Intuitive sensorbasierte, editierbare, kinästhetische Programmierung von Pick-and-Place-Aufgaben für Mehrrobotersysteme

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

von
Michael Riedl
aus Tirschenreuth

1. Gutachter: Prof. Dr. Dominik Henrich
2. Gutachter: Prof. Dr.-Ing. Jörg Krüger

Tag der Einreichung: 26.08.2021
Tag des Kolloquiums: 26.01.2022

Danksagung

An dieser Stelle ist es an der Zeit, all jenen zu danken, die mich in den vergangenen Jahren beim erfolgreichen Abschluss meiner Promotion unterstützt haben. Mein besonderer Dank gilt meinem Doktorvater Prof. Dr. Dominik Henrich für die Möglichkeit, an seinem Lehrstuhl zu promovieren, für die konstruktiven Diskussionen und hilfreichen Denkanstöße bei der Konkretisierung und Bearbeitung des Themas.

Bedanken möchte ich mich auch bei meinen Kollegen am Lehrstuhl für die organisatorische und technische Unterstützung, die aufschlussreichen Diskussionen und die zahlreichen (nicht nur) fachlichen Kaffeeküchengespräche. Weiterhin möchte ich mich bei meinen Studenten bedanken, die mit ihren Arbeiten dafür gesorgt haben, dass neue Ideen und Aspekte dieser Arbeit vorangetrieben wurden.

Nicht zuletzt möchte ich mich bei meiner Familie und meinen Freunden bedanken, die mich auf dem Weg meiner Promotion begleitet und mich stets unterstützt und motiviert haben und damit dazu beitrugen, dass diese ein Erfolg wurde.

Zusammenfassung

Intuitive sensorbasierte, editierbare, kinästhetische Programmierung von Pick-and-Place-Aufgaben für Mehrrobotersysteme

Das Programmieren von Robotern ist meist kosten- und zeitintensiv und wird in der Regel von Systemintegratoren durchgeführt. Für kleine und mittlere Unternehmen (KMU) stellt dies ein Problem dar, da durch geringe Losgrößen Robotersysteme häufig neu programmiert werden müssen. Um Robotersysteme für KMU wirtschaftlicher zu machen muss der Programmierprozess günstiger und schneller gemacht werden. Eine Möglichkeit dies zu erreichen ist, dass Werker:innen in KMU, anstelle von Systemintegratoren, die Roboter mit Hilfe eines intuitiven Programmiersystems für Nicht-Expert:innen programmieren.

Diese Arbeit stellt ein neuartiges Konzept für ein intuitiv zu bedienendes, schnelles Mehrroboter Programmiersystem für Pick-and-Place Anwendungen vor. Es basiert auf der kinästhetische Programmierung von Robotern in Form der Playback Programmierung. Ziel ist es, Domänen-Expert:innen ohne textuelle Programmierkenntnisse das Programmieren von Robotern zu ermöglichen.

Der vorgeschlagene Ansatz setzt auf vier Kernaspekte: Darstellen, Editieren, Kontrollieren und Synchronisieren. Unter dem Aspekt *Darstellen* wird zunächst eine kontextfreie Grammatik für die kinästhetisch demonstrierten Roboterprogramme in Form einer Backus-Naur-Form (BNF) entwickelt. Die Elemente dieser BNF werden anschließend übersetzt in eine skalierbare und geschichtete graphische Darstellung entlang einer Zeitleiste. Für den Aspekt *Editieren* wird ein Konzept zum graphischen Editieren der kinästhetisch demonstrierten Roboterprogramme entlang der Zeitleisten präsentiert. Unter dem Aspekt *Kontrollieren* werden Konzepte für sensorbasierte Schleifen und Verzweigungen in kinästhetisch demonstrierten Roboterprogrammen vorgestellt. Zusätzlich wird ein Konzept für Schleifen-Inkremente definiert, welche das Programmieren von Stapel- und Palettieraufgaben ermöglichen. Für den Aspekt *Synchronisieren* wird ein Konzept zur zeitlichen Synchronisation mehrerer Roboter mit Hilfe von Synchronisationspunkten und Synchronisationsintervallen erarbeitet. Dieses Konzept wird auf eine Synchronisation zwischen Mensch und Roboter erweitert. Alle Konzepte der vier Aspekte werden in ein prototypisches Programmiersystem integriert.

Das Programmiersystem wird in einer Studie mit 42 Teilnehmer:innen hinsichtlich intuitiver Bedienbarkeit, bestehend aus Effektivität, mentaler Effizienz und Zufriedenheit, evaluiert. Die Ergebnisse der Evaluation zeigen ein intuitiv zu bedienendes Programmiersystem.

Abstract

Intuitive Sensor-based, Editable, Kinesthetic Programming of Pick-and-Place Tasks for Multi-Robot-Systems

Programming robots is usually costly and time-consuming and is usually carried out by system integrators. This poses a problem for small and medium-sized enterprises (SMEs), as robot systems often have to be reprogrammed due to small batch sizes. In order to make robot systems more economical for SMEs, the programming process must be made cheaper and faster. One possibility to achieve this is that workers in SMEs, instead of system integrators, program the robots with the help of an intuitive programming system for non-experts.

This thesis presents a novel concept for an intuitive, fast multi-robot programming system for pick-and-place applications. It is based on the kinesthetic robot programming approach in the form of playback programming. The goal is to enable domain experts without textual programming knowledge to program robots.

The proposed approach is based on four core aspects: displaying, editing, controlling and synchronizing. Under the aspect of *displaying*, a context-free grammar for the kinesthetically demonstrated robot programs is developed as a Backus-Naur form (BNF). The elements of this BNF are then translated into a scalable, layered graphical representation along a timeline. For the *editing* aspect, a concept for the graphical editing of the kinesthetically demonstrated robot programs along the timelines is presented. Under the aspect *controlling*, concepts for sensor-based loops and branches in kinesthetically demonstrated robot programs are introduced. In addition, a concept for loop-increments is defined, which enables the programming of stacking- and palletizing tasks. For the aspect *synchronizing* a concept for the temporal synchronization of multiple robots with the help of synchronization points and synchronization intervals is developed. This concept is extended to a synchronization between humans and robots. All concepts of the four aspects are integrated into a prototype of the programming system.

The programming system is evaluated in a study with 42 participants in terms of intuitive usability, consisting of effectiveness, mental efficiency and satisfaction. The results of the evaluation show a programming system that is intuitive to use.

Inhaltsverzeichnis

KAPITEL 1

Einleitung

Inhalt

Durch die Verfügbarkeit von neuen und immer günstiger werdenden Robotersystemen, welche im Bereich der kollaborativen Roboter anzusiedeln sind, wird die Anschaffung eines Robotersystems auch für kleine und mittlere Unternehmen (KMU) immer interessanter. Trotz der günstigen Anschaffungspreise besteht weiterhin das Problem, dass Expert:innen im Bereich der Roboterprogrammierung benötigt werden, um die Roboter programmieren zu können. Durch die häufig wechselnden Aufgaben und kleinen Losgrößen in KMU ist es notwendig, dass ein schnelles und intuitives Roboter-Programmiersystem existiert, welches von den Mitarbeiter:innen der KMU ohne zusätzliche Robotik- und Programmierkenntnisse verwendet werden kann [Schraft2006].

Diese Arbeit präsentiert ein neuartiges Konzept für ein schnelles, intuitives, kinästhetisches Roboter-Programmiersystem für Benutzer:innen ohne Robotik- und Programmierkenntnisse. Dazu wird zunächst in Abschnitt 1.1 das Thema intuitive Roboterprogrammierung anhand von KMU motiviert. Anschließend werden in Abschnitt 1.2 die Ziele dieser Arbeit erläutert indem die Aufgabenstellung und die daraus abgeleiteten Forschungsfragen aufgezeigt werden. In Abschnitt 1.3 werden die Rahmenbedingungen festgelegt, für welche das System erstellt wurde. Abschließend gibt Abschnitt 1.4 einen Überblick über die Kapitel dieser Arbeit.

1.1 Motivation

Einer der größten Faktoren, welche Firmen davon abhalten, Robotersysteme anzuschaffen sind die Kosten und das fehlende Robotik-Wissen [Kildal2018]. Die Kosten für Robotersysteme setzen sich zusammen aus den Anschaffungskosten für Roboter und Peripherie, den Projektmanagementkosten und den Systems Engineering Kosten, zu denen auch die Programmierkosten gehören. Die Preisentwicklung für Industrieroboter in den USA wird in [statista2015] dargestellt. Dabei zeigen die Prognosen für das Jahr 2025, dass die Kosten für die Anschaffung und auch die Projektmanagementkosten sinken sollen. Am stärksten prognostiziert ist der Rückgang der Systems Engineering Kosten, zu denen auch die Programmierkosten zählen. Dies ist dadurch erreichbar, dass das Programmieren von Robotern in Zukunft auch von Nicht-Expert:innen ohne Robotik- und Programmierkenntnisse mit einem intuitiven System erledigt werden kann [OConnor2016]. Mit dieser Art der Endnutzer-Programmierung wird dafür gesorgt, dass Roboter für die Kleinserienautomatisierung in KMU interessanter werden. Dadurch werden sowohl die Kosten der Robotersysteme gesenkt, als auch das notwendige Robotik-Wissen, welches die Firmen besitzen müssen.

Als Folge der günstiger werdenden Roboter steigt sowohl der Umsatz [statista2019] der mit diesen erwirtschaftet wird, als auch die Anzahl der Roboter [statista2020a] weltweit kontinuierlich an. Beides soll prognostiziert auch in den nächsten Jahren weiter steigen. Insbesondere bei den mittleren Unternehmen mit 50 bis 249 Mitarbeitern hat sich der Anteil an Unternehmen im verarbeitenden Gewerbe in Deutschland, welche Industrieroboter einsetzen, zwischen 2018 und 2020 von 20% auf 27% erhöht, was eine Steigerung um 35% bedeutet [statista2020b]. Diese Zahlen zeigen, dass Automatisierung in KMU ein zentrales Thema ist. Für das Feld der Endnutzer-Roboterprogrammierung, also der intuitiven Roboterprogrammierung welche von Werker:innen ohne Programmier- und Robotik-Kenntnisse durchgeführt wird, bietet sich hier großes Potential. Dies ist insbesondere der Fall, weil in der EU 99.4% aller Unternehmen zu den KMU zählen [DESTATIS2018].

Derzeit stellt sich Situation in KMU so dar, dass meist kleine Stückzahlen gefertigt werden müssen, wodurch ein Robotersystem häufig umprogrammiert werden muss. In Abbildung 1.1 ist der zeitliche Ablauf der Phasen eines Robotersystems dargestellt. Dabei ist erkennbar, dass die (Um-)Konfigurationszeit, zu der auch die Programmierdauer gehört, prozentual einen immer größeren Anteil an der Gesamtzeit eines Robotersystems einnimmt, je kürzer die Produktionszeiten sind. Durch die direkte Proportionalität zwischen Produktionszeit und Losgröße ist bei KMU mit kleinen Losgrößen der Anteil der Produktionszeit an der Gesamtzeit besonders klein. Dementsprechend muss oft umkonfiguriert bzw. umprogrammiert werden, wodurch Stillstandszeiten entstehen. Es ist also notwendig, die Programmierdauer zu senken, damit die Produktionszeiten von Robotern in KMU erhöht werden können.

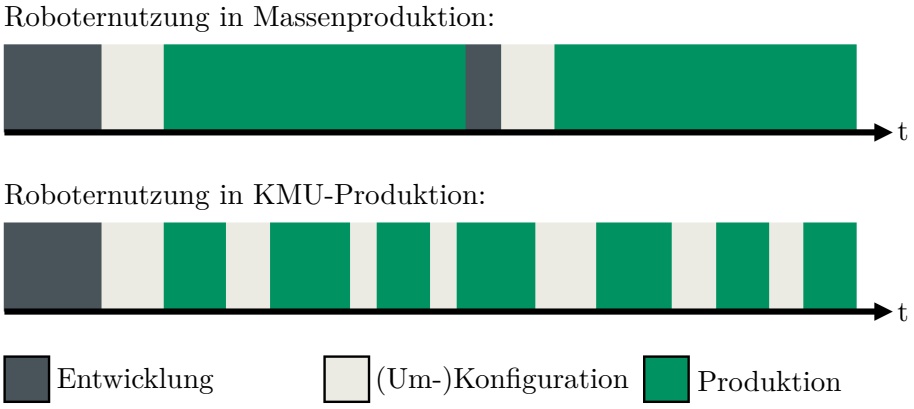


Abbildung 1.1: Vergleich der Phasen der Roboternutzung in Massenproduktion und KMU-Produktion nach [Dietz2012].

Dabei ist es nicht unbedingt notwendig, ein optimales Roboterprogramm im Hinblick auf die Ausführungszeit zu erstellen. Wichtiger ist bei KMU die schnelle und einfache Programmierbarkeit, sodass die bereits erwähnten Stillstandszeiten verringert werden können.

Um die Verbreitung von Robotern in KMU weiter voranzubringen ist es notwendig, ein schnelles und intuitives Roboter-Programmiersystem bereitzustellen. Dieses Programmiersystem muss auf kleine Losgrößen und Nicht-Exptert:innen ohne Robotik- und Programmierkenntnisse ausgerichtet sein, um auf den teuren und häufigen Einsatz von Systemintegratoren zum Programmieren der Roboter verzichten zu können [Grüninger2009]. In diese Richtung der schnellen und intuitiven Roboter-Programmierung durch Nicht-Expert:innen soll mit dieser Arbeit ein weiterer Schritt gemacht werden.

1.2 Ziele dieser Arbeit

Das Ziel dieser Arbeit ist, ein neuartiges Konzept für ein intuitives, sensorbasiertes Roboter-Programmiersystem zu entwickeln, implementieren und evaluieren, welches Benutzer:innen ohne Robotik- und textuelle Programmierkenntnisse ein schnelles Programmieren von Pick-and-Place Aufgaben für Mehrrobotersysteme erlaubt. Dafür soll die Eigenschaft des kinästhetischen Führens von Robotern verwendet werden, da diese im Allgemeinen als intuitiv bewertet werden kann [Villani2018]. Zusätzlich sollen die entstandenen Roboterprogramme nachträglich editiert und in den Roboterprogrammen auf äußere Einflüsse mittels lokalen, am Roboter befestigten Sensoren, reagiert werden können.

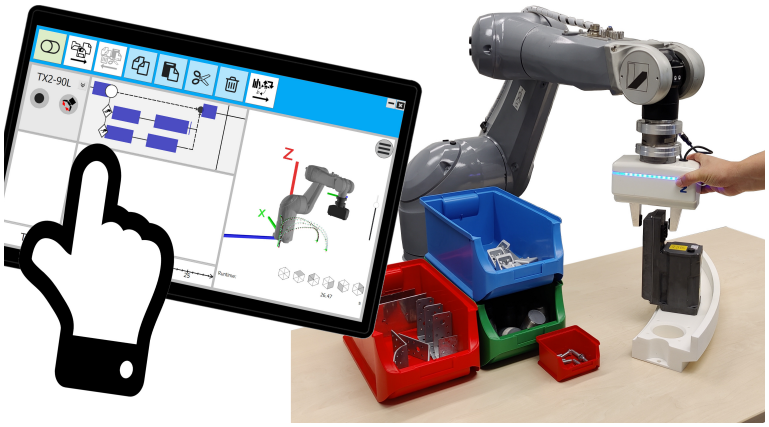


Abbildung 1.2: Fotomontage eines potentiellen intuitiven kinästhetischen Programmiersystems für Pick-and-Place Aufgaben.

Die Benutzer:innen sollen nach einer kurzen Einführung in der Lage sein, die Roboter mit Hilfe des Programmiersystems schnell und intuitiv programmieren können. Weiterhin soll die Anzahl der Roboter skalierbar sein. Dabei soll das Programmiersystem so konzipiert sein, dass es beliebig viele Roboter inklusive deren Programme darstellen, bearbeiten und ausführen kann. Bei mehreren vorhandenen Robotern soll eine Möglichkeit bestehen, diese untereinander zu koordinieren, um bei der Ausführung gemeinsame Aufgaben erledigen, oder um einen gemeinsamen Arbeitsraum absichern zu können. Dabei soll im Rahmen dieser Arbeit eine Beschränkung auf die Domäne der Pick-and-Place Aufgaben erfolgen. Abbildung 1.2 zeigt schematisch, wie ein solches Programmiersystem für einen Roboter aussehen könnte.

Zum Lösen der Aufgabe waren folgende Rahmenbedingungen gegeben: Ein Robotersystem mit beliebig vielen stationären sechs- bzw. siebenachsigen Roboterarmen, die kinästhetisch geführt werden können. Zusätzlich ist an jedem Roboterarm ein Greifer montiert und es steht eine 2D-RGB Eye-in-Hand Kamera an einem Roboter zur Verfügung. Die Benutzungsoberfläche des Programmiersystems soll auf einem Tablet-PC mit Touchscreen ausgeführt werden. Es sollen keine weiteren externen Sensoren und auch kein Weltmodell zur Lösung des Problems verwendet werden.

Zur Bearbeitung der vorgestellten Aufgabenstellung wurde sechs Forschungsfragen formuliert. Diese werden im Verlauf dieser Arbeit in den einzelnen Kapiteln diskutiert und beantwortet.

Zentraler Aspekt bei der Programmierung von Robotern ist die Darstellung der entstandenen Roboterprogramme. Insbesondere bei kinästhetisch programmierten Robotersystemen ist die Darstellung verantwortlich dafür, ob ein Programmiersystem als intuitiv bedienbar bewertet werden kann. Aus diesem Grund wird zuerst folgende Fragestellung im Rahmen dieser Arbeit behandelt:

F1 Wie lassen sich geschachtelte Roboterprogramme strukturiert und skalierbar graphisch entlang einer Zeitleiste darstellen?

Das nachträgliche Editieren von kinästhetisch demonstrierten Roboterprogrammen wird, sofern es überhaupt zur Verfügung gestellt wird, meist in einer Simulationsumgebung durch anpassen der einzelnen Konfigurationen ermöglicht. Zum graphischen Editieren eines kinästhetisch demonstrierten Roboterprogramms in einer Zeitleistendarstellung konnte im Rahmen der Recherche zu dieser Arbeit keine Veröffentlichung gefunden werden. Aus diesem Grund ergibt sich folgende Fragestellung:

F2 Auf welche Art und Weise lassen sich kinästhetisch demonstrierte Roboterprogramme graphisch entlang von Zeitleisten editieren?

Kinästhetisch demonstrierte Roboterprogramme bestehen meist aus aufgezeichneten Trajektorien, welche anschließend exakt wieder ausgeführt werden können. Während der Ausführung kann dabei nicht auf die Umwelt reagiert, und somit auch nicht der Kontrollfluss des Programms verändert werden. Dies soll im Rahmen dieser Arbeit ermöglicht werden, wodurch sich folgende zwei Fragestellungen ergeben haben:

F3 Wie lassen sich Konzepte der imperativen Programmierung wie Schleifen und Verzweigungen auf das kinästhetische Programmierkonzept übertragen?

F4 Wie lässt sich das Konzept von Schleifeninkrementen auf die kinästhetische Roboterprogrammierung übertragen?

In der Literatur finden sich kinästhetische Programmieransätze in der Regel nur für einzelne Roboter. Sollen diese Ansätze auf Mehrrobotersysteme erweitert werden, wird ein Konzept zum Koordinieren der Roboter benötigt, so dass geteilte Arbeitsräume abgesichert oder gemeinsame Aufgaben definiert werden können. Dies führt zu folgender Fragestellung:

F5 In wie weit lässt sich das Programmierkonzept auf die Zusammenarbeit mehrerer Roboter und auf die Zusammenarbeit zwischen Mensch und Roboter erweitern?

Die Ergebnisse der fünf vorhergehenden Fragen sollen in ein Gesamtsystem integriert werden. Zu diesem Gesamtsystem soll herausgefunden werden, ob es für Benutzer:innen ohne Robotik- und Programmierkenntnisse intuitiv bedienbar ist. Daraus ergibt sich folgende Fragestellung:

F6 Inwiefern lässt sich das entstandene kinästhetische Programmierkonzept und dessen prototypische Implementierung intutiv bedienen? Wie spiegelt sich das in Effektivität, mentaler Effizienz und Zufriedenheit der Programmierung wider?

Forschungsfrage F6, welche auf die Evaluation des entstandenen Programmierkonzepts abzielt, wird noch mit weiteren Unterfragen spezifiziert. Dadurch sollen zusätzliche Eigenschaften des Konzepts evaluiert werden:

F6.1 Wie hoch ist die erwartete Komplexität des Programmiersystems im Vergleich zur tatsächlich empfundenen?

F6.2 Gibt es bei der Bewertung des Systems Unterschiede zwischen Robotik-Expert:innen, Domänen-Expert:innen und Nicht-Expert:innen?

F6.3 Wie verständlich sind die für die prototypische Implementierung verwendeten Symbole der graphischen Benutzungsoberfläche?

F6.4 Welche der beiden entworfenen Darstellungsmöglichkeiten von Bewegungsbefehlen entlang einer Zeitleiste (Graph- oder Blockdarstellung) bevorzugen die Benutzer:innen des Programmiersystems?

F6.5 Bevorzugen Benutzer:innen bei repetitiven Aufgaben eher Kopieren&Einfügen oder eine Schleife?

Die Bearbeitung und Beantwortung der Forschungsfragen findet sukzessiv im Verlauf dieser Arbeit statt. Kapitel 4 behandelt Forschungsfrage F1 und Kapitel 5 Forschungsfrage F2. Die Forschungsfragen F3 und F4 werden in Kapitel 6 bearbeitet. In Kapitel 7 wird die Forschungsfrage F5 betrachtet, bevor in Kapitel 8 Forschungsfrage F6 mit den Unterfragen F6.1 bis F6.5 behandelt werden.

1.3 Abgrenzung

Eine Aufteilung der Möglichkeiten zur Mensch-Roboter-Kooperation in Kollaboration, Kooperation und Koexistenz wird in [Onnasch2016] definiert. Auf diese Arbeit übertragen lässt sich festhalten, dass die Programmierung der Roboter als Kollaboration eingestuft werden kann, da dabei eine direkte Zusammenarbeit zwischen Mensch und Roboter stattfindet. Die Ausführung der Roboterprogramme hingegen findet in der Regel in Koexistenz statt, da sich Mensch und Roboter zeitlich und räumlich nur sehr begrenzt treffen und jeder eigenständig arbeitet. Eine Ausnahme davon bilden die Zeitpunkte, an denen eine Mensch-Roboter-Synchronisation stattfindet. Diese ist als Kooperation einzustufen, da an diesen Stellen sowohl Mensch als auch Roboter ein gemeinsames Ziel verfolgen und der Mensch dabei in der Nähe des Roboters arbeitet.

Die Konzepte dieser Arbeit werden für die Domäne Kleinserienfertigung in KMU entwickelt. Dabei sollen nur Pick-and-Place Aufgaben betrachtet werden. Andere Aufgabenarten, wie zum Beispiel Oberflächenbehandlung oder Sprühprozesse, werden in dieser Arbeit nicht behandelt.

Das verwendete Programmierparadigma dieser Arbeit ist die kinästhetische Roboterprogrammierung in Form der Playback bzw. Teach-In Programmierung. Diese zeichnet sich dadurch aus, dass keine Lernverfahren, wie zum Beispiel beim Programmieren durch Vormachen, verwendet werden. Bei dem hier verwendeten Paradigma ist zu jedem Zeitpunkt deterministisch vorhersagbar, welche Bewegungen der Roboter ausführen wird. Dies ist der Fall, da während der Programmierung die Trajektorie des Roboters exakt aufgezeichnet wird und anschließend identisch wieder abgespielt werden kann.

Des Weiteren wird in dieser Arbeit nicht das Problem der Steuerung des Roboters und der Umsetzung des kinästhetischen Führens behandelt. Es wird angenommen, dass die verwendeten Roboter eine Funktion zum kinästhetischen Führen bereitstellen, und dass die Roboter eine Schnittstelle besitzen, über welche Fahrbefehle gesendet werden können.

Abschließend wird in dieser Arbeit nicht das Thema Sicherheit in der Mensch-Roboter-Kooperation betrachtet. Für das Robotersystem werden MRK zertifizierte Komponenten verwendet und dabei angenommen, dass alle Anforderungen nach ISO 10218 [ISO10218] erfüllt sind. Für den produktiven Einsatz in einem Unternehmen müsste die konkrete Anwendung zusätzlich mit einer Risikoanalyse nach ISO/TS 15066 [TS15066] bewertet und gegebenenfalls weitere Zertifikate eingeholt und Überprüfungen durchgeführt werden.

1.4 Kapitelübersicht

Zunächst wird in Kapitel 2 der Stand der Forschung für die unterschiedlichen Aspekte dieser Arbeit behandelt. Dabei wird zunächst die Roboterprogrammierung, dann graphische Benutzungsoberflächen und zum Schluss Intuitivität betrachtet.

Daraufhin wird in Kapitel 3 das Grundkonzept der Arbeit im Hinblick auf die kinästhetische Programmierung, die Aufteilung in Programmier- und Ausführungsphase und den Aufbau und die Architektur des Programmiersystems behandelt.

In Kapitel 4 wird der Aspekt *Darstellen* näher erläutert. Dabei wird zunächst eine Backus-Naur-Form für kinästhetisch demonstrierte Roboterprogramme entworfen. Anschließend wird eine zur Backus-Naur-Form duale skalierbare graphische Darstellung von kinästhetisch demonstrierten Roboterprogrammen entlang einer Zeitleiste vorgestellt, bevor zum Abschluss die Evaluationsergebnisse für die Verständlichkeit der graphischen Darstellung präsentiert werden.

Der Aspekt *Editieren* wird in Kapitel 5 genauer betrachtet. Dabei wird näher auf die Kopieren&Einfügen Funktionen für das zeitliche Editieren kinästhetischer Roboterprogramme in den Zeitleisten eingegangen. Im weiteren Verlauf wird die Drag-and-Drop Funktionalität von Kontrollfluss-Elementen in der Zeitleiste präsentiert. Abschließend werden Konzepte aufgezeigt, um mit dem Problem der Sprungstellen beim Editieren der Trajektorien umgehen zu können. Hierfür wird ein automatischer Ansatz und ein Ansatz mit kinästhetischem Führen gezeigt.

Kapitel 6 behandelt den Aspekt *Kontrollieren*. Dabei wird ein Konzept vorgestellt, wie mit Sensorwerten, die auf Ähnlichkeit verglichen werden können, sensorbasierte Kontrollstrukturen in Form von Schleifen und Verzweigungen für die kinästhetische Roboterprogrammierung umgesetzt werden können. Dadurch erhält das Programmiersystem die Möglichkeit auf Umwelteinflüsse reagieren zu können. Zuletzt wird das Schleifen-Inkrement vorgestellt, welches eine Erweiterung sensorbasierter Schleifen ist, um Aufgaben wie Stapeln und Palettieren lösen zu können, ohne dass alle Teilbewegungen dieser Aufgaben explizit kinästhetisch von den Benutzer:innen demonstriert werden müssen.

Zur zeitlichen Koordination mehrerer Roboter mit Hilfe des in dieser Arbeit entwickelten Programmiersystems wird in Kapitel 7 der Aspekt *Synchronisieren* vorgestellt. In diesem Kontext werden zunächst Synchronisationspunkte und Synchronisationsintervalle definiert. Anschließend werden Konzepte für die Synchronisation zwischen mehreren Robotern und die Synchronisation zwischen Roboter und Mensch präsentiert.

Die Evaluation des entstandenen Programmiersystems wird in Kapitel 8 vorgestellt. Dazu wird zuerst der Gesamtdemonstrator erläutert, bevor das Studiendesign, die von den Teilnehmer:innen zu lösende Aufgabe und die Ergebnisse der Evaluation hinsichtlich Intuitivität, Schnelligkeit und weiteren Merkmalen vorgestellt werden. Abschließend werden die Grenzen des entwickelten Programmierkonzepts erläutert.

Den Abschluss dieser Arbeit bildet das Fazit in Kapitel 9. Darin werden die Ergebnisse der Arbeit kurz zusammengefasst und anschließend die bereits vorgestellten Forschungsfragen beantwortet. Abschließend wird ein Ausblick gegeben, welche potentiellen Stoßrichtungen zur Weiterentwicklung des in dieser Arbeit vorgestellten Programmierkonzeptes angestrebt werden können.

KAPITEL 2

Stand der Forschung

Inhalt

In diesem Kapitel wird ein Überblick über den Stand der Forschung zu den Aspekten dieser Arbeit gegeben. Zunächst wird in Abschnitt 2.1 das Gebiet der Roboterprogrammierung genauer betrachtet und insbesondere verschiedene Ansätze zur intuitiven und zur kinästhetischen Programmierung von Robotern aufgezeigt. Anschließend wird in Abschnitt 2.2 näher auf das Thema graphische Benutzungsoberflächen, sowohl für die Roboterprogrammierung, als auch für Videoschnitt-Programme eingegangen. Da das Ziel dieser Arbeit ist, ein intuitiv zu bedienendes Programmierkonzept zu entwickeln, wird abschließend in Abschnitt 2.3 das Thema intuitive Bedienbarkeit definiert und erläutert, wie diese evaluiert werden kann.

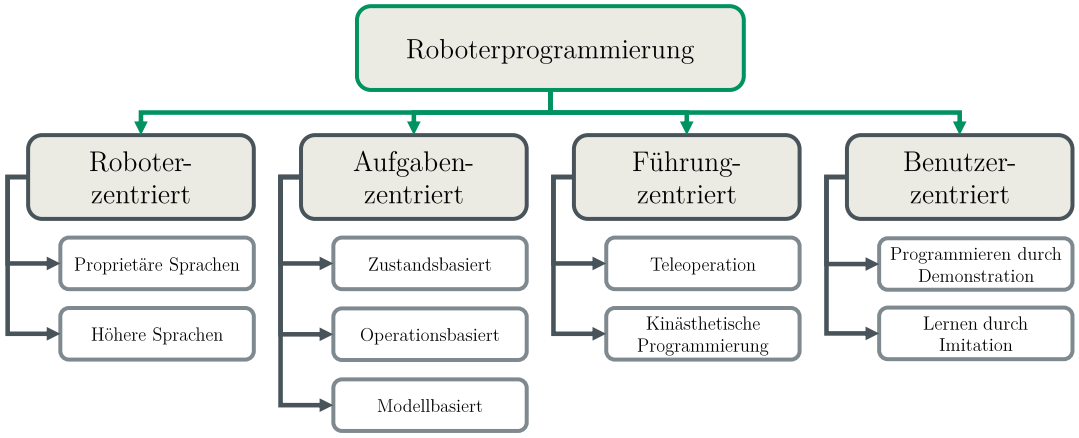


Abbildung 2.1: Einteilung der Roboterprogrammierung in sogenannte Zentrierungen nach [Orendt2019].

2.1 Roboterprogrammierung

Das Thema *Roboterprogrammierung* ist zentraler Bestandteil dieser Arbeit. Aus diesem Grund wird in diesem Abschnitt dieses Gebiet aufgegliedert und potentielle Lücken aufgezeigt. Hierfür wird zunächst das Feld der allgemeinen Roboterprogrammierung strukturiert und erläutert. Anschließend wird auf die verschiedenen intuitiven Ansätze näher eingegangen bevor im letzten Abschnitt das Thema Erweiterungen für die kinästhetische Roboterprogrammierung erläutert wird.

2.1.1 Allgemeine Roboterprogrammierung

Die Arten Roboter zu programmieren werden in der Literatur nach unterschiedlichen Kriterien gruppiert. So lässt sich zum Beispiel nach dem Ort der Eingabe gruppieren. Mit dieser Aufteilung lassen sich Programmiersysteme in On-line, Off-line und hybrid unterteilen (siehe [Krot2019] oder [Pan2012]). Unter On-line versteht man dabei die Programmierung direkt am Roboter (z.B. [Maeda2015]), unter Off-line die Programmierung unabhängig vom Roboter (z.B. [Gan2013]) und unter hybrid eine Mischform aus beiden (z.B. [Guhl2019]).

Eine weitere Möglichkeit Roboter-Programmiermethoden zu unterteilen ist nach der Art der Eingabe. Diese Art der Unterteilung kann insbesondere für die Off-line Programmiermethoden verwendet werden. In [Orendt2019] wird dabei unterteilt in textuell, graphisch und Icon-basiert. Bei der textuellen Programmierung wird das Roboterprogramm durch schreiben von Quellcode in einer textuellen Programmiersprache erstellt (z.B. [Blank2003]). Unter graphischer Programmierung versteht man im Allge-

meinen die Programmierung eines Roboters mit Hilfe eines Simulationssystems, welches eine Modellierung des Roboters und dessen Umwelt enthält (z.B. [Stumm2016]). Bei der Icon-basierten Programmierung wiederum können vordefinierte Bausteine aneinandergereiht werden und so Roboterprogramme erstellt werden (z.B. [Bischoff2002]).

In dieser Arbeit wird eine weitere Art der Gruppierung von Roboter-Programmiermethoden verwendet. Diese teilt das Feld der Roboterprogrammierung sogenannte Zentrierungen, welche nach dem zentrale Aspekt der jeweiligen Programmierart unterteilen. Nach [Lozano-Perez1983] lassen sich Arten der Roboterprogrammierung einteilen in Roboter-zentriert, Aufgaben-zentriert und Führungs-zentriert. In [Orendt2019] wurde diese Kategorisierung erweitert um die Benutzer-zentrierte Roboterprogrammierung. Abbildung 2.1 zeigt eine Übersicht über die Einteilung in Zentrierungen.

Bei der *Roboter-zentrierten* Programmierung steht der Roboter selbst im Mittelpunkt. Dabei müssen Benutzer:innen dem Robotersystem explizite Bewegungsbefehle, meist mit Hilfe von textuellen Programmiersprachen, geben. Die Roboter-zentrierte Programmierung lässt sich weiter unterteilen in proprietäre Programmiersprachen der Roboterhersteller und höhere Sprachen, welche herstellerübergreifend verwendet werden können. Beispiele für proprietäre Programmiersprachen der Roboterhersteller sind KRL von Kuka [Kuka2014], RAPID von ABB [ABB2010] oder VAL3 von Stäubli [Stäubli2016]. Zu den höheren Sprachen gehören zum einen die Schnittstellen der Roboterhersteller zur externen Ansteuerung ihrer Robotersysteme (z.B. Fast Research Interface (FRI) von Kuka [Schreiber2010] oder uniVAL drive von Stäubli [Stäubli2013]). Zum anderen gehören dazu aber auch Bestrebungen, eine universelle, für alle Roboterhersteller verwendbare Programmiersprache zu erstellen. Ein Beispiel dafür ist die aufgrund mangelnder Unterstützung der Hersteller nicht weiter entwickelte Industrial Robotic Language (IRL) [DIN66312].

Die *Aufgaben-zentrierte* Programmierung stellt die zu lösende Aufgabe in das Zentrum der Programmierung. Dabei müssen Benutzer:innen mit Hilfe von Zuständen oder Operationen den Programmablauf beschreiben. Die Programmierung erfolgt dabei entweder zustandsbasiert, operationsbasiert oder modellbasiert. Bei der zustandsbasierten Programmierung werden Startzustand, Endzustand und gegebenenfalls mehrere Zwischenzustände definiert, anhand derer das Robotersystem das Roboterprogramm erstellt (z.B. [Dennet2006]). Bei der operationsbasierten Programmierung wird eine Aufgabe als Folge von Operationen dargestellt, welche vom Programmiersystem nacheinander ausgeführt werden können (z.B. [Steinmetz2018], [DeSchutter2007]). Die modellbasierte Programmierung kombiniert den zustandsbasierten und operationsbasierten Ansatz, indem sowohl Zustände, als auch Operationen verwendet werden können. Daraus entstehen sogenannte Diskrete Ereignissysteme wie zum Beispiel Zustandsautomaten oder Aktionsprimitivnetze [Cassandras2009].

Bei der *Führungs-zentrierten* Programmierung ist das Führen des Roboters im Zentrum. Dabei programmieren Benutzer:innen ein Robotersystem entweder durch Teleoperation oder durch kinästhetisches Führen direkt am Roboterarm. Die Teleoperation funktioniert so, dass mit Hilfe einer Master-Kinematik der reale Roboter ferngesteu-

ert wird. Dies kann sowohl über ein reales Modell des Roboters (z.B. [Hokayem2006]) oder über ein virtuelles Modell mittels Virtual Reality oder Augmented Reality erfolgen (z.B. [Kohn2018]). Das kinästhetische Führen hingegen findet direkt am Roboterarm statt und kann entweder als Playback Programmierung oder als Teach-In Programmierung realisiert sein. Bei der Playback Programmierung wird Bewegung während der Demonstration aufgezeichnet und anschließend exakt wieder abgespielt (z.B. [Bascetta2013]). Dies wird teilweise auch Walk-Through Programmierung genannt [OSHA]. Bei der Teach-In Programmierung hingegen werden nur markante Punkte der Bewegung gespeichert und die Bewegung zwischen diesen Punkten bei der Wiedergabe interpoliert. Teilweise wird dieses Vorgehen auch Lead-Through Programmierung genannt [OSHA]. Anstelle des kinästhetischen Führens kann die Teach-In Programmierung auch mit Hilfe eines Teach-Pendants erfolgen.

Die *Benutzer-zentrierte* Programmierung hat die Benutzer:innen des Programmiersystems als zentralen Bestandteil. Darunter fallen Ansätze des Programmierens durch Vormachens, bei denen die Benutzer:innen eine Aufgabe demonstrieren und das Programmiersystem daraus das Roboterprogramm lernt. Dabei kann das Programmieren durch Vormachen sowohl durch Imitation (z.B. [Wu2010]), also dem Beobachten der Aufgabe mit Hilfe von Kameras und anderen Sensoren, als auch durch Demonstration (z.B. [Calinon2007]), also dem direkten zeigen der Aufgabe am Roboterarm erfolgen. Unterschieden wird dabei noch zwischen einmaligem Demonstrieren, sogenannten One-Shot Verfahren (z.B. [Orendt2016]), und mehrmaligem Demonstrieren, sogenannten Multi-Shot Verfahren (z.B. [Mollard2015]). Aus den demonstrierten Aufgaben leitet das Programmiersystem dann das Roboterprogramm ab, welches anschließend in angepasster Form für veränderte Situationen wieder ausgeführt werden kann.

Vergleicht man die vier Zentrierungen, so ist offensichtlich, dass die Roboter-zentrierte Programmierung nicht als intuitiv eingestuft werden kann, da textueller Programmcode geschrieben werden muss. Auch ist die Aufgaben-zentrierte Programmierung bis auf wenige Ausnahmen (z.B. [Steinmetz2018]) nicht als intuitiv einzustufen, da dabei Wissen über die Fähigkeiten der Roboter und die notwendige Modellierung der Zustände und Operationen verwendet werden muss. Bei der Führungs-zentrierten Programmierung hingegen muss der Roboter entweder direkt oder indirekt von den Benutzer:innen geführt werden, wohingegen bei der Benutzer-zentrierten Programmierung von den Benutzer:innen die Aufgaben demonstriert werden. Beide Zentrierungen können als grundsätzlich intuitiv eingestuft werden, da nur wenig Vorkenntnisse zum Programmieren der Roboter notwendig ist [Villani2018]. Im nächsten Abschnitt werden diese beiden Arten der Zentrierungen als Grundlage für die intuitive Roboterprogrammierung näher betrachtet.

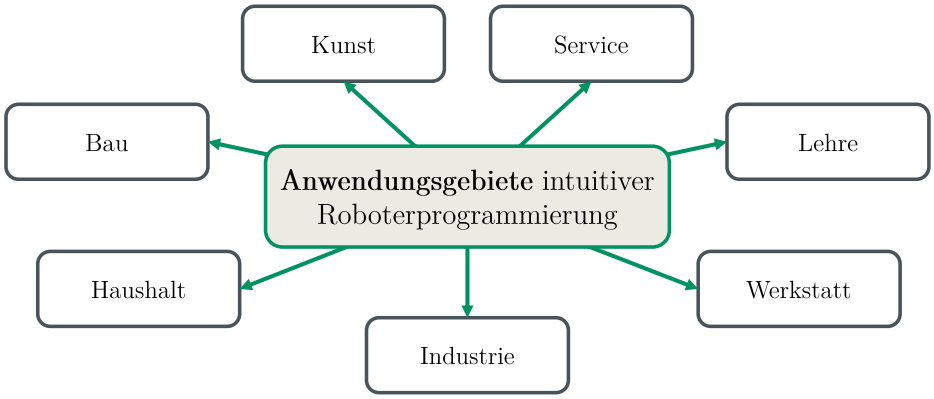


Abbildung 2.2: Die häufigsten Anwendungsgebiete intuitiver Roboterprogrammierung.

2.1.2 Intuitive Roboterprogrammierung

Eine Vielzahl von Roboter-Programmierverfahren beanspruchen von sich, intuitiv zu sein. Diese lassen in der Regel sich in die Führungs-zentrierte oder Benutzer-zentrierte Roboterprogrammierung aus dem letzten Abschnitt einordnen, wie Studien zu diesem Thema zeigen. Diese Studien geben entweder einen Überblick über die Mensch-Roboter-Kollaboration (z.B. [Chandrasekaran2015], [ElZaatari2019]) oder über die einfache und intuitive Roboter-Programmierung (z.B. [Ajaykumar2020], [Krot2019], [Rossano2013]). In dieser Arbeit wird wird das Feld der intuitiven Roboterprogrammierung mit Hilfe der unterschiedlichen Anwendungsgebiete und der verwendeten Eingabemodalitäten gegliedert.

Die intuitive Roboterprogrammierung wird in verschiedensten Anwendungsgebieten verwendet und erforscht. Ein Überblick über die häufigsten Anwendungsgebiete ist in Abbildung 2.2 dargestellt. So werden intuitive Ansätze zur Roboterprogrammierung zum Beispiel in der Lehre eingesetzt, um den Schüler:innen und Student:innen Grundlagen der Informatik und Robotik zu vermitteln [Bravo2017]. Dabei gibt es Systeme, die auf Basis von textueller Programmierung (z.B. [Blank2003]) oder visuellem Programmieren (z.B. [Jost2014], [Cross2013], [Rahul2014]) arbeiten. Auch für Serviceroboter (z.B. [Datta2012]) werden Konzepte zum Programmieren dieser entwickelt. Des Weiteren gibt es Ansätze, welche in der Kunst (z.B. [Vick2014]) oder auf Baustellen (z.B. [Stumm2016]) verwendet werden.

Ein weiteres Anwendungsgebiet der intuitiven Programmierung ist der Haushalt bzw. die Werkstatt. In diesem Gebiet gibt es Ansätze im Bereich des Programmieren durch Vormachen (z.B. [Groth2014a], [Liang2018]), sprachbasierte Ansätze (z.B [Wölfel2021]) und CAD-basierte Ansätze (z.B. [Stumm2018]).

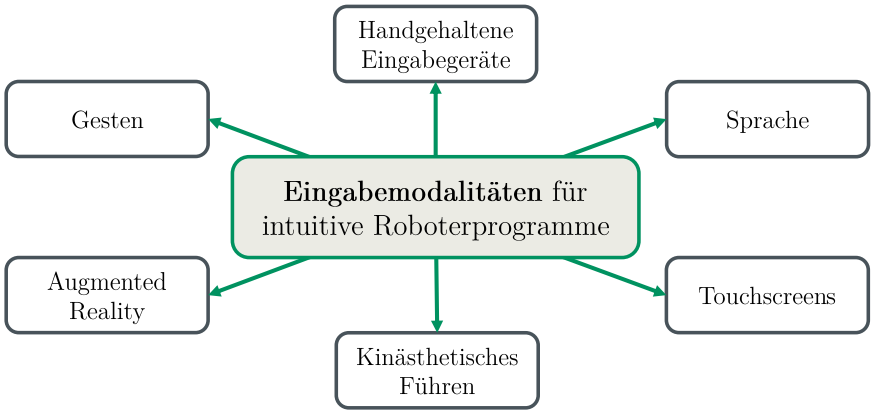


Abbildung 2.3: Die häufigsten Eingabemodalitäten für intuitive Roboterprogramme.

Das größte Anwendungsgebiet für die intuitive Roboterprogrammierung sind in der aktuellen Forschung die industriellen Anwendungen, insbesondere für kleine und mittlere Unternehmen [Heimann2020]. Hierbei werden die Robotersysteme in der Regel für die Oberflächenbehandlung, Handhabung oder Qualitätskontrolle eingesetzt. Bei der Oberflächenbehandlung gibt es intuitive Ansätze für Schweißen (z.B. [Takarics2008], [Ang2000]), Kleben (z.B. [Ni2017], [Kim2004]), Entgraten (z.B. [Bascetta2013]), Fasterspritzen (z.B. [Schmidt2020]) und Lackieren (z.B. [Ferraguti2017]). Robotersysteme für die Handhabung von Werkstücken können das Bestücken von Maschinen (z.B. [Cherubini2016]) oder die Montage von Baugruppen (z.B. [Rozo2016]) übernehmen. In der Qualitätskontrolle werden die intuitiven Robotersysteme häufig zum Testen fertiger Bauteile oder Geräte eingesetzt (z.B. [Massa2015]).

Zur Realisierung eines intuitiven Roboter-Programmiersystems ist es notwendig, passende Modalitäten für die Programmeingabe zu verwenden. In der Literatur finden sich unterschiedlichste Ansätze zur Eingabe von Roboterprogrammen. Eine Übersicht über Schnittstellen für die Mensch-Roboter-Interaktion ist in [Berg2020] zu finden. Die am häufigsten verwendeten Eingabemodalitäten sind in Abbildung 2.3 dargestellt. So können Roboterprogramme mit handgehaltenen Eingabegeräten wie Controller von Spielekonsolen (z.b. [Neto2010]), Stifte (z.B. [Pires2007]) oder Messsonden (z.B. [Antonelli2013]) erstellt werden. Für die Programmierung mittels eines Stiftes gibt es mit Wandelbots Teaching [Wandelbots] mittlerweile auch ein kommerzielles Produkt. Weitere Eingabemethoden sind die Verwendung von Gesten (z.B. [Berg2019], [Heimann2017]) oder Touchscreens zur Programmierung von Robotersystemen (z.B. [Kraft2017], [Mateo2014]). Bei der Programmierung mittels Sprache gibt es sowohl Ansätze, welche auf einer Spracherkennung (z.B. [Wölfel2021], [Akan2010]) basieren, als auch Ansätze, welche Chatbots (z.B. [Beschi2019]) verwenden.

Die beiden häufigsten Eingabemodalitäten für intuitive Robotersysteme sind die Programmeingabe mittels Augmented Reality und mittels kinästhetischen Führens. Beim Programmieren durch Augmented Reality gibt es sowohl Ansätze, welche Smartphones oder Tablet-PCs (z.B. [Zhang2018], [Frank2016], [Stadler2016]), als auch welche, die spezielle Augmented Reality Brillen verwenden (z.B. [Gadre2019], [Guhl2017], [Rudorfer2018]). Zusätzlich wird das Thema der nicht vorhandenen haptischen Rückmeldung bei der Programmierung mittels Augmented Reality behandelt (z.B. [Ni2017]).

Das Programmieren von Robotern mittels kinästhetischen Führens ist sehr verbreitet (z.B. [Schmidt2020], [Ferraguti2017], [Stumm2016], [Park2009], [Ang1999]). Dabei gibt es Ansätze, welche das Führen des Roboters mittels externer Kraft-Momenten-Sensorik umsetzen (z.B. [Fujii2016]), aber auch welche, die das Führen ohne externe oder interne Sensorik, sondern nur mit Hilfe der Motorströme realisieren (z.B. [Lee2016]). Die häufigste Methode ist jedoch die Verwendung aktueller zur Mensch-Roboter-Kollaboration zugelassenen Roboter, welche bereits das kinästhetische Führen mittels interner Kraft-Momenten-Sensorik unterstützen (z.b. [Landi2016]).

Bei aktuellen intuitiven Roboter-Programmiersystemen wird selten nur eine Methode der Eingabe des Roboterprogramms verwendet. Meist handelt es sich um multimodale Ansätze, welche eine Kombination aus verschiedenen Eingabemethoden verwenden. So gibt es zum Beispiel Systeme, welche Spracheingabe in Kombination mit einem Touchscreen [Makris2014], Spracheingabe in Kombination mit kinästhetischem Führen [Pires2009], Gestensteuerung in Kombination mit einem Bildschirm [Perzylo2016] oder Augmented Reality in Kombination mit Controller und kinästhetischem Führen [Quintero2018] verwenden.

In [Quintero2018] werden die beiden Modalitäten Augmented Reality und kinästhetisches Führen verglichen. Dabei wird zum Ergebnis gekommen, dass der Augmented Reality Ansatz etwas schneller ist und weniger motorische Beanspruchung benötigt, jedoch die mentale Beanspruchung beim kinästhetischen Ansatz geringer ist. Dadurch, dass für die Intuitivität eines Systems, wie in Abschnitt 2.3.1 gezeigt wird, die mentale Beanspruchung einer der drei Hauptaspekte neben der Effizienz und der Zufriedenheit ist, wird in dieser Arbeit als Eingabemodalität das kinästhetische Führen verwendet. In Kombination mit einer weiteren Modalität, der graphischen Benutzungsoberfläche auf Touchscreens, wird ein intuitives Roboter-Programmierkonzept entwickelt. Dazu untersucht der nächste Abschnitt die verwandten Arbeiten zu möglichen Erweiterungen der kinästhetischen Roboterprogrammierung.

2.1.3 Erweiterungen der kinästhetischen Roboterprogrammierung

Die kinästhetische Roboterprogrammierung wurde aufgrund der Intuitivität für das Erarbeiten eines Programmierkonzepts für Nicht-Expert:innen ausgewählt. Dadurch, dass das Programmierkonzept die vier Aspekte Darstellen, Editieren, Kontrollieren und Synchronisieren abbilden soll, wird der Stand der Forschung für Erweiterungen der kinästhetischen Roboterprogrammierung zu diesen Aspekten im Folgenden genauer betrachtet.

Der Aspekt *Darstellen* behandelt insbesondere die graphische Darstellung der kinästhetisch demonstrierten Roboterprogramme. Um diese strukturiert darstellen zu können wird eine formale Grundlage in Form einer Grammatik benötigt. Dadurch, dass die kinästhetisch demonstrierten Roboterprogramme eine Art Programmiersprache darstellen, eignet sich eine kontextfreie Grammatik in Form einer Backus-Naur-Form [McCracken2003] besonders dafür, da diese häufig für die Definition der Syntax von Programmiersprachen verwendet wird (z.B. Java [Møller] oder Python [Python]). Diese Art der textuellen Darstellung muss anschließend in eine graphische Darstellung übersetzt werden, um sie im Programmierkonzept verwenden zu können. Diese kann erstellt werden, indem Elemente aus UML-Diagramme [Fowler2003] und Ablaufdiagramme aus ISO 5807 [ISO5807] verwendet und angepasst werden. Dadurch, dass die zeitliche Komponente in der entwickelten Zeitleistendarstellung weiterhin vorhanden sein soll, wird in Kapitel 4 eine graphische Darstellung von kinästhetisch demonstrierten Roboterprogrammen entlang von Zeitleisten auf Basis einer BNF und von Ablaufdiagrammen entwickelt.

Mit dem Aspekt *Editieren* wird das zeitliche Editieren der kinästhetisch demonstrierte Trajektorien behandelt. In der Roboterprogrammierung gibt es zwar Ansätze, welche das räumliche Editieren von Trajektorien mit Hilfe einer graphischen Benutzungsoberfläche erlauben (z.B. [Meyer2007]), jedoch sind keine Ansätze aus der Literatur bekannt, welche ein manuelles zeitliches Editieren einer Roboter-Trajektorie entlang von Zeitleisten ermöglichen. Dies ist insbesondere dann sinnvoll, wenn die Roboterprogramme in Form von Trajektorien entlang von Zeitleisten visuell dargestellt werden und anschließend per Kopieren und Einfügen verändert werden sollen. Insbesondere in Texteditoren, aber auch bei Videoschnittprogrammen ist das Editieren mit Kopieren und Einfügen weit verbreitet [Tesler2012]. Dafür ist es nötig, mit dem Problem der Sprungstellen in der Trajektorie umzugehen, welche beim Entfernen oder Hinzufügen von Konfigurationen zu einer Trajektorie entstehen können. Das Programmiersystem muss an diesen Sprungstellen entweder automatisch eine Transferbewegung generieren, oder aber, wie in Kapitel 5.3 beschrieben, die Möglichkeit des kinästhetischen Führens von der Start- zur Zielposition dieser Sprungstelle bereitstellen. Für die automatische Erzeugung von Zwischenbahnen gibt es zum Beispiel zeitoptimale Ansätze im Gelenkraum, welche Randbedingungen wie maximale Beschleunigung oder maximale Geschwindigkeit einhalten [Craig2005]. Für das kinästhetische Führen von der Startkonfiguration zur Zielkonfiguration gibt es in der Literatur Ansätze, welche

präzises Führen ([Safeea2017], [Ragaglia2016]) oder Führen entlang eines vorgegebenen Pfades [Hanses2016] ermöglichen, jedoch keine, welche die Benutzer:innen dabei unterstützen, einen Roboter von einer Start- zu einer Zielposition manuell zu Führen. Auch gibt es Ansätze, welche die Steifigkeitswerte der verwendeten Roboter beim kinästhetischen Führen verändern ([Tykal2016], [Ficuciello2015], [Tsumugiwa2001]), so dass die Benutzer:innen haptische Rückmeldung während des Führens erhalten. Da keiner der in der Literatur zu findenden Ansätze den Anforderungen dieser Arbeit entspricht wird in Kapitel 5.3.2 ein multimodaler Ansatz mit veränderbaren Steifigkeitswerte zum Führen zu einer vorgegebenen Zielposition vorgestellt.

Der Aspekt *Kontrollieren* behandelt die Verwendung von Schleifen und Verzweigungen in Kombination mit am Roboter montierten Sensoren. In der Literatur gibt es Ansätze zu Kontrollstrukturen in Form von Schleifen und Verzweigungen in prozeduralen Programmiersprachen [Ullenboom2020]. Dabei sind Schleifen so aufgebaut, dass diese eine Schleifenbedingung und einen Schleifenrumpf besitzen. Je nach Schleifenart wird vor oder nach dem Durchlaufen der Schleife die Schleifenbedingung ausgewertet und abhängig davon entweder der Schleifenrumpf ein weiteres Mal ausgeführt oder übersprungen. Analog dazu ist eine Verzweigung aus beliebig vielen Zweigen aufgebaut. Jeder Zweig besitzt eine Bedingung und einen Rumpf. Wird bei der Programmausführung die Verzweigung erreicht, so werden der Reihe nach alle Zweige überprüft und die Bedingung evaluiert. Der erste Zweig, dessen Bedingung wahr ist wird ausgeführt, die anderen Zweige werden ignoriert. Dieses Konzept ist so in der Literatur nicht für kinästhetisch demonstrierte Roboterprogramme entlang von Zeitleisten zu finden. In dieser Arbeit wird in Kapitel 6 das Konzept der Kontrollstrukturen aus prozeduralen Programmiersprachen verwendet und mit Hilfe von Sensorwerten, die als Bedingungen fungieren, auf kinästhetisch demonstrierte Roboterprogramme angepasst. Um auch Stapel- und Palettieraufgaben mit dem Programmierkonzept lösen zu können, ohne alle Teilbewegungen explizit programmieren zu müssen, ist ein Verfahren zur Adaption der Trajektorie notwendig. Dieses Verfahren soll nach jedem Schleifendurchlauf die Konfigurationen innerhalb der Schleife um ein vorher definiertes Inkrement im kartesischen Raum verschieben. In der Literatur gibt es verschiedene Ansätze, um eine Roboter-Trajektorie anzupassen. Diese sind insbesondere beim Programmieren durch Vormachen zu finden. Dabei gibt es Ansätze, die aus einer Demonstration generalisieren und die Bewegung anpassen (z.B. [Wu2010], [Groth2014b]), aber auch solche, die aus mehreren Demonstrationen eine allgemeine Bewegung lernen, welche bei der Ausführung angepasst wird (z.B. [Nicolescu2003], [Cederborg2010]). Dadurch, dass in dieser Arbeit jedoch kinästhetisch demonstrierte Trajektorien verwendet werden, welche in der Ausführungsphase exakt so abgespielt werden sollen wie sie demonstriert wurden, sind diese Ansätze nicht verwendbar. Für das in dieser Arbeit entwickelte Konzept des Schleifen-Inkrements wurde in Kapitel 6.4 ein eigener Algorithmus zum Adaptieren der Trajektorie entwickelt, welcher durch die Funktion des proportionalen Editierens von Blender [Blender] motiviert wurde.

Unter dem Aspekt *Synchronisieren* werden Ansätze aus der Literatur zum Thema kinästhetisch programmierte Mehrrobotersysteme diskutiert. Wenige Arbeiten behandeln dieses Thema (z.B. [Park2009], [Zöllner2004]). Bei diesen Systemen wird aus mehreren demonstrierten Trajektorien eine neue Bewegungen gelernt und so die Roboterarme koordiniert. Dies ist für die in dieser Arbeit verwendete kinästhetische Programmierung nicht verwendbar, da nur eine Demonstration stattfindet und das Roboterprogramm exakt so ausgeführt werden soll, wie es demonstriert wurde. Deshalb wird zur zeitlichen Koordination der unterschiedlichen Roboter ein Synchronisationsmechanismus für die Zeitleistendarstellung benötigt. In Anlehnung an Barrieren aus der parallelen Programmierung [Ullenboom2020] wird in Kapitel 7 ein Konzept entwickelt, um kinästhetisch programmierte Roboter zeitlich aufeinander synchronisieren zu können. Dabei sollen die Benutzer:innen die Möglichkeit haben, Wartepunkte in die Roboterprogramme einzufügen, an denen einzelne Roboter aufeinander warten, und Bereich festzulegen, in denen die beteiligten Roboter gemeinsam eine Aufgabe ausführen.

Dieser Abschnitt hat gezeigt, dass in der Literatur zwar Ansätze für die Aspekte Darstellen, Editieren, Kontrollieren und Synchronisieren bestehen, es aber bisher keine Konzepte oder Systeme gibt, welche diese Ansätze mit der kinästhetischen Roboterprogrammierung in Kombination mit der Zeitleistendarstellung der Roboterprogramme vereinen. Zusätzlich zur Programmeingabe mittels kinästhetischen Führens wird im nächsten Abschnitt das Thema graphische Benutzungsoberflächen betrachtet.

2.2 Graphische Benutzungsoberflächen

Zusätzlich zur Roboterprogrammierung ist für den in dieser Arbeit vorgestellten Programmieransatz eine *graphische Benutzungsoberfläche (GUI)* Bestandteil des Konzepts. Deshalb wird im nächsten Abschnitt das Feld der graphischen Benutzungsoberflächen für Roboter-Programmiersysteme betrachtet. Anschließend wird näher auf Videoschnitt Programme eingegangen, welche Parallelen zu Roboter Programmiersystemen aufweisen.

2.2.1 GUI zur Roboterprogrammierung

In den letzten Jahren stieg aufgrund der großen Anzahl neu entwickelter kollaborativer Roboter auch die Zahl der Programmiersysteme für diese. Die meisten Programmiersysteme setzen dabei auf graphische Benutzungsoberflächen, welche auf Bildschirmen angezeigt werden, die entweder mit Maus und Tastatur oder mit Toucheingabe gesteuert werden. Unterteilt werden können die GUI zur Roboterprogrammierung zum einen in die Programmiersysteme, welche die Roboterhersteller zur Verfügung stellen, und zum anderen in die GUI, welche im Rahmen von Forschungsprojekten entwickelt wurden.

Herstellereigene Programmierumgebungen speziell für deren Robotersysteme wurden in den letzten Jahren immer präsenter. Anders als bei der Roboter-zentrierten Programmierung setzen die Hersteller dabei nicht mehr auf textuelle Programmeingaben, sondern meist auf eine Art Icon-basierter Programmierung bei der vordefinierte Programmblöcke aneinandergereiht und parametriert werden müssen. Beispiele dafür sind die Programmieroberfläche von Universal Robots [UR] oder die Programmieroberfläche von Franka Emika [Franka]. Auch gibt es immer mehr Firmen, welche Software zum einfachen Programmieren von Robotern bereitstellen. Beispiele dafür sind Artiminds [Artiminds], Drag&Bot [DragAndBot] oder RoboDK [RoboDK]. Alle Benutzungsoberflächen der Roboterhersteller haben gemeinsam, dass sie nicht für Mehrrobotersysteme geeignet sind. Weiterhin haben sowohl die Benutzungsoberflächen der Roboterhersteller, als auch die der weiteren Firmen gemeinsam, dass diese die Roboterprogramme nicht entlang einer Zeitleiste anzeigen und so den zeitlichen Aspekt eines Roboterprogramms nicht darstellen können.

Bei den GUI, welche aus Forschungsprojekten entstanden, sind gibt es verschiedene Herangehensweisen zur Erstellung eines Programmiersystems. Die meisten dieser GUI setzen auf den Visual Programming Ansatz [Coronado2020]. Darunter ist zu verstehen, dass die Programmierung der Roboter graphisch ohne textuellen Programmcode erfolgt. Hierfür gibt es verschiedene Ansätze. Ein Ansatz ist, Programmblöcke mit fest vorgegebenen Aktionen aneinander zu reihen um ein Roboterprogramm zu erstellen (z.B. [Chung2020], [Huang2017], [Rahul2014], [Kim2007]). Andere Arbeiten hingegen verwenden eine Art Ablaufdiagramm um Sequenzen von auszuführenden Aktionen inklusive Kontrollfluss zu modellieren (z.B. [Alexandrova2015], [Charntaweekhun2006],

Abbildung 2.4: Bildschirmaufnahme des OpenShot Video Editors [OpenShot].

[Cross2013], [Arai1997]). Eine dritte Möglichkeit, welche die Visual Programming Ansätze für die Definition von Roboterprogrammen erlauben, ist die Eingabe als Zustandsautomaten (z.B. [Datta2012], [Cox1998]). Alle Visual Programming Ansätze haben gemeinsam, dass sie aus einer Kontrollleiste bestehen, über welche die einzelnen Bausteine der Roboterprogramme ausgewählt und über einen Programmierbereich, in dem die Bausteine zu einem Programm verbunden werden.

Dadurch, dass für eine intuitive Bedienung eines Systems eine gewisse Vertrautheit des Aufbaus und der Funktionen für die Benutzer:innen notwendig sind, werden im nächsten Abschnitt Videoschnitt-Programme genauer betrachtet, weil Endanwender:innen häufig bereits Erfahrung mit dem Thema Videoschnitt - zum Beispiel in Form von Urlaubsvideos - haben.

2.2.2 GUI von Videoschnitt-Programmen

Videoschnitt-Programme existieren bereits deutlich länger als graphische Benutzungsoberflächen für Roboterprogrammiersysteme und sind in einer breiten Form kommerziell erhältlich. Dadurch, dass die Zielgruppe der Videoschnitt-Programme häufig Endanwender:innen ohne Expertenwissen sind, wird der Aufbau und die Funktionalität der Programme so weiterentwickelt, dass diese möglichst intuitiv verwendet werden können. Aus diesem Grund werden in diesem Abschnitt kommerzielle Programme in Form von closed-source und open-source Videoschnitt-Programme betrachtet. Zusätzlich werden Beispiele für Prototypen aus der Forschung präsentiert. Aus den Programmen werden Gemeinsamkeiten extrahiert und Möglichkeiten zur Anwendung in der Roboterprogrammierung dargelegt.

Sowohl kommerzielle Videoschnitt-Programme, als auch Prototypen aus der Forschung besitzen in der Regel einen ähnlichen Aufbau. Dieser ist in drei Teile gegliedert. Zum einen gibt es einen Bereich mit Zeitleisten, in dem die Video- und Tonspuren dargestellt werden. Des Weiteren gibt es einen Bereich, in dem ein Vorschaufenster den aktuellen Stand des Videos zeigt, sowie eine Kontrollleiste, in der die Funktionalitäten des jeweiligen Programms ausgewählt werden können. Ein Beispiel dafür ist in Abbildung 2.4 dargestellt. Beispiele für closed-source Programme sind Adobe Premiere [Adobe] oder Magix Video Deluxe [Magix]. Bei den open-source Programmen sind OpenShot [OpenShot] und Shotcut [Shotcut] als weit verbreitete Beispiele zu nennen. Im Bereich der Forschung gibt es dadurch, dass Videoschnittprogramme schon lange kommerziell erhältlich sind, relativ wenig neue Entwicklungen. Beispiele für Videoschnitt-Programme aus der Forschung sind in [Casares2002] und [Meng1997] zu finden.

Der allgemeine Aufbau von Videoschnitt-Programmen kann in abgewandelter Form auch für GUI von kinästhetischen Roboterprogrammiersystemen verwendet werden. Dabei können die Erkenntnisse, welche aus Abschnitt 2.2.1 zu Visual Programming Oberflächen gewonnen wurden mit denen aus diesem Abschnitt kombiniert werden. Die Kontrollleiste soll dabei weiterhin die Funktionalitäten des Programmiersystems beinhalten. Das Vorschaufenster aus Videoschnitt-Programmen wird durch ein Simulationsfenster ersetzt, welches die einzelnen Roboter mit ihren Programmen und die Umgebung der Roboter in einer 3D-Darstellung anzeigen kann. Aus den Zeitleisten mit den Video- und Tonspuren wird der Bereich, in dem die kinästhetisch demonstrierten Roboterprogramme dargestellt werden. Die Roboterprogramme sollen ähnlich wie Ablaufdiagramme, die für das Visual Programming benutzt werden, dargestellt werden. Zusätzlich soll der zeitliche Aspekt in Form von Zeitleisten aus den Videoschnitt-Programmen weiterverwendet werden. Diese neuartige Darstellungsform für kinästhetisch demonstrierte Roboterprogramme wird in Kapitel 4 ausgearbeitet.

2.3 Intuitive Bedienbarkeit

In diesem Abschnitt das Thema *Intuitivität* genauer betrachtet. Dafür wird zunächst der Begriff intuitive Bedienbarkeit definiert und anschließend verschiedenen Möglichkeiten der Evaluation derselbigen präsentiert.

2.3.1 Definition intuitiver Bedienbarkeit

Die IUUI Research Group hat sich mit dem Thema *intuitive Bedienbarkeit* beschäftigt und eine Definition dafür erarbeitet. Diese lautet folgendermaßen:

> „Ein technisches System ist im Rahmen des spezifischen Kontextes einer Aufgabenstellung in dem Maße intuitiv benutzbar, in dem der jeweilige Benutzer mit ihm durch unbewusste Anwendung von Vorwissen effektiv interagieren kann.“ [Mohs2006a]

Mit dieser Definition ist erkennbar, dass intuitive Bedienbarkeit nur für den Kontext der Aufgabenstellung vorliegen kann. Es ist wichtig, dass die Benutzer:innen effektiv mit dem System interagieren und dabei unbewusst ihr Vorwissen verwenden. Nicht das Produkt selbst ist intuitiv, sondern der Prozess der Bedienung [Mohs2006b]. Soll die Bedienbarkeit eines Produkts intuitiv gestaltet werden, so muss auf Vorwissen zurückgegriffen werden, welches von möglichst vielen Menschen geteilt wird [Mohs2006c].

Ein weiterer Begriff, der in diesem Kontext wichtig ist, ist die *Gebrauchstauglichkeit* eines Produkts. Diese wird in der Norm EN ISO 9241 Teil 11 [ISO9241-11] definiert als eine Kombination aus Effektivität, Effizienz und Zufriedenheit. Weitere wichtige Teile zur Gestaltung gebrauchstauglicher Prozesse sind Teil 13 [ISO9241-13] (Benutzerführung), Teil 110 [ISO9241-110] (Grundsätze der Dialoggestaltung) und Teil 210 [ISO9241-210] (Prozess zur Gestaltung gebrauchstauglicher interaktiver Systeme).

Nach [Wegerich2012] stellt die intuitive Bedienbarkeit ein Teilkonzept der Gebrauchstauglichkeit aus EN ISO 9241 dar. In EN ISO 9241 wird unter Effizienz die motorische Effizienz, also zum Beispiel die Anzahl Klicks zum Erreichen eines Ziels verstanden. Bei der intuitiven Bedienbarkeit hingegen ist es so, dass die mentale Beanspruchung in den Vordergrund rückt und so anstelle der motorischen Effizienz die mentale Effizienz wichtig ist [Schmitt2014]. Zusätzlich wird bei der intuitiven Bedienbarkeit auf Vorwissen der Benutzer:innen gesetzt, was bei der Gebrauchstauglichkeit nicht der Fall ist [Wegerich2012].

Mit diesen Erkenntnissen setzt sich die Intuitivität demnach zusammen aus den drei Aspekten Effektivität, mentale Effizienz und Zufriedenheit. Nach [ISO9241-11] ist die *Effektivität* definiert als die Vollständigkeit und Genauigkeit, mit der Benutzer:innen die Ziele einer Aufgabe erreichen. Die *mentale Effizienz* ist definiert als die subjektiv empfundene Beanspruchung der Benutzer:innen beim Lösen einer Aufgabe [Schmitt2014]. Abschließend ist die *Zufriedenheit* definiert als subjektive Konsequenzen intuitiver Bedienung [Schmitt2014].

Um eine Aussage über die intuitive Bedienbarkeit eines Systems treffen zu können, müssen demnach diese drei Aspekte gemessen werden. Der nächste Abschnitt beschäftigt sich mit Werkzeugen, welche verwendet werden können, um die Intuitivität zu evaluieren.

Genauigkeit:		Vollständigkeit:	
3	Der Nutzer hat das Programmiersystem ohne Probleme genutzt (bewusst; gezielt)	2	Die Aufgabe wurde vollständig erfüllt
2	Der Nutzer hat das Programmiersystem mit Trial & Error genutzt (auf gut Glück; teilweise unbewusst)	1	Die Aufgabe wurde teilweise erfüllt
1	Der Nutzer hat das Programmiersystem mit einem einzigen Hinweis des Moderators genutzt	0	Die Aufgabe wurde nicht erfüllt
0	Der Nutzer hat das Programmiersystem mit permanenter Unterstützung des Moderators genutzt		

Abbildung 2.5: Bewertungskriterien zur Messung der Effektivität als Kombination aus Vollständigkeit und Genauigkeit.

2.3.2 Evaluation intuitiver Bedienbarkeit

Zur Messung der intuitiven Bedienbarkeit eines Systems wird in [Wegerich2012] eine Toolbox vorgestellt. Diese wird in [Orendt2017] verwendet und speziell auf die Roboterprogrammierung angepasst. Beide Ansätze evaluieren die intuitive Bedienbarkeit, indem die drei Aspekte Effektivität, mentale Effizienz und Zufriedenheit gemessen werden. In [Orendt2017] wird zusätzlich noch die Komplexität des Systems gemessen. Dafür werden in [Wegerich2012] und [Orendt2017] verschiedene Messinstrumente zur Verfügung gestellt, auf die im Folgenden näher eingegangen wird.

Die Effektivität setzt sich zusammensetzt aus der Genauigkeit und der Vollständigkeit der Bearbeitung einer Aufgabe. Hierfür werden in [Wegerich2012] Bewertungskriterien veröffentlicht, wobei die Genauigkeit mit Werten zwischen 0 und 3 Punkten und die Vollständigkeit mit Werten zwischen 0 und 2 Punkten bewertet wird. Bei der Genauigkeit bedeuten 0 Punkte, dass die Aufgabe mit permanenter Unterstützung des Moderators bearbeitet und 3 Punkte, dass die Aufgabe ohne Probleme gelöst wurde. Für die Vollständigkeit bedeuten 0 Punkte, dass die Aufgabe nicht erfüllt wurde und 2 Punkte, dass diese vollständig erfüllt wurde. Die Bewertungskriterien für die Effektivität sind in Abbildung 2.5 dargestellt.

Zur Auswertung der Effektivität wird je Aufgabe und Teilnehmer:in die Summe der Punkte aus Genauigkeit und Vollständigkeit berechnet. Anschließend wird dieser Wert in den prozentualen Wert umgerechnet, wobei 5 Punkte einer Effektivität von 100% entsprechen. Pro Aufgabe kann so aus allen Teilnehmer:innen der Mittelwert der Effektivität berechnet werden und sowohl für die Einschätzung der Effektivität des Produkts als auch zum Vergleich mehrerer Produkte verwendet werden [Schmitt2014].

Für die Messung der mentalen Effizienz in Form der mentalen Beanspruchung wird in [Wegerich2012] die Verwendung der Skala zur Erfassung von subjektiv erlebter Anstrengung (SEA) [Eilers1986] vorgeschlagen (Abbildung 2.6a). Hierbei bewerten die Teilnehmer:innen selbst auf einer Skala von 0 bis 220, wie anstrengend sie die Be-

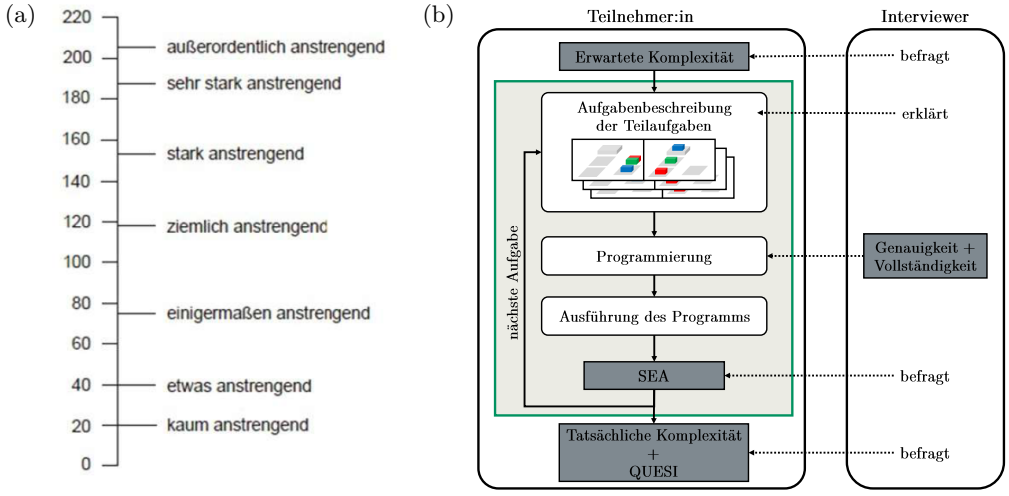


Abbildung 2.6: Abbildung (a) zeigt die SEA-Skala zur Messung der mentalen Effizienz in Form von mentaler Beanspruchung der Teilnehmer:innen. Abbildung (b) zeigt den Ablauf einer Evaluation hinsichtlich intuitiver Bedienung von Roboterprogrammiersystemen nach [Orendt2017].

dienung des Produkts empfunden haben. Die Skala ist an mehreren Stellen mit einer textuellen Beschreibung annotiert. Diese soll die Teilnehmer:innen dabei unterstützen, den passenden Wert für die empfundene mentale Beanspruchung zu finden. Ein Wert von 20 ist beispielsweise mit „kaum anstrengend“, ein Wert von 208 mit „außerordentlich anstrengend“ annotiert.

Zur Auswertung der mentalen Effizienz wird je Aufgabe der Mittelwert aus den Bewertungen aller Teilnehmer:innen gebildet. So kann festgestellt werden, welche Aufgaben am anstrengendsten zu bearbeiten waren. Zusätzlich kann aus den Mittelwerten jeder Aufgabe ein Gesamtwert für die mentale Effizienz des Produkts berechnet werden. Dieser Wert kann anschließend dafür verwendet werden um das Produkt selbst einzuschätzen, aber auch um es mit anderen Produkten zu vergleichen [Schmitt2014].

Die Zufriedenheit wird in [Wegerich2012] über die subjektiven Konsequenzen intuitiver Benutzbarkeit mit dem QUESI-Fragebogen [Naumann2010] gemessen. Der Fragebogen besteht aus 14 positiv formulierten Fragen in 5 Kategorien. Alle Fragen werden auf der fünfstufigen Likertskala mit Werten von 1 („trifft gar nicht zu“) bis 5 („trifft völlig zu“) bewertet. Höhere Werte entsprechen dabei einer höheren Zustimmung und damit einer höheren Zufriedenheit mit dem System. Der Fragebogen ist in Frage 6 des Fragebogens F3 im Anhang abgebildet. Die einzelnen Kategorien der Fragen sind:

- K: Wahrgenommene Kognitive Beanspruchung (Fragen 1, 6, 11)
- Z: Wahrgenommene Zielerreichung (Fragen 2, 7, 12)
- L: Wahrgenommener Lernaufwand (Fragen 3, 8, 13)

- V: Vertrautheit/Vorwissen (Fragen 4, 9, 14)
- Fe: Wahrgenommene Fehlerrate (Fragen 5, 10)

Zur Auswertung der Zufriedenheit wird der Mittelwert aller Antworten zu den Fragen einer Kategorie gebildet. Dadurch erhält man den Wert für diese Kategorie. Der Mittelwert über alle Kategorien ergibt den QUESI-Wert. Dieser Wert kann nicht für sich allein stehend interpretiert werden, sondern muss mit den Werten anderer Systeme verglichen werden [Wegerich2012]. Mit Hilfe der Kategorien kann jedoch herausgefunden werden, in welchen Kategorien das System noch verbessert werden muss.

Zusätzlich zu den Werkzeugen zur Messung der drei Aspekte intuitiver Bedienung wurde in [Orendt2017] noch der Aspekt der erwarteten und tatsächlichen Komplexität als Indiz für die Intuitivität eines Systems eingeführt. Hierfür wird vor der Benutzungsstudie die erwartete Komplexität auf einer Skala von -5 („sehr kompliziert“) bis $+5$ („sehr leicht“) erhoben. Auf der gleichen Skala wird nach der Benutzungsstudie die tatsächlich empfundene Komplexität ermittelt. Damit kann über die Differenz aus beiden Werten eine Aussage darüber getroffen werden, ob die Teilnehmer:innen das System als leichter oder schwieriger zu Bedienen eingeschätzt haben als es tatsächlich ist.

Weiterhin wurde im Toolkit in [Orendt2017] ein Prozess entworfen, um Roboterprogrammierkonzepte hinsichtlich intuitiver Bedienung evaluieren zu können. Dieser Prozess ist in Abbildung 2.6b dargestellt. Dabei ist das Vorgehen wie folgt: Vor Beginn der Benutzungsstudie wird die erwartete Komplexität abgefragt. Anschließend erklärt der Interviewer den Teilnehmer:innen die Aufgabenbeschreibung. Diese wird danach von diesen am Robotersystem programmiert, während der Interviewer die Werte für Vollständigkeit und Genauigkeit erhebt. Nach dem Programmieren geben die Teilnehmer:innen auf der SEA-Skala an, wie hoch ihre mentale Beanspruchung bei der Programmierung war. Anschließend wird mit der nächsten Aufgabe weitergemacht, welche der Interviewer den Teilnehmer:innen wieder erklärt. Sind alle Aufgaben der Benutzungsstudie erledigt müssen die Teilnehmer:innen noch den QUESI-Fragebogen ausfüllen und die tatsächlich empfundene Komplexität des Programmiersystems angeben.

2.4 Schlussfolgerung

Die Auswertung des Stands der Forschung hat gezeigt, dass es einfach zu bedienende Programmiersysteme für Roboter gibt. Einige davon verwenden das kinästhetische Führen, welches in der Literatur als intuitiv angesehen wird. Meist sind diese Programmiersysteme jedoch auf einen Roboter beschränkt und unterstützen keine Mehrrobotersysteme. Für die Benutzungsoberfläche bietet sich ein Aufbau ähnlich eines Videoschnittprogramms an, da Videoschnittprogramme auch im privaten Bereich häufig verwendet werden und so der Aufbau und die Funktionalität des Programms vertraut ist. Die Gestaltung der Benutzungsoberfläche wurde nach allgemein anerkannten Maßstäben aus Design-Richtlinien durchgeführt.

Um abschließend eine Aussage zur intuitiven Bedienbarkeit des Programmiersystems treffen zu können wurde in der Literatur nach Möglichkeiten der Evaluation der intuitiven Bedienung gesucht. Dabei wurden mit der ibis-Toolbox aus [Wegerich2012] und der Erweiterung auf Roboterprogrammiersysteme aus [Orendt2017] passende Werkzeuge gefunden. Diese werden verwendet, um eine Aussage über die intuitive Bedienbarkeit des im Laufe dieser Arbeit entwickelten neuartigen Programmierkonzeptes treffen zu können.

Das nächste Kapitel beschäftigt sich mit dem Grundkonzept, welches für das Programmierkonzept verwendet wird. Dabei werden verschiedene Begriffe der kinästhetischen Roboterprogrammierung definiert, die Aufteilung des Systems in Programmier- und Ausführungsphase erläutert und der Aufbau und die Architektur des Programmiersystems vorgestellt.

KAPITEL 3

Grundkonzept

Inhalt

Bevor in den folgenden Kapiteln auf die einzelnen Konzepte des kinästhetischen Programmiersystems eingegangen wird, gibt es in diesem Kapitel einen Überblick über das Grundkonzept des Programmiersystems. Auf diese Basis bauen im weiteren Verlauf alle Konzepte auf.

Hierfür wird zunächst in Abschnitt 3.1 die kinästhetische Programmierung genauer beleuchtet und notwendige Begriffe in diesem Zusammenhang definiert. Anschließend wird in Abschnitt 3.2 die Trennung zwischen Programmierphase und Ausführungsphase im entwickelten Programmiersystem erläutert. Abschließend wird in Abschnitt 3.3 der Aufbau der graphischen Benutzungsoberfläche mit Hilfe von Videoschnittprogrammen aufgezeigt.

3.1 Kinästhetische Programmierung

Für das intuitive Programmiersystem wird in dieser Arbeit die kinästhetische Programmierung als grundlegendes Programmiersystem verwendet, da im Stand der Forschung festgestellt wurde, dass diese Art der Programmeingabe im Allgemeinen als intuitiv zu bezeichnen ist [Villani2018]. Die kinästhetische Programmierung umfasst

(a) Playback (b) Teach-In

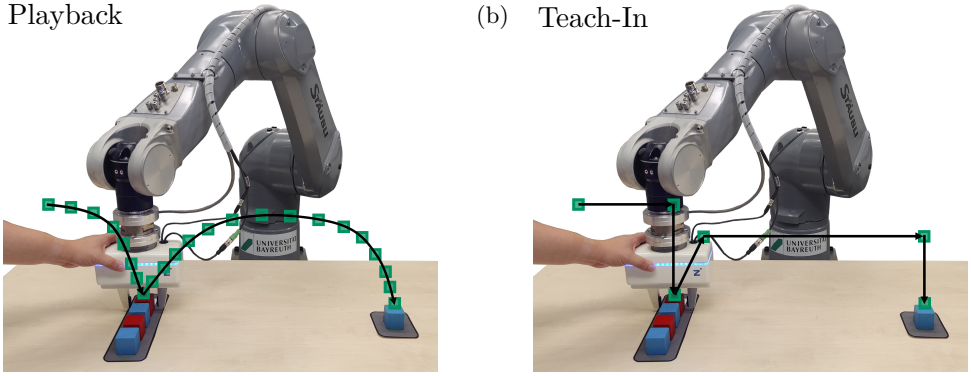


Abbildung 3.1: Vergleich beider in dieser Arbeit verwendeten Arten der kinästhetischen Programmierung: (a) Bei der Playback Programmierung wird in isochronen Zeitabständen die Konfigurationen (grün) gespeichert, woraus eine Trajektorie (schwarz) entsteht. (b) Bei der Teach-In Programmierung müssen die Konfigurationen (grün) explizit festgelegt werden. Zwischen diesen verfährt der Roboter mit einer Linearbahn im Arbeitsraum (schwarz).

sowohl die Playback Programmierung als auch die Teach-In Programmierung. Die Benutzer:innen können zur Laufzeit des Programmiersystems entscheiden, ob für die zu programmierende Aufgabe der Playback- oder der Teach-In-Ansatz verwendet werden soll. Da in der Literatur die Definitionen von Playback bzw. Teach-In Programmierung nicht einheitlich sind, wird im Folgenden definiert, was im Rahmen dieser Arbeit unter Playback Programmierung (Abschnitt 3.1.1) und was unter Teach-In-Programmierung (Abschnitt 3.1.2) zu verstehen ist.

Sowohl bei der Playback, als auch bei der Teach-In Programmierung erhält man als Ergebnis nach dem Programmieren eine Trajektorie T als Sequenz von $n \in \mathbb{N}^+$ Gelenkkonfigurationen q_t mit $t \in \{0, 1, \ldots, n-1\}$ und $q_t \in \mathbb{R}^d$ für Roboter mit $d \in \mathbb{N}^+$ Achsen. Zusätzlich wird für die Trajektorie zwischen je zwei Konfigurationen q_t und q_{t+1} eine Zeitdauer definiert, welche die benötigte Verfahrzeit des Roboters von q_t nach q_{t+1} angibt. Bei der Playback Programmierung, wie sie in dieser Arbeit verwendet wird, ist die Zeitdauer zwischen allen Paaren aus Gelenkkonfigurationen q_t und q_{t+1} isochron, d.h. immer eine feste Anzahl an Millisekunden. Im Gegensatz dazu können die Zeitdauern bei der Teach-In Programmierung variieren, je nachdem wie die Konfigurationen und Zeiten von den Benutzer:innen festgelegt werden. Im weiteren Verlauf dieser Arbeit werden die entwickelten Konzepte in der Regel für die Playback Programmierung vorgestellt. Diese sind jedoch problemlos auf die Teach-In Programmierung anwendbar, da alle Konzepte auf Trajektorie-Basis entwickelt werden.

3.1.1 Playback Programmierung

In der Literatur wird die Playback Programmierung auch als Walk-Through Programmierung bezeichnet [Villani2018]. Sie besteht aus einer Programmierphase und einer Ausführungsphase. In der Programmierphase programmieren die Benutzer:innen den Roboter, indem sie diesen kinästhetisch führen. Dabei geben die Benutzer:innen durch das Führen des Roboters und das Ansteuern des montierten Werkzeuges die Trajektorie T vor. Parallel zum kinästhetischen Führen zeichnet das verwendete Playback-Programmiersystem in isochronen Abständen die jeweils aktuellen Gelenkkonfigurationen q_c des Roboters auf und erstellt so sukzessiv die Trajektorie T. Dies ist schematisch in Abbildung 3.1a dargestellt

In der Ausführungsphase wird die vorher per kinästhetischem Führen definierte Trajektorie T vom Roboter wieder abgespielt. Wichtig dabei ist, dass der zeitliche Verlauf exakt so eingehalten wird, wie er in der Programmierphase aufgezeichnet wurde. Das sorgt dafür, dass der Roboter sich identisch zu der kinästhetisch geführten Trajektorie bewegt und so Aufgaben ausführen kann, bei denen die exakte Trajektorie benötigt wird. Darunter fallen Aufgaben wie zum Beispiel Schweißen, Kleben oder Lackieren. Auch ist es bei Bewegungen in einem Raum mit vielen Hindernissen sinnvoll, die exakte Trajektorie aufzuzeichnen, damit keine Kollisionen bei der Ausführung des Roboterprogramms entstehen.

3.1.2 Teach-In Programmierung

Die Teach-In Programmierung ist, so wie sie in dieser Arbeit verwendet wird, ebenso in eine Programmierphase und eine Ausführungsphase aufgeteilt. In der Literatur wird teilweise auch der Begriff Lead-Through Programmierung verwendet. Die Programmierung des Robotersystems findet in der Programmierphase ebenfalls durch kinästhetisches Führen der Benutzer:innen statt. Der Spezialfall hierbei ist aber, dass von den Benutzer:innen explizit angegeben werden muss, welche Gelenkkonfigurationen vom Programmiersystem zur Trajektorie hinzugefügt werden sollen. Dies funktioniert so, dass der Roboter zu einer der gewünschten Gelenkkonfigurationen geführt wird und anschließend über das Programmiersystem das Hinzufügen der aktuellen Konfiguration q_c zur Trajektorie T ausgelöst wird. Für die benötigte Zeit zwischen den einzelnen Konfigurationen wird nach Hinzufügen einer neuen Konfiguration entweder eine zeitoptimale Bewegung im Gelenkraum oder eine Linearbahn im Arbeitsraum von der letzten zur aktuellen Konfiguration berechnet. Alternativ können die Benutzer:innen eine Zeitdauer angeben, welche größer der berechneten minimalen Zeitdauer sein muss. Welche Art der Verbindungsbahn zwischen den Konfigurationen verwendet werden soll, wird vor der Programmierphase von den Benutzer:innen festgelegt. Dies ist schematisch in Abbildung 3.1b dargestellt.

In der Ausführungsphase wird die vorher durch kinästhetisches Führen definierte Trajektorie T vom Roboter wieder abgespielt. Dabei wird zwischen den einzelnen Konfigurationen - wie von den Benutzer:innen festgelegt - entweder mit einer zeitoptimalen Bewegung im Gelenkraum oder einer Linearbahn im Arbeitsraum verfahren. Dies sorgt dafür, dass nicht die von den Benutzer:innen demonstrierte Trajektorie wieder abgespielt wird, sondern nur die Bewegung des Roboters zwischen den manuell abgespeicherten Gelenkkonfigurationen. Hierfür eignen sich Aufgaben, bei denen kein exaktes abspielen einer demonstrierten Bewegung benötigen. Darunter Fallen einfache Montage- oder Pick-and-Place Aufgaben. Sollten jedoch komplexere Bewegungen zur Montage oder zum Pick-and-Place notwendig sein, so kann mit Hilfe der Playback Programmierung unter Umständen schneller ein passendes Roboterprogramm erzeugt werden.

Die Teach-In Programmierung, wie sie in dieser Arbeit verwendet wird, ist ein Spezialfall der Playback Programmierung mit variablen Zeiten zwischen den aufgezeichneten Punkten. Dadurch werden im Folgenden in dieser Arbeit die weiteren Konzepte auf Basis der Playback Programmierung als kinästhetische Programmierung erläutert.

3.2 Aufteilung in Programmier- und Ausführungsphase

Wie im letzten Abschnitt erwähnt, sind beide Arten der kinästhetischen Roboterprogrammierung aufgeteilt in Programmier- und Ausführungsphase. Dies ist in der graphischen Benutzungsoberfläche des Programmiersystems durch die strikte Trennung in Programmiermodus (Abschnitt 3.2.1 und Ausführungsmodus (Abschnitt 3.2.2) realisiert. Beide Phasen sind in Abbildung 3.2 beispielhaft dargestellt.

3.2.1 Programmierphase

In der Programmierphase ist bei der kinästhetischen Roboterprogrammierung der Roboter so eingestellt, dass er von den Benutzer:innen von Hand kinästhetisch geführt werden kann. Dies passiert in der Regel durch Greifen des Roboters am Flansch oder Werkzeug und anschließendem Führen entlang der gewünschten Trajektorie. Die Benutzer:innen können dadurch den Roboter mit Muskelkraft bewegen und so ein neues Roboterprogramm in Form einer Trajektorie T aufzeichnen. Ob dies per Playback oder Teach-In Programmierung passiert, ist den Benutzer:innen überlassen.

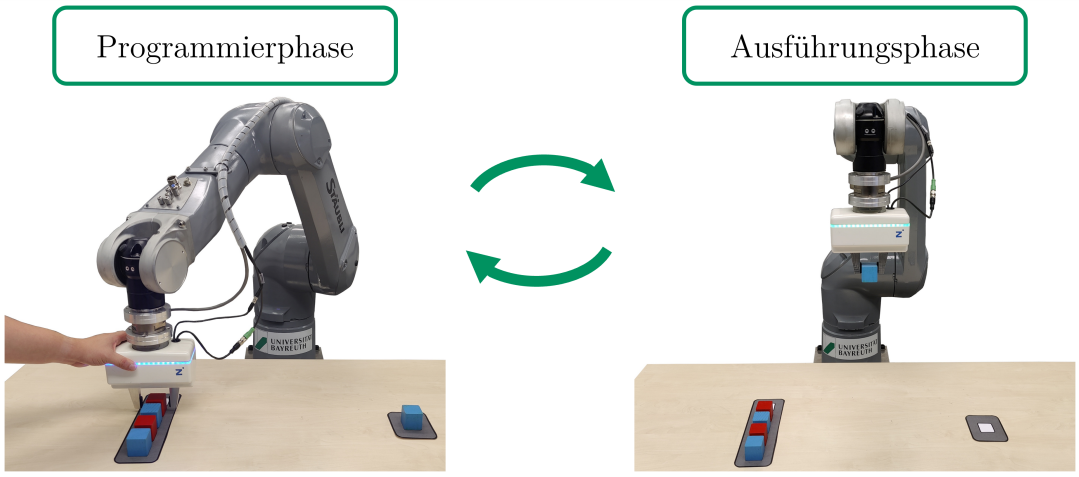


Abbildung 3.2: Unterschied zwischen Programmier- und Ausführungsphase. In der Programmierphase kann der Roboter kinästhetisch programmiert und anschließend dessen Programm graphisch bearbeitet werden. In der Ausführungsphase führt der Roboter das erstellte Roboterprogramm aus und darf dabei nicht mehr berührt werden. Von der Programmierphase kann erst in die Ausführungsphase gewechselt werden, wenn das erstellte Roboterprogramm valide ist.

Wichtig in der Programmierphase ist, dass der Roboter sich in einem Modus befindet, der für die Mensch-Roboter-Kollaboration zulässig ist. Durch die auftretenden Kräfte und Geschwindigkeiten darf es also nicht möglich sein, dass der Mensch durch den Roboter verletzt werden kann. Auch sollte sich der Roboter während der Programmierphase nur nach expliziter Bestätigung durch die Benutzer:innen bewegen, so dass für die Benutzer:innen keine Gefahr aufgrund von unvorhersehbaren Bewegungen besteht.

3.2.2 Ausführungsphase

In der Ausführungsphase kann der Roboter nicht mehr kinästhetisch geführt werden, da er sich wieder im Positionskontrollmodus befindet. Wird das Ausführen eines kinästhetisch demonstrierten Roboterprogramms gestartet, so führt der Roboter ohne Hilfe der Benutzer:innen das vorher aufgezeichnete Roboterprogramm in Form der Trajektorie T aus.

(a) (b)

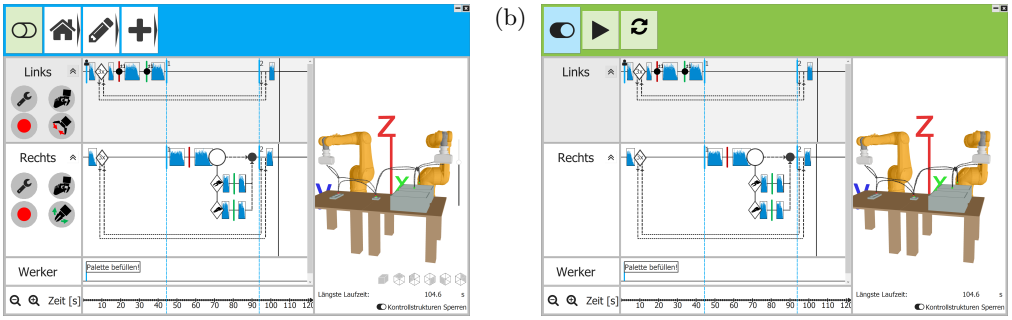


Abbildung 3.3: Die graphische Benutzungsoberfläche des Programmiersystems in beiden Modi. Abbildung (a) zeigt das Programmiersystem im Programmiermodus, Abbildung (b) im Ausführungsmodus. In beiden Modi ist der dreigeteilte Aufbau in Anlehnung an Videoschnittsoftware zu erkennen. Oben befindet sich die Kontrollleiste, unten links die Zeitleisten der einzelnen Roboter und unten rechts das Simulationsfenster. Der Aufbau und die Farbgestaltung der Oberfläche wurde mit Hilfe von Design-Richtlinien aus der Literatur entwickelt (Kapitel 4.2.1).

Um schnelle Ausführungen zu garantieren, kann die Bewegungsgeschwindigkeit des Roboters nach oben skaliert werden. Demnach dürfen sich bei der Ausführung keine Menschen in der Sicherheitszone des Roboters aufhalten, da sie unter Umständen verletzt werden könnten. Hierbei muss der Roboter nicht den Richtlinien der Mensch-Roboter-Kollaboration folgen.

Durch die strikte Trennung zwischen Programmier- und Ausführungsphase ergeben sich zwei Vorteile. Zum einen ist es möglich, ein Roboterprogramm durch Nicht-Expert:innen schnell umprogrammieren zu können, da in der Programmierphase auf die Playback bzw. Teach-In Programmierung gesetzt wird. Zum anderen erlaubt die Trennung aber auch, dass in der Ausführungsphase, in der sich die Benutzer:innen nicht im Gefahrenbereich des Roboters aufhalten dürfen, das Roboterprogramm beschleunigt abgespielt werden kann. Dies eliminiert einen Nachteil der Mensch-Roboter Kollaboration, nämlich die verhältnismäßig langsame Ausführungsgeschwindigkeit von Roboterprogrammen aufgrund von Sicherheitsbestimmungen. Das hat aber zur Folge, dass während der Ausführung eines kinästhetisch demonstrierten Roboterprogramms keine direkte Interaktion mit dem Menschen stattfinden kann.

3.3 Aufbau des Programmiersystems

Videoschnittprogramme gibt es sowohl für klassische Desktop-Systeme, als auch für Touch-Oberflächen wie Smartphones oder Tablet-PCs. Es hat sich gezeigt, dass der grundsätzliche Aufbau immer sehr ähnlich ist. Bei genauerer Betrachtung des Aufbaus von Videoschnittsoftware zeigt sich, dass die einzelnen Komponenten auf Bestandteile der kinästhetischen Roboterprogrammierung abgebildet werden können. Die in dieser Arbeit entwickelte Benutzungsoberfläche wurde für Touchscreens entworfen und mit Hilfe eines iterativen, Benutzer-zentrierten Design-Prozesses weiterentwickelt. Dieser Design-Prozess und die Weiterentwicklung wurden bereits in [Colceriu2020] veröffentlicht.

Videoschnittsoftware besteht, wie der Stand der Forschung gezeigt hat, in der Regel aus drei Komponenten. Einem Bereich, der entlang von Zeitleisten die einzelnen Video- bzw. Tonspuren anzeigt, eine Art Vorschaufenster, welches den aktuellen Stand des bearbeiteten Videos anzeigt und einen Kontrollbereich, welcher die Funktionen der Anwendung beinhaltet. Unter diesen Gesichtspunkten wurde das Design für die Benutzungsoberfläche des Programmiersystems entwickelt. Die graphische Benutzungsoberfläche, sowohl im Programmiermodus, als auch im Abspielmodus ist in Abbildung 3.3 dargestellt.

Der *Zeitleistenbereich* besteht in der entworfenen Benutzungsoberfläche aus *Zeitachsen* (Abbildung 3.3, unten links). In diesen Zeitachsen werden die jeweiligen Roboterprogramme ähnlich eines Ablaufdiagramms entlang einer Zeitleiste dargestellt. Dabei kommt eine skalierbare, geschichtete Darstellung zum Einsatz, welche in Kapitel 4 genauer erläutert wird. Diese Darstellung ermöglicht es, auch komplexe und tief geschachtelte Roboterprogramme übersichtlich auf begrenztem Raum darzustellen. Diese Zeitleistendarstellung kann mit Hilfe der Funktionen aus der Kontrollleiste editiert und mit Zusatzfunktionen erweitert werden, sodass beliebig geschachtelte Roboterprogramme entstehen können.

Der *Vorschaubereich* wird in der entworfenen Benutzungsoberfläche umgesetzt als *Simulationsfenster* (Abbildung 3.3, unten rechts). In diesem Simulationsfenster werden Modelle der am Programmiersystem angemeldeten Roboter und ihre geometrische Lage zueinander dargestellt. Zusätzlich besteht die Möglichkeit, 3D-Modelle der Umgebung zu laden. Weiterhin wird jedes in der Zeitleiste dargestellte Roboterprogramm in der Simulation als Kette von Würfeln dargestellt, welche die Konfigurationen der zugrundeliegenden Trajektorie repräsentieren. Wenn ein Bereich in der Zeitleistendarstellung durch die Benutzer:innen markiert wird, so wird dieser Bereich auch im Simulationsfenster hervorgehoben.

Der *Kontrollbereich* ist in der entworfenen Benutzungsoberfläche die *Kontrollleiste* (Abbildung 3.3, oben). Diese Leiste beinhaltet alle Funktionen, welche in den Kapiteln 5, 6 und 7 vorgestellt werden. Aufgerufene Funktionen der Kontrollleiste werden auf den jeweils in den Zeitleisten markierten Bereich oder Punkt eines Roboterprogramms ausgeführt. Mit Hilfe der Kontrollleiste kann zwischen Programmier- und Ausführungsphase gewechselt werden. Ein Wechsel in die Ausführungsphase ist nur möglich, wenn alle Roboterprogramme valide und damit für das Robotersystem ausführbar sind.

Da Videoschnittsoftware häufig auch im nichtprofessionellen Bereich beispielsweise für Urlaubsvideos verwendet wird und damit die Bedienung durch Nicht-Expert:innen erfordert, wird ein ähnlicher Aufbau im Rahmen dieser Arbeit verwendet. Dadurch soll eine intuitive Programmierung für kinästhetisch demonstrierte Roboterprogramme ermöglicht werden. Die konkrete Umsetzung des Gesamtdemonstrators des Programmiersystems wird in Kapitel 8 in Abschnitt 8.1 dargestellt. Dort werden auch die einzelnen Teile des Programmiersystems näher erläutert.

KAPITEL 4

Darstellen

Inhalt

Zentraler Bestandteil des Programmierkonzepts ist die Darstellung der kinästhetischen Roboterprogramme. Für ein intuitives Gesamtsystem ist es notwendig, die graphische Darstellung so zu gestalten, dass diese für Nicht-Expert:innen schnell und einfach verständlich ist. Um dies zu erreichen wird eine kontextfreie Grammatik in Form einer Backus-Naur-Form (BNF) entwickelt, welche die formale Grundlage für die kinästhetisch demonstrierten Roboterprogramme bildet. Zusätzlich wurde eine skalierbare Darstellung ähnlich Ablaufdiagrammen entlang einer Zeitleiste auf Basis der BNF entworfen. Mit Hilfe der BNF ist es auch möglich, die graphischen Roboterprogramme in textuelle Roboterprogramme zu übersetzen und umgekehrt.

In diesem Kapitel wird in Abschnitt 4.1 die BNF inklusive eines Beispielprogramms genauer erläutert. Anschließend wird in Abschnitt 4.2 die graphische Darstellung entlang einer Zeitleiste auf Basis der BNF erklärt und in Abschnitt 4.3 die Evaluation der Verständlichkeit der graphischen Darstellung vorgestellt.

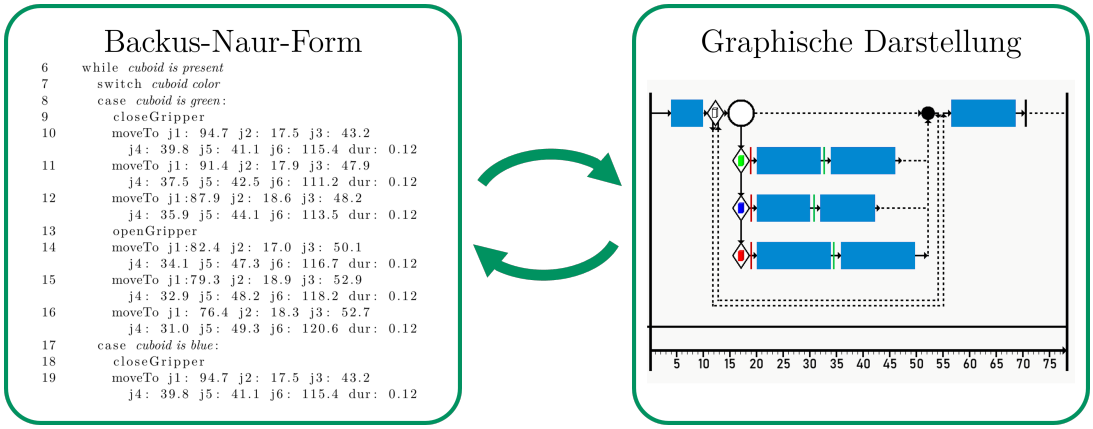


Abbildung 4.1: Schematische Darstellung der Dualität zwischen der Backus-Naur-Form und der graphischen Darstellung. Beide Darstellungsformen können verlustfrei ineinander umgewandelt werden. Die BNF links zeigt einen Ausschnitt des Roboterprogramms, welches rechts graphisch dargestellt ist.

In diesem Kapitel wird folgende Forschungsfrage behandelt:

F1 Wie lassen sich geschachtelte Roboterprogramme strukturiert und skalierbar graphisch entlang einer Zeitleiste darstellen?

Eine frühere Version der BNF und die zugehörige skalierbare graphische Darstellung inklusive Evaluation wurde in [Riedl2020a] veröffentlicht.

4.1 Backus-Naur-Form (BNF)

Als formale Grundlage der graphischen Darstellung kinästhetisch demonstrierter Roboterprogramme wird in dieser Arbeit zunächst eine kontextfreie Grammatik in Form einer BNF [McCracken2003] entworfen. Eine BNF als Grammatik wurde ausgewählt, da die Syntax vieler textueller Programmiersprachen mit Hilfe einer BNF oder erweiterten BNF definiert wird (z.B. Java [Møller] oder Python [Python]). Dadurch wird eine eigenständige textuelle Programmiersprache definiert, welche die kinästhetisch demonstrierten Roboterprogramme, die mit dem entwickelten Programmiersystem erstellbar und abspielbar sind, abbilden kann. Die Elemente der BNF können in eine zugehörige graphische Repräsentation übersetzt werden und umgekehrt, da beide Darstellungsformen für kinästhetisch demonstrierte Roboterprogramme äquivalent sind (Abbildung 4.1).

Diese Art der Dualität bietet den positiven Nebeneffekt, dass zusätzlich zu der graphischen Bearbeitung der kinästhetisch demonstrierten Roboterprogramme wenn gewünscht auch die textuelle Repräsentation bearbeitet werden kann. Expert:innen können dadurch Optimierungen am textuellen Quellcode vornehmen statt an der graphischen Repräsentation, wohingegen Nicht-Expert:innen weiterhin die Möglichkeit haben, alle Funktionen des Programmiersystems auch graphisch zu verwenden.

Dieser Abschnitt befasst sich zunächst in Abschnitt 4.1.1 mit dem formalen Aufbau der BNF. Anschließend wird in Abschnitt 4.1.2 genauer auf die einzelnen Elemente der BNF eingegangen, bevor in Abschnitt 4.1.3 ein Beispielprogramm, welches mit Hilfe der BNF erzeugt wurde, erläutert wird.

4.1.1 Formale Modellierung der BNF

Backus-Naur-Formen sind weit verbreitet um die Syntax einer Programmiersprache zu beschreiben. Bei der im Folgenden beschriebenen BNF wurde Wert darauf gelegt, dass alle Funktionen des Programmiersystems in der BNF abgebildet werden. Die einzelnen Elemente der BNF sollen anschließend übersetzt werden in die graphische Darstellung der kinästhetisch demonstrierten Roboterprogramme.

Auflistung 4.1 zeigt die Struktur der BNF welche im Rahmen dieser Arbeit entwickelt wurde und in einer früheren Version in [Riedl2020a] veröffentlicht wurde. In der früheren Version waren noch Elemente für Unterprogramme vorgesehen, welche letztendlich nicht realisiert wurden. Stattdessen wurden die Schleifen-Inkremente zu des aktuellen Version der BNF hinzugefügt. Nichtterminalsymbole sind in spitzen Klammern geschrieben (z.B. ⟨robot program⟩), Terminalsymbole als Klartext (z.B. `main`), und Kommentare sind kursiv geschrieben (z.B. *integer*). Kommentare werden im Folgenden genutzt, um die Grammatik abzukürzen, so dass die BNF auf das Wesentliche reduziert und lesbarer wird.

Auflistung 4.1: Backus-Naur-Form der kinästhetisch demonstrierten Roboterprogramme des in dieser Arbeit entwickelten Programmiersystems.

⟨robot program⟩ ⊨ ⟨main program⟩
⟨main program⟩ ⊨ `main` ⟨stmt block⟩ `endMain`
⟨stmt block⟩ ⊨ ⟨stmt⟩ | ⟨stmt⟩ ⟨stmt block⟩
⟨stmt⟩ ⊨ ⟨motion stmt⟩ | ⟨tool stmt⟩ | ⟨while stmt⟩ | ⟨do-while stmt⟩ | ⟨for stmt⟩ | ⟨switch stmt⟩ | ⟨sync stmt⟩

⟨motion stmt⟩ ⊨ `moveTo` ⟨conf⟩ | `moveTo` ⟨conf⟩ ⟨motion stmt⟩
⟨conf⟩ ⊨ ⟨increment conf⟩ | ⟨joint conf⟩ ⟨duration⟩
⟨increment conf⟩ ⊨ ⟨increment⟩ ⟨conf⟩

⟨increment⟩ ⊨ `increment x:` ⟨float⟩ `y:` ⟨float⟩ `z:` ⟨float⟩
`rx:` ⟨float⟩ `ry:` ⟨float⟩ `rz:` ⟨float⟩
⟨joint conf⟩ ⊨ ⟨conf 6D⟩ | ⟨conf 7D⟩
⟨conf 6D⟩ ⊨ `j1:` ⟨float⟩ `j2:` ⟨float⟩ `j3:` ⟨float⟩
`j4:` ⟨float⟩ `j5:` ⟨float⟩ `j6:` ⟨float⟩
⟨conf 7D⟩ ⊨ ⟨conf 6D⟩ `j7:` ⟨float⟩
⟨duration⟩ ⊨ `dur:` ⟨float⟩

⟨tool stmt⟩ ⊨ ⟨tool cmd⟩ | ⟨tool cmd⟩ ⟨tool stmt⟩
⟨tool cmd⟩ ⊨ ⟨gripper cmd⟩ | ⟨other tool cmd⟩
⟨gripper cmd⟩ ⊨ `openGripper` | `closeGripper`
⟨other tool cmd⟩ ⊨ *cmd to control other tools*

⟨while stmt⟩ ⊨ `while` ⟨expression⟩ `do` ⟨stmt block⟩ `endWhile`
⟨do-while stmt⟩ ⊨ `do` ⟨stmt block⟩ `while` ⟨expression⟩ `endDoWhile`
⟨for stmt⟩ ⊨ `for` ⟨positive int⟩ `times do` ⟨stmt block⟩ `endFor`

⟨switch stmt⟩ ⊨ `switch` ⟨cur value⟩ ⟨switch block⟩ `endSwitch`
⟨switch block⟩ ⊨ ⟨switch label⟩ ⟨stmt block⟩ |
⟨switch label⟩ ⟨stmt block⟩ ⟨switch block⟩
⟨switch label⟩ ⊨ `case` ⟨ref value⟩ `:` | `default :`

⟨sync stmt⟩ ⊨ ⟨sync point⟩ | ⟨sync interval⟩ |
⟨human point⟩ | ⟨human interval⟩
⟨sync point⟩ ⊨ `syncPoint id:` ⟨uuid⟩
⟨sync interval⟩ ⊨ `syncInterval id:` ⟨uuid⟩ ⟨stmt block⟩ `endSyncInterval`
⟨human point⟩ ⊨ `humanSyncPoint id:` ⟨uuid⟩ `description:` ⟨string⟩
⟨human interval⟩ ⊨ `humanSyncInterval id:` ⟨uuid⟩ `description:` ⟨string⟩
⟨stmt block⟩ `endHumanSyncInterval`

⟨expression⟩ ⊨ ⟨ref value⟩ `==` ⟨cur value⟩
⟨ref value⟩ ⊨ ⟨stimulus⟩
⟨cur value⟩ ⊨ ⟨stimulus⟩
⟨stimulus⟩ ⊨ *image* | *gripper value* | *other stimulus type*
⟨int⟩ ⊨ *integer*

⟨positive int⟩ ⊨ *integer > 0*
⟨float⟩ ⊨ *floating-point number*
⟨uuid⟩ ⊨ *universally unique identifier*
⟨char⟩ ⊨ ⟨letter⟩ | *space*
⟨letter⟩ ⊨ `A` | `B` | ... | `Z` | `a` | `b` | ... | `z`
⟨string⟩ ⊨ ⟨char⟩ | ⟨char⟩ ⟨string⟩

4.1.2 Elemente der BNF

Die im vorherigen Abschnitt dargestellte BNF bildet die Funktionen des Programmiersystems ab. Hauptelement der BNF ist das ⟨robot program⟩, welches zur Definition eines kinästhetisch demonstrierten Roboterprogramms verwendet wird. Ein Roboterprogramm besteht aus Elementen, welche den Kontrollfluss innerhalb des Programms modellieren. Diese werden in der BNF ⟨stmt⟩ Elemente genannt.

Die Basiselemente eines ⟨stmt⟩ Elements sind ⟨motion stmt⟩ und ⟨tool stmt⟩. Diese werden verwendet, um die Trajektorie des Roboters zu beschreiben und um diese um Werkzeugkommandos zu erweitern. Unter Werkzeugkommandos fallen zum Beispiel Befehle zum Öffnen oder Schließen des Greifers. Die ⟨motion stmt⟩ Elemente sind die `moveTo`-Befehle des Roboterprogramms, welche neben den anzufahrenden Gelenkwinkeln `j1` bis `j6` bzw. `j7` noch eine Zeitdauer `dur` in Sekunden angeben, welche der Roboter zum Anfahren der gegebenen Position benötigen darf. Das Editieren dieser Basiselemente wird in Kapitel 5 im Detail vorgestellt. Sollte ein ⟨motion stmt⟩ eine veränderbare Konfiguration beinhalten, also eine sogenannte Inkrement-Konfiguration (siehe Kapitel 6.4), so wird bei diesem zusätzlich zu den Gelenkwinkeln und der Zeitdauer noch die Veränderung `increment` angegeben. Dieses wird in Form von kartesischen Koordinaten und Eulerwinkeln definiert und wird nach jedem Schleifendurchlauf auf die zugehörige Konfiguration gerechnet.

Zusätzlich zu den Basiselementen werden die Elemente zum Modellieren des Kontrollflusses im ⟨stmt⟩ Element definiert. Darunter fallen unter anderem sensorbasierte Schleifen in Form von ⟨while stmt⟩ und ⟨do-while stmt⟩ Elementen um While und Do-While Schleifen realisieren zu können. Weiterhin werden ⟨switch stmt⟩ Elemente unterstützt, damit sensorbasierte Verzweigungen abgebildet werden können. Alle Bedingungen, welche für Schleifen und Verzweigungen genutzt werden, sind Sensorwerte, die durch das ⟨stimulus⟩ Element repräsentiert werden. Darunter fallen zum Beispiel Aufnahmen von Kameras oder Öffnungswinkel von Greifern. Ergänzend zu den sensorbasierten Kontrollstrukturen gibt es als Element noch eine Zählschleife, welche durch

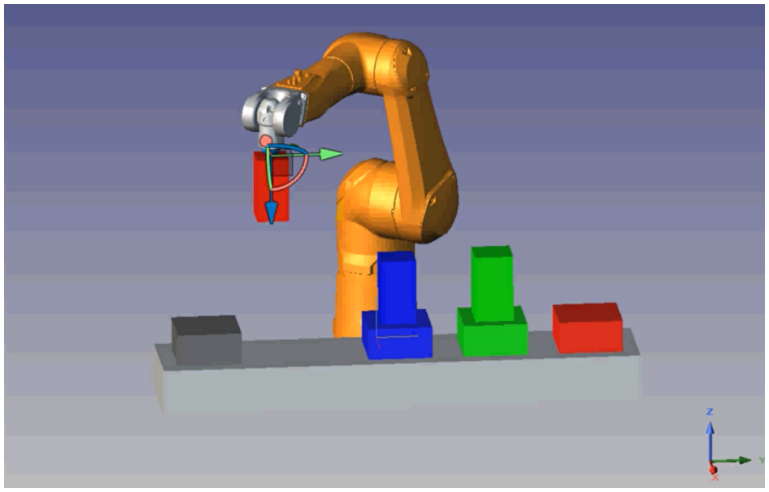


Abbildung 4.2: Bildschirmaufnahme aus der Stäubli Robotics Suite [SRS]. Der Roboter führt das Programm aus Auflistung 4.2 aus.

das ⟨for stmt⟩ repräsentiert wird. Diese hat die Funktion, dass in der Programmierphase angegeben werden kann, wie oft die Schleife während der Ausführungsphase durchlaufen werden soll. Sowohl die Schleifen, als auch die Verzweigungen, werden in Kapitel 6 im Detail behandelt.

Neben den Elementen für die Kontrollstrukturen sind auch Elemente zum Synchronisieren mehrerer Roboter und auch zum Synchronisieren von Roboter mit Menschen Teil des ⟨stmt⟩ Elements. Das allgemeine Element für Synchronisation ist das ⟨sync stmt⟩ Element. Dieses wird aufgeteilt in Elemente für Synchronisationspunkte (⟨sync point⟩) und Synchronisationsintervalle (⟨sync interval⟩) in der Roboter-Roboter Synchronisation und in Elemente für Synchronisationspunkte (⟨human point⟩) und Synchronisationsintervalle (⟨human interval⟩) in der Mensch-Roboter Synchronisation. Der Synchronisationsmechanismus wird in Kapitel 7 genauer betrachtet.

4.1.3 Beispielprogramm

Nachdem der formale Aufbau der BNF und deren Elemente erläutert wurden, wird in diesem Abschnitt ein verkürztes Beispielprogramm genauer betrachtet. Auflistung 4.2 zeigt ein Beispielprogramm, welches Quader der Farbe nach sortiert, so lange ein Quader an der Aufgreifposition vorhanden ist. Eine Bildschirmaufnahme aus der Simulation der Stäubli Robotics Suite [SRS], welches einen Roboter zeigt, der dieses Programm ausführt, ist in Abbildung 4.2 dargestellt. Die *kursiv* gedruckten Teile des Quellcodes wurde aufgrund der besseren Lesbarkeit vereinfacht und sind nicht Teil

der Syntax welche aus der BNF abgeleitet werden kann. Insbesondere bei den Bedingungen der `while` Schleife und der `switch` - `case` Verzweigung müssten Kamerabilder referenziert werden. Diese Kamerabilder wurden in der Auflistung durch die textuelle Umschreibung ersetzt.

Auflistung 4.2: Beilspielquellcode eines kinästhetisch demonstrierten Roboterprogramms, welches Quader der Farbe nach sortiert. Das Programm ist abgeleitet aus der BNF aus Listing 4.1.

```
1  main
2    moveTo j1: 18.5 j2: 16.5 j3: 78.4 j4: 12.3
       j5: 49.2 j6: 15.2 dur: 0.12
3    moveTo j1: 25.5 j2: 17.9 j3: 65.9 j4: 30.4
       j5: 47.0 j6: 71.6 dur: 0.12
4    moveTo j1: 53.9 j2: 17.0 j3: 59.1 j4: 36.2
       j5: 43.6 j6: 99.7 dur: 0.12
5    moveTo j1: 94.7 j2: 17.5 j3: 43.2 j4: 39.8
       j5: 41.1 j6: 115.4 dur: 0.12
6    while cuboid is present
7      switch cuboid color
8      case green:
9        closeGripper
10       moveTo j1: 94.7 j2: 17.5 j3: 43.2
           j4: 39.8 j5: 41.1 j6: 115.4 dur: 0.12
11       moveTo j1: 91.4 j2: 17.9 j3: 47.9
           j4: 37.5 j5: 42.5 j6: 111.2 dur: 0.12
12       moveTo j1:87.9 j2: 18.6 j3: 48.2
           j4: 35.9 j5: 44.1 j6: 113.5 dur: 0.12
13       openGripper
14       moveTo j1:82.4 j2: 17.0 j3: 50.1
           j4: 34.1 j5: 47.3 j6: 116.7 dur: 0.12
15       moveTo j1:79.3 j2: 18.9 j3: 52.9
           j4: 32.9 j5: 48.2 j6: 118.2 dur: 0.12
16       moveTo j1: 76.4 j2: 18.3 j3: 52.7
           j4: 31.0 j5: 49.3 j6: 120.6 dur: 0.12
17     case blue:
18       closeGripper
19       moveTo j1: 94.7 j2: 17.5 j3: 43.2
           j4: 39.8 j5: 41.1 j6: 115.4 dur: 0.12
20       moveTo j1: 95.2 j2:16.2 j3: 46.3
           j4: 38.2 j5: 46.2 j6: 133.0 dur: 0.12
21       openGripper
22       moveTo j1: 97.8 j2:18.3 j3: 42.1
           j4: 33.9 j5: 49.9 j6: 141.2 dur: 0.12
23       moveTo j1: 100.3 j2: 19.7 j3: 41.1
```

```
            j4: 32.4 j5: 53.4 j6: 152.8 dur: 0.12
24      case red:
25        closeGripper
26        moveTo j1: 94.7 j2: 17.5 j3: 43.2
            j4: 39.8 j5: 41.1 j6: 115.4 dur: 0.12
27        moveTo j1: 92.1 j2: 18.6 j3: 41.0
            j4: 35.6 j5: 43.6 j6: 132.8 dur: 0.12
28        moveTo j1: 90.5 j2: 20.9 j3: 39.9
            j4: 37.1 j5: 47.9 j6: 145.0 dur: 0.12
29        moveTo j1: 89.3 j2: 16.3 j3: 39.6
            j4: 39.7 j5: 52.0 j6: 149.2 dur: 0.12
30        openGripper
31        moveTo j1: 95.6 j2: 15.9 j3: 39.1
            j4: 43.8 j5: 58.1 j6: 159.3 dur: 0.12
32        moveTo j1: 96.8 j2: 12.4 j3: 37.6
            j4: 47.9 j5: 61.7 j6: 168.5 dur: 0.12
33        moveTo j1: 99.3 j2: 8.9 j3: 36.2
            j4: 45.9 j5: 67.4 j6: 174.2 dur: 0.12
34      endSwitch
35    endWhile
36    moveTo j1: 92.3 j2: 15.8 j3: 47.0 j4: 34.8
        j5: 46.3 j6: 133.7 dur: 0.12
37    moveTo j1: 55.5 j2: 17.1 j3: 62.4 j4: 28.1
        j5: 48.3 j6: 102.6 dur: 0.12
38    moveTo j1: 18.7 j2: 16.5 j3: 78.2 j4: 12.3
        j5: 49.2 j6: 45.2 dur: 0.12
39  endMain
```

Das dargestellt Beispielprogramm zum Sortieren von Quadern ist so aufgebaut, dass mit den `moveTo` Befehlen in den Zeilen `2-5` der Roboter von seiner Startposition zur Aufgreifposition verfahren wird. Anschließend wird vor jedem Schleifendurchlauf in Zeile `6` überprüft, ob noch ein Quader an der Aufgreifposition vorhanden ist.

Ist dies der Fall, so wird mit der Switch-Anweisung in Zeile `7` abhängig von der Farbe des Quaders ein `case` Block aufgerufen. Alle drei `case` Blöcke sind ähnlich aufgebaut. Zunächst wird der Quader gegriffen, dann wird der Roboter je nach Farbe des Quaders zu einer der Ablagepositionen verfahren, anschließend der Quader abgelegt und der Roboter zurück zur Aufgreifposition verfahren.

Liegt kein Quader mehr an der Aufgreifposition, so wird der Rumpf der `while` Schleife übersprungen und mit den `moveTo` Befehlen in den Zeilen `36-38` der Roboter zurück zur Startposition verfahren. Anschließend terminiert das Programm.

4.2 Skalierbare Graphische Darstellung entlang einer Zeitleiste

Zentraler Bestandteil der kinästhetischen Roboterprogrammierung ist die Darstellung der Roboterprogramme. Dadurch, dass die Zielgruppe des Programmiersystems Nicht-Expert:innen mit wenig bis keinen textuellen Programmierkenntnissen sind, muss die Darstellung strukturiert, verständlich und ohne Quellcode sein. Struktur erhält die graphische Darstellung dadurch, dass die BNF aus Abschnitt 4.1 als Grundlage dient. Diese Struktur und die Elemente der BNF werden in diesem Abschnitt in eine graphische Darstellung übersetzt.

Die leichte Verständlichkeit der graphischen Darstellung soll erreicht werden, indem diese in Anlehnung an Ablaufdiagramme entworfen wird. Die Symbole dafür werden in Anlehnung an ISO 5807 [ISO5807] erstellt. Dies sorgt dafür, dass sowohl der Kontrollfluss, als auch Synchronisation mit anderen Robotern oder Menschen und die Dauern der einzelnen Teile des Roboterprogramms aus der graphischen Darstellung ablesbar sind. Durch die schichtweise Anordnung der Roboterprogramme und der Möglichkeit Schachtelungen in den Programmen darzustellen, wird die Skalierbarkeit der Darstellung erreicht.

Zunächst wird in Abschnitt 4.2.1 auf allgemeine Design-Richtlinien für graphische Benutzungsoberflächen eingegangen, bevor in Abschnitt 4.2.2 die graphischen Darstellungen der einzelnen Elemente der BNF erarbeitet wird. Anschließend wird in Abschnitt 4.2.3 erläutert, auf welche Art und Weise geschachtelte Roboterprogramme dargestellt werden können, um abschließend in Abschnitt 4.2.4 die graphische Darstellung zum Beispielprogramm aus Auflistung 4.2 genauer zu betrachten.

4.2.1 Design-Richtlinien für GUI

Um bei der Erstellung der graphischen Benutzungsoberfläche diese speziell für Touchscreens anzupassen, wurden verschiedene Design-Richtlinien verwendet. Der grundsätzliche Aufbau des Programmiersystems sollte, wie im vorhergehenden Abschnitt beschrieben, ähnlich eines Videoschnitt-Programms sein. Dieser wurde für Touchscreens mit Hilfe des Android Developers Guide [AndroidDevelopers] und des Material Guide [MaterialDesign] entworfen.

Hierfür wurde ein Gitter-Layout aus dem Material Guide erstellt, welches je nach Auflösung, Seitenverhältnis und Pixeldichte auf allen Bildschirmen ein ähnliches Erscheinungsbild liefert und gleichzeitig eine Skalierbarkeit der Inhalte erlaubt. Dies kann erreicht werden, indem Zeilen und Spalten zum Gitter hinzugefügt oder entfernt werden können. Die Aufteilung der Benutzungsoberfläche mit Hilfe des Gitters ist in Abbildung 4.3 dargestellt, wobei eine Zeile für die Kontrollleiste zur Verfügung steht

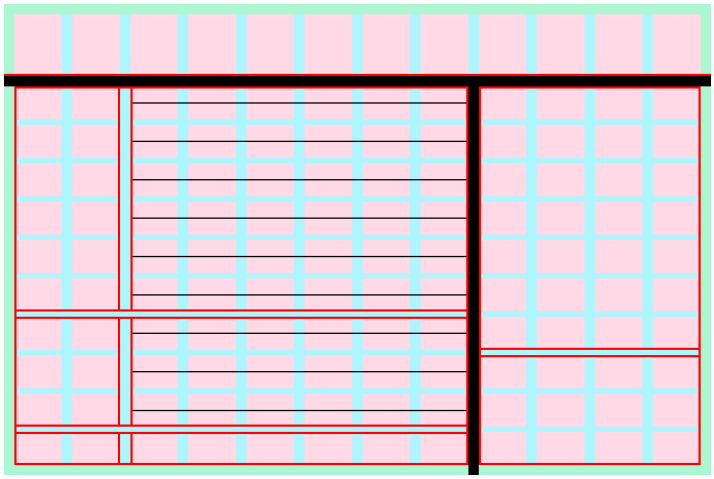


Abbildung 4.3: Gitter-Layout als Basis des Roboter-Programmiersystems auf Basis des Material Guide [MaterialDesign].

(Abbildung 4.3, oben). Die restliche Oberfläche wird in ein Untergitter aufgeteilt, welches initial acht Spalten und zehn Zeilen für die Zeitleisten der Roboter und deren Programme (Abbildung 4.3, unten links) und vier Spalten für das Simulationsfenster (Abbildung 4.3, unten rechts) zur Verfügung stellt.

Für die Bedienung des Programmiersystems, insbesondere der Zeitleisten und des Simulationsfensterswurden allgemein bekannten Touch-Gesten aus [Villamor2010] verwendet. So sorgt zum Beispiel eine Pinch-Geste sowohl in den Zeitleisten, als auch im Simulationsfenster dafür, dass der Ausschnitt vergrößert bzw. verkleinert wird.

Bei der Farbgebung der Benutzungsoberfläche wurde auf Farben aus dem Material Design zurückgegriffen. Dabei wurde mit Hilfe der Material Design Palette [MaterialPalette] jeweils ein Farbschema für den Programmiermodus und ein Farbschema für den Ausführungsmodus erstellt. Durch diese strikte farbliche Trennung zwischen Editieren und Abspielen ist für die Benutzer:innen schnell ersichtlich, in welchem Modus sich das Programmiersystem befindet und welche Funktionen verwendet werden können. Für den Programmiermodus wurde ein blaues Farbschema verwendet, wohingegen für den Ausführungsmodus ein grünes ausgewählt wurde.

Mit Hilfe der Design-Richtlinien wurde so eine Benutzungsoberfläche nach allgemein anerkannten Standards erstellt. Diese wird im weiteren Verlauf dieser Arbeit für die intuitive kinästhetische Roboterprogrammierung angepasst und mit den jeweiligen Funktionen erweitert.

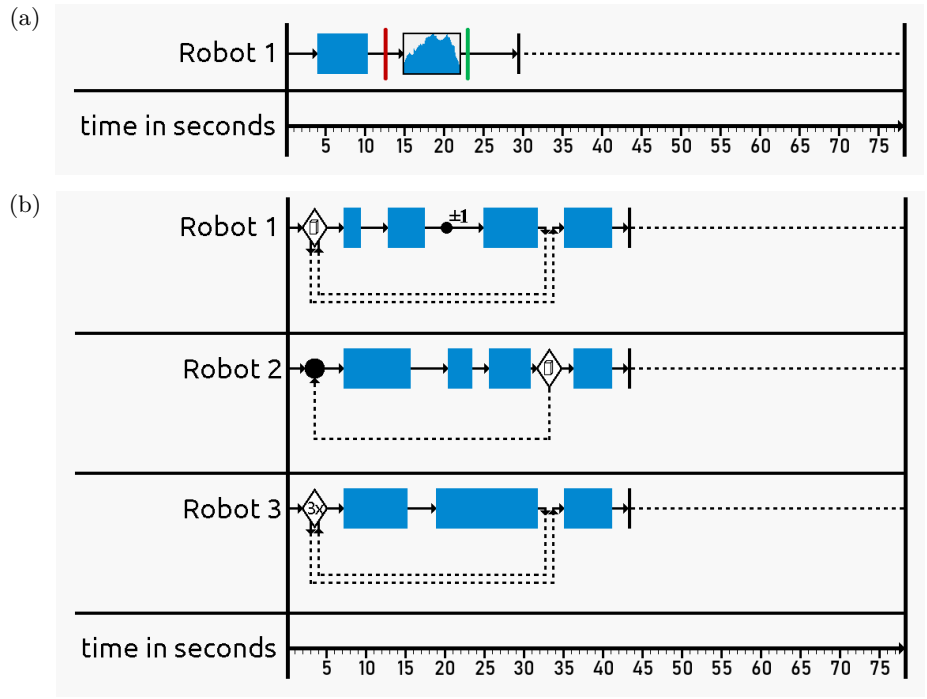


Abbildung 4.4: Graphische Darstellung von Bewegungsbefehlen, Werkzeugbefehlen und Schleifen. Abbildung (a) zeigt die graphischen Darstellungen für ⟨motion stmt⟩ und ⟨tool stmt⟩. Abbildung (b) zeigt die graphische Darstellung für ⟨while stmt⟩ (Roboter 1), ⟨do-while stmt⟩ (Roboter 2) und ⟨for stmt⟩ (Roboter 3).

4.2.2 Elemente der BNF graphisch darstellen

Die kinästhetisch demonstrierten Roboterprogramme werden je Roboter entlang einer Zeitleiste von links nach rechts dargestellt. Für das ⟨motion stmt⟩ Element der BNF gibt es zwei Darstellungsmöglichkeiten. Die *Blockdarstellung* und die *Graphdarstellung*. Bei der Blockdarstellung wird ein ein blaues Rechteck dargestellt, wenn sich der Roboter bewegt. Bei der Graphdarstellung wird anstelle des Blocks das Geschwindigkeitsprofil des Tool-Center-Point in blau dargestellt. Bewegt sich der Roboter nicht, wird bei beiden Darstellungsarten ein schwarzer Strich gezeichnet. Zusätzlich werden die Greiferbefehle des ⟨tool stmt⟩ Elements als rote vertikale Linie gezeichnet, wenn der Greifer geschlossen wird und als grüne vertikale Linie, wenn der Greifer geöffnet wird. Ein Beispiel dazu ist in Abbildung 4.4a dargestellt. Der Abschnitt zwischen Sekunde 5 und Sekunde 10 zeigt die Blockdarstellung, der Abschnitt zwischen Sekunde 15 und 22 die Graphdarstellung.

Sensorbasierte Schleifen aus den ⟨while stmt⟩ und ⟨do-while stmt⟩ Elementen der BNF werden ähnlich dargestellt. Die Schleifenbedingung, welche erfüllt werden muss um eine weitere Iteration der Schleife auszuführen, wird in einer Raute dargestellt. Je nachdem, ob es sich um eine While- oder eine Do-While-Schleife handelt wird die Bedingung am Anfang oder am Ende des Schleifenrumpfes dargestellt. Handelt es sich um eine While-Schleife, so wird die Bedingung an den Anfang der Schleife gestellt, handelt es sich um eine Do-While-Schleife, so wird sie an das Ende gestellt. Der Ort der Bedingung symbolisiert zusätzlich, wann diese ausgewertet wird. Ein schwarzer Kreis symbolisiert, dass an dieser Stelle im Roboterprogramm mehrere Kontrollflüsse zusammengeführt werden. Eine Zählschleife aus den ⟨for stmt⟩ Elementen der BNF wird analog zu While-Schleifen dargestellt. Anstelle der Bedingung wird die Anzahl an Schleifendurchläufen in der Raute dargestellt. Beide Arten sensorbasierter Schleifen und die Zählschleife sind in Abbildung 4.4b dargestellt. Enthält der Rumpf einer Schleife ein ⟨increment conf⟩ Element, so wird diese als schwarzer Kreis mit einem ± 1 daneben visualisiert. Dies ist im Programm von Roboter 1 in Abbildung 4.4b zu sehen.

Zusätzlich zu den sensorbasierten Schleifen werden sensorbasierte Verzweigungen aus den ⟨switch stmt⟩ Elementen der BNF als schwarze Ringe mit Rauten für jeden Zweig in der Zeitleiste graphisch dargestellt. In den Rauten werden die Sensorwerte der Zweige dargestellt. Wird eine Verzweigung in einem Roboterprogramm erreicht, so wird der aktuelle Sensorwert ausgewertet und mit den Referenzwerten der einzelnen Zweige verglichen. Der Zweig mit dem ähnlichsten Wert wird anschließend ausgeführt. Ist keiner der Referenzwerte ähnlich zum aktuellen Sensorwert, so wird der Default-Zweig, welcher rechts neben dem schwarzen Ring dargestellt wird, ausgeführt. Ein Beispiel dazu ist in Abbildung 4.5a dargestellt.

Die letzten Elemente der BNF, welche eine graphische Darstellung benötigen, sind die ⟨sync stmt⟩ Elemente für die Synchronisation. Dabei kann es sich entweder um einen Synchronisationspunkt oder ein Synchronisationsintervall handeln. Beide werden entweder verwendet um mehrere Roboter aufeinander zu synchronisieren oder um einen Menschen mit einem Roboter zu synchronisieren. Intervalle werden als eckige Klammern links und rechts vom Intervall im Roboterprogramm dargestellt, Punkte als vertikale Linien. Neben den Klammern bzw. Linien wird jeweils die Synchronisations-ID des Intervalls oder Punkts angegeben. Die Klammern und Linien sind dabei hellblau eingefärbt. Sind die Zeitleisten der zu synchronisierenden Roboter nicht direkt übereinander, so werden die Klammern bzw. Linien mit gepunkteten Linien über die nicht an der Synchronisation beteiligten Roboter verbunden. Gleiches gilt wenn die Zeitleiste für den Roboter und die für den Menschen nicht direkt übereinander liegen. Zusätzlich werden bei Roboter-Roboter Synchronisationen diese Linien bis zur Zeitachse verlängert. Ein Beispiel dazu ist in Abbildung 4.5b dargestellt.

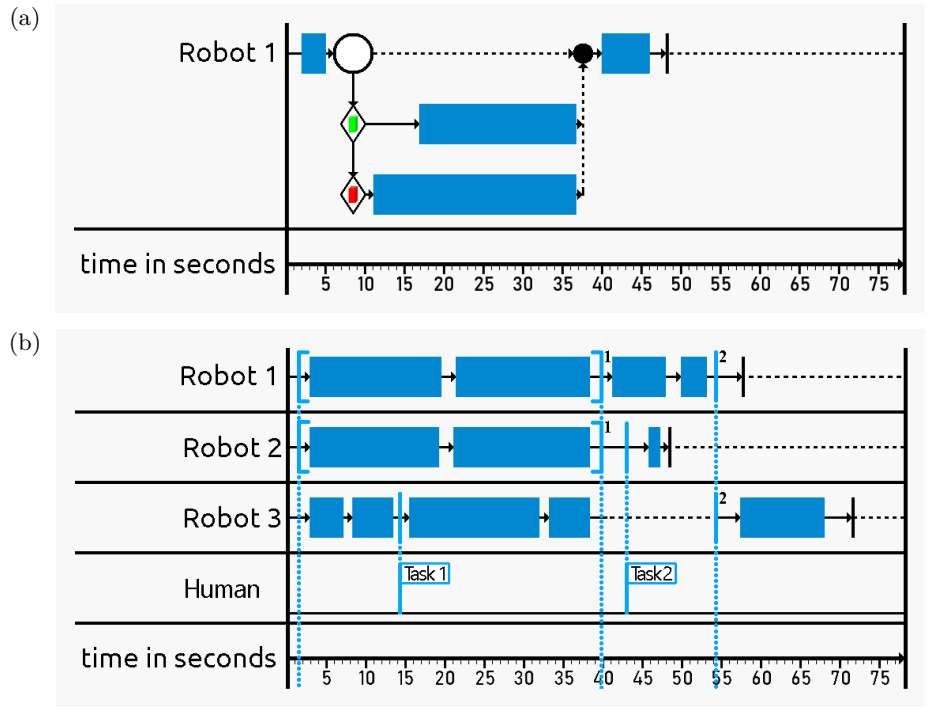


Abbildung 4.5: Graphische Darstellung von Verzweigungen und Synchronisationen. Abbildung (a) zeigt die graphischen Darstellungen für ein ⟨switch stmt⟩ mit zwei Zweigen. Abbildung (b) zeigt die graphische Darstellung für ⟨sync stmt⟩ Elemente, wobei Roboter 1 und Roboter 2 mit einem ⟨sync interval⟩ (Zwischen Sekunde 1.5 und 40) und Roboter 1 und Roboter 3 mit einem ⟨sync point⟩ (Sekunde 54) synchronisiert werden. Zusätzlich wird Roboter 3 (Sekunde 14.5) und Roboter 2 (Sekunde 43) jeweils mit einem ⟨human point⟩ synchronisiert.

4.2.3 Geschachtelte Programme graphisch darstellen

Wenn mehrere Roboter am Programmiersystem angemeldet sind und jeder Roboter ein geschachteltes Roboterprogramm mit mehreren Kontrollstrukturen besitzt kann die graphische Darstellung dieser Programme unübersichtlich werden. Aus diesem Grund wird eine skalierbare Schichtstruktur der graphischen Elemente der BNF entwickelt, so dass auch komplizierte und geschachtelte Roboterprogramme übersichtlich dargestellt werden können.

Um Skalierbarkeit zu erhalten wird der Bereich für die graphische Darstellung der Zeitleisten in äquidistante Schichten aufgeteilt, was durch die roten Linien in Abbildung 4.6 dargestellt ist. Jede Komponente der BNF hat eine feste Menge an Schichten, welche sie benötigt um dargestellt zu werden. Die Elemente ⟨motion stmt⟩, ⟨tool stmt⟩ und ⟨sync stmt⟩ benötigen eine Schicht, ⟨while stmt⟩, ⟨do-while stmt⟩ und

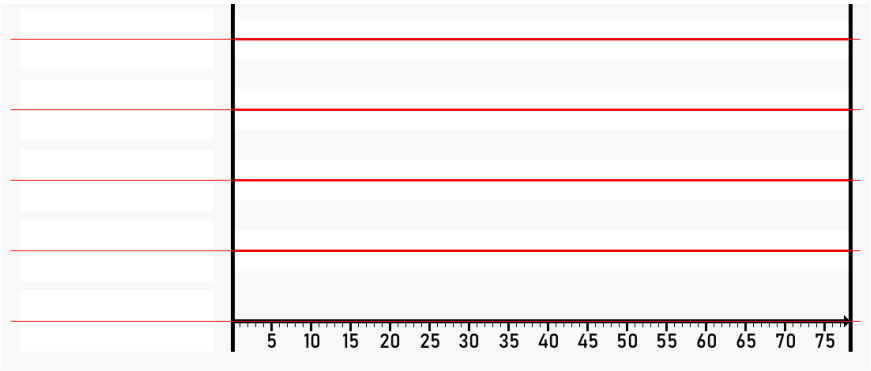


Abbildung 4.6: Schematische Darstellung der Schichten der Zeitleiste. Durch diese Aufteilung der Zeitleisten wird eine skalierbare Darstellung der Roboterprogramme ermöglicht.

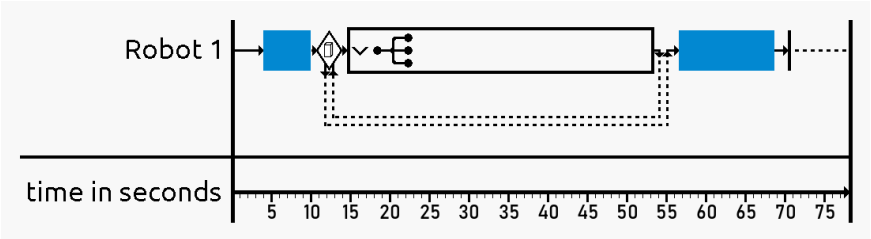


Abbildung 4.7: Darstellung eines eingeklappten Elements in der Zeitleiste. Die Black Box zeigt als Symbol das ⟨stmt⟩ an, welches eingeklappt wurde. In diesem Fall handelt es sich dabei um eine Verzweigung. Die ausgeklappte Version dieses Beispielprogramms ist in Abbildung 4.8 dargestellt.

⟨for stmt⟩ benötigen jeweils zwei Schichten, ein ⟨switch stmt⟩ benötigt eine Schicht für den Default-Zweig und je eine weitere Schicht für jeden Zweig der zur Verzweigung hinzugefügt wird. Zusätzliche Roboter und deren Programme werden in einer neuen Schicht unterhalb der letzten Schicht des zuletzt hinzugefügten Roboters dargestellt. Sind Elemente innerhalb eines Roboterprogramms geschachtelt, so erhöht sich die Menge an Schichten die benötigt werden abhängig von den Komponenten die geschachtelt sind.

Sind alle Schichten zum Darstellen von Roboterprogrammen in Verwendung, ist es möglich die Höhe der existierenden Schichten zu reduzieren und so Platz für weitere Schichten hinzuzufügen. Eine weitere Möglichkeit alle Schichten anzuzeigen besteht darin, scrollen zu ermöglichen. Auf der einen Seite kann man bei reduzierter Schicht-

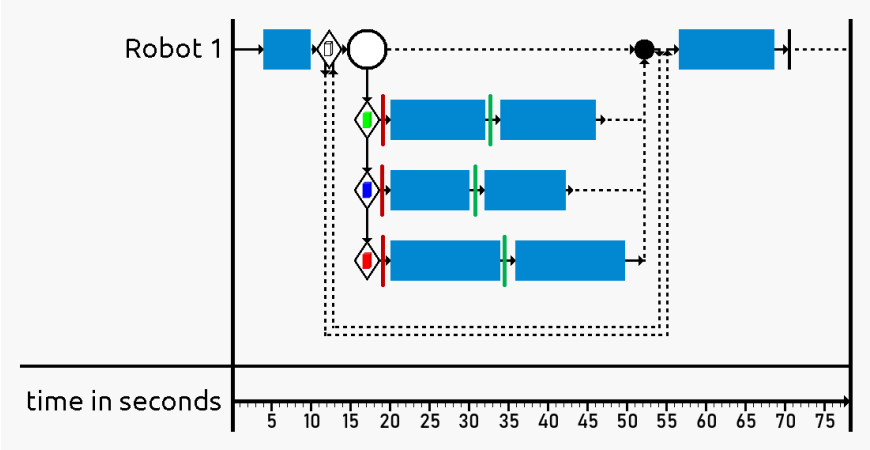


Abbildung 4.8: Graphische Darstellung zum Beispielprogramm aus Auflistung 4.2. Dargestellt ist ein Sortiervorgang, welcher so lange Quader nach deren Farbe sortiert, bis kein Quader mehr an der Auswerteposition vorhanden ist.

höhe immer noch alle Schichten der Roboterprogramme sehen, wobei dabei unter Umständen Details im Roboterprogramm nicht mehr sichtbar sind. Auf der anderen Seite erlaubt das Scrollen die Darstellung der Schichten in ihrer ursprünglichen Größe, aber es können nicht alle Schichten gleichzeitig angezeigt werden.

Deshalb gibt es die Option, einzelne Elemente, welche mehr als eine Schicht benötigen, einzuklappen. Dies sorgt dafür, dass die Benutzer:innen entscheiden können, solche Elemente einzuklappen, bei denen der Inhalt für sie irrelevant ist. Diese werden dann als Black-Box dargestellt, welche nur eine Schicht benötigt. Ein Beispiel dazu ist in Abbildung 4.7 dargestellt.

Die Kombination aus Änderung der Schichthöhe, Scrollen und Einklappen sorgt dafür, dass die graphische Darstellung skalierbar wird. Dadurch wird es prinzipiell möglich, mehrere Roboter mit unendlich tief geschachtelten Roboterprogrammen darzustellen.

4.2.4 Beispielprogramm

Nachdem in Auflistung 4.2 bereits eine Sortieraufgabe als Beispielprogramm in der BNF dargestellt wurde, wird in diesem Abschnitt das selbe Programm als graphische Darstellung entlang einer Zeitleiste gezeigt. Abbildung 4.8 zeigt die duale graphische Darstellung zur textuellen Darstellung aus Auflistung 4.2.

Man kann vor der While-Schleife die Bewegung von der Startposition zur Auswerteposition (Sekunde 0 bis 12) erkennen. Als Bedingung in der Raute der Schleife ist ein Quader erkennbar, dessen Farbe egal ist. Die Farbe des Quaders wird anschließend in der Verzweigung ausgewertet, und je nachdem ob es ein grüner, blauer oder roter Quader ist wird einer der drei Zweige ausgeführt. Zum Ende der Verzweigung werden die drei Kontrollflüsse im schwarzen Kreis wieder zusammengeführt und danach erneut die Bedingung der Schleife ausgewertet. Wird kein Quader mehr erkannt, so wird an das Ende des Schleifenrumpfes gesprungen und die Bewegung zurück zur Startposition ausgeführt (Sekunde 55 bis 70). Anschließend terminiert das Programm.

4.3 Evaluation der Verständlichkeit der graphischen Darstellung

Mit der Evaluation soll gezeigt werden, dass die entwickelte graphische Darstellung von geschachtelten, kinästhetisch demonstrierten Roboterprogrammen entlang von Zeitleisten für die Benutzer:innen leicht verständlich ist. Aus diesem Grund wurde vom 02.10.2019 bis zum 09.10.2019 eine Onlineumfrage mit 53 Teilnehmer:innen durchgeführt, welche die graphische Darstellung vorher noch nicht gesehen hatten. Zunächst wird in Abschnitt 4.3.1 das Studiendesign erläutert bevor anschließend in Abschnitt 4.3.2 die Ergebnisse der Onlineumfrage genauer betrachtet werden.

4.3.1 Studiendesign

Zu Beginn der Umfrage wurde den Teilnehmer:innen eine kurze schriftliche Einleitung gegeben, in der die unterschiedlichen Elemente der graphischen Darstellung erläutert wurden. Anschließend mussten alle Teilnehmer:innen fünf Fragen beantworten. Jede der Fragen beinhaltete ein Video, welches die Ausführung eines kinästhetisch demonstrierten Roboterprogramms in einer Robotersimulation zeigte. Nachdem das Video fertig abgespielt wurde, mussten die Teilnehmer:innen aus einer Menge von vier ähnlichen graphischen Darstellungen von Roboterprogrammen diejenige Darstellung auswählen, welche das im Video gezeigte Roboterprogramm darstellt. Die vier Darstellungen unterschieden sich zum Beispiel darin, dass einzelne Teile der Roboterbewegungen unterschiedlich lang waren, Greifer schließen und Greifer öffnen vertauscht war oder das eine Schleife mit einer Verzweigung vertauscht war. Eine der Fragen ist in Abbildung 4.9 abgebildet. In der Umfrage wurden zwei Kriterien evaluiert: Die Zeit, welche zum Ausfüllen der Umfrage benötigt wurde und die Menge an korrekt gegebenen Antworten in der Umfrage.

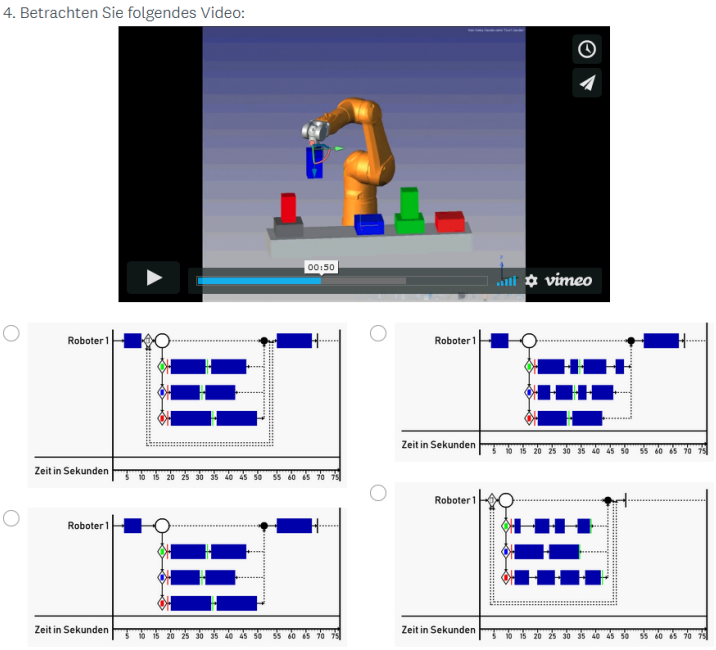


Abbildung 4.9: Eine der fünf Fragen, welche die Teilnehmer:innen im Laufe der Umfrage beantworten mussten.

4.3.2 Ergebnisse

Die 53 Teilnehmer:innen der Umfrage gaben insgesamt 265 Antworten, fünf pro Teilnehmer:in. Die Ergebnisse der Antworten sind in Abbildung 4.10a dargestellt. Von den 265 Antworten waren 258 korrekt, was 97.36% entspricht. Die falschen Antworten können in drei unterschiedliche Kategorien eingeteilt werden. *Greiferfehler* sind insgesamt drei Mal aufgetreten und haben einen Anteil von 1.13%. Darunter wurden falsche Antworten zusammengefasst, bei denen die Greiferbefehle vertauscht wurden (Greifer schließen mit Greifer öffnen und umgekehrt) oder ganz vergessen wurden. *Schleifenfehler* traten insgesamt drei Mal auf, was auch einem Anteil von 1.13% entspricht. Darunter wurden falsche Antworten zusammengefasst, bei denen entweder eine Schleife vergessen wurde oder eine Schleife mit einer Verzweigung vertauscht wurde. Die letzte Fehlerkategorie, *Bewegungsfehler*, trat nur einmal auf, was einem Anteil von 0.38% entspricht. Dabei wurde eine Antwort ausgewählt, bei der ein Bewegungsbefehl vergessen wurde. Insgesamt wurden sieben falsche Antworten gegeben, was einer Fehlerrate von 2.64% entspricht. Dies deutet darauf hin, dass die geschichtete graphische Darstellung von kinästhetisch demonstrierten Roboterprogrammen auf Basis der BNF mit Hilfe einer kurzen Einleitung leicht verständlich ist.

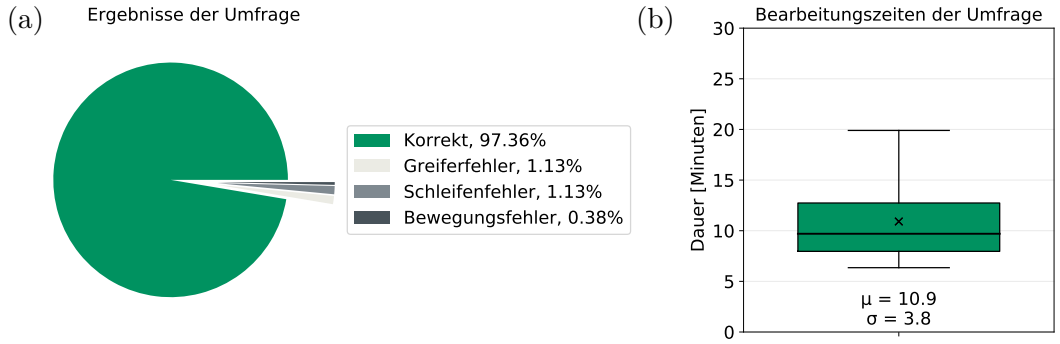


Abbildung 4.10: Die Ergebnisse der Evaluation zur Verständlichkeit der graphischen Darstellung. Das Kreisdiagramm (a) zeigt die die Anzahl korrekt gegebener Antworten. Von den insgesamt 265 Antworten der 53 Teilnehmer:innen waren 97.36% korrekt. 1.13% der Antworten waren ein Greiferfehler, 1.13% ein Schleifenfehler und 0.38% ein Bewegungsfehler. Der Boxplot (b) zeigt die Zeitdauern, die zum Beantworten des Fragebogens benötigt wurden.

Das zweite Kriterium, welches ausgewertet wurde, war die Gesamtzeit, welche die Teilnehmer:innen benötigten, um die Umfrage auszufüllen. Die Ergebnisse davon sind in einem Boxplot in Abbildung 4.10b dargestellt. Die Gesamtlaufzeit der fünf gezeigten Videos lag bei 5:42 Minuten. Die erwartete Minimalzeit um die Umfrage zu beenden sollte demnach höher liegen als dieser Wert. Die Minimalzeit zum Ausfüllen der Umfrage lag bei 6:21 Minuten, die Maximalzeit bei 19:54 Minuten. Der Mittelwert über alle Teilnehmer:innen lag bei 10:65 Minuten. Insgesamt 75% der Teilnehmer benötigten weniger als 12:46 Minuten zum Ausfüllen der Umfrage. Die Ausreißer welche ungefähr 20 Minuten benötigten können möglicherweise damit erklärt werden, dass diese die Umfrage begonnen haben, dann unterbrochen wurden, und erst mit einer Verzögerung abgeschlossen haben. Insgesamt hat das Ausfüllen im Mittel demnach nicht länger als die doppelte Gesamtlaufzeit der Videos betragen. Dies ist ein weiterer Indikator dafür, dass die Darstellung der Roboterprogramme leicht und schnell verständlich ist.

4.4 Schlussfolgerung

Abschließend kann festgehalten werden, dass in diesem Kapitel zwei zueinander duale, äquivalente Darstellungen für geschachtelte, sensorbasierte, kinästhetisch demonstrierte Roboterprogramme entwickelt wurden. Dies ist zum einen die BNF für die strukturierte textuelle Darstellung und zum anderen die skalierbare, geschichtete grafische Darstellung entlang von Zeitleisten. Beide Darstellungsarten lassen sich verlustfrei in-

einander überführen, so dass sowohl der Quellcode, als auch die graphische Darstellung editiert werden können, je nachdem welche Darstellung die Benutzer:innen bevorzugen. Abschließend wurde in einer Evaluation die Verständlichkeit der graphischen Darstellung bestätigt.

Forschungsfrage F1 kann damit beantwortet werden, dass mit Hilfe einer formalen Modellierung der Roboterprogramme als BNF und anschließender Übersetzung der Elemente der BNF in eine geschichtete Darstellung entlang einer Zeitleiste dafür gesorgt werden kann, dass geschachtelte Roboterprogramme strukturiert und skalierbar graphisch entlang einer Zeitleiste dargestellt werden können. Die Evaluation hat gezeigt, dass diese entwickelte graphische Darstellung leicht und schnell verständlich ist.

Nachdem in diesem Kapitel die graphische Darstellung von kinästhetisch demonstrierten Roboterprogrammen näher erläutert wurde, wird im folgenden Kapitel näher darauf eingegangen, wie die Roboterprogramme graphisch editiert werden können und gleichzeitig weiterhin ein valides Roboterprogramm bestehen bleibt.

KAPITEL 5

Editieren

Inhalt

Die kinästhetische Roboterprogrammierung in ihrer bisherigen Form kann nur die Trajektorie aufzeichnen und exakt wieder abspielen. Um die Art der Programmierung zu Erweitern, werden in diesem Kapitel graphische Editierfunktionen vorgestellt, welche auf der Zeitleistendarstellung eines kinästhetisch demonstrierten Roboterprogramms aus Kapitel 4 arbeiten. Diese Editierfunktionen bieten den Vorteil, dass existierende Roboterprogramme angepasst und verändert werden können, ohne dass das ganze Roboterprogramm neu aufgezeichnet werden muss. Insbesondere überflüssige Bewegungen und Stillstandszeiten können dadurch aus dem Roboterprogramm optimiert werden.

Die Art des Editierens ist vergleichbar mit dem Schneiden und neu Anordnen von Videosequenzen in einem Videoschnittprogramm oder dem Editieren durch Kopieren, Einfügen, Ausschneiden und Löschen in einem Textdokument. Jedoch ist es möglich, dass durch das Editieren der Roboterprogramme Probleme in Form von Sprungstellen an den Enden des eingefügten oder gelöschten Bereichs auftreten. Auch dafür wird in diesem Kapitel ein Konzept präsentiert, um die Validität des Roboterprogramms gewährleisten zu können. Zusätzlich wird ein Konzept vorgestellt, wie die Kontrollfluss-

Elemente der Roboterprogramme, welche in Kapitel 6 und 7 erläutert werden, per Drag-and-Drop innerhalb der Zeitleiste verschoben werden können. Dadurch wird es den Benutzer:innen ermöglicht, nach dem Programmieren die Roboterprogramme zu editieren, so dass Fehler ausgebessert, die Ausführungszeit optimiert, oder die Ausführungsreihenfolge umgestellt werden kann.

In Abschnitt 5.1 wird die Funktion Kopieren&Einfügen zum graphischen Editieren näher betrachtet. Anschließend zeigt Abschnitt 5.2 das Verschieben von Kontrollfluss-Elementen in der Zeitleiste mittels Drag-and-Drop. Abschließend wird in Abschnitt 5.3 darauf eingegangen, wie durch das Editieren entstandene Sprungstellen behandelt werden können.

In diesem Kapitel wird folgende Forschungsfrage behandelt:

F2 Auf welche Art und Weise lassen sich kinästhetisch demonstrierte Roboterprogramme graphisch entlang von Zeitleisten editieren?

Das graphische Editieren von kinästhetisch demonstrierten Roboterprogrammen wurde in Auszügen bereits in [Riedl2016], [Riedl2019] und [Colceriu2020] veröffentlicht. Das Behandeln von Sprungstellen, indem Zwischenbahnen durch die Benutzer:innen demonstriert werden, wurde bereits in [Riedl2018] veröffentlicht.

5.1 Kopieren&Einfügen

In diesem Abschnitt werden die unter dem Schlagwort *Kopieren&Einfügen* zusammengefassten Funktionen behandelt. Dies sind Kopieren, Einfügen, Ausschneiden und Löschen. Dabei handelt es sich um Funktionen, welche in ähnlicher Form in Videoschnittprogrammen und auch in Texteditoren vorhanden sind. Zunächst werden in Abschnitt 5.1.1 die Funktionsweisen der Einzelfunktionen erläutert und wie diese im vorgestellten Programmiersystem verwendet werden können. Anschließend wird in Abschnitt 5.1.2 auf das Problem der Sprungstellen eingegangen, welches beim Editieren eines Roboterprogramms auftreten kann.

5.1.1 Funktionsweise

Um eine der vier Kopieren&Einfügen-Funktionen verwenden zu können, muss von den Benutzer:innen entweder ein Zeitpunkt oder ein Zeitintervall in der Zeitleiste des zugehörigen Roboterprogramms ausgewählt werden. Abbildung 5.1 zeigt das Programmiersystem mit aufgeklappten Editierfunktionen in der Kontrollleiste und einem ausgewählten Zeitintervall zwischen Sekunde 15.0 und Sekunde 26.6. Gleichzeitig ist im Simulationsfenster der Teil des Roboterprogramms in blau hervorgehoben, der ausgewählt ist. Bei einem ausgewählten Zeitintervall kann die Kopieren, Ausschneiden oder Löschen-Funktion verwendet werden, bei einem ausgewählten Zeitpunkt die Einfügen-Funktion. Zeitpunkte können im Programmiersystem entweder per Mausklick mit der

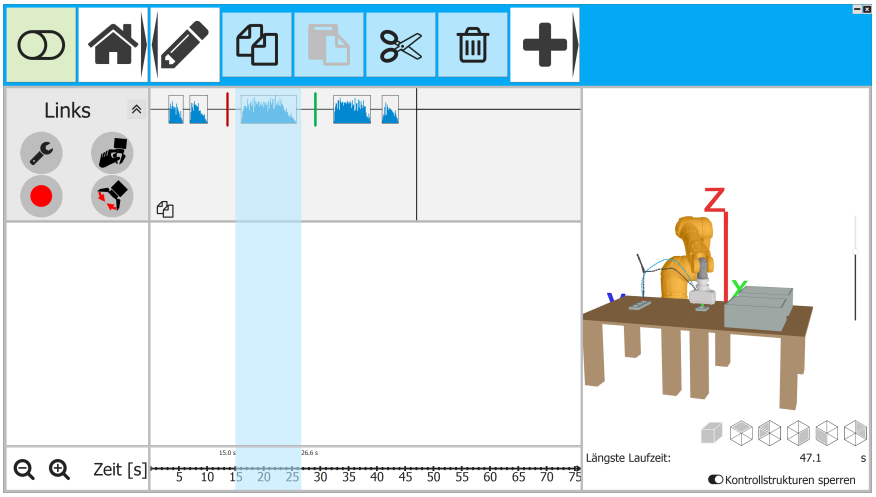


Abbildung 5.1: Bildschirmaufnahme des Programmiersystems im Programmiermodus. In der Zeitleiste des Roboters ist ein Bereich zwischen Sekunde 15.0 und 26.6 markiert und die Editierfunktionen der Kontrollleiste sind ausgeklappt.

linken Maustaste oder durch antippen mit dem Finger markiert werden. Zeitintervalle hingegen können entweder durch drücken und halten der linken Maustaste und bewegen der Maus über den auszuwählenden Bereich oder durch wischen über den auszuwählenden Bereich mit dem Finger markiert werden.

Wird die *Kopieren*-Funktion verwendet, so wird der markierte Teil des Roboterprogramms in der Zwischenablage gespeichert, so dass dieser später an einer beliebigen anderen Stelle wieder eingefügt werden kann. Befindet sich ein Teil eines Roboterprogramms in der Zwischenablage, so wird dies durch das *Kopiert*-Symbol im unteren linken Bereich der Zeitleiste symbolisiert. Dies ist in Abbildung 5.1 zu sehen. Am bestehenden Roboterprogramm selbst wird durch die Kopieren-Funktion nichts geändert.

Bei Verwendung der *Löschen*-Funktion wird der markierte Teil aus dem Roboterprogramm gelöscht. So wird bei dem Beispiel in Abbildung 5.1 das Roboterprogramm so verkürzt, dass nach dem Programmteil vor Sekunde 15.0 direkt der Programmteil nach Sekunde 26.6 ausgeführt wird. Der Bereich dazwischen wird entfernt und das Roboterprogramm selbst wird durch diese Funktion verändert.

Die *Ausschneiden*-Funktion kombiniert die beiden Funktionen Kopieren und Löschen. Wird auf ein ausgewähltes Zeitintervall die Ausschneiden-Funktion aufgerufen, so wird der markierte Programmabschnitt zuerst in die Zwischenablage kopiert und anschließen aus dem Roboterprogramm gelöscht. Dadurch wird sowohl die Zwischenablage befüllt, als auch das Roboterprogramm verändert, indem der kopierte Bereich aus dem Roboterprogramm entfernt wird.

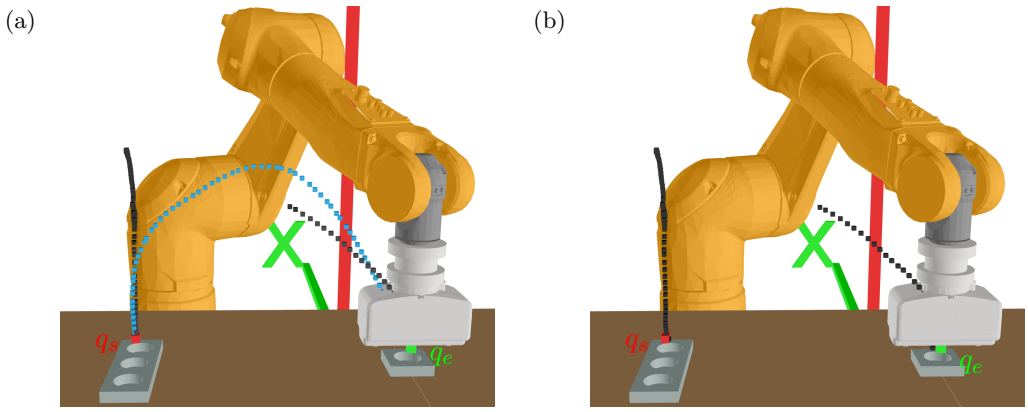


Abbildung 5.2: Bildschirmaufnahmen des Simulationsfensters. Bildschirmaufnahme (a) zeigt eine valide Trajektorie mit einer hellblau markierten Teiltrajektorie zwischen der roten Konfiguration q_s und der grünen Konfiguration q_e. Bildschirmaufnahme (b) zeigt die gleiche Trajektorie, nur dass der Bereich zwischen q_s und q_e gelöscht wurde und eine Sprungstelle in der Trajektorie existiert, da nicht klar ist, wie sich der Roboter von q_s nach q_e bewegen soll.

Die in der Zwischenablage gespeicherten Teil-Roboterprogramme können mit Hilfe der *Einfügen*-Funktion am markierten Zeitpunkt wieder in ein bestehendes Roboterprogramm eingefügt werden. Dazu wird der Teil des Roboterprogramms rechts vom markierten Zeitpunkt um die Länge des einzufügenden Programmteils nach rechts in der Zeitleiste verschoben. Auch diese Funktion verändert das bestehende Roboterprogramm.

Mit Hilfe der Kopieren&Einfügen-Funktionen kann auf eine einfache Art und Weise ein sich wiederholender Programmabschnitt mehrfach ausgeführt werden, indem dieser so oft kopiert und wieder eingefügt wird, bis die gewünschte Anzahl an Wiederholungen erreicht ist. Mit der Löschen-Funktion hingegen kann ein bestehendes Roboterprogramm von Hand optimiert werden, indem zum Beispiel unnötige Bewegungen oder zu lange Stillstandszeiten des Roboters entfernt werden. Bei Verwendung dieser Editierfunktionen kann es jedoch dazu kommen, dass das Roboterprogramm Sprungstellen in der Trajektorie aufweist. Dies wird im nächsten Abschnitt genauer erläutert.

5.1.2 Problem beim Editieren

Wird eine der Editierfunktionen angewendet, welche das bestehende Roboterprogramm verändern, so kann es passieren, dass ein Roboterprogramm nicht mehr valide und damit nicht mehr ausführbar ist. Dies ist der Fall, da eine Sprungstelle in einem Roboterprogramm entsteht, wenn nicht zueinander passende Trajektorien aneinandergereiht werden. Wird zum Beispiel in dem Programm aus Abbildung 5.1 der markierte Bereich gelöscht, so passen die Konfiguration q_s vor dem gelöschten Bereich und die Konfiguration q_e nach dem gelöschten Bereich nicht mehr zueinander und eine Sprungstelle entsteht. Dies ist in Abbildung 5.2 dargestellt.

Dieses Problem tritt bei jeder der Editierfunktionen auf, welche das Roboterprogramm verändern, da im Allgemeinen nicht davon ausgegangen werden kann, dass die zusammengefügten Trajektorien des Roboterprogramms stetig sind. Für dieses Problem muss demnach eine allgemein gültige Lösung gefunden werden, um auch nach einem Editierschritt eine valide Trajektorie zwischen q_s und q_e zu erhalten.

Der nächste Abschnitt behandelt die Editiermöglichkeit des Drag-and-Drop von Elementen. Anschließend wird auf zwei Möglichkeiten der Erzeugung von Zwischenbahnen zur Vermeidung von Sprungstellen eingegangen.

5.2 Drag-and-Drop von Elementen in der Zeitleiste

Nachdem im letzten Abschnitt als Editiermöglichkeit für kinästhetisch demonstrierte Roboterprogramme die Kopieren&Einfügen-Funktionen vorgestellt wurden, wird in diesem Abschnitt näher auf eine weitere Editiermöglichkeit eingegangen. Dabei handelt es sich um Drag-and-Drop von Kontrollfluss-Elementen der kinästhetisch demonstrierten Roboterprogramme innerhalb der zugehörigen Zeitleiste. Dafür wird zunächst das Konzept zum Drag-and-Drop und die Besonderheiten dabei erläutert, anschließend wird genauer auf die einzelnen Elemente, welche verschoben werden können, eingegangen.

5.2.1 Konzept

Das Verschieben von Elementen des Roboterprogramms innerhalb der Zeitleisten ist eine weitere Funktion zum Editieren der Struktur von Roboterprogrammen. Während der Erstellung eines Programms kann es vorkommen, dass die Positionen einzelner Elemente, wie zum Beispiel Schleifen oder Verzweigungen, innerhalb der Zeitleiste angepasst werden müssen. Mit Hilfe der Drag-and-Drop Funktion muss dies nicht durch das Entfernen des alten Elements und das Hinzufügen des gleichen Elements an der neuen Position erfolgen, sondern kann durch verschieben des ursprünglichen Elements erreicht werden.

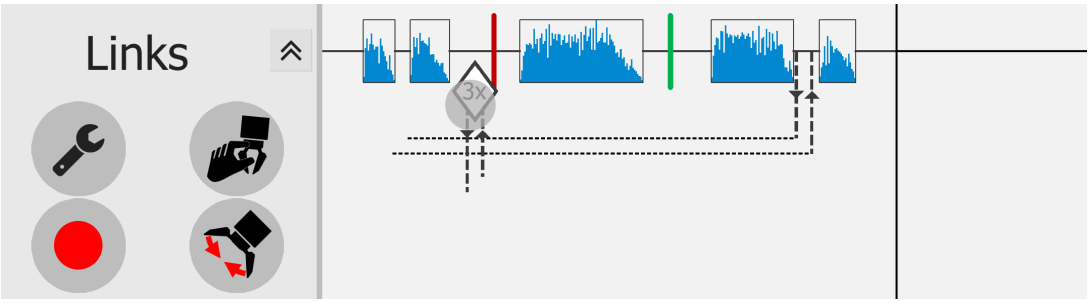


Abbildung 5.3: Bildschirmaufnahme eines Drag-and-Drop Vorgangs. Dabei wird der Beginn der Schleife von Sekunde 6 nach rechts verschoben. Der graue Kreis zeigt die Stelle, an der gerade mit dem Finger auf den Touchscreen gedrückt wird.

Eine Drag-and-Drop Operation funktioniert im Programmiersystem für die kinästhetische Roboterprogrammierung im Allgemeinen so, dass das zu verschiebende Element entweder mit der Maus, oder mit dem Finger innerhalb der Zeitleiste verschoben werden kann. In Abbildung 5.3 ist zu sehen, wie der Anfang einer Schleife innerhalb der Zeitleiste verschoben wird.

Eine Besonderheit beim Drag-and-Drop in kinästhetisch demonstrierten Roboterprogrammen besteht darin, dass die Elemente nur in Bereiche des Roboterprogramms abgelegt werden dürfen, an denen der Roboter still steht. Dies ist notwendig, da bei den jeweiligen Elementen Aktionen ausgeführt werden müssen, die es erfordern, dass der Roboter sich nicht bewegt. So muss unter Umständen ein Sensorwert ausgelesen oder auf einen anderen Roboter gewartet werden. Um auch nach der Drag-and-Drop-Operation noch ein valides Roboterprogramm sicherstellen zu können wird die Bedingung, dass nur bei Stillstand ein Element abgelegt werden darf, erzwungen.

Zusätzlich zur Möglichkeit zum Drag-and-Drop von Elementen der kinästhetischen Roboterprogrammierung sorgt ein Klick auf ein Element dafür, dass das Kontextmenü des jeweiligen Elements geöffnet wird. In diesem Kontextmenü können die Benutzer:innen die jeweiligen Elemente konfigurieren und so zum Beispiel weitere Zweige zu einer Verzweigung hinzufügen oder die Synchronisations-ID eines Synchronisationspunktes verändern.

5.2.2 Verschiebbare Elemente

Im Allgemeinen können alle Kontrollfluss-Elemente, welche in kinästhetisch demonstrierte Roboterprogramme eingefügt werden können, auch per Drag-and-Drop innerhalb dessen verschoben werden. Dabei handelt es sich in dieser Arbeit um Schleifen, Verzweigungen, Synchronisationspunkte und Synchronisationsintervalle. Zusätzlich zu den Kontrollfluss-Elementen ist es auch noch möglich, Greiferbefehle zu verschieben.

Greiferbefehle bilden eine Ausnahme zu der im vorherigen Abschnitt aufgestellten Bedingung, dass Elemente nur abgelegt werden dürfen, wenn der Roboter still steht. Diese dürfen auch an Stellen verschoben werden, an denen der Roboter nicht still steht, da einen Greifer öffnen oder schließen auch in der Bewegung möglich ist.

Dadurch, dass der Beginn und das Ende einer Schleife verschoben werden können, lässt sich auch nach dem Einfügen von sensorbasierten Schleifen deren Rumpf nachträglich verändern. So können zum Beispiel weitere Bewegungsfragmente eines Roboterprogramms zum Schleifenrumpf hinzugefügt oder entfernt werden. Wird die Begrenzung einer Schleife verschoben, welche den Sensorwert enthält, so wird auch gleichzeitig die Konfiguration, an welcher der Sensorwert ausgewertet wird, angepasst.

Bei der Verschiebung einer Verzweigung wird nicht nur die Position der Verzweigung und deren Zweige innerhalb der Zeitleiste verschoben, sondern auch die Position, an welcher der Sensor ausgewertet wird. Dadurch wird es unter Umständen notwendig, neue Soll-Stimuli für die einzelnen Zweige der Verzweigung aufzuzeichnen, da der Sensorwert von einer neuen Position des Roboters aus aufgezeichnet wird.

Abschließend ist es noch möglich, Synchronisationspunkte und -intervalle in der Zeitleiste eines Roboters zu verschieben. Dadurch wird es ermöglicht, dass ein Intervall vergrößert werden kann ohne das ein neues in die Bewegung eingefügt werden muss. Zusätzlich kann die Position eines Punktes verschoben werden, wenn sich zum Beispiel bei Testläufen herausgestellt hat, dass eine andere Position besser zur Synchronisation der Bewegung geeignet ist.

5.3 Zwischenbahnen zur Vermeidung von Sprungstellen

Wie im vorhergehenden Abschnitt dargestellt, ergibt sich durch das Editieren mit Kopieren&Einfügen bei kinästhetisch demonstrierten Roboterprogrammen das Problem der Unstetigkeit durch Sprungstellen in der Trajektorie. Um aus der unstetigen Trajektorie des Roboterprogramms wieder eine stetige und damit valide Trajektorie zu erhalten, muss an den Unstetigkeiten zwischen q_s und q_e eine Zwischenbahn eingefügt werden.

Zum Einfügen dieser Zwischenbahn werden im Folgenden zwei Methoden vorgestellt. Zum einen das automatische Erzeugen der Zwischenbahn indem im Gelenkraum Start- und Zielposition zeitoptimal verbunden werden, ein Geschwindigkeitsprofil mit Randbedingungen bezüglich Start- und Zielgeschwindigkeit berechnet wird und anschließend mit einer vorgegebenen Abtastrate die Konfigurationen der Zwischenbahn erzeugt werden. Zum anderen wird ein Konzept vorgestellt, welche es den Benutzer:innen erlaubt, den Roboter vom Start zum Ziel der Sprungstelle manuell zu Führen. Unterstützt werden die Benutzer:innen dabei mittels visuellem und haptischem Feedback.

5.3.1 Automatische Erzeugung der Zwischenbahnen

Die automatische Erzeugung der Zwischenbahn erfolgt so, dass zwischen der Startkonfiguration q_s und Endkonfiguration q_e der Sprungstelle mit vorgegebenen Randbedingungen eine zeitoptimale Punkt-zu-Punkt-Bewegung (PTP-Bewegung) im Gelenkraum erzeugt und diese anschließend mit einer vorgegebenen Abtastrate abgetastet wird. Die Randbedingungen sind, dass die Geschwindigkeit q'_s an der Startkonfiguration und q'_e an der Endkonfiguration gleich den Geschwindigkeiten an den Sprungstellen des Roboterprogramms sein sollen und das die maximale Beschleunigung $q''_{\max}$ und die maximale Geschwindigkeit $q'_{\max}$ eines Gelenks nicht überschritten werden darf.

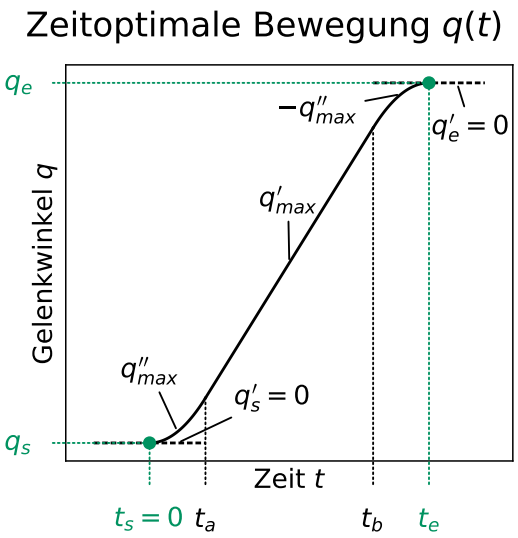


Abbildung 5.4: Gelenkwinkelprofil einer zeitoptimalen Bewegung nach [Craig2005].

Das automatische Erzeugen hat für die Benutzer:innen den Vorteil, dass beim Editieren immer eine valide Trajektorie erzeugt wird. Sollten jedoch Hindernisse existieren, so kann eine Kollision mit diesen nicht ausgeschlossen werden, da im Allgemeinen für eine zeitoptimale PTP-Bewegung der Verlauf im Arbeitsraum nicht intuitiv vorhergesagt werden kann. Für diesen Fall ist die Demonstration durch die Benutzer:innen mit Hilfe von multimodalem Feedback die bessere Lösung. Folgende Abschnitte zeigen die konkrete Berechnung und Abtastung der zeitoptimalen PTP-Bewegung, bevor danach näher auf die Demonstration der Zwischenbahn durch die Benutzer:innen eingegangen wird.

Berechnung der PTP-Bewegung

Zur Berechnung der PTP-Bewegung zwischen der Startkonfiguration q_s und der Endkonfiguration q_e mit der durch die Randbedingungen vorgegebenen Geschwindigkeit q'_s and der Startposition und q'_e an der Endposition wird der Ansatz der zeitoptimalen Bewegung verwendet. Dieser wird für den Spezialfall $q'_s = 0$ und $q'_e = 0$ in [Craig2005] in Kapitel 7.3 unter der Überschrift „Linear function with parabolic blends“ präsentiert.

Die zeitoptimalen Bewegungen haben den Vorteil, dass sowohl eine vorher definierte maximale Geschwindigkeit $q'_{\max}$ als auch eine maximale Beschleunigung $q''_{\max}$ eingehalten werden. Der Ansatz funktioniert so, dass zu Beginn der Bewegung an q_s mit maximaler Beschleunigung $q''_{\max}$ so lange beschleunigt wird, bis die maximale Geschwindigkeit $q'_{\max}$ erreicht wird. Anschließend bewegt sich das Gelenk mit konstanter

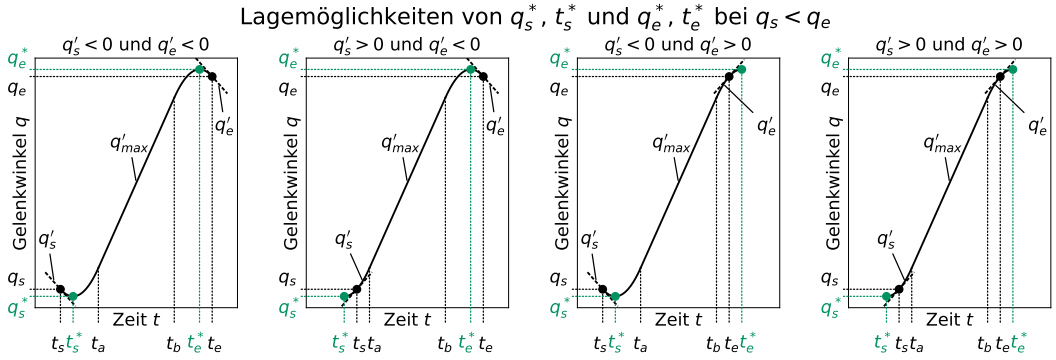


Abbildung 5.5: Die vier verschiedenen Lagemöglichkeiten von q_s^*, t_s^* zu q_s, t_s und q_e^*, t_e^* zu q_e, t_e für den Fall, dass $q_s < q_e$ gilt. Analog dazu können die Lagemöglichkeiten für den Fall das $q_s > q_e$ gilt angegeben werden.

Geschwindigkeit weiter, bis zum Erreichen von q_e mit maximaler Beschleunigung abgebremst wird. Dadurch entsteht ein zeitlicher Gelenkwinkelverlauf, welcher in drei Teile eingeteilt werden kann. Im Bereich $t_s = 0 \leq t < t_a$ verläuft die Kurve quadratisch, im Bereich $t_a \leq t < t_b$ linear und im Bereich $t_b \leq t \leq t_e$ wieder quadratisch, wobei q_s zum Zeitpunkt $t = t_s = 0$ und q_e zum Zeitpunkt $t = t_e$ erreicht werden. Die Zeiten t_a, t_b und t_e können nach dem Ansatz von [Craig2005] berechnet werden. Ein beispielhafter Verlauf einer zeitoptimalen Bewegung mit allen relevanten Größen ist in Abbildung 5.4 dargestellt.

Um den Ansatz von [Craig2005] auf das Problem dieser Arbeit anwenden zu können, muss das hier vorliegende allgemeine Problem mit beliebigen Start- und Endgeschwindigkeiten auf das spezielle Problem mit $q_s' = q_e' = 0$ reduziert werden. Hierfür werden zusätzlich die Parameter q_s^*, t_s^*, q_e^* und t_e^* eingeführt, welche aus den gegebenen Werten für q_s, q_s', q_e und q_e' berechnet werden können. Die Parameter q_s^* und q_e^* werden so gewählt, dass die Geschwindigkeit an diesen Positionen 0 beträgt. Dadurch kann zwischen q_s^* und q_e^* mit Hilfe der zeitoptimalen Bewegung nach [Craig2005] ein Gelenkwinkelprofil berechnet werden.

Die Lagemöglichkeiten von q_s^*, t_s^* zu q_s, t_s und q_e^*, t_e^* zu q_e, t_e für den Fall, dass $q_s < q_e$ gilt, sind in Abbildung 5.5 dargestellt. Daraus ist zu erkennen, dass je nach Vorzeichen der Geschwindigkeiten q_s' und q_e' die Punkte q_s mit Geschwindigkeit q_s' und q_e mit Geschwindigkeit q_e' entweder zwischen q_s^* und q_e^* und somit auf dem mit [Craig2005] berechneten Gelenkwinkelprofil oder außerhalb dieses Gelenkprofils liegen. Sind die Punkte außerhalb des berechneten Gelenkwinkelprofils, so können diese durch verlängern der quadratischen Phase des Gelenkprofils erreicht werden. Zur Berechnung des Gelenkwinkelprofils ist demnach die Berechnung von q_s^* und q_e^* notwendig.

Zunächst werden die Zeiten Δt_s^* und Δt_e^* berechnet, welche benötigt werden, um von der gegebenen Start- bzw. Endgeschwindigkeit auf 0 abzubremsen:

$$\Delta t_s^* = \left| \frac{q_s'}{q_{\max}''} \right| \quad ; \quad \Delta t_e^* = \left| \frac{q_e'}{q_{\max}''} \right|$$

Mit diesen Zeiten wird der Bremsweg bzw. Beschleunigungsweg Δq_s^* und Δq_e^* berechnet:

$$\Delta q_s^* = \frac{1}{2} q_{\max}'' \cdot \Delta t_s^{*2} \quad ; \quad \Delta q_e^* = \frac{1}{2} q_{\max}'' \cdot \Delta t_e^{*2}$$

Aufgrund der Lage von q_s zu q_e kann nun q_s^* und q_e^* berechnet werden:

$$q_s^* = \begin{cases} q_s - \Delta q_s^* & \text{, wenn } q_s < q_e \\ q_s + \Delta q_s^* & \text{, sonst} \end{cases}$$

$$q_e^* = \begin{cases} q_e + \Delta q_e^* & \text{, wenn } q_s < q_e \\ q_s - \Delta q_s^* & \text{, sonst} \end{cases}$$

Mit den Gelenkwinkeln q_s^* und q_e^* kann mit Hilfe von [Craig2005] t_a, t_b, t_e^* und das Gelenkwinkelprofil $q(t)$ zwischen $t_s^* = 0$ und t_e^* berechnet werden. Dieses hat die Form:

$$q(t) = \begin{cases} \frac{1}{2} q_{\max}''(t - t_s^*)^2 + q_s^* & , t \in [t_s^*; t_a[\\ q_{\max}'' t_a (t - t_s^*) - \frac{1}{2} q_{\max}'' t_a^2 + q_s^* & , t \in [t_a; t_b[\\ q_{\max}''(t_a + t_b)(t - t_s^*) - \frac{1}{2} q_{\max}''(t - t_s^*)^2 - \frac{1}{2} q_{\max}''(t_a^2 + t_b^2) + q_s^* & , t \in [t_b; t_e^*] \end{cases}$$

Abtasten der PTP-Bewegung

Das Gelenkwinkelprofil $q(t)$ aus dem vorhergehenden Abschnitt muss vor der Abtastung noch angepasst werden, damit es von t_s bis t_e zur Generierung der Trajektorie verwendet werden kann. Hierfür muss zunächst t_s aus t_s^* und Δt_s^* berechnet werden und t_e aus t_e^* und Δt_e^*. Dafür wird die Signumfunktion $sgn(x)$ benötigt, welche einer reellen Zahl ihr Vorzeichne zuordnet:

$$t_s = \begin{cases} t_s^* - \Delta t_s^* & \text{, wenn } sgn(q_s - q_e) = sgn(q_s') \\ t_s^* + \Delta t_s^* & \text{, sonst} \end{cases}$$

$$t_e = \begin{cases} t_e^* - \Delta t_e^* & \text{, wenn } sgn(q_s - q_e) = sgn(q_e') \\ t_e^* + \Delta t_e^* & \text{, sonst} \end{cases}$$

Mit Hilfe dieser Zusammenhänge zwischen t_s, t_s^* und t_e, t_e^* kann das Gelenkwinkelprofil angepasst werden an eine zeitoptimale Bewegung mit Start- und Endgeschwindigkeiten ungleich Null. Dabei bleiben die Formeln gleich, nur die Definitionsbereiche für die Abschnitte von $q(t)$ ändern sich, so dass die Funktion jetzt im Bereich $t_s \leq t \leq t_e$ definiert ist:

$$q(t) = \begin{cases} \frac{1}{2} q''_{\max}(t - t_s^*)^2 + q_s^* & , t \in [t_s; t_a[\\ q''_{\max} t_a (t - t_s^*) - \frac{1}{2} q''_{\max} t_a^2 + q_s^* & , t \in [t_a; t_b[\\ q''_{\max}(t_a + t_b)(t - t_s^*) - \frac{1}{2} q''_{\max}(t - t_s^*)^2 - \frac{1}{2} q''_{\max}(t_a^2 + t_b^2) + q_s^* & , t \in [t_b; t_e] \end{cases}$$

Dieses Gelenkwinkelprofil kann mit der vom Programmiersystem vorgegebenen Abtastrate abgetastet werden um eine Trajektorie für die Zwischenbahn zu erzeugen. Diese erzeugte Trajektorie erfüllt sowohl die vorgegebenen Bedingungen zu Startposition, Startgeschwindigkeit, Zielposition und Zielgeschwindigkeit, als auch die Einhaltung der maximalen Geschwindigkeit und Beschleunigung des Robotersystems, so dass diese problemlos anstelle der Sprungstelle in das editierte Roboterprogramm eingefügt werden kann.

5.3.2 Demonstration durch Benutzer:innen

Wie im vorhergehenden Abschnitt beschrieben ist eine automatische Erzeugung der Zwischenbahn nicht immer sinnvoll. Dadurch, dass das in dieser Arbeit vorgestellt Programmiersystem kein Modell der Umwelt besitzt kann auch keine Bahnplanung zur automatischen Erzeugung der Zwischenbahnen bei Sprungstellen eingesetzt werden. Wie in Abbildung 5.6a dargestellt kann es bei einer automatisch erzeugten PTP-Bewegung zu Kollisionen kommen, wohingegen die Benutzer:innen den Roboter intuitiv um das Hindernis führen könnten. In dieser Abbildung ist zugleich das visuelle Feedback in Form einer Ampelfarbgebung als Indikator für die Entfernung des Roboters zur Zielposition dargestellt.

Um den Benutzer:innen das Demonstrieren der Bewegung zwischen Startkonfiguration q_s und Zielkonfiguration q_e der Sprungstelle zu erlauben wurde ein Konzept entwickelt, um diese mittels visuellem und haptischem Feedback zu unterstützen. Das visuelle Feedback wird dafür verwendet, um die Navigation zur Zielposition zu unterstützen, wohingegen das haptische Feedback verwendet wird, um mittels Erhöhen der Steifigkeitswerte des Roboters anzuzeigen, dass die Zielposition erreicht ist. Beide Feedback-Methoden werden in den nächsten Abschnitten näher erläutert. Dabei ist $p_s \in \mathbb{R}^3$ die kartesische Startposition zur Startkonfiguration q_s, $p_e \in \mathbb{R}^3$ die kartesische Endposition zur Endkonfiguration q_e und $p_c \in \mathbb{R}^3$ die aktuelle kartesische Position des Tool-Center-Points im Arbeitsraum des Roboters.

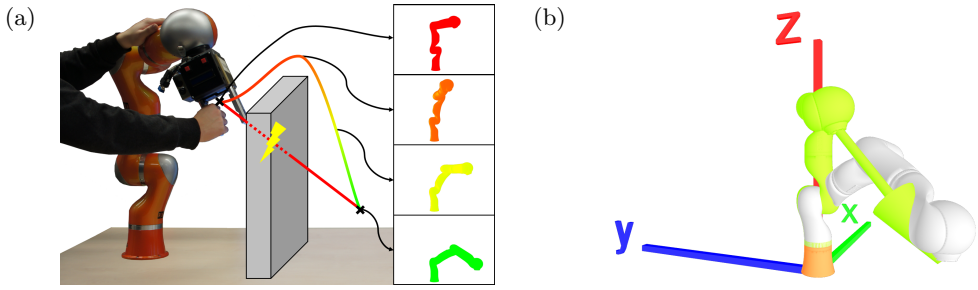


Abbildung 5.6: Abbildung (a) zeigt eine schematische Darstellung des zielgerichteten Führens mit multimodalem Feedback. Abbildung (b) zeigt eine Bildschirmaufnahme des Simulationsfensters während des Führens zu einer Zielposition. Dargestellt wird das Modell des Geisterroboters in grau und das Modell des realen Roboters eingefärbt nach dessen Erfüllungswert c. Beide Abbildungen sind aus [Riedl2018].

Visuelles Feedback

Der visuelle Teil des multimodalen Feedbacks zur Demonstration der Zwischenbahn durch die Benutzer:innen besteht daraus, dass im Simulationsfenster des Programmiersystems verschiedene Elemente angezeigt und eingefärbt werden. So wird ein Robotermodell in der aktuellen Konfiguration des Roboters, ein Geister-Robotermodell in der anzufahrenden Zielkonfiguration, und ein Pfeil zwischen den beiden Robotermodellen angezeigt. Abbildung 5.6b zeigt ein Bildschirmfoto des Simulationsfensters während der Demonstration einer Zwischenbahn.

Um Feedback zur Entfernung des Roboters zur Zielposition zu geben wird das Robotermodell, welches die aktuelle Position anzeigt, und der Pfeil in einem Ampelfarbschema mit einer Farbe zwischen rot und grün eingefärbt. Die Farbe wird ausgewählt mit Hilfe des Erfüllungswertes c, der beschreibt, zu wie viel Prozent die Zielposition erreicht ist.

$$c = 1 - \frac{||p_e - p_c||}{||p_e - p_s||}$$

Für alle Werte $c \leq 0$ wird das Modell und der Pfeil rot eingefärbt, für $c = 1$ grün. Für alle Werte $0 < c < 1$ wird eine Farbe zwischen rot und grün interpoliert um ein Ampel-Farbschema von rot über gelb nach grün zu erhalten, welches durch die Benutzer:innen leicht zu interpretieren ist. In Abbildung 5.6b hat der Roboter einen Erfüllungswert von $c \approx 0.75$.

Haptisches Feedback

Für den haptischen Teil des multimodalen Feedbacks ist es notwendig, dass der verwendete Roboter ein Anpassen der Steifigkeitswerte unterstützt. Das haptische Feedback besteht aus zwei Hauptbestandteilen, dem *Einrasten* und *Ausrasten* des Roboters. Um beide Bestandteile zu beschreiben wird zunächst ein kugelförmiger Bereich um p_e mit Radius ϵ definiert, der Einrastbereich genannt wird.

Der Roboter rastet ein, wenn die Benutzer:innen den Tool-Center-Point des Roboters in den Einrastbereich geführt haben und die Geschwindigkeit des Roboters kleiner als die Grenzgeschwindigkeit v_{thr} ist. Das Einrasten ist so realisiert, dass die Steifigkeitswerte des Roboters erhöht werden, um den Benutzer:innen zu signalisieren, dass die Zielposition erreicht wurde.

Befindet sich der Roboter im eingerasteten Zustand ist es für die Benutzer:innen trotzdem möglich, diesen Zustand wieder zu verlassen. Dieser Vorgang wird als Ausrasten bezeichnet. Der Roboter rastet aus, wenn eine höhere Kraft als die vordefinierte Grenzkraft F_{thr} auf den Tool-Center-Point aufgebracht wird. Ist dies der Fall, so werden die Steifigkeitswerte des Roboters wieder zurückgesetzt, damit dieser per Hand geführt werden kann.

Der Demonstrationsvorgang wird beendet, sobald sich der Roboter länger als die Grenzzeit t_{thr} im eingerasteten Zustand befindet. Ist diese Bedingung erfüllt, so wird der Roboter in Positionskontrolle geschaltet und kann nicht mehr geführt werden. Anschließend verfährt der Roboter selbstständig den restlichen Weg von p_c nach p_e. Diese Bewegung wird als *Snapping* bezeichnet, da es nur noch eine kurze Bewegung zur Zielposition ist. Dadurch, dass die Position p_c des Roboters am Ende des Demonstrationsvorgangs maximal ϵ entfernt ist von p_e findet diese Snapping-Bewegung in einem begrenzten Raum statt und kann gefahrloser ausgeführt werden als eine automatisch berechnete PTP-Bewegung von p_s nach p_e.

Als geeignete Grenzwerte für das haptische Feedback haben sich $\epsilon = 5$ cm, $v_{thr} = 2\ \frac{\text{cm}}{\text{s}}$, $F_{thr} = 10$ N und $t_{thr} = 2$ s herausgestellt. Diese Werte wurden vor der Evaluation empirisch in einer Testphase bestimmt und zur Durchführung der Evaluation verwendet.

Evaluation der Intuitivität der Demonstration

Um eine Aussage über die Intuitivität des multimodalen Feedbacks zur Demonstration von Zwischenbahnen treffen zu können, wurde vom 9. März 2018 bis zum 14. März 2018 eine Benutzungsstudie mit 19 Teilnehmer:innen im Roboterlabor der Universität Bayreuth durchgeführt. Die Ergebnisse dieser Evaluation wurde bereits in [Riedl2018] veröffentlicht.

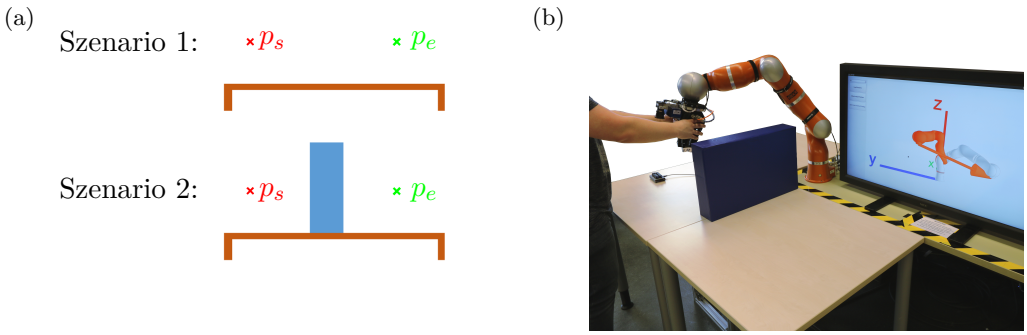


Abbildung 5.7: Abbildung (a) zeigt schematisch den Aufbau der beiden Szenarien, in welchen von den Teilnehmer:innen der Benutzungsstudie der Roboter von p_s nach p_e geführt werden musste. Abbildung (b) zeigt einen Teilnehmer beim Führen zur p_e in Szenario 2. Beide Abbildungen stammen aus [Riedl2018].

Jede:r Teilnehmer:in musste in zwei Szenarien einen Leichtbauroboter von einer Start- zu einer Zielposition führen. Das erste Szenario bestand daraus, den Roboter von p_s nach p_e ohne ein Hindernis dazwischen zu führen, um ein Gefühl für das visuelle und haptische Feedback zu bekommen. Im zweiten Szenario wurde die gleiche Start- und Zielposition verwendet, dieses Mal mit einem Hindernis dazwischen welches zu einer Kollision führt, wenn eine automatische PTP-Bewegung eingefügt wird. Beide Szenarien sind in Abbildung 5.7a skizziert. Ein Teilnehmer beim Ausführen von Szenario 2 ist in Abbildung 5.7b zu sehen.

Vor dem Ausführen der beiden Szenarien musste von den Teilnehmer:innen ein Fragebogen zu deren Vorkenntnissen im Bereich Robotik ausgefüllt werden. Anhand dieser Ergebnisse wurden die Teilnehmer:innen in zwei Gruppen, die Nicht-Expert:innen und die Expert:innen, eingeteilt. Während des Ausführens der Szenarien wurde vom Studienleiter ein Beobachtungsbogen ausgefüllt. Nach dem Ausführen der beiden Szenarien musste ein zweiter Fragebogen ausgefüllt werden. Mit diesem wurde die Zufriedenheit der Teilnehmer:innen mit Hilfe des QUESI-Fragebogens ermittelt. Der QUESI-Fragebogen wird verwendet, um einen numerischen Wert für die Intuitivität des Systems zu erhalten. Dabei werden 14 Fragen in fünf Kategorien gestellt, welche getrennt ausgewertet werden können und jeweils ein Indiz für die Intuitivität in der jeweiligen Kategorie sind. Der QUESI-Fragebogen wird in Kapitel 2.3.2 näher erläutert.

Die Teilnehmer:innen waren zwischen 21 und 52 Jahre alt mit einem Mittelwert von $\mu = 28.2$ Jahren und einer Standardabweichung von $\sigma = 7.2$ Jahren. Sie wurden in zwei Gruppen aufgeteilt, wobei als Expert:innen diejenigen klassifiziert wurden, welche schon Erfahrung mit Robotern hatten, und als Nicht-Expert:innen diejenigen, welchen noch nie etwas mit Roboter zu tun hatten. An der Studie nahmen elf Expert:innen und acht Nicht-Expert:innen teil. Sowohl der QUESI-Wert, als auch die Zeitdauern für beide Szenarien wurden evaluiert.

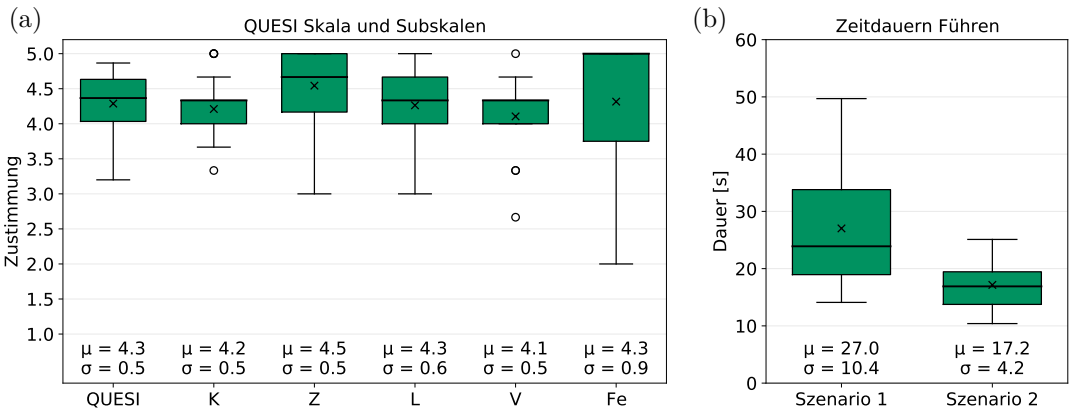


Abbildung 5.8: Die Ergebnisse der Evaluation des manuellen Führens mit multimodalem Feedback. Die Boxplots (a) zeigen den QUESI-Wert mit den Subskalen. Die Subskalen sind K: Wahrgenommene Kognitive Beanspruchung; Z: Wahrgenommene Zielerreichung; L: Wahrgenommener Lernaufwand; V: Vertrautheit/Vorwissen; Fe: Wahrgenommene Fehlerrate. Die Boxplots (b) zeigen die Zeitdauern für beide Szenarien, welche die Teilnehmer:innen der Studie zum manuellen Führen von der Startposition zur Zielposition benötigt haben.

Insgesamt ergab sich ein QUESI-Wert von im Mittel $\mu = 4.3$ (Standardabweichung $\sigma = 0.6$). Der maximale QUESI-Wert liegt bei 5. Vergleicht man diesen Wert mit Werten aus [Naumann2010], so zeigt sich, dass das zielgerichtete Führen ähnlich intuitiv zu Bedienen ist wie eine Nintendo Wii Konsole (QUESI-Wert 4.23) und intuitiver als ein Apple iPod Touch (QUESI-Wert 3.92). Dieses Ergebnis zeigt, dass die gesamte Intuitivität für den in dieser Arbeit vorgestellten Ansatz sehr gut ist. Aufgeteilt auf beide Teilnehmergruppen zeigt sich, dass Nicht-Expert:innen das System mit einem QUESI-Wert von $\mu = 4.4$ ($\sigma = 0.6$) leicht intuitiver bewerten als Expert:innen mit $\mu = 4.2$ ($\sigma = 0.7$). Da das System von beiden Gruppen als ähnlich intuitiv bewertet wurde, wird im Folgenden nicht weiter zwischen Expert:innen und Nicht-Expert:innen unterschieden sondern nur das Gesamtergebnis diskutiert.

Abbildung 5.8a zeigt den gesamten QUESI-Wert mit den Werten für jede Subskala. Der höchste Wert mit 4.5 wurde bei der wahrgenommenen Zielerreichung erzielt, was zeigt, dass ein Großteil der Teilnehmer:innen zufrieden war mit dem Ergebnis der Demonstration. Der niedrigste Wert hingegen wurde mit 4.1 in der Kategorie Vertrautheit/Vorwissen erzielt, was zeigt, dass für die Teilnehmer:innen nicht immer klar war, was getan werden musste. Einige Teilnehmer:innen gaben an, dass sie vor Szenario 1 nicht abschätzen konnten, wo der Einrastbereich des haptischen Feedbacks beginnt. Die verhältnismäßig große Standardabweichung bei der wahrgenommenen Fehlerrate kann so interpretiert werden, dass einige Teilnehmer:innen Probleme hatten das haptische Feedback zu spüren und deshalb den Roboter wieder ausgerastet haben, was als Fehler interpretiert wurde.

Die Zeitdauern für die beiden Szenarien sind in Abbildung 5.8b dargestellt. Für Szenario 1 ergibt sich ein Mittelwert von 27.0 s, für Szenario 2 von 17.2 s. Die deutlich geringeren Zeiten für Szenario 2 deuten darauf hin, dass eine Lernkurve für die multimodale Demonstration von Zwischenbahnen existiert. Obwohl aufgrund des Hindernisses die Länge der zu demonstrierenden Bahn für Szenario 2 länger ist, benötigten alle Teilnehmer:innen ohne Ausnahme geringere Zeitdauern für Szenario 2 als für Szenario 1.

Weitere Anmerkungen, welche von den Teilnehmer:innen gemacht wurden, sind, dass das visuelle Feedback auch Rückmeldung darüber geben sollte, wenn der Einrastbereich erreicht ist. Auch wurde angemerkt, dass verschiedene Ansichten des Simulationsfensters hilfreich wären, um einen besseren räumlichen Überblick zu erhalten. Zuletzt wurde von einigen Teilnehmer:innen genannt, dass vor dem ersten Versuch nicht klar war, wie sich das haptische Feedback anfühlen würde, aber danach das Führen zur Zielposition einfach war. Trotzdem kann festgehalten werden, dass das vorgestellte Verfahren zum multimodalen Demonstrieren einer Zwischenbahn von den Studienteilnehmern als intuitiv bewertet wurde.

5.4 Schlussfolgerung

Zusammenfassend wurden in dieser Arbeit zwei Konzepte zum graphischen Editieren eines kinästhetisch demonstrierten Roboterprogramms entworfen und implementiert. Zum einen das Editieren der Bewegung eines Roboterprogramms mit Hilfe von Kopieren&Einfügen, und zum anderen das Editieren der Struktur eines Roboterprogramms mit Drag-and-Drop. Da beim Editieren der Bewegung mittels Kopieren und Einfügen im Allgemeinen Sprungstellen entstehen wurden zusätzlich zwei Methoden entworfen und evaluiert um diese Sprungstellen einerseits automatisch zu behandeln und andererseits per manuellem Führen zu eliminieren.

Forschungsfrage F2 kann damit beantwortet werden, dass in dieser Arbeit zwei Konzepte des graphischen Editierens sensorbasierter Roboterprogramme entlang von Zeitleisten entwickelt wurden. Dabei kann der Editiervorgang ähnlich ablaufen wie in Videoschnittprogrammen oder in Texteditoren. Das auftretende Problem der Sprungstellen lässt sich mit Hilfe einer Demonstration durch die Benutzer:innen mit visuellem und haptischem Feedback oder mit Hilfe einer automatisch erzeugten Zwischenbahn lösen. Die Evaluation dieser Methode hat gezeigt, dass dieses Vorgehen für die Benutzer:innen intuitiv zu verstehen ist.

Bisher können kinästhetisch demonstrierte Roboterprogramme dargestellt und editiert werden. Um im weiteren Verlauf auch auf Umwelteinflüsse reagieren zu können, werden im nächsten Kapitel sensorbasierte Kontrollstrukturen behandelt.

KAPITEL 6

Kontrollieren

Inhalt

In diesem Kapitel werden die unterschiedlichen Möglichkeiten vorgestellt, ein bereits aufgezeichnetes Roboterprogramm zu kontrollieren. Dadurch erhält das Programmiersystem die Möglichkeit, nicht nur lineare Roboterprogramme zu erstellen, sondern auch alternative und sich wiederholende Programmstränge zu definieren. Diese Programmstränge werden dann abhängig von Sensorwerten ausgewählt oder wiederholt.

In Abschnitt 6.1 werden zunächst verschiedene Arten von möglichen Sensorwerten vorgestellt und aufgezeigt, wie diese miteinander verglichen werden können. Anschließend werden in Abschnitt 6.2 die zentralen Bestandteile dieses Kapitels, die sensorbasierten Kontrollstrukturen, präsentiert, welche auf dem Konzept von Schleifen und Verzweigungen prozeduraler Programmiersprachen basieren und für den Gebrauch mit kinästhetisch demonstrierten Roboterprogrammen angepasst wurden. Die Evaluation

dieser Kontrollstrukturen wird in Abschnitt 6.3 vorgestellt. Abschließend erläutert Abschnitt 6.4 eine Erweiterung der sensorbasierten Schleifen, das Schleifen-Inkrement. Dieses erweitert die Menge an Aufgaben, welche die Roboter lösen können, auf Stapel- und Palettieraufgaben.

In diesem Kapitel werden folgende Forschungsfragen behandelt:

F3 Wie lassen sich Konzepte der imperativen Programmierung wie Schleifen und Verzweigungen auf das kinästhetische Programmierkonzept übertragen?

F4 Wie lässt sich das Konzept von Schleifeninkrementen auf die kinästhetische Roboterprogrammierung übertragen?

Die sensorbasierten Kontrollstrukturen im Allgemeinen wurden in [Riedl2017] veröffentlicht, die Erweiterung der Schleifen um das Schleifen-Inkrement in [Riedl2021].

6.1 Sensorwerte

Wie bei Kontrollstrukturen in prozeduralen Programmiersprachen werden Bedingungen benötigt, welche ausgewertet werden können, um zu entscheiden, ob eine Schleife erneut ausgeführt werden soll oder welcher Zweig einer Verzweigung ausgeführt werden soll. Zur Realisierung von Kontrollstrukturen für die kinästhetische Roboterprogrammierung werden in dieser Arbeit Sensorwerte, auch Stimuli genannt, als Bedingungen verwendet. Dafür wird in der Programmierphase ein Soll-Sensorwert S_{soll} aufgezeichnet, welcher dann in der Ausführungsphase mit dem jeweils aktuellen Ist-Sensorwert S_{ist} verglichen wird. Sind beide Sensorwerte ähnlich, so wird die Bedingung als `TRUE` ausgewertet. Sind sich Soll-Sensorwert und Ist-Sensorwert hingegen nicht ähnlich, so wird die Bedingung als `FALSE` ausgewertet. Abhängig von der Kontrollstruktur, in der die Bedingung ausgewertet wird, werden je nach Ergebnis dieser Auswertung Aktionen durchgeführt oder übersprungen.

6.1.1 Notwendige Eigenschaften von Sensorwerten

Im Rahmen dieser Arbeit müssen Sensorwerte, die in Bedingungen verwendet werden sollen, zwei Eigenschaften erfüllen. Die erste notwendige Eigenschaft ist, dass diese auf Ähnlichkeit vergleichbar sein müssen. Das heißt, dass eine Funktion

`bool isSimilar(Stimulus` $S_{\texttt{ist}}$`, Stimulus` $S_{\texttt{soll}}$`)`

für jede Art von Sensorwert existiert, die `TRUE` zurückgibt, wenn beide Sensorwerte S_{ist} und S_{soll} ähnlich sind und `FALSE`, wenn nicht. Die zweite Eigenschaft besteht darin, dass zusätzlich zur alleinigen Aussage, ob zwei Sensorwerte ähnlich sind, noch ein numerisches Maß für die Ähnlichkeit berechnet werden kann. Dadurch kann ein Ist-Sensorwert mit mehreren Soll-Sensorwerten verglichen werden um herauszufinden, welcher am ähnlichsten zum Ist-Sensorwert ist. Notwendig für die interne Umsetzung ist dafür das Implementieren einer Funktion

`Stimulus getMostSimilarStimulus(Stimulus` S_{ist}`, Stimulus[]` S_{soll}`)`

welche den Sensorwert aus der Menge an Soll-Sensorwerten S_{soll} zurückgibt, welcher am ähnlichsten zu Sensorwert S_{ist} ist.

Im Rahmen der beiden oben genannten Forderungen können alle auf dem Feld der intelligenten Robotik gängigen Sensoren eingebunden werden. Beispiele für verwendbare Sensoren und deren zugehörige Sensorwerte sind unter anderem Farbkameras mit 2D-RGB-Kamerabildern, Tiefenkameras mit 3D-Punktewolken, Näherungssensoren mit Entfernungsangaben, Laserscanner mit 2D-Punktewolken oder auch Backen- oder Fingergreifer mit den jeweiligen Greiferöffnungswinkeln. Diese Aufzählung ist nicht vollständig und kann mit zusätzlichen Sensoren und Sensorwerten beliebig erweitert werden, falls diese die notwendigen Bedingungen erfüllen. In dieser Arbeit wurden als Sensoren Farbkameras mit 2D-RGB-Kamerabildern und Backengreifer mit Öffnungswinkel verwendet und implementiert. Der nächste Abschnitt zeigt beispielhaft für diese beiden Arten von Sensorwerten, wie die Vergleichsfunktion aufgebaut ist.

6.1.2 Vergleich von Sensorwerten

Nachdem im vorhergehenden Abschnitt die notwendigen Eigenschaften, welche für die kinästhetische Roboterprogrammierung an die Sensorwerte gestellt werden, und der daraus abgeleiteten beispielhaften Liste an möglichen Sensoren und Sensorwerten aufgezeigt wurden, wird nun in diesem Kapitel für zwei Arten von Sensorwerten exemplarisch gezeigt, wie die Vergleichbarkeit umgesetzt werden kann. Hierfür werden zum einen 2D-RGB-Kamerabilder aus Farbkameras und zum anderen Greiferöffnungswinkel aus Backengreifern verwendet. Beide Arten von Sensorwerten werden auch vom Demonstrator (siehe Kapitel 8.1) unterstützt.

2D-RGB-Kamerabilder

Dadurch, dass die zu vergleichenden 2D-RGB-Kamerabilder aufgrund der kinästhetischen Programmierung immer aus der gleichen Perspektive mit einer Eye-in-Hand Kamera, welche am Greifer befestigt ist, aufgenommen werden, wird auf eine Objekterkennung verzichtet und stattdessen ein Algorithmus verwendet, welcher auf den ganzen Bildern arbeitet um einen Wert für die Ähnlichkeit zu erhalten. Am geeignetsten hat sich dabei für die Konzepte dieser Arbeit bei empirischen Versuchen im Labor der Peak

Signal to Noise Ratio-Algorithmus (PSNR) herausgestellt. Dieser wurde ursprünglich verwendet, um die Qualität von Kompressionsalgorithmen für Bilder auszuwerten. In dieser Arbeit wird er dafür verwendet, um eine Aussage über die Ähnlichkeit zweier Bilder treffen zu können. Die Herleitung des PSNR basiert auf [Huynh-Thu2008], die Einheit des PSNR-Wertes ist Dezibel [dB].

Der Algorithmus arbeitet auf den beiden Stimuli S_{ist} und S_{soll} und den zugehörigen 2D-RGB-Bildern $i_{\text{ist}}(x, y)$ und $i_{\text{soll}}(x, y)$, welche eine identische Auflösung $m \times n$ mit $m, n \in \mathbb{N}^+$, $x \in \{0, 1, ..., m\}$, $y \in \{0, 1, ..., n\}$ und eine identische Menge an Kanälen $C \in \{r, b, g\}$ besitzen. Zunächst wird die mittlere quadratische Abweichung (englisch: Mean Squared Error, MSE) berechnet:

$$\text{MSE}(i_{\text{ist}}, i_{\text{soll}}) = \frac{1}{|C| \cdot m \cdot n} \sum_{e \in C} \sum_{k=0}^{m-1} \sum_{l=0}^{n-1} (i_{\text{ist},\, c}(k, l) - i_{\text{soll},\, c}(k, l))^2$$

Aus dem MSE und dem maximalen Kanalwert der Bilder $s_{\max}$ wird der PSNR-Wert folgendermaßen berechnet:

$$\text{PSNR}(i_{\text{ist}}, i_{\text{soll}}) = 10 \cdot \log_{10} \left(\frac{s_{\max}^2}{\text{MSE}(i_{\text{ist}}, i_{\text{soll}})} \right)$$

Bei 8-Bit-Kanälen erhält man für $s_{\max} = 2^8 - 1 = 255$. Der MSE besitzt einen Wertebereich von $W_{\text{MSE}} = [0; s_{\max}^2]$. Dadurch ist sichergestellt, dass das Argument des Logarithmus bei der Berechnung des PSNR-Wertes immer größer oder gleich 0 ist. Für den Spezialfall, dass i_{ist} und i_{soll} identisch sind ergibt sich ein Wert von MSE = 0, was dazu führt, dass bei der Berechnung des PSNR durch 0 geteilt werden muss. Tritt dieser Fall auf, so wird definiert, dass der PSNR Wert 0 dB ist. Dadurch ergibt sich für den PSNR ein Wertebereich von $W_{\text{PSNR}} = [0; \infty[$. Höhere PSNR-Werte bedeuten, dass die Ähnlichkeit zwischen den Bildern größer ist. Bei Bildern mit 8-Bit-Kanälen werden PSNR-Werte über 30 dB als annähernd identisch betrachtet, Werte über 20 dB noch als ähnlich. Bei der Anwendung in dieser Arbeit mit der am Greifer montierten Eye-in-Hand Kamera hat sich herausgestellt, dass ein Schwellwert von PSNR = 24 dB die besten Ergebnisse liefert. Bei kleineren Werten werden die Bilder als verschieden klassifiziert, bei größeren als ähnlich.

Die Ähnlichkeitsfunktion für zwei 2D-RGB-Bilder S_{ist} und S_{soll} sieht damit folgendermaßen aus:

$$\texttt{isSimilar}(S_{\text{ist}}, S_{\text{soll}}) = \begin{cases} \texttt{FALSE} & \text{, wenn } 0 \text{ dB} < \text{PSNR}(i_{\text{ist}}, i_{\text{soll}}) < 24 \text{ dB} \\ \texttt{TRUE} & \text{, sonst} \end{cases}$$

Greiferöffnungswinkel eines Backengreifers

Analog zu 2D-RGB-Kamerabildern können auch die Öffnungswinkel der an den Robotern montierten Backengreifer als Sensorwerte verwendet werden. Diese dienen dazu, um indirekt die Größe der gegriffenen Objekte messen zu können, und abhängig davon entscheiden zu können, ob die aktuell gegriffenen Objekte ähnlich zu den vorher gegriffenen Objekten sind. Dabei existieren zwei Werte. Zum einen gibt es den *Greiferwert* v, welcher für jedes Greifermodell unterschiedlich sein kann, und zum anderen den *prozentualen Greiferöffnungswinkel* p, welcher sich aus v berechnen lässt und angibt, zu wie viel Prozent der Greifer geöffnet ist.

Ein Backengreifermodell besitzt einen festen Greiferwert v_{closed}, den er im vollständig geschlossenen Zustand hat, und einen Greiferwert v_{open} im geöffneten Zustand. Es besteht ein linearer Zusammenhang zwischen dem Greiferwert und dem prozentualen Greiferöffnungswinkel. Der prozentualen Greiferöffnungswinkel p kann aus dem aktuell gemessenen Greiferwert v und den bekannten Greiferwerten für den geschlossenen und geöffneten Zustand wie folgt berechnet werden:

$$p(v) = \frac{v - v_{\text{closed}}}{v_{\text{open}} - v_{\text{closed}}} \cdot 100\%$$

Zur Messung der Ähnlichkeit zweier Greiferwerte eines Greifers v_{ist} und v_{soll} werden diese in die prozentualen Greiferöffnungswinkel p_{ist} und p_{soll} umgerechnet und die Differenz aus beiden gebildet. Je kleiner diese Differenz ist, desto ähnlicher sind sich beide Greiferwerte. Bei den in dieser Arbeit verwendeten Backengreifern hat sich ein Wert von $\epsilon = 2.5\%$ als am besten geeignet herausgestellt, um ähnliche von verschiedenen Werten zu unterscheiden. Dabei werden Differenzen kleiner 2.5% als ähnlich angesehen.

Für zwei zu vergleichende Stimuli S_{ist} und S_{soll} mit den zugehörigen Greiferwerten v_{ist} und v_{soll} sieht die Ähnlichkeitsfunktion folgendermaßen aus:

$$\texttt{isSimilar}(S_{\text{ist}}, S_{\text{soll}}) = \begin{cases} \texttt{FALSE} & \text{, wenn } |p(v_{\text{ist}}) - p(v_{\text{soll}})| > 2.5\% \\ \texttt{TRUE} & \text{, sonst} \end{cases}$$

Die in den letzten beiden Abschnitten gezeigten Beispiele für Sensorarten und wie diese auf Ähnlichkeit verglichen werden können sollen einen kleinen Einblick in die Vielfalt unterschiedlicher Sensoren geben, welche für die in dieser Arbeit vorgestellten sensorbasierten Kontrollstrukturen verwendet werden können. Neue Sensoren, welche für die kinästhetische, sensorbasierte Roboterprogrammierung verwendet werden sollen, müssen dabei lediglich eine Funktion implementieren, welche es erlaubt, die Sensorwerte auf Ähnlichkeit hin zu vergleichen.

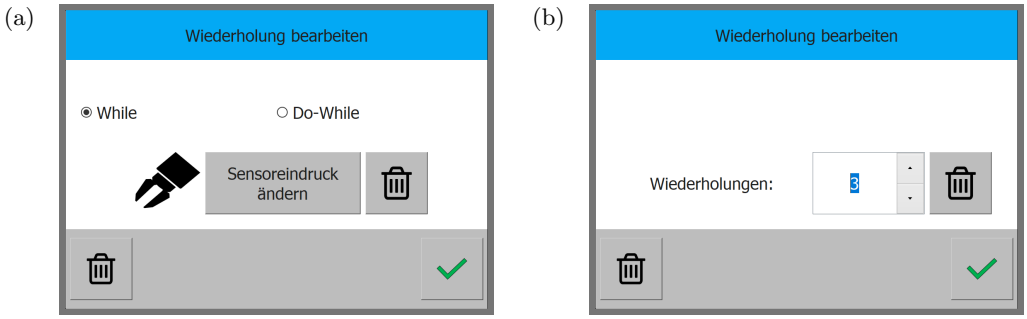


Abbildung 6.1: Die Menüs zum Konfigurieren einer Sensorschleife (a) und einer Zählschleife (b).

6.2 Sensorbasierte Kontrollstrukturen

Um auf Umwelteinflüsse reagieren zu können werden die sensorbasierten Kontrollstrukturen für kinästhetisch demonstrierte Roboterprogramme eingeführt. Zentraler Bestandteil dieser Kontrollstrukturen sind die in Abschnitt 6.1 vorgestellten vergleichbaren Sensorwerte. Mit deren Hilfe können je nach erkannter Szene bzw. je nach erkanntem Objekt Programmteile ausgeführt oder übersprungen werden. Dabei werden als Grundlage Kontrollstrukturen aus prozeduralen Programmiersprachen verwendet und deren Konzept auf die kinästhetische Roboterprogrammierung angepasst.

In dieser Arbeit werden Konzepte für sensorbasierte Kontrollstrukturen in Form von Schleifen und Verzweigungen ausgearbeitet und evaluiert. Bei den Schleifen werden in Abschnitt 6.2.1 sowohl While-, als auch Do-While-Schleifen dargestellt, bei den Verzweigungen in Abschnitt 6.2.2 Switch-Anweisungen.

6.2.1 Schleifen

In prozeduralen Programmiersprachen gibt es Schleifen in der Regel sowohl in Form von While-, als auch Do-While-Schleifen. Beiden Arten ist gemein, dass sie sowohl eine Schleifenbedingung als auch einen Schleifenrumpf besitzen. Der Unterschied liegt darin, an welcher Stelle die Schleifenbedingung ausgewertet wird. Bei der While-Schleife wird die Bedingung vor dem Durchlaufen des Schleifenrumpfes ausgewertet und entschieden, ob dieser ausgeführt werden soll, bei der Do-While-Schleife hingegen wird nach dem Schleifendurchlauf überprüft, ob der Schleifenrumpf erneut ausgeführt werden soll. Rümpfe von Do-While-Schleifen werden demnach mindestens einmal ausgeführt, Rümpfe von While-Schleifen können auch übersprungen werden.

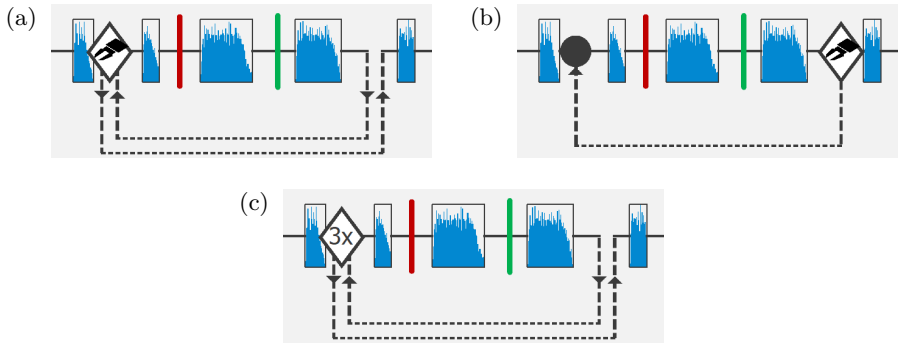


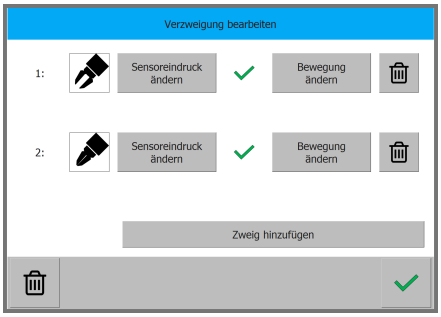
Abbildung 6.2: Bildschirmaufnahme der Zeitleistendarstellungen für While (a), Do-While (b) und For (c) Schleifen. Die sensorbasierten Schleifen haben jeweils einen Greiferöffnungswinkel als Soll-Sensorwert.

Das Konzept der prozeduralen While- und Do-While-Schleifen wurde im Rahmen dieser Arbeit verwendet und als sensorbasierte Schleife auf die kinästhetische Roboterprogrammierung übertragen. Dabei werden als Bedingungen für die Schleifen die in Abschnitt 6.1 beschriebenen Sensorwerte verwendet. In der Programmierphase werden für alle Bedingungen Soll-Sensorwerte aufgezeichnet, welche dann in der Ausführungsphase mit den aktuellen Ist-Sensorwerte verglichen werden. Sind beide Sensorwerte ähnlich, so wird der zugehörige Schleifenrumpf ausgeführt. Zur Berechnung der Ähnlichkeit beider Sensorwerte wird die Funktion `isSimilar` aus Abschnitt 6.1.1 verwendet. Als Schleifenrumpf dient, ähnlich wie bei prozeduralen Programmiersprachen, auch bei der kinästhetischen Roboterprogrammierung wieder ein vollständiges, kinästhetisch demonstriertes Roboterprogramm.

Zusätzlich zu den Sensorschleifen wurde im Rahmen dieser Arbeit noch eine Zählschleife realisiert, angelehnt an die For-Schleifen in prozeduralen Programmiersprachen. Diese kann immer dann verwendet werden, wenn vorher bekannt ist, wie oft ein Schleifenrumpf ausgeführt werden soll. Dabei wird in der Programmierphase ein Zähler festgelegt, welcher die Anzahl an Schleifendurchläufen beschreibt.

Im Rahmen des entstandenen Programmiersystems ist es möglich, Schleifen mit Hilfe der graphischen Benutzungsoberfläche zu vorher kinästhetisch demonstrierten Roboterprogrammen hinzuzufügen. Zum Hinzufügen einer Schleife muss der Teil des bestehenden Roboterprogramms in der Zeitleiste markiert werden, der den Schleifenrumpf darstellen soll. Anschließend kann über die Schaltfläche *Schleife hinzufügen* in der Kontrollleiste eine Schleife um diesen markierten Bereich definiert werden. In dem sich öffnenden Untermenü muss festgelegt werden, ob es sich um eine Sensor- oder Zählschleife handelt, und ob es sich um eine While- oder Do-While-Schleife handelt. Zum Schluss muss bei Sensorschleifen der Soll-Sensorwert über ein Konfigurationsmenü aufgezeichnet werden und bei Zählschleifen die Anzahl an Schleifendurchläufen festgelegt werden. Eine Bildschirmaufnahme des Konfigurationsmenüs für Sensorschleifen ist in

(a)



(b)

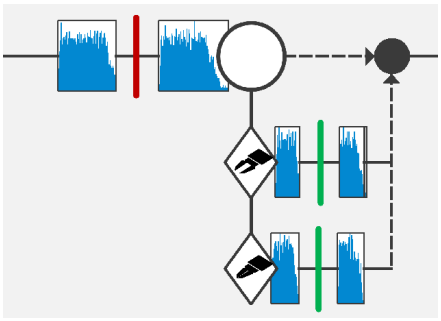


Abbildung 6.3: Das Konfigurationsmenü einer sensorbasierten Verzweigung (a) und die Zeitleistendarstellung dazu (b).

Abbildung 6.1a und für Zählschleifen in Abbildung 6.1b dargestellt. Zeitleistendarstellungen von Roboterprogrammen mit eingefügten Sensorschleifen in Form einer While- oder Do-While-Schleife bzw. mit einer eingefügten Zählschleife sind in Abbildung 6.2 dargestellt.

6.2.2 Verzweigungen

Analog zu den Schleifen werden die Verzweigungen in dieser Arbeit in Anlehnung an prozedurale Programmiersprachen konzeptioniert. Dabei werden als Grundlage Switch-Anweisungen verwendet. Eine Switch-Anweisung in prozeduralen Programmiersprachen ist so aufgebaut, dass diese beliebig viele Zweige besitzt. Jedem Zweig ist ein Soll-Wert für den Vergleich mit dem Ist-Wert und ein Zweigrumpf zugeordnet. Dieser Rumpf enthält Programmcode, welcher ausgeführt wird, wenn der Soll- und der Ist-Wert übereinstimmen. Optional kann ein Zweig ohne Soll-Wert, der sogenannte Default-Zweig, existieren, der ausgeführt wird, wenn keiner der Soll-Werte mit dem aktuellen Ist-Wert übereinstimmt.

Dieses Konzept wurde im Rahmen dieser Arbeit verwendet und als sensorbasierte Verzweigung auf die kinästhetische Roboterprogrammierung angepasst. Dabei werden - ähnlich wie bei den Schleifen - als Soll- und Ist-Werte die in Abschnitt 6.1 vorgestellten Sensorwerte verwendet. In der Programmierphase werden für alle Zweige Soll-Sensorwerte aufgezeichnet, welche in der Ausführungsphase mit dem aktuellen Ist-Sensorwert verglichen werden. Der Zweig, dessen Soll-Sensorwert am ähnlichsten zum Ist-Sensorwert ist, wird ausgewählt und dessen zugehöriger Zweigrumpf wird ausgeführt. Zur Berechnung des ähnlichsten Sensorwertpaares wird die Funktion `getMostSimilarStimulus` aus Abschnitt 6.1.1 verwendet. Existiert kein Soll-Sensorwert, der ähnlich zu dem Ist-Sensorwert ist, so wird der Default-Zweig der Verzweigung ausgeführt. Existiert auch kein Default-Zweig, so wird die Verzweigung übersprungen. Als Zweigrumpf dient wieder ein kinästhetisch demonstrierten Roboterprogramm

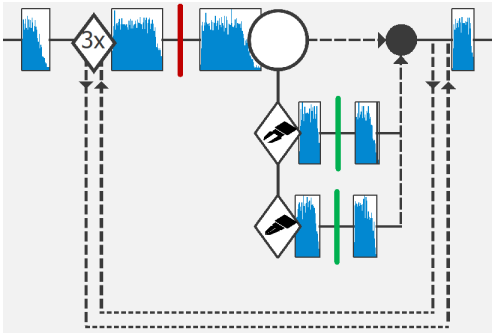


Abbildung 6.4: Zeitleistendarstellung geschachtelter Kontrollstrukturen. In diesem Beispiel ist in einer Zählschleife, welche drei Mal ausgeführt wird, eine Verzweigung mit zwei Zweigen geschachtelt.

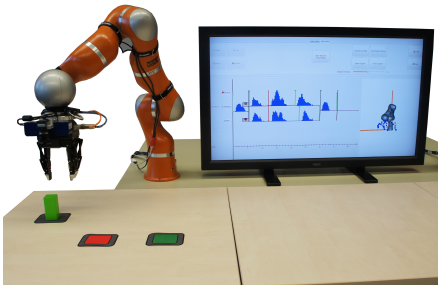
Zum Einfügen einer Verzweigung in ein vorhandenes kinästhetisch demonstriertes Roboterprogramm muss in der Zeitleiste die Stelle markiert werden, an der die Verzweigung eingefügt werden soll. Anschließend kann über die Schaltfläche *Verzweigung hinzufügen* in der Kontrollleiste eine Verzweigung an dieser Stelle des Roboterprogramms eingefügt werden. In dem sich öffnenden Untermenü muss anschließend ein Zweig für jeden Soll-Sensorwert, den das Programm unterscheiden können soll, hinzugefügt werden. Zu diesen Zweigen müssen über die Schaltfläche *Sensorwert ändern* die Soll-Sensorwerte hinzugefügt werden, und über die Schaltfläche *Bewegung ändern* die Roboterprogramme, welche zu diesem Sensorwerten gehören, kinästhetisch aufgezeichnet werden. Eine Bildschirmaufnahme des Untermenüs ist in Abbildung 6.3a zu sehen. Eine Zeitleistendarstellung von einem Roboterprogramm mit eingefügter Verzweigung ist in Abbildung 6.3b dargestellt.

6.2.3 Schachtelung von Kontrollstrukturen

Im Zusammenhang mit Kontrollstrukturen ist es wichtig, dass diese nicht nur linear hintereinander ausgeführt werden können, sondern auch ineinander geschachtelt werden können. Aus diesem Grund wurde das Konzept der sensorbasierten Kontrollstrukturen für kinästhetisch demonstrierte Roboterprogramme so erweitert, dass sowohl Schleifen, als auch Verzweigungen beliebig ineinander geschachtelt werden können.

Dadurch, dass die Schleifen- und Zweigrümpfe eigenständige Roboterprogramme sind, ist es möglich in diesen Roboterprogrammen auch wieder Schleifen und Verzweigungen einzufügen. Formal definiert wurde dies bereits in der Backus-Naur-Form in Kapitel 4.1, welche die kinästhetisch demonstrierten Roboterprogramme dieser Arbeit beschreiben kann. Somit lassen sich auch komplexere Aufgaben, welche das Zusammenspiel mehrerer Kontrollstrukturen benötigen, mit Hilfe der kinästhetischen Roboterprogrammierung realisieren.

(a)



(b)

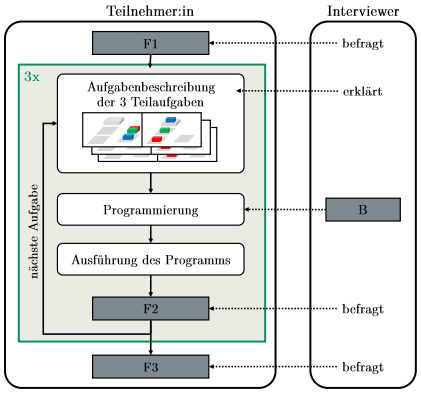


Abbildung 6.5: Abbildung (a) zeigt den Aufbau für die Evaluation. Auf dem Monitor ist eine frühe Version des Programmiersystems zu sehen, welche zur Evaluation der sensorbasierten Kontrollstrukturen verwendet wurde. Die Zielbereiche sind rot und grün markiert. Abbildung (b) zeigt den Ablauf eines Durchgangs der Evaluation.

Ein Beispiel für diese Art Aufgaben ist eine Sortieraufgabe, bei der Objekte von einem Förderband immer an der gleichen Stelle aufgegriffen werden sollen und der Größe nach sortiert werden sollen. Ein beispielhaftes Roboterprogramm welches diese Sortieraufgabe löst und eine schematische Darstellung der Aufgabe ist in Abbildung 6.4 zu sehen.

6.3 Evaluation der Kontrollstrukturen

Um eine Aussage über die Intuitivität der sensorbasierten Kontrollstrukturen treffen zu können wurde vom 17.05.2016 bis 25.05.2016 eine Benutzungsstudie mit insgesamt 20 Teilnehmer:innen im Roboterlabor der Universität Bayreuth durchgeführt. Dabei sollte herausgefunden werden, wie intuitiv die sensorbasierten Kontrollstrukturen in Kombination mit den kinästhetisch demonstrierten Roboterprogrammen sind.

In Abschnitt 6.3.1 wird das Studiendesign erläutert und anschließend werden in Abschnitt 6.3.2 die Ergebnisse der Evaluation im Hinblick auf Intuitivität vorgestellt. Diese Evaluation wurde bereits in [Riedl2017] veröffentlicht.

6.3.1 Studiendesign

Zur Evaluation der Kontrollstrukturen und des Programmiersystems wurde die damalige prototypische Implementierung verwendet. Jede:r Studienteilnehmer:in musste drei Teilaufgaben (TA) mit Hilfe des Programmiersystems programmieren.

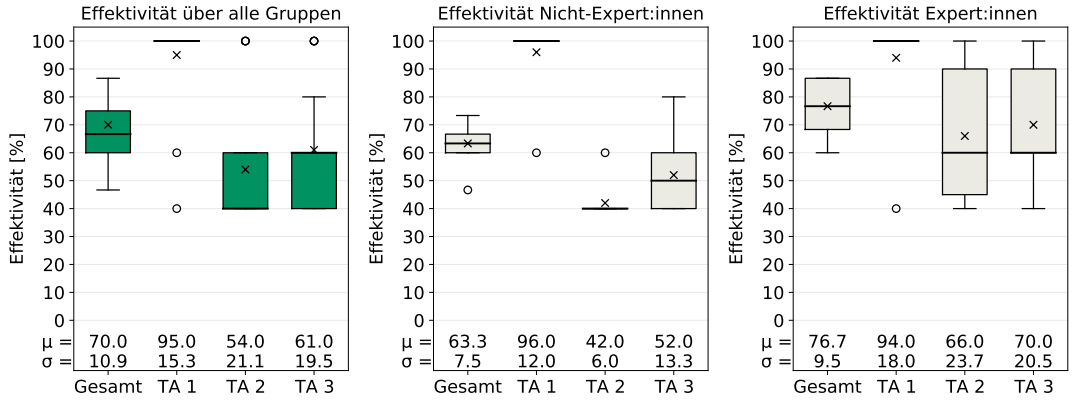


Abbildung 6.6: Ergebnisse für die Vollständigkeit und Genauigkeit zur Messung der Effektivität der sensorbasierten Kontrollstrukturen aufgeteilt nach Gruppen für die Gesamtaufgabe und die Teilaufgaben TA1 bis TA3.

Bei der ersten Teilaufgabe handelt es sich um eine Pick-and-Place Aufgabe um den Teilnehmer:innen ein Gefühl für den Roboter und das Programmiersystem an sich zu geben. Die Teilnehmer:innen mussten den Roboter so programmieren, dass dieser den Quader aufgreift und in den markierten Zielbereich ablegt. In der zweiten Teilaufgabe mussten die Teilnehmer:innen eine sensorbasierte Schleife programmieren. Dabei sollte eine Bewegung demonstriert werden, welche Objekte von der Aufgreifposition aufnimmt und in eine Kiste wirft, so lange rote Würfel erkannt werden. Ein grüner Würfel wurde als Abbruchkriterium verwendet. Als letzte Teilaufgabe sollten die Teilnehmer:innen eine Verzweigung programmieren, welche dafür sorgt, dass der Roboter je nach Farbe der Quader diese auf unterschiedliche Zielgebiete ablegt. Der Aufbau an dem die Teilnehmer:innen die Teilaufgaben programmieren sollten ist in Abbildung 6.5a dargestellt.

Ähnlich zu der Gesamtevaluation des finalen Prototypen dieser Arbeit in Kapitel 8 mussten auch bei der Evaluation der sensorbasierten Kontrollstrukturen drei Fragebögen durch die Teilnehmer:innen ausgefüllt werden und ein Beobachtungsbogen durch den Studienleiter. Der Ablauf dafür ist in Abbildung 6.5b dargestellt. Eine Erklärung der verwendeten Werkzeuge zur Messung der Teilaspekte der Intuitivität ist in Abschnitt 2.3.2 zu finden. Fragebogen F1 musste vor Beginn der Studie ausgefüllt werden und fragte die Vorkenntnisse im Hinblick auf Roboterprogrammierung ab. Fragebogen F2 musste nach jeder Aufgabe erneut ausgefüllt werden und dient dazu, die mentale Beanspruchung der Teilnehmer:innen und damit die mentale Effizienz des Programmiersystems mit Hilfe der SEA-Skala [Eilers1986] zu erfassen. Parallel dazu füllte der Studienleiter den Beobachtungsbogen B aus, um die Effektivität der Teilnehmer:innen in Form von Vollständigkeit und Genauigkeit bei der Bearbeitung der Aufgaben zu dokumentieren. Abschließend mussten die Teilnehmer:innen noch Fragebogen F3 ausfüllen, mit welchem die Zufriedenheit der Teilnehmer mit dem Programmiersystem gemessen wurde. Hierfür wurde der QUESI-Fragebogen [Naumann2010] verwendet.

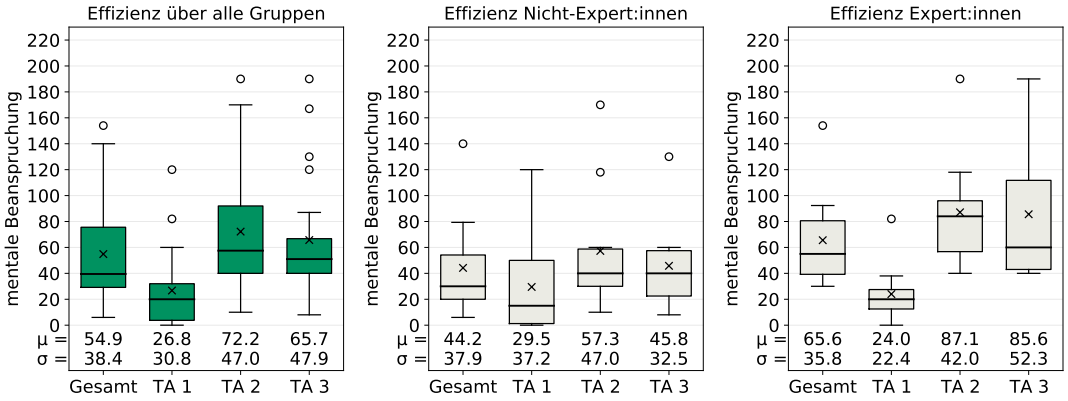


Abbildung 6.7: Ergebnisse für die mentale Beanspruchung zur Messung der mentalen Effizienz der sensorbasierten Kontrollstrukturen aufgeteilt nach Gruppen für die Gesamtaufgabe und die Teilaufgaben TA1 bis TA3.

6.3.2 Ergebnisse

Die 20 Teilnehmer:innen der Studie waren zwischen 20 und 36 Jahre alt mit einem Mittelwert vom $\mu = 25.5$ Jahren und einer Standardabweichung von $\sigma = 4.3$ Jahren. Sie wurden aufgeteilt in zwei Gruppen, Expert:innen und Nicht-Expert:innen, wobei Expert:innen definiert wurden als Personen, die bereits mit Robotern gearbeitet haben und Nicht-Expert:innen als Personen, die noch nie etwas mit Robotern zu tun hatten. Mit dieser Definition bestanden beide Gruppen aus je 10 Teilnehmer:innen.

Da in Kapitel 2.3 bereits herausgearbeitet wurde, dass die Intuitivität eines Systems definiert wird als Kombination aus Effektivität, mentaler Effizienz und Zufriedenheit, wurden in dieser Studie mit den Fragebögen und dem Beobachtungsbogen diese Werte erhoben. Für die *Effektivität* wurde mit Hilfe des Beobachtungsbogens B ausgewertet, wie zielstrebig und genau die jeweilige Aufgabe von den Teilnehmer:innen gelöst wurde. Insgesamt ergibt sich so je Aufgabe ein Wert für die Effektivität zwischen 0% und 100%, wobei höhere Werte ein effektiveres Vorgehen der Teilnehmer:innen repräsentieren. Die Ergebnisse für die Effektivität der beiden Gruppen ist in Abbildung 6.6 dargestellt. Für die Effektivität ergibt sich über alle Gruppen insgesamt ein Wert von $\mu = 70\%$, wobei die Nicht-Expert:innen das System mit $\mu = 63.3\%$ nicht ganz so effektiv bedient haben wie die Expert:innen mit $\mu = 76.7\%$. Dies kann auf die vergleichsweise hohe Steigerung des Komplexitätsgrades beim Schritt von TA1 zu TA2 zurückgeführt werden, der für die Nicht-Expert:innen potentiell schwierig war. Dies zeigt, dass vor allem für die Teilaufgaben 2 und 3, also für die sensorbasierten Kontrollstrukturen, die Bedienung des Programmiersystems noch verbessert werden musste.

Für die *mentale Effizienz* wurde mit Hilfe des Fragebogens F2 ausgewertet, wie anstrengend die jeweilige Aufgabe von den Teilnehmer:innen bewertet wurde. Die Werte auf der verwendeten Skala zur Erfassung subjektiv erlebter Anstrengung (SEA-Skala) reichten von 0 bis 220, wobei niedrigere Werte eine geringere mentale Beanspruchung

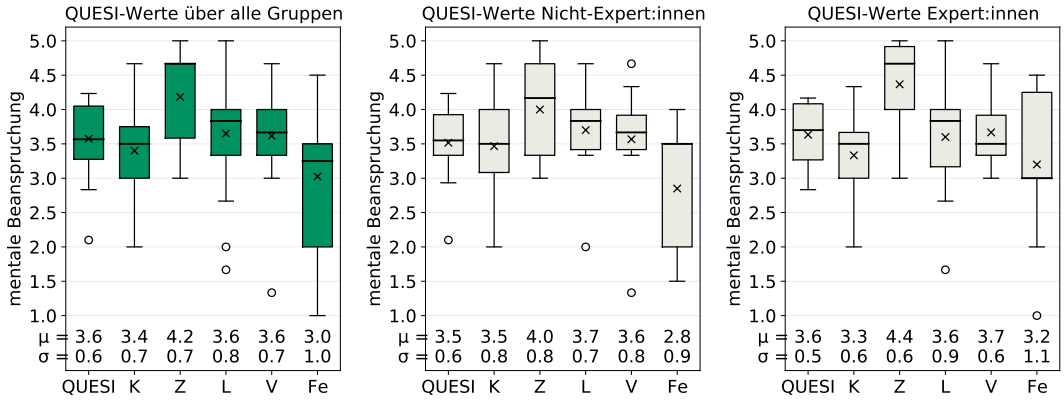


Abbildung 6.8: Ergebnisse für die subjektiven Konsequenzen intuitiver Benutzung (QUESI) zur Messung der Zufriedenheit mit den sensorbasierten Kontrollstrukturen für alle Gruppen. Die Werte sind aufgeteilt in den Gesamtwerte (QUESI) und die einzelnen Subskalen des QUESI. Diese sind: wahrgenommene kognitive Beanspruchung (K), wahrgenommene Zielerreichung (Z), wahrgenommener Lernaufwand (L), Vertrautheit/Vorwissen (V) und wahrgenommene Fehlerrate (Fe).

widerspiegeln. Die Ergebnisse für die mentale Beanspruchung ist in Abbildung 6.7 dargestellt. Der Mittelwert über alle Gruppen und Aufgaben lag bei $\mu = 54.9$. Interessant ist dabei, dass die Gruppe der Nicht-Expert:innen das Programmiersystem mit einem Wert von $\mu = 44.9$ als weniger anstrengend bewertet hat als die Gruppe der Expert:innen mit einem Wert von $\mu = 65.6$. Die relativ große Standardabweichung über alle Teilnehmer von $\sigma = 38.4$ zeigt, dass die Beanspruchung unterschiedlich wahrgenommen wird. Im Mittel ergibt sich für die mentale Beanspruchung aus der wörtlichen Übersetzung der SEA-Skala eine Bewertung von „ein bisschen bis etwas anstrengend“.

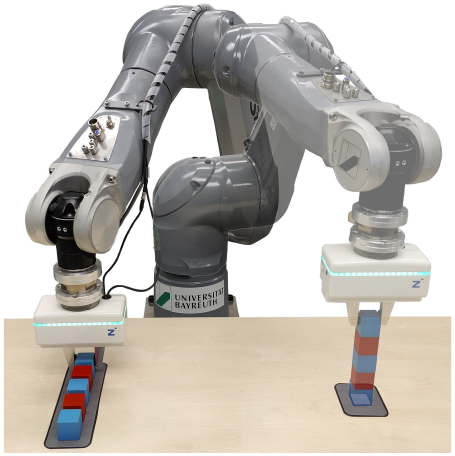
Als letzter Wert wurde die *Zufriedenheit* mit Hilfe des Fragebogens F3 erhoben. Dabei wurde mit dem standardisierten QUESI-Fragebogen in fünf Kategorien auf einer Skala von 1 bis 5 die Zufriedenheit gemessen, wobei höhere Werte eine höhere Zufriedenheit bedeuten. Die Ergebnisse in den fünf Kategorien sind in Abbildung 6.8 dargestellt. Der Mittelwert über alle Kategorien und Gruppen liegt bei $\mu = 3.6$, wobei die Bewertung der Nicht-Expert:innen mit $\mu = 3.5$ und die Bewertung der Expert:innen mit $\mu = 3.6$ ähnlich gut ist. Wie in Kapitel 2.3 bereits beschrieben müssen QUESI-Werte von Systemen mit anderen Systemen verglichen werden, um eine Aussage über die Zufriedenheit treffen zu können. In [Orendt2016] wurde ein Programmieren durch Vormachen System mit dem gleichen Fragebogen evaluiert. Dieses hat einen QUESI-Wert von 3.6 erzielt. Es kann also festgehalten werden, dass der damalige Stand des Programmiersystems ähnlich zufriedenstellend für die Benutzer:innen ist wie das System in [Orendt2016].

Diese Evaluation zeigte, dass bereits der frühe Prototyp als intuitiv angesehen werden kann, auch wenn gerade im Bereich der Effektivität noch Verbesserungspotential besteht. Bei der Weiterentwicklung des Prototypen wurden diese Ergebnisse berücksichtigt. Die Evaluation des Gesamtsystems in Kapitel 8 zeigt, dass dadurch eine Verbesserung in allen Kategorien erreicht wurde.

6.4 Erweiterung sensorbasierter Schleifen: Das Schleifen-Inkrement

Wie im letzten Abschnitt zu sehen, ist die kinästhetische Programmierung von Robotersystemen mit Hilfe von sensorbasierten Kontrollstrukturen bereits in einer ersten Version als intuitiv einzustufen. Durch die Kontrollstrukturen ist es möglich, auf die Umgebung in Form von Sensorwerten zu reagieren und so zu entscheiden, ob eine Bewegung noch einmal ausgeführt werden soll. Dies hat aber aufgrund der kinästhetischen Programmierung den Nachteil, dass immer nur exakt die gleiche Bewegung ausgeführt werden kann. Bereits bei kleinen Änderungen in jedem Schleifendurchlauf muss diese Bewegung komplett neu programmiert werden, so dass Stapel- bzw. Palettieraufgaben nur schwer mit dem bisherigen Umfang des Programmiersystems umgesetzt werden können.

(a)



(b)

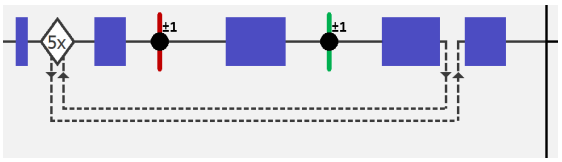


Abbildung 6.9: (a): Beispielaufgabe bestehend aus depalettieren (linker Roboter) und stapeln (rechter Roboter), welche mit Schleifen-Inkrementen programmiert werden kann, indem nur eine Pick-and-Place Trajektorie demonstriert wird.
(b): Zeitleistendarstellung der oben dargestellten Aufgabe im Programmiersystem.

Mit dem Schleifen-Inkrement wird in dieser Arbeit ein neues Konzept zur Erweiterung der kinästhetischen Roboterprogrammierung vorgestellt, um regelmäßige Positionsänderungen programmieren zu können, ohne jede einzelne Wieder-

holung der Bewegung explizit demonstrieren zu müssen. Diese Erweiterung ermöglicht es nun auch Stapeln, Palettieren und ähnliche Aufgaben zu programmieren, indem nur die Bewegung des ersten Schleifendurchlaufs demonstriert und anschließend mit Hilfe des Inkrements die kartesische Bahnveränderung an markanten Positionen der Bewegung definiert wird. Ein beispielhaftes Depalettier- und Stapelszenario, dass mit Hilfe des Schleifen-Inkrements programmiert werden kann, ohne dass die fünf einzelnen Pick-and-Place Bewegungen demonstriert werden müssen, ist in Abbildung 6.9a dargestellt.

Abschnitt 6.4.1 erläutert das Konzept des Schleifen-Inkrements näher, bevor anschließend in Abschnitt 6.4.2 verschiedenen Methoden zum Anpassen der Bewegung zu den durch das Schleifeninkrement neu festgelegten markanten Punkten aufgezeigt wird. Abschließend wird in Abschnitt 6.4.3 darauf eingegangen, welche der vorher aufgezeigten Methoden am Besten für die in dieser Arbeit verwendete kinästhetische Programmierung geeignet ist. Das Konzept des Schleifen-Inkrements wurde bereits in [Riedl2021] veröffentlicht.

6.4.1 Konzept

Allgemein werden Schleifen-Inkremente so angewendet, dass zunächst die Grundbewegung der auszuführenden Aufgabe kinästhetisch programmiert wird. Bei dem Szenario aus Abbildung 6.9 ist das die Pick-and-Place Bewegung vom ersten Würfel bei den Aufnahmepositionen links hin zur Ablageposition rechts. Anschließend legen die Benutzer:innen den sich wiederholenden Teil der Bewegung wie bei den übrigen Schleifenarten fest (Abschnitt 6.2.1). Nun kann das Schleifen-Inkrement verwendet werden, um in dem definierten Schleifenrumpf Konfigurationen zu definieren, welche nach jeder Iteration um einen definierten kartesischen Betrag - das Schleifen-Inkrement - verschoben werden sollen. Diese Konfigurationen innerhalb des Schleifenrumpfes, welchen ein Schleifen-Inkrement zugeordnet ist, werden Inkrement-Positionen genannt. Die Funktionsweise des Schleifen-Inkrements ist in Abbildung 6.10 dargestellt.

Bevor die Inkremente erläutert werden können, müssen einige Variablen definiert werden, die für die Berechnung benötigt werden. Für jede Inkrement-Konfiguration wird die initiale Pose P_0 benötigt und die Pose nach einmaligem anwenden des Schleifen-Inkrements P_1. Das Inkrement, also die Transformation zwischen beiden Posen wird als D bezeichnet. P_0, P_1 und D sind jeweils homogene 4×4 Matrizen mit $P_1 = P_0 \cdot D$. Mit Hilfe dieser Gleichung kann D als $D = P_0^{-1} \cdot P_1$ berechnet werden. Um D zu definieren, ist also nur P_0 und P_1 notwendig.

In dieser Arbeit werden zwei Möglichkeiten zum festlegen des Inkrements D durch die Benutzer:innen vorgeschlagen. Die erste Möglichkeit, auch als *Nicht-Expert:innen-Methode* bezeichnet, erlaubt es den Benutzer:innen den Roboter manuell zu P_1 zu führen. Um den Benutzer:innen das manuelle Führen zu erlauben wird der Roboter nach Festlegen der Inkrement-Konfiguration in der Zeitleiste automatisch zu P_0 verfahren und in den Handführen-Modus geschaltet. Anschließend können die Benut-

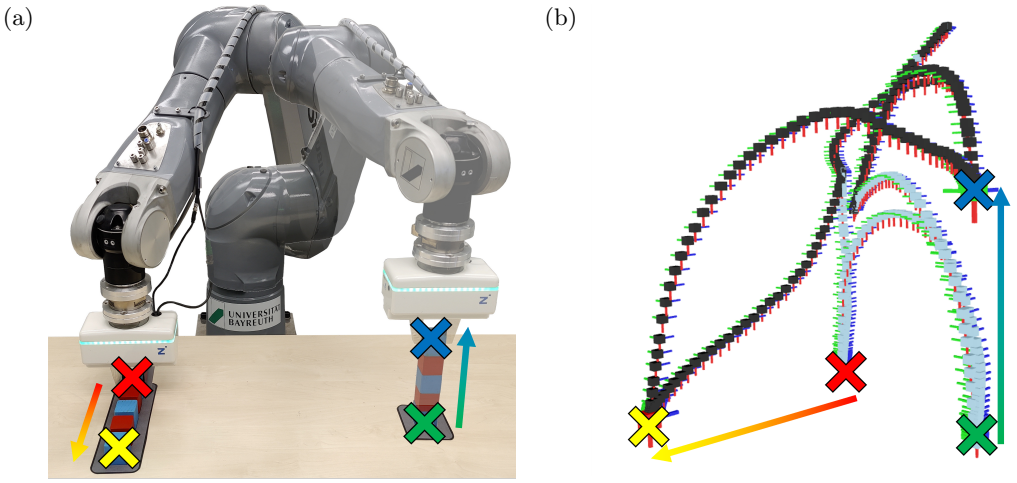
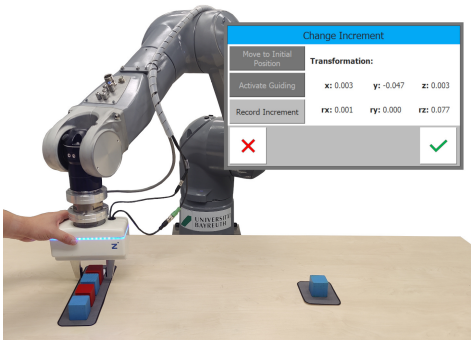


Abbildung 6.10: Funktionsweise des Schleifen-Inkrements. In (a) ist die Beispielaufgabe mit Stapeln und Palettieren dargestellt, in (b) zwei Trajektorien aus dem Simulationsfenster des Programmiersystems. Die hellblaue Trajektorie wurde kinästhetisch programmiert und soll vom Roboter beim ersten Durchlauf der Schleife ausgeführt werden. Die schwarze Trajektorie wird aus den Inkrementen an der roten und grünen Position und der hellblauen Trajektorie berechnet und soll beim fünften Schleifendurchlauf ausgeführt werden. Korrespondierende Konfigurationen sind in (a) und (b) mit der gleichen Farbe markiert.

zer:innen den Roboter zu P_1 führen. Während dieser Aktion zeigt das Programmiersystem in Echtzeit das Inkrement D als kartesische Translation und Euler-Winkel in der Benutzungsoberfläche an. Dies ist in Abbildung 6.11a dargestellt. Wenn die Benutzer:innen zufrieden mit der Transformation sind, kann diese abgespeichert werden. Nachträglich ist es noch möglich über das Expert:innen-Menü das Inkrement D händisch anzupassen.

Die zweite Möglichkeit, auch als *Expert:innen-Methode* bezeichnet, erlaubt es den Benutzer:innen D direkt über die Benutzungsoberfläche einzugeben. Dabei wird die Translation als kartesische Translation und die Rotation als Euler-Winkel angegeben. Die Benutzer:innen können dadurch die Translation als Verschiebung entlang der lokalen x-, y- und z-Achsen angeben und die Rotation um die lokalen x-, y- und z-Achsen. Eine Bildschirmaufnahme davon ist in Abbildung 6.11b abgebildet. Die Nicht-Expert:innen-Methode definiert das Inkrement D implizit, wohingegen bei der Expert:innen-Methode D explizit definiert wird.

(a)



(b)

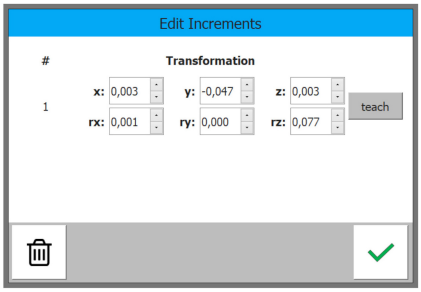


Abbildung 6.11: (a): Foto eine:r Nicht-Expert:in beim definieren eines Inkrements durch manuelles Führen. Zusätzlich ist eine Bildschirmaufnahme der Rückmeldung des Programmiersystems dargestellt. (b): Die Benutzungsschnittstelle für die Expert:innen um ein Inkrement als kartesische Transformation und Euler-Winkel im lokalen Koordinatensystem zu definieren. Beide Graphen stammen aus [Riedl2021].

6.4.2 Methoden zum Anpassen der Bewegung

Nachdem im letzten Abschnitt gezeigt wurde, wie das Inkrement definiert werden kann, wird jetzt näher darauf eingegangen, wie das Anpassen der ursprünglichen Trajektorie zur durch die Inkrement-Konfiguration veränderte Trajektorie funktioniert. Dazu wird zunächst eine Trajektorie T als Sequenz von $n \in \mathbb{N}$ Gelenkkonfigurationen q_t mit $0 \leq t \leq n$ und $q_t \in \mathbb{R}^d$ für Roboter mit $d \in \mathbb{N}$ Achsen definiert. Dadurch, dass eine Trajektorie auch immer eine Zeitdauer zwischen zwei Gelenkkonfigurationen enthält, wird festgelegt, dass diese Zeitdauer zwischen allen Paaren an Gelenkkonfigurationen q_t und q_{t+1} in T isochron sein soll und immer eine feste Anzahl an Millisekunden beträgt.

Je weiter zeitlich entfernt eine Konfiguration von der Inkrement-Konfiguration ist, desto weniger gewichtet wird das Inkrement auf die Konfiguration gerechnet. Zur Festlegung der Gewichtung eines Inkrements für eine Konfiguration werden zunächst zwei Methoden entwickelt. Diese geben an, auf welche Konfigurationen das Inkrement zu 100% angewendet wird. Die erste Methode wendet das Inkrement nur auf die definierten Inkrement-Konfiguration zu 100% (*Punkt-Methode*) an, wohingegen die zweite Methode das Inkrement auch auf alle zur Inkrement-Konfiguration benachbarten Konfigurationen zu 100% anwendet, so lange der Roboter an diesen Konfigurationen still steht (*Intervall-Methode*).

Durch die Intervall-Methode werden alle Konfigurationen identisch angepasst, welche zu einem Bereich gehören, in dem sich der Roboter nicht bewegt. Auch nach dem Anpassen der Bewegung durch das Inkrement steht der Roboter im gleichen Bereich weiterhin still. Bei der Punkt-Methode hingegen werden diese Bereiche ignoriert, so dass sich nach Anpassen der Bewegung der Roboter auch in Bereichen bewegen kann, in denen er vorher still stand.

Um alle Konfigurationen anhand ihrer Entfernung zur nächsten und vorherigen Inkrement-Konfiguration adaptieren zu können müssen erst Anpassungsfunktionen definiert werden, welche die Gewichtung von D für jede Konfiguration innerhalb von T berechnet. Die Anpassungsfunktionen müssen einen Wert zwischen 0 und 1 zurückgeben, der angibt, zu wie viel Prozent das zugehörige Inkrement auf die aktuelle Konfiguration gerechnet werden muss. Eine Anpassungsfunktion ist definiert als

$$f_{\text{adapt}} : i, j, x \to w \text{ mit } i, j, x \in \{0, \ldots, n\} \text{ und } w \in [0; 1].$$

Dabei ist i der Index der Inkrement-Konfiguration, j ist der Index der letzten Konfiguration, auf die das Inkrement eine Auswirkung hat, x ist der Index der aktuellen Konfiguration die mit dem Inkrement verändert werden soll und w beschreibt die Gewichtung für das Inkrement an Inkrement-Konfiguration i für Konfiguration x. Alle Anpassungsfunktionen müssen zwei Eigenschaften erfüllen. Einerseits muss $f_{\text{adapt}}(i, j, i) = 1$ und andererseits $f_{\text{adapt}}(i, j, j) = 0$ gelten. Die erste Bedingung bedeutet, dass die Inkrement-Konfiguration mit dem kompletten Anteil des Inkrements verändert werden soll, wohingegen die zweite Bedingung bedeutet, dass die letzte Konfiguration j keine Veränderung durch das Inkrement an Konfiguration i erfahren darf. Um eine geeignete Anpassungsfunktion zu finden, werden nun drei Funktionen, Linear-, Gauß- und Kosinus-Funktionen in Kombination mit der Punkt- und der Intervall-Methode verglichen.

Zur Definition der Linear-Funktion wird $f(x) = m \cdot x + t$ als Basisformel verwendet. Mit den beiden vorher definierten Eigenschaften die erfüllt werden müssen ergibt sich für die Linear-Funktion Gleichung 6.1.

$$f_{\text{adapt, linear}}(i, j, x) = \frac{1}{i - j}(x - j) \tag{6.1}$$

Für die Gauß-Funktion wird $f(x) = a \cdot \exp\left(-\frac{(x-b)^2}{2c^2}\right)$ als Basisfunktion verwendet. Mit Hilfe der ersten Bedingung erhält man $a = 1$ und $b = i$. Die zweite Bedingung ist für diese Art Formel nicht erfüllbar, da $\forall x \in \mathbb{R} : f(x) > 0$ gilt. Deshalb wird diese Bedingung aufgeweicht zu $f(i, j, j) \approx 0$. Mit dieser angepassten Bedingung kann nun ein passendes c gefunden werden, damit das Ergebnis klein genug ist, aber der Exponent nicht zu klein wird. Sollte der Exponent zu klein werden, so hat $f(x)$ einen

zu hohen Gradienten und die Anpassungsfunktion wird zu steil. Für die Verwendung in dieser Arbeit hat sich herausgestellt, dass $c = \frac{i-j}{d}$ mit $d = 3.5$ den besten Kompromiss aus zweiter Bedingung und nicht zu steilem Gradienten erfüllt. Mit diesen Annahmen ergibt sich für die Gauß-Funktion Gleichung 6.2.

$$f_{\text{adapt, gaussian}}(i, j, x) = \exp\left(-\frac{(x-i)^2}{2 \cdot \left(\frac{i-j}{3.5}\right)^2}\right) \tag{6.2}$$

Als letzte Anpassungsfunktion wird die Kosinus-Funktion hergeleitet. Hierfür wird $f(x) = a \cdot \cos(b + c \cdot x) + d$ als Basisformel verwendet. Zusätzlich zu den beiden Bedingungen die erfüllt werden müssen soll die Funktion einen Wertebereich zwischen 0 und 1 haben. Deshalb muss $a = 0.5$ und $d = 0.5$ gelten. Um die anderen beiden Bedingungen zu erfüllen muss $\cos(b + ci) = 1 \rightarrow b + ci = 0$ und $\cos(b + cj) = 0 \rightarrow b + cj = \pi$ gelten. Dies sorgt dafür, dass $b = -\frac{\pi}{j-i} \cdot i$ und $c = \frac{\pi}{j-i}$ gelten muss. Für die vereinfachte Kosinus-Funktion ergibt sich Gleichung 6.3.

$$f_{\text{adapt, cosine}}(i, j, x) = 0.5 \cdot \cos\left(\frac{\pi}{j-i} \cdot (x-i)\right) + 0.5 \tag{6.3}$$

Mit Hilfe der definierten Anpassungsfunktionen ist es jetzt möglich, die einzelnen Konfigurationen von T anzupassen. Nach jedem Schleifendurchlauf müssen alle Gelenkkonfigurationen $q_t \in T$ neu berechnet werden. Jede Gelenkkonfiguration wird von zwei Inkrementen beeinflusst, Inkrement D_l welches zur Inkrement-Konfiguration q_l links von q_t gehört und Inkrement D_r welches zur Inkrement-Konfiguration q_r rechts von q_t gehört. Mit einer der drei Gleichungen 6.1, 6.2 oder 6.3 können die Gewichte für jedes Inkrement berechnet werden als $w_l = f_{\text{blend, } b}(l, r, c)$ und $w_r = f_{\text{blend, } b}(r, l, c)$ mit $b \in \{\text{linear}, \text{gaussian}, \text{cosine}\}$.

Als nächster Schritt werden alle D_x mit $x \in \{l, r\}$ aufgeteilt in ihre kartesische Translation t_x und Euler-Winkel r_x. Daraus werden die anteiligen translatorischen ($t_{\text{shift, } x}$) und rotatorischen ($r_{\text{shift, } x}$) Verschiebungen berechnet als $t_{\text{shift, } x} = w_x \cdot t_x$ und $r_{\text{shift, } x} = w_x \cdot r_x$. Diese translatorischen und rotatorischen Verschiebungen werden nun wieder zurück transformiert zu einer homogenen 4×4 Transformationsmatrix $D_{\text{shift, } x}$. Mit Hilfe der Roboter-spezifischen Vorwärtskinematik $f_{\text{forward}} : \mathbb{R}^d \rightarrow \mathbb{R}^{4\times4}$, welche Gelenkwinkel in homogene kartesische Koordinaten des TCP transformiert und der Roboter-spezifischen Rückwärtskinematik $f_{\text{inverse}} : \mathbb{R}^{4\times4} \rightarrow \mathbb{R}^d$ welche homogene kartesischen Koordinaten des TCP in Gelenkkonfigurationen transformiert, ist es nun möglich die neuen Gelenkkonfigurationen $q_{t,\text{ neu}}$ mit den gewichteten Inkrementen $D_{\text{shift, } l}$ und $D_{\text{shift, } r}$ zu berechnen. Das Ergebnis ist in Gleichung 6.4 dargestellt.

$$q_{t,\text{ neu}} = f_{\text{inverse}}\left(f_{\text{forward}}(q_t) \cdot D_{\text{shift, } l} \cdot D_{\text{shift, } r}\right) \tag{6.4}$$

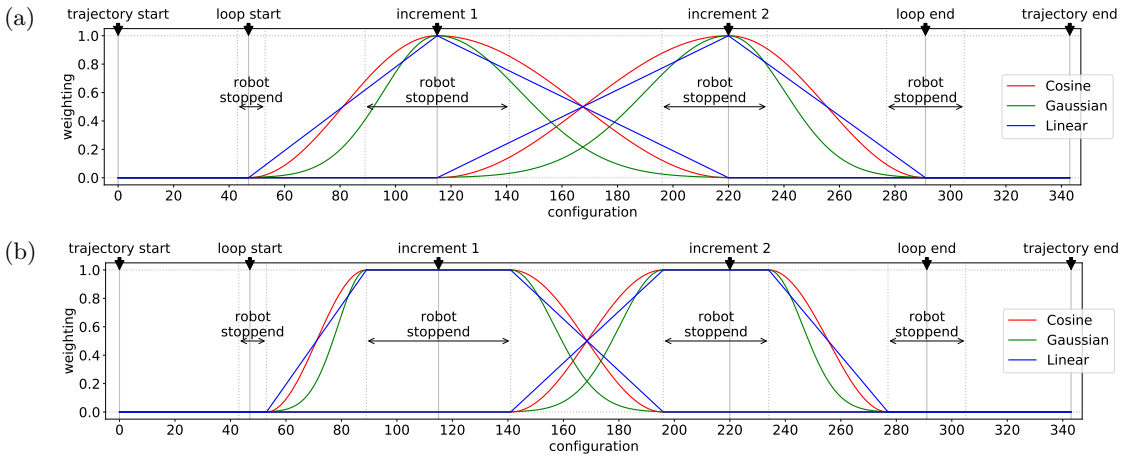


Abbildung 6.12: Vergleich der Anpassungsfunktionen für die Trajektorie der Beispielaufgabe in Abbildung 6.9. Graph (a) zeigt das Ergebnis der Anpassungsfunktionen für die *Punkt-Methode*, Graph (b) für die *Intervall-Methode*. Der blaue Graph zeigt jeweils die Linear-Funktion, der grüne Graph die Gauß-Funktion und der rote Graph die Kosinus-Funktion. Beide Graphen sind aus [Riedl2021].

Gleichung 6.4 wird verwendet, um die neuen Gelenkkonfigurationen innerhalb einer Schleife für jeden Schleifendurchlauf zu berechnen. Die daraus entstandene neue Trajektorie wird anschließend ausgeführt, so dass die sich wiederholende Aufgabe mit leicht abgeänderten Konfigurationen ausgeführt wird.

6.4.3 Vergleich und Evaluation der Anpassungsfunktionen

In diesem Abschnitt werden die drei vorgestellten Anpassungsfunktionen aus Abschnitt 6.4.2 verglichen und anschließend evaluiert. Dabei wird besonders darauf geachtet, welche der Anpassungsfunktion am besten geeignet ist für die kinästhetische Roboterprogrammierung.

Vergleich der Anpassungsfunktionen

Abbildung 6.12 zeigt die sechs unterschiedlichen Kombinationen der drei Anpassungsfunktionen mit jeweils der Punkt- und der Intervall-Methode für die Beispielaufgabe aus Abbildung 6.9 mit 343 Konfigurationen. In den Graphen sind Schleifenbeginn, Schleifenende und beide Inkrement-Konfigurationen dargestellt. Des Weiteren werden die Intervalle markiert, an denen der Roboter still steht, um den Unterschied zwischen der Punkt-Methode (Abbildung6.12a) und der Intervall-Methode (Abbildung 6.12b) hervorzuheben.

Beim Vergleich der Anpassungsfunktionen fällt auf, dass die Linear-Funktion und die Kosinus-Funktion punktsymmetrisch innerhalb eines Bereiches zwischen zwei Inkrementen sind. Im Vergleich dazu ist bei der Gauß-Funktion der Bereich, bei dem das Inkrement stärker gewichtet wird, kürzer als der Bereich, in dem das Inkrement weniger stark gewichtet wird. Zusätzlich besitzt die Gauß-Funktion den Nachteil, dass es an der letzten Konfiguration, die von einem Inkrement verändert werden soll, eine nicht stetige Trajektorie mit Sprungstelle erzeugt. Dies ist der Fall, da die Gauß-Funktion niemals den Wert 0 erreicht, was bereits in Gleichung 6.2 gezeigt wurde. Weiterhin fällt beim Vergleich der Linear-Funktion und Kosinus-Funktion auf, dass die Linear-Funktion keine glatte Bewegung an den Inkrement-Konfigurationen und am Schleifenbeginn und Schleifenende erzeugt, wohingegen mit der Kosinus-Funktion eine durchgehend glatte Bewegung entsteht. Aus mathematischer Sicht ist die Kosinus-Funktion im Vergleich zu den anderen beiden Methoden zu bevorzugen, unabhängig davon, ob die Punkt- oder Intervall-Methode verwendet wird.

Evaluation der Anpassungsfunktionen

Um festzustellen, ob die Punkt- oder Intervall-Methode besser für kinästhetisch demonstrierte Roboterprogramme geeignet ist, werden in diesem Abschnitt alle sechs Kombinationen aus Anpassungsfunktionen und Gewichtungsmethoden verglichen. Dafür wird die Beispielaufgabe aus Abbildung 6.9 verwendet und deren Trajektorie mit den jeweiligen Methoden verändert. Abbildung 6.13 zeigt für die sechs Kombinationen die resultierenden angepassten Trajektorien.

Bei Verwendung der Punkt-Methode fällt auf, dass unabhängig von der gewählten Anpassungsfunktion die Trajektorie kurz vor der grünen Konfiguration eine Delle hat. Sollten diese Trajektorien ausgeführt werden, würde diese Delle dafür sorgen, dass der Roboter mit den gestapelten Objekten kollidiert und somit die Aufgabe nicht ausführen kann. Mit Hilfe der Graphen in Abbildung 6.12 lässt sich diese Delle erklären. Um Inkrement-Konfiguration zwei existiert ein Intervall, in dem sich der Roboter nicht bewegt. Die Punkt-Methode jedoch sorgt dafür, dass in diesem Bereich bereits die Gewichtung des Inkrements verringert wird. Dies sorgt dafür, dass die Konfigurationen in der Nähe der grünen Konfiguration in z-Richtung nach unten gezogen werden, weil das Inkrement nicht mehr zu 100% auf die Konfigurationen gerechnet werden.

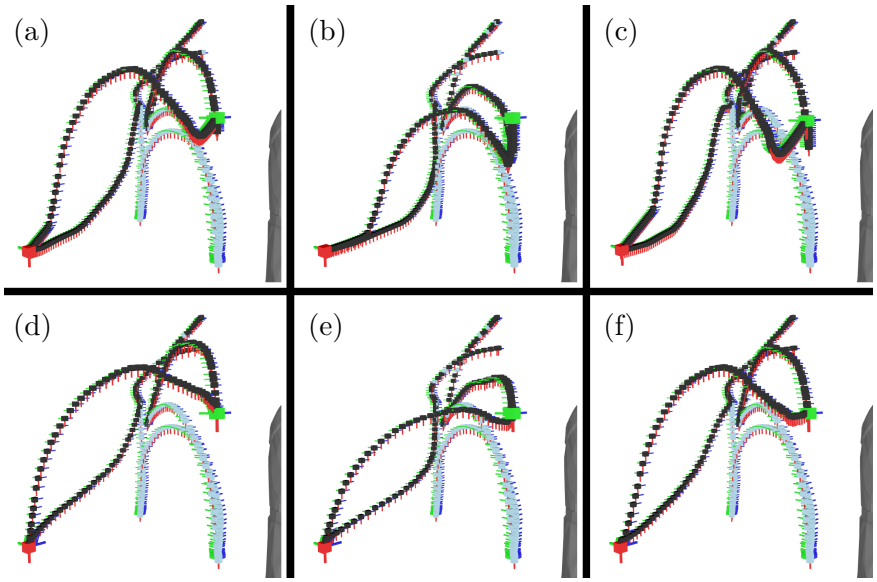


Abbildung 6.13: Simulation der Anpassungsfunktionen für die Beispielaufgabe aus Abbildung 6.9. Die Original-Trajektorie ist in den Bildschirmaufnahmen hellblau dargestellt, die angepasste Trajektorie schwarz. Die angepasste Trajektorie besitzt ein Inkrement in x-Richtung für die Depalettier-Konfiguration (roter Würfel in den Trajektorien) und ein Inkrement in z-Richtung für die Stapel-Konfiguration (grüner Würfel in den Trajektorien). (a), (b) und (c) zeigen die angepasste Trajektorie für die *Punkt-Methode*, (d), (e) und (f) für die *Intervall-Methode*. Die Kosinus-Funktion wird in (a) und (d) dargestellt, die Gauß-Funktion in (b) und (e) und die Linear-Funktion in (c) und (f). Alle Abbildungen sind aus [Riedl2021].

Um diese Art Fehler zu vermeiden ist es möglich die Intervall-Methode zu verwenden, da diese erst die Gewichte verändert wenn sich der Roboter bewegt. Das Ergebnis dieser Methode wird in (d), (e) und (f) von Abbildung 6.13 dargestellt. Es ist zu erkennen, dass die veränderten Trajektorien die Delle der Punkt-Methode nicht mehr besitzen und dementsprechend ohne Kollision ausgeführt werden können.

Aufgrund des Vergleichs der unterschiedlichen Anpassungsfunktionen und der Evaluation der Punkt- und Intervall-Methode wurde in dieser Arbeit die Kosinus-Funktion mit der Intervall-Methode als bevorzugte Kombination gewählt. Diese Kombination hat die besten Ergebnisse in der Simulation gezeigt und ist hinsichtlich mathematischer Aspekte wie Stetigkeit und Glattheit am besten geeignet.

6.5 Schlussfolgerung

Zusammenfassend kann festgehalten werden, dass im Bereich der Kontrollstrukturen für kinästhetisch programmierte Roboter zwei neue Konzepte in dieser Arbeit entworfen und vorgestellt wurden. Es wurde ein Ansatz für sensorbasierte Kontrollstrukturen in Form von Schleifen (Abschnitt 6.2.1) und Verzweigungen (Abschnitt 6.2.2) entwickelt. Mit diesem Ansatz ist es nun auch in der kinästhetischen Roboterprogrammierung möglich, auf Umwelteinflüsse anhand von Sensorwerten reagieren zu können und so zum Beispiel Programmteile erneut abzuspielen (Schleife) oder aus einer Menge von Programmteilen den zu einem Sensorwert passenden Programmteil auszuwählen (Verzweigung). Die Evaluation der sensorbasierten Kontrollstrukturen zeigte, dass sowohl die Kontrollstrukturen, als auch das Programmiersystem in einer frühen Ausbaustufe bereits als intuitiv bezeichnet werden können.

Zusätzlich wurde ein Konzept für Schleifen-Inkremente entwickelt, um den Funktionsumfang der kinästhetischen Roboterprogrammierung zu erweitern. Mit diesen Schleifen-Inkrementen ist es nun möglich, sich wiederholende Aufgaben, welche eine regelmäßige geometrische Veränderung nach jedem Schleifendurchlauf benötigen, nur mit der Demonstration des ersten Schleifendurchlaufs zu programmieren. Dadurch können nun Stapel- und Palettierszenarien mit Hilfe der kinästhetischen Programmierung Robotern beigebracht werden. Die Evaluation der verschiedenen Möglichkeiten zur Anpassung der Trajektorien hat gezeigt, dass für die kinästhetische Roboterprogrammierung die Kosinus-Funktion in Kombination mit der Intervall-Methode am besten geeignet ist.

Forschungsfrage F3 kann damit beantwortet werden, dass es möglich ist, sensorbasierte Kontrollstrukturen in Form von Schleifen und Verzweigungen aus prozeduralen Programmiersprachen so anzupassen, dass diese auch in der kinästhetischen Roboterprogrammierung verwendet werden können. Eine Evaluation eines ersten Prototypen hat bereits gezeigt, dass dies auch intuitiv möglich ist.

Mit dem Ausarbeiten eines Konzeptes für Schleifen-Inkremente kann Forschungsfrage F4 so beantwortet werden, dass auch Schleifen-Inkremente für die kinästhetische Roboterprogrammierung eingesetzt werden können. Als geeignete Methode zum anpassen der Trajektorie nach jedem Schleifendurchlauf hat sich dabei die Intervall-Methode in Kombination mit der Kosinus-Funktion herausgestellt.

Die in den bisherigen Kapiteln vorgestellten Konzepte für die kinästhetische Roboterprogrammierung können alle mit einzelnen Robotern verwendet werden. Welche Probleme bei der Verwendung mehrerer Roboter auftreten können und wie diese mit Hilfe von Synchronisationsmechanismen umgangen werden können wird im nächsten Kapitel näher beleuchtet.

KAPITEL 7

Synchronisieren

Inhalt

Nachdem in den bisherigen Kapiteln Konzepte vorgestellt wurden, welche auf einzelne Roboterprogramme beschränkt sind, wird in diesem Kapitel die Koordinierung mehrerer Roboter, sowie die Koordinierung zwischen Mensch und Roboter genauer betrachtet. Bei den bisher vorgestellten Konzepten ist es zwar auch möglich, die kinästhetischen Programme mehrerer Roboter gleichzeitig wiedergeben zu lassen, jedoch gibt es keinen Mechanismus, welcher sicherstellt, dass Teile der Programme, welche ein gemeinsames Arbeiten der Roboter erfordern, auch synchron ausgeführt werden. So kann es vorkommen, dass durch das Editieren eines Roboterprogramms oder durch die Verwendung von sensorbasierten Kontrollstrukturen Bereiche, welche synchron ausgeführt werden sollen, innerhalb der Zeitleisten verschoben werden. Dies ist insbesondere ein Problem, wenn die synchron auszuführenden Tätigkeiten in einem gemeinsamen Arbeitsraum stattfinden sollen und dadurch potentiell Kollisionen entstehen können. Ein Beispiel für einen geteilten Arbeitsraum eines Mehrrobotersystems ist in Abbildung 7.1 dargestellt. Um die synchrone Ausführung von Programmen auch nach dem Editieren sicherzustellen, wird in diesem Kapitel das Konzept der zeitlichen Synchronisation von mehreren Robotern bzw. zwischen Mensch und Roboter eingeführt.

Hierzu wird zunächst in Abschnitt 7.1 das Konzept der Synchronisationspunkte und Synchronisationsintervalle erläutert. Anschließend wird in Abschnitt 7.2 auf die Synchronisation mehrerer Roboter eingegangen, bevor in Abschnitt 7.3 ein Konzept für die Synchronisation zwischen Roboter und Mensch vorgestellt wird.

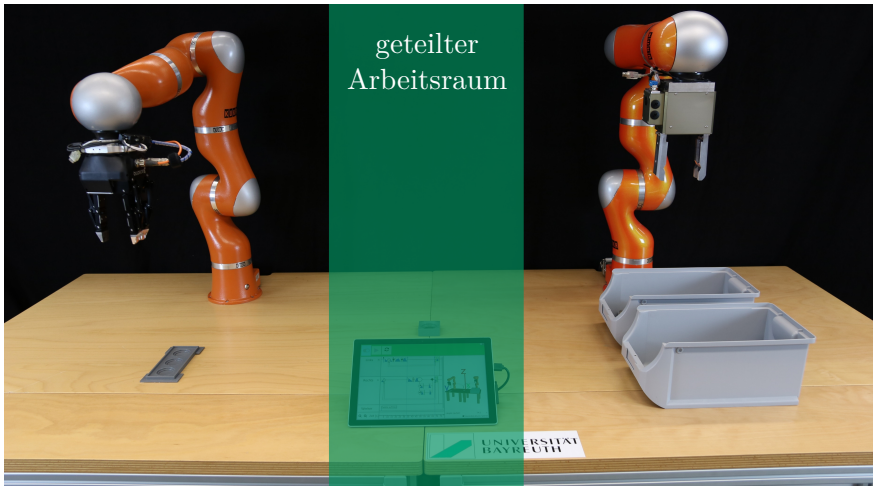


Abbildung 7.1: Schematische Darstellung eines geteilten Arbeitsraums. Der grüne Bereich kann von beiden Robotern erreicht werden, dadurch besteht die Gefahr einer Kollision.

In diesem Kapitel wird folgende Forschungsfrage behandelt:

F5 In wie weit lässt sich das Programmierkonzept auf die Zusammenarbeit mehrerer Roboter und auf die Zusammenarbeit zwischen Mensch und Roboter erweitern?

Eine frühe Version der zeitlichen Synchronisation mehrerer Roboter wurde bereits in [Riedl2016] und [Riedl2019] vorgestellt.

7.1 Synchronisationspunkte und Synchronisationsintervalle

In dieser Arbeit werden mit den Synchronisationspunkten und den Synchronisationsintervallen zwei Arten der zeitlichen Synchronisation vorgestellt, welche sich in ihrer Semantik unterscheiden. Beide Arten der Synchronisation werden dabei für beliebige Aktoren definiert, wobei ein Aktor im Kontext dieser Arbeit entweder ein Roboter oder ein Mensch ist. Bei Synchronisationspunkten warten die beteiligten Aktoren aufeinander und führen anschließend weiter ihre Aufgaben aus, wohingegen bei Synchronisationsintervallen die beteiligten Aktoren über einen definierten Zeitraum, dem Intervall, eine Aufgabe gemeinsam erledigen. Beide Arten der zeitlichen Synchronisation ha-

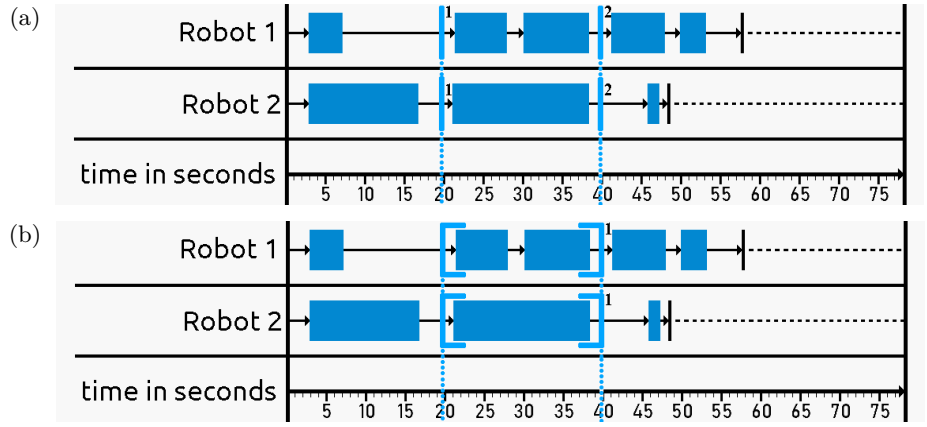


Abbildung 7.2: Zeitleistendarstellungen von synchronisierten Roboterprogrammen welche eine identische Aufgabe ausführen. Abbildung (a) zeigt die Synchronisation mit zwei Synchronisationspunkten, Abbildung (b) mit einem Synchronisationsintervall. Nur in Abbildung (b) ist durch die Klammern erkennbar, dass beide Roboter zwischen Sekunde 19 und Sekunde 39 gemeinsam eine Aufgabe ausführen.

ben den Vorteil, dass, sollte das kinästhetisch demonstrierte Roboterprogramm eines beteiligten Roboters vor der synchronisierten Stelle editiert werden, der Bereich, der synchron ausgeführt werden soll, auch nach dem Editieren noch synchron ausgeführt wird.

Ein Synchronisationspunkt ist für einen Aktor definiert als ein Zeitpunkt t und einer Synchronisations-ID *id*. Alle an diesem Synchronisationspunkt beteiligten Aktoren besitzen an beliebiger Stelle in ihrem Programm auch einen Synchronisationspunkt mit der gleichen Synchronisations-ID *id*. Die Funktionsweise von Synchronisationspunkten ist vergleichbar mit einer Barriere in der parallelen Programmierung. Alle beteiligten Aktoren warten an der definierten Position in ihrem Programm, bis die verbleibenden Aktoren den Synchronisationspunkt mit der gleichen *id* erreicht haben. Danach setzen alle Aktoren gleichzeitig die Abarbeitung des restlichen Programms fort. Aufgaben, bei denen Synchronisationspunkte benötigt werden sind zum Beispiel die Übergabe von Objekten zwischen Aktoren, das Warten auf neue Werkstücke, das Absichern eines gemeinsamen Arbeitsraums und ähnliche Aufgaben.

Analog sind Synchronisationsintervalle für einen Aktor definiert als Startzeitpunkt t_1 und Endzeitpunkt t_2 und einer Synchronisations-ID *id*. Auch hier besitzen alle beteiligten Aktoren an einer beliebigen Stelle des Programms ein Synchronisationsintervall mit der Synchronisations-ID *id*. Bei Synchronisationsintervallen ist es so, dass alle beteiligten Aktoren jeweils am Startpunkt t_1 des Intervalls im Programm warten, bis alle Aktoren in ihren Programmen den Startpunkt des Intervalls mit der gleichen *id* erreicht haben. Anschließend wird das innere des Synchronisationsintervalls synchron

ausgeführt, so dass von den beteiligten Aktoren eine Aufgabe gemeinsam ausgeführt werden kann. Am Ende des Synchronisationsintervalls warten wieder alle Aktoren zum Zeitpunkt t_2 aufeinander und führen den restlichen Teil des Programms im Anschluss daran aus. Vergleichbar ist dieses Vorgehen mit der Verwendung zweier Barrieren, welche einen synchron auszuführenden Bereich in der parallelen Programmierung umschließen. Die zweite Barriere ist für die Ausführung des Roboterprogramms nicht zwangsweise notwendig. Sie sorgt aber durch die Darstellung in der Benutzungsoberfläche dafür, dass verdeutlicht wird, dass in einem gewissen Bereich etwas von mehreren Aktoren gemeinsam ausgeführt werden soll (Abbildung 7.2). Sowohl das Programm in Abbildung 7.2a, als das in Abbildung 7.2b führen die gleiche Aufgabe aus. Durch die Darstellung in Abbildung 7.2b wird jedoch verdeutlicht, dass beide Roboter zwischen Sekunde 19 und Sekunde 39 gemeinsam eine Aufgabe ausführen. Aufgaben, bei denen Synchronisationsintervalle verwendet werden können sind zum Beispiel das gemeinsame Heben von schweren Objekten, die gemeinsame Montage von Bauteilen, das Überwachen des Roboters durch den Menschen bei besonders anspruchsvollen und fehleranfälligen Tätigkeiten und ähnliche Aufgaben.

Sowohl Synchronisationspunkte als auch Start und Ende von Synchronisationsintervallen dürfen nur an Stellen in ein kinästhetisch demonstriertes Roboterprogramm eingefügt werden, an denen der beteiligte Roboter still steht. Dies hat den Hintergrund, dass jeweils an den Rändern des Synchronisationsintervalls und am Synchronisationspunkt ein Warten auf die anderen beteiligten Aktoren notwendig sein kann. Würde der Roboter an diesen Stellen nicht still stehen, so müsste eine Vollbremsung eingeleitet werden, was zu einer Beschädigung der Roboter oder deren Umgebung führen kann.

Sowohl das Konzept der Synchronisationspunkte, als auch das Konzept der Synchronisationsintervalle wurde im Rahmen dieser Arbeit auf die Synchronisation zwischen mehreren Robotern und auf die Synchronisation zwischen Roboter und Mensch angewendet. Abschnitt 7.2 beschreibt die Synchronisation mehrerer Roboter, Abschnitt 7.3 die Synchronisation von Roboter und Mensch.

7.2 Synchronisation mehrerer Roboter

Bei der Synchronisation zwischen mehreren Roboter geht es darum, die Programme der unterschiedlichen Roboter zu koordinieren und abzusichern. Dies ist insbesondere wichtig, wenn die Roboter miteinander interagieren, oder in einem geteilten Arbeitsraum arbeiten sollen, da so die Gefahr einer Kollision und damit einer Beschädigung der Roboter, oder der verarbeiteten Bauteile, besteht. Potentielle Kollisionen können durch die in diesem Kapitel vorgestellten Synchronisationsmethoden verhindert werden.

Dadurch, dass sich Synchronisationspunkte und -intervalle nur in ihrer Semantik unterscheiden, ein Synchronisationsintervall aber als zwei Synchronisationspunkte, jeweils am Beginn und am Ende des Intervalls, darstellbar ist, ist die Funktionsweise für beide Arten der Synchronisierung gleich. Alle Roboter, welche am jeweiligen Synchronisationspunkt bzw. -intervall beteiligt sind, warten am Punkt bzw. den Start- oder Endpunkten des Intervalls aufeinander, bis alle Roboter diesen erreicht haben. Zusammengehörende Punkte oder Intervalle sind in den verschiedenen Roboterprogrammen durch die gleiche Synchronisations-ID erkennbar. Haben alle Roboter den Wartepunkt erreicht so wird dieser freigegeben und die beteiligten Roboter führen den Rest ihres Roboterprogramms aus. Dieses Vorgehen hat den Vorteil, dass durch geschicktes Einsetzen der Punkte und Intervalle Kollisionen vermieden werden können und und im gemeinsamen Arbeitsraum gearbeitet werden kann. Jedoch besitzt es auch den Nachteil, dass unter Umständen unnötige Wartezeiten erzeugt werden, wenn die Synchronisation ungeschickt gewählt wurde. Im schlechtesten Fall führen die Roboter sequentiell nacheinander die Programme aus, obwohl ein parallelisieren von Teilprogrammen möglich wäre. Dabei sind die Benutzer:innen des Programmiersystems in der Verantwortung, die Synchronisationspunkte und -intervalle so zu wählen, dass keine Deadlocks, also keine zyklischen Abhängigkeiten der Synchronisationspunkte und -intervalle, entstehen.

Soll im Programmiersystem die Roboter-Roboter Synchronisationsfunktion verwendet werden, so ist das sowohl für Synchronisationspunkte, als auch für Synchronisationsintervalle ähnlich möglich. Bei einem Punkt müssen die Benutzer:innen in der Zeitleistendarstellung des Roboterprogramms den Zeitpunkt markieren, an dem der Roboter auf andere Roboter warten soll, und dann über die Roboter-Roboter Synchronisationsfunktion einen Synchronisationspunkt einfügen. Anschließend muss für diesen Punkt noch die *id* definiert werden. Dieses Vorgehen wird für alle am Synchronisationspunkt beteiligten Roboter wiederholt, wobei immer die gleiche *id* angegeben werden muss. Für Synchronisationsintervalle ist das Vorgehen analog, nur dass anstelle eines Zeitpunkts ein Zeitbereich, welcher synchron ausgeführt werden soll, markiert werden muss. Nachdem die Punkte und Intervalle eingefügt wurden, aktualisiert sich automatisch die Zeitleistendarstellung der Roboterprogramme und zeigt die parallel auszuführenden Teile bzw. die Wartepunkte bei den Synchronisationspunkten auch auf den Zeitleisten der Roboter parallel zueinander an. Dadurch ist für die Benutzer:innen sofort ersichtlich, wie sich die Abarbeitung paralleler Roboterprogramme mit den Synchronisationspunkten und -intervallen zueinander verschoben hat. Eine Bildschirmaufnahme mit unsynchronisierten Robotern, bei der es zu einer Kollision der Roboter kommen würde, ist in Abbildung 7.3a dargestellt. Die gleichen Roboter mit synchronisierten Programmen ohne Kollision sind in Abbildung 7.3b zu sehen.

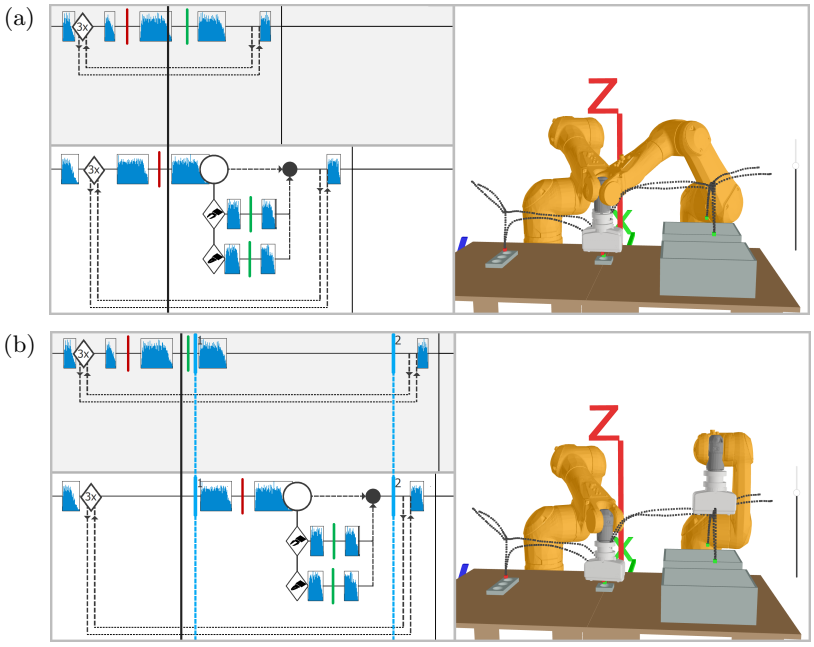


Abbildung 7.3: Abbildung (a) zeigt eine Bildschirmaufnahme eines nicht synchronisierten Roboterprogramms. Im Simulationsfenster ist zu sehen, dass es bei diesem Programm zu einer Kollision kommen würde. Abbildung (b) zeigt das gleiche Programm mit Synchronisationspunkten. Die Synchronisationspunkte werden übereinander angezeigt, so dass zu sehen ist, wie sich die Ausführung der Programme zueinander verschiebt. Im Simulationsfenster ist zu sehen, dass keine Kollision mehr auftritt. Der schwarze Balken ist jeweils die Fortschrittsanzeige. Sie zeigt an, welche Konfigurationen von den Robotern im Simulationsfenster dargestellt werden.

7.3 Synchronisation von Roboter und Mensch

Zusätzlich zur Synchronisation mehrerer Roboter untereinander kann es bei manchen Aufgaben notwendig sein, dass auch eine Synchronisation mit einem Menschen erfolgen muss. Dies ist insbesondere dann der Fall, wenn der Roboter nicht alle Aufgaben selbst ausführen kann, weil er zum Beispiel nicht das richtige Werkzeug besitzt oder ein Werkstück außerhalb seines Arbeitsraums platziert ist. In diesem Fall ist es sinnvoll, wenn im Roboterprogramm die Stelle markiert werden kann, an welcher der Mensch eingreifen muss, um so ein Zusammenspiel zwischen Mensch und Roboter zu ermöglichen.

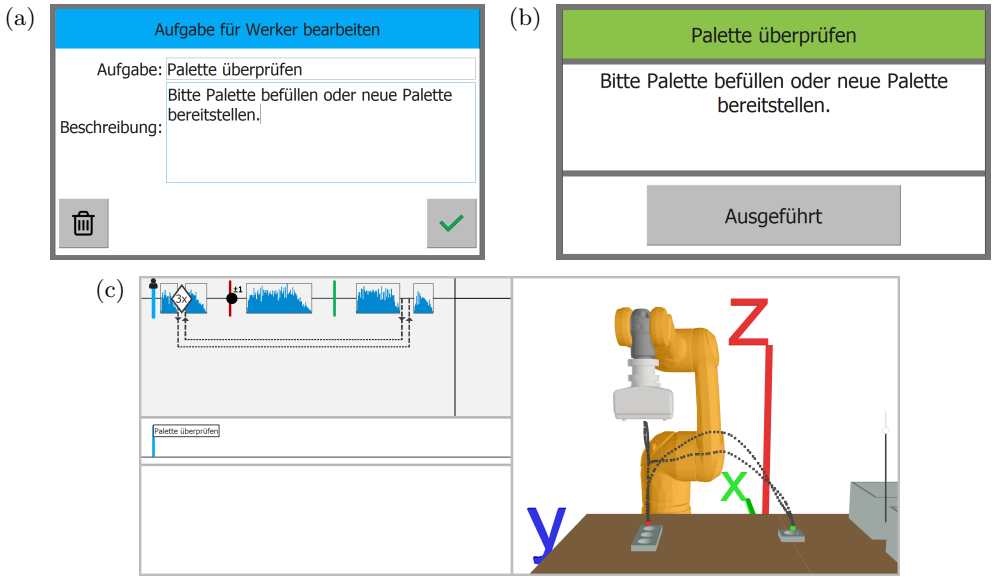


Abbildung 7.4: Bildschirmaufnahmen einer Mensch-Roboter Synchronisation. Abbildung (a) zeigt, wie die Mensch-Roboter Synchronisation definiert wird. Abbildung (b) zeigt die Meldung, welche die Benutzer:innen bei erreichen der Mensch-Roboter Synchronisation angezeigt bekommen. Abbildung (c) zeigt das Roboterprogramm und die Zeitleiste für die Benutzer:innen inklusive eingefügter Mensch-Roboter Synchronisation.

Die Mensch-Roboter Synchronisation funktioniert dabei so, dass analog zur Roboter-Roboter Synchronisation im kinästhetisch demonstrierten Roboterprogramm des beteiligten Roboters ein Synchronisationspunkt oder -intervall mit Hilfe der Mensch-Roboter Synchronisationsfunktion definiert wird. Anstelle einer Synchronisations-ID muss nun eine Beschreibung der Tätigkeit angegeben werden, welche vom Menschen auszuführen ist, wenn der jeweilige Synchronisationspunkt bzw. das jeweilig Synchronisationsintervall erreicht wird. Die eingegebene Kurzbeschreibung wird in einer separaten Zeitleiste im Programmiersystem angezeigt. Bei der Ausführung eines Roboterprogramms, welches eine Synchronisation mit dem Menschen beinhaltet, wartet der Roboter am Synchronisationspunkt bzw. -intervall und benachrichtigt den Menschen über einen Benachrichtigungskanal, dass dessen Mithilfe benötigt wird. Für die prototypische Implementierung ist der Benachrichtigungskanal der vorhandene Touchscreen und die Benachrichtigung wird textuell in natürlicher Sprache an den Menschen, verschickt. Hat der Mensch seine Aufgabe erfüllt bzw. ist bereit, parallel mit dem Roboter seine Aufgabe auszuführen, so muss dieser über den Benachrichtigungskanal die Aufgabe bestätigen. Im Prototypen des Programmiersystems muss dazu die Meldung auf dem Touchscreen bestätigt werden. Dadurch wird der Wartepunkt des Roboters freigegeben und dieser fährt mit der Ausführung des weiteren Roboterprogramms fort.

Ein Beispielprogramm, in dem der Mensch dazu aufgefordert wird zu überprüfen, ob eine Palette mit Werkstücken vor Ausführung des Roboterprogramms bereit steht, ist in Abbildung 7.4c zu sehen. Zusätzlich ist die Definition der Mensch-Roboter Synchronisation (Abbildung 7.4a) und die Meldung, welche den Benutzer:innen während der Ausführung des Roboterprogramms angezeigt wird (Abbildung 7.4b), dargestellt.

Das Konzept der Mensch-Roboter Synchronisation lässt sich erweitern zu einer Synchronisation mit externen Aktoren. So könnte mit dem gleichen Konzept zum Beispiel ein Roboterprogramm mit einer Speicherprogrammierbaren Steuerung (SPS) synchronisiert werden. Dabei ist es denkbar, dass das Roboterprogramm durch das Erreichen eines Synchronisationspunktes und der dadurch erfolgten Benachrichtigung einer SPS die Ausführung einer bestimmten Aufgabe anstößt. Nachdem die SPS über den gleichen Benachrichtigungskanal das Ausführen der Aufgabe quittiert hat, setzt der Roboter die Ausführung seines Programms fort.

7.4 Schlussfolgerung

Abschließend kann festgehalten werden, dass mit dem in diesem Kapitel vorgestellten Synchronisierungsmechanismen eine zeitliche Synchronisation sowohl zwischen mehreren Robotern, als auch zwischen Mensch und Roboter ermöglicht wird und so mögliche Kollisionen bei Verwendung mehrerer Roboter verhindert werden können. Mit den Synchronisationspunkten und Synchronisationsintervallen stehen den Benutzer:innen zwei Arten der Synchronisation zur Verfügung, welche sich nur in der Semantik, nicht aber im konzeptionellen Aufbau, unterscheiden. Es wurde sowohl ein Konzept zur zeitlichen Synchronisation mehrerer Roboter erarbeitet, als auch ein Konzept zur Synchronisation zwischen Mensch und Roboter. Letzteres Konzept kann als Beispiel einer Synchronisation mit einem beliebigen externen Aktor, beispielsweise einer SPS, gesehen werden.

Forschungsfrage F5 kann demnach damit beantwortet werden, dass das Konzept der kinästhetischen Roboterprogrammierung mit den in diesem Kapitel vorgestellten Synchronisationsmechanismen in der Lage ist, mehrere Roboter gemeinsam eine Aufgabe ausführen zu lassen, welche im gleichen Arbeitsraum stattfindet. Auch wurde mit der Synchronisation zwischen Mensch und Roboter ein Konzept entwickelt, welches es erlaubt, auch Aufgaben, welche von einem Roboter nicht erledigt werden können, trotzdem im Roboterprogramm abzubilden, und bei Bedarf die Benutzer:innen zur Hilfe zu holen.

Nachdem mit Abschluss dieses Kapitels alle inhaltlichen Konzepte dieser Arbeit vorgestellt wurden, wird im nächsten Kapitel die Frage geklärt werden, wie intuitiv das entstandene Programmiersystem zu bedienen ist. Dazu wurde eine Nutzungsstudie durchgeführt, deren Aufbau, Ablauf und Ergebnisse im Folgenden vorgestellt werden.

KAPITEL 8

Evaluation

Inhalt

In den letzten Kapiteln wurden alle neuen Konzepte zur Erweiterung und Verbesserung der Intuitivität für die sensorbasierten kinästhetischen Roboterprogrammierung vorgestellt. Mit dieser Gesamtevaluation soll herausgefunden werden, inwiefern die entwickelten Konzepte und der entstandene Prototyp als intuitiv zu bewerten ist. Des Weiteren soll evaluiert werden, welche Schwachstellen noch vorhanden sind und wie diese als Ausblick für spätere Arbeiten verbessert werden können.

Hierfür wird in Abschnitt 8.1 der Gesamtdemonstrator vorgestellt und dessen Aufbau erläutert. Anschließend wird in Abschnitt 8.2 die durchgeführte Nutzungsstudie, deren Ergebnisse und die daraus gewonnenen Erkenntnisse vorgestellt. Abschließend werden in Abschnitt 8.3 die Grenzen des vorgestellten Programmiersystems und mögliche Verbesserungen genauer erläutert.

In diesem Kapitel wird folgende Forschungsfrage und die dazugehörigen Unterfragen behandelt:

F6	Inwiefern lässt sich das entstandene kinästhetische Programmierkonzept und dessen prototypische Implementierung intutiv bedienen? Wie spiegelt sich das in Effektivität, mentaler Effizienz und Zufriedenheit der Programmierung wider?

Zu dieser Forschungsfrage gehören folgende Unterfragen, welche mit Hilfe der Studie beantwortet werden sollen:

F6.1	Wie hoch ist die erwartete Komplexität des Programmiersystems im Vergleich zur tatsächlich empfundenen?
F6.2	Gibt es bei der Bewertung des Systems Unterschiede zwischen Robotik-Expert:innen, Domänen-Expert:innen und Nicht-Expert:innen?
F6.3	Wie verständlich sind die für die prototypische Implementierung verwendeten Symbole der graphischen Benutzungsoberfläche?
F6.4	Welche der beiden entworfenen Darstellungsmöglichkeiten von Bewegungsbefehlen entlang einer Zeitleiste (Graph- oder Blockdarstellung) bevorzugen die Benutzer:innen des Programmiersystems?
F6.5	Bevorzugen Benutzer:innen bei repetitiven Aufgaben eher Kopieren&Einfügen oder eine Schleife?

Aufgrund der Corona-Pandemie konnte die Studie nicht wie geplant im Roboterlabor am realen Robotersystem durchgeführt werden. Stattdessen wurde die Studie mit Hilfe einer Simulation des echten Roboters gestaltet, welche alle für den Prototypen relevanten Funktionen des realen Roboters enthält. Dadurch war eine Teilnahme der Proband:innen an der Studie unabhängig vom Ort möglich. Diese erfolgte entweder mit Hilfe der Fernsteuermöglichkeit über Microsoft Teams oder unter befolgen der geltenden Hygieneregeln bei den Proband:innen zu Hause. Die einzige Einschränkung des simulierten Systems im Vergleich zum realen Robotersystem war, dass die Roboter nicht kinästhetisch geführt werden konnten. Um den Teilnehmer:innen trotzdem den Studienaufbau und die Roboter zu visualisieren, wurde mit Hilfe von Modellen der verwendeten Roboter und einem skalierten Aufbau des Szenarios als 3D-Druck ein Modell des Robotersystems ähnlich dem im Roboterlabor angefertigt. Obwohl aufgrund der pandemischen Lage das Roboterlabor nicht verwendet werden konnte, war es mit diesem alternativen Evaluationsaufbau möglich, die Bedienbarkeit des Programmiersystems zu evaluieren.

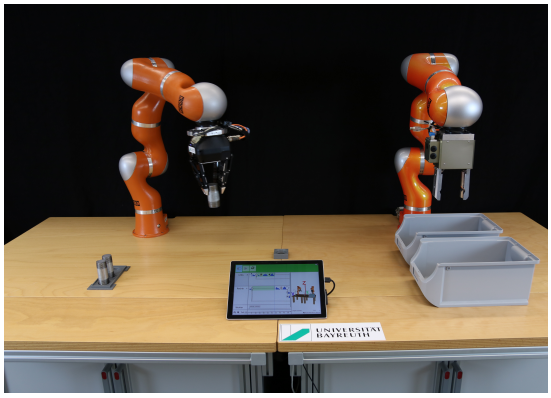
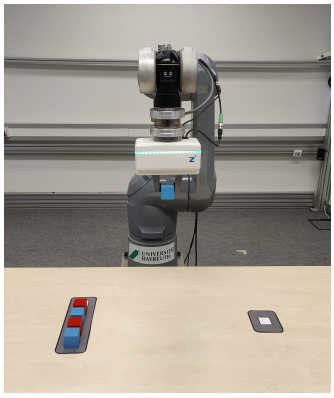
(a)  (b) 

Abbildung 8.1: Die vom Programmiersystem unterstützten Robotertypen. Abbildung (a) zeigt den Kuka LBR 4+ (links) und den Kuka LBR 4 (rechts). Abbildung (b) zeigt den Stäubli TX2-90L.

8.1 Gesamtdemonstrator

Bevor die Studie und die daraus gewonnenen Ergebnisse vorgestellt werden, wird in diesem Abschnitt der Gesamtdemonstrator des Programmiersystems präsentiert. Mit diesem Gesamtdemonstrator wurde die Studie durchgeführt. Für die Studie wurde aus den in Abschnitt 8.1.1 vorgestellten unterstützten Robotertypen der virtuelle Roboter in Kombination mit der in Abschnitt 8.1.2 vorgestellten Benutzungsoberfläche und der in Abschnitt 8.1.3 dargestellten Architektur auf einem Tablet-PC verwendet.

8.1.1 Unterstützte Robotertypen

Dadurch, dass das dem Programmiersystem zugrundeliegende Programmierparadigma die kinästhetische Roboterprogrammierung, und im speziellen Fall die Playback Programmierung, ist, müssen alle Robotertypen einen Modus zum kinästhetischen Führen besitzen. Dies kann entweder durch interne Sensorik in den Gelenken und einer sogenannten Gravitationskompensation umgesetzt sein, oder mit Hilfe einer am Flansch befestigten Kraftmessdose, deren Werte intern in Fahrbefehle übersetzt werden. Ist eine der beiden Möglichkeiten zum kinästhetischen Führen eines Robotertypen gegeben, so kann dieser als Client in die Architektur des Programmiersystems integriert werden, welche in Abschnitt 3.3 beschrieben wurde.

Konkret unterstützt werden zum einen Kuka LBR 4 und 4+ [DLR] über eine Ansteuerung über das Fast Research Interface [Schreiber2010]. Bei diesen Robotertypen wird die Funktionalität zum kinästhetischen Führen bereits vom Hersteller bereitgestellt und mit Hilfe interner Kraft-Momenten Sensorik umgesetzt. An den beiden im Robotiklabor des Lehrstuhls Angewandte Informatik 3 der Universität Bayreuth installierten Robotern sind verschiedene Werkzeuge und Sensoren vorhanden. Der Kuka

LBR 4+ besitzt eine UI-1220SE Farbkamera von iDS-Imaging [iDS-Imaging] als Sensor und einen Adaptiven 3-Finger-Greifer von Robotiq [Robotiq] als Werkzeug. Der Kuka LBR 4 besitzt einen Schunk PG70 Backengreifer [Schunk] und keine Kamera. Die beiden unterstützten Kuka LBR Robotersysteme sind in Abbildung 8.1a dargestellt.

Zum anderen werden Stäubli Roboter vom Typ TX2-90L [Stäubli] unterstützt, welche mittels der Kraftmessdose F/T Gamma von ATI [ATI] am Flansch und einem in [Stolka2003] entworfenen Regler um die Funktionalität des kinästhetischen Führens erweitert wurde. Als Werkzeug befindet sich ein Prototyp des für die Mensch-Roboter-Kollaboration zertifizierten Backengreifers HRC-01 der Firma Zimmer [Zimmer]. Angesteuert werden die Stäubli Roboter dadurch, dass das Roboterprogramm vom Programmiersystem in Stäubli-eigenen VAL3-Code übersetzt wird und dieser dann auf der Steuerung des Roboters ausgeführt wird. Das unterstützte Stäubli Robotersystem ist in Abbildung 8.1b dargestellt.

Zusätzlich zu den realen Robotern wurde ein virtueller Roboter implementiert, welcher alternativ im Programmiersystem verwendet werden kann. Dieser kann alle Befehle des Programmiersystems verarbeiten und sorgt dafür, dass die Simulation im Programmiersystem ohne realen Roboter verwendet werden kann. Einziger Nachteil des virtuellen Roboters ist, dass dieser aufgrund des fehlenden physikalischen Roboters nicht kinästhetisch programmiert werden kann. Als Ersatz dazu wurde eine Funktion implementiert, die bei Verwendung der Aufzeichnen-Funktion des Programmiersystems ein vorher am realen Roboter programmiertes Roboterprogramm an das Programmiersystem sendet. Beim Ausführen eines Roboterprogramms mit dem virtuellen Roboter bewegt sich das Robotermodell in der Simulation des Programmiersystems exakt so, wie es der reale Roboter machen würde. Somit können bis auf das kinästhetische Führen mit dem virtuellen Roboter alle Funktionen des Programmiersystems verwendet werden, obwohl kein realer Roboter verwendet wird.

8.1.2 Benutzungsoberfläche

Abbildung 8.2 zeigt die Benutzungsoberfläche des Programmiersystems, welche von den Teilnehmer:innen der Studie bedient wurde. Die Benutzungsoberfläche enthält alle Eigenschaften und Funktionen, welche in den vorhergehenden Kapiteln beschrieben wurden. Die Benutzungsoberfläche ist dafür entwickelt worden, auf Touchscreens ausgeführt zu werden. Für die Studie wurde das Programmiersystem auf einem Microsoft Surface Pro [Microsoft] betrieben, welches mittels Touch-Eingaben oder über die Fernsteuerfunktion von Microsoft Teams bedient wurde.

(a) (b)

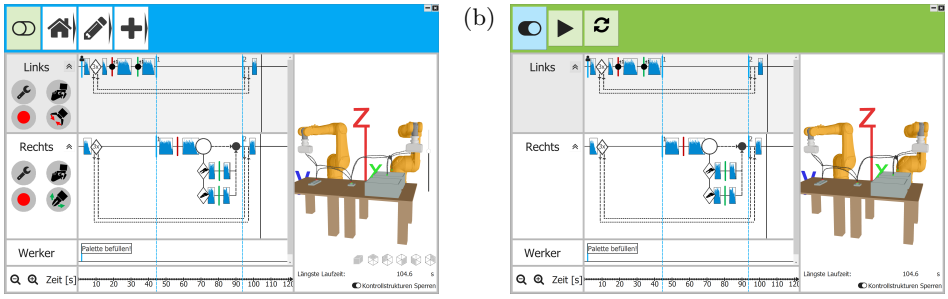


Abbildung 8.2: Die Benutzungsoberfläche des entstandenen Prototypen des Programmiersystems. Abbildung (a) zeigt die Oberfläche im Programmiermodus, Abbildung (b) im Ausführungsmodus.

Die Benutzungsoberfläche ist dreigeteilt aufgebaut. Oben befindet sich die *Kontrollleiste* mit Knöpfen um die jeweiligen Funktionen auf den ausgewählten Zeitpunkten oder Bereichen der Zeitleisten auszuführen. Unten links sind die *Zeitleisten* für die einzelnen Roboter und den Menschen in der geschichteten Darstellung angeordnet. Zuletzt befindet sich noch unten rechts ein *Simulationsfenster*, in dem die Robotermodelle und die Umgebung des Roboters angezeigt werden, und zugleich jeweils eine 3D-Darstellung der zugehörigen Roboterprogramme angezeigt wird.

Die Kontrollleiste kann zwei Zustände einnehmen, den Programmiermodus (Abbildung 8.2a) und den Ausführungsmodus (Abbildung 8.2b). Der erste Knopf links dient dazu, zwischen den beiden Modi zu wechseln. Durch die Farbgebung des Hintergrunds der Kontrollleiste ist sichergestellt, dass schnell erkennbar ist, in welchem Modus sich das Programmiersystem befindet. Ist der Hintergrund blau, so befindet es sich im Programmiermodus und die Roboterprogramme können in den Zeitleisten editiert und neu programmiert werden. Ist ein grüner Hintergrund sichtbar, so befindet sich das System im Ausführungsmodus und es können keine Änderungen mehr an den Roboterprogrammen vorgenommen werden. Im Ausführungsmodus ist es nur möglich, das Abspielen der Roboterprogramme zu starten und die automatische Wiederholung der Roboterprogramme einzuschalten.

Im Programmiermodus werden die einzelnen Funktionen der Kontrollleiste, welche in den vorhergehenden Kapiteln erläutert wurden, je nach Art gruppiert. Die erste Gruppierung enthält die Dateifunktionen. Mit diesen Funktionen kann ein leeres Roboterprogramm angelegt, ein gespeichertes geladen oder ein editiertes Roboterprogramm gespeichert werden. Zusätzlich gibt es die Funktion, eine Zeitleiste für einen Menschen zu den Zeitleisten hinzuzufügen, um die Mensch-Roboter Synchronisation zu verwenden. Die zweite Gruppierung umfasst die Editierfunktionen aus Kapitel 5. Darunter fallen Kopieren, Einfügen, Ausschneiden und Löschen. Mit der letzten Grup-

pierung werden die erweiterten Funktionen gruppiert, welche in den Kapiteln 6 und 7 vorgestellt wurden. Diese beinhalten: Einfügen einer Schleife und eines Schleifen-Inkrements; Einfügen einer Verzweigung; Einfügen einer Roboter-Roboter Synchronisation; Einfügen einer Mensch-Roboter Synchronisation.

In den Zeitleisten werden die kinästhetisch demonstrierten Roboterprogramme in der geschichteten, skalierbaren Darstellung aus Kapitel 4 dargestellt. Zusätzlich werden links neben den Zeitleisten für die Roboter noch die Roboter-spezifischen Funktionen angezeigt. Darunter fallen öffnen/schließen des Greifers, aktivieren/deaktivieren des kinästhetischen Führens, Aufzeichnung eines neuen Roboterprogramms beginnen und Roboter-spezifische Einstellungen öffnen. In diesen Einstellungen können Parameter für die Roboter festgelegt werden, so zum Beispiel maximale Beschleunigung und maximale Geschwindigkeit. Die Zeitleisten der Roboter können ein- und ausgeklappt werden, um im ausgeklappten Zustand die Bewegungsbefehle in Graphdarstellung anzuzeigen und im eingeklappten Zustand die Blockdarstellung anzuzeigen. Zusätzlich kann für ein präziseres Editieren in den Zeitleisten gezoomt und nach links und rechts gescrollt werden.

Das Simulationsfenster stellt die Modelle der Roboter dar, welche mit dem Programmiersystem verbunden sind. Sobald ein Roboter ein kinästhetisch demonstriertes Roboterprogramm geladen hat, wird dessen Trajektorie im Simulationsfenster durch schwarze Würfel für jede Konfiguration angezeigt. Greiferbefehle werden in den gleichen Farben wie in der Zeitleiste - grün für öffnen und rot für schließen - angezeigt. Im unteren Bereich des Simulationsfensters gibt es noch Knöpfe für Standardansichten und ein Informationsfeld, in welchem die Laufzeit des längsten Roboterprogramms angezeigt wird. Die Benutzer:innen des Programmiersystems können den Blickwinkel im Simulationsfenster nach belieben verändern, um die für sie am besten geeignete Ansicht zu finden.

8.1.3 Architektur

Zusätzlich zum Aufbau der graphischen Benutzungsoberfläche des Programmiersystems ist auch die Architektur hinter dem Programmiersystem wichtig. Dabei geht es konkret darum, wie die einzelnen Komponenten miteinander verbunden sind und wie die Kommunikation zwischen diesen aussieht. Die einzelnen Komponenten des Programmiersystems sind zum einen der PC oder der Tablet-PC, auf dem die graphische Benutzungsoberfläche dargestellt wird und zum anderen die Roboter mit ihren Steuerungen und den damit verbundenen Steuerungs-PCs. Die grundlegende Architektur ist eine Client-Server Architektur, wobei der PC oder der Tablet-PC mit der Benutzungsoberfläche den Server darstellt und die einzelnen Roboter mit ihren Steuerungen und den Steuerungs-PCs als Clients fungieren. Schematisch ist dies in Abbildung 8.3 dargestellt.

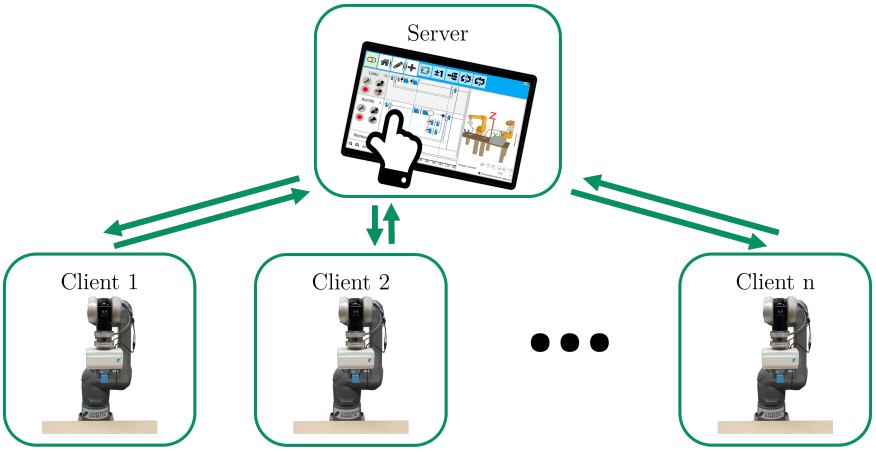


Abbildung 8.3: Architektur des Programmiersystems. Die graphische Benutzungsoberfläche übernimmt die Funktion des Servers. Mit diesem sind beliebig viele Clients in Form von Robotern verbunden. Die Kommunikation erfolgt nur zwischen Server und Client, nicht zwischen Client und Client.

Der Server, also die graphische Benutzungsoberfläche, hat in der Architektur die Aufgabe, die angemeldeten Clients zu verwalten und ihnen Befehle zu schicken. Dies ist unter anderem daran zu erkennen, dass je angemeldetem Client eine neue Zeitleiste in der Benutzungsoberfläche hinzugefügt wird. Durch das Verschicken der Befehle an die Clients ist der Server der zentrale Bestandteil des Programmiersystems. Beispielhafte Befehle sind, dass ein neues kinästhetisch demonstriertes Roboterprogramm aufgezeichnet werden soll, oder dass ein vorhandenes Roboterprogramm abgespielt werden soll.

Die Clients, also die Roboter, welche mit dem Programmiersystem angesteuert werden sollen, melden sich nach dem Start des Client-Programms am Server an. Dadurch kennt der Server zu jedem Zeitpunkt, wie viele und welche Roboter zur Verfügung stehen. Anschließend warten Clients auf Steuerbefehle des Servers. Die Logik des Programmiersystems befindet sich im Server, die Clients hingegen reagieren nur auf die Befehle des Servers und besitzen die Fähigkeit, neue kinästhetisch demonstrierte Roboterprogramme aufzuzeichnen und vorhandene Roboterprogramme abzuspielen. Das Editieren und Erweitern der Roboterprogramme findet ausschließlich auf dem Server statt. Um einen neuen Robotertyp zu unterstützen, muss für diesen lediglich das Client-Programm mit der Schnittstelle zum Server implementiert werden. Anschließend kann der neue Robotertyp mit dem Programmiersystem verwendet werden.

Über das Netzwerk werden in serialisierter Form Roboterprogramme, Roboter-spezifische Parameter und Steuerbefehle geschickt. Die Roboterprogramme werden von einem Client zum Server geschickt, nachdem ein neues Roboterprogramm aufgezeichnet wurde, oder vom Server zum Client, nachdem auf dem Server ein vorhandenes Roboterprogramm editiert wurde. Roboter-spezifische Parameter, wie die maximale Beschleunigung und maximale Geschwindigkeit, werden zwischen Client und Server ausgetauscht, wenn diese sich verändert haben. Mit Steuerbefehlen kann zum Beispiel die Aufzeichnung eines neuen Roboterprogramms oder das Abspielen eines alten Roboterprogramms gestartet werden, oder auch das am Roboter montierte Werkzeug bedient werden.

Als Netzwerk zwischen Server und Clients wird ein TCP/IP Netzwerk über Ethernet und WLAN verwendet. Kommunikationskanäle bestehen nur zwischen Server und Client, nicht jedoch zwischen Client und Client. Somit ist der Server für die Koordination der Clients zuständig, wohingegen die Clients nur auf die Befehle des Servers reagieren. Durch die Client-Server Architektur wird dafür gesorgt, dass mit einem Server konzeptionell beliebig viele Clients koordiniert werden können.

8.2 Studie

Die im Folgenden vorgestellte Studie soll eine Antwort auf Forschungsfrage F6 geben und zeigen, ob das Programmiersystem aus dieser Arbeit intuitiv zu bedienen ist. Zusätzlich soll mit der Studie herausgefunden werden, welche Aspekte des Programmiersystems verbessert werden sollen und ob bzw. welche zusätzlichen Funktionen von den Teilnehmer:innen gewünscht sind. Dazu wird zunächst in Abschnitt 8.2.1 das Studiendesign vorgestellt. Anschließend wird in Abschnitt 8.2.2 die Gesamtaufgabe, welche die Teilnehmer:innen der Studie programmieren mussten, inklusive der einzelnen Teilaufgaben näher erläutert. Abschließend werden in Abschnitt 8.2.3 die Ergebnisse der Studie, sowohl im Hinblick auf die Intuitivität, als auch auf die zusätzlich evaluierten Fragestellungen vorgestellt.

8.2.1 Studiendesign

Zur Evaluation der Benutzungsoberfläche des Gesamtsystems wurde der in Abschnitt 8.1 beschriebene Stand des Gesamtdemonstrators verwendet. Dadurch, dass die Intuitivität nach Abschnitt 2.3 eine Kombination aus Effektivität, mentaler Effizienz und Zufriedenheit bei der Bedienung eines Systems ist, wurden für die Evaluation die Methoden aus dem Mineric-Toolkit [Orendt2017] verwendet, welche diese drei Teilaspekte der Intuitivität messen. Die verwendeten Fragebögen sind im Anhang aufgelistet. Dort findet sich auch die Beschreibung des Programmiersystems und der auszuführenden Teilaufgaben, welche die Teilnehmer:innen der Studie erhalten haben. Die Studie wurde initial so entworfen, dass diese am realen Robotersystem im Roboti-

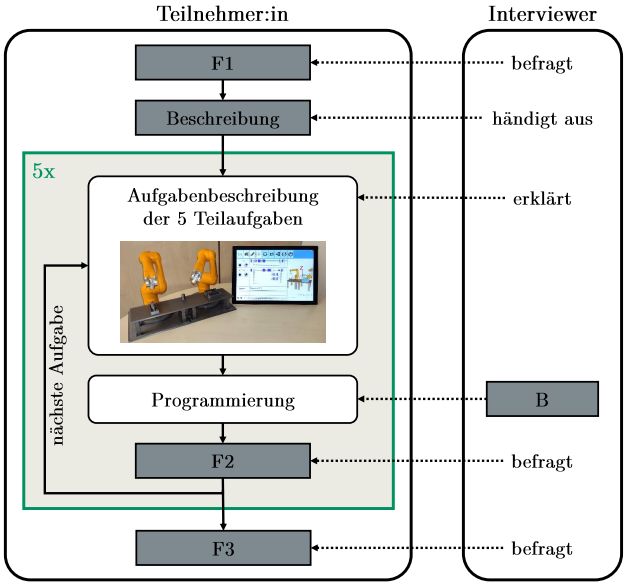


Abbildung 8.4: Ablauf der Studie für eine:n Teilnehmer:in in Anlehnung an Abbildung 2.6b. Die Fragebögen F1, F2 und F3, der Beobachtungsbogen B und auch die Beschreibung, welche die Teilnehmer:innen vor Beginn der Studie erhalten haben, sind im Anhang aufgelistet.

klabor der Universität Bayreuth durchgeführt werden kann. Aufgrund der Corona-Pandemie wurde hierauf verzichtet und die Evaluation mit Hilfe eines virtuellen Roboters dezentral durchgeführt, entweder online über Microsoft Teams oder bei den Studienteilnehmer:innen zu Hause unter Einhaltung der Hygienevorschriften.

Die Studie wurde mit dem Ziel entworfen, dass bei der Bearbeitung der einzelnen Teilaufgaben die Teilnehmer:innen nach und nach alle Aspekte des Programmiersystems verwenden müssen. Dabei wurden die Teilaufgaben so gewählt, dass diese jeweils einen Teil der Funktionalitäten des Programmiersystems evaluieren. Zusätzlich bauen die Teilaufgaben aufeinander auf, so dass das Ergebnis der letzten Teilaufgabe als Start für die nächste Teilaufgabe verwendet werden soll.

Abbildung 8.4 zeigt den Ablauf der Studie für eine:n Teilnehmer:in. Das Szenario, welches für die Evaluation der Aufgabe verwendet wurde, ist in Abbildung 8.5 dargestellt. Bevor die Teilnehmer:innen Informationen zum Programmiersystem oder zur Studie erhielten, musste Fragebogen F1 ausgefüllt werden. Danach wurde die Beschreibung des Programmiersystems und der zu erledigenden Gesamtaufgabe ausgehändigt. Anschließend wurden nacheinander die fünf Teilaufgaben erklärt. Diese mussten dann von den Teilnehmer:innen mit Hilfe des Programmiersystems gelöst werden. Parallel dazu mussten die Teilnehmer:innen je Teilaufgabe eine Frage von Fragebogen F2 ausfüllen und der Interviewer notierte Beobachtungen auf dem Beobachtungsbogen B.

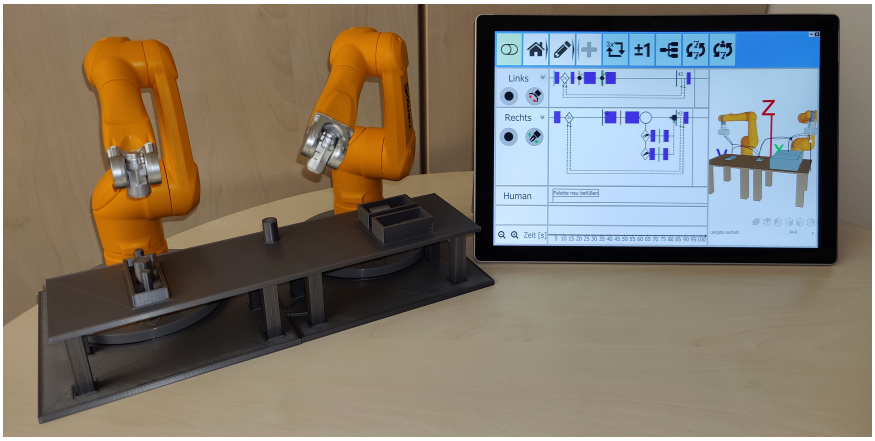


Abbildung 8.5: Evaluationsaufbau für die Teilnehmer:innen der Studie. Das 3D-gedruckte Modell der Umgebung und die beiden Roboter-Modelle sollen den Teilnehmer:innen die Aufgabe und den Aufbau mit realen Robotern im Labor visualisieren. Rechts auf einem Tablet-PC ist das Programmiersystem zu sehen, welches evaluiert wurde.

Nachdem alle Teilaufgaben von den Teilnehmer:innen programmiert wurden, musste zum Abschluss der Studie Fragebogen F3 ausgefüllt werden. Im Folgenden werden die Inhalte der Fragebögen F1, F2 und F3, des Beobachtungsbogens B und der Beschreibung näher erläutert.

In Fragebogen *F1* werden die Teilnehmer:innen zunächst nach demographischen Daten wie Alter, Geschlecht, Schulabschluss und die Arbeit in einem technischen Beruf gefragt. Anschließend werden die Vorkenntnisse im Bereich Audio-/Videoschnitt und Robotik erhoben. Diese werden dazu verwendet, um die Teilnehmer:innen gruppieren zu können. Teilnehmer:innen, die nicht in einem technischen Beruf tätig sind, werden als *Nicht-Expert:innen* eingestuft. Teilnehmer:innen in einem technischen Beruf, jedoch ohne Robotik-Kenntnisse werden als *Domänen-Expert:innen* eingestuft, Teilnehmer:innen mit Robotik-Kenntnissen als *Robotik-Expert:innen*. Mit dieser Gruppierung kann im späteren Verlauf festgestellt werden, ob Unterschiede in der Bewertung des Systems je nach Gruppe vorliegen. Anschließend wird die erwartete Komplexität des Programmiersystems auf einer Skala von -5 (sehr kompliziert) bis $+5$ (sehr einfach) erhoben. Im letzten Teil von F1 müssen die Teilnehmer:innen die Symbole, welche für die Funktionalitäten des Programmiersystems verwendet werden, mit einem Stichwort beschreiben. Diese Information soll genutzt werden, um zu erkennen, welche Symbole im Kontext der Roboterprogrammierung ohne Erklärung verständlich sind und welche Symbole noch überarbeitet werden müssen.

Nachdem Fragebogen F1 ausgefüllt wurde, wird den Teilnehmer:innen die *Beschreibung* des Programmiersystems und der Teilaufgaben ausgehändigt. In dieser wird zunächst das Szenario kurz beschrieben und die Gesamtaufgabe erläutert. Anschließend wird zu den Symbolen, welche in Fragebogen F1 beschrieben werden sollten, die rich-

tige Funktionalität genannt. Der letzte Teil der Beschreibung erklärt textuell die fünf Teilaufgaben, welche die Teilnehmer:innen im Verlauf der Studie programmieren sollen. Diese Beschreibung kann während der Bearbeitung der Teilaufgaben von den Teilnehmer:innen jederzeit zu Hilfe gezogen werden.

Auf dem Beobachtungsbogen *B* werden während der Abarbeitung der Teilaufgaben vom Interviewer Notizen zum Vorgehen der Teilnehmer:innen angefertigt. Dabei enthält der Beobachtungsbogen zunächst allgemeine Informationen zur Studie, wie Datum, Beginn, Ende, Dauer, den Namen des Interviewers und allgemeine Anmerkungen. Darüber hinaus wird für jede Teilaufgabe der Beginn, das Ende, die Dauer, Beobachtungen und ein Wert für Genauigkeit und Vollständigkeit erhoben. Die Kombination aus Genauigkeit und Vollständigkeit dient als Wert für die *Effektivität* der Bedienung des Programmiersystems durch die Teilnehmer:innen. Der Wert für die Genauigkeit liegt zwischen 0 und 3 Punkten, wobei ein höherer Wert aussagt, dass die Benutzung des Programmiersystems zielgerichteter und bewusster durch die Teilnehmer:innen erfolgte. Die Vollständigkeit wird auf einer Skala zwischen 0 und 2 Punkten gemessen, wobei ein höherer Wert bedeutet, dass das Ziel vollständig erreicht wurde. Aus den beiden Werten wird nach [Orendt2017] der Wert für die Effektivität des Programmiersystems berechnet. Die Kriterien für die Bepunktung der Genauigkeit und Vollständigkeit können dem Stand der Forschung (Kapitel 2.3.2, Abbildung 2.5) entnommen werden.

Nach Abschluss jeder Teilaufgabe wird von den Teilnehmer:innen in Fragebogen *F2* die mentale Beanspruchung als Wert für die *mentale Effizienz* des Programmiersystems bei dieser Teilaufgabe eingetragen. Die mentale Beanspruchung wird auf der Skala zur Erfassung subjektiv erlebter Anstrengung (SEA-Skala) mit Werten zwischen 0 und 220 gemessen. Einzelne Werte sind dabei annotiert mit einer textuellen Beschreibung wie anstrengend dieser Wert einzustufen ist. Beispiele für die Annotationen sind 20 mit *kaum anstrengend*, 119 mit *ziemlich anstrengend* und 205 mit *außerordentlich anstrengend*. Die vollständige Skala ist im Stand der Forschung (Kapitel 2.3.2, Abbildung 2.6a) dargestellt. Mit den erhobenen Werten aus F2 wird in der Auswertung ein Wert für die mentale Effizienz berechnet.

Zum Ende der Studie muss Fragebogen *F3* ausgefüllt werden, welcher zunächst die Zufriedenheit der Teilnehmer:innen mit dem Programmiersystem mit Hilfe des *QUESI*-Fragebogens erhebt. Dieser misst die subjektiven Konsequenzen intuitiver Benutzung als Wert für die *Zufriedenheit*. Dabei werden 14 positiv formulierte Fragen in 5 Kategorien gestellt, welche mit Werten von 1 (trifft gar nicht zu) bis 5 (trifft völlig zu) beantwortet werden. Je höher der QUESI-Wert, desto zufriedener sind die Teilnehmer:innen mit dem evaluierten System. Näheres zum QUESI-Fragebogen ist im Stand der Forschung (Kapitel 2.3.2) zu finden. Neben dem QUESI-Wert für die Zufriedenheit der Teilnehmer:innen wird auch die empfundene Komplexität erhoben, welche in Relation zur erwarteten Komplexität gestellt und ausgewertet wird. Zusätzlich wird erhoben, welche der beiden Darstellungsarten für Bewegungen in den Zeitleisten (Graphdarstellung oder Blockdarstellung) von den Teilnehmer:innen bevorzugt

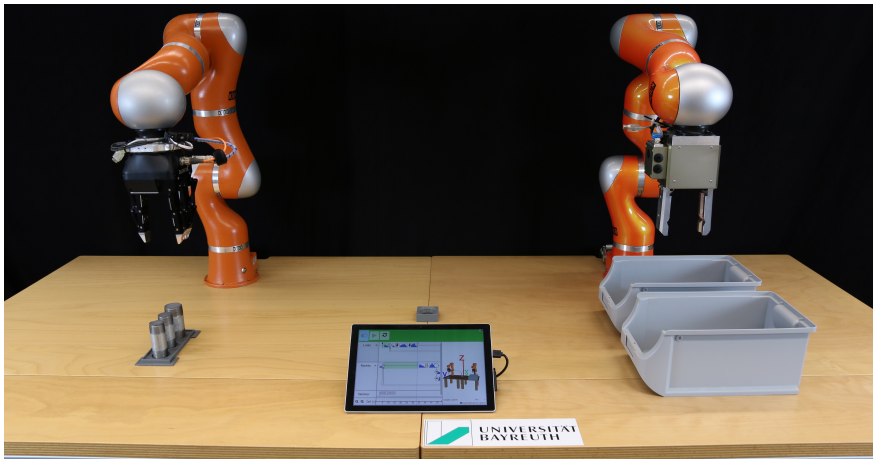


Abbildung 8.6: Der Aufbau der Gesamtaufgabe am realen Roboter. Links ist die Palette mit den Werkstücken, in der Mitte die Übergabeposition und rechts die beiden Kisten zum Sortieren der Werkstücke. Auf dem Tablet-PC wird das Programmiersystem ausgeführt.

wird und ob ein Kopieren und mehrfaches Einfügen oder eine Zählschleife zum realisieren von sich wiederholenden Aufgaben bevorzugt wird. Abschließend besteht für die Teilnehmer:innen noch die Möglichkeit, in einem Freitextfeld Kommentare und Anmerkungen zum Programmiersystem einzutragen.

8.2.2 Gesamtaufgabe und Teilaufgaben

Für die Evaluation der Benutzungsoberfläche des Programmiersystems wurde eine Aufgabe entwickelt, welche alle in dieser Arbeit entworfenen und vorgestellten Aspekte der kinästhetischen Roboterprogrammierung beinhaltet. Die Aufgabe muss an einem Mehrrobotersystem mit zwei Robotern („Links“ und „Rechts“), an welchen jeweils ein Greifer befestigt ist, programmiert werden. Die *Gesamtaufgabe*, welche in mehreren Teilaufgaben von den Teilnehmer:innen erledigt werden soll, ist eine Übergabe von Werkstücken vom linken zum rechten Roboter mit anschließender Sortierung der Objekte nach deren Größe. Dabei soll der linke Roboter die Werkstücke von einer Palette aufgreifen und an der Übergabeposition in der Mitte zwischen den beiden Robotern ablegen. Anschließend soll der rechte Roboter das Werkstück von der Übergabeposition aufgreifen und abhängig davon, ob es ein großes (Durchmesser: 41 mm) oder kleines (Durchmesser: 33 mm) Objekt ist, dieses in eine der beiden vorhandenen Kisten legen. Wenn die Palette abgearbeitet ist, soll der linke Roboter automatisch eine Rückmeldung an eine:n Werker:in geben, damit eine neue Palette gebracht wird. Der reale Aufbau im Robotiklabor mit den beiden Robotern, der Palette links, der Übergabeposition in der Mitte und den beiden Kisten rechts ist in Abbildung 8.6 dargestellt.

Mit *Teilaufgabe 1 (TA1)* sollen die Teilnehmer:innen an den Aufbau und das Verhalten des Programmiersystems herangeführt werden. Hierfür soll zunächst der rechte Roboter so programmiert werden, dass dieser zur Übergabeposition fährt, dort ein Werkstück aufgreift, und dieses anschließend in eine der Kisten wirft. Um diese Aufgabe abzuschließen, muss als erstes die Aufzeichnung eines neuen Roboterprogramms für „Rechts“ begonnen werden. An einem realen Roboter würde anschließend die Bewegung des Roboters demonstriert werden. Da die Evaluation nur mit Hilfe eines virtuellen Roboter durchgeführt wurde, musste dieser Schritt entfallen. Abschließend muss die Aufzeichnung des Roboters beendet werden und das aufgezeichnete Roboterprogramm sollte von den Teilnehmer:innen in der Benutzeroberfläche inspiziert werden. Dadurch, dass mit einem virtuellen Roboter gearbeitet wurde, wurde statt eines demonstrierten Roboterprogramms ein vorher am realen Robotersystem aufgezeichnetes Roboterprogramm an das Programmiersystem gesendet.

Mit *Teilaufgabe 2 (TA2)* sollen die Editierfunktionen des Programmiersystems evaluiert werden. Dabei soll das Roboterprogramm aus TA1 so verändert werden, dass es insgesamt drei Werkstücke an der Übergabeposition aufgreift und in eine Kiste wirft. Hierfür sollen die Teilnehmer:innen die Editierfunktion in Form von Kopieren und Einfügen verwenden. Damit soll der Teil des Roboterprogramms, der wiederholt ausgeführt werden soll, kopiert und zwei Mal eingefügt werden.

Mit *Teilaufgabe 3 (TA3)* sollen sowohl die sensorbasierte Verzweigungsfunktion, als auch die Schleifenfunktion evaluiert werden. Dafür soll in das Roboterprogramm von „Rechts“ nach TA1 mit Hilfe der Verzweigungsfunktion das Sortieren der Werkstücke nach großen und kleinen Werkstücken programmiert werden. Zusätzlich soll mit Hilfe der Schleifenfunktion eine Zählschleife eingefügt werden, damit die Sortierbewegung insgesamt drei Mal ausgeführt wird. Dazu muss zunächst an der richtigen Stelle des Roboterprogramms eine Verzweigung bestehend aus 2 Zweigen eingefügt werden. Ein Zweig behandelt hierbei den Fall für die großen Werkstücke mit 41 mm Durchmesser, einer für die kleinen Werkstücke mit 33 mm Durchmesser. Anschließend muss um den zu wiederholenden Bereich eine Zählschleife eingefügt werden, welche drei Mal ausgeführt wird. Mit erfüllen dieser Aufgabe ist das Roboterprogramm für „Rechts“ zunächst fertig.

Mit *Teilaufgabe 4 (TA4)* sollen Schleifen in Kombination mit einem Schleifeninkrement evaluiert werden. Hierfür soll in ein vorgegebenes Roboterprogramm von „Links“, welches ein Werkstück auf der ersten Position der Palette aufgreift und auf der Übergabeposition ablegt, eine Schleife mit einem Schleifeninkrement eingefügt werden. Die Schleife wird benötigt, um drei Werkstücke zur Übergabeposition zu befördern. Anschließend muss mit dem Schleifen-Inkrement definiert werden, dass nach jedem Schleifendurchlauf die nächste Position auf der Palette angefahren wird. Ist diese Teilaufgabe fertig programmiert, so kann im Simulationsfenster des Programmiersystems gesehen werden, dass beide Roboter an der Übergabeposition kollidieren können, wenn diese nicht abgesichert wird.

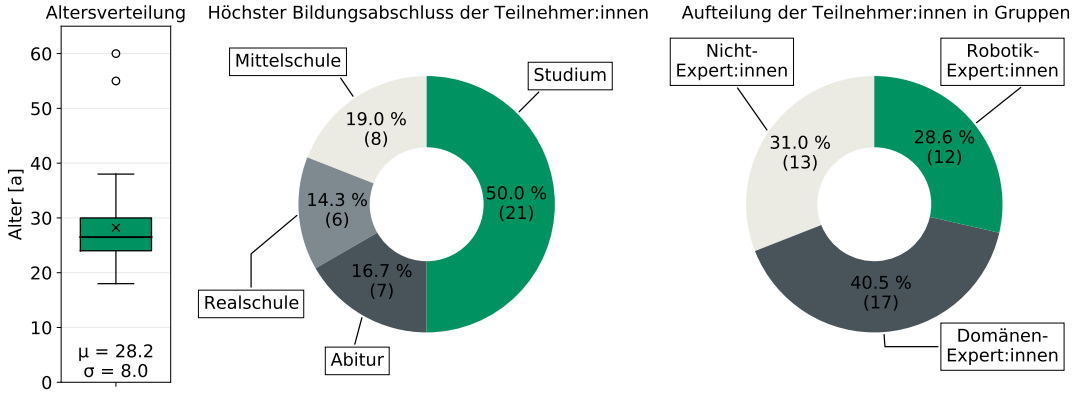


Abbildung 8.7: Alter, Bildungsabschluss und Aufteilung der Studienteilnehmer:innen in Gruppen je nach Robotik-Vorkenntnissen.

Mit *Teilaufgabe 5 (TA5)* soll sowohl die Roboter-Roboter-, als auch die Mensch-Roboter-Synchronisation des Programmiersystems evaluiert werden. Dafür muss mit Roboter-Roboter-Synchronisationspunkten in beiden Roboterprogrammen die Übergabeposition so abgesichert werden, dass diese immer nur von einem Roboter gleichzeitig belegt werden kann. Anschließend muss von den Teilnehmer:innen der Studie noch eine Mensch-Roboter-Synchronisation eingefügt werden, damit zu Beginn jeder Ausführung der Roboterprogramme sichergestellt ist, dass eine volle Palette vor dem linken Roboter steht. Sind alle fünf Teilaufgaben erfolgreich bearbeitet worden, so können die entstandenen Roboterprogramme im Ausführungsmodus des Programmiersystems mit Hilfe der virtuellen Roboter im Simulationsfenster abgespielt werden. Dabei können die Teilnehmer:innen noch einmal überprüfen, ob die einzelnen Teilaufgaben erfolgreich bearbeitet wurden. Eine beispielhafte Bildschirmaufnahme der fertig programmierten Gesamtaufgabe ist in Abbildung 8.2 dargestellt.

8.2.3 Evaluationsergebnisse

Die Studie wurde vom 14. Dezember 2020 bis zum 22. Januar 2021 mit 16 Teilnehmer:innen (38.1%) online über Microsoft Teams und mit 26 Teilnehmer:innen (61.9%) bei den Teilnehmer:innen zu Hause durchgeführt. Insgesamt nahmen 42 Teilnehmer:innen an der Studie teil. Die Altersverteilung, der höchste Bildungsabschluss und die Aufteilung in die drei Gruppen ist in Abbildung 8.7 dargestellt. Alle Teilnehmer:innen waren zwischen 18 und 60 Jahre alt, mit einem Mittelwert von $\mu = 28.2$ Jahre bei einer Standardabweichung von $\sigma = 8.0$ Jahren. Es nahmen 10 Frauen (23.8%) und 32 Männer (76.2%) an der Studie Teil, wobei 8 Personen (19.0%) als höchsten Bildungsabschluss einen Mittelschulabschluss besaßen, 6 (14.3%) einen Realschulabschluss, 7 (16.7%) Abitur und 21 (50.0%) ein abgeschlossenes Studium.

Aufgrund der angegebenen Vorkenntnisse in Fragebogen F1 wurden alle Teilnehmer:innen, wie in Abschnitt 8.2.1 erläutert, in drei Gruppen eingeteilt. Hieraus soll erkannt werden, ob Personen mit unterschiedlichen Vorkenntnissen das Programmiersystem unterschiedlich bewerten. Die Gruppe der Nicht-Expert:innen umfasste 13 Personen, was einem Anteil von 31.0% aller Teilnehmer:innen entspricht. Die Gruppe der Domänen-Expert:innen war mit 17 Personen und einem Anteil von 40.5% die größte Gruppe. In die Gruppe der Robotik-Expert:innen wurden 12 Personen eingeteilt, was einen Anteil von 28.6% entspricht.

Intuitivität

Zur Messung Intuitivität eines Systems müssen, wie im Stand der Forschung in Kapitel 2.3 erklärt, die drei Eigenschaften *Effektivität*, *mentale Effizienz* und *Zufriedenheit* gemessen werden. Der allgemeine Ablauf und die Fragebögen dafür wurden in Abschnitt 8.2.1 erläutert. Zusätzlich zu Effektivität, Effizienz und Zufriedenheit wurde die Einschätzung der Teilnehmer:innen zur erwarteten und tatsächlichen Komplexität des Programmiersystems erhoben. Durch die Kombination dieser vier Werte kann eine Aussage über die Intuitivität des Systems erfolgen.

Die Ergebnisse der Vollständigkeit und Genauigkeit zur Messung der *Effektivität* der Benutzungsoberfläche aufgeteilt nach Gruppen sind in Abbildung 8.8 als Boxplots dargestellt. Der Gesamtwert für die Effektivität des Programmiersystems über alle Gruppen beträgt $\mu = 85.0\%$ bei einer Standardabweichung $\sigma = 10.1\%$. Im Vergleich dazu wurde in der Evaluation einer früheren Version des Programmiersystems im Jahr 2016 in Abschnitt 6.3.2 für die Effektivität ein Wert von 70.0% erreicht. Somit ist die Benutzungsoberfläche im Vergleich zur Messung aus 2016 hinsichtlich Effektivität deutlich verbessert worden und kann als sehr gut eingestuft werden.

Bei betrachten der einzelnen Gruppen ist zu erkennen, dass die Robotik-Expert:innen einen um ca. 6% höheren Wert für die Effektivität erzielten als die Nicht-Expert:innen und die Domänen-Expert:innen. Die Robotik-Expert:innen haben demnach eine höhere Vollständigkeit und Genauigkeit bei der Bedienung des Programmiersystems erreicht. Dies könnte damit erklärt werden ist, dass diese schon vor der Teilnahme an der Studie mit Robotersystemen und deren Programmierung vertraut waren.

Beim Vergleich der Effektivitätswerte der einzelnen Teilaufgaben fällt auf, dass für alle Gruppen die Effektivität zwischen TA1 und TA2 ansteigt, dann deutlich abfällt, um bei TA4 und TA5 wieder anzusteigen. Der Abfall ist durch die in TA3 erstmalig eingeführten sensorbasierten Kontrollstrukturen erklärbar, welche als erstes von den komplexeren und für Nicht-Expert:innen und Domänen-Expert:innen neuen Konzepten angewendet werden. Der anschließende Anstieg des Effektivitätswertes zu TA4 und TA5 bei allen Gruppen zeigt jedoch, dass nach Verwenden der sensorbasierten Kontrollstrukturen ein Lerneffekt einsetzt.

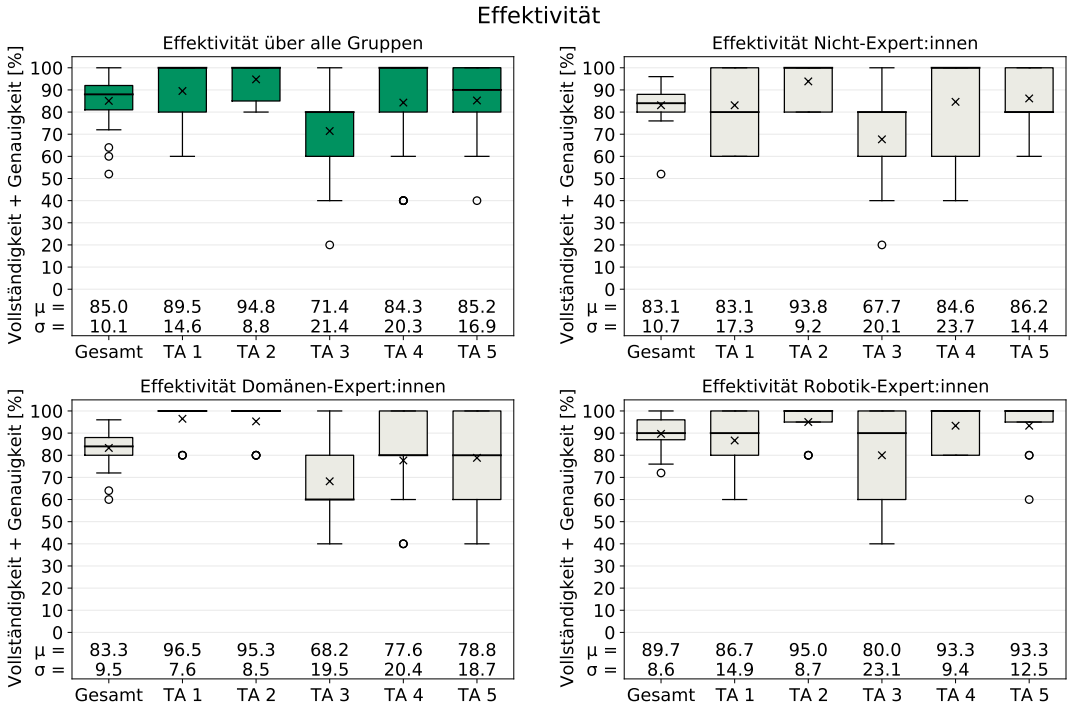


Abbildung 8.8: Ergebnisse für die Messung der Vollständigkeit und Genauigkeit als Wert für die Effektivität der Benutzungsoberfläche des Programmiersystems aufgeteilt nach Gruppen jeweils für die Gesamtaufgabe und die Teilaufgaben TA1 bis TA5.

Die Ergebnisse der mentalen Beanspruchung zur Messung der *mentalen Effizienz* aufgeteilt nach Gruppen sind in Abbildung 8.9 als Boxplots dargestellt. Die Teilnehmer:innen bewerteten die mentale Effizienz über alle Gruppen im Mittel mit $\mu = 43.2$ bei einer Standardabweichung von $\sigma = 17.5$ auf der SEA-Skala von 0 (nicht anstrengend) bis 220 (extrem anstrengend). Dies kann direkt übersetzt werden in „etwas anstrengend“. Im Vergleich zur Evaluation 2016 in Abschnitt 6.3.2, bei der für die mentale Effizienz ein Wert von 54.9 erreicht wurde zeigt sich auch hier eine deutliche Verbesserung, obwohl das Programmiersystem in der Zwischenzeit mehr Funktionen erhalten hat.

Beim betrachten der einzelnen Gruppen fällt auf, dass die Gruppe der Robotik-Expert:innen die mentale Effizienz im Mittel um 6 Punkte besser bewertet haben als die Domänen-Expert:innen. Diese wiederum haben die mentale Effizienz im Mittel auch um 6 Punkte besser bewertet als die Nicht-Expert:innen. Die Nicht-Expert:innen empfanden die Benutzung der graphischen Oberfläche demnach leicht anstrengender als die Domänen-Expert:innen und die Robotik-Expert:innen. Dies könnte analog zur Effektivität damit erklärt werden kann, dass Nicht-Expert:innen und Domänen-Ex-

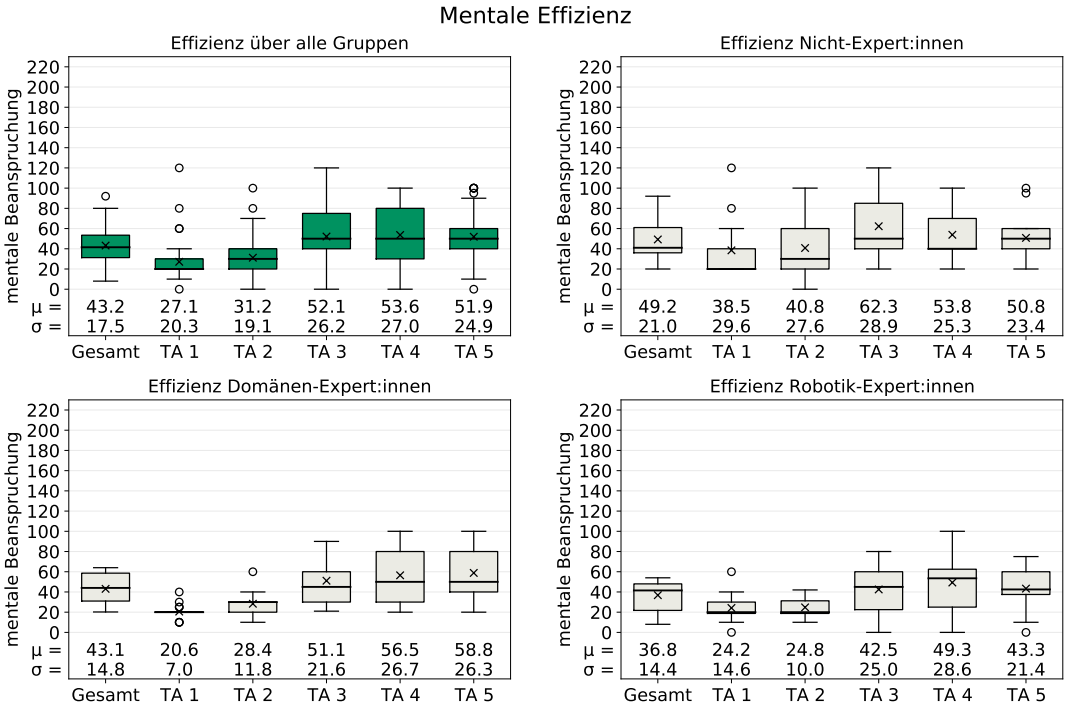


Abbildung 8.9: Ergebnisse für die Messung der mentalen Beanspruchung als Wert für die mentale Effizienz des Programmiersystems aufgeteilt nach Gruppen jeweils für die Gesamtaufgabe und die Teilaufgaben TA1 bis TA5.

pert:innen vorher noch kein Robotersystem bedient haben und dementsprechend keine Referenzwerte für andere Robotersysteme kennen. Auch ist denkbar, dass für Nicht-Expert:innen und Domänen-Expert:innen die Bedienung einer graphischen Benutzungsoberfläche im Allgemeinen anstrengender ist.

Beim Vergleich der Effizienzwerte der einzelnen Teilaufgaben ist erkennbar, dass die mentale Beanspruchung über alle Gruppen bei TA1 und TA2 nahezu identisch ist. Zwischen TA2 und TA3 gibt es bei allen Gruppen einen deutlichen Sprung um ca. 20 Punkte. Bei TA3, TA4 und TA5 zeigt sich im Mittel über alle Gruppen eine gleichbleibende mentale Beanspruchung. Bei den einzelnen Gruppen zeigen sich ebenfalls unterschiedliche Ergebnisse. Bei den Nicht-Expert:innen fällt die Bewertung für die mentale Beanspruchung nach TA3 wieder leicht ab bis TA5. Bei den Domänen-Expert:innen hingegen steigt diese zwischen TA3 und TA5 leicht an, und bei den Robotik-Expert:innen steigt die mentale Beanspruchung zwischen TA3 und TA4 leicht an um dann bei TA5 wieder auf das Niveau von TA3 zu fallen. Das Bild, dass TA1 und TA2 immer deutlich weniger mental anstrengend als TA3, TA4 und TA5 bewertet

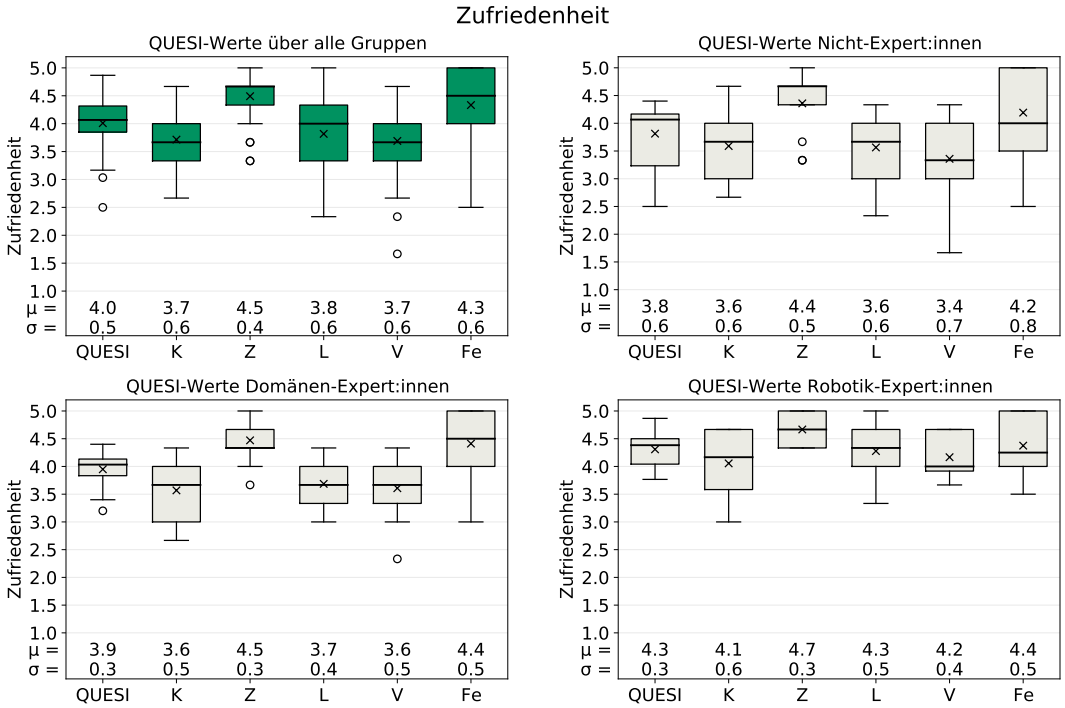


Abbildung 8.10: Ergebnisse für die Messung der subjektiven Konsequenzen intuitiver Benutzung (QUESI) als Wert für die Zufriedenheit mit dem Programmiersystem für alle Gruppen. Die Werte sind aufgeteilt in den Gesamtwerte (QUESI) und die einzelnen Subskalen des QUESI. Diese sind: wahrgenommene kognitive Beanspruchung (K), wahrgenommene Zielerreichung (Z), wahrgenommener Lernaufwand (L), Vertrautheit/Vorwissen (V) und wahrgenommene Fehlerrate (Fe)

werden, ist über alle Gruppen zu erkennen. Dies ist wie bei der Effektivität damit zu erklären, dass Aufgaben, welche ab TA3 gelöst werden mussten, komplexere Konzepte der kinästhetischen Roboterprogrammierung verwenden und deswegen deutlich mehr mentale Anstrengung von den Teilnehmer:innen notwendig war.

Die Ergebnisse für den dritten Aspekte der Intuitivität, der subjektiven Konsequenzen intuitiver Benutzung (QUESI) zur Messung der *Zufriedenheit*, sind in Abbildung 8.10 als Boxplots dargestellt. Die Teilnehmer:innen bewerteten die Zufriedenheit über alle Gruppen im Mittel mit $\mu = 4.0$ bei einer Standardabweichung von $\sigma = 0.5$ auf der QUESI-Skala von 1 bis 5, wobei 5 der bestmögliche Wert ist. Auch hier zeigt sich im Vergleich zur Evaluation aus 2016 in Abschnitt 6.3.2, bei der für die Zufriedenheit ein QUESI-Wert von 3.6 erreicht wurde, eine deutliche Verbesserung.

Betrachtet man die einzelnen Gruppen, so fällt auf, dass die Gruppe der Robotik-Expert:innen um 0.4 bzw. 0.5 Punkte zufriedener mit dem Programmiersystem waren als die Domänen-Expert:innen und Nicht-Expert:innen. Eine Hypothese für diesen Unterschied bei der Zufriedenheit ist, dass Robotik-Expert:innen wissen, wie kompliziert ein Robotersystem in der Regel zu programmieren ist, und deswegen von dem Programmiersystem positiv überrascht wurden. Nichtsdestotrotz ist eine Bewertung von 3.8 bei den Nicht-Expert:innen und 3.9 bei den Domänen-Expert:innen immer noch als intuitiv zu interpretieren.

Bei den fünf Kategorien, aus denen der QUESI-Wert zusammengesetzt wird, zeigt sich über alle Gruppen ein ähnliches Bild. Sowohl die *wahrgenommene Zielerreichung (Z)* mit einer mittleren Bewertung von 4.5 als auch die *wahrgenommene Fehlerrate (Fe)* mit einer mittleren Bewertung von 4.3 werden äußerst positiv bewertet. Beide Werte zeigen, dass die Teilnehmer:innen einen Großteil ihrer Ziele mit Hilfe des Programmiersystems erreicht haben und dabei sehr wenige Fehler wahrgenommen wurden. Die anderen drei Kategorien wurden im Mittel mit Werten zwischen 3.7 (*wahrgenommene kognitive Beanspruchung (K)* und *Vertrautheit/Vorwissen (V)*) und 3.8 (*wahrgenommener Lernaufwand (L)*) bewertet. Dies zeigt, dass eine gewisse kognitive Beanspruchung und ein gewisser Lernaufwand bestand. Zugleich war den Teilnehmer:innen nicht alles vertraut. Diese Werte können nichtsdestotrotz als gut interpretiert werden.

Als letzter Baustein für die Messung der Intuitivität wurde die erwartete und die tatsächlich empfundene Komplexität des Programmiersystems erhoben. Die Ergebnisse der erwarteten und tatsächlichen *Komplexität* und der Differenzwert aus beiden ist in Abbildung 8.11 dargestellt. Tatsächliche und erwartete Differenz wurden auf einer Skala von -5 bis $+5$ erhoben, wobei -5 bedeutet, dass das System sehr kompliziert ist und $+5$, dass es sehr einfach ist. Der Differenzwert wurde dabei aus den beiden Werten für die erwartete (K_e) und tatsächliche (K_t) Komplexität als $K_\mathrm{diff} = K_\mathrm{t} - K_\mathrm{e}$ und hat demnach einen Wertebereich von -10 bis $+10$. Positive Werte für die Differenz sind so zu interpretieren, dass das Programmiersystem tatsächlich einfacher zu bedienen war als erwartet.

Betrachtet man die Ergebnisse für die Komplexität über alle Gruppen, so ist zu erkennen, dass im Mittel die Teilnehmer:innen das Programmiersystem um $\mu = 2.5$ Punkte leichter zu bedienen empfanden als vorher erwartet. Nur drei Teilnehmer:innen haben das Programmiersystem tatsächlich komplexer bewertet als erwartet. Auch der absolute Wert für die tatsächliche Komplexität von $\mu = 2.5$ ist als sehr gut einzustufen.

Vergleicht man die Ergebnisse der einzelnen Gruppen miteinander, so erkennt man, dass der Differenzwert sowohl bei den Nicht-Expert:innen, als auch bei den Domänen-Expert:innen um 0.7 bzw. 0.8 Punkte im Mittel höher liegt als bei dem Robotik-Expert:innen. Dies kann damit erklärt werden, dass die Gruppe der Robotik-Expert:innen mit einer erwarteten Komplexität von 1.2 in die Evaluation gegangen ist, wohingegen die Nicht-Expert:innen diese mit -0.8 und die Domänen-Expert:innen mit -0.2 bewertet haben. Dies zeigt, dass die Robotik-Expert:innen schon mit einer höheren Erwartung an die Komplexität des Programmiersystems herangetreten sind.

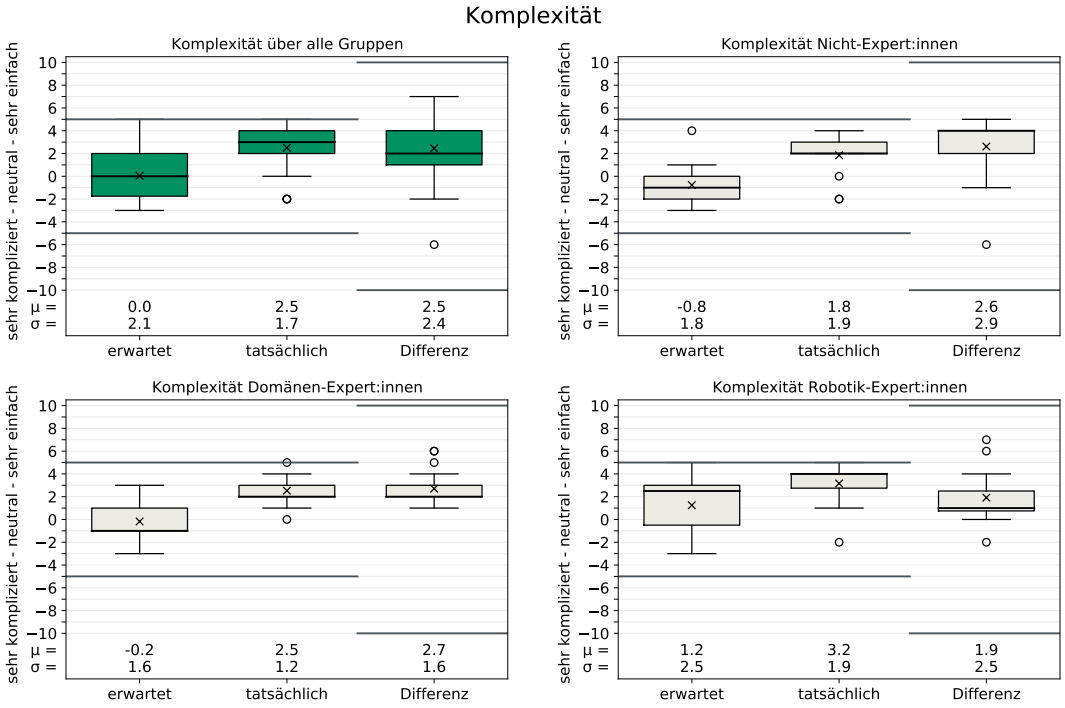


Abbildung 8.11: Ergebnisse für die erwartete und tatsächliche Komplexität des Programmiersystems für alle Gruppen. Zusätzlich wird noch die Differenz zwischen tatsächlicher und erwarteter Komplexität dargestellt, um eine Aussage darüber treffen zu können, ob die einzelnen Teilnehmer:innen das System nach Benutzung als einfacher einschätzen als vorher oder umgekehrt. Die Skala reicht von „sehr kompliziert“ (−5) bis „sehr einfach“ (+5)

Die Evaluation in den vier Kategorien Effektivität, mentale Effizienz, Zufriedenheit und Komplexität hat gezeigt, dass die Benutzungsoberfläche des Programmiersystems in der aktuellen, prototypischen Form intuitiv bedienbar ist. Über alle Kategorien hinweg wurden gute bis sehr gute Ergebnisse erzielt, sodass das Programmiersystem dieser Arbeit als intuitiv eingestuft werden kann. Die Aufteilung der Teilnehmer:innen in Gruppen zeigt, dass trotz kleinerer Unterschiede die einzelnen Gruppen das Programmiersystem ähnlich bewerten und somit die Aussage über die Intuitivität unabhängig von den Vorkenntnissen der Teilnehmer:innen getroffen werden kann.

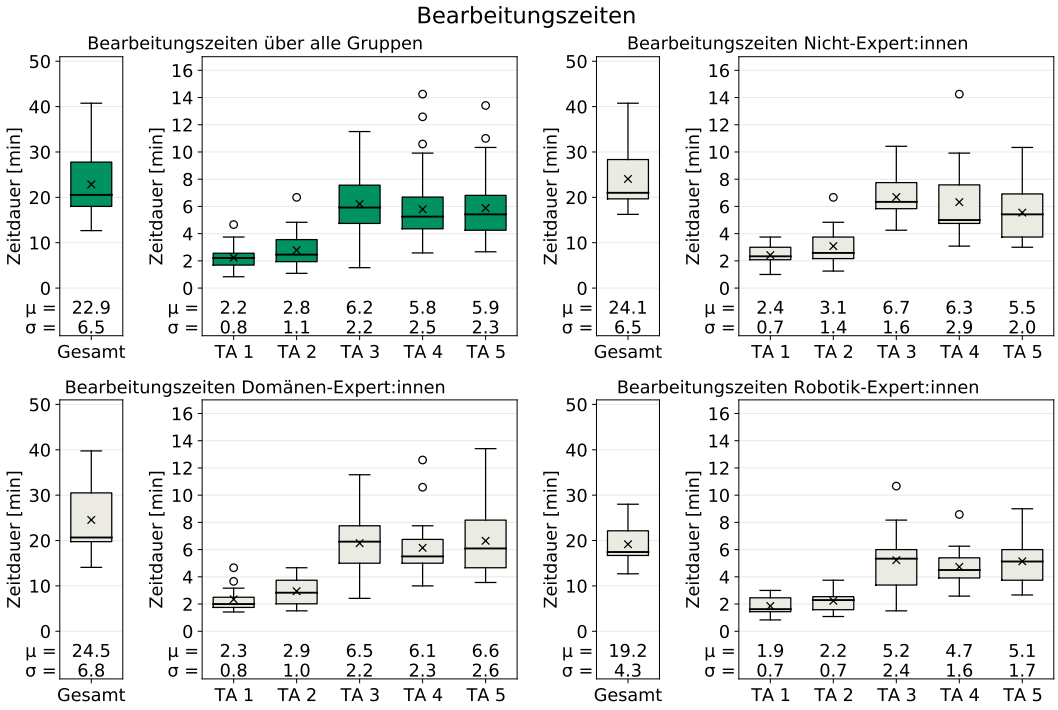


Abbildung 8.12: Ergebnisse für die Bearbeitungszeiten der einzelnen Teilaufgaben TA1 bis TA5 und der gesamten Bearbeitungszeit aller Teilaufgaben für alle Gruppen.

Bearbeitungsdauer

Zusätzlich zur Intuitivität des Programmiersystems wurde die Schnelligkeit der Programmierung in Form der Zeit, welche zum Erfüllen der einzelnen Teilaufgaben benötigt wurde, gemessen. Die Ergebnisse davon sind in Abbildung 8.12 als Boxplots dargestellt. Dabei wurde die Zeit zum Bearbeiten einer Teilaufgabe von Beginn des Durchlesens der Beschreibung der jeweiligen Teilaufgabe bis zur letzten Aktion zum erfüllen der Teilaufgabe gemessen. Die Gesamt-Bearbeitungszeiten werden aus der Summe der Bearbeitungszeiten der Teilaufgaben berechnet.

Über alle Gruppen hinweg wurden für das Bearbeiten von TA1 und TA2 im Mittel $\mu = 2.2$ Minuten bzw. $\mu = 2.8$ Minuten benötigt. Ab TA3 steigt die Bearbeitungszeit deutlich an, was mit den umfangreicheren Aufgaben bei TA3, TA4 und TA5 erklärt werden kann. Dabei wurden für TA3 im Mittel $\mu = 6.2$ Minuten, für TA4 $\mu = 5.8$ Minuten und für TA5 $\mu = 5.9$ Minuten benötigt. Die Gesamtbearbeitungszeit aller Teilaufgaben liegt bei $\mu = 22.9$ Minuten.

Vergleicht man die Bearbeitungszeiten der Gruppen untereinander, fällt auf, dass die Robotik-Expert:innen im Mittel ungefähr 5 Minuten schneller waren als die Gruppen ohne Robotik-Kenntnisse, wohingegen sowohl die Nicht-Expert:innen, als auch die Domänen-Expert:innen ungefähr gleich lang zum Bearbeiten der Aufgaben benötigt haben. Die 5 Minuten Differenz teilen sich auf alle Teilaufgaben ungefähr gleichmäßig auf, was damit erklärt werden kann, dass bei der Arbeit mit Robotern mit mehr Erfahrung Aufgaben schneller ausgeführt werden.

Weitere Evaluationsergebnisse

Zusätzlich zur Intuitivität und zur Programmierdauer wurden in der Studie noch weitere Aspekte des Programmiersystems untersucht. Zunächst sollte vor Beginn der Studie die Teilnehmer:innen in Frage 4 des Fragebogens F1 (siehe Anhang) zu den in der Benutzungsoberfläche verwendeten Symbolen angeben, was diese im Kontext der kinästhetischen Roboterprogrammierung bedeuten könnten. Dadurch wurde eruiert, welche Symbole bereits ohne Erklärung verständlich sind und bei welchen Symbolen eine zusätzliche Erläuterung oder eine Umgestaltung des Symbols notwendig ist. Die Ergebnisse dieser Frage sind in Tabelle 8.1 dargestellt. Die Farbcodierung hat in der Tabelle einen höheren Grünanteil, je mehr Teilnehmer:innen das Symbol richtig beschrieben haben.

Bei Betrachtung des Anteils richtiger Antworten hinsichtlich einzelner Symbole ist zu erkennen, dass die Symbole, welche unabhängig von der Roboterprogrammierung sind und in dieser Form häufig auch in anderen Anwendungen vorkommen, alle zu einem hohen Prozentsatz richtig benannt wurden (Tabelle 8.1 a) bis j)). Bei den Symbolen, welche Funktionen der kinästhetischen Roboterprogrammierung beschreiben, fallen insbesondere die Symbole für das Schleife-Inkrement m), die Roboter-Roboter-Synchronisation o) und die Mensch-Roboter-Synchronisation p) auf. Diese wurden am seltensten richtig benannt, was damit zu erklären ist, dass die Funktionen, welche diese Symbole beschreiben, außerhalb der kinästhetischen Roboterprogrammierung nicht vorhanden sind. Bei den Symbolen zur Verzweigungsfunktion n), zum Aktivieren des Handführens r) und zum Deaktivieren des Handführens s) zeigt sich, dass diese teilweise verstanden werden, obwohl die Funktionen an sich nicht in anderen Anwendungen vorkommen. Für die Symbole m), o) und p) ist es unter Umständen sinnvoll, aussagekräftigere Symbole zu entwerfen. Alternativ ist es hilfreich, Benutzer:innen vor der Erstbenutzung des Programmiersystems die Symbole, welche speziell für Funktionen der kinästhetischen Roboterprogrammierung entwickelt wurden, ausführlich zu erklären. Insgesamt lässt sich trotzdem feststellen, dass ein Großteil der verwendeten Symbole von den Teilnehmer:innen der Studie korrekt benannt wurde.

Tabelle 8.1: Auswertung der Verständlichkeit der in der Benutzungsoberfläche verwendeten Symbole. Angegeben wird, wie viele Teilnehmer:innen die Symbole richtig bzw. falsch benannt haben (Frage 4 aus Fragebogen F1 im Anhang). Die dritte Spalte wird abhängig von der Prozentzahl richtiger Antworten in einem Farbschema von grün (100%) über gelb (50%) nach rot (0%) eingefärbt.

	Symbol		**Richtig**		**Falsch**	
a)	Start/Home		40	95.2%	2	4.8%
b)	Neu		38	90.5%	4	9.5%
c)	Öffnen		32	76.2%	10	23.8%
d)	Speichern		42	100.0%	0	0.0%
e)	Benutzer:in hinzufügen		36	85.7%	6	14.3%
f)	Bearbeiten		38	90.5%	4	9.5%
g)	Kopieren		40	95.2%	2	4.8%
h)	Einfügen		33	78.6%	9	21.4%
i)	Ausschneiden		42	100.0%	0	0.0%
j)	Löschen		41	97.6%	1	2.4%
k)	Zusatzfunktionen		31	73.8%	11	26.2%
l)	Schleifenfunktion		41	97.6%	1	2.4%
m)	Schleifen-Inkrement	±1	5	11.9%	37	88.1%
n)	Verzweigungsfunktion		20	47.6%	22	52.4%
o)	Roboter-Roboter-Sync.		10	23.8%	32	76.2%
p)	Mensch-Roboter-Sync.		9	21.4%	33	78.6%
q)	Einstellungen		21	50.0%	21	50.0%
r)	Handführen aktivieren		21	50.0%	21	50.0%
s)	Handführen deaktivieren		21	50.0%	21	50.0%
t)	Greifer schließen		39	92.9%	3	7.1%
u)	Greifer öffnen		39	92.9%	3	7.1%
v)	Aufzeichnung starten		25	59.5%	17	40.5%

Zusätzlich zur Verständlichkeit der Symbole sollte herausgefunden werden, ob die Teilnehmer:innen der Studie eher die Graphdarstellung (Abbildung 8.13a) oder die Blockdarstellung (Abbildung 8.13b) für Roboterprogramme entlang von Zeitleisten bevorzugen. Des Weiteren wurde evaluiert, ob die Teilnehmer:innen bei sich wiederholenden Aufgaben eher Kopieren&Einfügen oder Zählschleifen präferieren. Diese Fragen wurden den Teilnehmerinnen in Fragebogen F3, Fragen 8 und 9 gestellt (siehe Anhang). Die Ergebnisse dieser Befragung sind in Tabelle 8.2 dargestellt.

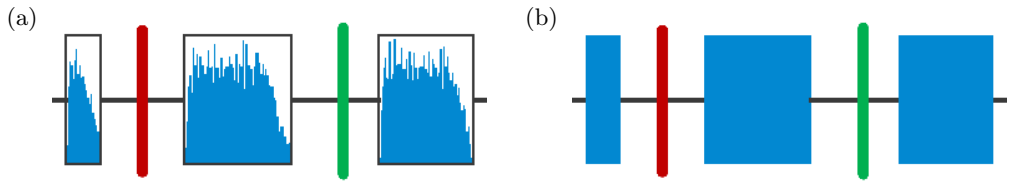


Abbildung 8.13: Unterschiedliche Darstellungsarten für Bewegungen entlang der Zeitleisten. Abbildung (a) zeigt die *Graphdarstellung*, bei der ein Geschwindigkeitsprofil des TCP angezeigt wird, Abbildung (b) die *Blockdarstellung*, bei der nur angezeigt wird, ob sich der TCP des Roboters bewegt oder nicht.

Tabelle 8.2: Auswertung der Abstimmung der Teilnehmer:innen, welche Darstellungsart und welche Art zum Programmieren wiederholender Aufgaben bevorzugt wird (Frage 8 und Frage 9 aus Fragebogen F3 im Anhang).

		Graph	**Block**
Frage 8	Welche der beiden Darstellungsarten (Graphdarstellung oder Blockdarstellung) für Roboterprogramme entlang der Zeitleiste bevorzugen Sie?	27 (64.3%)	15 (35.7%)

		Kopieren&Einfügen	**Zählschleife**
Frage 9	Welche Art zur Eingabe von mehrfach auszuführenden Bewegungen bevorzugen Sie? Kopieren und mehrfaches Einfügen oder eine Zählschleife?	3 (7.1%)	39 (92.9%)

Die Auswertung von Frage 8 zeigt, dass die Teilnehmer:innen eine leichte Tendenz zur Graphdarstellung besitzen. Jedoch wurde von 7 der 42 Teilnehmer:innen im Freitextkommentar angegeben, dass beide Darstellungsformen gewünscht sind. Bei Bedarf soll auch weiterhin zwischen beiden Darstellungen umgeschaltet werden können. Bei den Darstellungsformen gibt es demnach keinen klaren Favoriten, so dass die Wahlmöglichkeit weiter in der Benutzungsoberfläche bestehen bleibt und jede:r Benutzer:in nach den eigenen Vorlieben die Darstellungsform auswählen kann. Bei Frage 9 hingegen

zeigt sich, dass eine sehr große Mehrheit der Studienteilnehmer:innen die Zählschleife für die Umsetzung von sich wiederholenden Aufgaben bevorzugen. Hauptargument für die Befürwortung der Zählschleife war, dass die Roboterprogramme dadurch übersichtlicher und somit leichter verständlich bleiben.

Abschließend kann festgehalten werden, dass ein Teil der verwendeten Symbole bereits von den Teilnehmer:innen der Studie verstanden wurde. Bei spezifischen Symbolen, welche nur im Kontext der kinästhetischen Roboterprogrammierung vorkommen, ist eine vorherige Erläuterung nötig. Bei der Frage, ob die Graph- oder Blockdarstellung bevorzugt wird, ergibt sich kein einheitliches Bild, so dass beide Darstellungsformen weiterhin im Programmiersystem zur Verfügung gestellt werden und zwischen beiden umgeschaltet werden kann. Bei der Frage, ob eine Zählschleife oder Kopieren und Einfügen für sich wiederholende Programmteile bevorzugt wird, haben sich die Teilnehmer:innen mit sehr großer Mehrheit für die Verwendung der Zählschleife ausgesprochen.

8.3 Grenzen des Ansatzes

Der Ansatz, welcher für die kinästhetische Roboterprogrammierung entworfen und implementiert wurde, besitzt auch Grenzen, welche in diesem Abschnitt erläutert werden. Dabei wird zunächst in Abschnitt 8.3.1 auf die Limitierung des Programmiersystems auf Grobbewegungen eingegangen. Anschließend wird in Abschnitt 8.3.2 die Beschränkung auf Pick-and-Place Aufgaben erläutert, bevor zum Abschluss in Abschnitt 8.3.3 die Verbesserungsvorschläge der Teilnehmer:innen der Studie vorgestellt werden.

8.3.1 Grobbewegungen

Das Konzept des kinästhetischen Programmiersystems wurde im Hinblick auf Grobbewegungen entworfen. Unter Grobbewegungen sind Bewegungen zu verstehen, bei denen der Abstand zu Hindernissen deutlich größer ist als die geometrischen Ungenauigkeiten, wohingegen bei Feinbewegungen der Abstand zu Hindernissen in der Größenordnung der Ungenauigkeiten liegt. Unter Grobbewegungen fallen Aufgaben wie die Bewegung von einer Startposition zu einer Zielposition, das Abfahren einer Trajektorie mit ausreichend Abstand zu Hindernissen oder das Greifen und Ablegen von Objekten mit relativ großen Toleranzen. Bei Feinbewegungen hingegen ist in der Regel direkter Kontakt mit der Umgebung vorhanden. Beispiele dafür sind die Verfolgung einer Kontur mit einem Werkzeug das am Flansch montiert ist, oder das Peg-in-hole Problem, bei dem ein gegebenes Werkstück in ein zur Form des Werkstücks passendes Loch gesteckt werden soll.

Bei Feinbewegungen müssen demnach in der Regel Randbedingungen wie Position auf einer Werkstückoberfläche oder Kraft die auf die Umgebung aufgebracht werden soll eingehalten werden. Dies ist bei Grobbewegungen nicht der Fall. Dadurch wird für diese Version des kinästhetischen Roboter-Programmiersystems der Komplexitätsgrad gesenkt. Als weitere Ausbaustufe des Programmiersystems sollte eine Erweiterung auf Feinbewegungen in Betracht gezogen werden, da dadurch die Arten der programmierbaren Aufgaben um eine neue Aufgabenklasse erweitert wird. Insbesondere überwachte und nachgiebige Bewegungen als Teilmenge der Feinbewegungen können dabei von Interesse sein. Mit Hilfe von überwachten Bewegungen können Aufgaben realisiert werden, die so lange ausgeführt werden sollen, bis eine Randbedingung (z.B. Position oder Kraft) erfüllt ist. Bei nachgiebigen Bewegungen ist es möglich Randbedingungen, zum Beispiel in Form von Kraftaufbringung in eine bestimmte Richtung, zu definieren, welche während der Programmausführung aufrecht erhalten werden sollen.

8.3.2 Pick-and-Place Aufgaben

Zusätzlich zur Beschränkung auf Grobbewegungen wurde das kinästhetische Programmiersystem in der ersten Ausbaustufe nur für Pick-and-Place Aufgaben entworfen. Dies hat den Vorteil, dass die Konzepte, welche für die kinästhetische Roboterprogrammierung entworfen wurden, in der ersten Version nicht universell einsetzbar sein müssen, sondern nur mit den Pick-and-Place Aufgaben funktionieren müssen. Auch wurde die Studie so entworfen, dass die Konzepte des Programmiersystems im Hinblick auf Pick-and-Place Aufgaben evaluiert wurden.

Durch die Verwendung der Playback Programmierung als Grundlage für die kinästhetische Programmierung ist es möglich, dass mit Hilfe des Programmiersystems in der aktuellen Form auch Aufgaben zur Oberflächenbearbeitung, wie zum Beispiel Lackieren, Faserspritzen oder Schweißen, ausgeführt werden können. Diese Art Aufgaben wurden jedoch nicht explizit getestet. Um eine Aussage über die Tauglichkeit des Programmiersystems für Aufgaben aus dem Bereich Oberflächenbearbeitung treffen zu können, muss dieses in weiteren Evaluationen untersucht werden. Zusätzlich werden einige Konzepte auf die neue Aufgabenart angepasst werden müssen.

Für zukünftige Ausbaustufen des kinästhetischen Programmiersystems ist eine Erweiterung der Aufgabenarten sinnvoll. Dadurch wird die Bandbreite der programmierbaren Aufgaben erhöht und so das Programmiersystem universeller einsetzbar. Die Evaluation bezüglich Intuitivität und Geschwindigkeit in dieser Arbeit zeigt jedoch, dass mit dem verwendeten Ansatz Pick-and-Place Aufgaben zum Stapeln und Palettieren und auch zum Bestücken von Maschinen intuitiv programmierbar sind.

8.3.3 Verbesserungsvorschläge aus der Studie

Die Teilnehmer:innen der Studie hatten in einem Freitextfeld am Ende der Evaluation die Möglichkeit, Verbesserungsvorschläge und Kommentare zum evaluierten Programmiersystem zu geben. In diesem Abschnitt werden die häufigsten Vorschläge vorgestellt und kurz erläutert, wie diese umgesetzt werden könnten.

Eine Art Hilfefunktion wurde von zwei Teilnehmer:innen der Studie gewünscht. Dabei gab es unterschiedliche Vorschläge, wie diese Hilfefunktion umgesetzt werden soll. Zum einen wünschte sich ein:e Teilnehmer:in, dass mit Hilfe einer Art Tutorial beim ersten Start des Programmiersystems die einzelnen Funktionen erklärt werden sollen. Ein:e andere:r Teilnehmer:in wiederum wünschte sich eine Art Hilfe-Knopf, mit dem zu den Symbolen in den Zeitleisten und zu den einzelnen Bedienelementen kurze Erklärungen eingeblendet werden können. Beide Vorschläge haben zum Ziel, das Programmiersystem einsteigerfreundlicher zu machen und sollten in weiteren Ausbaustufen berücksichtigt werden.

Für die Editierfunktionen mit Hilfe von Kopieren, Einfügen, Ausschneiden und Löschen haben sich drei Teilnehmer:innen eine Rückgängig-Funktion gewünscht. Grund dieses Wunsches war es, dass ein gelöschter oder eingefügter Programmteil in der aktuellen Version des Programmiersystems nicht rückgängig gemacht werden kann. Dies sorgt dafür, dass Fehler beim Editieren nur durch weiteres Editieren, nicht aber durch zurücksetzen des Roboterprogramms ausgebessert werden können. Ein Konzept für eine Rückgängig-Funktion der Editierfunktionen ähnlich den Rückgängig-Funktionen in bekannten Texteditoren sollte für zukünftige Versionen des Programmiersystems entwickelt und implementiert werden.

Weiter Verbesserungsvorschläge gab es im Bereich der Gestaltung der Benutzungsoberfläche. So war für zwei Teilnehmer:innen die Hervorhebung der ausgewählten Zeitleiste nicht deutlich genug. Sie wünschten sich ein kräftigeres Einfärben der Zeitleiste oder eine Umrandung der ausgewählten Zeitleiste. Zusätzlich wünschte sich ein:e Teilnehmer:in, dass die Größe des Koordinatensystems im Simulationsfenster veränderbar sein sollte, um dieses bei Bedarf vergrößern oder verkleinern zu können.

Eine Beobachtung, die bei vier Teilnehmer:innen gemacht wurde ist, dass das Untermenü für die Verzweigungen beim ersten Anblick etwas verwirrend ist. Dies könnte dadurch verbessert werden, dass statt dem Untermenü ein mehrschrittiger Dialog verwendet wird, welche die Benutzer:innen durch das Erstellen einer Verzweigung führt. So könnten die Benutzer:innen des Programmiersystems Schritt für Schritt durch die Erstellung einer Verzweigung geführt werden.

8.4 Schlussfolgerung

Zusammenfassend kann festgehalten werden, dass mit diesem Kapitel die Konzepte der vorherigen Kapitel vereint in einem Gesamtdemonstrator hinsichtlich ihrer Intuitivität evaluiert wurden. Dazu wurde zunächst der Gesamtdemonstrator mit den unterstützten Robotertypen und der zugehörigen Benutzungsoberfläche vorgestellt. Anschließend wurde näher auf die durchgeführte Studie eingegangen, insbesondere auf das Studiendesign, die Aufgabe welche von den Teilnehmer:innen mit Hilfe des Programmiersystems gelöst werden musste sowie die Evaluationsergebnisse. Abschließend wurden die Grenzen des Ansatzes und Verbesserungsvorschläge für das Programmiersystem, welche aus der Studie entstanden sind, dargelegt.

Forschungsfrage F6 lässt sich damit beantworten, dass die Evaluationsergebnisse hinsichtlich der Intuitivität mit ihren drei Bestandteilen Effektivität, mentale Effizienz und Zufriedenheit eindeutig gezeigt haben, dass das Programmiersystem sowohl von Benutzer:innen mit, als auch ohne Robotik-Kenntnissen als intuitiv bewertet wurde. Die Antwort auf Forschungsfrage F6.1 lautet konkret, dass von den insgesamt 42 Teilnehmer:innen nur 3 das Programmiersystem tatsächlich komplexer empfunden haben, als erwartet. Alle anderen Teilnehmer:innen fanden das Programmiersystem leichter zu bedienen als erwartet. Zu Forschungsfrage F6.2 kann gesagt werden, dass Unterschiede in der Bewertung des Systems durch die einzelnen Gruppen kaum vorhanden sind. Über alle Gruppen hinweg zeichnet sich ein ähnliches Bewertungsmuster ab. Die Verständlichkeit der verwendeten Symbole aus Forschungsfrage F6.3 kann damit beantwortet werden, dass die allgemein bekannte Symbole (z.B. Speichern, Kopieren, Löschen, ...) ohne weitere Erklärung verständlich sind, jedoch die für die kinästhetische Roboterprogrammierung eingeführten Symbole je nach Komplexität der beschriebenen Funktion neuen Benutzer:innen erklärt werden müssen. Bei der Auswertung von Forschungsfrage F6.4 zeigt sich eine leichte Präferenz der Benutzer:innen des Programmiersystems zur Graphdarstellung, jedoch gibt es auch das Bedürfnis, dass die Benutzer:innen die Wahl zwischen beiden Darstellungsformen haben sollen. Die letzte Forschungsfrage F6.5 kann damit beantwortet werden, dass nur 3 der 42 Teilnehmer:innen der Studie Kopieren und Einfügen anstelle von Schleifen bei repetitiven Aufgaben bevorzugen würden.

Nachdem in den letzten Kapiteln die einzelnen Aspekte des Programmiersystems näher erläutert wurden und in diesem Kapitel die Konzepte in Form des Gesamtdemonstrators evaluiert wurden, gibt das folgende Kapitel ein Fazit zu dieser Arbeit. Dazu wird der Inhalt dieser Arbeit zusammengefasst, eine Schlussfolgerung gezogen und ein Ausblick auf mögliche folgende Arbeiten gegeben.

KAPITEL 9

Fazit

Inhalt

Dieses Kapitel zieht ein Fazit zu dieser Arbeit. Dazu wird in Abschnitt 9.1 der Inhalt dieser Arbeit zusammengefasst und die Forschungsfragen aus Kapitel 1.2 diskutiert. Abschließend wird in Abschnitt 9.2 ein Ausblick auf mögliche zukünftige Arbeiten für weitere Schritte Richtung der Vision eines intuitiven, universellen Roboter-Programmiersystems gegeben.

9.1 Zusammenfassung

Robotersysteme sind für kleine und mittlere Unternehmen (KMU) zur Zeit noch nicht sehr rentabel, da diese aufgrund meist kleiner Losgrößen oft umprogrammiert werden müssen. Das Umprogrammieren muss bei konventionellen Robotern in der Regel von Systemintegratoren oder Expert:innen auf dem Gebiet der Roboterprogrammierung erledigt werden. Um Robotersysteme für KMU attraktiver zu gestalten ist ein universelles Programmiersystem notwendig, welches intuitiv bedienbar ist. Dieses sollte so konstruiert sein, dass es von Nicht-Expert:innen ohne textuelle Programmierkenntnisse verwendet werden kann, um ein potentielles Mehrrobotersystem programmieren zu können.

Aus dem Stand der Forschung hat sich gezeigt, dass vor allem im Bereich der Roboterprogrammierung durch das Aufkommen der kollaborativen Roboter einfache Programmiersysteme entstanden sind. Insbesondere die kinästhetische Roboterprogrammierung zeigt sich dabei für Nicht-Expert:innen als intuitiv zu bedienen.

In dieser Arbeit wurde ein neues Konzept für ein Roboterprogrammiersystem auf Basis der kinästhetischen Roboterprogrammierung präsentiert. Das *Darstellen* der Roboterprogramme erfolgt dabei entlang von Zeitleisten mit Hilfe angepasster Ablaufdiagramme. Das Programmiersystem bietet den Benutzer:innen die Möglichkeit, bestehende Programme in der Zeitleiste zu *editieren*, mit Hilfe von Sensoren zu *kontrollieren*, und mit Synchronisationspunkten und -intervallen zu *synchronisieren*. Abschließend wurde in einer Studie die Intuitivität des Programmiersystems evaluiert.

Im Folgenden werden die Forschungsfragen aus Kapitel 1.2 noch einmal betrachtet und diskutiert. In der ersten Frage wurde die Darstellung von geschachtelten Roboterprogrammen entlang von Zeitleisten thematisiert:

F1 Wie lassen sich geschachtelte Roboterprogramme strukturiert und skalierbar graphisch entlang einer Zeitleiste darstellen?

Diese Frage wurde in Kapitel 4 behandelt. Um eine strukturierte und skalierbare Darstellung von geschachtelten, kinästhetischen Roboterprogrammen zu erreichen wurde zunächst eine kontextfreie Grammatik in Form einer BNF entwickelt, welche die kinästhetisch demonstrierten Roboterprogramme dieser Arbeit abbilden kann. Für die einzelnen Elemente dieser BNF wurde anschließend eine graphische Darstellung entlang von Zeitleisten basierend auf Ablaufdiagrammen konzipiert. Diese wurde geschichtet und skalierbar entworfen, damit das Programmiersystem potentiell beliebig viele Roboter und beliebig tief geschachtelte Roboterprogramme darstellen kann. Die Evaluation der graphischen Darstellung entlang von Zeitleisten hat gezeigt, dass kinästhetisch demonstrierte Roboterprogramme, welche auf diese Art und Weise dargestellt werden, für Benutzer:innen leicht und schnell verständlich sind.

Die zweite Frage behandelte das Thema der Editierbarkeit von graphischen Roboterprogrammen:

F2 Auf welche Art und Weise lassen sich kinästhetisch demonstrierte Roboterprogramme graphisch entlang von Zeitleisten editieren?

Bei dieser Frage stand die graphische Editierbarkeit der kinästhetisch demonstrierten Roboterprogramme im Mittelpunkt. Diese wurde in Kapitel 5 behandelt. Dabei wurden zunächst die Kopieren&Einfügen-Funktionen für die kinästhetische Roboterprogrammierung vorgestellt. Zusätzlich wurde im Rahmen dieser Arbeit ein Konzept zum Drag-and-Drop von Kontrollfluss-Elementen innerhalb der Zeitleisten entwickelt. Dadurch ist es beispielsweise möglich, Verzweigungen oder Schleifen innerhalb der Roboterprogramme auch nachträglich zu verschieben. Zum Lösen der durch das Editieren entstehenden Probleme in Form von Sprungstellen in den Trajektorien der Roboter wurden zwei Konzepte entwickelt. Zum einen ist es möglich, die Zwischenbahn zum Verbinden der Sprungstelle automatisch als PTP-Bewegung generieren zu lassen, zum anderen wurde ein neuartiges Konzept entwickelt, welches die Benutzer:innen mit mul-

timodalem Feedback dabei unterstützt, den Roboter manuell vom Start der Sprungstelle zu deren Ende zu Führen. Die Evaluation des manuellen Führens von einer Start- zu einer Zielposition hat gezeigt, dass dies für die Benutzer:innen schnell und intuitiv möglich ist.

In der dritten Forschungsfrage wurde überprüft, in wie weit sich Konzepte für Schleifen und Verzweigungen aus der imperativen Programmierung auf die kinästhetische Roboterprogrammierung übertragen lassen:

F3 Wie lassen sich Konzepte der imperativen Programmierung wie Schleifen und Verzweigungen auf das kinästhetische Programmierkonzept übertragen?

Diese Frage wurde in Kapitel 6 behandelt. Mit Hilfe von Sensoren, deren Sensorwerte auf Ähnlichkeit vergleichbar sind, wurden sensorbasierte Kontrollstrukturen in Anlehnung an Kontrollstrukturen aus prozeduralen Programmiersprachen konzipiert. Dadurch ist es für das Robotersystem möglich, während der Programmausführung auf Umwelteinflüsse zu reagieren. Es wurden Konzepte für zwei Arten von sensorbasierten Kontrollstrukturen erarbeitet, den Schleifen und den Verzweigungen. Bei den Schleifen wird nach bzw. vor jedem Schleifendurchlauf der Soll-Sensorwert mit dem aktuell aufgenommenen Ist-Sensorwert verglichen und anhand der Ähnlichkeit der beiden Sensorwerte die Schleife entweder erneut ausgeführt oder übersprungen. Bei den Verzweigungen wird der aktuelle Ist-Sensorwert mit den Soll-Sensorwerten der einzelnen Zweige verglichen wird und derjenige Zweig ausgeführt, der den ähnlichsten Sensorwert hat. Als Beispielsensorwerte wurden 2D-RGB-Kamerabilder und Greiferöffnungswinkel in dieser Arbeit vorgestellt. Die Evaluation des Konzeptes der sensorbasierten Kontrollstrukturen in Form einer Studie hat gezeigt, dass diese in einer prototypischen Variante des Programmiersystems durch die Benutzer:innen intuitiv verwendbar sind.

Aufbauend auf die dritte Forschungsfrage wurde mit der vierten Frage das Thema Schleifeninkremente zum Programmieren von Stapel- und Palettierszenarien diskutiert:

F4 Wie lässt sich das Konzept von Schleifeninkrementen auf die kinästhetische Roboterprogrammierung übertragen?

Dadurch, dass bei der kinästhetischen Programmierung die Roboter die Bewegung exakt gleich ausführen wie sie programmiert wurde, wurde in Kapitel 6 das Schleifen-Inkrement als neuartiges Konzept für die Adaption der Roboterbewegungen nach jedem Schleifendurchlauf entwickelt. Mit Hilfe des Schleifen-Inkrements müssen bei Stapel- und Palettieraufgaben nicht alle Teilbewegungen explizit demonstriert werden, sondern nur die erste der Pick-and-Place Bewegungen. Anschließend werden die Inkremente entweder durch Führen des Roboters oder durch Eingeben der kartesischen Transformation definiert. Das Programmiersystem berechnet mit Hilfe der Inkremente nach jedem Schleifendurchlauf die neue Trajektorie und führt diese aus. Dadurch wer-

den vom Programmierkonzept Stapel- und Palettieraufgaben unterstützt. Unter den Verfahren, welche zur Anpassung einer Trajektorie verglichen wurden, hat sich die Intervall-Methode in Kombination mit der Kosinus-Funktion als am besten geeignet für die Aufgaben dieser Arbeit herausgestellt.

Bisher wurde außer Acht gelassen, dass potentiell beliebig viele Roboter mit dem Programmiersystem gesteuert werden können. Dabei kann es beim Editieren aus F2 und bei den Kontrollstrukturen und dem Inkrement aus F3 und F4 dazu kommen, dass die Programme der beteiligten Roboter sich zueinander zeitlich verschieben und Kollisionen entstehen. Um dies zu vermeiden, wurde in der fünften Fragestellung das Thema zeitliche Synchronisation zwischen Roboter und Roboter und zwischen Mensch und Roboter behandelt:

F5 In wie weit lässt sich das Programmierkonzept auf die Zusammenarbeit mehrerer Roboter und auf die Zusammenarbeit zwischen Mensch und Roboter erweitern?

Diese Frage wurde in Kapitel 7 bearbeitet. Zunächst wurde ein Konzept zur zeitlichen Synchronisierung mehrerer Roboter entwickelt. Dabei wurden sowohl Synchronisationspunkte, als auch Synchronisationsintervalle definiert. Die Aufgabe der Punkte ist es, Zeitpunkte in den kinästhetisch demonstrierten Roboterprogrammen festzulegen, an denen mehrere Roboter aufeinander warten. Dies kann zum Beispiel dafür verwendet werden, einen gemeinsamen Arbeitsraum zweier Roboter gegeneinander abzusichern. Die Intervalle hingegen können verwendet werden, wenn mehrere Roboter eine Aufgabe gemeinsam ausführen sollen, wie zum Beispiel ein Objekt zu heben. Zusätzlich zur Synchronisation mehrerer Roboter wurde ein Konzept zur Synchronisation zwischen Mensch und Roboter als Beispiel für die Synchronisation mit einem externen Aktor entwickelt. Auch hier gibt es Intervalle und Punkte, wobei der Roboter den externen Aktor über einen Benachrichtigungskanal informiert, sobald ein Punkt oder Intervall erreicht wurde, damit dieser die vorher definierte Aufgabe ausführt. Erst nach quittieren der Aufgabe setzt der Roboter die Ausführung seines Programms fort. Dies ist von Vorteil, wenn Teilaufgaben zum Beispiel aufgrund falschen Werkzeugs nicht vom Roboter gelöst werden können, sondern externe Hilfe benötigt wird.

Die letzte Fragestellung, welche betrachtet wurde, untersuchte die Intuitivität als Kombination aus Effektivität, mentaler Effizienz und Zufriedenheit des entwickelten Programmierkonzeptes in Form einer Studie mit dem Prototypen des Programmiersystems:

F6 Inwiefern lässt sich das entstandene kinästhetische Programmierkonzept und dessen prototypische Implementierung intutiv bedienen? Wie spiegelt sich das in Effektivität, mentaler Effizienz und Zufriedenheit der Programmierung wider?

Die einzelnen Aspekte aus den Kapiteln 4, 5, 6 und 7 wurden in einem Gesamtdemonstrator vereint und in Kapitel 8 hinsichtlich Intuitivität evaluiert. Dabei wurden aus der Literatur anerkannte Verfahren und Fragebögen verwendet, welche die drei Teilaspekte der Intuitivität, nämlich Effektivität, mentale Effizienz und Zufriedenheit messen. In einer Studie mit 42 Teilnehmer:innen sollte von diesen eine Übergabe- und Sortieraufgabe programmiert werden. Über alle Teilnehmer:innen und Teilaufgaben ergab sich für die Effektivität ein Wert von 85.0%, was als sehr gut zu bewerten ist. Auch die mentale Effizienz ist mit einem Wert von 43.2 auf der SEA-Skala als äußerst gut bewertet worden. Dieser Wert kann wörtlich mit „etwas anstrengend“ übersetzt werden. Bei der Zufriedenheit wurde mit einem Wert von 4.0 auf der QUESI-Skala ebenfalls ein exzellenter Wert erreicht. Das Gesamtergebnis für die Konzepte dieser Arbeit und das graphische Programmiersystem kann demnach in allen drei Kategorien als intuitiv bewertet werden.

Zusammenfassend kann festgehalten werden, dass mit dieser Arbeit ein Schritt in Richtung eines universell einsetzbaren, intuitiven Roboterprogrammiersystems gemacht wurde. Dabei wurden anhand der vier Aspekte Darstellen, Editieren, Kontrollieren und Synchronisieren Konzepte für eine editierbare, sensorbasierte kinästhetische Programmierung von Mehrrobotersystemen für Pick-and-Place Aufgaben definiert und implementiert. Die Evaluation hat gezeigt, dass dieses Programmiersystem von Gruppen mit unterschiedlichen Vorkenntnissen als intuitiv bewertet wird und somit die Aufgabenstellung dieser Arbeit erfüllt wurde.

9.2 Ausblick

Die intuitive, kinästhetische Roboterprogrammierung kann in verschiedene Richtungen erweitert werden, um das Fernziel eines universellen, intuitiven Roboterprogrammiersystems zu erreichen. Vier Stoßrichtungen, welche sich aus dieser Arbeit ergeben haben, werden im Folgenden kurz beschrieben. Dabei handelt es sich um die Erweiterung der Anwendungsgebiete, die Unterstützung weiterer Bewegungsarten, das Hinzufügen einer automatischen Bahnverbesserung und die Möglichkeit des Wiederverwendens parametrierbarer Programmteile.

Eine mögliche Entwicklungsrichtung ist das Untersuchen weiterer Anwendungsgebiete zusätzlich zu Pick-and-Place Aufgaben. Die kinästhetische Programmierung und insbesondere die Playback Programmierung eignet sich durch ihre Eigenschaft, dass Bewegungen exakt so ausgeführt werden wie sie demonstriert wurden, für Anwendungen, bei denen die Trajektorie des Werkzeuges relevant ist. Dies ist zum Beispiel bei Schweißen, Lackieren oder Sprühen der Fall. Dabei wäre es denkbar Expert:innen-Wissen in die Programme mit einfließen zu lassen, indem Domänen-Expert:innen auf dem jeweiligen Fachgebiet das Robotersystem kinästhetisch programmieren.

Eine zusätzliche Erweiterung des Programmierkonzeptes wäre denkbar, indem weitere Bewegungsarten unterstützt werden. Insbesondere durch hybride Bewegungen lässt sich der Anwendungsbereich der kinästhetischen Programmierung auf Oberflächenbehandlung mit Kontakt zwischen Roboter und Werkstück erweitern. Unter hybride Bewegungen fallen dabei sowohl geregelte, als auch überwachte Bewegungen. Bei ersteren wird während der Ausführung des Programms eine Randbedingung, beispielsweise eine Kraft in eine bestimmte Raumrichtung, aufrecht erhalten. Bei überwachten Bewegungen hingegen wird eine Bewegung so lange ausgeführt, bis eine Randbedingung erfüllt ist, beispielsweise das drücken eines Schalters mit einer vordefinierten Kraft.

Des Weiteren ist es denkbar, im Bereich des Editierens automatische Bahnverbesserungen zu untersuchen. Diese könnten auf der Trajektorie des Roboterprogramms arbeiten und die Bahn zum Beispiel automatisch glätten. Auch wäre eine automatische Optimierung der Trajektorie hinsichtlich verschiedener Kriterien, wie zum Beispiel eine möglichst gleichmäßige Geschwindigkeit oder einer Minimierung der Stillstandszeiten des Roboters, denkbar.

Eine abschließende Stoßrichtung für die kinästhetische Roboterprogrammierung ist das Hinzufügen des Aspekts des Wiederverwendens zu den vier in dieser Arbeit konzipierten Aspekten. Dies könnte in Form einer Programmbibliothek realisiert werden, welche häufig verwendete Teilprogramme zum Einfügen in ein bestehendes Roboterprogramm bereitstellt. Diese Teilprogramme müssen parametrierbar sein, damit sie in die Trajektorie des bestehende Roboterprogramms nahtlos eingefügt werden können. Dies hätte den Vorteil, dass dadurch Roboterprogramme nicht von Grund auf neu erstellt werden müssen, sondern das bereits vorhandene Programmteile zum Erstellen des Programms wiederverwendet werden können.

Es gibt demnach noch diverse Problemstellungen, welche bis zum Erreichen der anfangs dargelegten Vision gelöst werden sollten. Gleichwohl wurde mit dieser Arbeit bereits ein wesentlicher Schritt in Richtung eines universell einsetzbaren, intuitiven Roboterprogrammiersystems für Nicht-Expert:innen gemacht.

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[ABB2010] ABB. *Technical reference manual - RAPID Instructions, Functions and Data types*, 2010.

[Adobe] Adobe. *Adobe Premiere Pro.* `https://www.adobe.com/de/products/premiere.html`. Abgerufen: 20.08.2021.

[Ajaykumar2020] G. Ajaykumar & C.-M. Huang. *User Needs and Design Opportunities in End-User Robot Programming.* In Companion of the 2020 ACM/IEEE International Conference on Human-Robot Interaction, Seiten 93–95, 2020.

[Akan2010] B. Akan, B. Cürüklü, G. Spampinato & L. Asplund. *Towards Robust Human Robot Collaboration in Industrial Environments.* In 2010 5th ACM/IEEE International Conference on Human-Robot Interaction (HRI), Seiten 71–72, 2010.

[Alexandrova2015] S. Alexandrova, Z. Tatlock & M. Cakmak. *RoboFlow: A Flow-based Visual Programming Language for Mobile Manipulation Tasks.* In 2015 IEEE International Conference on Robotics and Automation (ICRA), Seiten 5537–5544, 2015.

[AndroidDevelopers] Google. *Android Developers Guide - Get started with large screens.* `https://developer.android.com/guide/topics/ui/responsive-layout-overview`. Abgerufen: 20.08.2021.

[Ang1999] M.H. Ang Jr., W. Lin & S.-Y. Lim. *A walk-through programmed robot for welding in shipyards.* Industrial Robot: An International Journal, Band 26, Nr. 5, Seiten 377–388, 1999.

[Ang2000] M.H. Ang Jr., W. Lin & S.-Y. Lim. *An Industrial Application of Control of Dynamic Behavior of Robots - A Walk-Through Programmed Welding Robot.* In 2000 IEEE International Conference on Robotics and Automation, Seiten 2352–2357, 2000.

[Antonelli2013] D. Antonelli, S. Astanin, M. Galetto & L. Mastrogiacomo. *Training by demonstration for welding robots by optical trajectory tracking.* In 8th CIRP Conference on Intelligent Computation in Manufacturing Engineering, Seiten 145–150, 2013.

[Arai1997] T. Arai, T. Itoko & H. Yago. *A graphical robot language developed in Japan.* Robotica, Band 15, Seiten 99–103, 1997.

[Artiminds] *Artiminds.* `https://www.artiminds.com/`. Abgerufen: 20.08.2021.

[ATI] ATI. *F/T Sensor Gamma SI-32-2.5*. `https://www.ati-ia.com/products/ft/ft_models.aspx?id=Gamma`. Abgerufen: 20.08.2021.

[Bascetta2013] L. Bascetta, G. Ferretti, G. Magnani & P. Rocco. *Walk-through programming for robotic manipulators based on admittance control*. Robotica, Band 31, Nr. 7, Seiten 1143–1153, 2013.

[Berg2019] J Berg, A. Lottermoser, C. Richter & G. Reinhart. *Human-Robot-Interaction for mobile industrial robot teams*. Procedia CIRP, Band 79, Seiten 614–619, 2019.

[Berg2020] J. Berg & S. Lu. *Review of Interfaces for Industrial Human-Robot Interaction*. Current Robotics Reports, Band 1, Seiten 27–34, 2020.

[Beschi2019] S. Beschi, D. Fogli & F. Tampalini. *CAPIRCI: A Multi-modal System for Collaborative Robot Programming*. In International Symposium on End User Development, Seiten 51–66, 2019.

[Bischoff2002] R. Bischoff, A. Kazi & M. Seyfarth. *The MORPHA style guide for icon-based programming*. In 11th IEEE International Workshop on Robot and Human Interactive Communication, 2002.

[Blank2003] D. Blank, D. Kumar, L. Meeden & H. Yanco. *Pyro: A Python-based Versatile Programming Environment for Teaching Robotics*. ACM Journal of Educational Resources in Computing, Band 3, Nr. 4, Seiten 1–15, 2003.

[Blender] Blender Documentation Team. *Blender 2.79 Manual*. `https://docs.blender.org/manual/en/2.79/editors/3dview/object/editing/transform/control/proportional_edit.html`. Abgerufen: 20.08.2021.

[Bravo2017] F.A. Bravo, A.M. Gonzalez & E. Gonzalez. *A Review of Intuitive Robot Programming Environments for Educational Purposes*. In 2017 IEEE 3rd Colombian Conference on Automatic Control, Seiten 1–6, 2017.

[Calinon2007] S. Calinon & A. Billard. *Active Teaching in Robot Programming by Demonstration*. In 16th IEEE International Conference on Robot and Human Interactive Communication, Seiten 702–707, 2007.

[Casares2002] J. Casares, A.C. Long, B.A. Myers, R. Bhatnagar, S.M. Stevens, L. Dabbish, D. Yocum & A. Corbett. *Simplifying video editing using metadata*. In DIS'02: 4h conference on Designing interactive systems: processes, practices, methods, and techniques, Seiten 157–166, 2002.

[Cassandras2009] C.G. Cassandras & S. Lafortune. *Introduction to Discrete Event Systems*. Springer Science & Business, 20019.

[Cederborg2010] T. Cederborg, M. Li, A. Baranes & P.-Y. Oudeyer. *Incremental Local Online Gaussian Mixture Regression for Imitation Learning of Multiple Tasks*. In 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Seiten 267–274, 2010.

[Chandrasekaran2015] B. Chandrasekaran & J.M. Conrad. *Human-robot collaboration: A survey.* In Proceedings of the IEEE SoutheastCon 2015, Seiten 1–8, 2015.

[Charntaweekhun2006] K. Charntaweekhun & S. Wangsiripitak. *Visual Programming using Flowchart.* In 2006 International Symposium on Communications and Information Technologies, Seiten 1062–1065, 2006.

[Cherubini2016] A. Cherubini, R. Passama, A. Crosnier, A. Lasnier & P. Fraisse. *Collaborative manufacturing with physical human–robot interaction.* Robotics and Computer-Integrated Manufacturing, Band 40, Seiten 1–13, 2016.

[Chung2020] M.J.-Y. Chung, M. Nakura, S.H. Neti, A. Lu, E. Hummel & M. Cakmak. *ConCodeIt! A Comparison of Concurrency Interfaces in Block-Based Visual Robot Programming.* In 2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN), Seiten 245–252, 2020.

[Coronado2020] E. Coronado, F. Mastrogiovanni, B. Indurkhya & G. Venture. *Visual Programming Environments for End-User Development of intelligent and social robots, a systematic review.* Journal of Computer Languages, Band 58, Seiten 1–20, 2020.

[Cox1998] P.T. Cox & T.J. Smedley. *Visual programming for robot control.* In 1998 IEEE Symposium on Visual Languages, Seiten 217–224, 1998.

[Craig2005] J.J. Craig. Introduction to robotics: Mechanics and control. Pearson Education International, 2005.

[Cross2013] J. Cross, C. Bartley, E. Hamner & I. Nourbakhsh. *A Visual Robot-Programming Environment for Multidisciplinary Education.* In 2013 IEEE International Conference on Robotics and Automation, Seiten 445–452, 2013.

[Datta2012] C. Datta, C. Jayawardena, I.H. Kue & B.A. MacDonald. *RoboStudio: A Visual Programming Environment for Rapid Authoring and Customization of Complex Services on a Personal Service Robot.* In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Seiten 2352–2355, 2012.

[Dennet2006] D.C. Dennet. *The Frame Problem of AI.* In J.L. Bermudez, Herausgeber, Philosophy of Psychology - Contemporary Reading, Seiten 433–454. Routledge, 1996.

[DeSchutter2007] J. De Schutter, T. De Laet, J. Rutgeerts, W. Decre, R. Smits, E. Aertbelien, K. Claes & H. Bruyninckx. *Constraint-based Task Specification and Estimation for Sensor-Based Robot Systems in the Presence of Geometric Uncertainty.* The International Journal of Robotics Research, Band 26, Nr. 5, Seiten 433–455, 2007.

[DESTATIS2018] *Shares of small and medium-sized enterprises in selected variables, 2018.* `https://www.destatis.de/EN/Themes/Economic-Sectors-Enterprises/Enterprises/Small-Sized-Enterprises-Medium-Sized-Enterprises/Tables/total-cik.html`. Abgerufen: 20.08.2021.

[Dietz2012] T. Dietz, U. Schneider, M. Barho, S. Oberer-Treitz, M. Drust, R. Hollmann & M. Hägele. *Programming System for Efficient Use of Industrial Robots for Deburring in SME Environments.* In Robotik 2012, Seiten 428–433, 2012.

[DIN66312] *DIN 66312 - Programmiersprache Industrial Robotic Language (IRL).* Deutsches Institut für Normung (DIN) e.V., 1996.

[DLR] Deutsches Zentrum für Luft-und Raumfahrt Institut für Robotik und Mechatronik. *Geschichte des LBR.* `https://www.dlr.de/rm/desktopdefault.aspx/tabid-12464/21732_read-44586/`. Abgerufen: 20.08.2021.

[DragAndBot] *drag&bot.* `https://www.dragandbot.com/de/`. Abgerufen: 20.08.2021.

[Eilers1986] K. Eilers, F. Nachreiner & K. Hänecke. *Entwicklung und Überprüfung einer Skala zur Erfassung subjektiv erlebter Anstrengung.* Zeitschrift für Arbeitswissenschaft, Band 40, Nr. 4, Seiten 215–224, 1986.

[ElZaatari2019] S. El Zaatari, M. Marei, W. Li & Z. Usman. *Cobot programming for collaborative industrial tasks: An overview.* Robotics and Autonomous Systems, Band 116, Seiten 162 – 180, 2019.

[Ferraguti2017] F. Ferraguti, C.T. Landi, C. Secchi, C. Fantuzzi, M. Nolli & M. Pesamosca. *Walk-through programming for industrial applications.* In 27th International Conference on Flexible Automation and Intelligent Manufacturing, Seiten 31–38, 2017.

[Ficuciello2015] F. Ficuciello, L. Villani & B. Siciliano. *Variable Impedance Control of Redundant Manipulators for Intuitive Human-Robot Physical Interaction.* IEEE Transactions on Robotics, Band 31, Nr. 4, Seiten 850–863, 2015.

[Fowler2003] M. Fowler. Uml konzentriert - eine kompakte einführung in die standardobjektmodellierungssprache. Addison-Wesley, 2003.

[Frank2016] J.A. Frank, M. Moorhead & V. Kapila. *Realizing mixed-reality environments with tablets for intuitive human-robot collaboration for object manipulation tasks.* In 2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), Seiten 302–307, 2016.

[Franka] *Franka Emika.* `https://www.franka.de/de/robot-system`. Abgerufen: 20.08.2021.

[Fujii2016] M. Fujii, H. Murakami & M. Sonehara. *Study on Application of a Human-Robot Collaborative System Using Hand-Guiding in a Production Line.* IHI Engineering Review, Band 49, Nr. 1, Seiten 24–29, 2016.

[Gadre2019] S.Y. Gadre, E. Rosen, G. Chien, E. Phillips, S. Tellex & G. Konidaris. *End-User Robot Programming Using Mixed Reality.* In 2019 International Conference on Robotics and Automation, Seiten 2707–2713, 2019.

[Gan2013] Y. Gan, X. Dai & D. Li. *Off-Line Programming Techniques for Multirobot Cooperation System.* International Journal of Advanced Robotic Systems, Band 10, Nr. 7, Seiten 1–17, 2013.

[Grüninger2009] R. Grüninger, E. Kus & R. Huppi. *Market study on adaptive robots for flexible manufacturing systems.* In 2009 IEEE International Conference on Mechatronics, Seiten 1–7, 2009.

[Groth2014a] C. Groth & D. Henrich. *Single-Shot Learning and Scheduled Execution of Behaviors for a Robotic Manipulator.* In ISR/Robotik 2014 - 41st International Symposium on Robotics, 2014.

[Groth2014b] C. Groth & D. Henrich. *One-shot robot programming by demonstration using an online oriented particles simulation.* In 2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014), 2014.

[Guhl2017] J. Guhl, S. Tung & J. Krüger. *Concept and Architecture for Programming industrial Robots using Augmented Reality with Mobile Devices like Microsoft HoloLens.* In 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Seiten 1–4, 2017.

[Guhl2019] J. Guhl, S. Nikoleizig, O. Heimann, J. Hügle & J. Krüger. *Combining the Advantages of On- and Offline Industrial Robot Programming.* In 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Seiten 1567–1570, 2019.

[Hanses2016] M. Hanses, R. Behrens & N. Elkmann. *Hand-guiding robots along predefined geometric paths under hard joint constraints.* In 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation, Seiten 1–5, 2016.

[Heimann2017] O. Heimann, J. Hügle & J. Krüger. *Gesture based robot programming using process knowledge — An example for welding applications.* In 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Seiten 1–4, 2017.

[Heimann2020] O. Heimann & J. Guhl. *Industrial Robot Programming Methods: A Scoping Review.* In 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Seiten 696–703, 2020.

[Hokayem2006] P. Hokayem & M. Spong. *Bilateral teleoperation: An historical survey.* Automatica, Band 42, Nr. 12, Seiten 2035–2057, 2007.

[Huang2017] J. Huang & M. Cakmak. *Code3: A System for End-to-End Programming of Mobile Manipulator Robots for Novices and Experts.* In Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction, Seite 453–462. Association for Computing Machinery, 2017.

[Huynh-Thu2008] Q. Huynh-Thu & M. Ghanbari. *Scope of validity of PSNR in image/video quality assessment.* Electronics Letters, Band 44, Seiten 800–801, 2008.

[iDS-Imaging] iDS Imaging. *UI-1220SE.* `https://de.ids-imaging.com/store/ui-1220se.html`. Abgerufen: 20.08.2021.

[ISO10218] *ISO 10218 - Robots and robotic devices — Safety requirements for industrial robots.* 2011.

[ISO5807] *ISO 5807:1985 - Information processing — Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts.* 2019.

[ISO9241-11] *Teil 11: Gebrauchstauglichkeit - Begriffe und Konzepte.* In DIN EN ISO9241 - Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten, 2018.

[ISO9241-110] *Teil 110: Grundsätze der Dialoggestaltung.* In DIN EN ISO9241 - Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten, 2006.

[ISO9241-13] *Teil 13: Benutzerführung.* In DIN EN ISO9241 - Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten, 2000.

[ISO9241-210] *Teil 210: Prozess zur Gestaltung gebrauchstauglicher interaktiver Systeme.* In DIN EN ISO9241 - Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten, 2010.

[Jost2014] B. Jost, M. Ketterl, R. Budde & T. Leimbach. *Graphical Programming Environments for Educational Robots: Open Roberta - Yet another One?* In 2014 IEEE International Symposium on Multimedia, Seiten 381–386, 2014.

[Kildal2018] J. Kildal, A. Tellaeche, I. Fernández & I. Maurtua. *Potential users' key concerns and expectations for the adoption of cobots.* Procedia CIRP, Band 72, Seiten 21 – 26, 2018.

[Kim2004] J.Y. Kim. *CAD-Based Automated Robot Programming in Adhesive Spray Systems for Shoe Outsoles and Uppers.* Journal of Robotic Systems, Band 21, Nr. 11, Seiten 625–634, 2004.

[Kim2007] S.H. Kim & J.W. Jeon. *Programming LEGO Mindstorms NXT with visual programming.* In International Conference on Control, Automation and Systems 2007, Seiten 2468–2472, 2007.

[Kohn2018] S. Kohn, A. Blank, D. Puljiz, L. Zenkel, O. Bieber, B. Hein & J. Franke. *Towards a Real-Time Environment Reconstruction for VR-Based Teleoperation Through Model Segmentation.* In Proceedings of 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Seite 6, 2018.

[Kraft2017] M. Kraft & M. Rickert. *How to Teach Your Robot in 5 Minutes: Applying UX Paradigms to Human-Robot-Interaction.* In 2017 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), Seiten 942–949, 2017.

[Krot2019] K. Krot & V. Kutia. *Intuitive Methods of Industrial Robot Programming in Advanced Manufacturing Systems.* In ISPEM 2018: Intelligent Systems in Production Engineering and Maintenance, Seiten 205–214, 2019.

[Kuka2014] Kuka. *KUKA System Software 8.3 - Bedien- und Programmieranleitung für Systemintegratoren*, 2014.

[Landi2016] C.T. Landi, F. Ferraguti, C. Secchi & C. Fantuzzi. *Tool compensation in walk-through programming for admittance-controlled robots.* In IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society, Seiten 5335–5340, 2016.

[Lee2016] S.-D. Lee, K.-H. Ahn & J.-B. Song. *Torque Control based Sensorless Hand Guiding for Direct Robot Teaching.* In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems, Seiten 745–750, 2016.

[Liang2018] Y.S. Liang, D. Pellier, H. Fiorino, S. Pesty & M. Cakmak. *Simultaneous End-User Programming of Goals and Actions for Robotic Shelf Organization.* In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, Seiten 6566–6573, 2018.

[Lozano-Perez1983] T. Lozano-Perez. *Robot Programming.* Proceedings of the IEEE, Band 71, Nr. 7, Seiten 821–841, 1983.

[Maeda2015] Y. Maeda & T. Nakamura. *View-based teaching/playback for robotic manipulation.* ROBOMECH Journal, Band 2, Nr. 2, Seiten 1–12, 2015.

[Magix] MAGIX Software. *Magix Video Deluxe 2021.* `https://www.magix.com/de/videos-bearbeiten/video-deluxe/`. Abgerufen: 20.08.2021.

[Makris2014] S. Makris, P. Tsarouchi, D. Surdilovic & J. Krüger. *Intuitive dual arm robot programming for assembly operations.* CIRP Annals, Band 63, Nr. 1, Seiten 13–16, 2014.

[Massa2015] D. Massa, M. Callegari & C. Cristalli. *Manual guidance for industrial robot programming.* Industrial Robot, Band 42, Nr. 5, Seiten 457 – 465, 2015.

[Mateo2014] C. Mateo, A. Brunete, E. Gambao & M. Hernando. *Hammer: An Android Based Application for End-User Industrial Robot Programming.* In 2014 IEEE/ASME 10th International Conference on Mechatronic and Embedded Systems and Applications (MESA), 2014.

[MaterialDesign] Google. *Material Design Guide.* `https://material.io/design/guidelines-overview`. Abgerufen: 20.08.2021.

[MaterialPalette] *material design palette.* `https://www.materialpalette.com/`. Abgerufen: 20.08.2021.

[McCracken2003] D.D. McCracken & E.D. Reilly. Backus-naur form (bnf), Seite 129–131. John Wiley and Sons Ltd., 2003.

[Meng1997] J. Meng & F.-F. Chang. *CVEPS - a Compressed Video Editing and Parsing System.* In Proceedings of the Fourth ACM International Conference on Multimedia, MULTIMEDIA '96, Seite 43–53. Association for Computing Machinery, 1997.

[Meyer2007] C. Meyer, R. Hollmann, C. Parlitz & M. Hägele. *Programmieren durch Vormachen für Assistenzssysteme - Schweiß- und Klebebahnen intuitiv programmieren.* it - Information Technology, Band 49, Nr. 4, Seiten 238–246, 2007.

[Microsoft] Microsoft. *Surface Pro (5. Gen).* `https://www.microsoft.com/de-de/surface/`. Abgerufen: 20.08.2021.

[Møller] A. Møller. *Java BNF - Java Syntax Specification.* `https://cs.au.dk/~amoeller/RegAut/JavaBNF.html`. Abgerufen: 20.08.2021.

[Mohs2006a] C. Mohs, J. Hurtienne, M.C. Kindsmüller, J.H. Israel & H.A. Meyer. *IUUI Intuitive Use of User Interfaces: Auf dem Weg zu einer wissenschaftlichen Basis für das Schlagwort 'Intuitivität'.* In Sandro Leuchter, Leon Urbas & Martin R. K. Baumann, Herausgeber, MMI Interaktiv - Aufmerksamkeit und Situationawareness beim Autofahren: Vol. 1, No. 11, Seiten 75–84, 2006.

[Mohs2006b] C. Mohs, J. Hurtienne, J.H. Israel, A. Naumann, M.C. Kindsmüller, H. Meyer & A. Pohlmeyer. *IUUI - Intuitive Use of User Interfaces.* In Tagungsband UP06, Seiten 130–133, 2006.

[Mohs2006c] C. Mohs, J. Hurtienne, D. Scholz & M. Rötting. *Intuitivität: definierbar, beeinflussbar, überprüfbar!* Useware 2006 - Nutzergerechte Gestaltung technischer Systeme, VDI-Berichte 1946, Seiten 215–224, 2006.

[Mollard2015] Y. Mollard, T. Munzer, A. Baisero, M. Toussaint & M. Lopes. *Robot Programming from Demonstration, Feedback and Transfer.* In 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Seiten 1825–1831, 2015.

[Naumann2010] A. Naumann & J. Hurtienne. *Benchmarks for Intuitive Interaction with Mobile Devices.* In 2010 MobileHCI, Seiten 401–402, 2010.

[Neto2010] P. Neto, J.N. Pires & A.P. Moreira. *High-level programming and control for industrial robotics: using a hand-held accelerometer-based input device for gesture and posture recognition.* Industrial Robot: An International Journal, Band 37, Nr. 2, Seiten 137–147, 2010.

[Ni2017] D. Ni, A.W.W. Yew, S.K. Ong & A.Y.C. Nee. *Haptic and visual augmented reality interface for programming welding robots.* Advanced Manufacturing, Band 5, Seiten 191 – 198, 2017.

[Nicolescu2003] M.N. Nicolescu & M.J. Mataric. *Natural Methods for Robot Task Learning: Instructive Demonstrations, Generalization and Practice.* In AAMAS '03: Second International Joint Conference on Autonomous Agents and Multiagent Systems, Seiten 241–248, 2003.

[OConnor2016] A. O'Connor, A. Link & O. Oliver. *Economic Analysis of Technology Infrastructure Needs for Advanced Manufacturing: Advanced Robotics and Automation*, 2016.

[Onnasch2016] L. Onnasch, X. Maier & T. Jürgensohn. *Mensch-Roboter-Interaktion - Eine Taxonomie für alle Anwendungsfälle.* In baua: Fokus, Seiten 1–12, 2016.

[OpenShot] OpenShot Studios. *OpenShot Video Editor*. `https://www.openshot.org/de/`. Abgerufen: 20.08.2021.

[Orendt2016] E.M. Orendt, M. Fichtner & D. Henrich. *Robot Programming by Non-Experts: Intuitiveness and Robustness of One-Shot Robot Programming*. In IEEE 25th International Symposium on Robot and Human Interactive Communication (RO-MAN), Seiten 192–199, 2016.

[Orendt2017] E.M. Orendt, M. Fichtner & D. Henrich. *MINERIC Toolkit: Measuring Instruments to Evaluate Robustness and Intuitiveness of Robot Programming Concepts*. In 2017 26th International Symposium on Robot and Human Interactive Communication (RO-MAN), Seiten 1379–1386, 2017.

[OSHA] United States Departement of Labor. *Occupational Safety and Health Administration (OSHA) Technical Manual: Section IV: Chapter 4: Industrial Robots and Robot System Safety*. `https://www.osha.gov/otm/section-4-safety-hazards/chapter-4`. Abgerufen: 20.08.2021.

[Pan2012] Z. Pan, J. Polden, N. Larkin, S. van Duin & J. Norrish. *Recent progress on programming methods for industrial robots*. Robotics and Computer-Integrated Manufacturing, Band 28, Nr. 2, Seiten 87–94, 2012.

[Park2009] C. Park, K. Park, D.I. Park & J.-H. Kyung. *Dual Arm Robot Manipulator and Its Easy Teaching System*. In 2009 IEEE International Symposium on Assembly and Manufacturing, Seiten 242–247, 2009.

[Perzylo2016] A. Perzylo, N. Somani, S. Profanter, I. Kessler, M. Rickert & A. Knoll. *Intuitive Instruction of Industrial Robots: Semantic Process Descriptions for Small Lot Production*. In 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Seiten 2293–2300, 2016.

[Pires2007] J.N. Pires, T. Godinho & R. Araujo. *Using digital pens to program welding tasks*. Industrial Robot: An International Journal, Band 34, Nr. 6, Seiten 476–486, 2007.

[Pires2009] J.N. Pires, G. Veiga & R. Araujo. *Programming-by-demonstration in the coworker scenario for SMEs*. Industrial Robot: An International Journal, Band 36, Nr. 1, Seiten 73–83, 2009.

[Python] Python. *Full Grammar Specification*. `https://docs.python.org/3/reference/grammar.html`. Abgerufen: 20.08.2021.

[Quintero2018] C.P. Quintero, S. Li, M. Pan, W.P. Chan, H.F.M. van der Loos & E. Croft. *Robot Programming Through Augmented Trajectories in Augmented Reality*. In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, Seiten 1838–1844, 2018.

[Ragaglia2016] M. Ragaglia., A.M. Zanchettin, L. Bascetta & P. Rocco. *Accurate sensorless lead-through programming for lightweight robots in structured environments*. Robotics and Computer-Integrated Manufacturing, Band 39, Seiten 9 – 21, 2016.

[Rahul2014] R. Rahul, A. Whitchurch & M. Rao. *An open source graphical robot programming environment in introductory programming curriculum for undergraduates.* In 2014 IEEE International Conference on MOOC, Innovation and Technology in Education, Seiten 96–100, 2014.

[RoboDK] *RoboDK.* `https://robodk.com/`. Abgerufen: 20.08.2021.

[Robotiq] Robotiq. *Adaptiver 3-Finger-Robotergreifer.* `https://robotiq.com/de/produkte/adaptiver-3-finger-robotergreifer`. Abgerufen: 20.08.2021.

[Rossano2013] G.F. Rossano, C. Martinez, M. Hedelind, S. Murphy & T.A. Fuhlbrigge. *Easy Robot Programming Concepts: An Industrial Perspective.* In 2013 IEEE International Conference on Automation Science and Engineering (CASE), Seiten 1119–1126, 2013.

[Rozo2016] L. Rozo, S. Calinon, D.G. Caldwell, P. Jiménez & C. Torras. *Learning Physical Collaborative Robot Behaviors From Human Demonstrations.* IEEE Transactions on Robotics, Band 32, Nr. 3, Seiten 513–527, 2016.

[Rudorfer2018] M. Rudorfer, J. Guhl, P. Hoffmann & J. Krüger. *Holo Pick'n'Place.* In 2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA), Seiten 1219–1222, 2018.

[Safeea2017] M. Safeea, R. Bearee & P. Neto. *End-Effector Precise Hand-Guiding for Collaborative Robots.* In ROBOT 2017: Third Iberian Robotics Conference, Seiten 595–605, 2017.

[Schmidt2020] E. Schmidt, J. Winkelbauer, G. Puchas, D. Henrich & W. Krenkel. *Robot-Based Fiber Spray Process for Small Batch Production.* In Annals of Scientific Society for Assembly, Handling and Industrial Robotics, 2020.

[Schmitt2014] H. Schmitt, A. Hess, S. Hess, A. Maier, D. Löffler & J. Hurtienne. *Intuitive Benutzbarkeit messen - Eine Evaluationstoolbox für Software, Apps und technische Produkte.* In UP14 - Kurzvorträge, Stuttgart, 2014. German UPA.

[Schraft2006] R. D. Schraft & C. Meyer. *The Need for an Intuitive Teaching Method for Small and Medium Enterprises.* In Joint Conference of the International Symposium on Robotics (ISR) and the German Conference on Robotics (ROBOTIK), Seiten 95–105, 2006.

[Schreiber2010] G. Schreiber, A. Stemmer & R. Bischoff. *The Fast Research Interface for the KUKA Lightweight Robot.* In Workshop on IEEE ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications - How to Modify and Enhance Commercial Controllers, 2010.

[Schunk] Schunk. *PG70 Product Information.* `https://schunk.com/fileadmin/pim/docs/IM0004247.PDF`. Abgerufen: 20.08.2021.

[Shotcut] Meltytech. *Shotcut.* `https://shotcut.org/`. Abgerufen: 20.08.2021.

[SRS] Stäubli. *Stäubli Robotics Suite*. `https://www.staubli.com/de-de/robotics/produktprogramm/robotersoftware/staeubli-robotics-suite/`. Abgerufen: 20.08.2021.

[Stadler2016] S. Stadler, K. Kain, M. Giuliani, N. Mirnig, G. Stollnberger & M. Tscheligi. *Augmented Reality for Industrial Robot Programmers: Workload Analysis for Task-based, Augmented Reality-supported Robot Control*. In 2016 25th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), Seiten 179–184, 2016.

[statista2015] *Prognose zur Preisentwicklung eines Industrieroboters in den USA nach Posten bis zum Jahr 2025*. In Statista. `https://de.statista.com/statistik/daten/studie/416525/umfrage/preisentwicklung-eines-industrieroboters/`, 2015. Abgerufen: 20.08.2021.

[statista2019] *Umsatz mit Industrierobotern weltweit in den Jahren von 2018 bis 2025*. In Statista. `https://de.statista.com/statistik/daten/studie/870571/umfrage/umsatz-von-industrierobotern-weltweit/`, 2019. Abgerufen: 20.08.2021.

[statista2020a] *Geschätzter Bestand von Industrierobotern weltweit in den Jahren 2009 bis 2019*. In Statista. `https://de.statista.com/statistik/daten/studie/250212/umfrage/geschaetzter-bestand-von-industrierobotern-weltweit/`, 2020. Abgerufen: 20.08.2021.

[statista2020b] *Anteil von Unternehmen im Verarbeitenden Gewerbe mit Industrierobotern in Deutschland nach Anzahl der Beschäftigten in den Jahren 2018 und 2020*. In Statista. `https://de.statista.com/statistik/daten/studie/947397/umfrage/nutzung-von-industrierobotern-im-verarbeitenden-gewerbe-in-deutschland/`, 2020. Abgerufen: 20.08.2021.

[Steinmetz2018] F. Steinmetz, A. Wollschläger & R. Weitschat. *RAZER—A HRI for Visual Task-Level Programming and Intuitive Skill Parameterization*. IEEE Robotics and Automation Letters, Band 3, Nr. 3, Seiten 1362–1369, 2018.

[Stolka2003] P. Stolka & D. Henrich. *A Hybride Force Following Controller for Multi-Scale Motions*. In SYROCO 2003 - 7th International Symposium on Robot Control, Seiten 247–252, 2003.

[Stäubli] Stäubli. *TX2-90 Sechsachs-Roboter*. `https://www.staubli.com/de-de/robotics/produktprogramm/roboterarme/6-achs-roboter/tx2-90/`. Abgerufen: 20.08.2021.

[Stäubli2013] Stäubli. *uniVAL drive - innovative robot control*. `https://www.unival-drive.com/`, 2013. Abgerufen: 20.08.2021.

[Stäubli2016] Stäubli. *VAL 3 Reference Manual*, 2016.

[Stumm2016] S. Stumm, J. Braumann & S. Brell-Cokcan. *Human-Machine Interaction for Intuitive Programming of Assembly Tasks in Construction.* In 6th CIRP Conference on Assembly Technologies and Systems, Seiten 269–274, 2016.

[Stumm2018] S. Stumm, P. Devadass & S. Brell-Cokcan. *Haptic programming in construction.* Construction Robotics, Band 2, Seiten 3–13, 2018.

[Takarics2008] B. Takarics, P.T. Szemes, G. Nemeth & P. Korondi. *Welding trajectory reconstruction based on the Intelligent Space concept.* In 2008 Conference on Human System Interactions, Seiten 791–796, 2008.

[Tesler2012] L. Tesler. *A Personal History of Modeless Text Editing and Cut/Copy-Paste.* Interactions, Band 19, Nr. 4, Seiten 70–75, 2012.

[TS15066] *ISO/TS 15066 - Roboter und Robotikgeräte - Kollaborierende Roboter.* 2016.

[Tsumugiwa2001] T. Tsumugiwa, R. Yokogawa & K. Hara. *Variable Impedance Control with Virtual Stiffness for Human-Robot Cooperative Peg-in-Hole Task.* In 2002 IEEE/RSJ International Conference on Intelligent Robots and Systems, Seiten 1075–1081, 2002.

[Tykal2016] M. Tykal, A. Montebelli & V. Kyrki. *Incrementally Assisted Kinesthetic Teaching for Programming by Demonstration.* In 2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI), Seiten 205–212, 2016.

[Ullenboom2020] C. Ullenboom. Java ist auch eine insel. Rheinwerk Verlag, 2020.

[UR] *Universal Robots.* `https://www.universal-robots.com/de/`. Abgerufen: 20.08.2021.

[Vick2014] A. Vick, D. Surdilovic, A.K. Dräger & J. Krüger. *The industrial robot as intelligent tool carrier for human-robot interactive artwork.* In The 23rd IEEE International Symposium on Robot and Human Interactive Communication, Seiten 880–885, 2014.

[Villamor2010] C. Villamor, D. Willis & L. Wroblewski. *Touch Gesture Reference Guide.* 2010.

[Villani2018] V. Villani, F. Pini, F. Leali, C. Secchi & C. Fantuzzi. *Survey on Human-Robot Interaction for Robot Programming in Industrial Applications.* IFAC-PapersOnLine, Band 51, Nr. 11, Seiten 66–71, 2018.

[Wandelbots] *Wandelbots Teaching.* `https://wandelbots.com/de/roboterprogrammierung/`. Abgerufen: 20.08.2021.

[Wegerich2012] A. Wegerich, D. Löffler & A. Maier. *Handbuch zur IBIS Toolbox.* Seiten 1–23, 2012.

[Wölfel2021] K. Wölfel. Spiro sprachbasierte instruktion kraftbasierter roboterbewegungen. 2021.

[Wu2010] Y. Wu & Y. Demiris. *Towards One Shot Learning by Imitation for Humanoid Robots.* In 2010 IEEE International Conference on Robotics and Automation, Seiten 2889–2894, 2010.

[Zhang2018] Y. Zhang & T.-H. Kwok. *Design and Interaction Interface using Augmented Reality for Smart Manufacturing.* Procedia Manufacturing, Band 26, Seiten 1278 – 1286, 2018.

[Zimmer] Zimmer. *HRC-01 Kooperativer 2-Backen-Parallelgreifer.* `https://www.zimmer-group.com/de/technologien-komponenten/komponenten/handhabungstechnik/greifer/mrk/kooperativ/2-backen-parallelgreifer-mit-grossem-hub/hrc-01`. Abgerufen: 20.08.2021.

[Zöllner2004] R. Zöllner, T. Asfour & R. Dillman. *Programming by Demonstration: Dual-Arm Manipulation Tasks for Humanoid Robots.* In 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems, Seiten 479–484, 2004.

Eigene Publikationen

[Colceriu2020] C. Colceriu, M. Riedl, D. Henrich, S. Brell-Cokcan & V. Nitsch. *User-Centered Design of an Intuitive Robot Playback Programming System.* In T. Schüppstuhl, K. Tracht & D. Henrich, Herausgeber, Annals of Scientific Society for Assembly, Handling and Industrial Robotics, Seiten 193–202. Springer, 2020.

[Orendt2017b] E.M. Orendt, M. Riedl & D. Henrich. *Robust One-Shot Robot Programming by Demonstration Using Entity-Based Resources.* In C. Ferraresi & G. Quaglia, Herausgeber, Advances in Service and Industrial Robotics. RAAD 2017. Mechanisms and Machine Science, vol 49, Seiten 573–582. Springer, 2017.

[Riedl2016] M. Riedl, J. Baumgart & D. Henrich. *Editing and synchronizing multi-robot playback programs.* In Proceedings of ISR 2016: 47th International Symposium on Robotics, Seiten 200–207, 2016.

[Riedl2017] M. Riedl, E.M. Orendt & D. Henrich. *Sensor-Based Loops and Branches for Playback-Programmed Robot Systems.* In C. Ferraresi & G. Quaglia, Herausgeber, Advances in Service and Industrial Robotics. RAAD 2017. Mechanisms and Machine Science, vol 49, Seiten 183–190. Springer, 2017.

[Riedl2018] M. Riedl & D. Henrich. *Guiding Robots to Predefined Goal Positions.* In ISR 2018: 50th International Symposium on Robotics, Seiten 388–393, 2018.

[Riedl2019] M. Riedl & D. Henrich. *Fast Playback Robot Programming Using Video Editing Concepts.* In T. Schüppstuhl, K. Tracht & J. Roßmann, Herausgeber, Tagungsband des 4. Kongresses Montage Handhabung Industrieroboter, Seiten 259–268. Springer, 2019.

[Riedl2020a] M. Riedl & D. Henrich. *Scalable Visual Representation of Sensor-Based, Nested Robot Programs.* In 2020 Fourth IEEE International Conference on Robotic Computing (IRC), Seiten 232–239, 2020.

[Riedl2021] M. Riedl & D. Henrich. *Playback Robot Programming With Loop Increments.* In T. Schüppstuhl, K. Tracht & A. Raatz, Herausgeber, Annals of Scientific Society for Assembly, Handling and Industrial Robotics, Seiten 375–386. Springer, 2021.

Sonstige eigene Arbeiten

[Berger2021] J. Berger, C. Colceriu, A. Blank, J. Franke, C. Härdtlein, T. Hellig, D. Henrich, L. Heuss, M. Hiller, M. Krä, B. Leichtmann, A. Lottermoser, S. Lu, V. Nitsch, G. Reinhart, M. Riedl, S. Roder, K. Schäfer, J. Schilp, L. Vogt & M.F. Zäh. *Abschlussbericht: FORobotics - mobile, ad-hoc kooperierende Roboterteams.* Fraunhofer Publica: `http://publica.fraunhofer.de/documents/N-624794.html`, 2021. Abgerufen: 03.08.2021.

[Orendt2019] E. M. Orendt, M. Riedl, & D. Henrich. *Kapitel 5.3: Programmierung von Robotern.* In R. Müller, J. Franke, D. Henrich, B. Kuhlenkötter, A. Raatz & A. Verl, Herausgeber, Handbuch Mensch-Roboter-Kollaboration, Seiten 182–203. Carl Hanser Verlag, 2019. ISBN: 978-3-446-45016-5.

[Riedl2020b] M. Riedl, E. Schmidt & D. Henrich. *Videoschnitt trifft intuitive Roboterprogrammierung.* In handling 05/2020, Seiten 20–21, 2020.

Anhang

Inhalt

Der Anhang enthält die Dokumente, welche für die Evaluation des Gesamtsystems in Kapitel 8 verwendet wurden. Dazu gehören die Aufgabenbeschreibung welche den Studienteilnehmer:innen ausgehändigt wurde, Fragebogen F1 der von den Teilnehmer:innen vor Beginn der Studie ausgefüllt werden musste, Fragebogen F2, von welchem nach jeder Teilaufgabe eine Frage von den Teilnehmer:innen beantwortet werden musste, Fragebogen F3, der nach Erledigung aller Teilaufgaben ausgefüllt werden musste und der Beobachtungsbogen B, welcher vom Studienleiter ausgefüllt wurde, während ein:e Teilnehmer:in die Aufgaben der Studie löst.

Aufgabenbeschreibung der Evaluation

Evaluation zum Thema

„Kinästhetische, sensorbasierte Programmierung von Mehrrobotersystemen“

Die Programmierung von Industrierobotern ist in der Regel zeitintensiv, kostspielig und nur von Experten:innen machbar. Dies soll sich durch das in dieser Studie zu untersuchende Programmiersystem ändern. Das System erlaubt die Programmierung von Robotersystemen durch Nicht-Experten:innen ohne (textuelle) Programmierkenntnisse und nur mit Domänenwissen („Was soll der Roboter machen?“). Inwiefern diese Programmierung einfach und intuitiv ist soll im Rahmen dieser Studie herausgefunden werden.

Im Folgenden sollen mehrere Teilaufgaben zum Lösen der Gesamtaufgabe mit dem bereitgestellten Prototyp des Programmiersystems erledigt werden. Aufgrund der Corona-Pandemie wird dafür das Programmiersystem mit den simulierten Robotern verwendet. Das 3D-gedruckte Modell der Umgebung mit den beiden Robotermodellen ist ein Abbild des Laboraufbaus, der zur Veranschaulichung der auszuführenden Aufgabe dienen soll (siehe Abbildung 1). Die gesamte Bedienung des Programmiersystems erfolgt ausschließlich über das bereitgestellte Tablet.

Die Gesamtaufgabe, welche in mehreren Teilschritten erledigt werden soll, ist eine Übergabe von Werkstücken vom linken Roboter zum rechten Roboter mit anschließender Sortierung der Objekte nach deren Größe. Dabei soll der linke Roboter die Werkstücke von einer Palette aus aufgreifen und an der Übergabeposition ablegen. Anschließend soll der rechte Roboter das Werkstück von der Übergabeposition aufgreifen, und abhängig davon, ob es ein großes oder kleines Objekt ist, dieses in eine der beiden vorhandenen Kisten legen. Wenn die Palette leer ist, soll der linke Roboter automatisch eine Rückmeldung an den Werker geben, dass dieser eine neue Palette bringen soll. Folgende Teilaufgaben müssen zur Erfüllung der Gesamtaufgabe erledigt werden:

1. Programmieren des rechten Roboters
2. Editieren des Roboterprogramms des rechten Roboters mit Kopieren/Einfügen/Ausschneiden/Löschen
3. Verzweigung und Schleife in Roboterprogramm des rechten Roboters einfügen
4. Schleife und Schleifeninkrement zum leeren der Palette in das Roboterprogramm des linken Roboters einfügen
5. Synchronisation zwischen den Robotern und mit den Werker:innen in die Roboterprogramme einfügen

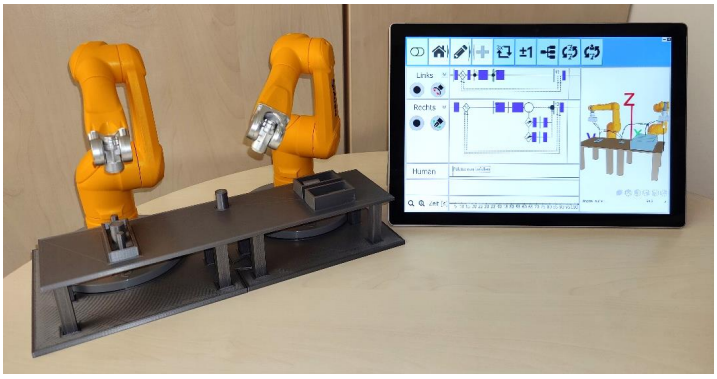


Abbildung 1: Aufbau der Studie.

Aufbau des Programmiersystems:

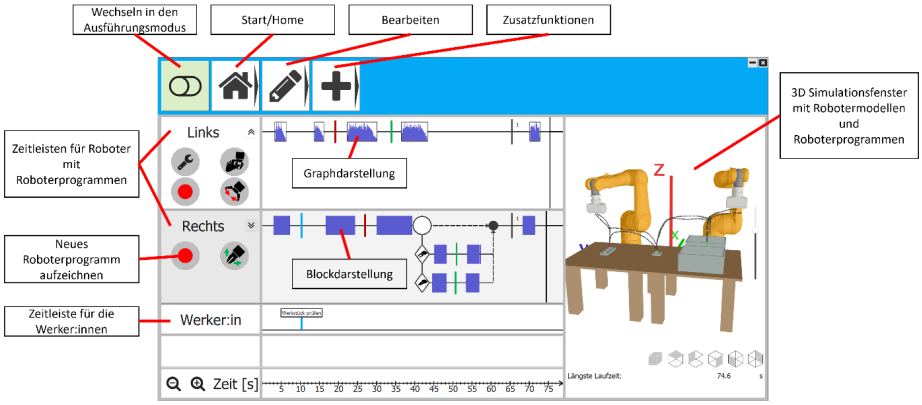
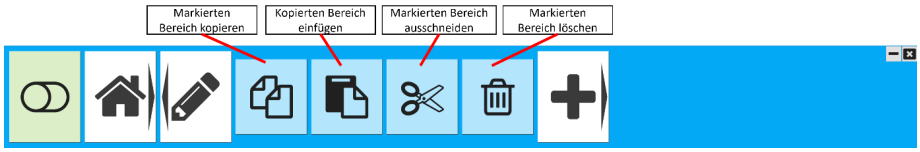
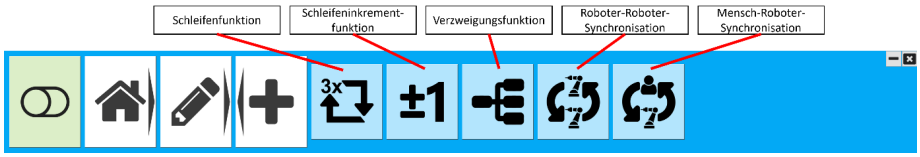


Abbildung 2: Bildschirmaufnahme des Programmiersystems mit zwei Robotern („Links“ und „Rechts“) und eine:r Werker:in. Die Funktionen der Kontrollleiste (oben, blau) sind eingeklappt. Der grüne Button wechselt vom Programmiermodus in den Ausführungsmodus.

Funktionen der Kontrollleiste:





Ausgangslage:

Zu Beginn der Studie befindet sich vor Ihnen das Programmiersystem mit zwei Robotern („Links“ und „Rechts“) und einer Zeitleiste für die Werker:innen. Für Roboter „Links“ wurde bereits ein Roboterprogramm aufgezeichnet. Mit den einzelnen Teilaufgaben soll nun durch Sie das gesamte System so programmiert werden, dass es die Übergabe- und Sortieraufgabe lösen kann.

Teilaufgabe 1:

Programmieren des rechten Roboters

Bei der kinästhetischen Programmierung werden die Roboter mit der Hand entlang der auszuführenden Roboterbahn geführt. Während des Führens zeichnet das Programmiersystem die Bahn auf, so dass diese anschließend exakt wieder abgespielt werden kann. In dieser Teilaufgabe sollen Sie den Roboter „Rechts“ so programmieren, dass er von der Übergabeposition in der Mitte des Tisches ein Werkstück aufgreift und in eine der Kisten wirft. Hierzu muss folgendermaßen vorgegangen werden:

(1) Starten Sie die Aufzeichnung eines neuen Roboterprogramms für „Rechts“.
(2) Führen sie den Roboter entlang der gewünschten Bahn und öffnen und schließen Sie den Greifer an den richtigen Positionen.
(Dadurch, dass diese Evaluation ohne realen Roboter durchgeführt wird, muss dieser Schritt entfallen)
(3) Beenden Sie die Aufzeichnung des Roboterprogramms und inspizieren Sie das Ergebnis in der graphischen Benutzungsoberfläche.

Bitte **Frage a)** von Fragebogen **F2** ausfüllen.

Teilaufgabe 2:

Editieren des Roboterprogramms mit Kopieren/Einfügen/Ausschneiden/Löschen

Wie Sie im Simulationsfenster bzw. am 3D-gedruckten Aufbau sehen können, besitzt die Palette drei Plätze für Werkstücke. Insgesamt muss der rechte Roboter also drei Mal ein Objekt von der Übergabeposition aufnehmen und in die Kiste werfen. In dieser Teilaufgabe sollen sie das Roboterprogramm des rechten Roboters mit den Bearbeitungsfunktionen so editieren, dass der Roboter „Rechts“ insgesamt drei Werkstücke in eine der Kisten wirft. Hierzu muss folgendermaßen vorgegangen werden:

(1) Kopieren Sie den Bereich des Roboterprogramms von „Rechts“, in dem der Roboter das Werkstück aufgreift und anschließend in die Kiste wirft.
(2) Fügen Sie den Kopierten Teil des Roboterprogramms an der passenden Stelle in das Roboterprogramm so oft wieder ein, bis „Rechts“ insgesamt drei Werkstücke in die Kiste wirft.
(3) Inspizieren Sie das Ergebnis und überprüfen Sie, ob tatsächlich drei Werkstücke in die Kiste geworfen werden.

Bitte **Frage b)** von Fragebogen **F2** ausfüllen.

Teilaufgabe 3:

Verzweigung und Schleife in Roboterprogramm des rechten Roboters einfügen

Das Roboterprogramm des Roboters „Rechts“ wird nun zurückgesetzt auf den Ausgangszustand. Nun soll vor Ablegen des Werkstücks durch eine Verzweigung festgestellt werden, wie groß das Werkstück im Greifer ist und dieses abhängig davon in eine der beiden Kisten sortiert werden (große Werkstücke sollen in die hintere Kiste, kleine Werkstücke in die vordere). Anschließend soll mit der Schleifenfunktion anstatt der Editierfunktionen dafür gesorgt werden, dass die Bewegung zur Übergabeposition und das Sortieren insgesamt drei Mal ausgeführt werden. Hierzu muss folgendermaßen vorgegangen werden:

(1) Fügen Sie mit der Verzweigungsfunktion eine Verzweigung an der Stelle von „Rechts“ ein, an der sich der Roboter nach dem Aufgreifen des Werkstückes wieder über den Kisten befindet.
(2) Fügen Sie zu dieser Verzweigung zwei Zweige hinzu und ändern Sie für jeden der beiden Zweige den Sensorwert und die auszuführende Bewegung.
(3) Markieren Sie nun in der Zeitleiste den Bereich des Roboterprogramms, der wiederholt ausgeführt werden soll und fügen Sie mit Hilfe der Schleifenfunktion eine Zählschleife in das Roboterprogramm ein, welche drei Mal ausgeführt werden soll.
(4) Inspizieren Sie das Ergebnis und überprüfen Sie, ob die Teilaufgabe korrekt ausgeführt wird.

Bitte **Frage c)** von Fragebogen **F2** ausfüllen.

Teilaufgabe 4:

Schleife und Schleifeninkrement zum leeren der Palette in Roboterprogramm des linken Roboters einfügen

Nun ist das Roboterprogramm des Roboters „Rechts“ fertig. Für den Roboter „Links“ wird ein Programm bereitgestellt, welches den Roboter zur ersten Position auf der Palette verfährt, dort das Werkstück greift, zur Übergabeposition verfährt, das Werkstück ablegt und wieder zurück zu einer Position über der Palette verfährt. Der linke Roboter soll die Palette leeren, die Werkstücke zur Übergabeposition bringen und dort ablegen. Das Problem hierbei ist, dass bedingt durch die kinästhetische Programmierung des Roboters dieser nur exakt die gleiche Bewegung wie aufgezeichnet abspielen kann. Mit Hilfe der Schleifeninkrement-Funktion kann das geändert werden. Hierbei kann für eine Position innerhalb einer Schleife ein sogenanntes Inkrement (Veränderung der Position in x-, y- und z-Richtung) angegeben werden, damit der Roboter nach jeder Wiederholung der Schleife die ausgewählte Inkrement-Position um das Inkrement verändert und somit diese neue Position anfährt. In dieser Teilaufgabe soll dieses Schleifeninkrement dafür verwendet werden, dass der linke Roboter die Palette leert, ohne dass explizit jede Bewegung des Roboters vorgegeben werden muss. Hierzu muss folgendermaßen vorgegangen werden:

(1) Markieren Sie den Bereich des Roboterprogramms von „Links“, der wiederholt ausgeführt werden soll und fügen Sie mit Hilfe der Schleifenfunktion eine Zählschleife in das Roboterprogramm ein, welche drei Mal ausgeführt werden soll.
(2) Fügen Sie an der Position, an der das Werkstück auf der Palette gegriffen wird, in das Programm von „Links“ mit Hilfe der Schleifeninkrement-Funktion eine Inkrement-Position ein. Setzen Sie das Inkrement auf „-0,070“ m in y-Richtung (Die Werkstücke auf der Palette sind 7 cm auseinander). Dadurch wird nach jedem Schleifendurchlauf die neue Position auf der Palette erreicht.

→ Weiter auf der nächsten Seite

(3) Fügen Sie an der Position, an der das Werkstück auf der Übergabeposition abgelegt wird, in das Programm von „Links“ mit Hilfe der Schleifeninkrement-Funktion eine Inkrement-Position ein. Lassen Sie alle Einträge des Inkrements auf „0,000“, damit der Roboter bei jedem Schleifendurchlauf exakt die gleiche Position anfährt.
(4) Inspizieren Sie das Ergebnis und überprüfen Sie, ob die Teilaufgabe korrekt ausgeführt wird.

Bitte **Frage d)** von Fragebogen **F2** ausfüllen.

Teilaufgabe 5:

Synchronisation zwischen den Robotern und den Werker:innen

Wie Sie in der Simulation erkennen können, kann es an der Übergabeposition zwischen den beiden Robotern zu einer Kollision kommen. Um dies zu verhindern müssen beide Roboterprogramme aufeinander synchronisiert werden, so dass immer nur einer der beiden Roboter an der Übergabeposition ist. Zugleich sollen Werker:innen vor Beginn einer jeden Ausführung des Roboterprogramms von „Links“ benachrichtigt werden, dass die Palette neu befüllt werden muss oder ausgetauscht werden muss. Dies soll in dieser Teilaufgabe gelöst werden. Hierzu muss folgendermaßen vorgegangen werden:

(1) Fügen Sie zwischen der letzten Bewegung von „Links“ in der Schleife und dem Schleifenende mit Hilfe der Roboter-Roboter Synchronisationsfunktion einen Synchronisationspunkt mit ID „1“ ein.
(2) Fügen Sie zwischen dem Schleifenbeginn von „Rechts“ und der ersten Bewegung in der Schleife mit Hilfe der Roboter-Roboter Synchronisationsfunktion einen Synchronisationspunkt ebenfalls mit ID „1“ ein.
(3) Fügen Sie zwischen dem Synchronisationspunkt mit ID 1 und dem Schleifenende in „Links“ mit Hilfe der Roboter-Roboter Synchronisationsfunktion einen Synchronisationspunkt mit ID „2“ ein.
(4) Fügen Sie zwischen dem Ende der Verzweigung und dem Ende der Schleife in „Rechts“ mit Hilfe der Roboter-Roboter Synchronisationsfunktion einen Synchronisationspunkt ebenfalls mit ID „2“ ein.
(5) Fügen Sie vor Beginn der Schleife von „Links“ mit Hilfe der Mensch-Roboter Synchronisationsfunktion einen Synchronisationspunkt ein mit dem Titel „Palette befüllen“ und der Beschreibung „Bitte Palette befüllen oder neue Palette bereitstellen.“ ein.
(6) Wechseln Sie das Programmiersystem in den Abspielmodus und starten Sie das Abspielen der Roboterprogramme. Inspizieren Sie dabei im Simulationsfenster, ob die Gesamtaufgabe korrekt erfüllt wird.

Bitte **Frage e)** von Fragebogen **F2** ausfüllen.

Bitte Fragebogen **F3** ausfüllen.

Fragebogen F1

1. Allgemeine Angaben

a) Alter: [ ] Jahre

b) Geschlecht:
- ○ Weiblich
- ○ Männlich
- ○ Anderes (bitte angeben): [ ]

c) Was ist ihr höchster Bildungsabschluss? Wenn Sie derzeit noch in Ausbildung sind, welchen Bildungsabschluss streben Sie an?
- ○ Kein Schulabschluss
- ○ Haupt-/Mittelschulabschluss
- ○ Realschulabschluss
- ○ (Fach-) Abitur
- ○ Abgeschlossenes (Fach-) Hochschulstudium

d) Arbeiten Sie in einem technischen Beruf (Mechatroniker, Mechaniker, Maschinenführer, …), haben Sie eine Berufsausbildung in einem technischen Beruf, studieren Sie einen technischen Studiengang oder haben Sie ein abgeschlossenes Hochschulstudium mit technischer Ausrichtung (Ingenieurwesen, Naturwissenschaften, Informatik, …)?
- ○ Ja
- ○ Nein

2. Vorkenntnisse

Haben Sie Erfahrung mit folgenden Tätigkeiten?

a) Textverarbeitung mit Maus und Tastatur? (Texte kopieren, einfügen, …)
- ○ Ja
- ○ Nein

b) Textverarbeitung per Touchscreen?
- ○ Ja
- ○ Nein

c) Audio- oder Videoschnitt mit Maus und Tastatur?
- ○ Ja
- ○ Nein

d) Audio- oder Videoschnitt per Touchscreen?
- ○ Ja
- ○ Nein

e) (Textuelles) Programmieren im Allgemeinen?
- ○ Ja
- ○ Nein

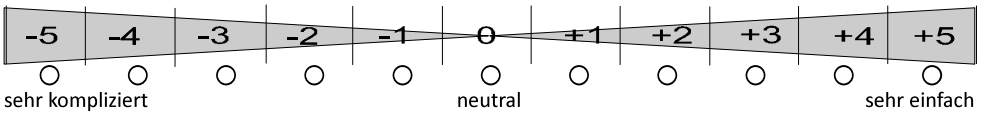
f) Bedienung von Haushaltsrobotern? (Saugroboter, Mähroboter, …)
- ○ Ja
- ○ Nein

g) Bedienung von Industrierobotern? (Roboterarme)
- ○ Ja
- ○ Nein

h) Programmierung von Industrierobotern?
- ○ Ja
- ○ Nein

3. Erwartete Komplexität

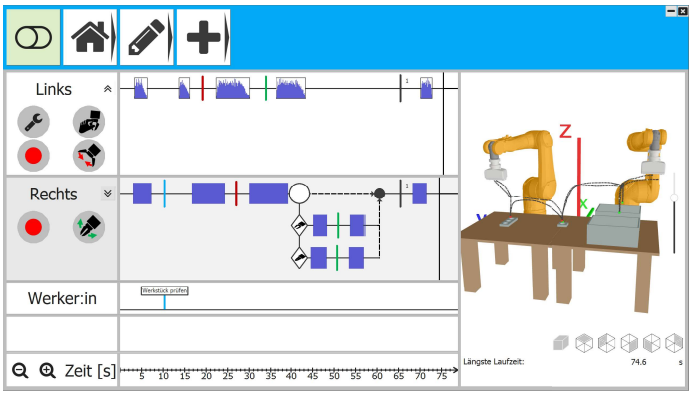
Wie schätzen Sie die Komplexität der Bedienung des vorliegenden Programmiersystems ein?



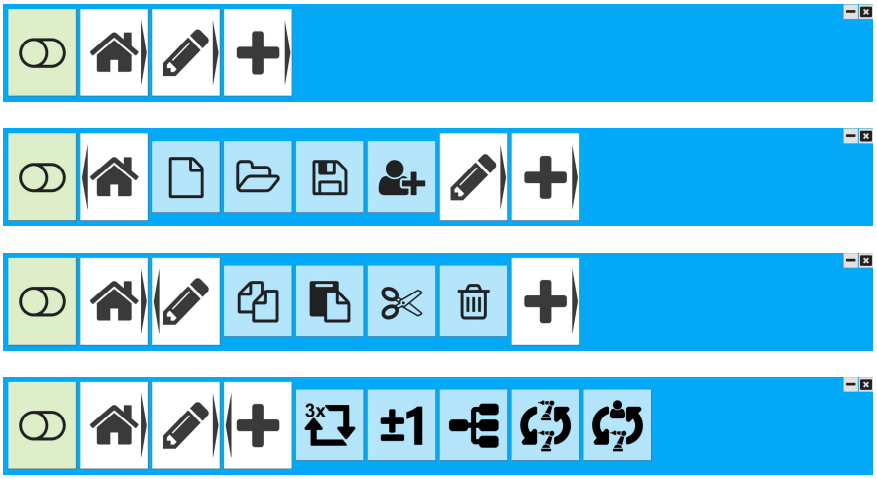
4. Bedeutung der Symbole der Kontrollleiste

Die graphische Benutzungsoberfläche des Programmiersystems versucht so weit wie möglich auf Text zu verzichten und verwendet stattdessen Symbole, insbesondere in der Kontrollleiste. Mit den nächsten Fragen soll herausgefunden werden, inwieweit diese Symbole im Kontext der kinästhetischen Roboterprogrammierung verständlich sind.

Der Aufbau des Programmiersystems wird in folgender Bildschirmaufnahme gezeigt. Insbesondere die Symbole der Bedienelemente der Kontrollleiste (oben, blau) und der Roboter-spezifischen Funktionen (links, rund, grau) sind von Interesse.



Es gibt zwei Arten von Bedienelementen in der Kontrollleiste: Übergeordnete Ordner (weißer Hintergrund) und Roboterfunktionen (blauer Hintergrund), welche von den Ordnern gruppiert werden.
Die Kontrollleiste kann Folgende Zustände einnehmen:



Evaluation „Kinästhetische Programmierung von Mehrrobotersystemen“
Fragebogen F1

Bitte beschreiben Sie im Folgenden kurz mit einem Stichwort bzw. einem Satz:
Was wird mit dem gezeigten Ordner (weißer rechteckig) gruppiert?
bzw.
Was für eine Funktion hat das jeweilige Bedienelement (blau rechteckig bzw. grau rund)?

a) 

b) 

c) 

d) 

e) 

f) 

g) 

h) 

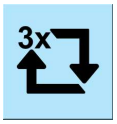
i) 

j) 

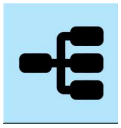
Weitere Symbole auf der nächsten Seite →

Evaluation „Kinästhetische Programmierung von Mehrrobotersystemen“
Fragebogen F1

k) 

l) 

m) 

n) 

o) 

p) 

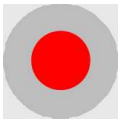
q) 

r) 

s) 

t) 

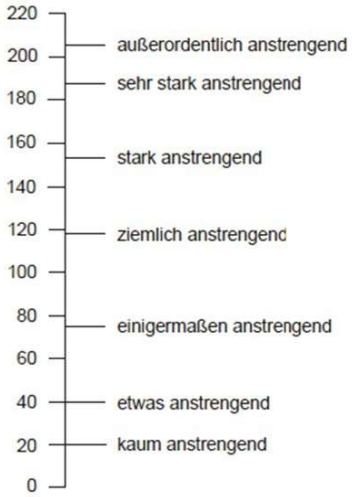
u) 

v) 

Fragebogen F2

5. Mentale Beanspruchung

Bitte geben Sie anhand der folgenden Skala einen Wert für die mentale Beanspruchung je Teilaufgabe für die Benutzung des Programmiersystems an.
Der Wert muss zwischen 0 und 220 liegen.



a) Teilaufgabe 1: ☐

b) Teilaufgabe 2: ☐

c) Teilaufgabe 3: ☐

d) Teilaufgabe 4: ☐

e) Teilaufgabe 5: ☐

Fragebogen F3

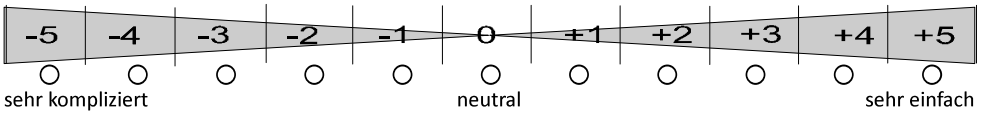
6. QUESI – Questionnaire for Intuitive Use

Versuchen Sie Ihre Einschätzungen des Programmiersystems ausschließlich auf die Benutzung des Systems zu beziehen (und nicht z.B. auf die Schwierigkeit der Aufgabe an sich). Es gibt keine richtigen oder falschen Antworten. Bitte antworten Sie spontan und lassen Sie keine Fragen aus.

		trifft gar nicht zu	trifft wenig zu	trifft teils-teils zu	trifft ziemlich zu	trifft völlig zu
a)	Es gelang mir, das System ohne Nachdenken zu benutzen.	○	○	○	○	○
b)	Ich habe erreicht, was ich mit dem System erreichen wollte.	○	○	○	○	○
c)	Mir war sofort klar, wie das System funktioniert.	○	○	○	○	○
d)	Der Umgang mit dem System erschien mir vertraut.	○	○	○	○	○
e)	Bei der Benutzung des Systems sind keine Probleme aufgetreten.	○	○	○	○	○
f)	Die Systembenutzung war unkompliziert.	○	○	○	○	○
g)	Es gelang mir, meine Ziele so zu erreichen, wie ich es mir vorgestellt habe.	○	○	○	○	○
h)	Es fiel mir von Anfang an leicht, das System zu benutzen.	○	○	○	○	○
i)	Mir war immer klar, was ich tun musste, um das System zu benutzen.	○	○	○	○	○
j)	Die Benutzung des Systems verlief reibungslos.	○	○	○	○	○
k)	Ich musste mich kaum auf die Benutzung des Systems konzentrieren.	○	○	○	○	○
l)	Das System hat mich dabei unterstützt, meine Ziele vollständig zu erreichen.	○	○	○	○	○
m)	Die Benutzung des Systems war mir auf Anhieb klar.	○	○	○	○	○
n)	Ich tat immer automatisch das Richtige, um mein Ziel zu erreichen.	○	○	○	○	○

7. Empfundene Komplexität

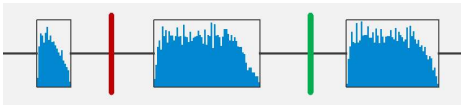
Wie komplex war die Bedienung des vorliegenden Programmiersystems tatsächlich?



8. Darstellungsarten

Welche der beiden Darstellungsarten (Graphdarstellung oder Blockdarstellung) für Roboterprogramme entlang der Zeitleiste bevorzugen Sie?

○ Graphdarstellung:



○ Blockdarstellung:

9. Kopieren/Einfügen oder Schleife

Welche Art zur Eingabe von mehrfach auszuführenden Bewegungen bevorzugen Sie? Kopieren und mehrfaches Einfügen (siehe Teilaufgabe 2) oder eine Zählschleife (siehe Teilaufgabe 3 bzw. Teilaufgabe 4)?

○ Kopieren und mehrfaches Einfügen

○ Zählschleife

10. Sonstiges

Haben Sie weitere Anmerkungen oder Kommentare zum evaluierten Programmiersystem?

Vielen Dank für Ihre Teilnahme!

Beobachtungsbogen B

11. Zeiterfassung und Allgemeines

a) Datum:

b) Beginn Studie:

c) Ende Studie:

d) Dauer Studie:

e) Interviewer:

f) Allgemeine Anmerkungen:

12. Vollständigkeit und Genauigkeit

Bewertungskriterien für die Genauigkeit und die Vollständigkeit. Für jede Teilaufgabe wird die Genauigkeit mit bis zu 3 Punkten und die Vollständigkeit mit bis zu 2 Punkten bewertet.

Genauigkeit:		Vollständigkeit:	
3	Der Nutzer hat das Programmiersystem ohne Probleme genutzt (bewusst; gezielt)	2	Die Aufgabe wurde vollständig erfüllt
2	Der Nutzer hat das Programmiersystem mit Trial & Error genutzt (auf gut Glück; teilweise unbewusst)	1	Die Aufgabe wurde teilweise erfüllt
1	Der Nutzer hat das Programmiersystem mit einem einzigen Hinweis des Moderators genutzt	0	Die Aufgabe wurde nicht erfüllt
0	Der Nutzer hat das Programmiersystem mit permanenter Unterstützung des Moderators genutzt		

a) Teilaufgabe 1:

i. Beginn:

ii. Genauigkeit: ○3 ○2 ○1 ○0

iii. Vollständigkeit: ○2 ○1 ○0

iv. Ende:

v. Dauer:

vi. Beobachtung:

Evaluation „Kinästhetische Programmierung von Mehrrobotersystemen“
Beobachtungsbogen B

b) Teilaufgabe 2:

i. Beginn: ☐

ii. Genauigkeit: ○3 ○2 ○1 ○0

iii. Vollständigkeit: ○2 ○1 ○0

iv. Ende: ☐

v. Dauer: ☐

vi. Beobachtung:

c) Teilaufgabe 3:

i. Beginn: ☐

ii. Genauigkeit: ○3 ○2 ○1 ○0

iii. Vollständigkeit: ○2 ○1 ○0

iv. Ende: ☐

v. Dauer: ☐

vi. Beobachtung:

d) Teilaufgabe 4:

i. Beginn: ☐

ii. Genauigkeit: ○3 ○2 ○1 ○0

iii. Vollständigkeit: ○2 ○1 ○0

iv. Ende: ☐

v. Dauer: ☐

vi. Beobachtung:

e) Teilaufgabe 5:

i. Beginn: ☐

ii. Genauigkeit: ○3 ○2 ○1 ○0

iii. Vollständigkeit: ○2 ○1 ○0

iv. Ende: ☐

v. Dauer: ☐

vi. Beobachtung:

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

Michael Riedl