, dass in den Daten gar keine Trends erkennbar sind, aber dass diese nicht verlässlich auf die Gesamtpopulation schließen lassen. Zukünftige Arbeiten könnten an dieser Stelle ansetzen, und für einzelne zu betrachtende Aspekte eine breitere Datengrundlage erheben.



---

8.1. Zusammenfassung und Diskussion . . . . .	113
8.2. Ausblick . . . . .	117
8.2.1. Technische Verbesserungen am existierenden System . . . . .	117
8.2.2. Breitere Datengrundlage . . . . .	117
8.2.3. Aspekte aus der initialen Abgrenzung . . . . .	118
8.2.4. Neue Stoßrichtungen . . . . .	119

---

Im abschließenden Kapitel werden zunächst die bisherigen Inhalte und Ergebnisse der Arbeit rekapituliert, insbesondere, inwieweit die eingangs identifizierten wissenschaftlichen Fragestellungen beantwortet werden konnten. Anschließend wird noch ein Ausblick auf offene Fragen gegeben, die teilweise aus der zu Anfang getroffenen Eingrenzung des Themas hervorgehen, teilweise im Rahmen der Bearbeitung aufgekommen sind.

## 8.1. Zusammenfassung und Diskussion

In dieser Arbeit wurde ein automatenbasierter Ansatz zur Roboterprogrammierung für Nichtexperten erforscht. Dabei wurde von vornherein die Motivation formuliert, auf eine grafische Schnittstelle zu verzichten, unter der Annahme, dass eine Menge an Aufgaben existiert, die zwar Programmstrukturen jenseits von reinem Playback erfordern, deren Struktur aber Nutzende gleichzeitig ausreichend gut im Kopf haben, um die Programmierung allein anhand des aktuellen Weltzustands Schritt für Schritt durchzuführen. Das System wurde in den Kontext verwandter Arbeiten eingeordnet. Insbesondere wurde ein Überblick über Verfahren zur Roboterprogrammierung im Allgemeinen gegeben, unterteilt in die Aspekte der Eingabe von Posen und der Eingabe des Programminhalts selbst. Weiterhin wurde existierende Forschung vorgestellt, die auf endlichen Automaten und deren Varianten basiert. Als Fazit ließ sich feststellen, dass ein automatenbasiertes Verfahren zur Erzeugung strukturierter Programme ohne grafischen Editor aktuell eine Forschungslücke bildet, die in der Arbeit erkundet wurde.

Im Folgenden wird der weitere Inhalt der Arbeit anhand der eingangs aufgestellten Forschungsfragen zusammengefasst, beginnend mit den spezifischeren Teilfragen F2 bis F6 und abschließend mit der übergreifenden Frage F1.

**F2** Mit welcher Funktionalität lässt sich das ursprüngliche Modell erweitern? Mit welchen Operationen können Nutzende solche erweiterten Automaten anlegen?

Das zugrunde liegende Automatenmodell der *Extended Robot State Automata* (ERSA) wurde aus dem einfacheren Modell der *Robot State Automata* entwickelt und vorgestellt, sowie auf das damit verbundene Programmiersystem eingegangen. Die Funktionalität, die durch diese Erweiterung abgedeckt wird, beinhaltet relative Bewegungen, die für inkrementelle Änderungen in Schleifenstrukturen essenziell sind, die Spezifikation von Bewegungszielen relativ zu Objektposen, was ein dynamisches Aufgreifen an nicht vollständig spezifizierten Positionen und in Bereichen ermöglicht, und die Verwendung von Posenvariablen, wodurch zur Programmierzeit unbekannte Orte zur Laufzeit gespeichert und wieder angefahren werden können. Die Spezifikation dieser Funktionen als Updates im Automaten wird durch jeweils eine eigene Eingabeoperation umgesetzt. Um diese Programmieroperationen durch die Nutzenden auslösen zu lassen, sind dabei die haptischen Interaktionen ein zentraler Aspekt:

**F3** Mit welchen haptischen Gesten lassen sich die nötigen Programmieroperationen umsetzen? Welchen Einfluss haben haptische Eingaben auf die Intuitivität des Systems?

Dazu wurde eine Vorstudie in Form einer Onlineumfrage durchgeführt, aus deren Ergebnissen sich die in dieser Arbeit implementierten Interaktionen begründen. Verwendet werden insbesondere Ruckbewegungen senkrecht in Weltkoordinaten, Ruckbewegungen in einzelnen Gelenken, und eine horizontale Kreisgeste in Weltkoordinaten. Auch in der späteren Erweiterung auf mehrere Roboterarme in Multi-ERSA werden haptische Eingaben für die Synchronisationsoperationen angewendet, genauer gesagt horizontale Ruckgesten in Weltkoordinaten entlang der Achse zwischen den beiden Roboterbasen bzw. senkrecht dazu. Der Einfluss auf die Intuitivität wurde im Zusammenhang mit dem Multi-ERSA-System durch den Vergleich mit reinen Tasteneingaben auf einem am Greifer montierten Tastenblock evaluiert. Dabei ist im direkten Vergleich abzulesen, dass haptische Interaktionen die Komplexität für die Teilnehmenden erhöhten, was bei Experten und Nichtexperten unterschiedliche Konsequenzen zur Folge hatte (höhere mentale Anstrengung bzw. weniger erfolgreiche Programmierung).

In einer Vergleichsstudie unter Verwendung der Franka Emika Desk Schnittstelle<sup>1</sup> wurde zudem eine der motivierenden Annahmen hinter haptischen Eingaben evaluiert:

**F4** Welchen Einfluss haben Wechsel der Eingabemodalität und der dadurch anfallende Mehraufwand auf die Benutzbarkeit?

---

<sup>1</sup><https://www.franka.de/interaction>, besucht: 21.11.2022

Dazu wurden drei Varianten der Programmeingabe mit diesem existierenden, grafischen System verglichen: Erstens über einen Desktop-Rechner an einem separaten Tisch, zweitens über denselben Rechner in Kombination mit dem aus Tasten am Roboter selbst bestehenden Pilot-Interface, und drittens über ein Tablet direkt neben dem Roboter (ebenfalls unter Hinzunahme des Pilot). Diese drei Abstufungen reichen von eher hohem Aufwand für Wechsel der Eingabemodalität (häufig erforderlich, und benötigen einen Standortwechsel der Nutzenden) zu eher niedrigem (Wechsel durch Pilot seltener nötig, und durch das Tablet direkt neben dem Roboter kaum Distanz zurückzulegen). In den Resultaten erzeugten diese drei Varianten wenig Unterschiede in Bezug auf die Benutzbarkeitswerte, doch in puncto Effektivität zeigte sich eine Tendenz, dass höherer Aufwand für Eingabewechsel einen negativen Einfluss auf den Programmiererfolg hatte.

Ein Aspekt aus den Ergebnissen der ersten durchgeführten Benutzerstudie zum ERSA-System an sich war die Erkenntnis, dass das manuelle Schließen von Schleifen von Nutzern als mental anstrengend bewertet wurde. Infolgedessen beschäftigte sich ein weiterer Teil der Arbeit mit der automatisierten Struktursynthese:

**F5** Wie können sich wiederholende Strukturen in den Automaten erkannt und ergänzt werden?

Dabei wurde zur einfacheren Programmierung von Schleifenstrukturen ein entsprechendes Verfahren entwickelt, das die Schleifen aus partieller Wiederholung des Schleifenrumpfs ableitet. Dieses besteht aus einem Markierungsalgorithmus, der für Zustandspaare in ERSA beschreibt, ob diese potenziell vereinigbar sind oder sich widersprechen, und falls ersteres, welcher Umfang an Hinweisen (d.h. welche Länge an Überlappung in den zugehörigen Zustandsfolgen) für eine Vereinigung vorliegt. Auch die eigentliche Unifikationsprozedur wurde beschrieben, sowie die Auswertung im realen System auf einer der Beispielaufgaben, womit die Anwendbarkeit des Verfahrens verifiziert wurde.

In einer weiteren Erweiterung wurde die Anwendbarkeit des Ansatzes auf mehrere Roboterarme betrachtet:

**F6** Wie lässt sich der vorgestellte Ansatz in Automatenmodell und Programmierung auf mehrere Roboter erweitern? Inwieweit sind Nutzende damit in der Lage, mehrere Roboter ohne grafische Schnittstelle zu programmieren?

Dafür wurde die Automatenvariante der Multi-ERSA entwickelt. Die darin umgesetzten Synchronisationsmöglichkeiten von Präzedenzen, Rendezvous und synchronen Aktionen beruhen dabei auf einer Übersicht verwandter Arbeiten zur Multiroboterprogrammierung. Auch für Multi-ERSA wurde das Programmiersystem beschrieben, in dem wie erwähnt wiederum haptische Interaktionen eine zentrale Rolle spielen.

Zur Beantwortung des zweiten Teils der Frage wurde das erweiterte Multi-ERSA-System in einer Benutzerstudie auf zwei kleineren Programmieraufgaben evaluiert. Sowohl Experten als auch Nichtexperten waren mit dem System in der Lage, die formulierten Aufgaben durch Programmierung zweier Leichtbauroboter umzusetzen. Die Intuitivität wurde dabei in einzelnen Teilaspekten recht unterschiedlich bewertet. Positiv sticht die wahrgenommene Erfüllung der eigenen Ziele heraus; Im Vergleich fallen Aspekte

wie die mentale Belastung eher neutral aus und bilden daher Anstoßpunkte für weitere Verbesserungen des Ansatzes.

Im Abschnitt zur Evaluation wurde auch die Auswertung der initialen, übergreifenden Forschungsfrage vorgestellt:

**F1** Inwieweit sind Robotik-Nichtexperten als Endanwendende in der Lage, Roboter ohne grafische Schnittstelle zu programmieren?

Neben der Studie zu Multi-ERSA beschäftigte sich die erste durchgeführte Benutzerstudie dabei mit dem ERS-Modell selbst, zu welchem Teilnehmenden eine einzelne, größere Programmieraufgabe gestellt wurde. Auch hier konnte aus den Ergebnissen die Schlussfolgerung gezogen werden, dass auch Nichtexperten in der Lage waren, mit dem System erfolgreich zu programmieren, dass aber die Intuitivität des Systems als noch verbesserungsfähig bewertet wurde. Insbesondere fielen beispielsweise die mentale Belastung und der Lernaufwand bei dieser größeren Aufgabe deutlich negativer aus als bei den kleineren Aufgaben der Multi-ERSA-Studie.

Innerhalb der weiter oben erwähnten Vergleichsstudie wurden Ergebnisse bei weitgehend gleicher Aufgabe und Rahmenbedingungen mit dem Franka Emika Desk System erfasst. Im direkten Vergleich zeigt sich unter dem ERS-System mit seinem Verzicht auf eine grafische Schnittstelle eine Verschlechterung in den erfassten Werten zur Benutzbarkeit. Dennoch sind die erzielten Resultate für sich genommen nicht zu negativ zu bewerten. In allen durchgeführten Studien zeigten sich interessante Effekte von Robotik-Vorerfahrung der Teilnehmenden: Tendenziell erhöhte solche den Erfolg der Programmierung, aber gleichzeitig auch die wahrgenommene mentale Belastung und die abgegebene Einschätzung der Komplexität. Andererseits zeigte sich, dass etwa der Unterschied im Programmiererfolg unter verschiedenen Eingabemethoden und -systemen bei Experten qualitativ ähnlich war wie bei Nichtexperten.

An dieser Stelle lässt sich folgendes Gesamtfazit ziehen:

Die erzielten Werte für Effektivität und Intuitivität der Programmierung unter dem ERS-System bieten einigen Raum für Interpretation. Einerseits sind letztere wie oben erwähnt zumeist im neutralen bis leicht negativen Bereich einzuordnen. Andererseits ist a priori nicht selbstverständlich, dass Nutzende und insbesondere Nichtexperten in einem System ohne grafische Repräsentation überhaupt in der Lage sind, Aufgaben erfolgreich zu programmieren. Unter diesem Gesichtspunkt stellt diese Arbeit einen ersten grundlegenden Ansatz dar, an dem weitere Forschung ansetzen könnte. In Antwort auf die oben formulierte Frage lässt sich hieraus zumindest eine untere Schranke formulieren: Für kleinere Aufgaben und unter einiger mentaler Anstrengung ist die Programmierung durch Nichtexperten offenbar möglich, wie am vorliegenden System demonstriert.

Auch die Verwendung haptischer Interaktionen ist differenziert zu bewerten. Dem offenkundigen Zuwachs an Komplexität für Nutzende steht der Kontext gegenüber, dass es sehr wenige Systeme gibt, die anderweitig haptische Interaktionen verwenden, und mit denen die Teilnehmenden vorher Kontakt gehabt haben könnten. Im Gegensatz dazu sind Tasten, Touchscreens oder verbale Anweisungen Interaktionsmodalitäten, mit denen die meisten Nutzenden vertraut sein dürften. Um diesen Aspekt fehlender Vertrautheit



mit der Modalität zu evaluieren oder zu kompensieren, wären für zukünftige Forschung Longitudinalstudien interessant, bei denen Nutzende über einen längeren Zeitraum mit ähnlichen Systemen und Ansätzen vertraut werden können.

## 8.2. Ausblick

In diesem Abschnitt werden Aspekte der Arbeit und der Materie identifiziert, die für zukünftige Forschung von Interesse sein könnten.

### 8.2.1. Technische Verbesserungen am existierenden System

Die erste und kurzfristigste Kategorie bezieht sich direkt auf Kritikpunkte an Details des existierenden Systems, die im Rahmen der Studien aufgefallen sind oder als Feedback genannt wurden. Dazu gehörten u.a. die häufigen Probleme mit der Verknüpfungsoperation, die die Entwicklung der Verfahren aus Kapitel 5 motiviert haben.

Eine andere Idee war, bei der Programmierung von Multi-ERSA die Operation für synchrone Bewegungen kontextabhängig zu machen. Die Operation an der zuletzt angefahrenen Roboterpose auszulösen, könnte die Bewegung davor als synchron markieren, was eine wiederholte Fehlerquelle (falsche Operation bzw. Reihenfolge seitens der Nutzenden) abschwächt. Auch die Frage, ob das Anlegen einer Verzweigung direkt die Aufnahme der ersten Zweigbedingung auslösen sollte, wurde häufiger gestellt. Weiterhin wurde mehrmals vorgeschlagen, die Aufnahme von Trajektorien statt einzelnen Zielposen zumindest als Option anzubieten, z.B. darüber, eine Taste gedrückt zu halten. Dies könnte die Programmierung von Aufgabenteilen nahe an Hindernissen, wie z.B. das Eintauchen eines Objekts in einen Behälter in der ERSA-Studie, erleichtern, da mögliche Probleme durch die Interpolation zwischen Zielposen dadurch wegfallen.

Ein Erweiterungsvorschlag zu den Tasten selbst war, über LED-Beleuchtung zu signalisieren, welche Eingaben in einer bestimmten Situation jeweils überhaupt akzeptiert werden. Auch zu den verwendeten Icons gab es sehr verschiedene Anmerkungen und Verbesserungsvorschläge.

Die Tasten sind jedoch ohnehin als Übergangslösung zu verstehen. Tasten sind in manchen Konfigurationen eventuell schlecht erreichbar und erfordern unter Umständen ein Umgreifen aus der Führung [104], auch wenn letzteres durch die gewählte Anbringung des Tastenblocks vermieden werden soll. Langfristiges Ziel wäre es, die nötigen Schnittstellen gänzlich haptisch umzusetzen. Die zugunsten einer einfachen Implementierung verwendeten Tasten ließen sich natürlich auch auf andere Weise ersetzen, etwa durch taktile Sensorelemente auf der Oberfläche des Roboters.

### 8.2.2. Breitere Datengrundlage

Als nächstes gibt es auf der Ebene der bereits vorgestellten Ansätze verschiedene Möglichkeiten, die Datengrundlage zu verbreitern. Dazu zählen neben schlicht größeren Teilnahmezahlen für statistische Signifikanz auch die im letzten Abschnitt genannten Longitudinalstudien, um Effekte von fehlender Vertrautheit mit der haptischen Interaktions-

modalität isoliert zu betrachten. Auch allgemein könnte die Erhebung weiterer Daten aus Benutzerstudien helfen, die Ansätze feiner einzuordnen, insbesondere z.B. der weitere Vergleich mit anderen, existierenden Programmieransätzen (unter möglichst denselben Bedingungen und mit denselben Aufgaben).

Auch die Vorstudie zu den Gestenentwürfen, eine reine Onlineumfrage, könnte durch eine weitere Studie am realen Roboter gestützt werden. Damit ließe sich verifizieren, ob sich die erzielten Bewertungen anhand abstrakter Beschreibungen tatsächlich auf die Anwendung übertragen lassen.

### 8.2.3. Aspekte aus der initialen Abgrenzung

Eine Reihe weiterer Ansatzpunkte ergibt sich aus den Einschränkungen, die zu Anfang der vorliegenden Arbeit dargestellt wurden. Dazu zählt beispielsweise der Verzicht auf Fähigkeiten zur Feinbewegung (also auf Basis von Kräften und Momenten überwachte oder geregelte Bewegungen). Tatsächlich sind etwa laut [29], einer Analyse aus einer Haushaltsdomäne, fast die Hälfte der dortigen Aufgaben Reinigungsaufgaben, und diese erfordern zu ca. 95% die Anwendung eines Werkzeugs (z.B. Wischlappen, Schwamm o.ä.) auf ein Objekt oder eine Oberfläche. Solche Interaktionen sind rein positionsgeregelt meist nicht möglich. Um Feinbewegung in die vorgestellten Automatenmodelle zu integrieren, wäre es nötig, z.B. entsprechende Primitive zu identifizieren, die in die Menge der Updateschritte aufgenommen werden könnten. Ähnlich zur Multiarmprogrammierung sollte hier zunächst untersucht werden, wie existierende Arbeiten Feinbewegungen auf einer niedrigen Abstraktionsebene umsetzen. Außerdem ist nicht offensichtlich, in welchem Umfang solche Fähigkeiten in welchen Aufgabendomänen erforderlich sind, seien es kraftüberwachte, seien es kraftgesteuerte oder hybride Bewegungen.

In der vorliegenden Arbeit wurde auch der Aspekt der Objekterkennung bewusst außen vor gelassen. Es wird allgemein die Annahme getroffen, dass aus der am Roboter montierten Kamera bekannte Objekte und deren Posen in Weltkoordinaten identifiziert werden können. Die verwendeten Verfahren zur Objekterkennung selbst sollten zum Rest des Systems orthogonal sein. Es wäre aber interessant, Möglichkeiten zum Umgang mit unbekanntem Objekten in das System zu integrieren. Darunter fallen z.B. Ansätze, die aus den (RGB-D-) Kamerabildern Teilmodelle generieren können. Solange die Objekte danach immer in grob ähnlicher Orientierung vorliegen, sollte das in vielen Fällen bereits ausreichend sein für eine Wiedererkennung. Ein größerer Änderungsschritt wäre es, wenn bei Vorlage eines unbekanntem Objekts dieses tatsächlich zuerst aus mehreren Perspektiven eingelesen würde, umgedreht vom menschlichen Anwender oder auch dem Roboter selbst. Dies würde unter Umständen einen weiteren Moduswechsel einführen (neben Programmierung und Ausführung nun auch das Einlesen, das von der eigentlichen Aufgabe unabhängig ist).<sup>2</sup>

Auch die Bahnplanung (zwischen einzelnen anzufahrenden Posen) wird in der vorliegenden Arbeit außen vor gelassen. In der Implementierung handelt es sich dabei um simple Point-to-Point-Bewegungen. Verschiedene Varianten sind hier denkbar und könnten eva-

---

<sup>2</sup>Der Gedanke an ein automatisiertes Einlernverfahren geht auf [66] zurück.

hüert werden, angefangen mit einem Wechsel auf Linearbewegungen. Wenn dem System weitere Perzeptionsfähigkeiten oder ein internes Weltmodell zur Verfügung gestellt würden, könnten hier tatsächlich Verfahren zur kollisionsfreien Bahnplanung angewendet werden. Darüber hinaus könnten ausführlichere Planungsverfahren auch ein Scheitern der Planung erkennen, wenn etwa der Weg im Arbeitsraum durch Anwesenheit von aufgabenfremden Personen oder Objekten verstellt wäre.<sup>3</sup>

#### 8.2.4. Neue Stoßrichtungen

Die letzte Kategorie an Themen für weitere Forschung sind Aspekte, die aus der Bearbeitung oder den Ergebnissen aufgetreten sind. In Richtung Objekthandhabung war z.B. eine häufige Bemerkung aus den Benutzerstudien, dass die reine Gravitationskompensation des Roboters insofern unbequem ist, als dass der Roboter bei gegriffenen Objekten in der Führung unter deren Gewicht nach unten driftet und unterstützt werden muss. Dazu wäre es hilfreich, wenn direkt nach dem Aufgreifen Gewicht, Schwerpunkt und Trägheitsmomente der Objekte schätzungsweise ermittelt werden könnten, sodass dieser Effekt vermieden wird. Auch schon eine reine Gewichtsanpassung dürfte eine Verbesserung darstellen, da etwa die in den Benutzerstudien gehandhabten Objekte sich im mittleren zweistelligen bis niedrigen dreistelligen Grammbereich bewegten, und die Schwerpunkte innerhalb der Greifergeometrie lagen. Eine solche ließe sich über eine Interaktion umsetzen, in der der Roboter bewusst neutral gehalten, dann kurz losgelassen wird, sodass aus der Kraftspitze am Endeffektor das Gewicht berechnet werden kann.

Im Feld der Mehrarmprogrammierung decken Multi-ERSA die grundlegenden identifizierten Synchronisationsfähigkeiten ab. Es gibt aber hier eine ganze Reihe weiterer Aspekte, die umgesetzt werden könnten. Eine häufig genannte weitere Möglichkeit ist die Programmierung von relativen Bewegungen eines Roboters zum beweglichen Referenzkoordinatensystem des anderen (teils auch als *Master/Slave* betitelt). In so einem Fall stellen sich einige Fragen für die Eingabe der Bewegung selbst, z.B. ob zuerst die relative oder die Referenzbewegung angelegt wird, und, damit verbunden, ob die relative Bewegung bei einem stehenden zweiten Arm programmiert wird oder parallel zu dessen Bewegung.

In Bezug auf das vorgestellte Verfahren zur Struktursynthese öffnen sich mehrere Forschungsrichtungen. Zunächst ist die Anwendung auf das Multi-ERSA-Modell in dieser Arbeit nicht mehr erfolgt und wäre ein natürlicher weiterer Schritt. Daneben stellt sich wie im Text erwähnt das Problem der Syntheschwelle. Diese wurde in der Verifikation jeweils manuell gesetzt, abhängig von der Aufgabe. Erstens wäre hier eine ausführlichere Betrachtung interessant, wie sich dieser Wert allgemein sinnvoll (fest) setzen lässt. Noch vorteilhafter wären Methoden, den Wert automatisch zu bestimmen (idealerweise vermutlich in Abhängigkeit von der Aufgabe). In Kapitel 5 wurde dazu bereits ein an das Page-Fault-Frequency-Verfahren angelehnter Gedanke vorgestellt, wozu Nutzende auf Unifikationsvorschläge mit Bestätigung oder Ablehnung reagieren müssten, um den

---

<sup>3</sup>Solche Rückmeldungen werden etwa im Kontext einer natürlichsprachlichen Roboterkommandierung in [168] betrachtet.

Schwellwert im Lauf der Zeit anzupassen. Insbesondere kleinere Aufgaben könnten dafür allerdings zu wenig Raum bieten (was im Gegenzug zur Notwendigkeit für über mehrere Aufgaben hinweg ermittelte Schwellwerte führen könnte).

Ein weiterer spannender Forschungsbereich öffnet sich in den haptischen Interaktionen. Wie bereits erwähnt existieren dazu nur wenige Vorarbeiten. Im Kontrast dazu ist der Spielraum an Bewegungen, die für Operationen eingesetzt werden könnten, praktisch grenzenlos, und die Menge der verwendbaren Verfahren allein aus der Erkennung menschlicher Gesten immens. Möglichkeiten für eine gründlichere, isolierte Betrachtung dieses Aspekts beginnen bei der Vorstudie: Eine Annahme dahinter war, dass die theoretische Bewertung der verschiedenen Gesten ähnliche Werte erzielt wie deren praktischer Einsatz. Diese Annahme könnte durch Studien an realen Robotern überprüft werden. Weitere Aspekte sind hier wie oben erwähnt die Auswahl und Umsetzung der zu erkennenden Gesten, aber auch ausführlichere Vergleiche mit anderen Interaktionsmodalitäten. Insbesondere stellt sich für die einzelnen Operationen die Frage nach einer Evaluation, welche Modalität von Nutzenden dafür bevorzugt wird. Aus dem freien Feedback der Benutzerstudien ging hervor, dass etwa zur Eingabe einzelner Zielposen die kurzen Anpinngesten als einfach und flüssig wahrgenommen wurden. Auch existierende multimodale Ansätze könnten, unabhängig von der speziellen Ausrichtung der vorliegenden Arbeit, Erweiterungen in dieser Richtung in Betracht ziehen.

Ebenfalls eine mögliche technische Erweiterung wäre die Verwendung von haptischem Feedback vom Roboter zum Menschen. Anbieten würde sich dies z.B. bei der Auswahl von existierenden Zuständen beim Verbinden. Der Roboter könnte hier in der Nähe von bekannten Posen eigenständig in deren Richtung driften. Ähnliche Konzepte existieren etwa in [101, 167].

Zwischen diesen stark unterschiedlichen Ansatzpunkten bietet sich von dieser Arbeit ausgehend zusammengefasst ein breites Potenzial, die Roboterprogrammierung durch Nichtexperten zukünftig weiter voranzutreiben.

## ANHANG **A**

---

### Studienergebnisse

---

		QUESTI																
	PAC-U	SSEE	COM-C	COM-D	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Experten	4	150	-4	-1	4	3	3	3	4	5	4	3	2	4	3	5	3	4
	2	90	0	-2	1	5	2	2	4	2	5	1	3	4	1	4	1	2
	3	190	-3	-1	3	4	3	3	4	3	4	3	3	3	3	4	3	3
	1	175	-3	0	2	5	4	3	3	4	5	4	4	3	2	5	4	4
Nicht- Experten	4	80	1	4	4	5	5	5	5	5	5	5	5	5	5	5	5	5
	4	120	1	4	4	4	5	5	2	3	3	5	4	2	4	3	4	5
	0	120	5	0	1	4	3	2	3	2	4	3	4	3	1	4	3	3
	2	120	3	2	4	3	3	4	2	4	3	2	4	3	4	4	3	3
	3	120	2	3	5	5	5	5	3	4	5	5	4	4	4	4	5	5

Tabelle A.1.: Vollständige Daten aus den Experimenten zu ERSA (sh. Abschnitt 7.1).

		QUESTI																	
		PAC-U	SSEE	COM-C	COM-D	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Experten, PC	1	75	2	0	3	3	4	4	3	4	4	4	4	4	3	2	3	4	3
	1	30	3	5	3	5	4	5	5	5	5	4	5	4	5	4	5	5	5
	2	140	-2	-2	2	5	3	3	4	3	4	2	4	3	4	3	2	3	3
	2	80	2	0	4	3	5	4	3	4	3	4	4	4	4	4	3	5	4
Experten, PC + Pilot	1	60	0	-2	3	4	4	4	3	4	4	4	4	5	5	3	4	4	3
	1	30	2	0	3	5	4	4	5	5	5	5	5	5	5	4	5	4	5
	1	85	2	-1	3	5	5	4	5	4	2	4	4	4	4	3	4	4	5
	1	75	0	0	3	4	4	3	4	4	4	4	3	4	3	2	2	4	4
Experten, Tablet + Pilot	1	80	-1	1	2	4	3	2	4	4	4	4	5	4	2	2	4	3	4
	0	100	-2	0	1	5	4	3	4	3	5	4	3	4	3	4	1	4	3
	1	80	-4	-1	2	4	5	3	1	2	2	3	1	3	2	4	3	1	
Nicht- Experten, PC	2	50	-2	0	3	4	3	4	5	3	4	3	3	3	4	2	1	3	4
	3	70	-2	0	3	3	3	2	3	3	3	3	3	3	3	3	2	3	3
	3	60	1	2	4	2	3	4	3	4	4	4	4	3	3	3	2	4	3
Nicht- Experten, PC + Pilot	4	190	-4	-3	1	2	3	2	2	2	2	1	2	3	2	1	2	3	1
	1	60	3	1	3	5	5	5	4	5	5	5	5	5	4	4	5	5	5
	1	40	-2	-6	3	4	4	5	3	4	2	4	4	4	3	4	2	4	3
	4	30	0	-1	3	4	4	2	4	3	2	3	2	3	5	3	3	4	2
Nicht- Experten, Tablet + Pilot	4	90	1	3	4	4	3	5	4	3	4	2	3	4	2	4	4	4	3
	0	20	5	3	4	5	5	4	5	5	5	5	5	5	5	5	5	5	5
	0	110	2	-1	3	5	4	4	4	4	4	2	3	3	5	4	4	5	3
	1	30	4	2	4	5	4	3	5	4	4	5	4	5	5	4	4	4	4

Tabelle A.2.: Vollständige Daten aus den Vergleichsexperimenten zu ERSA mit dem Franka Desk Interface.

	Aufgabe 1		Aufgabe 2		QUESTI																
	PAC-U	SSEE	PAC-U	SSEE	COM-C	COM-D	1	2	3	3	4	5	6	7	8	9	10	11	12	13	14
Experten, haptisch	2	80	1	60	2	4	4	3	3	3	4	5	4	4	3	2	4	3	5	3	4
	2	185	2	150	-2	-4	1	5	2	2	4	4	2	5	1	3	4	1	4	1	2
	0	70	1	145	-3	-3	3	4	3	3	4	4	3	4	3	3	3	3	4	3	3
Experten, Tasten	0	20	1	20	2	4	4	5	5	5	4	4	5	5	5	4	4	3	5	5	3
	2	50	3	75	2	2	2	5	3	4	3	3	3	3	3	2	4	3	5	2	2
	2	40	1	60	0	2	3	2	4	3	1	3	3	2	4	5	2	3	2	4	4
Nicht- Experten, haptisch	1	60	0	50	2	3	4	5	4	3	3	4	5	4	4	5	4	3	4	4	5
	5	60	2	160	2	5	2	3	4	4	1	2	4	4	3	3	3	4	4	4	3
	0	20	1	60	4	6	3	4	3	4	2	4	4	5	4	3	3	2	4	4	3
Nicht- Experten, haptisch	0	95	2	125	3	3	2	5	5	5	3	3	4	5	2	4	3	2	4	4	3
	2	70	3	90	-1	-3	1	3	4	2	3	3	3	4	3	4	3	3	4	4	3
	0	20	1	70	2	0	4	4	4	3	3	4	4	4	4	4	4	3	3	4	5
Nicht- Experten, Tasten	1	20	1	75	3	6	3	4	4	2	4	4	5	4	4	3	4	3	5	3	3
	2	80	3	120	-2	1	2	3	3	2	1	3	3	3	1	3	2	1	3	2	2
	0	10	0	30	4	1	4	5	5	5	5	5	5	5	5	5	5	5	5	5	5
Nicht- Experten, Tasten	0	20	1	0	3	6	4	4	5	5	2	3	3	3	5	4	2	4	4	3	4
	1	80	0	120	-1	-4	1	4	3	2	3	2	4	4	3	4	3	1	4	4	3
	2	20	1	40	3	1	4	3	3	4	2	4	3	2	4	4	3	4	4	3	3
Nicht- Experten, Tasten	0	20	0	20	2	3	5	5	5	5	3	4	4	5	5	4	4	4	4	5	5
	2	160	3	80	-2	1	2	1	2	1	1	2	2	2	2	2	2	1	3	2	2

Tabelle A.3.: Vollständige Daten aus den Experimenten zu Multi-ERSA (sh. Abschnitt 7.2).



	PAC-U		SSEE		COM-C	COM-D	QUESI
ERSA	0.8810		0.1349		0.2381	0.5952	0.7302
Franka Desk	0.2455		0.2191		0.8379	0.9608	0.7843
Multi-ERSA	0.7626	0.6942	0.3917	0.9013	0.1264	0.6038	0.8734

**Tabelle A.4.:**  $p$ -Werte aus Mann-Whitney-U-Tests zur Unterscheidung zwischen Experten und Nichtexperten (sh. Abschnitt 7.5). Bei der Multi-ERSA-Studie sind die Werte für PAC-U und SSEE der beiden Aufgaben angegeben. QUESI meint hier den erzielten Gesamtwert.

	PAC-U		SSEE		COM-C	COM-D	QUESI
1	0.1732	0.1575	0.0033	0.5382	0.7684	0.4994	

**Tabelle A.5.:**  $p$ -Werte aus Mann-Whitney-U-Tests zur Unterscheidung zwischen haptischen und Tasteneingaben in der Multi-ERSA-Studie (sh. Abschnitt 7.5). Für PAC-U und SSEE sind jeweils Werte zu beiden Aufgaben angegeben.

PAC-U	SSEE	COM-C	COM-D	QUESI
0.1363	0.7669	0.9223	0.1116	0.6323

**Tabelle A.6.:**  $p$ -Werte aus Kruskal-Wallis-Tests zur Unterscheidung zwischen den drei Eingabevarianten in der Vergleichsstudie mit dem Franka Desk Interface (sh. Abschnitt 7.5).

Ziel Markieren	Greifer Auslösen	Kamera Auslösen	Verbinden	Zahl Eingeben
0.0004	0.1660	0.4380	0.0148	0.0054

**Tabelle A.7.:**  $p$ -Werte aus Friedman-Tests für Unterschiede in der Bewertung verschiedener Gestenvorschläge aus der Vorstudie (sh. Abschnitt 4.3). Für Operationen mit  $p < 0.05$  folgen paarweise ausdifferenzierte Tabellen.

	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
1	0.1548	0.0088	1.0	0.1146
2		1.0	1.0	1.0
3			0.5317	1.0
4				1.0

**Tabelle A.8.:**  $p$ -Werte aus Friedman-Test für Unterschiede in der Bewertung zu Vorschlägen zum Markieren von Zielen aus der Vorstudie (sh. Abschnitt 4.3). Vorschläge sind: **1:** Ruckbewegung nach unten, **2:** Ruckbewegung horizontal, **3:** visuelle Geste „Daumen hoch“, **4:** Kreuz führen, **5:** visuelle Geste Kreuz zeichnen.

	<b>2</b>	<b>3</b>	<b>4</b>
1	0.5945	1.0	0.0469
2		1.0	1.0
3			0.7854

**Tabelle A.9.:**  $p$ -Werte aus Friedman-Test für Unterschiede in der Bewertung zu Vorschlägen zum Verbinden aus der Vorstudie (sh. Abschnitt 4.3). Vorschläge sind: **1:** visuelle Geste Kreis aus Daumen und Zeigefinger, **2:** visuelle Geste Kreis zeichnen, **3:** Kreis führen, **4:** Ellenbogengelenk des Roboters einknicken.

	<b>2</b>	<b>3</b>
1	1.0	0.0898
2		0.0446

**Tabelle A.10.:**  $p$ -Werte aus Friedman-Test für Unterschiede in der Bewertung zu Vorschlägen zur Zahleneingabe aus der Vorstudie (sh. Abschnitt 4.3). Vorschläge sind: **1:**  $n$ -maliges Antippen, **2:** visuelle Geste Anzahl Finger, **3:** abgestufte Drehung Handgelenk.

## ANHANG B

---

### Vorstudie zu haptischen Interaktionen

---

Im Folgenden sind die Fragen der Onlineumfrage zu haptischen Interaktionen (sh. Abschnitt 4.3) als Screenshots reproduziert.

## Anhang B. Vorstudie zu haptischen Interaktionen

---

### Einleitung

Meine Forschung beschäftigt sich mit der intuitiven Programmierung von Robotern. Nutzende führen den Roboter dazu Schritt für Schritt und Position für Position durch die Aufgabe. Es können auch Fallunterscheidungen für verschiedene mögliche Objekte eingefügt werden. Bei der Programmierung gibt es verschiedene Operationen, die durch Bewegungen am Roboter (haptische "Kommandos") ausgelöst werden sollen.

Die Interaktionen sind dabei zunächst folgendermaßen beschränkt: Der Roboter (ein Tischarm mit Greifer als Werkzeug und einer daran montierten Kamera) darf auf beliebige Weise bewegt werden, am Greifer oder am Arm selbst, mit jeder Bewegungs- und Drehrichtung. Die Interaktion darf beliebig kurz oder lang sein.

In Teil 1 der Umfrage werden Sie gebeten, sich jeweils eine Interaktion für die fünf Operationen zu überlegen. In Teil 2 können Sie für die Operationen jeweils verschiedene Interaktionsvorschläge bewerten, wie gut sie dazu passen. In Teil 3 haben Sie schließlich nochmal die Gelegenheit, jeweils die insgesamt passendste Interaktion zu nennen.

Die Teilnahme an der Umfrage dauert insgesamt ca. 25 Minuten, mit ca. 15 Minuten für Teil 1, ca. 5 Minuten für Teil 2, und ca. 5 Minuten für Teil 3. Sie können die Umfrage auch an einer beliebigen Stelle unterbrechen und zu einem späteren Zeitpunkt fortsetzen; Dazu einfach vor der Unterbrechung den Code rechts oben notieren.

### Abbildung B.1.: Einführungstext der Onlineumfrage

### Eigene Angaben

Wie würden Sie sich selbst beschreiben? \*

- Männlich
- Weiblich
- Divers
- Keine Angabe

Bitte geben Sie Ihr Alter in Jahren an: \*

Wieviel Erfahrung haben Sie im Umgang mit Robotern? \*

- 0 - Keine
- 1 - Wenig
- 2 - Einige
- 3 - Viel

### Abbildung B.2.: Erfassung demographischer Daten

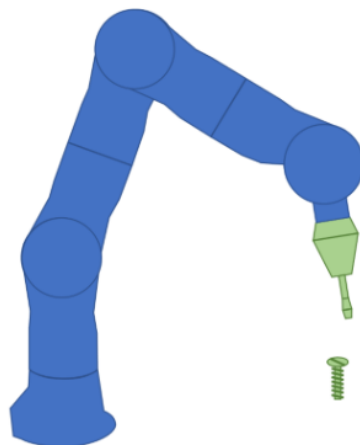
---

## Teil 1: Interaktionen beschreiben

Im Folgenden werden fünf verschiedene Operationen bei der Programmierung des Roboters beschrieben. Bitte überlegen Sie sich für jede dieser Operationen, wie sie sie durch Bewegung des Roboters auslösen würden. Des Weiteren geben Sie bitte eine Merkhilfe (eine "Eselsbrücke") für diese Interaktion an.

Die Beschreibung sollte detailliert genug sein, dass eine dritte Person damit die Bewegung grob reproduzieren könnte.

### Beispiel:

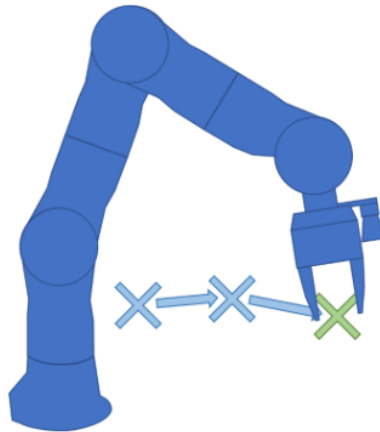


Operation: auslösen, dass mit einem Schraubwerkzeug eine Schraube angezogen wird.  
Interaktion: Drehe das Werkzeug über der Schraube eine halbe Umdrehung im Uhrzeigersinn um das letzte Gelenk.  
Merkhilfe: Deute den Schraubvorgang an.

**Abbildung B.3.:** Einführung für den ersten Teil der Umfrage ("Beschreiben") mit Beispiel

**Teil 1: Interaktionen beschreiben**

Nächstes Ziel markieren



Um Bewegungen zu programmieren, wird der Roboter entlang seines Pfades geführt, und dabei wiederholt die jeweils aktuelle Position als nächstes Ziel markiert (als "Keypoint" gesetzt).

**Beschreiben Sie eine passende Bewegung zum Auslösen dieser Operation: \***

**Was ist eine Merkhilfe dafür? \***

**Abbildung B.4.:** Beschreibungsfrage zur Eingabe von Bewegungen

---

## Teil 1: Interaktionen beschreiben

Greifer öffnen/schließen



Den Greifer öffnen und schließen.

Hinweis: Sie können zwei einzelne Interaktionen beschreiben, oder eine, die je nach Situation die zutreffende Operation auslöst ("toggle").

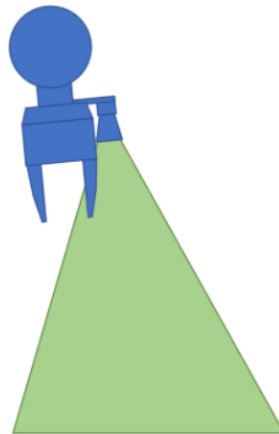
Beschreiben Sie eine passende Bewegung zum Auslösen dieser Operation: \*

Was ist eine Merkhilfe dafür? \*

**Abbildung B.5.:** Beschreibungsfrage zum Operieren des Greifers

**Teil 1: Interaktionen beschreiben**

**Bild aufnehmen**



Die Aufnahme eines Bildes mit der Kamera am Greifer auslösen.

**Beschreiben Sie eine passende Bewegung zum Auslösen dieser Operation: \* ⓘ**

**Was ist eine Merkhilfe dafür? \* ⓘ**

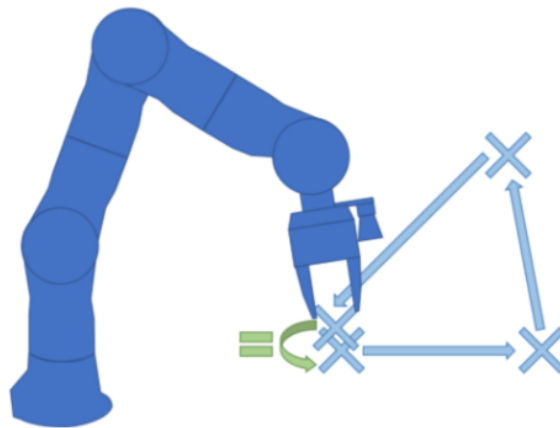
**Abbildung B.6.:** Beschreibungsfrage zum Auslösen der Kamera



---

## Teil 1: Interaktionen beschreiben

### Schleife schließen



Die generierten Programme sind intern eine Abfolge von Zuständen. Indem man einen Zustand mit einem früheren gleichsetzt, kann man also Schleifen schließen, sodass sich der Ablauf dazwischen wiederholt.

Beschreiben Sie eine passende Bewegung zum Auslösen dieser Operation: \* 


Was ist eine Merkhilfe dafür? \* 

Abbildung B.7.: Beschreibungsfrage zum Verbinden

**Teil 1: Interaktionen beschreiben**

Zahl eingeben



Eine kleine Zahl eingeben (bis etwa 5).  
Hinweis: Diese Operation wird z.B. benutzt, um zwischen verschiedenen Positionsvariablen auszuwählen

Beschreiben Sie eine passende Bewegung zum Auslösen dieser Operation: \*

Was ist eine Merkhilfe dafür? \*

**Abbildung B.8.:** Beschreibungsfrage zur Eingabe von Zahlen

**Teil 2: Interaktionen bewerten**

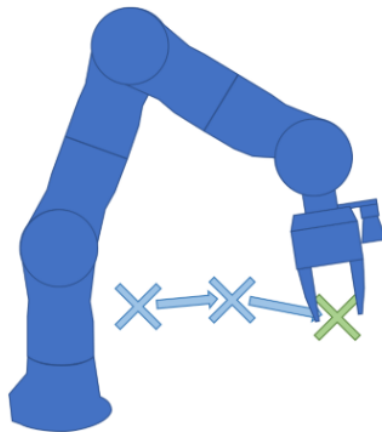
Im Folgenden werden Ihnen für die fünf Operationen eine Reihe von Interaktionen vorgeschlagen. Dies beinhaltet sowohl haptische Interaktionen als auch gestische, die vor der Kamera am Greifer mit einer Hand der Nutzenden ausgeführt werden. Bitte bewerten Sie für diese jeweils, wie gut sie ihres Empfindens zu der genannten Operation passen.

**Abbildung B.9.:** Einführung für den zweiten Teil der Umfrage ("Bewerten")

---

## Teil 2: Interaktionen bewerten

Nächstes Ziel markieren



Die aktuelle Position als nächstes Ziel markieren.

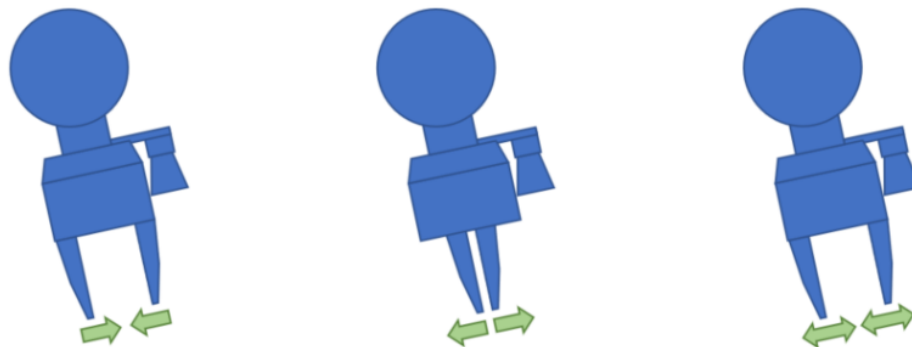
Wie passend finden Sie die folgenden Interaktionskandidaten für diese Operation? \*

	Perfekt	Gut	Kaum	Gar nicht
Greifer kurz nach unten stupsen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Greifer kurz hin und her wackeln	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Geste "Daumen hoch" vor Kamera	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Mit Greifer ein kleines Kreuz führen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Mit Zeigefinger ein Kreuz zeichnen vor Kamera	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

**Abbildung B.10.:** Bewertungsfrage zur Eingabe von Bewegungen inklusive Vorschläge

Teil 2: Interaktionen bewerten

Greifer öffnen/schließen



Den Greifer öffnen und schließen.

Wie passend finden Sie die folgenden Interaktionskandidaten für diese Operation? \*

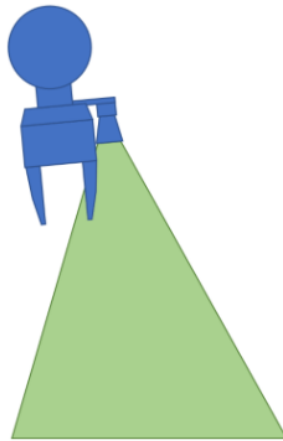
	Perfekt	Gut	Kaum	Gar nicht
Schließen: Greifer kurz nach unten stupsen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Öffnen: Greifer kurz nach oben stupsen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Öffnen: Greifer kurz hin und her wackeln	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Schließen: Daumen und übrige Finger flach zusammengedrücken vor Kamera	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Öffnen: Daumen und übrige Finger aus flach zusammengedrückter Stellung öffnen vor Kamera	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Toggle: Letztes Gelenk kurz im und gegen den Uhrzeigersinn drehen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Abbildung B.11.: Bewertungsfrage zum Operieren des Greifers inklusive Vorschläge

---

## Teil 2: Interaktionen bewerten

### Bild aufnehmen



Die Aufnahme eines Bildes mit der Kamera am Greifer auslösen.

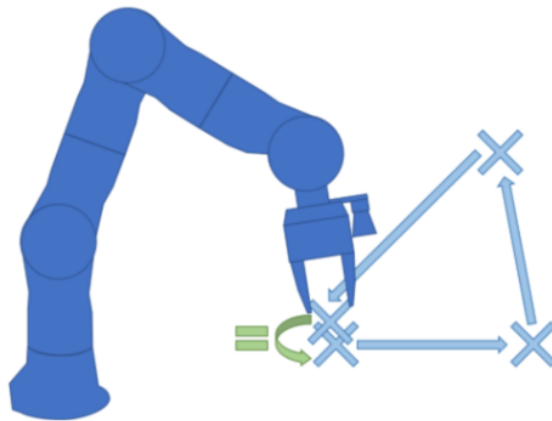
Wie passend finden Sie die folgenden Interaktionskandidaten für diese Operation? \*

	Perfekt	Gut	Kaum	Gar nicht
Mit flacher Hand wischen vor Kamera ("Blende")	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Ausgestreckten Zeigefinger einknicken ("Auslöser") vor Kamera	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Zeigefinger und Daumen im rechten Winkel halten ("Bilderrahmen") vor Kamera	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Greifer in kleinem Quadrat flach zur Bildebene führen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

**Abbildung B.12.:** Bewertungsfrage zum Auslösen der Kamera inklusive Vorschläge

Teil 2: Interaktionen bewerten

Schleife schließen



Durch Gleichsetzen des aktuellen Zustands des Programms mit einem anderen, bereits existierenden eine Schleife schließen.

Wie passend finden Sie die folgenden Interaktionskandidaten für diese Operation? \*

	Perfekt	Gut	Kaum	Gar nicht
Daumen und Zeigefinger zu Kreis schließen vor Kamera	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Mit Zeigefinger Kreis zeichnen vor Kamera	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Greifer in kleinem Kreis führen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Ellenbogengelenk des Roboters kurz einknicken	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Abbildung B.13.: Bewertungsfrage zum Verbinden inklusive Vorschläge

---

## Teil 2: Interaktionen bewerten

### Zahl eingeben



Eine kleine Zahl eingeben (bis etwa 5).

Wie passend finden Sie die folgenden Interaktionskandidaten für diese Operation? \*

	Perfekt	Gut	Kaum	Gar nicht
Roboter so oft antippen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
So viele Finger vor Kamera zeigen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Letztes Gelenk um Zahl mal (Pi geteilt durch größtmögliche Zahl) weiter drehen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Abbildung B.14.: Bewertungsfrage zur Eingabe von Bewegungen inklusive Vorschläge

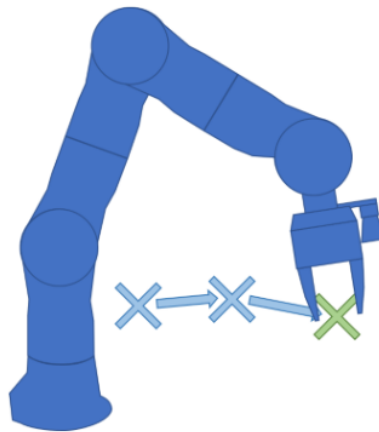
## Teil 3: Favoriten nennen

Im Folgenden können Sie zu jeder der Operationen angeben, welche Interaktion Sie nach Beantwortung der Umfrage am passendsten finden. Das können haptische und/oder gestische Interaktionen sein, sowohl die von Ihnen anfangs beschriebenen, als auch die im zweiten Teil vorgeschlagenen Kandidaten, als auch beliebige andere.

Abbildung B.15.: Einführung für den dritten Teil der Umfrage ("Favoriten nennen")

**Teil 3: Favoriten nennen**

Nächstes Ziel markieren



Die aktuelle Position als nächstes Ziel markieren.

Welche Interaktion (haptisch und/oder gestisch) finden Sie für diese Operation insgesamt am passendsten? \* ⓘ

**Abbildung B.16.:** Favoritenabfrage zur Eingabe von Bewegungen (Fragen für weitere Interaktionen analog)



Die deutschsprachigen Versionen der MINERIC-Fragebögen<sup>1</sup> nach [113], die zur in Kapitel 7 beschriebenen Evaluation verwendet wurden, sind auf den folgenden Seiten abgebildet. Dabei handelt es sich beispielhaft um die Fragebögen zur ERSA-Studie. In der Studie zu Multi-ERSA kamen SSEE und PAC-U jeweils zweimal vor, entsprechend der zwei Einzelaufgaben.

---

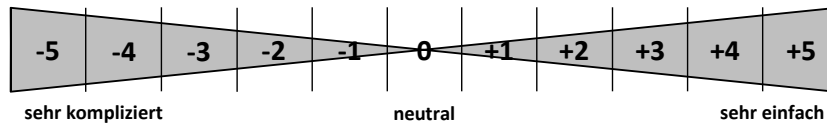
<sup>1</sup><https://www.ai3.uni-bayreuth.de/de/software/evaluation-toolkit/index.html>, besucht: 26.01.2023

**Fragebogen F**

Wie viel Erfahrung haben Sie mit...

	Keine	Wenig	Einige	Viel
<b>Programmierung im Allgemeinen?</b>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Falls zutreffend, welche?				
<b>Programmierung / Bedienung von Robotern?</b>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Falls zutreffend, welche?				

Wie schätzen Sie die Komplexität bei der Bedienung des Systems ein?

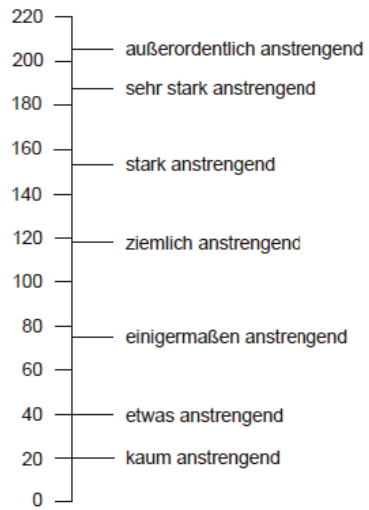


**Abbildung C.1.:** Fragebogen zu Vorerfahrung in Programmierung und Robotik, sowie erwartete Komplexität COM-E

---

**Fragebogen F**

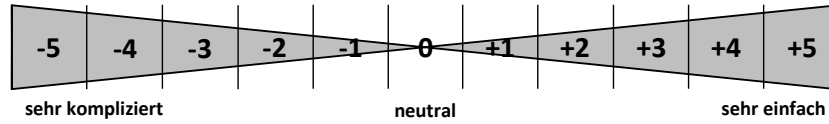
**Bitte kreuzen Sie auf der folgenden Skalen Ihre Gesamtbewertung bezüglich der Anstrengung bei der Lösung der Aufgabe an.**



**Abbildung C.2.:** SSEE-Fragebogen zur subjektiven mentalen Belastung

Fragebogen F

Wie schätzen Sie die Komplexität bei der Bedienung des Systems ein?



Wie schätzen Sie die Benutzung des Systems ein?

	trifft gar nicht zu	trifft wenig zu	teils, teils	trifft ziemlich zu	trifft völlig zu
Ich konnte das System benutzen, ohne darüber nachzudenken.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich habe erreicht, was ich mit dem System erreichen wollte.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Mir war sofort klar, wie das System funktioniert.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Der Umgang mit dem System erschien mir vertraut.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Bei der Benutzung des Systems sind keine Probleme aufgetreten.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Die Systembenutzung war unkompliziert.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich konnte meine Ziele so erreichen, wie ich es mir vorgestellt hatte.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Es fiel mir von Anfang an leicht, das System zu benutzen.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Mir war immer klar, was ich tun musste, um das System zu benutzen.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Die Benutzung des Systems verlief reibungslos.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich musste mich kaum auf die Benutzung des Systems konzentrieren.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Das System half mir dabei, meine Ziele vollständig zu erreichen.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Die Benutzung des Systems war mir auf Anhieb klar.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Ich tat automatisch das Richtige, um mein Ziel zu erreichen.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Haben Sie weitere Anmerkungen zum System / zur Studie?

Herzlichen Dank!

Interview aufgenommen am: ..... um: .....

Abbildung C.3.: Fragebogen zu wahrgenommener Komplexität COM-C und QUESI-Aussagen





2.1. Übersicht über verschiedene Varianten der Poseneingabe, welche Ressourcen sie seitens des Systems voraussetzen, und welche Fehler damit an sich möglich sind. Die mit ✗ markierten Einträge bedeuten, dass Fehler dieser Art bei einer bestimmten Eingabeart automatisch vermieden werden können; So findet die kinästhetische Eingabe etwa immer in der realen Umgebung statt, sodass Posen, die mit der Umwelt kollidieren, auffallen. Durch zusätzliches Wissen (seitens der Nutzenden oder seitens des Systems) können Fehler erkannt werden. In der untersten Zeile ist aufgeführt, welches Zusatzwissen zur Vermeidung der einzelnen Fehlerarten benötigt wird. Zusätzlich sind zwei häufige Arten markiert, wie den Nutzenden dieses Wissen an die Hand gegeben wird: Mit einer Visualisierung des Roboters, oder einer vollständigen Simulation bzw. der Durchführung On-line, am realen System. Die kinästhetische Führung benötigt zwar seitens des Systems vergleichsweise viele Ressourcen, erlaubt dafür aber von Haus aus weniger Fehler. . . . .	14
2.2. Übersicht über Varianten der Programmeingabe. Die roten, negativ bewerteten Eigenschaften zeigen die jeweiligen Einschränkungen existierender Ansätze auf. Gesucht ist ein Verfahren, das in allen Aspekten zumindest akzeptabel ist. Die visuelle Programmierung, die für die Zielsetzung dieser Arbeit ansonsten am besten zu bewerten ist, hat den oben erwähnten Nachteil zusätzlicher Eingabegeräte und deren Wechsel, was die in dieser Arbeit vorgestellte Methode umgehen möchte. Es geht also um den Entwurf eines Systems, das ähnliche Eigenschaften ohne eine grafische Schnittstelle erzielt. . . . .	18

2.3. Übersicht über Automatenmodelle in verwandten Arbeiten in der Roboterprogrammierung. Die Spalte „Kategorie“ gibt die Einordnung gemäß dem Fließtext an. In der Spalte „Bezeichnung“ ist der Titel des verwendeten Modells oder des Frameworks angegeben (falls von der Kategoriebezeichnung verschieden). Die Spalte „Programmierung“ gibt ggfs. wieder, auf welche Anwendenden die Arbeiten abzielen, oder andere Details zum Programmiervorgang – Falls nichts genaueres angegeben ist, wird von einer Verwendung durch Robotikexperten ausgegangen. Die Spalte „Ref.“ gibt die Referenzen zu einem Tabelleneintrag an. . . . .	28
2.3. Fortsetzung. . . . .	29
A.1. Vollständige Daten aus den Experimenten zu ERSA (sh. Abschnitt 7.1).	122
A.2. Vollständige Daten aus den Vergleichsexperimenten zu ERSA mit dem Franka Desk Interface. . . . .	123
A.3. Vollständige Daten aus den Experimenten zu Multi-ERSA (sh. Abschnitt 7.2).	124
A.4. $p$ -Werte aus Mann-Whitney-U-Tests zur Unterscheidung zwischen Experten und Nichtexperten (sh. Abschnitt 7.5). Bei der Multi-ERSA-Studie sind die Werte für PAC-U und SSEE der beiden Aufgaben angegeben. QUESI meint hier den erzielten Gesamtwert. . . . .	125
A.5. $p$ -Werte aus Mann-Whitney-U-Tests zur Unterscheidung zwischen haptischen und Tasteneingaben in der Multi-ERSA-Studie (sh. Abschnitt 7.5). Für PAC-U und SSEE sind jeweils Werte zu beiden Aufgaben angegeben.	125
A.6. $p$ -Werte aus Kruskal-Wallis-Tests zur Unterscheidung zwischen den drei Eingabevarianten in der Vergleichsstudie mit dem Franka Desk Interface (sh. Abschnitt 7.5). . . . .	125
A.7. $p$ -Werte aus Friedman-Tests für Unterschiede in der Bewertung verschiedener Gestenvorschläge aus der Vorstudie (sh. Abschnitt 4.3). Für Operationen mit $p < 0.05$ folgen paarweise ausdifferenzierte Tabellen. . . . .	126
A.8. $p$ -Werte aus Friedman-Test für Unterschiede in der Bewertung zu Vorschlägen zum Markieren von Zielen aus der Vorstudie (sh. Abschnitt 4.3). Vorschläge sind: <b>1</b> : Ruckbewegung nach unten, <b>2</b> : Ruckbewegung horizontal, <b>3</b> : visuelle Geste „Daumen hoch“, <b>4</b> : Kreuz führen, <b>5</b> : visuelle Geste Kreuz zeichnen. . . . .	126
A.9. $p$ -Werte aus Friedman-Test für Unterschiede in der Bewertung zu Vorschlägen zum Verbinden aus der Vorstudie (sh. Abschnitt 4.3). Vorschläge sind: <b>1</b> : visuelle Geste Kreis aus Daumen und Zeigefinger, <b>2</b> : visuelle Geste Kreis zeichnen, <b>3</b> : Kreis führen, <b>4</b> : Ellenbogengelenk des Roboters einknicken. . . . .	126
A.10. $p$ -Werte aus Friedman-Test für Unterschiede in der Bewertung zu Vorschlägen zur Zahleneingabe aus der Vorstudie (sh. Abschnitt 4.3). Vorschläge sind: <b>1</b> : $n$ -maliges Antippen, <b>2</b> : visuelle Geste Anzahl Finger, <b>3</b> : abgestufte Drehung Handgelenk. . . . .	126



---

## Abbildungsverzeichnis

---

1.1.	Konzeptgrafik zu klassischer Industrierobotik (links) gegenüber der Vision von Robotik in kleineren Unternehmen (rechts). Oben: typisches Verwendungsszenario. In der klassischen Industrierobotik ist der Roboter räumlich abgezüunt, und die vorwiegend textuelle Programmierung erfordert hohe Expertise. In der modernen Anwendung kann der Mensch sich im Arbeitsraum des (Leichtbau-) Roboters aufhalten, und die Programmierung soll auch für Nichtexperten unter Kenntnis der Aufgabe machbar sein. Unten: Zeitlicher Verlauf von Nutzungsphasen, nach [44]. In der klassischen Industrierobotik liegen zwischen den Phasen der Neukonfiguration oder -programmierung lange Produktionsperioden. In kleineren Unternehmen sind häufigere Aufgabenwechsel erforderlich. . . . .	3
1.2.	Kostenverteilung für eine kooperative Roboterzelle, übersetzt und adaptiert aus [123]. Betriebskosten stellen mit Abstand den größten Posten dar. . . . .	4
2.1.	Unterteilung der Roboterprogrammierung nach Zentrierungen (Roboter, Aufgabe, Führung, Benutzer) und deren Unterkategorien, nach [111] . . .	12
2.2.	Zwei Beispiele endlicher Automaten aus verwandten Arbeiten. Links: Ein in [129] per genetischen Algorithmen erzeugter Automat, um ein Objekt zu verschieben. Wie anhand der Benennung deutlich, repräsentieren die Zustände einzelne Arbeitsschritte des Roboters, und Zustandsübergänge finden jeweils nach Beendigung eines Schritts statt. Rechts: Beispiel aus [114], das die Umsetzung von Kontrollstrukturen illustriert. Automatenzustände stellen hier direkt den Aufgabenzustand dar, und Transitionen definieren einen Zustandsübergang unter in dem Moment wahrgenommenen Bedingungen. In dem Fall sind auch die Ausgaben, d.h. die Roboteraktionen, an den Transitionen annotiert. . . . .	21

---

2.3.	Oben: Automatenerweiterung aus [81]. An den Transitionen sind komplexe Bedingungen über den zusätzlichen Variablen notiert, die kontinuierlich überwacht werden. Unten: Timed Automaton aus [126]. Die reellwertigen Zeitvariablen laufen während der Ausführung mit, und werden entlang der Transitionen geprüft bzw. zurückgesetzt. . . . .	23
2.4.	Zwei Beispiele für komplexere Programme unter Verwendung von Statecharts (bzw. Varianten davon), beide aus dem Kontext des Roboterfußballs. Oben: Automat im grafischen Editor des RAFCON-Systems aus [27]. Unten: Darstellung aus [102]. . . . .	25
3.1.	Eine einfache Beispielaufgabe für die Verwendung von RSA. Ein Roboter verteilt Objekte zwischen Förderbändern um. Links ankommende Objekte werden nach ihrer Farbe auf eines der beiden ausgehenden Bänder rechts gelegt. Es sind sechs Posen $\bar{p}_0$ bis $\bar{p}_5$ markiert, die in einer einfachen Umsetzung relevant wären: die Aufgreif- und Ablageorte auf den drei Förderbändern, und jeweils eine Pose darüber, die eine kollisionsfreie Annäherung ermöglicht. . . . .	35
3.2.	RSA zur Beispielaufgabe aus Abbildung 3.1. Der Automat teilt sich zu Beginn in zwei Zweige, und am Ende beider Zweige wird durch eine Kante zurück zum Verzweigungszustand die Schleife geschlossen. An jeder Kante von $q$ zu $q'$ (d.h. zu jeder Transition $\delta(q, \sigma) = q'$ ) ist eine Zielpose gemäß $u(q, \sigma) = \bar{p}$ notiert (wobei Posen $\bar{p}'$ einen geschlossenen Greifer bei ansonsten gleicher Pose wie $\bar{p}$ symbolisieren). Insofern dabei $\sigma \neq \varepsilon$ ist, ist dieses ebenfalls an der Kante notiert, in der Form „ $\sigma$ :“ vor der entsprechenden Zielpose. Die Benennung der einzelnen Posen ist dabei identisch wie in Abbildung 3.1. . . . .	35
3.3.	Defizite des (einfachen) RSA-Modells, von links nach rechts: Ausdruck relativer Bewegungen z.B. für Schleifeninkremente, Ausdruck von relativ zu Objekten definierten Zielposen, z.B. um mit ungenauer Platzierung umzugehen, Speichern und anschließende Verwendung von zur Programmierzeit unbekanntem Posen (zur Laufzeit). . . . .	36
3.4.	Eine Beispielaufgabe für die Verwendung von ERSA. Ein Roboter palettiert Objekte auf einer Palette unbekannter Größe, die an einer gegebenen Ecke bereitgelegt wird. Die mit Zahlen nummerierten $\bar{p}_0$ und $\bar{p}_1$ stehen für feste Posen (Ecke der Palette und Ankunftsort der Objekte), die mit Buchstaben gekennzeichneten $\bar{p}_x$ und $\bar{p}_y$ für (relative) Transformationen, nämlich das Inkrement von einer Zeile bzw. Spalte zur nächsten. . . . .	39



3.8.	Beispiel für den verwobenen Programmierprozess und das Überschreiben aus der Ausführung. Der Automat aus den vorherigen Beispielen wurde vervollständigt, wie ursprünglich in Abbildung 3.2 dargestellt. Nun soll die Ablageposition für den unteren Zweig verändert werden. Dazu wird zunächst bis direkt vor der entsprechenden Bewegung ausgeführt (3.8a). Anschließend wird eine neue Aktion programmiert, in diesem Fall eine Bewegung zur neuen Zielpose $\bar{p}_6$ . Da keine Wahrnehmung spezifiziert wurde, wird diese Bewegung wieder an eine $\varepsilon$ -Transition zu einem neuen Zustand annotiert. Diese überschreibt die bisherige Transition (3.8b). Damit kann von dieser Stelle aus weiterprogrammiert werden. Der dahinter liegenden Teil des Zweigs wird unerreichbar und in der Folge aus dem Automaten entfernt. . . . .	45
4.1.	Skizze aus [153] zur dortigen Verwendung haptischer Eingaben. Neben der ursprünglichen Eingabe (links) und späteren Einstellung von Posenparametern (Mitte) wird dort eine Schlüsselgeste zur Übergabe von Objekten verwendet: Ein Auslenken des Roboters nach oben bzw. unten zum Öffnen bzw. Schließen des Greifers. . . . .	48
4.2.	Konzeptgrafiken aus [167] zu verschiedenen Varianten für eine Schlüsselgeste zum Öffnen und Schließen des Greifers. Es werden dort kraftbasierte und positionsbasierte Richtungswechsel vorgeschlagen (Mitte), in beliebigen Achsen oder solchen des NSA-Koordinatensystems (links), sowie rotationsbasierte Richtungswechsel (rechts). Letztere Möglichkeit wurde in den Experimenten in der Arbeit verwendet. . . . .	49
4.3.	Varianten für bestimmte Richtungen bei Ruckbewegungen, von links nach rechts: in Weltkoordinaten (Beispiel: nach unten), in NSA-Koordinaten (Beispiel: in Richtung der A-Achse), im Koordinatensystem eines Objekts (Beispiel: in einer Achsrichtung), in Richtung des Objekts selbst. Die Referenzkoordinatensysteme sind jeweils angedeutet (soweit zutreffend). . .	52
4.4.	In der Onlineumfrage verwendete Skizzen, von links nach rechts: Bewegung, Greiferoperation, Kameraaufnahme, Verbinden und Zahleneingabe	57
4.5.	Daten zu Teilnehmenden der Vorstudie. Links: Angegebenes Alter in Jahren. Rechts: Prozentuale Verteilung der angegebenen Vorerfahrung im Umgang mit Robotern auf der Skala von 0 („keine“) bis 3 („viel“). . . .	58
4.6.	Bewertungen der Interaktionsvorschläge für die Operation zum Markieren einer Zielpose, von 0 („passt gar nicht“) bis 3 („passt perfekt“). . . . .	59
4.7.	Bewertungen der Interaktionsvorschläge zum Bedienen des Greifers. . . .	60
4.8.	Bewertungen der Interaktionsvorschläge zum Auslösen der Kamera. . . .	61
4.9.	Bewertungen der Interaktionsvorschläge zum Schließen von Schleifen (Verbinden). . . . .	62
4.10.	Bewertungen der Interaktionsvorschläge zur Eingabe einer Variablennummer, d.h. einer Zahl $n$ . . . . .	63

- 
- 4.11. Links: Konzeptgrafik der im ERSA-System verwendeten haptischen Interaktionen. Pinngesten für Bewegungen rot, horizontale Kreisgeste zum Verbinden grün, Drehung im letzten Gelenk zum Speichern/Laden orange. (Eine optionale Nickgeste im vorletzten Gelenk zum Auslösen der Kamera, hier türkis, wurde probenhalber implementiert, aber nicht in den Experimenten verwendet.) Rechts: Merkhilfe für die Benutzerstudie. Diese wurde auf den Roboter aufgeklebt (der obere Teil zu Bewegungen und Verbinden auf das letzte Gelenk, der Teil zum Speichern und Laden von Posenvariablen auf den Flansch). . . . . 66
- 5.1. Links: Versuchsaufbau, wie bei der ERSA-Benutzerstudie. Objekte werden im schwarz markierten Areal angeliefert und müssen entweder in den Behälter eingetaucht und an die genaue Aufnahmeposition zurückgelegt werden, oder an der ersten markierten Stelle platziert werden, die noch frei ist. Rechts: Anzahl an Zuständen über der Anzahl an programmierten Schritten in einem Beispieldurchlauf. In jedem Schritt wird ein neuer Zustand hinzugefügt. In Schritt 14, 25 und 30 wird ein neuer Zustand generiert, aber anschließend werden durch die Unifikation  $m+1 = 4$  Zustände als unerreichbar entfernt (genau die Zustände entlang der drei überlappenden Transitionen, die redundant zu ihrer ursprünglichen Repräsentation sind). . . . . 75
- 5.2. Teilautomaten vor (5.2a) und nach (5.2b) der ersten Unifikation. Direkt vor der Vereinigung hat das ganz linke Zustandspaar eine Bestätigungstiefe von  $3 \geq m = 3$ . Von diesem Paar ausgehend, werden Zustandspaare vereinigt, was jeweils die Vereinigung der direkten Nachfolgerzustände nach sich zieht. Der aktuelle Zustand nach der Unifikation ist der Repräsentant für den aktuellen Zustand vor der Unifikation (jeweils mit einer doppelten Umrandung markiert). . . . . 76
- 5.3. Ein weiterer partieller Automat für die gleiche Beispielaufgabe, vor (5.3a) und nach (5.3b) Synthese der Schleife über Ablageorte. Die abgebildete Kette an Zuständen und Übergängen erreicht eine ausreichende Bestätigungstiefe (genauer: die ersten beiden Zustände darin). Die Zustände werden entlang der Kette paarweise vereinigt, was dazu führt, dass am Ende nur ein einzelner Zustand übrig bleibt, mit einer Transition zu sich selbst mit der entsprechenden relativen Bewegung. Durch die Übereinstimmung entlang der Kette sind auch andere Zustandspaare als nicht widersprüchlich markiert, aber deren Bestätigungstiefe führt weniger schnell zu einer Unifikation. Z.B. könnte auch ein Schleifenrumpf aus jeweils zwei der Übergänge gebildet werden. . . . . 77

- 5.4. Teilautomaten im Fall, dass die äußere Schleife nicht sofort geschlossen wurde, bzw. in 5.4a genauer genommen nach der Rückkehr in den gemeinsamen Verzweigungszustand. Die Schleifen der beiden Zweige können nun nicht sofort beide geschlossen werden. Je nach Reihenfolge können zwei Fälle entstehen: Bei erneuter Programmierung der ersten Aktionen aus dem unteren Zweig wird die Schleife für diesen geschlossen (5.4b). Bei erneuter Programmierung der ersten Aktionen aus dem oberen Zweig wird stattdessen die Sequenz der beiden Zweige zu einer großen Schleife zusammengelegt (5.4c). In beiden Fällen kann in der Folge durch weitere Überlappung die korrekte Gesamtstruktur erzeugt werden. . . . . 78
- 6.1. Partieller Multi-ERSA am Beispiel für zwei Roboterarme. Transitionen sind wie üblich beschriftet, mit dem Zusatz, dass das Subskript am Doppelpunkt die betroffene Transitionsfunktion angibt (d.h. „:1“ markiert einen Übergang gemäß  $\delta_1$ ). Links des Doppelpunkts steht die erforderliche Eingabe (ein  $\sigma \in \Sigma$  oder  $\varepsilon$ ) sowie das erforderliche Signal (ein  $b \in [m]$ ) falls zutreffend. Falls kein Signal erforderlich ist, wird der zweite Teil (d.h.  $\varepsilon$ ) weggelassen. Auf der rechten Seite steht das aus der Updatefunktion definierte Kommando, sowie ggfs. das zu inkrementierende Signal (das ansonsten, also im Fall  $\varepsilon$ , wiederum weggelassen wird). Zustände sind per Superskript markiert, zu welchen Teilautomaten sie gehören, d.h. ein Superskript von „1,2“ bedeutet etwa, dass der Zustand sowohl in  $Q_1$  als auch in  $Q_2$  enthalten ist. . . . . 87
- 6.2. Übersicht über haptische Gesten im Rahmen der Synchronisation, wie in der Benutzerstudie (sh. Abschnitt 7.2) als Merkhilfe verwendet: Weiterhin Anpinngeste vertikal nach unten für (absolute) Bewegung, Anpinngeste in Richtung der anderen Roboterbasis für einen Sendepunkt, von der anderen Roboterbasis weg für einen Wartepunkt (Präzedenz oder Rendezvous), senkrecht zu den beiden anderen Achsen für synchrone Aktionen. (In der Benutzerstudie wurde außerdem der Greifer auf eine Drehbewegung um das letzte Gelenk des Roboters abgebildet.) . . . . . 89
- 7.1. Links: experimenteller Aufbau, mit angedeuteten Schritten analog zu Abbildung 7.2 eingefärbt. Vor dem Roboter werden Objekte im schwarz umrandeten Bereich abgelegt. Je nach Objektart werden diese entweder in den Behälter rechts eingetaucht und zurückgelegt, oder an den mit einem schwarzen Punkt markierten Ablageorten aufgereiht. Rechts: Letzte Segmente des Roboters mit den darauf angebrachten Piktogrammen (sh. Abbildung 4.11) zur Erinnerung an die in Kapitel 4 beschriebenen haptischen Interaktionen (Ruckbewegungen nach unten/oben für absolute/relative Bewegungen, horizontale Kreisbewegung zum Verbinden, Ruckbewegungen nach rechts/links im letzten Gelenk zum Speichern/Laden von Posenvariablen). . . . . 93

7.2. Vereinfachter Automat für die Aufgabe aus der ERSA-Studie. Kanten, die mit „ $\sigma$ “ beschriftet sind, werden nur dann ausgeführt, wenn ein entsprechender Stimulus aus der Objektwahrnehmung auftritt. Für die einzige auftretende Posenvariable wird $d$ verwendet (statt der entsprechenden Komponente $d_1$ ). Die jeweiligen Updates sind ähnlich einer Pseudocode-Zuweisung notiert, z.B. „ $d \leftarrow p$ “ für das Abspeichern der aktuellen Roboterpose in der Posenvariable. Die einzelnen Programmiermöglichkeiten aus der Automatendefinition sind abgedeckt: Verzweigung und Bewegung relativ zur aktuellen Pose des wahrgenommenen Objekts zum Aufnehmen (rot), Speichern der Aufnahmepose für gelbe Teeschachteln (oberer Pfad, hellblau), Eintauchen in den Behälter mit absoluten Bewegungen (grün) und Anfahren der gespeicherten Pose aus der Posenvariablen (magenta), Schleife über Ablagepositionen mit relativen Bewegungen für fliederfarbene Teeschachteln (unterer Pfad, dunkelblau). . . . .	94
7.3. Ergebnisse aus der Studie zum ERSA-System. Von links nach rechts: PAC-U-Wert zum Erfolg der Programmierung (Abstand vom Optimum, d.h. niedriger ist besser), SSEE-Wert zur mentalen Belastung (niedriger ist besser), wahrgenommene Komplexität COM-C (positiv bedeutet einfach, negativ bedeutet komplex), und Zustimmung zu den positiven QUESI-Aussagen (höher ist besser). . . . .	97
7.4. Ergebnisse mit dem ERSA-System für PAC-U, SSEE, COM-C und QUESI-Durchschnitt, aufgeteilt in Robotikexperten („Exp.“) und Robotiknichtexperten („N.-e.“). . . . .	98
7.5. Experimenteller Aufbau zur Evaluation des Multi-ERSA-Systems, bestehend aus zwei Leichtbaurobotern, einer bedruckten Matte mit den relevanten Ablageorten, und den verwendeten Objekten auf der linken Seite. Die Roboter sind jeweils mit einem Greifer und einem Tastenblock ausgestattet. . . . .	99
7.6. Tastensymbole für Synchronisationsoperationen, von links nach rechts mit zugrunde liegender Assoziation: Sendepunkt (Senden eines Signals), Wartepunkt (Warten auf ein Signal), Rendezvous (Erstellen einer Synchronisationsbarriere), und synchrone Aktion (gemeinsame Ausführung/Kante). . . . .	99
7.7. Ausgangs- und Zielzustand der ersten Programmieraufgabe. Ein Bogen muss mithilfe der beiden neben der Basisplatte verbleibenden Blöcke aufgebaut werden, in korrekter Reihenfolge. . . . .	100
7.8. Vereinfachter Automat für Aufgabe 1. Im oberen Teil des Automaten wird vom ersten Roboter der zweite kleinere Bauklotz (in Abbildung 7.7 grün, vorne links) neben dem bereits vorhandenen abgelegt und anschließend ein Signal gesendet. Im unteren Automatenteil wartet der zweite Roboter zunächst auf dieses Signal, d.h. bis die zweite „Säule“ platziert wurde und der Arbeitsraum wieder frei ist. Erst danach kann der „Balken“, der größere Bauklotz, auf den beiden kleineren abgelegt werden. . . . .	100

---

7.9. Ausgangs- und Zielzustand der zweiten Programmieraufgabe. Zunächst muss in korrekter Reihenfolge ein Quadrat aus den vier flachen Verbindungselementen gelegt, anschließend die gesamte Platte von beiden Robotern gemeinsam nach hinten bewegt werden. . . . .	101
7.10. Vereinfachter Automat für Aufgabe 2. Zunächst wird von beiden Robotern unabhängig voneinander jeweils ein Verbindungselement in der unteren Ebene platziert, auf gegenüberliegenden Seiten des Quadrats. Es folgt ein Rendezvous, um sicherzustellen, dass die untere Ebene abgeschlossen ist, bevor Verbindungselemente in der oberen Ebene platziert werden dürfen. Nach dem Rendezvous wird dieser Schritt wieder von beiden Robotern unabhängig voneinander ausgeführt. Zuletzt greifen beide Roboter gemeinsam die Basisplatte, bewegen sie entlang einer gemeinsamen Kante im Automaten durch eine synchrone Bewegung nach hinten und legen sie dort wieder ab. . . . .	101
7.11. MINERIC-Ergebnisse über alle Teilnehmenden. A1 und A2 bezeichnen die erste und zweite Programmieraufgabe. . . . .	102
7.12. Ergebnisse für PAC-U, SSEE und COM-C, aufgeteilt in Robotikexperten und Robotiknichtexperten. . . . .	103
7.13. Ergebnisse der Studie zum Mehrarmsystem, aufgeteilt in haptische Eingaben („Hapt.“) und Tasteneingaben („Tast.“). . . . .	104
7.14. Ergebnisse der Studie zum Mehrarmsystem innerhalb der Nichtexperten-Gruppe, aufgeteilt in haptische und Tasteneingaben. . . . .	105
7.15. Ergebnisse der Studie zum Mehrarmsystem innerhalb der Expertengruppe, aufgeteilt in haptische und Tasteneingaben. . . . .	106
7.16. Aufbau in der Vergleichsstudie, links: bei Verwendung des Surface-Tablets (Aufnahme von vorne), rechts: bei Verwendung des Desktop-PCs (Aufnahme von der Seite). . . . .	107
7.17. Ergebnisse unter dem ERSA-System und der Franka Desk Schnittstelle für PAC-U, SSEE, COM-C und QUESI-Durchschnitt im Vergleich. . . . .	108
7.18. Ergebnisse unter der Franka Desk Schnittstelle für PAC-U, SSEE, COM-C und QUESI-Durchschnitt, aufgeteilt in Robotikexperten und Robotiknichtexperten. . . . .	109
7.19. Ergebnisse unter der Franka Desk Schnittstelle für PAC-U, SSEE, COM-C und QUESI-Durchschnitt, aufgeteilt in die drei Eingabevarianten: Eingabe nur über den Desktop-Rechner auf dem Nebentisch („Deskt.“), über den Desktop-Rechner unter Zuhilfenahme der Pilot-Schnittstelle auf dem Roboter („D+P“) und über das Tablet direkt neben dem Roboter („Tabl.“). . . . .	110
B.1. Einführungstext der Onlineumfrage . . . . .	128
B.2. Erfassung demographischer Daten . . . . .	128
B.3. Einführung für den ersten Teil der Umfrage ("Beschreiben") mit Beispiel . . . . .	129
B.4. Beschreibungsfrage zur Eingabe von Bewegungen . . . . .	130
B.5. Beschreibungsfrage zum Operieren des Greifers . . . . .	131



---

B.6. Beschreibungsfrage zum Auslösen der Kamera . . . . .	132
B.7. Beschreibungsfrage zum Verbinden . . . . .	133
B.8. Beschreibungsfrage zur Eingabe von Zahlen . . . . .	134
B.9. Einführung für den zweiten Teil der Umfrage ("Bewerten") . . . . .	134
B.10. Bewertungsfrage zur Eingabe von Bewegungen inklusive Vorschläge . . .	135
B.11. Bewertungsfrage zum Operieren des Greifers inklusive Vorschläge . . . . .	136
B.12. Bewertungsfrage zum Auslösen der Kamera inklusive Vorschläge . . . . .	137
B.13. Bewertungsfrage zum Verbinden inklusive Vorschläge . . . . .	138
B.14. Bewertungsfrage zur Eingabe von Bewegungen inklusive Vorschläge . . .	139
B.15. Einführung für den dritten Teil der Umfrage ("Favoriten nennen") . . . . .	139
B.16. Favoritenabfrage zur Eingabe von Bewegungen (Fragen für weitere Inter- aktionen analog) . . . . .	140
C.1. Fragebogen zu Vorerfahrung in Programmierung und Robotik, sowie er- wartete Komplexität COM-E . . . . .	142
C.2. SSEE-Fragebogen zur subjektiven mentalen Belastung . . . . .	143
C.3. Fragebogen zu wahrgenommener Komplexität COM-C und QUESI-Aussagen	144
C.4. PAC-U-Beobachtungsbogen zu Erfolg der Programmierung . . . . .	145



- [1] Ajaykumar, Gopika, Maureen Steele und Chien Ming Huang: *A Survey on End-User Robot Programming*. ACM Computing Surveys, 54(8), 2022. <https://doi.org/10.1145/3466819>.
- [2] Akgun, Baris, Maya Cakmak, Jae Wook Yoo und Andrea Lockerd Thomaz: *Trajectories and keyframes for kinesthetic teaching: a human-robot interaction perspective*. In: *Proceedings of the seventh annual ACM/IEEE international conference on Human-Robot Interaction*, Seiten 391–398. Association for Computing Machinery, 2012. <https://doi.org/10.1145/2157689.2157815>.
- [3] Akl, Ahmad und Shahrokh Valaee: *Accelerometer-based gesture recognition via dynamic-time warping, affinity propagation, & compressive sensing*. In: *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, Seiten 2270–2273, Dallas, 2010. IEEE. <https://doi.org/10.1109/ICASSP.2010.5495895>.
- [4] Albu-Schäffer, Alin, Christian Ott und Gerd Hirzinger: *A unified passivity-based control framework for position, torque and impedance control of flexible joint robots*. International Journal of Robotics Research, 26(1):23–39, 2007. <https://doi.org/10.1177/0278364907073776>.
- [5] Alexandrova, Sonya, Maya Cakmak, Kaijen Hsiao und Leila Takayama: *Robot Programming by Demonstration with Interactive Action Visualizations*. In: *Robotics: Science and Systems Foundation*, 2014. <https://doi.org/10.15607/rss.2014.x.048>.
- [6] Alexandrova, Sonya, Zachary Tatlock und Maya Cakmak: *RoboFlow: A flow-based visual programming language for mobile manipulation tasks*. In: *IEEE International Conference on Robotics and Automation*, Seiten 5537–5544. IEEE, 2015. <https://doi.org/10.1109/ICRA.2015.7139973>.
- [7] Alizadeh, Tohid und Milad Malekzadeh: *Identifying the relevant frames of reference in programming by demonstration using task-parameterized Gaussian mixture regression*. In: *2016 IEEE/SICE International Symposium on System Integration*

- (SII), Seiten 453–458, Sapporo, Japan, 2017. IEEE. <https://doi.org/10.1109/SII.2016.7844040>.
- [8] Alur, R., J. Esposito, M. Kim, V. Kumar und I. Lee: *Formal Modeling and Analysis of Hybrid Systems: A Case Study in Multi-robot Coordination*. In: Wing, Jeannette M., Jim Woodcock und Jim Davies (Herausgeber): *FM'99 — Formal Methods*, Seiten 212–232, Berlin, Heidelberg, 1999. Springer. [https://doi.org/10.1007/3-540-48119-2\\_14](https://doi.org/10.1007/3-540-48119-2_14).
- [9] Alur, Rajeev, Thao Dang, Joel Esposito, Yerang Hur, Franjo Ivancic, Vijay Kumar, Insup Lee, Pradyumna Mishra, George J. Pappas und Oleg Sokolsky: *Hierarchical modeling and verification of embedded systems*. *Proceedings of the IEEE*, 91(1):11–28, 2003. <https://doi.org/10.1109/JPROC.2002.805817>.
- [10] Andersen, Michael S., Rune S. Jensen, Thomas Bak und Michael M. Quottrup: *Motion planning in multi-robot systems using timed automata*. *IFAC Proceedings Volumes*, 37(8):597–602, 2004. [https://doi.org/10.1016/S1474-6670\(17\)32043-8](https://doi.org/10.1016/S1474-6670(17)32043-8).
- [11] André, Charles: *SyncCharts: A Visual Representation of Reactive Behaviors*. Technischer Bericht RR 96-56, I3S Sophia Antipolis, 1996.
- [12] Arai, Toshiaki und Frieder Stolzenburg: *Multiagent Systems Specification by UML Statecharts Aiming at Intelligent Manufacturing*. In: *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 1, AAMAS '02*, Seite 11–18, New York, NY, USA, 2002. Association for Computing Machinery. <https://doi.org/10.1145/544741.544745>.
- [13] Argall, Brenna D., Sonia Chernova, Manuela Veloso und Brett Browning: *A survey of robot learning from demonstration*. *Robotics and Autonomous Systems*, 57(5):469–483, 2009. <https://doi.org/10.1016/j.robot.2008.10.024>.
- [14] Arici, Tarik, Sait Celebi, Ali S. Aydin und Talha T. Temiz: *Robust gesture recognition using feature pre-processing and weighted dynamic time warping*. *Multimedia Tools and Applications*, 72(3):3045–3062, 2014. <https://doi.org/10.1007/s11042-013-1591-9>.
- [15] Bakala, Ewelina, Jorge Visca, Gonzalo Tejera, Andres Sere, Guillermo Amarin und Leonel Gomez-Sena: *Designing child-robot interaction with Robotito*. In: *2019 28th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. IEEE, 2019. <https://doi.org/10.1109/RO-MAN46459.2019.8956448>.
- [16] BCG: *Prognose zur Preisentwicklung eines Industrieroboters in den USA nach Posten bis zum Jahr 2025*. <https://de.statista.com/statistik/daten/studie/416525/umfrage/preisentwicklung-eines-industrieroboters/>, besucht: 31.03.2023.

- [17] Beeck, M. von der: *A Comparison of Statecharts Variants*. In: Langmaack, H., WP. de Roever und J. Vytopil (Herausgeber): *Formal Techniques in Real-Time and Fault-Tolerant Systems. FTRTFT ProCoS 1994. Lecture Notes in Computer Science*, Band 863, Seiten 128–148. Springer, Berlin, Heidelberg, 1994. [https://doi.org/10.1007/3-540-58468-4\\_163](https://doi.org/10.1007/3-540-58468-4_163).
- [18] Bender, Manfred, Martin Braun, Peter Rally und Oliver Scholtz: *Lightweight robots in manual assembly – best to start simply!* Technischer Bericht, Fraunhofer Institute for Industrial Engineering IAO, Stuttgart, 2016.
- [19] Biggs, Geoffrey und Bruce Macdonald: *A Survey of Robot Programming Systems*. In: *Australasian Conference on Robotics and Automation*, Band 1, Seiten 1–10, 2003.
- [20] Billard, Aude und Sylvain Calinon: *Robot programming by demonstration*. In: Siciliano, Bruno und Oussama Khatib (Herausgeber): *Handbook of Robotics*, Seiten 1371–1394. Springer, Berlin, Heidelberg, 1. Auflage, 2008. [https://doi.org/10.1007/978-3-540-30301-5\\_60](https://doi.org/10.1007/978-3-540-30301-5_60).
- [21] Bischoff, Rainer, Johannes Kurth, Günter Schreiber, Ralf Koeppe, Alin Albu-Schäffer, Alexander Beyer, Oliver Eiberger, Sami Haddadin, Andreas Stemmer, Gerhard Grunwald und Gerhard Hirzinger: *The KUKA-DLR Lightweight Robot arm - A new reference platform for robotics research and manufacturing*. In: *ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics)*, Band 2, Seiten 741–748, München, 2010. VDE.
- [22] Bobick, Aaron F. und Andrew D. Wilson: *A state-based approach to the representation and recognition of gesture*. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(12):1325–1337, 1997. <https://doi.org/10.1109/34.643892>.
- [23] Bohren, Jonathan, Radu Bogdan Rusu, E. Gil Jones, Eitan Marder-Eppstein, Caroline Pantofaru, Melonee Wise, Lorenz Mösenlechner, Wim Meeussen und Stefan Holzer: *Towards autonomous robotic butlers: Lessons learned with the PR2*. In: *IEEE International Conference on Robotics and Automation*, Seiten 5568–5575, Shanghai, China, 2011. IEEE. <https://doi.org/10.1109/ICRA.2011.5980058>.
- [24] Branicky, Michael S. und Siddharth R. Chhatpar: *A computational framework for the verification and synthesis of force-guided robotic assembly strategies*. In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Band 3, Seiten 1471–1476, Maui, HI, USA, 2001. IEEE. <https://doi.org/10.1109/IR0S.2001.977188>.
- [25] Brause, Rüdiger: *Prozesse*, Seiten 27–120. Springer Vieweg, Berlin, Heidelberg, 4. Auflage, 2017. [https://doi.org/10.1007/978-3-662-54100-5\\_2](https://doi.org/10.1007/978-3-662-54100-5_2).
- [26] Brooks, R.: *A robust layered control system for a mobile robot*. *IEEE Journal on Robotics and Automation*, 2(1):14–23, 1986. <https://doi.org/10.1109/JRA.1986.1087032>.

- [27] Brunner, Sebastian G., Franz Steinmetz, Rico Belder und Andreas Dömel: *RAF-CON: A graphical tool for engineering complex, robotic tasks*. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Seiten 3283–3290. IEEE, 2016. <https://doi.org/10.1109/IROS.2016.7759506>.
- [28] Brunner, Sebastian Georg, Andreas Domel, Peter Lehner, Michael Beetz und Freek Stulp: *Autonomous parallelization of resource-aware robotic task nodes*. IEEE Robotics and Automation Letters, 4(3):2599–2606, 2019. <https://doi.org/10.1109/LRA.2019.2894463>.
- [29] Cakmak, Maya und Leila Takayama: *Towards a comprehensive chore list for domestic robots*. In: *2013 8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Band 4, Seiten 93–94. IEEE, 2013. <https://doi.org/10.1109/HRI.2013.6483517>.
- [30] Calinon, Sylvain: *Learning from Demonstration (Programming by Demonstration)*. In: Ang, Marcelo H, Oussama Khatib und Bruno Siciliano (Herausgeber): *Encyclopedia of Robotics*. Springer, Berlin, Heidelberg, 2018. [https://doi.org/10.1007/978-3-642-41610-1\\_27-1](https://doi.org/10.1007/978-3-642-41610-1_27-1).
- [31] Calinon, Sylvain und Aude G. Billard: *What is the teacher's role in robot programming by demonstration?* Interaction Studies. Social Behaviour and Communication in Biological and Artificial Systems, 8(3):441–464, 2007. <https://doi.org/10.1075/is.8.3.08cal>.
- [32] Calinon, Sylvain, Danilo Bruno und Darwin G. Caldwell: *A task-parameterized probabilistic model with minimal intervention control*. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*, Seiten 3339–3344. IEEE, 2014. <https://doi.org/10.1109/ICRA.2014.6907339>.
- [33] Carlisle, Martin C., Terry A. Wilson, Jeffrey W. Humphries und Steven M. Hadfield: *RAPTOR: A Visual Programming Environment for Teaching Algorithmic Problem Solving*. In: *36th SIGCSE technical symposium on Computer science education*, Seiten 176–180. Association for Computing Machinery, 2005. <https://doi.org/10.1145/1047344.1047411>.
- [34] Carter, Jim und David Fourney: *Research Based Tactile and Haptic Interaction Guidelines*. In: *Guidelines On Tactile and Haptic Interactions Conference (GOTHI-05)*, Seiten 84–92, 2005.
- [35] Celebi, Sait, Ali S. Aydin, Talha T. Temiz und Tarik Arici: *Gesture recognition using skeleton data with weighted dynamic time warping*. Proceedings of the International Conference on Computer Vision Theory and Applications - Volume 1: VISAPP, (VISIGRAPP 2013), 1:620–625, 2013. <https://doi.org/10.5220/0004217606200625>.

- 
- [36] Chen, Feng Sheng, Chih Ming Fu und Chung Lin Huang: *Hand gesture recognition using a real-time tracking method and hidden Markov models*. Image and Vision Computing, 21(8):745–758, 2003. [https://doi.org/10.1016/S0262-8856\(03\)00070-2](https://doi.org/10.1016/S0262-8856(03)00070-2).
- [37] Chu, Wesley W. und Holger Opderbeck: *The page fault frequency replacement algorithm*. In: *Proceedings of the 1972 Fall Joint Computer Conference (AFIPS '72 Fall), Part I*, Seiten 597–609, New York, 1972. Association for Computing Machinery. <https://doi.org/10.1145/1479992.1480077>.
- [38] Cooper, Stephen, Wanda Dann und Randy Pausch: *ALICE : A 3-D Tool for Introductory Programming Concepts*. Journal of Computing Sciences in Colleges, 15(5):107–116, 2000.
- [39] Coronado, Enrique, Fulvio Mastrogiovanni, Bipin Indurkha und Gentiane Venture: *Visual Programming Environments for End-User Development of intelligent and social robots, a systematic review*. Journal of Computer Languages, 58, 2020. <https://doi.org/10.1016/j.j.cola.2020.100970>.
- [40] Corradini, A.: *Dynamic time warping for off-line recognition of a small gesture vocabulary*. In: *Proceedings IEEE ICCV Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems*, Seiten 82–89. IEEE, 2001. <https://doi.org/10.1109/RATFG.2001.938914>.
- [41] Côté, Carle, Dominic Létourneau, François Michaud, Jean Marc Valin, Yannick Brosseau, Clément Raïevsky, Mathieu Lemay und Victor Tran: *Code reusability tools for programming mobile robots*. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Band 2, Seiten 1820–1825. IEEE, 2004. <https://doi.org/10.1109/iros.2004.1389661>.
- [42] Crandall, Jacob W. und Michael A. Goodrich: *Experiments in adjustable autonomy*. In: *2001 IEEE International Conference on Systems, Man and Cybernetics*, Band 3, Seiten 1624–1629. IEEE, 2001. <https://doi.org/10.1109/icsmc.2001.973517>.
- [43] Darrell, Trevor und Alex Pentland: *Space-Time Gestures*. In: *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Seiten 335–340. IEEE, 1993. <https://doi.org/10.1109/CVPR.1993.341109>.
- [44] Dietz, Thomas, Ulrich Schneider, Marc Barho, Susanne Oberer-Treitz, Manuel Drust, Rebecca Hollmann und Martin Hägele: *Programming System for Efficient Use of Industrial Robots for Deburring in SME Environments*. In: *7th German Conference on Robotics*. IEEE, 2012.
- [45] Dijkstra, Edsger W.: *Cooperating Sequential Processes*. In: Hansen, Per Brinch (Herausgeber): *The Origin of Concurrent Programming*, Seiten 65–138. Springer, New York, 1968. [https://doi.org/10.1007/978-1-4757-3472-0\\_2](https://doi.org/10.1007/978-1-4757-3472-0_2).

- [46] Dümmel, Nikita: *MuLE A Multi-Paradigm Language for Education*. Dissertation, Universität Bayreuth, 2022. [https://doi.org/10.15495/EPub\\_UBT\\_00006408](https://doi.org/10.15495/EPub_UBT_00006408).
- [47] Eiband, Thomas, Christoph Willibald, Isabel Tannert, Bernhard Weber und Dongheui Lee: *Collaborative programming of robotic task decisions and recovery behaviors*. *Autonomous Robots*, 47, 2022. <https://doi.org/10.1007/s10514-022-10062-9>.
- [48] Eickeler, S., A. Kosmala und G. Rigoll: *Hidden Markov model based continuous online gesture recognition*. In: *Fourteenth International Conference on Pattern Recognition*, Band 2, Seiten 1206–1208. IEEE, 1998. <https://doi.org/10.1109/icpr.1998.711914>.
- [49] Eker, Steven, Thomas J Lee und Melinda T Gervasio: *Iteration Learning by Demonstration*. In: *AAAI Spring Symposium: Agents that Learn from Human Teachers*, Seiten 40–47, 2009.
- [50] Elbrechter, Christof, Robert Haschke und Helge Ritter: *Folding paper with anthropomorphic robot hands using real-time physics-based modeling*. In: *IEEE-RAS International Conference on Humanoid Robots*, Seiten 210–215. IEEE, 2012. <https://doi.org/10.1109/HUMANOIDS.2012.6651522>.
- [51] European Strategic Robotics Platform (EUROP): *Robotics Visions to 2020 and beyond – The Strategic Research Agenda for robotics in Europe, 07/2009*, 2009.
- [52] Franka Emika GmbH: *User Manual Panda Research*. München, 2017.
- [53] Fraser, Neil: *Ten Things We’ve Learned from Blockly*. In: *2015 IEEE Blocks and Beyond Workshop*, Seiten 49–50. IEEE, 2015. <https://doi.org/10.1109/BLOCKS.2015.7369000>.
- [54] Friedrich, Holger und Rudiger Dillmann: *Robot programming based on a single demonstration and user intentions*. In: *3rd European workshop on learning robots at ECML*, 1995.
- [55] Galil, Zvi und Giuseppe F. Italiano: *Data structures and algorithms for disjoint set union problems*. *ACM Computing Surveys (CSUR)*, 23(3):319–344, 1991. <https://doi.org/10.1145/116873.116878>.
- [56] Galler, Bernard A. und Michael J. Fisher: *An improved equivalence algorithm*. *Communications of the ACM*, 7(5):301–303, 1964. <https://doi.org/10.1145/364099.364331>.
- [57] Gan, Yahui, Xianzhong Dai und Dongwei Li: *Off-Line Programming Techniques for Multirobot Cooperation System*. *International Journal of Advanced Robotic Systems*, 10(7), 2013. <https://doi.org/10.5772/56506>.



- 
- [58] Godart, Peter, Johannes Gross, Rudranarayan Mukherjee und Wyatt Ubellacker: *Generating real-time robotics control software from SysML*. In: *2017 IEEE Aerospace Conference*. IEEE, 2017. <https://doi.org/10.1109/AERO.2017.7943610>.
- [59] Grollman, Daniel H. und Odest Chadwicke Jenkins: *Can We Learn Finite State Machine Robot Controllers from Interactive Demonstration?* In: Sigaud, Olivier und Jan Peters (Herausgeber): *From Motor Learning to Interaction Learning in Robots. Studies in Computational Intelligence*, Band 264, Seiten 407–430, Berlin, Heidelberg, 2010. Springer. [https://doi.org/10.1007/978-3-642-05181-4\\_17](https://doi.org/10.1007/978-3-642-05181-4_17).
- [60] Groth, Christian: *Online-Adaption und synchronisierte Ausführung einmalig demonstrierter Roboterverhalten*. Dissertation, Universität Bayreuth, 2022.
- [61] Groth, Christian und Dominik Henrich: *One-shot robot programming by demonstration by adapting motion segments*. In: *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, Seiten 1068–1075. IEEE, 2014. <https://doi.org/10.1109/ROBIO.2014.7090474>.
- [62] Groth, Christian und Dominik Henrich: *One-shot robot programming by demonstration using an online oriented particles simulation*. In: *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, Seiten 154–160. IEEE, 2014. <https://doi.org/10.1109/ROBIO.2014.7090323>.
- [63] Grüninger, Rolf, Ezbieta Kus und Richard Hüppi: *Market Study on Adaptive Robots for Flexible Manufacturing Systems*. In: *2009 IEEE International Conference on Mechatronics*. IEEE, 2009. <https://doi.org/10.1109/ICMECH.2009.4957143>.
- [64] Hägele, Martin, Klas Nilsson, J. Norberto Pires und Rainer Bischoff: *Industrial Robotics*. In: Siciliano, Bruno und Oussama Khatib (Herausgeber): *Springer Handbook of Robotics*, Seiten 1385–1422. Springer International Publishing, Cham, 2016. [https://doi.org/10.1007/978-3-319-32552-1\\_54](https://doi.org/10.1007/978-3-319-32552-1_54).
- [65] Harel, David: *Statecharts: a visual formalism for complex systems*. *Science of Computer Programming*, 8(3):231–274, 1987. [https://doi.org/10.1016/0167-6423\(87\)90035-9](https://doi.org/10.1016/0167-6423(87)90035-9).
- [66] Harrer, David: *Roboterbasiertes und automatisiertes Einlernen von vollständigen Werkstückmodellen*. Masterarbeit, Universität Bayreuth, 2019.
- [67] Hoaglin, David C.: *John W. Tukey and Data Analysis*. *Statistical Sciences*, 18(3):311–318, 2003.
- [68] Hong, Pengyu, M. Turk und T. S. Huang: *Gesture modeling and recognition using finite state machines*. In: *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition*, Seiten 410–415, Grenoble, 2000. IEEE. <https://doi.org/10.1109/AFGR.2000.840667>.

- [69] Hopcroft, J. E. und J. D. Ullman: *Set merging algorithms*. SIAM Journal on Computing, 2(4):294–303, 1973. <https://doi.org/10.1137/0202024>.
- [70] Hopcroft, John und Jeffrey D. Ullman: *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley Publishing Company, 1979.
- [71] Hu, Yuxiao und H Levesque: *Planning with loops: Some new results*. In: *ICAPS Workshop on Generalized Planning*, Seiten 35–42, Thessaloniki, 2009.
- [72] Huang, Justin, Tessa Lau und Maya Cakmak: *Design and evaluation of a rapid programming system for service robots*. In: *ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Seiten 295–302. IEEE, 2016. <https://doi.org/10.1109/HRI.2016.7451765>.
- [73] Hurdus, J.G.: *A Portable Approach to High-Level Behavioral Programming for Complex Autonomous Robot Applications*. Masterarbeit, Virginia Polytechnic Institute and State University, 2008.
- [74] IFR: *Geschätzter Bestand von Industrierobotern weltweit in den Jahren 2011 bis 2021*. <https://de.statista.com/statistik/daten/studie/250212/umfrage/geschaezter-bestand-von-industrierobotern-weltweit/>, besucht: 31.03.2023.
- [75] International Federation of Robotics: *Robots and the Workplace of the Future (Positioning Paper)*, 2018.
- [76] Jokinen, Kristiina und Graham Wilcock: *Multimodal Open-Domain Conversations with the Nao Robot*. In: Mariani, J., S. Rosset, M. Garnier-Rizet und L. Devillers (Herausgeber): *Natural Interaction with Robots, Knowbots and Smartphones*, Seiten 213–224, New York, 2014. Springer. [https://doi.org/10.1007/978-1-4614-8280-2\\_19](https://doi.org/10.1007/978-1-4614-8280-2_19).
- [77] Kain, Kevin Sebastian, Susanne Stadler, Manuel Giuliani, Nicole Mirnig, Gerald Stollnberger und Manfred Tscheligi: *Tablet-Based Augmented Reality in the Factory*. In: *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, Seiten 151–152. Association for Computing Machinery, 2017. <https://doi.org/10.1145/3029798.3038347>.
- [78] Kammel, Sören, Julius Ziegler, Benjamin Pitzer, Moritz Werling, Tobias Gindele, Daniel Jagzent, Joachim Schröder, Michael Thuy, Matthias Goebel, Felix von Hundelshausen, Oliver Pink, Christian Frese und Christoph Stiller: *Team ANNIEWAY's Autonomous System for the 2007 DARPA Urban Challenge*. Journal of Field Robotics, 25:615–639, 2008. <https://doi.org/10.1002/rob.20252>.
- [79] Kela, Juha, Panu Korpipää, Jani Mäntyjärvi, Sanna Kallio, Giuseppe Savino, Luca Jozzo und Sergio Di Marca: *Accelerometer-based gesture control for a design environment*. Personal and Ubiquitous Computing, 10:285–299, 2006. <https://doi.org/10.1007/s00779-005-0033-8>.

- 
- [80] Kildal, Johan, Alberto Tellaecche, Izaskun Fernández und Iñaki Maurtua: *Potential users' key concerns and expectations for the adoption of cobots*. *Procedia CIRP*, 72:21–26, 2018. <https://doi.org/10.1016/J.PROCIR.2018.03.104>.
- [81] König, Lukas, Sanaz Mostaghim und Hartmut Schmeck: *Decentralized Evolution of Robotic Behavior Using Finite State Machines*. *International Journal of Intelligent Computing and Cybernetics*, 2(4):695–723, 2009. <https://doi.org/10.1108/17563780911005845>.
- [82] Kormushev, Petar, Sylvain Calinon und Darwin G. Caldwell: *Imitation Learning of Positional and Force Skills Demonstrated via Kinesthetic Teaching and Haptic Input*. *Advanced Robotics*, 25(5):581–603, 2011. <https://doi.org/10.1163/016918611X558261>.
- [83] Kortenkamp, David, Reid Simmons und Davide Brugali: *Robotic Systems Architectures and Programming*. In: Siciliano, Bruno und Oussama Khatib (Herausgeber): *Springer Handbook of Robotics*, Seiten 283–306. Springer International Publishing, Cham, 2016. [https://doi.org/10.1007/978-3-319-32552-1\\_12](https://doi.org/10.1007/978-3-319-32552-1_12).
- [84] Kubus, Daniel, Arne Muxfeldt, Konrad Kissener, Jan Haus und Jochen Steil: *Robust recognition of tactile gestures for intuitive robot programming and control*. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Seiten 1643–1650. IEEE, 2017. <https://doi.org/10.1109/IROS.2017.8205974>.
- [85] Kurakin, A., Z. Zhang und Z. Liu: *A real time system for dynamic hand gesture recognition with a depth sensor*. In: *2012 Proceedings of the 20th European Signal Processing Conference (EUSIPCO)*, Seiten 1975–1979. IEEE, 2012.
- [86] Lau, Tessa, Pedro Domingos und Daniel S. Weld: *Learning programs from traces using version space algebra*. In: *Proceedings of the 2nd International Conference on Knowledge Capture (K-CAP)*, Seiten 36–43, New York, 2003. Association for Computing Machinery. <https://doi.org/10.1145/945645.945654>.
- [87] Lau, Tessa A. und Daniel S. Weld: *Programming by Demonstration Using Version Space Algebra*. *Machine Learning*, 53:111–156, 2003. <https://doi.org/10.1023/A:1025671410623>.
- [88] Lee, Jangwon: *A survey of robot learning from demonstrations for Human-Robot Collaboration*, 2017. <https://doi.org/10.48550/arXiv.1710.08789>.
- [89] Lei, Wentai, Xinyue Jiang, Long Xu, Jiabin Luo, Mengdi Xu und Feifei Hou: *Continuous gesture recognition based on time sequence fusion using mimo radar sensor and deep learning*. *Electronics*, 9(5), 2020. <https://doi.org/10.3390/electronics9050869>.
- [90] Levesque, Hector J.: *Planning with loops*. *IJCAI International Joint Conference on Artificial Intelligence*, Seiten 509–515, 2005.

- [91] Link, Albert N., Zachary T. Oliver und Alan C. O'Connor: *Economic Analysis of Technology Infrastructure Needed for Advanced Manufacturing - Advanced Robotics and Manufacturing*, 2016. <https://doi.org/10.6028/NIST.GCR.16-005>.
- [92] Loetzsch, Martin, Max Risler und Matthias Jünger: *XABSL - A pragmatic approach to behavior engineering*. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Seiten 5124–5129. IEEE, 2006. <https://doi.org/10.1109/IROS.2006.282605>.
- [93] Lozano-Pérez, Tomás: *Robot Programming*. *Proceedings of the IEEE*, 71(7):821–841, 1983. <https://doi.org/10.1109/PROC.1983.12681>.
- [94] Maloney, John, Natalie Rusk, Leo Burd, Brian Silverman, Yasmin Kafai und Mitchel Resnick: *Scratch: A sneak preview*. In: *Second International Conference on Creating, Connecting and Collaborating Through Computing*, Seiten 104–109, 2004. <https://doi.org/10.1109/C5.2004.1314376>.
- [95] Marcel, Sebastien, Olivier Bernier, Jean Emmanuel Viallet und Daniel Collobert: *Hand Gesture Recognition using Input-Output Hidden Markov Models*. In: *Proceedings Fourth IEEE International Conference on Automatic Face and Gesture Recognition*, Seiten 456–461, Grenoble, 2000. IEEE. <https://doi.org/10.1109/AFGR.2000.840674>.
- [96] Marino, Alessandro, Lynne Parker, Gianluca Antonelli und Fabrizio Caccavale: *Behavioral control for multi-robot perimeter patrol: A Finite State Automata approach*. In: *2009 IEEE International Conference on Robotics and Automation*, Seiten 831–836. IEEE, 2009. <https://doi.org/10.1109/ROBOT.2009.5152710>.
- [97] Miranda, Leandro, Thales Vieira, Dimas Martinez, Thomas Lewiner, Antonio W. Vieira und Mario F.M. Campos: *Real-time gesture recognition from depth data through key poses learning and decision forests*. In: *2012 25th SIBGRAPI Conference on Graphics, Patterns and Images*, Seiten 268–275. IEEE, 2012. <https://doi.org/10.1109/SIBGRAPI.2012.44>.
- [98] Mitra, Sushmita und Tinku Acharya: *Gesture recognition: A survey*. *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, 37(3):311–324, 2007. <https://doi.org/10.1109/TSMCC.2007.893280>.
- [99] Moni, M. A. und A. B.M. Shawkat Ali: *HMM based hand gesture recognition: A review on techniques and approaches*. In: *2009 2nd IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, Seiten 433–437. IEEE, 2009. <https://doi.org/10.1109/ICCSIT.2009.5234536>.
- [100] Montebelli, Alberto, Franz Steinmetz und Ville Kyrki: *On handing down our tools to robots: Single-phase kinesthetic teaching for dynamic in-contact tasks*. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*, Seiten 5628–5634. IEEE, 2015. <https://doi.org/10.1109/ICRA.2015.7139987>.

- 
- [101] Müller, Florian: *Assistierende virtuelle Kraftfelder bei handgeführten Robotern*. Dissertation, Technische Universität Chemnitz, 2018.
- [102] Murray, Jan und Frieder Stolzenburg: *Hybrid State Machines with Timed Synchronization for Multi-Robot System Specification*. In: *2005 portuguese conference on artificial intelligence*, Seiten 236–241, 2005. <https://doi.org/10.1109/EPIA.2005.341221>.
- [103] Muxfeldt, Arne, Sugeeth Gopinathan, Thilo Coenders und Jochen Steil: *A user study on human-robot-interactive recovery for industrial assembly problems*. In: *26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, Seiten 824–830. IEEE, 2017. <https://doi.org/10.1109/ROMAN.2017.8172398>.
- [104] Muxfeldt, Arne, Jan Niklas Haus, Jingyuan Cheng und Daniel Kubus: *Exploring tactile surface sensors as a gesture input device for intuitive robot programming*. In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, Seiten 7–10. IEEE, 2016. <https://doi.org/10.1109/ETFA.2016.7733658>.
- [105] Muxfeldt, Arne, Jan Henrik Kluth und Daniel Kubus: *Kinesthetic teaching in assembly operations – a user study*. In: *Simulation, Modeling, and Programming for Autonomous Robots (SIMPAN) 2014. Lecture Notes in Computer Science*, Band 8810, Seiten 533–544, Cham, 2014. Springer. [https://doi.org/10.1007/978-3-319-11900-7\\_45](https://doi.org/10.1007/978-3-319-11900-7_45).
- [106] Muxfeldt, Arne, Daniel Kubus und Friedrich M. Wahl: *Developing new application fields for industrial robots - Four examples for academia-industry collaboration*. In: *2015 IEEE 20th Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2015. <https://doi.org/10.1109/ETFA.2015.7301652>.
- [107] Ng, Chan Wah und Surendra Ranganath: *Real-time gesture recognition system and application*. *Image and Vision Computing*, 20(13):993–1007, 2002. [https://doi.org/10.1016/S0262-8856\(02\)00113-0](https://doi.org/10.1016/S0262-8856(02)00113-0).
- [108] Nguyen, Hai, Matei Ciocarlie, Kaijen Hsiao und Charles C. Kemp: *ROS commander (ROSCo): Behavior creation for home robots*. In: *2013 IEEE International Conference on Robotics and Automation*, Seiten 467–474, 2013. <https://doi.org/10.1109/ICRA.2013.6630616>.
- [109] Niemüller, Tim, Alexander Ferrein und Gerhard Lakemeyer: *A Lua-based behavior engine for controlling the humanoid robot Nao*. *RoboCup 2009: Robot Soccer World Cup XIII. RoboCup 2009. Lecture Notes in Computer Science*, 5949:240–251, 2010. [https://doi.org/10.1007/978-3-642-11876-0\\_21](https://doi.org/10.1007/978-3-642-11876-0_21).
- [110] Olsen, D. R. und Michael A. Goodrich: *“Metrics for Evaluating Human-Robot Interaction*. In: *PERMIS: Performance Metrics for Intelligent Systems*, 2003.

- [111] Orendt, E. M., M. Riedl, und D. Henrich: *Kapitel 5.3: Programmierung von Robotern*. In: Müller, R., J. Franke, D. Henrich, B. Kuhlenkötter, A. Raatz und A. Verl (Herausgeber): *Handbuch Mensch-Roboter-Kollaboration*, Seiten 182–203. Carl Hanser Verlag, 2019.
- [112] Orendt, Eric M., Myriell Fichtner und Dominik Henrich: *Robot Programming by Non-Experts: Intuitiveness and Robustness of One-shot Robot Programming*. In: *2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, Seiten 192–199. IEEE, 2016. <https://doi.org/10.1109/ROMAN.2016.7745110>.
- [113] Orendt, Eric M., Myriell Fichtner und Dominik Henrich: *MINERIC toolkit: Measuring instruments to evaluate robustness and intuitiveness of robot programming concepts*. In: *2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, Seiten 1379–1386, 2017. <https://doi.org/10.1109/ROMAN.2017.8172484>.
- [114] Orendt, Eric M. und Dominik Henrich: *Control flow for robust one-shot robot programming using entity-based resources*. In: *2017 18th International Conference on Advanced Robotics (ICAR)*, Seiten 68–74, 2017. <https://doi.org/10.1109/ICAR.2017.8023498>.
- [115] Orendt, Eric M., Michael Riedl und Dominik Henrich: *Robust One-Shot Robot Programming by Demonstration Using Entity-Based Resources*. In: Ferraresi, Carlo und Giuseppe Quaglia (Herausgeber): *Advances in Service and Industrial Robotics (RAAD) 2017. Mechanisms and Machine Science*, Band 49, Seiten 573–582, Cham, 2018. Springer International Publishing. [https://doi.org/10.1007/978-3-319-61276-8\\_60](https://doi.org/10.1007/978-3-319-61276-8_60).
- [116] Ott, C., A. Albu-Schaffer, A. Kugi, S. Stramigioli und G. Hirzinger: *A Passivity Based Cartesian Impedance Controller for Flexible Joint Robots Part I: Torque Feedback and Gravity Compensation*. In: *IEEE International Conference on Robotics and Automation*, Band 3, Seiten 2666–2672, 2004. <https://doi.org/10.1109/ROBOT.2004.1307462>.
- [117] Pan, Zengxi, Joseph Polden, Nathan Larkin, Stephen Van Duin und John Norrish: *Recent progress on programming methods for industrial robots*. *Robotics and Computer-Integrated Manufacturing*, 28(2):87–94, 2012. <https://doi.org/10.1016/j.rcim.2011.08.004>.
- [118] Pardowitz, Michael, Bernhard Glaser und Rüdiger Dillmann: *Learning repetitive robot programs from demonstrations using version space algebra*. In: Schilling, K. (Herausgeber): *Proceedings of the 13th IASTED International Conference on Robotics and Applications, RA 2007 and Proceedings of the IASTED International Conference on Telematics*, Seiten 394–399. ACTA Press, 2007.

- [119] Park, Dae Hyung, Heiko Hoffmann, Peter Pastor und Stefan Schaal: *Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields*. In: *Humanoids 2008 - 8th IEEE-RAS International Conference on Humanoid Robots*, Seiten 91–98, 2008. <https://doi.org/10.1109/ICHR.2008.4755937>.
- [120] Paternò, Fabio und Carmen Santoro: *End-user development for personalizing applications, things, and robots*. *International Journal of Human Computer Studies*, 131:120–130, 2019. <https://doi.org/10.1016/j.ijhcs.2019.06.002>.
- [121] Paxton, Chris, Andrew Hundt, Felix Jonathan, Kelleher Guerin und Gregory D. Hager: *CoSTAR: Instructing collaborative robots with behavior trees and vision*. In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*, Seiten 564–571, Singapore, 2017. <https://doi.org/10.1109/ICRA.2017.7989070>.
- [122] Pek, Christian, Arne Muxfeldt und Daniel Kubus: *Simplifying synchronization in cooperative robot tasks - An enhancement of the Manipulation Primitive paradigm*. In: *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2016. <https://doi.org/10.1109/ETFA.2016.7733552>.
- [123] Perzylo, Alexander, Markus Rickert, Björn Kahl, Nikhil Somani, Christian Lehmann, Alexander Kuss, Stefan Profanter, Anders Billesø Beck, Mathias Haage, Mikkel Rath Hansen, Malene Tofveson Nibe, Máximo A. Roa, Olof Sörnmo, Sven Gestegård Robertz, Ulrike Thomas, Germano Veiga, Elin Anna Topp, Ingmar Kessler und Marinus Danzer: *SMErobotics: Smart Robots for Flexible Manufacturing*. *IEEE Robotics and Automation Magazine*, 26(1):78–90, 2019. <https://doi.org/10.1109/MRA.2018.2879747>.
- [124] Peters, Jan, Daniel D. Lee, Jens Kober, Duy Nguyen-Tuong, J. Andrew Bagnell und Stefan Schaal: *Robot Learning*. In: Siciliano, Bruno und Oussama Khatib (Herausgeber): *Springer Handbook of Robotics*, Seiten 357–398. Springer International Publishing, Cham, 2016. [https://doi.org/10.1007/978-3-319-32552-1\\_15](https://doi.org/10.1007/978-3-319-32552-1_15).
- [125] Purkayastha, Sagar N., Nick Eckenstein, Michael D. Byrne und Marcia K. O’Malley: *Analysis and comparison of low cost gaming controllers for motion analysis*. In: *2010 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, Seiten 353–360. IEEE, 2010. <https://doi.org/10.1109/AIM.2010.5695830>.
- [126] Quottrup, M.M., T. Bak und R.I. Zamanabadi: *Multi-robot planning: a timed automata approach*. In: *IEEE International Conference on Robotics and Automation (ICRA)*, Band 5, Seiten 4417–4422, 2004. <https://doi.org/10.1109/ROBOT.2004.1302413>.
- [127] Ramamoorthy, Aditya, Namrata Vaswani, Santanu Chaudhury und Subhashis Banerjee: *Recognition of dynamic hand gestures*. *Pattern Recognition*, 36(9):2069–2081, 2003. [https://doi.org/10.1016/S0031-3203\(03\)00042-6](https://doi.org/10.1016/S0031-3203(03)00042-6).

- [128] Ravichandar, Harish, Athanasios S. Polydoros, Sonia Chernova und Aude Billard: *Recent Advances in Robot Learning from Demonstration*. Annual Review of Control, Robotics, and Autonomous Systems, 3(1):297–330, 2020. <https://doi.org/10.1146/annurev-control-100819-063206>.
- [129] Riano, Lorenzo und T. M. Mcginnity: *Compositional Skills Building*, 2011. Preprint.
- [130] Riedelbauch, Dominik und Jonathan Hümmer: *A Benchmark Toolkit for Collaborative Human-Robot Interaction*. In: *2022 31st IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*, Seiten 806–813. IEEE, 2022. <https://doi.org/10.1109/ro-man53752.2022.9900790>.
- [131] Riedl, Michael: *Intuitive sensorbasierte, editierbare, kinästhetische Programmierung von Pick-and-Place-Aufgaben für Mehrrobotersysteme*. Dissertation, Universität Bayreuth, 2022. [https://doi.org/10.15495/EPub\\_UBT\\_00005968](https://doi.org/10.15495/EPub_UBT_00005968).
- [132] Riedl, Michael, Johannes Baumgartl und Dominik Henrich: *Editing and synchronizing multi-robot playback programs*. In: *Proceedings of ISR 2016: 47th International Symposium on Robotics*, Seiten 1–8. VDE, 2016.
- [133] Riedl, Michael, Eric M. Orendt und Dominik Henrich: *Sensor-based loops and branches for playback-programmed robot systems*. In: Ferraresi, Carlo und Giuseppe Quaglia (Herausgeber): *Advances in Service and Industrial Robotics. RAAD 2017. Mechanisms and Machine Science*, Band 49, Seiten 183–190, Cham, 2018. Springer International Publishing. [https://doi.org/10.1007/978-3-319-61276-8\\_21](https://doi.org/10.1007/978-3-319-61276-8_21).
- [134] Rigoll, Gerhard, Andreas Kosmala und Stefan Eickeler: *High performance real-time gesture recognition using hidden markov models*. Gesture and Sign Language in Human-Computer Interaction. GW 1997. Lecture Notes in Computer Science, 1371:69–80, 1998. <https://doi.org/10.1007/BFb0052990>.
- [135] Ritschel, Nico, Vladimir Kovalenko, Reid Holmes, Ronald Garcia und David C Shepherd: *Comparing Block-Based Programming Models for Two-Armed Robots*. IEEE Transactions on Software Engineering, 48(5):1630–1643, 2022. <https://doi.org/10.1109/TSE.2020.3027255>.
- [136] Ritter, Helge, Robert Haschke und Jochen J. Steil: *A dual interaction perspective for robot cognition: Grasping as a "Rosetta Stone"*. Perspectives of Neural-Symbolic Integration. Studies in Computational Intelligence, 77:159–178, 2007. [https://doi.org/10.1007/978-3-540-73954-8\\_7](https://doi.org/10.1007/978-3-540-73954-8_7).
- [137] Rossano, G. F., C. Martinez, M. Hedelind, S. Murphy und T. A. Fuhlbrigge: *Easy robot programming concepts: An industrial perspective*. In: *2013 IEEE International Conference on Automation Science and Engineering (CASE)*, Seiten 1119–1126. IEEE, 2013. <https://doi.org/10.1109/CoASE.2013.6654035>.



- [138] Rossano, Gregory F., Carlos Martinez, Mikael Hedelind, Steve Murphy und Thomas A. Fuhlbrigge: *Easy robot path programming concepts: An industrial perspective on path creation*. In: *44th International Symposium on Robotics (ISR)*, Seiten 1119–1126. IEEE, 2013. <https://doi.org/10.1109/ISR.2013.6695710>.
- [139] Sang, Yu, Laixi Shi und Yimin Liu: *Micro hand gesture recognition system using ultrasonic active sensing*. *IEEE Access*, 6:49339–49347, 2018. <https://doi.org/10.1109/ACCESS.2018.2868268>.
- [140] Schaal, Stefan: *Dynamic Movement Primitives - A Framework for Motor Control in Humans and Humanoid Robotics*. In: Kimura, Hiroshi, Kazuo Tsuchiya, Akio Ishiguro und Hartmut Witte (Herausgeber): *Adaptive Motion of Animals and Machines*, Seiten 261–280. Springer, Tokyo, 2006. [https://doi.org/10.1007/4-431-31381-8\\_23](https://doi.org/10.1007/4-431-31381-8_23).
- [141] Schlenzig, Jennifer, Edd Hunter und Ramesh Jain: *Recursive identification of gesture inputs using hidden markov models*. In: *Proceedings of 1994 IEEE Workshop on Applications of Computer Vision*, Seiten 187–194, 1994. <https://doi.org/10.1109/acv.1994.341308>.
- [142] Schöning, Uwe: *Theoretische Informatik - kurz gefasst*. Spektrum Akademischer Verlag, Heidelberg, 5. Auflage, 2008.
- [143] Schou, C., J. S. Damgaard, S. Bogh und O. Madsen: *Human-robot interface for instructing industrial tasks using kinesthetic teaching*. In: *2013 44th International Symposium on Robotics (ISR)*. IEEE, 2013. <https://doi.org/10.1109/ISR.2013.6695599>.
- [144] Sefidgar, Yasaman S., Prerna Agarwal und Maya Cakmak: *Situated Tangible Robot Programming*. In: *ACM/IEEE International Conference on Human-Robot Interaction*, Seiten 473–482, New York, 2017. Association for Computing Machinery. <https://doi.org/10.1145/2909824.3020240>.
- [145] Siciliano, Bruno und Oussama Khatib (Herausgeber): *Springer Handbook of Robotics*. Springer International Publishing, Cham, 2016. <https://doi.org/10.1007/978-3-319-32552-1>.
- [146] Smid, Michiel H.M.: *A data structure for the union-find problem having good single-operation complexity*. Technischer Bericht CT-88-06, Prepublication Series Computation and Complexity Theory, University of Amsterdam, Institute for Language, Logic and Information, 1988.
- [147] Statistisches Bundesamt: *Anteil der Unternehmen im Verarbeitenden Gewerbe mit Industrierobotern in Deutschland nach Anzahl der Beschäftigten in den Jahren 2018 und 2020*. <https://de.statista.com/statistik/daten/studie/947397/umfrage/nutzung-von-industrierobotern-im-verarbeitenden-gewerbe-in-deutschland/>, besucht: 28.10.2022.

- [148] Statistisches Bundesamt (Destatis): *Shares of small and medium-sized enterprises in selected variables, 2020*. <https://www.destatis.de/EN/Themes/Economic-Sectors-Enterprises/Enterprises/Small-Sized-Enterprises-Medium-Sized-Enterprises/Tables/total-cik.html>, besucht: 31.03.2023.
- [149] Steil, Jochen J. und Günter W. Maier: *Kollaborative Roboter: universale Werkzeuge in der digitalisierten und vernetzten Arbeitswelt*. In: Maier, Günter W, Gregor Engels und Eckhard Steffen (Herausgeber): *Handbuch Gestaltung digitaler und vernetzter Arbeitswelten*, Seiten 1–24. Springer, Berlin, Heidelberg, 2018. [https://doi.org/10.1007/978-3-662-52903-4\\_15-1](https://doi.org/10.1007/978-3-662-52903-4_15-1).
- [150] Steinmetz, Franz, Annika Wollschläger und Roman Weitschat: *RAZER - A HRI for Visual Task-Level Programming and Intuitive Skill Parameterization*. IEEE Robotics and Automation Letters, 3(3):1362–1369, 2018. <https://doi.org/10.1109/LRA.2018.2798300>.
- [151] Stenmark, Maj, Elin Topp, Mathias Haage und Jacek Malec: *Knowledge for Synchronized Dual-Arm Robot Programming*. In: *Artificial Intelligence for Human-Robot Interaction. AAAI Technical Reports FS-17-01*, 2017.
- [152] Stolzenburg, Frieder und Toshiaki Arai: *From the Specification of Multiagent Systems by Statecharts to Their Formal Analysis by Model Checking: Towards Safety-Critical Applications*. In: Schillo, Michael, Matthias Klusch, Jörg Müller und Huaglorry Tianfield (Herausgeber): *Multiagent System Technologies (MATES) 2003. Lecture Notes in Computer Science*, Band 2831, Seiten 131–143, Berlin, Heidelberg, 2003. Springer. [https://doi.org/10.1007/978-3-540-39869-1\\_12](https://doi.org/10.1007/978-3-540-39869-1_12).
- [153] Stumm, Sven, Pradeep Devadass und Sigrid Brell-Cokcan: *Haptic programming in construction*. Construction Robotics, 2:3–13, 2018. <https://doi.org/10.1007/s41693-018-0015-9>.
- [154] Taft, S. Tucker, Robert A. Duff, Randall L. Brukaradt, Erhard Ploedereeder, Pascal Leroy und Edmond Schonberg (Herausgeber): *Ada 2012 Reference Manual. Language and Standard Libraries*. Springer Berlin, Heidelberg, 2013. <https://doi.org/10.1007/978-3-642-45419-6>.
- [155] Tanenbaum, Andrew S.: *Processes and Threads*, Seiten 85–180. Pearson Education Inc., New Jersey, 4. Auflage, 2015.
- [156] Thomas, Ulrike, Gerd Hirzinger, Bernhard Rumpe, Christoph Schulze und Andreas Wortmann: *A new skill based robot programming language using UML/P Statecharts*. In: *2013 IEEE International Conference on Robotics and Automation*, Seiten 461–466, 2013. <https://doi.org/10.1109/ICRA.2013.6630615>.
- [157] Tousignant, Steve, Eric Van Wyk und Maria Gini: *XRobots: A flexible language for programming mobile robots based on hierarchical state machines*. In: *2012 IEEE*

- International Conference on Robotics and Automation*, Seiten 1773–1778. IEEE, 2012. <https://doi.org/10.1109/ICRA.2012.6225145>.
- [158] Tractica: *Umsatz mit Industrierobotern weltweit in den Jahren von 2018 bis 2025*. <https://de.statista.com/statistik/daten/studie/870571/umfrage/umsatz-von-industrierobotern-weltweit/>, besucht: 31.03.2023.
- [159] Tsarouchi, Panagiota, Sotiris Makris und George Chryssolouris: *Human-robot interaction review and challenges on task planning and programming*. *International Journal of Computer Integrated Manufacturing*, 29(8):916–931, 2016. <https://doi.org/10.1080/0951192X.2015.1130251>.
- [160] Tsinganos, Panagiotis, Bruno Cornelis, Jan Cornelis, Bart Jansen und Athanassios Skodras: *Improved Gesture Recognition Based on sEMG Signals and TCN*. In: *2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Seiten 1169–1173. IEEE, 2019. <https://doi.org/10.1109/ICASSP.2019.8683239>.
- [161] Universal Robots A/S: *PolyScope Manual Version 5.1.0*, 2018.
- [162] Villani, Valeria, Fabio Pini, Francesco Leali und Cristian Secchi: *Survey on human-robot collaboration in industrial settings: Safety, intuitive interfaces and applications*. *Mechatronics*, 55:248–266, 2018. <https://doi.org/10.1016/j.mechatronics.2018.02.009>.
- [163] Wang, Qinyi, Yexin Zhang, Junsong Yuan und Yilong Lu: *Space-time event clouds for gesture recognition: From RGB cameras to event cameras*. 2019 IEEE Winter Conference on Applications of Computer Vision (WACV), Seiten 1826–1835, 2019. <https://doi.org/10.1109/WACV.2019.00199>.
- [164] Watson, B.: *A taxonomy of finite automata minimization algorithms*. Technischer Bericht Computing Science Report 93/44, Eindhoven University of Technology, 1995.
- [165] Wilson, Andrew D. und Aaron F. Bobick: *Learning visual behavior for gesture analysis*. *Proceedings of the IEEE International Conference on Computer Vision (ISCV)*, Seiten 229–234, 1995. <https://doi.org/10.1109/iscv.1995.477006>.
- [166] Wilson, Daniel und Andy Wilson: *Gesture Recognition Using The XWand*. Technischer Bericht CMU-RI-TR-04-31, Carnegie Mellon University, 2004.
- [167] Wölfel, Kim: *Force-Response-Konzepte zur manuellen, kraftbasierten Robotersteuerung*. Masterarbeit, Universität Bayreuth, 2016.
- [168] Wölfel, Kim: *SpIRo Sprachbasierte Instruktion kraftbasierter Roboterbewegungen*. Dissertation, Universität Bayreuth, 2021. [https://doi.org/10.15495/EPub\\_UBT\\_00005667](https://doi.org/10.15495/EPub_UBT_00005667).

- [169] Wösch, Thomas und Wendelin Feiten: *Reactive motion control for human-robot tactile interaction*. In: *Proceedings 2002 IEEE International Conference on Robotics and Automation*, Band 4, Seiten 3807–3812. IEEE, 2002. <https://doi.org/10.1109/robot.2002.1014313>.
- [170] Wrede, Sebastian, Christian Emmerich, Ricarda Grünberg, Arne 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*, 2(1):56–81, 2013. <https://doi.org/10.5898/jhri.2.1.wrede>.
- [171] Wu, Yan und Yiannis Demiris: *Towards one shot learning by imitation for humanoid robots*. In: *2010 IEEE International Conference on Robotics and Automation*, Seiten 2889–2894. IEEE, 2010. <https://doi.org/10.1109/ROBOT.2010.5509429>.
- [172] Xinhui, Wang, Yu Chang, Fu Jiawei, Zhou Hangyu und Han Tianlin: *Team TH-MOS*, 2018.
- [173] Yamane, Katsu und Wataru Takano: *Human Motion Reconstruction*. In: Siciliano, Bruno und Oussama Khatib (Herausgeber): *Springer Handbook of Robotics*, Seiten 1819–1834. Springer International Publishing, Cham, 2016. [https://doi.org/10.1007/978-3-319-32552-1\\_68](https://doi.org/10.1007/978-3-319-32552-1_68).
- [174] Yannakakis, Mihalis: *Hierarchical State Machines*. *Theoretical Computer Science: Exploring New Frontiers of Theoretical Informatics (TCS)*. *Lecture Notes in Computer Science*, 1872:315–330, 2000. [https://doi.org/10.1007/3-540-44929-9\\_24](https://doi.org/10.1007/3-540-44929-9_24).
- [175] Zhu, Zuyuan und Huosheng Hu: *Robot Learning from Demonstration in Robotic Assembly: A Survey*. *Robotics*, 7(2), 2018. <https://doi.org/10.3390/robotics7020017>.
- [176] Zieliński, Cezary: *By how much should a general purpose programming language be extended to become a multi-robot system programming language?* *Advanced Robotics*, 15(1):71–95, 2001. <https://doi.org/10.1163/156855301750095587>.
- [177] Ziparo, V. A., L. Iocchi, D. Nardi, P. F. Palamara und H. Costelha: *Petri Net Plans: A Formal Model for Representation and Execution of Multi-Robot Plans*. In: *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1, AAMAS '08*, Seiten 79–86, Richland, SC, 2008. International Foundation for Autonomous Agents and Multiagent Systems.
- [178] Zollner, R., T. Asfour und R. Dillmann: *Programming by demonstration: dual-arm manipulation tasks for humanoid robots*. In: *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Band 1, Seiten 479–484, 2004. <https://doi.org/10.1109/IROS.2004.1389398>.

---

## Eigene Publikationen

---

- [179] Sauer, Lukas und Dominik Henrich: *Extended State Automata for Intuitive Robot Programming*. In: Zegloul, Saïd, Med Amine Laribi und Juan Sandoval (Herausgeber): *Advances in Service and Industrial Robotics. RAAD 2021. Mechanisms and Machine Science*, Band 102, Seiten 61–68, Cham, 2021. Springer International Publishing. [https://doi.org/10.1007/978-3-030-75259-0\\_7](https://doi.org/10.1007/978-3-030-75259-0_7).
- [180] Sauer, Lukas und Dominik Henrich: *Structure Synthesis for Extended Robot State Automata*. In: Müller, Andreas und Mathias Brandstötter (Herausgeber): *Advances in Service and Industrial Robotics. RAAD 2022. Mechanisms and Machine Science*, Band 120, Seiten 71–79, Cham, 2022. Springer International Publishing. [https://doi.org/10.1007/978-3-031-04870-8\\_9](https://doi.org/10.1007/978-3-031-04870-8_9).
- [181] Sauer, Lukas und Dominik Henrich: *Synchronisation in Extended Robot State Automata*. In: *2022 Sixth IEEE International Conference on Robotic Computing (IRC)*, Seiten 360–363. IEEE, 2022. <https://doi.org/10.1109/IRC55401.2022.00070>.
- [182] Sauer, Lukas, Dominik Henrich und Wim Martens: *Towards Intuitive Robot Programming Using Finite State Automata*. In: Benzmüller, Christoph und Heiner Stuckenschmidt (Herausgeber): *KI 2019: Advances in Artificial Intelligence. KI 2019. Lecture Notes in Computer Science*, Band 11793, Seiten 290–298, Cham, 2019. Springer International Publishing. [https://doi.org/10.1007/978-3-030-30179-8\\_25](https://doi.org/10.1007/978-3-030-30179-8_25).



---

## Sonstige eigene Arbeiten

---

- [183] Sauer, Lukas: *Erweiterung der intuitiven Roboterprogrammierung mit deterministischen endlichen Automaten*. Masterarbeit, Universität Bayreuth, 2018.





## **Eidesstattliche Versicherung**

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die von mir angegebenen Quellen und Hilfsmittel verwendet habe. Weiterhin erkläre ich, dass ich die Hilfe von gewerblichen Promotionsberatern bzw. -vermittlern oder ähnlichen Dienstleistern weder bisher in Anspruch genommen habe, noch künftig in Anspruch nehmen werde. Zusätzlich erkläre ich hiermit, dass ich keinerlei frühere Promotionsversuche unternommen habe.

Bayreuth, den 13. April 2024

Lukas Sauer