Technische Universität München
Fakultät für Informatik

Herstellerunabhängige Integration von Industrierobotersystemen in den automobilen Karosseriebau

Maxim Simeon Taschew

Vollständiger Abdruck der von der Fakultät der Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Vorsitzender: Prof. Dr. Uwe Baumgarten

Prüfer der Dissertation:

1. Prof. Dr.-Ing. Alois Knoll

2. Prof. Dr.-Ing. Bernd Kuhlenkötter, Ruhr-Universität Bochum

Die Dissertation wurde am 13.04.2021 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 18.08.2021 angenommen.

Kurzfassung

Der automobile Karosseriebau stellt seit Beginn des Zeitalters der Industrierobotik den Hauptanwender und zentralen Anforderungsgeber für Automatisierungstechnik in der Großserienproduktion dar. Ebenso lang unterliegt die Integration von Industrierobotern erhöhten Aufwänden aufgrund von proprietärer und herstellerspezifischer Systemgestaltung, die den Einsatz von Automatisierungslösungen sowohl für Klein- als auch Großunternehmen erschwert. Obwohl das Thema der Herstellerneutralität bereits seit den 90er Jahren in unterschiedlichsten Facetten direkt und implizit in Forschung und Wissenschaft aufgegriffen wird, ist die Abhängigkeit von Roboterherstellern weiterhin eine ungelöste und zentrale Problemstellung in der industriellen Gegenwart der Automobilproduktion. Die vorliegende Arbeit adressiert diese Problematik und erarbeitet einen Lösungsvorschlag zur herstellerunabhängigen Integration von Robotersystemen in den automobilen Karosseriebau anhand der folgenden vier Themenschwerpunkte: Anforderungsermittlung und -ableitung aus Anwendersicht, hersteller-unabhängige Steuerung von Robotersystemen, herstellerunabhängige Programmierung von Robotersystemen, prototypische Umsetzung und Evaluierung.

Im ersten Schritt werden der gegenwärtige Integrationsprozess von Robotersystemen im Karosseriebau anhand eines Fallbeispiels beschrieben, bestehende Einschränkungen durch die herstellerspezifischen Systeme identifiziert und zentrale Anforderungen an die vorhandenen Arbeitspakete abgeleitet.

Anschließend wird die Frage der passenden herstellerunabhängigen Steuerung von Robotersystemen bearbeitet. Hierzu werden zur Verfügung stehende Methoden der externen Ansteuerung analysiert und anhand ihrer Integrationstiefe kategorisiert, um darauf aufbauend einen Bewertungskatalog zu entwickeln, aus Karosseriebausicht zu valutieren und das geeignetste Verfahren zu bestimmen. Offene Fragestellungen in Bezug auf die Leistungsfähigkeit und Vorteile für die herstellerneutrale Applikationsstandardisierung werden nachfolgend in Laborversuchen bearbeitet und die Einsetzbarkeit des gewählten Steuerungskonzeptes erfolgreich nachgewiesen. Ebenso werden die Defizite gegenwärtiger steuerungstechnischer Angebote aufgezeigt und Lösungsvorschläge dargestellt.

Die Möglichkeit, dem proprietären Robotersystem Applikationslogiken zu entziehen, bietet neue Gestaltungsfreiheiten in der Programmierung und führt zur Fragestellung der geeigneten Programmiermethode und -sprache. Zur Bearbeitung dieser Thematik wird der Status quo der textuellen Roboterprogrammierung sowohl auf Lieferanten- als auch auf Anwenderebene untersucht und anhand der quantitativen Instruktionsanalyse von Roboterprogrammen zweier Automobilwerke ein Anforderungsbild aus Karosseriebausicht abgeleitet. Mittels der gewonnenen Erkenntnisse wird die Eignung unterschiedlicher Programmiersprachen und -paradigmen aus Roboter-, Automatisierungs- und Softwaretechnik bewertet. Aufgrund mangelnder Tauglichkeit einer einzelnen Option wird ein hybrides Programmier-sprachenkonzept vorgeschlagen, das bei der herstellerunabhängigen Programmierung von Robotersystemen die Vorteile domänenspezifischer und allgemeiner Hochsprachen verbindet.

Im letzten Teil der Arbeit erfolgt die prototypische Umsetzung und Evaluierung zunächst anhand eines Programmiersystems für Werksanwender und einer Softwareentwicklungsumgebung für Roboterexperten. Die abschließende, übergreifende Validierung geschieht auf Basis einer herstellerunabhängigen Karosseriebauzelle mit Robotern unterschiedlicher Hersteller und der Implementierung zentraler Roboterapplikationen in einer Open-Source-Steuerung.

Abstract

Since the beginning of the industrial robot age, the automotive body shop has been the main user and central demand generator for automation technology in large-scale production. For just as long, the integration of industrial robots has been subject to increased efforts due to proprietary and manufacturer-specific system design, which makes the use of automation solutions difficult for both small- and large-scale companies. Although the topic of manufacturer neutrality has been directly and implicitly addressed in research and science in a wide variety of facets since the 90s, the dependence on robot manufacturers remains an unresolved and central problem in the industrial present of automotive production.

This thesis addresses these issues and develops a solution proposal for the manufacturer-independent integration of robot systems in the automotive body shop based on four main topics: identification and derivation of requirements from a user perspective, manufacturer-independent control of robot systems, manufacturer-independent programming of robot systems, prototypical implementation and evaluation.

First of all, the current integration process of robot systems in the automotive body shop is described on the basis of a case study, existing limitations due to the manufacturer-specific systems are identified and central requirements for the existing work packages are derived.

The question of a suitable manufacturer-independent control of robot systems is addressed next. For this purpose, available methods of external control are analyzed and categorized according to their level of integration. On that basis, an evaluation catalog is developed, valuated from the point of view of a body shop, and the most suitable method determined. Furthermore open questions regarding the performance and advantages for manufacturer-independent application standardization are subsequently addressed and successfully demonstrated in laboratory tests. Likewise, the deficits of current control technology offered are noted and solutions presented.

The possibility of removing application logic from proprietary robotic systems offers new design freedom in programming and leads to the question of the appropriate programming method and language. To answer this question, the status quo of textual robot programming is investigated both on supplier and user level. Using quantitative instruction analysis of the robot programs of two automotive plants, a requirement picture from the body shop perspective is derived. Based on the findings, the suitability of different programming languages and paradigms from robotics, automation and software engineering are evaluated. Since a single option alone is not suitable, a hybrid programming language concept is proposed which combines the advantages of domain-specific and general high-level languages in the manufacturer-independent programming of robot systems.

In the last part of the thesis, the prototypical implementation and evaluation is first carried out by using a programming system for factory users and a software development environment for robot experts. Subsequently, the final overarching validation is conducted on the basis of a manufacturer-independent body shop cell with robots from different manufacturers and the implementation of central robotic applications in an open-source controller.

Vorwort des Autors

Die vorliegende Arbeit hat mehrere Jahre meines Lebens stark geprägt und wäre ohne die Hilfe und das Engagement einer Reihe von Personen während meiner Tätigkeit in der Vorentwicklung, Standardisierung und Integration bei der BMW AG in dieser Form nicht möglich gewesen. Ob Betreuer, Vorgesetzte, Kollegen, Projektpartner oder Freunde und Familie: Mein Dank gebührt unter anderem folgenden Menschen, die mich im Laufe meines umfangreichen Dissertationsvorhabens unterstützt, aufgebaut und mir den notwendigen Freiraum gewährt haben.

Mein besonderer Dank gilt Herrn Professor Alois Knoll für die Übernahme der langjährigen Betreuung meiner Doktorarbeit, die durchgängige Gesprächsbereitschaft und das starke Interesse an wissenschaftlichen Fragestellungen in Kombination mit den industriellen und praxisrelevanten Heraus- und Anforderungen. Herrn Professor Bernd Kuhlenkötter danke ich sehr für das entgegengebrachte Interesse an meiner Arbeit, die Übernahme des Korreferats und den finalen Input für den Feinschliff der Publikation meiner Dissertation.

Auf Unternehmensseite möchte ich Dr. Thomas Herzinger, Dr. Till Werneck, Dr. Peter Weber, Stefan Bartscher und Dr. Stephan Oertelt hervorheben und danken. Das ab dem ersten Arbeitstag entgegengebrachte Vertrauen, die Förderung, der gewährte Freiraum, die Unterstützung, die gelegentlich z. T. berechtigte Kritik, sowie die Bereitstellung von Ressourcen und einer zuverlässigen Betreuung, trotz Hektik und Stress des Tagesgeschäfts, sind nur einige der nennenswerten Gründe, warum diese Personen maßgeblich zur initialen Idee, der Durchführung und dem Gelingen dieser Arbeit beigetragen haben.

Auf fachlicher Seite möchte ich mich herzlich bei Dr. Markus Rickert, Daniel Bunse, Ahmed Ghazi, Adolf Ermer, Dr. Max Kossmann und insbesondere bei Dr. Johannes Bix bedanken, sowohl für die produktive gemeinsame Arbeit an Projekten und die spannende Zeit im Roboterlabor als auch für die telefonische Notfallhilfe bei diversen wissenschafts- (Markus), roboter- bzw. taschenrechner- (Daniel), karosseriebau- (Adi), formatierungsspezifischen (Max) und natürlich universellen Fragestellungen jeglicher Art (Johannes).

Darüber hinaus geht mein freundschaftlicher Dank an Menschen, die mich während meiner Dissertation begleitet, durchweg unterstützt, aber auch für zahlreiche Ablenkung gesorgt haben und zu sehr guten Freunden geworden sind. Neben Daniel Bunse, Dr. Johannes Bix, Dr. Jakob Dinse und Dr. Markus Hagenmaier ist insbesondere der „Konspirative Austausch“, stellvertretend für das soziale Gefüge des BMW-Doktorandenprogramms, zu nennen. Ihr habt mir das Doktorandendasein nicht nur halbwegs erträglich, sondern beinahe wiederholenswert und mit Sicherheit unvergesslich gemacht. Es war eine tolle Zeit!

Zu guter Letzt und mit tiefstem und herzlichem Dank möchte ich meine Eltern Nora und Ilija sowie meine Freundin erwähnen. Ohne Euch hätte ich weder die Einstellung noch die notwendigen Freiräume, Ressourcen und ausreichende Unterstützung gehabt, mich derart intensiv meiner akademischen Ausbildung zu widmen – ohne die nicht-akademischen Aspekte und Pflichten im Leben dabei nicht zu sehr zu vernachlässigen.

Burgthann und München, im Februar 2022 Maxim Taschew

Inhaltsverzeichnis

1. Einleitung ... 1
   1.1. Problemstellung und Motivation ... 1
   1.2. Zielstellung, Prämissen und Aufbau der Arbeit ... 3
2. Stand von Wissenschaft und Technik ... 5
   2.1. Industrieroboter – Grundlagen ... 5
   2.2. Industrierobotermarkt ... 9
   2.3. Herstellerabhängigkeit in der Industrierobotik ... 11
   2.4. Lösungsansätze in Forschung und Industrie ... 15
3. Abgrenzung, zentrale Forschungsfragen und Vorgehen ... 30
4. Roboterintegration am Beispiel der BMW AG ... 36
   4.1. Robotereinsatz in den Technologien der Automobilproduktion ... 36
   4.2. Steuerungsarchitektur von Karosseriebauzellen ... 38
   4.3. Roboterapplikationsstandardisierung im Karosseriebau ... 39
   4.4. Integrationsprozess von Roboteranwendungen ... 42
   4.5. Beeinträchtigungen durch herstellerspezifische Robotersysteme ... 44
   4.6. Schlussfolgerung und Anforderungskonkretisierung der Forschungsfragen ... 45
5. Herstellerunabhängige Steuerung von Robotersystemen ... 48
   5.1. Methoden der externen Steuerung von Robotern ... 50
   5.1.1. Funktionsanalyse anhand der Integrationstiefe ... 51
   5.1.2. Gesamtübersicht ... 56
   5.1.3. Anforderungskatalog und Bewertung aus Endanwendersicht ... 58
   5.2. Leistungsmessungen ... 63
   5.2.1. Untersuchungs- und Auswertemethodik ... 65
   5.2.2. Test der Posekenngrößen ... 73
   5.2.3. Test der Bahnkenngrößen ... 80
   5.2.4. Zusammenfassung der Messergebnisse ... 92
   5.3. Integrationsprozess von externen Steuerungssystemen ... 94
   5.3.1. Identifizierte Defizite und Verbesserungsvorschläge ... 94
   5.3.2. Plug & Produce-Verfahren ... 95
   5.4. Zusammenfassung ... 98
6. Herstellerunabhängige Programmierung von Robotersystemen ... 99
   6.1. Methoden und Sprachen der Roboterprogrammierung ... 100
   6.2. Fragestellung und Vorgehen ... 105
   6.3. Analyse der textuellen Roboterprogrammierung im Karosseriebau ... 107
   6.3.1. Programmiersprachen für Industrieroboter ... 107
   6.3.2. Programmierung von Automobilapplikationen am Beispiel der BMW AG ... 115

6.3.3. Schlussfolgerung und Fazit 125
6.4. Alternative herstellerunabhängige Programmiersprachen für Robotersysteme 126
6.5. Anforderungen an die Programmierung aus Anwendersicht 129
6.6. Vorschlag eines hybriden Programmiersprachenkonzeptes 133
6.7. AIRL – Automotive Industrial Robot Language 136
6.8. Zusammenfassung 138
7. Prototypische Umsetzung und Evaluierung 139
7.1. Herstellerunabhängige Programmierung auf Instandhaltungsebene 140
7.1.1. Anforderungen 141
7.1.2. Umsetzung 142
7.1.3. Evaluierung 144
7.2. Herstellerunabhängige Programmierung auf Expertenebene 146
7.2.1. Anforderungen 147
7.2.2. Umsetzung 147
7.2.3. Evaluierung 150
7.3. Herstellerunabhängige Karosseriebauroboterzelle 151
7.3.1. Anforderungen 151
7.3.2. Exemplarische Karosseriebauapplikationen 151
7.3.3. Layout und Komponenten der Testzelle 153
7.3.4. Steuerungsarchitektur und Sicherheitskonzept 157
7.3.5. Umsetzung 160
7.3.6. Evaluierung 163
7.4. Bewertung aus Gesamtunternehmenssicht 165
8. Zusammenfassung und Ausblick 168
9. Anhang 172
9.1. Test der Posekenngrößen 172
9.2. Test der Bahnkenngrößen 190
9.3. Quellcode der Testprogramme 202
9.4. Herstellerunabhängige Programmierung von Robotersystemen 202
9.5. Prototypische Umsetzung und Evaluierung 216
9.6. Präzisierung Eigenanteil und Zuarbeiten 217
9.7. Komprimierte Darstellung der Forschungsschwerpunkte und -ergebnisse 219
Abbildungsverzeichnis 220
Tabellenverzeichnis 224
Verzeichnis Abkürzungen und Akronyme 225
Literaturverzeichnis 227

1. Einleitung

1.1. Problemstellung und Motivation

Digitalisierung, Industrie 4.0, Internet of Things, Cyber-Physische Produktionssysteme, Servicerobotik: Die derzeitigen Trends und Schlagworte sowohl in Gesellschaft als auch Automatisierungsindustrie scheinen, ebenso wie die damit einhergehenden Potenziale, beinahe grenzenlos. Eine übergreifende Gemeinsamkeit besteht darin, Optimierung und Problemlösung in der softwareorientierten Umsetzung von Aufgabenstellungen zu sehen und den Einsatz von Automatisierungstechnik auf neue Anwendungsfelder zu erweitern oder in bestehenden Systemen zu verbessern. Flexibilität, Interoperabilität und Kompatibilität stellen zentrale Schwerpunkte dar, die jedoch nicht erst neuerdings in das Betrachtungsfeld von Forschung und Wissenschaft gelangt sind, sondern bereits seit Beginn der rechnergestützten Fertigung vor mehr als 50 Jahren eine bedeutende Rolle spielen.

So vielseitig die Potenziale der Automatisierungstechnik sind, so vielseitig sind ebenso die aktuellen Herausforderungen eines ihrer Hauptanwender – der Automobilindustrie. Ein offener technologischer Wandel auf Antriebsseite, der Eintritt zahlreicher neuer Wettbewerber, zunehmende Regulierung mit strengeren Gesetzesvorgaben, globale Handelskonflikte und Zollunsicherheiten bei gleichzeitig stagnierendem Gesamtmarkt und rückgängigen Gewinnen charakterisieren das Polylemma, welchem die Automobilindustrie und damit insbesondere auch der Wirtschaftsstandort Deutschland bereits vor den COVID-19-bedingten Marktturbulenzen ausgesetzt war. Flexible und interoperable Automatisierungslösungen sind daher ein wichtiger Beitrag, um das automobile Produktionssystem hinsichtlich Aufbau, Produkt und Standort zügig anzupassen, auf Marktveränderungen entsprechend reagieren zu können und die Wettbewerbsfähigkeit sicherzustellen. Aufgrund seiner hohen Automatisierungsquote ist der Karosseriebau diesbezüglich der wichtigste Anwendungsbereich innerhalb eines Automobilwerkes und den mit dem Einsatz von Automatisierungstechnik verbundenen Vor- und Nachteilen am stärksten ausgesetzt. Dies manifestiert sich insbesondere im Einsatz einer der prägendsten Komponenten des Karosseriebaus – dem Industrieroboter. Zwar bietet dieser im Kern ein sehr flexibles Produktionswerkzeug, mit dem von Handhabung über Schweißen bis hin zum Vermessen von Bauteilen unterschiedlichste Produktionsprozesse realisiert werden können. Dem täglichen Roboterbetrieb geht allerdings ein langjähriger Befähigungs- und Integrationsprozess voraus, der neben der fertigungstechnischen Eignung auch die Interoperabilität und Kompatibilität mit über- und untergeordneten Komponenten des unternehmensspezifischen Produktionssystems sicherstellt.

Auf diese Weise ist es für ein Automobilunternehmen zwar möglich, Roboteranlagen weltweit nach einem vorgegebenen Standard zügig in Betrieb zu nehmen, da sowohl Fertigungsprozesse als auch Technik definiert sind und auf bereits vorhandenen Softwarebausteinen und Schnittstellen aufgesetzt werden können. Jedoch ist dies angesichts der Proprietarität der Robotersysteme nicht lieferantenübergreifend einheitlich umsetzbar. Zudem erschweren unterschiedliche Bedienkonzepte die Gestaltung einer universellen, auf den Karosseriebau ausgelegten Handhabung der Robotersysteme. Die Integration von Robotersystemen in die Automobilproduktion ist ein zeit- und kostenintensiver Prozess, der sowohl fertigungs- und steuerungstechnische als auch organisatorische Aspekte betrifft und mit jedem Roboterlieferanten redundant durchlaufen werden muss. Die Folgen sind erhöhte Aufwände, eine geringere Flexibilität und eine langsamere Integration von Fertigungsinnovationen sowie die eingeschränkte Möglichkeit der kontinuierlichen Verbesserung – in Summe eine reduzierte

Wettbewerbsfähigkeit. Diese Problematik betrifft nicht nur die Großserienproduktion. Auch kleine und mittelständische Unternehmen scheuen häufig den mit dem Robotereinsatz verbundenen technologischen und organisatorischen Aufwand, der sich beispielsweise in Form von Mitarbeiterschulungen oder Umrüstkosten für Produktumstellungen äußert. Ebenso erschwert die herstellerspezifische Systemgestaltung sowohl den Wechsel eines bereits in einem Unternehmen etablierten Lieferanten als auch die übergreifende Verwendung von Robotersystemen unterschiedlicher Hersteller.

Die verbesserte Integration von Roboteranwendungen und die interoperable, herstellerneutrale Verwendung von Robotersystemen stellen neben den erwähnten industriellen Problemstellungen auch Schwerpunkte vergangener und aktueller Forschungsarbeiten in Steuerungstechnik und Robotik dar. Ebenso steigt die Anzahl industrieller Ansätze, die produkt-, prozess- und schnittstellenseitig die Einbindung von Robotersystemen herstellerübergreifend vereinfachen sollen, zunehmend. Die industrielle Gegenwart der Automobilproduktion zeigt, dass weder Lösungsvorschläge der Wissenschaft noch der Industrie bisher eine umfassende Erleichterung der herstellerunabhängigen Integration von Robotersystemen bewirken konnten. Zudem fokussiert sich das Anforderungsbild von Forschungsarbeiten häufig auf die Befähigung von Automatisierungslösungen bei kleinen und mittelständischen Betrieben, da diese aufgrund der geringen Automatisierungsquote als das attraktivere Anwendungsfeld erscheinen. Neben der grundsätzlichen Realisierung der Anwendung stehen dabei häufig die Systemauslegung für kleine Losgrößen und die Gestaltung einfacher Bedienkonzepte für Anwender ohne oder mit nur geringen Kenntnissen von Roboter- oder Automatisierungstechnik im Fokus. Sogleich diese Ansätze zweifelsohne auch Potenziale für den Karosseriebau bieten, erfolgte bei der Entwicklung der existierenden Konzepte weder eine Analyse noch ein Abgleich mit den sich aus dem Status quo ergebenden prozessualen und technischen Anforderungen der industriellen Großserienfertigung.

Ähnliche Defizite in Bezug auf die Systemauslegung bieten häufig technologiegetriebene Arbeiten, die diverse Steuerungs-, Programmiersprachen- oder Middlewarekonzepte entwickeln, um unterschiedlichsten Problemstellungen zu begegnen, dabei auch das Thema der mangelnden Interoperabilität aufgreifen und in ihre Lösungsvorschläge einarbeiten. Die „Herstellerunabhängigkeit“ stellt hierbei meist nicht den zentralen Forschungsschwerpunkt, sondern lediglich einen Randaspekt bzw. Befähiger dar. Ebenso sind die Einsatzfelder nicht auf klassische „Industrierobotik“, sondern auf Servicerobotik-Lösungen fokussiert, um beispielsweise bisher nicht automatisierte komplexe Anwendungsfelder in der Montage oder im Dienstleistungsbereich zu realisieren. Der „automobile Karosseriebau“ als Hauptanwender von Industrierobotern wird ebenso wie sein technologisches und organisatorisches Anforderungsspektrum größtenteils außer Acht gelassen.

Die vorliegende Arbeit greift die beschriebenen Problemstellungen der „herstellerunabhängigen Integration“ von „Industrierobotersystemen“ in den „automobilen Karosseriebau“ fokussiert und kombiniert auf. Ziel ist es, sowohl die damit verbundenen Herausforderungen als auch die notwendigen anwendungsbezogenen Anforderungen zu identifizieren, potenzielle Lösungsvorschläge zu entwickeln und an einem praxisgerechten Prototyp zu implementieren und zu evaluieren.

1.2. Zielstellung, Prämissen und Aufbau der Arbeit

Für die herstellerunabhängige Roboterintegration im automobilen Karosseriebau existiert gegenwärtig weder eine prozessuale noch eine technische Betrachtung der Anforderungs- und Lösungsseite. Diese zentrale Fragestellung wird in der vorliegenden Arbeit aufgegriffen und unter folgenden Zielstellungen und Prämissen bearbeitet:

- Anwendungsbezug Karosseriebau
  Die Automobilindustrie ist der Hauptabnehmer von Industrierobotern. Der Karosseriebau ist wiederum der größte Anwender innerhalb eines Automobilwerkes. Die Anwendungen des Karosseriebaus stellen einen erheblichen Anteil der Applikationen von Industrierobotern dar und bieten deshalb eine Untersuchungsgrundlage von allgemeiner und hoher Bedeutung. Der Automobilkarosseriebau ist zwar in der Industrierobotik allgegenwärtig, aber in der gängigen Literatur hinsichtlich der Integration von Robotersystemen nur unzureichend beschrieben. Die Erarbeitung des Status quo anhand des Karosseriebaus stellt daher einen Schwerpunkt dar.

- Anforderungsableitung aus Anwendersicht
  Die Fokussierung des Untersuchungsgegenstands auf einen industriellen Anwendungsbereich bedingt zunächst eine entsprechende Anforderungsableitung. Ziel der Arbeit ist es daher, durch konsequente Orientierung an den identifizierten Industrieanforderungen einen technischen und prozessualen Lösungsraum zu erarbeiten und den geeignetsten Lösungsvorschlag zu evaluieren. Eine umfassende Ableitung und Erarbeitung aller Anforderungen und Anwendungen des Karosseriebaus ist leistungsseitig nicht erfüllbar. Die Arbeit konzentriert sich deshalb auf zentrale exemplarische Anwendungen und deren Anforderungen zur Umsetzung.

- Nachweis von Realisierbarkeit und technischer Leistungsfähigkeit
  Aufgrund des starken Anwendungsbezuges ist die Praxistauglichkeit der erarbeiteten Lösungen eine signifikante Grundanforderung. Die grundsätzliche Realisierbarkeit und die erforderliche technische Leistungsfähigkeit werden für die essenziellen Inhalte nicht nur theoretisch erarbeitet, sondern durch konkrete Versuchsaufbauten und Experimente nachgewiesen.

- Zielvision und Fallbeispiel
  Um die geforderten Belange hinsichtlich Anwendungsbezug, Anforderungsableitung und Leistungserfüllung realitätsgetreu bearbeiten zu können, wird ein konkretes Fallbeispiel eines Automobilunternehmens, der BMW AG, als Untersuchungsgegenstand gewählt. Neben der Bereitstellung von Spezifikationsgrundlagen sowie fertigungs- und steuerungstechnischer Expertise konnte anhand der Anwenderanforderung des Fallbeispiels ein Zukunftsszenario als Zielvision und Orientierungshilfe entwickelt werden. Dieses zeigt in Abbildung 1 ein Visionsbild der Industrieroboterintegration im Karosseriebau, in dem eine hochflexible Steuerungsarchitektur die Anwenderlogik von der eingesetzten Hardware trennt, die flexible Einbindung relevanter Soft- und Hardware ermöglicht und so eine schnelle und einfache produktorientierte Umgestaltung des Produktionssystems ermöglicht.

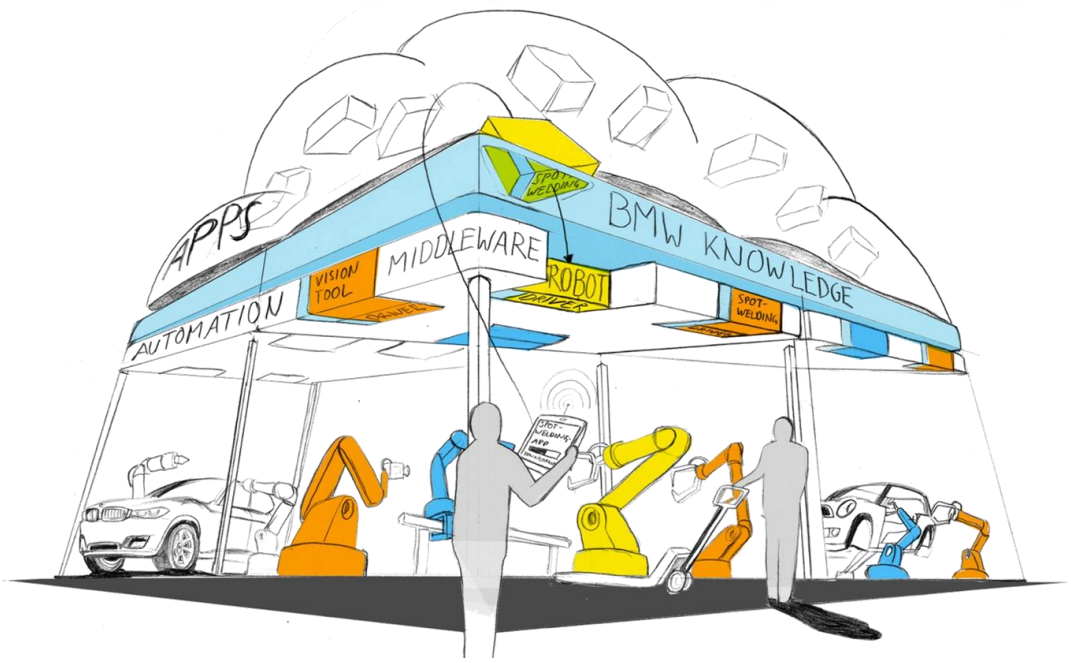


Abbildung 1 – Visionäres Zielbild am Fallbeispiel der BMW AG

Aufbau der Arbeit

Dem allgemeinen Stand von Wissenschaft und Technik wird sich in **Kapitel 2** gewidmet. Neben der Erläuterung von grundlegenden Begriffen, technischen Zusammenhängen und den gegenwärtigen Auswirkungen von Herstellerabhängigkeit werden wissenschaftliche und industrielle Lösungsansätze beschrieben, welche die Integration und Verwendung von Industrierobotern herstellerneutral gestalten sollen. Darauf aufbauend erfolgt in **Kapitel 3** die Abgrenzung zu den existierenden Ansätzen, die Entwicklung der zentralen Forschungsfragen der Arbeit und die Erläuterung der Methodik sowie des gewählten Fallbeispiels zur Bearbeitung dieser Fragestellungen, die in den vier darauffolgenden Kapiteln beantwortet werden.

In **Kapitel 4** wird das Anwendungsfeld Karosseriebau konkretisiert, indem die Einsatzgebiete und der Integrationsprozess von Industrierobotern beschrieben und die bestehenden Beeinträchtigungen dargestellt werden, die sich aufgrund der Proprietarität von Robotersystemen ergeben. Abschließend werden die grundsätzlichen Anforderungen für die Hauptteile der Arbeit in Kapitel 5, 6 und 7 abgeleitet. In **Kapitel 5** wird die herstellerunabhängige Steuerung von Industrierobotersystemen auf Basis externer Steuerungskonzepte betrachtet. Diese werden hinsichtlich ihrer Eignung für den Karosseriebau theoretisch sowie praktisch analysiert und ein geeignetes Steuerungskonzept für den Karosseriebau identifiziert. Darüber hinaus werden Defizite gegenwärtiger Umsetzungen und Ansätze zur Verbesserung aufgezeigt. **Kapitel 6** beinhaltet das Themengebiet der herstellerunabhängigen Programmierung, indem Roboterprogrammiersprachen und -methoden sowie die gegenwärtigen Anforderungen des Karosseriebaus untersucht werden und ein anwendergerechtes Programmiersprachenkonzept für die unterschiedlichen Nutzergruppen entwickelt wird. Die prototypische Umsetzung und Evaluierung der Lösungsvorschläge erfolgt in **Kapitel 7** sowohl anhand der Programmierung von Roboterapplikationen als auch der herstellerunabhängigen Realisierung einer Karosseriebautestzelle mit Robotersystemen verschiedener Hersteller.

Abschließend werden in **Kapitel 8** die Ergebnisse zusammengefasst und darauf aufbauende Fragestellungen erörtert, die potenzielle Forschungsschwerpunkte zukünftiger Arbeiten bieten.

2. Stand von Wissenschaft und Technik

2.1. Industrieroboter – Grundlagen

Begriffserläuterung und Einsatzbereiche

Der Ursprung der Terminologie „Roboter“ ist in der Science-Fiction-Kultur der 20er Jahre des letzten Jahrhunderts zu finden. In Karel Čapes sozialutopischem Drama „R.U.R.“ produziert das Unternehmen „Rossums Universal Robots“ Androiden als willige und günstige Arbeitskräfte, die im Laufe des Stückes ein eigenes Bewusstsein entwickeln, gegen ihre Erschaffer rebellieren und die Menschheit zerstören [1]. Im literarischen Sinne abgeleitet aus den tschechischen Wörtern „robota“ und „robotník“, die für Zwangsdienst bzw. Fronarbeiter stehen [2], hat sich der Begriff „Roboter“ anschließend als Bezeichnung für diverse technische Apparaturen etabliert.

Auch heute sind die vorhandenen Definitionen des Wortes ebenso unterschiedlich wie die darunter zu verstehenden technischen Gebilde und deren Anwendungsgebiete. Humanoide, mobile Roboter, Servicerobotik, automatisierte Softwaresysteme, autonome Fahrzeuge – um nur einige Beispiele zu nennen. Im Kontext dieser Arbeit ist vor allem der Begriff des Industrieroboters von Bedeutung, der sich in der deutschen Normierungslandschaft unter anderem in den folgenden zwei Definitionen wiederfindet:

„Industrieroboter sind universell einsetzbare Bewegungsautomaten mit mehreren Achsen, deren Bewegungen hinsichtlich Bewegungsfolge und Wegen bzw. Winkeln frei (d. h. ohne mechanischen Eingriff) programmierbar und gegebenenfalls sensorgeführt sind. Sie sind mit Greifern, Werkzeugen oder anderen Fertigungsmitteln ausrüstbar und können Handhabungs- und/oder Fertigungsaufgaben ausführen.“ (VDI-Richtlinie 2860 [3])

„Automatisch gesteuerter, frei programmierbarer Mehrzweck-Manipulator, der in drei oder mehr Achsen programmierbar ist und zur Verwendung in der Automatisierungstechnik entweder an einem festen Ort oder beweglich angeordnet sein kann.“ (DIN EN ISO 8373 [4])

Eingesetzt werden Industrieroboter hauptsächlich für Handhabungs- und Schweißanwendungen in Produktionsbetrieben. Zwei von drei verkauften Robotern werden nur für diese beiden Anwendungsbereiche eingesetzt. Trotz des stark zunehmenden Angebots an Robotersystemen für Montageprozesse stellt diese Kategorie mit ca. 10 % Abnahmeanteil lediglich das dritthäufigste Einsatzgebiet von Industrierobotern dar [5].

Eine ähnlich klare Tendenz ist in Bezug auf die Abnehmerbranchen zu erkennen. Mehr als 50 % aller ausgelieferten Systeme finden in der Automobil- und Elektronikfertigung Anwendung. Mit über 105.000 installierten Einheiten war die Automobilindustrie im Jahr 2019 vor der Elektronikbranche (88.000 Stück) weiterhin der größte Verwender von Industrierobotern (Abbildung 2). Aufgrund der außerordentlichen Dominanz dieser beiden Industriezweige bezüglich des Einsatzes von Robotersystemen bieten sie sehr repräsentative Untersuchungsgegenstände für wissenschaftliche Arbeiten in der Industrierobotik. Dem Forschungs- und Industriestandort Deutschland kommt angesichts des hohen Anteils der Automobilindustrie an der gesamtwirtschaftlichen Wertschöpfung eine besonders große Bedeutung zu. Daher konzentriert sich vorliegende Arbeit konsequenterweise auf dieses Einsatzgebiet.

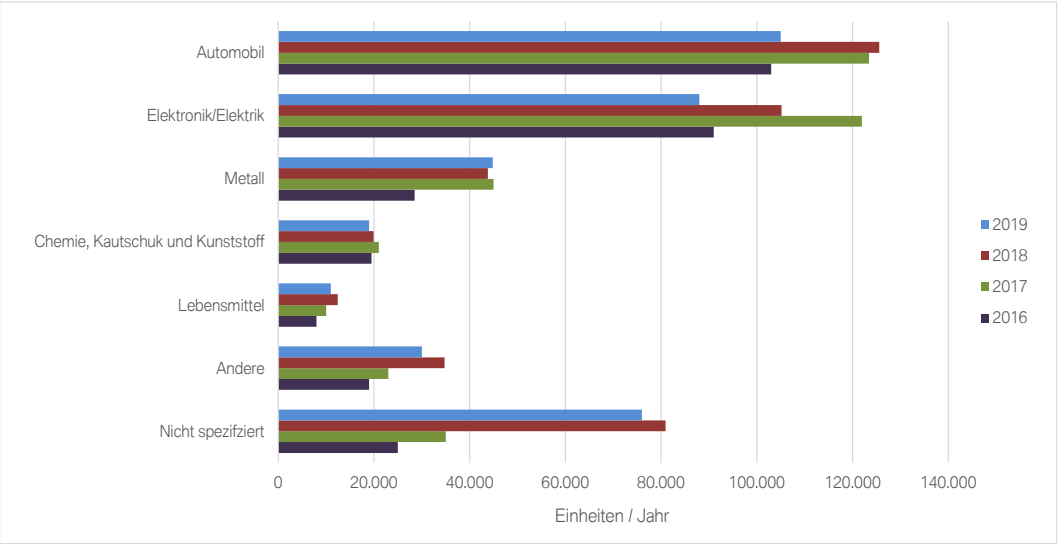


Abbildung 2 – Abnehmer von Industrierobotern nach Branchen für die Jahre 2016 bis 2019 (Datenquelle: [5, 6])

Aufbau eines Industrieroboters

Gemäß DIN EN ISO 8373 [4] sind unter einem Industrieroboter sowohl der Manipulator einschließlich Aktorik als auch die Steuerung samt Hardware und Software zu verstehen. Diese Komponenten sollen im Folgenden hinsichtlich Aufbau und Funktionsweise näher erläutert werden. Abbildung 3 stellt die zentralen Elemente Manipulator (1), Elektrische Verbindung (2) zur Steuerung (3) und das Programmierhandgerät (4) für ein Kuka-Robotersystem dar.

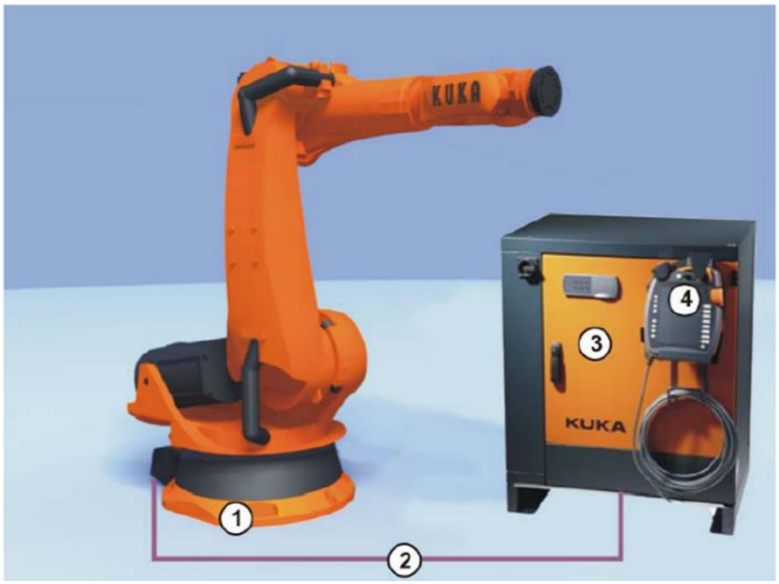


Abbildung 3 – Komponenten eines Industrieroboters [7]

Unter Manipulator, im allgemeinen Sprachgebrauch auch Roboterarm bezeichnet, ist die Robotermechanik zu verstehen. Gemäß der Norm [4] handelt es sich dabei um eine *„Maschine, deren Mechanismus aus einer Folge von Komponenten besteht, durch Gelenke oder gegeneinander verschieblich verbunden, mit dem Zweck, Gegenstände (Werkstücke oder Werkzeuge) zu greifen und/oder zu bewegen, normalerweise mit mehreren Freiheitsgraden“.* Darunter fallen nicht nur der in Abbildung 3 dargestellte Gelenk- bzw. Knickarmroboter, sondern auch andere Kinematiktypen wie Portal- oder SCARA-Roboter. In der Produktion eines

Automobilherstellers sind derzeit allerdings fast ausschließlich 6-Achs-Knickarmroboter im Einsatz.

In der Mechatronik dienen Aktoren als Stellglieder zwischen elektrischen Signalen und physikalischen Größen in einem technischen Prozess. Im Bereich der Robotik ist unter Aktorik folglich das angebrachte Werkzeug, wie z. B. ein Greifer oder eine Schweißzange zu verstehen, die ebenfalls als Endeffektor bezeichnet werden und je nach Anwendung variieren. Eine weitere Komponente des Roboterarms stellen elektrisch betriebene Motoren dar, die über Umrichter der Robotersteuerung geregelt werden und die Bewegung des Manipulators ausführen.

Die Aufgaben der Robotersteuerung sind neben der angesprochenen Koordination und Durchführung der Roboterbewegung auch die Funktionsüberwachung des Systems, die Kommunikation mit dem Anwender [8] und die Kontrolle der angeschlossenen Peripherie über die Verarbeitung von digitalen und analogen Ein- und Ausgangssignalen. Um die funktionalen Zusammenhänge besser verstehen zu können, wird der Aufbau einer Robotersteuerung anhand Abbildung 4 detailliert erläutert.

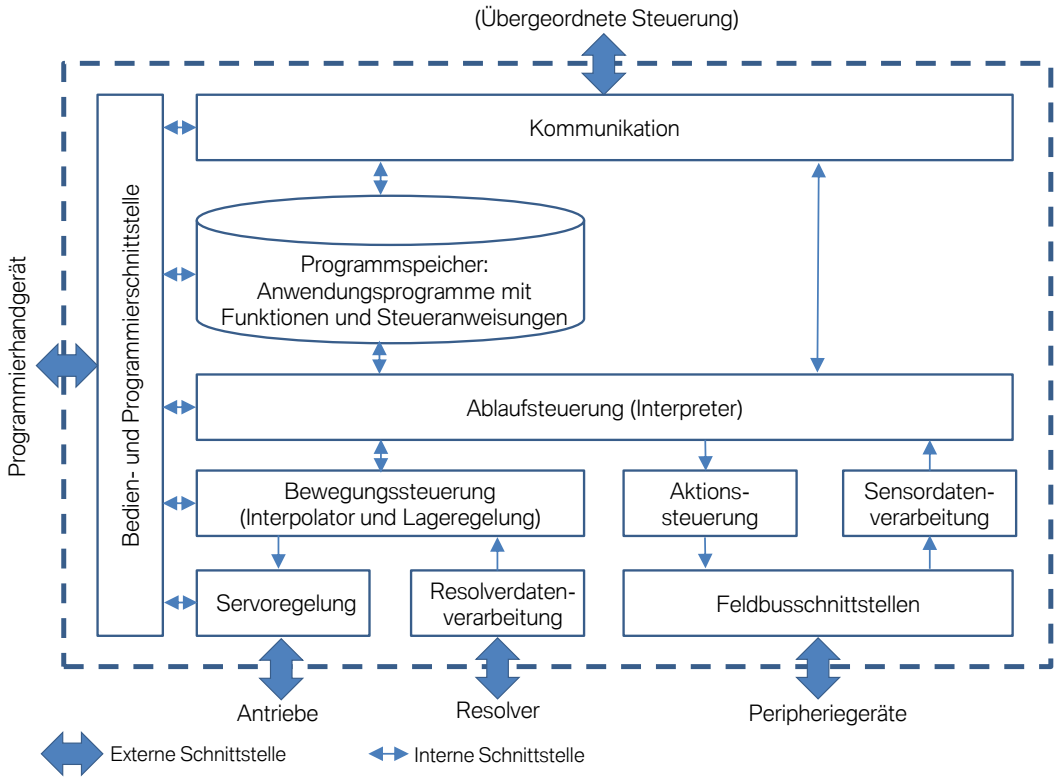


Abbildung 4 – Aufbau einer Industrierobotersteuerung in Anlehnung an [9]

Aus Benutzersicht ist das zentrale Logikelement einer Robotersteuerung der Programmspeicher mit den Anwenderprogrammen. Letztere legen die wesentlichen Aktionen des Roboters fest, die sowohl über Bewegungs- als auch Peripherieinstruktionen zu definieren sind. Bearbeitet werden diese Programmbefehle unter anderem durch das Programmierhandgerät, welches die zentrale Schnittstelle zum Anwender ist. Darüber hinaus dient dieses, auch „Teach Pendant“ genannte, Bedieninstrument dazu, Einstellungen am System vorzunehmen oder den aktuellen Systemzustand zu überwachen.

Für die Verarbeitung der Applikationsprogramme ist die sogenannte Ablaufsteuerung zuständig. Diese interpretiert Anwenderprogramme und führt die notwendigen Aktionen auf Ebene der

Bewegungs- und Aktionssteuerung durch. Gleichzeitig müssen eingehende Signale verarbeitet werden, da diese wiederum Einfluss auf einzelne Operationen und die Ablaufreihenfolge haben können.

Unter Berücksichtigung der statischen und dynamischen Randbedingungen berechnet die Bewegungssteuerung eine sich aus den Bewegungsbefehlen ergebende Roboterbahn. Diese sogenannte Trajektorie besteht aus einzelnen, interpolierten Zwischenpunkten, die dem Antriebssystem als Sollwerte vorgegeben werden und in Motorströmen und Bewegungen resultieren. Gleichzeit werden über Resolver Ist-Positionen erfasst, die zur Lageregelung des Manipulators notwendig sind und Rückmeldung über die Position des Roboterarms geben. Über Feldbussysteme oder elektrische Schnittstellen wird sowohl mit über- als auch untergeordneter Peripherie kommuniziert. Dabei werden z. B. Aufträge angenommen oder Aktionen ausgeführt.

Software

Software bzw. Roboterprogramme stellen einen Kernbestandteil in der Funktionserfüllung einer Robotersteuerung dar. Dabei ist zwischen der System- und der Anwendersoftware zu unterscheiden. Während erstere die grundsätzliche Funktionalität der im vorangegangenen Abschnitt beschriebenen Aufgaben einer Steuerung sicherstellt, ist letztere für die Definition der Fertigungsaufgaben einer konkreten Roboterinstanz notwendig.

Diese wird über eine Roboterprogrammiersprache festgelegt, die notwendige Instruktionen zur Bewegung des Roboters, der Ansteuerung der Ein- und Ausgänge sowie der Ablauflogik enthält. Die Programmierung der Anwendersoftware kann entweder direkt am Roboter (online) oder indirekt (offline) erfolgen. Während bei Online-Verfahren in der Regel das Bedienpanel der Robotersteuerung verwendet wird, um Instruktionen vorzugeben, erfolgt die Offline-Programmierung meist über Entwicklungs- oder Simulationsumgebungen von einem PC-Arbeitsplatz aus. Die entwickelte Software wird dann nachträglich auf das Robotersystem übertragen. Eine detailliertere Ausführung der unterschiedlichen Programmiermethoden wird in Kapitel 6 aufgezeigt.

Peripherie

Roboter sind in der Regel in einem Peripherieverbund eingebunden, der sich in einen unter- und einen übergeordneten Teil differenziert und jeweils aus weiteren Automatisierungstechnik-komponenten besteht, mit denen der Roboter über Feldbuskommunikation oder elektrische Ein- und Ausgänge kommuniziert.

Die bereits erwähnten Endeffektoren stellen Beispiele für die untergeordnete Peripherie dar. Sie kann aber auch aus einfachen Sensorikkomponenten wie Lichtschranken, Initiatoren oder komplexeren Elementen wie Kamerasystemen bestehen. Diese ermöglichen es, Zustände außerhalb des Robotersystems zu erfassen und zurückzuspielen, um als Eingangsparameter im Programmablauf verarbeitet zu werden.

Unter übergeordneter Peripherie sind Komponenten zu verstehen, die dem Robotersystem in der Automatisierungspyramide hierarchisch vorangestellt sind. Meist handelt es sich dabei um eine sogenannte speicherprogrammierbare Steuerung (SPS). Diese orchestriert die Zusammenarbeit mehrerer Roboter in einem Roboterzellenverbund für einen Fertigungsauftrag und stellt die Schnittstelle zur Prozessleitebene dar. Je nach Produktionsstruktur und Steuerungskonzepten kann diese Struktur auch variieren und beispielsweise Aufgaben der Zellensteuerung vom Roboter selbst übernehmen.

2.2. Industrierobotermarkt

Das Zeitalter der industriellen Roboternutzung hat seinen Ursprung in der Patentanmeldung „Programmed Article Transfer“ [10] von George Devol und dem daraus entstandenen ersten kommerziell verfügbaren Industrieroboter „Unimate“ der Firma Unimation zu Beginn der 60er Jahre des letzten Jahrhunderts. Zeitgleich wurde der hohe Nutzen dieser neuen Technologie für den Automobilbau erkannt und die ersten roboterbasierten Automatisierungslösungen im Karosseriebau von General Motors eingesetzt. Mit einem Zeitverzug von circa zehn Jahren wurden diese auch in der deutschen Automobilindustrie, beginnend bei Mercedes Benz mit Unterstützung der Firma Kuka, in Betrieb genommen. Zunächst lediglich als Unterhändler für Unimation tätig, startete Kuka mit der Veröffentlichung des weltweit ersten elektromechanischen 6-Achs „Famulus“ im Jahr 1973 den Beginn der eigenen Industrieroboterentwicklung und -produktion. [11]

Das starke Wachstum dieses Marktes in den letzten 50 Jahren zeigt sich heute an der vorhandenen Roboterdichte und der Anzahl verkaufter Industrieroboter. Laut dem neusten World Robotics Report der International Federation of Robotics erreichte die Anzahl an weltweit in Betrieb befindlichen Industrierobotern im Jahr 2019 ein neues Hoch von über 2,7 Millionen Stück. Mit insgesamt 373.240 Einheiten wurde der dritthöchste Wert an jährlich abgesetzten Systemen seit Beginn der Datenerfassung erzielt, wobei sich die Anzahl im Vorjahresvergleich jedoch um 12 % verringerte. Der Anteil kooperativer Robotersysteme liegt dabei bei 5 % und stellt trotz hoher Wachstumsdynamik weiterhin nur einen kleinen Teil des Gesamtmarktes dar. Mit 20.473 neu installierten Robotern ist Deutschland nach China, Japan, den USA und Südkorea der fünftgrößte Abnehmermarkt von Industrierobotern. Ebenso belegt Deutschland mit einer Roboterdichte von 346 Industrierobotern auf 10.000 Beschäftigte hinter Singapur (918), Südkorea (855) und Japan (364) einen der vorderen Plätze in der globalen Betrachtung. Infolge der COVID-19-Pandemie wird im Jahr 2020 kurzfristig mit einem starken Rückgang des Roboterabsatzes gerechnet[1]. Für die Folgejahre ist mit einer Rückkehr auf den vorherigen Wachstumspfad sowie mit jährlichen Steigerungsraten im zweistelligen Prozentbereich zu rechnen. [5]

Trotz dieses potenzialträchtigen und dynamischen Industriezweiges, der zunehmend neue Technologien, Anwendungsgebiete und Wettbewerber hervorbringt, teilt sich der Großteil des Marktes auf nur einige wenige etablierte Roboterhersteller auf. Analystenschätzungen zufolge besitzen alleine die vier größten Industrieroboterunternehmen Fanuc, Yaskawa, ABB und Kuka zusammen einen Marktanteil von über 55 % [12]. Auch ist der Roboter- und Automatisierungsmarkt im Vergleich zur IT-Branche oder dem Consumer Electronics Umfeld sehr konservativ. Gründe hierfür liegen in der relativ langen Laufzeit der erworbenen Betriebsmittel (Ø 12 Jahre bei Industrierobotern [5]) und der hohen Bedeutung von Zuverlässigkeit und Fehlerfreiheit im Produktionsbetrieb [13]. Dies führt dazu, dass aufgrund des gegebenen Anforderungsspektrums sowohl bei Kunden als auch bei Anbietern häufig die sicherere und zuverlässigere Lösung der innovativen, vermutlich noch fehlerbehafteten Alternative vorgezogen wird.

Ein weiteres Merkmal der konservativen Ausrichtung der Anbieter stellt der hohe Grad an Herstellerspezifität in den unterschiedlichen Dimensionen eines Robotersystems dar. Dies führt dazu, dass der Aufwand des Wechsels eines einmal in Betrieb genommenen Systems zu einem

[1] Zum Zeitpunkt der Einreichung vorliegender Arbeit lagen noch keine veröffentlichten Absatzzahlen für das Jahr 2020 vor.

anderen Hersteller sehr hoch ist, da ein Großteil der bereits vorhandenen Implementierungs- und Integrationsarbeit nicht transferiert werden kann. Man spricht von einem sogenannten „Vendor Lock-in“ oder „User Lock-in“ Effekt [14]. So sind z. B. Roboterprogrammiersprachen herstellerübergreifend nicht kompatibel. Ein einmal entwickeltes Roboterprogramm für den Roboter eines Herstellers kann dementsprechend nicht ohne Weiteres auf dem Roboter eines anderen Herstellers verwendet werden.

Die Charakteristika des Robotermarktes manifestierten sich ebenfalls in den vergangenen Produktverbesserungen der Hersteller von Industrierobotern. Wie am Beispiel von Kuka in Abbildung 5 zu sehen ist, standen insbesondere hardware- und regelungstechnische Themen der Robotermechanik, z. B. Kostenminimierung, Wartbarkeit und Performance, im Vordergrund.

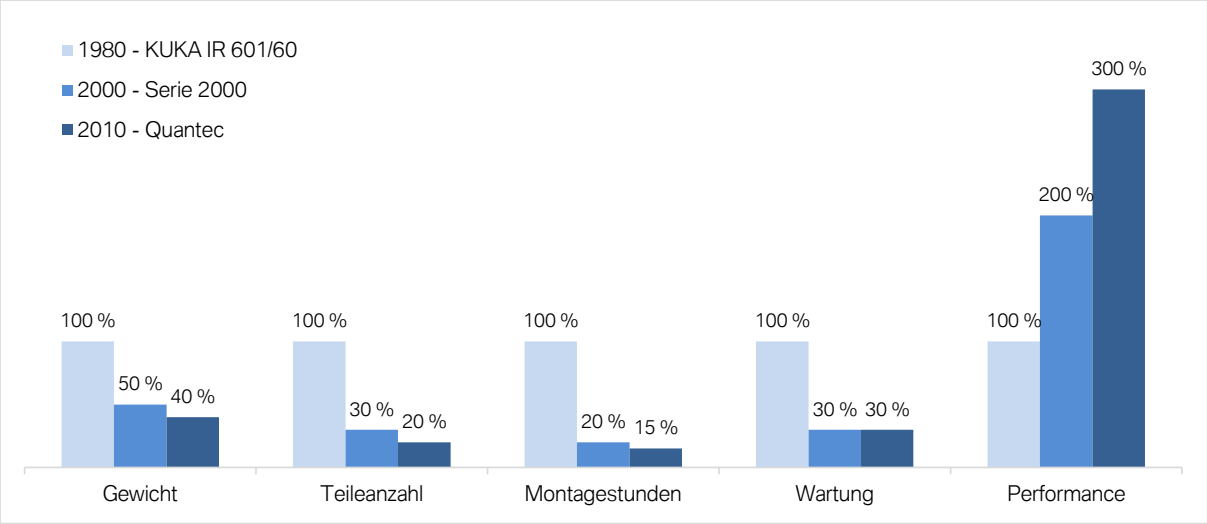


Abbildung 5 – Produktverbesserungen Kuka-Industrieroboter in Anlehnung an [15]

Der Entwicklungsfortschritt ist in Bezug auf Softwarearchitektur, Schnittstellen und Bedienungskonzepte im direkten Vergleich zur Servicerobotik oder anderen dynamischen und technologisch getriebenen Wachstumsbranchen im IT-Umfeld sehr begrenzt. So wurden zentrale Konzepte entweder nicht verändert oder nur marginal angepasst. Beispielsweise finden Programmiersprachen für Industrieroboter ihren Ursprung in Pascaldialekten, die Kommunikation mit Peripheriekomponenten oder Leitsystemen erfolgt bitweise über I/Os und die zentrale Bedienkomponente stellt weiterhin das Bedienpanel vor Ort dar. Ob Agentensysteme, Gestikprogrammierung oder Middlewaretechnologien – technologische und konzeptionelle Ansätze, Ideen und Weiterentwicklungsmöglichkeiten existieren sowohl in Industrie und Wissenschaft ausreichend. Eine umfassende Diffusion in den realen Produktionsbetrieb der klassischen Industrierobotik ist bisher ausgeblieben und wird durch die angesprochene Herstellerabhängigkeit und Proprietarität der Systeme zusätzlich erschwert, da neue Lösungen auf Anwenderseite zunächst aufwändig integriert und bestehende Applikationen nicht systemübergreifend skaliert werden können.

Im nachfolgenden Abschnitt sollen die Termini „herstellerabhängig“ und „proprietär“ daher genauer beschrieben und in Bezug auf ein Industrierobotersystem sowie die damit verbundenen Nachteile aus Anwendersicht erläutert werden.

2.3. Herstellerabhängigkeit in der Industrierobotik

Obwohl „herstellerabhängig“ bzw. „herstellerunabhängig“, insbesondere in der Softwareindustrie, häufig verwendete Begriffe sind, lassen sich weder in der gängigen deutschen noch englischsprachigen wissenschaftlichen und technischen Literatur allgemeingültige Definitionen hierfür finden. Aus rein sprachlicher Beschreibung bietet der Duden mit dem Eintrag zu *„herstellerunabhängig“*[2] und der zugehörigen Bedeutung *„unabhängig vom Hersteller [funktionierend]“* eine Aussage, dass etwas *„unabhängig“* von einem *„Hersteller“* funktioniert und somit gemäß weiterer Definition der Begrifflichkeiten als *„nicht abhängig, souverän, frei, von etwas losgelöst, nicht von etwas beeinflusst“* von einer *„Person oder Firma, die etwas industriemäßig herstellt; Produzent einer Ware“* anzusehen ist. Dementsprechend lässt sich „herstellerabhängig“ als etwas nur „abhängig vom Hersteller funktionierend“ definieren.

Mehr definitorische Klarheit verschafft der inhaltliche Kontext, in dem die „Herstellerabhängigkeit“ verwendet wird, beispielsweise im Zusammenhang mit bzw. ausgelöst durch geschlossene oder „proprietäre“ Systeme, welche wiederum ihren Wortursprung im lateinischen Wort für Eigentum, „proprietas“, besitzen. Darunter sind in der Informatik herstellergebundene[3] bzw. nicht standardisierte und herstellerspezifische Produkte oder Verfahren zu verstehen, die anderen Marktteilnehmern nicht offenstehen [16] und der Hersteller somit der alleinige Verfügungsberechtigte an dem Standard ist [17].

Komplementär dazu steht häufig der Begriff des „offenen Systems“, welches aus technischer Sicht genaueren Definitionen unterliegt, beispielsweise durch Pritschow et al., die in Anlehnung an das Standardisierungsvorhaben POSIX durch IEEE, ein offenes System als die *„Spezifikation von Schnittstellen und Diensten (Funktionen) seiner Komponenten, in diesem Fall der Applikationssoftware, sowie von Mechanismen für ihre Kombination zu Gesamtsystemen“* festlegt. Zudem muss *„Eine nach der Spezifikation ordnungsgemäß gefertigte Applikationssoftware [...] über Systemgrenzen portierbar (**Portabilität**) sein, mit anderen Applikationen zusammenarbeiten (**Interoperabilität**) und für den Benutzer in einheitlicher Form erscheinen (**Oberflächenkonsistenz**)“* [18].

In Bezug auf das in Abschnitt 2.1 definierte Robotersystem kann sich der Grad der Offenheit bzw. vice versa der Geschlossenheit, an der Portabilität, Interoperabilität und Oberflächenkonsistenz messen lassen. Dieser hat implizit Auswirkung auf die Abhängigkeit des Anwenders bzw. Kunden in Bezug zum Hersteller des Systems. Das liegt z. B. darin begründet, dass entwickelte Softwarebausteine nicht portierbar bzw. nicht wiederverwendbar sind und neu erstellt werden müssen oder Schulungen für Mitarbeiter notwendig sind, um die Anwendung neuer Bedienkonzepte (Oberflächenkonsistenz) zu erlernen. Das stellt zwar keine unüberwindbare, aber zumindest eine kostenintensive Hürde dar, erschwert den herstellerübergreifenden Wechsel und erhöht somit die Abhängigkeit von dem bereits eingesetzten und integrierten System. Da die aufgeführte Definition insbesondere die Systemgrenzen, also die Schnittstellen zu anderen technischen oder menschlichen Systemen, anspricht, sollen diese im Folgenden in Bezug auf die Ebenen der Mechanik, Elektrik, Software, Benutzer und deren Offenheit beschrieben werden.

[2] Duden Online – Begriff „herstellerunabhängig“, abgerufen am 23.05.2020.
[3] Duden Online – Begriff „proprietär“ und dessen Bedeutung, abgerufen am 23.05.2020.

Mechanische Schnittstellen

Hierunter sind Elemente mit direktem physischen Kontakt zur Systemwelt des Kunden zu verstehen. Beispiele sind Roboterfuß, Roboterflansch und Befestigungselemente der Steuerung.

Der Roboterfuß stellt das Verbindungselement zum Hallenboden dar. In der Regel erfolgt die Befestigung über eine Bodenplatte oder einen Robotersockel. Da Anzahl und Position der Befestigungselemente am Roboterfuß je nach Robotertyp und Hersteller variieren, ist ein einfacher Wechsel zu einem Roboter eines anderen Lieferanten i. d. R. nicht möglich. Bodenplatte oder Robotersockel können also nicht ohne Weiteres wiederverwendet werden. Für eine geringe Anzahl an Robotern mag dies ein zu vernachlässigender Faktor sein – bei einem Automobilwerk mit mehreren hundert Robotern sind die Aufwände für Konstruktion, Beschaffung und Einbau hingegen beträchtlich. Eine ähnliche Problematik ergibt sich bei der Robotersteuerung selbst. So sind die äußeren Befestigungselemente ebenfalls unterschiedlich aufgeführt und die Steuerungen variieren in Größe und Aufbau. Ein einfacher Ein- und Ausbau, analog zu IT-Komponenten in Server-Racks oder von Elektronikbausteinen auf Hutschienen, ist nicht möglich.

Auf mechanischer Seite zeigen sich sowohl die Vorteile als auch die in der Praxis vorhandenen Probleme einer offenen und standardisierten Schnittstelle am Beispiel des Roboterflansches. Mit der ISO 9409-1 [19] existiert zwar eine Norm, die Maße, Position und Art der Schraubverbindungen für unterschiedliche Flanschdurchmesser definiert und somit die herstellerübergreifende Kompatibilität in der Werkzeug-Roboter-Verbindung grundsätzlich gewährleisten kann. Jedoch sind in der Norm unterschiedliche Varianten aufgeführt. Außerdem bietet nicht jeder Hersteller einen normkonformen Roboterflansch an. Folglich kann das Lochbild zwischen den einzelnen Herstellern variieren und eine direkte systemübergreifende Wiederverwendung von Werkzeugen oder Werkzeugwechslern nicht immer sichergestellt werden. Der Einsatz von Adapterplatten, anderen Schraubverbindungen oder eine Modifikation des Lochbilds auf Werkzeugseite kann unter Umständen weiterhin erforderlich sein.

Elektrische Schnittstellen

Neben der proprietären Ausführung von mechanischen Elementen ist auch eine Reihe von elektrischen und mechatronischen Komponenten ein Hindernis für den einfachen Austausch von Robotersystemen. Zwar sind grundsätzliche Aspekte wie die externe elektrische Spannungsversorgung (i. d. R. 400 V) oder physische Kommunikationsschnittstellen (z. B. RJ45) infolge gängiger Industrieanforderungen und -baukästen interoperabel, allerdings sind Steckertypen und Pin- bzw. Aderbelegung unterschiedlich ausgeführt. Die elektrische Verbindung zu Werkzeugen oder externen Komponenten muss somit individuell geplant und ausgeführt werden – ein einfaches An- und Abstecken ist nicht möglich.

Während sich der Integrationsaufwand für die direkte physische Verdrahtung aufgrund von Technologien wie Feldbussystemen zunehmend reduziert, erhöht er sich für die logische Konfiguration der Anbindung – also der funktionalen Interpretation der Ein- und Ausgänge eines Robotersystems. Hier existiert kein übergreifender Standard. Sowohl die Menge und Reihenfolge als auch die Bedeutung dieser I/Os wird von jedem Hersteller individuell bzw. von der auf dem System laufenden Software bestimmt. Da diese Ebene der Anbindung ausschlaggebend für die funktionale Interaktion mit über- und untergeordneten Peripheriekomponenten ist, stellt sie ein maßgebliches Hindernis für den einfachen Austausch von Robotersystemen dar.

Softwareschnittstellen

Die zentrale Verbindung zwischen der Software eines Robotersystems und dem Anwender ist die Programmiersprache und die zugehörige Programmiersoftware bzw. Entwicklungsumgebung. Mit dieser wird die für Roboteraufgaben notwendige Prozesslogik definiert, welche insbesondere Bewegungs-, Logik-, und Schaltbefehle umfasst. Diese Anweisungen sind von essenzieller Bedeutung bei der Realisierung von Automatisierungslösungen. Ausnahmslos alle namhaften Hersteller von klassischen Industrierobotern für die Automobilproduktion verwenden eigene, proprietäre Programmiersprachen. Diese unterscheiden sich sowohl in Befehlssyntax und -semantik als auch in der Menge und Art an Funktionen. Der Programmcode eines Roboters kann auf dem System eines anderen Herstellers nicht verwendet werden. Teilweise ist sogar die Nutzung von Programmen unterschiedlicher Systemgenerationen eines Herstellers problematisch, da Befehlsdefinitionen weggefallen sind oder sich verändert haben.

Neben der Programmiersprache unterscheiden sich die zugehörigen Softwaretools zudem erheblich. Sie werden i. d. R. vom Hersteller mitgeliefert und sind für dessen Programmiersprache und Robotersysteme ausgelegt. Dabei handelt es sich meist nicht nur um reine Entwicklungsumgebungen, sondern um Anwendungen, die eine Reihe von Inbetriebnahme- und Konfigurationsmöglichkeiten für die jeweiligen Robotersysteme bieten. Eine interoperable Verwendung dieser Tools ist nicht möglich. Anwendungsentwickler müssen sowohl in Bezug auf die jeweilige Sprache als auch hinsichtlich der Programmiertools geschult werden. Ein reibungsloser Wechsel wird somit verhindert.

Bedienschnittstellen

Die Bedienung von Robotersystemen erfolgt primär mit dem Teach Pendant, auch als Programmierhandgerät (PHG) bezeichnet. Es dient dazu, Achsbewegungen manuell durchzuführen, Roboterpositionen zu speichern (das sogenannte Teachen), Roboterprogramme aufzurufen, diese geringfügig zu bearbeiten, den Systemstatus anzuzeigen oder Einstellungen zu modifizieren. Obwohl sich die Bedienfunktionen der Roboterhersteller aus funktionaler Sicht zum Großteil überschneiden, unterscheiden sich die Bedienpanels unter anderem in Menüführung, Tastenbelegung und Bedienkonzept. Ähnlich wie bei der Programmiersoftware ist auch hier ein Umstieg zwischen verschiedenen Systemen nicht ohne Weiteres möglich, sondern erfordert zumindest eine Umschulung des Personals.

Wie sich übergreifende Standardisierung positiv auswirken kann, zeigt ein Blick auf die sicherheitstechnischen Umfänge der Bedienschnittstellen. So sind beispielsweise der Not-Halt-Schalter und die Zustimmeinrichtung in mehr oder weniger ähnlicher Ausprägung herstellerübergreifend vorhanden. Ein erfahrener Anwender benötigt daher, unabhängig vom eingesetzten Hersteller, keine weitere Erläuterung der Funktionsweise.

Die vorangegangenen Punkte verdeutlichen, in welch unterschiedlichen Dimensionen sich die Geschlossenheit und geringe Interoperabilität der industriellen Robotersysteme zeigt und welche Auswirkungen diese Faktoren in der industriellen Praxis haben - insbesondere für die in vorliegender Arbeit relevanten Aspekte der Software- und Bedienschnittstellen. Es sei angemerkt, dass die Ursachen hierfür nicht allein auf der Seite der Hersteller und deren nachvollziehbarem Interesse an Wettbewerbsdifferenzierung zu finden sind, sondern auch am mangelnden Interesse an Standardisierungsvorhaben auf Abnehmerseite oder an der Komplexität von Schnittstellen liegen können. So besitzt beispielsweise die deutsche Automobilindustrie mit der AIDA (Automatisierungsinitiative Deutscher Automobilhersteller) zwar ein Gremium, das Vereinheitlichung und Standardisierung in der Automatisierungstechnik

vorantreibt und in Bezug auf die Robotik auch erfolgreiche hardwarenahe Umsetzungen, wie z. B. ProfiNet-Standard und Ethernet-Stecker [20, 21], vorzuweisen hat, jedoch auf Ebene der Softwarelogik und -schnittstellen keinen nennenswerten Standardisierungsfortschritt verzeichnet.

Ebenso existieren, wie im nachfolgenden Kapitel beschrieben, auch im Forschungsbereich Ansätze und Methoden, um den Effekt der Herstellerabhängigkeit durch offene Systemgestaltung zu mindern. Diese konnten sich allerdings bisher nicht in der industriellen Praxis etablieren bzw. sind nicht für das zu untersuchende Anwendungsfeld des Karosseriebaus konzipiert.

2.4. Lösungsansätze in Forschung und Industrie

Bereits seit über 30 Jahren beschäftigen sich Wissenschaft und Industrie direkt oder implizit mit dem Thema der herstellerunabhängigen Verwendung von Automatisierungstechnik und Robotern. Dies geschieht häufig im Zusammenhang mit der Gestaltung offener, standardisierter, hardwareunabhängiger, modularer oder herstellerübergreifender Systeme und Lösungen, welche in unterschiedlicher Form in Dissertationen, Forschungsprojekten oder Normungsgremien behandelt und diskutiert wurden – oft als Kern der eigentlichen Arbeit, aber auch als Vorteil oder Notwendigkeit, die eine technologische Neuerung mit sich bringt.

Die bedeutendsten Arbeiten der Vergangenheit in Bezug auf herstellerunabhängige und offene Robotersysteme werden in Abbildung 6 chronologisch dargestellt und im Folgenden detaillierter erläutert.

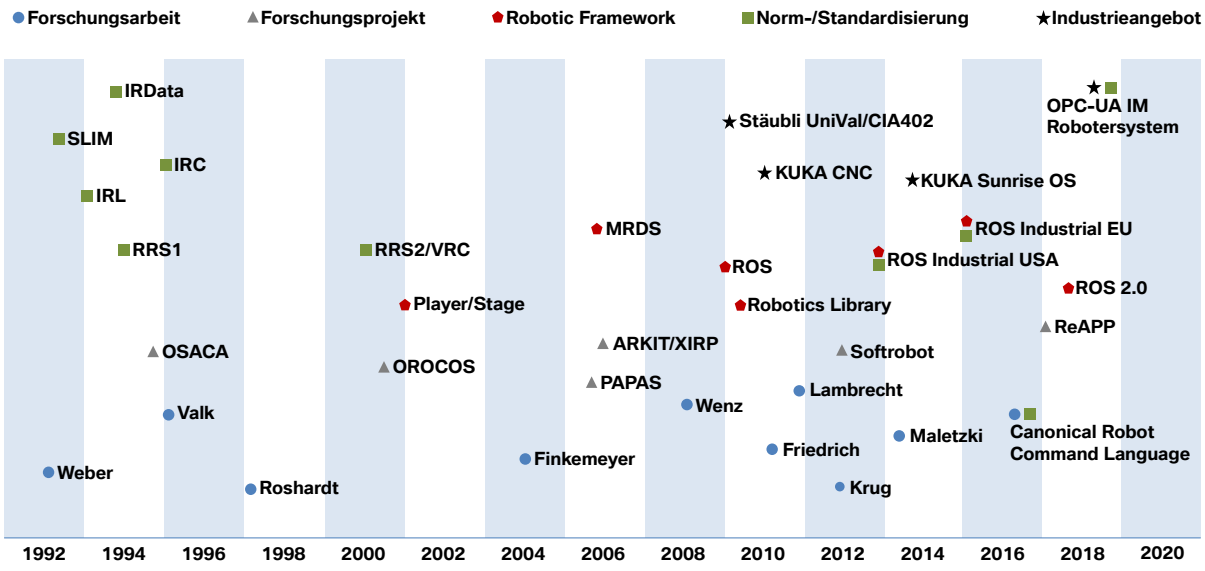


Abbildung 6 – Beiträge zur herstellerunabhängigen Robotik im Zeitverlauf[4]

Normierungsvorhaben und Forschungsarbeiten vor der Jahrtausendwende

Der Zeitraum vor der Jahrtausendwende ist insbesondere von Standardisierungsvorhaben im Bereich der industriellen Roboter- und NC-Maschinensteuerung und der Entwicklung PC-basierter Robotersteuerungsaufbauten gekennzeichnet.

IRDATA und ICR

Einer der ersten Versuche, die Ansteuerung von Robotersteuerungen zu standardisieren, wurde im Rahmen der Arbeit des VDI zur „Programmierung numerisch gesteuerter Handhabungseinrichtungen“ unternommen und 1987 als technische Regel [22] veröffentlicht. In den darauffolgenden Jahren wurden die Ergebnisse geringfügig modifiziert und 1994 in einen Entwurf bzw. 1997 in die finale Norm DIN 66314-1 [23] mit dem Titel „Schnittstelle zwischen Programmierung und Robotersteuerung – IRDATA“ überführt. Mit der Spezifikation einer Zwischensprache wird eine Schnittstelle definiert, die den Datenverkehr zwischen einem externen, steuerungsunabhängigen Programmiersystem und einer Industrierobotersteuerung beschreibt [23]. Die Verbindung zwischen der Programmierung und der Programmausführung des Roboters durch die Robotersteuerung soll so getrennt und austauschbar gestaltet werden.

[4] Einordnung anhand des Zeitpunktes der zugehörigen ersten Veröffentlichung(en), dem Abschluss der Forschungsprojektes oder der Veröffentlichung von Quellcode.

Technisch betrachtet stellt IRDATA eine Definition von numerisch codierten Befehlssätzen dar, die zur Ausführung und Definition von unterschiedlichen Roboterfunktionen und -parametern wie Bewegungen, Arbeitsbereiche oder Logikoperationen dienen. Anforderungsseitig muss das Programmiersystem in der Lage sein, IRDATA-Code auszugeben und die Robotersteuerung einen IRDATA-Interpreter besitzen, um die empfangenen Anweisungen auszuführen.

Gleiche Ziele wie IRDATA besitzt der 1995 veröffentliche ISO-Spezifikations-Entwurf ISO/TR 10562:1995 – „Manipulating industrial robots - Intermediate Code for Robots (ICR)“ [24]. Analog zu IRDATA sollte mit einem Zwischencode eine standardisierte neutrale Schnittstelle zwischen Programmier- und Steuerungssystem geschaffen werden. Im Gegensatz zur deutschen Norm definiert ICR Anweisungen nicht nur in numerischen Befehlen, sondern ebenfalls in für Menschen leicht lesbare Mnemonics. Obwohl die Arbeit an IRDATA und ICR laut [23] in Abstimmung erfolgt ist, sind beide Spezifikationen nicht vollständig kompatibel, da sich die Satznummern geringfügig voneinander unterscheiden.

IRL und SLIM

Ein weiteres deutsches Normierungsvorhaben ist die durch den Normenausschuss Maschinenbau erarbeitete und im Jahr 1993 als DIN 66312-1 [25] veröffentlichte Industrial Robot Language (IRL). Diese Norm definiert Syntax und Semantik einer an PASCAL angelehnten, einheitlichen höheren Programmiersprache für Industrieroboter. Sie soll es Anwendern ermöglichen, Bewegungen und logische Programmflüsse unabhängig von speziellen Industrierobotern und -steuerungen zu programmieren [26]. Die Spezifikation umfasst eine Vielzahl an Anweisungen, die übliche Roboterprogrammierbefehle wie Programmdeklarationen oder Bewegungs-, Ein- und Ausgangs- sowie Logikoperationen universell definieren. Eine genauere Betrachtung des Funktionsumfangs erfolgt in Kapitel 6.

Das asiatische Pendant zur IRL ist die im japanischen Industrie Standard JS B 8439 [27] spezifizierte Programmiersprache SLIM (Standard Language for Industrial Manipulators) aus dem Jahr 1992. Bei SLIM werden roboterspezifische Befehle mit dem Fokus auf Montageanwendungen definiert. Im Gegensatz zur IRL dient BASIC als Ausgangssprache und der spezifizierte Funktionsumfang fällt im direkten Vergleich geringer aus. Allerdings hat die Norm heute noch Bestand, während die DIN 66132-1 ebenso wie die Normen zu IRDATA und ICR offiziell zurückgezogen wurden. Anwendung findet SLIM als Basis für die Programmiersprachen der japanischen Roboterhersteller Denso und Nachi, welche jedoch herstellerspezifische Erweiterungen und Anpassungen in ihren proprietären Programmiersprachen vorgenommen haben und eine übergreifende Kompatibilität so verhindern.

RRS

Während sich die vorangegangenen Normierungen vor allem auf die Programmierung und Steuerung des realen Roboters bezogen haben, befasste sich das von 1992 bis 1994 von der Automobilbranche initiierte Projekt „Realistic Robot Simulation“ (RRS) mit Standardisierungsschwerpunkten im Simulationsumfeld von Roboterapplikationen. Mit zunehmender Verbreitung von Computer-Aided-Robotics Methoden im Produktionsplanungsumfeld stieg damals der Bedarf, das reale Steuerungsverhalten von Industrierobotern möglichst realitätsnah zu simulieren. Ausgangslage war, dass Anbieter von Simulationssoftware zwar die Möglichkeit hatten, Roboterkinematiken in ihren Tools mechanisch realitätsgetreu abzubilden, das exakte Bewegungsverhalten hingegen nicht reproduzierbar war, da diese von den Steuerungsalgorithmen der jeweiligen Roboterhersteller abhängig und nicht einsehbar sind. Aus diesem

Grund wurde mit dem RRS-Interface ein Standard [28] erschaffen, der es ermöglicht, reale SW-Komponenten der Robotersteuerung eines Herstellers in die Entwicklungsumgebung eines Simulationssoftwareanbieters zu integrieren.

Der RRS-Standard spezifiziert ein prozedurales Interface, das sowohl den Datenaustausch als auch die Ausführung der Steuerungs-SW-Komponenten für die Bahnplanung definiert. Die Algorithmen werden als Blackbox in einem sogenannten RCS-Modul zur Verfügung gestellt, das von der Simulationssoftware über definierte C-Befehle angesprochen und ausgeführt werden kann. Dadurch soll sichergestellt werden, dass das Know-how des Roboterherstellers verborgen bleibt, aber die notwendige Portabilität für Simulationslieferanten und Anwender gegeben ist. Im Gegensatz zu den vorherigen Standardisierungsvorhaben konnte sich RRS als Industriestandard etablieren und stellt auch heute noch den technischen Status quo dar.

RRS 2

Während sich RRS lediglich auf die Simulationseinbindung von Bewegungsmodulen fokussiert, wurde im Jahr 1998 aufgrund der erfolgreichen Umsetzung das Folgeprojekt RRS 2 gestartet. Ziel war es, die Robotersteuerung noch umfassender in die Simulationswelt zu integrieren und die Anbindung mit der realen Produktionsumgebung zu verbessern, indem Anwenderprogramme ohne weitere Modifikation auf realen Anlagen ausgeführt werden können [29]. Aus diesem Grund wurden die Projektpartner um Anlagenbauer und Kalibriertechniker erweitert. Das Ergebnis des Projektes war die Spezifikation des Virtual Robot Controller (VRC). Im Gegensatz zum RRS-Interface wurden zusätzliche Schnittstellen bereitgestellt, die Aspekte wie Bedieninterface, Dateisystem, I/O-Handhabung, Handhabung von Simulationsinstanzen oder Programmcodegenerierung beinhalteten [30]. Dieser Standardisierungsvorschlag konnte sich indes nicht etablieren.

OSACA

Während man sich in den zuvor beschriebenen Vorhaben auf die Definition von Schnittstellen und die Domäne Robotik konzentrierte, war der Anwendungsbereich des von 1992 bis 1995 laufenden Projektes OSACA (Open System Architecture for Controls within Automation Systems) weiter gefasst. Ziel des mit Forschungsinstituten und Werkzeugmaschinenherstellern besetzten Projektes war die Festlegung eine Softwarearchitektur für herstellerübergreifende offene Steuerungssysteme der Automatisierungstechnik.

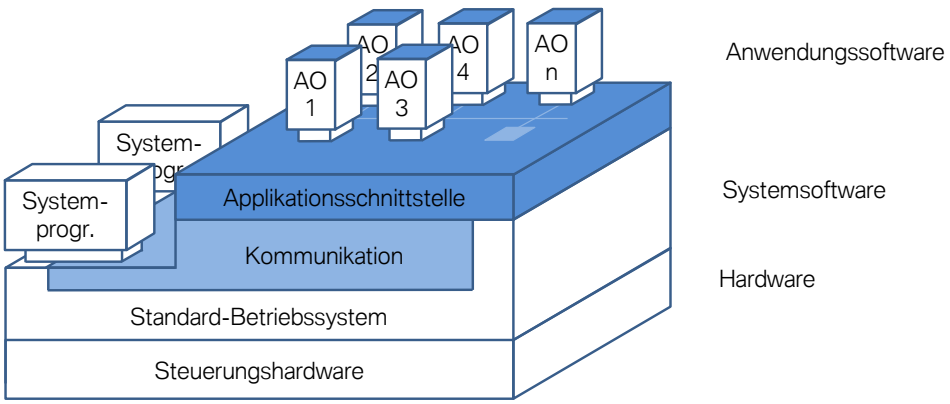


Abbildung 7 – OSACA-Steuerungsplattform und Anwendungssoftware in Anlehnung an [18]

Die entwickelte OSACA-Steuerungsplattform soll als „Bindeglied zwischen Software für steuerungstechnische Funktionen (Applikationssoftware) und der Hardware fungieren“ [18]. Auf diese Weise wird es Herstellern ermöglicht, sich auf die eigentliche Steuerungssoftware zu konzentrieren und sich nur in geringem Maße mit den notwendigen Infrastrukturthemen befassen zu müssen. Zudem kann eine „einmal erschaffene Software mit wenig Aufwand auf Steuerungsplattformen unterschiedlicher Hersteller portiert und eingesetzt werden“ [18].

Anhand der definierten Referenzarchitektur und der zugehörigen einheitlichen API können relevante Softwarekomponenten der Anwendungssoftware einer Steuerung als sogenannte Architekturobjekte (AO) unabhängig von der Zielplattform entwickelt werden. Zusätzlich können diese einzelne AOs auf Basis der zur Verfügung gestellten Kommunikationsinfrastruktur zu einem modularen Gesamtsystem konfiguriert werden (Abbildung 7). Auf diese Weise wird die Komplexität der verwendeten Hardware und der Kommunikation vor dem Programmierer verborgen und die Softwareelemente werden modular, plattformunabhängig und austauschbar gestaltet. Ein Prinzip, das ebenfalls in klassischen General Purpose-Betriebssystemen[5] und Middleware-Technologien zum Tragen kommt.

Weber, Valk, Roshardt

Neben den dargestellten Projekten befassten sich in den 90er Jahren auch eine Reihe von Dissertationen mit der Aufgabe, Robotersysteme unabhängiger von der verwendeten Hardware oder der zu verwendenden Programmierschnittstelle zu gestalten.

Weber präsentierte 1992 in seiner Arbeit [31] eine hardwareunabhängige Robotersteuerung mit offenen Programmierschnittstellen. Die auf einem Echtzeitbetriebssystem basierende Steuerung ist mit modularen, in Hochsprachen konzipierten Softwarekomponenten umgesetzt und bildet die wesentlichen Funktionen einer industriellen Steuerung ab. Dank der gewählten Architektur ist diese Steuerungssoftware weitestgehend plattformunabhängig und mit geringen Aufwänden auch auf andere Roboter und Steuerungsrechner-Hardware portierbar. Zudem bietet ein IRDATA-Interpreter die Möglichkeit der Ausführung von Roboterprogrammen über eine herstellerübergreifende Schnittstelle an.

Die Herausforderung der Hardwareunabhängigkeit ging van der Valk mit der Konzeption und Realisierung einer Low-Cost Robotersteuerung auf Basis von PC-Standardkomponenten an [32]. Als Software wurde das um Echtzeiteigenschaften erweiterte Standardbetriebssystem MS-DOS gewählt, wobei die Lageregelung in separaten Achsregelbaugruppen ausgeführt wird. Mit IRDATA und IRL bietet die Steuerung ebenfalls eine offene Programmierschnittstelle. Die prototypische Realisierung und Validierung wurde mit Robotern kleinerer Nutzlast (Manutec r2, Mitsubishi RV-M1) durchgeführt.

Roshardt entwickelte in seiner Arbeit [33] ebenfalls eine PC-basierte Robotersteuerung. Anwendungsfokus bestand darin, die Anforderungen einer automatisierten Kleinserien- und Einzelfertigung zu erfüllen, welche aus seiner Sicht zukünftig die zentrale Rolle in der Robotertechnik spielen sollten, aber aufgrund mangelnder Flexibilität, Störanfälligkeit und hohen

[5] Mit General Purpose werden Technologien bezeichnet, die einen universellen und breiten statt einen engen und spezifischen Anwendungszweck haben, z.B. ein Windows-Betriebssystem anstatt eines spezifischen Roboterbetriebssystems oder z.B. eine Standardprogrammiersprache wie Java anstatt einer spezifischen Programmiersprache wie Kuka KRL.

Programmieraufwänden der damals aktuellen Systeme nicht wirtschaftlich umsetzbar waren. Zentrale Ansätze zur Lösung dieser Herausforderungen sah er neben neuen Mensch-Maschine-Interaktionsmethoden in der vereinfachten Sensorintegration, besserer Sensordaten-verarbeitung und der Nutzung des technologischen Fortschritts in der Informationsverarbeitung. Aus diesem Grund präsentierte er eine offene Steuerung, die in einer modularen und objektorientierten SW-Architektur mit klaren HW-SW-Schnittstellen umgesetzt wurde und ein „hardwareunabhängiges Peripheriesystem ermöglicht". Dadurch können Anwenderprogramme von der Sensorik getrennt und Komponenten zur Laufzeit ausgetauscht werden, ohne die Steuerung modifizieren zu müssen.

Beiträge zur herstellerunabhängigen Robotik neuerer Zeit

Während der Zeitraum vor der Jahrtausendwende stark durch die Standardisierungs- und Normierungsversuche in Gremienarbeit geprägt war, spielten in den darauffolgenden Jahren insbesondere offene Frameworks und Middlewareplattformen eine bedeutende Rolle in der Standardisierungsarbeit und der hardwareunabhängigen Entwicklung von Robotersystemen und -anwendungen.

Player/Stage

Gestartet als ein Softwareprojekt, um Bedürfnisse hinsichtlich Schnittstellen- und Simulationsanforderungen für Multi-Robot-Systeme zu erfüllen, hat sich das Player/Stage-Projekt der University of Southern California zu einer der ersten offenen und weltweit verbreiteten Softwareplattformen für Robotik etabliert. Motivation war, Forscher von der ermüdenden, aber notwendigen Infrastrukturimplementierung zu entlasten, grundlegende Algorithmen über standardisierte Schnittstellen zur Verfügung zu stellen und die Entwicklung schneller, einfacher und effizienter zu gestalten [34–36]. Analog zu Betriebssystemen in der PC-Technik wurde dabei die Grundphilosophie verfolgt, eine Standardplattform in der Robotik zu erschaffen, welche die Komplexität der Hard- und Software vor dem Entwickler verbirgt und durch generische Tools eine einheitliche Entwicklungsumgebung für unterschiedliche Systeme bietet.

In Player wird dies durch die Abstraktion von Aktorik und Sensorik mit sogenannten Device-Models realisiert, die über TCP[6]-Sockets einheitliche Schnittstellen zur Verfügung stellen und von Clients angesprochen werden können. Auf diese Weise wird die Systemarchitektur modular gehalten und Clients können in unterschiedlichen Programmiersprachen implementiert werden. Zudem ermöglicht die Socket-Kommunikation die Realisierung von verteilten Systemen, d. h. Client und Device müssen nicht zwangsläufig auf einem System laufen, was insbesondere für Multi-Robot-Systeme einen Mehrwert darstellt. Die Player-Standard-Schnittstellen werden ebenfalls von der zugehörigen Stage-Simulationsumgebung genutzt und transferieren die Vorteile der Plattform- und Programmiersprachenunabhängigkeit auch in die Robotersimulations- und Modellerstellung.

OROCOS

Eine ähnliche Intention wie das Player/Stage-Projekt verfolgte das Open Robot Control Software-Projekt (OROCOS). Auch hier war es Ziel, ein General Purpose-Softwaresystem in einem Open-Source-Ansatz zu entwickeln, jedoch mit dem Fokus, Steuerungs- und Regelungsaspekte von Robotersystemen zu vereinfachen, wiederverwendbarer zu gestalten

[6] Transmission Control Protocol

und neue Methoden der Softwareentwicklung zu integrieren. Aus Sicht von Bruyninckx ist es der Roboterindustrie in der Vergangenheit zwar gelungen, neue Technologien und Standards in Bezug auf Elektronik und Mechanik zu integrieren, aber seitens der Softwarearchitektur auf einem Stand der Informationstechnologie von vor 20 Jahren stehen geblieben zu sein [14]. Insbesondere die proprietäre Ausprägung von Algorithmen, Datenstruktur und Programmiersprachen verhindert die Möglichkeit, Software plattformübergreifend auszutauschen oder wiederzuverwenden. Dies führt zu einem klassischen Vendor Lock-in und hohen Wechselkosten. Sämtliche Standardisierungsvorhaben der Vergangenheit sieht Bruyninckx als gescheitert. Als Ursache hierfür wird insbesondere die Charakteristik des Industrierobotermarktes angeführt. Zum einen handelt es sich um einen Nischenmarkt, der zwar in Volumen und Stückzahl eine hohe Ausprägung besitzt, allerdings aus Anwendungssicht bisher nur ein geringes Spektrum des Produktionsumfelds abdeckt. Zum anderen sind die Hauptabnehmer Industrieunternehmen, die vor allem Langzeitinvestitionen mit relativ statischen Anforderungen tätigen und insbesondere Wert auf Zuverlässigkeit und Robustheit legen. Kunden mit dynamischen, schnell wechselnden Anforderungen und Anwendungen (z. B. Forschungsinstitute), die von offenen Plattformen und austauschbarer sowie wiederverwendbarer Software besonders profitieren würden, stellen lediglich eine Minderheit dar. Daher ergibt sich seitens der Hersteller keine Notwendigkeit, Robotersysteme diesbezüglich zu optimieren [14].

Um diesen Anforderungszweck zu erfüllen, stellt Orocos eine C++-Softwarebibliothek aus vier entkoppelten Teilsystemen zur Verfügung, die es ermöglichen sollen, Robotersteuerungen plattform- bzw. roboterunabhängig zu entwickeln. Dazu gehören eine auf CORBA basierende, echtzeitfähige Middleware („Real-Time Toolkit“ (RTT)), Algorithmen zur Steuerung und Regelung („Orocos Component Library“), diverse stochastische Filtermethoden (The Bayesian Filtering Library) und Werkzeuge zur Berechnung von dynamischen und kinematischen Aspekten (Kinematics and Dynamics Library (KDL)) [37, 38].

MRDS

Während die bisher erwähnten Softwarebibliotheken aus dem Forschungsumfeld mit wissenschaftlichem Anforderungsschwerpunkt initiiert wurden, ist im Jahr 2006 mit dem Microsoft Robotics Developer Studio[7] erstmals eine roboterunabhängige Softwareplattform von einem großen IT-Unternehmen veröffentlich worden. Analog zum Orocos- und Player/Stage-Projekt wurden Hardwareabhängigkeit und mangelnde Austauschbarkeit bzw. fehlende Wiederverwendbarkeit von Robotersoftware als Hindernisse identifiziert, die den Fortschritt, insbesondere in der Servicerobotik, verzögern. Analog zur eigenen Firmenhistorie und der PC-Industrie in den 80er Jahren sah Microsoft den Bedarf für eine einheitliche und plattformunabhängige Softwareumgebung gegeben, die es ermöglicht, einmal entwickelte Software auf unterschiedlichen Systemen betreiben zu können. Dadurch sollten Unternehmen und Entwickler in die Lage versetzt werden, sich auf die für sie wertschöpfenden und differenzierenden Themen, wie Algorithmik und Verhalten der Roboter, zu fokussieren, statt ihre Ressourcen für Infrastrukturthemen zu verwenden [39, 40].

Die Kerntechnologien des auf Windows basierenden MRDS stellen eine Bibliothek & Runtime für verteilte Systeme, eine graphische Programmiersprache und eine Simulationsumgebung dar. Dadurch kann modulare und verteilte Software in Form von Services entwickelt und über

[7] Zunächst als Microsoft Robotics Studio veröffentlicht und im Jahr 2008 in Microsoft Robotics Developer Studio umbenannt.

Kommunikationstechnologien aus dem Web-Service Bereich (SOAP, HTTP, REST) zu einem flexiblen, robusten System verbunden werden. Zudem besteht mit der Visual Programming Language die Möglichkeit, Programme sowohl mit klassischen textuellen Programmiersprachen wie C#, VBA oder C++ als auch graphisch zu entwickeln. Im Gegensatz zu den bisher beschriebenen Frameworks ist MRDS nicht als Open-Source-Software veröffentlicht worden, sondern als frei verfügbare Entwicklungsplattform, für die bei kommerzieller Anwendung eine Lizenzgebühr erhoben wird.

ROS

Ein weiteres Open-Source-Framework, welches initial von einem Unternehmen entwickelt und veröffentlicht wurde, stellt das Robot Operating System (ROS) dar [41]. Motivation war hierbei ebenfalls, mit Hilfe eines einheitlichen Softwareframeworks und standardisierten Schnittstellen, die Entwicklung von Robotersystemen sowie die plattformübergreifende Wiederverwendung von Software zu vereinfachen und zu beschleunigen. Kern von ROS stellt eine eigene Middleware dar, die TCP- und UDP-basierte synchrone und asynchrone Kommunikation zwischen unterschiedlichen Modulen, sogenannten ROS-Nodes, ermöglicht. Hierzu ist eine Reihe von Standard-Schnittstellen in Form von ROS-Messages festgelegt, um Modularität und Austauschbarkeit im System zu gewährleisten. Darauf aufbauend bietet ROS eine Vielzahl von diversen Tools und Libraries an, die den Anwender bzw. Entwickler bei unterschiedlichsten Aufgaben, beispielsweise bei der Kinematisierung oder Simulation und Visualisierung eines Robotersystems, unterstützen.

Ende 2017 wurde neben der beschriebenen ersten Version von ROS parallel eine zweite Version ROS 2.0 eingeführt, die technologische Schwachpunkte des Vorgängers verbessert sowie beispielsweise Echtzeitfähigkeit unterstützt und die eigene Middlewareimplementierung durch den Data Distribution Standard (DDS) der Object Management Group ersetzt [42].

Interessant sind bei ROS neben den technischen Eigenschaften auch die wirtschaftlichen Randbedingungen und die weite Verbreitung in Forschung und Wissenschaft. Ursprünglich entwickelt und veröffentlicht wurde ROS im Rahmen der Serviceroboter-Entwicklung des Start-Ups Willow Garage im Jahr 2009. Eigentümer und Investor war Scott Hassan, einer der ersten Google-Architekten und ein in der Technologie-Szene bekannter Milliardär, der die autonome Robotik mit seiner Investition in ROS und dem PR2-Roboter im privaten Bereich revolutionieren wollte [43, 44]. Jedoch hat Willow Garage auf Dauer kein funktionierendes Geschäftsmodell etablieren können und wurde im Jahr 2013 liquidiert, wobei die Wartung und Weiterentwicklung von ROS an das Non-Profit-Unternehmen Open Source Robotics Foundation übertragen wurde [45, 46]. Mit der ROSCon existiert zudem weiterhin eine jährlich stattfindende Konferenz, die speziell auf Entwickler und Anwender von ROS ausgerichtet ist.

Im Gegensatz zu den bisher beschriebenen Arbeiten und Projekten, konnte sich ROS als einziges Framework im wissenschaftlichen Bereich als eine Art De-facto-Standard etablieren und findet sich, insbesondere in der mobilen und der Servicerobotik, auch in Serienprodukten wieder [47]. Bei der BMW AG wird ROS sowohl als Prototyping Plattform für die Serienentwicklung von hochautomatisierten Fahrfunktionen bei PKW [48, 49] als auch in der Entwicklung von mobilen Robotern des Produktionssektors verwendet [50].

Bezogen auf die klassische industrielle Automatisierungstechnik und industrielle Robotik ist das Konsortium ROS Industrial zu erwähnen. In diesem bemühen sich seit 2013 Forschungsinstitute sowie Industrie- und IT-Unternehmen, ROS für industrielle Anwendungen zu befähigen und

weiterzuentwickeln [51, 52]. So werden durch das Konsortium beispielsweise Hardwaretreiber für Robotersysteme und Aktoren bereitgestellt, auf Industrieanwender ausgelegte ROS-Schulungen durchgeführt und die Implementierung von industriellen Anwendungen mit ROS vorangetrieben.

Robotics Library

Während ROS eine Kombination aus Middleware und einer Vielzahl von unterschiedlichen Roboterbibliotheken darstellt, verfolgt Rickert eine fokussiertere Vorgehensweise in Form einer einheitlichen C++ Bibliothek. Die Entwicklungsarbeit zur Robotics Library (RL) wurde im Jahr 2004 aufgrund mangelnder Verfügbarkeit von leistungsfähigen Bahnplanen initiiert und fand anschließend in einer Reihe von Industrie- und Forschungsprojekten Anwendung, bevor sie im Jahr 2009 als Open-Source-Bibliothek auch der Allgemeinheit zur Verfügung gestellt wurde. Ziel der RL ist, die funktionalen Kernanforderungen mit einem objektorientierten Ansatz und einer klaren Schnittstellendefinition für die Entwicklung von Roboteranwendungen von Hardwareabstraktion über Visualisierung bis zur Bahnplanung vollständig abzudecken. Zudem wurde in der Implementierung auf ein hohes Maß an Wiederverwendbarkeit und Plattformunabhängigkeit geachtet [53].

Die RL kann daher im Gegensatz zu ROS oder MRDS auch auf echtzeitfähigen und sowohl auf Linux- als auch auf Windows-Systemen ohne Einschränkung betrieben werden, wohingegen die genannten Technologien primär für den Einsatz auf Linux (ROS) oder Windows-Plattformen (MRDS) konzipiert wurden. Die klaren und fokussierten Designprinzipien gehen allerdings mit dem bewussten Verzicht auf eine Middlewaretechnologie einher. Für den Einsatz auf verteilten Systemen bedeutet dies, dass der Anwender die Kommunikationstechnologie frei wählen kann, diese jedoch C++ Unterstützung aufweisen und die Implementierung eigenständig durchgeführt werden muss. In Bezug auf die unterstützte Roboterhardware konzentriert sich die RL vor allem auf statische Manipulatorkinematiken mit unterschiedlichstem Aufbau.

Weitere Roboterframeworks

Neben den beschriebenen Roboterframeworks sind im wissenschaftlichen Umfeld noch eine Vielzahl weiterer Frameworks und Libraries (u. a. MIRA, CARMEN, ORCA, YARP) entstanden, die Aspekte der plattform-, hardware- oder herstellerunabhängigen Integration von unterschiedlichen Robotersystemen behandeln. Diese sollen in vorliegender Arbeit nicht vertieft werden. Es wird auf Arbeiten von Elkady et al. [54] oder Zug et al. [55] verwiesen, die einen übergreifenden und detaillierten Vergleich der verschiedenen Lösungen durchführen. Zusammenfassend lässt sich festhalten, dass die analysierten Frameworks meist das gleiche Ziel besitzen, indem sie die Wiederverwendbarkeit von Robotersoftware durch Abstraktion von Hard- und Softwarebestandteilen erhöhen und den Aufwand für Infrastrukturaufgaben reduzieren sollen. Technisch wird dies i. d. R. mit universell definierten Schnittstellen von Softwarekomponenten und über Kommunikationsmechanismen in Form von Middlewaretechnologien realisiert.

Forschungsprojekte – PAPAS, SMERobots, SMERobotics, ReApp, ARKIT

Während sich die beschriebenen Roboterframeworks vor allem auf die Entwicklung von Robotersystemen konzentrieren, beschäftigte sich eine Reihe von Forschungsprojekten insbesondere mit der einfacheren Inbetriebnahme von Robotern im Produktionsumfeld. Fokus stellte häufig die Rekonfiguration von Peripheriekomponenten oder die einfache Programmierung von Robotersystemen dar. Die Hersteller- bzw. Hardwareunabhängigkeit bezieht sich in diesem Fall meist auf einzubindende Komponenten bzw. den zugehörigen Konfigurationsprozess oder die Programmier- und Bedienmethode.

Zu erwähnen sind hierbei die öffentlich geförderten Projekte PAPAS, SMERobots, ReAPP oder das im Projekt ARKIT entstandene XIRP-Protokoll. Ein Schwerpunkt in dem von 2003 bis 2006 laufenden Verbundprojekt PAPAS war, die Konfiguration und die Verwendung von Peripheriekomponenten für Leichtbauroboter mittels an USB-Bus-Technologie angelehnten Plug & Play-Konzepten zu vereinfachen. Zu diesem Zweck wurde zwischen Feldgeräten und Robotersystem ein offenes und standardisiertes Protokoll festgelegt, das unabhängig vom Feldbussystem und der Geräte- bzw. Systemsoftware ist und bestimmte Geräteklassen (z. B. Kraft-Momenten-Sensor oder Greifer) definiert. PAPAS-konforme Peripheriekomponenten und Robotersysteme sollen so automatisch und ohne manuellen Aufwand konfiguriert werden können [56].

Während das PAPAS-Protokoll auf herkömmliche industrielle Feldbussysteme aufbaut, verwendet das im Rahmen des ARKIT entstandene Protokoll XIRP eine konventionelle TCP/IP-Kommunikation und ein XML-Datenformat, um für die Maschinen-Maschinen-Schnittstelle einen Protokollstack zwischen komplexer Sensorik und dem Steuerungssystem festzulegen [57]. Ziel ist ebenfalls, den wiederkehrenden Aufwand für Peripherieintegration, insbesondere komplexe Sensorik, zu vereinfachen. Die XIRP-Spezifikation wurde nach dem Projekt im VDMA-Einheitsblatt 66430-1 [58] festgehalten und legt Mechanismen fest, wie der Verbindungsaufbau und Befehlsaustausch geregelt ist.

Das Projekt ReApp ging hingegen nicht den Weg der Definition eines eigenen Nachrichtenprotokolls, sondern entschloss sich, aufgrund der weiten Verwendung auf dem Framework ROS und dessen Middlewarespezifikation aufzubauen, um die herstellerunabhängige Wiederverwendung von Software für Roboterapplikationen zu ermöglichen. Hierzu wurde eine prototypische Integrationsplattform entwickelt, mit der über einen App-Store bereitgestellte Softwaremodule und Hardware-Treiber heruntergeladen und für Roboterapplikationen verwendet werden können. Diese Roboter-Apps werden über eine einheitliche Taxonomie und Ontologie in ihrer Funktionalität beschrieben und kategorisiert. Auf diese Weise soll die Austausch-, Vergleich- und Verwaltbarkeit innerhalb des App-Stores und des Steuerungssystems gewährleistet sein. Zudem bietet eine Reihe von Tools, die einfache, modellbasierte Erstellung von neuen Applikationen und die Simulation von Roboteranwendungen über eine Cloud-Simulationsanbindung an. Sowohl als Anforderungsgeber als auch Validierungsinstrument wurden innerhalb des Projektes industrielle Anwendungsfälle umgesetzt. Die BMW AG beteiligte sich in dem Projekt mit Entwicklungsarbeit und der Bereitstellung eines Anwendungsfalls aus der Türenvormontage. Ein Montageprozess für das Anrollen von Türdichtungen, der bereits sowohl in einer Standard-Industrieroboter-Lösung als auch in einer vollautomatisierten Mensch-Roboter-Kollaboration-Anwendung existierte, wurde im Rahmen von ReApp zusätzlich mit der auf ROS-basierenden Integrationsplattform umgesetzt [59, 60].

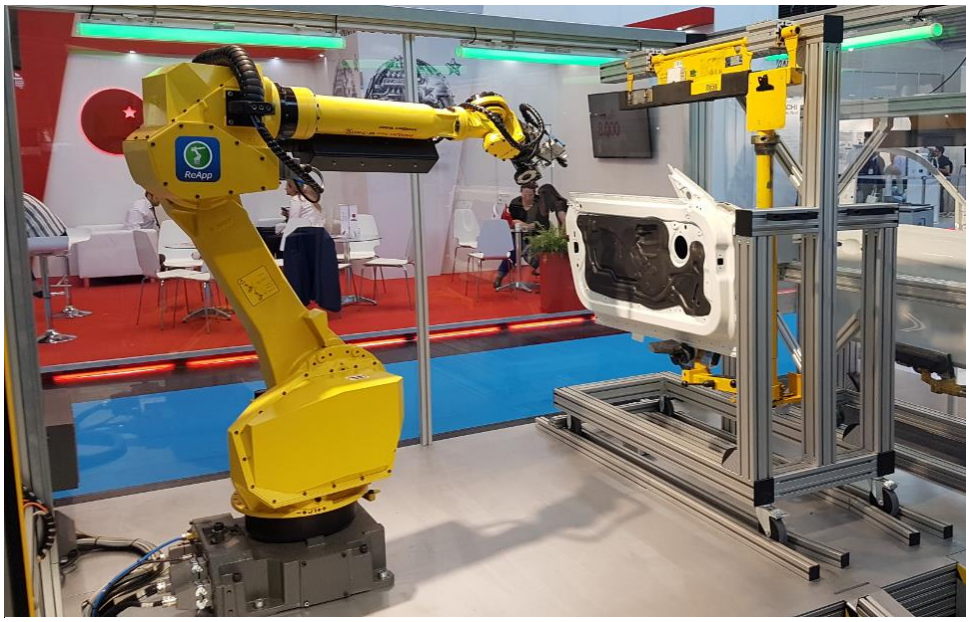


Abbildung 8 – ReApp Messestand mit Anwendungsfall der BMW AG

Die ReApp Werkzeugkette zielt neben der Herstellerunabhängigkeit auch darauf ab, Robotersoftwareentwicklung für Anwender möglichst intuitiv und ohne hohen Programmieraufwand zu ermöglich, um die Erschließung neuer Anwendungsfelder, insbesondere für Unternehmen kleinerer und mittlerer Größe, wirtschaftlicher zu gestalten [61].

Das Ziel der einfachen Bedienung stand ebenfalls bei einer Vielzahl weiterer Forschungsprojekte im Fokus. SMERobots [62], SMERobotics [63] oder Rosetta [64] sind hier zu nennen. In diesen Projekten wurden intuitive, CAD-unterstützte und wissensbasierte Programmiermethoden entwickelt, welche die automatische Konfiguration und Generierung von Roboterprogrammen ermöglichen. Während die genannten Projekte insbesondere auf Produktionsmitarbeiter mit geringer Roboterexpertise abzielten, fokussierte sich Softrobot auf die Zielgruppe der Softwareentwickler, in dem eine Softwarearchitektur für Robotersysteme entwickelt wurde, die eine herstellerunabhängige und objektorientierte Programmierung mit der Standard-Hochsprache Java ermöglicht [65].

Festzuhalten ist, dass die beschriebenen Forschungsprojekte meist die Abstraktion der proprietären Programmiersprache durch modellbasierte, graphische Programmiermethoden oder alternative Hochsprachen gewählt haben, um die Programmierung oder Konfiguration eines Robotersystems zu vereinfachen und dieses Vorgehen mit der Reduktion der Abhängigkeit von einem spezifischen Hersteller einhergeht.

Forschungsarbeiten – Finkemeyer, Friedrich, Krug, Lambrecht, Wenz

Der Ansatz, Programmlogik durch Abstraktion aus dem eigentlichen Robotersystem zu entziehen und in alternative Steuerungssysteme zu verlegen, findet sich ebenfalls in einer Reihe unterschiedlicher Forschungs- und Doktorarbeiten wieder.

So schlägt Finkemeyer in seiner Dissertation [66] ein Programmierkonzept vor, durch das die Komplexität bei der Integration von Sensorik und der Programmierung von sensorgeführten oder -überwachten Bewegungen durch ein High-Level-Interface verborgen wird. Zur Realisierung dieser Programmierung mit sogenannten Aktionsprimitiven wird ein modulares, Middleware-basiertes Steuerungssystem namens MIRPA vorgestellt. Sensorik und Aktorik werden über

Gerätetreiber in die MIRPA-Plattform eingebunden. Für die in der Arbeit aufgezeigten Beispielanwendungen wird ein Manipulator der Firma Manutec verwendet, welcher über eine spezielle Erweiterung der Leistungselektronik mit einer 1 kHz-Regelung und einem Interpolationstakt von 500 Hz angesteuert wird.

Friedrich hingegen fokussiert sich mit dem „Technologieorientierten Programmier- und Steuerungssystem für Industrieroboter“ nicht auf die Einbindung und Verwendung von Sensorik. Er legt Wert auf eine KMU-gerechte Programmierung und Steuerung des Gesamtsystems, um den Anforderungen an die Produktion kleiner Losgrößen gerecht zu werden. Das System soll so ausgelegt sein, dass Facharbeiter in der Lage sind, Konfiguration und Programmierung unabhängig vom eingesetzten Robotersystem durchführen zu können [67]. Das Gesamtsystem besteht aus den vier Kernkomponenten Programmierleitstand, Datenbanksystem, Prozess- und Robotersteuerung. Prozessschritte werden in sogenannte Elementaranweisungen zerlegt. Dabei handelt es sich um kleine, in sich abgeschlossene generische Roboterkommandos wie das Schalten eines Ausganges oder das Ausführen einer Punkt-zu-Punkt-Bewegung. Die Gesamtreihenfolge wird über den Programmierleitstand und das Datenbanksystem definiert und über die Prozesssteuerung an die eingebundenen Robotersysteme übermittelt. Diese setzen die erhaltenen Anweisungen mit der proprietären Robotersteuerung in Bewegungen oder Schaltaktionen um.

Die Methode, generisch definierte Roboterbefehle zu verwenden, welche anschließend über einen Interpreter der Robotersteuerung ausgeführt werden, wird ebenfalls durch Maletzki und Lambrecht angewandt. Maletzki fordert in seiner Arbeit [68] „neue Methoden zur herstellerunabhängigen und applikationsübergreifenden Programmierung von Roboteranwendungen“, um neue Anwendungsbereiche wie flexible Fertigungssysteme in der Robotik zu erschließen. Zudem sieht er einen systematischen, durchgängigen Entwicklungsprozess als notwendig an, um die zunehmende Komplexität der Anwendungen durch Aktorik und Sensorik zu beherrschen. Diesen zeigt er anhand eines Prototyps mit einer durchgängigen Werkzeugkette und einer aufgabenorientierten Programmierung, die Entwicklung und Steuerung von Roboterprogrammen auf Matlab-Basis umsetzt. Im Gegensatz zu Friedrich werden Aktorik und Sensorik über den Matlab-PC und nicht über die Robotersteuerung angesprochen, da Maletzki durch die Ansteuerung mit allgemeinen Hochsprachen statt der herstellerspezifischen Programmiersprache KRL Vorteile hinsichtlich Flexibilität und Funktionalität sieht.

Die Notwendigkeit einer herstellerunabhängigen Schnittstelle zur Ansteuerung von Robotern äußert Lambrecht im Zusammenhang mit der Entwicklung seiner „natürlich-räumlichen Industrieroboterprogrammierung auf Basis von markerloser Gestenerkennung und mobiler Augmented Reality“, um die breite Anwendbarkeit und einfache Implementierung auf beliebigen Industrierobotersteuerungen zu garantieren [69]. Hierzu stellt er ein System vor, das ein aufgabenorientiertes und universelles Roboterprogramm, ausgehend von einem mobilen Endgerät, über eine generische Roboterschnittstelle an einem Kuka-Roboter ausführt. Die Übertragung erfolgt durch einen zwischengeschalteten Industrie-PC, der die Befehle über eine RS-323-Schnittstelle an das KRL-Interpreterprogramm in der Kuka-Steuerung weitergibt. Die vorgestellte Schnittstelle soll sowohl für diverse Programmiergeräte als auch für übergeordnete Steuerungssysteme wie SPS oder Prozessleitsysteme verwendbar sein, da sich auch hier die Notwendigkeit der herstellerunabhängigen Ansteuerung ergibt [70].

Einen weiteren, übergeordneten Ansatz zur vereinheitlichten Ansteuerung von Robotersystemen bietet die Canonical Robot Command Language (CRCL) aus dem Forschungsbereich des National Institute of Standards and Technology. Entgegen der Namensgebung definiert die CRCL keine Roboterprogrammiersprache, sondern ein generisches Informationsmodell für Roboteraufgaben, die im XML-Format spezifiziert sind. Diese Beschreibung kann verwendet werden, um mit unterschiedlichen Kommunikationsmechanismen oder Programmiersprachen umgesetzt zu werden und Instruktionen an Robotersysteme zu versenden. Proctor et al. zeigen in ihrem Journalbeitrag [71] die Umsetzung des Konzeptes für Montageanwendungen anhand der Ansteuerung eines Fanuc LR Mate 200i und eines Kuka LBR iiwa Roboters auf Basis eines XML-Interfaces. Ebenso wird die CRCL-Definition verwendet, um eine Anbindung über die ROS-Middleware in die zugehörige Simulationsumgebung zu realisieren. Das Ziel der Standardisierung von CRCL durch die IEEE Robotics and Automation Society wird von den Autoren zwar erwähnt, ist aber bisher nicht erfolgt.

Während in den beschriebenen Arbeiten ein externes System verwendet wird, um ein Robotersystem und Peripherie zu steuern, zeigt Krug mit der von ihm vorgeschlagenen Plug & Produce-Methode [9], wie nicht der Betrieb, sondern die Konfiguration eines Robotersystems und der angeschlossenen Peripherie durch ein externes System erfolgen kann. Hierbei wird im Inbetriebnahmeprozess einer Roboteranlage ein automatisierter Konfigurationsprozess durchgeführt. Die einzubindenden Geräte stellen hierfür Gerätebeschreibungsdateien zur Verfügung, die einzelne Aspekte der Konfiguration wie Feldbuskonfiguration, Ein- und Ausgangszuordnung oder funktionale Abläufe beschreiben. Durch die konsequente Orientierung an offenen Standards wird versucht, diese Beschreibung möglichst herstellerunabhängig und universell zu halten. Funktionale Abläufe der Peripherie werden beispielsweise in IRL definiert und im Konfigurationsprozess zu den herstellerspezifischen Programmdeklarationen transformiert und in der Robotersteuerung hinterlegt. Der Betrieb ist so ohne ein externes Steuerungssystem und eine Kommunikationszwischenschicht möglich, allerdings muss die Bedienung weiterhin am Robotersystem des Herstellers erfolgen.

Dass ein automatischer Konfigurationsprozess nicht nur für Aktorik und Sensorik von Bedeutung ist, stellt Wenz in seiner Dissertation mit dem Thema „Automatische Konfiguration der Bewegungssteuerung von Industrierobotern“ dar [37]. Er legt in seiner Arbeit den Fokus nicht direkt auf die Anforderungen der Endanwender von Robotersystemen, sondern auf die Entwicklung von Steuerungssoftware für unterschiedliche Robotermodelle. Diese ist bis dato stark eingeschränkt, da Bewegungssteuerung und Software stark an Robotertypen gekoppelt sind. Insbesondere mangelnde Erweiter-, Wart- und Portierbarkeit sind seines Erachtens nach hier zu nennen. Er entwirft daher ein modulares Softwaresystem mit roboterunabhängigen Komponentenschnittstellen, mit dem ein Benutzer ohne fundiertes Expertenwissen eine Bewegungssteuerung inklusive Roboterkinematik und Dynamik konfigurieren kann. Durch einen IRL-Interpreter wird eine herstellerunabhängige Einbindung von Anwenderprogrammen sichergestellt. Die Offenheit des Gesamtsystems soll durch die Open-Source-Implementierung gewährleistet werden.

Industrielle Ansätze der Automatisierungstechnik

Die im vorangegangenen Abschnitt beschriebenen Lösungen stellen Arbeiten und Anwendungen im wissenschaftlichen Umfeld und durch Forschungsinstitute initiierte Projekte dar. Es existieren jedoch auch in der Automatisierungsindustrie Anwendungen und Produkte, die sich mit der hersteller- und hardwareunabhängigen Gestaltung der Programmierung, Steuerung und Integration von Robotersystemen beschäftigen.

Programmierung

Das Unternehmen Kuka bietet seit dem Jahr 2012 die Möglichkeit an, seine Robotersysteme mit G-Code statt der Kuka-eigenen Sprache Kuka Robot Language (KRL) zu programmieren [72]. Bei G-Code handelt es sich um eine normierte Programmiersprache aus dem Werkzeugmaschinenbereich, die es ermöglicht, Steuerprogramme für NC-Maschinen einheitlich und hardwareunabhängig festzulegen [73]. Mit dem Softwarepaket Kuka.cnc wird das Robotersystem um einen passenden Interpreter erweitert und NC-Programme können ohne Umwandlung direkt von der Robotersteuerung verarbeitet werden. Zusätzlich ist das Bedienkonzept an Werkzeugmaschinensysteme angelehnt. Ziel ist es, den Roboter für Standardaufgaben der Metallbearbeitung wie Fräsen und Drehen verwenden zu können, wobei der Maschinen- bzw. Roboterbediener kein roboterspezifisches Know-how mehr besitzen muss [74].

Einen weitaus radikaleren Ansatz unternimmt Kuka im Bereich der Servicerobotik mit dem Leichtbauroboter iiwa und der zugehörigen Steuerungsplattform Kuka Sunrise [75]. In dieser wird statt der proprietären Programmiersprache KRL die objektorientierte Hochsprache Java zur Ablaufprogrammierung verwendet. Dadurch sollen Anwendungen offener und modularer gestaltet werden können und durch externe Softwarebibliotheken einfacher erweiterbar und wiederverwendbarer sein, um komplexe Applikationen schneller, flexibler und intelligenter zu entwickeln und somit den Anforderungen der Servicerobotik besser gerecht zu werden [15]. Ein Transfer der Programmiermethodik und -sprache aus dem Kuka-Sunrise-System auf die Industrierobotersysteme der Firma Kuka hat noch nicht stattgefunden und ist für die Zukunft bisher auch nicht kommuniziert.

Gängige Praxis stellen in der industriellen Programmierung von Robotersystemen zudem Tools der Offline-Programmierung dar. Diese ermöglichen es, Roboterprogramme in einer Simulationsumgebung mit Hilfe eines PCs vom Büroarbeitsplatz aus zu entwickeln, anstatt am Robotersystem an einer Anlage vor Ort. Zwar unterstützen die Softwarewerkzeuge der Roboterhersteller nur die Programmierung ihrer eigenen Systeme, allerdings existieren bereits seit den 80er Jahren Drittanbieter von Offline-Programmiersoftware. Mit dieser können Roboterprogramme in einer roboterherstellerunabhängigen Simulationsumgebung entwickelt werden, wobei der Programmcode anschließend mittels Postprocessing für eine herstellerspezifische Steuerung ausgeleitet werden kann. Da diese Tools häufig aus der CAD- bzw. CAM-Umgebung stammen, stellen sie teure Expertensysteme wie Delmia Robotics [76] oder Process Simulate Robotics [77] dar und legen den Fokus auf die Modellierung und Generierung von Bewegungsbahnen statt auf die Entwicklung komplexer Applikationslogiken.

Befördert durch die steigende Anzahl an Anbietern von günstigen und leichter handzuhabenden Servicerobotern sind in den letzten Jahren Softwareprodukte entstanden, die Offline- und Online-Programmierung verschmelzen lassen und zum Ziel haben, die Programmentwicklung durch PC-basierte Programmier- und Bedientools auch für weniger erfahrene Anwender zu ermöglichen, beispielsweise ArtiMinds RPS [78] oder drag&bot [79]. Beide Produkte bieten eine intuitive und moderne graphische Programmieroberfläche, mit der Roboterprogramme anhand

vorgefertigter Funktionsmodule erstellt, simuliert und inklusive Peripherieanbindung einfach in Betrieb genommen werden können, jedoch unterscheiden sich die Konzepte in der steuerungstechnischen Ausführung. Während ArtiMinds das Programm, identisch zur klassischen Simulationssoftware, in die proprietäre Robotersprache übersetzt und dieses anschließend isoliert auf der Robotersteuerung ausgeführt wird, verfolgt drag&bot ein Konzept analog zu den vorgestellten wissenschaftlichen Arbeiten wie Friedrich, Lambrecht, Maletzki oder ReApp, da ein PC mit einer auf ROS basierenden spezifischen drag&bot Runtime die Steuerung des Gesamtsystems inkl. Roboter und Peripherie übernimmt [80].

Steuerungskonzepte

Die Strategie, Roboterherstellerabhängigkeit durch die Verwendung zusätzlicher externer Steuerungssysteme zu reduzieren, verfolgen auf industrieller Seite auch Steuerungstechnikhersteller wie Keba [81], B&R [82] oder Siemens [83]. Diese bieten eigenständige Softwaresysteme zur Programmierung und Steuerung von Robotern über Industrie-PCs oder speicherprogrammierbare Steuerungen an. Sie sind allerdings ebenso auf die Unterstützung durch den jeweiligen Roboterhersteller angewiesen, um ein Robotersystem über elektrische sowie logische Schnittstellen korrekt anzusteuern. Zudem ist je nach Anwendung spezifisches, z. T. wettbewerbsrelevantes Know-how bezüglich der Roboterkinematik erforderlich. Die Wahl der Schnittstelle und des Steuerungskonzeptes fällt daher sehr unterschiedlich aus. Entscheidend ist oft nicht der Anwendungsfall, sondern welche Art der Anbindung die Roboterhersteller bereit sind zuzulassen. Dies reicht von der direkten elektrischen Regelung des Manipulators [84] über digitale Feldbusschnittstellen zur Drehmoment- oder Positionsübermittlung einer Trajektorie [85] bis hin zur Vorgabe in sich abgeschlossener Bewegungsbefehle oder Schaltvorgänge [86]. Ziel derartiger Lösungen ist in der Regel die bessere Integration des Roboters in das Gesamtsystem einer Automatisierungsanlage, die beispielsweise über eine klassische SPS oder einen Industrie-PC gesteuert wird. Auf Inbetriebnahme- und Bedienseite soll so vermieden werden, mit unterschiedlichen Programmiersprachen arbeiten zu müssen, da der Roboter aus der übergeordneten Steuerung und der dort verwendeten Programmiersprache angesprochen wird.

Es existieren ebenfalls industrielle Anwendungen, die neben den Aspekten der Bedienbarkeit auch aus funktionalen Gründen ein externes Steuerungskonzept für den Betrieb eines Robotersystems wählen, beispielsweise wie bei einer komplexen Handlingsapplikation für eine „Griff-in-die-Kiste“ Automatisierungslösung bei BMW in Steyer [87]. Dabei ist ein hohes Maß an Bildverarbeitung erforderlich, um Schüttgut zu identifizieren und individuelle, kollisionsfreie Bahnen für jede Handlingsoperation zu generieren. Programmiersprache und Architektur des Robotercontrollers sind darauf nicht ausgelegt, sodass ein Großteil der Algorithmik von einem PC-System durchgeführt und anschließend an das Robotersystem übergeben werden muss. Auch die Einbindung von Robotern in PC-basierte universelle Entwicklungsumgebungen für die Mess- und Prüftechnik stellen Anwendungen dar, die externe Schnittstellen erforderlich machen. So bietet z. B. die Firma DigiMetrix verschiedene Bibliotheken an, die es ermöglichen, unterschiedliche Robotersysteme in LabView Applikationen zu integrieren [88].

Auch wenn die dargestellten Lösungen die Problematik der Abhängigkeit an einen spezifischen Roboterhersteller und dessen Softwareschnittstellen aus Anwendersicht zum Teil mindern, handelt es sich dabei häufig nur um Individuallösungen, die einen hohen Implementierungsaufwand besitzen und neue Abhängigkeiten schaffen, beispielsweise zum Lieferanten der übergeordneten Steuerung.

Standardisierungsvorhaben

Im Umfeld der industriellen Automatisierungstechnik für die Integration von antriebstechnischen Komponenten sind im Bereich der Standardisierungsinitiativen insbesondere CiA 402 [89], ProfiDrive [90] und PLCOpen MotionControl [91] zu nennen. Diese definieren in Form von Zustandsautomaten und Befehlsaufrufen auf Feldbus- bzw. Funktionsbausteinebene eine standardisierte Schnittstelle, um einzelne Antriebskomponenten oder Mehrachssysteme über externe Schnittstellen anzusprechen. In der Robotik ist dies primär für das interne Steuerungssystem relevant, da durch die erwähnten Standards Interoperabilität für die verwendeten Komponenten der Antriebstechnik geschaffen wird. CiA 402, ProfiDrive und PLCOpen Motion Control können jedoch auch genutzt werden, um in Anlehnung an die im vorangegangenen Abschnitt erläuterten Integrationskonzepte Teilumfänge der Robotersteuerung auf Antriebsebene mit Hilfe von standardisierten Schnittstellen zu ersetzen. Allerdings muss dies, wie erwähnt, auch vom Hersteller unterstützt werden.

Ein weiteres Standardisierungsumfeld deckt die OPC Foundation mit ihrer OPC UA-Architektur ab. Diese soll als plattformunabhängiger und serviceorientierter Standard den Austausch von Daten und Informationen in der Automatisierungstechnik dienen und somit eine herstellerübergreifende universelle Kommunikation von der Sensor- bis zur Leitstandsebene ermöglichen [92]. Für das Anwendungsspektrum der Industrierobotik existiert seit 2018 ein VDMA-Normentwurf, der ein OPC UA-Informationsmodell für Robotersysteme definiert [93]. Dieses soll genutzt werden, um hersteller- und typunabhängig Informationen von Robotersystemen an übergreifende Leitstandsysteme zu übermitteln. Das Kommunikationskonzept zielt nicht auf die Steuerung eines Roboters ab, sondern auf Zustandsüberwachung und Wartungsfunktionalitäten.

Jüngere wissenschaftliche Veröffentlichungen zeigen indes den Bedarf auf, auch Steuerungsfunktionen über Funktionsschnittstellen mittels OPC UA zu implementieren und neue Ansätze wie Plug & Produce-Konzepte bei Dorofeev et al. [94] oder wie generische Tool- und Manipulatorschnittstellen bei Kaspar et al. [95] umzusetzen, häufig in Verbindung mit den bereits im Rahmen der wissenschaftlichen Ansätze dargestellten Technologien wie ROS oder der Robotics Library [96].

3. Abgrenzung, zentrale Forschungsfragen und Vorgehen

Im Rahmen der bisherigen Arbeit wurde beschrieben, dass die Integration von Robotersystemen in ein Automobilwerk ein langwieriger und mehrjähriger Prozess ist, da aufgrund proprietärer Schnittstellen und Herstellerspezifitäten von Industrierobotern eine Vielzahl von Entwicklungs- und Inbetriebnahmearbeiten erforderlich ist. Diese kostenintensiven Initialaufwände führen zu einer hohen Herstellerabhängigkeit, werden mit steigender Lieferantenanzahl zunehmend verstärkt und schränken die Flexibilität und Innovationsgeschwindigkeit der Automobilproduktion ein, da Karosseriebauapplikationen auf den proprietären Robotersystemen entwickelt werden müssen und eine herstellerübergreifende Wiederverwendung von Roboterprogrammen nicht möglich ist.

Der existierende Stand von Wissenschaft und Technik legt dar, dass bereits diverse Forschungs- und Standardisierungsarbeiten, aber auch Industrieprodukte, Möglichkeiten aufzeigen, die Gestaltung und Inbetriebnahme von Robotersystemen unabhängiger von der verwendeten Hardware bzw. dem eingesetzten Hersteller zu realisieren. Die industrielle Realität zeigt indes, dass weder die angebotenen Lösungen noch die zugrunde liegenden Methoden im automobilen Karosseriebau Anwendung finden. Es existiert bis dato auch keine theoretische Betrachtung oder praktische Untersuchung, welche die Anwendbarkeit und Eignung von offenen Systemlösungen für die Automobilproduktion und deren Anforderungen untersucht.

Aus Sicht des Autors ergeben sich folglich vier zu bearbeitende Themengebiete und zugehörige Forschungsfragen, deren systematische Beantwortung notwendig ist, um die herstellerunabhängige Integration von Industrierobotern in den automobilen Karosseriebau zu ermöglichen.

Anforderungsbild

Eine Vielzahl der beschriebenen wissenschaftlichen Arbeiten und Projekte konzentriert sich auf Anwendungen und somit auch auf das Anforderungsbild von kleinen und mittleren Unternehmen. Insbesondere die hochflexible Produktion mit intuitiven und einfachen Bedien- und Programmierkonzepten sowie schnellen Umrüstmöglichkeiten wird häufig als Ziel definiert - eine Vision, die sich zwar auch mit den Zielen der Automobilindustrie überschneidet [97, 98], aber noch weit entfernt von der industriellen Praxis der Großserienproduktion ist. Zudem wird das damit einhergehende postulierte Ziel der Losgröße 1 in der Automobilindustrie als weniger relevant angesehen [99]. Es kann somit nicht davon ausgegangen werden, dass eine schnelle Umrüstung oder einfache Bedienung in der Großserienfertigung eine ebenso hohe Bedeutung wie im Produktionsumfeld von KMU besitzt. Das industrielle Lösungsangebot wie das Java-basierte Kuka Sunrise OS oder die integrierten Steuerungskonzepte von B&R sind zwar stärker auf die Großserienproduktion ausgerichtet, fokussieren sich aber vor allem auf die Servicerobotik oder den allgemeinen Maschinen- und Anlagenbau. Des Weiteren ist, insbesondere bei Roboterherstellern, davon auszugehen, dass die herstellerübergreifende Austauschbarkeit keinen bedeutenden Schwerpunkt in der Entwicklungsarbeit darstellt.

Es lässt sich festhalten, dass sich keine der betrachteten Arbeiten und existierenden Technologien konkret mit dem Hauptanwendungsfeld der industriellen Robotik beschäftigt – dem Karosseriebau in der Automobilindustrie. Dementsprechend ist bisher keine Betrachtung erfolgt, die den dazugehörigen Entwicklungsprozess sowie den Anforderungsbedarf analysiert, der sich aus der herstellerunabhängigen Integration von Robotersystemen ergibt.

Die sich daraus ableitende Forschungsfrage lautet:
Welche Anforderungen bestehen an die herstellerunabhängige Integration von Robotersystemen im automobilen Karosseriebau?

Abstraktion durch Externalisierung

Eine weitere Gemeinsamkeit, die im Rahmen der Betrachtung des Standes der Wissenschaft und Technik deutlich wird, ist, dass die Reduktion der Herstellerabhängigkeit durch Externalisierung von Prozesslogiken in alternativen Steuerungssystemen ein valides und häufig verwendetes Prinzip ist. Ebenso ist erkennbar, dass eine breite Möglichkeit hinsichtlich der Art und Weise gegeben ist – sowohl aus konzeptioneller Sicht als auch in der technischen Umsetzung.

Allerdings begründet keine der vorliegenden Arbeiten das gewählte Vorgehen und geht dabei umfassend auf die zur Verfügung stehenden Möglichkeiten ein. Darüber hinaus findet weder ein Abgleich mit den Leistungsanforderungen im Allgemeinen noch mit den spezifischen Anforderungen des Karosseriebaus statt. So existiert beispielsweise keine Untersuchung der Bahngenauigkeit und ähnlicher Kriterien in den beschriebenen Arbeiten – obwohl diese zentrale Anforderungen für den Produktionsbetrieb darstellen. Zusätzliche oder reduzierte Integrationsaufwände werden nicht betrachtet und die beschriebenen Plug & Produce-Methoden beschäftigen sich vor allem mit der Integration von Aktorik und Sensorik für ein Robotersystem – nicht mit der Einbindung des Robotersystems an sich.

Es ergibt sich folgende Forschungsfrage:
Welche Methode der externen Ansteuerung von Robotersystemen ist für den automobilen Karosseriebau am besten geeignet?

Abstraktion durch Programmierung

Die Verlagerung der Programmlogik aus dem Robotersystem in andere Steuerungssysteme bietet die Möglichkeit, alternative Programmiersprachen und -methoden anzuwenden und stellt somit einen weiteren Ansatzpunkt dar, die Abhängigkeit vom Herstellersystem zu reduzieren. Das angewandte Spektrum erstreckt sich dabei von klassischen Hochsprachen und domänenspezifischen Programmiersprachen über visuelle und simulationsbasierte Ansätze bis hin zum Programmieren durch Vormachen oder gestenbasierte Anweisungen. Analog zu den ersten beiden Forschungsfragen ist der Anwendungs- und Anforderungsschwerpunkt nicht der Karosseriebau einer Automobilproduktion. Middlewarebasierte Ansätze wie ROS oder MRDS legen sich nicht auf ein einzelnes Programmierkonzept oder eine einzelne Sprache fest, sondern überlassen diese Entscheidung dem Entwickler und Anwender selbst. Zudem sind die zur Verfügung stehenden Methoden sehr IT-getrieben und gehen nicht zwangsläufig auf die Anforderungen von Industrieroboteranwendern ein.

Es wurde bisher nicht betrachtet, welchen Funktionsumfang Programmiersprachen aus Sicht der Automobilproduktion abdecken müssen und inwiefern diese von den im Stand der Wissenschaft und Technik definierten Möglichkeiten erfüllt werden. Ebenfalls ungeklärt ist, welche Anwendergruppen hiervon betroffen sind und welche Auswirkung dies auf die Programmierkonzepte haben kann.

Die dritte Forschungsfrage lautet folglich:
Welche Roboterprogrammiermethode und -sprache deckt die notwendigen Funktionalitäten und Anforderungen im Planungs-, Entwicklungs- und Produktionsprozess der Automobilproduktion im Karosseriebau am besten ab?

Prototyp und Evaluierung

Die Gesamtevaluierung der vorgeschlagenen Konzepte und deren technische Machbarkeit variieren stark und reichen von einer rein theoretischen Betrachtung über kleinere Prototypen bis hin zu industriellen Anlagen mit mehreren Robotern. Jedoch ist analog zum definierten Anforderungsbild häufig das KMU-Umfeld Gegenstand der technischen Umsetzung und Evaluierung. Der Erfüllungsgrad und die Vor- und Nachteile werden dementsprechend auch nur in diesem Bereich bzw. für einen spezifischen Anwendungsfall besprochen.

Es existiert bisher noch keine Arbeit, die mit den vorangestellten Forschungsfragen verbundene Lösungsansätze für die herstellerunabhängige Roboterintegration im Karosseriebau an einem technischen Prototyp aufzeigt, deren Erfüllungsgrad bewertet und Vor- und Nachteile aus Großunternehmenssicht diskutiert.

Die zu bearbeitende Forschungsfrage lautet:
Wie und mit welchem Ergebnis ist die herstellerunabhängige Integration von Robotersystemen an einem Prototyp für den automobilen Karosseriebau zu evaluieren und welche Vor- und Nachteile ergeben sich aus Unternehmenssicht?

Vorgehen

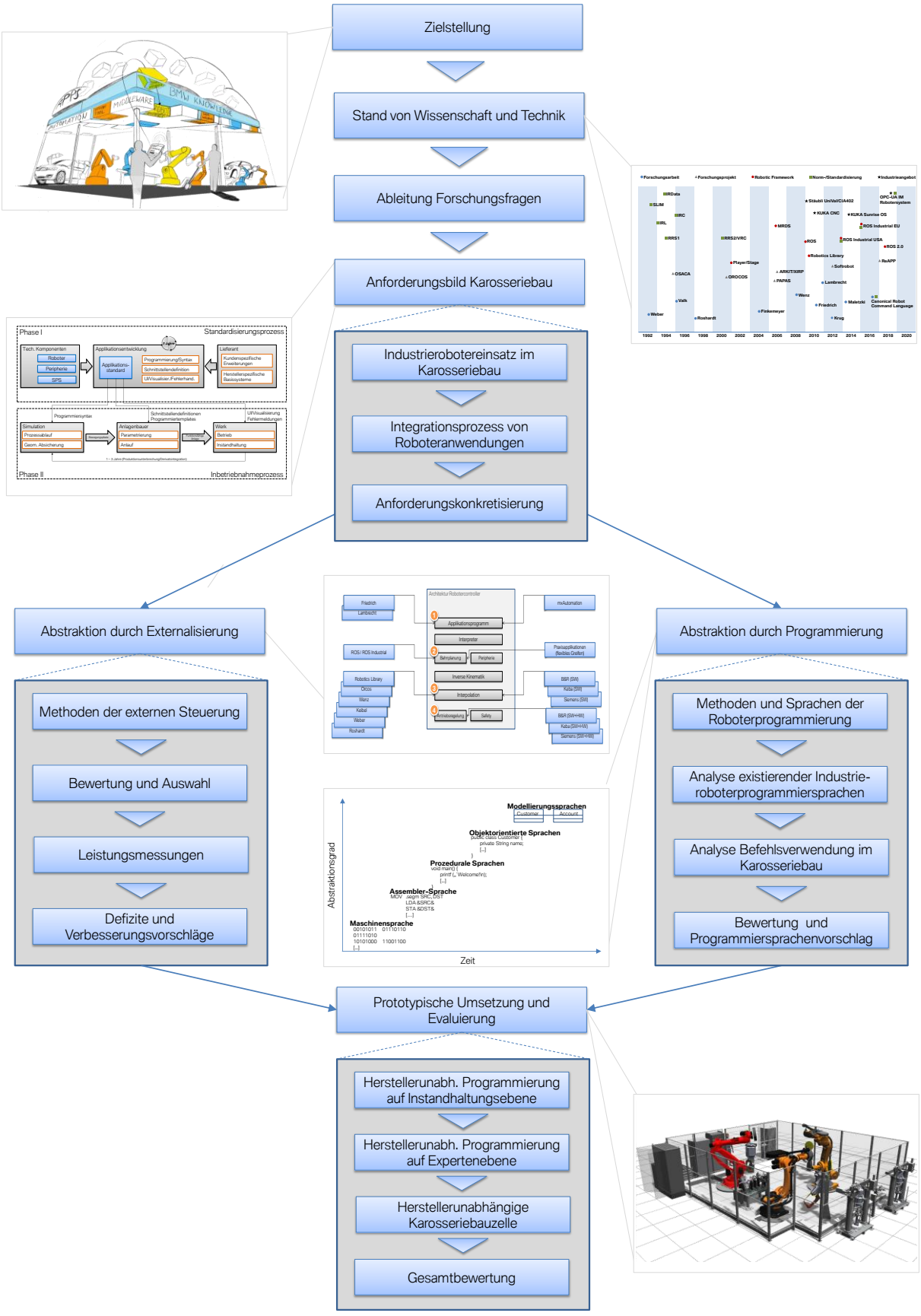


Abbildung 9 – Vorgehen innerhalb der Arbeit

Im vorherigen Abschnitt wurde das Thema dieser Arbeit – „Herstellerunabhängige Integration von Industrierobotersystemen in den automobilen Karosseriebau“ – in vier Forschungsfragen unterteilt, welche die zentralen Arbeitspakete darstellen. Diese werden im Folgenden detaillierter erläutert und in Abbildung 9 anhand des übergreifenden Vorgehens der Arbeit zugeordnet.

Im ersten Schritt wird auf die Anforderungsermittlung aus Sicht des automobilen Karosseriebaus eingegangen. Hierfür wird zunächst der grundsätzliche Aufbau von Karosseriebauzellen erläutert, um ein klares Bild der Steuerungsarchitektur zu vermitteln und die zentrale Bedeutung des Roboters im Produktionsprozess zu unterstreichen. Anschließend wird der Softwareentwicklungs- und Integrationsprozess für Robotersysteme in der Automobilproduktion analysiert und aufgezeigt, welche Rolle die unternehmensinterne Standardisierung von Robotertechnik und Applikationen hat und welche Einschränkungen aufgrund der proprietären Systeme entstehen. Zudem wird untersucht, ob und in welcher Form Anforderungsunterschiede für Unternehmen der Großserienproduktion im Gegensatz zu kleineren und mittleren Unternehmen bestehen und welche Auswirkung dies auf eine herstellerunabhängig zu gestaltende Roboteranwendung hat.

In Kapitel 5 wird sich dem Thema der externen Schnittstellen von Robotersystemen gewidmet. Im ersten Abschnitt werden die vorhandenen Methoden und Abstraktionsebenen der externen Ansteuerung erläutert und klassifiziert, wobei auch auf die entsprechenden Vor- und Nachteile eingegangen wird. Anschließend wird ein Bewertungsmodell entwickelt, das je nach Anwender oder Anwendungsfall eine objektive Grundlage zur Entscheidungsfindung bietet, um eine geeignete Integrationsmethode zu bestimmen. Darüber hinaus werden die Leistungsfähigkeit und Einsetzbarkeit der Schnittstellen im externen Steuerungsverbund für die Automobilproduktion diskutiert und für essenzielle technische und prozessuale Anforderungen durch lasertrackerbasierte Performancemessungen evaluiert. Anhand der im Rahmen der steuerungstechnischen Umsetzung erlangten Erkenntnisse werden anschließend Defizite und Verbesserungsvorschläge bzgl. der untersuchten Schnittstellen aufgezeigt und ein neues Verfahren zur Integration von Robotersystemen mit externen Steuerungen definiert, um den Bedürfnissen sowohl auf Anwender- als auch Lieferantenseite bei einer flexiblen und automatisierten Plug & Produce-Lösung gerecht zu werden.

Das sechste Kapitel beinhaltet die Programmierung und Bedienung von Robotersystemen. Zu Beginn werden die unterschiedlichen Programmiermethoden und -sprachen im Umfeld der Robotik erläutert. Im nächsten Schritt werden Programmiersprachen von Industrierobotern in Bezug auf ihre funktionale Mächtigkeit analysiert und es wird eine vergleichende Befehlsdatenbank aufgestellt. Anschließend wird der Fokus auf reale Anwendung und deren Anforderung in der Automobilproduktion gelegt und diskutiert, inwiefern die vom Hersteller angebotenen Programmfunktionen den notwendigen Befehlsumfang abdecken. Hierfür werden die Roboterlinien zweier Automobilwerke hinsichtlich der Befehlsverwendung analysiert und die Befehlsdatenbank diesbezüglich erweitert. Anhand der Untersuchungsergebnisse werden die Anforderungen der unterschiedlichen Nutzerkreise der Systeme beschrieben und es wird dargestellt, welche universellen Programmiersprachenalternativen zur Verfügung stehen und inwiefern diese für das jeweilige Anwendungsspektrum geeignet sind. Auf dieser Basis wird ein herstellerunabhängiges und nutzergerechtes hybrides Programmiersprachenkonzept vorgeschlagen, welches die Vorteile einer einfachen domänenspezifischen Roboterprogrammiersprache für den Karosseriebau mit den Vorzügen einer objektorientierten allgemeinen Hochsprache verbindet und so den funktionalen und technischen Anforderungen der Automobilproduktion gerecht wird.

Im letzten Kapitel werden die Lösungen der vorangegangenen Forschungsfragen in einem technischen Prototyp synthetisiert, deren Umsetzbarkeit nachgewiesen und die Anforderungserfüllung evaluiert. Hierfür wird zunächst die Eignung des hybriden Programmiersprachenkonzeptes sowohl anhand einer herstellerunabhängigen Bedien- und Programmierumgebung für Werksanwender als auch an einer offenen Entwicklungsumgebung auf Expertenebene aufgezeigt. Anschließend erfolgt die gesamthafte Umsetzung der herstellerunabhängigen Roboterintegration in Form einer Karosseriebautestzelle mit Robotersystemen unterschiedlicher Hersteller. Anhand dieses Prototyps konnten relevante Anwendungsfälle des Karosseriebaus mit einem herstellerunabhängigen Steuerungskonzept exemplarisch umgesetzt und die Anforderungserfüllung sowie die technische Umsetzbarkeit erfolgreich nachgewiesen werden. Abschließend werden noch die einhergehenden Vor- bzw. Nachteile aus Großunternehmenssicht erörtert.

Fallbeispiel und Untersuchungsgegenstand

Da sich die vorliegende Arbeit auf das Umfeld des automobilen Karosseriebaus beschränkt und die Notwendigkeit von Anforderungsidentifikation und -abgleich hinsichtlich der unterschiedlichen prozessualen, technischen und anwenderbezogenen Aspekte hervorhebt, kommt der fundierten Analyse der gegenwärtigen industriellen Praxis ein hoher Stellenwert zu. Aus diesem Grund war es dem Autor wichtig, ein konkretes Fallbeispiel in Form eines automobilen OEM[8] untersuchen zu können, weshalb sich in Deutschland insbesondere die Unternehmen Audi, BMW, Daimler und Volkwagen als Untersuchungsgegenstand anbieten, da hierzulande sowohl deren Produktionsstätten als auch die technischen Planungsabteilungen ansässig sind.

Als zentraler Anforderungsgeber dieser Dissertation konnte die BMW AG gewonnen werden, deren Produktionsumfeld sich für die Erarbeitung des Themenschwerpunkts besonders gut eignet, da mit ABB und Kuka zwei der bedeutendsten Industrieroboterhersteller bereits langjährige Lieferanten sind. Zudem wurde parallel zum Bearbeitungszeitraum dieser Arbeit mit Fanuc ein dritter Systemlieferant in den BMW-Produktionsstandard integriert. Im gewählten Fallbeispiel waren die Herausforderungen und Aufgaben in Bezug auf die Integration von Robotersystemen somit sehr präsent. Neben der zentralen Anforderung, eine ausreichende Anzahl an Roboterlieferanten und fundierte Integrationserfahrung zu besitzen, boten sowohl das Unternehmensumfeld der Serien- als auch der Vorentwicklung geeignete Rahmenbindungen zur Erarbeitung der definierten Forschungsfragen an, da die technischen Produktionsstandards industrieweit einen innovativen und nachhaltigen Ruf besitzen und sowohl der branchenübergreifenden Standardisierung als auch der wissenschaftlichen Erarbeitung von Themen der Roboter- und Automatisierungstechnik im Unternehmen ein hoher Stellenwert zukommt. Dies zeigt sich z. B. an der zeitnahen Adaption und Mitgestaltung neuer Technologien und Methoden der Steuerungs- und Robotertechnik [20, 100, 101] und der Mitarbeit an Standardisierungs- und Forschungsvorhaben wie RRS [28], ROS-Industrial [51] oder ReApp [60].

[8] Original Equipment Manufacturer – in der Automobilbranche sind unter dem Begriff OEM die Fahrzeughersteller zu verstehen, z. B. Volkswagen, BMW.

4. Roboterintegration am Beispiel der BMW AG

4.1. Robotereinsatz in den Technologien der Automobilproduktion

Die Herstellung von Kraftfahrzeugwagen ist heute üblicherweise in einer Fließfertigung in Großserienproduktion organisiert. Ein klassisches automobiles Vollwerk deckt die zentralen Fertigungstechnologien mit den Prozessabschnitten Presswerk, Karosseriebau, Lackiererei und Montage ab.

Diese vier Kerntechnologien unterscheiden sich neben den verwendeten Produktionstechnologien auch in den verwendeten Betriebsmitteln, der Automatisierungsquote und der Anwendungsbreite von Robotersystemen [102].

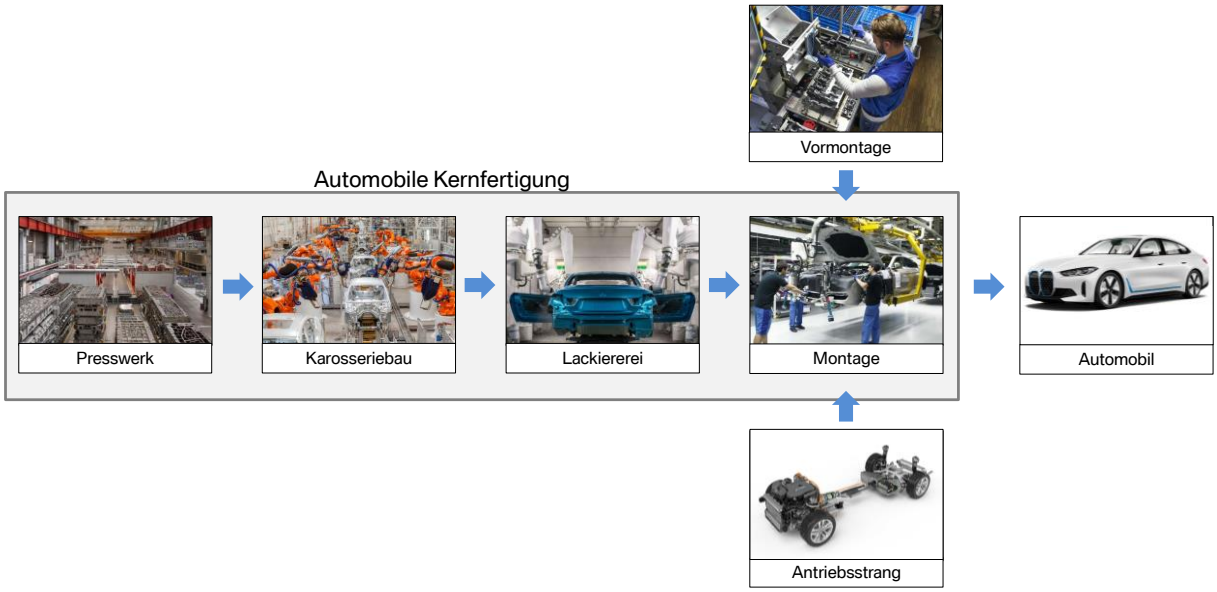


Abbildung 10 – Produktionsschritte in der Automobilfertigung in Anlehnung an [103]

Das Presswerk ist durch fertigungstechnische Umformprozesse sowie Pressstraßen charakterisiert. Roboter finden insbesondere für einfache Handlings- und Palettierprozesse Anwendung. Sie werden i. d. R. als Gesamteinheit mit der Presse erworben und der Integrationsprozess der Robotersysteme wird durch den Lieferanten durchgeführt. Eine hohe spezifische Softwareanpassung durch den OEM ist nicht erforderlich.

Die darauffolgende Kerntechnologie stellt der Karosseriebau dar. Das Produktionsbild ist vor allem durch aneinandergereihte Roboterzellen ausgezeichnet (Abbildung 11), in denen mit Werkzeugen ausgestattet Roboter die einzelnen Blechteile durch unterschiedlichste Fügeverfahren zunächst zu einzelnen Baugruppen und letztendlich zur Karosserie zusammensetzen. Im Gegensatz zum Presswerk existiert eine hohe Anzahl an komplexen Roboterapplikationen, die infolge der hohen Verzahnung zum Produkt und dem Produktionsprozess stark durch OEM-spezifische Anforderungen geprägt sind. Die Integration des Robotersystems ist ein langwieriger Prozess, an dem sowohl die Roboter- und Prozessexperten des OEM als auch die Peripherie-, Werkzeug- und Roboterlieferanten beteiligt sind.

Abbildung 11 – Karosseriebaulinie (Quelle: BMW AG)

Die Oberflächenbearbeitung der Rohkarosse erfolgt in der Lackiererei. Diese ist vor allem durch Sonderbetriebsmittel wie industrielle Lackierroboter, Trocknungsöfen und spezielle Fördertechnik geprägt [102]. Analog zum Presswerk werden die Lackierroboter zusammen mit einer schlüsselfertigen Lackieranlage erworben. Die durchgeführten Anwendungen wie z. B. Zerstäuben sind stark prozessspezifisch und stehen in enger Abhängigkeit zur Gesamtanlage. Die Integration erfolgt aus diesem Grund über das Gesamtsystem des Lieferanten.

Die geringste Automatisierungsquote und Roboterdichte weißt die letzte Kerntechnologie auf – die Montage. Sie ist durch eine Vielzahl manuell durchgeführter Tätigkeiten charakterisiert, welche die lackierte Rohkarosse mit Interieur, Exterieur und Antriebstechnik ausstatten und so zum finalen Produkt, dem Automobil, werden lassen. Aufgrund der hohen Anzahl unterschiedlichster Fertigungsprozesse und Varianten in der Montage ist eine Automatisierung oft nicht wirtschaftlich. Der Robotereinsatz beschränkt sich folglich auf einige wenige Applikationen wie Scheibenkleben, Cockpiteinbau oder der Montage von Schiebehebedächern. Dabei handelt es sich meist um individuelle Einzelanwendungen, die zusammen mit einem Anlagenlieferanten entwickelt werden.

Betrachtet man die Kerntechnologien hinsichtlich des Robotereinsatzes, zeigt sich, dass der Karosseriebau das zentrale Anwendungsumfeld in der Automobilproduktion ist. Die hohe Roboterdichte, das erforderliche OEM-spezifische Know-how und die hohe Anzahl skalierbarer Applikationen sind maßgebliche Gründe, sich insbesondere diesem Einsatzgebiet in Bezug auf die Herausforderungen und Potenziale der Herstellerunabhängigkeit von Robotersystemen zu widmen.

4.2. Steuerungsarchitektur von Karosseriebauzellen

In Kapitel 2.1 wurde bereits der grundsätzliche Aufbau eines Industrieroboters erläutert. Im Folgenden soll auf die Steuerungsarchitektur und Fertigungsvorgänge von Karosseriebauroboterzellen eingegangen werden, um anschließend den notwendigen Engineering- und Integrationsprozess für Robotersysteme beschreiben zu können. Eine gute Orientierungshilfe für die relevanten Hierarchiestufen und deren Aufgaben bietet das Ebenenmodell der Automatisierungspyramide, welches in Abbildung 12 dargestellt und für den Karosseriebau instanziiert ist.

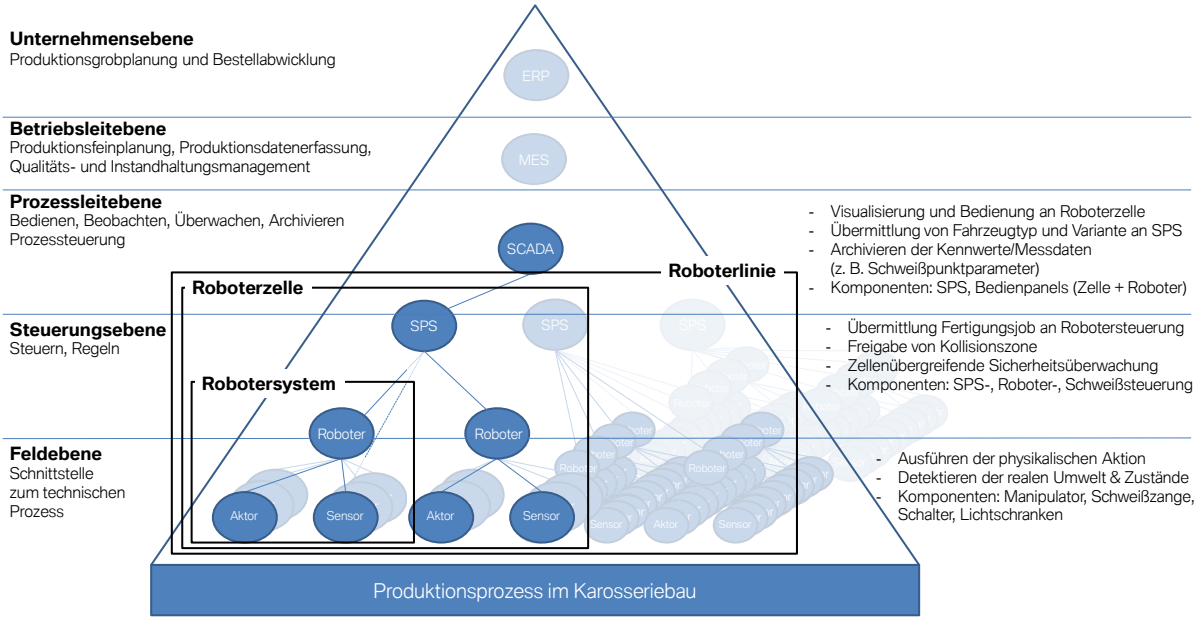


Abbildung 12 – Automatisierungspyramide am Beispiel des automobilen Karosseriebaus in Anlehnung an [104] und [9]

Aus steuerungstechnischer Sicht stellen SPS und Roboter das Gehirn des Karosseriebaus dar und umfassen den Großteil der Prozesslogik. In ihrer Rolle als übergeordnete Steuerung ist die SPS für die übergreifende Koordination der Fertigungsvorgänge zuständig. Zu diesen Aufgaben gehören beispielsweise die Steuerung des Materialflusses, Überwachung der Zugänge sowie Einlegestationen und die Kommunikation mit den Leitstandssystemen. Über ein Bedienpanel stellt die SPS zudem die zentrale Schnittstelle zum Bediener dar und ermöglicht elementare Funktionen wie den Aufruf von Programmstart und -ende oder einer Status- bzw. Fehlerübersicht für den zu steuernden Produktionsbereich. An einer SPS sind in der Regel mehrere, oft eine zweistellige Zahl Roboter angebunden, die von dieser Befehle erhalten. Dazu gehören grundsätzliche Fahr- und Bewegungsfreigaben, das Starten von Fertigungssequenzen in Form von Roboterprogrammen oder die Statuserfassung und das Fehlerhandling aus Zellensicht. Die Kommunikation erfolgt mittels analoger und digitaler Ein- und Ausgänge über ein Feldbussystem.

Im Gegensatz zur SPS ist die zentrale Aufgabe eines Robotercontrollers nicht die übergreifende Steuerung der Abläufe in einer Anlagenzelle des Karosseriebaus, sondern die Ausführung einzelner Fertigungsprozesse wie Punktschweißen, Bördeln oder Kleben. Darunter ist in erster Linie die Manipulation von Bauteilen oder Werkzeugen und die Kommunikation des Roboters mit dem erforderlichen Equipment wie Schweißzangen, Klebeanlagen, Lichtschranken und ähnlichem zu verstehen. Diese Aktorik und Sensorik kann entweder direkt durch den Roboter angesprochen werden oder indirekt durch eine eigene Prozesssteuerung, die mit dem Roboter

kommuniziert. Diese werden häufig eingesetzt, wenn hohes prozessspezifisches Know-how zur Regelung der Aktorik erforderlich ist, z. B. bei der Kontrolle des Schweißstromes. Darüber hinaus wertet die Robotersteuerung Prozessdaten aus, meldet Statusinformationen an übergreifende Systeme und ist für das Fehlerhandling zuständig. Ein Großteil des fertigungstechnischen Prozess-Know-hows befindet sich im Roboterprogramm und ist stark abhängig vom eingesetzten Equipment sowie dem herzustellenden Produkt. Zwar bieten Roboterhersteller Softwarepakete für diese fertigungstechnischen Abläufe an, diese sind aber angesichts der erwähnten Abhängigkeiten nicht oder nur mit großer Anpassung für einen Automobilhersteller verwendbar.

Die Entwicklung der notwendigen Software erfolgt dementsprechend unternehmensspezifisch und stellt einen Großteil notwendiger Integrationsleistungen für Robotersysteme im automobilen Karosseriebau dar. Bei der BMW AG wird von einem sogenannten Roboterapplikationsstandard gesprochen. Dieser stellt unternehmensübergreifend die zentrale SW-Logik für Roboteranwendungen zur Verfügung, muss allerdings aufgrund der proprietären Systemlandschaft mit jedem Roboterlieferanten individuell entwickelt werden und ist somit ein Flaschenhals für flexible und herstellerunabhängige Roboteranwendungen im Karosseriebau. Es bietet sich folglich an, die Forschungsfrage am Beispiel eines Roboterapplikationsstandards zu konkretisieren und sowohl prozessuale, steuerungstechnische und anwenderbezogene Anforderungen hieraus abzuleiten. Im Folgenden wird daher konkreter auf Roboterapplikationen in der Automobilproduktion sowie dem zugehörigen Entwicklungsprozess und den Abhängigkeiten zu Roboterlieferanten eingegangen. Dieser Sachverhalt wird derzeit in keiner gängigen Literatur detaillierter behandelt und soll daher anhand des Fallbeispiels der BMW AG dargestellt und konkretisiert werden. Auf Basis von unternehmensinterner Dokumentation und Expertenbefragungen wird eine Beschreibung von Karosseriebauapplikationen durchgeführt und die aus der Herstellerabhängigkeit resultierenden Nachteile für den Entwicklungs- und Produktionsprozess abgeleitet.

4.3. Roboterapplikationsstandardisierung im Karosseriebau

Ein großer Vorteil beim Einsatz von Robotersystemen im Produktionsumfeld ist die flexible Einsetzbarkeit für eine Vielzahl an unterschiedlichen Anwendungen. Eine gute Übersicht zu industriellen Roboterapplikationen bietet die Klassifizierung der International Federation of Robotics. Diese ist in Tabelle 1 aufgelistet und um den Einsatzort in den Produktionsabschnitten der automobilen Kernfertigung bei der BMW AG ergänzt. Es ist ersichtlich, dass die Technologie Karosseriebau als einziger Bereich sämtliche Applikationskategorien abdeckt und mit vierzehn Anwendungsfeldern mehr als die Hälfte aller aufgelisteten Anwendungen beheimatet. Des Weiteren verbergen sich in den Feldern „Sonstige Schweißprozesse“ (2.4.) und „Befestigen, Einpressen“ (4.1.) noch eine Vielzahl weiterer Applikationen, die durch eine hohe Prozesskomplexität gekennzeichnet sind. Dies unterstreicht, dass der Anwendungsfall Karosseriebau eine geeignete Wahl für die Untersuchung industrierobotertechnischer Fragestellungen ist, da dieser eine breite Abdeckung industrieller Roboterapplikationen besitzt.

Tabelle 1 – Industrieroboterapplikationen nach der IFR-Klassifikation [6] und deren Anwendung in der Automobilproduktion

			Einsatz in			
Nr	**Bezeichnung**	**Kommentar**	**Presswerk**	**Karosseriebau**	**Lackiererei**	**Montage**
1.	**Handling/Maschinenbeschickung**					
1.1.	Handling für Metallgußteile					
1.2.	Handling für Spritzgußteile					
1.3.	Handling für Pressen, Schmieden, Biegen		x			
1.4.	Handling für Werkzeugmaschinen					
1.5.	Maschinenbestückung für sonstige Prozesse					
1.6.	Handling für Messen, Untersuchen, Testen		x	x		x
1.7.	Handling für Palletierung		x	x		x
1.8.	Handling für Packetierung, Pick & Place			x		x
1.9.	Sonstinge Handlingoperationen			x		
2.	**Schweißen und Löten**					
2.1.	Lichtbogenschweißen			x		
2.2.	Punktschweißen			x		
2.3.	Laserschweißen			x		
2.4.	Sonstige Schweißprozesse	Ultraschall-, Gas-, Bolzensch.		x		
2.5.	Löten			x		
3.	**Dosieren**					
3.1.	Lackieren, Emalieren				x	
3.2.	Kleben und Dichten			x		
3.3.	Sonstige Dosier- und Sprühanwendungen			x	x	
4.	**Montage/Demontage**					
4.1.	Befestigen, Einpressen	Schrauben, Clinchen		x		x
4.2.	Montieren, Befestigen, Einsetzen	auch Vorpositionierung		x		x
4.3.	Demontieren					
4.4.	Sonstige Montageprozesse					x
5.	**Sonstiges**					
5.1.	Reinraum für FPD (Flat Panel Display)					
5.2.	Reinraum für Halbleiter					
5.3.	Reinraum für sonstige Anwendungen					
5.4.	Sonstige Anwendungen			x		

Jede dieser Roboterapplikationen muss steuerungstechnisch mit entsprechender Hard- und Software umgesetzt werden. In der Automobilproduktion erfolgt dies nicht durch anlagenindividuelle Programmierung im Werk, sondern unterliegt einem unternehmensübergreifenden Standardisierungs- und Integrationsprozess, in dem Applikationsstandards zunächst zentral entwickelt werden und die Basis für die Realisierung der Automatisierungslösungen in den Produktionsstätten sind.

Im Kern definiert ein Applikationsstandard welche Komponenten, sowohl hard- als auch softwareseitig, für eine Automatisierungsanwendung einzusetzen sind. Zudem sind Softwarebausteine und Schnittstellendefinitionen enthalten, die bereits zentrale Anwendungslogiken für den Fertigungsvorgang und die Peripherie- bzw. Leitstands-kommunikation enthalten und somit das unternehmensspezifische Prozesswissen abbilden.

Dabei handelt es sich allerdings nicht um die reine Entwicklung von Standardroboterprogrammen auf Anwenderebene, sondern um eine tiefe und umfassende Verankerung im herstellerspezifischen Robotersystem. Diese umfassen kundenspezifische Programmierbefehle, Bedien- und Visualisierungskonzepte sowie Fehlerhandhabungsroutinen. Zum Teil kann diese kundenspezifische Anpassung bis auf die Ebene der Trajektoriencharakteristik von Roboterbahnen gehen.

Die Entwicklung und Definition dieser Applikationsstandards erfolgt bei der BMW AG durch eine Zentralstelle der Planung in direkter Zusammenarbeit mit den Roboterherstellern und Komponentenlieferanten. Zeitlich richtet sich der Prozess an den jeweiligen Fahrzeugprojekten aus, um einen zeitgerechten Anlauf mit aktueller Steuerungstechnik sicherzustellen. Die Hauptverantwortung obliegt den Roboter- und Steuerungstechnikexperten, die in Zusammenarbeit mit den Fachstellen der Fertigungsprozesse, Werksinstandhaltung und den jeweiligen Lieferanten ca. vier Jahre vor Produktionsbeginn mit dem Entwicklungsprozess beginnen. Die Gültigkeit eines Applikationsstandards orientiert sich ebenso am Produktlebenszyklus eines Fahrzeugprojekts und beträgt daher mindestens sieben Jahre, kann aber infolge von Fahrzeugderivaten[9] auch eine zweistellige Lebensdauer erreichen. Steuerungstechnische Anpassungen und Erweiterungen eines bestehenden Standards erfolgen nach Serienstart nur in geringem Maße und fokussieren sich auf Optimierungsschleifen. Aufgrund der Proprietarität der Robotersysteme muss für jeden Roboterlieferant ein eigener Applikationsstandard redundant entwickelt werden. Eine Wiederverwendung bestehender Softwaremodule ist nicht möglich. Selbst bei einem Steuerungsgenerationswechsel eines einzelnen Herstellers sind die Änderungen häufig so hoch, dass auch hier nur eine eingeschränkte Wiederverwendbarkeit existiert.

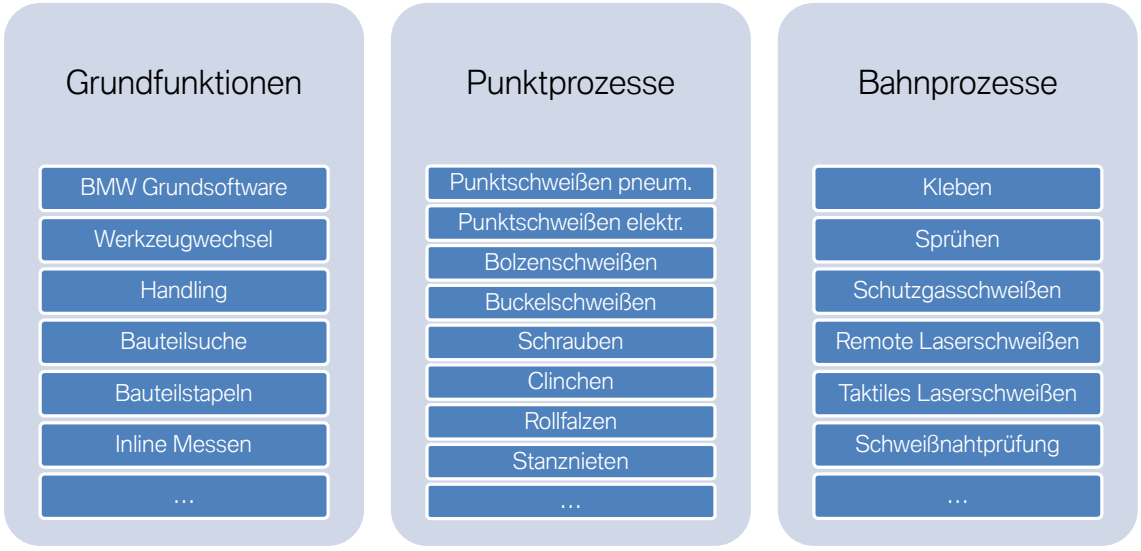


Abbildung 13 – Standardisierte Roboterapplikationen im Karosseriebau der BMW AG

Die bei der BMW AG vorhandenen Roboterapplikationsstandards orientieren sich an den Fertigungsanwendungen und decken sich im Kern mit der Kategorisierung des IFR. Es existieren jedoch zudem Applikationssoftwares wie „BMW Grundsoftware“ und „Werkzeugwechsel“ Standardmodule, die unabhängig von einzelnen Fertigungsprozessen übergreifend zur Verfügung stehen und Funktionen für sämtliche Applikationen zur Verfügung stellen. Darüber

[9] Unter Fahrzeugderivaten sind von einem Basismodell abgeleitete Modelle zu verstehen.

hinaus sind Roboterstandards häufig granularer als in der IFR-Einteilung definiert, um relevante fertigungstechnische und lieferantenseitige Unterschiede besser abdecken zu können, z. B. untergliedert sich Punktschweißen in eine pneumatisch und eine servoelektrische Variante. Eine Übersicht der bei BMW identifizierten und standardisierten Roboterapplikationen ist in Abbildung 13 zu sehen.

4.4. Integrationsprozess von Roboteranwendungen

Die Verwendung eines Applikationsstandards beeinflusst die Integration von Roboteranwendungen in den Karosseriebau erheblich. Die Entwicklung und Programmierung der konkreten Roboteranwendung erfolgt nicht individuell durch Anlagenbauer oder Roboterprogrammierer, sondern auf Basis der vorgegebenen Programmier- und Schnittstellenstandards und der vorentwickelten Softwarepakete. Auf diese Weise wird der Integrationsprozess zweigeteilt – in einen übergreifenden Standardisierungsprozess (Phase I) und den werk- und anlagebezogenen Inbetriebnahmeprozess (Phase II). Während Phase I bereits im vorangegangenen Abschnitt beschrieben wurde, sollen im Folgenden, aufbauend auf dem Applikationsstandard, die zweite Phase und die beteiligten Stakeholder detaillierter erläutert werden.

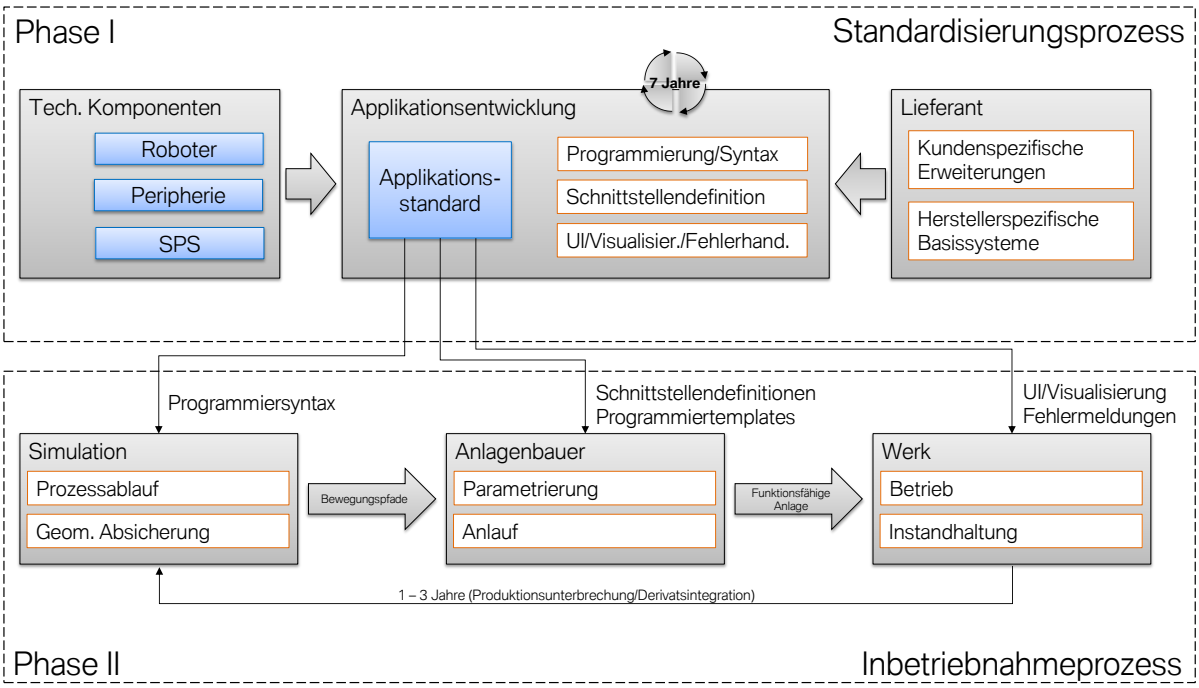


Abbildung 14 – Integrationsprozess von Roboterapplikationen im Karosseriebau

Zu Beginn der Inbetriebnahmephase wird im Rahmen der Anlagenplanung eine Roboteranlage digital in Betrieb genommen, um den Prozessablauf zu optimieren, die geometrische Realisierbarkeit zu gewährleisten und die Bewegungspfade der Roboter zu bestimmen. Da Schnittstellen und Peripherie bereits definiert und über Softwarepakte steuerungstechnisch realisiert sind, muss keine digitale Inbetriebnahme der Steuerungstechnik oder die Entwicklung komplexer Anwendungslogiken erfolgen und es kann sich auf die Bewegungsabläufe und Pfade der Roboter fokussiert werden. Als Ergebnis liegt der konkrete physische Aufbau der Roboterzelle inklusive der Roboterprogramme vor. Die Erweiterungen und Modifikationen des herstellerspezifischen Programmierbefehlssatzes müssen somit bereits auf Simulationsebene berücksichtig werden, um die korrekte Programmausleitung zu gewährleisten.

Im nachfolgenden Schritt, dem Anlagenaufbau durch den Systemintegrator, wird neben der physischen Installation auch die steuerungstechnische Inbetriebnahme durchgeführt. Zentrale Aufgabe ist hierbei das Einbinden der relevanten Roboterprogrammbausteine, die Konfiguration der I/O-Schnittstellen und Testläufe der Robotersysteme mit angeschlossener Peripherie. Bei der Softwareeinbindung und Kommunikationskonfiguration wird der Roboterprogrammierer durch den Applikationsstandard toolseitig unterstützt, so dass ein manueller Konfigurationsprozess in vielen Bereichen nicht notwendig ist. Der größte Programmieraufwand liegt in der Anpassung der Roboterbahnen an den realen Zellenaufbau, da dieser häufig kleine Differenzen zur Simulationswelt aufweist und geringfügige Programmkorrekturen notwendig sind. Es handelt sich dabei allerdings lediglich um Teach-Vorgänge, also dem Anlernen bzw. Korrigieren von Bahnpunkten und nicht um komplexe Programmänderungen.

Nach abgeschlossener steuerungstechnischer Inbetriebnahme übernimmt die Instandhaltung im Werk den Betrieb und die Wartung der Robotersysteme. Die Programmiertätigkeit beschränkt sich diesbezüglich i. d. R. auf kleine Optimierungsanpassungen zu Produktionsbeginn. Anschließend liegt der Fokus auf der Bedienung und der Statusüberwachung, um Output und Qualität der Großserienproduktion zu gewährleisten. Wobei die Robotersysteme nicht einzeln, sondern über die Zellensteuerung bzw. dem Leitstand überwacht werden. Aufgrund der langen Lebensdauer der Produkte und der stabilen Produktgestaltung sind größere Programmanpassungen in den bestehenden Robotersystemen nicht mehr notwendig bzw. erfolgen erst bei der Integration neuer Derivate oder zu dem LCI[10] eines Fahrzeuges. Diese Anpassungen erfolgen in längeren Phasen der Produktions-unterbrechung und prozessual analog zu einem Fahrzeugneuanlauf inklusive der Simulation relevanter Anlagenveränderungen. Aus Sicht der Standardisierung werden die Robotersysteme insbesondere in Bezug auf die Konzepte und Gestaltung von Bedienung, Visualisierung und Fehlerhandling angepasst, um den Werksbetrieb anwender- und produktionssystemgerecht zu gestalten.

Zusammenfassend lässt sich festhalten, dass durch die Applikationsstandardisierung auf Basis der proprietären Steuerungssysteme ein Großteil der Integrations- und Programmierarbeit von Robotern zentralisiert, automatisiert und anwendergerecht gestaltet wird. Zudem wird durch Aufgabenteilung und Spezialisierung der Integrationsprozess selbst so definiert, dass die unterschiedlichen Stakeholder sich auf ihr Aufgabengebiet und ihre Expertise fokussieren können und die Menge an nicht wertschöpfenden Arbeiten reduziert wird. Die dadurch entstehenden Vor- und Nachteile sollen im Folgenden kurz erläutert werden.

[10] Life-Cycle-Impulse: Maßnahme zur Beibehaltung, ggf. Steigerung der Produktattraktivität im fortgeschrittenen Vermarktungszeitraum, z. B. Facelift, Zusatz-/erweiterte Serienausstattung.

Vorteile der Applikationsstandardisierung

Der Hauptvorteil des standardisierten Einsatzes von Software und Betriebsmitteln ist, einen hohen Anteil der Entwicklungsleistung von Roboterapplikationen und Programmen nur einmal und werkeübergreifend durchzuführen, statt individuell für einzelne Anlagen, Produktionslinien oder Werke. Dies bietet neben kostenseitigen Vorteilen einen enormen Effizienzgewinn in Bezug auf den Inbetriebnahmeaufwand einer Anlage, da die bestehenden Softwarekomponenten den Systemintegratoren vorgegeben werden und eine individuelle Entwicklung nicht notwendig ist. Zusätzlich erhöht die einheitliche Software die Qualität der Produktionsprozesse, da mehr Ressourcen und Expertise einfließen können als bei einer dezentralisierten Entwicklung. Fehlerverbesserungen und Prozessoptimierungen können in kurzer Zeit allen Systemen und nicht nur einzelnen Anlagen zur Verfügung gestellt werden. Die Robotersysteme sind zudem werkeübergreifend vergleichbar, Betrieb und Wartung standardisiert und Erfahrungen können über den lokalen Standort hinaus ausgetauscht werden. Dies minimiert Schulungsaufwände und vereinfacht die Handhabung der Systeme, da zentrale Elemente der Bedienung und Überwachung durch den Anwenderstandard vorgegeben sind und nicht alleine auf lieferantenspezifischen Konzepten beruhen.

Nachteile der Applikationsstandardisierung

Als Nachteil ist der insgesamt verlängerte Entwicklungsprozess und die Komplexität der Software zu nennen, da diese nicht nur für eine einzelne Anwendung in einer Anlage ausgelegt wird, sondern als eine übergreifende Lösung, die den multiplen Anforderungen von unterschiedlichen Anwendungsfällen und Anwendern gerecht werden muss. Darüber hinaus sind mit der Festlegung auf ein Standardgerüst an Robotersystemen und Komponenten zwangsläufig Kompromisse erforderlich und eine geringere Flexibilität verbunden. Dies hat unter Umständen zur Folge, dass die technisch beste Option für einen einzelnen Anwendungsfall zu Gunsten einer gesamtunternehmensoptimierten Lösung weichen muss. Auch der zügige Wechsel von noch nicht befähigten Zulieferern oder das Einbinden neuer Peripheriekomponenten und Prozessapplikationen ist aufwändiger und langwieriger.

4.5. Beeinträchtigungen durch herstellerspezifische Robotersysteme

Eine zentrale Ursache für die Nachteile in der Applikationsstandardisierung stellen die proprietären Steuerungssysteme der Roboter dar, da die unternehmensspezifischen Anpassungen immer in den jeweiligen herstellerspezifischen Entwicklungsumgebungen und Softwareökosystemen der Roboterlieferanten erfolgen müssen. Faktisch verdoppelt sich infolge der mangelnden Interoperabilität der Systeme der Aufwand der Applikationsstandardisierung bereits bei dem Einsatz von zwei statt einem Roboterhersteller, da sämtliche Anpassungen redundant in beiden Systemen der Lieferanten durchzuführen sind. Dadurch ist eine hohe Abhängigkeit zum jeweiligen Roboterlieferanten gegeben und ein einfacher Wechsel vom Robotersystem oder den eingebundenen Komponenten ist nicht möglich. Oft ist selbst die Wiederverwendung eines entwickelten Standards von Roboter- zu Robotergeneration eines Herstellers aufgrund von Systemanpassung nicht möglich. Zudem ist die Gestaltungsfreiheit, insbesondere bei Anwender- und Bedienkonzepten, durch die Robotersysteme eingeschränkt und herstellerübergreifend identische Standards sind auch mit sehr hohem Aufwand nicht realisierbar.

Erhöht bzw. erschwert werden die beschriebenen Aufwände und Problemfelder zusätzlich dadurch, dass in der Automobilindustrie üblicherweise Single Sourcing, also dem Bezug einer Ware von lediglich einem einzigen Lieferanten, aus strategischer Sicht häufig vermieden wird,

beispielsweise um weltweite Liefersicherheit zu gewährleisten, Abhängigkeiten zu verringern und leistungs- und kostenseitigen Wettbewerb unter den Zulieferern zu fördern. Dies erfordert jedoch einen redundanten Standardisierungsprozess mit unterschiedlichen Lieferanten. Außerdem wird die Innovationsgeschwindigkeit reduziert, da neue Fertigungsapplikationen mit mehreren Lieferanten integriert werden müssen, um eine werkeübergreifende Einsetzbarkeit sicherzustellen.

Mehraufwände entstehen ebenfalls in der Anlagenplanung. Zwar sind die eingesetzten Tools bereits grundsätzlich darauf ausgelegt, roboterherstellerunabhängig über Postprocessing die spezifischen Systeme zu bedienen, allerdings erfordern die anwenderspezifischen Erweiterungen der Programmiersyntax zusätzliche Modifikationen in den Simulationssystemen, um die Ausleitung des proprietären Programmcodes weiterhin zu gewährleisen.

Aus Sicht des Betreibers führt diese Mehrlieferantenstrategie ebenso zu erhöhtem Aufwand, beispielsweise für Schulungen, da durch die eingeschränkte Modifizierbarkeit der Robotersysteme ein übergreifend vergleichbares Bedienkonzept nicht realisierbar ist und mit jeder neuen Fahrzeugintegration der eingesetzte Roboterlieferant variieren kann. Teilweise werden sogar nebeneinanderliegende Karosseriebaulinien eines Standorts mit unterschiedlichen Roboterherstellern ausgestattet.

4.6. Schlussfolgerung und Anforderungskonkretisierung der Forschungsfragen

Betrachtet man den beschriebenen Standardisierungs- und Integrationsprozess, zeigt sich, dass sowohl die Art und Weise der Entwicklung und Programmierung von Roboterapplikationen als auch die damit verbundenen Anforderungen von den im Stand der Wissenschaft und Technik beschriebenen Arbeiten und Anwendungsfällen differieren.

Beispielsweise ist die schnelle und einfache Programmierung von Roboterapplikationen oder die automatische Integration von Peripheriekomponenten wünschenswert und hilfreich, doch ist die Notwendigkeit durch das Standardisierungsvorgehen sowie die Produktionsrandbedingungen nicht gegeben oder zumindest wesentlich geringer als in Betrieben kleiner und mittelständischer Größe. So werden komplexe Programmieraufgaben von Roboter- und Fertigungstechnikexperten statt Anwendern im Werk durchgeführt. Ebenso wird die aufwändige Konfiguration von Peripheriekomponenten vorab realisiert und durch unternehmensinterne Tools und Methoden weitestgehend automatisiert. Des Weiteren ist festzuhalten, dass von einem Peripheriekomponententyp (z. B. einer Schweißsteuerung) nicht beliebig viele unterschiedliche Varianten oder Hersteller zum Einsatz kommen. Vielmehr wird versucht, dies auf ein notwendiges Minimum für eine Standardarchitektur zu reduzieren, um Einkaufsskaleneffekte zu realisieren und die technische Handhabbarkeit zu gewährleisten. Plug & Produce-Konzepte, die einen flexiblen und schnellen Tausch von Peripheriekomponenten unterschiedlicher Varianten oder Lieferanten ermöglichen, bieten in diesem Fall weniger Nutzen, da die Anzahl der eingesetzten Komponenten stark begrenzt und vorgegeben ist. Zudem wird der initiale Einmalaufwand bei der Applikationsstandardisierung anlagen- und werkeübergreifend zusammengefasst und so die notwendige Infrastrukturarbeit auf Anlagenebene stark automatisiert und reduziert. Sie spielt dementsprechend in der operativen Anlagenintegration vor Ort keine nennenswerte Rolle.

Durch die arbeitsteilige Programmierung der Roboteranwendungen und die langen Produktlaufzeiten ergibt sich ein anderes Anforderungsbild als beispielsweise in den Arbeiten von Perzylo et al. [63], Krug [9] oder Naumann et al. [64]. Es existiert kein Bedarf, komplette

Roboterprogramme zügig auf Anwenderebene zu entwickeln, um neue Produkte flexibel in eine Fertigung zu integrieren oder für Losgröße Eins zu produzieren. Natürlich besteht auch in der Automobilindustrie der Wunsch, das Produktionsportfolio rasch und aufwandsarm anpassen zu können, allerdings ist die Roboterprogrammierung hierbei nicht der alleinig zu überwindende Flaschenhals. Themen wie die Abkehr von der klassischen Linienfertigung oder die Realisierung von hochflexiblen Werkzeug- und Spannvorrichtungskonzepten stellen ebenso hohe Hürden dar. Nicht zu vernachlässigen sind hierbei die zahlreichen produktbezogenen Themen, die eine stark reduzierte Losgröße mit sich bringen würden, beispielsweise die erhöhten Aufwände und Komplexitäten in Entwicklung, Absicherung und Einkauf. Außerdem wird durch die applikationsspezifische Standardisierung von Programmbausteinen, Programmierbefehlen und Bedienkonzepten bereits der Anforderung Rechnung getragen, Anwendern im Werk eine möglichst einfache und produkt- bzw. aufgabenbezogene Bedienung und Programmierung von Robotersystemen zu ermöglichen, zumindest insoweit dies unter den gegebenen Randbedingungen der Herstellersysteme realisierbar ist.

Vergleicht man den beschriebenen Integrationsprozess mit den im Stand von Wissenschaft und Technik aufgeführten Arbeiten, zeigt sich, dass die dargestellten Problemstellungen durch die angewandte Methodik bereits gemildert bzw. gelöst sind (z. B. Plug & Produce-Werkzeugkonfiguration) oder angesichts anderer Randbedingungen für den zu untersuchenden Anwendungsfall „automobiler Karosseriebau“ nicht existieren (z. B. Reprogrammierung der Anlage aufgrund eines schnell wechselnden Produktionsprogramms). Es wird jedoch deutlich, dass die Systemproprietarität erhebliche Auswirkungen auf die einzelnen Aufgaben im Roboterintegrationsprozess hat und folglich die Beantwortung der Forschungsfragen einen relevanten Mehrwert für die Roboteranwendungen in der Automobilproduktion besitzt. Analog zu den unterschiedlichen Tätigkeiten der beteiligten Stakeholder und den mit der Herstellerabhängigkeit verbundenen Problemen unterscheiden sich die Schwerpunkte und Anforderungen an eine herstellerneutrale Lösung. Beispielsweise ist aus Standardisierungssicht vor allem die Möglichkeit der universellen Programmierung von Applikationsprogrammen auf Expertenebene und einer freien Gestaltung der unternehmenseigenen Anforderungen von besonderer Bedeutung, während aus Simulationssicht insbesondere eine einheitliche Robotersprachbefehlsspezifikation wünschenswert wäre, um die Aufwände in Bezug auf proprietäre Roboterprogrammiersprachen und spezifische Erweiterungen zu reduzieren. Aus Sicht der produktionsnahen Tätigkeiten im Werk und der Systemintegration kommt vor allem der einfachen, anwendungsbezogenen, universellen Parametrierung von Programmen und Einstellungen eine wichtige Rolle zu.

Abbildung 16 fasst anhand des beschriebenen Integrationsprozesses die einzelnen Nutzergruppen, deren Tätigkeit und die Implikationen durch die Proprietarität der Robotersysteme übergreifend zusammen. Zudem sind die jeweiligen Anforderungen der Stakeholder an eine herstellerneutrale Integrationslösung beschrieben und als Randbedingungen für die beantwortende Fragestellung in Bezug auf Externalisierung, Programmierung und prototypische Umsetzung herunter gebrochen.

Prozessschritte	Applikations-standardisierung	Anlagenplanung	Systemintegration	Betrieb
Stakeholder	• Roboterexperten • Steuerungstechniker • Fertigungstechniker	• Anlagenplaner • Simulationsexperten	• Anlageninbetriebnehmer • Roboterprogrammierer	• Roboterprogrammierer • Anlagen-/Maschinenbediener
Tätigkeit	• Spezifikation und Entwicklung der Applikationsprogramme, Programmierbefehle und Bedienkonzepte	• Entwicklung des Anwenderprogramms (insb. Roboterbahnen) auf Basis der Applikationsbefehle und -programme des Standards	• Konfiguration der Peripherie und Aufspielen der Programmbausteine • Reteachen der Punkte bei Differenzen Simulation/Aufbau	• Überwachen und Optimieren der Fertigungsprozesse • Reteachen von Punkten, Anpassung von Applikationsparametern
Implikationen durch Herstellerabhängigkeit – Merkmale	• Redundante Programmierung der gleichen Applikationsprogramme • Erweiterung der proprietären Roboterprogrammiersprachen mit unternehmensspezifischen Befehlssätzen • Eingeschränkte Gestaltungsfreiheit • Keine Wiederverwendung bestehender Standards • Wechsel der Lieferanten eines Standards aufwändig und langwierig • Langsame Integration von Prozessinnovationen, da Applikationen mit jedem Roboterlieferanten entwickelt werden müssen • Tools und Methoden für Folgeprozesse müssen spezifisch für Herstellersysteme entwickelt werden (z.B. Peripheriekonfiguration)	• Implementierung der unternehmensspezifischen Erweiterungen in Simulationstools • Ansonsten relativ gering, da eingesetzte Tools bereits für verschiedene Roboterhersteller ausgelegt sind	• Konfiguration der Peripherie, Programmsetup und Teachen von Bahnpunkten erfolgt in proprietären Systemen • Roboterprogrammierer sind meist nur im Umgang mit ein oder zwei Roboterherstellern erfahren bzw. spezialisiert => Erhöhter Aufwand aufgrund von Einarbeitung oder Schulungen bei der Verwendung anderer Hersteller • Aufwände in der Inbetriebnahme werden jedoch durch Tools und Methoden des Standardisierungs- und Integrationsprozesses reduziert => (teil-)automatisierte Inbetriebnahme	• Programmanpassung und Überwachung erfolgt in proprietären Systemen • Schulungen notwendig, um Mitarbeiter für den jeweilig eingesetzten Robotertyp eines Werks/Linie zu befähigen • Unternehmensspezifische Anpassungen durch Applikationsstandards gleichen Systeme teilweise herstellerübergreifend an
Implikationen durch Herstellerabhängigkeit – Grad der Betroffenheit	• Hoch	• Gering	• Gering bis mittel	• Hoch
Anforderung an eine herstellerneutrale Lösung	• Entwicklung der Applikationslogik nur einmal für unterschiedliche Robotersysteme • Entwicklungssystem kann/soll Expertensystem sein • Gestaltungsfreiheit in der anwendungsbezogenen Umsetzung der Standards • Fertigungsqualität darf nicht schlechter werden	• Einheitliche Befehlsdefinition für unterschiedliche Robotersysteme, um aufwändige Anpassungen zu vermeiden	• Inbetriebnahmeprozedur der Peripherie unabhängig vom eingesetzten Robotersystem	• Programmanpassung und -überwachung universell und unabhängig vom eingesetzten Robotersystem • Bedienung möglichst anwender- und anwendungsbezogen • Bedienung möglichst einfach und verständlich • Fertigungsqualität und -output darf nicht schlechter werden

Abgeleitet Anforderungen an Forschungsfragen

Externalisierung

- Bestmögliche Abstraktion von Applikationslogik
- Möglichkeit der Gestaltungsfreiheit
- Umsetzbarkeit mit Lieferanten
- Leistungsfähigkeit, um Fertigungsqualität zu realisieren
  - Roboterperformance
  - Peripherieperfomance

Programmierung

- Entwicklung/Programmierung/Bedienung herstellerunabhängig und anwenderbezogen für Roboterexperten und -bediener
- Abdecken der spezifischen Automobilerweiterungen
- Universelle Integrierbarkeit in Simulationssoftware

Prototyp

- Synthese der Ergebnisse der Forschungsfragen der Roboterabstraktion durch Externalisierung und Programmierung
- Universelle Abstraktionslogik von Karosseriebauapplik.
- Einsatz mehrerer Robotersysteme untersch. Hersteller
- Universelle Peripheriekonfiguration

Abbildung 16 – Implikationen durch Herstellerabhängigkeit in der Roboterintegration und sich daraus ableitende Anforderungen an eine herstellerneutrale Lösung

5. Herstellerunabhängige Steuerung von Robotersystemen

Im Rahmen der Erläuterungen zum Stand der Wissenschaft und Technik wurde aufgezeigt, dass die externe Ansteuerung von Robotersystemen eine häufig verwendete Methode in Forschung und Industrie ist, um Steuerungsintelligenzen von proprietären Systemen zu lösen und auf alternative, übergeordnete Steuerungssysteme zu überführen. Gründe für die bewusste Externalisierung von Logiken sind meist die Sicherstellung von Hardware- und Herstellerunabhängigkeit oder die Notwendigkeit, aufgrund von mangelnder technischer Umsetzbarkeit von Anwendungsanforderungen, der Applikationsumsetzung im konventionellen Steuerungssystem zu entgehen.

Dabei variieren Art und Weise der externen Ansteuerung erheblich. Zentrale Einflussfaktoren sind die zur Verfügung stehenden Schnittstellen der Robotercontroller, die Leistungsfähigkeit der externen Steuerungssysteme und die sich aus der Anwendung ergebenden Anforderungen.

Ebenfalls variiert der Typ des externen Steuerungssystems stark. Dieser reicht von klassischen Steuerungskomponenten wie speicherprogrammierbaren Steuerungen zur Einbindung in ein traditionelles Automatisierungsumfeld bis hin zu Standard-PCs mit Windows- und Linux-Systemen, die durch standardisierte Hard- und Software eine hohes Maß an Flexibilität und Funktionalität zur Verfügung stellen. Auf diese Weise ist es auch möglich, Geräte aus der Consumer Electronic in Automatisierungssysteme einzubinden und Visualisierungs- oder Bedienkonzepte zu realisieren, die mit einer proprietären Robotersteuerung nicht möglich wären.

Die gewonnene Flexibilität ist allerdings auch mit stärkerer Verantwortung auf Umsetzungsseite verbunden. Je nach Anbindung des Roboters müssen teilweise sehr komplexe Robotergrundfunktionen in das externe System implementiert werden, die ohne spezifische Konstruktionskenntnisse des Manipulators nicht optimal lösbar sind. Das externe Steuerungssystem muss zudem gegebenenfalls hohe Echtzeit- und Zuverlässigkeits-anforderungen erfüllen, um einen robusten und korrekten Betrieb zu ermöglich. Auch muss die externe Ansteuerbarkeit des Roboters überhaupt erst ermöglicht werden, entweder durch die inhärente Funktionalität des Robotersystems oder durch die eigene Implementierung von passenden Schnittstellen im Controller.

Diese notwendigen Aufwände haben sich in den letzten Jahren durch technologische Entwicklungen in Wissenschaft und Industrie massiv verringert. Zum einen haben die in Kapitel 2.4 erwähnten Forschungsprojekte dazu beigetragen, notwendige Methoden der Steuer- und Regelung für Robotersysteme offenzulegen und als wiederverwendbare Open-Source-Bibliotheken der Allgemeinheit zur Verfügung zu stellen. Zum anderem ist dabei eine Vielzahl an Schnittstellenimplementierungen für Robotersteuerungen entstanden, die eine Externalisierung von Steuerungslogiken ermöglichen.

Neben diesen Forschungsarbeiten bieten auch die Roboterhersteller selbst zunehmend Lösungen an, um mit geringerem Integrationsaufwand auf Endkundenseite eine externe Ansteuerung der Systeme zu realisieren. Anwendung finden diese häufig in Speziallösungen der Automatisierungstechnik, beispielsweise im Bereich der Werkzeugmaschinen oder bei sensorgeführten Anwendungen (z. B. Force-Control), allerdings ist die Verbreitung bisher begrenzt und die Schnittstellen unterscheiden sich in Protokoll und Integrationstiefe erheblich. Letztere reichen von der Stromregelungsebene der Motoren bis hin zu abgeschlossenen Bewegungsbefehlen auf Applikationsebene. Die damit verbundenen Integrationsaufwände auf Endkundenseite sowie der notwendige Informationsbedarf hinsichtlich Manipulatormechanik

unterscheiden sich stark und es ist kein einheitlicher Standard definiert, der Roboterhersteller und Endanwender sowohl in Schnittstellengestaltung als auch in der Auswahl und Implementierung der steuerungstechnischen Umsetzung unterstützt. Zudem wird von Herstellerseite das Angebot der externen Ansteuerung insbesondere für die Erschließung neuer Anwendungsfelder betrachtet und nicht für den flexiblen und interoperablen Betrieb der Robotersysteme an sich. Die Herausgabe der Schnittstellenfreigabe und Spezifikationen sowie der notwendigen Manipulatorinformationen erfolgt daher häufig nur restriktiv.

Zusammenfassend lässt sich festhalten: Es existieren vielfältige Möglichkeiten, Roboter extern anzusteuern und Herstellerabhängigkeit zu verringern. In Forschung und Industrie finden diese in unterschiedlichen Anwendungen Einsatz. Jedoch stellen die verwendeten und zur Verfügung stehenden Schnittstellen nicht den Fokus der betrachteten Arbeiten dar, sondern sind meist eine technische Notwendigkeit, um Anwendungsanforderungen praktikabel umzusetzen.

Es ist bisher ungeklärt, welche Methode für welche Anwendung aus welchem Grund eingesetzt werden soll. Es fehlt eine gesamtübersichtliche Darstellung, Charakterisierung, Einordnung und Bewertung der Integrationsmethoden. Es existiert keine Aussage über die Leistungsfähigkeit von extern betriebenen Industrierobotersystemen, ebenso wenig ein Abgleich mit den Anforderungen aus Sicht des Karosseriebaus. Eine systematische Beantwortung dieser Fragen ist notwendig, um Abstraktion durch Externalisierung aus organisatorischer und technischer Sicht korrekt zu bewerten, passende Konzepte zu entwickeln und diese im industriellen Einsatz zukünftig großflächig umsetzen zu können. Diese Fragestellungen sollen anhand des in Abbildung 17 dargestellten Vorgehens erarbeitet werden:

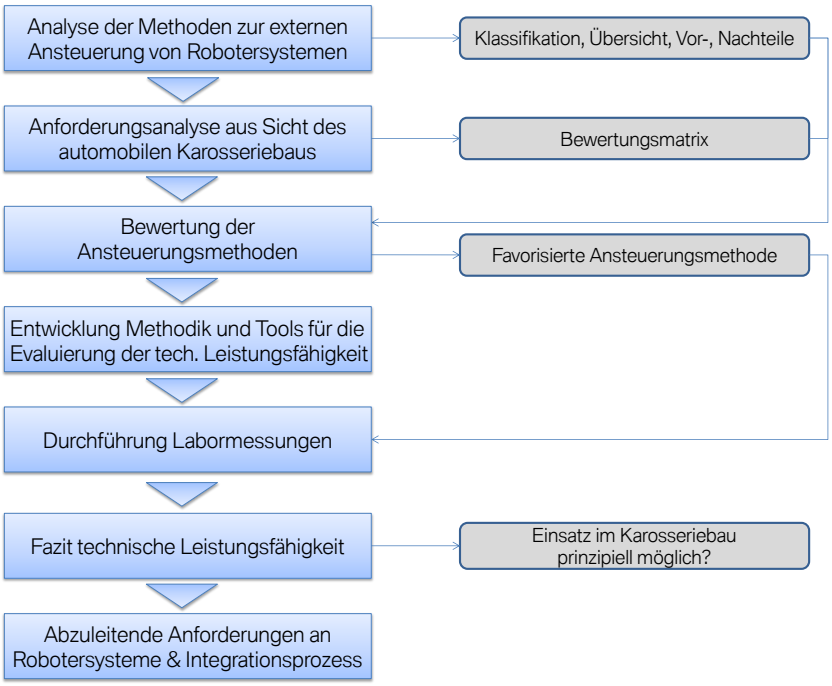


Abbildung 17 – Vorgehen zur Ermittlung einer geeigneten Methode zur herstellerunabhängigen Steuerung von Robotersystem im automobilen Karosseriebau

Im ersten Schritt werden die Verfahren der externen Ansteuerung von Robotersystemen hinsichtlich ihrer Eigenschaften charakterisiert, Vor- und Nachteile beschrieben und kategorisiert. Anschließend wird ein mehrdimensionales Bewertungsmodell erstellt, mit dem die unterschiedlichen Konzepte in Bezug auf Abstraktionsfähigkeit, Geschäftsmodell,

grundsätzliche Leistungsfähigkeit und Integrationsaufwand beurteilt und priorisiert werden, um das für den Karosseriebau geeignetste Verfahren zur herstellerunabhängigen Steuerung von Robotersystemen zu identifizieren. Im Anschluss wird die Fragestellung der konkreten technischen Leistungsfähigkeit überprüft. Hierfür musste zunächst eine geeignete Untersuchungs- und Auswertungsmethode definiert und die zugehörigen Werkzeuge entwickelt werden, um anschließend im Rahmen von Labormessungen die Erfüllbarkeit der Anforderungen des automobilen Karosseriebaus zu evaluieren.

Die zu erarbeitenden Erkenntnisse und identifizierten Bedürfnisse stellen zum einen die Basis dar, um eine Aussage hinsichtlich der Abstrahierbarkeit der Bewegungsansteuerung von Karosseriebaurobotern zu treffen und darauf aufbauend, die vollständige Umsetzung einer herstellerunabhängigen Applikationsstandardisierung für die Automobilproduktion durchzuführen (Kapitel 7.3). Zum anderen werden auf diese Weise Defizite identifiziert, die eine einfache und zügige herstellerübergreifende Integration von Robotersystemen in externe Steuerungen erschweren. Die sich daraus ableitenden Anforderungen und vorgeschlagenen Verbesserungsmaßnahmen werden kapitelabschließend in ein Plug & Produce-Schnittstellenkonzept überführt, welches der Vielfalt an unterschiedlichen Anwendungsanforderungen sowie Schnittstellenkonzepten Rechnung trägt und einen Vorschlag zur Standardisierung bietet, der eine einfache, flexible und automatisierte Integration von Robotersystemen in externe Steuerungen ermöglicht.

5.1. Methoden der externen Steuerung von Robotern

Im Stand der Wissenschaft und Technik sind bereits einige Forschungsarbeiten und Industrielösungen zur externen Ansteuerung von Robotersteuerungen dargestellt worden. Folgender Abschnitt soll die zugrunde liegenden Methoden detaillierter betrachten und klassifizieren, um Gemeinsamkeiten und Unterschiede darzustellen, Anforderungen an das externe Steuerungssystem zu identifizieren und eine übersichtliche Gesamtbetrachtung zu ermöglichen.

Zentrale Gemeinsamkeit der beschriebenen Lösungen ist, Funktionen bzw. Intelligenzen, die üblicherweise im Robotercontroller enthalten sind, zu externalisieren und so von der proprietären Steuerung zu lösen. Hauptunterschied stellt die Menge an Aufgaben dar, die anschließend von einer externen Steuerung anstatt des nativen Robotersystems durchgeführt wird. Zwar variiert diese natürlich mit den gegebenen Anforderungen und Zielen einer Anwendung. Jedoch ist aus steuerungstechnischer Sicht insbesondere die Ebene der Schnittstelle von zentraler Bedeutung, da sich aus dieser die zu übernehmende Aufgabenverantwortung für das externe Steuerungssystem definieren lässt, um die Gesamtfunktionalität zu erhalten.

Es wird deshalb vorgeschlagen, die Klassifizierung der externen Ansteuerung anhand der Architektur einer Robotersteuerung und der Integrationstiefe der externen Schnittstelle durchzuführen. Auf diese Weise ist es möglich, Aufgabenverteilung, Implementierungsaufwand, Verfügbarkeit der Schnittstelle, Anforderungen an das externe Steuerungssystem, Abstraktionsleistung der herstellerspezifischen Bestandteile und weitere Charakteristika systematisch zu beschreiben und die Unterschiede zu verdeutlichen (Abbildung 18). Mit steigender Integrationstiefe übernimmt die externe Steuerung zunehmend Aufgaben des nativen Robotercontrollers, der Abstraktionsgrad zum Herstellersystem steigt, ebenso wie die Anforderungen in Bezug auf die Leistungsfähigkeit des ausgelagerten Steuerungsanteils.

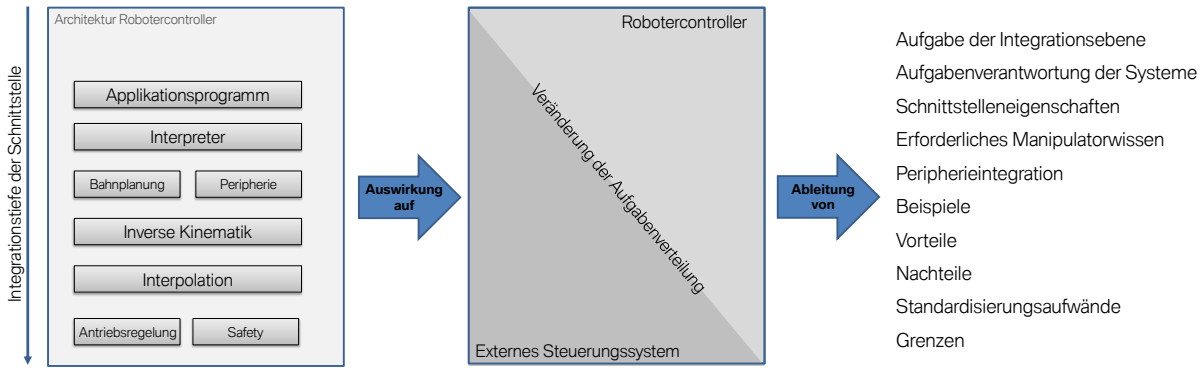


Abbildung 18 – Integrationstiefe von Roboterschnittstellen und daraus abzuleitende Eigenschaften

5.1.1. Funktionsanalyse anhand der Integrationstiefe

I - Applikationsprogramm

Das Applikationsprogramm enthält die zentrale Anwenderintelligenz innerhalb eines Roboterprogramms, da in diesem sämtliche aus Produktionssicht relevanten Manipulatorbewegungen und Schaltvorgänge definiert sind. Abstraktionskonzepte bzw. Schnittstellen, die auf dieser Ebene ansetzen, müssen die externe Ansteuerung des Roboterbefehlssatzes und der zugehörigen Parameter von einem alternativen Steuerungssystem ermöglichen. Die Schnittstelle besitzt damit die gleiche Granularität wie die Programmiersprache des Robotersystems. Soll der gesamte Befehlsumfang abstrahiert werden, muss die Schnittstelle also einen entsprechenden Umfang abbilden können, um diesen komplett abzudecken.

Da die externe Steuerung den Programmablauf logisch definiert und verarbeitet, die Aktionsausführung aber weiterhin vollständig auf dem Robotersystem erfolgt, sind prinzipiell keine hohen zeitlichen Anforderungen an das externe Steuerungssystem erforderlich. Die Schnittstelle kann dementsprechend auf herkömmlicher TCP-Kommunikation aufbauen, aber ebenso mit echtzeitfähigen Bussystemen umgesetzt werden. Allerdings muss sie die Breite der physischen Verbindung, also die Anzahl an parallel übertragbaren Bits, die Menge an Informationen der logischen Schnittstelle abdecken.

Sämtliche Funktionalitäten, die die Bewegungsplanung, -steuerung und -regelung des Manipulators betreffen, werden weiterhin von der Robotersteuerung ausgeführt. Kenntnisse über technische Daten des Roboterarms sind somit grundsätzlich nicht notwendig.

Ein weiterer Vorteil ist, dass die Schnittstelle die zentrale Anwenderlogik vom proprietären Steuerungssystem abstrahiert und sowohl die Bedienung als auch die Steuerung über ein externes System ermöglicht, wobei dies auf Basis des Befehlsumfangs des Robotersystems erfolgt. Dies führt zu einer großen Schnittstellendefinition und zu einer hohen Redundanz, da beide Systeme auf Basis von Roboterbefehlen Steuerungslogiken definieren und verarbeiten.

Die herstellerübergreifende Abstraktion der Bewegungssteuerung der Robotersysteme ist zudem in mehreren Dimensionen beschränkt. Zum einen führt der Fokus auf die nativen Roboterbefehle dazu, dass proprietäre Funktionalitäten und Parametrisierung in der Schnittstelle enthalten bleiben. Des Weiteren erschwert die erforderliche Breite der Schnittstelle eine herstellerübergreifende Vereinheitlichung. Da die Ausführung der Befehle weiterhin vollständig auf der Robotersteuerung erfolgt, bleiben Bahnplanung und Regelung herstellerspezifisch. Dies

bedeutet beispielsweise, dass bei absolutgenauen Robotern[11] unterschiedlicher Hersteller die externe Vorgabe einer Pose-Fahrt mit kartesischem Zielpunkt zwar zu einer identischen TCP-Position führen würde, aber Trajektorie oder Achsstellungen unterschiedlich ausfallen können, da diese nicht im Abstraktionsumfang der Schnittstellenkonzepte auf Applikationsebene enthalten sind.

Implementierungsbeispiele für eine externe Ansteuerung auf Applikationsebene sind bei Friedrich [67] und den von ihm definierten Elementaranweisungen oder den bei Lambrecht [69] verwendeten Generic Commands zu finden. Auf industrieller Seite ist Kuka mxAutomation [86] als Beispiel zu nennen. Mit dieser Programmierschnittstelle werden Bewegungsbefehle in einer SPS definiert und an die Robotersteuerung gesendet.

Schnittstellenkonzepte auf Applikationsebene ermöglichen eine skalierbare Abstraktion der Programmlogik. Nicht alle Programminhalte müssen zwangsläufig externalisiert werden, beispielweise kann für bildverarbeitungsunterstützte Handhabungsanwendungen nur die notwendige Bewegung zur Aufnahme der Teile über das externe System erfolgen. Andere Abläufe, wie das Ablegen der Teile oder das Anfahren von Vorpositionen, können weiterhin isoliert im Robotercontroller stattfinden.

II - Bahnplanung

Die nächst tiefere Ebene in der Architektur eines Robotercontrollers, auf der eine Vielzahl von Schnittstellen ansetzen, stellt die Bahnplanung dar. Der Terminus „Bahnplanung“ kann in der Robotik ja nach Anwendungsfall unterschiedliche Bedeutungen besitzen und ist daher genauer zu definieren. Im vorliegenden Fall ist mit „Bahnplanung“ keine übergeordnete geometrische Bahnplanung zu verstehen, deren Aufgabe es beispielsweise ist, einen kollisionsfreien Pfad zwischen einem Ausgangs- und einem Zielpunkt zu bestimmen [105]. Im gegebenen Kontext ist unter dem Begriff „Bahnplanung“ eine Funktion der Bewegungssteuerung eines Roboters definiert. Aufgabe dieser Bahnplanung ist es, Bewegungsbefehle in konkrete zeitliche Bewegungsbahnen umzurechnen, man spricht hierbei auch von Trajektorien [106]. Diese Trajektorien geben aufgrund des Zeitbezuges nicht nur Auskunft über die Zwischenpunkte einer Manipulatorbewegung, sondern definieren ebenso den Geschwindigkeits- und Beschleunigungsverlauf.

Externe Schnittstellen und Steuerungskonzepte, die auf der Bahnplanungsebene ansetzen, zielen darauf ab, neben der vollständigen Applikationslogik auch Umfänge der Bahnplanung zu abstrahieren und dem Robotersystem zu entziehen. Dabei soll die Bewegung eines Roboters nicht nur über die üblichen Bewegungsbefehle und Parameter für Start-, Endpunkt sowie Geschwindigkeit definiert und vom Robotersystem alleinig bestimmt und ausgeführt, sondern über Zwischenpunkte gezielt die Bahnplanungsalgorithmik des externen Steuerungssystems mit eingebunden werden. Hierzu wird eine abzufahrende Bahn im externen System bestimmt und über eine Kommunikationsschnittstelle an das Robotersystem übertragen. Die empfangene Trajektorie wird anschließend in einem Roboterprogramm auf dem Robotercontroller verarbeitet und auf Basis von roboterspezifischen Bewegungsbefehlen abgefahren, wobei mit Hilfe von Überschleifungen versucht wird, die Roboterbewegung durchgängig und ohne Halten in den übermittelten Zwischenpunkten durchzuführen. Die einzelnen Roboterpositionen der Trajektorie

[11] Roboter werden als absolutgenau bezeichnet, wenn sie über ein spezielles Mess- und Parametrierverfahren in ihrer Leistungsfähigkeit optimiert wurden und somit bessere Genauigkeitskennwerte als nicht vermessene Roboter erreichen (vgl. Kapitel 5.2.2).

können dabei sowohl als Achsstellungen als auch kartesisch definierte Posen[12] vorgegeben werden, jedoch wird im Rahmen der bekannten Umsetzungen fast ausschließlich die achsbasierte Ansteuerung verwendet, da dies eine bessere Kontrolle des Robotersystem ermöglicht.

Für die korrekte Berechnung von Koordinatentransformationen und Roboterbahnen benötigt das externe System allerdings erweitertes Manipulatorwissen – insbesondere Kenntnisse über den kinematischen Aufbau des Roboterarms, um anhand der Gelenkstruktur des Roboters die Vorwärts- und Inverskinematik bestimmen zu können. Dies ist erforderlich, wenn innerhalb des externen Steuerungssystems Programmanweisungen anhand des Werkzeugarbeitspunktes erfolgen und die zugehörige Pose in die Gelenkwinkel des Manipulators umgerechnet werden muss. Man spricht von der inversen Kinematik bzw. Rückwärtstransformation. Die umkehrte mathematische Beziehung, also der Rückschluss auf den Werkzeugarbeitspunkt anhand der Positionen der Einzelachsen wird als Vorwärts- bzw. direkte Kinematik bezeichnet. Tiefere Kenntnisse über die statischen und dynamischen Eigenschaften der Robotermechanik sind nicht erforderlich, da die notwendigen Berechnungen weiterhin im nativen Robotercontroller und nicht in der externen Steuerung erfolgen.

Implementierungs- und Schnittstellenbeispiele für die Bahnplanung zeigen sich bei vielen Umsetzungen im Umfeld von ROS Industrial [107], da die Bahnplanungsebene die tiefst mögliche Integrationsmöglichkeit ist, welche ohne Unterstützung des Roboterherstellers verwendet werden kann. Aufgrund der Fokussierung auf die Übermittlung von Positionswerten fallen die Schnittstellen schmal aus, da im Gegensatz zu den Steuerungskonzepten auf Applikationsebene nicht der gesamte Befehlssatz definiert werden muss. Die Zeitanforderungen sind bei beiden Konzepten vergleichbar. Echtzeitanforderungen existieren auch bei Bahnplanungsschnittstellen nicht, da die Roboterbahnen entweder vorab oder über einen gepufferten Stream während der Ausführung übermittelt werden können. Der Informationsaustausch zwischen Roboter- und externer Steuerung erfolgt daher meist über TCP- bzw. Socket-Kommunikation.

Aus Sicht der Externalisierungslogik von Roboterfunktionen zielen Schnittstellen auf Bahnplanungseben zwar darauf ab, die Bahnplanung eines Robotersystems zu abstrahieren, sie müssen aus technischer Sicht jedoch über Bewegungsbefehle im Applikationsprogramm implementiert werden. Die zentrale Bahnplanungsfunktion des Robotersystems bleibt intakt, da derartige Ansteuerungskonzepte nur darauf aufbauen und sie nicht ersetzen. Die Integrationsebene wird deshalb in der weiteren Betrachtung innerhalb vorliegender Arbeit als „hybride Bahnplanung“ bezeichnet.

Die redundante Ausführung der Bahnplanung ist ebenso der zentrale Nachteil solcher Ansteuerungskonzepte, da eine Trajektorie zwar im externen System berechnet wird – die genaue Ausführung sowohl in geometrischer als auch in zeitlicher Dimension aufgrund der Verarbeitung im Robotercontroller nicht garantiert oder vorhergesagt werden kann. Zudem kann je nach Implementierung der Schnittstelle die Geschwindigkeit des Robotersystems nicht ausgeschöpft werden.

[12] Eine Pose ist definiert als Kombination von Position und Orientierung im dreidimensionalen Raum [4].

III - Interpolation

Die geschilderte Problematik der redundant ausgeführten Bahnplanung wird mit externen Steuerungskonzepten auf Interpolationsebene vermieden, da bei diesen die Bahnplanung des Robotersystems deaktiviert ist. Grundsätzlich ist unter Interpolation die Berechnung von Zwischenpunkten einer Roboterbahn zu verstehen, welche anschließend als Sollwerte an die Gelenkregelkreise im Interpolationstakt des Roboters übergeben werden.

Bei Ansteuerungskonzepten mit einer derartigen Schnittstelle bietet die Robotersteuerung die Möglichkeit an, die zyklische Vorgabe der Sollpositionen von einem externen System übernehmen zu lassen. Die Anforderung an dieses erhöht sich im Vergleich zu den vorherigen Konzepten enorm. Zum einen müssen sämtliche Applikationslogik und Bahnplanungs-algorithmik innerhalb des externen Systems vorgehalten und ausgeführt werden. Zum anderen muss die Übermittlung der Positionen im Gegensatz zu den vorherigen Konzepten innerhalb der definierten Zeitanforderungen des Robotersystems erfolgen. Diese können je nach Schnittstellenauslegung von mehreren Millisekunden bis hin zu teilweise nur einigen wenigen hundert Mikrosekunden liegen. Die Nutzung von nicht echtzeitfähigen externen Steuerungssystemen ist nur noch bedingt möglich. Ebenso entfällt die Verwendung von herkömmlichen TCP-basierten Kommunikationsmechanismen angesichts mangelnder Performance. Je nachdem, ob die Positionsübermittlung auf Basis kartesischer Koordinaten oder anhand von Achswinkeln erfolgt, ist die Berechnung der inversen Kinematik noch im Robotersystem oder bereits in der externen Steuerung enthalten. Allerdings ist für die Zielanwendung der Bewegungsabstraktion die reine Ansteuerung über Achswinkel zu bevorzugen, da nur auf diese Weise Manipulatorstellungen eindeutig definiert und Steuerungsoperationen sowohl auf Gelenk- als auch kartesischer Ebene (mit eigener Umrechnung) möglich sind.

Grundsätzlich bietet die Ansteuerung auf Interpolationsebene den Vorteil, dass die Bahnplanung nun ausschließlich über das externe System erfolgt und die Regelung des Manipulators vollständig in der Robotersteuerung verbleibt. Dadurch ist im Gegensatz zur vorherigen Schnittstelle die Robotertrajektorie ausschließlich über die externe Steuerung festgelegt, da eine klare Systemgrenze gegeben ist und die Definition der Schnittstelle handhabbar und schmal bleibt. Jedoch muss dieses Ansteuerungskonzept auch vom Roboterhersteller angeboten werden. Eine rein kunden- bzw. anwenderseitige Implementierung ist nicht möglich. Kuka RSI [108], Comau C5GOpen [84] oder Stäubli UniVal [85] sind hier als Beispiele zu nennen. Exemplarische Anwendungen sind bei OROCOS [38], Rickert [109], und Keibel [29] zu finden.

Einen weiteren Nachteil bzw. Umsetzungsaufwand stellen die Anforderungen an das externe Steuerungssystem in Bezug auf das notwendige Robotermodell dar. Insbesondere für performante Anwendungen sind im Vergleich zu den vorherigen Konzepten zusätzlich Kenntnisse über Geschwindigkeits- und Beschleunigungs- bzw. Drehmomentgrenzen und Massen des Manipulators erforderlich, um ein Dynamikmodell im externen Steuerungssystem aufbauen zu können und die Positionsvorgabe an der Leistungsgrenze zu ermöglichen. Auch können Funktionalitäten, wie z. B. die Absolutgenauigkeit des Robotersystems, verloren gehen oder das Gesamtsystem, wie bei Werner [110], verschlechterte Leistungsparameter (z. B. geringere Beschleunigungswerte) als das native System aufweisen. Eine Beantwortung der Performance-Eigenschaften für Interpolationsschnittstellen kann dementsprechend nicht rein theoretisch erfolgen.

IV - Antriebsregelung

Die letzte und somit tiefste Schnittstellenebene für eine externe Ansteuerung von Robotersystemen stellt die Antriebsregelung dar. Aufgabe dieser Ebene ist die elektrische Ansteuerung der Elektromotoren mit Frequenzumrichtern, um die vorher berechneten Sollpositionen der einzelnen Achsstellungen korrekt anzufahren. Bei dieser rein elektrischen Schnittstelle entfällt die Robotersteuerung des Herstellers komplett und wird durch das externe Steuerungssystem ersetzt. Die notwendigen Kenntnisse über den Roboterarm sind konsequenterweise sehr hoch. Um diese Schnittstelle umzusetzen, sind nicht nur Informationen zum mechanischen Modell des Roboters notwendig, sondern auch Details zu Getriebe und Motoren wie Reibeffekte, Drehzahl, Frequenzen und Drehmomente. Letztendlich alle Spezifikationen, die der Roboterhersteller selbst benötigt, um eine Robotersteuerung zu entwickeln.

Von Vorteil ist, dass sämtliche Steuerungs- und Regelungsaufgaben von der externen Steuerung übernommen werden und eine klare Systemtrennung gegeben ist. Eine Beeinflussung der Performance durch ein hybrides Steuerungskonzept ist somit ausgeschlossen und es können je nach Leistungsfähigkeit des externen Controllers die gleichen Performancewerte des reinen Herstellersystems bzw. sogar bessere erreicht werden. Von Nachteil sind nicht nur die hohen erforderlichen Kenntnisse über den Manipulator, sondern ebenfalls über grundlegende Regelungs- und Steuerungsfunktionen, da faktisch eine alternative Robotersteuerung entwickelt werden muss. Trotzdem bietet diese Schnittstelle eine Möglichkeit, das klassische Robotersystem zu abstrahieren und eine modularere Lösung der Roboterintegration zu schaffen.

Offiziell unterstützen indessen nur sehr wenige Hersteller von Robotersystemen ein solches Konzept. So bietet bei den etablierten Industrieroboterherstellern lediglich Comau die Möglichkeit an, Roboterarme ohne Controller und mit den notwendigen Manipulatorinformationen zu erwerben [84]. Auf Seite der externen Steuerung haben beispielsweise B&R, Keba oder Siemens Antriebs- und Steuerungslösungen im Produktportfolio, um die Aufgaben des Roboterherstellercontrollers zu übernehmen. Anwendung finden diese Lösungen häufig im Sondermaschinenbau oder bei Werkzeugmaschinen, da hier auf die bereits bestehende, steuerungstechnische Infrastruktur aufgebaut werden kann und die Manipulatoren integriert werden können. Des Weiteren existieren derartige Steuerungskonzepte auch bei einigen Anwendungen in der Prozesstechnik, z.B. bei Lackierrobotern.

Peripherieeinbindung

Eine wichtige Rolle nimmt in allen Ansteuerungskonzepten die Anbindung der Peripherie ein. In einer herkömmlichen Robotersteuerung liegt die Peripheriekommunikation parallel zur Interpolations- und Antriebsebene. Die Auswertung und Verarbeitung der Ein- und Ausgänge erfolgt anhand der im Applikationsprogramm definierten Logik.

Da die Verarbeitung der beiden obersten Schnittstellen, Applikationsprogramm und hybride Bahnplanung, auf Applikationsebene erfolgt, kann die Peripherieverbindung entweder weiterhin im Robotercontroller oder auch eigenständig über das externe Steuerungssystem erfolgen. Auf Interpolationsebene ist dies prinzipiell auch möglich, wenn das Robotersystem dies unterstützt. Für Steuerungskonzepte auf Antriebsebene ist die Peripherieanbindung nur noch im externen System darstellbar. Mit der zusätzlichen Einbindung von Aktorik und Sensorik können sich die zeitlichen Anforderungen anwendungsabhängig erhöhen. Folglich ist die Verlagerung der

Peripheriekommunikation ins externe Steuerungssystem bei Schnittstellen auf Applikationslogikebene nicht geeignet für die Ausführung von Anwendungen mit sensorgeführten Bewegungen und engen Regelschleifen, jedoch ist die Kommunikation mit weniger hohen Zeitanforderungen, z. B. mit einem übergeordneten Leitstandsystem, sehr gut möglich.

5.1.2. Gesamtübersicht

Abbildung 19 stellt die beschriebenen Schnittstellenebenen und Beispiele für die Umsetzung von Ansteuerungskonzepten über externe Steuerungen anhand der Architektur eines Robotercontrollers dar. Tabelle 2 enthält zusammenfassend die zugehörigen Charakteristiken und Vor- bzw. Nachteile für die vier definierten Interface-Kategorien.

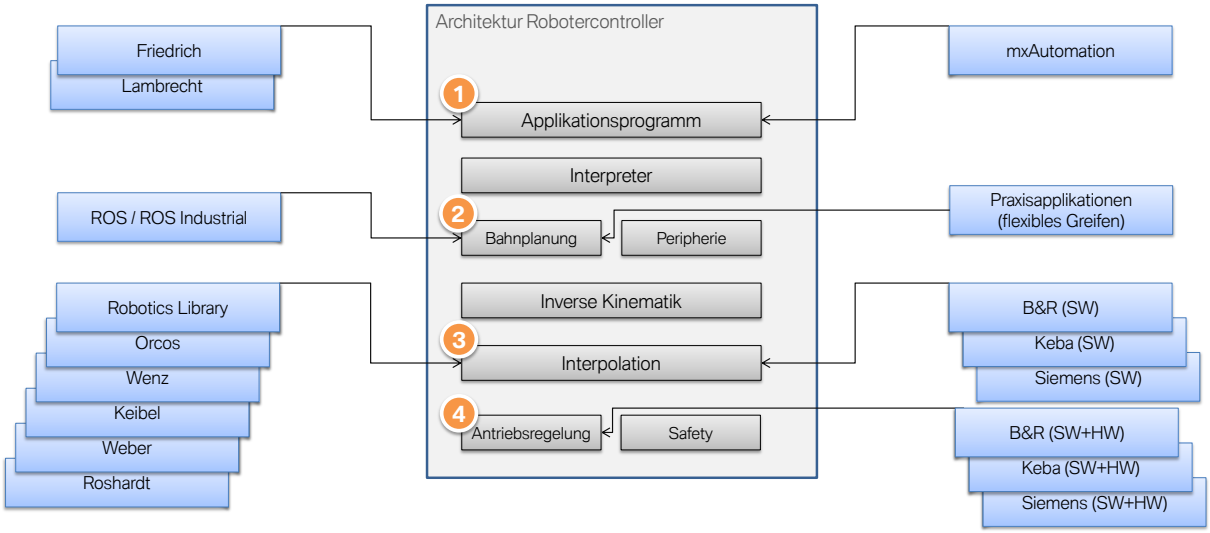


Abbildung 19 – Existierende Umsetzungen zur externen Roboteransteuerung und der verwendeten Schnittstellen

Es zeigt sich, dass sowohl in Industrie als auch Wissenschaft unterschiedlichste Schnittstellen definiert und verwendet wurden, um Steuerungsintelligenzen aus der proprietären Robotersteuerung zu entziehen. Jede dieser Anwendungen bietet daher einen potenziellen Lösungsansatz für die zentrale Fragestellung der herstellerunabhängigen Integration von Roboterapplikationen im Karosseriebau. Allerdings beinhaltet keine der beschriebenen Arbeiten bzw. Anwendungen eine fundierte Übersicht der Abstraktionsmöglichkeiten und Begründungen für die Wahl der jeweiligen Schnittstellen. Durch die Kategorisierung anhand der Integrationstiefe konnte eine geeignete Gruppierung der angewandten Methoden bzw. der existierenden Schnittstellen vollzogen und der Betrachtungsraum generalisiert werden. Auf dieser Basis soll im Folgenden ein Anforderungskatalog entwickelt und die zur Verfügung stehenden Abstraktionsebenen bewertet werden, um eine Eignungsaussage für die Roboterintegration zu erhalten.

Tabelle 2 – Externalisierungsebenen und deren Charakteristika

Integrationsebene / Kategorie	Applikationsprogramm	(Hybride) Bahnplanung	Interpolation	Antriebsregelung
Aufgabe der Ebene	• Definition der Anwendungslogik auf Basis des im Robotersystems vorhandenen Befehlssatzes	• Berechnung der Trajektorie des Robotersystems • Sowohl kartesisch als auch achsbasiert	• Berechnung der interpolierten Positionen und Übergabe der Achssollwerte an Gelenkregelkreise	• Steuerung und Regelung der Motoren über Frequenzumrichter/Antriebsverstärker
Externe Schnittstelle	• Übermittlung der auszuführenden Roboterbefehle und Parameter • Rückmeldung der Befehlsausführung bzw. -status • Statusinformationen über Systemzustände (z.B. Werkzeug- und Roboterposition) • Eigene Implementierung möglich	• Übermittlung von Sollpositionen/Trajektorien • Exakt getaktete Übermittlung nicht erfoderlich => Vorabübermittlung möglich, um Zeitanforderungen zu reduzieren • Statusrückmeldung erforderlich • Teilumfänge aus Applikationsschnittstelle möglich • Eigene Implementierung möglich	• Zyklische Übemittlung von Achspositionen oder kartesischen Positionen • Zykluszeiten varieren von mehreren hundert Mikrosekunden bis zu zweistelligen Millisekunden • Rückmeldung der aktuellen Position • Ggf. weitere Optionen, wie I/O-Steuerung möglich • Muss vom Hersteller zur Verfügung gestellt sein	• Elektrische Schnittstelle • Regelung der Motorströme des Roboterarms
Schnittstellenbreite	• U. U. sehr breit, da gesamter Befehlsumfang inklusive -parameter definiert werden muss, um vollständige Ansteuerung zu ermöglichen	• Gering, da lediglich Positionen im Zeitverlauf übermittelt werden • Jedoch Overhead zur Verarbeitung der Positionen notwendig, da Schnittstellenimplementierung im Applikationsprogrammm	• Sehr gering, da nur Achspositionen oder kartesische Positionen	• Gering - physikalische Basiswerte wie Frequenz, Spannung, Stromstärke
Schnittstellentyp	• TCP • Feldbus	• TCP	• UDP • Feldbus	• Elektrische Verbindung
Externes System	• Definition und Ausführung der Anwenderlogik • Auswertung der Rückgabewerte und logische Verarbeitung • Visualisierung und Bedienung des Systems	• Umfänge der Applikationsschnittstelle • Bahnplanungsumfänge des Robotersystems	• Umfänge der Bahnplanungschnittstelle • Interpolation der Trajektorie	• Eigenständige Robotersteuerung
Robotersystem	• Empfang und Ausführung der Steuerungsbefehle • Ein Großteil der Funktionalitäten bleibt erhalten • Visualisierung/Bedienung entfällt • Applikationsprogramm kann entfallen, aber auch für Implementierung genutzt werden	• Empfang und Abfahrt von vorgegebenen Positionen, bzw. Zwischenpositionen • Ansonsten wie Applikationsprogramm	• Empfang der interpolierten Positionen und direktes Anfahren dieser • Keine eigene Bahnplanung	• Nur der Roboterarm bleibt erhalten • Randkonzepte mit Roboterarm und Umrichter möglich
Manipulatorwissen	• Nicht erforderlich	• Nicht zwingend erforderlich • Kann aber je nach Umsetzung und Anwendung notwendig sein	• Kinematischer Aufbau bei Achspositionsteuerung zwingend erforderlich, um inverse Kinematik bzw. Vorwärtskinematik zu berechnen • Dynamikparameter für hochperformante Anwendungen	• Sehr hoch • Kinematischer Aufbau, Massen, etc. • Informationen zu Motoren erforderlich
Einbindung Peripherie	• Über Roboter- oder externe Steuerung möglich	• Über Roboter- oder externe Steuerung möglich	• Primär über externe Steuerung • Je nach Schnittstellenausführung auch über Robotersystem	• Nur über externe Steuerung möglich
Beispiele	• Friedrich, Lambrecht • Kuka MXAutomation	• ROS Industrial-Schnittstellen	• Comau C5Gopen, Stäubli UniVAL • Kuka RSI, ABB EGM	• Comau-Roboter mit B&R-, Keba-, Siemens Steuerung • Dürr-Lackierroboter (Kuka-Mechanik + Keba-Steuerung)
Vorteile	• Zentrale Aspekte werden abstrahiert • Steuer- und Regelungsfunktionalitäten für Manipulator bleiben im Robotercontroller • Performance der Manipulatorbewegung bleibt unverändert • Kein Manipulatorwissen erforderlich • Prinzipiell keine hohen zeitlichen Anforderungen an das externe Steuerungssystem (Applikationsbedingt jedoch auch höher)	• Zentrale Aspekte werden über externes System ausgeführt • Teile der Bahnplanung können ausgelagert werden • Interface kann schmaler als bei Applikationsprogramm ausfallen • Prinzipiell keine hohen zeitlichen Anforderungen an das externe Steuerungssystem (Applikationsbedingt jedoch auch höher)	• Abstraktion sämtlicher applikationsrelevanten Logiken • Regelung bleibt im Robotercontroller • Klare Systemgrenzen grundsätzlich vorhanden, um Gewährleistungsverantwortung trennen zu können • Bahn des Roboterarms vollständig durch externes System bestimmbar, da Bahnplanung des Robotercontrollers deaktiviert	• Vollständige Steuerung und Regelung über alternative Robotersteuerung • Perfomance je nach Qualität des externen Steuerungssystems, kann also schlechter oder besser als natives System sein
Nachteile	• Komplexe und breite Schnittstelle • Bewegung des Roboters bleibt herstellerabhängig • Teilweise hohe Systemredundanzen, da Befehlsbasis in beiden System verarbeitet werden muss • Trägheit des Systems, da Rückmeldung über Controller, aber Auswertung im externen System • Systemperformance bei Regelung mit Sensorik kritisch	• Redundante Ausführung der Bahnplanung • Keine klaren Systemgrenzen • Robotersystem beeinflusst weiterhin die Bahn • Genaue Trajektorie und Zeitverhalten u.U. nicht vorhersagbar • Trägheit des Systems	• Für hohe Performance sind Kenntnisse über Dynamikparameter des Manipulators notwendig • Hohe Anforderungen an das externe Steuerungsystem hinsichtlich Echtzeitanforderungen • Hohes Roboter-Know-how erforderlich	• Sehr hohes Know-how notwendig - sowohl über Manipulator als auch Steuerungs- und Regelungstechnik • Nur von sehr wenigen Herstellern unterstützt • Keine klare Trennung der Gewährleistungsverantwortung
Einschätzung der Standardisierbarkeit von Applikationen	• Applikationslogik u. Visualisierung abstahierbar • Keine Beeinflussbarkeit des Bewegungsverhaltens • Redundanz in Peripheriesteuerung • Aufwändige und herstellerspez. Roboterschnittstellen • Kein Manipulatorwissen notwendig	• Applikationslogik u. Visualisierung abstrahierbar • Herstellerabhängigkeit in Bewegungen weiterhin vorhanden • Schlechte Vorhersagbarkeit der Bewegung • Redundanz in Peripheriesteuerung • Roboterschnittstelle relativ einfach	• Sämtliche logische Komponenten einer Applikation können abstrahiert werden • Schmale Schnittstelle, deren Standardisierung herstellerübergreifend möglich ist • Hohes Roboter-Know-how erforderlich	• Modularisierung eines geschlossenen Systems • Vermutlich sehr hohe Barrieren bei Roboterherstellern • Sehr hohes Roboter-Know-how erforderlich • Abstraktion von Roboterfunktionen, die aus Applikationssicht keine Relevanz haben

5.1.3. Anforderungskatalog und Bewertung aus Endanwendersicht

Im folgenden Kapitel sollen die definierten Abstraktionsebenen nun hinsichtlich Ihrer Eignung für die herstellerunabhängige Ansteuerung von Robotersystemen im Karosseriebau bewertet werden. Hierfür wurden im Rahmen eines Expertenworkshops mit Teilnehmern aus BMW-Vorentwicklung und Roboterstandardisierung zunächst Anforderungen definiert, die aus OEM-Sicht entscheidend für eine erfolgreiche Anwendung von derartigen Abstraktionsmethoden sind. Anschließend wurden die Anforderungen hinsichtlich Ihrer Wichtigkeit beurteilt, die identifizierten Lösungsräume bewertet und zusammen mit dem Status quo gegenübergestellt.

Für die Bewertungsgrundlage wurde ein Anforderungskatalog aus 16 Elementen definiert, die sich in folgende vier Hauptkategorien einteilen lassen:

Abstraktion der Standardisierungsumfänge

Aus Endanwendersicht müssen sich die in Kapitel 4 beschriebenen Aufwände der Standardisierung aus Unternehmenssicht reduzieren. Die Entwicklung von Applikationslogiken, Bedienkonzepten und der Visualisierung soll interoperabel und nicht herstellerspezifisch erfolgen. Ebenfalls soll die aufwändige Integration von über- und untergeordneten Peripheriekomponenten reduziert werden. Diese umfasst die initiale Feldbuskonfiguration, sowie die notwendigen Logiken zur Ansteuerung und Auswertung dieser. Zudem sollte die verwendete Schnittstelle selbst systemübergreifend anwendbar sein, um eine herstellerübergreifende Standardisierung zu ermöglichen.

Geschäftsmodell

Anhand der gewählten Abstraktionsebene muss sich ein Geschäftsmodell entwickeln lassen, das sowohl für Endanwender als auch für aktuelle und zukünftige Zulieferer ein schlüssiges Gesamtbild ergibt. Andernfalls besteht das Risiko, dass externe Schnittstellen nicht angeboten werden, der notwendige Zuliefermarkt sich nicht entwickelt und eine Verwendung des Systems sich aus Endanwender- und Lieferantensicht nicht rentiert. Es sollte vermieden werden, Funktionen unnötig redundant in mehreren Systemen umzusetzen. Verantwortlichkeiten sollten zwischen den Beteiligten klar voneinander abgrenzbar sein und sicherheitsrelevante Funktionen möglichst vom Lieferanten des Manipulators umgesetzt werden, da von der bewegenden Mechanik auch das höchste Gefährdungspotenzial ausgeht.

Leistungsfähigkeit

Die Leistungsfähigkeit des Gesamtsystems bei der Ansteuerung über eine externe Steuerung muss dem Status quo heutiger Robotersysteme bzw. den Anforderungen an die Fertigungsprozesse im Karosseriebau genügen. Dies bedeutet, dass die Robotersysteme in der Lage sein müssen, vergleichbare Traglasten handhaben zu können und die Genauigkeit der Roboterbahnen, insbesondere bzgl. der prozessrelevanten Bahnpunkte, muss ausreichend sein. Außerdem muss die Ansteuerung der Peripherie in Kombination mit der Roboterbewegung die notwendigen Zeitanforderungen der Karosseriebauanwendungen erfüllen.

Zusätzlicher Integrationsaufwand

Aufwände, die durch die Externalisierung entstehen, sollten so gering wie möglich gehalten werden. Dies bedeutet, dass Breite und Varianz der Schnittstellen idealweise einen überschaubaren Umfang besitzen. Darüber hinaus sollte das erforderliche Wissen über die Robotermechanik möglichst gering, leicht zugänglich und aufwandsarm implementierbar sein. Bereits in den Systemen vorhandene Funktionalitäten sollten bestmöglich wieder verwendet werden und nicht neu erstellt werden müssen.

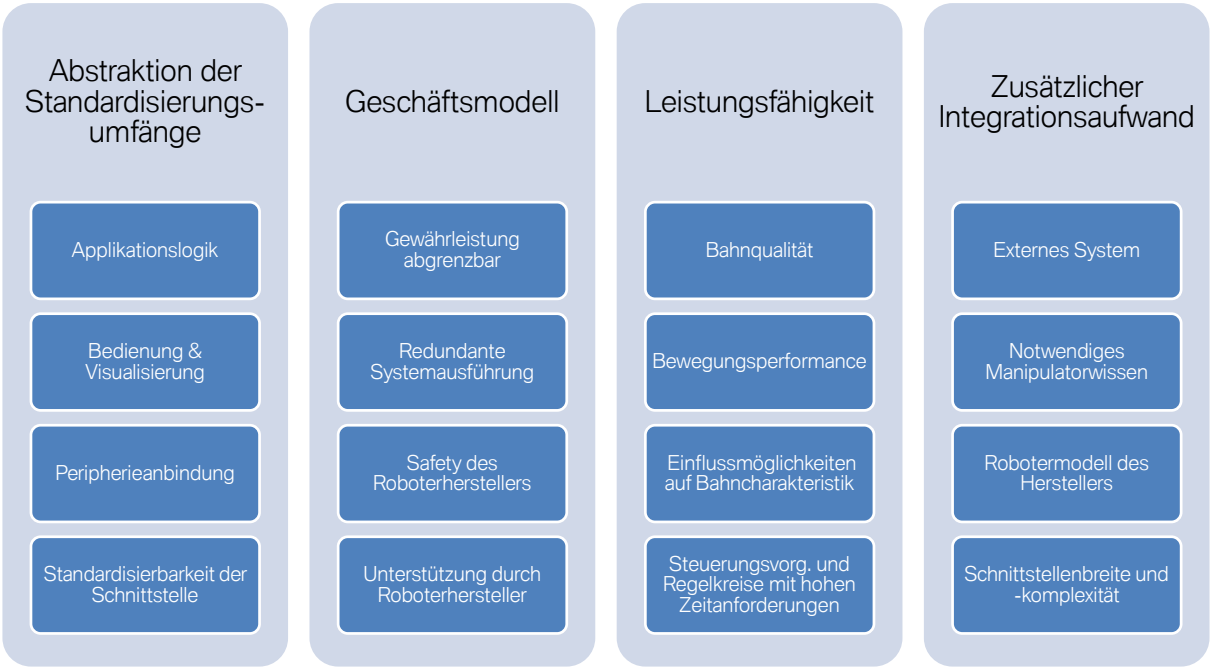


Abbildung 20 – Bewertungskategorien und -elemente

Aus den Hauptkategorien sind insgesamt 16 Bewertungselemente abgeleitet worden. Im ersten Schritt wurde zu jedem Element die Relevanz für den Integrationsprozess von Roboterapplikationen diskutiert und zwischen 0 (nicht relevant) und 3 (sehr relevant) bewertet. Bewertungspunkte, deren Umsetzung bzw. ausreichende Erfüllung unabdingbar für die erfolgreiche Realisierbarkeit der Externalisierungsmethode sind, wurden als Elementarkriterien definiert. Anschließend wurden für jeden Bewertungspunkt die einzelnen Steuerungskonzepte hinsichtlich ihrer Eignung bestimmt, welche mit einem Wert von 0 (nicht geeignet) bis zu 3 (sehr geeignet) bewertet wurden.

Die Bewertungselemente und deren Bedeutung lauten wie folgt:

Applikationslogik
Inwieweit ist es möglich, die aus Endanwender- und Prozesssicht relevante Logik zu abstrahieren?

Bedienung & Visualisierung
Wie stark können Visualisierungs- und Bedienkomponenten für den Endanwender abstrahiert werden?

Peripherieanbindung
Inwieweit ist es möglich, die Konfiguration und Ansteuerung des Bussystems sowie die Einbindung der über- und untergeordneten Peripherie zu abstrahieren?

Standardisierbarkeit der Schnittstelle
Wie gut kann auf Basis dieser Schnittstellenebene ein industrieübergreifender Standard definiert werden?

Gewährleistung abgrenzbar
Wie klar sind die Bereiche Mechanik, Regelung und Applikation voneinander getrennt, um im Gewährleistungsfall eine eindeutige Zuordnung der Verantwortlichkeiten zu ermöglichen?

Redundante Systemausführung
Inwieweit sind Roboterfunktionalitäten sowohl in Roboter- als auch externer Steuerung vorhanden und führen aus Sicht des Gesamtsystems zu Redundanzen und unnötigen Mehrkosten?

Safety des Roboterherstellers
Inwiefern können die Safety-Funktionen des Roboter- bzw. Manipulatorherstellers wieder- bzw. weiterverwendet werden, um die Zuordnung der Sicherheitsfunktionen möglichst nah an den Komponenten mit dem höchsten Gefährdungspotenzial zu belassen?

Unterstützung Roboterhersteller (Elementarkriterium)
Wie hoch ist die Wahrscheinlichkeit, dass etablierte Roboterhersteller ein Konzept der Externalisierung mittragen und die notwendigen Schnittstellen nativ anbieten?

Bahnqualität (Elementarkriterium)
Wie gut ist die Bahnqualität des Gesamtsystems im Vergleich zu konventionellen Lösungen?

Bewegungsperformance (Elementarkriterium):
Wie gut ist die Bewegungsperformance in Bezug auf Geschwindigkeit und Beschleunigung des Gesamtsystems im Vergleich zu konventionellen Lösungen?

Einflussmöglichkeit auf Bahncharakteristik
Wie stark kann der Endanwender Einfluss auf die Bahnplanung bzw. -charakteristik des Roboters nehmen, um beispielsweise Überschleifverhalten prozessseitig zu beeinflussen oder Kalibriermethoden endkundenspezifisch zu implementieren?

Steuerungsvorgänge und Regelkreise mit hohen Zeitanforderungen
Inwieweit ist die Umsetzung von Applikationen mit Steuerungs- und Regelungsvorgängen mit geringen Zykluszeiten und einer hohen Synchronität möglich? Beispielsweise für sensorgeführte Anwendungen.

Externes System
Wie hoch ist der Aufwand zur Entwicklung des externen Systems, um die notwendigen Funktionalitäten abzubilden, die bereits im Robotersystem des Herstellers enthalten sind?

Erforderliches Manipulatorwissen
Wie sehr sind Kenntnisse über den Manipulator notwendig (z. B. DH-Parameter, Massen, Massenträgheit, Reibung, Antriebssystem), um das Robotersystem bestmöglich steuern zu können?

Robotermodell des Herstellers
Inwiefern ist das Robotermodell des Roboterherstellers integriert, um vorhandenes Wissen und Erfahrungen bezüglich der Einflüsse auf das dynamische und kinematische Verhalten des Robotersystems wiederzuverwenden?

Schnittstellenbreite und -komplexität
Wie umfangreich und komplex ist die Schnittstelle?

Bewertungsergebnisse

Die Ergebnisse der Bewertung sind in Tabelle 3 dargestellt. In dieser sind die einzelnen Anforderungskategorien zeilenweise aufgelistet und bezüglich ihrer Relevanz eingestuft. Auf der rechten Seite der Tabelle befinden sich die Abstraktionskonzepte mit der jeweiligen Wertung für die einzelnen Anforderungskategorien.

Tabelle 3 – Bewertung der Abstraktionsverfahren aus Sicht des Karosseriebaus

Bewertungsmatrix				Abstraktionsverfahren				
Kategorie	Aspekt	Relevanz-multiplikator	Elementarkrit.	Status Quo	Applikations-programm	Hybride Bahnplanung	Interpolation	Antriebs-regelung
Abstraktion Standard.umf.	Applikationslogik	3		0	2	2	3	3
	Bedienung & Visualisierung	3		0	2	2	3	3
	Peripherieanbindung	2		0	2	1	3	3
	Standardisierbarkeit der Schnittstelle	3		0	1	2	3	3
Geschäfts-modell	Gewährleistung abgrenzbar	3		3	3	3	3	1
	Redundante Systemausführung	1		3	1	1	3	2
	Safety des Roboterherstellers	2		3	3	3	3	2
	Unterstützung Roboterhersteller	2	x	3	3	1	2	1
Leistungs-fähigkeit	Bahnqualität	3	x	3	3	1	2*	3
	Bewegungsperformance	2	x	3	3	2	2*	3
	Einflussmöglichkeit auf Bahncharakt.	3		0	0	1	2*	3
	Regelzyklen mit hohen Zeitanford.	1		2	0	1	2*	3
Integrations-aufwand	Externes System	3		3	2	1	1	0
	Manipulatorwissen notwendig	2		3	3	2	1	0
	Robotermodell des Herstellers	2		3	3	3	1	0
	Schnittstellenbreite und Komplexität	2		3	0	1	3	2
Ø pro Kategorie	Abstraktion der Standardisierungsumfänge			0,00	1,73	1,82	3,00	3,00
	Geschäftsmodell			3,00	2,75	2,25	2,75	1,38
	Leistungsfähigkeit			1,89	1,67	1,22	2,00*	3,00
	Zusätzliche Integrationsaufwände			3,00	2,00	1,67	1,44	0,44
	Gesamtbewertung			**1,84**	**2,00**	**1,73**	**2,32**	**2,03**

Bewertung
3 – sehr gut
↕
0 – schlecht

*hohe Unsicherheit in der Bewertung

Es zeigt sich, dass alle Schnittstellenkonzepte als Verbesserung zum Status quo in Bezug auf die Abstraktion der Applikationsstandardisierung eingestuft werden, jedoch wird bei der Applikationsschnittstelle die Breite und Proprietarität der Schnittstelle als kritisch angesehen. Die Implementierung wird deshalb als aufwändig und eine herstellerübergreifende Normierung als unwahrscheinlich eingeschätzt. Schnittstellen der hybriden Bahnplanung sind zwar schmaler und werden daher grundsätzlich als einfacher zu standardisieren eingeschätzt. Die unsaubere Trennung der Bahnplanung wird aber insbesondere in Kombination mit der Ansteuerung von Peripherie als kritisch angesehen. Den anderen beiden Konzepten wird dank der besseren Handhabbarkeit der Schnittstellen und der Möglichkeit, die Trajektorie des Roboters vollkommen über die externe Steuerung zu definieren, eine bessere Standardisierbarkeit und Abstraktionsleistung zugerechnet. Zudem zeigt sich, dass Applikations- und Bahnplanungsebenen nicht alle Aspekte der Applikationsstandardisierungsabstraktion ermöglichen, während bei der Antriebsregelung ein hoher Teil nicht applikationsrelevanter

Aspekte unnötigerweise abstrahiert wird. Nachteilig werden hier ebenso der Integrationsaufwand und das umsetzbare Geschäftsmodell angesehen, da letztendlich eine alternative Robotersteuerung für ein bestehendes System entwickelt bzw. konfiguriert werden muss und die klare Trennung für Garantie- oder Sicherheitsthemen nicht gegeben sein könnte.

Aktuell wird die Partizipationsbereitschaft der meisten Roboterhersteller an einem solchen Geschäftsmodell als gering eingeschätzt, da Know-how in Bezug auf Mechanik und Mechatronik des Manipulators offen gelegt werden muss. Außerdem könnte die Befürchtung bestehen, sich durch das veränderte Geschäftsmodell zu einem reinen Hardwarelieferanten des Manipulators zu entwickeln, auch wenn hier ebenso die Chance gesehen wird, Marktanteile bzw. Umsatz durch die isolierte Veräußerung von Roboterarmen und Robotersteuerungen zu steigern.

Das Thema der Leistungsfähigkeit war von intensivem Diskurs geprägt. Während diese bei Applikationsschnittstellen als sehr hoch eingeschätzt wird, da das Steuerungs- und Regelungskonzept, wenn auch u. U. anders verteilt, dem heutigen entspricht, wurden die Schnittstellen auf Bahnplanungsebene aufgrund unklarer Trennung und Vermischung der Bahnplanung als nicht besonders leistungsfähig bewertet.

Bei der Ansteuerung über die Antriebsebene bestand der Konsens, dass die Leistungsfähigkeit theoretisch identisch zu heutigen Systemen bzw. sogar besser sein kann. Da das Steuerungskonzept im Prinzip dem eines klassischen Industrieroboters, nur in anderer Aufteilung, entspricht, hängt die Leistungsfähigkeit insbesondere vom externen System ab. Die steuerungstechnische Kompetenz und die Verfügbarkeit der Informationen bzgl. der Roboterkinematik ist somit entscheidend, um eine dem Herstellersystem adäquate, alternative Robotersteuerung zu entwickeln, die je nach Qualität des Originalsystems auch ein höheres Leistungsniveau erreichen kann.

Anders verhält sich die Sachlage bei Steuerungskonzepten auf Interpolationsebene. Die Leistungsfähigkeit ergibt sich durch den Systemverbund aus externem Steuerungssystem und dem Robotercontroller. Sie hängt von vielen Faktoren ab und eine einfache theoretische Beantwortung der Frage ist daher nicht möglich. Zum einen sind derartige Schnittstellen bei den klassischen Roboterherstellern vergleichsweise neu. Trotz der konzeptionellen Ähnlichkeit können sie zwischen den Herstellern diverse Unterschiede aufweisen und sich in Parametrierbarkeit, Integrierbarkeit und Leistungsfähigkeit unterscheiden. Zum anderen existieren keine veröffentlichten Untersuchungen bezüglich der Leistungsfähigkeit im steuerungstechnischen Gesamtverbund – insbesondere nicht für die Eignung für Anwendungen in der Automobilindustrie. Aus diesem Grund wurden die Bahnqualität und die Peripherieansteuerbarkeit vorläufig mit einem mittleren Performancewert bewertet.

In der Gesamtbewertung und der Gegenüberstellung aller Schnittstellen wird die Interpolationsansteuerung als am besten geeignete Methode angesehen, ein steuerungstechnisches Lösungskonzept zu entwickeln, das eine herstellerunabhängige Standardisierung von Roboterapplikation ermöglicht. Im Vergleich zu den anderen Externalisierungsalternativen bietet diese eine hohe Abstraktionsfähigkeit bzgl. der applikationsrelevanten Bestandteile, hat zudem nur einen überschaubaren Mehraufwand an zusätzlicher Integrationsarbeit und könnte in einem für alle Seiten sinnhaften Geschäftsmodell umgesetzt werden. Jedoch ist eine finale Beantwortung der Fragestellung der Leistungsfähigkeit notwendig, da die Erfüllung dieser als Elementarkriterium definiert und unabdingbar für einen erfolgreichen Einsatz ist. Das nächste Kapitel widmet sich deshalb dieser Thematik.

5.2. Leistungsmessungen

Anforderungen an Robotersysteme im Karosseriebau

Die Eignung eines Robotersystems für den Einsatz im Karosseriebau hängt insbesondere von der Erfüllung der anwendungsspezifischen und unternehmensübergreifenden Anforderungen ab. Diese sind innerhalb dieser Arbeit anhand interner Spezifikationsunterlagen und Expertenbefragungen am Beispiel der BMW AG analysiert, gruppiert und übergreifend zusammengefasst. Abbildung 21 stellt sie unterteilt nach Art der Anforderung in Fertigungsprozess- oder Unternehmensprozessanforderung dar. Aus Sicht der Verwendungen eines Steuerungskonzeptes auf Interpolationsebene ist neben der grundsätzlichen Leistungserfüllung die Systemverantwortung von Interesse. Deshalb wurde für die einzelnen Kriterien zusätzlich bewertet, ob die Erfüllbarkeit der Anforderung alleinig von der externen Steuerung, dem kombinierten Steuerungssystem aus Roboter und externer Steuerung, oder alleinig vom Robotersystem abhängt.

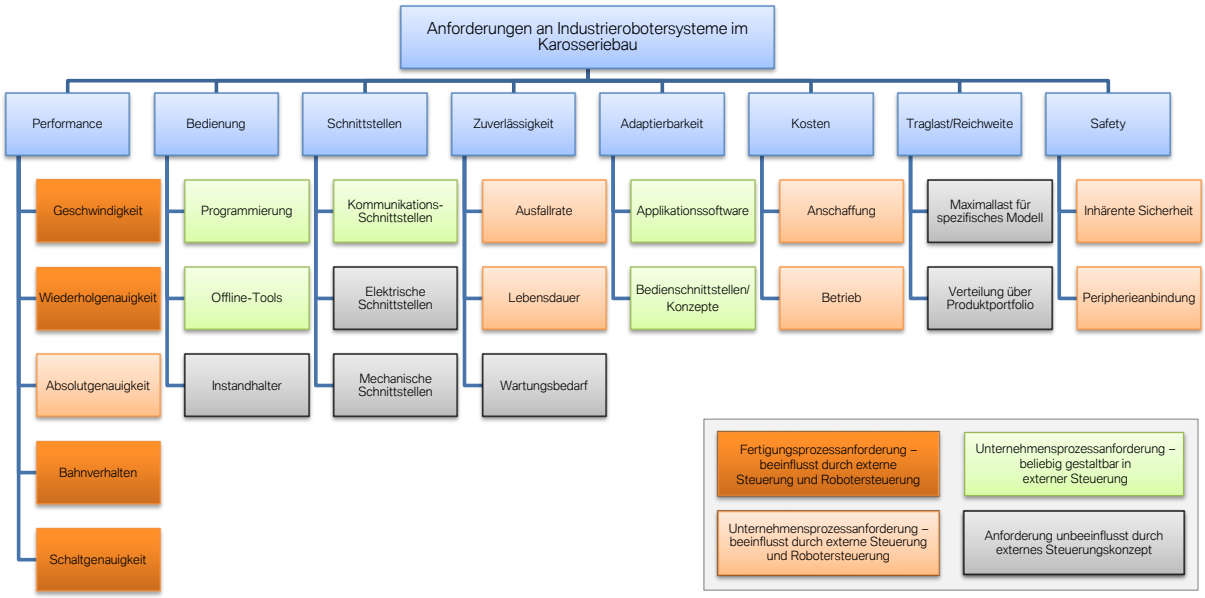


Abbildung 21 – Kategorisierte Anforderungen an Industrierobotersysteme im Karosseriebau bewertet nach Anforderungstyp (Fertigungs- oder Unternehmensprozess) und Beeinflussbarkeit durch externe Steuerung

Es zeigt sich, dass einige Leistungskriterien wie Traglast, Reichweite, Wartungsintervall oder elektrische Schnittstellen prinzipiell unbeeinflusst durch das veränderte Steuerungskonzept bleiben, da der grundlegende Mechanik- und Steuerungsaufbau nicht modifiziert wird. Andere Kriterien, wie die Programmierung oder Adaptierbarkeit werden nun alleinig von der externen Steuerung unabhängig vom Robotersystem ausgeführt.

Für unbeeinflusste Kriterien ist somit keine Reevaluation in Bezug auf den Einsatz der Interpolationsschnittstelle notwendig, da die Verantwortung weiterhin vollständig dem Herstellersystem obliegt und sich die Anforderungserfüllung nicht verändert. Ebenso sind Anforderungen, die nur vom externen Steuerungssystem zu erfüllen sind, nicht bewertungsrelevant im Zusammenhang mit dem Einsatz der Interpolationsschnittstelle, da die Erfüllung der Leistungsfähigkeit alleinig in der Gestaltung des externen Systems verantwortet ist. Das externe System muss diese Anforderungen natürlich auch erfüllen, die verwendete Schnittstelle hat allerdings hierbei keine Auswirkung auf die Zielerreichung.

Anders verhält es sich bei den Kriterien, die durch die Kombination der Roboter- und der externen Steuerung beeinflusst werden können, da hierzu weder eine rein theoretische Beantwortung erfolgen kann, noch Erfahrungswerte vorliegen und auch kein Abgleich mit den Anforderungen aus Automobilsicht existiert.

Insbesondere die Erfüllung der fertigungsprozesstechnischen Anforderungen spielt eine entscheidende Rolle, da diese in direktem Zusammenhang mit der Produktqualität stehen und somit entscheidend in Bezug auf die Einsatzfähigkeit der externen Steuerungskonzepte im Karosseriebau sind. Beispielsweise stellen Wiederholgenauigkeit und Geschwindigkeit bei Punktprozessen bedeutende Leistungskriterien dar, da diese ausschlaggebend sind, um in ausreichender Toleranz und Taktzeit einen Fertigungsvorgang durchzuführen. Bei Bahnprozessen sind Bahngenauigkeit, Geschwindigkeitstreue und Schaltgenauigkeit entscheidend, da diese z. B. das gleichmäßige und genaue Auftragen einer Kleberaupe gewährleisten. Für einige Anforderungen bietet die Externalisierung der Bahnplanung zudem einige Potenziale in Bezug auf die herstellerübergreifende Standardisierung, beispielsweise eine robotersystemübergreifende Modifikation des Bahnverhaltens, um das Überschleifverhalten unabhängig vom eingesetzten Hersteller gleichermaßen zu gestalten.

Neben den Fertigungsprozessanforderungen existiert eine Reihe von prozessualen Anforderungen wie Zuverlässigkeit oder Sicherheitsaspekte, die für eine breite Verwendung im Karosseriebau erforderlich sind, jedoch aus Fertigungsgesichtspunkten keine Bedeutung haben und deshalb für die grundlegende Tauglichkeitsbewertung zunächst außer Acht gelassen werden können.

Für die Einsetzbarkeit der Interpolationsschnittstellen sind folgende Fragen mittels Laboruntersuchungen zu klären:

1. Wie leistungsfähig sind Robotersysteme in Bezug auf:
    a. Geschwindigkeit und Beschleunigung
    b. Punkt- und Bahngenauigkeit
    c. Ansteuerung von Peripherie

2. Welchen Einfluss haben die Einstellungsparameter der Schnittstellen und des externen Steuerungssystems?

3. Ermöglicht die externe Ansteuerung eine bessere herstellerunabhängige Standardisierung von Roboterbahnen?

4. Ist es prinzipiell möglich, mit derartigen Steuerungskonzepten automobile Applikationen zu realisieren?

5. Wo liegen die Verbesserungspotenziale und Herausforderungen für einen breiten industriellen Einsatz in der Automobilindustrie und weiteren Branchen?

5.2.1. Untersuchungs- und Auswertemethodik

Zur Ermittlung und dem Vergleich der Leistungsfähigkeit der Steuerungslösungen diente ein Laborversuch mit zwei Robotersystemen für das Abfahren von Referenzbahnen, einem Lasertracker zur Erfassung der realen TCP-Position[13] der Roboterbahnen und einer Auswertungswerkzeugkette, welche die Gütekriterien der Messpunkte automatisiert analysiert und eine individuelle graphische Darstellung ermöglicht.

Normbasis, notwendige Modifikationen und Erweiterungen

Grundlage für die durchzuführenden Versuche stellte die ISO 9283 dar [111]. Diese Norm definiert wichtige Leistungskenngrößen für Roboter und beinhaltet Vorschläge, wie diese geprüft werden sollen. Eine umfassende Überprüfung aller dort definierten Kennwerte ist nicht sinnvoll, da diese zum Teil nicht durch die externe Steuerung beeinflusst werden oder keine Relevanz für Karosseriebauanwendungen besitzen. Zudem sollte folgende Versuchsreihe keine gesamtübergreifende Leistungsbewertung durchführen, sondern sich auf die in der Anforderungsanalyse definierten Kriterien für eine Tauglichkeitsbewertung externer Ansteuerungskonzepte fokussieren. In Abweichung zur Norm wurden die meisten Versuche nur bei mittlerer und nicht bei voller Traglast durchgeführt, da dies die reale Roboternutzung im Werksumfeld besser repräsentiert. Hintergrund ist, dass aus ökonomischen und instandhaltungstechnischen Gründen Robotertypen anwendungsseitig nicht immer an der Leistungsgrenze der Gewichtslast eingesetzt werden. Häufig wird sich auf ein Volumenmodell konzentriert, das mit Traglast und Reichweite ein möglichst großes Anwendungsfeld abdeckt und im realen Einsatz in der Produktion meist in einem mittleren statt im maximalen Auslastungsbereich arbeitet.

Darüber hinaus bietet die ISO 9283 keine Bewertungskriterien und Versuchsvorschriften für die Messung der Schaltgenauigkeit. Hierzu musste eine Versuchserweiterung entwickelt werden, die Peripherieansteuerung in die Leistungsmessung mit einbezieht. Ergänzend zur Norm wurden zusätzlich Parameter bezüglich der externen Schnittstellen in die Messreihen mit aufgenommen.

Messdesign

Aus Gründen der Praktikabilität der Labormessungen und der Relevanz für die fertigungstechnische Qualität wurde sich bei der Genauigkeitsmessung auf die Positions-wiederholgenauigkeit fokussiert. Ein Einbezug der Absolut- und Orientierungsgenauigkeit findet nicht statt, da dies die Anforderungen an das Messsystem und die Versuchsdurchführung in Bezug auf die gegebene Zielstellung unverhältnismäßig erhöht hätte und für die Beurteilung der fertigungsprozessualen Leistungsfähigkeit nur von geringerer Bedeutung ist. Beispielsweise bietet die Absolutgenauigkeit zwar den Vorteil, Robotersysteme leicht austauschen zu können und ohne Programmanpassungen der Zielpunkte fortzufahren, die Produktqualität wird aber insbesondere von der Fähigkeit des Systems beeinflusst, denselben Punkt möglichst genau wiederholt anzufahren.

[13] Tool Center Point (TCP) – Bezeichnung für den Werkzeugarbeitspunkt eines Roboters.

Versuchsaufbau

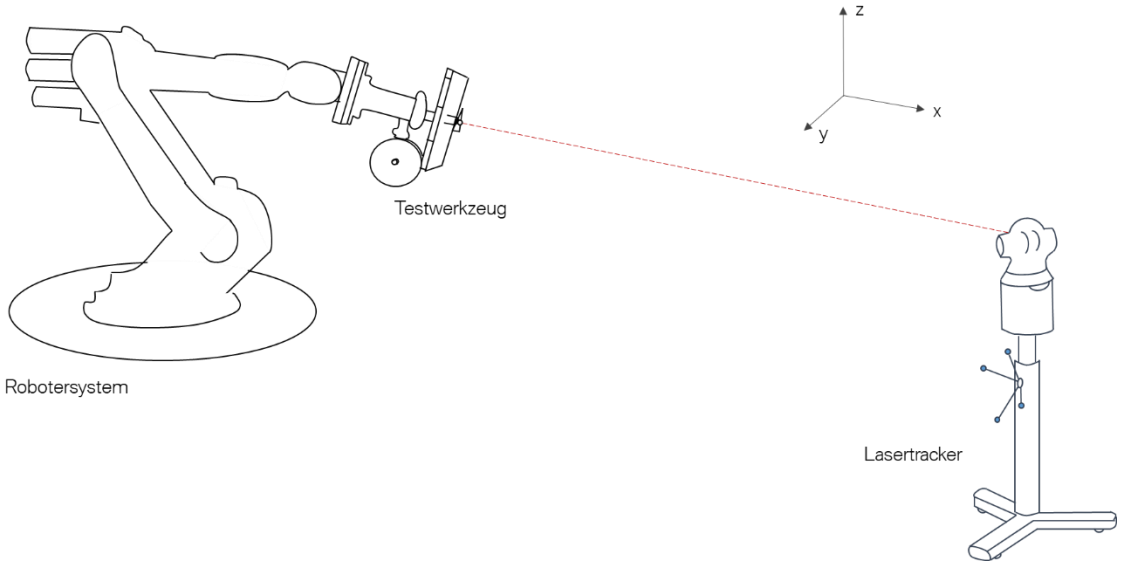


Abbildung 22 – Skizze Versuchsaufbau

Messmittel

Als Messwerkzeug zur Aufnahme der TCP-Positionen des Referenzwerkzeuges wurde der Lasertracker „FARO Laser Tracker ION“ verwendet. Dieser ermöglicht mit einer Auflösung von 0,5 µm und einer Abtastrate von bis zu 1000 Hz das Verfolgen der X-, Y-, Z-Koordinaten von statischen und dynamischen Raumpunkten, die mittels eines Retroreflektors erfasst und zurückgegeben werden. Die Genauigkeit hängt nicht nur vom Messsystem selbst, sondern auch von Umweltbedingungen wie Temperatur, Luftfeuchtigkeit, Luftdruck und dem Abstand zum Messobjekt ab. Für die im Versuchsaufbau gegebene geometrische Anordnung wird vom Hersteller laut Datenblatt eine Messgenauigkeit von mindestens 0,025 mm zugesichert. Gemäß ISO 9283 soll die Gesamtunsicherheit der Leistungsmessung für Industrieroboter nicht mehr als 25 % der zu messenden Größe überschreiten [111]. Die Erfassung von klassischen Robotergenauigkeiten im Bereich von 1/10 mm ist daher zwar prinzipiell möglich, für die ISO-konforme Messung von darunter liegenden Genauigkeitswerten sollte ein Eignungsnachweis für den gewählten Messprozess durchgeführt werden.

Für den konkreten Versuchsaufbau wurden daher Messunsicherheitsuntersuchungen durchgeführt und eine erweiterte Messunsicherheit von maximal 13 µm (Kuka-Messung) bzw. 21 µm (Comau-Messung) bestimmt. Demnach ist eine normkonforme Messung bis zu einer Genauigkeit von 52 µm bzw. 84 µm möglich. Da die von den Herstellern angegebenen Wiederholgenauigkeitswerte darüber liegen, ist der gewählte Messprozess folglich als geeignet einzustufen.

Zudem ist darauf hinzuweisen, dass die exakte Genauigkeitsermittlung nicht zentrales Ziel der Untersuchung ist, sondern der Vergleich von Genauigkeiten in Bezug auf die unterschiedlichen Steuerungskonzepte. Die sich für ein Robotersystem ergebenden Messungenauigkeiten erstrecken sich daher sowohl auf die Messreihen des nativen als auch des externe Steuerungssystems und fallen bei dem Vergleich von Durchschnittswerten gleichermaßen ins Gewicht.

Testwerkzeug

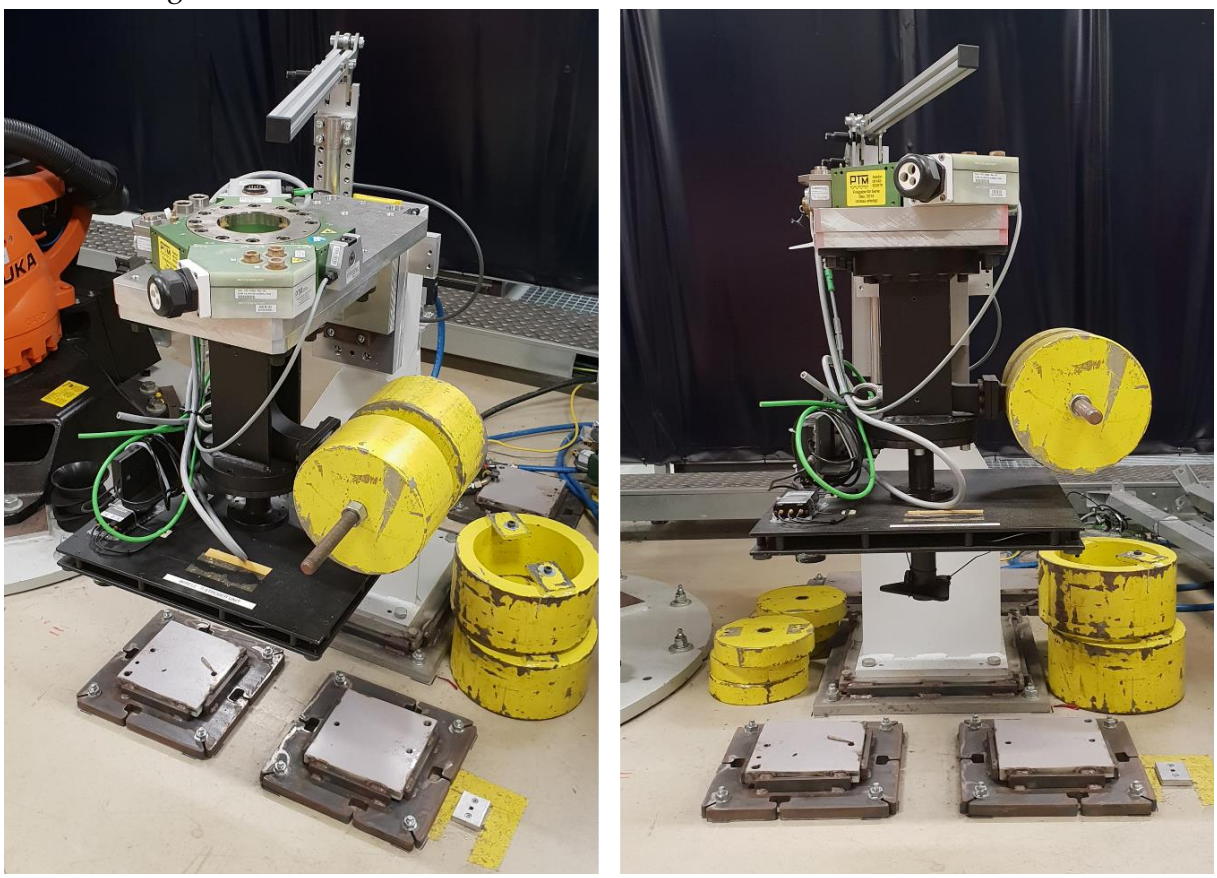


Abbildung 23 – Testwerkzeug mit entkoppeltem Gewicht und Aufnahme für den Retroreflektor des Lasertrackers

Das für die Versuche verwendete Testwerkzeug ist in Abbildung 23 dargestellt und kann über ein Werkzeugwechslersystem leicht zwischen den zu untersuchenden Robotersystemen getauscht werden. Das Testwerkzeug hat eine Gewichtslast von 100 kg, die in 10 kg Schritten bis zu 240 kg skalierbar ist. Um eine Beeinflussung der Messgenauigkeit durch die Trägheitskräfte im Werkzeug zu reduzieren, ist der Retroreflektor für den Lasertracker über eine mechanisch entkoppelte Fläche am Tool Center Point befestigt. Für die vorliegenden Versuche wurde das Messgewicht primär auf ein Gewicht von 140 kg eingestellt, da eine mittlere Traglast eine übliche Auslastung der Robotersysteme im Karosseriebau beträgt. Darüber hinaus sollte bewusst die ISO 9283 Gewichtsvorgabe der vollen Traglast vermieden werden, um etwaige ISO-Test spezifische Optimierungen der Hersteller zu reduzieren.

Robotersysteme

Für den Versuch wurden Robotersysteme der Firmen Comau und Kuka eingesetzt, die in Traglast (220 bis 240 kg) und Reichweite (2,4 bis 2,5 m) der Volumenmodellklassifizierung des Karosseriebaus entsprechen. Detaillierte Leistungsdaten zu den Robotersystemen befinden sich auf Seite 154. Die beiden Robotersteuerungen bieten mit RSI- und C5GOpen-Schnittstellen auf Interpolationsebene, deren Ansteuerungszyklus variierbar ist. Die Roboter können daher sowohl isoliert über die konventionelle Robotersteuerung des Herstellers (nativer Betrieb) oder über zusätzliche externe Systeme gesteuert werden (externer Betrieb).

Für die Abfahrt der Testbahnen wurde in beide Roboter ein variables ISO-Cube-Programm implementiert, das sämtliche geometrische Details und Untersuchungsgrößen als einstellbare Parameter vorhält und so eine schnelle und flexible Anpassung des Testaufbaus ermöglicht.

Externes Steuerungssystem

Als externes Steuerungssystem wurde ein Industrie-PC mit einem Linux-Betriebssystem und einem für Echtzeitanwendungen gepatchten Kernel[14] verwendet, um die Zeitanforderungen der Roboterschnittstellen zu gewährleisten. Die Roboterfunktionalitäten der Bahnplanung, Inverskinematikberechnung und Trajektorieninterpolation werden mit der Robotics Library (S. 22) durchgeführt. Diese wurde im Rahmen des Prototyps (Kapitel 7.3) mit den notwendigen Robotertreibern erweitert. Darauf aufbauend wurde analog zu den nativen Steuerungen das variable ISO-Cube-Programm im externen Steuerungssystem implementiert. Dank der Robotersystemunabhängigkeit musste dies nur einmal für beide Robotersysteme statt individuell für jeden Hersteller einzeln erstellt werden.

Prüfbahnen und Prüfkriterien

Wie bereits erwähnt, ist eine vollumfassende Analyse der 19 Leistungskriterien der ISO-Norm nicht zielführend. Die zu untersuchenden Kriterien wurden daher auf die relevanten beschränkt und um den Kennwert der Schaltgenauigkeit für Bahnprozesse und Einstellparameter bzgl. der Interpolationsschnittstelle und Peripherieanbindung erweitert. Grundlage für alle Untersuchungen sind die durch den ISO-Cube definierten Bewegungsbahnen für die Ermittlung von Pose- und Bahnkenngrößen. Neben den klassischen, einstellbaren Parametern wie Geschwindigkeit wurden aufgrund des externen Ansteuerungskonzeptes weitere Untersuchungsgrößen in Form von Zykluszeit der Schnittstelle, kartesischer bzw. achsbasierter Ansteuerung und verschiedener Beschleunigungs- und Bremsprofile integriert. Zudem wurde für jede Testreihe die Bahncharakteristik verglichen, um die Abstraktionsleistung in Bezug auf die herstellerunabhängige Bahnplanung zu diskutieren und zu bewerten. Eine Übersicht der durchgeführten Versuche und Untersuchungsgegenstände ist in Tabelle 4 dargestellt. Die Beschreibung der gewählten Testbahnen ist ebenfalls in den jeweiligen Abschnitten aufgeführt.

Tabelle 4 – Versuchsübersicht

Bewegungstyp	Testbahn Nr. (Seite)	Untersuchungsgegenstand
Punkt-zu-Punkt	Testbahn 1 (S. 73)	Bahncharakteristik
		Bahncharakteristik (Bewegungsprofile)
		Geschwindigkeit und Beschleunigung
		Wiederholgenauigkeit
		Stabilisierungszeit / Überschwingen
		Zykluszeit Interpolationsschnittstelle
Linearbahn	Testbahn 2 (S. 80)	Bahncharakteristik
		Geschwindigkeit
		Genauigkeit
		Zykluszeit Interpolationsschnittstelle
	Testbahn 2 + Aktorschaltung (S. 83)	Schaltgenauigkeit
		Zykluszeit Interpolationsschnittstelle
		Zykluszeit Peripherie
Kreisbahn	Testbahn 3 (S. 86)	Bahncharakteristik
		Kennwerte
Überschleifen	Testbahn 4 (S. 86)	Bahncharakteristik
		Einstellbarkeit

[14] PREEMT_RT Patch [112]

Für die Versuchsfahrten zur Ermittlung von ISO-Kriterien wurden die Testbahnen jeweils 30-mal abgefahren. Dies wurde in drei sequenziellen Versuchen durchgeführt und das Robotersystem vorab in einer 15-minütigen Dauerfahrt betrieben. Ziel war es, thermische Effekte, die beim Start des Robotersystems auf die Genauigkeit Einfluss nehmen können, zu minimieren und einzelne Messstreuungen auszugleichen.

Bewegungsprofile

Essenzielle Auswirkung auf die Charakteristik und damit ebenso auf die Qualität der Bahn einer Roboterbewegung hat die Art und Weise der Beschleunigung der Antriebsachsen. Durch die Externalisierung der Bahnplanung auf Interpolationsebene entfällt die Vorgabe von Geschwindigkeit und Beschleunigung implizit auf das externe Steuerungssystem. Eine gute Orientierung für die Ausführung der Bewegungen auf Gelenkebene bietet die VDI 2143 [113] mit den Bewegungsgesetzen für Kurvengetriebe. Diese umfassen grundlegende Gestaltungsrichtlinien und deren mathematische Hintergründe, um Bewegungen nach unterschiedlichen Aspekten auszulegen und zu berechnen. Im Kern wird hierbei mittels analytischen Funktionen beschrieben, wie sich Position, Geschwindigkeit und Beschleunigung einer Getriebebewegung im Zeitverlauf verhalten und welche Kräfte und Schwingungen auf eine Maschine einwirken. In der Regel sind Ziel, Stoß- und Ruckzustände[15] zu vermeiden und die Performance des Systems zu optimieren. Die VDI-Norm ist lediglich eine Grundlage. Die aufgeführten Regeln stellen folglich nur einen Teilausschnitt dar und es existieren noch zahlreiche weitere Ausführungen von Bewegungsprofilen. Deren Gestaltung und Optimierung ist Kern-Know-how eines Roboterherstellers und geht durch die Externalisierung im Steuerungsverbund des Versuchsaufbaus verloren und muss daher innerhalb des externen Steuerungssystems reimplementiert werden. Neben dem Nachteil des erhöhten Aufwands und der damit zu beantwortenden Fragestellung, ob eine ausreichend hohe Performance für den Produktionsbetrieb gewährleistet ist, bringt die Verwendung roboterherstellerunabhängiger Beschleunigungsprofile den Vorteil, Charakteristiken der Manipulatorbewegungen zu beeinflussen und diese auf die auszuführende Aufgabe zu optimieren.

Die Auslegung und Optimierung von Bewegungsgesetzen auf ein spezifisches System stellt eine aufwändige und komplexe Aufgabe dar, welche nicht Fokus dieser Arbeit ist. Aus diesem Grund werden für die Laboruntersuchungen untransformierte Standardfunktionen verwendet. Diese werden im Folgenden kurz erläutert.

Polynom 3. und 5. Ordnung für Rast-in-Rast-Bewegung

Für Rast-in-Rast Bewegungen werden kubische und quintische Polynome verwendet. Insbesondere die 5. Potenz ist ein in der Praxis häufig angewandtes Bewegungsgesetz, da es ruckfrei, leicht herzuleiten und relativ ausgewogen ist. Das Polynom 3. Ordnung hingegen ist bei Start und Rückführung in die Rast ruckbehaftet, welche sich aus nicht stetigen Änderungen der Beschleunigung ergibt. Die Ableitung der Beschleunigung (Ruckfunktion) weist einen endlichen Beschleunigungssprung auf. Nachteilig erweist sich bei beiden Polynomfunktionen, dass die maximal erreichbaren Geschwindigkeiten nur für einen kurzen Moment am Wendepunkt erreicht und nicht über einen längeren Zeitraum ausgeführt werden. Zur besseren Verständlichkeit und Interpretierbarkeit sind in Abbildung 24 die Übertragungsfunktionen für Position, Geschwindigkeit, Beschleunigung und Ruck der verwendeten Polynomfunktionen dargestellt.

[15] Unter Stoß ist ein theoretisch unendlicher Beschleunigungssprung zu verstehen, während ein Ruck ein endlicher Sprung in der Beschleunigungsfunktion (Zweite Ableitung des Weges/Winkels) darstellt.

Polynome 4. und 6. Ordnung mit Rast-in-Geschwindigkeit-in-Rast-Bewegung

Für die dauerhafte Ausführung einer konstanten, idealerweise maximalen Geschwindigkeit bietet sich die Kombination von Bewegungsgesetzen für „Rast-in-konstante Geschwindigkeit“ und „konstante Geschwindigkeit-in-Rast“-Übertragungsfunktionen an. Dabei wird eine Polynomfunktion verwendet, um die Geschwindigkeitsspitze zu erreichen und mit einer linearen Übertragungsfunktion konstant abgefahren zu werden, um abschließend mit einer Polynomfunktion erneut in einen Rastzustand überzugehen.

Innerhalb der Arbeit werden hierfür einmal quartische und einmal sextische Polynome verwendet. Erstere bieten analog zu den Polynomen 5. Ordnung den Vorteil, keinen Beschleunigungssprung und damit einen ruckfreien Bewegungsverlauf zu haben. Die Funktion des 6. Grades vermeidet Rucksprünge sogar komplett und bietet durch den stetigen Verlauf der vierten Ableitung einen noch weicheren Beschleunigungsverlauf. Dies geht jedoch mit einem späteren Erreichen der Maximalgeschwindigkeit einher oder, analog zu der Übersichtsdarstellung in Abbildung 24, mit einem höheren Beschleunigungsbedarf, um zeitgleich mit der quartischen Bewegung das Geschwindigkeitsplateau zu erreichen.

Im nachfolgenden Teil der Arbeit werden für die bessere Lesbarkeit die erläuterten Bewegungsprofile wie folgt abgekürzt:

P3 - Polynom 3. Ordnung
P5 - Polynom 5. Ordnung
P414 - Polynome 4. Ordnung mit konstanter Geschwindigkeit
P616 - Polynome 6. Ordnung mit konstanter Geschwindigkeit

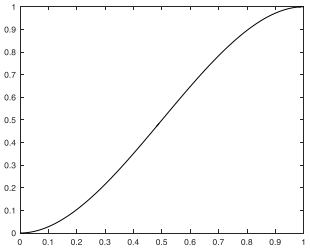
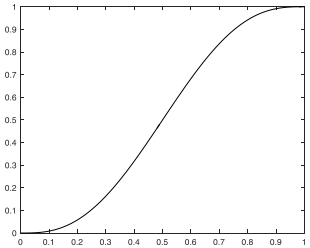
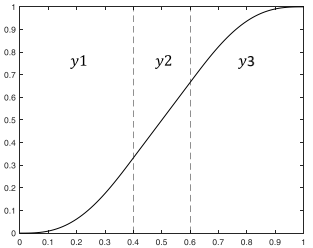
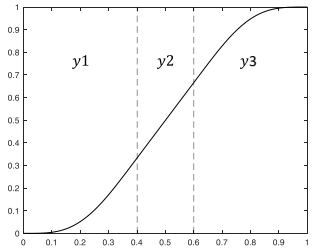
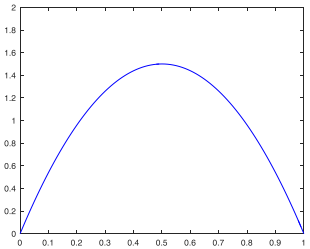
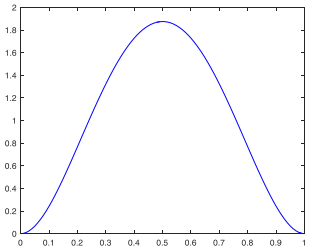
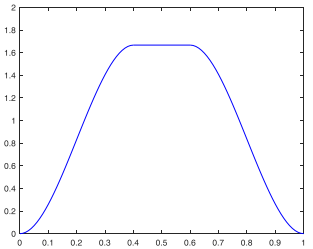
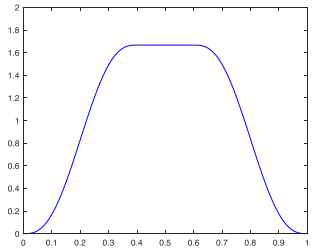
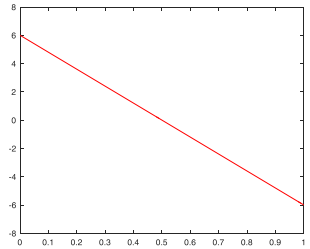
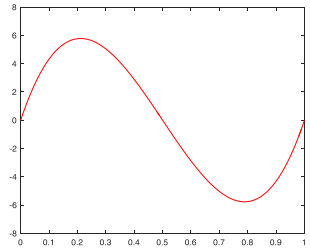
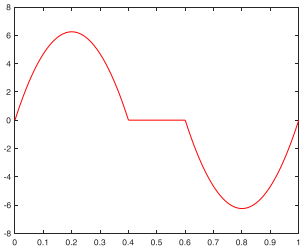
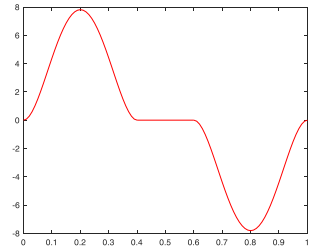
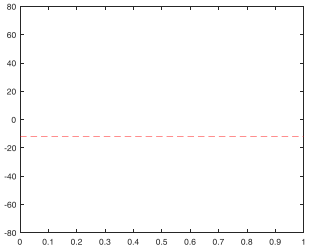
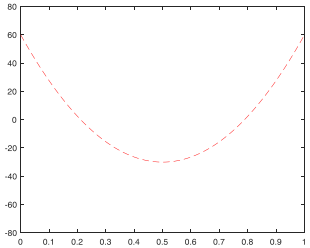
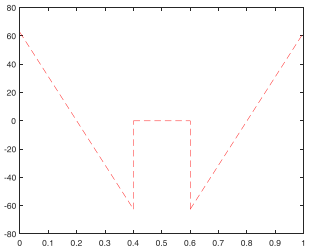
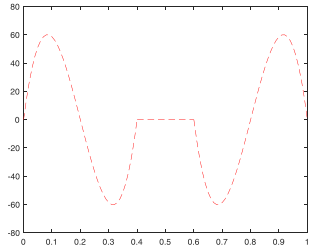
	P3	P5	P414	P616
Position	 $y = ax^3 + bx^2 + c$	 $y = ax^5 + bx^4 + cx^3 + dx^2 + ex + f$	 $y1, y3 = ax^4 + bx^3 + cx^2 + dx + e$ $y2 = vx$	 $y1, y3 = ax^6 + bx^5 + cx^4 + dx^3 + ex^2 + fx + g$ $y2 = vx$
Geschwindigkeit	 $y' = 3ax^2 + 2bx$	 $y' = 5ax^4 + 4bx^3 + 3cx^2$	 $y1', y3' = 4ax^3 + 3bx^2$ $y'2 = v$	 $y1', y3' = 6ax^5 + 5bx^4 + 4cx^3$ $y'2 = v$
Beschleunigung	 $y'' = 6ax + 2b$	 $y'' = 20ax^3 + 12bx^2 + 6cx$	 $y1'', y3'' = 12ax^2 + 6bx^2$ $y''2 = 0$	 $y1'', y3'' = 30ax^4 + 20bx^3 + 12cx^2$ $y2'' = 0$
Ruck	 $y''' = 6a$	 $y''' = 60ax^2 + 24bx$	 $y1''', y3''' = 24ax + 12b$ $y'''2 = 0$	 $y1''', y3''' = 120ax^3 + 60bx^2 + 24cx$ $y'''2 = 0$

Abbildung 24 – Übertragungsfunktionen für Rast-in-Rast und Rast-in-Geschwindigkeit-in-Rast-Bewegungen mit Polynomen

Auswertungswerkzeugkette

Zur Aufnahme der Lasertrackermessungen wurde die Software Cam2Measure von Faro verwendet, welche allerdings keine Leistungsbeurteilung nach den ISO-Kriterien ermöglicht. Aus diesem Grund musste zur Auswertung der Leistungskennwerte und graphischen Betrachtung der aufgenommenen Bahnen eine eigene Werkzeugkette entwickelt werden, welche ein automatisiertes Einlesen und Verarbeiten der Rohdaten des Faro-Systems ermöglicht und folgende Funktionen aufweist:

- Formatieren der Messdaten
- Vorfiltern und Korrigieren von Verarbeitungsfehlern der Messdaten
- Interpolation und Normalisieren der Messpunkte
- Berechnen der ISO-Genauigkeitswerte für Punkt-, Linear- und Kreisbahnen
- Berechnen der Schaltgenauigkeit für Linearfahrten
- Graphische Darstellung der Messreihen

Mit RoboDK [114] und RoboDyn [115] existieren zwar inzwischen ebenfalls Softwareprodukte, die eine Performanceanalyse nach ISO 9283 Standard ermöglichen, jedoch waren zum einen die ISO-Auswertefunktionen zu Beginn der Leistungsmessungen z. T. noch nicht erhältlich. Zum anderen sind die geforderten Analyseanforderungen der durchzuführenden Tests nicht alle abgedeckt. So bieten die Tools z. B. keine Auswertung der zur ISO ergänzten Untersuchungen bzgl. der Schaltgenauigkeit. Ebenso existieren keine spezifischen Bedarfe wie die Betrachtung der Stabilisierungsphase von Anfahrpunkten oder Funktionen zur automatisierten Verarbeitung und Auswertung einer großen Anzahl an Messreihen.

Ergebnisse und Bezeichnungen

Die Resultate der Messungen werden im nachfolgenden Teil der Arbeit erläutert. Zur besseren Verständlichkeit wurden die Bezeichnungen wie folgt gewählt:

Kuka: Messungen des nativen Kuka-Herstellersystems, ohne externe Steuerungstechnik.
KukaR: Messungen des extern gesteuerten Kuka-Robotersystems.

Comau: Messungen des nativen Comau-Herstellersystems, ohne externe Steuerungstechnik.
ComaR: Messungen des extern gesteuerten Comau-Robotersystems.

Die dahinter folgenden Abkürzungen P3, P5, P415 und P616 beschreiben die Art der gewählten Bewegungsprofile im externen Steuerungsmodus. Für einige Tests wurde für eine bessere Vergleichbarkeit die Absolutgenauigkeitsfunktion in den nativen Robotersystemen bewusst deaktiviert. Diese Messreihen sind mit dem Kürzel „o.Ag.“ (ohne Absolutgenauigkeit) gekennzeichnet.

Die Ergebnisse werden anhand der in Tabelle 4 dargestellten Reihenfolge besprochen. Dabei werden zunächst die abzufahrende Testbahn und Kenngrößen grundsätzlich beschrieben und deren Bedeutung für den realen Fertigungsprozess erläutert. Anschließend werden die Bahncharakteristiken der getesteten Systeme verglichen und die ermittelten Leistungskennwerte besprochen. Aufgrund des Umfanges der Messungen werden die zentralen Resultate im Rahmen des Hauptteils der Arbeit dargestellt. Die detaillierten Ergebnisse inklusive Wertetabellen, Bahnkurven und weiteren Vergleichsgrafiken befinden sich in Anhang 9.1 und 9.2.

5.2.2. Test der Posekenngrößen

Bei der Testbahn für Posekenngrößen werden fixe Positionen im ISO-Cube (Abbildung 25) hintereinander wiederholt und ausgehend vom Mittelpunkt (Punkt 1) in absteigender Reihenfolge abgefahren.

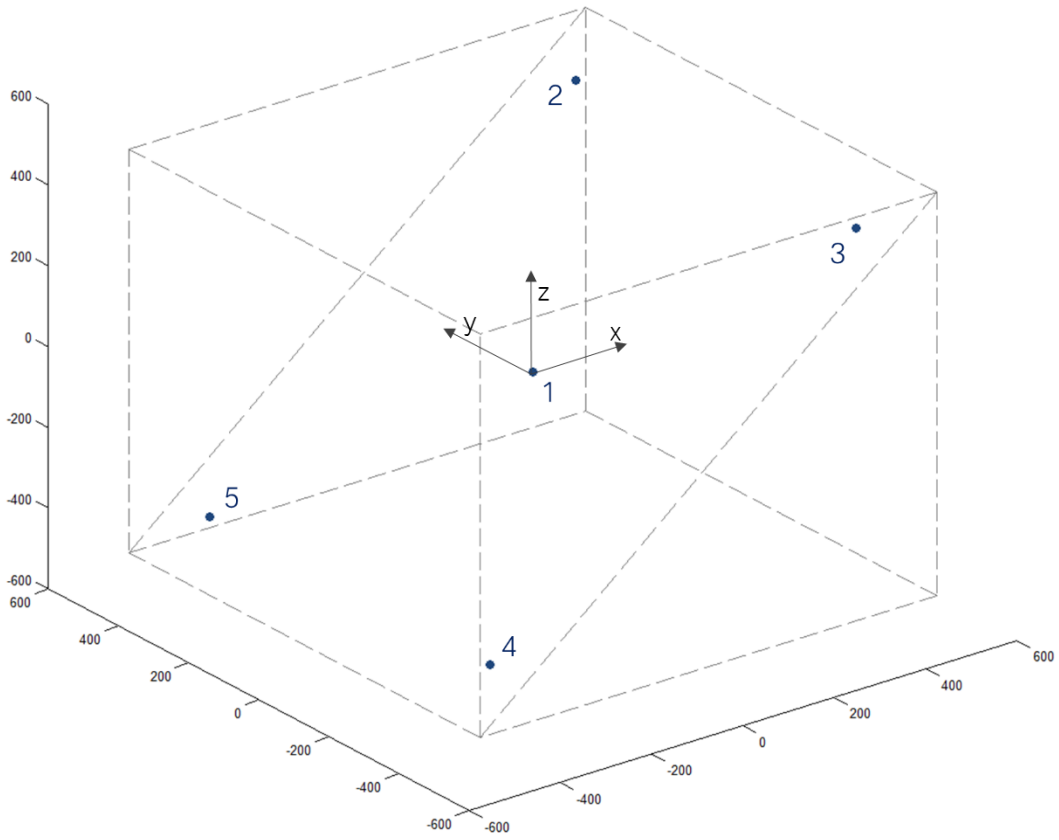


Abbildung 25 – ISO-Cube mit Zielpunkten für Pose-Tests

Zentrales Ziel ist die Erfassung der Differenz von Soll- und Ist-Positionen und die Ermittlung des Genauigkeit- bzw. des Wiederholgenauigkeitskennwertes. Der Unterschied zwischen diesen besteht darin, ob die vom Roboter angefahrenen Ist-Positionen mit dem Schwerpunkt der real angefahrenen Positionen (Pose- bzw. Positionswiederholgenauigkeit) oder mit einem idealen Sollpunkt (Pose- bzw. Positionsgenauigkeit), also dem programmierten Zielpunkt, verglichen werden. Für den regulären Produktionsbetrieb ist insbesondere die Wiederholgenauigkeit von Bedeutung, da diese die Abweichung einer fest einprogrammierten dauerhaft abgefahrenen Roboterbahn beschreibt und damit direkt Auswirkung auf die Produktqualität haben kann. Die Positions-Genauigkeit, auch Absolutgenauigkeit genannt, ist aus Inbetriebnahme- und Instandhaltungssicht von Bedeutung, da ein guter Kennwert eine geringe Differenz des realen Systems zum idealen Sollpunkt beschreibt und kürzere Reteachzeiten bei Erstinbetriebnahme oder Robotertausch zur Folge hat. Weitere Kriterien zur Beurteilung der Absolutgenauigkeit stellen die Kenngrößen Abstandsgenauigkeit und Austauschbarkeit dar, welche die Abweichung des Abstands zweier Punkte eines Robotersystems bzw. den Genauigkeitsvergleich zweier unterschiedlicher Robotersysteme des gleichen Typs beschreiben. Neben der Bewertung der reinen Endposition bieten Positions-Stabilisierungszeit und Positions-Überschwingung Untersuchungskriterien, um die Qualität der Positionserreichung zu beurteilen. Diese sind aus Gesichtspunkten der Taktzeit und der generellen Beanspruchung relevant, da eine schnellere

Stabilisierung einen früheren Start des darauffolgenden Prozessschrittes ermöglicht und geringere Schwingungen zu einer reduzierten mechanischen Belastung führen. Im Vergleich zu der Gesamtprozesszeit und -belastung der Robotersysteme wirken diese sich nur sehr gering aus. Die beschriebenen Posekenngrößen stellen ebenfalls Bewertungskriterien der technischen Anforderungen für Robotersysteme der BMW AG dar und sind daher auch aus Anwendersicht für den Einsatz von Robotern im Produktionsumfeld bzw. alternativer Steuerungslösungen von Bedeutung. Neben den klassischen ISO-Kriterien soll zusätzlich die allgemeine Trajektoriencharakteristik betrachtet werden, um die Auswirkung der unterschiedlichen Bahnplanung zwischen nativer Robotersteuerung und externem Steuerungssystem zu betrachten und die Modifizierbarkeit aufzuzeigen. Zudem wird mit Testbahn 1 ebenfalls eine Geschwindigkeits- und Beschleunigungsbetrachtung durchgeführt.

Von zentraler Bedeutung ist aus den genannten Gründen die Wiederholgenauigkeit (RP):

$$RP_i = \bar{l}_i + 3S_i;$$

$\boldsymbol{RP_i}$: *Positionswiederholgenauigkeit für Pose i*

$$S_i = \sqrt{\frac{\sum_{j=1}^{n}\left(l_{ij}-\bar{l}_i\right)^2}{n-1}}$$

$\boldsymbol{i}$: *Poseindex* $\boldsymbol{j}$: *Zyklusindex*

$\boldsymbol{n}$: *Anzahl Messzyklen*

$\boldsymbol{S_i}$: *Standardabweichung*

$$\bar{l}_i = \frac{1}{n}\sum_{j=1}^{n} l_{ij}$$

$\bar{\boldsymbol{l}}_{\boldsymbol{i}}$: *Mittelwert der Abstände für Pose i*

$\boldsymbol{l_{ij}}$: *Abstand zwischen Ist- und Schwerpunkt der j-ten Pose für Position i*

$$l_{ij} = \sqrt{\left(x_{ij}-\bar{x}_i\right)^2 + \left(y_{ij}-\bar{y}_i\right)^2 + \left(z_{ij}-\bar{z}_i\right)^2}$$

$\bar{\boldsymbol{x}}_{\boldsymbol{i}}, \bar{\boldsymbol{y}}_{\boldsymbol{i}}, \bar{\boldsymbol{z}}_{\boldsymbol{i}}$: *Schwerpunkte der Koordinaten, nach n-facher Wiederholung für Pose i*

$$\bar{x}_i = \frac{1}{n}\sum_{j=1}^{n} x_j\,; \quad \bar{y}_i = \frac{1}{n}\sum_{j=1}^{n} y_j\,; \quad \bar{z}_i = \frac{1}{n}\sum_{j=1}^{n} z_j$$

$\boldsymbol{x_j}, \boldsymbol{y_j}\ \boldsymbol{z_j}$: *Koordinaten der j-ten Ist-Pose*

Ergebnisse

Bahncharakteristik – native Roboterbahn im Vergleich zur externen Ansteuerung

Die Untersuchung der Robotersysteme hinsichtlich der Bahncharakteristik (Anhang 9.1.1) zeigte zunächst, dass die vorgegebenen Punkte der ISO-Cube-Testbahn grundsätzlich vom Kuka- und vom Comau-Robotersystem abgefahren werden konnten. Dies war sowohl mit dem nativen Robotersteuerungs- als auch dem externen Steuerungssystem möglich. Auffallendster Unterschied stellte der Kurvenverlauf zwischen den Punkten dar, da dieser angesichts der unterschiedlichen Kinematik je nach Robotersystem unterschiedlich ausfällt (Anhang - Abbildung 78). Der Kuka-Roboter fuhr die Testbahn ohne Haltepunkte in einer Zeit von 4,67 Sekunden ab, während der Comau-Roboter 5,34 s in der höchsten Geschwindigkeitseinstellung benötigte. Der unterschiedliche Verlauf zeigt sich sowohl in der nativen als auch der externen Ansteuerung, wobei für beide Roboter die externe Bahn im Vergleich zur jeweiligen nativen Bahn einen deutlichen Versatz (primär in Z-Richtung) aufwies. Dieser betrug beim Comau-Robotersystem 9,1 mm und für das Kuka-System 2,7 mm. Ursache hierfür liegt in der Absolutgenauigkeitskorrektur, die bei externer Ansteuerung von den Robotersystemen nicht durchgeführt wird. Die Deaktivierung der Absolutgenauigkeit in der nativen Robotersteuerung

führte bei beiden Robotersystemen dementsprechend auch zu einer Reduktion der Abweichung auf ca. 0,3 mm im Vergleich zum externen Steuerungskonzept (Anhang - Tabelle 14).

Bahncharakteristik – Einfluss der Bewegungsprofile (P3, P5, P414, P616)

Die Wirkung unterschiedlicher Bewegungsprofile zeigte sich insbesondere bei der Anfahrt der Haltepunkte und ist in Anhang 9.1.2 dargestellt. Insbesondere zwischen den stark unterschiedlichen Profilen P3 und P616 konnte beobachtet werden, dass sich über die externen Ansteuerungsmethoden und Bremsprofile das Verhalten der Roboterbahnen signifikant beeinflussen lässt und anwendungsabhängig auf beispielsweise reduzierte Fahrtdauer (P3) oder reduziertes Schwingungsverhalten (P616) optimieren lässt. Dies ist mit herkömmlichen Robotersystemen in dieser Form als Endkunde nicht möglich, bzw. kann nur implizit über die Anpassung der Geschwindigkeitskennwerte erfolgen.

Leistungskennwerte

Im nächsten Untersuchungsschritt wurden die Roboterbahnbewertung und der Vergleich zwischen den verschiedenen Ansteuerungsmöglichkeiten und Bahnprofilen durch die Erfassung von Kennwerten quantifiziert. Die Gesamtübersicht der Messungen und Bestimmung der Leistungswerte für unterschiedliche Parameter sind in Anhang 9.1.3 dargestellt und besprochen.

TCP-Geschwindigkeit und Beschleunigung

Zur Ermittlung der TCP-Geschwindigkeit und Beschleunigung wurden die nativen Testbahnen mit der größtmöglichen Programmgeschwindigkeitsvorgabe der nativen Robotersysteme abgefahren. Anschließend wurden die extern gesteuerten Systeme mit größtmöglicher Geschwindigkeit und Beschleunigung abgefahren, wobei diese iterativ bestimmt wurden, in dem die Werte kontinuierlich bis zur Leistungsgrenze der Robotersysteme erhöht wurden. Dabei zeigte sich ein unterschiedliches Verhalten. Während der Kuka-Roboter bei zu hohen Beschleunigungswerten stoppte und die Fehlermeldung einer Drehmomentverletzung ausgab, konnte der Comau-Roboter über die Leistungsgrenze hinaus gefahren werden und gab zunächst nur Warnmeldungen bzgl. zu hoher Motorströme aus. Bei einer weiteren Erhöhung der Beschleunigung folgte ein Not-Halt-Eingriff aufgrund von Positionsfehlern.

Die Darstellung der Geschwindigkeits- und Beschleunigungskurven zeigt (Anhang - Abbildung 87 bis 90), dass mit den extern gesteuerten Robotersystemen grundsätzlich vergleichbare Leistungswerte erreicht werden können wie mit dem rein nativen Steuerungskonzept. Durch die Beeinflussung der Beschleunigungsprofile wurden für die Untersuchungsbahn auch höhere TCP-Beschleunigungs- (P3) oder Geschwindigkeitswerte (P616) im Vergleich zum nativen Herstellersystem erreicht. Ebenso lag die Gesamtfahrtdauer der Testbahnen mit den verschiedenen Steuerungskonzepten in gleicher Größenordnung und variierte entsprechend des gewählten Beschleunigungsprofils. So war bei dem Kuka-System eine externe P3-Bahn mit 17,3 s leicht schneller als die native Steuerung (17,5 s) und eine ruckoptimierte P616-Bahn leicht langsamer (17,9 s). Ein ähnliches Verhalten zeigte sich beim Comau-System.

Es lässt sich zusammenfassend festhalten, dass anders als bei Werner [110] keine Limitierung der Geschwindigkeit oder Beschleunigung der extern gesteuerten Systeme vorlag und dass durch die Auswahl des Beschleunigungsprofils ein erheblicher Einfluss auf die Roboterbahn genommen werden kann.

TCP- Wiederholgenauigkeit

Einer der bedeutendsten Kennwerte in der Bewertung von Robotersystemen und Hauptbestandteil der ISO 9283 ist die Genauigkeit eines Robotersystems. Diese kann wie erwähnt in unterschiedlichen Ausprägungen definiert sein. Die Pose- bzw. Positionsgenauigkeit und die Bahngenauigkeit sind hier sowohl in der Absolut- als auch Wiederholreferenz zu nennen. Insbesondere die Positionswiederholgenauigkeit wird in Industrie und Praxis als zentraler Kennwert angesehen und ist daher auch in jeder Roboterspezifikation zu finden. Für die in der Untersuchung verwendeten Roboter geben die Hersteller eine Wiederholgenauigkeit von ± 0,06 mm (Kuka) und ± 0,15 mm (Comau) an. Ob extern angesteuerte Roboter für industrielle Anwendungen eingesetzt werden können, hängt vor allem von der Erreichung dieser Leistungsgröße ab.

Abbildung 26 zeigt die Anfahrt der Zielpunkte 5 bis 1 für das Kuka-Robotersystem der Testbahn mit höchstmöglicher Geschwindigkeit und einer Haltezeit von 3 Sekunden, sowohl für den nativen als auch externen Betrieb mit P3- und P616-Bewegungsprofilen. Die horizontale grüne Linie visualisiert die zu erfüllende Wiederholgenauigkeit. Die eingezeichnete Roboterbahn verfärbt sich von Rot zu Blau, sobald die Stabilisierungsphase abgeschlossen ist und damit kein Überschwingen des TCP über die einzuhaltende Wiederholgenauigkeit mehr erfolgt.

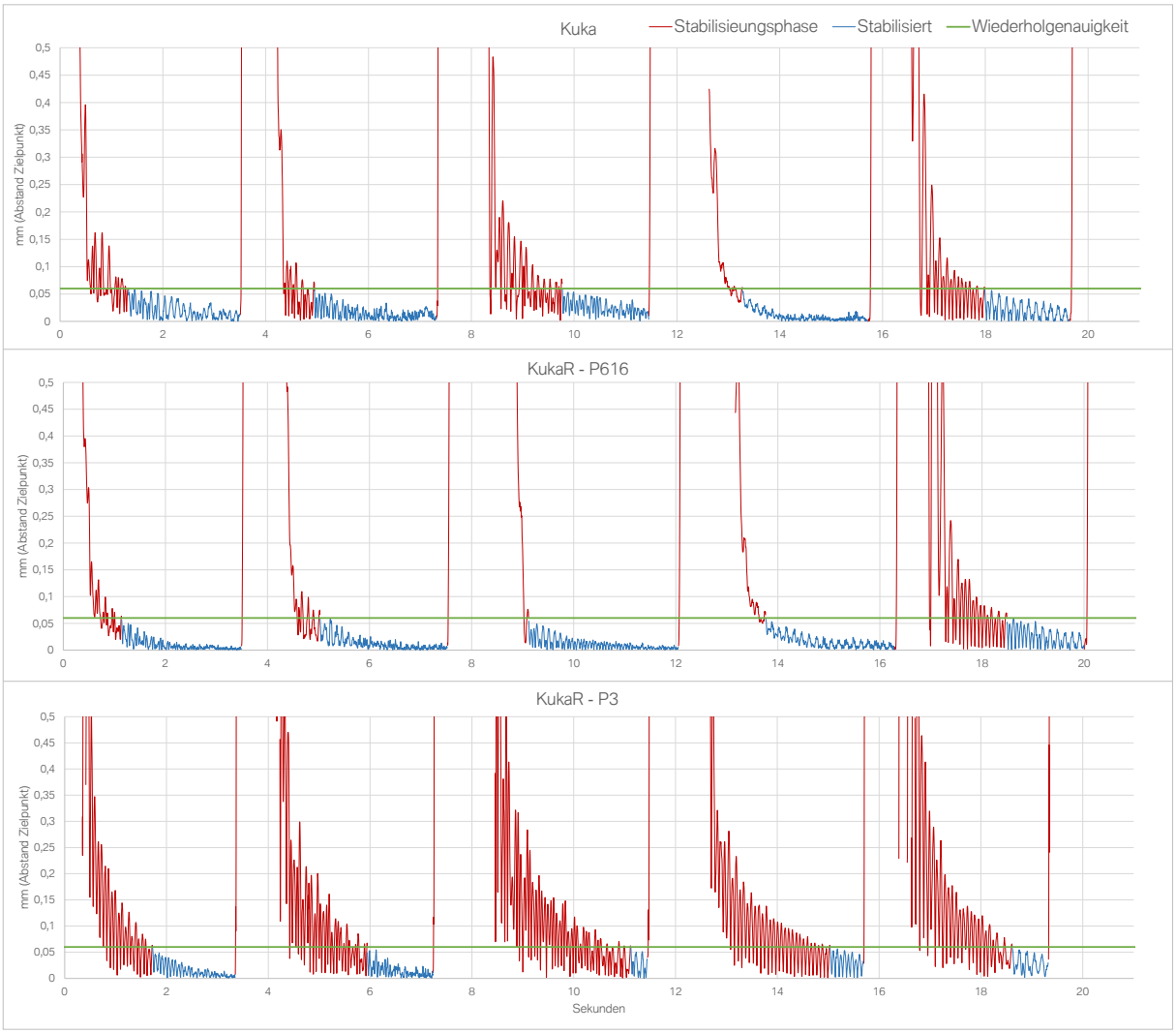


Abbildung 26 – Stabilisierungsprozess des Kuka-Roboters für eine ISO-Cube-Pose-Fahrt

Die Messergebnisse zeigen, dass diese Genauigkeitsangaben von beiden Robotersystemen sowohl nativ als auch extern grundsätzlich eingehalten wurden, jedoch die Wahl des Beschleunigungsprofils Auswirkungen auf die Dauer der Haltephasen hat. So ist die Stabilisierungsdauer des nativen Kuka-Roboters und des extern gesteuerte Kuka-Roboters mit einem P616-Bewegungsprofil deutlich kürzer als bei einem P3-Bewegungsprofil, welches dafür die Durchfahrt der Testbahn am schnellsten absolviert. Dabei ist anzumerken, dass es sich hier um keine direkte Leistungsverbesserung des Robotersystems handelt, sondern vielmehr es dem Anwender ermöglicht wird, das Bahnverhalten gezielt zu parametrieren (vgl. Anhang S.181).

Bei dem Comau-Robotersystem wurde im nativen und externen Betrieb die vorgegebene Genauigkeit des Herstellersystems erreicht (Abbildung 27), allerdings dauert das Erreichen des Zielwerts aufgrund der Regelungsmethodik des Comau-Antriebssystems wesentlich länger als im Kuka-System, obwohl der einzuhaltende Kennwert mit 0,15 mm im direkten Vergleich mehr als doppelt so hoch ist. Das untersuchte Comau-Robotersystem kommt z. T. bereits bei einem Abstand von 0,5 mm zum Stillstand und nähert sich in den folgenden Sekunden erst dem Zielpunkt, wobei auch in der finalen Position Regelungsschwankungen von bis zu 0,1 mm auftreten. Die Anfahrtskurven für die unterschiedlichen Betriebsmodi des Comau-Robotersystems sind im Anhang aufgeführt (Abbildung 83 bis 86).

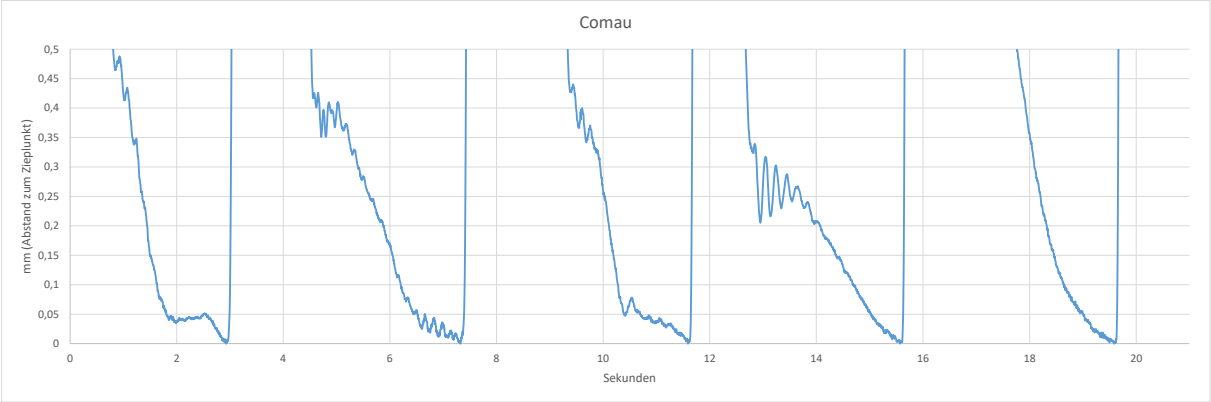


Abbildung 27 – Stabilisierungsprozess des Comau-Roboters für eine ISO-Cube-Pose-Fahrt

Dieses Bahnverhalten wird insbesondere bei längeren Wartezeiten deutlich sichtbar (Abbildung 28) und tritt sowohl im nativen Betrieb als auch bei externer Ansteuerung auf. Es zeigt sich, dass die potenziellen Genauigkeitsunterschiede durch die externe Steuerungsmethode und Bewegungsprofilwahl unbedeutend im Vergleich zur grundlegenden Antriebsregelung bei diesem Robotersystem sind.

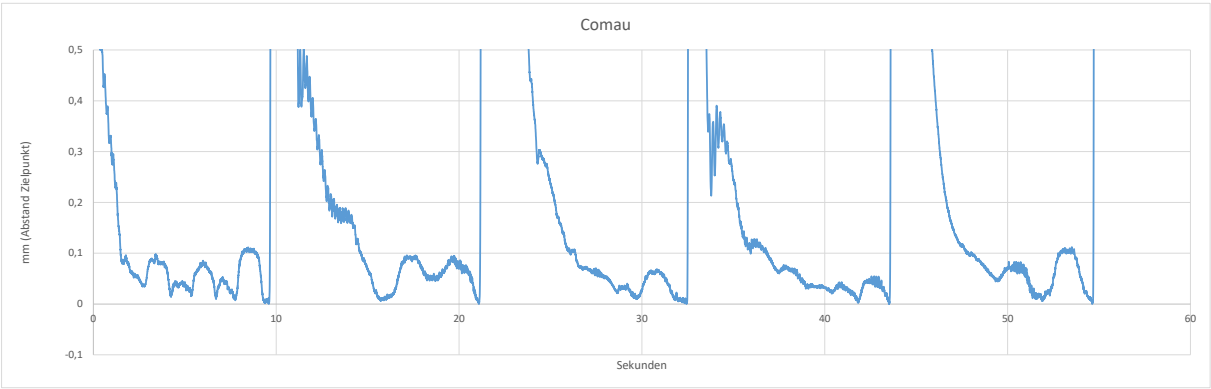


Abbildung 28 – Stabilisierungsprozess des Comau-Roboters für eine ISO-Cube-Pose-Fahrt mit 10 s Haltezeit

TCP-Stabilisierungszeit und Überschwingverhalten

Neben der reinen Positionsgenauigkeit spielt auch die Dauer des TCP-Einschwingvorgangs bei einer Posefahrt eine wichtige Rolle. Dies wird in der ISO-Norm mit Hilfe der Stabilisierungszeit bezeichnet. Das System gilt als positioniert sobald ein vorgegebener Genauigkeitskennwert dauerhaft unterschritten ist und nicht mehr überschwungen wird. Dabei ist sowohl die Anzahl an Überschwingungen als auch der größte Abstand eines Überschwungvorgangs (Überschwinghöhe) zu erfassen. Diese Kennwerte wurden im Rahmen der Messungen für den Kuka-Roboter ermittelt und sind in Tabelle 5 dargestellt. Angesichts des hohen Einflusses der Antriebsregelungsmethodik des Comau-Roboters wurde auf eine detaillierte Betrachtung der Kennwerte verzichtet, da diesbezüglich kein nennenswerter Unterschied zwischen nativ oder extern gesteuertem System vorhanden ist.

Tabelle 5 – Kennwerte Positionsstabilisierung Kuka-Robotersystem

	Ø Einschwingdauer (s)			**Ø Anzahl Überschwinger (n)**			**Ø Überschwinghöhe (mm)**		
	Kuka	KukaR P616	KukaR P3	Kuka	KukaR P616	KukaR P3	Kuka	KukaR P616	KukaR P3
Pose 5	0,884	0,522	0,931	11,8	6,7	14,4	0,152	0,118	0,212
Pose 4	0,711	0,512	1,452	9,1	5,4	22,4	0,115	0,107	0,302
Pose 3	1,667	0,159	2,330	20,0	1,5	29,3	0,489	0,072	0,662
Pose 2	0,205	0,138	1,973	3,8	2,4	25,5	0,070	0,072	0,247
Pose 1	1,105	1,587	1,827	14,2	17,0	21,2	0,401	1,020	0,468
Ø	**0,914**	**0,584**	**1,703**	**11,8**	**6,6**	**22,6**	**0,245**	**0,278**	**0,378**

Es zeigt sich, dass auch in der quantifizierten Betrachtung der Pose-Zielerreichung das P616-Bewegungsprofil Vorteile im Vergleich zum nativen Kuka-System bietet, sowohl mit einer reduzierten Einschwingdauer als auch einer geringeren Anzahl und Höhe der Überschwinger. Das P3-Profil weist hingegen jeweils schlechtere Werte im externen Betrieb auf.

Einfluss der Zykluszeit der externen Schnittstelle

Neben der Art des Bewegungsprofils standen die einstellbare Zykluszeit der externen Schnittstelle und deren Einfluss auf die Leistungskennwerte im Fokus der Versuche. Während im Kuka-Robotersystem diese mit 4 ms oder 12 ms auswählbar ist, bietet Comau die Optionen 400 µs, 4 ms, 8 ms oder 16 ms als Taktfrequenz an. Die Auswahl der Zykluszeit hat einen entscheidenden Einfluss auf die Anforderungen an das externe System. Während Kommunikationsraten im zweistelligen Millisekundenbereich von Standard-PC-Systemen erreicht werden können, sind im Mikrosekundenbereich Echtzeitsysteme bzw. -erweiterungen erforderlich. Aus technischer Sicht ist insbesondere bei Applikationen mit engen Regelkreisen (wie z. B. bei Force-Torque-Control) eine geringe Taktrate nicht unbedingt von Vorteil, da das externe System die Einhaltung der Zykluszeit auch zuverlässig gewährleisten muss.

Die durchgeführten und in Anhang 9.1.4 beschriebenen Messungen zeigen, dass beim Kuka-Roboter die Wahl der Zykluszeit prinzipiell keinen Einfluss auf die Leistungskennwerte des Robotersystems besaß. Bei der Ansteuerung mit 12 ms wurden jedoch im Singularitätsbereich des Roboters Abbrüche aufgrund Geschwindigkeitsverletzungen festgestellt, welche bei einer 4 ms Ansteuerung nicht auffällig waren. Sowohl Geschwindigkeit, Beschleunigung als auch Genauigkeit und Einschwingverhalten sind unverändert geblieben, unabhängig davon ob die Ansteuerung mit 12 ms oder 4 ms erfolgte. Anders verhielt es sich bei dem extern gesteuerten Comau-Robotersystem. Während mit Zykluszeiten von 400 µs und 4 ms Geschwindigkeit und Beschleunigung identisch blieben, fielen diese bei der Ansteuerung im 8 ms bzw. 16 ms Takt geringer aus und erhöhten die Gesamtdauer der Fahrt. Zudem wies die Roboterbewegung eine unrhythmischere Akustik auf.

Fazit

Zusammenfassend lässt sich festhalten, dass die erreichbaren Wiederholgenauigkeiten mit externalisierten Steuerungskonzepten in der gleichen Größenordnung wie die nativen Systeme liegen. Am Beispiel des Comau-Roboters ist zu sehen, dass antriebsspezifische Regelungsvorgänge einen vielfachen Einfluss auf die Art und Weise der Zielerreichung besitzen können. Aus qualitativer Fertigungsprozesssicht sind die untersuchten Steuerungslösungen dementsprechend ausreichend gut, um ohne Genauigkeitsverlust produzieren zu können. Darüber hinaus zeigte sich im Rahmen der Betrachtung unterschiedlicher Bewegungsprofile und deren Auswirkung auf die Gesamtprozessdauer bzw. Stabilisierungscharakteristik, dass Leistungskennwerte stark beeinflusst und z. T. verschlechtert aber auch verbessert werden können. Die Leistungsfähigkeit bzw. -charakteristik ist also durch die externe Ansteuerung hochflexibel parametrierbar und kann auf Kundenanforderung in Abhängigkeit zum Fertigungsprozess noch besser ausgelegt werden. So könnte bei relativ groben Handhabungsprozessen, Profile mit guten Geschwindigkeits- aber schlechteren Stabilisierungseigenschaften gewählt und Taktzeit gespart werden, während sensiblere Prozesse mit schonenden und ruckfreien Bewegungscharakteristiken durchfahren werden könnten.

Als Defizit ist insbesondere das Fehlen von Absolutgenauigkeit bei der externen Ansteuerung zu nennen. Dies hat zwar keine direkten Auswirkungen auf die Fertigungsqualität, kann aber aus Inbetriebnahme- und Wartungssicht zu erhöhten Aufwänden in Form von Rekalibrierungsvorgängen führen. Deshalb wäre die Fähigkeit, Korrekturen der Absolutgenauigkeit durch das Robotersystem auch bei externer Ansteuerung durchzuführen, eine wünschenswerte Verbesserung aus Anwendersicht. Alternativ besteht die Möglichkeit, eine individuelle Absolutkorrektur im externen Steuerungssystem zu implementieren. Vorteil hierbei ist, dass diese unabhängig vom Robotersystem umgesetzt werden kann und eine bessere Anpassung an die einsatz- bzw. anwenderbezogenen Randbedingungen ermöglicht und zeitgleich eine herstellerübergreifende Lösung darstellt, die vor allem aus Werkssicht den Einsatz unterschiedlicher Manipulatorlieferanten vereinfachen würde.

In Bezug auf die Anwendbarkeit und Anforderungserfüllung für den Karosseriebau am Beispiel der BMW AG zeigen die Messungen, dass bei der Erfüllung der Leistungskriterien durch das native Herstellersystem, diese auch durch das externe System eingehalten werden können. Der BMW-Anforderungskatalog gibt eine Wiederhol-Positionsgenauigkeit von mindestens 1/10 mm vor. Ein Einsatz der Kuka-Robotersysteme wäre dementsprechend sowohl im nativen als auch externen Betrieb möglich, während das Comau-Robotersystem die Spezifikationsvorgabe unabhängig vom Ansteuerungskonzept nicht erfüllt. Dieser Sachverhalt war zu erwarten, da die von Comau angegebene Positionsgenauigkeit für den eingesetzten Roboter ± 0,15 mm beträgt[16]. Das Kriterium der Absolutgenauigkeit wird von beiden Systemen im externen Betrieb nicht erfüllt und würde einen Einsatz gemäß Spezifikationsrichtlinien nicht zu lassen, jedoch wäre wie beschrieben eine Umsetzung dieser Funktion auch im externen System möglich. Die prinzipielle Tauglichkeit der externen Systeme für den Produktionsbetrieb ist somit gegeben, müssten hierzu aber erst befähigt werden.

[16] Hierzu sei angemerkt, dass ein Vergleich der Genauigkeitswerte zwischen den beiden Herstellern nicht zwingend zielführend ist, da der Comau-Roboter aufgrund seiner sogenannten Hollow Wrist-Mechanik eine andere Manipulatorstruktur besitzt. Ebenso bietet Comau weitere Robotersysteme an, die den spezifizierten BMW-Kennwert erfüllen, aber im durchgeführten Laborversuch nicht betrachtet wurden.

5.2.3. Test der Bahnkenngrößen

Ein weiterer zentraler Bewertungsschwerpunkt neben den Kenngrößen für Punktbewegungen ist die Leistungscharakteristik von Bahnbewegungen. Hierfür sollen im Folgenden die Aspekte der Genauigkeit und Wiederholgenauigkeit in Bezug auf Position und Geschwindigkeit einer abzufahrenden linearen Bahn betrachtet werden. Der ISO-Test sieht hierfür eine gerade Bahn über die in Abbildung 25 dargestellten Punkte 2, 1 und 4 vor, die vom Robotersystem mit einer vorgegebenen Lineargeschwindigkeit abgefahren werden soll.

Im Gegensatz zur Testbahn 1 ist bei der Ermittlung der Bahnqualität auch die allgemeine Genauigkeit und nicht nur die Wiederholgenauigkeit von Bedeutung, da diese sich direkt auf die Fertigungsqualität auswirken kann. Beispielsweise erfordert eine gerade Klebenaht auch eine Trajektorie möglichst nahe an der idealen Sollkontur. Die wiederholte Abfahrt dieser Sollbahn mit einer exakt reproduzierten Abweichung von der Idealform würde zwar einen guten Wiederholgenauigkeitskennwert liefern, aber trotzdem negative Auswirkungen auf die Güte der Fügeoperation besitzen. Unter allgemeiner Genauigkeit kann sowohl die Genauigkeit in Bezug zum absoluten Welt- bzw. dem Roboterkoordinatensystem als auch die relative Genauigkeit in Bezug zur idealen geometrischen Sollbahn verstanden werden. Analog zur Testbahn 1 ist der erste Kennwert stark durch das mechanische System bedingt und durch Reteachen der Programmpunkte in der Roboterzelle vor Ort optimierbar, während letztere nicht durch das Korrigieren vorhandener Programmpunkte optimiert werden kann. Am Beispiel einer Linie führt eine Korrektur von Start- und Endpunkt also lediglich zur Verbesserung in Bezug auf das Weltkoordinatensystem. Eine Abweichung von der Sollbahn zwischen diesen Punkten wird nicht korrigiert und hat so direkten Einfluss auf die Fertigungsqualität. Eine Verbesserung wäre zwar auch hier möglich, hätte aber Programmmodifikationen und das Einführen zusätzlicher Stützpunkte auf der Linearbahn zur Folge. Daher werden nachfolgend die Genauigkeits-betrachtung auch in Bezug auf die ideale geometrische Sollbahn und nicht nur die mittlere Istbahn und deren Wiederholgenauigkeit besprochen.

Des Weiteren ist im Gegensatz zu den Pose-Fahrten des ersten Tests die Geschwindigkeitscharakteristik der abgefahrenen Bahn bei Bahnprozessen nicht nur ein taktzeitbezogenes Einflusskriterium, sondern besitzt auch eine qualitätsbeeinflussende Auswirkungen, da der Fertigungsvorgang während der Bewegung (z. B. Kleben) und nicht in der Halteposition erfolgt (z. B. Punktschweißen).

Aus dem gleichen Grund ist neben dem Geschwindigkeitsverlauf auch der genaue Start des Fertigungsprozesses in Bezug zur Roboterposition notwendig. Deshalb wurde ergänzend zu dem vorherigen Test das Leistungskriterium Schaltgenauigkeit betrachtet.

Auf Versuche mit unterschiedlichen Bewegungsprofilen wurde verzichtet, da die Untersuchungen sich auf das Verhalten von Linearfahrten mit gleichbleibender Geschwindigkeit fokussiert haben. Die Beeinflussbarkeit durch Parametrierung beschränkt sich daher auf die Zykluszeit, welche direkt in die Kennwertdiskussion miteinfließt.

Ergebnisse

Bahncharakteristik

Die erfassten Robotertrajektorien für Linearfahrten sind in Anhang 9.2.1 graphisch dargestellt und hinsichtlich ihrer Charakteristik bewertet. Die abzufahrende Linearbahn konnte von beiden Robotersystemen sowohl mit nativen als auch externen Ansteuerungskonzepten durchfahren werden. Aufgrund der klaren geometrischen Vorgabe der Linearbahn ist die zu bewertende Bahn weitestgehend identisch. Größter Unterschied ist erneut ein Versatz in Z-Richtung, verursacht durch die fehlende Absolutgenauigkeit. Ebenso sind die Vorpositionierungs- und Rückfahrtstrecken der nativen Robotersysteme unterschiedlich, da diese wiederum über Pose-Fahrten durchgeführt wurden. Bei den extern gesteuerten Systemen erfolgte im Gegensatz zum nativen Roboterprogramm eine lineare Rückfahrt. Diese Bahnen unterscheiden sich folglich weniger.

Leistungskennwerte

Analog zur Testbahn 1 wurden die Messungen mit einem Testgewicht von 140 kg durchgeführt. Die TCP-Geschwindigkeitsvorgabe wurde ausgehend von 0,1 m/s bis zur maximal durchführbaren Geschwindigkeit festgelegt. Im Anhang ist in eine Gesamtübersicht der erfassten Messungen dargestellt. Im Folgenden wird sich für eine bessere Vergleichbarkeit und Verständlichkeit auf eine Lineargeschwindigkeit von 0,5 m/s fokussiert, welche die maximale Prozessgeschwindigkeit des Großteils aller Klebeapplikationen bei der BMW AG abdeckt.

TCP-Geschwindigkeit

Die TCP-Geschwindigkeit ist für Bahnbewegungen ein zentraler Leistungskennwert. Im Gegensatz zu Punkt-zu-Punkt-Bewegungen kommt es dabei nicht auf eine möglichst hohe Geschwindigkeit und Beschleunigung an, sondern auf einen reproduzierbar gleichbleibenden Geschwindigkeitsverlauf, da der Fertigungsprozess während der Linearbewegung durchgeführt und somit vom Robotersystem stark beeinflusst wird. Für das Kuka-Robotersystem ist der Geschwindigkeitsverlauf über die Messstrecke für die unterschiedlichen Steuerungskonzepte und Parameter in Abbildung 29 dargestellt.

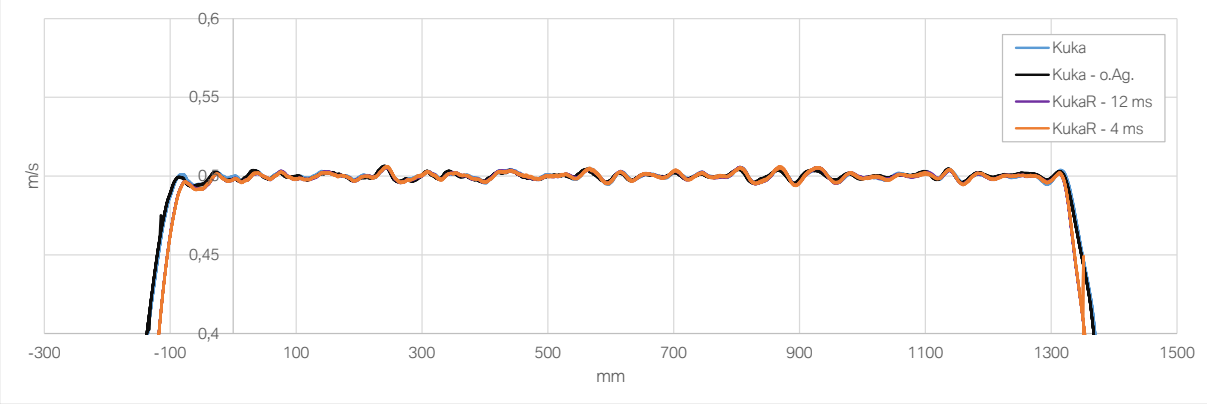


Abbildung 29 – TCP-Geschwindigkeit im Streckenverlauf für das Kuka-System

Es ist ersichtlich, dass die Geschwindigkeitskurven für alle Messungen einen sehr ähnlichen Verlauf aufweisen und die extern gesteuerte Bahn weitestgehend dem des nativen Kuka-Robotersystems entspricht. Dieser optische Eindruck bestätigt sich ebenfalls in den Kennwerten. Die durchschnittliche Geschwindigkeit entspricht in allen Messreihen der Sollvorgabe von 50 cm/s. Ebenso sind die Maximalgeschwindigkeiten identisch (50,6 cm/s) und die Mindestgeschwindigkeit mit 49,5 cm/s (Kuka) zu 49,4 cm/s (KukaR) fast identisch.

Die Leistungskennwerte des extern angesteuerten Comau-Systems mit 0,4 ms und 4 ms sind im direkten Vergleich mit der nativen Comau-Steuerung ähnlich wie die Messungen des Kuka-Systems (Anhang - Abbildung 101), jedoch liegen die Geschwindigkeitsschwankungen mit Grenzwerten von 49,4 cm/s und 50,8 cm/s sowohl für das externe als auch native Comau-System geringfügig höher. Die Durchschnittsgeschwindigkeit entspricht ebenfalls der Sollvorgabe. Die Problematik der 8 ms und 16 ms Interpolationstaktung zeigt sich bei den Linearbahnen gleichermaßen wie bei den Pose-Fahrten und führt zu einer zu geringen mittleren Geschwindigkeit. Für beide Robotersysteme wurde außerdem die Schwerpunktgeschwindigkeitsdifferenz erfasst, um die Reproduzierbarkeit der Geschwindigkeitstreue zu ermitteln und zu vergleichen (Anhang - Abbildung 102). Hier zeigten sich ebenfalls keine Unterschiede zwischen den Ansteuerungskonzepten. Für das Kuka-Robotersystem beträgt die maximale Abweichung 1 mm/s. Für das Comau-System 3 mm/s.

TCP-Genauigkeit

Wie eingangs erwähnt, ist bei Linearfahrten im Gegensatz zur Genauigkeitsbewertung von Punkt-zu-Punkt Bewegungen die reine Wiederholgenauigkeit nicht allein entscheidend für die Qualität des Fertigungsprozesses. Vielmehr ist die Abweichung von der idealgeometrischen Sollform von Bedeutung. Im folgenden Abschnitt wird deshalb insbesondere die Y-Z-Abweichung diskutiert, welche die vertikale und horizontale Abweichung der Linearbahn von einer idealen Linie in X-Richtung beschreibt. Die graphischen Auswertungen hierzu befinden sich für den Kuka-Roboter in Abbildung 30 und für den Comau im Anhang in Abbildung 103 und Abbildung 104. Die tabellarische Gesamtübersicht ist in Tabelle 17 im Anhang aufgeführt.

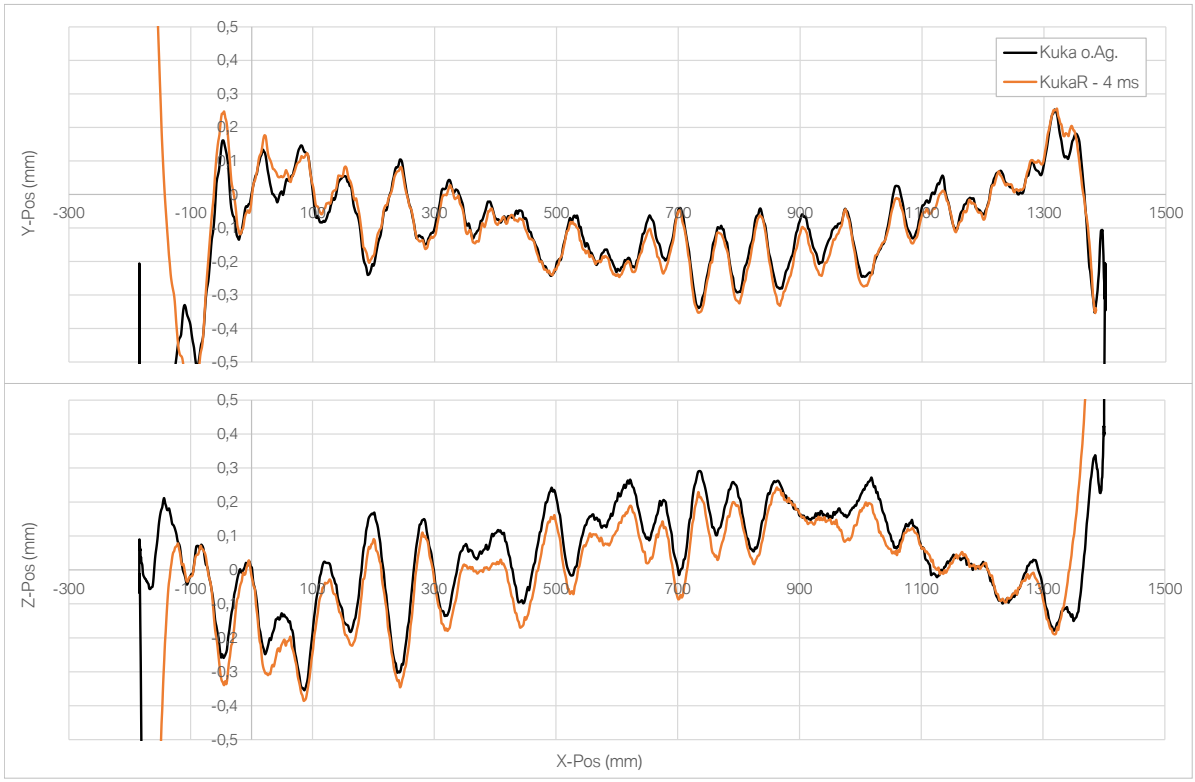


Abbildung 30 – Vergleich der horizontalen (Y) und vertikalen Abweichung (Z) des TCP bei Kuka-Linearfahrten

Im Vergleich der Kuka-Roboter-Bahnen zeigt sich, dass die horizontale Abweichung in Y-Richtung beider Steuerungslösungen geringer ausfällt als in vertikaler Richtung (Z-Achse). In beiden Fällen besitzt der extern gesteuerte Roboter mit 0,354 mm (Y) und 0,390 mm (Z) eine geringfügig höhere maximale Abweichung von der Sollbahn als der native Kuka-Roboter (0,337 mm (Y), 0,354 mm (Z)). Die mittlere Abweichung für die Gesamtstrecke fällt mit 0,127 mm (Y), 0,116 mm (Z) (KukaR) und 0,115 mm (Y), 0,130 mm (Z) (Kuka) nahezu identisch aus. Bei dem Comau-Robotersystem fallen die Differenzen sowohl zwischen den Steuerungslösungen als auch im Vergleich zum Kuka-Roboter stärker aus. Dies zeigt sich besonders bei der Höhenabweichung, die im Maximum 1,490 mm bzw. 1,524 mm und im Mittel 0,896 mm (Comau) bzw. 0,923 mm (ComauR 4ms) beträgt. Bei der vertikalen Betrachtung beträgt die maximale Abweichung für das native 0,567 mm und für das externe System 0,613 mm. Die mittleren Abweichungen betragen jeweils 0,208 mm bzw. 0,292 mm.

Im Gegensatz zur Positionswiederholgenauigkeit wird die Bahnwiederholgenauigkeit gemäß ISO-Norm nicht für die Wiederholgenauigkeit einzelner Punkte der Bahn bewertet, sondern anhand der Maximalwerte aller Punkte. Im Rahmen der Messungen mit einer Bahngeschwindigkeit von 0,5 m/s zeigte sich analog zu den Posegenauigkeitstests, dass mit den externen Systemen vergleichbare Ergebnisse wie mit den nativen Robotersteuerungen zu erzielen sind, wobei der Kuka Roboter mit 0,43 mm (Kuka) bzw. 0,45 mm (KukaR - 4 ms) wiederum etwas besser als das Comau-System mit 0,162 mm (Comau) bzw. 0,167 mm (Comau R) war. Zudem besaß die Zykluszeit bei dem Kuka-System keinen Einfluss auf die Kennwerte, wohingegen bei dem Comau-System die Taktzeiten von 8 und 16 ms zu einer Verschlechterung der Bahngenauigkeit führten. Alle erfassten Bahnen lagen allerdings unter dem Grenzwert der BMW-Bahnwiederholgenauigkeitsanforderung von 0,5 mm.

TCP-Schaltgenauigkeit

Ein weiteres, für den Produktionsbetrieb sehr wichtiges, Leistungskriterium bei bahnsynchronen Prozessen ist die Schaltgenauigkeit. Diese gibt an, mit welcher Genauigkeit ein Robotersystem seine Aktorik während der Manipulatorbewegung ansteuern kann. Relevant ist dies beispielsweise für Start- und Endpunkte von Klebeprozessen, da die geometrische Genauigkeit dieser Punkte in direkter Abhängigkeit zur Schaltgenauigkeit des Robotersystems steht.

Obwohl die Schaltleistung eines Roboters eine hohe Relevanz für den Produktionsbetrieb hat, bietet die ISO 9283 keine Informationen hierzu an. Auch ist in den technischen Spezifikationen der Roboterhersteller in der Regel keine Aussage diesbezüglich zu finden. Aus diesem Grund musste für die Bewertung der bahnsynchronen Schaltgenauigkeit auf Basis der Norm eine Testerweiterung formuliert werden. Als Grundlage hierzu diente die im vorangegangenen Kapitel besprochene Testbahn für Bahngenauigkeit. Zur Erfassung der Schaltgenauigkeit werden zusätzlich Start- und Endschaltpunkte auf der Diagonalfahrt definiert. Diese dienen als Referenzpunkte, zu denen das Robotersystem seine Aktorik, also einen Steuerungsausgang, schalten soll. Ab der vom Messsystem zu erfassenden Aktivierung des Ausgangs beginnt die Erfassung der TCP-Positionen. Die Messung der Bahn endet mit der erneuten Umschaltung des Ausgangs der Robotersteuerung. Auf diese Weise wird die geometrische Werkzeugposition des Roboters in Abhängig zum Schaltvorgang des Robotersystems gebracht und eine objektive Bewertung in Anlehnung an die ISO 9283 ermöglicht. Die Messung erfolgt analog zu den vorherigen Tests mit 30 Wiederholungen. Anhand der erfassten Messpunkte kann analog zu Test 1 und Test 2 die Wiederholgenauigkeit des Schaltpunktes berechnet werden. Der Verlauf des TCP zwischen diesen Punkten ist nicht von Relevanz, da dieser gleichbleibend zur Linearbahn des Bahngenauigkeitstests ist und ausreichend bewertet wurde.

Während im vorangegangenen Abschnitt nicht nur die in der Norm definierte Wiederholgenauigkeit insgesamt, sondern insbesondere die Y- und Z-Werte von Bedeutung waren, stellt bei der Bewertung der Schaltgenauigkeit die X-Position das entscheidende Qualitätskriterium dar, da dieses als einziges von der Schaltgeschwindigkeit und -genauigkeit des Robotersystems direkt beeinflusst wird.

Die Untersuchungsparameter Geschwindigkeit und Schnittstellenzykluszeit sind ebenso Betrachtungsschwerpunkte wie in den vorangegangenen Tests, sie mussten infolge des Einbezugs der Schaltaktorik jedoch um die Taktfrequenz der ProfiNet-Peripherieeinbindung des Robotersystems erweitert werden. Über diese ProfiNet-Schnittstelle ist ein I/O-Modul angeschlossen, welches ein Relais schaltet und so den Messvorgang des Lasertrackers starten und beenden kann. Als Referenz wurde die Bahngeschwindigkeit von 50 cm/s analog zu den vorherigen Tests beibehalten. Der Kuka-Roboter wurde im externen Steuerungsmodus mit einer Schnittstellentaktung von 4 ms und 12 ms betrieben, während der Comau-Roboter implementierungsbedingt ausschließlich mit einem Takt von 400 µs angesteuert wurde, allerdings wurden die Zykluszeiten der Schaltaktorik sowohl beim nativen als auch externen Comau-System als zusätzliche Parameter mit den Werten 2 ms und 16 ms in die Untersuchungen miteinbezogen. Die Genauigkeit der Messung wird im vorliegenden Versuch insbesondere von der Geschwindigkeit (0,5 mm/ms) und der Abtastrate des Trackers (1000 Hz) in Bezug zum Triggersignal bestimmt und liegt folglich bei einer Auflösung von 0,5 mm.

Aufgrund der unabhängig voneinander ausgeführten Regelschleifen von Roboter bzw. externem System und der Schaltaktorik verteilen sich die Messpunkte um einen mittleren Ist-Schaltpunkt. Die Ergebnisse werden zusammengefasst in Abbildung 31 als Boxplot-Diagramm dargestellt.

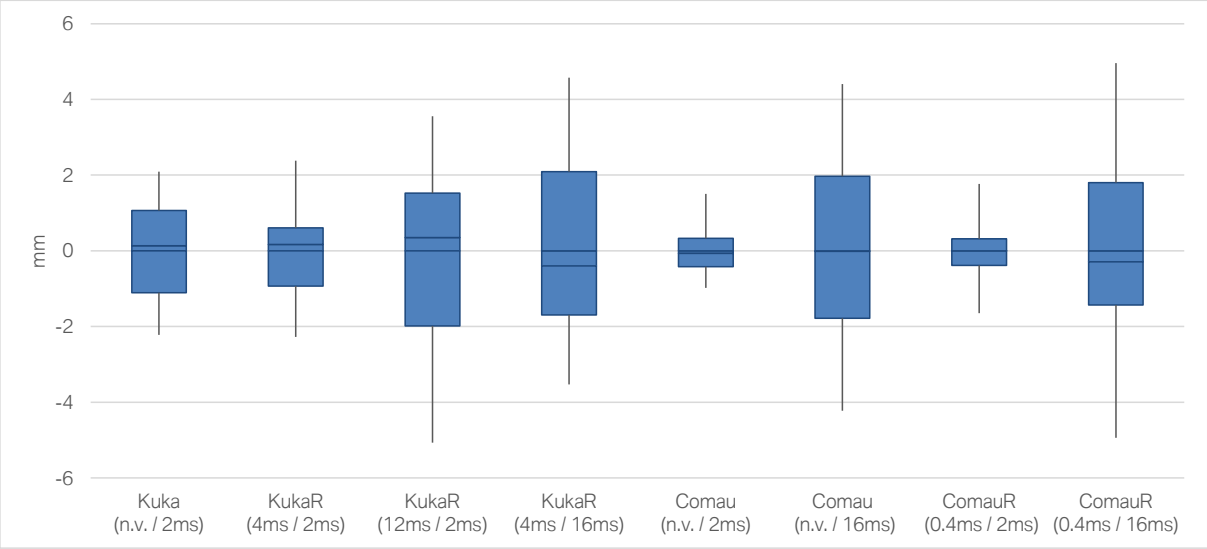


Abbildung 31 – TCP-Schaltgenauigkeit mit Robotersystem und Zykluszeit (ext. Schnittstelle, ProfiNet)

In der Vergleichsdarstellung zeigt sich, dass der Kuka- und der KukaR-Roboter bei einer Taktzeit von 4 ms und einer ProfiNet-Zykluszeit von 2 ms in einem Bereich von ca. ±2 mm die Schaltpunkte auslösen und davon 75 % in einem Bereich von ±1 mm liegen. Für die externe Lösung wurde eine gemessene X-Wiederholgenauigkeit von 1,15 mm bestimmt, für die native Version ein Wert von 1,36 mm. Bei einer Erhöhung der Zykluszeit der Roboter- oder ProfiNet-Schnittstelle im externen Betrieb erhöhen sich die Werte um mehr als das Doppelte. Die X-Wiederholgenauigkeit liegt in einem Bereich von 2,7 mm bzw. 2,8 mm. Hierbei wird die Einflussnahme der unterschiedlichen und nicht synchronisierten Steuerungskreise deutlich.

Ein ähnliches Bild zeigt sich bei hoher Zykluszeit und dem nativ und extern gesteuerten Comau-Roboter. Der Großteil der Triggerpunkte liegt ebenfalls in einem Bereich von ±2 mm, allerdings sind die restlichen Punkte bei den ComauR-Messungen mit bis zu ±5 mm verteilt. Dies zeigt sich auch in der Wiederholgenauigkeit des Schalpunktes, die bei 3,2 mm (Comau) und 3,4 mm (ComauR) liegt. Eine wesentliche Verbesserung bringt die Erhöhung der ProfiNet-Zykluszeit mit sich. Die Genauigkeit verbessert sich dadurch auf 0,74 mm bzw. 0,95 mm. In der Boxplot-Darstellung wird dies deutlich, da 75 % der Punkte in einem Fenster von ±0.35 mm liegen.

Fazit Kennwerte Linearbahn

Zusammenfassend lässt sich festhalten, dass die extern gesteuerten Systeme auch bei den Bahnkenngrößen vergleichbare Leistungen wie die nativen Systeme erreichen. Sowohl bei der Geschwindigkeitskonstanz als auch bei der grundsätzlich erreichbaren Geschwindigkeit sind die erfassten Kennwerte in einer vergleichbaren Größenordnung. Die Bahngenauigkeit muss etwas differenzierter betrachtet werden. Die Wiederholgenauigkeit ist bei den externen Steuerungskonzepten jeweils mit dem nativen Robotersystem gleichwertig. Ebenso die Genauigkeit in Bezug zur idealen Sollbahn des Kuka-Roboters. Der Comau-Roboter hingegen wies bei der externen Ansteuerung einen leicht höheren Versatz in Y-Richtung auf.

Auch hat die Wahl der Zykluszeit der externen Schnittstellen bei dem Kuka-Robotersystem keinen Einfluss auf die Ergebnisqualität der Bahngenauigkeit. Für Zykluszeiten bis 4 ms deckt sich diese ebenso mit dem extern gesteuerten Comau-Roboter. Bei darüber hinaus gehenden Taktzeiten treten dieselben Effekte wie bei Testbahn 1 auf. Die Geschwindigkeit liegt leicht unter Sollvorgabe, zudem zeigen sich Schwankungen in der Bewegungskonstanz. Anders verhält sich die Einflussnahme der Zykluszeit im Aspekt der Schaltgenauigkeit. Eine Veränderung der Taktrate sowohl von Interpolationsschnittstelle als auch Kommunikationstakt beeinflusst die Leistungsfähigkeit des Gesamtsystems erheblich. Es ist also applikationsabhängig zu erwägen, ob mit reduzierter Taktfrequenz gearbeitet werden kann.

Der Gesamtüberblick verdeutlicht, dass die grundsätzlichen Anforderungen für den Einsatz im Karosseriebau erfüllt sind, da die Leistungswerte in vergleichbarer Größenordnung liegen. Die untersuchten Robotersysteme weisen jedoch bereits zum Teil erhebliche Unterschiede auf, die sich in den Kennwerten des Betriebs mit externer Ansteuerung fortführen. Ob die Anforderungen für eine spezifische Applikation erfüllt sind, hängt also primär vom eingesetzten Robotersystem und nicht vom Ansteuerungskonzept ab.

Um die konkrete Fragestellung der Anwendbarkeit für Bahnapplikationen zu beantworten, ist noch der Abgleich mit den Anwendungsanforderungen für Bahnapplikationen erforderlich. Für den Anwendungsfall der BMW AG ist die erforderliche Bahngenauigkeit auf eine Wiederholgenauigkeit von 0,5 mm definiert, die von allen untersuchten Systemen erfüllt wird. Für die allgemeine Genauigkeit in Bezug auf die Sollgeometrie liegen keine Anforderungen vor, allerdings führt ein schlechterer Wert dazu, dass über Stützpunkte der Bahnverlauf korrigierend angepasst werden muss. Die fertigungstechnischen Anforderungen zur Erreichung der notwendigen Produktqualität sind so erfüllbar, aber mit einer längeren Inbetriebnahme verbunden. Während der Durchführung und Analyse der Untersuchungen lag für die Schaltgenauigkeit der Bahnapplikationen bei der BMW AG noch keine allgemeine Spezifikationsanforderung vor. Da die Leistungsfähigkeit der externen Systeme in vergleichbarer Größenordnung zu den Robotersteuerungen der Hersteller liegt, kann geschlussfolgert werden, dass bei einer Tauglichkeit des nativen Systems die externe Steuerungslösung ebenso für den Produktionseinsatz geeignet ist.

Bahnverhalten für runde Bahnen und Überschleifungen

Neben den bisher besprochenen Punkt-zu-Punkt- und Linearfahrten stellt das Bahnverhalten für runde Bahnen ein weiteres Leistungskriterium im ISO-Cube dar. Die ISO 9283 sieht für die Bahngenauigkeitsbeurteilung von runden Bahnen Tests mit Kreis- statt Linearbewegungen vor. Zudem bietet die Norm Beurteilungskriterien für Überschwingen und Verrundungsfehler beim Abfahren von scharfen und runden Ecken. Eine Bewertung der Leistungskenngrößen von Kreis- und Eckfahrten wie in den vorangegangenen Tests sieht der Anforderungskatalog der BMW AG nicht vor. Auch sind Kreisbewegungen selbst keine üblichen Bewegungsarten im Karosseriebauumfeld und demnach nur ein weniger wichtiges Gütekriterium.

Das Verfahren von Ecken ist insbesondere im Zusammenhang mit Überschleifbewegungen eine relevante Bewegungsform im Karosseriebau. Mit Überschleifen werden Bewegungsvorgänge bezeichnet, bei denen das Robotersystem nicht am programmierten Zwischenpunkt hält, sondern bereits vor dem Erreichen eines Zwischenpunktes in die dem Zwischenpunkt nachfolgende Bewegung übergeht. Aus produktionstechnischer Sicht sind diese Bewegungsformen für Fügeprozesse wie Kleben oder Bahnschweißen von Bedeutung, aber auch für die grundsätzliche Taktzeitoptimierung von Bewegungsabläufen, bei der durch das Überschleifen von Bahnpunkten der Stillstand bzw. eine zu hohe Geschwindigkeitsreduktion und der damit einhergehende Zeitverlust vermieden werden soll. Überschleifvorgänge sind dabei nicht auf Eckfahrten begrenzt, sondern für beliebige Bewegungsarten möglich. Eine übergreifende Beurteilung von Überschleifvorgängen auf Basis des heutigen Normstandes ist daher nicht sinnhaft, jedoch bietet der ISO-Cube mit dem Verfahren von Ecken zumindest eine passende Referenzbahn für die Betrachtung des Überschleifvorgangs bei Linearfahrten.

Aufgrund der fehlenden konkreten Anforderungsspezifikation wurden die Untersuchungen hinsichtlich des Bahnverhaltens von runden Bahnen auf die Erfüllbarkeit der Bewegungsmuster und die Beeinflussbarkeit der Bewegungscharakteristik durch die externe Steuerungslogik beschränkt und kein ausführlicher Leistungsvergleich wie bei den Pose- und Bahnkenngrößen durchgeführt. Im Folgenden wird die Ergebnisdiskussion der Kreisbahnen daher auf den Kuka-Roboter beschränkt und die Betrachtung der Überschleiffahrten auf die herstellerübergreifende Gestaltung der Trajektorie fokussiert.

Testbahn 3 – Kreisbahnen

Vorgabe bei der Testbahn für Kreisfahrten ist ein Kreis, der mit vorgegebener, gleichbleibender Geschwindigkeit und einem identischen Radius wiederholt hintereinander abzufahren ist. Die detaillierten Ergebnisse hierzu sind in Anhang 9.2.4 dargestellt. Die Sollbahn konnte sowohl vom nativen System als auch dem externen System mit einer Bahngenauigkeit analog zu den Linearbahnen abgefahren werden, allerdings bot die native Robotersteuerung Vorteile in Bezug auf die Dynamik. Diese konnte bis zu einer maximalen Bahngeschwindigkeit von 1,5 m/s abgefahren werden, während das extern gesteuerte System nur einen Höchstwert von 1,4 m/s erreichen konnte. Jedoch wies dies hingegen eine bessere Treue zum vorgegebenen Radius und der vorgegebenen Geschwindigkeit auf, da diese auch bei Maximalgeschwindigkeit lediglich um 0,2 % abwichen, während das native System Abweichungen bis zu 1,8 % hatte.

Testbahn 4 – Überschleifen von Eckfahrten

Wir bereits einleitend erwähnt, stand bei den Untersuchungen zum Überschleifen von Eckfahrten der Nachweis der grundsätzlichen Realisierbarkeit und Einstellbarkeit im Fokus. Von besonderem Interesse war dabei die Bahncharakteristik einer hersteller- bzw. robotersystemunabhängigen Trajektorie, auf die im Folgenden detaillierter eingegangen werden

soll. Hintergrund ist, dass im Gegensatz zu Linear- oder Kreisbahnen beim Überschleifen von Ecken keine Standard-Sollgeometrie abgefahren wird, sondern sich die Trajektorie anhand der vom Hersteller gewählten Algorithmik bestimmt. Ein unterschiedliches Bahnverhalten war zwar bereits bei den Posefahrten zwischen den Zielpositionen ersichtlich, allerdings ist dies vor allem der unterschiedlichen Mechanik und nicht der Bahnplanungsmethodik geschuldet. Darüber hinaus wird während Punkt-zu-Punkt-Fahrten in der Regel kein direkter Fertigungsprozess durchgeführt. Überschleifbewegungen werden dagegen häufig bei Bahnprozessen und einem aktiven Fügeprozess eingesetzt, um z. B. Ecken mit einer möglichst gleichbleibenden Geschwindigkeit und ohne Stillstand zu überfahren.

Der Überschleifalgorithmus hat somit direkte Auswirkungen auf die Geometrie des Verarbeitungsprozesses. Aus fertigungstechnischen Gesichtspunkten des OEM-Anwenders ist deshalb ein homogenes Überschleifverhalten, unabhängig vom eingesetzten Robotersystem, wünschenswert. Dies reduziert den Aufwand in der Prozessstandardisierung, da die Abstimmung mit der Sensorik und Aktorik besser wiederverwendet werden kann, um gleiche bzw. ähnliche Bahnen zu generieren und so lieferantenübergreifend einen identischen Prozess zu gewährleisten. Ebenso von Bedeutung sind gut einstellbare Bahneigenschaften wie die exakte Festlegbarkeit eines Überschleifpunktes und die Einhaltung von Geschwindigkeitsvorgaben.

Die Prüfbahn für das Überschleifen von Ecken ist gemäß Norm ein in die Diagonale des ISO-Cubes projiziertes Viereck, welches in den Randpunkten mit Verrundungen abgefahren wird und inklusive der Messaufnahmen in Abbildung 32 dargestellt ist. Als Überschleifparameter wurden Werte von 20, 50, 100 und 200 mm gewählt. Der Wert gibt dabei die geometrischen Vorgaben für eine Überschleifbewegung an und kann je nach Hersteller in der Interpretation variieren, zudem haben weitere Randbedingungen wie Werkzeuggewicht oder gewählte Geschwindigkeit zusätzlich Einfluss auf die Überschleifbewegung. Beispielsweise wird beim Kuka-System anhand des Überschleifparametes in Millimetern definiert, mit welchem Abstand zum zu überschleifenden Punkt das Robotersystem die Ausgangsbahn frühestens verlassen darf. Der Comau-Roboter bietet für Linearbewegungen noch weitere Optionen hinsichtlich der Parametrierung an, beispielsweise den Millimeterwert als einzuhaltenden Mindestabstand zum Überschleifpunkt zu interpretieren, statt als Beginn der Überschleifbewegung. Ebenso können weitere Vorgaben wie z. B. mechanische Belastung in die Bewegungsplanung mit einbezogen werden. Für den durchzuführenden Versuch wurde eine Funktionalität analog des Kuka-Systems gewählt. Der Vollständigkeit halber sei erwähnt, dass Kuka neben den herkömmlichen und verwendeten Anweisungen auch Linearbewegungen mit einer größeren Parametrierbarkeit auf Spline-Basis anbietet, die aber im Folgenden nicht betrachtet wurden, da sie gegenwärtig keine verwendete Bewegungsform im Karosseriebau des Fallbeispiels darstellen. Um die Unterschiede von möglichen Überschleifalgorithmen besser aufzuzeigen, wurde in der externen Steuerung eine andere Überschleiflogik verwendet. Im Gegensatz zu den nativen Systemen wird der festgelegte Abstand nicht als Punkt, ab dem die Bahn verlassen werden kann, interpretiert, sondern als Punkt, ab dem eine notwendige Geschwindigkeitsreduktion durchgeführt werden darf. Die Geschwindigkeitsvorgabe innerhalb der Versuche orientierte sich an der maximal ausführbaren Geschwindigkeit der extern gesteuerten Systeme bei einem gewählten Überschleifparameter von 50mm. Dies entsprach 0,9 m/s beim Kuka-Roboter und 0,8 m/s beim Comau-System.

In Abbildung 32 sind die Versuchsfahrten für den Comau und den Kuka-Roboter sowohl mit nativer als auch externer Ansteuerung dargestellt. Die Bahnen zeigen dabei jeweils einen eingestellten Überschleifradius von 200 mm und 100 mm. In der Vergrößerung ist ersichtlich,

dass die unterschiedliche Parametrierung sowohl im nativen als auch im externen Betrieb eine Auswirkung auf den Bahnverlauf hat und dass die nativen Roboterbahnen dabei jeweils vorher mit der Überschleifbewegung starten. Die Fahrten mit den Überschleifradien von 50 mm und 20 mm sind der Übersichtlichkeit halber im Diagramm nicht dargestellt. Beide Größen konnten mit dem nativen System aber problemlos abgefahren werden. Ein Abfahren der Zielbahnen mit den extern gesteuerten Systemen war bei dem kleineren Überschleifradius von 20 mm nicht möglich, da die Richtungsänderungen Drehmomentüberschreitungen ausgelöst haben. Zudem konnten die Bahnen mit den nativen Systemen grundsätzlich jeweils mit einer höher eingestellten Geschwindigkeit als im externen Betrieb abgefahren werden.

Festzuhalten ist, dass Überschleifbewegungen durch das externe System grundsätzlich realisierbar und die Bahnen bzgl. der herstellerübergreifenden Reproduzierbarkeit auf den ersten Blick in vergleichbarer Größenordnung sind, allerdings konnten aus dynamischer Sicht nicht die Leistungswerte der nativen Robotersysteme erreicht werden. Beides soll deshalb nachfolgend noch einmal detaillierter betrachtet werden.

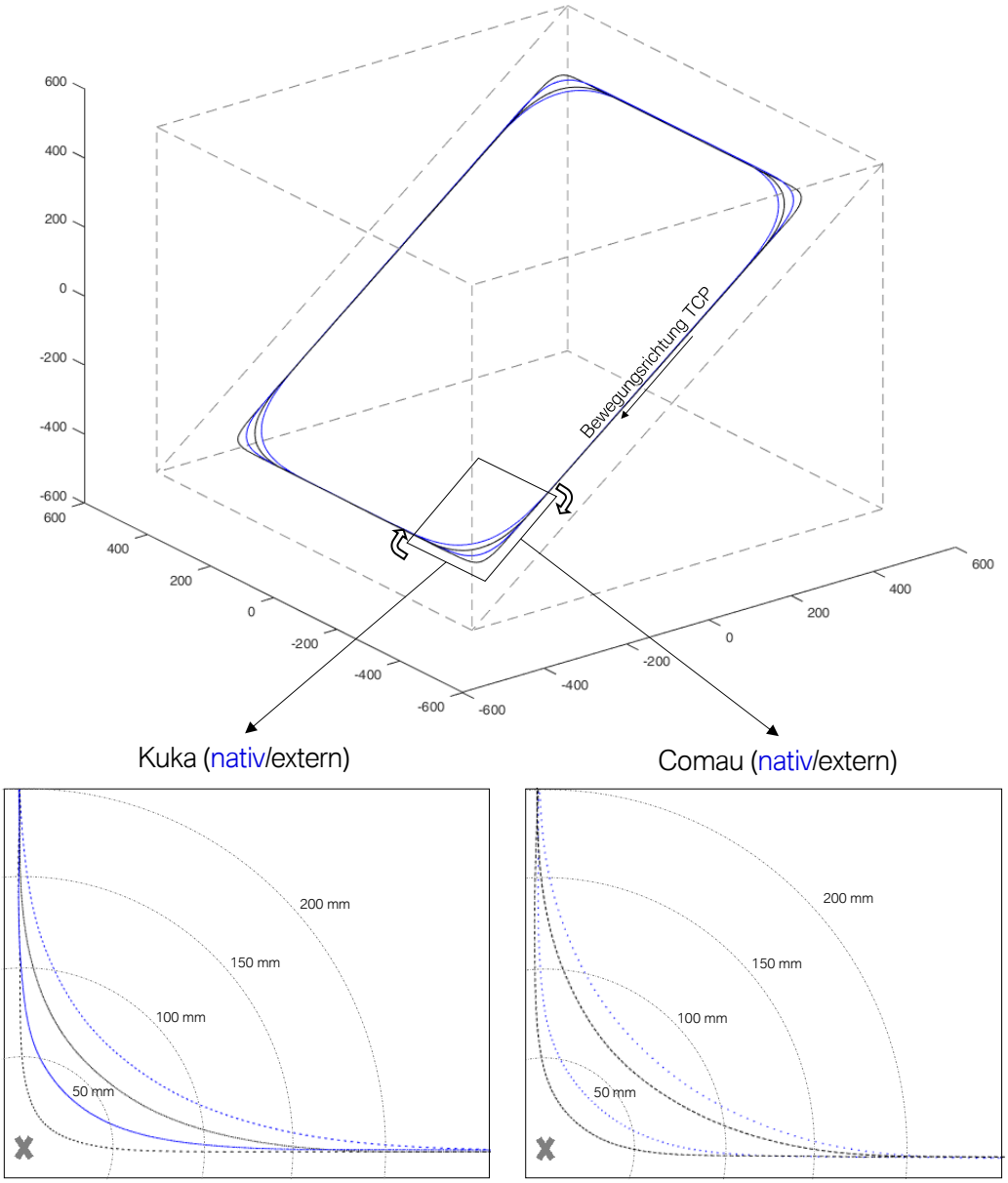


Abbildung 32 – Aufgezeichnete ISO-Cube-Roboterbahnen für Überschleifbewegungen

Die folgenden Grafiken zeigen die Testbahnen des Kuka-Systems mit den Überschleifradien 100 mm und 200 mm in einer 2-dimensionalen Darstellung, bei der die X-Achse die Position des TCPs auf der Ausgangslinearbahn und dessen Abstand zum anzufahrenden und zu überschleifenden Eckpunkt in Millimetern wiedergibt. Der rechte Teil der Grafik zeigt den Abschluss eines Überschleifvorgangs mit dem Übergang zur Linearbahn (1150 mm ⇨ 850 mm) und der linke Teil den Beginn einer Überschleiffahrt (300 mm ⇨ 0 mm). Während Abbildung 33 die Geschwindigkeit des TCPs in der Y-Achse zeigt, stellt Abbildung 34 den Abstand zur Ausgangslinearbahn in einer perspektivischen Draufsicht des Testbahnrechteckes dar.

In der Darstellung des Geschwindigkeitsverlaufs ist ersichtlich, dass sowohl das externe als auch das native Robotersystem eine gleichbleibende Geschwindigkeit von 0,9 m/s einhalten. Die Geschwindigkeitstreue in den überschneidenden Bereichen liegt dabei in ähnlicher Größenordnung, soll aber hier nicht weiter vertieft werden. Es wird auf die Ergebnisse der Untersuchungen zu Linearbahnen verwiesen (Seite 81).

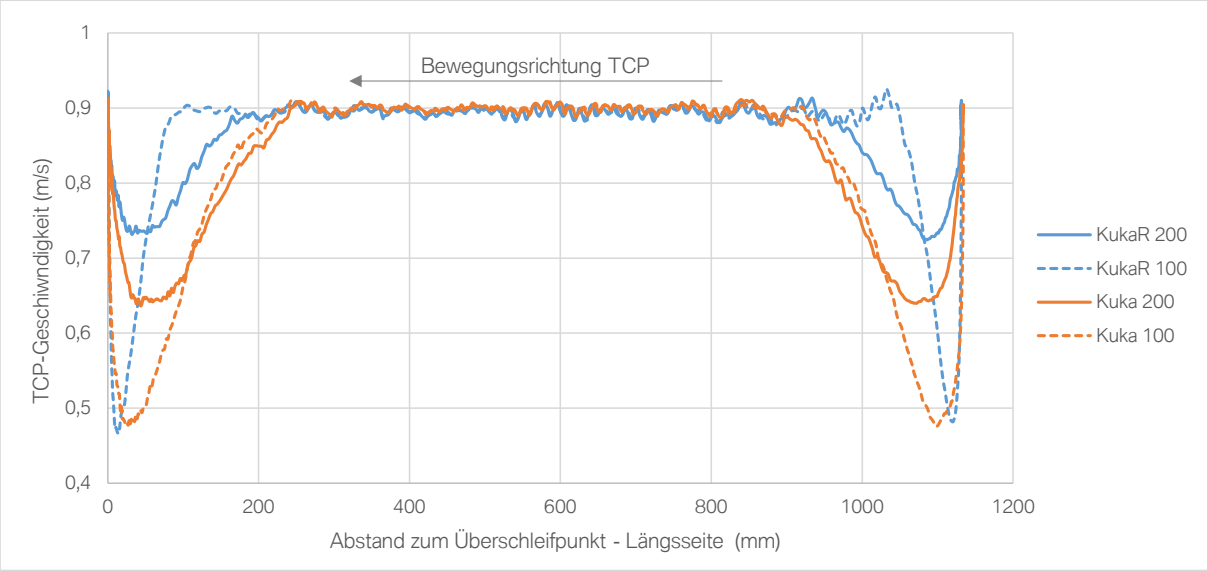


Abbildung 33 – Geschwindigkeitsverlauf beim Anfahren eins Überschleifpunktes

Auffallend ist, dass das native Kuka-System bei beiden Überschleiffahrten ca. 280 mm vor dem Überschleifpunkt beginnt, die Bahngeschwindigkeit zu reduzieren. Bei der extern gesteuerten Bahn hingegen, wird erst ab einem Abstand von 200 bis 100 mm zum Überschleifpunkt die Geschwindigkeit verringert. Dies ist in der unterschiedlichen Gestaltung der Überschleifalgorithmen begründet. Während im originären Roboteralgorithmus lediglich die Positionstreue auf der Ausgangslinie als einzuhaltendes Kriterium zugesichert wird, wurde im Algorithmus der externen Steuerung die Geschwindigkeitskonstanz als weitere Randbedingung priorisiert. Die damit verbundene stärkere positive und negative Beschleunigung führt jedoch auch geringfügig zu höheren Geschwindigkeitsschwankungen bei engeren Überschleifradien. Dies zeigt sich im Übergang zur Linearfahrt (ca. 1050 mm) der KukaR 100-Bahn.

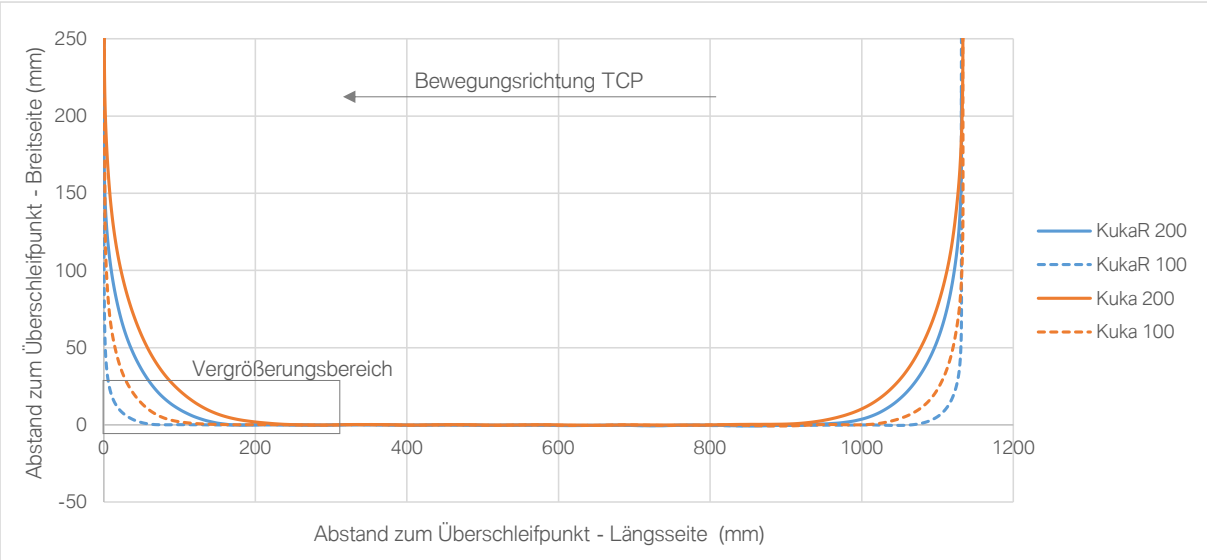


Abbildung 34 – Überschleifbahn (Draufsicht)

Im Positionsdiagramm der Abbildung 34 ist der Parameterunterschied der beiden Kuka-Bahnen noch besser ersichtlich, da das Abweichen von der Linearbahn deutlich unterschiedlich und jeweils in der Größenordnung des eingestellten Parameters (200 mm / 100 mm) erfolgt. Ebenso sind die unterschiedlichen Parametrierungen der extern gesteuerten Bahnen erkennbar, die analog zu den Originalbahnen einen Versatz aufweisen und im direkten Vergleich zu diesen mit der Abweichung später beginnen. Für eine bessere Betrachtung des Sachverhalts ist der Übergang der Fahrten in Abbildung 35 vergrößert dargestellt. Zusätzlich wurde die Grafik um die Bahnen des Comau-Robotersystems ergänzt. Um eine bessere Interpretation der Bahncharakteristik zu gewährleisten, ist die Auflösung der Y-Achse höher als die der X-Achse.

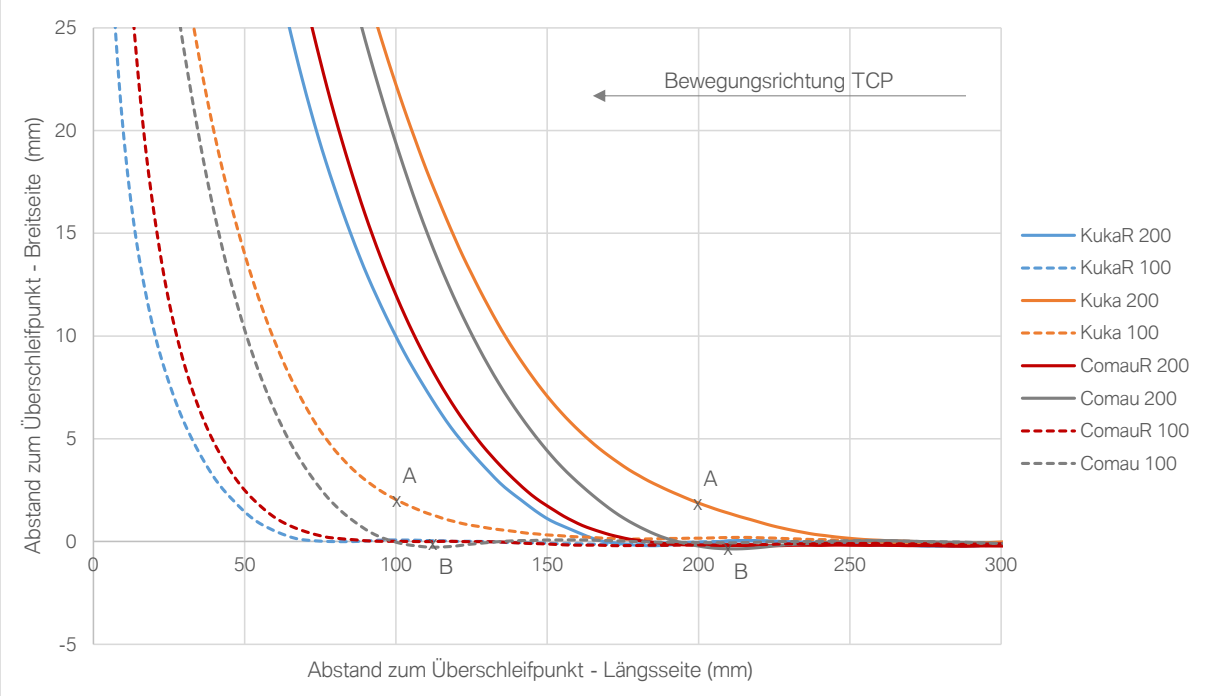


Abbildung 35 – Vergrößerte Darstellung eines Eckpunktes

Die vergrößerte Darstellung verdeutlicht zunächst den bereits erwähnten Unterschied zwischen den originalen und externen Bahnen des Kuka-Robotersystems. Die native Überschleifbahn wird eher eingeleitet als dies bei der KukaR-Bahn der Fall ist. Auf Höhe der im Programm definierten Überschleifbahn (100 mm bzw. 200 mm) beträgt der Abstand bei beiden Kuka-Fahrten jeweils bereits ca. 2 mm zur Ausgangslinearbahn (Markierungen A). Die Bahnen des extern gesteuerten Roboters sind beim gewählten Überschleifpunkt noch direkt auf der Linie. Der Übergang in die Überschleiffahrt beginnt mit der Reduktion der Geschwindigkeit, das Verlassen der Bahn erfolgt erst später. Dieses Verhalten, insbesondere im direkten Vergleich, verdeutlicht sehr gut die unterschiedliche Interpretation einer Überschleiffahrt und die damit verbundene differente Bahncharakteristik des mechanisch identischen Robotersystems.

Noch deutlicher wird dies, wenn man die Testbahnen des Comau-Robotersystems in die Betrachtung einbezieht. Die native Robotertrajektorie verlässt an der eingestellten Überschleifdistanz die Linearbahn, hat aber bei beiden Fahrten kurz zuvor eine geringfügige Abweichung in die entgegengesetzte Richtung (Markierungen B). Der extern gesteuerte Comau-Roboter weist dieses Verhalten nicht auf, sondern entspricht in seiner Bahncharakteristik dem des extern gesteuerten Kuka-Systems. Der geringe horizontale Versatz zwischen den extern gesteuerten Systemen ist in der fehlenden Absolutkalibrierung und der damit einhergehenden, unterschiedlichen Rechtecklänge bzw. der unterschiedlichen Roboterkinematik begründet. In der industriellen Praxis hat ein derartiger Versatz keinen negativen Einfluss, da durch Feinparametrierung in der Roboterzelle, dieser ohne große Schwierigkeiten ausgeglichen werden kann. Entscheidend ist die grundsätzliche Bahncharakteristik, da diese vom Anwender in den herkömmlichen Systemen nur unter den gegebenen Möglichkeiten einer Steuerung und nicht individuell und herstellerübergreifend beeinflusst werden kann. Mit der externalisierten Bahnplanung und Steuerung ist dies jedoch, wie der Test zeigt, sehr gut möglich.

Bezüglich der Überschleifcharakteristik des nativen Kuka-Roboters ist zu erwähnen, dass das beobachte Verhalten nicht exakt dem zu erwartenden entspricht, da gemäß Handbuch der Überschleifbeginn nicht vor dem eingestellten Parameterwert starten soll. Ursachen für die Abweichungen können im Messaufbau, der Programmierung oder im Robotersystem liegen. Da ein derartiges Verhalten nicht bei den Messungen des Comau-System aufgetreten ist und die Programmierwerte überprüft wurden, ist die Ursache mit hoher Wahrscheinlichkeit im Robotersystem verortet, wobei dies sowohl am spezifischen Roboter oder am Roboter bzw. der -steuerungssoftware im Allgemeinen liegen kann. Die in einem anderen Kontext von Wu et al. durchgeführten Untersuchungen eines Kuka-Roboters zeigen ein derartiges Überschleifverhalten nicht [116], allerdings sind Testaufbau, Robotertyp, Programmparameter, Gewicht und der Fokus unterschiedlich zu den hier betrachteten Messungen und eine direkte Vergleichbarkeit ist nicht geben. Zudem kann das Bahnverhalten eines Roboters in Abhängigkeit der SW-Version auch variieren. Von einer tiefergehenden Analyse des Überschleifverhaltens wurde abgesehen, da dieses aus fertigungstechnischer Sicht unkritisch ist bzw. über Parameteranpassungen korrigiert werden kann und keine Relevanz für den Untersuchungsfokus der externen Steuerungslösungen besitzt.

Fazit Messung Kreis- und Überschleifbewegungen

Die Messergebnisse verdeutlichen, dass Kreisbahnen und Überschleifbewegungen mit den extern gesteuerten Systemen grundsätzlich abfahrbar sind und die Genauigkeitskennwerte in Höhe der Messungen von Pose- und Bahnbewegungen liegen. Aus Anforderungssicht der BMW AG waren insbesondere die Parametrierbarkeit und die robotersystemübergreifende Vergleichbarkeit von Überschleifbewegungen von Interesse. Für beide Aspekte konnten mit den

Messungen die Möglichkeiten und Vorteile durch externe Steuerungssystem aufgezeigt werden, da sowohl die Anpassung der Überschleifalgorithmik als auch die verbesserte manipulatorübergreifende Homogenität der Überschleifbahn nachgewiesen wurde.

5.2.4. Zusammenfassung der Messergebnisse

Für die durchgeführten Messungen lässt sich abschließend festhalten, dass die Leistungsfähigkeit der extern gesteuerten Systeme grundsätzlich in vergleichbarer Größenordnung mit den rein nativen Steuerungssystemen liegt. Bietet das Herstellersystem also ausreichend Leistungsfähigkeit, werden die Anforderungen für den BMW-Karosseriebau vom externen Steuerungskonzept ebenso erfüllt. Ausnahme stellt die fehlende Absolutgenauigkeit dar, welche jedoch im externen System implementiert werden könnte. Die zentralen Ergebnisse der Messungen und Anforderungsbewertung sind in Tabelle 6 dargestellt.

Tabelle 6 – Ergebnisse und Anforderungsabgleich der Leistungsmessungen

Bewegungstyp	Ergebnis für externe Ansteuerung	Anforderungsbewertung BMW
Punkt-zu-Punkt	• Zentrale Leistungskenngrößen sind in vergleichbarer Größenordnung zu denen des nativen Herstellersystems - Wiederholgenauigkeit - Beschleunigung und Geschwindigkeitskennwerte - Stabilisierungszeit und Überschwingverhalten • Starke Beeinflussbarkeit durch Bewegungsprofile ➔ Einzelkriterien können im direkten Vergleich schlechter oder besser ausfallen • Interpolationszyklus der Schnittstelle prinzipiell ohne signifikanten Einfluss ➔ für Pose-Fahrten könnte Schnittstellenzykluszeit reduziert werden • Keine Absolutgenauigkeit mehr gegeben	• Wiederholgenauigkeit ausreichend gut (< 0,1 mm), wenn Herstellersystem ausreichend leistungsfähig • Beeinflussbarkeit der Beschleunigungsprofile bietet neue Ansätze der Optimierung von Roboterapplik. • Weitere Optimierung der generischen Beschleunigungsprofile jedoch sinnvoll • Befähigung der Absolutgenauigkeit im externen System möglich • ➔ Tauglichkeit der Systeme gegeben
Linearbahn	• Zentrale Leistungskenngrößen sind in vergleichbarer Größenordnung zu denen des nativen Herstellersystems - Wiederholgenauigkeit - Geschwindigkeitskonstanz, -profil - Erreichbare Maximalgeschwindigkeiten - Schaltgenauigkeit • Zykluszeit hat keinen Einfluss auf Bahnqualität, jedoch starken Einfluss auf die Schaltgenauigkeit	• Bahnwiederholgenauigkeit (< 0,5 mm) gegeben • Ebenso sind Geschwindigkeitseigenschaften und Schaltgenauigkeit ausreichend gut, da vergleichbar zu nativem System • ➔ Tauglichkeit der Systeme gegeben
Kreisbahn und Überschleifverhalten	• Kreisbahnen sind realisier- und gut parametrierbar • Besseres Dynamikverhalten des nativen Systems • Besseres Genauigkeitsverhalten (zur Sollgeometrie) des externen Systems • Überschleifungen realisier- und gut parametrierbar • Homogeneres Überschleifverhalten der Robotersysteme herstellerübergreifend darstellbar	• Keine Genauigkeitsanforderung gegeben, jedoch geometrische Genauigkeit analog zu Test 1 und 2 • Homogenisierung von Überschleifverhalten bietet hohes Potential in der Standardisierung • ➔Tauglichkeit der Systeme gegeben

Sowohl in Bezug auf Punktgenauigkeit als auch Bahngenauigkeit lag die durchschnittliche Abweichung im Rahmen der von den Herstellern spezifizierten Kennwerte. Folglich ist die Einsatztauglichkeit der extern gesteuerten Systeme für den automobilen Karosseriebau gegeben, wenn bereits das Herstellersystem die vorgegebenen Anforderungen erfüllt. Diese können je nach Anwendungsfall variieren und betragen am gewählten Fallbeispiel der BMW AG 0,1 mm in Bezug auf die Positions- und 0,5 mm für die Bahngenauigkeit. Das Kuka-Robotersystem liegt sowohl nativ als auch extern gesteuert innerhalb der vorgegebenen Genauigkeitswerte, während das Comau-Robotersystem die BMW-Anforderungen in beiden Fällen nicht erfüllt.

Im externen Betrieb erhalten beide Systeme die Absolutgenauigkeit nicht und sind deshalb aus formaler Sicht für den Einsatz im Karosseriebau der BMW AG nicht geeignet. Die Notwendigkeit

für Absolutgenauigkeit ist nicht fertigungstechnisch, sondern prozessual bedingt, da die Anforderung hierfür vor allem aus dem Bedarf eines Robotertausches ohne Korrektur der Prozesspunkte resultiert. Aus Sicht der Produktqualität spricht einer Verwendung von extern gesteuerter Systeme demnach nichts entgegen. Die Erhaltung der Absolutgenauigkeit der nativen Robotersysteme bei externer Ansteuerung wäre aus Anwendersicht wünschenswert, um dies nicht im externen System umsetzen zu müssen. Die Implementierung der Absolutgenauigkeit im externen System bietet allerdings die Möglichkeit, das zugrunde liegende Verfahren unabhängig vom eingesetzten Robotersystem direkt an den Anforderungen des Karosseriebaus auszulegen und so Vorteile für eine übergreifende lieferantenunabhängige Inbetriebnahme- und Instandhaltungsstrategie zu realisieren.

In Bezug auf Geschwindigkeit und Beschleunigung konnten die Leistungsdaten der nativen Systeme prinzipiell erreicht werden. Es zeigte sich jedoch, dass die Robotersteuerungen herstellerseitig optimierter sind und daher mit weniger Beschleunigung und einer niedrigeren Geschwindigkeit vergleichbare Prozesszeiten erreichen. Da die verwendeten Algorithmen im externen Steuerungssystem vollkommen generisch und ohne jegliche Optimierung für die verwendeten Robotersysteme ausgelegt wurden, ist dieses Resultat nicht überraschend und es bietet sich noch ausreichend Potenzial, die Leistungsfähigkeit zu erhöhen.

Die Ergebnisse zum Stabilisierungsverhalten für unterschiedliche Bewegungsprofile unterstreichen dies. Der Vorteil der Individualisierbarkeit zeigte sich auch bei den Untersuchungen zu Kurvenbewegungen, da die Bahncharakteristik herstellerübergreifend beeinflusst und das Überschleifverhalten vereinheitlich werden konnte. Dadurch ist beispielsweise eine anwender- bzw. produktbezogene Gestaltung der Trajektorie, unabhängig vom eingesetzten Robotersystem, vorstellbar.

Die Taktung der externen Ansteuerung der Robotersysteme erwies sich als prinzipiell unerheblich für die Leistungskennwerte der Pose- und Bahnbewegungen, da die erfassten Werte in Punkto Genauigkeit, Geschwindigkeit und Beschleunigung sowohl bei dem Kuka-System mit 4 ms sowie 12 ms als auch bei dem Comau-System mit 400 µs und 4 ms identisch geblieben sind. Jedoch zeigte das Comau-System bei Zykluszeiten von 8 ms und 16 ms bereits bei der Durchführung der Versuchsfahrten einen akustisch wahrnehmbaren, unrunden Lauf, der sich zudem auch in Positionsungenauigkeiten und Abweichungen von der Sollgeschwindigkeit bemerkbar gemacht hatte. Obwohl das Kuka-System im direkten Vergleich ohne Abweichungen mit der fast identischen ext. Steuerung und Software bei einer Taktung von 12 ms betrieben wurde, kann die Ursache sowohl im externen als auch nativen System liegen (vgl. Anhang 9.1.4).

Einen systematischen Einfluss auf die Fertigungsprozessqualität hatte die Zykluszeit in Bezug auf das Leistungskriterium der Schaltgenauigkeit, da sich die Verteilung der Triggerpunkte mit steigender Taktzeit der externen Ansteuerung und des ProfiNet-Zyklus der I/O-Ansteuerung vergrößert. Für Bahngeschwindigkeiten von 0,5 m/s erwies sich die externe Systemlösung für beide Robotertypen mit einer Genauigkeit von ca. ±2 mm ähnlich leistungsfähig wie die nativen Steuerungen. Das Leistungskriterium der Schaltgenauigkeit ist weder in den Roboterspezifikationsanforderungen der BMW AG noch in der ISO-Norm formuliert. Eine quantitative Bestätigung der Tauglichkeit anhand eines spezifizierten Wertes kann daher nicht erfolgen. Aufgrund der vergleichbaren Schaltgenauigkeit von nativen und externen Steuerungskonzepten ist davon auszugehen, dass aus Leistungssicht der Einsatz externer Steuerungslösungen für Karosseriebauanwendungen ebenso praktikabel ist.

5.3. Integrationsprozess von externen Steuerungssystemen

5.3.1. Identifizierte Defizite und Verbesserungsvorschläge

Die vorangegangenen Kapitel zeigen, dass die Verwendung von externen Steuerungskonzepten grundsätzlich eine Möglichkeit ist, Steuerungslogik vom proprietären Robotersystem zu entkoppeln und Fertigungsprozesse des Karosseriebaus weiterhin mit ausreichender Genauigkeit und Geschwindigkeit auszuführen. Nichtsdestotrotz konnten während der Analyse und der Implementierung der steuerungstechnischen Konzepte auf Interpolationsebene einige konkrete Defizite identifiziert werden, welche die Umsetzung erschweren. In Tabelle 7 sind die zentralen Mängel kurz beschrieben und Lösungsvorschläge formuliert, die eine Integration von externen Steuerungssystemen aus Anwendersicht erleichtern würden.

Tabelle 7 – Defizite und Verbesserungsvorschläge bei der Integration von Robotersystemen mittels Steuerungskonzepten auf Interpolationsebene

Defizit	Beschreibung	Verbesserungsvorschläge
Absolutgenauigkeit	• Im externen Betriebsmodus nicht vorhanden	• Implementierung im externen System als herstellerunabhängige Lösung • Bereitstellung einer manipulatorspezifischen Korrekturbibliothek durch den Hersteller • Erhalt der Absolutgenauigkeit in der Robotersteuerung auch im externen Betrieb
Verhalten bei Übersteuerung	• Roboter-Not-Halt wird aufgrund von Drehmomentverletzung ausgeführt • Bei Comau-Roboter ist Übersteuerung mit Warnhinweisen am Teach Pendant möglich	• Verhalten bei Übersteuerung auswählbar gestalten • Rückmeldung an das externe Steuerungssystem vor Not-Halt Eingriff • Automatische Reduktion der Geschwindigkeit & Beschleunigung im Robotersystem
Robustheit	• Bei Übersteuerung oder kurzer Kommunikations-verzögerung geht Robotersystem in Not-Halt • Programmneustart notwendig	• Vorstufe zu Not-Halt anbieten und somit Wiederaufnahmefähigkeit verbessern • Schwelle zur Unterbrechung je nach Anwendung parametrierbar gestalten
T1-Betriebsart übersteuerbar	• Keine Geschwindigkeitslimitierung in der T1-Betriebsart* bei externer Ansteuerung * Der T1-Betriebsart stellt den Handbetriebsmodus dar, in dem die TCP-Geschwindigkeit auf <250 mm/s limitiert ist	• Geschwindigkeitslimitierung auch in T1-Betriebsart aktiv • Zwingender Handshake zwischen externem Steuerungssystem und Robotersteuerung in Schnittstelle für Übersteuerungserlaubnis
Aktivierung des externen Steuerungsmodus	• Lediglich über natives Roboterprogramm aufrufbar • Teilweise relativ aufwändige Aktivierungs-Prozedur notwendig (z. B. Neustart des Robotercontrollers)	• Aktivierung über externe Schnittstelle ermöglichen • Beschleunigen der Aktivierungs-Prozedur
Dynamikmodell in externer Steuerung	• Aufwändige Implementierung in externer Steuerung • Notwendige Daten werden von Herstellern meist nicht zur Verfügung gestellt	• Bereitstellen von Dynamikbibliotheken, die Grenzwerte (z. B. mögliche Beschleunigung) für Achpositionen berechnen • Bereitstellen von Trajektorienbibliotheken (Mit RRS erfolgt das auch schon heute in der Simulation) • Behandlung und Rückmeldung bei Übersteuerung
Herstellerspezifische Umsetzung der Schnittstellen und Konzepte	• Zwar grundsätzlich gleiches Konzept, aber Unterschiede in Protokoll, Schnittstellenbelegung Zykluszeiten, Zustandsmaschine	• Vereinheitlichung von zentralen Elementen (z. B. einh. Zustandsmaschine), vergleichbare Protokolle • Bereitstellen der unterschiedlichen Umsetzungen über Konfigurationsdateien

Die festgehaltenen Umsetzungshindernisse wurden zwar primär anhand der Ebene der Interpolationsschnittstelle erarbeitet. Sie können aber ebenso andere der vorgestellten Steuerungskonzepte betreffen. Beispielsweise wirkt sich die herstellerspezifische Schnittstellenbelegung bei Externalisierungskonzepten auf Applikationsebene sogar noch stärker aus, da die Breite der Schnittstelle um ein Vielfaches größer ist. Bei Schnittstellen auf

Antriebsebene kommt der Implementierung und Verwendung des korrekten Dynamikmodells eine noch höhere Bedeutung zu, da die Robotersteuerung des Herstellers komplett entfällt.

Die beschriebenen Defizite können dahingehend unterteilt werden, ob für eine Verbesserung Herstellerunterstützung unabdingbar oder ob die Problemstellung zumindest teilweise ohne den Roboterlieferanten lösbar ist. So sind die Punkte Verhalten bei Übersteuern, Robustheit, T1-Betriebsmodus und Aktivierung des externen Steuerungsmodus alleinig im Robotersystem umsetzbar, während Absolutgenauigkeit, Dynamikmodel und die Schnittstellenanbindungen auch durch die Implementierungen außerhalb der nativen Robotersteuerung realisierbar sind, wobei die damit verbundenen Aufwände durch die Unterstützung des Herstellers allerdings stark reduziert werden könnten.

Eine Reduktion der mit der Umsetzung von externen Steuerungskonzepten einhergehenden Integrationsaufwände ist besonders bei Anwendungen außerhalb der Großserienproduktion von Bedeutung, da eine wirtschaftliche Umsetzung in Unternehmen kleinerer und mittlerer Größe durch diese notwendige Initialleistung erschwert wird. Analog zur Inbetriebnahme von Robotersystemen mit herkömmlichen Steuerungsarchitekturen sollte somit auch die Inbetriebnahme von Robotersystemen mit externen Steuerungskonzepten möglichst einfach gestaltet werden. Ein Vorschlag hierzu wird im Folgenden erläutert.

5.3.2. Plug & Produce-Verfahren

Die Entkoppelung der Programmlogik von der nativen Robotersteuerung des Herstellers hat zur Folge, dass in den Steuerungsverbund des Robotersystems ein neues technisches Element in Form einer zusätzlichen „externen Steuerung“ aufgenommen wird, welches die Anwendungsintelligenz des Fertigungsprozesses beinhaltet und die Steuerungshoheit übernimmt. Die Robotersteuerung wird dadurch zur Peripherie der externen Steuerung. Ebenso wie für die Einbindung von Peripherie in eine Robotersteuerung, ist auch für die Einbindung einer Robotersteuerung als Peripherieelement in eine externe Steuerung ein Inbetriebnahmeprozess notwendig. Dieser ist aus folgenden Gründen sehr komplex:

- Unterschiedliche Schnittstellen sind möglich und legitim
- Schnittstellen variieren je nach Integrationsebene sehr stark in Breite und Inhalt
- Schnittstellenprotokoll & Belegung sind je nach Hersteller unterschiedlich
- Klassische Roboterfunktionen sind im externen System umzusetzen
- Vielzahl an Informationen über Manipulator notwendig
- Diese Informationen sind zum Teil nicht verfügbar / Know-how-Schutz
- Variierende Unterstützung vom Hersteller / Know-how-Schutz
- Manuelle Implementierung und Parametrierung notwendig

Mit den Vorschlägen von Naumann et al. [117], Krug [9] oder Dorofeev et al. [94] existieren Plug & Produce-Verfahren, die eine ähnliche Problemstellung für die steuerungstechnische Inbetriebnahme von Roboterapplikationen und Peripheriekomponenten lösen. Zentrale Gemeinsamkeit der Arbeiten ist, dass Automatisierungstechnikkomponenten durch Selbstbeschreibung ihrer Schnittstellen die Möglichkeit besitzen, Fähigkeiten sowie technischen Eigenheiten zu kommunizieren und auf diese Weise vereinfacht und automatisiert in Steuerungssysteme eingebunden oder angesprochen werden können. Allerdings betrachtet keines der Verfahren den Roboter als eigenständige Peripheriekomponente, die über unterschiedliche Schnittstellenebenen herstellerunabhängig integriert werden kann. Es wird deshalb auch nicht auf Beschreibungsinhalte und Verfahrensdetails eingegangen, welche

notwendig sind, um ein Robotersystem automatisiert und unter Berücksichtigung der identifizierten Ansteuerungskonzepte (Kapitel 5.1) und Defizite (Kapitel 5.3.1) in ein externes Steuerungssystem möglichst aufwandsarm zu integrieren. Aus diesem Grund wurde innerhalb der Umsetzung dieser Arbeit ein Verfahren definiert, das die genannten Mängel und Komplexitätshindernisse aufgreift. Kerngedanke ist, die Selbstbeschreibung des Robotersystems anhand eines mehrstufigen Modells zu gestalten, das anhand von Integrationsebenen die notwendigen Informationen sowohl für Manipulatoraufbau als auch Schnittstellenverbindung beschreibt. Diese kombinierten kinematischen und steuerungstechnischen Beschreibungsdaten werden dem externen Steuerungssystem neben Roboterbasisinformationen über eine initiale Datenverbindung maschinenlesbar zur Verfügung gestellt. Das Robotersystem bzw. der -hersteller kann selbst bestimmen, welche Integrationsebenen unterstützt werden sollen. Je nach Integrationstiefe unterscheiden sich die zur Verfügung zu stellenden Beschreibungsdaten in Menge und Inhalt. Dem Roboterhersteller steht somit frei, welche Art an Informationen und damit aus seiner Sicht u. a. wettbewerbsrelevanten Inhalte preisgegeben werden sollen. Abbildung 36 fasst das „Beschreibungsmodell“ für die „mehrstufige Integration von Robotersystemen“ (Multi-Tier-Robot-Integration-Model) mit den zentralen Informationenbedarfen zusammen. Aufgrund der Erfahrungen im Rahmen der Implementierungsarbeit und der Bewertungsergebnisse in Kapitel 5.1 wurde die hybride Bahnplanungsebene nicht mit aufgenommen und die Interpolationsebene in eine statische und eine dynamische Ausführung unterteilt. So soll den Belangen des Know-how-Schutzes entgegengekommen werden, da auf diese Weise die Interpolationsebene für Anwendungen mit geringeren Leistungsanforderungen unterstützt werden kann, ohne Dynamikinformationen übermitteln zu müssen.

Basisinformationen			Herstellerinformationen (Integrationsebene, Name, Seriennummer, etc.) Anlauf- und Programmstartinformationen Buskonfiguration (Typ, Konfigurationsdateien, IP-Adresse, Speicherbreite) Visualisierung Roboter	
Integrationsebene			**Manipulatorinformationen**	**Steuerungsinformationen**
1	**Applikations-programm**		Keine notwendig	Applikationsbefehle (PTP, LIN, …) Parameter Schnittstellenbelegung (I/O)
2	**A**	**Interpolation statisch**	Anzahl Achsen/Endanschläge DH-Parameter Messdaten f. Absolutgenauigkeit	Zykluszeit/Jitter + Bewegungslatenz Schnittstellenbelegung Einheiten
	B	**Interpolation dynamisch**	Massen Massenträgheit Schwerpunkte	Bahnplanungsmodul Absolutgenauigkeitskorrektur Max. Beschleunigung/Jerk
3	**Antriebsregelung**		Getriebeübersetzung Motordaten/Encoder/Resolver Statische/Dynamische Reibung	Schnittstellenbelegung

Integrationstiefe

Abbildung 36 – Zentrale Inhalte des Multi-Tier-Robot-Integration-Models

Anhand der angebotenen Integrationsebenen und der zu realisierenden Anwendung kann im nachfolgenden Schritt das externe Steuerungssystem die geeignete Anbindung auswählen und einen Konfigurationsprozess durchführen. Dabei werden die erhaltenen kinematischen Roboterinformationen der Beschreibungsdaten verwendet, um ein passendes internes statisches, und falls notwendig, dynamisches Robotermodell automatisch zu erstellen. Ebenso können Bahnplanungsbibliotheken vom Robotersystem übermittelt werden. Dadurch kann die Modellerstellung vereinfacht und die vom Hersteller für den Manipulator optimierte Algorithmik wie z. B. Absolutgenauigkeit direkt verwendet werden.

Die Schnittstellenbeschreibung dient dazu, die korrekte Konfiguration der steuerungstechnischen Verbindung durchzuführen. Zum einen umfasst dies die grundsätzliche Bus- oder Ethernet-Kommunikation zwischen externer Steuerung und Robotersystem, zum anderen die korrekte Signalbelegung und das Kommunikationsprotokoll. Nach erfolgreicher Erstellung des Robotermodells und der Konfiguration der Kommunikationsverbindung kann das externe Steuerungssystem die Steuerungshoheit übernehmen. Als Ergebnis steht ein steuerungstechnischer Gesamtverbund auf Basis einer externen Schnittstellenlösung zur Verfügung, der anhand eines Plug & Produce-Selbstbeschreibungskonzeptes automatisiert durchgeführt werden kann und sowohl die Belange der Roboterhersteller (z. B. Know-how-Schutz) als auch der Anwender (einfache Inbetriebnahme, reduzierter Aufwand in der Modellerstellung) einbezieht. Auf Basis der klar definierten Integrationsebenen und der notwendigen Beschreibungsinformationen wird die Heterogenität der heutigen Schnittstellenausprägung und der Informationsverfügbarkeit durch ein transparentes Zielbild vereinfacht.

Auf eine Implementierung und Evaluierung des Verfahrens wurde innerhalb der Arbeit verzichtet, da sowohl für die automatisierte Konfiguration eines Robotermodells als auch die Selbstbeschreibung und Konfiguration einer Feldbuskommunikation von Automatisierungskomponenten mit den Arbeiten von Wenz [37] und Krug [9] ausreichend tief umgesetzt wurde und die Realisierbarkeit eines kombinierten Ansatzes als gegeben angesehen werden kann. Zudem bestand der Fokus in diesem Teil der Arbeit zunächst in der grundsätzlichen Analyse, Bewertung und Anwendbarkeit von steuerungstechnischen Lösungen zur Reduktion der Herstellerabhängigkeiten und nicht in der Entwicklung eines Plug & Produce-Verfahrens für die vereinfachte Inbetriebnahme dieser Automatisierungskonzepte. Jedoch lässt sich der zukünftige Bedarf daran sowohl mit steigendem Einsatz externer Steuerungslösungen auf Kundenseite als auch dem zunehmenden technologischen Angebot der Roboter- und Steuerungstechnikhersteller ableiten. Aus diesem Grund wurden die essenziellen Bestandteile des beschriebenen Plug & Produce-Verfahrens in Patentschriften festgehalten. Für eine weitere Detaillierung der Verfahrensschritte und Merkmale wird auf folgende Dokumente verwiesen:

EP 3 356 894 A1 [118] Erteilungsabsicht nach Regel 71 (3) EPÜ am 29.01.2021 erhalten.
Verfahren zur automatischen Konfiguration eines externen Steuerungssystems zur Steuerung und/oder Regelung eines Robotersystems

US 10,786,898 B2 [119] Patent erteilt am 29.09.2020.
Method for the Automatic Configuration of an External Control System for the Open-Loop and/or Closed-Loop Control of a Robot System

CN 107710082 B [120] Patent erteilt am 26.01.2021.
用于控制和/或调节机器人系统的外部控制系统的自动配置方法

Anhängige Patentanmeldungen:
DE 10 2015 218 699 A1 [121] veröffentlich am 30.03.2017.
Verfahren zur automatischen Konfiguration eines externen Steuerungssystems zur Steuerung und/oder Regelung eines Robotersystems

DE 10 2015 218 697 A1 [122] veröffentlicht am 30.03.2017.
Verfahren zur automatischen Konfiguration eines externen Steuerungssystems zur Regelung und/oder Steuerung eines Robotersystems

5.4. Zusammenfassung

Im Fokus dieses Kapitels stand die steuerungstechnische Lösung der Herstellerabhängigkeit von Robotersystemen. Ziel war es, zu ermitteln, welche Methoden der externen Ansteuerung grundsätzlich zur Verfügung stehen und welche hiervon für den Karosseriebau am geeignetsten ist. Um die Fragestellung systematisch zu bearbeiten, musste Aufgrund der Vielzahl von bereits vorhandenen unterschiedlichen Verfahren zur Entkopplung von Anwendungslogik zunächst eine geeignete Einteilung identifiziert und durchgeführt werden. Dies erfolgte anhand der Integrationstiefe von Schnittstellen zur externen Ansteuerung und den daraus resultierenden Kategorien: Applikationsprogramm, Bahnplanung, Interpolation und Antriebsregelung. Für diese Ebenen wurden anschließend die zentralen Charakteristika analysiert, Vor- und Nachteile dargestellt und existierende Umsetzungen zugeordnet. Im nächsten Schritt wurden ein Anforderungskatalog aus Anwendersicht entwickelt und die Abstraktionsmethoden hinsichtlich ihrer Eignung für den Einsatz im Karosseriebau bewertet. Hierbei konnten Schnittstellen auf Interpolationsebene als geeignetste Ansteuerungsmethode ermittelt werden, wobei die Themen der Leistungsfähigkeit und Einsetzbarkeit als noch zu beantwortende Fragestellungen für praktische Untersuchungen identifiziert wurden.

Zu diesem Zweck wurde ein Laborexperiment auf Basis der ISO 9283 konzipiert, um einen Leistungsvergleich zwischen den nativen und extern gesteuerten Systemen durchzuführen und mit den zentralen Anforderungen des Karosseriebaus abzugleichen. Neben der Anpassung und Erweiterung der ISO 9283 an spezifische Erfordernisse wie z. B. Schaltgenauigkeit oder zusätzliche Untersuchungsparameter durch das externe Steuerungskonzept, musste zudem eine Software-Werkzeugkette entwickelt werden, die eine automatisierte Verarbeitung der Messdaten und eine quantitative und graphische Auswertung der Ergebnisse ermöglichte. Die Untersuchungen eines Kuka- und eines Comau-Roboters zeigten, dass die Erreichung der nativen Leistungsdaten bei Punkt- und Bahnprozessen ebenso mit extern gesteuerten Systemen erreichbar und der Einsatz im Karosseriebau grundsätzlich möglich ist, wenn das Herstellersystem von sich aus die jeweiligen Anforderungen bereits erfüllt. Darüber hinaus konnte anhand von Überschleiffahrten nachgewiesen werden, dass die externe Ansteuerung Vorteile in der herstellerübergreifenden Auslegung von Bewegungsalgorithmen besitzt und die Roboterapplikationsstandardisierung erleichtern kann.

Auch wenn die prinzipielle Einsatztauglichkeit erfolgreich nachgewiesen werden konnte, zeigte sich im Integrationsprozess eine Reihe von Defiziten, welche den Einsatz von externen Steuerungskonzepten in der Automobilproduktion erschweren. Diese Mängel wurden identifiziert und mit Verbesserungsvorschlägen aufgeführt. Die Problematik des erhöhten Inbetriebnahmeaufwandes infolge der erweiterten Steuerungsarchitektur wurde mit einer passenden Plug & Produce-Methode aufgegriffen und ein Verfahren „zur automatisierten Konfiguration von Robotersystemen mit externer Steuerung“ als Lösungsvorschlag entworfen. In diesem Vorschlag werden die Möglichkeiten der externen Ansteuerung anhand der vorgenommenen Aufteilung der Ansteuerungsebene untergliedert und ein passendes, mehrstufiges Beschreibungsmodell definiert, das zur Selbstbeschreibung von Robotersystemen dient und die automatische Konfiguration der Kommunikationsanbindung und des Robotermodells in der externen Steuerung ermöglicht. Durch die systematische Einteilung der Ansteuerungsebenen und Gewährleistung der Auswählbarkeit von Integrationstiefe und -daten wird ein flexibles Konfigurationskonzept definiert, welches die notwendigen Belange sowohl auf Anwender- als auch auf Anbieterseite berücksichtigt. Die zentralen Inhalte wurden in deutschen und internationalen Patentschriften festgehalten.

6. Herstellerunabhängige Programmierung von Robotersystemen

Während sich das vorangegangene Kapitel den steuerungstechnischen Aspekten der herstellerunabhängigen Anbindung und Integration von Robotersystemen widmete, wird im folgenden Teil das Gebiet der herstellerunabhängigen Roboterprogrammierung von Automobilapplikationen betrachtet.

Obwohl die Programmiersprache von zentraler Bedeutung ist und mit ihr die applikative Intelligenz von Roboteranwendungen formuliert und festgehalten wird, stehen im industriellen Status quo des Karosseriebaus ausschließlich proprietäre Programmiersprachen der Hersteller zur Verfügung. Mit dem Normierungsversuch der Industrial Robot Language (DIN 66312) wurde dieser Problematik bereits Aufmerksamkeit gewidmet, jedoch konnte diese im Gegensatz zu Standardisierungsvorhaben wie G-Code bei NC-Maschinen oder ST in der SPS-Programmierung nicht in die industrielle Praxis überführt werden. Wie aus dem Stand der Wissenschaft und Technik (Kapitel 2.4) hervorgeht, haben eine Reihe von Forschungsarbeiten und -projekten das Thema der robotersystemunabhängigen Programmierung direkt oder implizit mit unterschiedlichen Herangehensweisen bearbeitet, beispielsweise durch anwendungsspezifische Bedienumgebungen [67], PC-basierte Hochsprachen [65] oder innovative Programmiermethoden mit Gestiksteuerung und Augmented Reality-Unterstützung [70, 123].

Bisherige Arbeiten zeigen zwar isolierte Möglichkeiten der Entkopplung von Programmierung zum eingesetzten Robotersystem auf, konnten sich aber nicht als industrieller Standard etablieren. Zudem wurde der Fokus sehr häufig auf KMUs oder spezifische Anwendungen gelegt und der eigentliche Hauptverwender von Robotersystemen – die Automobilindustrie – außer Acht gelassen. Die Anforderungen unterscheiden sich allerdings oft erheblich. So unterliegt die Großserienproduktion sowohl prozess- als auch produktseitig wesentlich starreren und langlebigeren Strukturen. Die häufig als Zielsetzung formulierte, hochflexible Produktion mit minimalen Losgrößen und fließender Umprogrammierung und Umkonfiguration der Anlagen ist nicht, wenn auch zum Teil wünschenswert, direkt auf die Fließbandproduktion der Automobilindustrie übertragbar, da diese wie in Kapitel 4 erläutert, aufgrund der Gesamtkomplexität und Kostenintensität des Produktes und Herstellungsprozesses andere Anforderungen hat.

Im Folgenden soll deshalb die herstellerunabhängige Programmierung von Robotersystemen mit dem Fokus des automobilen Karosseriebaus betrachtet werden. Insbesondere soll analysiert werden, welche Logiken und Funktionen von welchen Anwendergruppen definiert und verwendet werden, welche Anforderungen sich hieraus ergeben und welche Programmiersprachen und -methoden diese bestmöglich abdecken. Zunächst werden grundsätzliche Methoden und Sprachen der Roboterprogrammierung erläutert. Anschließend sollen die gängigen Industrierobotersprachen für die Automobilproduktion hinsichtlich ihrer Funktionalität analysiert und mit der realen Verwendung am Beispiel von Karosseriebauproduktionsstätten der BMW AG verglichen werden. Ziel ist, ein aus Anwendersicht definierten Befehlskatalog für eine domänenspezifische Sprache zur herstellerunabhängigen Programmierung von Karosseriebauanwendungen zu erstellen. Auf dieser Basis sollen unterschiedliche alternative Programmieroptionen bewertet und ein Vorschlag erarbeitet werden, der die notwendigen funktionalen Karosseriebauumfänge abdeckt, roboterherstellerunabhängig ist und den unterschiedlichen Anwenderanforderungen gerecht wird.

6.1. Methoden und Sprachen der Roboterprogrammierung

Für die Programmierung eines Roboters bzw. die Entwicklung von Anwenderprogrammen steht in Industrie und Wissenschaft eine Vielzahl an Werkzeugen und Methoden zur Verfügung. Diese können sich beispielsweise nach Benutzer (Experte oder normaler Anwender), Robotersystemtyp (Industrieroboter oder Serviceroboter) oder Einsatzort (Anlage oder Planungsbüro) in ihrem Konzept und Anwendungsfokus unterscheiden. In der Literatur existieren folglich verschiedene Einordnungskriterien, die Art und Weise des Vorgehens bei der Programmerstellung abgrenzen sollen.

So verwendet Lozano-Pérrez 1982 in einer ersten Studie [124] zu diesem Thema die Einteilung in geführte, roboternahe und aufgabenorientierte Programmierung. Biggs und MacDonald orientieren sich 2003 in ihrer Analyse [125] zu Roboterprogrammiersystemen am Automatisierungsgrad der Programmerstellung, um diese zunächst in manuelle und automatisierte Programmierung einzuteilen und anschließend anhand der Unterkategorien in textuell und graphische Programmierung (manuell) bzw. in lernende Systeme, Programmierung durch Vormachen und Programmierung durch Instruktionen (automatisch) weiter zu untergliedern. Vogl verwendet in seiner Arbeit [126] eine Kombination aus Abstraktionsgrad und Durchführungsort, um Programmiermethoden zu kategorisieren und übersichtlich darzustellen (Abbildung 37). Diese Aufteilung soll im Folgenden verwendet werden, um kurz auf die zentralen Methoden der Roboterprogrammierung einzugehen.

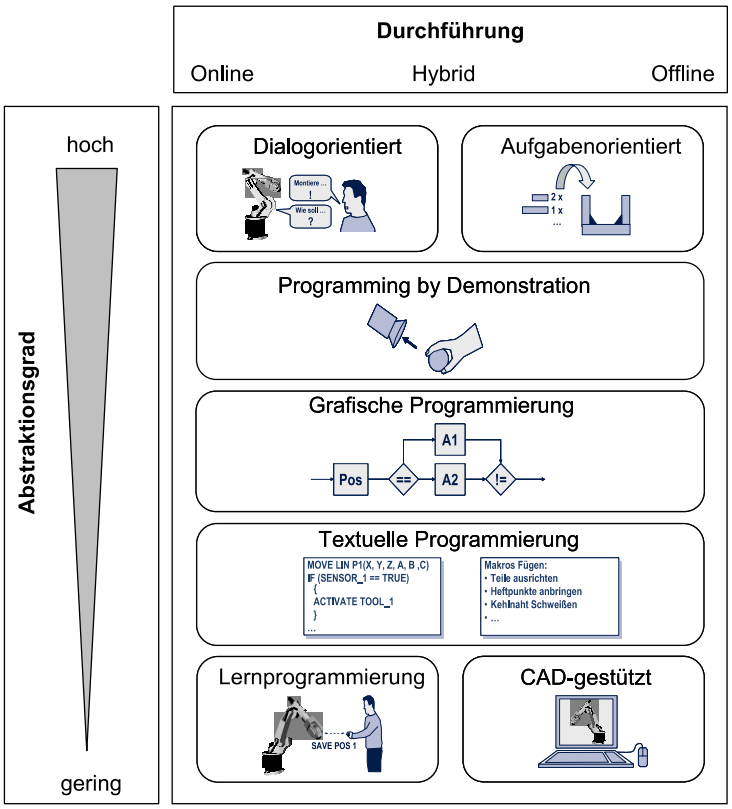


Abbildung 37 – Übersicht der Programmierverfahren für Industrieroboter [126]

Der Abstraktionsgrad beschreibt, wie nah die Programmierung an dem konkreten Fertigungsablauf ausgerichtet ist. Auf unterster Ebene ist die explizite Festlegung der Roboterbahn zu verstehen. Darunter fällt beispielsweise die klassische Teach-In Programmierung, bei der der Anwender am Robotersystem vor Ort die konkreten Bewegungspunkte und Abläufe manuell abfährt und einspeichert. Auf gleicher Ebene sind CAD-

basierte Programmiersysteme einzuordnen. Mit diesen werden Bewegungsabläufe der Roboter in einer Simulationsumgebung evaluiert und definiert. Die Entwicklung von Roboterprogrammen in einer virtuellen Umgebung bietet den Vorteil, dass zunächst keine reale Anlage benötigt wird und eine Vielzahl an Parametern wie Zugänglichkeit, Kollisionsüberprüfung oder Taktzeit untersucht und vor Baustellenbeginn optimiert werden kann. Dies erfolgt in der Regel von einem klassischen PC-Arbeitsplatz aus. Demzufolge spricht man von einem Offline-Programmierverfahren, da im Gegensatz zur Online-Programmierung die Arbeit nicht direkt am Robotersystem erfolgt.

Die textuelle und graphische Programmierung sind Methoden mit mittlerem Abstraktionsgrad, die sowohl vor Ort als auch offline erfolgen können. Im Gegensatz zur reinen Teach-In Programmierung werden hierbei zusätzlich Logikelemente und Peripherieaktionen oder Sensorüberwachungen definiert. Für Biggs und MacDonald [125], Weck und Brecher [127] ist die graphische Programmierung in Form von Flussdiagrammen, Graphen- und Diagrammsichten eine einfachere Alternative zur textuellen Programmierung, da sie leichter zu bedienen ist und keine Syntaxkenntnisse erforderlich sind. Auch bietet sie einen teilweise höheren Automatisierungsgrad, da Programmcode teilweise automatisch generiert werden kann, jedoch geht dies mit eingeschränkter Flexibilität und Funktionalität einher.

Das gleiche Phänomen zeigt sich bei den Programmiermethoden mit hohem Abstraktionsgrad. Bei diesen Verfahren erfolgt die Programmierung der Roboter implizit und weitestgehend losgelöst von systemnahen Programmierbefehlen. Der gewünschte Ablauf soll vielmehr durch die Spezifikation des Produktionsprozesses erfolgen, welcher anschließend systemseitig in Roboteraktionen heruntergebrochen wird [127]. Als zentrales Beispiel ist die aufgabenorientierte Programmierung zu nennen. Die Definition der Aufgabe kann in unterschiedlicher Form erfolgen und sich an Anwendung und Anwender ausrichten. So dienen aufgabenspezifische Module in Simulationsumgebungen dazu, den Simulationsexperten ohne Fachwissen von Verbindungstechnologien bei der Definition von Schweißprozessen zu unterstützen [128]. Der normale Produktionsmitarbeiter, ohne Roboterexpertenwissen, soll hingegen an der Anlage vor Ort in der Bedienung und Programmierung der Robotersysteme durch Spracheingabe oder Gestensteuerung unterstützt werden. Unter Programmierung durch Vormachen sind Verfahren zu verstehen, bei denen das Robotersystem die Aufgabenausführung des Menschen durch geeignete Sensorik wahrnimmt, verarbeitet und anschließend konkret oder abstrahiert ausführen bzw. wiedergeben kann [126]. Dabei lernt das System vom Prozessexperten.

Nachteil der Verfahren mit höherem Abstraktionsgrad ist die Inflexibilität aufgrund des eingeschränkten Einsatzraums. Zwar existiert dieses Forschungsfeld bereits seit längerem und es sind eine Vielzahl von Anwendungen umgesetzt worden, allerdings konnte sich in der Industrie bisher keines davon etablieren bzw. sind nur auf sehr spezifische und kleine Anwendungsfelder begrenzt. Laut Weck und Brecher ist die Bereitstellung eines flexibel einsetzbaren, aufgabenorientierten Programmiersystems noch nicht gelöst [127] und bleibt somit ein zentraler Forschungsgegenstand. Diese Methode bietet zwar einen hohen Abstraktionsgrad und High-Level Funktionalität hinsichtlich der klassischen Roboterbedienung und -programmierung, jedoch ist sie ebenfalls nicht zwingend hersteller- oder systemunabhängig. Außerdem müssen Methoden eines höheren Abstraktionsgrades zunächst auch erst einmal implementiert werden [125]. Die textuelle Programmierung dient somit weiterhin als Basis für eine Vielzahl an alternativen Programmiermethoden. Klassische, textuelle Roboterprogrammiersprachen stellen daher auch künftig ein Kernelement in der Roboterprogrammierung dar und werden im Folgenden noch detaillierter betrachtet.

Roboterprogrammiersprachen

Ähnlich wie bei Programmiermethoden lassen sich Programmiersprachen ebenfalls in unterschiedlichster Art und Weise kategorisieren. Analog zum vorherigen Abschnitt wurde aus diesem Grund mit Abbildung 38 eine Darstellungsform gewählt, die ebenfalls den Abstraktionsgrad als ein Einteilungskriterium besitzt und die Entwicklung der klassischen Programmiersprachenkonzepte im Zeitverlauf zeigt. Ausgehend von hardwarenahen Maschinen und Assemblersprachen haben sich mit fortschreitender Entwicklung Programmiersprachen zunehmend von der eingesetzten Hardware abstrahiert.

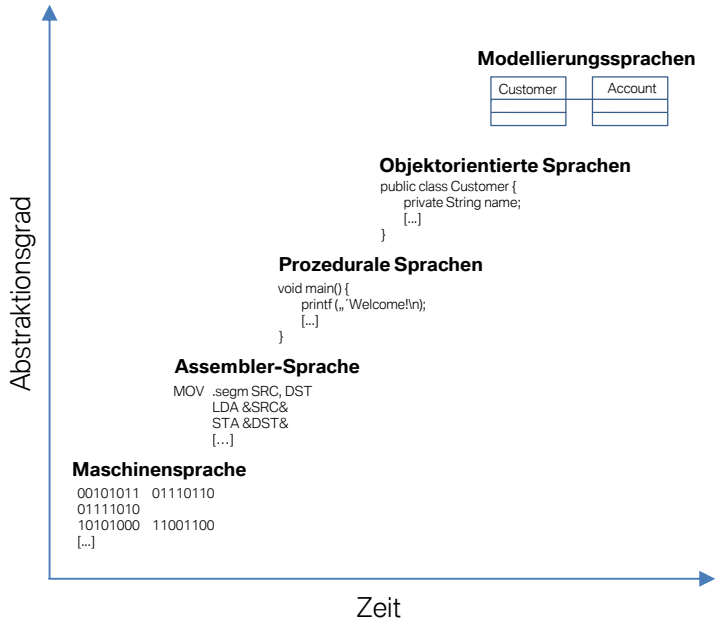


Abbildung 38 – Steigender Abstraktionsgrad mit Evolution der Programmiersprachen in Anlehnung an [129]

In der klassischen Industrierobotik werden fast ausschließlich herstellerspezifische, prozedurale Programmiersprachen verwendet. Beispielsweise stellen Kuka mit KRL und ABB mit RAPID proprietäre Textsprachen im Basic Dialekt zur Verfügung, die zur Definition der Anwenderprogramme auf den jeweiligen Systemen dienen. Zwar ordnen Weck und Brecher [127] diese inzwischen angesichts des im Laufe der Zeit stark gewachsenen Funktionsumfanges auch den Hochsprachen zu, eine Interoperabilität analog zu PC-basierten Hochsprachen ist allerdings nicht gegeben. Außerdem müssen Hersteller ihre eigenen Sprachen kontinuierlich pflegen und funktional erweitern.

Von Vorteil sind die Einfachheit und der Anwendungsbezug. Im Vergleich zu komplexen Hochsprachen können Befehle und Logikanweisung relativ simpel formuliert und ausgeführt werden. Des Weiteren bietet das zugehörige Ökosystem der Hersteller und Drittlieferanten zahlreiche Werkzeuge an, die Roboteranwender in Programmierung und Planung unterstützen und auf Basis der Programmiersprache einen Austausch über Systemgrenzen hinweg ermöglichen. Die Anbieter der Tools (z. B. von Simulationswerkzeugen) müssen jedoch bei der Verwendung mehrerer Roboterhersteller aufgrund der Proprietarität für jeden Dialekt eine eigene Anbindung implementieren.

Auch der relativ junge Hersteller von Servicerobotern Universal Robots bietet, trotz moderner Steuerungsarchitektur und innovativen Bedienkonzeptes, mit URScript [130] lediglich eine proprietäre Scriptsprache zur Anwendungsentwicklung an. Einzig Kuka stellt mit seiner Sunrise-

Plattform die Hochsprache Java als Programmiersprache zur Verfügung, wobei der Einsatz auf den Leichtbauroboter LBR iiwa und die mobile Plattform KMR iiwa beschränkt ist [131]. Eine Umsetzung für Standardindustrieroboter existiert nicht. Zudem ist die API Kuka-spezifisch und bezüglich der roboterbezogenen Befehle nicht standardisiert. Eine standardisierte Industrierobotersprache auf prozeduraler Ebene ist die IRL, welche sich aber nicht durchsetzen konnte.

Der Einsatz von plattformunabhängigen und universell einsetzbaren, allgemeinen Programmiersprachen, sogenannten General Purpose Languages (GPL), stellt im wissenschaftlichen Kontext eine häufig gewählte Umsetzungsmethode dar [125]. Ob C++, Java oder Python – die Breite und Vielfalt an verwendeten Programmiersprachen entspricht auch der zur Verfügung stehenden Menge an Sprachen im PC-Umfeld. Die Wahl der Sprache ist i. d. R. eine Entscheidung im jeweiligen Projekt und welche Anforderungen bzw. Ziele im Vordergrund stehen [125]. Durch die Wahl einer generischen Programmiersprache wird der Abstraktionsgrad in zweierlei Hinsicht erhöht. Zum einen sind generische Programmiersprachen nicht proprietär oder systemgebunden und bieten daher grundsätzlich ein höheres Maß an Portabilität. Zum anderen unterstützen sie die Konzepte der Objektorientierung bereits von sich aus, beispielweise das Verbergen von Low-Level-Funktionalitäten oder die Vererbung von Funktionalitäten über Objekthierarchien.

Ein weiterer Grund für die Verwendung von Hochsprachen in Forschung und Wissenschaft ist, dass häufig PC-basierte Systeme zur Realisierung der gestellten Aufgabe ergänzend oder substitutiv zur klassischen Automatisierungstechnik verwendet werden. Dadurch stehen dem Entwickler eine Vielzahl an Softwarebibliotheken, modernen und intuitiven Ein- bzw. Ausgabegeräten (z. B. Tiefenkameras) oder spezieller Sensorik wie Laserscanner zur Verfügung, die zügig, flexibel und i. d. R. relativ günstig in das eigene Projekt eingebunden werden können. Diese Möglichkeit steht mit dem proprietären Herstellersystem der Robotertechnik üblicherweise nicht zur Verfügung.

Es ist anzumerken, dass die alleinige Verwendung von Hochsprachen mit dem Ziel, projektspezifisch Klassen sowie Methoden zu definieren und so Funktionalitäten abstrahiert zur Verfügung zu stellen, nicht automatisch zu einer systemübergreifenden Portabilität führt, da die entwickelte Software häufig an das spezifische Robotersystem gebunden ist [125]. Middlewaresysteme und Bibliotheken wie ROS, RL oder Softrobot widmen sich daher insbesondere dieser Fragestellung. Allerdings existiert auch hier keine standardisierte Vereinbarung von Schnittstellen oder Methodendefinitionen, vielmehr wird, insbesondere bei ROS, durch die hohe Verbreitung ein de-facto-Standard geschaffen.

Auch wenn objektorientierte Hochsprachen bereits seit langem in der Softwareentwicklung eingesetzt und somit auch fast ebenso lange als Programmiermethode im wissenschaftlichen Umfeld für Roboteranwendungen eingesetzt werden, sind sie in der Industrierobotik nur in vereinzelten Labor- oder Nischenanwendungen zu finden. Sie stellen aber weiterhin einen interessanten Forschungsgegenstand dar – insbesondere in Bezug auf industrielle Anwendungen.

Roboterprogrammierung im Karosseriebau

Die Programmierung von Industrierobotern im automobilen Karosseriebau wird von drei der vorgestellten Methoden dominiert: der textuellen, der CAD-basierten und der Teach-In Programmierung.

Die Entwicklung von Applikationsprogrammen zur Anbindung von Peripheriekomponenten und der Implementierung des fertigungstechnischen Know-hows erfolgt sowohl offline als auch online in Versuchszellen während des Integrationsprozesses eines Roboterlieferanten. Die Programmierung erfolgt in der Programmiersprache des jeweiligen Robotersystems und wird durch Roboter- und Prozessexperten des OEMs und der Lieferanten durchgeführt.

Die Programmierung des konkreten Bewegungsablaufes der Roboter einer Anlage wird während der Konstruktions- und Planungsphase der Fertigungslinie mit einer Simulationsanwendung durchgeführt. Dabei obliegt es dem jeweiligen OEM bzw. dem verantwortlichen Systemintegrator, ob dies mit den proprietären Softwaretools der Roboterhersteller oder mit universellen Simulationsanwendungen von Dritt-Herstellern geschieht. Allerdings löst auch die Verwendung von letzterem die Herstellerabhängigkeit nicht vollständig auf. Zwar können Roboter unterschiedlicher Hersteller mit einem Tool simuliert werden, jedoch müssen für jedes Robotersystem wiederum spezifische Programmelemente bzw. Definitionen aus der Applikationsentwicklung auch in die Simulationssuite überführt werden. Folglich entstehen je Roboterlieferant gesonderte Aufwände und die Abhängigkeit vom Lieferanten der Simulationssoftware wird aufgrund der kundenspezifischen Anpassungen erhöht.

Die beschriebene klassische Lern- bzw. Teach-In-Programmierung wird hauptsächlich während der Inbetriebnahme- und Anlaufphase von Fertigungsanlagen durchgeführt, wobei dank der digitalen Vorarbeit keine Bewegungsabläufe einprogrammiert, sondern lediglich kleine Korrekturen durchgeführt werden müssen, um Differenzen zwischen Simulationsmodell und realer Anlage auszugleichen. Da bereits in der Bauphase über Lasermesssysteme Geometrien und Abstände überprüft und in die Simulation zurückgeführt werden, sind i. d. R. nur sehr geringe Anpassungen von wenigen Millimetern bzw. gar keine Anpassungen notwendig. Nur in seltenen Fällen werden Korrekturen an ganzen Bewegungsabläufen vorgenommen, um Taktzeitverbesserungen oder prozessbedingte Optimierungen durchzuführen, beispielsweise aufgrund eines Bewegungsverlaufes, der mit einem kontinuierlichen Schleifen des Schlauchpaketes am Manipulator einhergeht.

Während der Inbetriebnahmephase müssen neben Bahnkorrekturen zudem die Applikationsbausteine für die jeweilige konkrete Anwendung parametriert werden. Dabei wird nicht in den komplexen Applikationsbausteinen programmiert, sondern mit den zur Verfügung gestellten Funktionsschnittstellen und -parametern gearbeitet. Dies erfolgt online mittels textueller Programmierung und einer vorgefertigten Toolkette.

6.2. Fragestellung und Vorgehen

Es zeigt sich, dass trotz der Vielzahl an Programmiermethoden und -sprachen die Programmentwicklung im automobilen Karosseriebau je nach Anwendungsfeld von den drei Methoden der textuellen Programmierung, der CAD-basierten Offline-Programmierung und der Teach-In-Programmierung dominiert wird, wobei letztere nur für Korrekturen sowie Optimierungen und nicht zur Entwicklung von kompletten Programmabläufen genutzt wird.

Im Vergleich zu vielen Anwendungen in wissenschaftlichen Arbeiten bieten Programmiermethoden mit einem sehr hohen Abstraktionsgrad, wie z. B. Gestik oder dialogorientierte Verfahren, die auf eine häufige und flexible Programmentwicklung und -anpassung fokussiert sind, im Karosseriebau weniger Mehrwert. Zweifelsohne sind derartige Konzepte für individuelle Anwendungen durchaus sinnvoll, jedoch richten sich heutige Produktionskonzepte im Automobilbau an langen Produktionszyklen (ca. siebe Jahre) und konstanten Programmabläufen aus. Eine einmal integrierte Roboteranwendung wird nach der Inbetriebnahme i. d. R. nicht mehr modifiziert. Eine Ausnahme stellt die Integration eines neuen Derivates in die Produktionslinie dar. Allerdings ist die Programmierung der Anlagentechnik gegenwärtig kein Engpass in der Flexibilisierung von Produktionskonzepten. Beispielweise sind die Fertigungsmittel selbst, in Form von bauteilspezifischen Werkzeugen, ein wesentlich größeres Hemmnis für die hochflexible Produktion. Programmierkonzepte, welche die tägliche Anpassung von Roboteranwendungen durch normale Produktionsmitarbeiter ermöglichen sollen, sind deswegen aktuell und mittelfristig nicht erforderlich und stehen nicht im Fokus dieser Arbeit.

Nichtsdestotrotz ist die Reduktion der Komplexität von Programmiersprachen auch für die Automobilproduktion erstrebenswert. Methoden wie aufgabenorientierte oder graphische Programmierung stellen mit Sicherheit interessante Ansätze dar. Jedoch bieten diese Methoden auch nicht per se eine Lösung für die in dieser Arbeit fokussierte Fragestellung der herstellerunabhängigen Programmierung, da system- und implementierungsbedingt weiterhin Unterschiede bestehen können. Außerdem basieren die Konzepte z.T. auf klassischen textuellen Robotersprachen und müssen letztendlich in proprietäre Befehle umgewandelt werden, da diese die Grundlage zur hardwareseitigen Ansteuerung und dem Roboterbetrieb sind. Auch existieren bei aufgabenorientierter Programmierung in Bezug zu Karosseriebauanwendungen mit den Themen der Machbarkeit und Modellbildung präsentere Forschungsschwerpunkte, die noch zu lösen sind.

Klassische textuelle Roboterprogrammiersprachen geben daher nicht nur seit Beginn der industriellen Robotik den Status quo in der Entwicklung wieder, sondern werden auf absehbare Zeit weiterhin eine wichtige Rolle spielen, entweder direkt als Programmierwerkzeug oder implizit als Schnittstellendefinition zu übergeordneten Programmiereinrichtungen wie Simulationsumgebungen. Umso mehr ergibt sich hieraus der Bedarf, zu ergründen, warum sich noch kein Standard in diesem Bereich etablieren konnte und wie sich der Abstraktionsgrad zur eingesetzten Hardware erhöhen lassen kann.

Im folgenden Teil der Arbeit soll anhand des in Abbildung 39 dargestellten Vorgehens ein herstellerunabhängiges und anwendergerechtes Programmiersprachenkonzept für Karosseriebauanwendungen entwickelt werden. Hierzu werden zunächst gängige Programmiersprachen für Industrieroboter betrachtet und hinsichtlich ihrer Konzepte und ihres Funktionsumfanges analysiert und eine Befehlsdatenbank aufgebaut (Kapitel 6.3.1). Zudem wird mit der IRL eine herstellerunabhängige Industrierobotersprache in den Vergleich mit

einbezogen. Das anschließende Kapitel betrachtet die Programmierung von Karosseriebauapplikationen am Beispiel der BMW AG (Kapitel 6.3.2). Ziel ist es, Defizite von klassischen Roboterprogrammiersprachen aufzuzeigen und einen funktionalen Anforderungskatalog aus Domänensicht des Karosseriebaus zu generieren. Hierfür werden zunächst auf Basis von Expertengesprächen und Dokumentationsanalysen die grundsätzlichen Konzepte, Vorgehensweisen und Bedarfe bei der Applikationsentwicklung dargestellt. Anhand der gewonnenen Erkenntnisse wird der Quellcode des Roboterbestands zweier automobiler Vollwerke mit Hilfe der Befehlsdatenbank analysiert und eine transparente Übersicht hinsichtlich der Verwendung von Robotergrundfunktionen und automobilspezifischen Erweiterungen des Karosseriebaus entwickelt. Anschließend werden herstellerunabhängige Alternativen für die Programmierung von Industrierobotern aus den Bereichen Automatisierungstechnik und klassischer Softwareentwicklung vorgestellt und beschrieben (Kapitel 6.4). Um die zur Verfügung stehenden Optionen in ihrer Eignung bewerten zu können, werden die Programmieranforderungen aus Endanwendersicht und den beteiligten Instanzen der Applikationsentwicklung erarbeitet (Kapitel 6.5) und mit den zur Verfügung stehenden Programmiersprachenkonzepten einem Anforderungsabgleich unterzogen (Kapitel 6.6). Abschließend wird auf Basis der gewonnenen Erkenntnisse ein herstellerneutrales und domänenspezifisches, hybrides Programmiersprachenkonzept erarbeitet, welches in Form der Automotive Industrial Robot Language entworfen und an konkreten Befehlsbeispielen verdeutlicht wird (Kapitel 6.7).

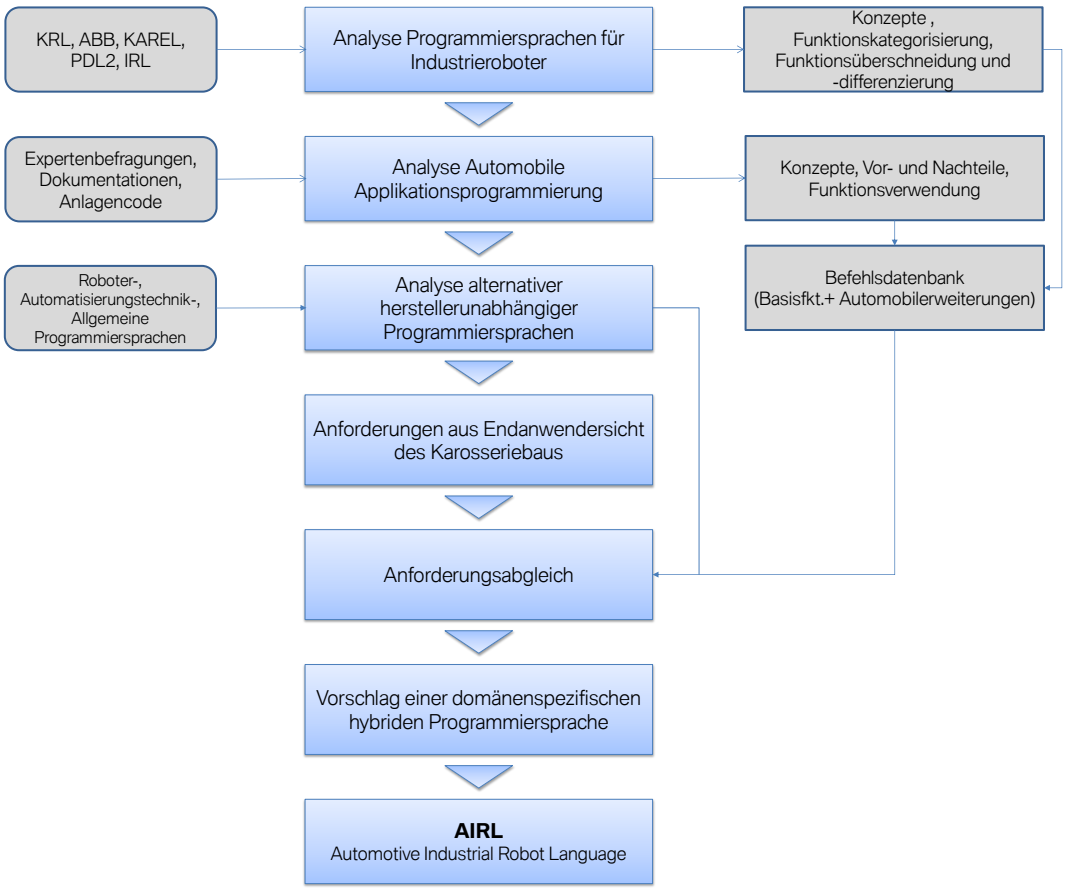


Abbildung 39 – Vorgehen im Kapitel herstellerunabhängige Programmierung von Robotersystemen

6.3. Analyse der textuellen Roboterprogrammierung im Karosseriebau

6.3.1. Programmiersprachen für Industrieroboter

Der Industrierobotermarkt wird trotz zunehmender Marktdynamik nur von einer überschaubaren Anzahl an Herstellern beherrscht. Betrachtet man das Einsatzgebiet des klassischen Karosseriebaus, ist die Auswahl auf die Hersteller ABB, Comau, Fanuc, Kawasaki Kuka, Nachi und Yaskawa eingeschränkt, wobei für den europäischen Automobilbau insbesondere ABB, Comau, Fanuc und Kuka von Bedeutung sind. Aus Tabelle 8 wird ersichtlich, dass sich trotz oder gerade wegen der begrenzten Anzahl an Roboterherstellern keine Sprache in der Programmierung von Industrierobotern als Standard etabliert hat. Sämtliche großen Roboterunternehmen verwenden Programmiersprachen, die sich unter anderem im Syntax, den Befehlsaufrufen und der Funktionsvielfalt unterscheiden. Die herstellerübergreifende Wiederverwendung von Softwarebausteinen und dem impliziten Fertigungsprozesswissen ist folglich nicht möglich.

Tabelle 8 – Übersicht Industrieroboterhersteller, Programmiersprache und OEM-Einsatz im Karosseriebau[17]

Roboterhersteller	Herkunftsland	Programmier-sprache	Traglast Maximum	Karosseriebau-einsatz OEM	Beispiele
ABB	Schweiz	RAPID	800 kg	ja	Audi, BMW, Toyota, Volvo, VW, Ford
ADEPT/OMRON	USA	V+	50 kg	nein	
COMAU	Italien	PDL2	650 kg	ja	FiatChrysler, Ford, Volvo
DENSO	Japan	PAC	20 kg	nein	
EPSON	Japan	SPEL+	20 kg	nein	
FANUC	Japan	TPE/KAREL	2300 kg	ja	AUDI, BMW, VW, Nissan,GM, PSA, Honda
KAWASAKI	Japan	AS	1500 kg	ja	Ford, Suzuki, Toyota
KUKA	Deutschland	KRL	1300 kg	ja	Audi, BMW, Daimler, Tesla,VW, Nissan
MITSUBISHI	Japan	Movemaster	20 kg	nein	
NACHI	Japan	RL	1000 kg	ja	Toyota, SsangYong's
PANASONIC	Japan	G3	220 kg	nein	
STÄUBLI	Schweiz	VAL3	190 kg	nein	
OTC DAIHEN	Japan	k.A.	350 kg	nein	
YAMAHA	Japan	YRL	50 kg	nein	
YASKAWA MOTOMAN	Japan	Inform III, Motoplus	900 kg	ja	Honda

Aus Aufwandsgründen wurde sich für die Analyse des Status quo der textuellen Industrieroboterprogrammiersprachen auf die Auswahl der bei allen deutschen OEMs eingesetzten Hersteller ABB, Fanuc und Kuka beschränkt. Global betrachtet sind diese ebenso die am meisten eingesetzten Roboter im Karosseriebau. Aufgrund der gewonnenen Implementierungserfahrung innerhalb des Prototyps wurde Comau mit in die Analyse aufgenommen. Datenbasis für die Untersuchungen sind die verfügbaren Handbücher der Hersteller, anhand derer die dokumentierten Programmierfunktionen erfasst, kategorisiert und gegenübergestellt wurden. Das Ziel der Analyse unterteilt sich in folgende Punkte:

- Erfassung und Gruppierung der Programmierbefehle
- Festhalten der syntaktischen und semantischen Überschneidungen und Differenzen
- Identifikation der Detail- und Konzeptunterschiede zwischen Herstellern
- Analyse und Darstellung der Funktionsvielfalt bzw. Mächtigkeit der Sprachen
- Erfassung des Sprachumfanges einer DZL für Industrieroboter von Herstellerseite
- Datenbasis für die Entwicklung und den Abgleich von alternativen Programmiersprachen
- Datenbasis für die Analyse der BMW-Roboterprogrammierstandards

[17] Die Auswahl der Hersteller orientiert sich an [132] und [133]. Weitere Informationen anhand Herstellerwebsite/-angaben und eigener Recherche zusammengestellt.

Vorgehen und Datengrundlage

Die Erstellung der Befehlsdatenbank erfolgte anhand des in Abbildung 40 aufgezeigten Vorgehens. Zunächst wurden die relevanten Herstellerdokumentationen (i. d. R. Programmierhandbücher) für die Hersteller ABB [134, 135], Kuka [136], Comau [137, 138] und Fanuc [139–142] ausgewählt und die dokumentierten Programmierbefehle mit Syntax, Parametrik, einer kurzen Beschreibung und einem Beispiel erfasst. Parallel wurde für jeden Programmierbefehl ein universeller Token spezifiziert, der die Semantik des Befehls unabhängig von der proprietären Definition beschreibt. Existierte bereits ein passender Token, beispielsweise infolge einer bereits erfassten funktionsgleichen Instruktion eines anderen Herstellers, wurde der Programmierbefehl diesem zugeordnet. Zudem konnten anhand der Token relevante Kategorien gebildet und Befehle zugeordnet werden. Des Weiteren wurde jeder Befehl dahingegen bewertet, ob er von hoher Bedeutung und Verwendung oder eher weniger relevant für die Programmentwicklung im Karosseriebau ist. Befehle mit hoher Relevanz wurden daher als „wichtig“ markiert.

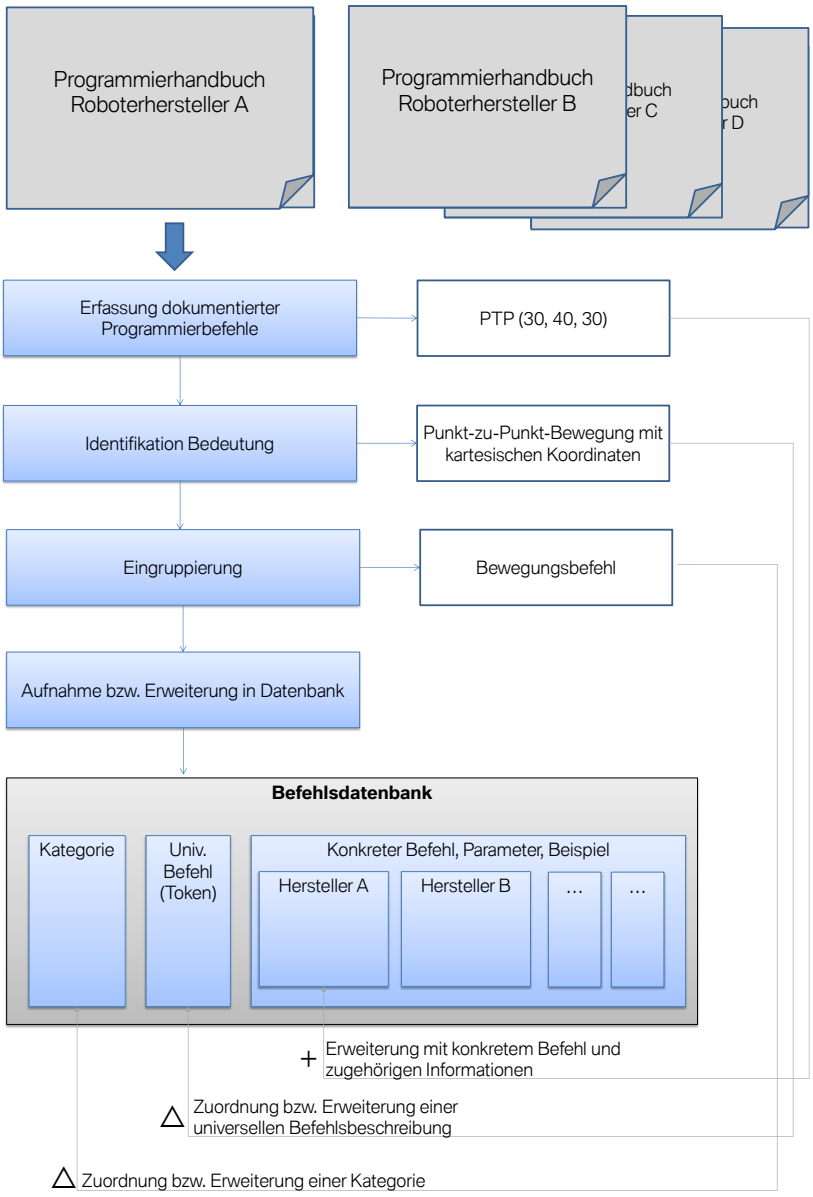


Abbildung 40 – Vorgehen bei der Erstellung der Befehlsdatenbank

Das Vorgehen soll kurz an einem klassischen Bewegungsbefehl in der Kuka-Programmiersprache KRL verdeutlicht werden:

PTP {X, Y, Z, A, B, C }

Dieser Befehl ermöglicht eine Punkt-zu-Punkt-Bewegung mit kartesischen Koordinaten (X, Y, Z) und einer Orientierungsangabe (A, B, C) als Zielposition. Er wird deshalb dem universellen Token „P2P_Kartesisch“ und der Kategorie „Bewegungsbefehle“ zugeordnet. Zudem wird der Syntax mit der Parameterangabe dokumentiert, mit einem einfachen Beispiel und einer kurzen Beschreibung verdeutlicht und aufgrund der hohen Bedeutung für die Programmentwicklung mit „wichtig“ gekennzeichnet. Sobald im späteren Verlauf der Dokumentanalyse der adäquate Bewegungsbefehl eines anderen Roboterherstellers (z. B. „MoveJ“ bei ABB oder „Move Joint To“ bei Comau) aufzunehmen ist, wird dieser ebenso inklusive der Parameter hinterlegt und dem bestehenden Token zugewiesen. Auf diese Weise werden die individuellen Programmierbefehle hinsichtlich der universellen Bedeutung zugeordnet und können herstellerübergreifend verglichen werden. Als Datengrundlagen dienen die jeweiligen Handbücher. Zusätzlich mussten bei unzureichender Dokumentationsqualität Roboterprogrammierer für spezifische Fragen konsultiert bzw. Befehlssyntax und -funktion vereinzelt am realen Roboter oder in der Simulation verifiziert werden. Ziel ist jedoch, den vom Hersteller dokumentierten Befehlsumfang festzuhalten. Für die Datenbasis wurden die Sprachen KRL (Kuka Robot Language) von Kuka, RAPID (Robot Application Programming Interface Dialog) von ABB, PDL2 (Process Description Language 2. Generation) und KAREL/TPE von Fanuc analysiert. Alle dieser Sprachen verfolgen grundsätzlich das Konzept, Roboteranweisungen und die notwendigen Hilfsfunktionen in einem prozeduralen Programmierkonzept zu definieren und weisen aufgrund dessen übergreifende Ähnlichkeiten in Struktur und Anwendung auf. Bei KRL, PDL2 und KAREL wird in den Handbüchern sogar explizit auf die Verwandtschaft zur Programmiersprache Pascal hingewiesen. Die auffälligsten Eigenheiten im Programmierkonzept zeigen Kuka und Fanuc.

Kuka verwendet zur vereinfachten Darstellung und Programmierung ein Verschachtelungskonzept auf Basis von Folds und Inline-Formularen. Mit Folds ist es möglich, Programmteile zu verbergen und den Code übersichtlicher darzustellen. Auf diese Weise können komplexere Programmaufrufe, die mehrere sequentielle Anweisungen benötigen, über eine einzelne Befehlsdefinition angezeigt werden. Am Teach Pendant erfolgt die Bearbeitung dieser Programmbefehle dann über Inline-Formulare. Dies sind textuelle Eingabemasken, die eine einfache Erstellung und Anpassungen von Fold Anweisungen inkl. Parameter ermöglichen und im Hintergrund die korrekte Umsetzung des versteckten Programmteils sicherstellen.

Fanuc bietet mit TP und KAREL[18] als einziger Hersteller zwei Programmiersprachen an, die je nach Anwendungsfokus verwendet werden. TP ist vor allem für die Programmierung von Roboterbewegungen und Anweisungsbefehlen gedacht. TP-Programme können wie bei den anderen Herstellern am Programmierhandgerät (Tech Pendant) angezeigt, bearbeitet und ausgeführt werden. KAREL Programme können am Robotersystem lediglich ausgeführt, aber nicht betrachtet werden. Sie müssen vorab am PC erstellt und kompiliert werden und dienen laut Hersteller insbesondere der Einrichtung des Robotersystems und nicht der Bewegungsprogrammierung. Für die Befehlsanalyse wurde TP und KAREL zusammenfassend betrachtet und mit den anderen Programmiersprachen verglichen.

[18] Die Bezeichnung KAREL stellt keine technische Abkürzung dar, sondern ist eine Anlehnung an den tschechischen Schriftsteller Karel Čapek (Seite 6).

Ergebnisse

Auf Basis der Handbücher der vier Roboterhersteller wurden insgesamt 1670 Instruktionen festgehalten und acht Befehlskategorien mit jeweils drei Unterkategorien identifiziert.

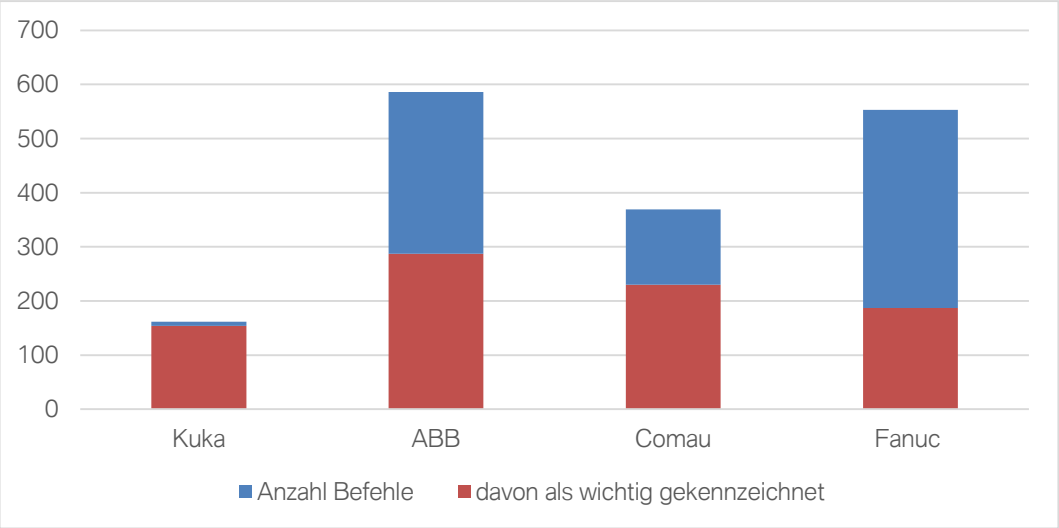


Abbildung 41 – Erfasste Programmierinstruktionen pro Hersteller

Befehlsmenge

Zunächst fällt auf, dass die Dokumentationen der Hersteller sowohl in Umfang als auch Detailliertheit stark variieren. Während Kuka im Standardhandbuch lediglich ca. 160 dokumentierte Befehle enthält, beschreibt ABB fast 600 Instruktionen (Abbildung 41). Die Differenzen zwischen den einzelnen Herstellern sind unter anderem auf konzeptionelle oder funktionale Unterschiede der einzelnen Robotersprachen zurückzuführen, beruhen jedoch auch auf der Tatsache, dass Programmierbefehle zwar vorhanden sind, aber nicht dokumentiert wurden. Beispielsweise sind im KRC4-Handbuch der Firma Kuka keine trigonometrischen Funktionen aufgeführt, die für die Vorgängerversion KRC2 noch beschrieben wurden und in aktuellen KRC4-Steuerungen ebenfalls verwendet werden können.

Kategorisierung

Abbildung 42 stellt die erfassten Befehlskategorien dar, welche im Folgenden kurz erläutert werden.

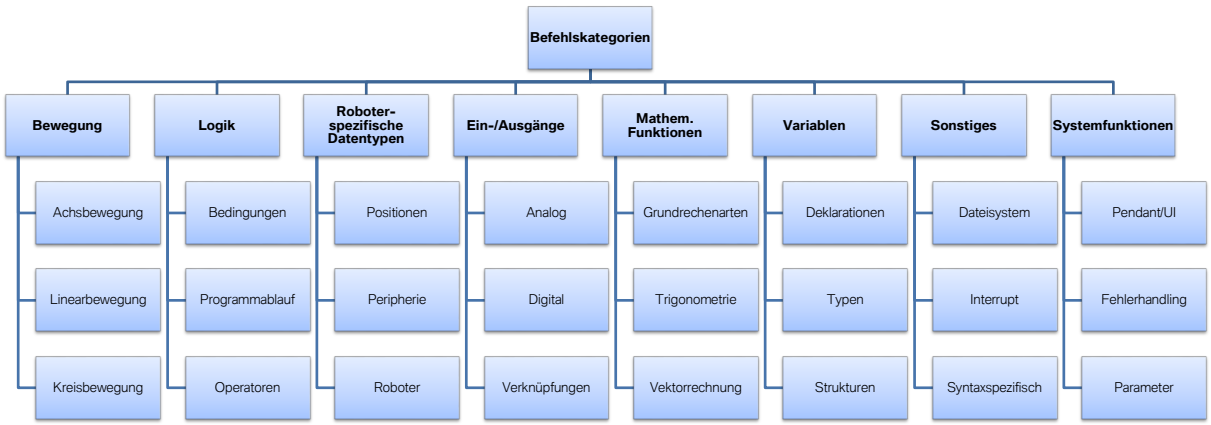


Abbildung 42 – Gewählte Kategorisierung der Roboterbefehle mit Gruppierungsbeispielen

Bewegung
Funktionen, die zu einer Bewegungsaktion des Manipulators führen. Zentrale Instruktionen stellen Achs- oder Bahnbewegungsbefehle dar.

Logik
Logikelemente zur Programmsteuerung. Dazu gehören Instruktionen zur Überprüfung von Bedingungen oder Fallunterscheidungen sowie Schleifen- und Sprungbefehle. Dabei handelt es sich meist um klassische Programmierbefehle, die auch in diversen Computerprogrammiersprachen enthalten sind.

Roboterspezifische Datentypen
Befehle zur Bestimmung und Bearbeitung von roboterspezifischen Datentypen wie Positionen, Lasten, Peripheriebausteine oder Koordinatensysteme.

Ein und Ausgänge
Anweisungen zum Setzen, Lesen sowie Verarbeiten von analogen und digitalen Ein- und Ausgängen zur Ansteuerungen von Peripherie über das Feldbussystem oder direkt verdrahtete Komponenten.

Mathematische Funktionen
Klassische mathematische Operationen für Grundrechenarten und Trigonometrie, die in den meisten Standardprogrammiersprachen ebenfalls definiert sind. Roboterprogrammiersprachen enthalten zudem erweiterte Instruktionen, die zur Verarbeitung und Umrechnung von Positionen und Rotationen verwendet werden. Beispielsweise Euler-Winkel und Quaternionen-Umrechnung.

Variablen
Befehle zur Deklaration, Typbestimmung und Verarbeitung von Standardvariablen wie Strings, Strukturen oder numerischen Werten.

Sonstiges
Instruktionen, die den vorherigen Kategorien nicht zugeordnet worden sind, beispielsweise herstellerspezifische Syntaxelemente, Interrupt-Handling oder Dateioperationen.

Systemfunktionen
Befehle mit starkem Systembezug, beispielsweise die Programmierung von spezifischen GUI Elementen, Ausgabe von Fehlermeldungen oder Zeitparameter.

Aufgrund des hohen unterschiedlichen Dokumentationsgrades und Funktionsweise der Systemfunktionen wurde von einer weiteren Betrachtung dieser Kategorie im Folgenden abgesehen.

Befehlsvergleich auf Kategorienebene

Um ein besseres Bild bezüglich der Abdeckung und Unterschiede zwischen den Roboterprogrammiersprachen zu erhalten, stellt Abbildung 43 die Anzahl der erfassten Befehle anhand der Kategorisierung dar.

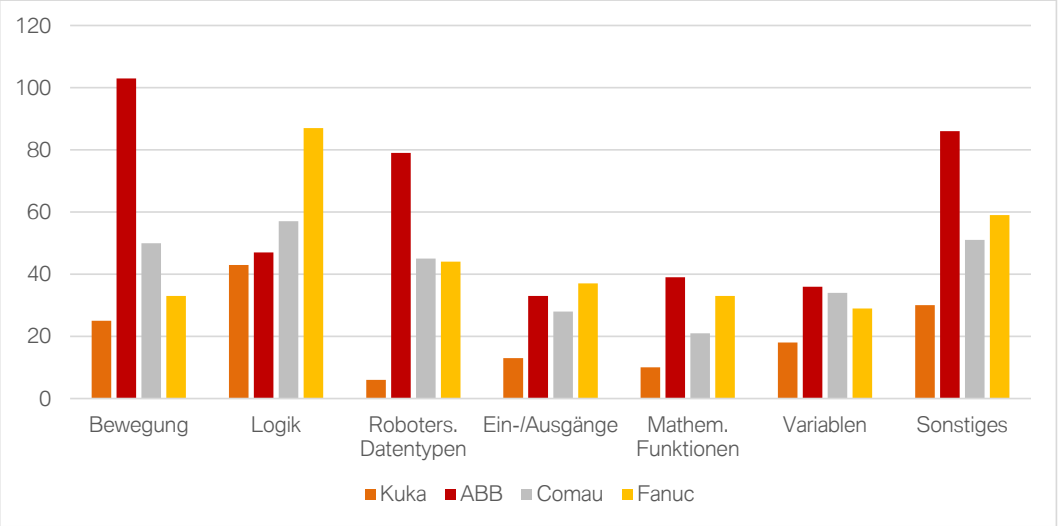


Abbildung 43 – Anzahl erfasster Programmierbefehle

Zunächst zeigt sich wie in der vorhergehenden Grafik, dass Kuka den geringsten Befehlsumfang pro Kategorie hat. Dies liegt zum einen an dem, insbesondere im Vergleich zu ABB, geringen Dokumentationsgrad und zum anderen an Konzeptunterschieden. So gibt ABB z. B. relativ viele Datentypen vor, während Kuka diese Spezifikation gröber hält und dem Nutzer eine weitere Detaillierung überlässt.

Ebenso zeichnet sich die ABB-Spezifikation durch hohe Vielfalt in der Kategorie der Bewegungsbefehle aus, da Triggerlogiken, Bahnaufzeichnungen und Synchronisationsbefehle sehr ausgeprägt dokumentiert sind und bei anderen Herstellern in dieser Form nicht vorkommen. Die große Anzahl an sonstigen Anweisungen ergibt sich durch Dateisystem- und Interrupt-Befehle. Dass FANUC ebenfalls vergleichsweise viele Befehle aufweist, liegt vor allem an dem Zwei-Sprachenkonzept, welches insbesondere im Bereich der Programmlogik zu einer Verdopplung der Befehlsmenge führt, da diese sowohl in KAREL als auch TP vorgehalten werden und in der ersten Erfassung z. T. doppelt aufgenommen wurden.

Um diese Eigenheit und die für den Automobilbereich weniger relevante Funktionen aus der Übersicht zu filtern, stellt Abbildung 44 den Kategorienvergleich lediglich für die als wichtig gekennzeichneten Befehle dar. Es wird ersichtlich, dass im Bereich Bedingungen, Logik und Variablen die Roboterhersteller jeweils eine ähnliche Anzahl an Befehlen definiert haben. Die eben beschriebenen Charakteristika im Bereich Bewegung, Datentypen und Sonstiges bleiben weiterhin bestehen. Die hohe Anzahl an ABB-Anweisungen in der Kategorie Ein- und Ausgangsoperationen wird durch die Spracheigenschaft verursacht, Gruppenoperationen durchzuführen und komplexere Wartelogiken auf Basis von I/O-Zuständen definieren zu können. Mit Gruppenports ist es möglich, nicht nur einzelne Bits für Ein- und Ausgänge auszuwerten und zu schalten, sondern auch komplexere Datentypen wie Bytes oder binär codierte Wörter zu versenden. Während Gruppenoperationen auch bei Fanuc und Comau direkt im Anwenderprogramm ausführbar sind, ist bei Kuka eine vorherige Definition in der I/O-Konfiguration nötig. Wartelogiken über spezifische Befehle bietet nur ABB an, bei den restlichen

Herstellern muss dies durch eigene Implementierung erfolgen. Die geringe Anzahl an mathematischen Operationen am Kuka-System ist vor allem durch die erwähnte mangelnde Dokumentation der trigonometrischen Funktionen bedingt. Die Funktionsvielfalt selbst ist gegeben.

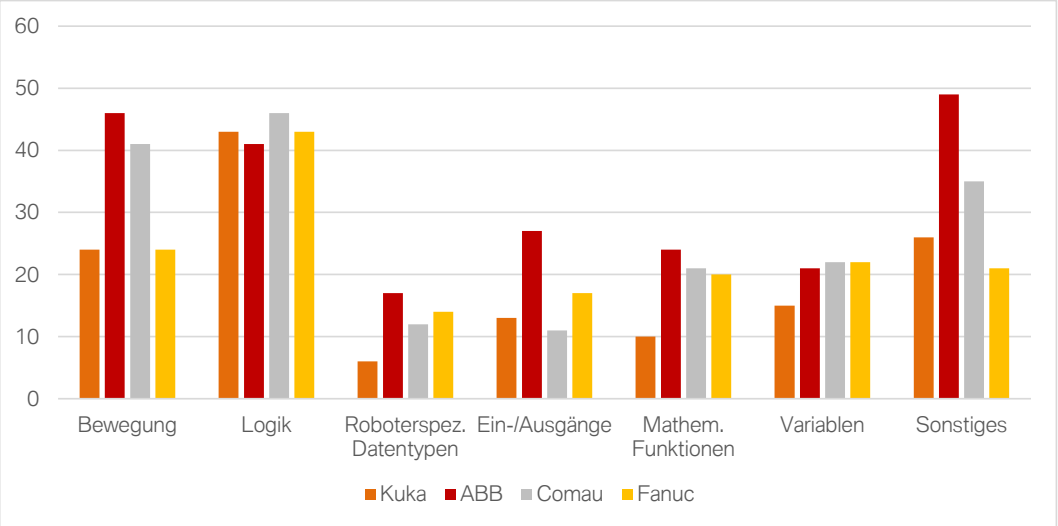


Abbildung 44 – Anzahl der als wichtig markierten Roboterbefehle

Durch die erstellte Kategorisierung und Aufzählung der Befehle wird deutlich, dass alle vier Hersteller grundsätzlich einen ähnlichen Befehlsumfang besitzen, der in Verteilung und Anzahl z. T. variiert. Die Differenzen ergeben sich vor allem durch Konzeptunterschiede (z. B. Datentypen bei Kuka/ABB oder dem TPE/KAREL Konzept bei Fanuc) bzw. unterschiedliche Qualität des Dokumentationsumfanges.

Eine Wertung hinsichtlich einer besser oder schlechter geeigneten Sprache kann und soll aus der vorliegenden Analyse nicht erfolgen. Im Rahmen von Expertengesprächen hat sich gezeigt, dass aus Sicht der Entwickler jede der untersuchten Programmiersprachen Vor- und Nachteile hat, jedoch die Präferenzen zu einem Sprachkonzept meist mit persönlicher Leidenschaft (z. B. erster Roboter) und Verwendungshäufigkeit eines Systems korreliert. Zudem kann aus der Anzahl an Befehlen keine direkte Anforderungseignung bzw. Qualität abgeleitet werden. Insbesondere Roboterexperten können sich auf Basis des gegebenen Funktionsumfanges ohnehin ihr spezifisches Befehlsset für die notwendigen Einsatzszenarien flexibel erstellen. Die Tatsache, dass alle vier Roboterhersteller im Karosseriebau eingesetzt werden können, unterstreicht, dass der etwaige unterschiedliche Sprachenumfang zumindest kein Einsatzhindernis darstellt.

Der Fokus der Analyse lag folglich vor allem im zahlenmäßigen Festhalten und Gruppieren des Befehlsumfanges, um eine Datenbasis zu schaffen, welche den Funktionsumfang der Herstellersysteme beschreibt und im Folgenden als Ausgangsbasis für weitere Untersuchungen und Anforderungsableitungen dient.

Herstellerunabhängige Programmiersprache IRL

Im Kapitel zum Stand von Wissenschaft und Technik wurden bereits einige Ansätze zur herstellerunabhängigen Programmierung von Robotersystemen formuliert, von denen sich bisher keiner im industriellen Umfeld durchsetzen konnte. Insbesondere die Industrial Robot Language (IRL) [26] stellt eine interessante Alternative dar, da sie in Aufbau und Syntax den gegenwärtigen Roboterprogrammiersprachen ähnelt und im Gegensatz zu allgemeinen Hochsprachen spezifisch auf Roboteranwendungen ausgelegt ist. Da sich die IRL nicht als industrieller Standard etablieren konnte, stellt sich die Frage, ob dies evtl. funktional bzw. technisch begründbar ist und ob diese sich für den industriellen Einsatz überhaupt eignet. Hierfür wurde die IRL in die entwickelte Befehlsdatenbank aufgenommen und abgeglichen, inwiefern der von den proprietären Roboterprogrammiersprachen vorgegebene Funktionsumfang durch die IRL abgedeckt wird und welche Defizite aus technischer Sicht vorhanden sind.

Es zeigt sich, dass mit insgesamt knapp 200 spezifizierten Befehlen, ein größerer Anweisungsumfang als mit der Kuka KRL definiert ist, jedoch die Funktionsmenge geringer als bei ABB, Comau und Fanuc ausfällt. Mit einem Anteil der wichtigen Befehle von 75 % bietet die IRL eine hohe Abdeckung relevanter Karosseriebaufunktionen.

Dies zeigt sich auch in der kategorisierten Gegenüberstellung der wichtigen Befehle mit dem Durchschnitt der proprietären Programmiersprachen in Abbildung 45.

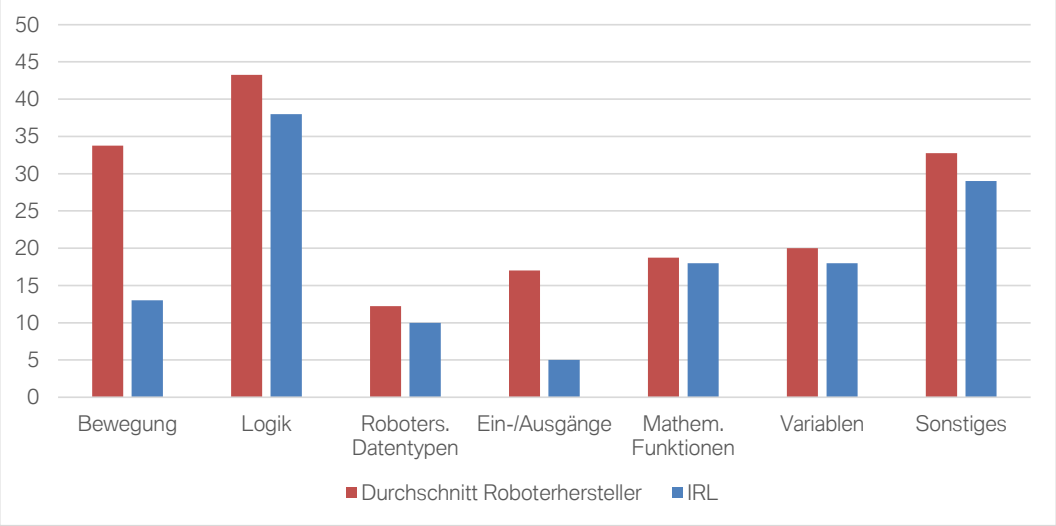


Abbildung 45 – Vergleich der Anzahl an als wichtig markierter Roboterbefehle der Hersteller und der IRL

In den Kategorien Bedingungen, Datentypen, Logik, mathematische Funktionen, Sonstiges und Variablen deckt die IRL sowohl zahlenmäßig als auch inhaltlich einen ähnlichen Befehlsumfang ab. Dies ergibt sich zum einen aus der quantitativen Vergleichbarkeit der Befehlsmenge. Zum anderen erschließt sich die semantische Überdeckung aus der Tatsache, dass die Datenbankeinbindung der IRL ohne neue Tokenerstellung für wichtige Roboterbefehle erfolgen konnte. Es mussten somit keine neuen Funktionen zu den vorhandenen Befehlssätzen definiert werden. Funktionale Defizite zeigen sich in den Kategorien Bewegung und Ein-/Ausgänge, da diesbezüglich IRL im Vergleich weniger Befehle bietet. Bei den Bewegungsbefehlen deckt die IRL-Norm zwar die grundsätzlichen Bewegungsbefehlssätze wie Punkt-zu-Punkt- und Bahnbewegungen ab, jedoch sind erweiterte Befehle (z. B. Pfadüberprüfungen) nicht gegeben und Instruktionen zu bahnsynchronen Operationen (für z. B. Kleben) nicht mit dem gleichen Umfang wie bei den Roboterherstellern vorhanden. Analog verhält es sich in der Gruppe der

Ein-/Ausgangs-Befehle. Die Basisfunktionen zum Schalten und Lesen von Ein- und Ausgängen sind definiert, allerdings sind keine erweiterten Funktionen wie Gruppensignale oder Prüfsignale spezifiziert. Auf eine Bewertung der syntaktischen Eignung von der IRL wurde verzichtet, da diese sich im Aufbau der KRL ähnelt und die Abweichung zu den anderen Programmiersprachen somit nicht größer ausfällt.

Zusammenfassend lässt sich festhalten, dass die IRL im Kern den notwendigen Basisbefehlsumfang abdeckt, aber einige Defizite bei vor allem erweiterten Funktionen existieren. Dies ist insbesondere dem Alter der Spezifikation geschuldet, da seit Veröffentlichung dieser, neue Funktionsausprägungen in den proprietären Sprachen integriert wurden und eine Anpassung der IRL-Norm nicht stattgefunden hat. Eine Verwendung von IRL für die Programmierung heutiger Systeme ist grundsätzlich möglich, wobei Spezifikationserweiterungen im Bereich der Bewegungs- und I/O-Operationen notwendig sind, um einen zeitgemäßen Instruktionsbestand abzubilden.

6.3.2. Programmierung von Automobilapplikationen am Beispiel der BMW AG

Bereits in Kapitel 4 wurde erläutert, dass die Integration von Industrierobotersystemen in ein Automobilwerk ein langwieriger Prozess ist, da eine Vielzahl von Anpassungen während der unternehmensspezifischen Standardisierung durchgeführt werden muss. Der folgende Abschnitt beschreibt am Beispiel der BMW AG zunächst den Aufbau eines Roboterapplikationsstandards und die notwendigen Integrationsanpassungen. Anschließend wird anhand realer Karosseriebaufertigungslinien der Funktionsbedarf an Roboteroperationen aus Anwendersicht ermittelt und mit dem existierenden Funktionsumfang der Robotersysteme in der Befehlsdatenbank verglichen, um den Anforderungsbedarf an eine herstellerunabhängige Programmiersprache für den Karosseriebau abzuleiten.

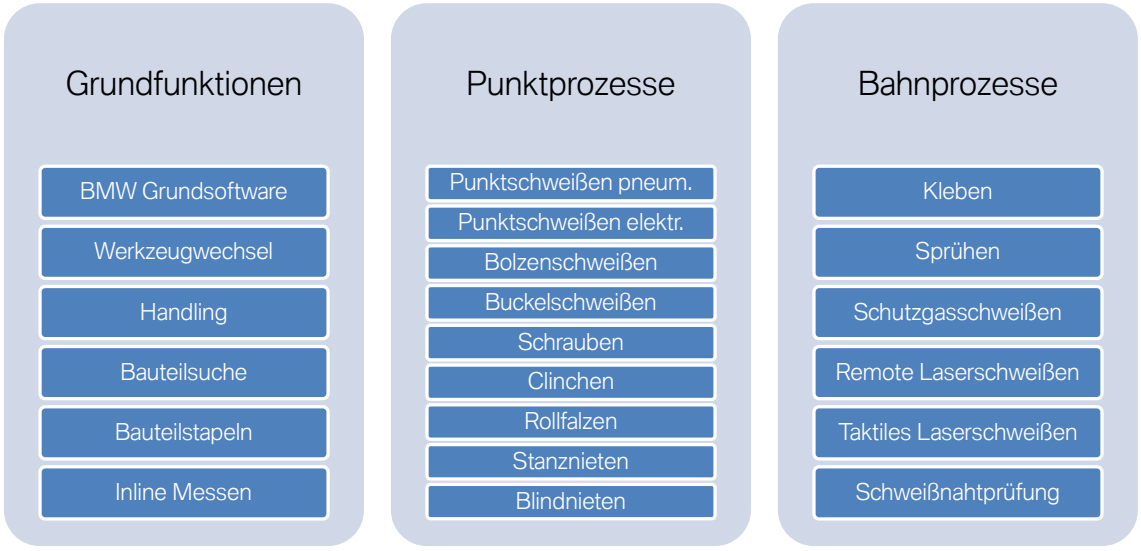


Abbildung 46 – Standardisierte Roboterapplikationen im Karosseriebau der BMW AG

Zentrales Element in der Roboterintegration der BMW AG sind die sogenannten Roboterapplikationsstandards (Abbildung 46). Diese orientieren sich an den Produktionsprozessen des Karosseriebaus und stellen deren softwaretechnische Umsetzung dar. Aufgrund der Systemproprietarität müssen diese für jedes Robotersystem eines Herstellers

individuell entwickelt werden. Neben Applikationen mit konkretem Fertigungsbezug existieren Basispakete, die übergreifend grundlegende Funktionen zur Verfügung stellen, beispielsweise für die Kommunikation mit der Zellensteuerung oder für Komponenten eines Werkzeugwechslersystems, das universell eingesetzt werden kann.

Anhand der Analyse der internen Dokumentation und der Befragung von Applikationsentwicklern hat sich gezeigt, dass sich die Applikationspakete aus übergreifender Sicht in der Umsetzung ähneln und sich ein Muster an wiederkehrenden Anforderungen ergibt, welches von den Softwarepaketen jeweils umgesetzt wird und die zentrale Fertigungsintelligenz einer Roboterapplikation im Karosseriebau enthält (Abbildung 47). Hierzu gehört zunächst das grundsätzliche Ablaufverhalten eines Prozesses in Form eines Zustandsautomaten mit den einzelnen Prozessschritten und der Überwachung des Vorgangs. Zudem sind zentrale Prozesseigenschaften parametrierbar, um Robotersystem bzw. zugehörige Komponenten für den Fertigungsvorgang einzustellen, wie etwa der Stellwinkel einer Klebedüse oder die Bahngeschwindigkeit beim Auftragen einer Klebenaht.

Ebenso ist die Kommunikation mit der übergeordneten und untergeordneten Peripherie ein essenzieller Bestandteil, um beispielsweise innerhalb der Anlagen mit korrektem Timing einen Prozess zu starten, den aktuellen Systemstatus mit dem Leitstand zu kommunizieren oder die einzelne Fertigungsoperation mit der angebundenen Aktorik und Sensorik durchzuführen. Hierbei spielt das intelligente Fehlermanagement eine bedeutende Rolle, da verschiedenartige Fertigungsprobleme rechtzeitig erkannt werden müssen und ein geschicktes Beheben der Ursachen notwendig ist, um teure Stillstandszeiten zu vermeiden, die sich schnell auch auf vor- und nachgelagerte Anlagen auswirken können. Aus diesem Grund sind verständliche und einfache Bedienfunktionen für Instandhalter wichtig, da diese einen transparenten und eindeutigen Status der Anlage kommunizieren sollen und gleichzeitig gewährleisten müssen, dass Eingriffe in den Ablauf und Anpassungen in das Fertigungsprogramm zügig vorgenommen werden können.

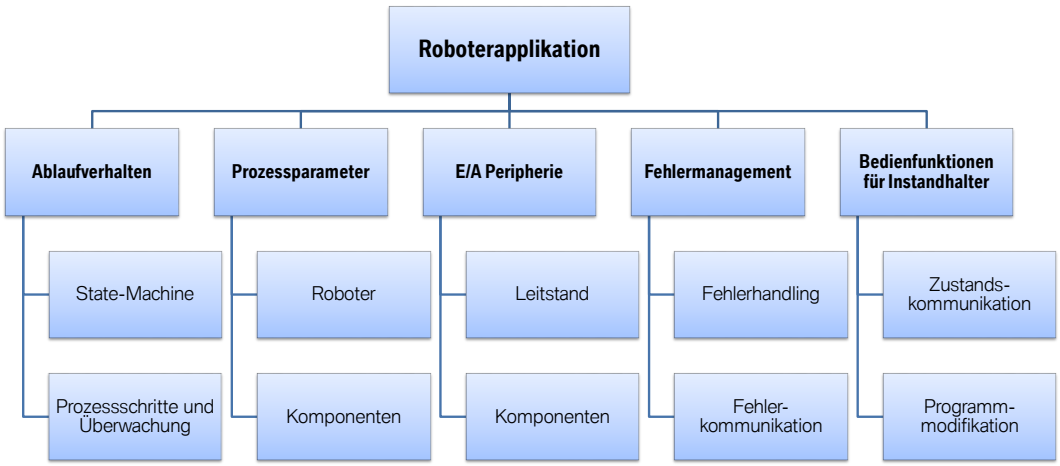


Abbildung 47 – Generische Bestandteile einer Roboterapplikation

Diese Softwarepakete geben die zentrale Prozesslogik und somit das fertigungstechnische Wissen wieder, besitzen aber keine anlagenspezifischen Bewegungsfolgen. Diese werden in der Offline-Simulation erzeugt und als Anwenderprogramm in die Anlagen übertragen. Die Applikationsprogramme sind die Grundlage für die eigentlichen Anwenderprogramme. Sie dienen als Basis, um Prozess- und Kommunikationskomplexität zu verbergen, Logiken werkeübergreifend zu vereinheitlichen, Qualitätsstandards zu sichern und Kosten zu reduzieren.

Abbildung 48 zeigt die Hierarchie der Programmlogik. Grundlage der Programmierung stellen die vom Roboterhersteller angebotenen Funktionsbefehle dar, die von sämtlichen Programmebenen verwendet werden. Über der Programmierebene des Roboterherstellers liegt die Ebene des Applikationsstandards. In dieser werden die dargestellten Basis- und Prozessfunktionen definiert. Auf oberster Ebene liegt das Anwenderprogramm. Dies ist für jedes eingesetzte Robotersystem unterschiedlich und baut auf den darunterliegenden Programmebenen auf.

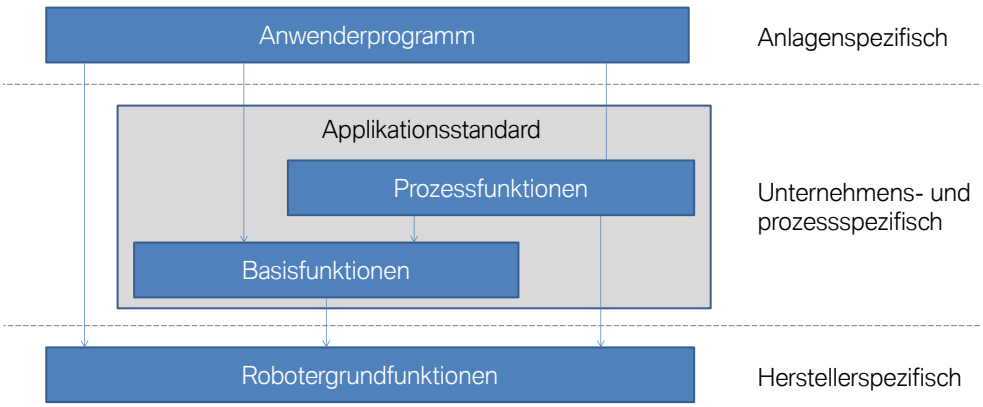


Abbildung 48 – Programmstruktur einer Roboteranwendung

Die zentrale Problematik bezüglich der Herstellerabhängigkeit liegt in der proprietären Programmiersprache in denen die Robotergrundfunktionen definiert sind. Sämtliche darauf aufbauende Befehle erben diese Abhängigkeit und Proprietarität, obwohl es sich um zunehmend anwenderspezifische Intelligenzen handelt. Zudem sind diese syntaktisch an die jeweilige Programmiersprache gebunden und in den verschiedenen Herstellersystemen unterschiedlich ausgeprägt.

Des Weiteren ist festzuhalten, dass auf Nutzerseite eine Zweiteilung der Anwender von Robotersystemen besteht. Zum einen existieren Roboter- und Prozessexperten, die zentrale Unternehmensstandards entwickeln. Zum anderen die Instandhalter- und Bedienerebene, die auf Basis dieser Standards die Verantwortung für den Betrieb der Anwenderprogramme trägt. Von Bedeutung ist, dass diese unterschiedlichen Nutzer auch unterschiedliche Erfahrung mit Robotersystemen und somit differenzierte Anforderungen an die Programmiersprache haben können.

Um das Thema der Herstellerabhängigkeit mit dem korrekten Anforderungsabbild anzugehen, ist es notwendig, den Befehlsbedarf auf den unterschiedlichen Ebenen zu ermitteln. Ziel ist, diesen Bedarf losgelöst von der proprietären Programmiersprache zu erfassen, um ein Anforderungsabbild zu generieren und alternative Programmiersprachen bewerten zu können. Hierfür wird das in Abbildung 49 dargestellte Vorgehen gewählt und eine Codeanalyse anhand von realer Roboteranlagen des Fallbeispiels durchgeführt.

Vorgehen

Im ersten Schritt werden sämtliche Roboterbackups der zu analysierenden Roboterlinien erfasst, welche die Datengrundlage der Analyse darstellen. Die Sicherungen werden anschließend in ihrer Programmstruktur analysiert und in Applikations- und Anwenderprogramme unterteilt. Im nächsten Schritt werden die in den Roboterprogrammen verwendeten Programmbefehle identifiziert und mit der Befehlsdatenbank abgeglichen. Jede Anweisung in der Befehlsdatenbank wird dabei um Verwendungsinformationen ergänzt, die Verwendungshäufigkeit, Befehlskategorie und den Programmtyp (Applikation/Anwender) dokumentieren. Darüber hinaus wird die Anzahl an Softwaremodulen, zugehörigen Applikationen und die Menge an analysierten Robotern festgehalten.

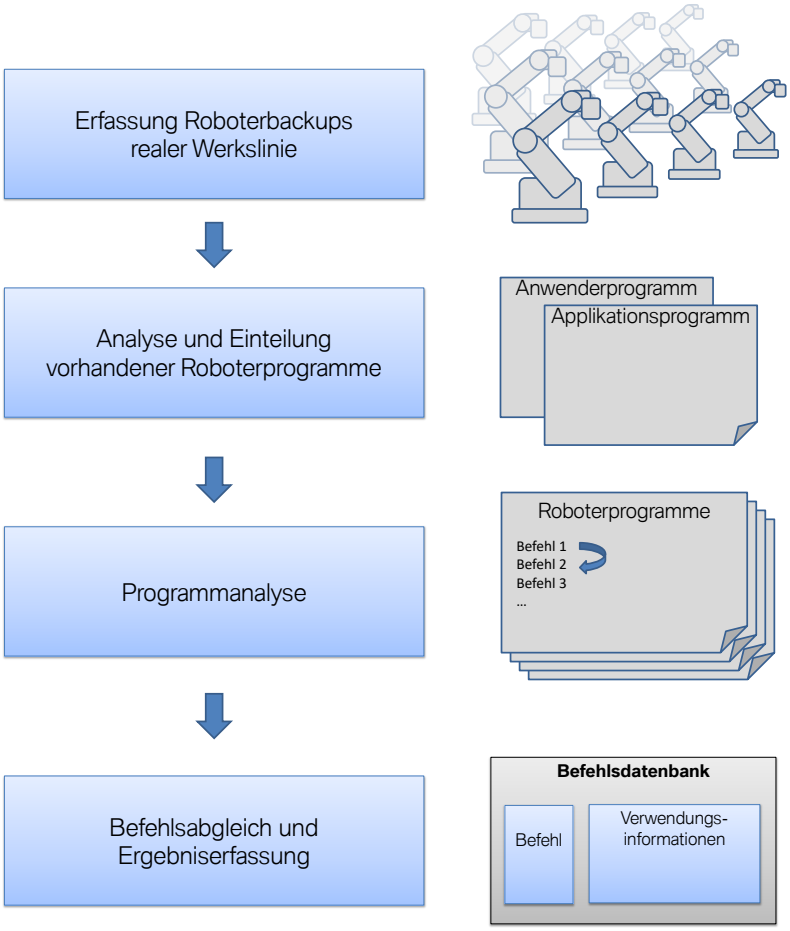


Abbildung 49 – Vorgehen Robotercodeanalyse

Da ein manuelles Verarbeiten und Abgleichen der Roboterbackups mit der Befehlsdatenbank sehr aufwändig wäre, wurde für das beschriebene Vorgehen ein Tool zur automatisierten Auswertung entwickelt. Mit dessen Hilfe ist es möglich, eine hohe Anzahl an Robotersoftwareständen in kurzer Zeit zu analysieren. Dies wurde am Beispiel zweier Werke der BMW AG durchgeführt. Die Ergebnisse werden im nachfolgenden Abschnitt beschrieben.

Ergebnis der Analyse

Allgemein

Im Rahmen der Arbeit wurden Karosseriebaulinien der BMW Werke Regensburg und Dingolfing analysiert. Im Werk Regensburg standen dafür die Backups von 899 ABB-Robotern zur Verfügung. Am Standort Dingolfing konnten 505 Kuka-Roboter analysiert werden. Insgesamt wurden über 32 Millionen Codezeilen verarbeitet und fast 24 Millionen Befehle identifiziert.

Tabelle 9 – Übersicht Robotercodeanalyse

Werk	Regensburg (ABB)	Dingolfing (Kuka)	Gesamt
Roboter	899	505	**1.404**
SW-Programme	21.145	48.717	**69.862**
Codezeilen	20.099.510	12.551.495	**32.651.005**
Befehle	13.761.448	9.528.478	**23.289.926**
Befehlstypen	381	198	
Davon BMW-spezifisch	101	89	
SW-Programme/Roboter	24	96	**50**
Codezeilen/Roboter	22.358	24.854	**23.256**
Befehle/Roboter	15.308	18.868	**16.588**

Die auf den einzelnen Roboter heruntergebrochenen Werte zeigen, dass die Karosseriebaulinie in Dingolfing eine höhere Anzahl an Software-Programmen besitzt. Zudem enthalten die SW-Bausteine im Durchschnitt mehr Codezeilen und Befehle pro Roboter. Zu erwähnen ist, dass der Regensburger Karosseriebau im Vergleich zu Dingolfing einen älteren Applikationsstandard verwendet. Des Weiteren unterscheiden sich beide Werke hinsichtlich der produzierten Automobile. Während in Regensburg vor allem PKW kleiner und mittlerer Fahrzeuggrößen (z. B. 1er, 3er) hergestellt werden, laufen in Dingolfing Fahrzeuge des oberen Produktsegments (z. B. 5er, 6er, 7er, 8er) der BMW AG vom Band. Diese erfordern aufgrund ihrer Größe und Karosseriestruktur eine höhere Anzahl an Fertigungsschritten, werden dafür aber in geringerer Stückzahl gebaut. Eine abweichende Charakteristik in der Programmkomplexität ist daher bereits angesichts der Produktions- und Produktstruktur gegeben, da mehr Fertigungsschritte pro Roboter ausgeführt werden müssen. Eine valide Aussage hinsichtlich des Zusammenhangs zwischen Roboterhersteller und Programmstruktur kann somit nicht direkt geschlussfolgert werden.

Eine bessere Vergleichbarkeit liefert die in Abbildung 50 dargestellte Befehlsmenge pro Roboter in Bezug auf den Programmtyp (Anwender-, Applikationsprogramm). Es ist ersichtlich, dass beide Werke eine wesentlich höhere Anzahl an Befehlen in standardisierten Applikationsprogrammen als in anlagenspezifischen Anwendungsprogrammen aufweisen.

Dies bestätigt, die im Rahmen der Dokumentenanalyse gewonnene Erkenntnis, dass das zentrale Prozess-Know-how in den Applikationspaketen und nicht in den Anwenderprogrammen enthalten ist, da davon auszugehen ist, dass mit steigender Befehlsmenge eine erhöhte Programmkomplexität und Prozessintelligenz einhergehen.

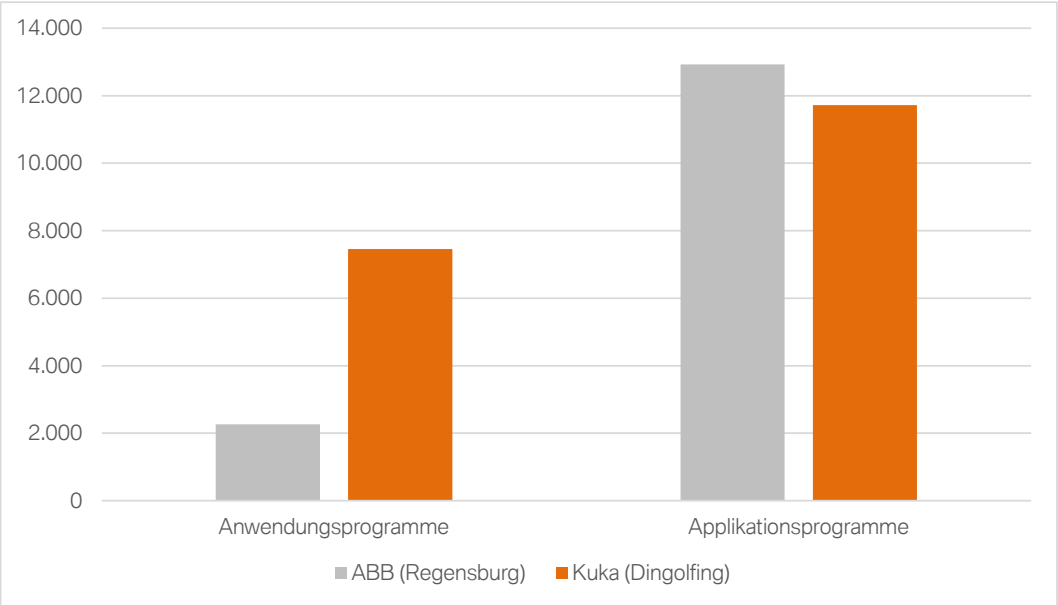


Abbildung 50 – Anzahl an Befehlen pro Roboter

Während die Befehlsanzahl pro Roboter in standardisierten Applikationsprogrammen für beide Karosseriebaustandorte auf ähnlichem Niveau liegt, unterscheidet sich diese Kennzahl bei den Anwendungsprogrammen um mehr als den Faktor 3. Zum einen kann dies mit der erwähnten, unterschiedlichen Produktstruktur der Produktionsstandorte erklärt werden, die mit anderen Taktzeiten, Produktgrößen und Produktkomplexitäten einhergeht und eine höhere Fertigungsschrittanzahl und somit auch eine höhere Befehlsmenge pro Roboter im Werk Dingolfing zur Folge hat. Zum anderen ist der Unterschied auch durch die Kuka-spezifischen FOLD-und ENDFOLD-Anweisungen bedingt, welche einen erheblichen Teil der identifizierten Instruktionen ausmachen.

Neben der Anzahl an Fertigungsschritten lässt sich anhand der Art der installierten Applikationsprogramme ebenfalls eine Aussage bezüglich der eingesetzten Fertigungstechnologien treffen. Abbildung 106 (Anhang 9.4.1) zeigt auf, welchen Anteil spezifische Applikation in den analysierten Roboterprogrammen haben. Sowohl in Dingolfing als auch in Regensburg sind die Anwendungen Handling, Punktschweißen und Werkzeugwechsel die am häufigsten eingesetzten Applikationsprogramme. Jedoch ist der Anteil an Punktschweißprozessen in der Fertigungslinie mit Kuka-Robotern zugunsten alternativer Prozesstechnologien wie Kleben und Nieten geringer.

Ursachen hierfür sind sowohl in der Produktionstechnologie als auch in der Produktstruktur begründet. Zum einen ist der Applikationsstandard in Dingolfing jünger, umfasst fertigungs- und steuerungstechnische Weiterentwicklungen und bietet dementsprechend die Möglichkeit, neuere und komplexere Applikationen einzusetzen, wie z. B zusätzliche Kamerasysteme, welche sich auch in dem höheren Anteil an Applikationen der „Bauteilsuche“ wiederfindet, oder zunehmend fertigungstechnische Alternativen zum klassischen Punktschweißen, wie z. B. Kleben. Zum anderen erlaubt die höhere Wertigkeit des Produktsegments in Dingolfing auch den Einsatz innovativerer und damit i. d. R. auch teurerer Technologien. Der Einsatz dieser Technologien wird häufig auch produktseitig gefordert, beispielsweise um notwendige Misch- und Leichtbaukonzepte im Premiumfahrzeugsegment zu realisieren.

Befehlstypen

Während im vorangegangenen Abschnitt die quantitative Befehlsverwendung der Karosseriebauanlagen betrachtet wurde, soll im folgenden Teil detaillierter auf die Art der verwendeten Instruktionen eingegangen werden.

Ein erster Eindruck ergibt sich aus der in Abbildung 51 dargestellten Statistik. Diese zeigt die Anzahl der verschiedenen Befehlstypen, welche in allen Roboterprogrammen und in den Teilen der Anwendungs- bzw. Applikationslogik identifiziert wurden. Zudem wird unterschieden, ob die Befehle proprietäre Anweisungen des Roboterherstellers oder BMW-spezifische Instruktionsspezifikationen sind.

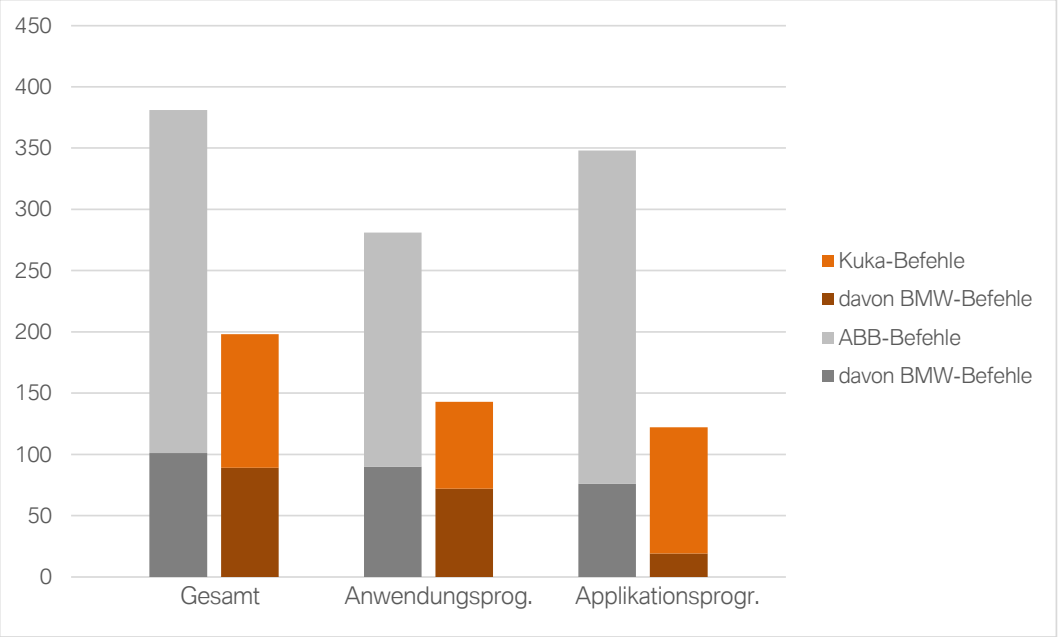


Abbildung 51 – Anzahl an Befehlstypen

Zunächst zeigt sich, dass die Rahmen der Programmiersprachenanalyse (Seite 110) ermittelte höhere Befehlsanzahl der ABB-Roboter sich auch im Anlagencode des Produktionsstandortes Regensburg wiederfindet. Insgesamt konnten 381 verwendete ABB-Instruktionen ermittelt werden. In den Steuerungen der Kuka-Roboter wurden 198 Programmanweisungen identifiziert. Während sich die Gesamtzahl an Befehlstypen damit fast um den Faktor zwei unterscheidet, ist die Menge an spezifischen BMW-Elementen mit 101 (ABB) zu 89 (Kuka) auf ähnlichem Niveau.

Das ermittelte Verhältnis zwischen BMW- und Roboterherstellerbefehlsmenge bestätigt quantitativ die in Abbildung 48 dargestellte Hierarchielogik. Anwendungsprogramme bauen auf Applikations- und Robotergrundfunktionen auf, während Applikationsprogramme insbesondere Robotergrundfunktionen verwenden, um BMW-spezifische Prozess- und Basisfunktionen zur Verfügung zu stellen.

Um eine besser Aussage in Bezug auf die verwendeten Befehls- und Programmtypen zu treffen, werden die identifizierten Anweisungen in Tabelle 10 auf konkrete Befehle heruntergebrochen. In der Tabelle werden die 15 meist verwendeten Befehle sowohl für Anwendungs- als auch Applikationslogik aufgelistet. Neben der jeweiligen Instruktion sind zusätzlich die Kategorie und die insgesamt erfasste Anzahl bzw. die durchschnittliche Verwendungshäufigkeit eines Befehls pro Roboter angegeben.

Tabelle 10 – Meist verwendete Befehle auf Anwendungs- und Applikationsprogrammebene

Anwendungsprogramme

ABB (Regensburg)				KUKA (Dingolfing)			
Befehl	**Kategorie**	**Anzahl**	**Ø pro Roboter**	**Befehl**	**Kategorie**	**Anzahl**	**Ø pro Roboter**
robtarget	Datentyp	142862	159	DECL	Variable	620188	1228
CONST	Variable	136680	152	ENDFOLD	Systemfunktion	499194	989
MoveJ	Bewegung	112693	125	FOLD	Systemfunktion	499127	988
PERS	Variable	99309	110	DECL GLOBAL	Variable	392621	777
IF	Logik	95491	106	FDAT	Datentypen	369657	732
Reset	Ein/Ausgänge	93920	104	E6POS	Datentypen	314114	622
MoveJG	BMW-Funkt.	78917	88	LDAT	Datentypen	211510	419
Stop	Sonstiges	72483	81	BAS	Bewegung	160443	318
MoveL	Bewegung	61569	68	PDAT	Datentypen	148242	294
ENDIF	Logik	57057	63	PTP	Bewegung	108105	214
MoveAbsJ	Bewegung	56362	63	LIN	Bewegung	37585	74
PROC	Programmdekl.	54335	60	CASE	Logik	21693	43
ENDPROC	Programmdekl.	54332	60	WAIT FOR	Logik	21138	42
MoveLG	BMW-Funkt.	37034	41	END	Programmdekl.	19071	38
OR	Logik	36286	40	Grp PartChk	BMW-Funktion	18898	37

Applikationsprogramme

ABB (Regensburg)				KUKA (Dingolfing)			
Befehl	**Kategorie**	**Anzahl**	**Ø pro Roboter**	**Befehl**	**Kategorie**	**Anzahl**	**Ø pro Roboter**
IF	Logik	1009967	1123	IF	Logik	485830	962
CASE	Logik	770138	857	ENDIF	Logik	485830	962
VAR	Variable	735111	818	$OUT	Ein/Ausgänge	328849	651
ENDIF	Logik	651011	724	$IN	Ein/Ausgänge	276644	548
string	Variable	455513	507	CONTINUE	Logik	256759	508
num	Variable	376945	419	GLOBAL INT	Variable	223116	442
AND	Logik	276714	308	DECL	Variable	178325	353
ELSE	Logik	254791	283	AND	Logik	152416	302
PERS	Variable	245385	273	CASE	Logik	139263	276
Reset	Ein/Ausgänge	237949	265	ENDFOLD	Systemfunktion	130124	258
Present	Systemfunktion	233047	259	FOLD	Logik	129516	256
Stop	Sonstiges	231411	257	INT	Variable	122738	243
NumToStr	Variable	208988	232	NOT	Logik	117418	233
RETRY	Systemfunktion	201993	225	END	Programmdekl.	105989	210
TEST	Logik	186304	207	WAIT FOR	Logik	102008	202

Bei einer ersten Betrachtung des oberen, anwendungsprogrammbezogenen Teils fällt auf, dass die durchschnittliche Befehlsanzahl pro Roboter bei den Kuka-Systemen weitaus höher liegt als bei den analysierten ABB-Maschinen. Mit ENDFOLD und FOLD sind zwei Systemfunktionen unter den drei am meisten verwendete Instruktionen. Eine einheitliche herstellerübergreifende Befehlscharakteristik in der Anwendungsebene ergibt sich daher nicht sofort und eine genauere Untersuchung und Erläuterung der Instruktionen ist lohnenswert.

Zunächst soll die Befehlsverteilung des ABB-Roboters in der Anwendungslogik (Tabelle 10 – oben links) besprochen werden. Bei dem Befehl ROBTARGET handelt es sich um einen Datentypen zur Erfassung erweiterter Positionsdaten von Robotern. Meist verwendet, um Ziele

für Roboterbewegungen zu berechnen bzw. vorzugeben. Das Kuka-Äquivalent lautet E6POS und ist ebenfalls in der Tabelle enthalten. Die auf Position zwei und drei eingeordneten Anweisungen sind klassische Programmieranweisungen zur Deklaration von Variablen (CONST) und zur Ausführung von Punkt-zu-Punkt-Bewegungen auf Achswertbasis (MoveJ). Zudem sind noch vier weitere Bewegungsbefehle aufgeführt. Neben den ABB-Instruktionen für Linear- (MoveL) und absoluten Punkt-zu-Punkt-Bewegungen (MoveAbsJ) sind mit MoveJG und MoveLG auch BMW-spezifische Funktionen aufgeführt, die Manipulatorbewegungen mit synchroner Ansteuerung der Schweißzangenaktorik ausführen.

Die Befehlsreihenfolge in Dingolfing ist sehr stark durch das Kuka-spezifische FOLD-Konzept geprägt, welches es ermöglicht, Anweisungen zu verschachteln und so Komplexität in der Programmhandhabung reduzieren soll. Auf diese Weise werden bei der Anzeige oder Eingabe von Bewegungsanweisungen notwendige Initialisierungsinstruktionen und -parameter wie BAS, FDAT, PDAT vor dem Bediener verborgen. Die aufgeführten Datentypen sind demensprechend stark mit Bewegungsoperationen für Punkt-zu-Punkt- und Linearbewegungen verbunden, welche in der Aufzählung mit PTP und LIN ebenso in einer hohen Anzahl vertreten sind. Da jede FOLD Anweisung mit einem ENDFOLD Befehl abgeschlossen werden muss, verwundert es, dass die Abschlussanweisung eine geringfügig höhere Anzahl aufweist. Begründet ist dies in der Existenz einiger ENDFOLD Anweisungen nach der eigentlichen Programmdefinition. Sie haben keinen Effekt auf die Anwendungslogik und stellen lediglich unnötige Coderelikte dar.

Ähnlich wie bei den ABB-Robotersystemen sind auch im Anwendungscode der Kuka-Steuerungen Deklarationen für Variable (DECL) und eine BMW-Funktion (Grp PartChk) aufgeführt, wobei letztere keine Bewegungsfunktion darstellt, sondern zur Abfrage der Bauteilkontrolle bei Handlingsoperationen dient. Im Bereich der Logikoperationen zeigt sich, dass bei den ABB-Robotern die Anweisungen IF bzw. ENDIF häufig verwendet werden, während diese Befehlstypen bei den Kuka-Anwenderprogrammen nicht in der Auflistung erscheinen. Dass sie grundsätzlich in KRL verfügbar sind und in den Anlagen verwendet werden, ist aus den Applikationsprogrammen im unteren Teil der Tabelle 10 ersichtlich. Ebenso zeigt sich hier eine weitere Herstellerspezifität. Während bei Kuka sowohl IF als auch ENDIF Anweisungen in exakt gleicher Menge vertreten sind, unterscheidet sich die Anzahl pro Roboter bei ABB mit 106 zu 63 in den Anwenderprogrammen. Hintergrund ist, dass bei KRL auch für eine einzelne IF-Anweisung ein ENDIF erforderlich ist, während bei RAPID dies erst bei der Eingabe von mindestens einer zweiten Bedingung mit ELSE erforderlich wird. Die in der Summe geringere Verwendung von IF-Befehlen bei Kuka-Robotern wird durch CASE-Anweisungen bei Fallunterscheidungen substituiert, da dieser Befehlstyp mit 43 Befehlen pro Roboter einen weitaus höheren Wert als bei ABB-Robotern (Ø 8 Instruktionen) besitzt. Zusammenfassend lässt sich für den Teil der Anwendungslogik festhalten, dass sowohl bei den untersuchten ABB- als auch Kuka-Robotern die Bewegungsbefehle und die dazugehörigen Datentypen einen Großteil der am häufigsten verwendeten Anweisungen darstellen. Nicht roboterspezifische Anweisungen sind vor allem bei der Variablendeklaration und Logikoperationen vorhanden.

Anders verhält es sich bei den analysierten Applikationspaketen im zweiten Teil der Tabelle. Bei beiden Herstellern sind in der Auflistung keine Bewegungsbefehle oder roboterspezifische Datentypen zu finden. Es werden vor allem klassische Programmieranweisungen wie Bedingungen, Variablentypen, Deklarationen und Logikelemente verwendet. Zudem erfolgt dies in einer weitaus höheren Zahl als auf Anwendungsebene. Die einzigen roboter- bzw. automatisierungsspezifischen Befehle sind Anweisungen zur Ansteuerung von Ein- und Ausgängen (z. B. $OUT, $IN), welche bei den Kuka-Systemen sehr häufig verwendet werden

und unter den fünf meist verwendeten Operationen liegen. Da ABB ein Konzept der variablen Namensgebung für I/Os verfolgt, ist hier lediglich RESET an Position zehn als Befehl für das Steuern von Ein- und Ausgängen aufgeführt.

Zwar zeigt sich in den aufgeführten Daten bereits eine klare Tendenz der Befehlstypen in Bezug auf die unterschiedlichen Roboterprogrammebenen, allerdings ist dies auf die meist verwendeten Operationen beschränkt. Aus diesem Grund wird in Abbildung 52 eine Darstellungsform gewählt, welche die prozentuale Verteilung der Befehlsmenge und -kategorien im Vergleich zueinander setzt sowie nach Programmkategorie und Roboterhersteller differenziert. Aufgrund der starken Herstellerspezifität und schlechten Vergleichbarkeit wurde die Datenbasis um die Gruppe der Systemfunktionen (z. B. FOLD) bereinigt.

Die auf Basis von Tabelle 10 getroffene Aussage, dass Bedingungen, Programmlogik und I/O-Befehle insbesondere auf Applikationsebene verwendet werden, bestätigt sich auch in dieser Auswertung. Es ist ebenfalls ersichtlich, dass Bewegungen, BMW-Funktionen und roboterspezifische Datentypen vor allem auf Anwendungsebene eingesetzt werden. Beide Aussagen treffen unabhängig vom eingesetzten Roboterhersteller zu. Lediglich die Kategorien Sonstiges und Variablen zeigen kein eindeutiges Bild. Die Kategorie Sonstiges ist infolge der heterogenen Befehlszusammensetzung nur schwer bewertbar. Bei der Gruppe der Variablen lässt sich festhalten, dass diese sowohl im Bereich der Anwendung- als auch Applikationslogik eine hohe Verwendung haben. In der Detailbetrachtung der Tabelle 10 zeigt sich zudem, dass auf Anwenderebene vor allem die Deklaration von Variablen mit DECL und CONST in Kombinationen mit den roboterspezifischen Datentypen (z. B. robtarget, FDAT) aufgeführt sind, wohingegen auf Applikationsebene verstärkt klassische Variablentypen wie INT, NUM, STRING verwendet werden. Begründet ist dies durch die Charakterstik der Programmstruktur, da Anwenderprogramme vor allem Bewegungen mit den zugehörigen Positionen definieren und abfahren, während Applikationsprogramme verstärkt Logikanweisungen enthalten und dementsprechend einen höheren Anteil an nicht roboterspezifischen Variablentypen verwenden. Die Aufteilung zeigt sich noch besser im direkten Vergleich der Bewegungsbefehle und der Logikbefehle in Abbildung 52. Der Anteil an Bewegungsbefehlen ist bei den Anwenderprogrammen um ein Vielfaches höher als bei den Applikationsprogrammen. Der Anteil an Logikoperationen ist wiederum auf Appliaktionsebene besonders groß.

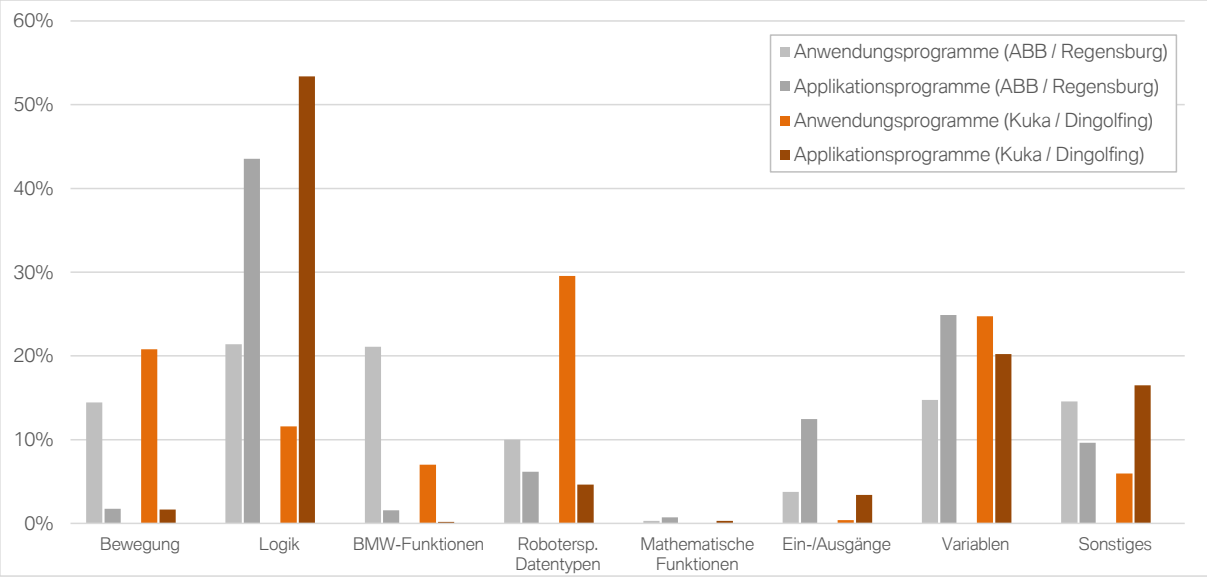


Abbildung 52 – Verteilung der verwendeten Instruktionen anhand der Befehlskategorie und Hersteller

6.3.3. Schlussfolgerung und Fazit

Aus den vorangegangenen Abschnitten der Befehlsanalyse auf Basis der Herstellerdokumente und Codeanalyse von realen Roboteranlagen können folgende Erkenntnisse festgehalten werden: Die dokumentierten Instruktionen lassen sich herstellerübergreifend kategorisieren, unterscheiden sich jedoch in der Gesamtzahl. Beispielsweise bietet ABB RAPID im vgl. zu Kuka KRL nach dem Dokumentationsstand ein Dreifaches an Befehlen an (Abbildung 41). Teilweise sind auch unterschiedliche Schwerpunkte in den einzelnen Kategorien ersichtlich. So ist die Menge der definierten Datentypen bei KRL geringer als bei anderen Roboterherstellern. Für die meisten Befehlskategorien zeigt sich allerdings ein einheitliches Verteilungsbild (Abbildung 44).

Trotz der unterschiedlichen Befehlsmenge und der teilweise unterschiedlichen Fokussierung lassen sich die gleichen Produkte mit den gleichen Fertigungsprozessen mit Robotern unterschiedlicher Hersteller produzieren. In der vorliegenden Arbeit wurden daher die BMW-Produktionsstandorte Regensburg und Dingolfing mit Kuka- bzw. ABB-Robotern genauer analysiert. Es zeigt sich, dass die Herstellerspezifitäten auch Auswirkungen auf die Produktionslinien haben. So ist die größere Befehlsmenge von RAPID im vgl. zu KRL auch im real verwendeten Befehlsumfang ersichtlich (Abbildung 51), wobei sowohl bei ABB als auch bei Kuka ca. 65 % der zur Verfügung stehenden Funktionen in den Anlagen verwendet werden. Ein Großteil der genutzten Befehle sind unternehmenseigene automobilspezifische Funktionen, die unabhängig vom Roboterlieferanten und dessen Programmiersprache eine Menge von ca. 100 Instruktionen umfasst. Die hinter diesen Instruktionen steckende Logik beinhaltet keine anlagenspezifischen Intelligenzen für eine konkrete Fertigungsinstanz. Vielmehr handelt es sich um einen Unternehmensstandard, der das relevante automobile Prozesswissen abbildet und unabhängig vom hergestellten Produkt Fertigungs-Know-how zur Verfügung stellt. Der Anteil dieser universellen Applikationsprogramme ist um ein Vielfaches höher, als das für eine bestimmte Anlage entwickelte Anwendungsprogramm (Abbildung 50).

Diese strukturelle Zweiteilung der Softwarekomponente zeigt sich ebenfalls in den darin verwendeten Befehlen und gibt Aufschluss darüber, welche programmiertechnischen Schwerpunkte in den jeweiligen Komponenten gesetzt werden bzw. welche Anforderungen an eine Programmiersprache daraus abgeleitet werden können. Dies wird noch einmal in Abbildung 53 verdeutlicht, die pro Befehlskategorie die Verwendungspräferenz der analysierten Programmkomponenten, Beispielbefehle und die zugehörige Domäne der Programmiersprache darstellt.

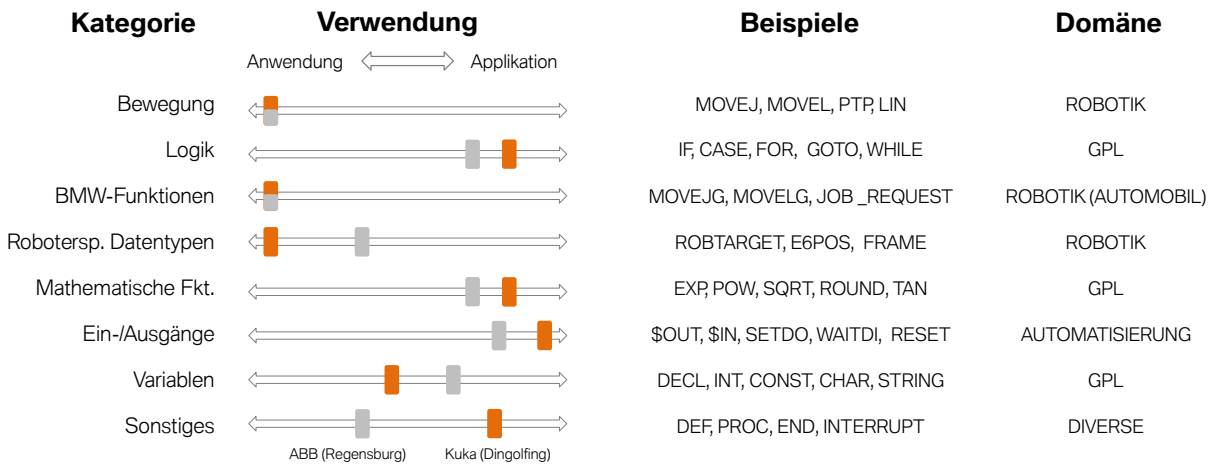


Abbildung 53 – Verteilung der verwendeten Befehle anhand Kategorie, Hersteller (BMW-Werk) und Domäne

6.4. Alternative herstellerunabhängige Programmiersprachen für Robotersysteme

Im vorangegangenen Kapitel wurden sowohl gegenwärtige Roboterprogrammiersprachen als auch die reale Befehlsverwendung in Produktionsanlagen analysiert, um ein funktionales Anforderungsbild in Form eines Befehlskataloges aus Sicht der Roboterhersteller (vorhandene Instruktionen) und des Karosseriebaus (tatsächlich genutzte Instruktionen und notwendige Erweiterungen) zu erstellen. Im nächsten Teil der Arbeit wird erörtert, welche Programmiersprachen neben den klassischen Robotersprachen als Alternative existieren, um den identifizierten Befehlsumfang herstellerunabhängig und nutzerzentriert umzusetzen. Aufgrund der durch die Externalisierung gewonnenen Gestaltungsfreiheit bietet sich aus technischer Sicht grundsätzlich die Möglichkeit, jede existierende Programmiersprache als Alternative einzusetzen oder ggf. auch ein eigenes Sprachenkonzept zu entwickeln. Da eine Evaluierung aller verfügbaren Programmiersprachen weder zielführend noch effizient ist, sollen diese zunächst anhand der Programmiersprachenkategorien beschrieben und an einzelnen Beispielen dargestellt werden.

Roboterprogrammiersprachen

Unter Roboterprogrammiersprachen sind Sprachen zu verstehen, die dezidiert für die Programmierung von Roboteranwendungen konzipiert und den domänenspezifischen Programmiersprachen zuzuordnen sind. Der Großteil der Programmiersprachen fällt dabei unter proprietäre roboterorientierte Programmiersprachen, wie beispielweise die in Kapitel 6.3.1 analysierten Programmiersprachen für Industrieroboter. Oft orientieren sich diese in Syntax an allgemeinen Programmiersprachen (wie z. B. Pascal). Des Weiteren werden textuelle aufgaben-orientierte Programmiersprachen ebenfalls den Roboterprogrammiersprachen zugeordnet [143]. Diese sind in der konventionellen Industrierobotik bisher nicht von Bedeutung. Darüber hinaus müssen diese final auch wieder in roboterorientierte Anweisungen übersetzt werden, um die Ausführung am eigentlichen Robotersystem spezifizieren und gewährleisten zu können.

Mögliche Unterteilungen:
- Proprietär und offen; roboterorientiert und aufgabenorientiert

Herstellerabhängigkeit:
- In der Regel vorhanden, da proprietäre Sprachen der Hersteller
- Ausnahmen stellen normierte Sprachen dar (z. B. IRL)

Beispiele: KRL, RAPID, PDL2, IRL

Automatisierungstechnik-Programmiersprachen

Eine zwar geringe, aber speziell für Nischenanwendungen zunehmende Rolle hat die Roboterprogrammierung mit klassischen Programmiersprachen der Automatisierungstechnik. Im Gegensatz zur Robotik existieren mit den Normen EN 61131-3 [144] und DIN 66025 [73] in diesem Bereich industriell etablierte Standards. Erstere spezifiziert die Programmiersprachen AWL (Anweisungsliste), KOP (Kontaktplan), FBS (Funktionsbausteinsprache), AS (Ablaufsprache) und ST (Strukturierter Text) für die Programmierung von speicherprogrammierbaren Steuerungen, wobei lediglich AWL und ST als Textsprachen definiert sind und AWL aufgrund seiner starken Ähnlichkeit zu Assemblersprachen als veraltet gekennzeichnet ist [144]. Zwar orientieren sich die SPS-Hersteller in der Regel an der Norm und erfüllen die definierte Spezifikation für grundlegende Operationen [145], jedoch sind in den Programmbibliotheken meist herstellerspezifische Umsetzungen und Erweiterungen enthalten, die eine direkte und vollständige Portierung zwischen den Systemen verhindern.

Ähnlich verhält es sich im Werkzeugmaschinenbereich. Die DIN 66025 bzw. ISO 6983 [146] definieren mit dem sogenannten „G/M-Code“ bereits seit mehreren Jahrzehnten sowohl Befehle als auch Programmaufbau für numerisch gesteuerte Arbeitsmaschinen. Allerdings ist angesichts der herstellerspezifischen Befehlserweiterungen eine Portierbarkeit zwischen unterschiedlichen Systemen nur eingeschränkt möglich und es bedarf spezifischer Postprozessoren, um den korrekten Herstellerdialekt zu generieren [147, 148]. Zudem besitzt G-Code konzeptbedingt keine maschinenübergreifende Kompatibilität, da TCP-Anweisungen in Abhängigkeit zu dem Koordinatensystem des Werkzeugs und nicht des Werkstücks formuliert werden [149]. Um diesem Nachteil zu entgegnen, wurde mit STEP-NC ein Programmierinterface in den Normen ISO 14649 [150] und DIN 10303-238 [151] definiert, das werkstückbezogene Arbeitsschritte inklusive Prozessparameter (z. B. Toleranzen) statt Werkzeugbewegungen definiert und die Verwendung von Postprozessoren obsolet machen soll. Jedoch konnte dieser Standard G-Code in der Industrie bisher nicht umfassend ablösen und die Entwicklung vollständiger STEP-NC Controller ist weiterhin ein gegenwärtiger Forschungsgegenstand [152]. Nichtsdestotrotz ist im Gegensatz zur EN 61131-3 oder Industrierobotersprachen die Portierbarkeit prinzipiell besser gegeben und ein zentrales Ziel der Normierungsinitiative. Allerdings ist das Arbeitsspektrum einer NC-Maschine im Vergleich zur Robotik oder SPS-Technik wesentlich begrenzter und damit auch leichter zu formalisieren.

Infolge der zunehmenden Verwendung von Robotersystemen im Spezialmaschinenbau existieren inzwischen auch vermehrt Lösungen, die Programmierkonzepte der Automatisierungstechnik für Anwendungen mit Robotersystemen integrieren. Beispielsweise die Verwendung von G-Code und STEP-NC zur Ansteuerung von Robotersystemen (Kuka NC, [153], [74] oder die Programmierung von Robotersystemen über ST bei den generischen Steuerungslösungen von B&R. Diese haben indes nur ein sehr eingeschränktes Anwendungsfeld (z. B. Fräsen) bzw. keine standardisierten Funktionsumfänge für Roboteranwendungen (ST).

Mögliche Unterteilungen:
- Anwendungsfeld (speicherprogrammierbare Steuerung, Werkzeugmaschinen)

Herstellerabhängigkeit:
- Infolge herstellerspezifischer Erweiterungen und Werkzeugorientierung meist vorhanden
- STEP- NC prinzipiell herstellerunabhängig, aber mangelnde industrielle Unterstützung

Beispiele: G-Code, STEP-NC, ST, AWL

Allgemeine Programmiersprachen

Als dritte relevante Programmiersprachenkategorie sind die allgemeinen Programmiersprachen (General Purpose Languages) zu nennen. Im Gegensatz zu domänenspezifischen Sprachen besitzen sie keine Ausrichtung auf einen speziellen Anwendungsbereich, sondern können für unterschiedliche Problemstellungen verwendet werden. Sie werden dementsprechend auch meistens in der klassischen Softwareentwicklung eingesetzt und sind mit Vertretern wie C++, Java oder Python weit verbreitet und bekannt. Ein zentrales Einordnungsmerkmal ist das Programmierparadigma, welches einer Sprache zu Grunde liegt [154]. Hierbei kann primär zwischen imperativen, deklarativen und objektorientierten Sprachen unterschieden werden, die sich nach Haun [143] und Fernández [155], wie folgt definieren:

Imperative Programmiersprachen bestehen aus Abfolgen von Anweisungsschritten, die zur Lösung eines Problems ausgeführt werden. Man nennt sie daher auch problemorientierte Sprachen. Sie stellen eins der ersten Programmierparadigmen dar. Maschinen- und Assemblersprachen sind ebenso den imperativen Programmiersprachen zuzuordnen. Im Laufe der Zeit haben sich aufgrund des Abstraktionsniveaus vor allem höhere imperative Sprachen als gängiges Werkzeug etabliert.

Beispiele: C, Fortran, Pascal

Deklarative Programmiersprachen unterteilen sich in logische und funktionale Programmiersprachen. Diese werden als deklarative Sprachen bezeichnet, da nicht beschrieben wird, wie ein Problem (logische Programmierung) oder eine Berechnung (funktionale Programmierung) durchgeführt werden soll, sondern wie ein Problem spezifiziert ist bzw. was berechnet werden soll.

Beispiele: Prolog, LISP, Haskell, Erlang

Objektorientiere Programmiersprachen ermöglichen es, Programme in einer Sammlung von hierarchisch aufgebauten Objekten zu definieren und auf deren Funktionalität und Daten über Methoden zuzugreifen. Die Objektorientierung wird auch als Ergänzung zur imperativen Programmierung gesehen. Es befinden sich allerdings ebenfalls Elemente der deklarativen Programmierung darunter.

Beispiele: C++, Java, Python, PHP

In der Robotik finden, insbesondere im Forschungsbereich, sämtliche Programmierparadigmen Anwendung [143]. In der industriellen Robotik erfolgt die Verwendung von allgemeinen Programmiersprachen auf Nutzerebene bisher nur sehr eingeschränkt. Lediglich Kuka bietet als einziger etablierter Roboterhersteller mit der Sunrise-Steuerung und Java eine objektorientierte Programmiersprache für Serviceroboter an. Aufgrund der Allgemeingültigkeit von objektorientierten Programmiersprachen unterliegen sie grundsätzlich keiner Herstellerabhängigkeit, jedoch existieren auch keine roboterspezifischen Sprachelemente. Aus diesem Grund werden meist projektspezifische Libraries entwickelt, die ähnliche bzw. gleiche Funktionen jeweils unterschiedlich implementieren. Eine direkte Portierung ist somit i. d. R. nicht möglich und mit Anpassungsaufwänden verbunden.

Herstellerabhängigkeit: Nicht gegeben, allerdings projektspezifische Implementierungen

Fazit

Die erwähnten Beispiele zeigen, dass sich in der Robotik prinzipiell alle Programmiersprachenkategorien und -paradigmen wiederfinden. Während in der Industrierobotik insbesondere roboterorientierte Sprachen verwendet werden, finden Allgemeinsprachen vor allem im wissenschaftlichen Bereich Anwendung. Automatisierungstechnische Sprachen werden vermehrt in Nischenanwendungen eingesetzt.

Um aus der in Kapitel 4.6 abgeleiteten Anforderung, ein anwendergerechtes und herstellerunabhängiges Programmierkonzept zu entwickeln, erscheinen zunächst alle beschriebenen Gruppen als valide Programmiersprachenoption. Daher sollen im Folgenden die Endanwenderanforderungen an Funktionalität und Herstellunabhängigkeit konkretisiert werden, um eine geeignete Auswahl für die Domäne des Karosseriebaus zu treffen.

6.5. Anforderungen an die Programmierung aus Anwendersicht

Abbildung 54 zeigt die Stakeholder eines Roboterprogramms in der Automobilproduktion, welche jeweils unterschiedliche Anwendungs- und somit auch unterschiedliche Anforderungsschwerpunke an die Roboterprogrammierung besitzen.

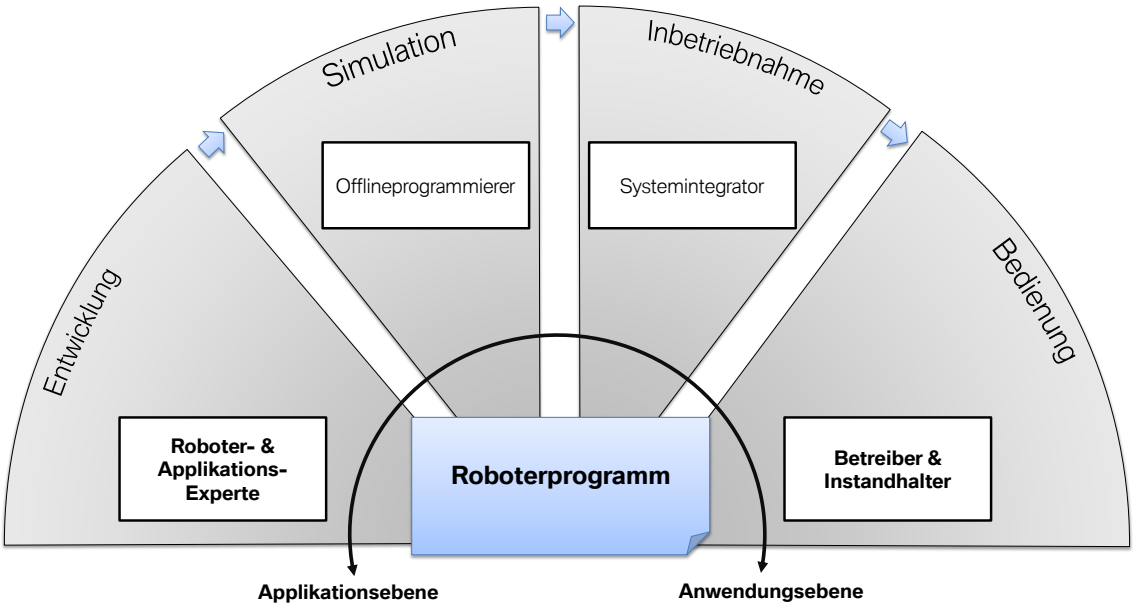


Abbildung 54 – Phasen und beteiligte Anwender an der Roboterprogrammentwicklung im Karosseriebau

Auf der linken Seite befindet sich die Rolle des Roboterexperten bzw. Applikationsentwicklers, deren Auftrag es ist, einen Unternehmensstandard für Roboterapplikationen zu entwickeln. Wie in Kapitel 6.3.2 anhand der Befehlsverwendung aufgezeigt, handelt es sich dabei um aufwändige und komplexe Programme, die das unternehmensspezifische Prozesswissen abbilden und die Roboterprogrammfunktionalitäten um kundenspezifische Merkmale erweitern. Die Softwareentwicklung erfolgt nicht mit dem Roboterbedienpanel, sondern in einer textuellen Offline-Entwicklungsumgebung sowohl bei dem OEM als auch den beteiligten Lieferanten. Die erstellten Programmbausteine dienen als Universallösungen, die an unterschiedlichen Produktionsstandorten in unterschiedlichen Anlagen Verwendung finden sollen. Sie haben also Einfluss auf alle anderen Stakeholder.

So liefert der vom Roboterhersteller und dem automobilen Roboterexperten definierte Befehlsumfang zunächst die Grundlage für die Programmierung und Simulation durch die Offline-Roboterprogrammierung in der Anlagenplanung. Anhand der vorgegebenen Bewegungs- und Applikationsbefehle werden konkrete Abläufe für spezifische Roboteranlagen untersucht und in einem passenden Anwendungsprogramm definiert. Applikationspakete und Anwendungsprogramm in Kombination sind wiederum die Softwarebasis, mit der ein Systemintegrator eine Roboteranlage im Rahmen der Inbetriebnahme konfigurieren und parametrieren kann. Die finale Nutzergruppe eines Roboterprogramms ist der Betreiber. Er hat den Auftrag, die Inbetriebnahme zu unterstützen, die finale Anlage abzunehmen und den anschließenden Produktionsbetrieb zu überwachen und zu optimieren. Dazu gehören beispielsweise minimale Anpassungen der Roboterpositionen im Anwendungsprogramm, um Differenzen zwischen Simulationsmodell und realer Welt zu korrigieren und den Anlagenablauf zu optimieren. Er arbeitet mit dem im vorangegangenen Prozess entstandenen Anwendungsprogramm. Prozessbausteine, Befehle und Reihenfolge sind dort bereits

festgelegt. Seine Aufgabe liegt nicht in der Programmentwicklung, sondern in der Bedienung, Überwachung und der Durchführung von notwendigen Anpassungen und Optimierungen der Anwenderprogramme.

Die vorgestellten Nutzergruppen zeigen, dass diese höchst unterschiedliche Anwendungsfelder haben, jedoch über die Roboterprogramme eine gemeinsame Datenbasis besitzen, welche sich im Laufe des Entwicklungs-, Inbetriebnahme- und Betriebsprozesses weiterentwickelt. Ähnlich verhält es sich auch mit der in dieser Arbeit im Fokus stehenden Herstellerabhängigkeit. Durch die heterogenen Anwendungsfelder ist diese ebenfalls in unterschiedlicher Ausprägung vorhanden, einen gemeinsamen Nenner stellen hingegen das Roboterprogramm und die damit einhergehende Programmiersprache dar.

So müssen Roboter- und Applikationsexperten mit dem durch den Hersteller spezifizierten Sprachenumfang arbeiten und um notwendige unternehmensspezifische Instruktionen erweitern. Darüber hinaus ist die Programmiertätigkeit an die zur Verfügung stehende Entwicklungsumgebung gebunden. Mit dem Umstieg des Robotersystems zu einem anderen Hersteller erfolgt folglich sowohl der Wechsel der Programmiersprache als auch der Entwicklungsumgebung.

Zwar existieren ebenfalls herstellerunabhängige Programmierumgebungen. Diese sind allerdings primär auf die Programmsimulation und nicht die Applikationsentwicklung ausgelegt. Aus diesem Grund hat die Systemproprietarität in der Offline-Programmierung eine geringere Auswirkung als in der Applikationsentwicklung. Tools wie Process Simulate Robotics [77], Delmia Robotics [76], Easy Rob [156] oder RoboDk [114] werden von Dritt-Anbietern angeboten und können somit unabhängig vom verwendeten Roboterlieferanten eingesetzt werden. Diese Flexibilität ist jedoch nur möglich, weil mit der durch die Automobilindustrie vorangetriebenen RRS-Spezifikation ein Standard vorhanden ist, der die übergreifende Simulation von Manipulatoren unterschiedlicher Hersteller ermöglicht. Auf der Ebene der Anwendungsprogrammgenerierung in der Simulationsumgebung zeigen sich bereits wieder Roboterherstellerabhängigkeiten. Da kein allgemeingültiger Programmierstandard existiert, müssen die Softwarehersteller für jede zu unterstützende Roboterprogrammiersprache Übersetzer in Form von Postprozessoren vorhalten. Zudem muss ebenfalls die automobilspezifische Befehlssyntax aufgenommen werden, um Anwendungscode mit dem korrekten Instruktionsumfang generieren zu können. Die dadurch entstehenden Aufwände bzw. Beeinträchtigungen liegen dementsprechend insbesondere in der Befähigung und nicht in der Verwendung der Simulationstools, da diese graphisch und robotersystemunabhängig erfolgt.

Eine stärkere Auswirkung hat die Proprietarität der Systeme und der Programmierung auf die Aufgaben des Systemintegrators. Dieser muss zwar keine Roboterprogramme entwickeln, allerdings müssen die vorhandenen Applikationsbausteine des OEM-Standards und die Anwenderprogramme aus der Simulation an der realen Anlage konfiguriert und in Betrieb genommen werden. Die Arbeit erfolgt zwangsläufig mit dem herstellerspezifischen Programmier- und Bediensystem. Die Aufwände können z. T. reduziert werden, da Installationsroutinen anhand des Unternehmensstandards automatisiert durchgeführt werden und die Integrationsarbeit meist durch externe und spezialisierte Lieferanten und nicht in Eigenleistung erfolgt. Des Weiteren stellt die SW-seitige Inbetriebnahme nur einen Teil der Integrationsarbeit dar, da ebenso eine mechanische und elektrische Installation sowie Inbetriebnahme notwendig ist.

Auf Werksebene zeichnet sich die Abhängigkeit sowohl in der Programmierung als auch Bedienung aus. Zum einen muss der Anwender in der Lage sein, Roboter- und Applikationsbefehle interpretieren zu können, um beispielsweise den Anlagenzustand zu bewerten und Fehler zu identifizieren. Zum anderen muss er mit dem Bedien- und Visualisierungskonzept vertraut sein, um geringe Modifikationen von Roboteraktionen oder Applikationsprozessen rasch durchzuführen. Im Gegensatz zur Inbetriebnahme unterliegen der Betrieb und die Instandhaltung einem stärkeren Zeitdruck, da technische Probleme in der laufenden Produktion rasch mit kostenintensiven Produktionsunterbrechungen einhergehen.

Es zeigt sich, dass die Roboterprogrammiersprache erhebliche Auswirkungen auf unterschiedliche Benutzergruppen, Unternehmensprozesse und Tools besitzt und einer Vielzahl an divergenten Anforderungen, denen sie gerecht werden muss, unterliegt. Betrachtet man neben der reinen Anwenderseite auch noch die Belange der Roboterhersteller in Bezug auf ihre Entwicklungsanforderungen und Differenzierungsbedürfnisse, wird ersichtlich, warum die Festlegung auf einen einheitlichen Standard mitunter langwierig und schwierig ist.

Möchte man die Anforderungen für die Automobilindustrie und dem Karosseriebau festlegen, sollten diese daher insbesondere aus Sicht der Anwender im Werk und der zentralen Entwicklungs- bzw. Applikationsingenieure erfolgen, da deren Arbeit am stärksten durch die Ausprägung der Programmiersprache beeinflusst ist. Abbildung 55 fasst die Tätigkeit der zentralen Entwicklung und Anwender von Roboterprogrammen im Rahmen der Integration von Robotersystemen im Karosseriebau und die daraus abgeleiteten Anforderungen an Roboterprogrammiersprachen zusammen.

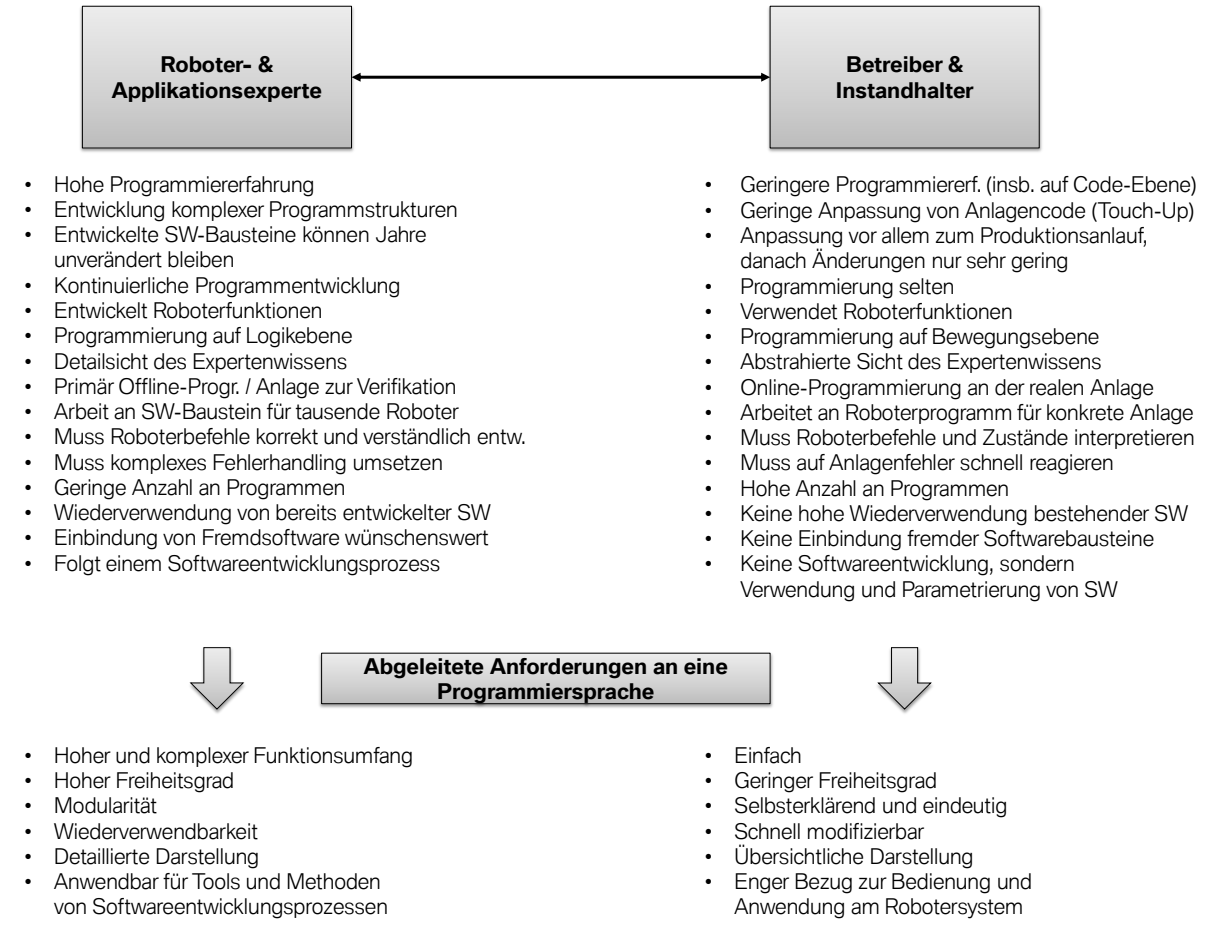


Abbildung 55 – Programmiercharakteristik und abgeleitete Anforderungen für zentrale Anwender im Karosseriebau

Es ist ersichtlich, dass sich der Arbeitsinhalt und damit auch die Anforderungen an eine Roboterprogrammiersprache signifikant unterscheiden. Diese decken sich auch mit den in Kapitel 6.3.2 identifizierten Befehlsausprägungen eines Anwender- bzw. Applikationsprogrammes.

Auf Seite des Roboterexpertens muss die Programmiersprache insbesondere die Umsetzung komplexerer Funktionen ermöglichen, die das spezifische Prozesswissen für den Anwender abstrahieren soll. Die Sprache muss in der Lage sein, dieses Fertigungs-Know-how in Algorithmen abzubilden. Der Roboterexperte agiert im Prinzip als klassischer Softwareentwickler und hat somit ähnliche Anforderungen in Bezug auf Modularität oder Wiederverwendbarkeit von Software. Er profitiert von modernen Konzepten in Softwarestruktur und Entwicklungsprozess und besitzt auch die Expertise diese anzuwenden. Obwohl er in der Domäne der Robotik arbeitet, verwendet er spezifische Roboterbefehle nur in geringem Maße, vielmehr arbeitet er mit Logikinstruktionen und Befehlen zur Ansteuerung von Peripheriekomponenten.

Der Roboterbediener bzw. Instandhalter im Werk hingegen arbeitet anwendungs- und prozessnah. Seine Aufgabe ist es, Roboterbewegungen und Applikationsbefehle richtig zu parametrieren und bzgl. Taktzeit und Prozessgenauigkeit zu optimieren und zu überwachen. Er arbeitet nicht in einer komplexen Logikebene, sondern mit domänenspezifischen Befehlen zur Steuerung des Manipulators und der angeschlossenen Peripherie. Der Freiheitsgrad ist vom Robotersystem mit dem Applikationsstandard vorgegeben und eingeschränkt, um so ein schnelles und zielgerichtetes Arbeiten zu ermöglichen und Programmfehler zu vermeiden. Daher ist die mit der Programmsprache verbundene Bedienung und Visualisierung von hoher Bedeutung, da diese die einfache Modifikation der vorhandenen Programmierbefehle und korrekte Zustandsbeschreibung des Systems ermöglichen muss.

Diese Betrachtung der Stakeholder eines Roboterprogrammes zeigt, dass neben unterschiedlichen Anforderungen an die Programmiersprache auch divergente Bedarfe in Bezug auf die Programmiermethode existieren und ein zu wählendes Programmiersprachenkonzept per se einem Zielkonflikt zwischen den einzelnen Verwendern unterliegt. Um diesen besser bewerten bzw. aufzulösen zu können, soll im Folgenden der Abgleich der Anforderungen mit den zur Verfügung stehenden Programmiersprachen erfolgen.

6.6. Vorschlag eines hybriden Programmiersprachenkonzeptes

In den vorangegangenen Kapiteln wurde beschrieben, welchem Zweck und welcher Verwendung Roboterprogrammiersprachen in der Automobilindustrie dienen und welche Anforderungen sich aus den unterschiedlichen Perspektiven des Karosseriebaus ergeben. Diese sollen im Folgenden noch einmal zusammengefasst und anhand der Ebenen Benutzer, Befehl und System kurz erläutert werden.

Benutzerebene

Die Sprache muss auf Anwenderebene im Werk einfach, verständlich, nachvollziehbar und möglichst anwendungsbezogen sein. Auf Expertenebene muss die Umsetzung komplexer Prozesslogiken und Softwarestrukturen realisierbar sein. Die an der Sprache sich orientierende Bedien- bzw. Programmierumgebung soll für die jeweilige Benutzergruppe ausgelegt sein.

Befehlsebene

Die Sprache muss den grundlegenden Befehlsumfang für Roboterapplikationen im Automobilbereich abdecken. Das bedeutet, es müssen relevante Instruktionen und Datentypen existieren, die sowohl Manipulatorbewegungen als auch Applikationsaktionen funktional beschreiben. Zudem muss ein ausreichendes Grundgerüst an Standardprogrammierbefehlen vorhanden sein.

Systemebene

Die Sprache soll herstellerunabhängig sein. Die Syntax und der Befehlssatz sollen folglich nicht einer herstellerspezifischen Robotersprache entsprechen. Außerdem sollte Interoperabilität auf Entwicklungs-, Planungs- und Ausführungsebene gegeben sein. Dies bedeutet, die Sprache muss in gängigen Entwicklungswerkzeugen unterstützt werden, in Simulationssysteme einles- und verwendbar und außerdem von unterschiedlichen Robotersystemen interpretierbar sein.

Bewertung vorhandener Sprachen

Eine vollumfängliche und detaillierte Bewertung der einzelnen Programmiersprachen ist wie in Kapitel 6.4 besprochen nicht sinnvoll, da die Überschneidungen und Unterschiede paradigmenbedingt ausreichend groß sind, um eine differenzierte Betrachtung auf Sprachgruppenebene vorzunehmen. Diese sollen im Folgenden anhand ihrer Anforderungsabdeckung diskutiert und in Abbildung 56 zusammenfassend gegenübergestellt werden.

Die Gruppe der Roboterprogrammiersprachen kann in herstellerspezifisch und herstellerunabhängig bzw. roboterorientiert und aufgabenorientiert unterteilt werden, wobei aus technischen Gesichtspunkten die aufgabenorientierte Programmierung letztendlich auf roboterorientierte Programmierung aufsetzen muss, um die notwendige Befehlsausführung des Robotersystems durchzuführen. Aus diesem Grund wird die aufgabenorientierte Programmierung im Folgenden nicht weiter betrachtet, sondern nur in der Herstellerabhängigkeit unterschieden.

In Bezug auf die Systemebene hat die herstellerunabhängige, roboterorientierte Programmierung mit IRL dank der universellen Normierung Vorteile im Vergleich zu den herstellerspezifischen Varianten. Die technische Interoperabilität zu anderen Systemen ist grundsätzlich vergleichbar, da der Sprachentypus derselbe ist, jedoch ist die IRL wegen der Normierung einfacher zu handhaben. Ebenso hat der gleiche Sprachentypus zur Folge, dass die Bewertung der Benutzerebene gleichmäßig ausfällt, allerdings wird angesichts der funktionalen

Begrenzung auf die Domäne der Robotik die Eignung für Expertenanwendungen schlechter als die für Werksanwender angesehen. Die Bewertung der Befehlsebene ergibt sich zudem aus der in Kapitel 6.3 durchgeführten Analyse. Die herstellerspezifischen Programmiersprachen besitzen aufgrund der kontinuierlichen Weiterentwicklung durch die Roboterhersteller Vorteile im Vergleich zur IRL. Dies gilt insbesondere für Instruktionen auf Ebene der klassischen Roboterfunktionen. Auf Applikationsebene wird der Bedarf aus Automobilsicht ebenso nur unzureichend abgedeckt und projektspezifische Erweiterungen sind erforderlich. Für die Sprachen KRL und RAPID existieren aber zumindest Implementierungen, die eine Spezifikationsdefinition erleichtern und auf die IRL übertragen werden können.

Die Gruppe der Normsprachen der Automatisierungstechnik stellt mit Structured Text und G-Code zwar grundsätzlich herstellerunabhängige und akzeptierte Standards dar, bildet die Domäne der Robotik überhaupt nicht oder nur zu einem sehr geringen Teil ab. Folglich bieten sie bereits in der Praxis eine hohe Interoperabilität auf Systemebene, jedoch sind applikationsspezifische Umfänge überhaupt nicht enthalten. Aufgrund der Programmierausrichtung für maschinenorientierte Achsbewegungen ist G-Code für die Anwenderseite auf Bedienerebene geeigneter. ST ist mit dem Einsatzzweck der Schalt- und Logikoperationen ein besseres Werkzeug auf Expertenebene.

Für die Bewertung der allgemeinen Programmiersprachen wurde die beschriebene Unterteilung in imperative, deklarative und objektorientierte Sprachen gewählt. Da sämtliche Sprachen keinen Domänenbezug zur Robotik haben, besitzen sie grundsätzlich weder Roboter- noch Applikationsbefehle und müssen diesbezüglich erweitert bzw. bibliotheksseitig implementiert werden. Objektorientierte Sprachen besitzen dank Ihres Konzeptes Vorteile im Verbergen komplexer Applikationslogiken. Sowohl in der imperativen als auch der objektorientierten Programmierung gibt es ausreichend Vorarbeiten, die eine Umsetzbarkeit von grundlegenden Roboteroperationen nachgewiesen haben (vgl. Seite 23 ff.). Außerdem verfolgen klassische Industrierobotersprachen ohnehin ein imperatives Programmierparadigma und sind somit auch gut in imperative GPLs zu transferieren. Deshalb werden diese auf Benutzerebene des Werksanwenders besser als alternative Sprachkonzepte bewertet, wobei die Objektorientierung infolge der Verwandtschaft zur imperativen Programmierung geeigneter ist als deklarative Konzepte, die einen komplett anderen Ansatz verfolgen und eine höhere Komplexität aufweisen.

Auf Expertenebene bieten objektorientierte Sprachen wegen ihrer Abstrahier-, Wiederverwend- und Strukturierbarkeit Vorteile bei der Entwicklung komplexer Programme und Logiken. Darüber hinaus sind GPLs kein herstellerspezifischer Standard und bei der imperativen und objektorientierten Ausprägung mit C, C++ oder Java sehr stark verbreitet, bieten aber keine Normierung für Roboteranweisungen und unterliegen häufig projektspezifischen Implementierungen. Dies birgt das Risiko, dass sich Parameter, Namensgebung, Datentypen und ähnliches unterscheiden und eine übergreifende Verwendung wieder eingeschränkt ist. Die weite Verbreitung und Anwendung der Sprachen haben auch Vorteile in Bezug auf die Interoperabilität, da die breite Unterstützung in Entwicklungsumgebungen und zahlreichen schon vorhandenen Softwarebibliotheken die Effizienz und Anwendbarkeit in der Entwicklung erhöhen.

		Programmiersprachentyp						
		Roboter-programmierung		Automatisie-rungstechnik		Allgemeine Programmiersprachen		
		Proprietär (KRL, RAPID, PDL)	Offen (IRL)	Ablauf-steuerungen (SPS - ST)	Werkzeug-maschinen (NC - G-Code)	Imperativ (Pascal)	Objektorientiert (C++, Java)	Deklarativ (Prolog, Lisp)
Bewertungsebene	**Befehlseben**	3	1	0	0	-1	1	-2
	Roboterbefehle	++	+	o	+	-	-	--
	Applikationsbefehle	+	o	o	-	o	++	o
	Benutzerebene	3	3	1	0	2	2	-1
	Werksanwender	++	++	o	+	+	o	--
	Roboterexperte	+	+	+	-	+	++	+
	Systemebene	-3	2	2	4	3	3	2
	Herstellerabhängigkeit	--	++	+	++	+	+	+
	Interoperabilität	-	o	+	++	++	++	+
Summe		**3**	**6**	**3**	**4**	**4**	**6**	**-1**

Abbildung 56 – Bewertung der Eignung unterschiedlicher Programmiersprachenarten

Schlussfolgerung und Lösungsvorschlag

In Abbildung 56 zeigt sich, dass keines der vorgestellten Programmiersprachenkonzepte allumfassend die definierten Anforderungen erfüllt. Jede Alternative besitzt individuelle Vor- und Nachteile. Insbesondere die Ansprüche, die sich aufgrund des unterschiedlichen Benutzerkontexts von Experte und Werksanwender ergeben, sind durch einzelne Programmiersprachen nicht übergreifend abgedeckt.

Daher wird in dieser Arbeit ein hybrides Programmiersprachenkonzept vorgeschlagen, das durch den kombinierten Einsatz einer roboterorientierten und einer allgemeinen Programmiersprache, sowohl die domänenspezifischen Aspekte auf Anwenderebene als auch die Freiheitsgrade und Gestaltungsmöglichkeiten einer GPL verbindet. Dadurch wird den unterschiedlichen Anforderungsbedürfnissen Rechnung getragen. Durch die Verwendung von IRL und C++ als Sprachbasis werden die Optionen mit der bestmöglichen Eignung auf Systemebene gewählt und Herstellerunabhängigkeit und Interoperabilität gewährleistet.

Beiden Sprache besitzen jedoch sowohl auf Roboter- als auch Applikationsbefehlsebene Defizite im Karosseriebaueinsatz. Deshalb wird der in Kapitel 6.3.2 identifizierte Funktionsumfang als automobilspezifische Normerweiterung der IRL vorgeschlagen, um dem notwendigen Funktionsumfang für Automobilanwendungen gerecht zu werden. Die entwickelte Spezifikation trägt dementsprechend den Namen:

AIRL - AUTOMOTIVE INDUSTRIAL ROBOT LANGUAGE

6.7. AIRL – Automotive Industrial Robot Language

Kerngedanke bei der AIRL-Konzeption war, sich anhand der bestehenden IRL-Norm an Funktionalität und Programmiersprachenaufbau zu orientieren und diese um die identifizierte notwendige automobilspezifische Befehlsmenge zu erweitern (Abbildung 57). Dadurch soll eine Spezifikation festgelegt werden, die möglichst schmal und anwendungsbezogen ist und die relevanten domänenspezifischen Aktionen des automobilen Karosseriebaus abdeckt. Durch Verwendung von IRL als Ausgangsbasis kann die vollkommene Neudefinition von Syntax und Semantik vermieden werden und sich auf die relevanten Funktionen, Parameter und Datentypen fokussiert werden.

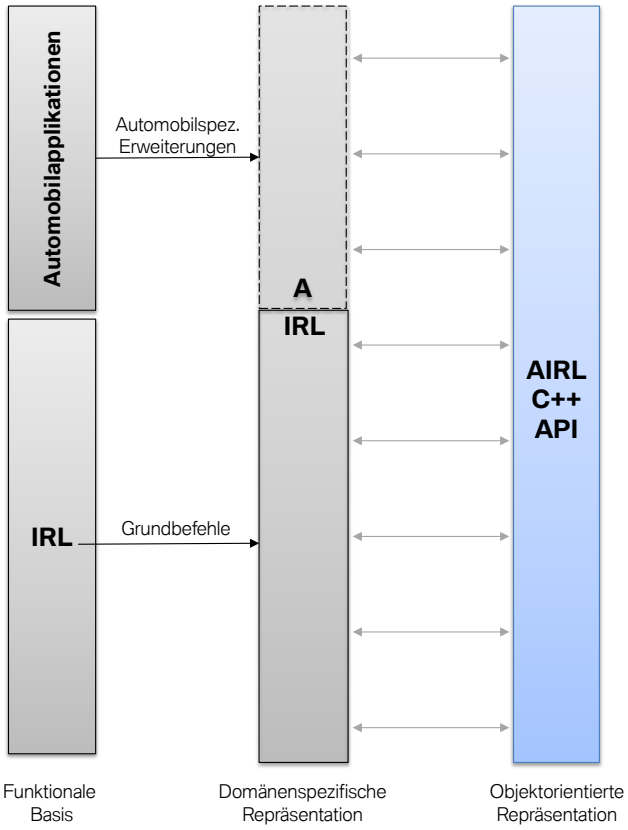


Abbildung 57 – Zusammensetzung der AIRL

Neben der Definition von AIRL als domänenspezifische Sprache ist die Spezifikation eines C++/AIRL-Interface ein weiteres Designprinzip, welches die Befehl- und Datentypsemantik der AIRL-DSL abbildet und so einen ansatzlosen Übergang zwischen prozeduraler, domänenspezifischer Sprache und objektorientierter Programmierung ermöglicht.

Zum einen kann dadurch die anwendungsbezogene und eingeschränkte Sicht einer DZL erhalten werden. Dies ist notwendig, um auf Werksseite alle erforderlichen Befehls- und Parametermodifikationen zu ermöglichen und die Anwendungskomplexität nicht zusätzlich zu erhöhen. Dank der stabilen und anwendungsbezogenen Spezifikationen bleibt die Interoperabilität zu weiteren Tools (z. B. Simulationssoftware) erhalten. Passende Bedien- und Visualisierungskonzepte an der Anlage können ebenfalls auf diesen Befehlssatz und der zugehörigen Funktionalität ausgelegt werden. Zum anderen bietet der objektorientierte Teil die Möglichkeit, sämtliche Vorteile einer GPL und die damit einhergehenden Methoden, Prozesse, Tools, Wiederverwendung, Abstraktion, etc. auf Expertenebene zu nutzen. Damit wird gewährleistet, komplexe Logiken bestmöglich implementieren zu können, ohne negative Auswirkungen auf andere Benutzergruppen zu haben. Durch die Verwendung von C++ bleibt die Plattforminteroperabilität ebenfalls erhalten und die Umsetzbarkeit in unterschiedlichen Steuerungssystemen ist gewährleistet.

Zum besseren Verständnis werden in den Abbildungen 107 bis 112 in Anhang 9.4.2 die Ausführung von klassischen Punkt-zu-Punkt- und Linear-Roboterbewegungen sowie eine BMW-spezifischen Punktschweißoperation in den Programmiersprachen Kuka KRL und ABB RAPID aufgezeigt und die analoge Ausführung in der AIRL-DZL und dem C++/AIRL-Interface dargestellt und erläutert. In Anhang 9.4.3 ist zudem der komplette Sprachumfang mit Instruktionen und Datentypen der AIRL-DZL im Vergleich zur KRL, RAPID und IRL aufgeführt.

Durch die vorgeschlagene Aufteilung der Programmierebenen auf Basis einer stabilen AIRL-Spezifikation können anwendergerechte Programmiermethoden verwendet und entsprechende Werkzeuge entwickelt werden, die als Transfermodell auf der AIRL-Spezifikation beruhen. Dies bietet den Vorteil, dass das für den jeweiligen Nutzerkreis ermittelte Anforderungsbild bestmöglich umsetzbar ist, sich aber anhand der AIRL-Basis an automobilen Programmieranforderungen für Roboterapplikationen orientiert.

Abbildung 58 fasst die zentralen Elemente und deren Eigenschaften für die AIRL-Spezifikation zusammen.

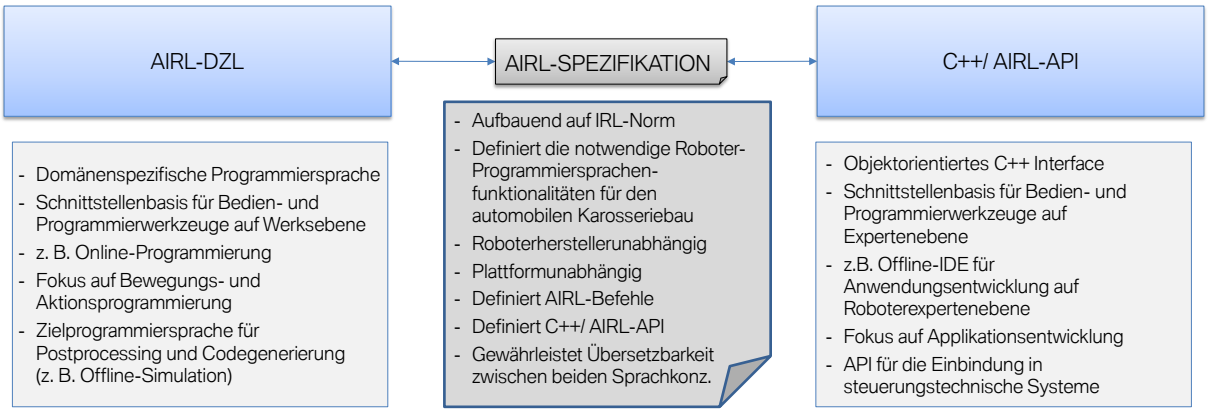


Abbildung 58 – AIRL-Spezifikation und zugehörige AIRL-DZL und C++/AIRL-API

Aufgrund der Verwendung einer herstellerunabhängigen, domänenspezifischen Programmiersprache ist damit die Interoperabilität und notwendige Funktionalität implizit vorgegeben. Für eine erfolgreiche Umsetzung einer herstellerübergreifenden Integration von Robotersystemen ist jedoch auch die Ableitung der daraus resultierenden Anforderungen für die Bedien- und Entwicklungsebene notwendig. Für den Anwendungsfall des Karosseriebaus bedeutete dies, dass die Programmier- und Bedienumgebung sowohl auf Werks- als auch auf Expertenebene die AIRL als gemeinsame Programmiergrundlage nutzen und zeitgleich das Ziel der Herstellerunabhängigkeit für den jeweiligen Nutzerkreis bestmöglich umsetzen müssen. Dies wird im Rahmen der prototypischen Umsetzung der Programmier- und Bedienumgebung aufgezeigt und evaluiert.

6.8. Zusammenfassung

Im vorangegangenen Teil der Arbeit wurden zunächst die grundlegenden Programmiermethoden und -sprachen der Softwareentwicklung und Robotik vorgestellt, für den Karosseriebau konkretisiert und begründet, weshalb die textuelle Programmierung aktuell und auch zukünftig die höchste Bedeutung in der industriellen Roboterprogrammentwicklung besitzt.

Im Anschluss wurde ein Vorgehen gewählt, das anhand der heute gegebenen Funktionalität von Roboterprogrammiersprachen und den z. T. abweichenden Anforderungen von Kundenseite ein universelles Anforderungsset für die Programmierung von Karosseriebauanwendungen formuliert. Hierfür erfolgte der Aufbau einer universellen Befehlsdatenbank, welche mit den Befehlssätzen der gängigen Roboterhersteller und der herstellerneutralen Programmiernorm IRL befüllt wurde. Auf diese Weise konnte eine herstellerübergreifende und vollumfassende Vergleichsbasis für industrielle Programmiersprachen erschaffen werden, die im Anschluss mit der kundenspezifischen Anforderungsseite am Beispiel der BMW AG anhand von realen Roboterlinien abgeglichen und ergänzt wurde.

Zum einen zeigte sich, dass sich die proprietären Programmiersprachen grundsätzlich ähneln und gleiche Funktionsumfänge, allerdings mit unterschiedlichen Schwerpunkten, abbilden. Zum anderen stellte sich heraus, dass diese gängigen Industrierobotersprachen trotzdem nur einen Teil der kundenseitigen funktionalen Anforderung abdecken und daher anwenderseitig Erweiterungen spezifischer Operationen notwendig sind, welche je nach Programmiersprache bis zu 50 % der verwendeten Befehlsmenge betragen können. Zudem konnte nachgewiesen werden, dass sich die verwendeten Befehlstypen und damit auch die Programmiercharakteristik zwischen Anwender- und Applikationsprogramm stark unterscheiden.

Aus diesem Grund wurde als herstellerunabhängige Lösung keine einzelne Programmiersprache, sondern ein hybrides Programmierkonzept vorgeschlagen, das aus einer domänenspezifischen und einer allgemeinen Programmiersprache besteht und sowohl den Anforderungen der Benutzerebene auf Werks- als auch auf Expertenebene gerecht wird. Kern des Konzeptes stellt die Automotive Industrial Robot Language (AIRL) dar, welche anhand der beschriebenen Vorarbeiten der Befehlsdatenbank und Programmanalyse abgeleitet und in Form einer IRL-Erweiterung entworfen wurde. Die Verknüpfung mit einer C++/AIRL-API ermöglicht den Übergang zwischen den beiden Programmiersprachenkonzepten. Auf diese Weise können sowohl die Vorzüge einer einfachen und anwendungsbezogenen DZL als auch die Vorteile einer objektorientierten allgemeinen Hochsprache genutzt werden, um komplexe Applikationsprogramme mit klassischen Softwareentwicklungsmethoden erstellen zu können.

Dieses Konzept bietet eine geeignete technische Grundlage, um für unterschiedliche Nutzergruppen passende Werkzeuge zur Simulation, Programmierung und Bedienung am Robotersystem selbst oder für den PC-Arbeitsplatz zu entwickeln. Durch die Verwendung der herstellerneutralen AIRL-Spezifikation werden einerseits die System- und Programmiersprachenineroperabilität sichergestellt. Andererseits ermöglicht es die Bereitstellung der notwendigen Funktionen, Parameter und Datentypen für die Entwicklung von Softwareapplikationen des automobilen Karosseriebaus.

Die sich daraus ergebenden Vorteile werden im folgenden Kapital im Rahmen der praktischen Umsetzung aufgezeigt und evaluiert.

7. Prototypische Umsetzung und Evaluierung

In der bisherigen Arbeit wurde in einer Kombination aus theoretischer und technischer Analysearbeit die herstellerunabhängige Roboterintegration im automobilen Karosseriebau bearbeitet. Anforderungen und Lösungskonzepte hinsichtlich Roboterschnittstellen bzw. Programmierkonzepten konnten aufgezeigt und diskutiert werden. Fragestellungen wie die Leistungsfähigkeit der Roboterschnittstellen in Bezug auf Bahngenauigkeit und -synchronität im Verbund mit externen Steuerungssystemen wurden durch Lasertrackermessungen evaluiert und die Möglichkeit der steuerungstechnischen Abstraktion der isolierten Robotersysteme nachgewiesen. Ein Abstraktionsvorschlag für die Programmierung von Karosseriebau-applikationen wurde anhand der Analyse von klassischen Roboterprogrammiersprachen mehrerer Hersteller und realer Roboterprogrammen in Automobilwerken entwickelt. Als Ergebnis wurde der AIRL-Spezifikationsentwurf in Form einer DZL und dem zugehörigen C++/AIRL-API-Interface definiert. Dieser stellt die Grundlage dar, um herstellerunabhängige Programmierung sowohl auf Instandhalter- als auch Expertenebene anhand einer definierten Instruktionsvorgabe interoperabel zu realisieren.

Sowohl die definierten Forschungsfragen der Herstellerunabhängigkeit in Bezug auf Steuerungstechnik als auch die Programmierung wurden somit jeweils beantwortet und zeigen auf, wie die herstellerunabhängige Integration von Robotersystemen in diesen Aspekten aussehen kann.

Insbesondere aus industrieller Forschungssicht spielt jedoch die praktische Umsetzung ebenso eine wichtige Rolle und findet sich somit auch als zentrale Forschungsfrage und abgeleitete Kernanforderungen in Kapitel 3 wieder. Im Folgenden soll daher die Überführung der zentralen Ergebnisse der vorherigen Arbeitspakete anhand der prototypischen Realisierung von automobilen Karosseriebauapplikationen an

- der herstellerunabhängigen Programmierung auf Instandhalterebene
- der herstellerunabhängigen Programmierung auf Expertenebene
- der gesamthaften Umsetzung einer herstellunabhängigen Roboterzelle

aufgezeigt, evaluiert und deren Machbarkeit nachgewiesen werden.

Um das vorgeschlagene hybride Programmierkonzept in Bezug auf praktische Einsetzbarkeit und technische Machbarkeit zu evaluieren, wurde für die unterschiedlichen Anwendergruppen jeweils ein offenes und herstellerneutrales Werkzeug zur Bedienung bzw. Programmierung von Roboteranwendungen implementiert. Für die Werks- und Bedienebene ist dies der RobotCell Commander, welcher anhand der AIRL-Funktionsspezifikation die Programmierung unterschiedlicher Robotersysteme mit graphischer Darstellung und Simulation vereinfacht und eine schnelle und fehlerfreiere Programmanpassung an der Anlage vor Ort ermöglicht. Auf Ebene der Applikations- und Roboterexperten wurde mit einer AIRL-Entwicklungsumgebung auf Eclipse-Basis ein Tool geschaffen, das den Anforderungen an moderne Software-entwicklungswerkzeuge gerecht wird und sowohl die Arbeit mit einer herstellerunabhängigen domänenspezifischen Programmiersprache als auch einer allgemeinen Programmiersprache unterstützt sowie die Entwicklung von AIRL- und C++-Programmen verbindet.

7.1. Herstellerunabhängige Programmierung auf Instandhaltungsebene

Für die herstellerunabhängige und anwendergerechte Programmierungen von Roboteranwendungen auf Werksebene wurde das Tool RobotCell Commander (RCC) entworfen. Es bildet Funktionalitäten von Roboter-Bedienpanels (Abbildung 59) ab. Diese sogenannten Teach Pendants werden insbesondere dazu eingesetzt, Roboter manuell zu steuern, Aktorik zu schalten und kleine Programmanpassungen durchzuführen.

Abbildung 59 – Teach Pendants der Roboterhersteller Kuka [157], ABB [158], Fanuc [159], Comau [160]

Die in Kapitel 6.3.1 identifizierten Unterschiede in den Roboterprogrammiersprachen und die damit verbundenen Nachteile lassen sich ebenso am Teach Pendant wiederfinden, da Programmier- und Bedienkonzept auf herstellerspezifischen Programmiersprachen beruhen. Neben den herstellerübergreifenden Unterschieden, sind auch die mangelnde Orientierung an den Anforderungen des Karosseriebaus und steuerungsgenerationsübergreifenden Mehraufwendungen als Problemfeld zu nennen. Aus Sicht des automobilen Endanwenders weisen Teach Pendants dementsprechend heute folgende Nachteile auf:

- Teach Pendants sind i. d. R. lediglich für eine Robotergeneration eines Herstellers ausgelegt. Eine Interoperabilität zwischen verschiedenen Herstellern ist nicht gegeben.
- Die Bedienkonzepte sind trotz gleicher Grundfunktionalität unterschiedlich, z. B. sind die Anordnung der Tasten und die Menüstruktur verschieden. Insbesondere bei Bedieneingriffen unter Zeitdruck kann es zu Fehlern und unnötigen Verzögerungen bei nicht ausreichend geschulten Mitarbeitern kommen.
- Die Programmierung mit einem Teach Pendant basiert auf den jeweiligen herstellerspezifischen Programmiersprachen. Die identifizierten Unterschiede wirken sich auch auf die Programmierung und Bedienung am Robotersystem selbst aus.
- Bei einigen Roboterherstellern wird die Teach-In und die textuelle Programmierung am Teach Pendant zwar mit Visualisierung unterstützt. Diese ist allerdings nur auf die Roboterbewegungen und nicht auf die Peripherieoperation oder Anlagenstruktur ausgerichtet.
- Für jeden Wechsel einer Robotergeneration oder eines -lieferanten sind in Anbetracht der genannten Unterschiede Fortbildungen für Mitarbeiter notwendig.
- Die Bedienkonzepte sind auf generische Roboteranwendungen ausgelegt und nicht auf spezifische Applikationen des Karosseriebaus.
- Die genannten Nachteile potenzieren sich bei der Anwendung einer Multilieferanten-Strategie, da mehrere Roboter mit unterschiedlichen Bedienkonzepten innerhalb eines Werkes eingesetzt werden können. Teilweise hat dies in der Praxis zur Folge, dass Mitarbeiter Bedieneingriffe zur Optimierung aus Angst vor Fehlbedienung scheuen.

7.1.1. Anforderungen

Die Vermeidung der genannten Nachteile durch Herstellerneutralität war eine zentrale Kernanforderung bei der Konzeption und Umsetzung des RobotCell Commanders [162]. Des Weiteren sollte das Bedienkonzept nicht vollkommen radikal sein, um die Integration in bestehende Strukturen zu vereinfachen. Ein aufgabenorientiertes Programmiersprachenkonzept wurde, wie im vorangegangenen Kapitel dargestellt, bewusst vermieden, da aufgrund der langen Produktlebenszyklen und der statischen Produktionsstruktur komplexe Anpassungen von Programmabläufen nur sehr selten durchgeführt werden müssen und Änderungen sich vielmehr auf die Modifikation von Parametern oder kleineren Programmanpassungen beschränken.

Der RCC ist roboterapplikationsorientiert konzeptioniert und orientiert sich an den definierten AIRL-Anweisungen, welche die Grundlage für die anwendungsgerechte Auslegung des Tools und die Fokussierung auf die relevanten automobilspezifischen Programmieranweisungen und Parameter sicherstellen. Diese erfolgt anhand eines Basisanweisungskonzeptes, indem Anweisungen für Roboteroperationen und Applikationsbefehle in der Programmübersicht durch reduzierte Anweisungsbausteine wiedergegeben werden, deren Konfiguration operations- bzw. applikationsspezifisch in weiteren Menümasken und durch graphische Unterstützung und Simulation erfolgen kann. Auf diese Weise soll die Komplexität in der übergreifenden Programmübersicht reduziert und auf spezifische Detailmenüs verteilt werden. Jeder Anweisungsbaustein repräsentiert einen AIRL-Befehlstyp und die dahinter stehenden Konfigurationsparameter.

Zusammenfassend wurden folgende Anforderungen im RobotCell Comander umgesetzt:

- Herstellerunabhängigkeit
    - Einsatz offener Technologien
    - AIRL als textuelle Programmiergrundlage
    - Unterstützung Roboter unterschiedlicher Hersteller
- Verbergen der Komplexität durch Anweisungskonzepte auf Basis der DZL AIRL
    - Zentrale Elemente sind Roboter- und Applikationsbefehle der AIRL
    - Detaillierung der Befehle in befehlsspezifischen Untermasken
    - Fokussierte Parametrierung und Konfiguration der Karosseriebaufunktionen
    - Zugriff bis auf I/O-Signalebene möglich
    - Komplexe Applikationsprogramme sind für Anwender nicht sichtbar
- Kombination unterschiedlicher Programmierkonzepte
    - Umschaltbar zwischen textueller AIRL-Programmierung und Basisanweisungen
    - Basisanweisungen konfigurierbar durch Untermenüs
    - Visualisierung der Bedienoperationen durch graphische Simulation

7.1.2. Umsetzung

Abbildung 60 zeigt die zentrale Programmansicht des RobotCell Commanders. Jede Zeile stellt eine Roboterbewegung oder Applikationsaktion dar. In der Übersicht sind die Auswahlparameter auf die notwendigsten Elemente beschränkt: Robotername (1), Applikationstyp (2), betreffende Peripheriekomponente (3) und abhängig von der Aktion weitere zentrale Informationen (4). Durch Auswahl eines Elements kann in die weitere Detailierung eingestiegen werden. Zudem ermöglicht die Betätigung des AIRL-Bedienelements (5) den Ansichtswechsel zur rein textuellen Umgebung mit der AIRL-DZL.

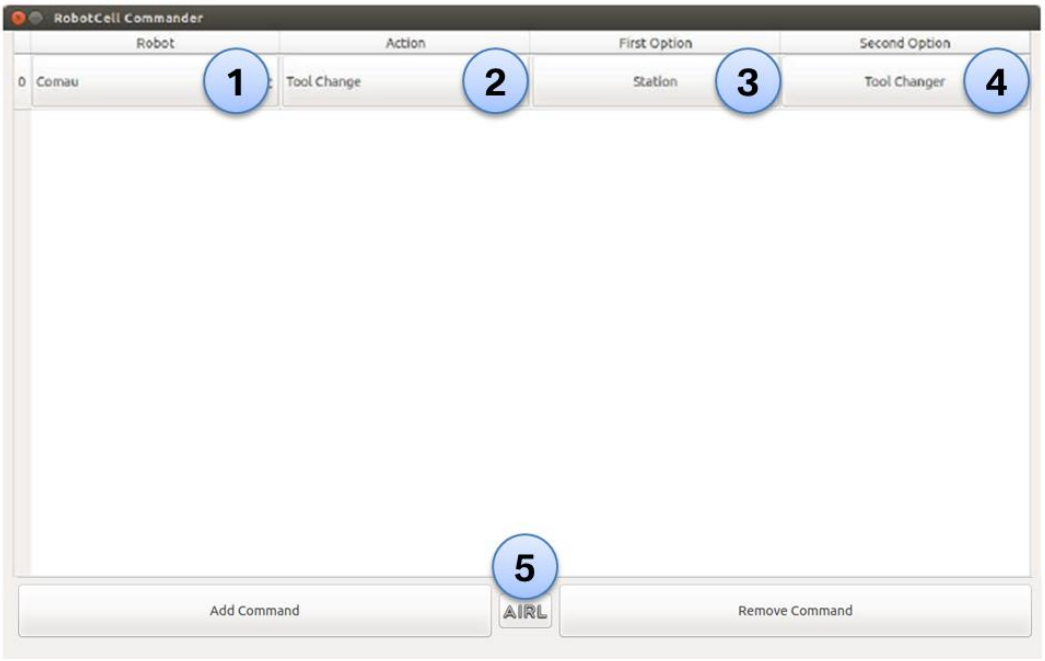


Abbildung 60 – RobotCell Commander Programmübersicht

Die Detailebene (Abbildung 61) ermöglicht das Einstellen von Befehlsparameter einzelner Aktionselemente, z. B. können bei Bewegungsbefehlen einzelne Punkte über Achspositionen (1), kartesische Positionen (2) und Bewegungsparamater wie Bewegungstyp (3) oder Überschleifwerte (4) angegeben werden. Zur visuellen Unterstützung wird ein digitales 3D-Modell analog einer Offline-Simulationsumgebung der gesamten Zelle angezeigt. Dabei werden beispielsweise Zielpositionen (5) und Bewegungseigenschaften, wie Überschleifradien (6) graphisch dargestellt, um den Anwender bei der Bewegungsprogrammierung zu unterstützen.

Bei Applikationsaktionen wird der Benutzer ebenfalls über ein Detailauswahlfeld unterstützt. Abbildung 62 zeigt dies am Beispiel einer Schweiß- und Werkzeugwechsleraktion. Die Einstellungsparameter richten sich an den definierten Parametersatz der AIRL-Spezifikation. Beispielsweise umfasst das bei einem Schweißbefehl die Einstellungen ID und Name des Schweißpunktes bzw. der Schweißzange (2) und das Zangenöffnungsmaß nach der Schweißung (3). In der Visualisierung wird das betreffende Werkzeug angezeigt. Außerdem wird die kinematische Veränderung gemäß den gewählten Parametern dargestellt, z. B. werden beim Öffnen und Schließen des Werkzeugwechslers die Stifte herein bzw. herausgefahren (4). Darüber hinaus kann ausgewählt werden, ob die Aktion an der realen Anlage oder nur in der Visualisierung erfolgen soll. Ein weiteres Bedienelement ermöglicht den Übergang zur I/O-Ebene (5).

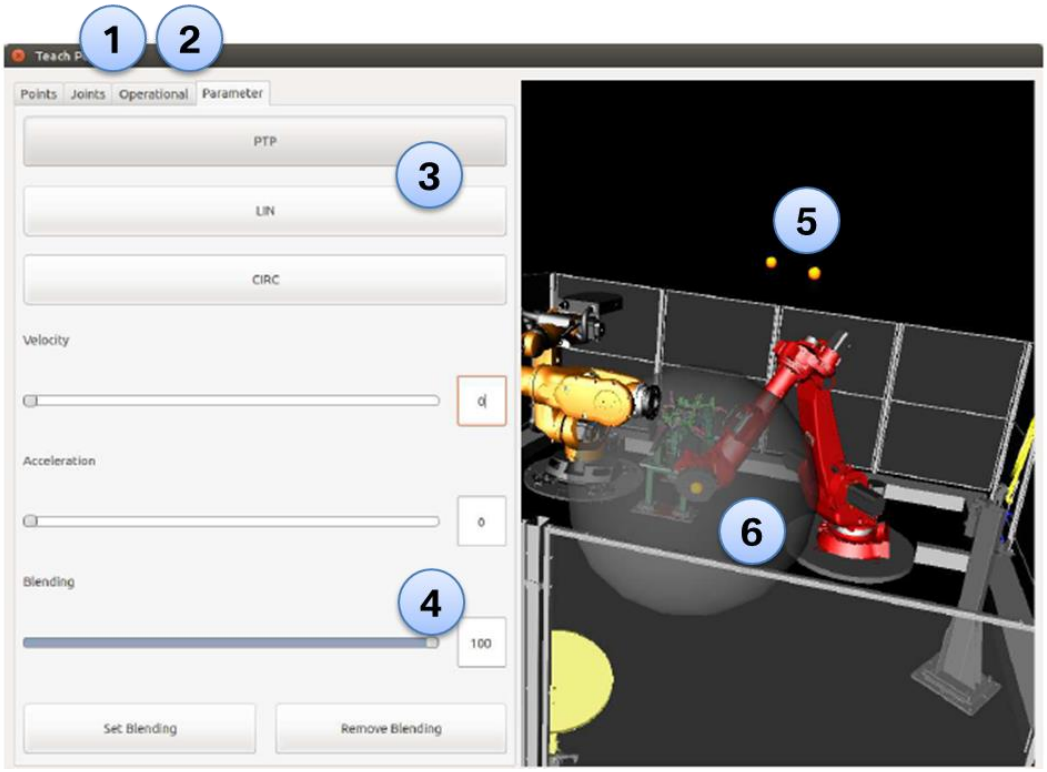


Abbildung 61 – Parametrierung einer Bewegungsoperation

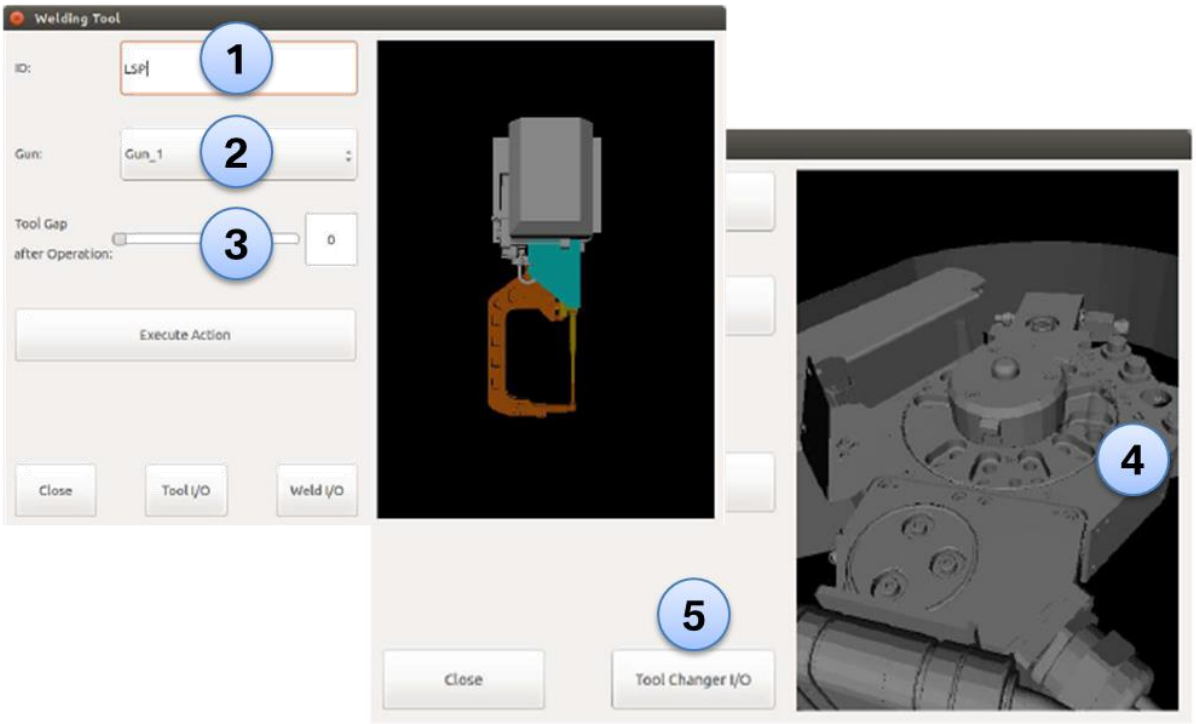


Abbildung 62 – Ansicht für Applikationseinstellungen am Beispiel Schweißen und Werkzeugwechseln

Auf I/O-Ebene erhält der Bediener einen detaillierten Einblick bzgl. der Kommunikationszustände auf Bit-Tiefe (Abbildung 63). Er sieht sowohl den Zustand der Aus-(1) als auch der Eingänge (2) für die jeweiligen Komponenten. Zusätzlich wird neben der jeweiligen Ein- und Ausgangsnummer über ein Kommentarfeld die Bedeutung (3) des Kanals erklärt. Sicherheitsrelevante Kanäle werden in der Farbe Gelb dargestellt (5). Über Schieberegler (4) kann Einfluss auf die Ausgänge des Robotersystems genommen werden. Die durch ein Umschalten des Kanalzustands erfolgte Systemänderung wird dem digitalen Modell zurückgespielt und visualisiert.

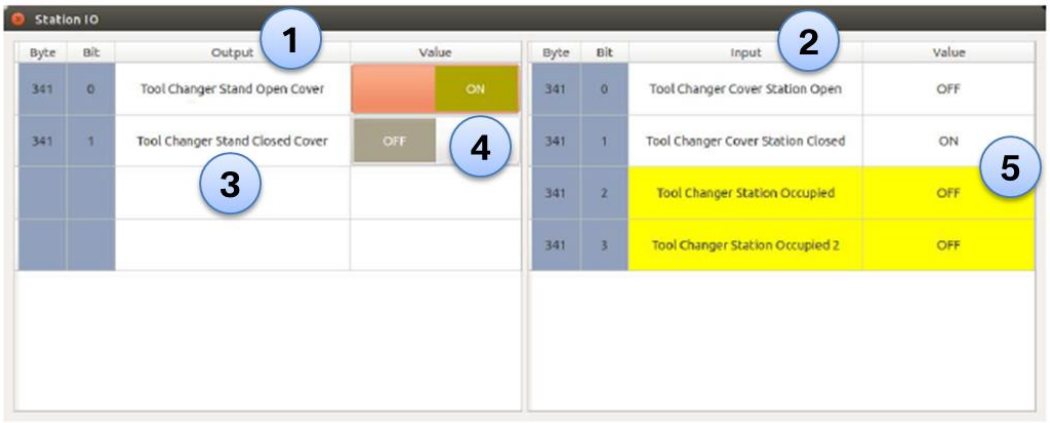


Abbildung 63 – Einstellung der I/O Parameter für eine Punktschweißoperation

7.1.3. Evaluierung

Ein zentrales Ziel bei der Entwicklung des Bedienkonzeptes und der prototypischen Umsetzung im RobotCell Commander war die Beseitigung der genannten Nachteile von gegenwärtigen proprietären Teach-Pendant Konzepten. Hierfür wurde durch die Verwendung von AIRL als Programmiersprachenbasis die konzeptionelle Grundlage geschaffen, den Programmablauf mehrerer Robotersysteme unterschiedlicher Hersteller gleichermaßen zu definieren. Im Rahmen der Implementierung wurde ein Standard-PC-System sowie offene Technologien (Linux-Betriebssystem, Qt-Framework[19]) verwendet und auf diese Weise ebenfalls ein plattformunabhängiger und nicht proprietärer Ansatz verfolgt. Sowohl Bedienkonzept als auch Umsetzung sind somit per se herstellerunabhängig und erfüllen das formulierte Anforderungsset, mehrere Roboter unterschiedlicher Hersteller auf Basis eines einheitlichen Bedien- und Befehlssets zu programmieren. Die Funktionsdefinition der AIRL stellt ebenso die Basis dar, um die geforderte Fokussierung auf Befehle und Parameter des Karosseriebaus zu gewährleisten und Applikationskomplexität zu verbergen. Durch angepasste Eingabemasken können die relevanten Veränderungen leicht eingegeben und modifiziert werden. Die dahinterliegende Programmstruktur und Befehle müssen nicht bearbeitet werden. Zudem wird der Nutzer durch die graphische Simulation unterstützt und erhält direktes optisches Feedback zu den vorgenommenen Programmänderungen.

Die sich daraus ergebenden Vorteile hinsichtlich Bediendauer und -verständlichkeit wurden im Rahmen einer Probandenstudie mit acht Teilnehmern untersucht, wobei vier der Teilnehmer keine bzw. nur sehr geringe Roboterprogrammiererfahrung besaßen und vier Teilnehmer erfahrene Anwender von Kuka-Robotersystemen waren. Die Aufgabenstellung bestand darin, innerhalb einer vorgegebenen Zeit Bedien- und Programmieroperationen am RCC, einem Kuka- und einem Comau-Roboter durchzuführen. Die Testpersonen sollten die notwendigen

[19] https://www.qt.io

Anpassungen zunächst ohne jegliche Einweisung durchführen (Versuch A). Anschließend wurde der Versuch mit der Erfahrung der ersten Versuchsreihe und einer Erklärung der jeweiligen Funktion erneut durchgeführt (Versuch B). Ein Überschreiten der Zeitgrenze innerhalb einer Aufgabe wurde als Fehlversuch gezählt und mit einer Bedienzeit von drei Minuten gewertet.

Die aggregierten Ergebnisse sind in Tabelle 11 dargestellt, die detaillierten Zeiten der einzelnen Teilnehmer in Anhang 9.5 aufgeführt. Wie zu erwarten, zeigte sich, dass die unerfahrenen Nutzer im ersten Versuch schlechter abschnitten als die erfahrenen Roboterprogrammierer. Sowohl die Anzahl an Fehlversuchen (1,58; 0,25), als auch die grundsätzliche Bedienzeit (1 m 46 s; 0 m 50 s) lag höher, wobei diese Differenz im zweiten Versuch erheblich reduziert wurde (25 s; 17 s). Bei fokussierter Betrachtung wird ersichtlich, dass die unerfahrenen Nutzer bei den unbekannten Systemen des Comau-Roboters und des RCC-Prototyps im ersten Versuch eine 30-prozentige (RCC) bzw. 50-prozentige (Comau) höhere Bedienzeit als die Roboterprogrammierer benötigten. Noch größer fiel die Differenz bei dem für die erfahrenen Anwender vertrauten Kuka-System aus, da diese mit 22 s im ersten Versuch um ein Vielfaches schneller waren als die unerfahrenen Nutzer (2 m 15 s). Auch in der isolierten Betrachtung der unerfahrenen Nutzer zeigte sich, dass die Bedienzeiten für RCC mit 38 s (Versuch A) und 14 s (Versuch B) geringer ausfielen als die Bedienzeiten mit den herkömmlichen Robotersystemen von Kuka (A: 2 m 15 s; B: 28 s) und Comau (A: 2m 26 s; B: 34 s). In der Expertengruppe konnte in Versuch A zwar weiterhin mit dem Kuka-System die schnellsten Bedienzeiten erreicht werden, jedoch war auch in diesem Fall, sobald erste Erfahrungen gesammelt wurden, der RCC in Versuch B das schnellste Bedienwerkzeug.

Tabelle 11 – Fehlversuche und Bedienzeiten für Programmoperationen im Vergleich

Aufgabenstellung			Fehlversuche (N) und Bedienzeiten für Versuch A und B in Minuten:Sekunden								
			Mittelwert			unerfahrene Nutzer			erfahrene Nutzer (Kuka)		
Bereich	Test	Panel	N	A	B	N	A	B	N	A	B
Roboter-operation	Punkte teachen	Comau	1	01:25	00:21	1	01:42	00:26	0	01:08	00:17
		Kuka	0	00:50	00:10	0	01:27	00:11	0	00:13	00:10
		RCC	0	00:31	00:05	0	00:37	00:06	0	00:26	00:05
	Achsbewegung	Comau	3	02:04	00:41	3	02:49	00:49	0	01:19	00:33
		Kuka	3	01:38	00:44	3	02:37	00:58	0	00:38	00:30
		RCC	0	00:28	00:20	0	00:28	00:20	0	00:29	00:19
	Überschleifradius einstellen	Comau	6	02:47	00:30	4	03:00	00:32	2	02:34	00:28
		Kuka	4	01:44	00:28	4	03:00	00:33	0	00:29	00:23
		RCC	0	00:50	00:21	0	00:53	00:21	0	00:46	00:22
Applikations-operation	Werkzeugwechsler öffnen/schließen	Comau	3	01:51	00:20	2	02:14	00:28	1	01:29	00:13
		Kuka	2	01:04	00:08	2	01:57	00:11	0	00:11	00:05
		RCC	0	00:25	00:06	0	00:33	00:06	0	00:16	00:06
Mittelwert aller Tests		Comau	3,25	02:02	00:28	2,5	02:26	00:34	0,75	01:37	00:23
		Kuka	2,25	01:19	00:22	2,25	02:15	00:28	0	00:22	00:17
		RCC	0,47	00:41	00:14	0,00	00:38	00:13	0,00	00:29	00:13
		Gesamt	1,99	01:21	00:21	1,58	01:46	00:25	0,25	00:50	00:17

Zusammenfassend lässt sich festhalten, dass mit dem reduzierten und fokussierten Bedienspektrum des RCC die Anzahl der Fehlversuche und die Bedienzeiten bei der Erstbedienung durch unerfahrene Benutzer im Vergleich zu den nativen Herstellerbedienpanels reduziert werden konnten. Ebenso war die Bediendauer für die gegebene Aufgabenstellung bei den unerfahrenen Benutzern mit dem RCC durchwegs geringer als mit den nativen Robotersystemen. Bei den erfahrenen Anwendern zeigten sich diese Vorteile des RCC im Vergleich zum nicht bekannten Comau-System gleichermaßen, während die verkürzte Bedienzeit des RCC in der Gegenüberstellung mit dem bekannten Kuka-Robotersystem erst im zweiten Versuch ersichtlich wurde.

7.2. Herstellerunabhängige Programmierung auf Expertenebene

Teach Pendants bzw. der RCC sind für das schnelle Ändern und Bedienen von Robotersystemen an der Produktionsanlage vor Ort konzipiert und daher nicht geeignet, komplexe Roboterprogramme und Applikationslogiken auf Expertenebene zu entwickeln. In der industriellen Praxis finden dabei textuelle Offline-Programmiertools der Hersteller Anwendung. Diese stellen Entwicklungsumgebungen für die proprietären Robotersysteme und deren Programmiersprachen dar und sind nicht universell oder herstellerübergreifend einsetzbar. Zudem unterscheiden sich die Tools in ihrer Funktionalität sehr stark. So bietet das ABB Robotstudio eine moderne und funktionsreiche IDE inkl. Simulationsumgebung und Buskonfiguration an, während andere Entwicklungsumgebungen keine dieser Funktionen zur Verfügung stellen, sondern über zusätzliche Software von Drittanbietern angeboten werden müssen. Diese Zusatzfunktionen stehen im Rahmen vorliegender Arbeit jedoch nicht im Fokus, da sowohl bei der Simulationsumgebung als auch der Feldbuskonfiguration die Problematik der Herstellerabhängigkeit weniger schwerwiegend als in der Programmierung und Applikationsentwicklung ist. Zum einen sind diese Themen beispielsweise mit RSS (Seite 16) oder der Plug & Produce-Methode (Seite 24) bereits technisch gelöst oder es existieren zumindest anwendbare Lösungskonzepte. Zum anderen reduziert ein standardisiertes Vorgehen und das bereitgestellte Tooling die Aufwände der Herstellerabhängigkeit in der Feldbuskonfiguration bei der Inbetriebnahme (Kapitel 4.4). Aus diesem Grund stellt die herstellerunabhängige textuelle Programmentwicklung auf Expertenebene die größten Potenziale in der Roboterapplikationsentwicklung für den Karosseriebau dar.

Neben den gegebenen Einschränkungen infolge der Proprietarität bieten die Herstellertools im direkten Vergleich zu Entwicklungsumgebungen aus dem Softwarebereich, wie z. B. Visual Studio oder Eclipse, meist einen geringeren Funktionsumfang in Bezug auf die textuelle Programmierung, da gängige Funktionen wie Codeüberprüfung in Echtzeit, Autocompletion oder Objektverzeichnisse nicht durchgängig existieren. Auch wird die Verwendung anderer Programmiersprachen neben der Herstellerprogrammierung nicht unterstützt und es existieren in der Regel auch keine Möglichkeiten, die Tools durch eigene Add-Ons oder Plugins benutzerspezifisch zu erweitern. Es ist zu erwähnen, dass ausgelöst durch die Servicerobotik auch eine Neuorientierung ersichtlich ist. Kuka bietet beispielweise mit der Sunrise Workbench eine auf Eclipse aufbauende Entwicklungsumgebung an, die stark von der vorhandenen Funktionalität des Open-Source-Tools profitiert und für Anwender mit Eclipse-Erfahrung einen gewissen Wiedererkennungswert bietet und so den Einstieg in die Roboterprogrammierung, zumindest für Softwareentwickler, erleichtert. Allerdings bietet das Tool lediglich die Programmierung mit Java und nicht die Verwendung oder Umschaltung zu einer domänenspezifischen Roboterprogrammiersprache an. Zudem steht die Sunrise Workbench aktuell nur für den Serviceroboter LBR iiwa sowie der mobilen Plattform KMR iiwa zur Verfügung, eine Integration in die Standard-Industrierobotersysteme für den Karosseriebau ist bisher nicht kommuniziert.

7.2.1. Anforderungen

Aus den gegebenen Randbedingungen wurden folgende Anforderungen an die Programmierumgebung gesetzt, die heute von keinem der existierenden Systeme in dieser Kombination erfüllt werden:

- Robotersystemherstellerunabhängig
- Unterstützung einer offenen domänenspezifischen Roboterprogrammiersprache (AIRL)
- Unterstützung einer Generalpurpose Language (C++)
- Parallele Anwendung beider Sprachkonzepte mit Kontextumschaltung
- Offene, plattformunabhängige und erweiterbare Technologie als Softwarebasis
- State of the Art Funktion aktueller IDE in der Softwareentwicklung wie
  z. B. Auto Completion, Syntax Highlighting, Objekt-, Struktur- und Fehlerinformationen

7.2.2. Umsetzung

Aufgrund der in den vorherigen Abschnitten definierten Eingrenzung und Anforderungsdefinition wurde im Rahmen dieser Arbeit eine IDE-Erweiterung entwickelt, welche die formulierten Forderungen exemplarisch umsetzt [163].

Die technologische Grundlage stellt die offene Entwicklungsumgebung Eclipse[20] dar. Die ursprünglich für Java-Programmierung konzipierte IDE der Eclipse Foundation bietet dank der Interoperabilität (Windows, Linux, Mac OS) und Vielsprachigkeit (primär Java, C++, PHP) Vorzüge gegenüber anderen Softwarewerkzeugen oder einer vollständigen Eigenentwicklung einer IDE.

Zudem unterstützen zahlreiche Plugins die Arbeit im Eclipse-Umfeld. Für die Definition und Verwendung von domänenspezifischen Programmiersprachen können sogenannte Language Workbenches [161] wie Rascal MPL[21], Spoofax[22] und Xtext[23] genutzt werden. Diese bieten die Möglichkeit, Grammatik und Semantik einer DSL zu definieren. Darauf aufbauend stehen in Verbindung mit Eclipse zahlreiche Funktionen automatisch zur Verfügung, welche die Arbeit mit einer Programmiersprache unterstützen sollen. Darunter sind infrastrukturelle Themen wie Compiler-, Interpreter-, Debugging-Funktionalitäten und die Entwicklungsumgebung selbst für die spezifische DSL zu verstehen. Darüber hinaus sind zahlreiche syntaktische und semantische Dienste zu nennen, welche die eigentliche Arbeit mit der DSL vereinfachen. Beispielsweise Auto Completion, Syntax- oder Fehler-Hervorhebung. Angesichts des Funktionsumfangs, der ausführlichen Dokumentation und der guten Integration in die Eclipse Community, die es ermöglichen, die formulierten Anforderungen abzudecken, wurde sich innerhalb der Arbeit für das XText-Framework entschieden.

AIRL-Erweiterung von Eclipse

Im Folgenden soll auf die AIRL-Entwicklungsumgebung auf Eclipse-Basis kurz eingegangen werden, die in Abbildung 64 dargestellt ist und die bekannte Eclipse-Fensteraufteilung zeigt. Die Ordner- und Dateistruktur eins Projektes werden im Projekt Explorer (1) aufgelistet. Offene Dateien werden über die auswählbaren Tab Elemente (2) dargestellt. Es sind sowohl AIRL (.airl) als auch C++ (.cpp) Dateien auswählbar, um die Entwicklung sowohl in der domänenspezifischen

[20] https://www.eclipse.org
[21] https://www.rascal-mpl.org
[22] http://www.metaborg.org
[23] https://www.eclipse.org/Xtext

AIRL als auch in der universellen Hochsprache C++ zu ermöglichen. Im Editorfenster (3) wird das aktuell bearbeitete Programm angezeigt. Das Fenster Outline (4) zeigt Daten und Funktionsstruktur des aktuellen Programms. Dabei werden Namen, Typen und aktuelle Werte angezeigt.

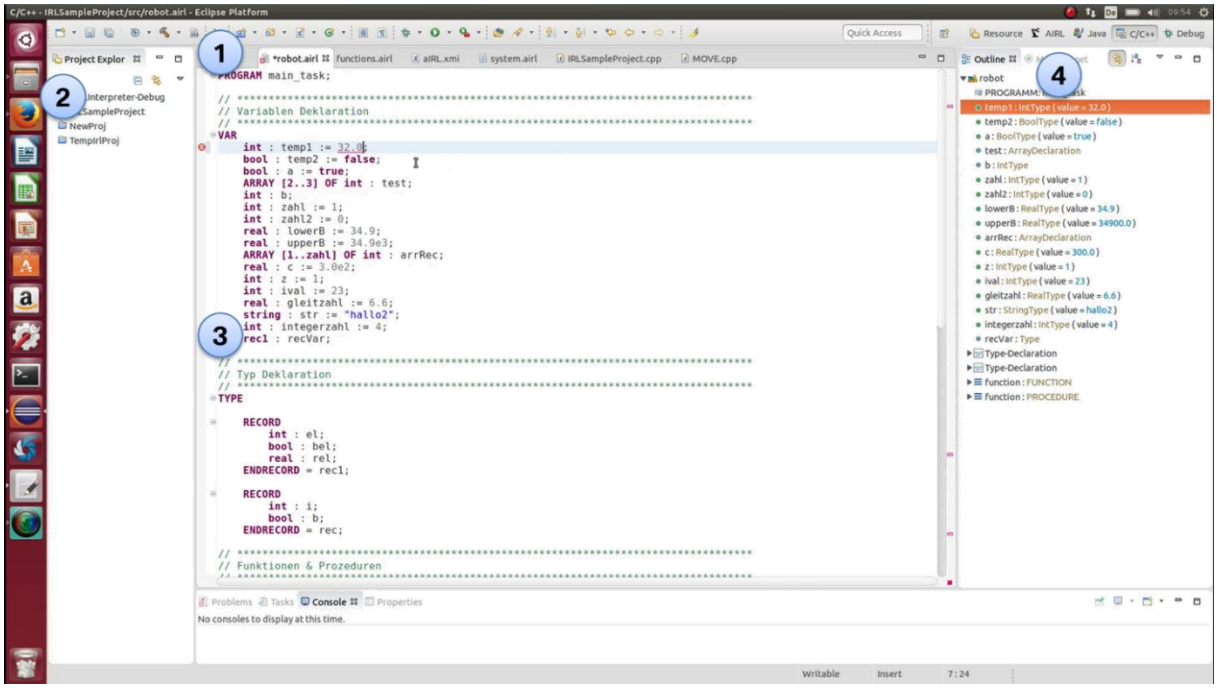


Abbildung 64 – Eclipse IDE mit AIRL-Programm

Unterstützungsfunktionen

Eine zentrale Anforderung an die integrierte Entwicklungsumgebung waren on-the-fly Methoden zur Unterstützung bei der Programmerstellung. Diese werden bei XText zum Teil vollständig automatisch aus der AIRL-Definition im Framework ausgeleitet oder können manuell mit der aus dem XText-Projekt entstandenen Sprache Xtend ergänzt werden. Für die C++ Programmbestandteile bietet Eclipse bereits die entsprechenden Funktionen von Haus aus an.

Syntax Coloring

Einzelne Programmbestandteile wie Funktionsdeklarationen, Variablentypen, Zeichenketten oder Kommentare werden in unterschiedlichen Farben dargestellt, um Programmierern die Unterscheidung zwischen Elementen und die Orientierung im Programm zu erleichtern.

Validierung

Zusätzlich bieten interpretative Zusatzfunktionen wie Duplikatsüberprüfung oder Typ- und Zuordnungsvalidierung die Möglichkeit, syntaktische und semantische Fehler zu vermeiden. Aufgrund der dynamischen Validierung werden diese Hilfsmittel bereits während der Programmentwicklung und nicht erst beim abschließenden Debuggen oder Compilieren angezeigt. Abbildung 65 zeigt unterschiedliche Validierungsfunktionen im Vergleich.

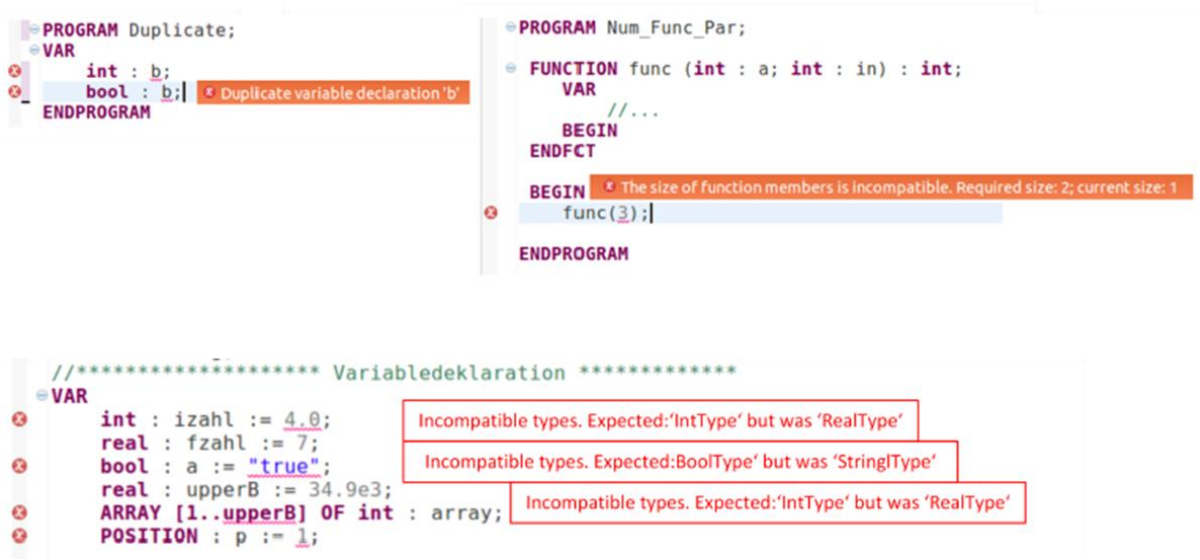


Abbildung 65 – Syntaxfehler Highlighting von AIRL-Syntax Fehlern

Content Assist und Code Completion

Ein weiteres Hilfsmittel, das den Nutzer bei der Programmierung unterstützt, sind sogenannte „Content Assist" Funktionen wie Code Completion. Mit diesen werden kontextabhängige Vorschläge zur automatischen Code-Vervollständigung gemacht. Dies umfasst sowohl statische Elemente, die in AIRL-Syntax definiert sind, als auch dynamische Ergänzungen, die sich auf ein durch den Programmierer erstelltes AIRL-Programm beziehen.

Abbildung 66 zeigt die automatische Vervollständigungsfunktion. Auf der linken Seite ist der Typ Position mit den kartesischen X, Y, Z Parametern zu sehen. Auf der rechten Seite befindet sich ein vom Anwender definierter Record Datentyp der AIRL. Dieser ist analog klassischer Struktur-Elemente in Hochsprachen definiert. Im Beispiel besteht dieser aus einer booleschen und einer ganzzahligen Variable.

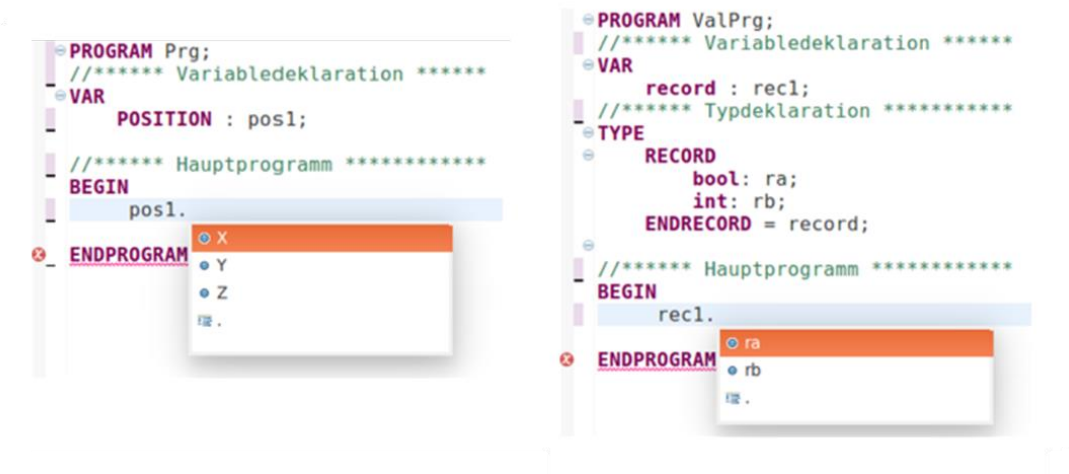


Abbildung 66 – Code Completion für AIRL-Variable

Quick Fix

Quick Fix stellen Bedienelemente dar, die Validierungs- und Content Assist Funktionen vereinen. Hierbei wird der Anwender durch automatisch eingeblendete Lösungsvorschläge unterstützt, um bei der Programmvalidierung identifizierte Fehler zu korrigieren.

Abbildung 67 zeigt dies am Beispiel zweier Bewegungsbefehle, in denen der Robotertyp nicht korrekt angegeben wurde und Korrekturoptionen angeboten werden.

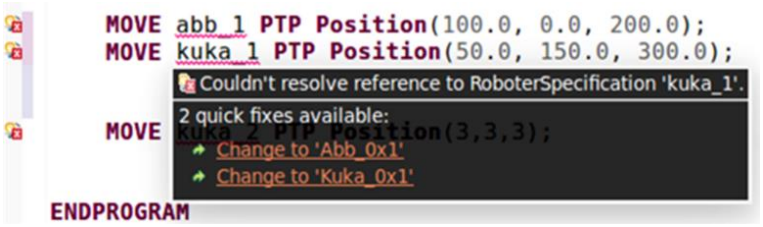


Abbildung 67 – Quick Fix Optionen für AIRL-Syntax Fehler

Hyperlinking zu AIRL und C++ Funktionen

Durch Hyperlinking wird dem Benutzer ermöglicht, zwischen Deklaration und Verwendung von Programmelementen zu springen. Dies ist nicht nur für Variablen und Funktionen innerhalb eines AIRL oder C++ Programms möglich, sondern auch zwischen den Elementen beider Programmiersprachen. Dadurch ist es möglich, bei der AIRL-Programmierung auf die damit verbundenen C++ Methoden zu springen. Insbesondere für die Applikationsentwicklung auf Expertenebene ermöglicht dies die Entwicklung komplexer Programme in C++ und die Einbindung in einfachere AIRL-Strukturen.

7.2.3. Evaluierung

Im Gegensatz zur Bedienerebene wurde für die AIRL-Expertenentwicklungsumgebung eine Evaluierung durch eine Probandenstudie nicht als zielführend betrachtet, da insbesondere die technische Umsetzbarkeit des hybriden Programmiersprachenkonzeptes mit AIRL und C++ in einer modernen Entwicklungsumgebung im Fokus stand. Zudem bietet die implementierte AIRL-Expertenentwicklungsumgebung aufgrund der Verwendung von Eclipse automatisch die Vorteile eines etablierten Softwareentwicklungswerkzeuges und ist damit den meisten proprietären Roboterentwicklungsumgebungen technologisch überlegen, da diese die gezeigten Funktionen häufig nicht vollständig aufweisen. Im direkten Vergleich zu den konventionellen Programmierumgebungen ist der entwickelte Prototyp in doppelter Hinsicht plattformunabhängig. Zum einen ist die Software selbst sowohl auf Windows- als auch Linux-Systemen lauffähig. Zum anderen bietet die AIRL-Ergänzung die Möglichkeit, sowohl roboterherstellerunabhängig domänenspezifische Roboterprogramme als auch im C++ Sprachenkontext zu entwickeln.

Die technische Machbarkeit in Form eines lauffähigen Gesamtsystems wurde anschließend auf Simulationsebene anhand eines AIRL-Interpreters in C++ überprüft, welcher ein in der AIRL-Eclipse IDE entwickeltes Roboterprogramm ausführen kann und den AIRL zugrunde liegenden C++/AIRL-Referenzbefehl zur Durchführung abruft. Dabei konnte von der Xtend-/Eclipse-basierten IDE-Umsetzung profitiert werden, da mittels EMF4CPP die bestehende AIRL-Modellierung der AIRL-DZL aus der Eclipse Java Umgebung des Modelling Frameworks auch in der C++ Implementierung des Interpreters wiederverwendet werden konnte. Eine Reimplementierung der Logik in C++ für die lexikalische und syntaktische Analyse konnte auf diese Weise vermieden werden.

7.3. Herstellerunabhängige Karosseriebauroboterzelle

7.3.1. Anforderungen

Ziel des Prototyps war es, die Erkenntnisse der vorangegangenen Arbeitspakete in einer Robotertestzelle mit herstellerunabhängigen Applikationsstandards zu realisieren und anhand zentraler Anforderungen aus Anwendersicht auszurichten und zu evaluieren (Tabelle 12).

Dies sollte anhand repräsentativer Karosseriebauapplikationen inklusive der zugehörigen Komponenten und Schnittstellenanbindungen des aktuellen Unternehmensstandards erfolgen, gegenwärtige Sicherheitsanforderungen beachten und zudem Erweiterungsmöglichkeiten vorhalten, um zukünftig Untersuchungen über die gegenwärtige Zielstellung hinaus zu ermöglichen.

Tabelle 12 – Anforderungen an eine herstellerunabhängige Karosseriebauroboterzelle

Anforderung	Unterpunkte
Synthese der vorherigen Arbeitspakete	• Externer Ansteuerung der Robotersysteme • Applikationssoftware in einer GPL • Anwendersoftware über graphische Benutzeroberfläche
Roboterherstellerunabhängige Umsetzung der Applikationsstandards	• Herstellerunabhängige Anbindung der Peripherie • Herstellerunabhängige Programmierung der Basislogik • Herstellerunabhängige Bedienung auf Anwenderebene
Hoher Praxisbezug und Aussagekraft für den automobilen Karosseriebau	• Auswahl repräsentativer Applikationen für den gesamten Karosseriebau • Reale Industriekomponenten eines Karosseriebaus • Mindestens zwei Robotersysteme unterschiedlicher Hersteller
Integrierbarkeit mit der realen Zellensteuerungsebene	• Verwendung des BMW Standards für SPS <-> Roboterkommunikation • Automatikbetrieb von extern gesteuerten Robotersystemen über SPS ermöglichen
Leistungsfähigkeit	• Genauigkeit und Performance der Manipulatoren • Schaltgenauigkeit • Paralleler Betrieb von Robotersystem und zugehörigen Peripheriekomponenten soll möglich sein
Valides Sicherheitskonzept	• Zentrale Sicherheitsmechanismen müssen gewährleistet sein, um eine Gefährdung von Personen auszuschließen
Erweiterbarkeit und Vergleichbarkeit der Testzelle	• Vorhalt für die Integration weiterer HW- und SW-Komponenten, um Untersuchungen zu alternativen steuerungstechnischen Konzepten zu ermöglichen • Parallele Umsetzung der klassischen Applikationsstandards in den Robotersystemen

7.3.2. Exemplarische Karosseriebauapplikationen

Bereits in Kapitel 4.3 und 6.3.2 wurden die verschiedenen Applikationen im automobilen Karosseriebau erläutert und hinsichtlich ihrer Verwendung im Werk analysiert. Eine vollumfängliche Umsetzung aller Anwendungen innerhalb des Prototyps wäre allerdings weder realistisch noch wirtschaftlich sinnvoll. Die Wahl der richtigen Referenzapplikationen ist daher von Bedeutung, um die geforderte Aufgabenstellung zu erfüllen und mit einer geringen Anzahl an Applikationen ein breites Anforderungsbild abzudecken. Aus diesen Grund wurden die Fügeprozesse Punktschweißen und Kleben sowie die Basisanwendungen BMW Grundsoftware, Werkzeugwechsel und Handling für die Applikationsumsetzung identifiziert. Zum einen adressieren diese sowohl Punkt- als auch Bahnprozesse (Abbildung 68) und decken alle Kategorien Applikationseinteilung des IFR (Seite 40) ab. Zum anderen sind abgesehen von der Bauteilsuche die Top 5 der real verwendeten Applikationen im Karosseriebau des Fallbeispiels abgedeckt.

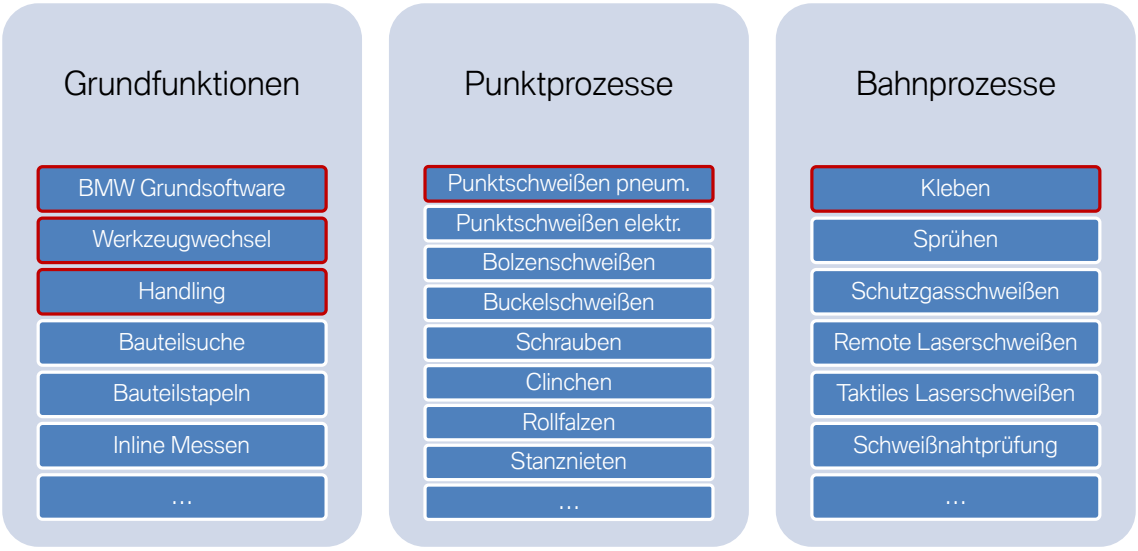


Abbildung 68 – Im Prototyp umzusetzende Karosseriebauapplikationen (rot markiert)

BMW Grundsoftware

Diese Applikation bildet die Kommunikation zum Leitstand bzw. der Zellensteuerung ab. Jeder eingesetzte Roboter besitzt dieses Softwarepaket, um notwendige Informationen bezüglich abzuarbeitender Aufträge, freier Arbeitsbereiche und Fehlerrückmeldungen mit der übergeordneten Steuerungsinstanz auszutauschen. Anforderungen ergeben sich vor allem im Nachrichtenaustausch auf I/O-Ebene und der Abbildung der notwendigen Logikverzweigungen.

Handling

Als Handling-Anwendungen werden Operationen mit pneumatischen oder elektrischen Greifern bezeichnet, die eingesetzt werden um Bauteile aufzunehmen, zu fixieren und abzulegen. Es handelt sich um die am häufigsten eingesetzte Applikation.

Punktschweißen

Punktschweißen ist der am meisten eingesetzte Fügeprozess im Karosseriebau. Über servopneumatische oder -elektrische Schweißzangen werden Bleche an spezifischen Punkten gefügt. Aufgabe der Steuerungstechnik ist die Positionierung der Schweißzange relativ zum Bauteil und der Vorhalt einer ausreichend großen Zangenweite. Zudem müssen notwendige Schweißpunktinformationen an die Schweißsteuerung weitergeleitet werden, welche für die Stromregelung der Schweißprozesse zuständig ist. Roboterseitig entsprechen Punktschweißen und Handling einem Anforderungsprofil, das auf sämtliche statische Fertigungsprozesse übertragbar ist.

Werkzeugwechsler

Die Integration der Anwendung Werkzeugwechsler ist an sich nicht erforderlich, da sie aus Anforderungssicht bereits über Punktschweißen bzw. Handling abgedeckt wird. Jedoch ist es aus Gründen der Praktikabilität notwendig, Werkzeuge im Prototyp zu tauschen, um auf die Messtechnik der Leistungsmessungsversuche zu wechseln und die Testzelle auch für weitere Untersuchungen flexibel gestaltbar zu halten.

Kleben

Bei der Anwendung Kleben wird über eine externe oder am Roboter montierte Klebedüse der zähflüssige Klebstoff aufgetragen. Die Regelung der Düse erfolgt nicht über die Roboter-, sondern über die Klebesteuerung. Zentrale Anforderung an das Robotersystem ist die korrekte und frühzeitige Übermittlung der aktuellen TCP-Geschwindigkeit an die Klebesteuerung, um zu gewährleisten, dass die korrekte Klebstoffmenge aufgetragen wird. Der Klebeprozess spiegelt die Anforderungen für andere bahnsynchrone Operationen wie Laserstrahlschneiden oder Bahnschweißen wider.

7.3.3. Layout und Komponenten der Testzelle

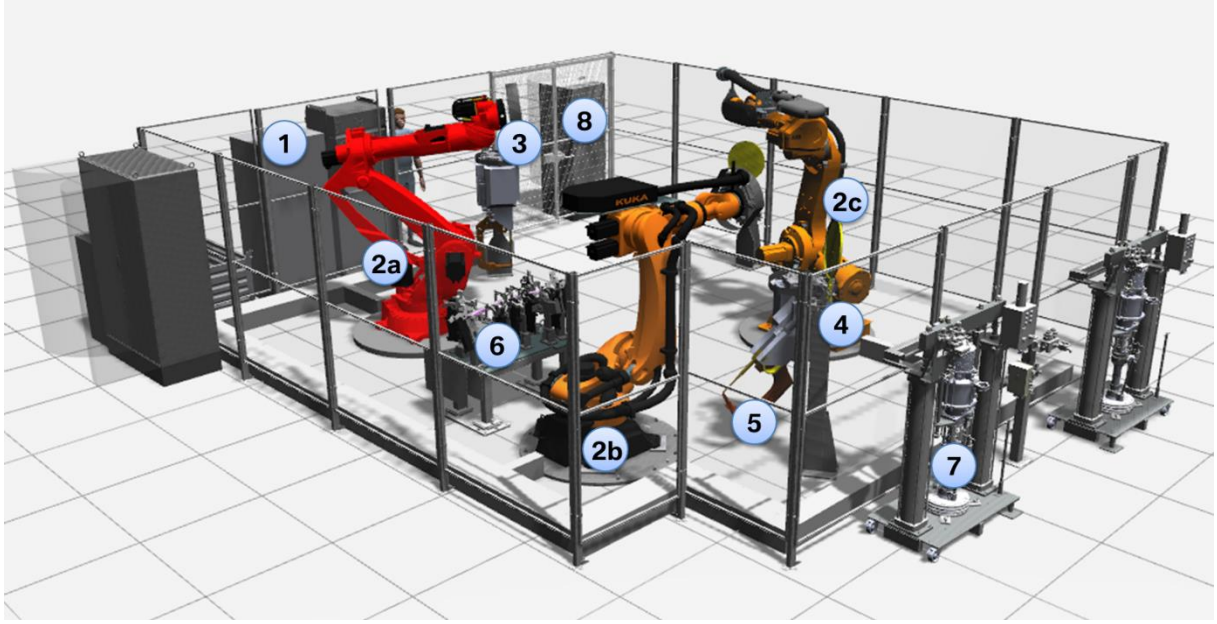


Abbildung 69 – 3D-Darstellung der Testzelle

Abbildung 69 zeigt den auf Basis der zu untersuchenden Applikationen entstandenen Planungsentwurf für eine Karosseriebautestzelle. Dabei wurde sich bei Layout und Komponenten an einer Ausschweißstation des BMW-Werks in München orientiert, von der Einlegevorrichtungen, Schweißzangen, Greiferkomponenten und Bauteile übernommen werden konnten. Für bahnsynchrone Operationen wurde die Testzelle um einen Klebeprozess mit den notwendigen Anlagenbestandteilen ergänzt. Des Weiteren wurden statt drei Roboter eines Herstellers, jeweils ein Roboter von drei unterschiedlichen Herstellern integriert, um eine möglichst große Vergleichsmenge abbilden zu können.

Die Roboterzelle und die eingesetzten Komponenten werden im Folgenden erläutert:

Zellensteuerung (1)

Die übergeordnete Steuerungslogik der Testzelle wird in ein S7-400 der Firma Siemens verarbeitet. Zum Einsatz kommen BMW-spezifische Schnittstellen zu den eingesetzten Peripheriekomponenten (insb. Roboter). Aufgabe der Zellensteuerung ist die Steuerung des übergeordneten Programmablaufs und die antriebstechnische Freigabe der Roboter. Zudem ist die Zellensteuerung die zentrale Sicherheitskomponente und übernimmt den zentralen Bedienerschutz.

Robotersysteme (2)

Die Testzelle wurde mit 6-Achs-Robotersystemen von Comau (2a), Kuka (2b) und ABB (2c) ausgestattet. Alle drei Manipulatoren erfüllen das übliche Karosseriebauleistungsspektrum von Traglasten zwischen 220 und 240 kg und einer Reichweite von ca. 2,5 Metern. Der Comau-Roboter weißt die Besonderheit einer Hollow Wrist-Mechanik auf, in der das Schlauchpaket innenliegend durch die Achsen geführt wird. Im Vergleich zum üblichen, außen anliegenden Schlauchpaket soll dies vor Schäden an der Medienversorgung im Produktionsbetrieb schützen. Des Weiteren verfügen alle Steuerungssysteme über native Schnittstellen auf Interpolationsebene. Die Kuka-Steuerung enthält mit MXAutomation herstellerseitig zusätzlich eine Schnittstelle auf Applikationsebene. Als einziger Hersteller bietet Comau die Möglichkeit, den Manipulator auch direkt elektrisch anzubinden und auf die herkömmliche Robotersteuerung zu verzichten. Zudem verfügt das Comau-System auch auf Ebene der Interpolationsschnittstelle über die Option, Zykluszeiten bis auf 400 µs reduzieren zu können. ABB hingegen stellt dem Anwender frei, ob die Interpolationsschnittstelle über eine UDP-Anbindung oder mit ProfiNet über eine Industrial Ethernet-Verbindung erfolgen soll.

Roboter	COMAU ROBOTICS	KUKA	ABB
Manipulator	SMART5 NJ4 220	KR240R2500 prime	IRB 6640
Traglast / Reichweite	220 kg / 2,4 m	240 kg / 2,5 m	235 kg / 2,55 m
RP (Posewiederholg. nach ISO 9283)	0,15 mm	0,06 mm	0,05 mm
Ausführung	Punktschweiß & Handling	Punktschweiß & Handling	Punktschweiß & Handling
Sonstiges	Hollow Wrist-Modell	-	-
Steuerung	C5G	KRC4	IRC5
Version	1.20.011	8.3.11	6.02.0106
IPO-Schnittstelle	C5GOpen	RSI	EGM
- Zykluszeit	400 µs, 4 ms, 8 ms, 16 ms	4 ms, 12 ms	4 ms, 8 ms, 12 ms, (n * 4 ms)
- Bus Kommunikation	Powerlink	UDP	ProfiNet / UDP
weitere (ext.) Schnittstelle	Antriebsregelung	Applikation (MXAutomation)	keine

Abbildung 70 – Eingesetzte Robotersysteme und Schnittstellen

Werkzeugwechsler (3)

PTM-Werkzeugwechslersystem bestehend aus Roboter- und Werkzeugseite mit Medienversorgung für 400 V, 24 V, Industrial Ethernet und Druckluft. Aufgabe des Werkzeugwechslersystems ist es, den automatisierten Austausch des angebrachten Werkzeugs zu ermöglichen und den flexiblen Einsatz des Robotersystems zu gewährleisten.

Werkzeugbahnhöfe (4)

Die Werkzeugbahnhöfe der Firma PTM werden dazu genutzt, um mit einem Werkzeugwechsler ausgestattetes Werkzeug stationär abzulegen. Angesteuert wird dieser durch ein I/O-geschaltetes Druckluftventil.

Punktschweißzangen (5)

Exemplarische Prozesswerkzeuge für Punktschweißen stellen servopneumatische X- und C-Zangen in der Testzelle dar. Die Regelung erfolgt über einen intelligenten Festo-Controller, der die passenden Sollwerte für Anstellposition und Anpressdruck aus der Applikationslogik der Roboter- bzw. externen Steuerung erhält. Während des Schweißprozesses verharrt das

Robotersystem in fester Position, jedoch erfolgt die Vorpositionierung während der Bewegungsvorgänge.

Greifer/Spannvorrichtung(6)

Zur Auf- und Abnahme von Bauteilen wird eine anwendungsspezifische Spannvorrichtung bzw. ein Pneumatikgreifer verwendet. Die Spanner werden über ein Druckluftventil mit den Positionen offen und geschlossen angesteuert und fixieren so die zu handhabenden Bauteile mechanisch.

Klebeanlage (7)

Die Intec-Klebeanlage besteht aus einer Klebesteuerung, einer Fasspumpe und einer Klebedüse. Die pneumatische Regelung des zu extrahierenden Klebemediums erfolgt über die Klebesteuerung. Die Menge wird geschwindigkeitsabhängig von der Applikationslogik des Robotersystems vorgegeben.

Neue Steuerungskomponente - Externe Steuerung (8)

Neben den beschriebenen konventionellen Komponenten wurde die Testzelle um die notwendige neue Komponente einer externen Steuerung erweitert, welche die Ansteuerung der Robotersysteme und die Umsetzung der Applikationsprogramme herstellerneutral übernehmen soll. Sowohl Hard- als auch Software sollten dabei möglichst offen und herstellerunabhängig sein, um möglichst keine neuen Abhängigkeiten zu erschaffen. Aus diesem Grund wurde ein Standard-Industrie-PC mit einem Ubuntu-Linux-Betriebssystem als Basis gewählt. Darauf aufbauend wurde mit der Echtzeiterweiterung Preempt-RT und der Robotics Library eine reine Open-Source-Softwareplattform zur Implementierung verwendet. Die externe Steuerung wird daher im nachfolgenden Teil der Arbeit auch als Open-Source-Steuerung bezeichnet.

Auf Basis der beschriebenen Testzelle und Komponenten (1-7) konnten zunächst die Referenzapplikationen mit den proprietären Standardrobotersystemen implementiert werden. Dabei konnte zum Teil auf bestehende BMW-Applikationsstandards zurückgegriffen werden. Jedoch mussten für einige Robotersysteme Teilumfänge der Software individuell entwickelt werden, da kein aktueller BMW-Standard für diesen Lieferanten oder die eingesetzte Steuerungsgeneration zur Verfügung standen. Nach erfolgreicher Implementierung wurden die gleichen Applikationen mit der externen Steuerung (8) und der Einbindung der Robotersteuerungen über die in Kapitel 5 untersuchten externen Schnittstellen umgesetzt. Weitere Details zur steuerungstechnischen Architektur und Umsetzung werden in den nachfolgenden Abschnitten beschrieben.

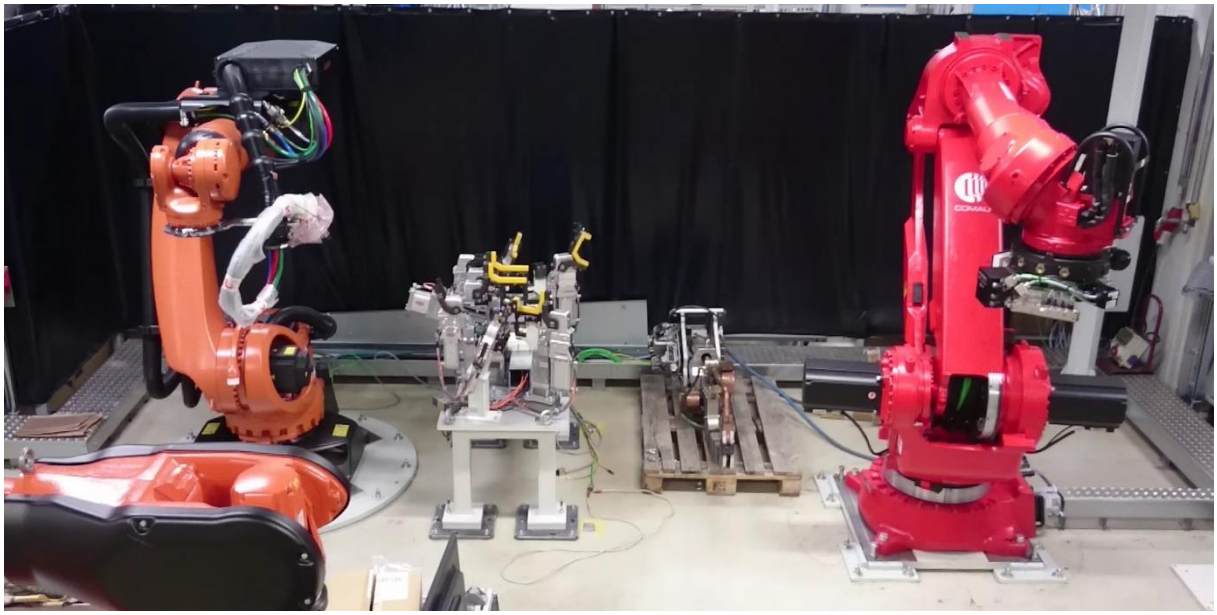


Abbildung 71 – Robotertestzelle in der Inbetriebnahmephase

Abbildung 72 – Steuerungstechnik der Testzelle mit Comau- (u. l.), ABB- (o. r.) und Kuka-Steuerung (u. r.)

7.3.4. Steuerungsarchitektur und Sicherheitskonzept

Integration Robotersysteme in Zellensteuerung

Eine zentrale Anforderung im Rahmen der Umsetzung des Prototyps war die Integration der externen Steuerung in den vorhandenen Zellensteuerungsstandard. Die größte Herausforderung lag dabei in der parallelisierten Ansteuerung der nativen und der Open-Source-Steuerungssysteme, da hierfür die Standardschnittstelle zwischen SPS und Roboter aufgeteilt werden musste. Die Basislogik für die Antriebsfreigabe und den Automatikprogrammaufruf ist dabei Bestandteil zwischen SPS und nativer Steuerung geblieben, während höherwertige Logiken wie Programm- und Jobaufrufe bzw. Fahrfreigaben in die externe Steuerung überführt wurden. Notwendig war dies, um sicherheitstechnische Aspekte zu gewährleisten. Zudem besaßen die externen Schnittstellen nicht die Funktionalität, eine Antriebsfreigabe des Grundsystems durchzuführen. Außerdem sollte die Robotereinbindung umschaltbar sein, um sowohl den klassischen Normalbetrieb (reine Verwendung der Herstellersysteme) und den herstellerunabhängigen Open-Source-Betrieb zu ermöglichen. Abbildung 73 zeigt das hierfür entwickelte Steuerungskonzept. Die Grafiken auf Seite 158 zeigen die Bedieneroberfläche an der SPS-Zellensteuerung, um Programme zu starten und die Konfiguration zu wechseln.

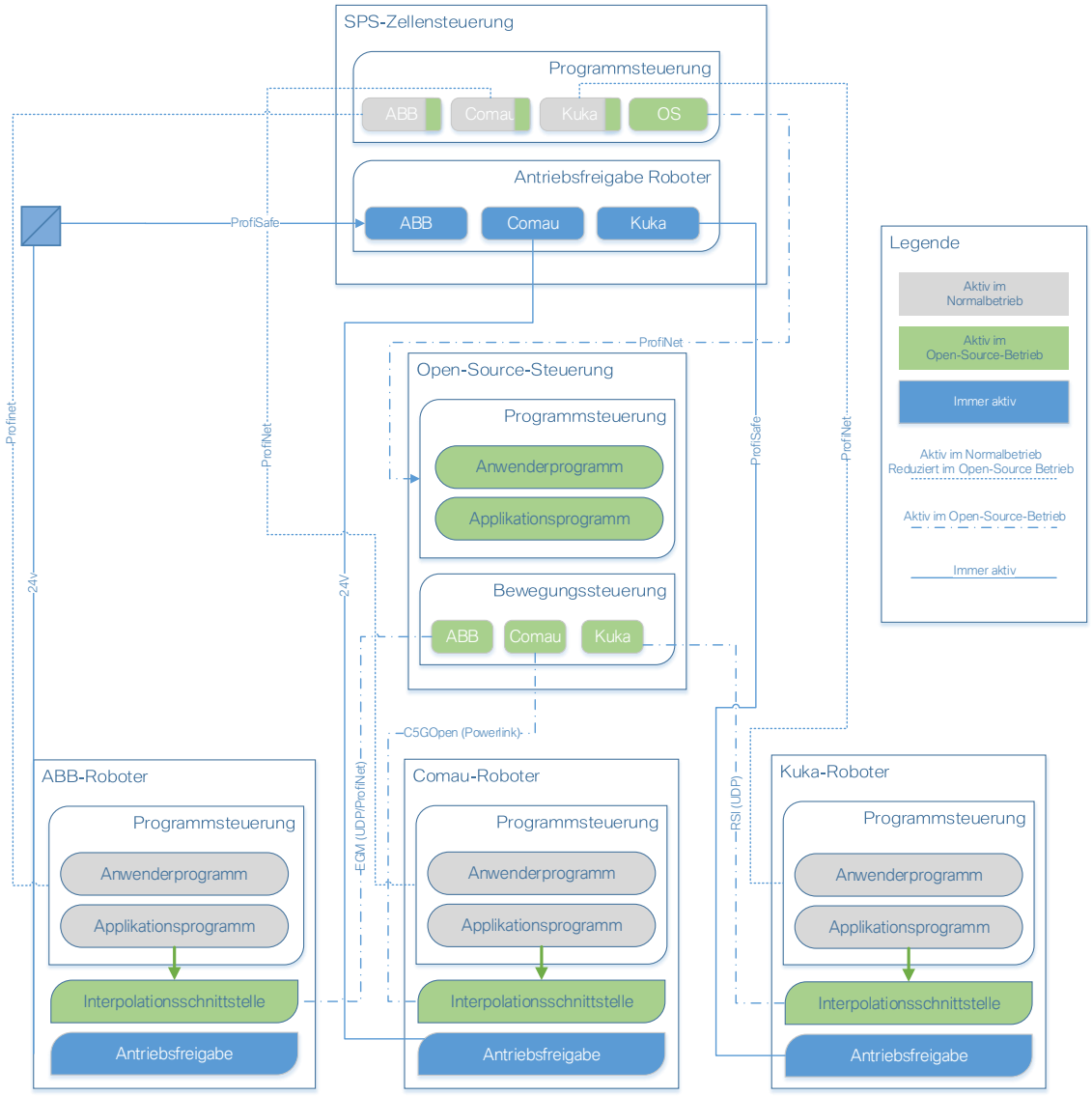


Abbildung 73 – Steuerungsarchitektur Robotertestzelle

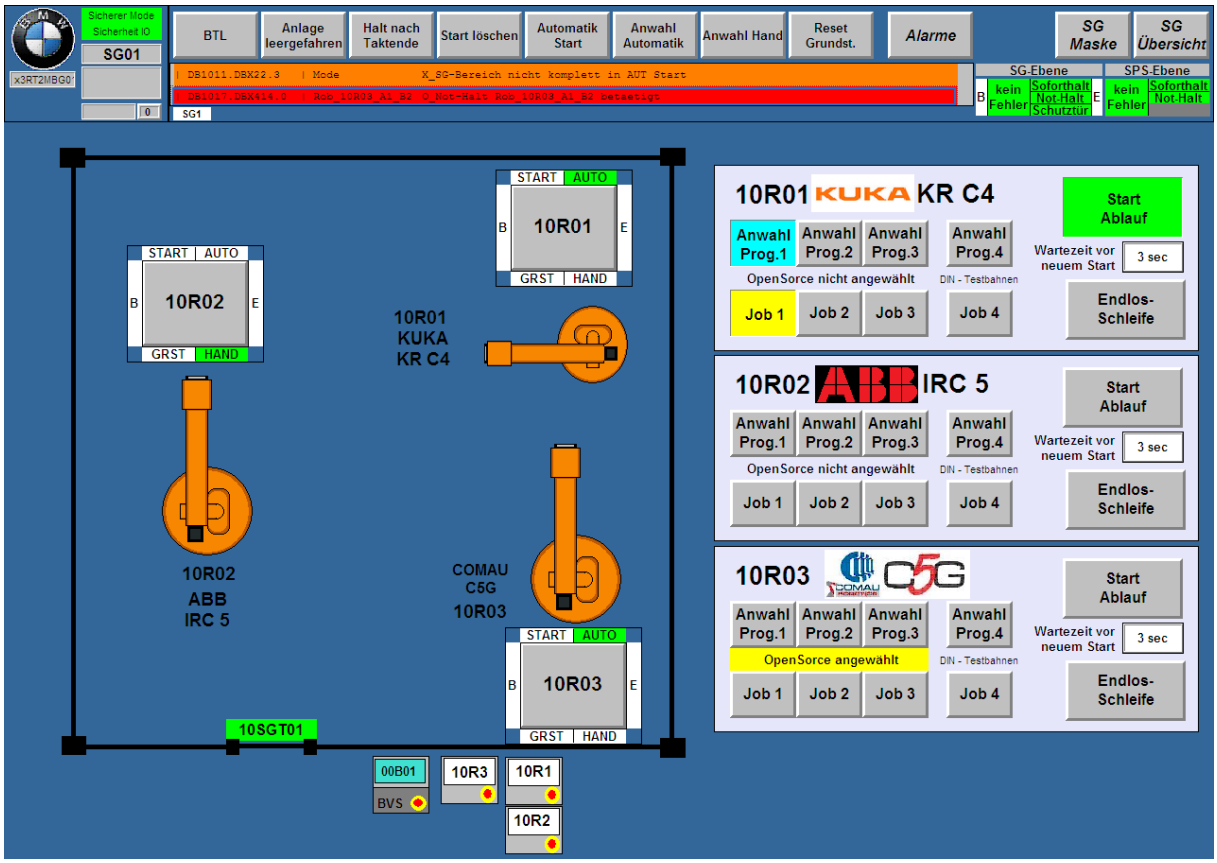


Abbildung 74 – Bedienpanel der SPS-Zellensteuerung: Programm- und Auftragsauswahl für Roboter

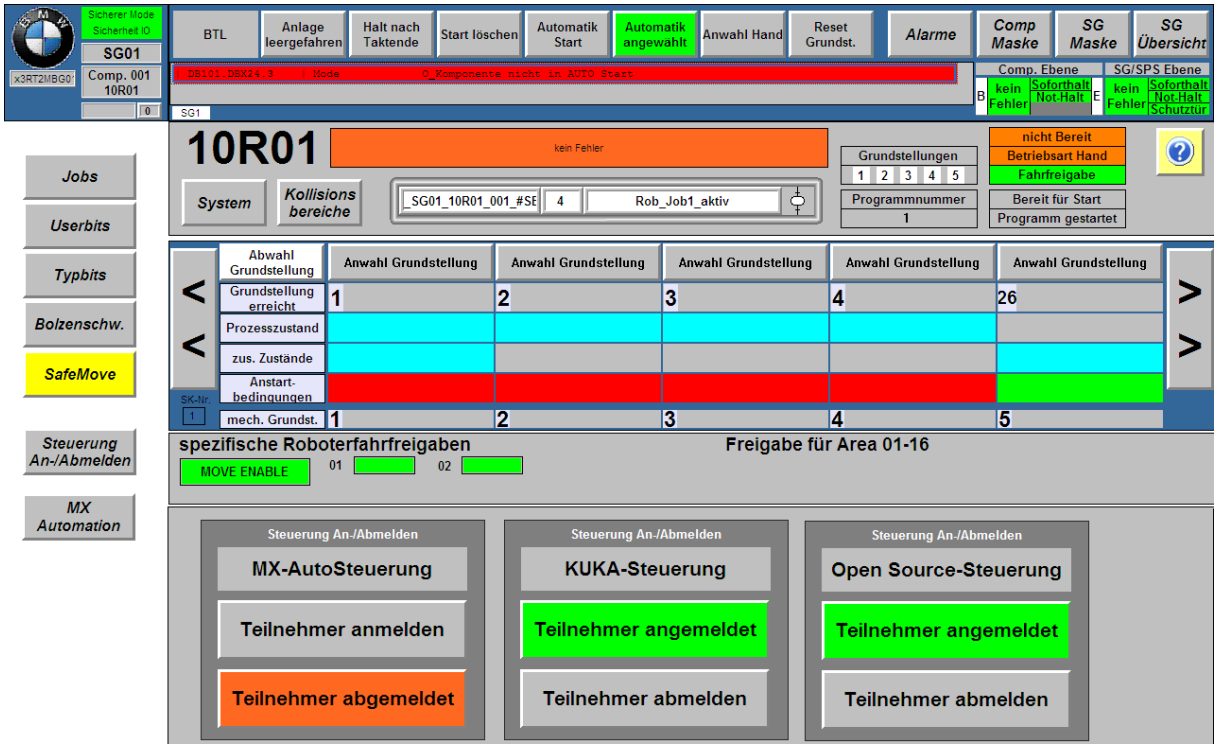


Abbildung 75 – Bedienpanel der SPS-Zellensteuerung: Steuerung der Roboterkonfiguration

Sicherheitskonzept

Aus Betreibersicht spielt das Thema Sicherheit eine entscheidende Rolle für die Integrier- und Verwendbarkeit von Automatisierungstechnik. Die vorhandenen Standards müssen natürlich auch von steuerungstechnischen Alternativen erfüllt bzw. zumindest erfüllbar sein. Eine Orientierung hierfür bietet die Deutsche Gesetzliche Unfallversicherung mit ihren Sicherheitsanforderungen für Industrieroboter [164], die im Kern folgende Themen behandelt:

- Betriebsarten
- Zustimmungseinrichtung
- Funktionale Sicherheit der Steuerung
- Sicher überwachte Steuerung
- Sicherheitshalt
- Not-Halt
- Anlagen-Not-Halt
- Schnellstmögliches Stillsetzen
- Achsbereichsbegrenzung

Durch die Verwendung des bestehenden Standards bezüglich SPS, Schutzeinrichtungen der Roboterzelle und Ansteuerung der Antriebsfreigabe der Robotersteuerung bleiben zunächst sämtliche Sicherheitskonzepte der Robotersysteme in Verbindung mit der Zellensteuerung erhalten. Insbesondere die Haltefunktionen in Verbindung mit dem Not-Taster und der Zelleneinrichtungen sind durch die externe Ansteuerung nicht beeinflusst und waren ebenfalls in der Testzelle grundsätzlich gegeben. Von den eingesetzten Robotersystemen besaß die Comau-Steuerung keine eigene Positionsüberwachung und musste über eine Achsbereichsbegrenzung in seiner Drehbewegung eingeschränkt werden. Der Kuka- und ABB-Roboter war jeweils mit einer Softwareoption zur Bereichsabgrenzung der Roboterbewegung ausgestattet. Sowohl die elektrische Achsbereichsabschaltung als auch die softwareseitige Umsetzung waren im externen Steuerungsbetrieb aktiv und schritten bei zu hohen geometrischen Abweichungen ein und brachten die Robotersysteme zum Stillstand.

Kritischer sind sowohl aus Sicht des Bedienkonzeptes als auch technischer Umsetzung die Anforderungen bzgl. Betriebsarten und Zustimmungseinrichtung zu sehen. Bei allen verwendeten Schnittstellenausführungen beendet die Aufhebung des Zustimmungstasters die Programmausführung und somit die Ansteuerbarkeit der externen Schnittstelle. Eine schnelle Wiederaufnahme ist infolge der durchzuführenden Startprozeduren nicht möglich und das Teachen von Roboterpositionen in den manuellen Betriebsarten T1 und T2 wird erschwert.

Darüber hinaus hat sich im Test gezeigt, dass Roboter ohne Überwachungsoption bei externer Ansteuerung auch in der Betriebsart T1 Geschwindigkeiten über der zulässigen Höchstgrenze von 250 mm/s zulassen. Zwar kann die Robotergeschwindigkeit beim Handbetrieb über das externe System ebenfalls in diesem begrenzt werden, jedoch sind in der derzeitigen Ausführung Interpolationsschnittstellen nicht auf die für diesen Fall notwendige funktional sichere Kommunikation ausgelegt. Ein sicherer Betrieb in T1 kann somit nur durch eine mit der Aktivierung der Interpolationsschnittstelle automatisch verbundenen Geschwindigkeits-überwachung oder Realisierung eines sicheren Schnittstellenkonzeptes gewährleistet werden.

7.3.5. Umsetzung

Konventionelle Steuerungstechnik

Für eine bessere Vergleichbarkeit und eine einfachere Implementierung der Applikationslogiken in die externen Steuerungssysteme wurden zunächst die konventionellen Robotersysteme mit der zu verwendenden Peripherie in Betrieb genommen. Dabei zeigten sich bereits die praktischen Herausforderungen, die durch den Einsatz von Robotersystemen unterschiedlicher Hersteller gegeben sind. Zum einen musste seitens der Zellensteuerung für den nicht im steuerungstechnischen BMW-Standard enthaltenen Comau-Roboter eine individuelle Anbindung für die Antriebs- und Programmfreigabe implementiert werden. Zum anderen mussten roboterseitig die Gegenstellen der bestehenden Standardmodule (Kuka, ABB) konfiguriert bzw. die Implementierung eines prototypischen Standards (Comau) realisiert werden. Darauf aufbauend konnten die notwendigen Anwendungsprogramme für den Testbetrieb erstellt werden. Neben dem grundsätzlichen Mehraufwand, die gleiche Logik aufgrund der proprietären Programmiersprachen mehrfach zu implementieren und nicht wiederverwenden zu können, ist die Tatsache, dass nur wenige Integratoren überhaupt die drei unterschiedlichen Robotersysteme gleichzeitig beherrschen, ein wesentlicher Nachteil, der sich während der Integrationsarbeit der Robotertestzelle zeigte.

Nach Abschluss der Inbetriebnahme lag eine lauffähige Testzelle mit drei Robotersystemen von unterschiedlichen Herstellern und der geforderten Applikationsperipherie, Punktschweißen, Werkzeugwechseln, Handling und Kleben vor.

Externe Steuerungstechnik

Auf Basis der vorhandenen Infrastruktur konnte anschließend, ausgehend von der Roboteransteuerung, mit der Einbindung und Entwicklung der externen Steuerungssysteme und der notwendigen Applikationslogik begonnen werden.

Roboteransteuerung

Zentrale Herausforderungen waren in diesem Kontext wiederum Aufwände, welche durch die proprietäre Schnittstellengestaltung und die unterschiedlichen Anforderungen an Nachrichtenspezifikation und Kommunikationsinfrastruktur gegeben war. Beispielweise unterstützt die Kuka-RSI-Schnittstelle nur UDP-Kommunikation und C5GOpen von Comau einzig Powerlink für die externe Ansteuerung der Robotersysteme. Lediglich ABB bietet mit ProfiNet und UDP die Wahl zwischen zwei Kommunikationskanälen.

Neben der erforderlichen Schnittstellenumsetzung war die Implementierung der kinematischen Robotermodelle notwendig, um eine korrekte Ansteuerung der Roboter auch bei kartesischer Punktevorgabe zu gewährleisten. Die notwendigen geometrischen Daten konnten hierfür jeweils aus den Handbüchern gewonnen werden. Wünschenswert wäre diesbezüglich mehr Unterstützung durch die Roboterhersteller selbst – beispielsweise durch die Bereitstellung von Bibliotheken zur Umrechnung der inversen Kinematik (vgl. Kapitel 5.3.2). Zwar bieten die Interpolationsschnittstellen der Robotersysteme auch die Option der kartesischen Ansteuerung, jedoch verliert hierbei das externe System die Möglichkeit der Kontrolle der Einzelachsen, zudem ergaben sich in den Versuchen Probleme im Singularitätsbereich, die zum kompletten Stillstand führten (Anhang 9.1.4).

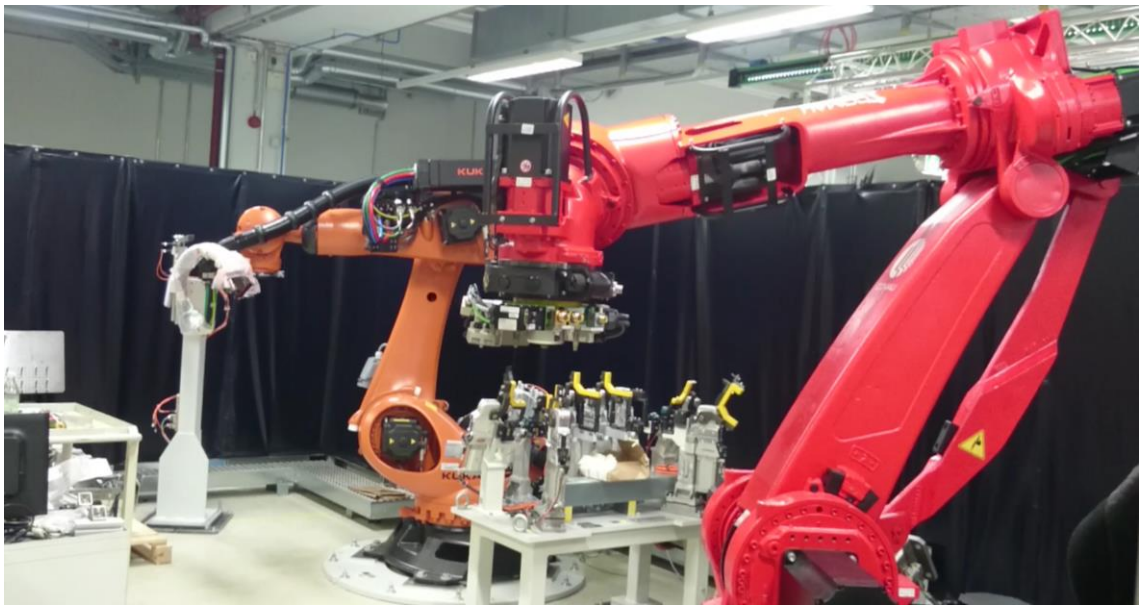


Abbildung 76 – Roboter in synchroner Bewegung bei kooperativer externer Ansteuerung

Mit der erfolgreichen Einbindung der Robotersysteme über die Zellensteuerung und dem externen Steuerungssystem war eine erste Bewertungsplattform geschaffen, um den praktischen Nachweis zu erbringen, dass eine herstellerunabhängige Steuerung möglich ist und ein ausreichend hohes Leistungsniveau für Automobilapplikationen mit sich bringt. Abbildung 76 zeigt beispielsweise eine Versuchsfahrt, in der über die universelle Programmlogik identische, kartesische Roboterbahnen zeitsynchron durch Vorgabe von Achspositionen auf Interpolationsebene abgefahren wurden, um die Abstraktion der Bewegungssteuerung unterschiedlicher Robotersysteme zu verdeutlichen.

Darüber hinaus wurde mittels Lasertracker das genaue Leistungsniveau der extern gesteuerten Robotersysteme mit der nativen Robotersteuerung ausführlich verglichen und bewertet. Hierfür sei auf die in Kapitel 5.2 aufgeführten Untersuchungen und Messreihen verwiesen.

Peripheriekommunikation

Neben der Verbindung zwischen externer Steuerung und Robotersystemen bestand aus Kommunikationsinfrastruktursicht die Notwendigkeit, ProfiNet als Industrial Ethernet Protokoll in die Open-Source-Steuerung zu implementieren, um mit Peripheriegeräten gemäß vorgegebenem BMW-Standard zu kommunizieren. Dies wurde mit einer PCI-Express-Kartenlösung der Firma Hilscher umgesetzt und über eine zugehörige C++ API in die RL integriert. Die technische Wahl der geschlossenen Lösung war der Tatsache geschuldet, dass zum Umsetzungszeitpunkt kein offener softwarebasierter ProfiNet-Masterstack existierte und die Komplexität und der Implementierungsaufwand in der Peripherieumsetzung gering gehalten werden sollten. Ein Austausch mit einer offenen ProfiNet-Echtzeitkommunikationsbibliothek wäre mit vertretbarem Aufwand möglich und auch wünschenswert, sollte diese zukünftig zur Verfügung stehen. Da die Anbindung des Comau-Roboters bereits über das offene Powerlink-Protokoll in der Open-Source-Steuerung erfolgt, kann die technische Umsetzbarkeit einer Echtzeit-Ethernet-Kommunikation in das bestehende System auch als gegeben angesehen werden.

Ergänzend zur grundsätzlichen Umsetzung der Peripheriekommunikation war aus Sicht der Leistungsfähigkeit insbesondere die Schaltgenauigkeit in Kombination mit den extern gesteuerten Robotern eine zu beantwortende Fragestellung, die auf Seite 83 f. im Kapitel der Bahnkenngrößen (5.2.3) behandelt wurde.

Applikationslogik

Nachdem die Bewegungssteuerung der Robotersysteme und die Peripheriekommunikation über das Open-Source-System erfolgreich umgesetzt und die Leistungsfähigkeit nachgewiesen wurde, konnte mit der Implementierung der herstellerunabhängigen Applikationslogik im Prototyp begonnen werden. Umzusetzende Elemente waren zum einen die Schnittstellen-kommunikation mit der Peripherie, ausgehend von der korrekten Bitbelegung über die Interpretation und Aufbereitung der empfangenen bzw. zu sendenden Daten bis hin zur Behandlung von Handshake- sowie Fehlerbehandlungsroutinen. Zum anderen wurden applikationsspezifische Prozessparameter und Ablaufverhalten definiert. Hierzu wurden in der externen Steuerung die relevanten Logiken für den Betrieb der Schweißzangencontroller, der I/O-Module für Werkzeugwechsler und Handling sowie die Anbindung der Klebesteuerung integriert.

Bei der Umsetzung der Applikationslogik wurden die positiven Eigenschaften der gewählten Steuerungsarchitektur und der verwendeten Programmiersprache C++ sehr deutlich. Zum einen konnten dank der Objektorientierung die bekannten Vorteile gegenüber der üblichen, prozeduralen Roboterprogrammierung nun im Kontext von Automobilapplikationen genutzt werden. Beispielsweise können die realen Komponenten und deren Funktionalitäten in wiederverwendbare Objekte und Methoden strukturiert und anhand von Vererbungs- und Interfacelogiken Standardisierungsvorgaben auch auf Programmiereben leicht einbindbar gestaltet und deren Nutzung sichergestellt werden. Auf diese Weise profitiert die Gestaltung der Karosseriebauapplikation durch klarere Vorgaben und Strukturen. Zum anderen kann die Karosseriebauapplikationslogik weitestgehend sowohl unabhängig von der eingesetzten Roboter- als auch von der Komponentenhardware gestaltet werden, da diese nicht auf dem proprietären Robotersystem definiert und ausgeführt wird und dank der Objektorientierung wesentlich bessere Methoden der Logikabstraktion zur Verfügung stehen. Des Weiteren wurde die bisherige Einschränkung aufgehoben, mit roboterspezifischen Entwicklungsumgebungen arbeiten zu müssen, da aufgrund der Verwendung einer GPL als Programmiersprachenbasis, die Implementierung der Softwareumfänge mit der offene Eclipse Plattform statt mit den herstellereigenen Tools erfolgen konnte.

Aus Anwendungssicht ergeben sich durch die Gestaltung der Applikationslogik in der externen Steuerung keine neuen Anforderungen an die bereits vorangegangene Leistungsbewertung, mit der die prinzipielle Tauglichkeit aus fertigungstechnischer Sicht bereits nachgewiesen wurde. Aus infrastruktureller Sicht war es jedoch von Interesse, ob das gewählte Hardwaresetup grundsätzlich die parallele Ansteuerung der Robotersysteme und mehrere Peripheriekomponenten zeitgleich bewältigen kann. Hierfür wurde der beschriebene Synchronlauf noch um die Ansteuerung der Komponenten Werkzeugwechsler, Werkzeugbahnhof und Schweißzange erweitert und erfolgreich getestet (Abbildung 77). Die herstellerunabhängige und synchrone Ansteuerung der Robotersysteme und der zugehörigen Peripherie über das externe System konnte so nachgewiesen werden.

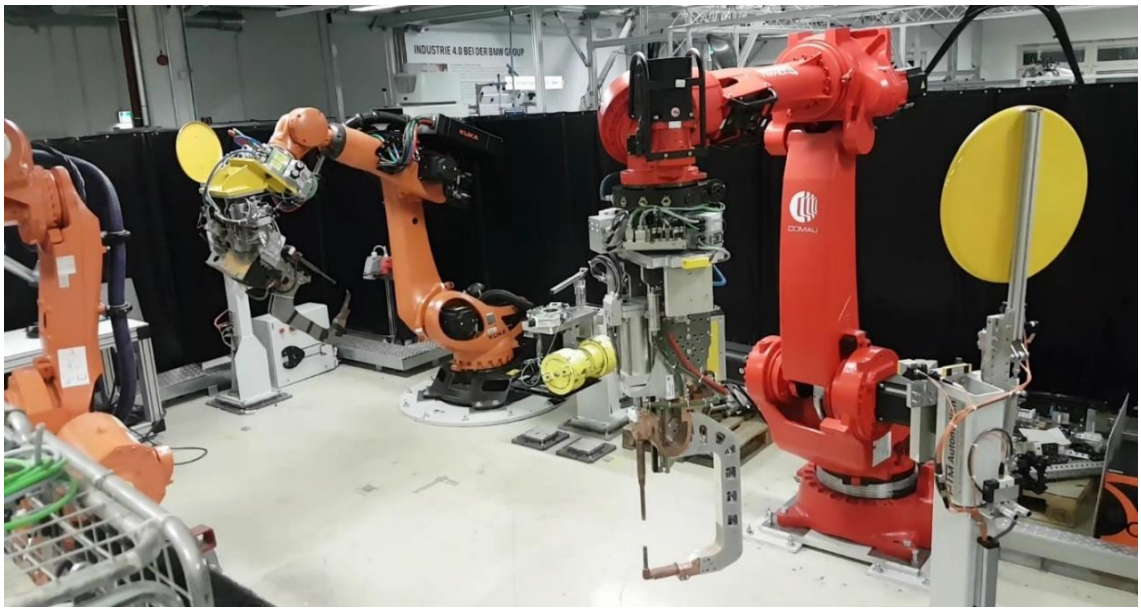


Abbildung 77 – Roboter im Testbetrieb einer Punktschweißapplikation mit Werkzeugwechsler

7.3.6. Evaluierung

Vergleicht man die in den vorangegangenen Abschnitten beschriebene Umsetzung und Ergebnisse des Prototyps mit den aus Anwendersicht identifizierten Anforderungspunkten, zeigt sich, dass diese erfolgreich adressiert und erfüllt wurden (Tabelle 13).

Tabelle 13 – Anforderungen und deren Realisierung im Prototyp

Anforderung	Unterpunkte	Realisierung der Anforderung
Synthese der vorherigen Arbeitspakete	• Externer Ansteuerung der Robotersysteme • Applikationssoftware in einer GPL • Anwendersoftware über graphische Benutzeroberfläche	• Externe Steuerung über Interpolationsschnittstelle • C++ Programmierung in RL und ROS auf Basis eines Linux-Systems • Spezifische Programmsteuerungsumgebung über Linux-System
Hoher Praxisbezug und Aussagekraft für den automobilen Karosseriebau	• Auswahl repräsentativer Applikationen für den gesamten Karosseriebau • Reale Industriekomponenten des Karosseriebaus • Mindestens zwei Robotersysteme unterschiedlicher Hersteller	• SPS, Punktschweißen, Werkzeugwechsler, Klebesteuerung identifiziert • Siemens S7, Schweißzangen, PTM-Werkzeugwechsler und -Bahnhof, Intec-Klebesteuerung • ABB-, Comau- und Kuka-Robotersystem
Integrierbarkeit mit der realen Zellen-steuerungsebene	• Verwendung des BMW-Standards für SPS <-> Roboter-Kommunikation • Automatikbetrieb von extern gesteuerten Robotersystemen via SPS	• BMW L7-Zellenstandard verwendet • Hybride Ansteuerung der nativen und externen Steuerung realisiert
Valides Sicherheitskonzept	• Zentrale Sicherheitsmechanismen müssen gewährleistet sein, um eine Gefährdung von Personen auszuschließen	• Verwendung des klassischen BMW-Zellenstandards • DGUV-Richtlinien abgeglichen und Schwachstellen identifiziert
Roboterhersteller-unabhängige Umsetzung der Applikationsstandards	• Herstellerunabhängige Anbindung der Peripherie • Herstellerunabhängige Programmierung der Basislogik • Herstellerunabhängige Bedienung auf Anwenderebene	• Schweißzangen, Werkzeugwechsler und -Bahnhof, Klebesteuerung über ProfiNet angebunden • Applikationslogik zur Kommunikation und Prozessablauf in C++ umgesetzt • Graphische Programmier- und Teach-Umgebung mittels RL implementiert
Leistungsfähigkeit	• Genauigkeit und Performance der Manipulatoren • Schaltgenauigkeit • Paralleler Betrieb von Robotersystem und zugehörigen Peripheriekomponenten soll möglich sein	• Nachweis über Lasertrackermessungen • Nachweis über Lasertrackermessungen mit Schalttrigger • Synchroner Ablauf mit Robotersystemen und Peripherie realisiert
Erweiterbarkeit und Vergleichbarkeit der Testzelle	• Vorhalt für die Integration weiterer HW- und SW-Komponenten, um Untersuchungen zu alternativen steuerungstechnischen Konzepten zu ermöglichen • Parallele Umsetzung der klassischen Applikationsstandards in den Robotersystemen	• ROS/ROS Industrial Anbindung und B&R-Steuerungstechnik integriert • Umsetzung der Applikation in den nativen ABB-, Comau-, Kuka-Systemen • MXAutomation Ansteuerung integriert

Auf Hardwareseite liegt eine Robotertestzelle mit drei Robotern unterschiedlicher Hersteller und der notwendigen Peripherie vor, um das klassische Karosserieumfeld repräsentativ wiederzugeben. Die Bewegungssteuerung der Robotersysteme wurde über Applikationsschnittstellen und externe Steuerungssysteme abstrahiert und das Gesamtsystem über eine Standardzellensteuerung und ein hybrides Ansteuerungskonzept für den Automatikbetrieb integriert. Ebenso konnten die zentralen sicherheitstechnischen Anforderungen auf Anlagenebene mit übernommen werden und das Optimierungspotenzial in Bezug auf den Handbetrieb identifiziert werden.

Nach erfolgreicher Umsetzung der notwendigen Infrastruktur konnten anschließend grundlegende Umfänge von Roboterapplikationsstandards in einem echtzeitfähigen und offenen Linux-System in objektorientierter statt prozeduraler und universeller statt proprietärer Programmiersprache implementiert werden. Für die Bediener- und Anwenderebene wurde eine graphische Programmier- und Bedienumgebung entwickelt, die ein roboterhersteller-unabhängiges Teachen, Parametrieren und Modifizieren von Roboterprogrammen ermöglicht.

Des Weiteren konnten anhand des Prototyps die zentralen Anforderungen in Bezug auf die notwendige Leistungsfähigkeit der extern gesteuerten Robotersysteme für sich und in Kombination mit der eingesetzten Peripherie erfolgreich überprüft werden. Zudem wurden weitere steuerungstechnische Alternativen und Schnittstellen implementiert, um aus Forschungs- und Projektsicht eine Bewertungs- und Testplattform für andere bzw. auf der vorliegenden Arbeit aufbauende Umsetzungskonzepte und Erweiterungen zu evaluieren.

7.4. Bewertung aus Gesamtunternehmenssicht

Die Evaluierung der prototypischen Umsetzung zeigt, dass die in Kapitel 4.6 formulierten Anforderungen erfolgreich in einer Testzelle synthetisiert werden konnten und eine herstellerunabhängige Integration von Robotersystemen in den Karosseriebau möglich ist. Überträgt man die im Rahmen dieser Arbeit aufgezeigten Potenziale auf die Gesamtorganisation eines Automobilunternehmens, ergeben sich eine Reihe von Vorteilen bei Beschaffung, Planung und Verwendung von Robotersystemen, aber auch neue Anforderungen und Risiken.

Einkauf

Die beschriebenen Schnittstellen zur externen Ansteuerung erlauben die Integration einer unabhängigen externen Steuerung und somit die Modularisierung des Robotersystems in einen hardware- (Manipulator und Bewegungssteuerung) und einen anwendungsorientierten Teil (Applikations- und Bedienumfänge). Die Grenzen können hierbei zwar je nach Schnittstelle variieren, sind aber aufgrund der vorgenommenen Einteilung in verschiedene Ebenen klar vorgegeben und differenzierbar. Diese Aufteilung eröffnet aus Sicht des Einkaufs neue Perspektiven, da das Robotersystem, welches bisher nur als Komplettlösung erhältlich ist, auf diese Weise zweigeteilt und somit grundsätzlich von unterschiedlichen Herstellern erwerbbar wird. Zum einen erhöht dies die potenzielle Anzahl an Lieferanten, da diese jeweils auch nur einen Teil bedienen können, statt ein vollständiges Robotersystem mit sämtlichen Anforderungen liefern zu müssen. Zum anderen kann das Robotersystem flexibler zusammengestellt und bezogen und so auf individuelle Anforderungen zugeschnitten werden. In Summe ist daher mit mehr Wettbewerb und passenderen Kundenlösungen zu rechnen. Die Flexibilisierung sollte dabei jedoch nicht nur als Nachteil für etablierte Systemlieferanten und Vorteil für neue Steuerungslieferanten betrachtet werden. Sie eröffnet etablierten Roboterherstellern ebenso die Möglichkeit, neue Anwendungsfelder zu erschließen, da beispielsweise der isolierte Manipulator oder die isolierte Steuerung in Bereichen eingesetzt werden kann, in denen das jeweilige Teilsystem im Gegensatz zum Gesamtsystem die Kundenanforderungen erfüllt.

Planung

Aus Sicht der Produktionsplanung überschneiden sich einige Vorteile mit denen des Einkaufs. So bietet die flexible Zusammenstellung neue Möglichkeiten der technischen Gestaltung von eingesetzten Robotersystemen. Vorstellbar wäre eine flexiblere Auswahl der Mechanik in Abhängigkeit von Leistung, Einsatzort oder Lieferverfügbarkeit, z. B. der Einsatz von leistungsstarken Mechaniken für Prozesse mit hohen Anforderungen und geringer Taktzeitvorgabe und die Verwendung von Low-Cost-Manipulatoren für Anwendungen mit niedrigerem Qualitätsanspruch. Ebenso sind Local-Sourcing-Konzepte vorstellbar, die es erlauben, die schwere Manipulator- und Antriebstechnik von einem dem Automobilwerk nahegelegenen Lieferanten zu beziehen. Dies bietet Vorteile sowohl in Bezug auf Nachhaltigkeit als auch hinsichtlich der Ersatzteilverfügbarkeit.

Realisierbar werden die beschriebenen Ansätze insbesondere durch den für die Produktionsplanung zentralen Vorteil bei der Integration von Robotersystemen – der Verwendung eines übergreifenden und herstellerunabhängigen Applikationsstandards. Aus steuerungstechnischer Sicht muss der bisher redundant durchzuführende Integrationsprozess dann nur noch einmal mit einer offenen Steuerung und einer einheitlichen Softwareplattform durchgeführt werden. Die relevante Anwender- und Applikationssoftware kann unabhängig von eingesetzten Lieferanten entwickelt und wiederverwendet werden. Neben der grundsätzlichen

Reduktion von Aufwänden durch die Vermeidung von Doppelarbeit birgt dies den Vorteil, Prozessinnovationen schneller in das Produktionsnetzwerk integrieren zu können, da die parallele Befähigung und Abstimmung mit mehreren Herstellern entfällt. Die Wiederverwendung von Applikationsstandards ist dabei nicht nur bei der Betrachtung von Robotersystemen unterschiedlicher Lieferanten nützlich, sondern auch in Bezug auf den Robotergenerationswechsel eines einzelnen Herstellers, da die bisher notwendige Reimplementierung von Prozesslogik und Bedienstandards nicht mehr erforderlich ist.

Auf Ebene der Applikationsentwicklung bietet die Verwendung allgemeiner Hochsprachen statt proprietärer Robotersprachen neben der übergreifenden Wiederverwendbarkeit zudem Potenziale in der Entwicklungswerkzeugkette. So stehen aufgrund der breiten Verwendung der Sprachen Tools aus der klassischen Softwareentwicklung zur Verfügung, die in den Standardisierungsprozess eingebunden werden können. Dies können beispielsweise Werkzeuge zur Codeanalyse oder der Versionsverwaltung sein. Mit gleichen Effekten ist ebenso bei der Festlegung auf eine universelle industrieübergreifende Roboterprogrammiersprache zu rechnen, da Aufwände für individuelle Anpassungen wegfallen und sich der potenzielle Abnehmermarkt aus der Perspektive der Toolhersteller vergrößert.

Produktion

Die universelle Entwicklung der Applikationsstandards bietet ebenso Vorteile beim Einsatz von Robotersystemen im Werk. So erleichtert die herstellerunabhängige Gestaltung von Bedien- und Programmierkonzepten die Durchführung von instandhaltungsrelevanten Aufgaben und reduziert die damit verbundenen Aufwände. Auch beim Einsatz unterschiedlicher Robotersysteme können notwendige Bedientätigkeiten einheitlich erfolgen. Mitarbeiter müssen sich nicht umstellen und sind routinierter im Umgang mit den Systemen. Fehler können schneller behoben und Optimierungen zügiger durchgeführt werden. Eine unter Umständen vorhandene Hemmschwelle, Robotersysteme zu bedienen, kann durch eine anwendungsorientierte und einfache Gestaltung des Bedienkonzeptes reduziert werden. Darüber hinaus können Schulungsaufwendungen verringert werden, da auch bei einem Lieferantenwechsel Bedienung und Programmierung der neuen Roboter unverändert bleiben. Des Weiteren gestattet die Modularisierung des Robotersystems die Implementierung neuer Konzepte der Wiederverwendung von Anlagentechnik im Werk. Es besteht die Möglichkeit, Robotermanipulatoren über eine längere Laufzeit zu betreiben, da notwendige steuerungstechnische Neuerungen unabhängig vom Manipulator durch die externe Steuerung umgesetzt werden können. Ebenso ist die einfachere Einführung neuer Applikationen in die bestehende Werksstruktur möglich, da kein aufwändiges Befähigen eines kompletten Altsystems, bestehend aus Robotermechanik und Steuerung, erfolgen muss, sondern über den Austausch oder dem Update der anwendungsrelevanten Umfänge in der externen Steuerung erfolgen kann.

Erfordernisse, Einschränkungen und Risiken

Neben den genannten Vorteilen beim Einsatz der vorgeschlagenen herstellerunabhängigen Steuerungslösung existieren jedoch auch Restriktionen und negative Implikationen, die ebenso berücksichtigt werden müssen, z. B. neue Aufwände und Abhängigkeiten, die durch die Einführung der externen Steuerung auftreten können. So ermöglicht die Verwendung externer Schnittstellen die Gestaltung von Applikationsstandards unabhängig vom Roboterlieferanten, wobei diese ebenso in einer alternativen Umsetzungsplattform implementiert werden müssen. Je nach Grad der Offenheit dieser Plattform können neue Verflechtungen zu einem Lieferanten

entstehen. Diese Abhängigkeiten fallen im Vergleich zum Status quo indes geringer aus, da sie sich nur auf einen Teil und nicht auf das gesamte Robotersystem beziehen. Gegensteuern kann man dieser Problematik mit der, wie am Prototyp gezeigten Verwendung einer Open-Source-Steuerung. Dabei kann die Implementierung von Funktionalitäten notwendig sein, die in den Systemen von Roboter- und Steuerungslieferanten bereits vorhanden sind. Der Nutzen des Einsatzes eines externen Steuerungssystems steigt folglich erst mit der Anzahl eingebundener Robotersysteme unterschiedlicher Lieferanten, da die Anwendung bei nur einem einzelnen Hersteller lediglich eine Verschiebung der Aufwände von Roboter- zur externen Steuerung darstellt. Ein weiterer Aufwandsfaktor ist zudem die Einbindung der Robotersysteme in die externe Steuerung, da aufgrund unterschiedlicher Implementierungen derzeit für jeden Schnittstellentyp eine eigene Umsetzung erforderlich ist. Für eine effiziente Anwendung sind daher die übergreifende Vereinheitlichung der Schnittstellenspezifikationen und die Verwendung von Plug & Produce-Konzepten notwendig. Hierfür ist die Mitarbeit der etablierten Roboterhersteller unabdingbar. Ebenso ist dies für die Sicherstellung der Gewährleistung notwendig, da die externe Steuerung in Abhängigkeit der Integrationstiefe signifikanten Einfluss auf das Gesamtsystem besitzt und eine klare Abstimmung zwischen den Lieferanten erfordert und zum Teil auch neue Absicherungsprozesse notwendig sein können. Diese Unterstützungsleistung der Roboterlieferanten kann nicht als automatisch gegeben angesehen werden und muss daher als Risiko ausgewiesen werden.

Ein weiteres, zu diskutierendes Erfordernis stellt der Umgang mit Open-Source-Software im steuerungstechnischen Umfeld dar. Zum einen sind lizenzrechtliche Belange zu berücksichtigen, die sich aus der Verwendung von quelloffener Software ergeben können. Zum anderen sind Erwartungen an deren Qualität und die Sicherstellung dieser zu beachten. So könnten beispielsweise ein intensiveres Testen oder die Einbindung zusätzlicher Entwicklungspartner notwendig sein, um die erforderliche Qualität der Software zu gewährleisten, welche nicht mehr direkt von einem verantwortlichen Lieferanten über den herkömmlichen Beschaffungsprozess bezogen wird. Außerdem ist zu bedenken, dass sich der Nutzen von Open-Source-Konzepten mit ihrem Verbreitungsgrad und der steigenden Anzahl an Anwendungen und Nutzern erhöht. Eine offene Steuerungsplattform und die darauf aufbauenden Applikationsstandards würden insbesondere von einer industrieübergreifenden Verwendung profitieren. Aus Sicht eines Automobilunternehmens ist daher zu diskutieren, ob die Offenlegung der notwendigen Erweiterung beim Einsatz einer Open-Source-Lösung notwendig und sinnvoll ist, um die Verbreitung auf diese Weise zu fördern. Gleiches gilt für Applikationsstandards und -programme. Insbesondere bei etablierten Standardprozessen wie dem des Punktschweißens ist zu überdenken, ob eine technisch offene Basisplattform dazu genutzt werden sollte, Prozesswissen nicht nur roboterhersteller-, sondern auch anwenderübergreifend zu definieren und zu implementieren.

8. Zusammenfassung und Ausblick

Die Integration von Industrierobotersystemen gestaltet sich auch nach über 50 Jahren des Robotereinsatzes in der Automobilindustrie weiterhin als langwierig und ressourcenintensiv. Die Komplexität der Systeme erfordert fachspezifisches und geschultes Personal – sowohl während der Inbetriebnahme als auch im täglichen Produktionsbetrieb. Die proprietäre Gestaltung von Softwarearchitektur, Bedienkonzepten, Programmiersprachen und Schnittstellen erschwert zudem den Einsatz von Robotersystemen unterschiedlicher Hersteller, verhindert die herstellerübergreifende Wiederverwendung von entwickelter Software und führt zu Herstellerabhängigkeiten in Form von Vendor Lock-in und Single-Sourcing-Effekten. Die Heterogenität und die Anzahl von Ansätzen für die offene, nutzergerechte und flexible Gestaltung von Robotersystemen belegen, dass die geschilderte Problematik nicht lediglich die Automobilindustrie, sondern generell jeden Anwender von Robotersystemen betrifft. Trotz der Vielzahl an Lösungsvorschlägen in Form von Normierungen, Produkten und Forschungsarbeiten fokussiert sich keiner auf das Anwendungsfeld des Karosseriebaus, obwohl dieses angesichts der Anzahl eingesetzter Robotersystemen und -applikationen das Haupteinsatzgebiet von Industrierobotern darstellt und ein spezifisches Anforderungsprofil besitzt. Die vorliegende Arbeit hat diese Betrachtungslücke aufgegriffen und liefert einen vierteiligen Beitrag zur herstellerunabhängigen Integration von Industrierobotersystemen in den Karosseriebau anhand folgender Umfänge am Fallbeispiel der BMW AG: Anforderungsableitung für die Roboterintegration, herstellerunabhängige Steuerung von Robotersystemen, hersteller-unabhängige Programmierung von Robotersystemen, und die kombinierte Umsetzung und Bewertung an einem Prototyp.

Im ersten Schritt wurden die Vorgehensweise bei der Roboterintegration in der Automobilindustrie und die damit verbundenen Herausforderungen in Bezug auf die Herstellerabhängigkeit betrachtet. Hierfür wurde der Einsatz von Robotersystemen in der Automobilproduktion organisatorisch und technisch analysiert und die zentrale Bedeutung des Karosseriebaus als Anforderungsgeber dargestellt. Darauf aufbauend konnten die Aufgaben und die daraus resultierenden Anforderungen der Prozessbeteiligten erfasst und für die Bearbeitung der weiteren Themenschwerpunkte abgeleitet werden.

Im Rahmen der herstellerunabhängigen Steuerung von Robotern wurde die Frage beantwortet, auf welche Art und Weise Robotersysteme extern angesteuert werden müssen, um die Anwendungslogik vom Herstellersystem zu entkoppeln und sowohl die technischen als auch die organisatorischen Anforderungen bestmöglich umzusetzen. Aufgrund der Vielzahl unterschiedlicher Schnittstellenkonzepte und Umsetzungen wurde zunächst eine Kategorisierung auf Basis der Integrationstiefe von Schnittstellen vorgenommen, bevor eine Beschreibung der entsprechenden Charakteristika sowie der Vor- und Nachteile erfolgte. Zudem wurden die sich durch den Einsatz der externen Steuerung ergebenden Aufwände und Verantwortungen erläutert. Anhand der Schnittstellengruppierung in Applikations-, Bahnplanungs-, Interpolations- und Antriebsebene wurde anschließend ein Bewertungsmodell für die Einsatzeignung von externen Steuerungskonzepten im Karosseriebau entwickelt und die Ansteuerung auf Interpolationsebene als geeignetste Schnittstelle identifiziert, um Robotersysteme unabhängig von einem Herstellersystem zu integrieren. Die Frage der konkreten Leistungsfähigkeit und Erfüllung der fertigungstechnischen und prozessualen Anforderungen in Verbindung mit einer Open-Source-Steuerung wurde anschließend im Rahmen von Lasertrackermessungen mit zwei Robotern unterschiedlicher Hersteller bearbeitet.

Grundlage dieser Versuche war die ISO-Norm 9283, welche jedoch durch spezifische Anforderungen des Karosseriebaus modifiziert und um Messungen zur Bestimmung der Schaltgenauigkeit erweitert wurde. Anhand der Ergebnisse konnte aufgezeigt werden, dass die Leistungsfähigkeit beim Einsatz von Interpolationsschnittstellen mit einer Open-Source-Steuerung grundsätzlich auf Höhe der nativen Robotersysteme liegt, die fertigungstechnischen Grundanforderungen von Karosseriebauanwendungen erfüllt und zusätzlich Vorteile in der herstellerübergreifenden Standardisierung von Robotersystemen bietet. Abschließend wurden Defizite und Verbesserungsvorschläge bei der Einbindung von Robotersystemen über externe Schnittstellen aufgezeigt und ein Plug & Produce-Konzept entwickelt. Dieses systematisiert die Ansteuerung und Datenbereitstellung mittels unterschiedlicher und fest definierter Integrationsebenen und ermöglicht so eine schnelle, flexible und automatisierte Konfiguration von externen Steuerungslösungen und herstellerunabhängigen Steuerungsarchitekturen.

Mit der Entkopplung der Anwendungslogik ergeben sich neue Freiheiten und Möglichkeiten in der herstellerunabhängigen Programmierung von Robotersystemen. Die damit verbundene Fragestellung, welche Programmiermethode und -sprache aus Anwendersicht des Karosseriebaus am besten geeignet ist, wurde daher ebenso nachgegangen. Hierfür wurden zunächst mögliche Roboterprogrammiermethoden betrachtet und die textuelle Programmierung als passendste Methode zur Entwicklung eines herstellerunabhängigen Programmierkonzeptes für Industrieroboter bestimmt. Zur Ermittlung des aktuellen Status quo in der Industrierobotik und des Anforderungsbildes aus Karosseriebausicht wurde anhand gängiger Industrierobotersprachen eine Datenbank aufgebaut und mit der Befehlsverwendung und dem Bedarf in Automobilwerken des Fallbeispiels automatisiert abgeglichen. Die Analyse ergab, dass kein eindeutiges Anforderungsbild ableitbar ist, da infolge des Integrationsprozesses in der Automobilindustrie eine arbeitsteilige Roboterprogrammentwicklung durchgeführt wird und sich sowohl die Programmstruktur als auch die Befehlsverwendung zwischen den Nutzergruppen unterscheiden. Mit Roboter- und Applikationsexperten auf der einen und Instandhaltungsmitarbeitern auf der anderen Seite existieren unterschiedliche Bedarfe bei Programmentwicklung und -verwendung. Dementsprechend wurde im darauffolgenden Anforderungsabgleich mit herstellerunabhängigen Programmiersprachen aus den Bereichen Robotik, Automatisierungstechnik und Softwareentwicklung ersichtlich, dass sich eine einzelne Sprache nur schwer als alleinige nutzergerechte Alternative eignen kann. Aus diesem Grund wird mit der Automotive Industrial Robot Language ein hybrides Programmiersprachenkonzept vorgeschlagen, das aus einer domänenspezifischen Roboterprogrammiersprache und einer zugehörigen objektorientierten Hochsprachen-API besteht und so den Belangen der zentralen Anforderungsgeber des Karosseriebaus, sowohl im Programmierparadigma als auch bezüglich des Funktionsumfangs, gerecht wird.

Im abschließenden Teil der Arbeit erfolgte die Synthese der vorangegangenen Analysearbeit von Anforderungsbild, Steuerungstechnik und Programmierung in Form einer prototypischen Umsetzung sowie deren Bewertung. Auf Basis der AIRL-Spezifikation wurde mit dem RobotCell Commander ein Tool zur herstellerunabhängigen, anwendergerechten und einfachen Programmierung und Anpassung von Roboteranwendungen für Mitarbeiter der Instandhaltung entwickelt und die Vorteile gegenüber herkömmlichen proprietären Roboterbedienpanels aufgezeigt. Ebenso diente die AIRL-Spezifikation als Grundlage für die Erweiterung einer Open-Source-Softwareentwicklungsumgebung, um Roboterprogramme herstellerunabhängig sowohl in AIRL als auch in C++ auf Expertenebene entwickeln zu können. Der technische Nachweis, dass herstellerunabhängige Roboterapplikationen für den Karosseriebau anforderungsgerecht realisierbar sind, wird auf Basis einer prototypischen Roboterzelle durchgeführt. In dieser wurden

Applikationsstandards des Fallbeispiels der BMW AG sowohl in den nativen Robotersystemen als auch in einer herstellerunabhängigen Variante mit einer externen Robotersteuerung realisiert. Es konnte nachgewiesen werden, dass es technisch und anforderungsgerecht möglich ist, klassische Roboteranwendungen des Karosseriebaus alternativ zur redundanten Umsetzung in proprietären Steuerungssystemen auch universell in eine flexible, offene und herstellerunabhängige Steuerungsarchitektur zu implementieren. Die anwenderrelevante Applikations-, Bedien- und Programmlogik konnte von den proprietären Steuerungssystemen entkoppelt und damit aufgezeigt werden, wie die herstellerunabhängige Integration von Robotersystemen in den Karosseriebau leistungsgerecht realisierbar ist. Neben der technischen Bewertung der Umsetzung wurden weitere organisatorische Auswirkungen auf das Gesamtunternehmen diskutiert.

Die vorliegende Arbeit schafft eine notwendige Grundlage, um die herstellerunabhängige Integration von Robotersystemen in den Karosseriebau zu industrialisieren. Zusätzlich bietet sie eine Ausgangsbasis für die Bearbeitung weiterer Forschungsfragen innerhalb anderer Domänen oder dezidierter Schwerpunkte aus dem behandelten Themengebiet.

Auf Seite der Programmierung liefern das ermittelte Anforderungsprofil und das vorgeschlagene hybride Programmiersprachenkonzept Ansatzpunkte, um weitere Programmiermethoden für die relevanten Nutzergruppen zu evaluieren. So können die ermittelten Funktionsbedarfe genutzt werden, um Programmmodifikations- und Bedienungsanforderungen von Werksanwendern des Karosseriebaus tiefer zu analysieren und aufgabenorientierte Programmiersysteme sowie graphische Programmiermethoden konkret auf diese Anwender auszulegen. Die verwendete Methodik zur Identifikation relevanter Befehlsbedarfe, Nutzergruppen und des Programmaufbaus bei der Roboterprogrammierung kann, neben der Automobilindustrie, für Betrachtungen weiterer Roboteranwendungsfelder genutzt werden. So erscheint die Anwendung auch für die Domäne der Elektronikproduktion sinnvoll, da diese das zweithäufigste Einsatzgebiet von Industrierobotern darstellt. Auf diese Weise könnte das Anforderungsbild an die textuelle Programmierung weiter konkretisiert werden und dazu beitragen, sich sukzessive der herstellübergreifenden Vereinheitlichung von Roboterprogrammiersprachen kunden- und anwendungsseitig zu nähern.

Mit dem Leistungsfähigkeitsnachweis der steuerungstechnischen Konzepte ergibt sich die Chance, darauf aufbauend weitere technische Innovationen und Verfahren zu betrachten, beispielsweise die Verwendung von verteilten Systemen statt eines Industrie-PCs als externes Steuerungsmedium. Hierbei können Middleware-Technologien und Cloud-Infrastrukturen einbezogen werden, um weitere Komponenten in eine Roboterzelle zu integrieren. Ebenso können Strategien und Anforderungen ermittelt werden, die eine übergreifende skalierbare Steuerungsinfrastruktur für mehrere Roboterzellen oder für ein gesamtes Automobilwerk ermöglichen. Zudem bietet der entwickelte Prototyp einer Karosseriebauzelle mit offener Steuerungsarchitektur eine ideale Plattform, um weitere Arbeiten hinsichtlich der Industrialisierung voranzutreiben, z. B. mit Dauerlaufversuchen oder der Einbindung weiterer Fertigungsapplikationen. Die identifizierten Verbesserungsvorschläge innerhalb der Schnittstellenkonzepte und die entwickelte Plug & Produce-Methode sollten genutzt werden, um eine Vereinheitlichung und Vereinfachung der externen Steuerungsanbindungen sowohl anbieter- als auch kundenseitig voranzutreiben. Nur durch eine fortschreitende Reduzierung der Implementierungsaufwände und durch die Abkehr von Insellösungen können die Potenziale derartiger Steuerungskonzepte effektiv genutzt werden und eine weitere Verbreitung und die wirtschaftliche Anwendung erfolgen.

9. Anhang

9.1. Test der Posekenngrößen

9.1.1. Bahncharakteristik - Nativer Roboter und extern gesteuertes Robotersystem

Abbildung 78 zeigt die nativen Bahnen des Kuka- und des Comau-Roboters im Vergleich. Die Roboter fahren die Bahn mit maximal möglicher Beschleunigung und Geschwindigkeit ab. Es ist ersichtlich, dass beide Roboter die vorgegebenen Punkte anfahren. Aufgrund der ungleichen Kinematik der Manipulatoren und der herstellerspezifischen Bahnplanung ist die Bahn zwischen den Zielpunkten jedoch unterschiedlich. Zudem differieren auch Dauer des Durchlaufs sowie die Geschwindigkeit und die Beschleunigung am Tool Center Point. So fährt der Kuka-Roboter die Testbahn in 4,67 Sekunden ab, während das Comau-System 5,34 Sekunden benötigt. Die Geschwindigkeit am TCP des Kuka-Roboters beträgt im Maximum 2,1 m/s. Die des Comau-Systems 1,7 m/s.

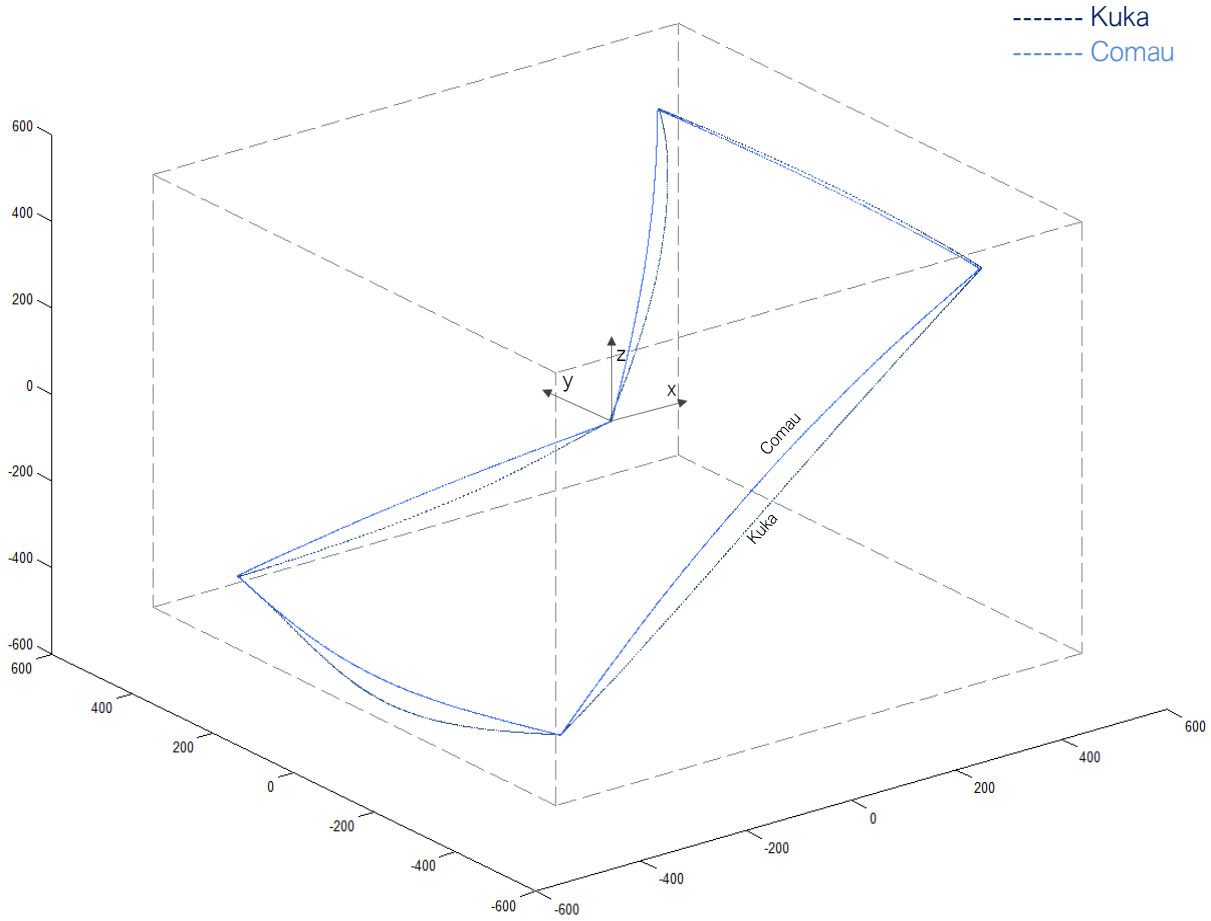


Abbildung 78 – Bahnverlauf Pose-Genauigkeitstest für einen nativen Kuka- und Comau-Roboter

In Abbildung 79 ist die vorangegangene Grafik um die Bahnen der extern angesteuerten Robotersysteme ergänzt. Es ist ersichtlich, dass der Bahnverlauf jeweils sehr nah an der nativen Robotertrajektorie verläuft, jedoch mit Versatz in negative Z-Richtung. Dies ist insbesondere bei den Comau-Roboterbahnen ersichtlich. Um eine bessere Betrachtung der Charakteristika zu ermöglichen, sind in Abbildung 80 und Abbildung 81 die Bahnverläufe an den Haltepunkten in vergrößerter Form für das jeweilige Robotersystem dargestellt. Zusätzlich wurden die nativen Roboterbahnen bei ausgeschalteter Absolutgenauigkeit ergänzt.

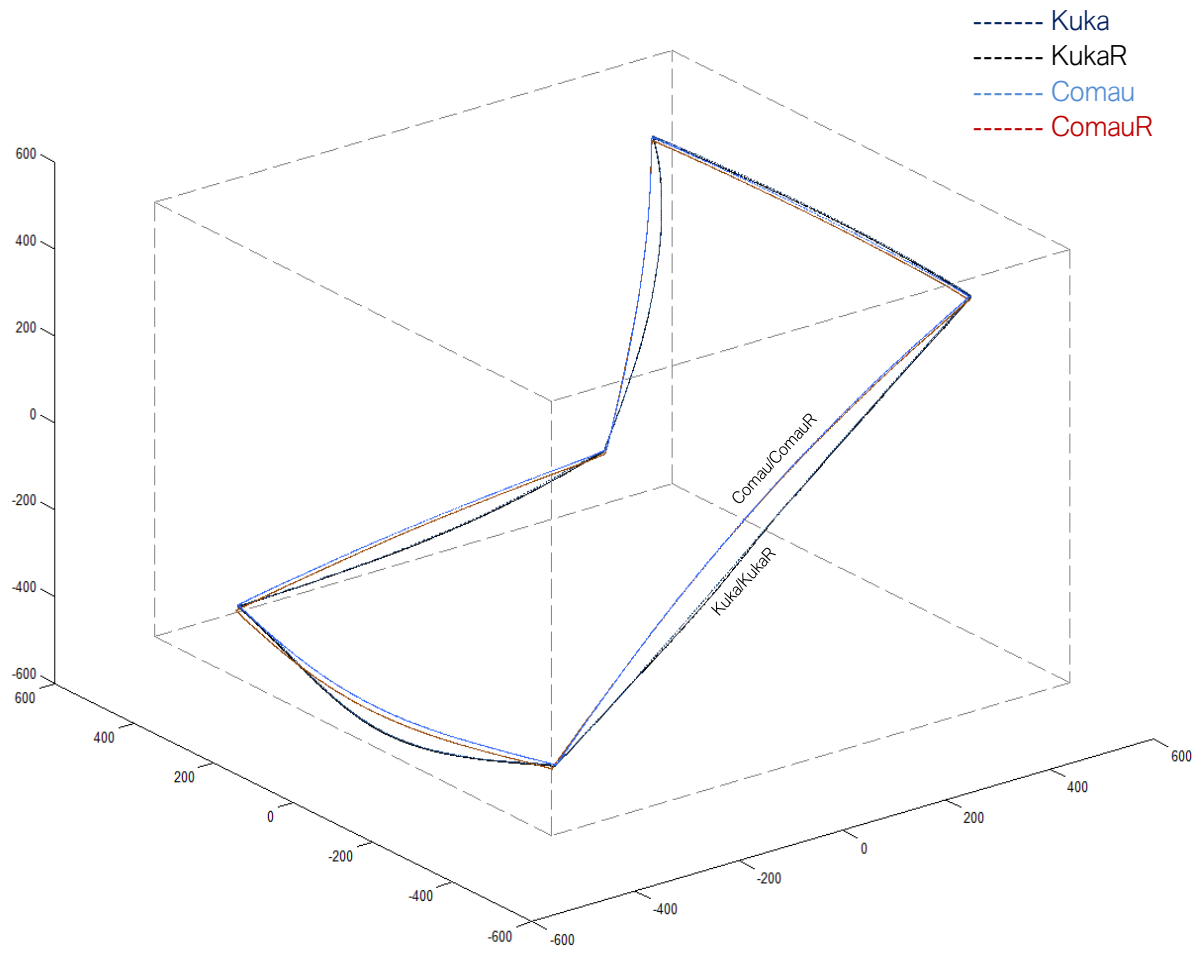


Abbildung 79 – Erweiterung von Abb. 78 um extern gesteuerte Roboterbahnen

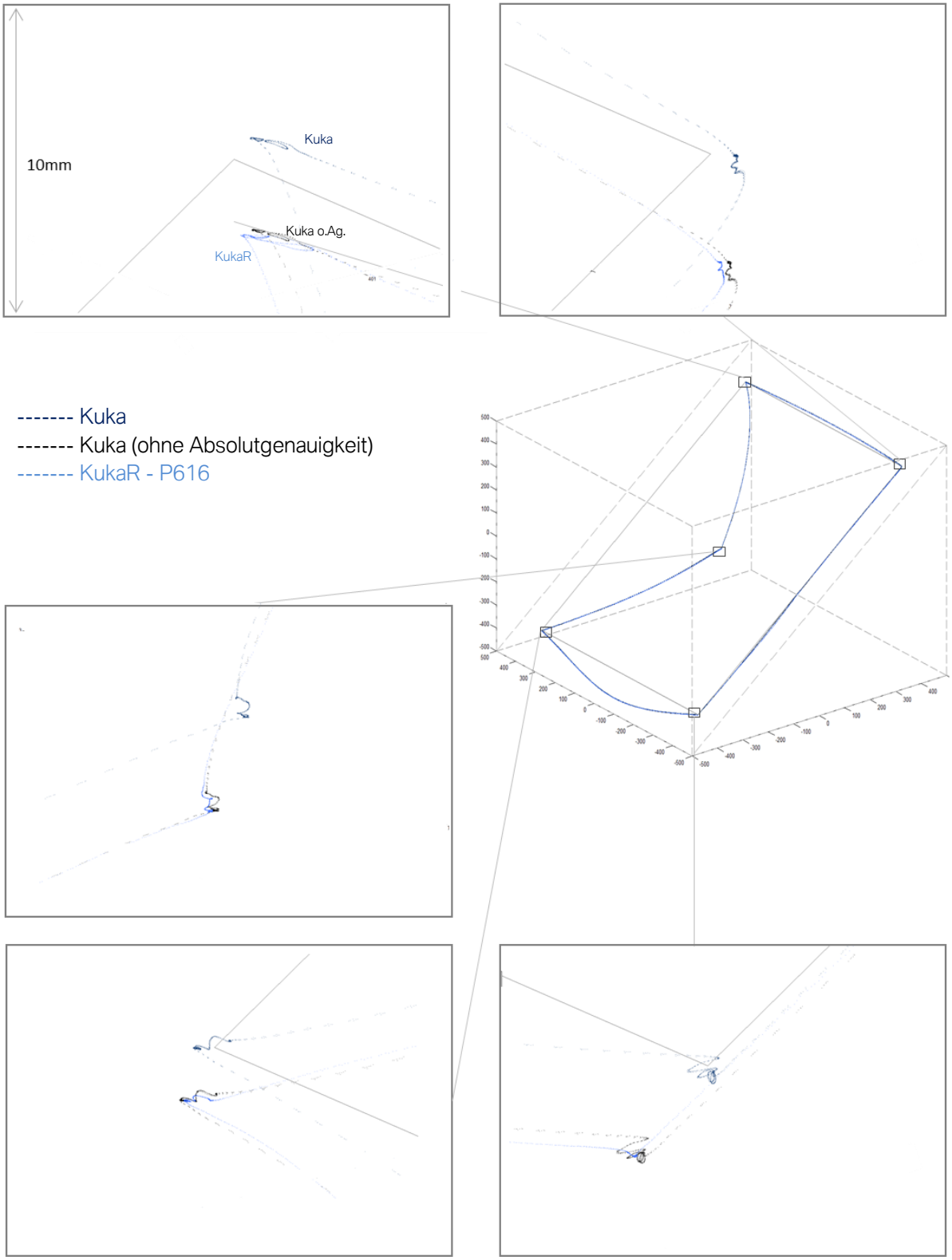


Abbildung 80 – Vergrößerte Darstellung der Kuka-Roboterbahnen (nativ mit und ohne Absolutgenauigkeit, extern)

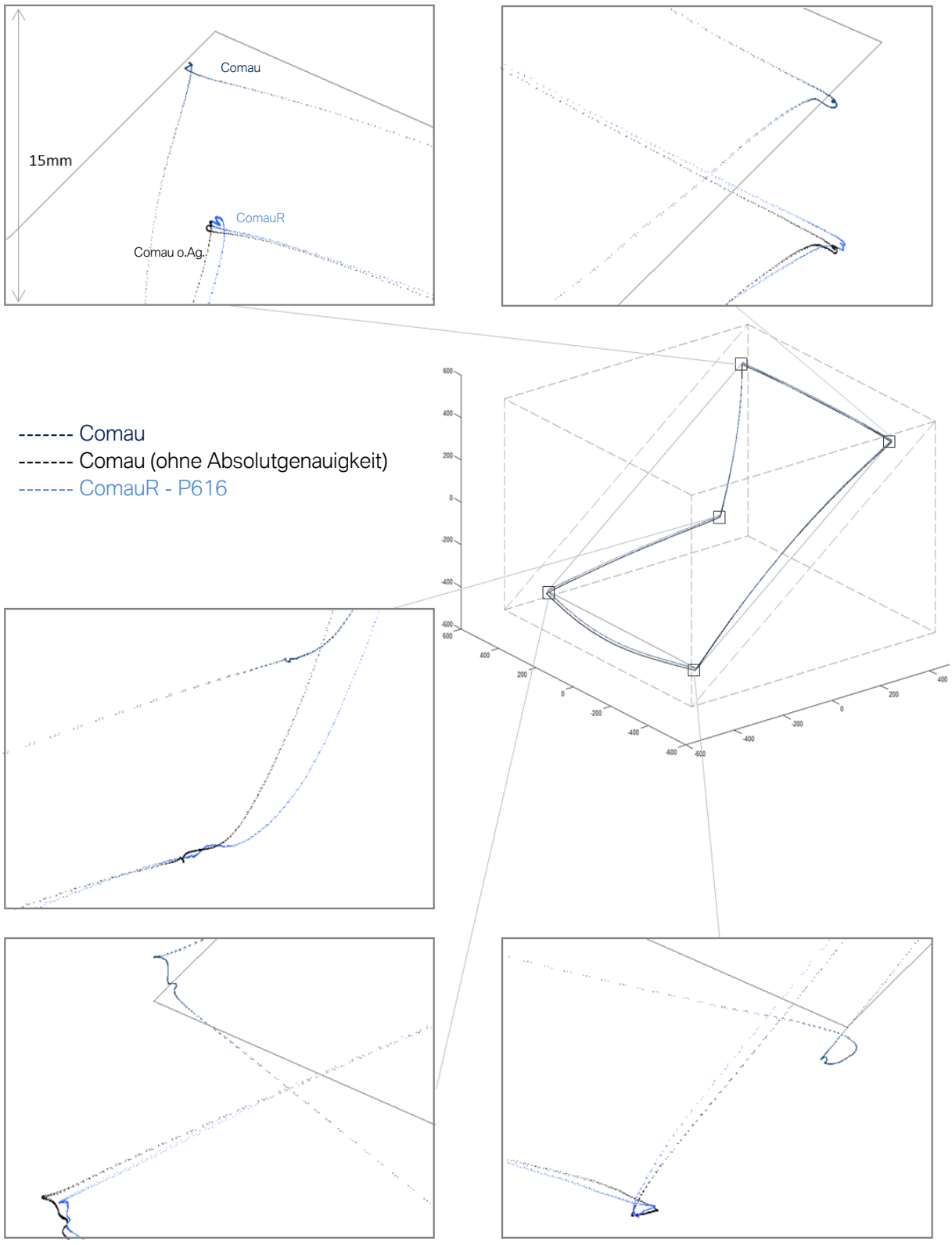


Abbildung 81 – Vergrößerte Darstellung der Comau-Roboterbahnen (nativ mit und ohne Absolutgenauigkeit, extern)

Aus den vergrößerten Grafiken wird deutlich, dass die nativen Bahnkurven bei ausgeschalteter Absolutgenauigkeit und die Roboterbahnen der externen Ansteuerung sehr nahe beieinander bzw. aufeinander liegen, während die Trajektorien der nativen, absolutgenauen Roboterbahnen einen Versatz primär in Z-Richtung aufweisen. Es lässt sich somit schlussfolgern, dass beide Robotersysteme keine Absolutgenauigkeitskorrektur bei der externen Ansteuerung durchführen. Dies erfolgt weder bei der kartesischen noch bei der achsbasierten Ansteuerung. Aus Endanwendersicht wäre diese Option wünschenswert. Die fehlende Absolutgenauigkeit zeigt sich auch am direkten Vergleich der Endlagen in den Haltepositionen der Testbahnen (Tabelle 14). Die Abweichungen zwischen den nativen Robotersystemen mit und ohne Absolutgenauigkeitskorrektur betragen 2,7 mm (Kuka) und 9,1 mm (Comau), während die Differenz zwischen den extern gesteuerten Bahnen und der nativen Roboterbahn ohne Absolutgenauigkeit lediglich 0,3 mm beträgt.

Bei diesen Vergleichswerten gilt zu beachten, dass Messsystem und -durchführung nicht auf die Erfassung von allgemeiner Genauigkeit, sondern lediglich von Wiederholgenauigkeiten eines Robotersystems ausgelegt wurden. Ein direkter Vergleich im niedrigen zehntel Millimeterbereich kann aus den vorliegenden Werten dementsprechend nicht abgeleitet werden. Es wird sich daher auf die angeführte Gegenüberstellung im Millimeterbereich beschränkt.

Tabelle 14 – Räumliche Abweichungen der Haltepunkte der Pose-Testbahn in mm

Haltepunkt / Vergleichsbahn	Pose 1	Pose 2	Pose 3	Pose 4	Pose 5	Ø
Kuka / Kuka (o.Ag.[24])	2,9	2,9	3,3	2,9	1,7	2,7
Kuka (o.Ag.) / KukaR	0,1	0,4	0,4	0,4	0,4	0,3
Comau / Comau (o.Ag.))	9,6	7,6	6,7	10,1	11,3	9,1
Comau (o.Ag.)) / ComauR	0,4	0,3	0,4	0,1	0,3	0,3

9.1.2. Bahncharakteristik - unterschiedliche Bewegungsprofile im externen Betrieb

Eine weitere zentrale Fragestellung bestand in der Beeinflussung und Auswirkung auf die reale Roboterbahn durch die Wahl unterschiedlicher Bewegungsprofile. Als Untersuchungsparameter standen die Profile P3, P5, P616 und P414 zur Verfügung. Abbildung 82 zeigt die gesamte Roboterbahn und die Posen 4 und 2 für den Kuka-Roboter im Detail. Die linke Vergrößerung zeigt jeweils das Interpolationsprofil P616 und P3. Auf der rechten Seite sind P616, P5 und P414 dargestellt.

Anhand der Darstellungen ist gut ersichtlich, welchen Einfluss die Wahl des Interpolations- bzw. Bahnplanungsalgorithmus auf die Charakteristik der TCP-Roboterbahn im Haltepunkt hat. Beispielsweise verursacht das Profil P3 durch seine starke positive und negative Beschleunigung insbesondere bei der Zielerreichung ein starkes Überschwingen, während die ruckfreie Beschleunigung im P616-Profil wesentlich weicher den Zielpunkt erreicht. Bei Betrachtung der unterschiedlichen Bahnen des Comau-Roboters in Abbildung 83 fällt auf, dass die Bahncharakteristik von P3 ebenfalls von P616 abweicht. P5, P414 und P616 sind indes wesentlich homogener ausgeprägt als dies bei dem Kuka-Robotersystem der Fall ist.

[24] o.Ag.: ohne Absolutgenauigkeit - Absolutgenauigkeit bei Roboter deaktiviert.

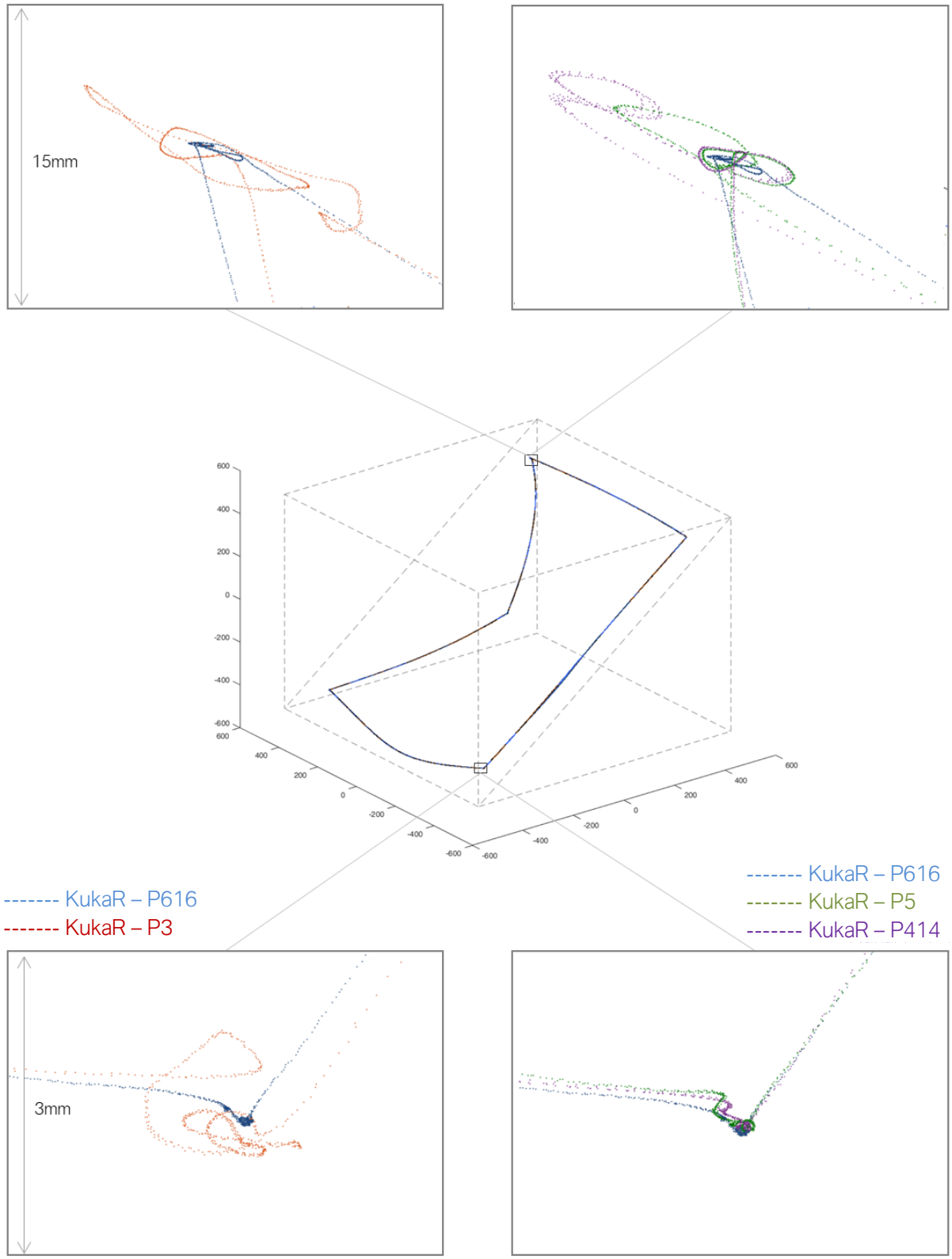


Abbildung 82 – KukaR-Roboterbahnen mit unterschiedlichen Bewegungsprofilen

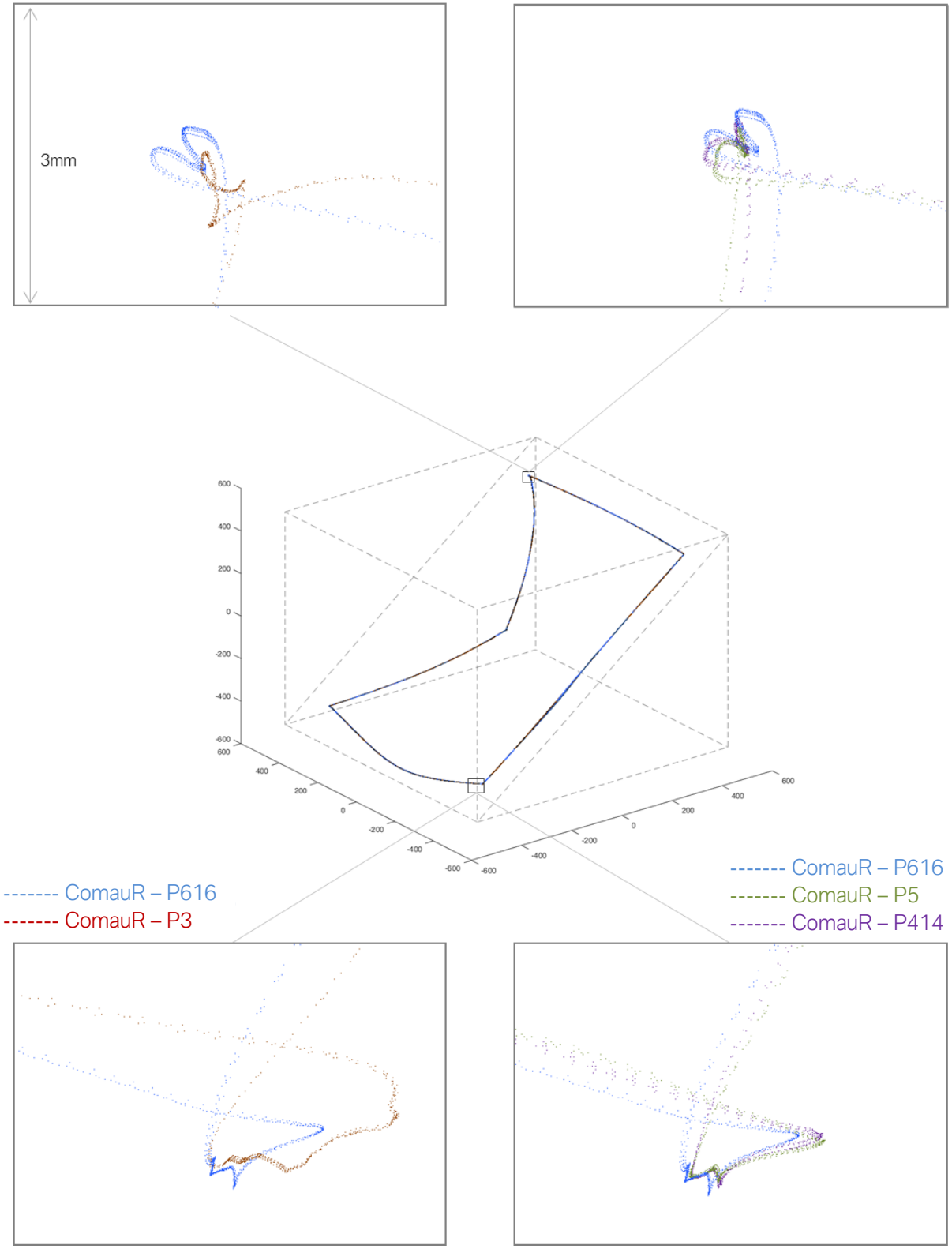


Abbildung 83 – ComauR-Roboterbahnen mit unterschiedlichen Bewegungsprofilen

9.1.3. Kennwerte

Zentrale Fragestellung bei der Kennwertbetrachtung ist, ob es mit den extern angesteuerten Robotersystemen möglich ist, Leistungswerte der nativen Robotersysteme zu erreichen. Als Kennwerte wurden daher die klassischen ISO-Zielgrößen Wiederholgenauigkeiten, Stabilisierungszeit und Überschwingungen als Einflussparameter auf die Prozessqualität untersucht. Zusätzlich wurden Kennwerte wie Geschwindigkeit und Beschleunigung als Einflussgrößen für die Taktzeit betrachtet. Für die Untersuchungen wurde eine Vielzahl an Messungen mit unterschiedlichen Eingangsgrößen hinsichtlich Geschwindigkeit, Beschleunigung, Beschleunigungsprofile, Wartedauer, Zykluszeit und Werkzeuggewicht durchgeführt. Im Folgenden wird aus Gründen der Übersichtlichkeit und Relevanz die Darstellung und Diskussion zunächst auf folgende Randbedingungen eingegrenzt.

- Das Messgewicht ist einheitlich 140 kg
- Die Wartezeit im Haltepunkt beträgt 3 Sekunden
- Die eingestellten Geschwindigkeits- und Beschleunigungswerte des nativen Systems stellen die Maximalgröße von 100 % dar
- Die Geschwindigkeit und Beschleunigung sind im Fall der externen Ansteuerung ebenso die maximal möglichen Werte. Die Bestimmung der maximalen Beschleunigungswerte wird aufgrund der fehlenden dynamischen Kennwerte iterativ bestimmt, wobei die Fahrtdauer in der Größenordnung des nativen Systems liegen sollte.
- Als Bewegungsprofile bei externer Ansteuerung wird sich auf P3- und P616-Randgrößen beschränkt, da diese die größten Unterschiede in der Bewegungsausprägung aufweisen
- Die ISO-Cube Kantenlänge beträgt einheitlich 1 Meter

TCP-Stabilisierungsprozess Comau

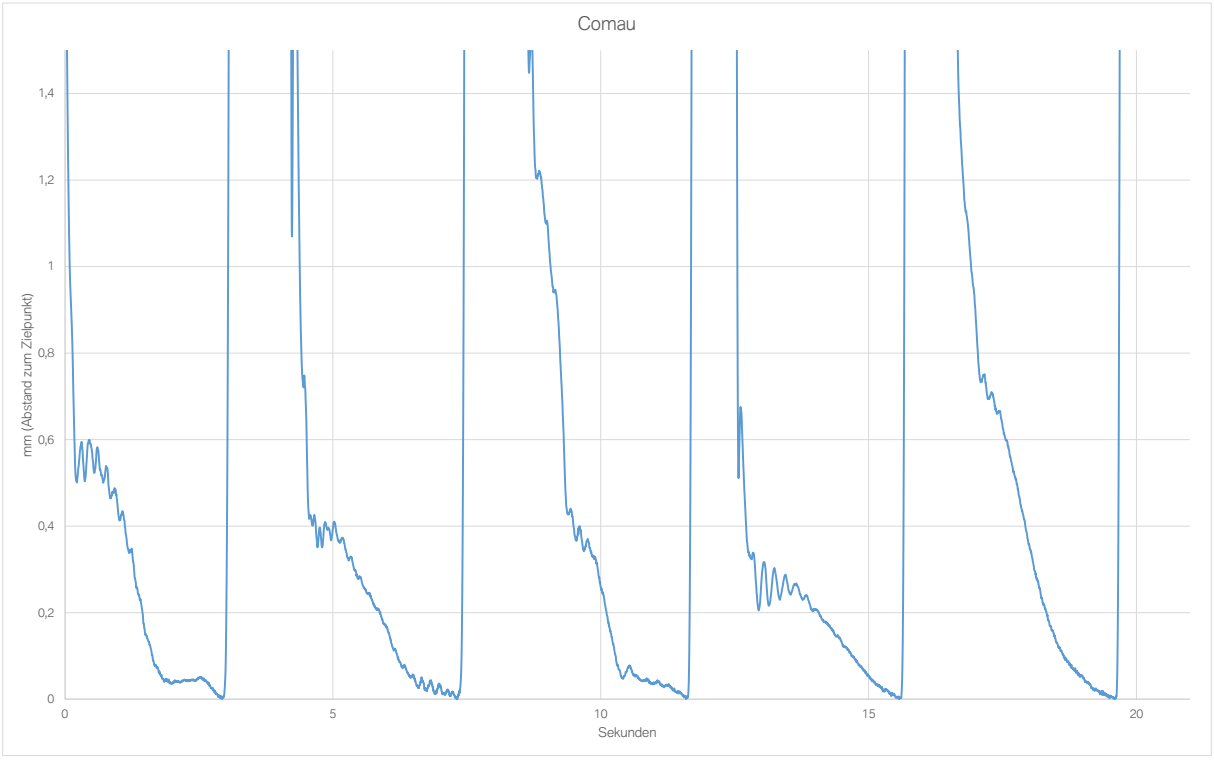


Abbildung 84 – Stabilisierungsprozess des nativen Comau-Roboters für eine ISO-Cube-Pose-Fahrt

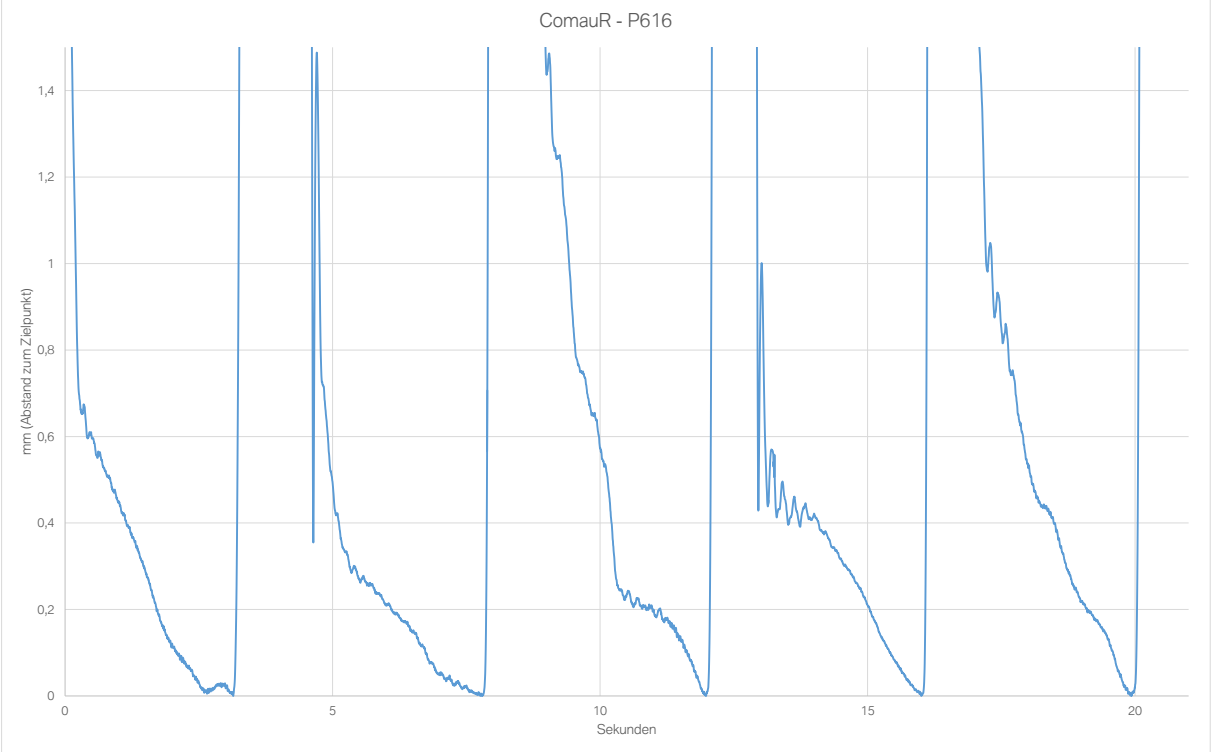


Abbildung 85 – Stabilisierungsprozess des Comau-Roboters im externen Betrieb mit P616-Bewegungsprofil für eine ISO-Cube-Pose-Fahrt

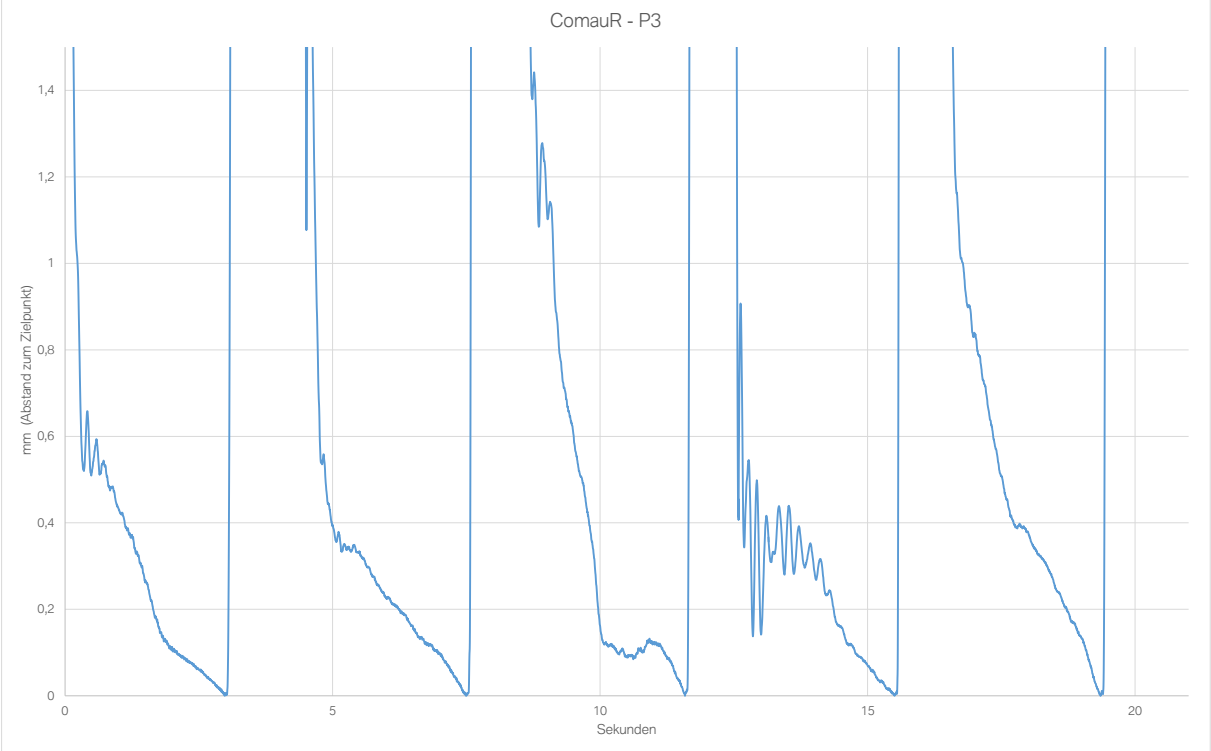


Abbildung 86 – Stabilisierungsprozess des Comau-Roboters im externen Betrieb mit P3-Bewegungsprofil für eine ISO-Cube-Pose-Fahrt

TCP-Geschwindigkeit und Beschleunigung

Auf Seite 182 sind die TCP-Geschwindigkeiten und -Beschleunigungen für die extern angesteuerten und nativ betriebenen Kuka- und Comau-Roboter im Zeitverlauf dargestellt. Position 0 der X-Achse definiert den Zeitpunkt 400 ms bevor der TCP den ISO-Cube-Mittelpunkt verlässt. Anschließend werden die einzelnen Zielposen angefahren und bei ca. 17 bis 18 Sekunden (Kuka) bzw. ca. 18 bis 19 Sekunden (Comau) der Ausgangspunkt wieder erreicht. Zur besseren Darstellung der Reproduzierbarkeit sind die Fahrten zu den ersten drei Posen des nachfolgenden Zyklus ebenfalls noch visualisiert.

Aus Abbildung 87 ist ersichtlich, dass die maximale TCP-Geschwindigkeit mit dem KukaR im P616-Interpolationsmodus bei 13,8 Sekunden erreicht wird. Die zweithöchste Geschwindigkeit wird im P3-Interpolationsmodus kurz vorher erreicht. Die native Roboterbahn ist im Vergleich zu dem Zeitpunkt etwas langsamer, jedoch zu den vorherigen Zeitpunkten bei 0-2 Sekunden und 4-6 Sekunden schneller. Die gesamte Bahn wird von KukaR - P3 mit 17,3 Sekunden geringfügig schneller als von der nativen Robotersteuerung mit 17,5 Sekunden abgefahren, während KukaR - P616 17,9 Sekunden benötigt. Ein schnelleres Abfahren im externen Steuerungsmodus war unter den gegebenen Bedingungen nicht möglich, da bei einer weiteren Erhöhung der Achsbeschleunigung Drehmomentfehler ausgelöst wurden und das Robotersystem in den Not-Halt versetzt wurde.

Analog zu den Geschwindigkeitswerten übertrifft das extern gesteuerte System die TCP-Beschleunigungen der nativen Steuerung zwischen 8 und 10 Sekunden bzw. zwischen 12 und 14 Sekunden (Abbildung 88), allerdings beschleunigt das Bewegungsprofil P3 dabei am stärksten. In den Zeitpunkten davor verhält sich die Beschleunigungsverteilung wiederum umgekehrt und die native Bahn übertrifft die externe Ansteuerung.

Ein ähnliches Bild zeigt sich bei den TCP-Geschwindigkeits- (Abbildung 89) und Beschleunigungswerten (Abbildung 90) des Comau-Robotersystems. Die Beschleunigung ist im Profil P3 und P616 mit beinah 8 m/s² bzw. 7 m/s² um ein Vielfaches größer als der native Roboter mit 3,5 m/s zu diesem Zeitpunkt zwischen 9 und 10 Sekunden. In den restlichen Bereichen sind die Beschleunigungswerte in ähnlicher Größenordnung.

Sowohl in Bezug auf das Einschwingverhalten als auch die erreichbaren Geschwindigkeiten und Beschleunigungen ist anzumerken, dass die Robotersysteme durch die veränderten Bewegungsprofile zwar Verbesserungen in isolierten Kriterien (z.B. Taktzeit) aufweisen, allerdings parallel hierzu andere Kriterien (z.B. Stabilisierungszeit) negativ beeinflusst werden. Die Gesamtleistung des extern gesteuerten Systems kann daher nicht direkt als verbesserte Steuerungsleistung im Vergleich zum Herstellersystem angesehen werden. Eine derartige Auslegung der Bahncharakteristik ist ebenfalls durch das Herstellersystem selbst möglich, jedoch steht die Möglichkeit, die Bahnplanung auf Interpolationsebene anwenderseitig im nativen Robotersystem derart zu modifizieren, nicht zur Verfügung. Die dargestellte Beeinflussung der Leistungskriterien des Robotersystem ist daher als Verbesserung der Parametrierbarkeit und damit eine verbesserte Auslegung des Gesamtsystems auf anwendungsspezifische Anforderung zu sehen und nicht als Verbesserung der Leistungsfähigkeit des Robotersystems an sich.

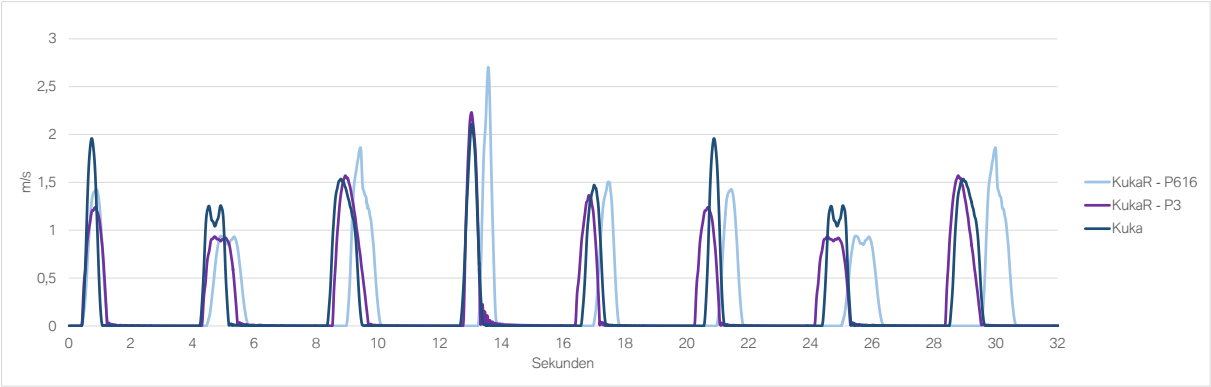


Abbildung 87 – TCP-Geschwindigkeit des nativ und extern gesteuerten Kuka-Roboters

Abbildung 88 – TCP-Beschleunigung des nativ und extern gesteuerten Kuka-Roboters

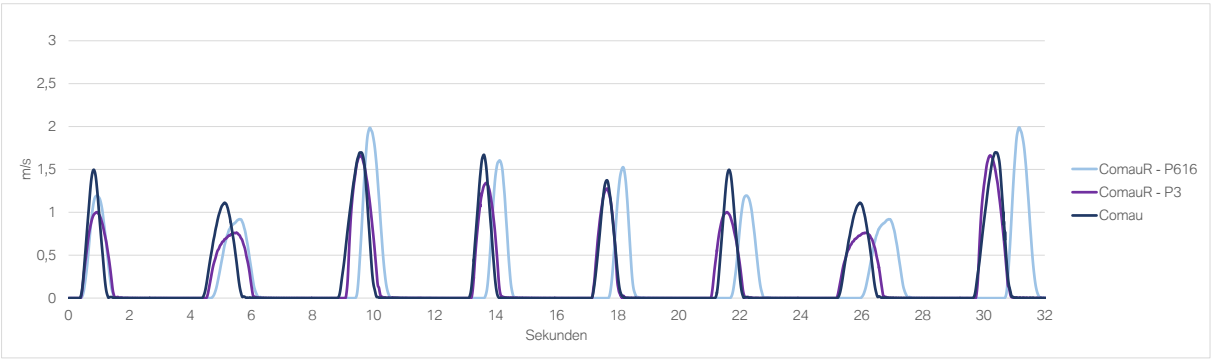


Abbildung 89 – TCP-Geschwindigkeit des nativ und extern gesteuerten Comau-Roboters

Abbildung 90 – TCP-Beschleunigung des nativ und extern gesteuerten Comau-Roboters

9.1.4. Beeinflussung durch Zykluszeit der externen Schnittstelle

Als Grundlage für Messungen der Beeinflussung durch die Zykluszeit dient die ISO-Cube-Bahn für Pose-Genauigkeit mit den gleichen Prämissen des Wiederholgenauigkeitsversuchs, jedoch wird sich im Rahmen des aufgeführten Vergleichs auf die P616-Bahn fokussiert.

Die Abbildung 91 bis 96 zeigen die im vorangegangenen diskutierten Kennwerte TCP-Geschwindigkeit, Beschleunigung und Wiederholgenauigkeit für die extern gesteuerten Kuka- und Comau-Roboter mit unterschiedlicher Zykluszeit. Bei dem Kuka-Roboter ist insbesondere aus Grafik 1 und 2 ersichtlich, dass bezüglich Geschwindigkeit, Beschleunigung und Dauer keine Unterschiede sichtbar sind, da die Linien der unterschiedlichen Messungen fast exakt übereinander liegen. Die Zykluszeit der externen Schnittstelle hat also keinen sichtbaren Einfluss auf die Roboterbahn des TCP.

Anders verhält es sich bei den Messwerten des Comau-Roboters. Während die Interpolationszeit von 400 µs und 4 ms übereinander liegende Messreihen erzeugen, sind Geschwindigkeit und Beschleunigung bei 8 und 16 ms bereits deutlich abweichend. Dementsprechend verlängert sich auch die Fahrt für die Gesamtstrecke.

Die Ursache für diese Abweichungen bei höheren Zykluszeiten kann sowohl im nativen Robotersystem als auch im externen Steuerungssystem begründet sein, obgleich die Software des externen Systems weitestgehend identisch zur externen Ansteuerung des Kuka-Systems ist. Zentraler Unterschied besteht hierbei im Kinematikmodell und dem Modul des Robotertreibers in der externen Steuerung, welches sich je nach anzusteuerndem Roboter unterscheidet, während das Modul der Bahninterpolation für beide Roboter identisch ist. Die Untersuchung des Sachverhaltes der Abweichungen bei höheren Zykluszeiten stand allerdings nicht im Fokus und wurde nicht weiterverfolgt, da die prinzipielle Leistungsfähigkeit mit zweistelligen Zykluszeiten des Gesamtsystems bereits über das extern gesteuerte Kuka-System ausreichend nachgewiesen wurde.

Ein Unterschied in der Genauigkeit kann zudem weder bei dem Comau- noch beim Kuka-Roboter in Abhängigkeit zur Zykluszeit aus den Darstellungen interpretiert werden. Auch eine genaue Betrachtung der Leistungskennwerte in Bezug auf den Einschwingvorgang zeigt keine Beeinflussung durch die gewählte Zykluszeit bei dem Kuka-Roboter. Kennwerte der Mindestpositionierzeit, Überschwinganzahl und Größe bleiben unverändert.

Von Relevanz ist die Tatsache, dass bei der achsbasierten Ansteuerung mit 12 ms vereinzelt Geschwindigkeitsverletzungen im Singularitätsbereich des Kuka-Roboters festgestellt wurden, die bei einer 4 ms Ansteuerung nicht auffällig waren. Bei Singularitäten handelt es sich in der Robotik um spezifische Konfigurationen eines Roboterarms, bei denen es theoretisch unendlich viele Achsstellungen für eine kartesische Sollposition gibt. An diesen Punkten kann es passieren, dass sich Achsen auch nur bei sehr kleinen Änderungen einer vorgegebenen kartesischen Position sehr schnell bewegen und es zu einer Geschwindigkeitsverletzung kommt [165]. Da die Steuerungskonfiguration bis auf die Zykluszeit unverändert geblieben ist, wird die Ursache der beobachteten Problematik des Kuka-Roboters in einer roboterinternen Umrechnung der Sollpositionen im externen Betrieb mit einem 12 ms-Takt vermutet. Eine Erklärung könnte sein, dass das Robotersystem den empfangenen Achspositionswert zunächst in einen kartesischen Zielwert umrechnet und diesen anschließend wieder in eine Achsposition als Sollvorgabe im 4 ms-Interpolationstakt des Roboters umwandelt.

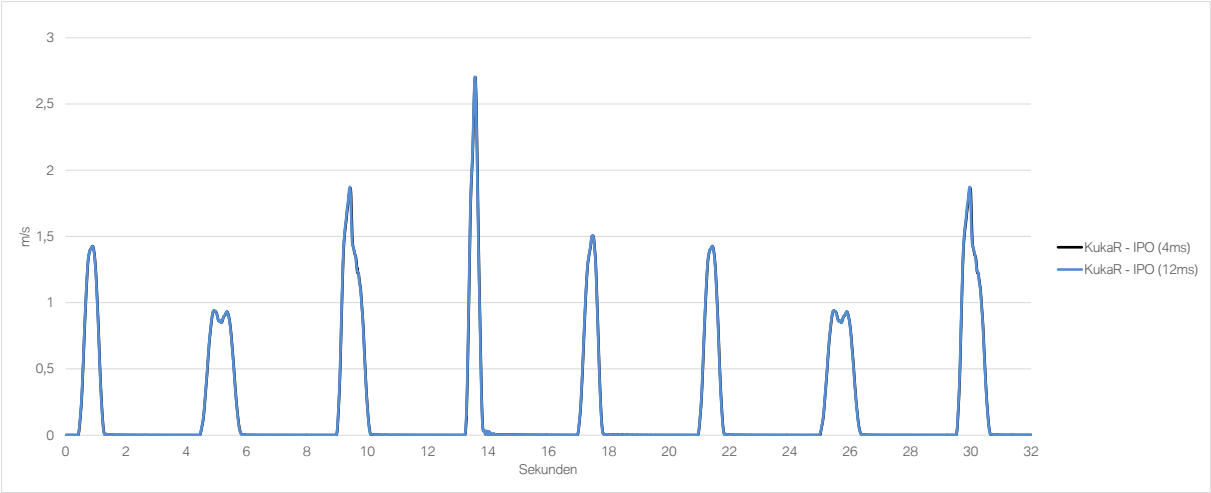


Abbildung 91 – TCP-Geschwindigkeit, im Vergleich bei unterschiedlicher Zykluszeit der externen Ansteuerung des Kuka-Roboters

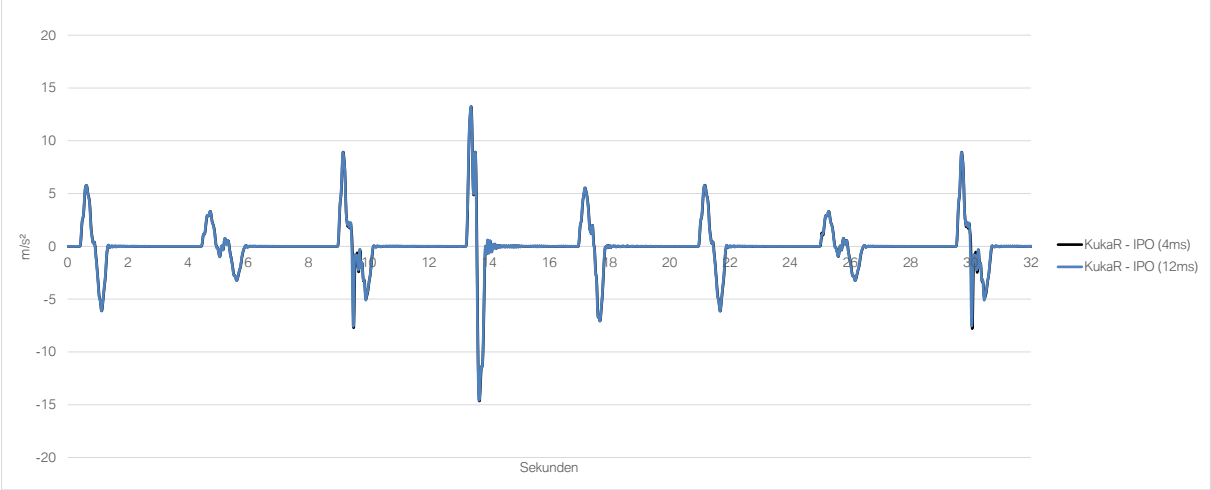


Abbildung 92 – TCP-Beschleunigung, im Vergleich bei unterschiedlicher Zykluszeit der externen Ansteuerung des Kuka-Roboters

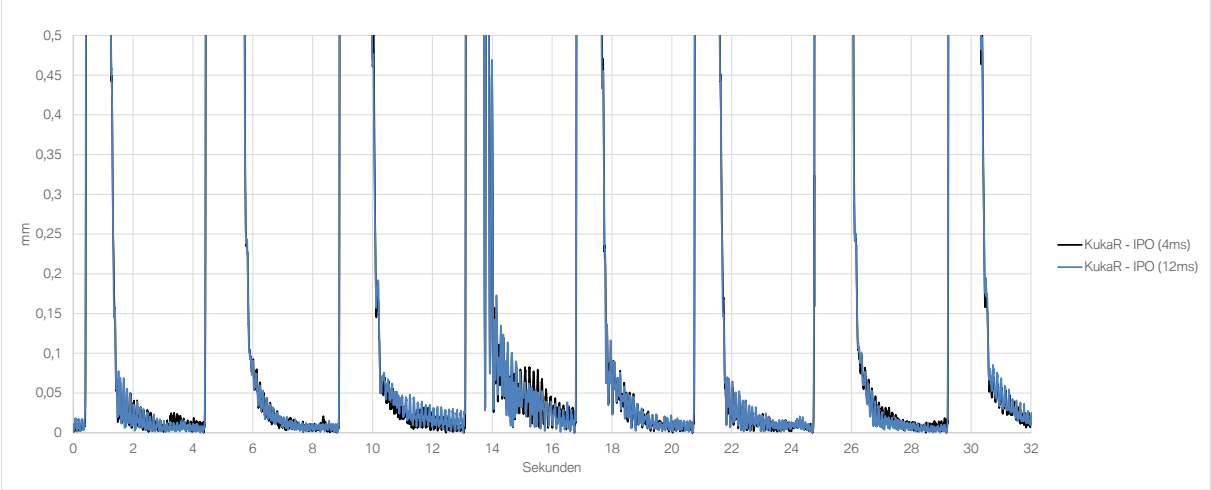


Abbildung 93 – TCP-Abstand zum Zielpunkt, im Vergleich bei unterschiedlicher Zykluszeit der externen Ansteuerung des Kuka-Roboters

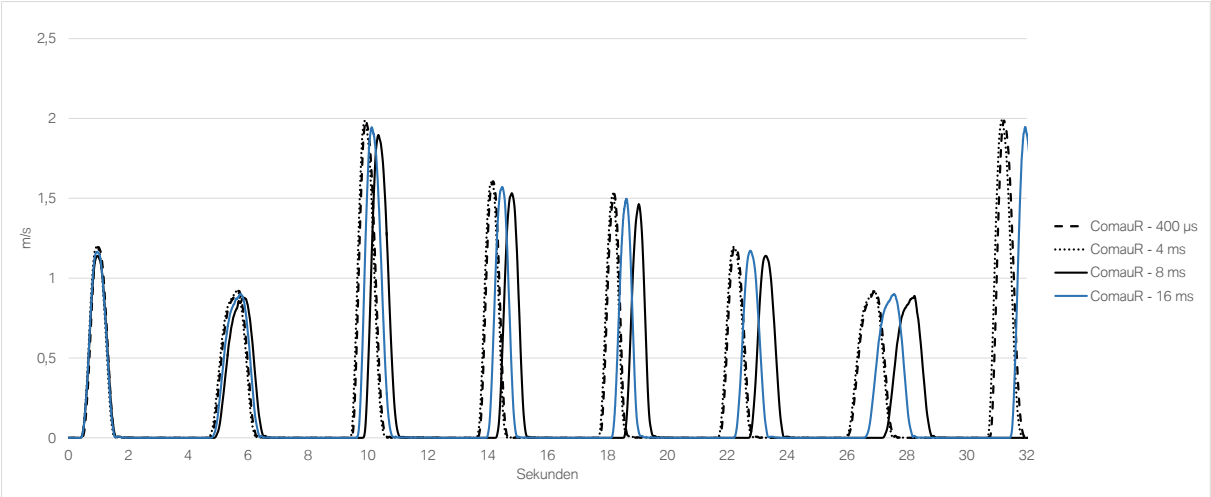


Abbildung 94 – TCP-Geschwindigkeit, im Vergleich bei unterschiedlicher Zykluszeit der externen Ansteuerung des Comau-Roboters

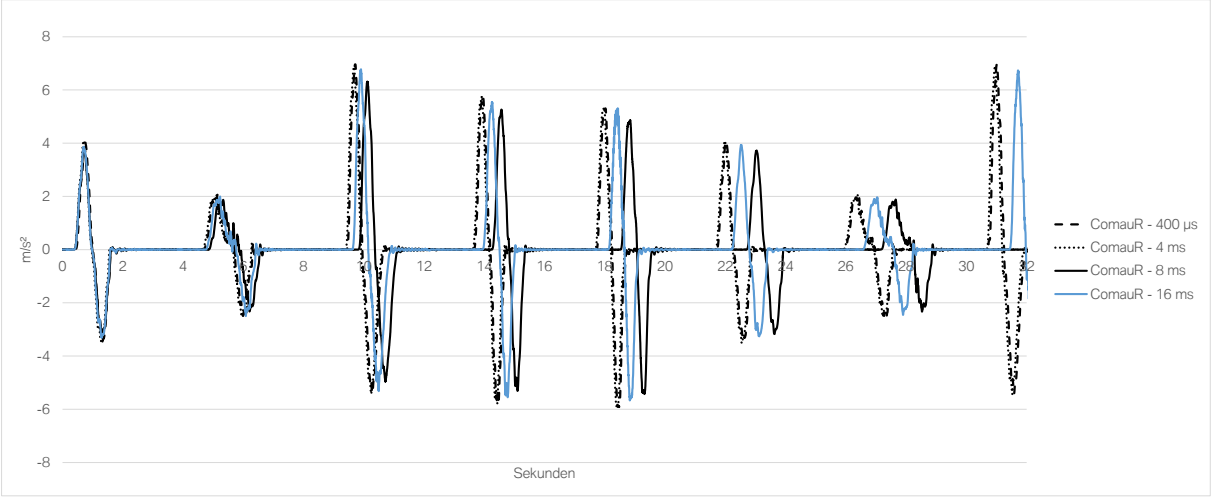


Abbildung 95 – TCP-Beschleunigung, im Vergleich bei unterschiedlicher Zykluszeit der externen Ansteuerung des Comau-Roboters

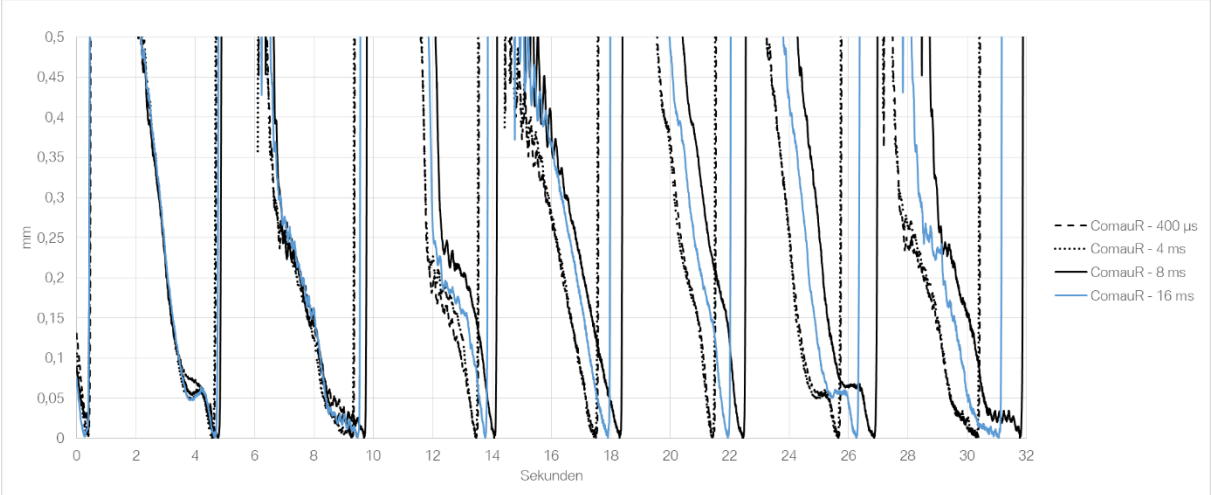


Abbildung 96 – TCP-Abstand zum Zielpunkt, im Vergleich bei unterschiedlicher Zykluszeit der externen Ansteuerung des Comau-Roboters

9.1.5. Messergebnisse im Überblick

Während in den vorangegangenen Abschnitten einzelne Ausschnitte der Untersuchungen für spezifische Kriterien aufgezeigt und diskutiert wurden, soll im Folgenden noch der übersichtliche Gesamtvergleich der Robotersysteme und der externen Ansteuerungskonzepte dargestellt werden. Für diese Messungen sind die gleichen Randbedingungen (Seite 179) gegeben, jedoch wurde die Haltezeit noch variiert, um den Einfluss auf die Wiederholgenauigkeit darzustellen. Da bei sehr niedrigen Haltezeiten die Stabilisierungsphase noch nicht abgeschlossen ist, aber trotzdem Genauigkeitsbewertungen durchgeführt werden sollten, wurde als zu vergleichende IST-Position diejenige gewählt, welche vor dem letzten Verlassen der Halteposition am nächsten an der Soll-Position liegt. Ebenso kann dadurch eine vergleichende Bewertung des Comau-Robotersystems durchgeführt werden, da die Einflussnahme auf der in Seite 77 dargestellten Regelungsbewegung reduziert ist.

Für jede Messreihe wurden drei hintereinanderliegende Messfahrten mit jeweils 30 Messpunkten durchgeführt. Die erfassten Leistungswerte sind die Durchschnittswerte dreier Messungen. Die Ergebnisse für den Kuka-Roboter sind in Tabelle 15, die des Comau-Roboters in Tabelle 16 dargestellt. Die Messabschnitte geben die jeweilige Haltezeit von 5, 3, 2, 1, 0.3 und 0 Sekunden wieder. Die Untersuchungsparameter stellen Roboter (nativ/extern), Bewegungsprofil (P3, P5, P414, P616), Zykluszeit (0,4 bis 16 ms), Haltezeit, Geschwindigkeit, Beschleunigung und aktive bzw. deaktivierte Absolutgenauigkeit dar. Geschwindigkeit und Beschleunigung wurden jeweils mit den höchstmöglichen Werten vorgegeben. Zusätzlich wurde bei den nativen Robotersystemen mit Haltezeiten von fünf und drei Sekunden die übergreifende Programmgeschwindigkeit (Programmoverride) soweit reduziert, dass die Gesamtbahn in einer etwa identischen Dauer zum externen P616-Bewegungspfad abläuft. Auf diese Weise soll eine bessere direkte Vergleichbarkeit der nativen und externen Systeme ermöglicht werden. Ergebnisgrößen sind die durchschnittliche Wiederholgenauigkeit (RP), die schlechteste Wiederholgenauigkeit einer Einzelmessung (RP max), die Dauer der Versuchsfahrt, die Geschwindigkeit (V-TCP) bzw. Beschleunigungsspitzen (A-TCP) der TCP-Position und die Wiederholgenauigkeiten der einzelnen Haltepositionen (RP-x).

Kernaussage der tabellarischen Übersicht ist, dass der zentrale Leistungskennwert von Robotersystemen — die Posewiederholgenauigkeit — bei der Ansteuerung durch externe Schnittstellen die gleiche Qualität wie die nativ betriebenen Robotersysteme erreicht. Jedoch hat die Wahl des Bewegungsprofils hierbei einen erheblichen Einfluss, da insbesondere das P616-Profil bessere Genauigkeitswerte bei Fahrten ohne Stillstand als die alternativen Profile erreicht. Gleiches gilt für die Dauer der Abfahrt der Testbahn, die durch Wahl des passenden Bewegungsprofils schneller als auch langsamer als die Referenzstrecke des nativen Systems durchfahren werden kann. Die Geschwindigkeits- und Beschleunigungskennwerte sind in zweierlei Hinsicht relevant. Zum einen übertreffen die Höchstwerte der externen Ansteuerungskonzepte die der nativen Systeme. Leistungsseitig scheint bei den alternativen Steuerungskonzepten also keine Einschränkung vorzuliegen. Zum anderen ist jedoch auch ersichtlich, dass die nativen Systeme effizienter die gleichen bzw. besseren Zeitwerte erreichen. Dies ist nicht überraschend, da die gewählten Bahnprofile exemplarische Bewegungsgesetz-Implementierungen sind und keinerlei Optimierung für die einsetzten Systeme vorgenommen wurde.

Im Vergleich zum Bewegungsprofil hat die Wahl der Zykluszeit der Ansteuerung bei dem Kuka-Roboter keinerlei Auswirkungen. Wiederholgenauigkeit, Geschwindigkeit, Beschleunigung und Stabilisierungscharakteristik blieben im Rahmen der Messungenauigkeit identisch. Anders

verhielt sich das Comau-Robotersystem, das wie im Vorangegangenen beschrieben, in Geschwindigkeit, Beschleunigung und Dauer variierte und zudem einen unruhigen Bewegungsablauf bzw. erhöhte Vibrationen auswies. Die Wiederholgenauigkeit verschlechterte sich dadurch aber nicht. Folglich zeigt sich, dass aus Punktprozesssicht ein Betrieb auch mit Zykluszeiten im zweistelligen Millisekundenbereich möglich ist und dass externe Steuerungssysteme nicht zwangsläufig auf einen sehr geringen Steuerungszyklus ausgelegt werden müssen. Dies kann sich für weitergehende Steuerungskonzepte von Vorteil erweisen, wenn Steuerungslogiken beispielsweise nicht vorberechnet und über verteilte Systeme oder Rechencluster bereitgestellt werden sollen, da die Zeitanforderungen nicht zwangsläufig im niedrigen Milli- oder Mikrosekunden Bereich liegen müssen.

Zusammenfassend lässt sich festhalten, dass die erreichbaren Wiederholgenauigkeiten mit den externalisierten Steuerungskonzepten in der gleichen Größenordnung wie die nativen Systeme liegen und, wie am Beispiel des Comau-Roboters zu sehen, antriebsspezifische Regelungsvorgänge einen vielfachen Einfluss auf die erzielbare Genauigkeit haben können. Aus qualitativer Fertigungsprozesssicht sind die untersuchten Steuerungslösungen also ausreichend gut, um ohne Genauigkeitsverlust produzieren zu können. Zudem zeigt sich im Rahmen der unterschiedlichen Bewegungsprofile und deren Auswirkung auf die Gesamtprozessdauer bzw. Stabilisierungscharakteristik, dass Leistungskennwerte beeinflusst und z. T. verschlechtert, aber auch verbessert werden können. Die Leistungsfähigkeit bzw. -charakteristik ist durch die externe Ansteuerung hochflexibel parametrierbar und kann auf Kundenanforderung in Abhängigkeit zum Fertigungsprozess noch besser ausgelegt werden. Bei relativ groben Handhabungsprozessen wäre es vorstellbar, Profile mit guten Geschwindigkeits-, aber schlechteren Stabilisierungseigenschaften zu wählen, während sensiblere Prozesse mit ruckfreien Bewegungscharakteristiken durchfahren werden könnten. Die Parametrierbarkeit der Roboterbahnen wäre neben der klassischen Vorgabe von Geschwindigkeits- und Beschleunigungswerten so noch detaillierter in Form von Geschwindigkeits- und Beschleunigungsprofilen festlegbar.

Als Defizit ist insbesondere das Fehlen von Absolutgenauigkeit bei der externen Ansteuerung zu nennen, das zwar keine direkten Auswirkungen auf die Fertigungsqualität besitzt, aber aus Inbetriebnahme und Wartungssicht zu erhöhten Aufwänden aufgrund von zusätzlicher Rekalibrierungsarbeit führt. Folglich wäre die Option, Korrekturen der Absolutgenauigkeit durch das Robotersystem auch bei der externen Ansteuerung durchzuführen, eine wünschenswerte Verbesserung aus Anwendersicht. Alternativ besteht die Möglichkeit eine individuelle Absolutkorrektur durch das externe Steuerungssystem umzusetzen. Vorteil wäre, dass diese unabhängig vom Roboterhersteller implementiert werden kann und eine bessere Anpassung an die einsatz- bzw. anwenderbezogene Randbedingungen ermöglicht und zeitgleich eine herstellerübergreifende Lösung darstellt, die vor allem aus Werkssicht den Einsatz unterschiedlicher Manipulatorhersteller vereinfachen würde.

Ein weiterer Nachteil zeigte sich in der Bewegungseffizienz der externen Systeme. Zwar konnten vergleichbare und schnellere Zykluszeiten wie mit dem nativen System erreicht werden, jedoch war dies auch mit höheren Geschwindigkeiten und Beschleunigen im TCP verbunden. Hier sind Optimierungen und Anpassungen auf die eingesetzte Hardware nötig, die im vorliegenden Versuch noch nicht durchgeführt wurden. Ebenso bietet sich die im Rahmen des Plug & Produce Konzepts (5.3.2) vorgeschlagene Integration und Verwendung mitgelieferter Bahnplanungsalgorithmen des Roboterherstellers an.

Tabelle 15 – Übersicht Messergebnisse Pose-Messungen Kuka-Roboter

Roboter	Bewegungs-profil	Zyklus-zeit	Halte-zeit	V (%)	A (%)	RP Ø (mm)	RP max (mm)	Dauer (s)	V-TCP (m/s)	A-TCP (m/s)	Temp (°C)	RP-1 (mm)	RP-2 (mm)	RP-3 (mm)	RP-4 (mm)	RP-5 (mm)
5 Sekunden																
Kuka			5	100	100	0,021	0,027	29,63	2,10	10,34	23,97	0,020	0,022	0,022	0,024	0,019
Kuka P90			5	90	90	0,028	0,034	30,02	1,91	8,74	23,17	0,024	0,027	0,029	0,032	0,027
KukaR	P616	4	5	100	100	0,021	0,025	30,07	2,67	12,94	23,80	0,019	0,024	0,022	0,018	0,022
3 Sekunden																
Kuka			3	100	100	0,027	0,037	19,67	2,10	10,62	23,06	0,024	0,029	0,024	0,038	0,025
Kuka P90			3	90	90	0,031	0,037	20,06	1,91	8,82	22,50	0,027	0,028	0,034	0,037	0,031
Kuka o.Ag.			3	100	100	0,028	0,034	19,67	2,10	10,49	22,74	0,024	0,034	0,030	0,028	0,027
KukaR	P616	4	3	100	100	0,025	0,033	20,06	2,69	13,08	23,02	0,023	0,033	0,025	0,024	0,021
KukaR	P616	12	3	100	100	0,025	0,037	20,07	2,68	13,07	22,97	0,025	0,037	0,024	0,021	0,020
KukaR	P3	4	3	100	100	0,026	0,033	19,37	2,22	14,34	22,91	0,023	0,025	0,027	0,025	0,033
KukaR	P3	12	3	100	100	0,039	0,028	19,37	2,19	14,90	22,91	0,023	0,026	0,022	0,022	0,025
KukaR	P414	4	3	100	100	0,025	0,035	19,76	2,70	12,28	23,04	0,023	0,035	0,023	0,018	0,025
KukaR	P5	4	3	100	100	0,020	0,026	19,87	2,70	13,38	23,04	0,018	0,023	0,026	0,020	0,019
2 Sekunden																
Kuka			2	100	100	0,033	0,068	14,63	2,10	10,53	23,97	0,025	0,069	0,022	0,025	0,028
KukaR	P616	4	2	100	100	0,026	0,035	15,07	2,69	13,14	23,58	0,025	0,035	0,028	0,023	0,022
KukaR	P3	4	2	100	100	0,026	0,033	14,37	2,23	14,52	23,38	0,029	0,021	0,022	0,030	0,028
1 Sekunde																
Kuka			1	100	100	0,033	0,043	09,65	2,11	10,55	23,65	0,035	0,029	0,028	0,043	0,034
KukaR	P616	4	1	100	100	0,030	0,039	10,06	2,70	13,16	23,96	0,028	0,038	0,031	0,026	0,027
KukaR	P3	4	1	100	100	0,030	0,045	09,37	2,23	14,71	23,25	0,030	0,023	0,034	0,042	0,026
0,3 Sekunden																
Kuka			0,3	100	100	0,027	0,033	06,17	2,11	10,55	22,98	0,022	0,028	0,030	0,031	0,027
Kuka o.Ag.			0,3	100	100	0,024	0,028	06,17	2,11	10,54	23,02	0,024	0,023	0,027	0,030	0,021
KukaR	P616	4	0,3	100	100	0,027	0,045	06,56	2,71	13,11	23,64	0,022	0,045	0,028	0,023	0,019
KukaR	P616	12	0,3	100	100	0,034	0,048	06,57	2,71	13,14	22,57	0,028	0,039	0,034	0,045	0,026
KukaR	P3	4	0,3	100	100	0,039	0,067	05,87	2,23	14,91	23,84	0,038	0,080	0,032	0,035	0,032
KukaR	P3	12	0,3	100	100	0,037	0,067	05,86	2,20	14,95	23,04	0,032	0,067	0,034	0,032	0,034
KukaR	P414	4	0,3	100	100	0,031	0,040	06,27	2,73	12,44	23,66	0,033	0,045	0,028	0,031	0,025
KukaR	P5	4	0,3	100	100	0,029	0,038	06,37	2,72	13,57	23,64	0,030	0,040	0,027	0,032	0,024
0 Sekunden																
Kuka			0	100	100	0,046	0,068	04,67	2,11	10,58	22,74	0,033	0,044	0,041	0,072	0,047
Kuka o.Ag.			0	100	100	0,043	0,065	04,67	2,11	10,56	22,74	0,031	0,051	0,038	0,066	0,034
KukaR	P616	4	0	100	100	0,051	0,109	05,06	2,72	13,22	22,98	0,040	0,119	0,048	0,036	0,027
KukaR	P616	12	0	100	100	0,048	0,097	05,07	2,71	13,24	22,98	0,032	0,113	0,044	0,040	0,029
KukaR	P3	4	0	100	100	0,057	0,084	04,37	2,25	14,91	22,74	0,111	0,177	0,056	0,099	0,086
KukaR	P3	12	0	100	100	Abbruch auf Grund Drehmomentenverletzung										
KukaR	P414	4	0	100	100	0,245	1,060	04,76	2,74	12,50	22,97	0,051	1,039	0,057	0,027	0,038
KukaR	P5	4	0	100	100	0,118	0,417	04,87	2,73	13,33	22,91	0,054	0,410	0,040	0,038	0,041

Legende:

V – Geschwindigkeit

A – Beschleunigung

RP – Positions-Wiederholgenauigkeit

RP max – Höchste RP einer Einzelmessung

RP-x – RP am Punkt x

TCP – Tool Center Point

Temp – Temperatur (Umgebung)

Tabelle 16 – Übersicht Messergebnisse Pose-Messungen Comau-Roboter

Roboter	Bewegungs-profil	Zyklus-zeit	Halte-zeit	V (%)	A (%)	RP Ø (mm)	RP max (mm)	Dauer (s)	V-TCP (m/s)	A-TCP (m/s)	Temp (°C)	RP-1 (mm)	RP-2 (mm)	RP-3 (mm)	RP-4 (mm)	RP-5 (mm)
5 Sekunden																
Comau			5	100	100	0,040	0,051	30,33	1,70	4,82	22,03	0,034	0,034	0,035	0,044	0,052
Comau P93			5	93	93	0,040	0,066	30,74	1,58	4,24	22,06	0,028	0,034	0,038	0,060	0,042
ComauR	P616	4	5	100	100	0,033	0,050	30,78	1,98	6,88	22,14	0,024	0,030	0,035	0,034	0,043
3 Sekunden																
Comau			3	100	100	0,040	0,064	20,34	1,70	4,82	22,07	0,025	0,039	0,029	0,062	0,044
Comau P93			3	93	93	0,043	0,067	20,75	1,58	4,26	22,06	0,033	0,043	0,036	0,054	0,047
Comau o.Ag.			3	100	100	0,037	0,051	20,34	1,70	4,79	22,06	0,031	0,036	0,031	0,033	0,050
ComauR	P616	0,4	3	100	100	0,039	0,053	20,78	1,98	6,85	22,56	0,033	0,053	0,027	0,044	0,036
ComauR	P616	4	3	100	100	0,036	0,044	20,79	1,98	6,83	22,45	0,032	0,035	0,041	0,034	0,036
ComauR	P616	8	3	100	100	0,036	0,042	21,79	1,89	6,25	22,42	0,029	0,036	0,038	0,040	0,035
ComauR	P616	16	3	100	100	0,036	0,048	21,29	1,94	6,66	22,50	0,028	0,042	0,044	0,034	0,036
ComauR	P3	4	3	100	100	0,032	0,045	20,17	1,66	7,71	22,43	0,027	0,032	0,032	0,031	0,041
ComauR	P414	4	3	100	100	0,033	0,043	20,17	2,16	6,79	22,50	0,027	0,044	0,035	0,027	0,031
ComauR	P5	4	3	100	100	0,030	0,039	20,08	2,08	6,89	22,43	0,024	0,028	0,032	0,031	0,036
2 Sekunden																
Comau			2	100	100	0,031	0,043	15,33	1,70	4,87	23,24	0,024	0,043	0,035	0,023	0,031
ComauR	P616	4	2	100	100	0,037	0,048	15,78	1,98	6,98	22,26	0,030	0,040	0,037	0,041	0,040
ComauR	P3	4	2	100	100	0,035	0,047	15,17	1,66	7,44	22,19	0,030	0,036	0,048	0,034	0,030
1 Sekunde																
Comau			1	100	100	0,035	0,048	10,34	1,70	4,89	23,24	0,023	0,048	0,037	0,032	0,034
ComauR	P616	4	1	100	100	0,033	0,039	10,78	1,98	7,03	22,59	0,034	0,038	0,027	0,031	0,036
ComauR	P3	4	1	100	100	0,037	0,051	10,16	1,66	7,48	22,43	0,032	0,045	0,045	0,029	0,036
0,3 Sekunden																
Comau			0,3	100	100	0,038	0,052	06,84	1,70	4,92	22,07	0,027	0,052	0,041	0,031	0,037
Comau o.Ag.			0,3	100	100	0,037	0,049	06,84	1,70	4,80	22,06	0,032	0,046	0,046	0,030	0,036
ComauR	P616	4	0,3	100	100	0,043	0,052	07,27	1,98	7,01	22,56	0,046	0,049	0,048	0,037	0,034
ComauR	P616	0,4	0,3	100	100	0,044	0,062	07,28	1,98	7,01	22,43	0,047	0,041	0,039	0,049	0,046
ComauR	P616	8	0,3	100	100	0,041	0,048	07,64	1,90	6,42	22,36	0,035	0,043	0,043	0,046	0,042
ComauR	P616	16	0,3	100	100	0,046	0,056	07,46	1,94	6,82	22,50	0,046	0,052	0,051	0,039	0,041
ComauR	P3	4	0,3	100	100	0,040	0,050	06,67	1,66	7,53	22,57	0,043	0,047	0,036	0,037	0,042
ComauR	P414	4	0,3	100	100	0,039	0,049	06,67	2,16	6,90	22,57	0,037	0,041	0,047	0,038	0,036
ComauR	P5	4	0,3	100	100	0,042	0,050	06,56	2,08	6,95	22,57	0,039	0,047	0,045	0,047	0,033
0,0 Sekunden																
Comau			0	100	100	0,078	0,104	05,34	1,70	4,92	22,07	0,104	0,059	0,084	0,063	0,086
Comau o.Ag.			0	100	100	0,071	0,090	05,35	1,70	4,78	22,04	0,084	0,083	0,082	0,045	0,089
ComauR	P616	0,4	0	100	100	0,066	0,097	05,78	1,98	6,97	22,49	0,078	0,056	0,060	0,041	0,099
ComauR	P616	4	0	100	100	0,073	0,105	05,78	1,98	6,94	22,56	0,081	0,068	0,066	0,047	0,107
ComauR	P616	4	0	100	100	0,065	0,096	05,28	2,16	8,34	22,07	0,084	0,058	0,053	0,045	0,091
ComauR	P616	8	0	100	100	0,065	0,082	06,06	1,91	6,37	22,42	0,072	0,057	0,070	0,048	0,081
ComauR	P616	16	0	100	100	0,068	0,091	05,92	1,95	6,80	22,50	0,090	0,050	0,074	0,049	0,084
ComauR	P3	4	0	100	100	0,277	0,424	05,17	1,66	7,73	22,59	0,422	0,341	0,356	0,152	0,119
ComauR	P414	4	0	100	100	0,091	0,155	05,17	2,17	6,85	22,56	0,153	0,083	0,075	0,048	0,106
ComauR	P5	4	0	100	100	0,103	0,182	05,07	2,09	7,17	22,59	0,181	0,102	0,099	0,050	0,087

Legende:
V – Geschwindigkeit
A – Beschleunigung
RP – Positions-Wiederholgenauigkeit
RP max – Höchste RP einer Einzelmessung
RP-x – RP am Punkt x
TCP – Tool Center Point
Temp – Temperatur (Umgebung)

9.2. Test der Bahnkenngrößen

9.2.1. Bahncharakteristik

Abbildung 97 zeigt die nativen Roboterbahnen des Kuka- und Comau-Roboters. Beide Roboter fahren die Diagonale mit möglichst gleichbleibender Geschwindigkeit ab. Der Ausgangspunkt wird über eine Punkt-zu-Punkt-Bewegung angefahren. Ähnlich wie in Testbahn 1 des Pose-Tests liegt die Rückfahrt auf unterschiedlichen Pfaden, während die Testbahnen selbst aufgrund der identischen Sollbahn übereinanderliegen.

Die Roboterbahnen des extern angesteuerten Systems sind in Abbildung 98 dargestellt. Im Gegensatz zur nativen Bahn wird die Rückfahrt bei beiden Roboter nicht über eine Punkt-zu-Punkt- sondern eine Linearfahrt durchgeführt. In der Grafiken wirken die Bahnen daher nahezu identisch.

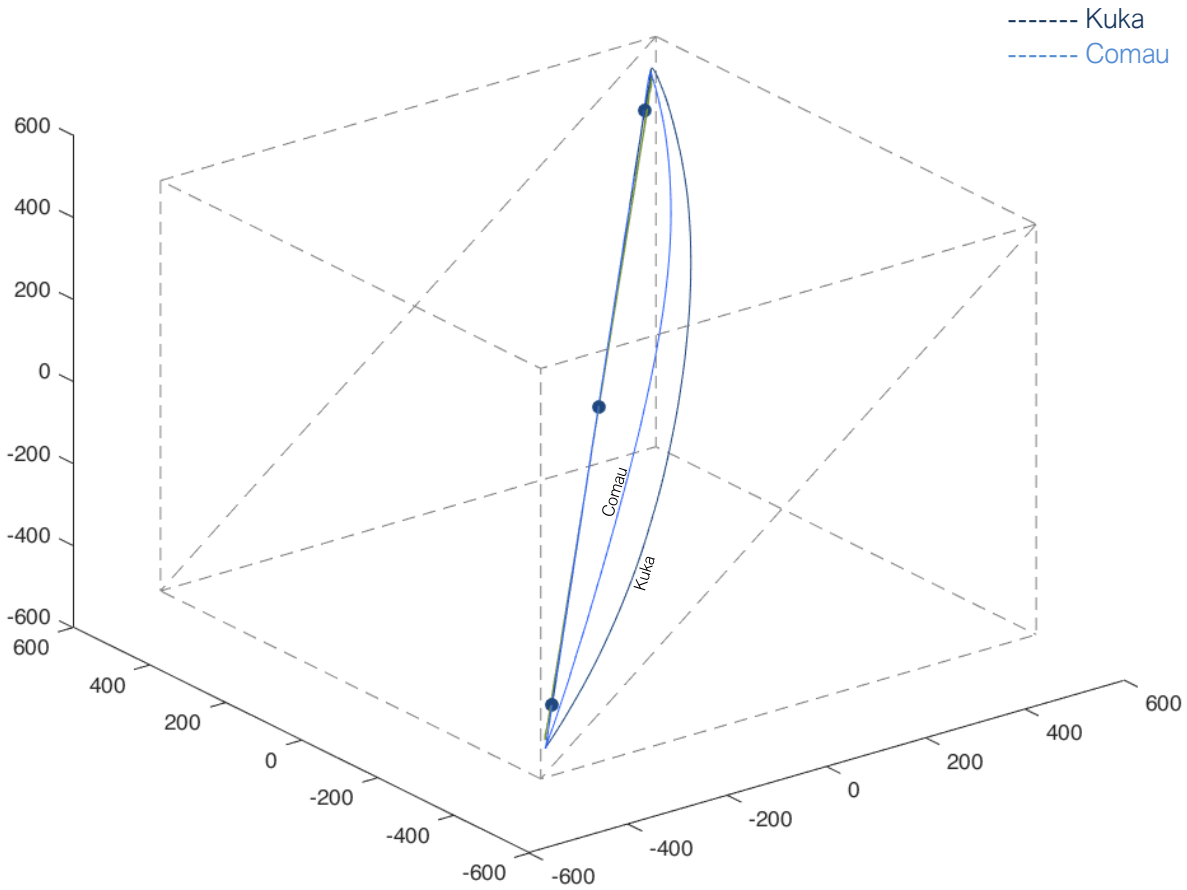


Abbildung 97 – Bahnverlauf der Bahn-Genauigkeitstests für einen nativen Kuka- und Comau-Roboter

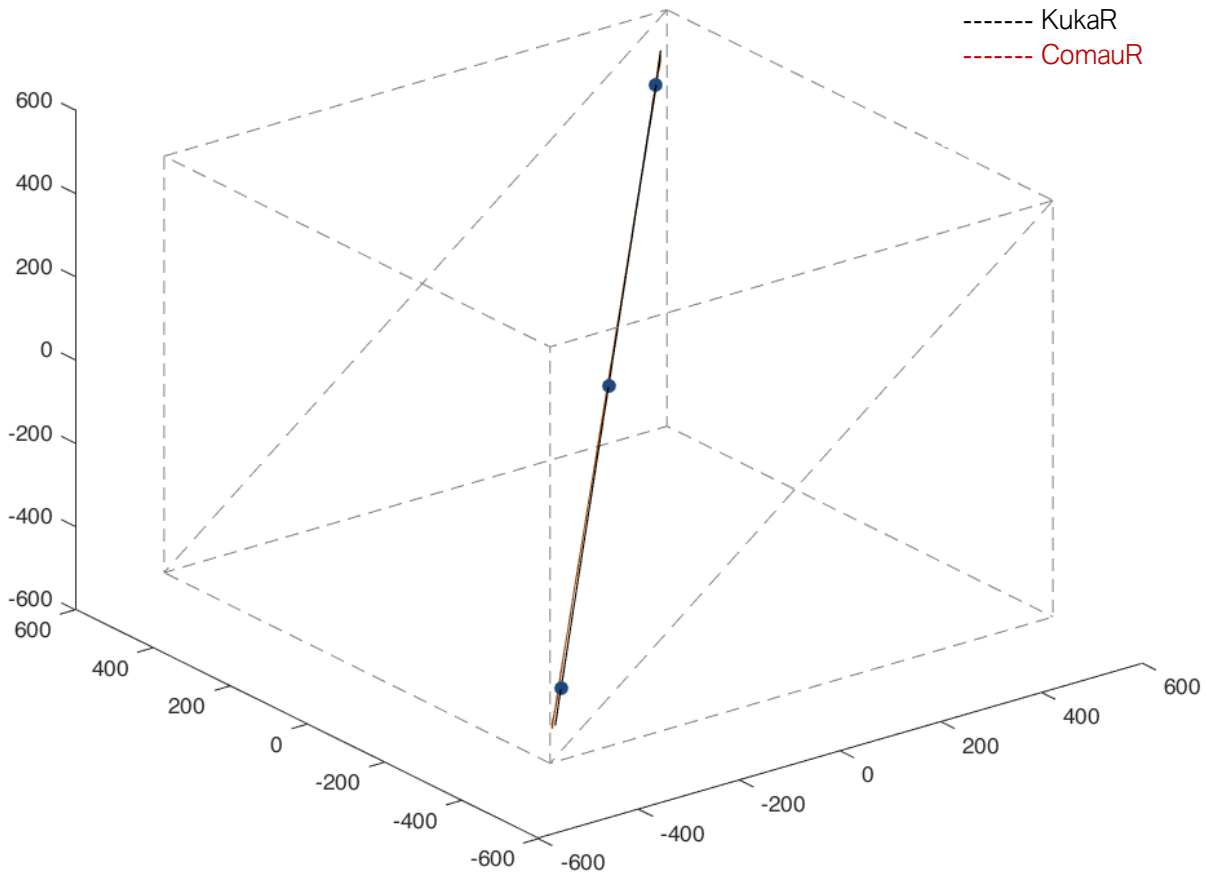


Abbildung 98 – Bahnverlauf der Bahn-Genauigkeitstests für extern gesteuerte Kuka- und Comau-Roboter

Eine detaillierte Visualisierung der Bahnen bietet die gedrehte Seitenansicht in Abbildung 99 und die zugehörige vergrößerte Darstellung der von rechts nach links gehenden Anfangs-, Mittel und Endpunkte der Roboterbahnen. Direkt und am deutlichsten ersichtlich sind die Unterschiede bei der Rückfahrt der nativen Kuka- und Comau-Roboter, da ersterer mit wesentlich weiterem Abstand zur Diagonalbahn fährt.

Auch in der Detailabbildung der Start- und Endpunkte werden die Differenzen in der Trajektorie zwischen den beiden Roboterbahnen ersichtlich. Neben dem erwähnten Abstand zur Diagonale zeigen sich auch Unterschiede im Beginn und am Ende der Diagonalfahrt, da der nativ angesteuerte Comau-Roboter eine Schleifenbewegung zunächst durchführt, während der Kuka-Roboter mit einer Kurvenbewegung die Linearfahrt beginnt.

Im Gegensatz zu den nativen Roboterbahnen zeigt sich bei den anderen beiden Trajektorien ein wesentlich identischeres Verhalten, da beide über die identische Sollpositionsvorgabe des externen Steuerungssystems angesteuert werden. Auffallendster Unterschied ist der Z-Versatz, der ebenso wie bei Testbahn 1 (vgl. Kapitel 9.1) durch die fehlende Absolutgenauigkeitskorrektur begründet ist und stark durch die unterschiedliche Kinematik- und Mechanik des Manipulators beeinflusst wird. Dies fällt insbesondere am Ende der Bahn des extern gesteuerten Comau-Roboters auf.

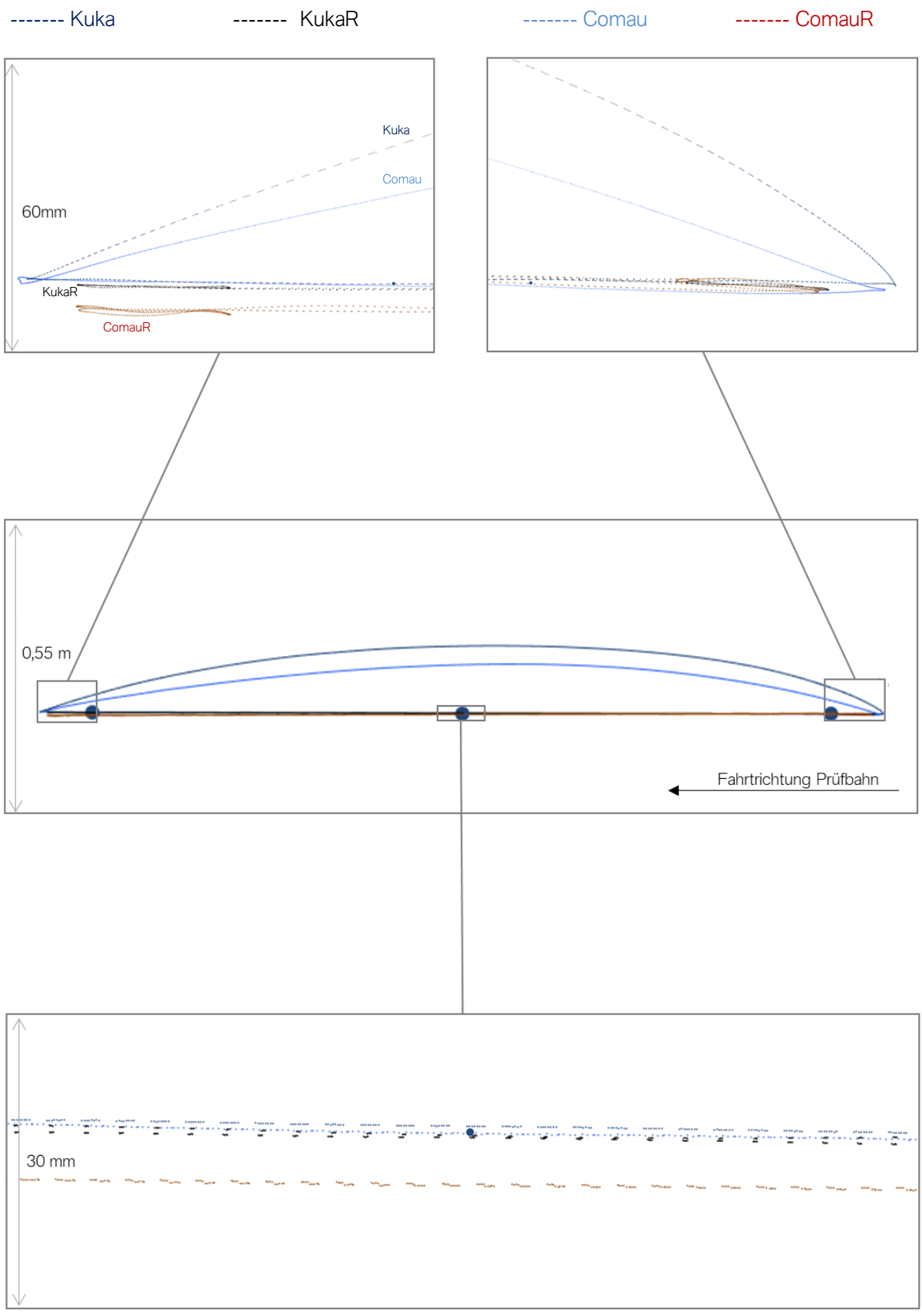


Abbildung 99 – Detailbetrachtung Bahncharakteristik

9.2.2. Leistungskennwerte

Analog zu den Pose-Tests betrug das Messgewicht 140 kg. Abgefahren wurde die Bahn mit einer Geschwindigkeitsvorgabe von 0,10 m/s bis hin zur maximal möglichen Lineargeschwindigkeit der Robotersysteme. Im folgenden Abschnitt wird als Referenz- und Vergleichswert eine Lineargeschwindigkeit von 0,50 m/s gewählt. Um die Betrachtung auf das Verhalten für Bereiche mit konstanten Geschwindigkeiten zu fokussieren, wurde die Bewertungsstrecke von 80% auf 70% der ISO-Cube-Diagonale begrenzt, da der Beschleunigungsvorgang für die Referenzgeschwindigkeit in diesem Punkt sicher abgeschlossen war.

TCP-Geschwindigkeit

Ein zentraler Leistungskennwert bei Bahnbewegungen ist die TCP-Geschwindigkeit. Im Gegensatz zu Punkt-zu-Punkt-Bewegungen ist nicht eine möglichst hohe Geschwindigkeit und Beschleunigung von großer Bedeutung, sondern eine reproduzierbar gleichbleibende Geschwindigkeit, da der Fertigungsprozess (z. B. Kleben) während der Linearbewegung durchgeführt und somit vom Robotersystem stark beeinflusst wird. Abbildung 100 und Abbildung 101 zeigen den Geschwindigkeitsverlauf für native und extern gesteuerte Kuka- bzw. Comau-Robotersysteme anhand der Position innerhalb der abzufahrenden Solldiagonale. Die zu bewertende Bahn startet bei dem X-Wert von 0 mm und reicht bis 1212 mm. In den Grafiken wurden jeweils zehn Fahrten direkt übereinandergelegt, um Geschwindigkeitsunterschiede visuell identifizierbar zu machen.

Aus den Diagrammen für den Kuka-Roboter ist ersichtlich (Abbildung 100), dass der Geschwindigkeitsverlauf jeweils nahezu identisch ist. Sowohl im Vergleich der Fahrten innerhalb eines steuerungstechnischen Konzeptes (Grafik A bzw. Grafik B) als auch im direkten Vergleich der Konzepte zueinander (Grafik C). Auch die Anpassung der Zykluszeit oder die Deaktivierung der Absolutgenauigkeit wirkten sich nicht wesentlich aus (Grafik C). Der auffälligste Unterschied ist im Verlauf vor und nach der eigentlichen Messstrecke zu sehen, da sich hier die unterschiedliche Bahnalgorithmik des externen und nativen Systems zeigt. Der optische Eindruck bestätigt sich auch in den ermittelten Werten. Für beide Steuerungsmethoden wurde eine mittlere Geschwindigkeit von 0,500 m/s erfasst. Die min. und max. Geschwindigkeiten waren mit 0,495 m/s und 0,506 m/s (Kuka) bzw. 0,494 m/s und 0,506 m/s (KukaR) nahezu identisch.

Bei dem Comau-Roboter war das Verhalten für das native und das extern betriebene System ähnlich, solange bei letzterem die Interpolationszeit nicht über 4 ms stieg. Mit Geschwindigkeiten von 0,493 m/s (min.) und 0,508 m/s (max.) fielen die Abweichungen für beide Betriebsarten im Vergleich zum Kuka-Roboter größer aus. Die Durchschnittsgeschwindigkeit war ebenfalls mit 0,500 m/s exakt im Sollwert. Anders verhielt es sich bei den Interpolationszeiten 8ms und 16ms. Zum einen waren die Durchschnittsgeschwindigkeiten mit 0,476 m/s und 0,487 m/s niedriger als die Sollgeschwindigkeiten. Die deckt sich auch mit den Erkenntnissen aus Testbahn 1. Zum anderen war die Reproduzierbarkeit der Geschwindigkeitsverläufe wesentlich schlechter. Dies ist an den unruhigeren Bahnverläufen (Grafik C und Grafik D in Abbildung 101) zu erkennen.

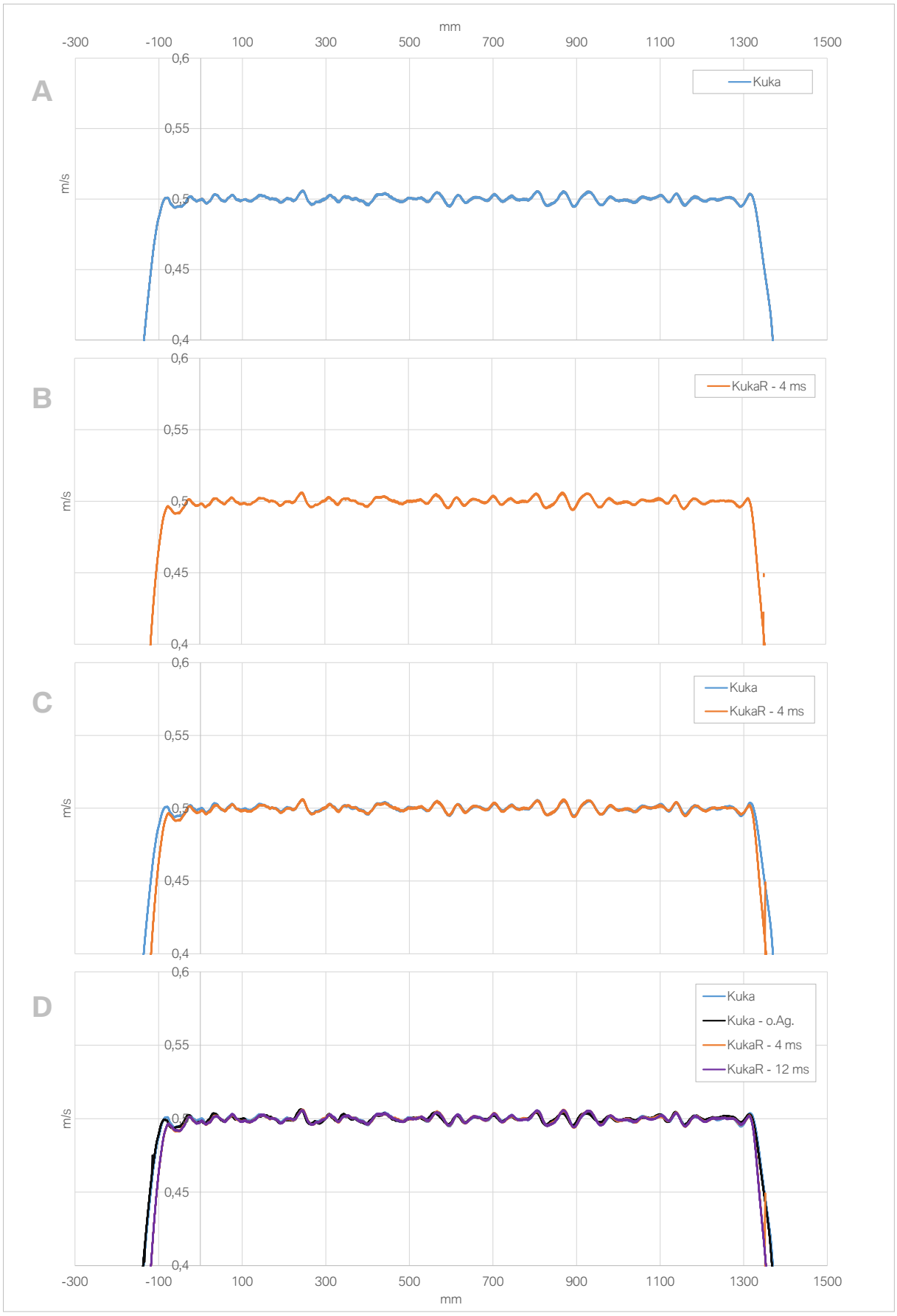


Abbildung 100 – TCP-Geschwindigkeitsverlauf von Linearbahnen (Kuka)

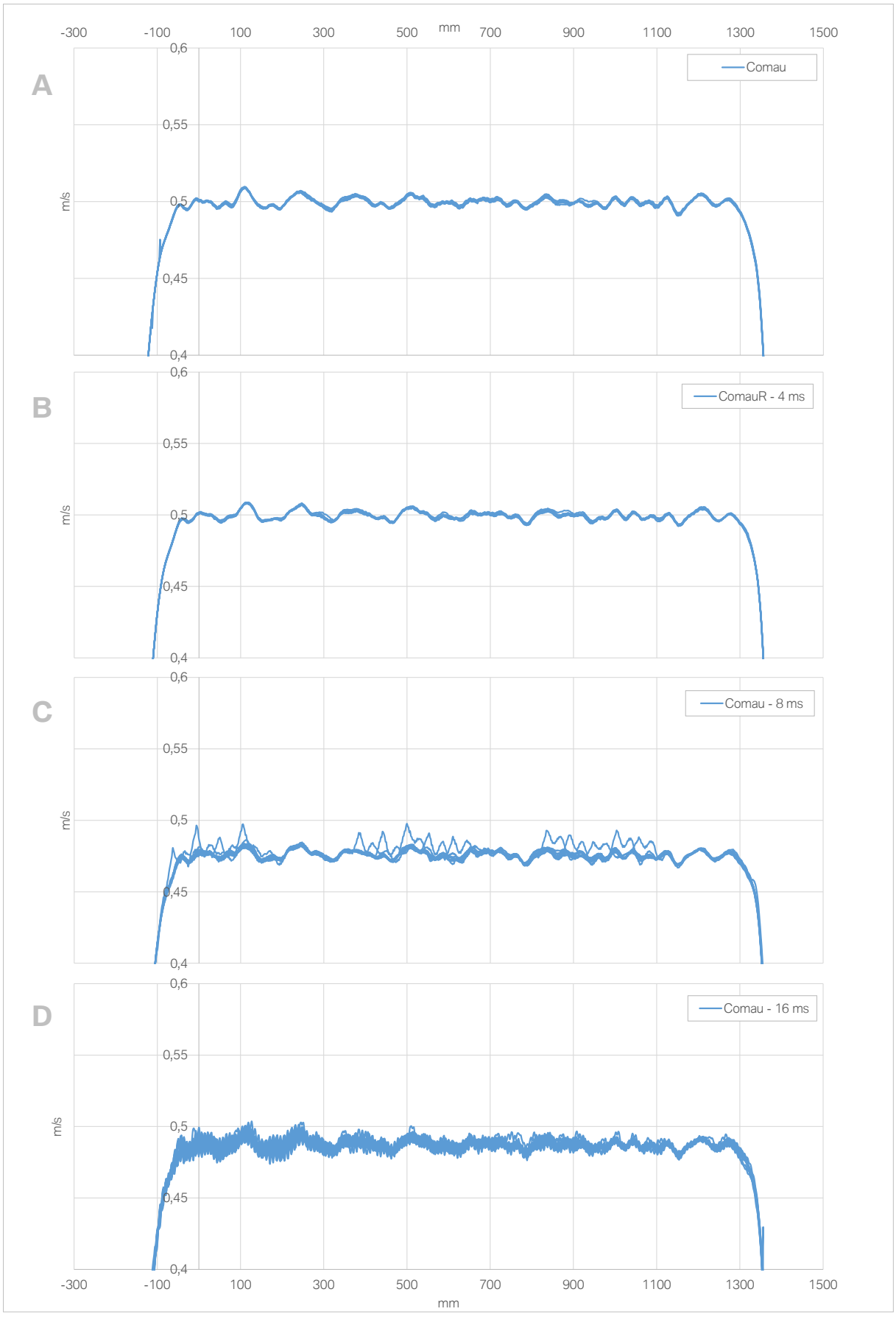


Abbildung 101 – TCP-Geschwindigkeitsverlauf von Linearbahnen (Comau)

Die Wiederholbarkeit der Geschwindigkeit wird in Abbildung 102 anhand der Darstellung der Differenzen zur Schwerpunktgeschwindigkeit detaillierter visualisiert, wobei für Grafik A und Grafik B die Skalierung im rechten Teil verändert wurde, um eine bessere Vergleichbarkeit zu ermöglichen. Die Schwerpunktgeschwindigkeit gibt wieder, wie groß die Geschwindigkeitsdifferenz einer Fahrt im Vergleich zur mittleren Geschwindigkeit aller Bahnen ist. Es zeigt sich, dass bei einer Taktrate von 4ms der extern gesteuerte Kuka-Roboter (Grafik A) etwas geringere Schwankungen als das externe Comau-System aufweist (Grafik B). Mit Differenzen von maximal 1 mm/s (KukaR) bzw. 3 mm/s (ComauR) bei einer Geschwindigkeit von 500 mm/s fallen diese jeweils marginal aus. Die Schwankungen im Betrieb mit 8 ms sowie 16 ms Zykluszeit des extern betriebenen Comau-Roboters fallen hingegen höher aus (Grafik C und D).

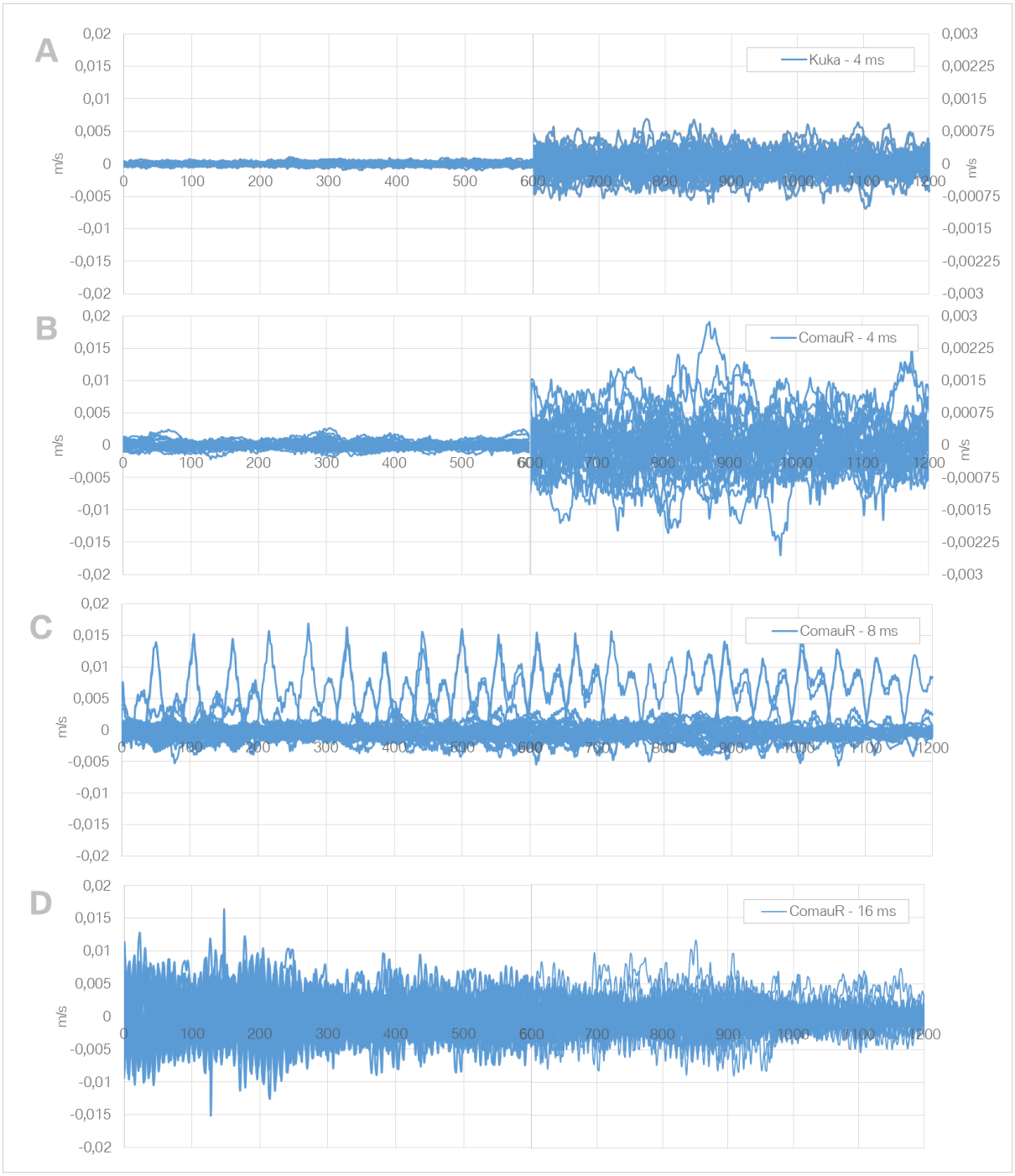


Abbildung 102 – Schwerpunktgeschwindigkeitsdifferenzen für Linearbahntestfahrten bei 50 cm/s

TCP-Genauigkeit

Die Abbildungen 103 und 104 zeigen die aufgezeichneten Linearfahrten des Comau-Roboters für den nativen und für den externen Betrieb. Die X-Position entspricht der Längsposition (Startpunkt 0) der Linearstrecke, während Y die horizontale und Z die vertikale Abweichung darstellen. Bei den meisten Fertigungsprozessen ist die Y-Abweichung von höherer Bedeutung für die Fertigungsqualität, da diese direkt Auswirkung auf die geometrische Position des Fügeprozesses hat. Eine geringe Abweichung der Z-Position, also in der Höhe, hat meist keine bzw. geringe Auswirkungen auf den Prozess. Beispielsweise führt bei einem Klebeprozess eine horizontale Abweichung zu einem krummen Klebeauftrag, während eine vertikale Abweichung zur Folge hat, dass das Klebemedium geringfügig verzögert aufgetragen wird. Geometrie und Menge der Klebenaht bleiben aber unverändert.

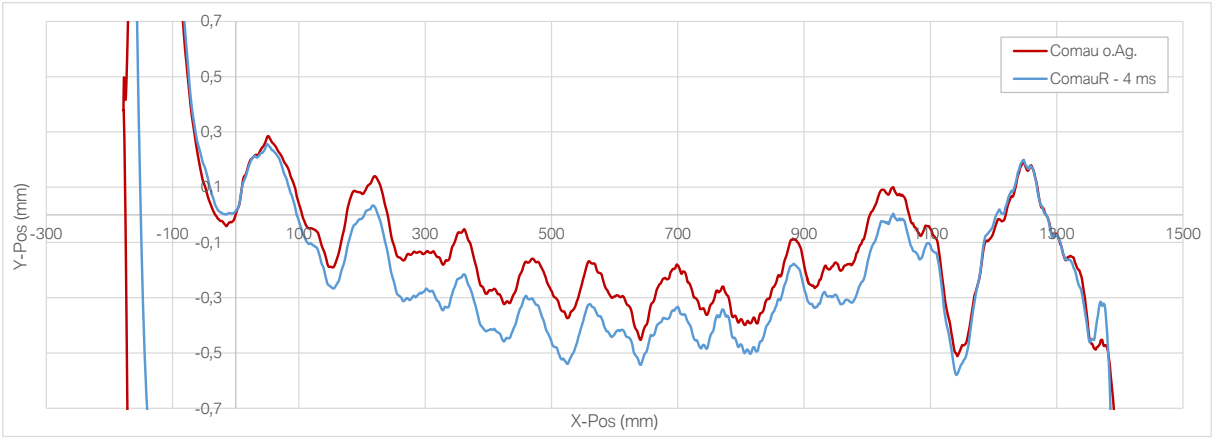


Abbildung 103 – Vergleich der horizontalen Abweichung (Y) des TCP bei einer Comau- und ComauR-Linearfahrt

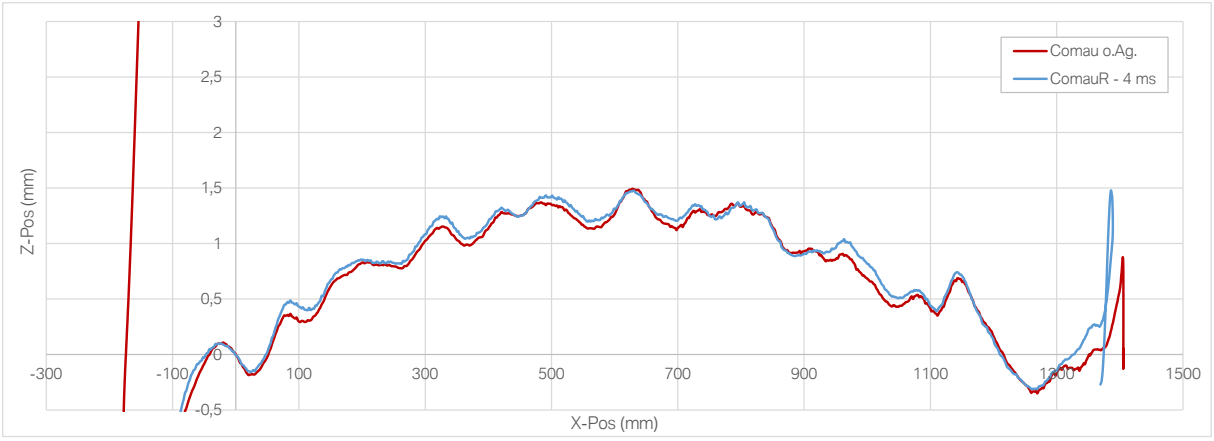


Abbildung 104 – Vergleich der vertikalen Abweichung (Z) des TCP bei einer Comau- und ComauR-Linearfahrt

9.2.3. Messergebnisse im Überblick

Die Übersicht der Gesamtergebnisse der Linearfahrten ist für die diskutierten Leistungskennwerte in Tabelle 17 dargestellt. Zusätzlich sind die Auswertungen für weitere Geschwindigkeitsvorgaben ebenso aufgeführt. Eine Zeile repräsentiert die mittleren Ergebnisse drei hintereinanderliegender Messungen. Im Gegensatz zur ISO-Norm wurden pro Messung 30 statt der vergebenen zehn Bahnen abgefahren, um die Aussagekraft zu erhöhen und eine bessere Vergleichbarkeit zu den Pose-Messungen zu gewährleisten.

Als Eingangsgrößen wurden die bis in die vierte Spalte aufgelisteten Parameter Robotertyp, Interpolationstakt der externen Schnittstelle, vorgegebene Bahngeschwindigkeit und ein- bzw. ausgeschalteter Absolutgenauigkeit gewählt. Ergebnisgrößen sind die maximale (RT max) und mittlere Wiederholgenauigkeit (RT Ø) der Linearbahnpunkte, wobei erstere der Bahnwiederholgenauigkeit (RT) gemäß ISO-Norm entspricht. Des Weiteren sind die Bahngeschwindigkeit im Durchschnitt (V Ø) sowie die minimale und maximal erfassten Geschwindigkeitsspitzen (V min; V max) als absoluter Wert und in Relation zur mittleren Geschwindigkeit angegeben. Die darauffolgenden Spalten beschreiben die mittlere und maximale Differenz für Y- und Z-Wert im Vergleich zur idealen Sollbahn, inklusive des Vorzeichens der Abweichung. Die letzten drei Kennwerte beschreiben den mittleren Abstandswert von Y, Z, und YZ zur idealen Sollbahn als Betragswert der vorher genannten Abweichung, also ohne Einbezug des Vorzeichens.

In der Gesamtbetrachtung lässt sich zunächst festhalten, dass die Bahngenauigkeitswerte der externen Systeme sowohl für den Comau- als auch den Kuka-Roboter in gleicher Größenordnung zu den nativen Systemen liegt, also kein Genauigkeitsverlust zwischen den Ansteuerungskonzepten festgestellt werden kann. Zudem ist ersichtlich, dass die Wiederholgenauigkeitskennwerte für den nativen und den extern gesteuerten Kuka-Roboter in der gleichen Größenordnung wie bei den Pose-Bewegungen sind. Die gemessenen Wiederholgenauigkeitswerte für den Comau-Roboter sind sowohl für externe als auch native Steuerung schlechter als bei Testbahn 1, liegen jedoch im Rahmen der angegebenen Genauigkeit und sind für beide Steuerungskonzepte in einer vergleichbaren Größenordnung, abgesehenen von den besprochen Ungenauigkeitseffekten bei Ansteuerungszykluszeiten von mehr als 4 Millisekunden.

In Bezug auf die Geschwindigkeitskennwerte zeigt sich, dass bis zu einer Geschwindigkeit von 0,5 m/s sämtliche getesteten Steuerungslösungen eine Durchschnittsgeschwindigkeit in exakter Höhe der Vorgabe besitzen. Zudem sind die Geschwindigkeitssenken und -spitzen für native und externe Systeme jeweils ähnlich, wobei das Comau-System grundsätzlich geringfügig höhere Abweichungen als das Kuka-System aufweist. Bei höheren Geschwindigkeitsvorgaben ist bei den nativen Systemen eine stetig wachsende Abweichung im Vergleich zu den externen Steuerungslösungen zu sehen. Ursache ist, dass im externen Steuerungssystem eine bessere Anfahrtsbahn gewählt wurde, als dies in den nativen Systemen der Fall war. Der Unterschied ist folglich nicht system- sondern implementierungsbedingt und lässt daher aufgrund der unterschiedlichen Beschleunigungsbahnen nur einen bedingten Vergleich der Leistungskennwerte im hohen Geschwindigkeitsbereich zu. Die Messungen zeigen aber, dass auch mit externen Steuerungssystemen die gleichen Geschwindigkeitswerte wie im nativen Betrieb erreicht werden konnten und keine Beeinträchtigung vorlag.

Die Abweichung in Bezug auf die ideale Sollgerade kann am besten anhand des Genauigkeitswertes in der vorletzten Spalte diskutiert werden. Es zeigt sich, dass die nativ und extern angesteuerten Systeme für fast alle Messungen Kennwerte in einem ähnlichen Leistungsspektrum aufweisen. Lediglich im sehr niedrigen Geschwindigkeitsbereich von 0,1 m/s besitzt das native Kuka-System eine wesentlich geringere Abweichung als das extern gesteuerte System, welches jedoch weiterhin die Genauigkeit des nativen Comau-Roboters übertrifft.

Tabelle 17 – Messergebnisse für lineare Testbahnen

Roboter	Zyklus-zeit (ms)	V (m/s)	RT max (mm)	RT Ø (mm)	V Ø (m/s)	V min (m/s)	V max (m/s)	V min (%)	V max (%)	Δ Y-TCP Ø (mm)	=> max (mm)	Δ Z-TCP Ø (mm)	=> max (mm)	lΔY-TCPl Ø (mm)	lΔZ-TCPl Ø (mm)	lΔYZ-TCPl Ø (mm)	Temp (°C)
0,1 m/s																	
Kuka		0,10	0,043	0,028	0,100	0,097	0,103	97,3	102,6	0,009	0,133	0,064	0,206	0,037	0,072	0,087	21,45
KukaR	4	0,10	0,057	0,030	0,100	0,098	0,103	97,7	102,9	-0,074	0,196	0,124	0,282	0,075	0,128	0,151	22,19
Comau		0,10	0,078	0,048	0,100	0,096	0,105	95,9	105,4	-0,223	0,471	0,785	1,302	0,228	0,786	0,821	21,80
ComauR		0,10	0,093	0,056	0,100	0,095	0,105	95,2	104,9	-0,247	0,519	0,753	1,294	0,257	0,754	0,800	21,78
0,25 m/s																	
Kuka		0,25	0,098	0,039	0,250	0,244	0,255	97,6	102,1	-0,029	0,195	0,193	0,358	0,053	0,195	0,207	21,63
KukaR	4	0,25	0,112	0,035	0,250	0,244	0,256	97,6	102,3	-0,109	0,292	0,154	0,337	0,126	0,165	0,212	22,01
Comau		0,25	0,134	0,080	0,250	0,245	0,254	98,1	101,5	-0,168	0,437	0,835	1,384	0,182	0,836	0,860	21,80
ComauR		0,25	0,153	0,092	0,250	0,246	0,254	98,4	101,7	-0,248	0,581	0,862	1,363	0,258	0,863	0,906	21,87
0,5 m/s																	
Kuka		0,50	0,043	0,026	0,500	0,495	0,506	99,0	101,2	-0,096	0,294	0,068	0,308	0,103	0,116	0,163	22,36
Kuka o.Ag.		0,50	0,056	0,035	0,500	0,495	0,506	98,9	101,1	-0,098	0,337	0,062	0,354	0,115	0,130	0,181	22,06
KukaR	4	0,50	0,045	0,027	0,500	0,494	0,506	98,8	101,2	-0,108	0,354	0,018	0,390	0,127	0,116	0,185	22,17
KukaR	12	0,50	0,040	0,026	0,500	0,494	0,506	98,9	101,2	-0,108	0,357	0,021	0,386	0,127	0,117	0,184	22,19
Comau		0,50	0,162	0,089	0,500	0,492	0,509	98,4	101,9	-0,178	0,458	0,917	1,523	0,208	0,920	0,951	21,86
Comau o.Ag.		0,50	0,191	0,094	0,500	0,493	0,508	98,6	101,7	-0,165	0,567	0,891	1,490	0,208	0,896	0,930	21,88
ComauR	4	0,50	0,167	0,092	0,500	0,493	0,508	98,6	101,7	-0,267	0,613	0,918	1,524	0,292	0,923	0,977	21,87
ComauR	0,4	0,50	0,186	0,118	0,500	0,493	0,508	98,6	101,7	-0,253	0,587	0,873	1,505	0,279	0,880	0,932	21,80
ComauR	8	0,50	0,240	0,144	0,476	0,468	0,492	98,3	103,2	-0,253	0,580	0,868	1,512	0,277	0,873	0,924	21,87
ComauR	16	0,50	0,423	0,181	0,487	0,477	0,500	97,8	102,6	-0,261	0,583	0,885	1,521	0,284	0,891	0,943	21,86
0,8 m/s																	
Kuka		0,80	0,038	0,026	0,800	0,787	0,807	98,4	100,9	0,124	0,558	0,048	0,363	0,137	0,114	0,200	22,36
KukaR	4	0,80	0,048	0,028	0,800	0,793	0,806	99,1	100,8	-0,209	0,456	0,055	0,354	0,211	0,119	0,254	22,14
Comau		0,80	0,197	0,098	0,795	0,708	0,809	89,1	101,7	-0,166	0,599	0,659	1,268	0,278	0,682	0,766	21,87
ComauR	4	0,80	0,178	0,104	0,799	0,791	0,810	99,0	101,4	-0,083	0,333	0,716	1,346	0,171	0,735	0,769	21,79
0,9 m/s																	
Kuka		0,90	0,038	0,025	0,899	0,840	0,908	93,5	101,0	0,112	0,594	0,090	0,344	0,164	0,139	0,249	22,19
KukaR	4	0,90	0,040	0,026	0,900	0,890	0,908	99,0	100,9	-0,296	0,570	0,138	0,349	0,296	0,157	0,342	22,09
Comau		0,90	0,159	0,087	0,885	0,722	0,908	81,6	102,6	-0,043	0,786	0,520	1,150	0,304	0,609	0,721	21,63
ComauR	4	0,90	0,189	0,104	0,899	0,890	0,909	99,0	101,1	-0,080	0,318	0,674	1,303	0,154	0,697	0,727	21,80
1,0 m/s																	
Kuka		1,00	0,040	0,027	0,994	0,870	1,010	87,5	101,6	-0,014	0,597	0,162	0,469	0,216	0,188	0,312	22,35
KukaR	4	1,00	0,042	0,028	0,999	0,981	1,011	98,2	101,1	-0,142	0,398	0,049	0,328	0,179	0,113	0,221	22,06
Comau		1,00	0,170	0,099	0,963	0,722	1,008	75,0	104,6	0,067	0,716	0,394	1,041	0,286	0,511	0,637	21,63
ComauR	4	1,00	0,170	0,094	0,999	0,990	1,008	99,0	100,9	-0,080	0,306	0,626	1,255	0,140	0,656	0,682	21,86

Legende: V – Geschwindigkeit; RT max – Bahnwiederholgenauigkeit (ISO Wert); RT Ø – mittlere Bahnwiederholgenauigkeit; Δ Y-TCP Ø – mittlere Y-Abweichung zur idealen Sollbahn; lΔ Y-TCPl Ø – mittlerer Betrag der Y-Abweichung zur idealen Sollbahn;

9.2.4. Kreisbahnen

Bei der Kreisprüfbahn werden im Gegensatz zur Testbahn 2 kreisförmige statt geradlinige Bahnkurven auf der diagonalen Ebene des ISO-Cubes mit gleichbleibender Geschwindigkeit abgefahren. Dieser Radius kann innerhalb der vorgegebenen Fläche der Größe nach variieren. In den durchgeführten Untersuchungen wurde ein Radius von 400 mm gewählt. Abbildung 105 zeigt die erfassten Testbahnen für den Kuka-Roboter mit nativer und externer Steuerung. In der vergrößerten Darstellung ist ersichtlich, dass sich mit steigender Geschwindigkeit der Radius beim nativen Kuka-Roboter leicht verändert, sich wohingegen das System mit externer Steuerung auch bei Geschwindigkeitsänderungen bahntreuer verhält. Mit dem externen System konnte jedoch maximal eine Bahngeschwindigkeit von 1,4 m/s abgefahren werden, während mit dem nativen Robotersystem Geschwindigkeiten von über 1,6 m/s erreicht wurden (Tabelle 18). Analog zur geschwindigkeitsabhängigen Veränderung des Radius ist die Abweichung zur Sollgeschwindigkeit beim nativen System mit steigender Bahngeschwindigkeit gewachsen, während sie im extern gesteuerten System weitestgehend gleich geblieben ist. Auch die mittlere und maximale Rundheitsabweichung war im extern gesteuerten System weniger stark ausgeprägt. Die Wiederholgenauigkeitswerte in Bezug auf Genauigkeit und Bahngeschwindigkeit entsprachen denen der Diagonalfahrten und wurde aus den o.g. Gründen nicht gesondert aufgeführt.

Zentrale Aussage des Kreisbahntestes ist, dass es auch mit dem externen Steuerungskonzept möglich ist, vorgegebene Kreisbahnen zu fahren. Im vorliegenden Fall sogar mit einer besseren Einhaltung der idealen Sollgeometrie als mit dem nativen System, allerdings mit Abstrichen hinsichtlich der dynamischen Leistungsfähigkeit in Bezug auf die maximal ausführbaren Geschwindigkeiten.

Tabelle 18 – Messergebnisse für Kreisbahnen

Roboter	V Soll (m/s)	R Soll (mm)	V (m/s)	Delta V Soll (%)	Radius (mm)	Delta R	Umfang (mm)	Delta U Soll (%)	Abw. Rundheit Ø (mm)	Abw. Rundheit Max (mm)
KukaR	0,20	400	0,200	-0,03	399,879	-0,03%	2512,517	-0,03	0,243	0,622
Kuka	0,20	400	0,200	-0,03	399,897	-0,03%	2512,626	-0,03	0,577	0,983
KukaR	0,50	400	0,500	-0,08	399,687	-0,08%	2511,307	-0,08	0,546	0,979
Kuka	0,50	400	0,499	-0,23	399,102	-0,22%	2507,629	-0,22	0,602	1,014
KukaR	1,00	400	0,999	-0,14	399,453	-0,14%	2509,836	-0,14	0,975	1,711
Kuka	1,00	400	0,992	-0,81	396,756	-0,81%	2492,894	-0,81	1,491	2,277
KukaR	1,40	400	1,397	-0,19	399,190	-0,20%	2508,186	-0,20	1,569	2,638
Kuka	1,50	400	1,473	-1,78	392,914	-1,77%	2468,753	-1,77	2,900	4,541
Kuka	100%	400	1,643		391,162	-2,21%	2457,746	-2,21	3,586	5,559

Legende:
V – Geschwindigkeit
R – Radius
U – Umfang

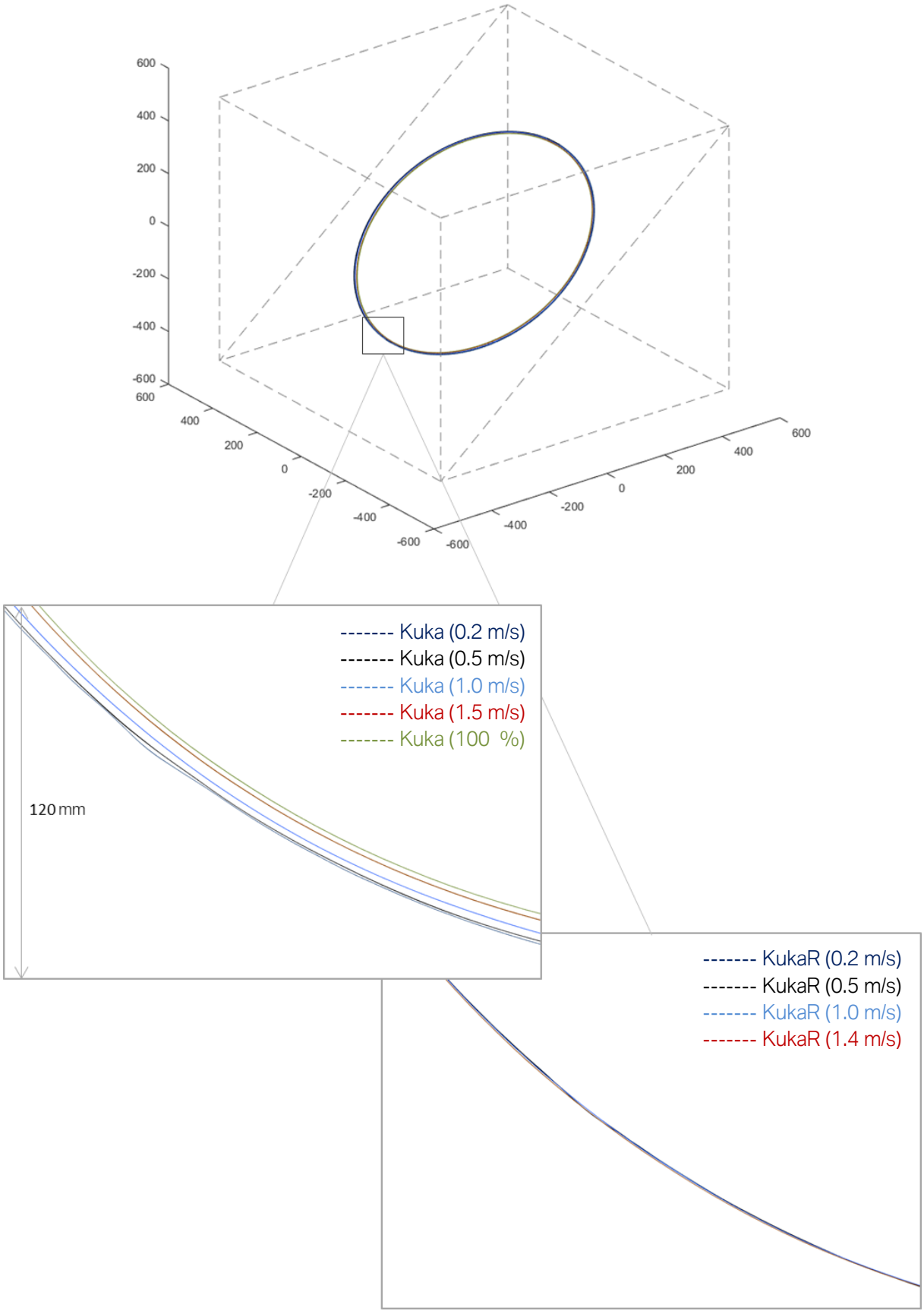


Abbildung 105 – Kreisbahnen mit nativ und extern gesteuertem Kuka-Roboter-System

9.3. Quellcode der Testprogramme

Der Quellcode der Testprogramme für die Durchführung der Leistungsmessungen kann über die Weiterleitungsadresse http://robot-code.taschew.de abgerufen werden.

9.4. Herstellerunabhängige Programmierung von Robotersystemen

9.4.1. Verteilung BMW-Applikationssoftware

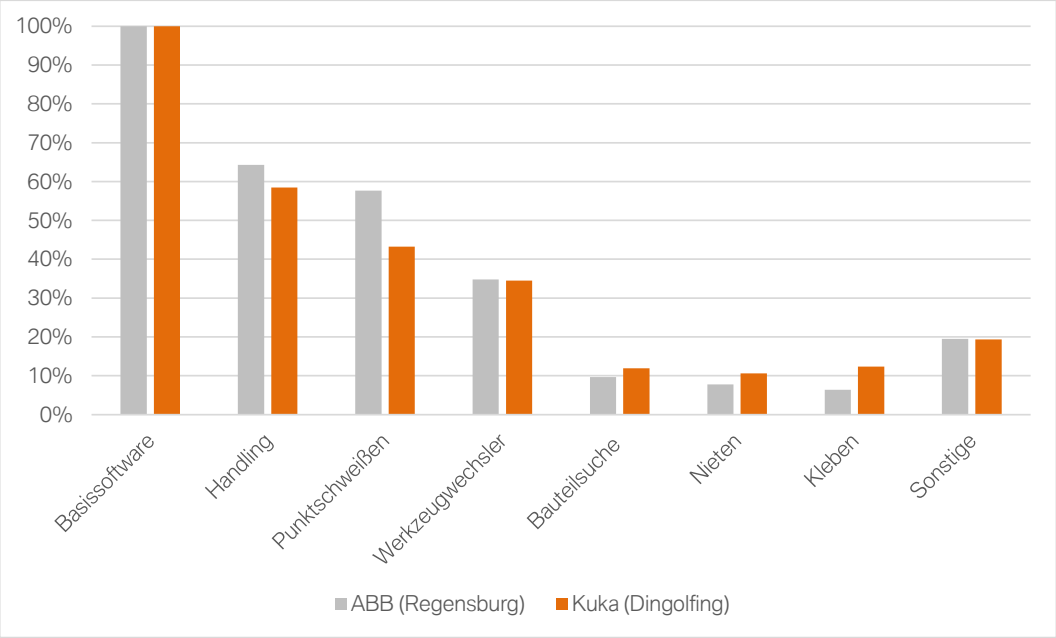


Abbildung 106 – Ermittelte Verteilung der BMW-Applikationssoftware auf Basis der durchgeführten Code-Analyse für die Werke Regensburg und Dingolfing

9.4.2. KRL-, RAPID-,AIRL und C++/AIRL-Befehle im Vergleich mit Erläuterung

```
;FOLD PTP HOME1 Vel=100 % … 1:PTP, 2:HOME1, 3: , 5:100, 7:PHOME1      Anweisung zur verschachtelten Darstellung des Befehl, wird von Interpreter ignoriert

$BWDSTART=FALSE                    Systemvariable, die nur noch aus Kompatibilitätsgründen existiert, keine Auswirkung
PDAT_ACT=PPHOME1                   Bewegungsparameter wird in Struct PDAT festgelegt. Enthält %-Werte für Geschwindigkeit, Beschleunigung, Überschleifparameter, Ruck
FDAT_ACT=FHOME1                    Struct FDAT zum Festlegen von Koordinatensystemeigenschaften und -bezügen (Werkzeug, Arbeitsraum)
BAS(#PTP_PARAMS,100)               Aufruf der BAS-Systemunterfunktion, verarbeitet Parameter und Vorbedingungen für Bewegungsablauf
PTP XHOME1                         Ausführen einer PTP-Bewegung zum Punkt XHOME mit aktuell festgelegten Systemparametern

;ENDFOLD                           Anweisung für das Ende der Verschachtelung eines Befehls, wird von Interpreter ignoriert
```

```
;FOLD PTP HOME1 Vel=100 % … 1:PTP, 2:HOME1, 3:C_DIS , 5:100, 7:PHOME1

$BWDSTART=FALSE
PDAT_ACT=PPHOME1
FDAT_ACT=FHOME1
BAS(#PTP_PARAMS,100)
PTP XHOME1 C_DIS

;ENDFOLD
```

C_DIS als Erweiterung für PTP-Bewegung mit Überschleifen des Zielpunktes.

Überschleifparameter in PPHOME1 enthalten. Variable ist in separater Konfigurationsdatei gespeichert

```
;FOLD LIN HOME1 Vel=2 … 1:LIN, 2:HOME1, 3: , 5:2, 7:PHOME1

$BWDSTART=FALSE
LDAT_ACT=LCPHOME1
FDAT_ACT=FHOME1
BAS(#CP_PARAMS,2)
LIN XHOME1

;ENDFOLD
```

LIN anstatt PTP für Linear- statt Punkt-zu-Punkt-Bewegung

Bewegungsparameter im Struct LDAT, zusätzlich Parameter wie Bahngeschwindigkeit und -beschleunigung

Angabe der Geschwindigkeit in m/s statt in prozentualer Maximalgeschwindigkeit

Aufruf der BAS-Systemfunktion mit Parametern für Bahnfunktionen

Abbildung 107 – Kuka KRL-Befehle für eine Punkt-zu-Punkt-Bewegung (o.), eine Punkt-zu-Punkt-Bewegung mit Überschleifen (m.) und eine Linearbewegung (u.)

```
MoveJ home1, vmax, fine , tool1\WObj:=wobj_01;      Bewegungsbefehl für Punkt-zu-Punkt-Bewegung mit den nachfolgenden Parametern
```

Zielpunkt für Bewegung

Geschwindigkeit für Bewegung

Parameter für Fahrt ohne Überschleifen

Zu verwendendes Werkzeug

Optionale Parameter: Werkraum

```
MoveJ home1, vmax, z30 , tool1\WObj:=wobj_01;
```

Angepasster Parameter für Überschleifen mit einer TCP-Zone von 30mm

```
MoveL home1, v1000, fine , tool1\WObj:=wobj_01;
```

Bewegungsbefehl für das Ausführen von Linearbewegungen

Geschwindigkeitsparameter für Linearbewegungen mit TCP-Geschwindigkeit 1000 mm/s

Abbildung 108 – ABB RAPID-Befehle für eine Punkt-zu-Punkt-Bewegung (o.), eine Punkt-zu-Punkt-Bewegung mit Überschleifen (m.) und eine Linearbewegung (u.)

```
;FOLD Swp Spotpoint PTP TypeId:22 SpotNo:5940 GunNo:1 GunOpen:50mm Vel:100% _Tool[1]:xGun1 Base[1]:40FX1 PartEnd:False ...
;... EqualOffset:False FastClose:False LeaveCtrl:Ctrl

$BWDSTART = FALSE                  BMW-spezifische FOLD-Anweisung für das Setzen eines Punktschweißpunktes mit der Positionierung durch einer PTP-Fahrt
PDAT_ACT = Pswp5940_22             Bewegungs- und Koordinatendaten für Zielpunkt swp5940_22 mit Werkzeug xGun1 im Werkraum 40FX1
FDAT_ACT = Fswp5940_22             BMW-spezifische Subroutinen bzw. Applikationsfunktionsaufrufe
BAS (#PTP_PARAMS,100)              Für den Schweißpunkt Nr. 5940 mit der Ident-Nr. 22 mit der Zange 1 und einem Zangenöffnungsmaß von 50mm
Swp_b_APP_READY=FALSE              Bei deaktiviertem Zangenausgleich (EqualOffset) und aktivierter Zangenkontrolle (LeaveCtrl)
Swp_AdvSpot (1,22,5940)
TRIGGER WHEN DISTANCE=1 DELAY=0 DO Swp_PreSpot (1,5940,0) PRIO=-1
PTP Xswp5940_22
Swp_Exe (1,22,5940,50,0,0,#Ctrl,False)
WAIT FOR Swp_b_APP_READY

;ENDFOLD
```

Abbildung 109 – BMW-spezifischer KRL-Befehl für eine Punkt-zu-Punkt-Bewegung mit Punktschweißoperation

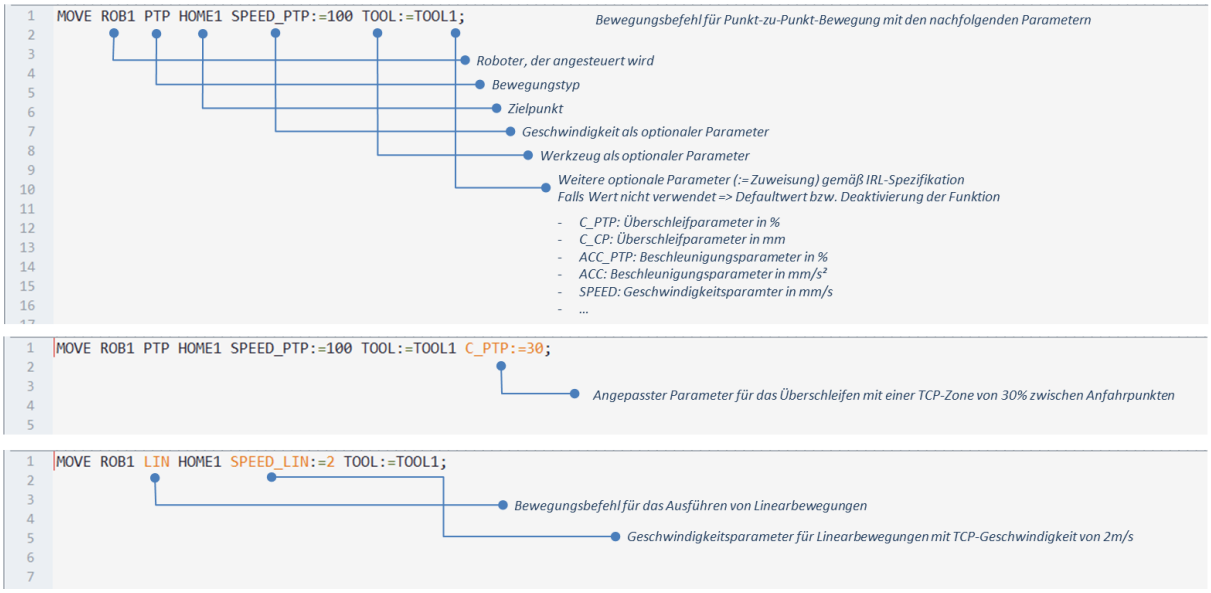


Abbildung 110 – AIRL-Befehle für eine Punkt-zu-Punkt-Bewegung (o.), eine Punkt-zu-Punkt-Bewegung mit Überschleifen (m.) und eine Linearbewegung (u.)

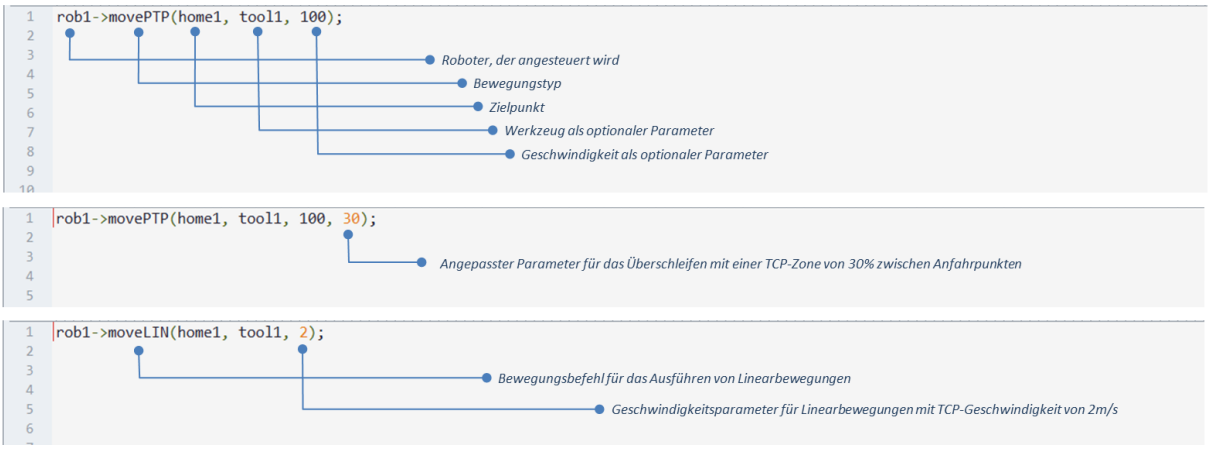


Abbildung 111 – C++/AIRL-Befehle für eine Punkt-zu-Punkt-Bewegung (o.), eine Punkt-zu-Punkt-Bewegung mit Überschleifen (m.) und eine Linearbewegung (u.)

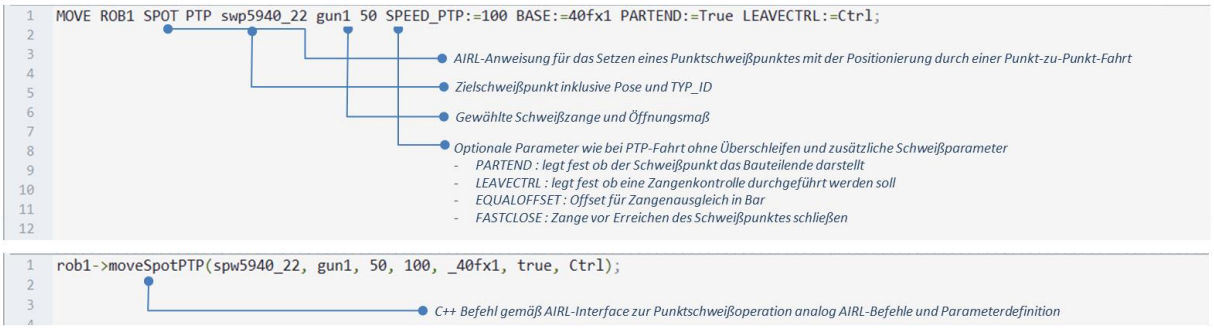


Abbildung 112 – AIRL-Befehl (o.) und C++/AIRL (u.) für eine Punkt-zu-Punkt-Bewegung mit Punktschweißoperation

9.4.3. AIRL-Sprachumfang

Eine Übersicht der durch die AIRL definierten Befehle und notwendigen Funktionalitäten ist in Tabelle 19 dargestellt. Jede Tabellenzeile beschreibt ein einzelnes Sprachelement mit der zugehörigen Kategorie, Gruppierung, Kurzbeschreibung und Anweisungsdefinition in unterschiedlichen Roboterprogrammiersprachen sowie der AIRL.

Sowohl in Lexik, Semantik und Syntaktik stellt die IRL-Norm die Grundlage für die AIRL dar. Um zu verdeutlichen, ob sich eine AIRL-Instruktion aus der IRL ableitet oder im Rahmen der AIRL-Spezifikation neu definiert werden musste, sind neben den AIRL-Anweisungen die zugehörigen IRL-Befehle aufgeführt. Zudem sind aus der IRL übernommene AIRL-Befehle durch eine helligkeitsreduzierte Schriftfarbe gekennzeichnet. Für neu definierte AIRL-Befehlsumfänge sind neben der Instruktion zusätzlich die applikationsspezifischen Eingangsgrößen aufgeführt, welche auf die Inhalte der Tabelle 20 verweisen und dort genauer deklariert sind. Bewegungsspezifische Parameter sowie Eingangsgrößen für abgeleitete Befehle sind hier nicht enthalten. Aufgrund der gemeinsamen Sprachbasis wird zur genaueren Spezifikation hierfür auf die IRL-Norm verwiesen.

Des Weiteren sind zur besseren Verständlichkeit und Vergleichbarkeit die adäquaten Befehle und Deklarationen der proprietären Programmiersprachen KRL und RAPID dargestellt, wobei die Zuordnung aufgrund unterschiedlicher Umsetzungen und Konzepte nicht immer als absolut identisch angesehen werden kann. Falls es nicht möglich war, einen passenden Befehl zu identifizieren, ist dies mit einem Minus-Zeichen gekennzeichnet. Das Fehlen von Befehlen muss nicht zwangsläufig ein Mangel an Funktionalität der jeweiligen Sprache bedeuten, sondern kann auch im geringen Dokumentationsgrad der Handbücher, Übernahmeungenauigkeiten während der manuellen Aufbereitung der Handbücher oder in alternativen Umsetzungskonzepten begründet sein. Automotive Befehle stellen in der Regel keine Instruktionen des Roboterherstellers dar, sondern wurden im Rahmen der Analyse der Anwendungen des Fallbeispiels identifiziert und in der Tabelle auf die ausgewählten Applikationsbeispiele des Prototyps beschränkt.

Zentrales Ziel der tabellarischen Darstellung ist den Befehlsumfang der AIRL verständlich und umfassend darzustellen und nicht eine exakte 1:1 Zuordnung von Instruktionen und Deklarationen, wie dies im Rahmen der Roboterprogrammiersprachenanalyse in Kapitel 6.3.1 erfolgt ist. Die Anzahl an Befehlen kann daher im Vergleich geringfügig variieren, da Funktionalitäten in einer Sprache beispielsweise über Befehlsparameter realisiert werden, während andere Programmiersprachen eigenständige Instruktionen hierfür verwenden. Aus demselben Grund kann es vorkommen, dass die Funktion eines einzelnen AIRL-Befehls mit mehreren Instruktionen in einer anderen Sprache umgesetzt werden muss. In diesem Fall sind die alternativen Befehle in der Tabelle durch ein Semikolon getrennt voneinander aufgelistet.

Die Befehlsparameter in Tabelle 20 sind analog zur Kategorisierung der Befehlsübersicht aufgeführt. Jeder Eingangsparameter ist mit einem Bezeichner, dem entsprechenden Datentypen und einer Beschreibung dargestellt. Des Weiteren hat jeder Parameter einen Identifier, der die Zuordnung in Tabelle 19 ermöglicht. Datentypen, die sich nicht aus der IRL ableiten, sind über die Spalte „Deklaration/Aufbau“ genauer spezifiziert. Handelt es sich um zusammengesetzte Datentypen sind die einzelnen Bestandteile direkt nach diesem Datentyp aufgeführt. Zudem enthalten die Identifier in diesem Fall jeweils ein Punkt-Zeichen mit anschließender Nummerierung der zugehörigen Elemente und ermöglichen so eine einfachere Identifikation der Untergruppierung.

Tabelle 19.1 – Befehls- und Datentypenübersicht der AIRL inkl. der adäquaten Instruktionen in KRL, RAPID und IRL

Nr.	Kategorie	Gruppe	Funktion/Kurzbeschreibung	Roboterprogrammiersprachen			AIRL				
				Proprietär		Norm		(Eingangs)parameter			
								Beweg.	Applikation		
				KRL	RAPID	IRL	AIRL-DZL	Pfl.	Opt.	Pflicht	Optional
1	Bewegungen	Punkt-zu-Punkt-Bew.	PTP-Bewegung mit absoluten Werten	PTP	MoveJ; MoveAbsJ	MOVE PTP	MOVE PTP	x			
2	Bewegungen	Punkt-zu-Punkt-Bew.	PTP-Bewegung mit relativen Werten	PTP_REL	MoveJ Offs	MOVE_INC PTP	MOVE_INC PTP	x			
3	Bewegungen	Linearbewegung	Linearbewegung mit absoluten Werten	LIN	MoveL	MOVE LIN	MOVE LIN	x			
4	Bewegungen	Linearbewegung	Linearbewegung mit relativen Werten	LIN_REL	MoveL Offs	MOVE_INC LIN	MOVE_INC LIN	x			
5	Bewegungen	Kreisbewegungen	Kreisbewegung mit absoluten Werten	CIRC	MoveC	MOVE CIRCLE	MOVE CIRCLE	x			
6	Bewegungen	Kreisbewegungen	Kreisbewegung mit relativen Werten	CIRC_REL	MoveC Offs	MOVE_INC CIRCLE	MOVE_INC CIRCLE	x			
7	Bewegungen	Aktionen	Trigger zu Bewegungsanfang/-ende mit zeitlichem Offset	TRIGGER WHEN	TriggeJ(L)(C()	SYNACT WHEN *distCond*	SYNACT WHEN *distCond*				
8	Bewegungen	Aktionen	Trigger pfadabhängig setzen	TRIGGER WHEN PATH	TriggLIOs	SYNACT WHEN	SYNACT WHEN *timeCond*				
9	Logik	Bedingungen	IF mit einer Bedingung - Kurzdeklaration	-	IF *cond*	-	IF *cond*				
10	Logik	Bedingungen	IF Else - Startdeklaration	IF *cond* THEN	IF *cond* THEN	IF *cond* THEN	IF *cond* THEN				
11	Logik	Bedingungen	IF Else - Deklaration weiterer IF-Statements	-	ELSEIF	-	ELSEIF				
12	Logik	Bedingungen	IF Else - Deklaration des alternativen Statements	ELSE	ELSE	ELSE	ELSE				
13	Logik	Bedingungen	IF Else - Enddeklaration	ENDIF	ENDIF	ENDIF	ENDIF				
14	Logik	Bedingungen	Switch/Case - Startdeklaration	SWITCH	TEST	CASE	CASE				
15	Logik	Bedingungen	Switch/Case - Fallunterscheidungsdeklaration	CASE	CASE	WHEN	WHEN				
16	Logik	Bedingungen	Switch/Case - Deklaration Default-Wert	DEFAULT	DEFAULT	DEFAULT	DEFAULT				
17	Logik	Bedingungen	Switch/Case- Enddeklaration	ENDSWITCH	ENDTEST	ENDCASE	ENDCASE				
18	Logik	Bedingungen	Und	AND	AND	AND	AND				
19	Logik	Bedingungen	Oder	OR	OR	OR	OR				
20	Logik	Bedingungen	Exlusives Oder	EXOR	XOR	EXOR	EXOR				
21	Logik	Bedingungen	Nicht	NOT	NOT	NOT	NOT				
22	Logik	Bitweise Operatoren	Bitweise UND-Verknüpfung	B_AND	BitAnd	-	B_AND			VA-1;VA-1	
23	Logik	Bitweise Operatoren	Bitweise ODER-Verknüpfung	B_OR	BitOr	-	B_OR			VA-1;VA-1	
24	Logik	Bitweise Operatoren	Bitweise exklusive ODER-Verknüpfung	B_EXOR	BitXOr	-	B_EXOR			VA-1;VA-1	
25	Logik	Bitweise Operatoren	Bitweise Invertierung	B_NOT	BitNeg	-	B_NOT			VA-1;VA-1	
26	Logik	Bitweise Operatoren	Löschen eines Bits	-	BitClear	-	B_CLEAR			VA-1;VA-1	
27	Logik	Bitweise Operatoren	Bit setzen	-	BitSet	-	B_SET			VA-1;VA-1	
28	Logik	Bitweise Operatoren	Prüfen eines Bits	-	BitCheck	-	B_CHECK			VA-1;VA-1	
29	Logik	Bitweise Operatoren	Logischer Left-Shift eines Bytes	-	BitLSh	-	B_LSH			VA-1;VA-1	
30	Logik	Bitweise Operatoren	Logischer Right-Shift eines Bytes	-	BitRSh	-	B_RSH			VA-1;VA-1	
31	Logik	Programmablauf	Vorlaufstop deaktivieren	CONTINUE	-	-	CONTINUE				
32	Logik	Programmablauf	Programmabarbeitung bis zum Stillstand des Robotersystems anhalten	-	WaitTime \InPos, 0	-	WAIT_HALT				
33	Logik	Programmablauf	FOR-Schleife Startdeklaration	FOR	FOR	FOR	FOR				
34	Logik	Programmablauf	Deklaration Start- und Endwert mit Variable i und Werten von x bis n	i = x to n	i FROM x TO n	i := x TO n	i := x TO n				
35	Logik	Programmablauf	Schrittweite	STEP	STEP	STEP	STEP				
36	Logik	Programmablauf	FOR-Schleife - Enddeklaration	ENDFOR	ENDFOR	ENDFOR	ENDFOR				
37	Logik	Programmablauf	While-Schleife mit Bedingung *cond* - Startdeklaration	WHILE *cond*	WHILE *cond* DO	WHILE *cond*	WHILE *cond*				
38	Logik	Programmablauf	While-Schleife - Enddeklaration	ENDWHILE	ENDWHILE	ENDWHILE	ENDWHILE				
39	Logik	Programmablauf	Repeat-Schleife - Startdeklaration	REPEAT	-	REPEAT	REPEAT				
40	Logik	Programmablauf	Abbruchbedingung/Enddeklaration	UNTIL *cond*	-	UNTIL *cond*	UNTIL *cond*				
41	Logik	Programmablauf	Schleife verlassen	EXIT	-	-	EXIT				
42	Logik	Programmablauf	Warten auf Eintreten einer Bedingung	WAIT FOR	WaitUntil	WAIT FOR	WAIT FOR				
43	Logik	Programmablauf	Bestimmte Zeit warten	WAIT SEC	WaitTime	WAIT SEC	WAIT SEC				
44	Logik	Programmablauf	Sprunganweisung	GOTO	GOTO	GOTO	GOTO				
45	Logik	Programmablauf	Sprungmarkendeklaration	Name:	Name:	Name:	Name:				
46	Logik	Programmablauf	Programmhalt	HALT	Stop	PAUSE	PAUSE				
47	Logik	Programmablauf	Programmstop	-	Break	HALT	HALT				
48	Logik	Programmablauf	Rücksprung mit und ohne Werte	RETURN	RETURN	RETURN	RETURN				
49	Logik	Prozesssynchron.	Auf Task/Signal warten	-	WaitSyncTask	SEMA_WAIT	SEMA_WAIT				
50	Logik	Prozesssynchron.	Synchronisationssignal	-	-	SEMA_SIGNAL	SEMA_SIGNAL				
51	Logik	Prozesssynchron.	Einen Prozess starten	-	-	START	START				
52	Logik	Prozesssynchron.	Einen Prozess stoppen	-	-	STOP	STOP				
53	Logik	Prozesssynchron.	Einen Prozess fortsetzen	-	-	CONTINUE	CONTINUE				

Tabelle 19.2 - Befehls- und Datentypenübersicht der AIRL inkl. der adäquaten Instruktionen in KRL, RAPID und IRL

Nr.	Kategorie	Gruppe	Funktion/Kurzbeschreibung	Roboterprogrammiersprachen			AIRL				
				Proprietär		Norm		(Eingangs)parameter			
								Beweg.	Applikation		
				KRL	RAPID	IRL	AIRL-DZL	Pfl.	Opt.	Pflicht	Optional
54	Logik	Prozesssynchron.	Einen Prozess abbrechen	-	-	CANCEL	CANCEL				
55	Logik	Prozesssynchron.	Status eines Prozesses abfragen	-	-	GETSTATUS	GETSTATUS				
56	Logik	Prozesssynchron.	Variable sperren	-	-	LOCK	LOCK				
57	Logik	Prozesssynchron.	Variable freigeben	-	-	UNLOCK	UNLOCK				
58	Logik	Vergleichsoperatoren	Wert entspricht Bedingung	==	=	=	=				
59	Logik	Vergleichsoperatoren	Wert kleiner Bedingung	<	<	<	<				
60	Logik	Vergleichsoperatoren	Wert größer Bedingung	>	>	>	>				
61	Logik	Vergleichsoperatoren	Wert kleiner-gleich Bedingung	<=	<=	<=	<=				
62	Logik	Vergleichsoperatoren	Wert größer-gleich Bedingung	>=	>=	>=	>=				
63	Logik	Vergleichsoperatoren	Wert ungleich Bedingung	<>	<>	<>	<>				
64	Robotersp. Datent.	Geomet. Datentypen	Position	-	pos	POSITION	POSITION				
65	Robotersp. Datent.	Geomet. Datentypen	Orientierung	-	orient	ORIENTATION	ORIENTATION				
66	Robotersp. Datent.	Geomet. Datentypen	Pose (Position + Orientierung)	FRAME	pose	POSE	POSE				
67	Robotersp. Datent.	Geomet. Datentypen	Pose des TCPs inkl. Zusatzachsen	E6POS	robtarget	ROBTARGET	ROBTARGET				
68	Robotersp. Datent.	Geomet. Datentypen	Position Roboterachsen	AXIS	robjoint	MAIN_JOINT	MAIN_JOINT				
69	Robotersp. Datent.	Geomet. Datentypen	Position von externen Achsen	-	extjoint	ADD_JOINT	ADD_JOINT				
70	Robotersp. Datent.	Geomet. Datentypen	Position Roboter und Zusatzachsen	E6AXIS	jointtarget	JOINT	JOINT				
71	Robotersp. Datent.	Geomet. Datentypen	Werkzeugdaten	-	tooldata	-	TOOL			RS-1.1;RS-1.2;RS-1.3;RS-1.4	
72	Robotersp. Datent.	Geomet. Datentypen	Lastdaten eines Werkzeugs	-	loaddata	-	LOAD			RS-1.4.1;RS-1.4.2;RS-1.4.3	
73	Robotersp. Datent.	Geomet. Datentypen	Trägheitsmomente eines Werkzeugs	-	loaddata.x(y)(z)	-	INERTIA			RS-1.4.4.1;RS-1.4.4.2; RS-1.4.4.3	
74	Robotersp. Datent.	Orientierung	Orientierung mit Drehung um X,Y,Z-Achse	-	-	ORIRS	ORIRS				
75	Robotersp. Datent.	Orientierung	Orientierung mit Drehung um X,Y',Z"-Achse	-	-	ORIXYZ	ORIXYZ				
76	Robotersp. Datent.	Orientierung	Orientierung mit Drehung um Y,X',Z"-Achse	-	-	ORIYXZ	ORIYXZ				
77	Robotersp. Datent.	Orientierung	Orientierung mit Drehung um Z,Y',X"-Achse	-	OrientZYX	ORIZYX	ORIZYX				
78	Robotersp. Datent.	Orientierung	Orientierung mit Drehung um Z,Y',Z"-Achse	-	-	ORIZYZ2	ORIZYZ2				
79	Robotersp. Datent.	Orientierung	Orientierung um die Achse V im Referenzsystem	-	-	ORIVANG	ORIVANG				
80	Robotersp. Datent.	Orientierung	Orientierung anhand von Quaternion	-	orient	ORIQUAT	ORIQUAT				
81	Robotersp. Datent.	Orientierung	Orientierung anhand einer Rotationsmatrix	-	-	ORIMAT	ORIMAT				
82	Robotersp. Datent.	Umwandlung	Vorwärtskinematik berechnken	FORWARD	CalcRobT	TRAFO	TRAFO				
83	Robotersp. Datent.	Umwandlung	Rückwärtskinematik berechnen	INVERSE	CalcJointT	TRAFOINV	TRAFOINV				
84	Ein-/Ausgänge	Datentypen	Analoger Eingang	-	signalai	(INPUT) real	(INPUT) real				
85	Ein-/Ausgänge	Datentypen	Analoger Ausgang	-	signalao	(OUTPUT) real	(OUTPUT) real				
86	Ein-/Ausgänge	Datentypen	Digitaler Eingang	-	signaldi	(INPUT) BOOL	(INPUT) BOOL				
87	Ein-/Ausgänge	Datentypen	Digitaler Ausgang	-	signaldo	(OUTPUT) BOOL	(OUTPUT) BOOL				
88	Ein-/Ausgänge	Datentypen	Digitale Eingangsgruppe	-	signalgi	(INPUT) INT	(INPUT) INT				
89	Ein-/Ausgänge	Datentypen	Digitale Ausgangsgruppe	-	signalgo	(OUTPUT) INT	(OUTPUT) INT				
90	Ein-/Ausgänge	Datentypen	Erweiteter analoger Eingang	-	-	-	AI_E			EA-7.1;EA-7.2	
91	Ein-/Ausgänge	Datentypen	Erweiteter analoger Ausgang	-	-	-	AO_A			EA-8.1;EA-8.2	
92	Ein-/Ausgänge	Datentypen	Erweiteter digitaler Eingang	-	-	-	DI_E			EA-9.1;EA-9.2	
93	Ein-/Ausgänge	Datentypen	Erweiteter digitaler Ausgang	-	-	-	DO_A			EA-10.1;EA-10.2	
94	Ein-/Ausgänge	Datentypen	Erweiteter digitaler Gruppeneingang	-	-	-	DGI_E			EA-11.1;EA-11.2	
95	Ein-/Ausgänge	Datentypen	Erweiteter digitaler Gruppenausgang	-	-	-	DGO_A			EA-12.1;EA-12.2	
96	Ein-/Ausgänge	Eingänge	Zyklische Verarbeitung starten	ANIN ON	-	-	ANIN ON				
97	Ein-/Ausgänge	Eingänge	Zyklische Verarbeitung beenden	ANIN OFF	-	-	ANIN OFF				
98	Ein-/Ausgänge	Eingänge	Zugriff auf analogen Eingang via Nr.	=$ANIN[n]	-	-	:'=$ANIN[n]				
99	Ein-/Ausgänge	Eingänge	Zugriff auf digitalen Eingang via Nr.	=$IN[n	-	-	:'=$IN[n]				
100	Ein-/Ausgänge	Eingänge	Zugriff auf Eingang via Alias	=Signalname	:=Signalname	':=SIGNALNAME	':=SIGNALNAME				
101	Ein-/Ausgänge	Eingänge	Zugriff auf digitale Guppeneingänge über Alias	=Gr_signalname	:=Gr_signalname; GInputDnum(Gr_signalname	-	:=GR_SIGNALNAME				
102	Ein-/Ausgänge	Eingänge	Auf Eingang warten	Wait FOR	WaitUntil;WaitAI(DI)(GI)	Wait FOR	Wait FOR				
103	Ein-/Ausgänge	Ausgänge	Zyklisches Schreiben starten	ANOUT ON	-	-	ANOUT ON				
104	Ein-/Ausgänge	Ausgänge	Zyklisches Schreiben beenden	ANOUT OFF	-	-	ANOUT OFF				
105	Ein-/Ausgänge	Ausgänge	Analogen Ausgang via Nr. lesen	=$ANOUT[n]	-	-	:=$ANOUT[n]				
106	Ein-/Ausgänge	Ausgänge	Digitalen Ausgang via Nr. lesen	=$OUT[n]	-	-	:=$OUT[n]				

Tabelle 19.3 - Befehls- und Datentypenübersicht der AIRL inkl. der adäquaten Instruktionen in KRL, RAPID und IRL

Nr.	Kategorie	Gruppe	Funktion/Kurzbeschreibung	Roboterprogrammiersprachen			AIRL				
				Proprietär		Norm		(Eingangs)parameter			
								Beweg.		Applikation	
				KRL	RAPID	IRL	AIRL-DZL	Pfl.	Opt.	Pflicht	Optional
107	Ein-/Ausgänge	Ausgänge	Analogen Ausgang via Alias lesen	=Signalname	Aoutput(Signalname)	:=SIGNALNAME	:=SIGNALNAME				
108	Ein-/Ausgänge	Ausgänge	Digitalen Ausgang via Alias lesen	=Signalname	Doutput(Signalname)	:=SIGNALNAME	:=SIGNALNAME				
109	Ein-/Ausgänge	Ausgänge	Setzen eines analogen Ausgangs via Nr.	$ANOUT[n]=x	-	-	$ANOUT[n]:=x			VA-2	
110	Ein-/Ausgänge	Ausgänge	Setzen eines digitalen Ausgangs via Nr.	$OUT[n]=True	-	-	$OUT[n];			VA-3	
111	Ein-/Ausgänge	Ausgänge	Setzen eines analogen Ausgangs via Alias	Signalname=x	SetAO(Signalname)	SIGNALNAME:=x	SIGNALNAME:=x				
112	Ein-/Ausgänge	Ausgänge	Setzen eines digitalen Ausgangs via Alias	Signalname=True	Set(Signalname)	SIGNALNAME:=True	SIGNALNAME:=True				
113	Ein-/Ausgänge	Ausgänge	Zurücksetzen eines digitalen Ausgangs	$OUT[n]= False	Reset(Signalname)	SIGNALNAME:=False	SIGNALNAME:=False				
114	Ein-/Ausgänge	Ausgänge	Digitalen Gruppenausgang setzen	Gr_signalname=	SetGO	-	GR_SIGNALNAME:=				
115	Ein-/Ausgänge	Ausgänge	Impuls mit bestimmter Dauer setzen	PULSE	PulseDO	PULSE	PULSE				
116	Ein-/Ausgänge	Ausgänge	Wert invertieren an digitalem Ausgang	-	InvertDO	-	INVERT			EA-4	
117	Ein-/Ausgänge	Ausgänge	Auf Ausgang warten	Wait FOR	WaitUntil;WaitAO	Wait FOR	Wait FOR				
118	Ein-/Ausgänge	Allgemein	Eingänge mit Alias definieren	SIGNAL	-	INPUT	INPUT				
119	Ein-/Ausgänge	Allgemein	Ausgänge mit Alias definieren	SIGNAL	-	OUTPUT	OUTPUT				
120	Math. Funktionen	Grundrechenarten	Plus	+	+	+	+				
121	Math. Funktionen	Grundrechenarten	Minus	"-"	"-"	"-"	"-"				
122	Math. Funktionen	Grundrechenarten	Mal	*	*	*	*				
123	Math. Funktionen	Grundrechenarten	Geteilt	/	/	/	/				
124	Math. Funktionen	Grundrechenarten	Wert um 1 erhöhen	+1	+1;Incr	+1	+1;++				
125	Math. Funktionen	Grundrechenarten	Wert um 1 reduzieren	-1	-1:Decr	-1	+1;--				
126	Math. Funktionen	Grundrechenarten	Ganzzahlendivision	-	DIV	DIV	DIV				
127	Math. Funktionen	Grundrechenarten	Modulo berechnen	-	MOD	MOD	MOD				
128	Math. Funktionen	Geometrische Oper.	Vektoraddition	:	+	+	+				
129	Math. Funktionen	Geometrische Oper.	Vektorsubtraktion	-	"-"	"-"	"-"				
130	Math. Funktionen	Geometrische Oper.	Vektormultiplikation	-	*	*	*				
131	Math. Funktionen	Geometrische Oper.	Relativ. Translation und Rotation	-	-	@	@				
132	Math. Funktionen	Weitere Funktionen	Berechnen des Exponentialwerts	-	Exp	-	EXP			VA-2	
133	Math. Funktionen	Weitere Funktionen	Logarithmus Naturalis	-	-	-	LN			VA-2	
134	Math. Funktionen	Weitere Funktionen	Zufallszahl generieren	-	-	-	RND				
135	Math. Funktionen	Weitere Funktionen	Potenzwert	-	Pow	-	POW			VA-2	
136	Math. Funktionen	Weitere Funktionen	Quadratwurzel	SQRT	Sqrt	-	SQRT			VA-2	
137	Math. Funktionen	Weitere Funktionen	Runden	-	Round	ROUND	ROUND				
138	Math. Funktionen	Weitere Funktionen	Numerischen Werte abschneiden	-	Trunc	TRUNC	TRUNC				
139	Math. Funktionen	Weitere Funktionen	Betrag	ABS	Abs	ABS	ABS				
140	Math. Funktionen	Trigonometrie	Sinus	SIN	Sin	SIN	SIN				
141	Math. Funktionen	Trigonometrie	Cosinus	COS	Cos	COS	COS				
142	Math. Funktionen	Trigonometrie	Tangens	TAN	Tan	TAN	TAN				
143	Math. Funktionen	Trigonometrie	Arcussinus	-	ASin	ASIN	ASIN				
144	Math. Funktionen	Trigonometrie	Arcuscosinus	ACOS	ACos	ACOS	ACOS				
145	Math. Funktionen	Trigonometrie	Arcustangens	-	ATan	-	ATAN			VA-2	
146	Math. Funktionen	Trigonometrie	Arcustangens2	ATAN2	ATan2	ATAN2	ATAN2				
147	Math. Funktionen	Vektor-/Winkelrechn.	Rotation um die X-Achse einer Orientierung abrufen	-	EulerZYX(\X)	ANGLEX	ANGLEX				
148	Math. Funktionen	Vektor-/Winkelrechn.	Rotation um die Y-Achse einer Orientierung abrufen	-	EulerZYX(\Y)	ANGLEY	ANGLEY				
149	Math. Funktionen	Vektor-/Winkelrechn.	Rotation um die Z-Achse einer Orientierung abrufen	-	EulerZYX(\Z)	ANGLEZ	ANGLEZ				
150	Math. Funktionen	Vektor-/Winkelrechn.	Rotationsvektor	-	-	ROTAXIS	ROTAXIS				
151	Math. Funktionen	Vektor-/Winkelrechn.	Rotationswinkel	-	-	ROTANGLE	ROTANGLE				
152	Math. Funktionen	Vektor-/Winkelrechn.	Skalarprodukt zweier Vektoren	-	DotProd	DOT	DOT				
153	Math. Funktionen	Vektor-/Winkelrechn.	Kreuzprodukt zweier Vektoren	-	*	CROSS	CROSS				
154	Math. Funktionen	Vektor-/Winkelrechn.	Einheitsvektor einer Rotationsmatrix (1-Zeile)	-	-	VECTORN	VECTORN				
155	Math. Funktionen	Vektor-/Winkelrechn.	Einheitsvektor einer Rotationsmatrix (2-Zeile)	-	-	VECTORO	VECTORO				
156	Math. Funktionen	Vektor-/Winkelrechn.	Einheitsvektor einer Rotationsmatrix (3-Zeile)	-	-	VECTORA	VECTORA				
157	Math. Funktionen	Vektor-/Winkelrechn.	1. Parameter der Quaternion-Spezifikation	-	orient.q1	QUATA	QUATA				
158	Math. Funktionen	Vektor-/Winkelrechn.	2. Parameter der Quaternion-Spezifikation	-	orient.q2	QUATB	QUATB				
159	Math. Funktionen	Vektor-/Winkelrechn.	3. Parameter der Quaternion-Spezifikation	-	orient.q3	QUATC	QUATC				
160	Math. Funktionen	Vektor-/Winkelrechn.	4. Parameter der Quaternion-Spezifikation	-	orient.q4	QUATD	QUATD				
161	Math. Funktionen	Vektor-/Winkelrechn.	Abstand zwischen zwei Punkten	-	Distance	DISTANCE	DISTANCE				

Tabelle 19.4 - Befehls- und Datentypenübersicht der AIRL inkl. der adäquaten Instruktionen in KRL, RAPID und IRL

Nr.	Kategorie	Gruppe	Funktion/Kurzbeschreibung	Roboterprogrammiersprachen			AIRL				
				Proprietär		Norm		(Eingangs)parameter			
								Beweg.		Applikation	
				KRL	RAPID	IRL	AIRL-DZL	Pfl.	Opt.	Pflicht	Optional
162	Math. Funktionen	Vektor-/Winkelrechn.	Invertieren einer Koordinatenverschiebung	INV_POS	PoseInv	INV	INV				
163	Variablen	Deklarationen	Standarddeklaration einer Variable	DECL	(LOCAL) VAR	VAR	VAR				
164	Variablen	Deklarationen	Persistente Variable deklarieren	DECL =	PERS	PERMANENT	PERMANENT				
165	Variablen	Deklarationen	Deklaration einer globalen Variable	DECL GLOBAL	VAR	SHARED	SHARED				
166	Variablen	Deklarationen	Variable einen Wert zuweisen	=	:=	:=	:=				
167	Variablen	Strukturen	Deklaration von Strukturen	STRUC	RECORD	TYPE	TYPE				
168	Variablen	Strukturen	Weitere Startdeklaration	-	-	RECORD	RECORD				
169	Variablen	Strukturen	Enddeklaration einer Struktur	-	ENDRECORD	ENDRECORD	ENDRECORD				
170	Variablen	Exportieren/Imp.	Variable in andere Programme exportieren	-	-	EXPORT	EXPORT				
171	Variablen	Exportieren/Imp.	Variable aus anderen Programmen/Datenlisten	IMPORT	-	IMPORT	IMPORT				
172	Variablen	Datentypen	Ganzzahl	INT	num	INT	INT				
173	Variablen	Datentypen	Gleitkommazahl	REAL	num	REAL	REAL				
174	Variablen	Datentypen	Boolescher Wert	BOOL	bool	BOOL	BOOL				
175	Variablen	Datentypen	Zeichen	CHAR	-	CHAR	CHAR				
176	Variablen	Datentypen	Zeichenkette	-	string	STRING	STRING				
177	Variablen	Datentypen	Konstante deklarieren	CONST	CONST	CONST	CONST				
178	Variablen	Konvertierung	Ganzzahl in Realzahl konvertieren	REAL = INT	-	FLOAT	FLOAT				
179	Variablen	Konvertierung	Aufzählungsdatentyp	ENUM	-	-	ENUM				
180	Variablen	Datentypen	Implizite Deklaration von TCP-Pose Variablen anschalten	-	-	DECLON	DECLON				
181	Variablen	Datentypen	Implizite Deklaration ausschalten	-	-	DECLOFF	DECLOFF				
182	Variablen	Datentypen - Listen	Deklaration von Listen eines Datentyps	-	-	LIST OF	LIST OF				
183	Variablen	Datentypen - Listen	Listenelement einfügen	-	-	LISTINS	LISTINS				
184	Variablen	Datentypen - Listen	Listenelement hinzufügen	-	-	LISTADD	LISTADD				
185	Variablen	Datentypen - Listen	Listenelement entfernen	-	-	LISTDEL	LISTDEL				
186	Variablen	Datentypen - Listen	Listenlänge abfragen	-	-	LISTLENGTH	LISTLENGTH				
187	Variablen	Datentypen - Listen	Index eines Listeneintrags abfragen	-	-	LISTINDEX	LISTINDEX				
188	Variablen	Datentypen - Arrays	Array mit Länge n und Datentyp bool	BOOL name[n]	bool name[n]	ARRAY [n] OF BOOL	ARRAY [n] OF BOOL				
189	Variablen	Datentypen - Arrays	Dimension abrufen	-	Dim	-	ARRAY_LEN			VA-4	
190	Sonstiges	Systemspez. Deklarat.	Startdeklaration eines Programms	DEF	MODULE	PROGRAMM	PROGRAMM				
191	Sonstiges	Systemspez. Deklarat.	Enddeklaration eines Programms	END	ENDMODULE	ENDPROGRAMM	ENDPROGRAMM				
192	Sonstiges	Systemspez. Deklarat.	Startdeklaration einer Prozedur	-	PROC	PROCEDURE	PROCEDURE				
193	Sonstiges	Systemspez. Deklarat.	Enddeklaration einer Prozedur	-	ENDPROC	ENDPROC	ENDPROC				
194	Sonstiges	Systemspez. Deklarat.	Startdeklaration einer parallelen Prozedur	-	-	PROCESS	PROCESS				
195	Sonstiges	Systemspez. Deklarat.	Enddeklaration einer parallelen Prozedur	-	-	ENDPROCESS	ENDPROCESS				
196	Sonstiges	Systemspez. Deklarat.	Startdeklaration einer Funktion	DEFFCT	FUNC	FUNCTION	FUNCTION				
197	Sonstiges	Systemspez. Deklarat.	Enddeklaration einer Funktion	ENDFCT	ENDFUNC	ENDFCT	ENDFCT				
198	Sonstiges	Systemspez. Deklarat.	Deklaration für Beginn der Befehlsausführung	-	-	BEGIN	BEGIN				
199	Sonstiges	Interrupts	Interrupt deklarieren	INTERRUPT DECL	intnum;CONNECT;Ipers;Isignalx;	INTERRUPT DECL	INTERRUPT DECL				
200	Sonstiges	Interrupts	Interrupt aktivieren	INTERRUPT ON	IWatch	INTERRUPT ON	INTERRUPT ON				
201	Sonstiges	Interrupts	Interrupt deaktivieren	INTERRUPT OFF	ISleep	INTERRUPT OFF	INTERRUPT OFF				
202	Sonstiges	Interrupts	Interrupt sperren	INTERRUPT DISABLE	IDisable	INTERRUPT DISABLE	INTERRUPT DISABLE				
203	Sonstiges	Interrupts	Interrupt freigeben	INTERRUPT ENABLE	IEnable	INTERRUPT ENABLE	INTERRUPT ENABLE				
204	Sonstiges	Interrupts	Roboter im Interrupt stoppen	BRAKE	StopMove	BRAKE	BRAKE				
205	Sonstiges	Interrupts	Wiederaufnahme der Bewegung	-	StartMove	-	STARTMOVE				
206	Sonstiges	Interrupts	Speichern der aktuellen Bewegungsbahn nach Interrupt/Fehler	-	StorePath	-	SAVEPATH				
207	Sonstiges	Interrupts	Wiederherstellen der gesp. Bewegungsbahn an Position vor Interrupt	$POS_INT	RestoPath	-	RELOADPATH				
208	Sonstiges	Interrupts	Interrupt(s) abbrechen und am Unterbrechungsort im Programm der Interruptdeklaration fortsetzen	RESUME	-	RESUME	RESUME				
209	Sonstiges	Interrupts	Interrupt am Ort des Interruptaufrufs fortsetzen	RETURN;END;	RETURN;ENDTRAP	RETURN;ENDPROC	RETURN;ENDPROC				
210	Sonstiges	Fehlerhandling	Startdeklaration des Fehlerhandlings	-	ERROR	ERROR	ERROR				
211	Sonstiges	Fehlerhandling	Enddeklaration des Fehlerhandlings	-	-	ENDERROR	ENDERROR				
212	Sonstiges	Fehlerhandling	Wiederaufnahme der Programmabarbeitung am Fehlerort	-	Retry	-	RETRY				

Tabelle 19.5 - Befehls- und Datentypenübersicht der AIRL inkl. der adäquaten Instruktionen in KRL, RAPID und IRL

Nr.	Kategorie	Gruppe	Funktion/Kurzbeschreibung	Roboterprogrammiersprachen			AIRL				
				Proprietär		Norm		(Eingangs)parameter			
								Beweg.		Applikation	
				KRL	RAPID	IRL	AIRL-DZL	Pfl.	Opt.	Pflicht	Optional
213	Sonstiges	Fehlerhandling	Neustart der Bewegung und der Programmabarbeitung nach Fehler	-	StartMoveRetry	-	RETRY_STARTMOVE				
214	Sonstiges	Fehlerhandling	Wiederaufnahme der Programmabarbeitung nach der fehlerverursachenden Methode	-	TryNext	-	RETRY_SKIP				
215	Sonstiges	Fehlerhandling	Restliche Anzahl an möglichen Wiederholungen	-	Remaining Retry	-	RETRY_REMAINING				
216	Sonstiges	Fehlerhandling	Reset der Anzahl der möglichen Wiederholungen	-	ResetRetryCount	-	RETRY_RESET				
217	Sonstiges	Events	Datentyp einer Eventroutine	-	event_type	-	EVENT_TYPE			EV-1	
218	Sonstiges	Events	Abrufen eines Eventtypen		EventType		GET_EVENT_TYPE				
219	Sonstiges	Stringhandling	Deklarierte Länge abfragen	StrDeclLen	-	SIZE	SIZE				
220	Sonstiges	Stringhandling	Initialisierte Länge abfragen	StrLen	StrLen	LENGTH	LENGTH				
221	Sonstiges	Stringhandling	Variable leeren	StrClear	Clear	-	CLEAR			VA-5	
222	Sonstiges	Stringhandling	Position einer Sequenz finden	StrFind	StrFind	LOCATE	LOCATE				
223	Sonstiges	Stringhandling	String um Zeichen erweitern	StrAdd	-	APPENDC	APPENDC				
224	Sonstiges	Stringhandling	Ein Zeichen im String ersetzen	-	-	SETCHR	SETCHR				
225	Sonstiges	Stringhandling	String um String erweitern	StrAdd	-	CONCATS	CONCATS				
226	Sonstiges	Stringhandling	Ein Zeichen aus String extrahieren	-	-	GETCHR	GETCHR				
227	Sonstiges	Stringhandling	Ein Zeichen im String ersetzen	-	-	SETCHR	SETCHR				
228	Sonstiges	Stringhandling	Zeichenfolge aus String extrahieren	-	StrPart	EXTRACTS	EXTRACTS				
229	Sonstiges	Stringhandling	Teilstring löschen	-	-	DELETEC	DELETEC				
230	Sonstiges	Stringhandling	Stringvariablen vergleichen	StrComp	-	COMPARE	COMPARE				
231	Sonstiges	Stringhandling	Zeichen aus ASCII-Code generieren	-	-	CHR	CHR				
232	Sonstiges	Stringhandling	ASCII-Code eines Zeichens abfragen	-	-	ORD	ORD				
233	Sonstiges	Datei/Seriellen Kanal	Öffnen einer Datei oder eines seriellen Kanals	COPEN	Open	OPEN	OPEN				
234	Sonstiges	Datei/Seriellen Kanal	Schließen einer Datei oder eines Kanals	CCLOSE	Close	CLOSE	CLOSE				
235	Sonstiges	Datei/Seriellen Kanal	Datei oder seriellen Kanal lesen	CREAD	ReadStr;ReadAnyBin	READ	READ				
236	Sonstiges	Datei/Seriellen Kanal	Zeilenweises Lesen einer Datei	-	-	READLN	READLN				
237	Sonstiges	Datei/Seriellen Kanal	Datei oder seriellen Kanal schreiben	CWRITE	WriteStrBin;WriteAnyBin	WRITE	WRITE				
238	Sonstiges	Datei/Seriellen Kanal	Zeilenweises Schreiben einer Datei	-	-	WRITELN	WRITELN				
239	Sonstiges	Kommentare	Kommentarbeginn Deklaration	;	!	{	{				
240	Sonstiges	Kommentare	Kommentarende Deklaration	-	-	}	}				
241	Automotive	Basisoperationen	Information über Jobstart an übergeordnete Steuerung senden	Job Started	MoveJ(L)_Job \Started; Job \Started	-	JOB STARTED		x	AB-1	
242	Automotive	Basisoperationen	Information über Jobende an übergeordnete Steuerung senden	Job Done	MoveJ(L)_Job \Done; Job \Done	-	JOB DONE		x	AB-1	
243	Automotive	Basisoperationen	Anfrage nach Jobparametern	Job Request	Job_Request	-	JOB REQUEST		x	AB-1	AB-2;AB-3;AB-4
244	Automotive	Basisoperationen	Alle Jobs auf erledigt setzen	Job Clear All	Job \ClearAll	-	JOB CLEAR_All		x	AB-5	
245	Automotive	Basisoperationen	Freigabe für Arbeitsbereich anfragen	Area Request	MoveJ(L)_AreaReq;	-	AREA REQUEST		x	AB-5	AB-4
246	Automotive	Basisoperationen	Arbeitsbereich freigeben	Area Release	MoveJ(L)_AreaRel; AreaRel	-	AREA RELEASE		x	AB-5	AB-4
247	Automotive	Basisoperationen	Freigabe für Kollisionsbereich anfragen	CollZone Request	MoveJ(L)_CollZoReq; CollZoReq	-	COLLZONE REQUEST		x	AB-6	AB-7
248	Automotive	Basisoperationen	Kollisionsbreich freigeben	CollZone Release	MoveJ(L)_CollZoRel; CollZoRel	-	COLLZONE RELEASE		x	AB-6	AB-7
249	Automotive	Basisoperationen	Kommunikation mit SPS bzw. übergeordneter Steuerung zur Programmsynchronisation	PLCCom	PLC_Comm	-	SCTRL_COM		x	AB-8	AB-9;AB-10;AB-8
250	Automotive	Punktschweißen	Schweißzange-Datentyp	-	weld_gun_dataGun	-	WELDGUN			AP-1.1...AP-1.10	
251	Automotive	Punktschweißen	Schweißpunkt-Datentyp	-	-	-	WELDPOINT			AP-2;AP-2.1;AP-2.2	
252	Automotive	Punktschweißen	Schweißpunkt setzen	Swp Spotpoint	SpotL	-	SPOT PTP/LIN	x		AB-11;AP-1;AP-2;AP-3	AP-5...AP-8
253	Automotive	Punktschweißen	Vorpositionierung	Swp Positioning	MoveL(J)G	-	SPOT PREPO PTP/LIN	x		AP-1;AP-3	
254	Automotive	Punktschweißen	Zangeninitialisierung	Swp Init	-	-	SPOT INIT			AP-1;AP-3	
255	Automotive	Punktschweißen	Startfräsen der Zange	Swp Firstdress	-	-	SPOT FIRSTDRESS			AP-1;AP-3	
256	Automotive	Punktschweißen	Fräsen der Zange	Swp Dress	swp_TipDress	-	SPOT DRESS			AP-1;AP-3	
257	Automotive	Punktschweißen	Referenzieren der Zange	Swp GunReference	swp_GunReference	-	SPOT GUNREF			AP-1;AP-3	
258	Automotive	Punktschweißen	Elektrodenwiderstand prüfen	Swp ResistorCheck	swp_MeasureResistance	-	SPOT RESISTOR_CHECK			AP-1;AP-3	
259	Automotive	Punktschweißen	Elektrodenwiderstand skalieren	Swp ResistorScale	-	-	SPOT RESISTOR_SCALE			AP-1;AP-3	
260	Automotive	Punktschweißen	Ansteuerung des Kappenwechslers	Swp TipChangeUnit	swp_TipChange	-	SPOT TIP_CHANGE			AP-1;AP-9	
261	Automotive	Punktschweißen	Kappenprüfung	Swp TipCheck	lTip_Check_High/Low	-	SPOT TIP_CHECK			AP-1;AP-10	

Tabelle 19.6 - Befehls- und Datentypenübersicht der AIRL inkl. der adäquaten Instruktionen in KRL, RAPID und IRL

Nr.	Kategorie	Gruppe	Funktion/Kurzbeschreibung	Roboterprogrammiersprachen			AIRL				
				Proprietär		Norm		(Eingangs)parameter			
								Beweg.		Applikation	
				KRL	RAPID	IRL	AIRL-DZL	Pfl.	Opt.	Pflicht	Optional
262	Automotive	Punktschweißen	Anpressen der Elektrodenkappe	Swp TipPress	swp_TipPress	-	SPOT TIP_PRESS			AP-1;AP-3;AP-11	AP-6
263	Automotive	Punktschweißen	Bestätigung des Kappenwechsels	Swp QuitTipChange	ICounterReset	-	SPOT TIP_CHANGE_CFRM			AP-1	
264	Automotive	Punktschweißen	Kühlwasser ein- und ausschalten	Swp Water	ITip_WaterOn/Off	-	SPOT WATER			AP-1;AP-12	
265	Automotive	Punktschweißen	Öffnen/Schließen der Spanner am Fräser	Swp MobTipDresser	swp_MobTipDress	-	SPOT MOB_TIPDRESS			AP-13;AP-14	
266	Automotive	Punktschweißen	Einlernen Zangenausgleich	Swp EqualCalib	swp_EqualCalib	-	SPOT CALIB_EQ			AP-1;AP-15	
267	Automotive	Punktschweißen	Aufruf Serviceprogramm	Swp ServiceProgramm	-	-	SPOT SERVICE			AP-1;AP-16	
268	Automotive	Punktschweißen	Aufruf Serviceprogramm für mobilen Kappenfräser	ServiceProgramMobDresser	-	-	SPOT SERVICE_MTD			AP-13;AP-17	
269	Automotive	Kleben	Klebesystem-Datentyp	-	glue_sys_data	-	GLUESYS			AK-1.1...AK-1.15	
270	Automotive	Kleben	Auswahl des Klebesystems und Start der Applikation	GI Select	GlueSelect	-	GLUE SELECT			AK-1;AK-2;AK-3	AK-4;AK-5;AK-6
271	Automotive	Kleben	Ende Klebeapplikation	GI End	GlueEnd	-	GLUE END			AK-7	
272	Automotive	Kleben	Spühlfreigabe eines Klebesystems setzen	GI PurgePermission	GluePurgeInhibit \Off	-	GLUE PURGE_PERMISSION			AK-1	
273	Automotive	Kleben	Spühlfreigabe eines Klebesystems zurücksetzen	GI PurgeInhibit	GluePurgeInhibit \On	-	GLUE PURGE_INHIBT			AK-1	
274	Automotive	Kleben	Spühlen der Klebedüse	GI Purge	GluePurge	-	GLUE PURGE			AK-1;AK-8	AK-9;AK-10
275	Automotive	Kleben	Klebesystem befüllen	GI Fill	GlueFill	-	GLUE FILL			AK-1	AK-11
276	Automotive	Kleben	Härter sperren	GI BoostLock	GlueL \BoostLock	-	GLUE BOOSTLOCK			AK-1;AK-12	AK-13
277	Automotive	Kleben	Erhöhen des Materialflusses	GI SpeedUp	-	-	GLUE SPEEDUP			AK-1;AK-14	AK-13
278	Automotive	Kleben	Abstandsregelung der Düse einschalten	GI WSN_On LIN	GlueFunc \Wsn_on	-	GLUE WSN_ON		x	AK-1;AK-3;AK-8	
279	Automotive	Kleben	Abstandsregelung der Düse ausschalten	GI WSN_Off LIN	GlueFunc \Wsn_off	-	GLUE WSN_OFF		x	AK-1;AK-3	
280	Automotive	Kleben	Stationäre Kamera zur Prüfung einschalten	-	GlueFunc \CamRec_on	-	GLUE CAM_ON		x	AK-1;AK-3;AK-8	
281	Automotive	Kleben	Stationäre Kamera zur Prüfung ausschalten	-	GlueFunc \CamRec_off	-	GLUE CAM_OFF		x	AK-1;AK-3	
282	Automotive	Kleben	Abstandsregelung kalibrieren	GI WSN_Calib	GlueFunc \Wsn_calib	-	GLUE WSN_CALIB		x	AK-1	
283	Automotive	Kleben	Lineare Klebebewegung starten	GI On Lin	GlueL \On	-	GLUE ON		x	AB-11;AK-15;AK-16;AK-3; AK-1;AK-8	AK-17;AK-18; AK-19
284	Automotive	Kleben	Lineare Klebebewegung beenden	GI Off Lin	GlueL \Off	-	GLUE OFF		x	AB-11;AK-15;AK-16;AK-3; AK-1;AK-8	AK-17;AK-18; AK-19
285	Automotive	Kleben	Lineare Klebebewegung durchführen	GL Move LIN	GlueL	-	GLUE LIN/CIRC	x		AB-11;AK-15;AK-16;AK-3; AK-1	AK-17;AK-20
286	Automotive	Kleben	Statischen Klebepunkt setzen	GL Point LIN	GluePoint	-	GLUE POINT LIN	x		AB-11;AK-15;AK-16;AK-3; AK-1;AK-8;AK-21;AK-22	AK-18;AK-23
287	Automotive	Handling	Greifer-Datentyp	-	-	-	GRIP_PER			AH-1.1;AH-1.2;AH-1.3;AH-	
288	Automotive	Handling	Greiferaktuator-Datentyp	-	grp_act	-	GRIP_ACT			AH-1.3.1...AH-1.3.7	
289	Automotive	Handling	Greiferventil-Datentyp	-	-	-	GRIP_ACT_VALVE			AH-1.3.6.1...AH-1.3.6.4	
290	Automotive	Handling	Greiferzylinder-Datentyp	-	-	-	GRIP_ACT_CYLINDER			AH-1.3.7.1...AH-1.3.7.4	
291	Automotive	Handling	Bauteilkontrolle-Datentyp	-	grp_part	-	GRIP_PRTCHK			AH-1.4.1;AH-1.4.2	
292	Automotive	Handling	Initialisierung der EA-Schnittstellen zur überg. Steuerung	Grp Init	Grplnit	-	GRIP INIT			AH-1	
293	Automotive	Handling	Stellglieder in Rück-Position setzen	Grp PosRet	Grp(L)(J) PosRet	-	GRIP POS_RET		x	AH-1.3[ ]	AH-2;AH-3
294	Automotive	Handling	Stellglieder in Vor-Position setzen	Grp PosAdv	Grp(L)(J) PosAdv	-	GRIP POS_ADV		x	AH-1.3[ ]	AH-2;AH-3
295	Automotive	Handling	Stellglieder in Rück-Position setzen (ohne Überwachung)	Grp PosRetNoChk	Grp(L)(J) PosRetNoChk	-	GRIP POS_RET_NC		x	AH-1.3[ ]	AH-2;AH-3
296	Automotive	Handling	Stellglieder in Vor-Position setzen (ohne Überwachung)	Grp PosAdvNoChk	Grp(L)(J) PosAdvNoChk	-	GRIP POS_ADV_NC		x	AH-1.3[ ]	AH-2;AH-3
297	Automotive	Handling	Bauteilkontrollen abfragen	Grp PartChk	Grp(J)(L)PartChk	-	GRIP PART_CHECK		x	AH-4;AH-1.4[ ]	AH-5
298	Automotive	Werkzeug wechseln	Werkzeughälfte-Datentyp	-	-	-	TC_TOOL			AW-1.1...AW-1.9	
299	Automotive	Werkzeug wechseln	Werkzeugbahnhof-Datentyp	-	Station_data	-	TC_STATION			AW-2.1...AW-2.9	
300	Automotive	Werkzeug wechseln	Prozedur zum Andocken eines Werkzeugs	ToolChg_Move Dock	ToolChg_Move \Dock	-	TOOLCH DOCK LIN	x		AW-1;AW-2	
301	Automotive	Werkzeug wechseln	Prozedur zum Abdocken eines Werkzeugs	ToolChg_Move Undock	ToolChg_Move \UnDock	-	TOOLCH UNDOCK LIN	x		AW-1;AW-2	
302	Automotive	Werkzeug wechseln	Überprüfen des Toolcodes der Werkzeugseite	ToolChg_Cmd Check_Tool_Code	ToolChg_Check	-	TOOLCH CODE_CHECK			AW-1	
303	Automotive	Werkzeug wechseln	Öffnen der Bahnhofsabdeckung	ToolChg_Cmd Open_Cover	ToolChg_CMD \OpenCover	-	TOOLCH OPEN_COVER			AW-2;AW-3	AW-4
304	Automotive	Werkzeug wechseln	Schließen der Bahnhofsabdeckung	ToolChg_Cmd Close_Cover	ToolChg_CMD \CloseCover	-	TOOLCH CLOSE_COVER			AW-2;AW-3	AW-4
305	Automotive	Werkzeug wechseln	Überprüfen der Bahnhofsbelegung	ToolChg_Cmd OccupiedCheck	ToolChg_CMD \StatFree(Occ)	-	TOOLCH STATFREE_CHECK			AW-3	

Tabelle 20.1 – Liste, Deklaration und Aufbau der AIRL-Eingangsparameter und zugehöriger Datentypen

Nr	Kategorie	Gruppe	Identifier	Bezeichnung	Datentyp	Beschreibung	Deklaration/Aufbau
1	Variablen	Datentypen	VA-1	Bit_Value	INT	Siehe IRL	
2	Variablen	Datentypen	VA-2	Value	REAL	Siehe IRL	
3	Variablen	Datentypen	VA-3	Value	BOOL	Siehe IRL	
4	Variablen	Datentypen	VA-4	Value	ARRAY	Siehe IRL	
5	Variablen	Datentypen	VA-5	Str_Value	STRING	Siehe IRL	
6	Robotersp. Datentypen	Geometrische Datentypen	RS-1	Tool_data	TOOL	Werkzeug-Datentyp	Description;Mounted;Tool_frame;Load_data
7	Robotersp. Datentypen	Geometrische Datentypen	RS-1.1	Description	STRING	Bezeichnung des Werkzeugs	
8	Robotersp. Datentypen	Geometrische Datentypen	RS-1.2	Mounted	BOOL	Werkzeug am Roboter angeschlossen	
9	Robotersp. Datentypen	Geometrische Datentypen	RS-1.3	Tool_frame	POSE	Werkzeugkoordinatensystem in Position und Orientierung	
10	Robotersp. Datentypen	Geometrische Datentypen	RS-1.4	Load_data	LOAD	Lastdaten-Datentyp	
11	Robotersp. Datentypen	Geometrische Datentypen	RS-1.4.1	Mass	REAL	Gewicht des Werkzeugs in kg	
12	Robotersp. Datentypen	Geometrische Datentypen	RS-1.4.2	CoG	POSITION	Schwerpunkt des Werkzeugs	
13	Robotersp. Datentypen	Geometrische Datentypen	RS-1.4.3	AoM	ORIENTATION	Drehmomentachsen des Werkzeugs	
14	Robotersp. Datentypen	Geometrische Datentypen	RS-1.4.4	Inertia	INERTIA	Massenträgheitsmoment Datentyp	X;Y;Z
15	Robotersp. Datentypen	Geometrische Datentypen	RS-1.4.4.1	X	REAL	Trägheitsmoment um die X-Achse in kgm²	
16	Robotersp. Datentypen	Geometrische Datentypen	RS-1.4.4.2	Y	REAL	Trägheitsmoment um die Y-Achse in kgm²	
17	Robotersp. Datentypen	Geometrische Datentypen	RS-1.4.4.3	Z	REAL	Trägheitsmoment um die Z-Achse in kgm²	
18	Sonstiges	Events	EV-1	Event_Type	EVENT_TYPE	Ereignis-Datentyp	E_None;E_Power_On;E_Power_Off;E_Start;E_Stop;E_QStop; E_Restart;E_Reset;E_Step;E_UserChPtr;E_UserChRtn;
19	Ein-/Ausgänge	Datentypen	EA-1	AInput	REAL	Siehe IRL	
20	Ein-/Ausgänge	Datentypen	EA-2	AOutput	REAL	Siehe IRL	
21	Ein-/Ausgänge	Datentypen	EA-3	DInput	BOOL	Siehe IRL	
22	Ein-/Ausgänge	Datentypen	EA-4	DOutput	BOOL	Siehe IRL	
23	Ein-/Ausgänge	Datentypen	EA-5	DGInput	INT	Siehe IRL	
24	Ein-/Ausgänge	Datentypen	EA-6	DGOutput	INT	Siehe IRL	
25	Ein-/Ausgänge	Datentypen	EA-7	AInput_E	AI_E	Datentyp für erweiterten analogen Eingang	
26	Ein-/Ausgänge	Datentypen	EA-7.1	Description	STRING	Bezeichnung des analogen Eingangs	
27	Ein-/Ausgänge	Datentypen	EA-7.2	AInput	REAL	Analoger Eingang	
28	Ein-/Ausgänge	Datentypen	EA-8	AOutput_E	AO_E	Datentyp für erweiterten analogen Ausgang	
29	Ein-/Ausgänge	Datentypen	EA-8.1	Description	STRING	Bezeichnung des analogen Ausgangs	
30	Ein-/Ausgänge	Datentypen	EA-8.2	DOutput	REAL	Analoger Ausgang	
31	Ein-/Ausgänge	Datentypen	EA-9	DInput_E	DI_E	Datentyp für erweiterten digitalen Eingang	
32	Ein-/Ausgänge	Datentypen	EA-9.1	Description	STRING	Bezeichnung des digitalen Eingangs	
33	Ein-/Ausgänge	Datentypen	EA-9.2	DInput	BOOL	Digitaler Eingang	
34	Ein-/Ausgänge	Datentypen	EA-10	DOutput_E	DO_E	Datentyp für erweiterten digitalen Ausgang	
35	Ein-/Ausgänge	Datentypen	EA-10.1	Description	STRING	Bezeichnung des digitalen Ausgangs	
36	Ein-/Ausgänge	Datentypen	EA-10.2	DOutput	BOOL	Digitaler Ausgang	
37	Ein-/Ausgänge	Datentypen	EA-11	DGInput_E	DGI_E	Datentyp für erweiterten digitalen Gruppeneingang	
38	Ein-/Ausgänge	Datentypen	EA-11.1	Description	STRING	Bezeichnung des digitalen Gruppeneingangs	
39	Ein-/Ausgänge	Datentypen	EA-11.2	DGInput	INT	Digitaler Gruppeneingang	
40	Ein-/Ausgänge	Datentypen	EA-12	DGOutput_E	DGO_E	Datentyp für erweiterten digitalen Gruppenausgang	
41	Ein-/Ausgänge	Datentypen	EA-12.1	Description	STRING	Bezeichnung des digitalen Gruppenausgangs	
42	Ein-/Ausgänge	Datentypen	EA-12.2	DGOutput	INT	Digitaler Gruppenausgang	
43	Automotive	Basisoperationen	AB-1	JobNr	INT	Nummer des aktuellen Auftrags	
44	Automotive	Basisoperationen	AB-2	TypRequest	BOOL	Abfrage des Fahrzeugtyps	
45	Automotive	Basisoperationen	AB-3	UserRequest	BOOL	Abfrage einer Detailunterscheidung	
46	Automotive	Basisoperationen	AB-4	Abort	ABORT	Verhalten bei Abbruch mit Roboterposition und Programmaufruf	Position;Programm
47	Automotive	Basisoperationen	AB-4.1	Position	ROBTARGET	Zielposition bei Programmabbruch	
48	Automotive	Basisoperationen	AB-4.2	Programm	STRING	Programmaufruf bei Programmabbruch	
49	Automotive	Basisoperationen	AB-5	Area_Num	INT	Bereich Nr. zwischen zwei Robotern	
50	Automotive	Basisoperationen	AB-6	Zone_Num	INT	Zonen Nr. in der Roboterzelle	

Tabelle 20.2 - Liste, Deklaration und Aufbau der AIRL-Eingangsparameter und zugehöriger Datentypen

Nr	Kategorie	Gruppe	Identifier	Bezeichnung	Datentyp	Beschreibung	Deklaration/Aufbau
51	Automotive	Basisoperationen	AB-7	SCtrl_AreaExt	BOOL	Alternativen übergeordneten Steuerungsbereich wählen	
52	Automotive	Basisoperationen	AB-8	SCtrl_Output Act	OUTPUT_ACT	Ausgangs-Aktion einer übergeordneten Steuerung	Action;Output_Num
53	Automotive	Basisoperationen	AB-8.1	Action	ENUM	Definition der durchzuführenden Aktion	Set_Output;Reset_Output
54	Automotive	Basisoperationen	AB-8.2	Output_Num	INT	Ausgangs-Nr. an der übergeordneten Steuerung	
55	Automotive	Basisoperationen	AB-9	SCtrl_Input_Act	INPUT_ACT	Eingangs-Aktion einer übergeordneten Steuerung	Action;Input_Num
56	Automotive	Basisoperationen	AB-9.1	Action	ENUM	Definition der durchzuführenden Aktion	Wait_Input_On;Wait_Input_Off
57	Automotive	Basisoperationen	AB-9.2	Input_Num	INT	Eingangs-Nr. an der übergeordneten Steuerung	
58	Automotive	Basisoperationen	AB-10	SCtrl_Gen_Act	GENERAL_ACT	Allgemeine Aktion einer übergeordneten Steuerung	Action;Number
59	Automotive	Basisoperationen	AB-10.1	Action	ENUM	Definition der durchzuführenden Aktion	Request_UserNum;Send_UserNum; Request_TypNum;Send_TypNum
60	Automotive	Basisoperationen	AB-10.2	Number	INT	Nummer die abgefragt oder versendet werden soll	
61	Automotive	Basisoperationen	AB-11	Type_ID	INT	Fahrzeugtyp	
62	Automotive	Punktschweißen	AP-1	WeldGun	SPOT_GUN	Schweißzangen-Datentyp	AP-1.1…AP-1.10
63	Automotive	Punktschweißen	AP-1.1	Name	STRING	Zangenname	
64	Automotive	Punktschweißen	AP-1.2	ID	INT	ID der Zange	
65	Automotive	Punktschweißen	AP-1.3	Tool_data	TOOL	Werkzeugdaten der Zange	
66	Automotive	Punktschweißen	AP-1.4	Max_Open_Dist	INT	Maximales Zangenöffnungsmaß	
67	Automotive	Punktschweißen	AP-1.5	Position	INT	Absolute Position/Vorhub	
68	Automotive	Punktschweißen	AP-1.6	Min_Force	INT	Minimale Kraft	
69	Automotive	Punktschweißen	AP-1.7	Partless_Force	INT	Kraftwert bei bauteillosem Betrieb	
70	Automotive	Punktschweißen	AP-1.8	Controller_Count	INT	Anzahl Schweißsteuerungen	
71	Automotive	Punktschweißen	AP-1.9	FastClose_Enabled	BOOL	Option für vorzeitiges Schließen der Zange zulassen	
72	Automotive	Punktschweißen	AP-1.10	LeaveCtrt_Enabled	BOOL	Freigabe der Zangenkontrollfunktion	
73	Automotive	Punktschweißen	AP-2	WeldPoint	WELDPOINT	Schweißpunkt-Datentyp	ID;Pose;Type_ID
74	Automotive	Punktschweißen	AP-2.1	ID	INT	Bezeichnung des Schweißpunktes	
75	Automotive	Punktschweißen	AP-2.2	Pose	POSE	Pose des Schweißpunktes	
76	Automotive	Punktschweißen	AP-2.3	Type_ID	INT	Fahrzeugtyp des Schweißpunktes	
77	Automotive	Punktschweißen	AP-3	GunOpen	INT	Zangenöffnungsmaß in mm	
78	Automotive	Punktschweißen	AP-4	PreCloseTime	INT	Vorzeitiges Öffnen der Schweißzange in ms	
79	Automotive	Punktschweißen	AP-5	PartEnd	BOOL	Aktivierung der Bauteilkontrolle am letzten Schweißpunkt	
80	Automotive	Punktschweißen	AP-6	EqualOffset	REAL	Offset für Zangenausgleich in bar	
81	Automotive	Punktschweißen	AP-7	FastClose	INT	Zange vor Erreichen des Schweißpunktes schließen in ms	
82	Automotive	Punktschweißen	AP-8	LeaveCtrl	ENUM	Zangenposition kontrollieren, bevor Bewegung fortgesetzt wird	Ctrl; None
83	Automotive	Punktschweißen	AP-9	Rotation	ENUM	Rotationsrichtung des Kappenwechslers	Left; Right
84	Automotive	Punktschweißen	AP-10	Dresser_State	ENUM	Prüfart des Kappenwechslers	Cap_Exisiting; Cap_Free
85	Automotive	Punktschweißen	AP-11	Force	INT	Anpressdruck für Elektrode in Newton	
86	Automotive	Punktschweißen	AP-12	Water_Switch	ENUM	Schalter für Kühlwasser	Water_ON; Water_OFF
87	Automotive	Punktschweißen	AP-13	DresserNr	INT	Nummer des Kappenfräsers	
88	Automotive	Punktschweißen	AP-14	Dresser_Commanc	ENUM	Anweisung für Kappenfräser	GripOpen; GripClose;BoltOpen; BoltClose
89	Automotive	Punktschweißen	AP-15	FixedTip	ENUM	Stoßrichtung	Up; Down
90	Automotive	Punktschweißen	AP-16	SV_Programm	ENUM	Programmwahl	TipDress; TipChange; EqualCali;GunChange;GunMaintain
91	Automotive	Punktschweißen	AP-17	SV_Programm_Dre	ENUM	Programmwahl für Kappenfräser	UnDockDresser; DockDresser
92	Automotive	Kleben	AK-1	GlueSys	GLUESYS	Klebesystem-Datentyp	AK-1.1…AK-1.15
93	Automotive	Kleben	AK-1.1	Name	STRING	Name des Klebesystems	
94	Automotive	Kleben	AK-1.2	ID	INT	ID des Klebesystems	
95	Automotive	Kleben	AK-1.3	Nozzle_Count	INT	Anzahl an Klebepistolen	
96	Automotive	Kleben	AK-1.4	2K_System_on	BOOL	2K-System in Betrieb	
97	Automotive	Kleben	AK-1.5	Static_Mixer_exst	BOOL	Statischer Mischer angeschlossen	
98	Automotive	Kleben	AK-1.6	OnlineCam_exst	BOOL	Kamera angeschlossen	
99	Automotive	Kleben	AK-1.7	Max_Purge_Time	INT	Maximale Spülzeit in s	
100	Automotive	Kleben	AK-1.8	Max_Fill_Time	INT	Maximale Füllzeit in s	

Tabelle 20.3 - Liste, Deklaration und Aufbau der AIRL-Eingangsparameter und zugehöriger Datentypen

Nr	Kategorie	Gruppe	Identifier	Bezeichnung	Datentyp	Beschreibung	Deklaration/Aufbau
101	Automotive	Kleben	AK-1.9	Delay_On_Time	REAL	Zeitangabe bei Verzögerung für Düse "auf" in ms	
102	Automotive	Kleben	AK-1.10	Delay_Off_Time	REAL	Zeitangabe bei Verzögerung für Düse "zu" in ms	
103	Automotive	Kleben	AK-1.11	Para_Switch_Time	REAL	Zeitangabe für Vorverzögerung bei Parameteranwahl in ms	
104	Automotive	Kleben	AK-1.12	Ref_Speed	INT	Geschwindigkeitsangabe in mm/sec	
105	Automotive	Kleben	AK-1.13	Delay_TCP_Speed	REAL	Zeitangabe für Vorverzögerung TCP-Fluss in s	
106	Automotive	Kleben	AK-1.14	Restart_GlueMode	ENUM	Restartverhalten nach Not-Halt	Dialog;Dry;Wet
107	Automotive	Kleben	AK-1.15	WSN_used	BOOL	Verwendung Breitschlitzdüse	
108	Automotive	Kleben	AK-2	Prog_Num	INT	Programmnummer, die zur Klebesteuerung gesendet wird	
109	Automotive	Kleben	AK-3	Para_Num	INT	Parameternummer, die zur Klebesteuerung gesendet wird	
110	Automotive	Kleben	AK-4	RefSpeed	INT	Referenzgeschwindigkeit zur Berechnung der TCP-Geschwind.	
111	Automotive	Kleben	AK-5	Stepping	BOOL	Stepnahtfunktion aktivieren	
112	Automotive	Kleben	AK-6	BoostLock	BOOL	Härter im 2K-Betrieb sperren	
113	Automotive	Kleben	AK-7	Fill_Condition	ENUM	Optionen zum Füllen des Dosierers	None;Fill;Fill_Conditionally
114	Automotive	Kleben	AK-8	GlueNozzle_Num	INT	Nummer der Klebepistole	
115	Automotive	Kleben	AK-9	Component	ENUM	Auswahl der zu spühlenden Klebekomponente	Mixed;Base;Hardener
116	Automotive	Kleben	AK-10	Purge_Condition	BOOL	Bedingtes Spühlen aktivieren	
117	Automotive	Kleben	AK-11	FillCtrl	ENUM	Warten bis Füllprozess abgeschlossen ist	Ctrl; None
118	Automotive	Kleben	AK-12	Boost_Lock	BOOL	Sperre an- oder ausschalten	
119	Automotive	Kleben	AK-13	Time_Delay	REAL	Zeitschverschiebung zum Schaltpunkt	
120	Automotive	Kleben	AK-14	Offset	REAL	Anheben des Signals für die TCP-Flussgeschwindigkeit	
121	Automotive	Kleben	AK-15	Point_Num	INT	Schweißpunkt-Nummer	
122	Automotive	Kleben	AK-16	Bead_Num	INT	Klebenaht-Nummer	
123	Automotive	Kleben	AK-17	Distance	INT	Anpassung Parameterwechsel in mm	
124	Automotive	Kleben	AK-18	CamRec	BOOL	Aktivierung der Kameraprüfung	
125	Automotive	Kleben	AK-19	CamDistance	INT	Wegangabe in mm für das Ein- oder Ausschalten der Kamera	
126	Automotive	Kleben	AK-20	PrePressure	BOOL	Vordruckregelung an und ausschalten	
127	Automotive	Kleben	AK-21	Time_Open	REAL	Öffnungszeit der Düse in Sekunden	
128	Automotive	Kleben	AK-22	FlowRate	REAL	Geschwindigkeit für die Materialflusssteuerung in %	
129	Automotive	Kleben	AK-23	CamTime	INT	Zeitangabe in ms, für das Ein- oder Ausschalten der Kamera	
130	Automotive	Handling	AH-1	Gripper	GRIP_PER	Greifer-Datentyp	Grp_Name;Grp_ToolData;Grp_Actuator;Grp_PartChk
131	Automotive	Handling	AH-1.1	Grp_Name	STRING	Name des Greifers	
132	Automotive	Handling	AH-1.2	Grp_ToolData	TOOL	Werkzeugdaten des Greifers	
133	Automotive	Handling	AH-1.3	Grp_Actuator	GRIP_ACT	Aktuator-Datentyp	Name;Type;TimeOffs;PLC_Check_On;Ctrl_Valve;Valve;Cylinder
134	Automotive	Handling	AH-1.3.1	Name	STRING	Name des Aktuators	
135	Automotive	Handling	AH-1.3.2	Type	ENUM	Typ des Aktuators	Pulse;Static;Vaccum;Clamp
136	Automotive	Handling	AH-1.3.3	TimeOffs	REAL	Wartezeit bis zu Fehlermeldungen	
137	Automotive	Handling	AH-1.3.4	PLC_Check_On	BOOL	Ein- und Ausschalten der übergeordneten Überwachung	
138	Automotive	Handling	AH-1.3.5	Ctrl_Valve	DI_E	Informationen zum Vorsteuerventil	
139	Automotive	Handling	AH-1.3.6	Valve	GRIP_ACT_VALVE	Ventil-Datentyp	posAdv_Do;posRet_Do;posAdvPulse_Time;posRetPulse_Time
140	Automotive	Handling	AH-1.3.6.1	posAdv_Do	DO	Ausgang für Positionssteuerung auf Vor-Position	
141	Automotive	Handling	AH-1.3.6.2	posRet_Do	DO	Ausgang für Positionssteuerung auf Rück-Position	
142	Automotive	Handling	AH-1.3.6.3	posAdvPulse_Time	REAL	Pulszeiten für Vor-Position	
143	Automotive	Handling	AH-1.3.6.4	posRetPulse_Time	REAL	Pulszeiten für Rück-Position	
144	Automotive	Handling	AH-1.3.7	Cylinder	GRIP_ACT_CYLINDE	Zylinder-Datentyp	posAdv_Di;posRet_Di;AdvChk_Off;AdvChk_Time
145	Automotive	Handling	AH-1.3.7.1	posAdv_Di	DI	Eingang für Positionssteuerung auf Vor-Position	
146	Automotive	Handling	AH-1.3.7.2	posRet_Di	DI	Eingang für Positionssteuerung auf Rück-Position	
147	Automotive	Handling	AH-1.3.7.3	AdvChk_Off	BOOL	Überwachen des Zylinders im bauteillosen Betrieb	
148	Automotive	Handling	AH-1.3.7.4	AdvChk_Time	REAL	Wartezeit bis zur positiven Rückmeldung bei ausgeschalteter Überwachung	
149	Automotive	Handling	AH-1.4	Grp_PartChk	GRIP_PRTCHK	Bauteilkontrolle-Datentyp	Label;Sensor
150	Automotive	Handling	AH-1.4.1	Label	STRING	Name der Bauteilkontrolle	

Tabelle 20.4 - Liste, Deklaration und Aufbau der AIRL-Eingangsparameter und zugehöriger Datentypen

Nr	Kategorie	Gruppe	Identifier	Bezeichnung	Datentyp	Beschreibung	Deklaration/Aufbau
151	Automotive	Handling	AH-1.4.2	Sensor	DI_E	Informationen zum Sensor	
152	Automotive	Handling	AH-2	Load_Variant	ENUM	Aktuelle Lastvariante am Greifer/Sauger	Grp_w_Part;Grp_wo_Part; Skr_w_Part;Skr_wo_Part
153	Automotive	Handling	AH-3	Part_Transfered	BOOL	Transfer des Bauteils in eine neue Station	
154	Automotive	Handling	AH-4	Grp_State	ENUM	Prüfart des Greifers	Grp_Occu;Grp_Free
155	Automotive	Handling	AH-5	SCtrl_Control	ENUM	Kontrolle der überg. Steuerung für Abfrage ausschalten	Control;No_Control
156	Automotive	Werkzeug wechseln	AW-1	ToolChanger_Tool	TC_TOOL	Werkzeughälfte-Datentyp	AW-1.1…AW-1.9
157	Automotive	Werkzeug wechseln	AW-1.1	Name	STRING	Name des Werkzeugs	
158	Automotive	Werkzeug wechseln	AW-1.2	ID	INT	ID des Werkzeugs	
159	Automotive	Werkzeug wechseln	AW-1.3	Tool_data	TOOL	Zugeordnete Werkzeugdaten	
160	Automotive	Werkzeug wechseln	AW-1.4	Nr_Devices	INT	Anzahl konfigurierter Netzwerkgeräte für eine Werkzeugseite	
161	Automotive	Werkzeug wechseln	AW-1.5	LockTimeOff	REAL	Zeit für den Druckaufbau nach dem Andocken	
162	Automotive	Werkzeug wechseln	AW-1.6	DockPosition	ROBTARGET	Position für das Andocken	
163	Automotive	Werkzeug wechseln	AW-1.7	UnDockPosition	ROBTARGET	Position für das Abdocken	
164	Automotive	Werkzeug wechseln	AW-1.8	AllowExternalStatio	BOOL	Externe Ablagen erlauben	
165	Automotive	Werkzeug wechseln	AW-1.9	AllowedStations	ARRAY	Freigegebene Stationen für dieses Werkzeug	
166	Automotive	Werkzeug wechseln	AW-2	ToolChanger_Stati	TC_STATION	Werkzeugbahnhof-Datentyp	AW-2.1…AW-2.9
167	Automotive	Werkzeug wechseln	AW-2.1	Name	STRING	Name des Werkzeugbahnhofs	
168	Automotive	Werkzeug wechseln	AW-2.2	ID	INT	ID des Bahnhofs	
169	Automotive	Werkzeug wechseln	AW-2.3	Base	POSE	Position des Bahnhofs	
170	Automotive	Werkzeug wechseln	AW-2.4	Dist_Dock	REAL	Abstand in Z-Richtung für Andocken	
171	Automotive	Werkzeug wechseln	AW-2.5	Dist_UnDock	REAL	Abstand in Z-Richtung für Abdocken	
172	Automotive	Werkzeug wechseln	AW-2.6	Dist_CheckC	REAL	Abstand in Z-Richtung für Belegungsabfrage	
173	Automotive	Werkzeug wechseln	AW-2.7	Cover_Installed	BOOL	Schwenkbarer Bahnhofdeckel installiert	
174	Automotive	Werkzeug wechseln	AW-2.8	Cover_TimeOut	REAL	Timeout bis zur Fehlermeldung, wenn Deckelposition nicht erreicht wird	
175	Automotive	Werkzeug wechseln	AW-2.9	Second_Sensor	BOOL	Zweiter Sensor für Bahnhofsbelegung installiert	
176	Automotive	Werkzeug wechseln	AW-3	Occupied_Chk	BOOL	Überprüfen, ob die Station belegt ist	
177	Automotive	Werkzeug wechseln	AW-4	Cover_Wait	BOOL	Warten bis Abdeckung offen ist	

9.5. Prototypische Umsetzung und Evaluierung

9.5.1. Probandenstudie

Tabelle 21 – Erfasste Zeiten der Probandenstudie (m:ss) / Fehlversuche mit "-" gekennzeichnet

Versuchsperson		**1**		**2**		**3**		**4**		**5**		**6**		**7**		**8**	
Einstufung		**unerfahr. Nutzer**		**unerfahr. Nutzer**		**unerfahr. Nutzer**		**unerfahr. Nutzer**		**Roboterexp. (Kuka)**		**Roboterexp. (Kuka)**		**Roboterexp. (Kuka)**		**Roboterexp. (Kuka)**	
Altersgruppe in Jahren		**30 bis 40**		**20 bis 30**		**20 bis 30**		**20 bis 30**		**30 bis 40**		**50 bis 60**		**40 bis 50**		**30 bis 40**	
Geschlecht		**m**		**m**		**m**		**m**		**m**		**m**		**m**		**m**	
Aufgabe	**Panel**	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**	**A**	**B**
Punkte teachen	**Comau**	1:27	0:11	1:00	0:24	-	0:37	1:21	0:30	1:04	0:20	0:55	0:13	1:24	0:17	1:10	0:19
	Kuka	1:59	0:12	0:29	0:10	2:59	0:14	0:19	0:06	0:09	0:07	0:12	0:09	0:18	0:12	0:11	0:10
	RCC	0:30	0:05	0:38	0:05	0:37	0:06	0:43	0:06	0:28	0:04	0:25	0:06	0:35	0:06	0:15	0:05
Achsbewegung	**Comau**	2:14	0:26	-	0:45	-	0:25	-	1:40	1:30	0:24	1:19	0:29	1:30	0:55	0:56	0:24
	Kuka	1:29	0:44	-	0:36	-	1:32	-	1:00	0:28	0:29	0:55	0:31	0:44	0:33	0:25	0:25
	RCC	0:34	0:21	0:22	0:19	0:25	0:18	0:30	0:22	0:29	0:20	0:31	0:19	0:28	0:23	0:26	0:15
Überschleif-radius einstellen	**Comau**	-	0:20	-	0:29	-	1:00	-	0:20	1:44	0:19	-	0:25	-	0:35	2:31	0:31
	Kuka	-	0:28	-	0:29	-	0:30	-	0:43	0:30	0:26	0:30	0:20	0:25	0:23	0:29	0:24
	RCC	0:32	0:18	0:46	0:19	1:00	0:25	1:13	0:20	0:40	0:20	0:45	0:21	0:30	0:25	1:10	0:20
Werkzeug-wechsler öffnen/schließen	**Comau**	1:09	0:34	1:46	0:14	-	0:09	-	0:53	1:10	0:14	-	0:21	0:56	0:10	0:50	0:08
	Kuka	0:19	0:04	1:29	0:08	-	0:22	-	0:10	0:05	0:04	0:06	0:05	0:25	0:05	0:06	0:06
	RCC	0:22	0:04	0:12	0:07	0:09	0:06	1:30	0:05	0:20	0:06	0:21	0:05	0:11	0:07	0:12	0:04

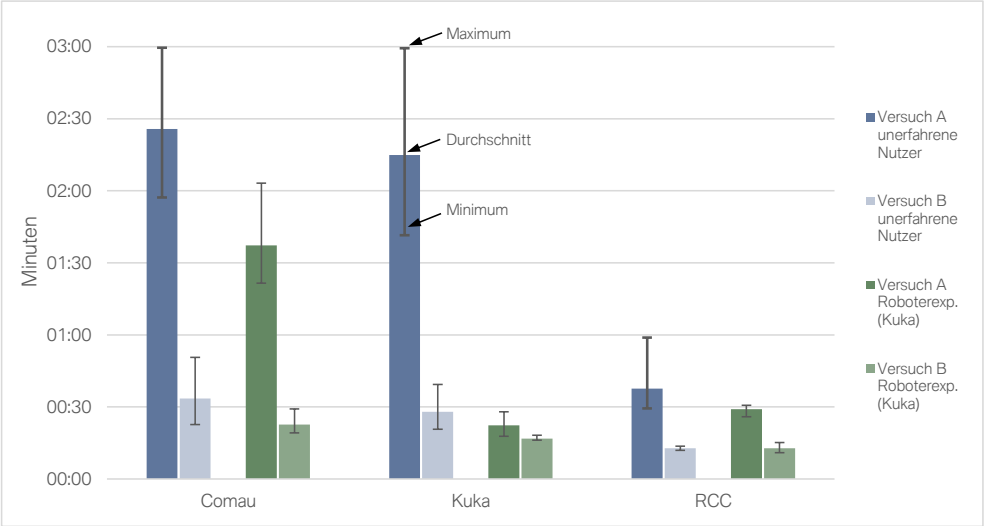


Abbildung 113 – Durchschnittliche Dauer der erfassten Bedienvorgänge

9.6. Präzisierung Eigenanteil und Zuarbeiten

Die initiale Idee und zentrale Grundlagen vorliegender Dissertation sind während meiner Arbeit in der BMW-Roboterstandardisierung und der Projektleitung des Vorentwicklungsprojektes SOAA („Serviceorientierte Automatisierungsarchitektur“) entstanden. Neben dem direkten Zugang zu Prozess- und Expertenwissen bot mir dies die Möglichkeit, Fragestellung der Arbeit umfassend und ganzheitlich zu bearbeiten und - insbesondere die aufwändige Umsetzung der Lösungsansätze – nicht allein, sondern in Zusammenarbeiten mit studentischen Mitarbeiten und Projektpartnern zu realisieren.

Im folgenden Abschnitt soll daher dargestellt werden, welche Umfänge dieser Dissertation in vollständiger Eigenleistung erfolgt sind und welche Teile mit Unterstützung von mir betreuten Studenten oder in Kooperation mit Projektpartner des Vorentwicklungsprojektes entstanden sind. Die Anfertigung vorliegenden Schriftstücks ist unabhängig von der operativen Unterstützung durchwegs vollständige Eigenleistung.

Tabelle 22 – Übergreifende Darstellung der Eigenanteile und Unterstützungsumfänge

Teil der Arbeit	**Kapitel**	**Vollständige Eigenleistung**	**Unterstützung in der operativen Umsetzung durch**	
			Studentische Arbeitkräfte / Abschluss-arbeiten	Projektpartner im Rahmen des VE-Projekts
Einleitung	1.	Ja	Nein	Nein
Stand von Wissenschaft und Technik	2.	Ja	Nein	Nein
Abgrenzung, zentrale Forschungsfragen und Vorgehen	3.	Ja	Nein	Nein
Roboterintegration am Beispiel der BMW AG	4.	Ja	Nein	Nein
Herstellerunabhängige Steuerung von Robotersystemen	5.			
Methoden der externen Steuerung von Robotern	5.1.	Ja	Nein	Nein
Leistungsmessungen	5.2.			
Konzept, Messdesign und Zielstellung	5.2.	Ja	Nein	Nein
Versuchsaufbau	5.2.	Nein	Nein	Ja
Durchführung	5.2.	Ja	Nein	Nein
Auswertung und Interpretation	5.2.	Ja	Nein	Nein
Integrationsprozess von externen Steuerungssystemen	5.3.	Ja	Nein	Nein
Zusammenfassung	5.4.	Ja	Nein	Nein
Herstellerunabhängige Programmierung von Robotersystemen	6.			
Methoden und Sprachen der Roboterprogrammierung	6.1.	Ja	Nein	Nein
Fragestellung und Vorgehen	6.2.	Ja	Nein	Nein
Analyse der textuellen Roboterprogramm. im Karosseriebau	6.3.			
Programmiersprachen für Industrieroboter	6.3.1.	Nein	Ja	Nein
Programmierung von Automobilapplikationen	6.3.2.	Nein	Ja	Nein
Schlussfolgerung und Fazit	6.3.3.	Ja	Nein	Nein
Alternative herstellerun. Programmiersp. für Robotersys.	6.4.	Ja	Nein	Nein
Anforderungen an die Programmierung aus Anwendersicht	6.5.	Ja	Nein	Nein
Vorschlag eines hybriden Programmiersprachenkonzeptes	6.6.	Ja	Nein	Nein
AIRL – Automotive Industrial Robot Language	6.7.	Ja	Nein	Nein
Zusammenfassung	6.8.	Ja	Nein	Nein
Protypische Umsetzung und Evaluierung	7.			
Herstellerun. Programmierung auf Instandhaltungsebene	7.1.	Nein	Ja	Nein
Herstellerun. Programmierung auf Expertenebene	7.2.	Nein	Ja	Nein
Herstellerunabhängige Karosseriebauroboterzelle	7.3.	Nein	Nein	Ja
Bewertung aus Gesamtunternehmenssicht	7.4.	Ja	Nein	Nein
Zusammenfassung und Ausblick	8.	Ja	Nein	Nein

Kommentierung des Eigenanteils zu den Kapiteln:

5.2. Leistungsmessungen

Der Eigenanteil im Rahmen des Versuchsaufbau für die Leistungsmessungen lag insbesondere in der Erstellung des Versuchskonzepts und des Messdesigns sowie der daraus resultierenden Definition von Anforderungen an Hardware, Software und Messtechnik. Der Aufbau der Robotertestzelle und die Inbetriebnahme der Versuchsprogramme ist in Kooperation mit Projektpartnern im Rahmen des Vorentwicklungsprojektes erfolgt. Softwarebasis für die externe Ansteuerung der Robotersysteme stellte die Robotics Library dar (S. 22) [53].

Der Aufbau des Messequipments in der Testzelle und die Überprüfung der Messqualität wurde mit Unterstützung des Lasertrackerherstellers durchgeführt. Die Durchführung der Versuchsreihen, die Optimierung und Parametrierung der Versuchsprogramme, die Messungen der Testbahnen, die Erstellung der Auswertungswerkzeugkette sowie die Auswertung und Interpretation der Messdaten ist in vollständiger Eigenleistung erfolgt.

6.3.1 und 6.3.2 – Programmiersprachen für Industrieroboter/Automobilapplikationen

In der Bearbeitung des Arbeitspakets der Industrieroboterprogrammiersprachen und Applikationen bestand der Eigenteil insbesondere in der Gesamtidee und Festlegung der Vorgehensweise. Ebenso war die Auswertung der Befehlsdatenbank, die finale Analyse der Roboterlinie und die Interpretation der Ergebnisse eigenständige akademische Arbeit. Die initiale Erstellung der Befehlsdatenbasis anhand der Roboterhersteller- und Standarddokumentation ist im Rahmen einer betreuten studentischen Abschlussarbeit entwickelt worden [166].

7.1 - Herstellerunabhängige Programmierung auf Instandhaltungsebene

Zentrale Eigenleistung im Kontext der herstellerunabhängigen Programmierung auf Instandhaltungsebene bestand in der Festlegung der Anforderungen an Bedienung und Sprachkonzept an den RobotCell Commander. Die erste Umsetzung des RobotCell Commanders und Benutzertests sind im Rahmen einer betreuten, studentischen Abschlussarbeit [162] und anschließenden studentischen Praktika erfolgt. Die übergreifende Auswertung und Interpretation der Ergebnisse ist vollständige Eigenleistung.

7.2. - Herstellerunabhängige Programmierung auf Expertenebene

Zentrale Eigenleistung im Kontext der herstellerunabhängigen Programmierung auf Expertenebene bestand in der Grundidee und der Anforderungsdefinition an das Programmiersystem sowie an die semantische und syntaktische Umsetzung auf Basis der AIRL-Spezifikation. Die technische Basis der Programmierumgebung wurde im Rahmen einer betreuten studentischen Abschlussarbeit entwickelt [163].

7.3. – Herstellerunabhängige Karosseriebauroboterzelle

Die Umsetzung einer Roboterzelle auf Basis von Open-Source-Technologie ist im Zuge des Vorentwicklungsprojektes der BMW AG entstanden. Gesamtidee, Steuerungs- und Sicherheitskonzept, Anforderungen an das Gesamtsystem und die Applikationszusammenstellung der Robotertestzelle sind erarbeitete akademische Eigenleistungen im Rahmen der Dissertation. Die Umsetzung in Form von Aufbau und Inbetriebnahme der Robotertestzelle ist in Kooperation mit den beteiligten Projektpartnern erfolgt.

9.7. Komprimierte Darstellung der Forschungsschwerpunkte und -ergebnisse

Ausgangssituation und Motivation

- Proprietäre Robotersysteme mit heterogenen Softwarearchitekturen, Bedienkonzepten, Programmiersprachen und Schnittstellen erschweren die Integration, Innovation und Flexibilität im Produktionsumfeld sowie die herstellerübergreifende Wiederverwendung von Software.
- Flexibilität in der globalen Bezugsstrategie und Vermeidung von Single-Sourcing ist essentiell um die Innovationsfähigkeit, Liefersicherheit und Wettbewerbsfähigkeit eines Automobilunternehmens sicherzustellen.
- Mangelnde Herstellerneutralität stellt sowohl historisch als auch gegenwärtig ein bedeutendes Defizit in der Robotik dar.
- Die interoperable, herstellerneutrale Verwendung von Robotersystemen ist seit den 1990er Jahren Forschungsgegenstand und es existiert eine Vielzahl von Ansätzen und Lösungsvorschlägen aus Forschungsarbeiten, Normierungsinitiativen und Industrielösungen.
- Der Hauptanwender von Industrierobotern, der „automobile Karosseriebau“, mit seinem technologischen und organisatorischen Anforderungsspektrum wird allerdings meist außer Acht gelassen. Die Integration von Industrierobotersystemen im Karosseriebau gestaltet sich daher weiterhin als sehr herstellerspezifisch, langwierig und ressourcenintensiv.
- => Forschungsbedarf in Bezug auf die kombinierte Schwerpunkte „Roboterintegration“, „Herstellerunabhängigkeit“ und „“Karosseriebau“.

- Bestehende Defizite	? Forschungsfrage	>> Vorgehen	+ Ergebnisse

Integrationsprozess von Robotersystemen

-	▪ Bestehende Arbeiten fokussieren sich häufig auf Servicerobotik oder die grundsätzliche Automatisierung von Prozessen mit kleinen Losgrößen und betrachten selten den Anwendungsfall einer bereits hochautomatisierten und standardisierten Großserienfertigung. ▪ Es fehlt eine Anwendungsfallanalyse und Anforderungsableitung für den Integrationsprozess von Robotern in den Karosseriebau.
?	▪ Welche Anforderungen bestehen an die herstellerunabhängige Integration von Robotersystemen im automobilen Karosseriebau?
>>	▪ Analyse, Beschreibung und Anforderungsableitung für den Integrationsprozess von Industrierobotersystemen an einem Fallbeispiel.
+	▪ Integrationsprozess und Einschränkungen durch herstellspezifi. Robotersysteme am Fallbeispiel eines Automobil-OEM objektiviert. ▪ Zweigeteilter Integrationsprozess mit mehreren Anwendern und unterschiedlichen Anforderungen identifiziert und beschrieben. ▪ Zentrale Bedarfe an steuerungs- und programmiertechnischen Anforderungen sowie an eine prototypische Umsetzung ermittelt.

Herstellerunabhängige Steuerung von Robotersystemen

-	▪ Es existiert eine Vielzahl an Möglichkeiten der steuerungstechnischen Umsetzung von Herstellerunabhängigkeit durch Externalisierung. ▪ Es fehlt eine übergreifende Betrachtung der Ansätze und der Eignung für spezifische Anwendungsfälle sowie eine Aussage bzgl. der Leistungsfähigkeit von extern gesteuerten Robotersystemen – insbesondere für den Karosseriebau.
?	▪ Welche Methode der externen Ansteuerung von Robotersystemen ist für den automobilen Karosseriebau am besten geeignet?
>>	▪ Analyse und Bewertung von existierenden Methoden der externen Ansteuerung. ▪ Entwicklung eines Bewertungsmodells für den allgemeine Integrationsprozess und Bewertung für das Fallbeispiel. ▪ Untersuchen der realen Leistungsfähigkeit und Eignung von extern gesteuerten Robotersystemen. ▪ Identifikation der steuerungstechnischen Defizite in Inbetriebnahme und Betrieb.
+	▪ Generische Kategorisierung und Charakterisierung von Methoden der externen Ansteuerung anhand der Integrationstiefe. ▪ Bewertungsmodell für die Integration von Robotersystemen mit Valutierung für den Anwendungsfall des Karosseriebaus. ▪ Definition von (für den Karosseriebau) angepassten und (um die Schaltgenauigkeit) erweiterten Leistungstests auf Basis der ISO 9283. ▪ Automatisierte Auswertungswerkzeugkette für Messdaten entwickelt und Leistungsmessungen durchgeführt. ▪ Bestätigte Leistungsfähigkeit und Eignung von extern gesteuerten Robotersystemen auf Interpolationsebene. ▪ Plug & Produce-Methode für die verbesserte Integration von extern gesteuerten Robotersystemen (inkl. stattgegebener Patente).

Herstellerunabhängige Programmierung von Robotersystemen

-	▪ Vielzahl von Lösungen, die Herstellerabhängigkeit in der Programmierung reduzieren können, jedoch ohne Karosseriebaufokus. ▪ Keine Betrachtung des Anwendungsfalls Karosseriebaus und dessen Anforderungen an die herstellerneutrale Programmierung. ▪ Keine Auslegung der Programmiersprachen und -methoden an die Anwendergruppen des Karosseriebaus.
?	▪ Welche Roboterprogrammiermethode und -sprache deckt die notwendigen Funktionalitäten und Anforderungen im Planungs-, Entwicklungs- und Produktionsprozess der Automobilproduktion im Karosseriebau am besten ab?
>>	▪ Analyse von Industrieroboterprogrammiermethoden und -sprachen sowie der Anwendungsprogrammierung im Karosseriebau. ▪ Ableiten der Anforderungen des Karosseriebaus an Programmiersprache und Roboterbefehle. ▪ Analyse von zur Verfügung stehenden Alternativen und Abgleich mit den Anforderungen der Anwendergruppen. ▪ Vorschlag einer geeigneten Programmiersprache für zentrale Anwendergruppen des Karosseriebaus.
+	▪ Vergleichende Befehlsdatenbank für unterschiedliche Roboterprogrammiersprachen als Analysebasis für reale Roboterprogramme. ▪ Funktionaler Anforderungskatalog auf Basis von analysierten Roboterlinien und Applikationsstandards. ▪ Anwendergerechte hybride und herstellerunabhängige Programmiersprachenspezifikation für den Karosseriebau (AIRL).

Prototypische Umsetzung und Evaluierung

-	▪ Keine existierende Umsetzung, um herstellerunabhängige Programmierung und steuerungstechnischen Gesamtverbund zu evaluieren. ▪ Keine Anforderungsdefinition an Programmierung und steuerungstechnische Umsetzung. ▪ Keine Bewertung der herstellerunabhängige Integration von Industrierobotersystemen aus Gesamtunternehmenssicht.
?	▪ Wie und mit welchem Ergebnis ist die herstellerunabhängige Integration von Robotersystemen an einem Prototyp für den automobilen Karosseriebau zu evaluieren und welche Vor- und Nachteile ergeben sich aus Unternehmenssicht?
>>	▪ Ableitung der Anforderungen aus den vorangegangenen Arbeitspaketen und Karosseriebauspezifikationen. ▪ Einteilung der Umsetzung in die Programmierung auf Instandhaltungs- und Expertenebene sowie der steuerungstechnischen Gesamtumsetzung anhand einer herstellerunabhängigen Karosseriebauroboterzelle mit geeigneten Applikationen. ▪ Evaluierung der Prototypen und des übergreifenden Integrationskonzeptes.
+	▪ Anforderungsdefinition für die prototypische Umsetzung von Programmierung, Steuerungstechnik und Sicherheitskonzept. ▪ Prototyp für herstellerunabhängige Programmierung auf Instandhaltungs- und Expertenebene auf Basis der AIRL. ▪ Prototyp einer herstellerunabhängigen Karosseriebauroboterzelle auf Basis von Open-Source-Technologie. ▪ Evaluierung und Bestätigung der Leistungsfähigkeit der prototypischen Umsetzungen. ▪ Bewertung der herstellerunabhängigen Integration von Industrierobotersystemen aus Gesamtunternehmenssicht.

Abbildung 114 - Komprimierte Darstellung der Forschungsschwerpunkte und -ergebnisse

Abbildungsverzeichnis

Abbildung 1 – Visionäres Zielbild am Fallbeispiel der BMW AG 4
Abbildung 2 – Abnehmer von Industrierobotern nach Branchen für die Jahre 2016 bis 2019 ... 6
Abbildung 3 – Komponenten eines Industrieroboters 6
Abbildung 4 – Aufbau einer Industrierobotersteuerung in Anlehnung an [9] 7
Abbildung 5 – Produktverbesserungen Kuka-Industrieroboter in Anlehnung an [15] 10
Abbildung 6 – Beiträge zur herstellerunabhängigen Robotik im Zeitverlauf 15
Abbildung 7 – OSACA-Steuerungsplattform und Anwendungssoftware in Anlehnung an [18] 17
Abbildung 8 – ReApp Messestand mit Anwendungsfall der BMW AG 24
Abbildung 9 – Vorgehen innerhalb der Arbeit 33
Abbildung 10 – Produktionsschritte in der Automobilfertigung in Anlehnung an [103] 36
Abbildung 11 – Karosseriebaulinie 37
Abbildung 12 – Automatisierungspyramide am Beispiel des automobilen Karosseriebaus in Anlehnung an [104] und [9] 38
Abbildung 13 – Standardisierte Roboterapplikationen im Karosseriebau der BMW AG 41
Abbildung 14 – Integrationsprozess von Roboterapplikationen im Karosseriebau 42
Abbildung 15 - Implikationen durch Herstellerabhängigkeit in der Roboterintegration und sich daraus ableitende Anforderungen an eine herstellerneutrale Lösung 46
Abbildung 16 – Implikationen durch Herstellerabhängigkeit in der Roboterintegration und sich daraus ableitende Anforderungen an eine herstellerneutrale Lösung 47
Abbildung 17 – Vorgehen zur Ermittlung einer geeigneten Methode zur herstellerunabhängigen Steuerung von Robotersystem im automobilen Karosseriebau 49
Abbildung 18 – Integrationstiefe von Roboterschnittstellen und daraus abzuleitende Eigenschaften 51
Abbildung 19 – Existierende Umsetzungen zur externen Roboteransteuerung und der verwendeten Schnittstellen 56
Abbildung 20 – Bewertungskategorien und -elemente 59
Abbildung 21 – Kategorisierte Anforderungen an Industrierobotersysteme im Karosseriebau bewertet nach Anforderungstyp (Fertigungs- oder Unternehmensprozess) und Beeinflussbarkeit durch externe Steuerung 63
Abbildung 22 – Skizze Versuchsaufbau 66
Abbildung 23 – Testwerkzeug mit entkoppeltem Gewicht und Aufnahme für den Retroreflektor des Lasertrackers 67
Abbildung 24 – Übertragungsfunktionen für Rast-in-Rast und Rast-in-Geschwindigkeit-in-Rast-Bewegungen mit Polynomen 71
Abbildung 25 – ISO-Cube mit Zielpunkten für Pose-Tests 73
Abbildung 26 – Stabilisierungsprozess des Kuka-Roboters für eine ISO-Cube-Pose-Fahrt 76
Abbildung 27 – Stabilisierungsprozess des Comau-Roboters für eine ISO-Cube-Pose-Fahrt 77
Abbildung 28 – Stabilisierungsprozess des Comau-Roboters für eine ISO-Cube-Pose-Fahrt mit 10 s Haltezeit 77
Abbildung 29 – TCP-Geschwindigkeit im Streckenverlauf für das Kuka-System 81

Abbildung 30 – Vergleich der horizontalen (Y) und vertikalen Abweichung (Z) des TCP bei Kuka-Linearfahrten 82
Abbildung 31 – TCP-Schaltgenauigkeit mit Robotersystem und Zykluszeit (ext. Schnittstelle, ProfiNet) 84
Abbildung 32 – Aufgezeichnete ISO-Cube-Roboterbahnen für Überschleifbewegungen 88
Abbildung 33 – Geschwindigkeitsverlauf beim Anfahren eins Überschleifpunktes 89
Abbildung 34 – Überschleifbahn (Draufsicht) 90
Abbildung 35 – Vergrößerte Darstellung eines Eckpunktes 90
Abbildung 36 – Zentrale Inhalte des Multi-Tier-Robot-Integration-Models 96
Abbildung 37 – Übersicht der Programmierverfahren für Industrieroboter 100
Abbildung 38 – Steigender Abstraktionsgrad mit Evolution der Programmiersprachen in Anlehnung an [129] 102
Abbildung 39 – Vorgehen im Kapitel herstellerunabhängige Programmierung von Robotersystemen 106
Abbildung 40 – Vorgehen bei der Erstellung der Befehlsdatenbank 108
Abbildung 41 – Erfasste Programmierinstruktionen pro Hersteller 110
Abbildung 42 – Gewählte Kategorisierung der Roboterbefehle mit Gruppierungsbeispielen 110
Abbildung 43 – Anzahl erfasster Programmierbefehle 112
Abbildung 44 – Anzahl der als wichtig markierten Roboterbefehle 113
Abbildung 45 – Vergleich der Anzahl an als wichtig markierter Roboterbefehle der Hersteller und der IRL 114
Abbildung 46 – Standardisierte Roboterapplikationen im Karosseriebau der BMW AG 115
Abbildung 47 – Generische Bestandteile einer Roboterapplikation 116
Abbildung 48 – Programmstruktur einer Roboteranwendung 117
Abbildung 49 – Vorgehen Robotercodeanalyse 118
Abbildung 50 – Anzahl an Befehlen pro Roboter 120
Abbildung 51 – Anzahl an Befehlstypen 121
Abbildung 52 – Verteilung der verwendeten Instruktionen anhand der Befehlskategorie und Hersteller 124
Abbildung 53 – Verteilung der verwendeten Befehle anhand Kategorie, Hersteller (BMW-Werk) und Domäne 125
Abbildung 54 – Phasen und beteiligte Anwender an der Roboterprogrammentwicklung im Karosseriebau 129
Abbildung 55 – Programmiercharakteristik und abgeleitete Anforderungen für zentrale Anwender im Karosseriebau 131
Abbildung 56 – Bewertung der Eignung unterschiedlicher Programmiersprachenarten 135
Abbildung 57 – Zusammensetzung der AIRL 136
Abbildung 58 – AIRL-Spezifikation und zugehörige AIRL-DZL und C++/AIRL-API 137
Abbildung 59 – Teach Pendants der Roboterhersteller Kuka, ABB, Fanuc, Comau 140
Abbildung 60 – RobotCell Commander Programmübersicht 142
Abbildung 61 – Parametrierung einer Bewegungsoperation 143
Abbildung 62 – Ansicht für Applikationseinstellungen am Beispiel Schweißen und Werkzeugwechseln 143
Abbildung 63 – Einstellung der I/O Parameter für eine Punktschweißoperation 144

Abbildung 64 – Eclipse IDE mit AIRL-Programm 148
Abbildung 65 – Syntaxfehler Highlighting von AIRL-Syntax Fehlern 149
Abbildung 66 – Code Completion für AIRL-Variable 149
Abbildung 67 – Quick Fix Optionen für AIRL-Syntax Fehler 150
Abbildung 68 – Im Prototyp umzusetzende Karosseriebauapplikationen (rot markiert) 152
Abbildung 69 – 3D-Darstellung der Testzelle 153
Abbildung 70 – Eingesetzte Robotersysteme und Schnittstellen 154
Abbildung 71 – Robotertestzelle in der Inbetriebnahmephase 156
Abbildung 72 – Steuerungstechnik der Testzelle mit Comau- (u. l.), ABB- (o. r.) und Kuka-Steuerung (u. r.) 156
Abbildung 73 – Steuerungsarchitektur Robotertestzelle 157
Abbildung 74 – Bedienpanel der SPS-Zellensteuerung: Programm- und Auftragsauswahl für Roboter 158
Abbildung 75 – Bedienpanel der SPS-Zellensteuerung: Steuerung der Roboterkonfiguration 158
Abbildung 76 – Roboter in synchroner Bewegung bei kooperativer externer Ansteuerung 161
Abbildung 77 – Roboter im Testbetrieb einer Punktschweißapplikation mit Werkzeugwechsler 163
Abbildung 78 – Bahnverlauf Pose-Genauigkeitstest für einen nativen Kuka- und Comau-Roboter 172
Abbildung 79 – Erweiterung von Abb. 78 um extern gesteuerte Roboterbahnen 173
Abbildung 80 – Vergrößerte Darstellung der Kuka-Roboterbahnen (nativ mit und ohne Absolutgenauigkeit, extern) 174
Abbildung 81 – Vergrößerte Darstellung der Comau-Roboterbahnen (nativ mit und ohne Absolutgenauigkeit, extern) 175
Abbildung 82 – KukaR-Roboterbahnen mit unterschiedlichen Bewegungsprofilen 177
Abbildung 83 – ComauR-Roboterbahnen mit unterschiedlichen Bewegungsprofilen 178
Abbildung 84 – Stabilisierungsprozess des nativen Comau-Roboters für eine ISO-Cube-Pose-Fahrt 179
Abbildung 85 – Stabilisierungsprozess des Comau-Roboters im externen Betrieb mit P616-Bewegungsprofil für eine ISO-Cube-Pose-Fahrt 180
Abbildung 86 – Stabilisierungsprozess des Comau-Roboters im externen Betrieb mit P3-Bewegungsprofil für eine ISO-Cube-Pose-Fahrt 180
Abbildung 87 – TCP-Geschwindigkeit des nativ und extern gesteuerten Kuka-Roboters 182
Abbildung 88 – TCP-Beschleunigung des nativ und extern gesteuerten Kuka-Roboters 182
Abbildung 89 – TCP-Geschwindigkeit des nativ und extern gesteuerten Comau-Roboters 182
Abbildung 90 – TCP-Beschleunigung des nativ und extern gesteuerten Comau-Roboters 182
Abbildung 91 – TCP-Geschwindigkeit, im Vergleich bei unterschiedlicher Zykluszeit der externen Ansteuerung des Kuka-Roboters 184
Abbildung 92 – TCP-Beschleunigung, im Vergleich bei unterschiedlicher Zykluszeit der externen Ansteuerung des Kuka-Roboters 184
Abbildung 93 – TCP-Abstand zum Zielpunkt, im Vergleich bei unterschiedlicher Zykluszeit der externen Ansteuerung des Kuka-Roboters 184

Abbildung 94 – TCP-Geschwindigkeit, im Vergleich bei unterschiedlicher Zykluszeit der externen Ansteuerung des Comau-Roboters 185
Abbildung 95 – TCP-Beschleunigung, im Vergleich bei unterschiedlicher Zykluszeit der externen Ansteuerung des Comau-Roboters 185
Abbildung 96 – TCP-Abstand zum Zielpunkt, im Vergleich bei unterschiedlicher Zykluszeit der externen Ansteuerung des Comau-Roboters 185
Abbildung 97 – Bahnverlauf der Bahn-Genauigkeitstests für einen nativen Kuka- und Comau-Roboter 190
Abbildung 98 – Bahnverlauf der Bahn-Genauigkeitstests für extern gesteuerte Kuka- und Comau-Roboter 191
Abbildung 99 – Detailbetrachtung Bahncharakteristik 192
Abbildung 100 – TCP-Geschwindigkeitsverlauf von Linearbahnen (Kuka) 194
Abbildung 101 – TCP-Geschwindigkeitsverlauf von Linearbahnen (Comau) 195
Abbildung 102 – Schwerpunktgeschwindigkeitsdifferenzen für Linearbahntestfahrten bei 50 cm/s 196
Abbildung 103 – Vergleich der horizontalen Abweichung (Y) des TCP bei einer Comau- und ComauR-Linearfahrt 197
Abbildung 104 – Vergleich der vertikalen Abweichung (Z) des TCP bei einer Comau- und ComauR-Linearfahrt 197
Abbildung 105 – Kreisbahnen mit nativ und extern gesteuertem Kuka-Roboter-System 201
Abbildung 106 – Ermittelte Verteilung der BMW-Applikationssoftware auf Basis der durchgeführten Code-Analyse für die Werke Regensburg und Dingolfing 202
Abbildung 107 – Kuka KRL-Befehle für eine Punkt-zu-Punkt-Bewegung (o.), eine Punkt-zu-Punkt-Bewegung mit Überschleifen (m.) und eine Linearbewegung (u.) 203
Abbildung 108 – ABB RAPID-Befehle für eine Punkt-zu-Punkt-Bewegung (o.), eine Punkt-zu-Punkt-Bewegung mit Überschleifen (m.) und eine Linearbewegung (u.) 203
Abbildung 109 – BMW-spezifischer KRL-Befehl für eine Punkt-zu-Punkt-Bewegung mit Punktschweißoperation 203
Abbildung 110 – AIRL-Befehle für eine Punkt-zu-Punkt-Bewegung (o.), eine Punkt-zu-Punkt-Bewegung mit Überschleifen (m.) und eine Linearbewegung (u.) 204
Abbildung 111 – C++/AIRL-Befehle für eine Punkt-zu-Punkt-Bewegung (o.), eine Punkt-zu-Punkt-Bewegung mit Überschleifen (m.) und eine Linearbewegung (u.) 204
Abbildung 112 – AIRL-Befehl (o.) und C++/AIRL (u.) für eine Punkt-zu-Punkt-Bewegung mit Punktschweißoperation 204
Abbildung 113 – Durchschnittliche Dauer der erfassten Bedienvorgänge 216
Abbildung 114 - Komprimierte Darstellung der Forschungsschwerpunkte und -ergebnisse. 219

Tabellenverzeichnis

Tabelle 1 – Industrieroboterapplikationen nach der IFR-Klassifikation [6] und deren Anwendung in der Automobilproduktion 40
Tabelle 2 – Externalisierungsebenen und deren Charakteristika 57
Tabelle 3 – Bewertung der Abstraktionsverfahren aus Sicht des Karosseriebaus 61
Tabelle 4 – Versuchsübersicht 68
Tabelle 5 – Kennwerte Positionsstabilisierung Kuka-Robotersystem 78
Tabelle 6 – Ergebnisse und Anforderungsabgleich der Leistungsmessungen 92
Tabelle 7 – Defizite und Verbesserungsvorschläge bei der Integration von Robotersystemen mittels Steuerungskonzepten auf Interpolationsebene 94
Tabelle 8 – Übersicht Industrieroboterhersteller, Programmiersprache und OEM-Einsatz im Karosseriebau 107
Tabelle 9 – Übersicht Robotercodeanalyse 119
Tabelle 10 – Meist verwendete Befehle auf Anwendungs- und Applikationsprogrammebene 122
Tabelle 11 – Fehlversuche und Bedienzeiten für Programmoperationen im Vergleich 145
Tabelle 12 – Anforderungen an eine herstellerunabhängige Karosseriebauroboterzelle 151
Tabelle 13 – Anforderungen und deren Realisierung im Prototyp 163
Tabelle 14 – Räumliche Abweichungen der Haltepunkte der Pose-Testbahn in mm 176
Tabelle 15 – Übersicht Messergebnisse Pose-Messungen Kuka-Roboter 188
Tabelle 16 – Übersicht Messergebnisse Pose-Messungen Comau-Roboter 189
Tabelle 17 – Messergebnisse für lineare Testbahnen 199
Tabelle 18 – Messergebnisse für Kreisbahnen 200
Tabelle 19.1 – Befehls- und Datentypenübersicht der AIRL inkl. der adäquaten Instruktionen in KRL, RAPID und IRL 206
Tabelle 20.1 – Liste, Deklaration und Aufbau der AIRL-Eingangsparameter und zugehöriger Datentypen 212
Tabelle 21 – Erfasste Zeiten der Probandenstudie (m:ss) / Fehlversuche mit "-" gekennzeichnet 216
Tabelle 22 – Übergreifende Darstellung der Eigenanteile und Unterstützungsumfänge 217

Verzeichnis Abkürzungen und Akronyme

ABB	Asea Brown Boveri
AIDA	Automatisierungsinitiative Deutscher Automobilhersteller
AIRL	Automotive Industrial Robot Language
API	Application programming interface
AWL	Anweisungsliste
BMW	Bayerische Motoren Werke
CAD	Computer-aided Design
DIN	Deutsches Institut für Normung
DSL	Domain-specific language
EGM	Externally Guided Motion
GPL	General-purpose language
GUI	Graphical user interface
HTTP	Hypertext Transfer Protocol
HW	Hardware
I/O	Input/Output
IDE	Integrated development environment
IEEE	Institute of Electrical and Electronics Engineers
IFR	International Federation of Robotics
IP	Internet Protocol
IRL	Industrial Robot Language
ISO	International Organization for Standardization
IT	Informationstechnik
KMU	Kleine und mittlere Unternehmen
KRL	KUKA Robot Language
LCI	Life-Cycle-Impulse
NC	Numerical Control
OEM	Original Equipment Manufacturer
P3	Polynom 3. Ordnung
P414	Polynome 4. Ordnung mit konstanter Geschwindigkeit
P5	Polynom 5. Ordnung
P616	Polynome 6. Ordnung mit konstanter Geschwindigkeit
PC	Personal Computer
PHG	Programmierhandgerät
POSIX	Portable Operating System Interface

PTP	Point-to-Point
RCC	RobotCell Commander
RL	Robotics Library
ROS	Robot Operating System
RP	Pose repeatability
RSI	Robot Sensor Interface
RT	Path repeatability
SOAP	Simple Object Access Protocol
SPS	Speicherprogrammierbare Steuerung
ST	Strukturierter Text
SW	Software
TCP	Tool Center Point
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VDI	Verein Deutscher Ingenieure
VDMA	Verband Deutscher Maschinen- und Anlagenbau
XML	Extensible Markup Language

Literaturverzeichnis

[1] Čapek, K., R.U.R. - Rossumovi Univerzální Roboti (Rossum's Universal Robots). Prag: Vydalo Aventinum, 1920.

[2] Brockhaus | NE GmbH, Roboter - Enzyklopädie - Brockhaus.de, [Website] https://brockhaus.de/ecs/enzy/article/roboter (Zugriff am: 19. August 2019).

[3] VDI 2860:1990-05, Montage und Handhabungstechnik Handhabungsfunktionen, Handhabungseinrichtungen; Begriffe, Definitionen, Symbole.

[4] DIN EN ISO 8373:2010-11, Roboter und Robotikgeräte - Wörterbuch (ISO/DIS 8373:2010).

[5] International Federation of Robotics, „World Robotics - Industrial Robots 2020“ Frankfurt, 2020.

[6] International Federation of Robotics, „World Robotics - Industrial Robots 2019“ Frankfurt, 2019.

[7] KUKA Roboter GmbH, „KUKA System Software 8.2: Bedien- und Programmieranleitung für Systemintegratoren“ [Handbuch], Augsburg, 2012.

[8] Kreuzer, E., Lugtenburg, J., Meißner, H. und Truckenbrodt, A., Industrieroboter: Technik, Berechnung und anwendungsorientierte Auslegung 1. Aufl. Berlin: Springer, 1994.

[9] Krug, S., Automatische Konfiguration von Robotersystemen (Plug&Produce) [Dissertation], München: Herbert Utz Verlag, 2012.

[10] Devol, G. C., “Programmed Article Transfer,” US 2,988,237 A, United States Patent Office, Jun 13, 1961.

[11] Unger, A., Die Geschichte der Robotertechnik begann 1954 - ke-next.de, 2017 Mai, [Website] https://www.ke-next.de/robotik/die-geschichte-der-robotertechnik-begann-1954-108.html (Zugriff am: 23. Mai 2020).

[12] Zhang, J., The Industrial Robot Market - 2019 - Interact Analysis, [Website] https://www.interactanalysis.com/the-industrial-robot-market-2019-infographic/ (Zugriff am: 7. April 2020).

[13] Cheng, H. H., Proctor, F., Michaloski, a. J. L. und Shackleford, W. P., „Real-Time Computing in Open Systems for Manufacturing“, Journal of Computing and Information Science in Engineering, Nr. 1, S. 92–99, 2001.

[14] Bruyninckx, H., „Open robot control software: the OROCOS project“ in Proceedings of the International Conference on Robotics & Automation, ICRA, Seoul, Korea, 2001, S. 2523–2528.

[15] Finkemeyer, B., „Software als Tor zu neuen Roboterapplikationen“, Cluster Mechatronik & Automation - Workshop - Frischer Wind in der Industrierobotik. Augsburg, Mrz. 13 2012.

[16] W. Greulich, Hg., Der Brockhaus Computer und Informationstechnologie: Hardware, Software, Multimedia, Internet, Telekommunikation. Mannheim: Brockhaus, 2003.

[17] Beth, S., Rechtsprobleme proprietärer Standards in der Softwareindustrie: Eine Untersuchung rechtlicher Fragestellungen der de facto-Standardisierung bei der Entwicklung von Softwareschnittstellen unter besonderer Berücksichtigung der essential facilities-Doktrin, Göttingen: Cuvillier, 2005.

[18] Pritschow, G., Sperling, W., Müller, J. und Daniel, C., „OSACA - Auf dem Weg zur herstellerübergreifenden offenen Steuerung“ in Fortschritte der Fertigung auf Werkzeugmaschinen, Komponenten und Schnittstellen für offene Steuerungssysteme, G. Pritschow, C. Daniel und Pritschow-Spur-Weck, Hg., München: Carl Hanser Verlag, 1996, S. 1–26.

[19] ISO 9409-1:2004-03, Manipulating industrial robots — Mechanical interfaces — Part 1: Plates.

[20] Heinze, R., Aida: viele Projekte mit Profinet, 2009 März, [Website] https://www.openautomation.de/detailseite/aida-viele-projekte-mit-profinet.html (Zugriff am: 24. August 2019).

[21] Scharf, A., Automobilindustrie setzt die Vernetzungsstandards - ingenieur.de, 2008 April, [Website] https://www.ingenieur.de/technik/fachbereiche/automation/automobilindustrie-setzt-vernetzungsstandards/ (Zugriff am: 23. Mai 2020).

[22] VDI 2863 - Blatt 1:1987-12, Programmierung numerisch gesteuerter Handhabungseinrichtungen; IRDATA; Allgemeiner Aufbau, Satztypen und Übertragung (zurückgezogen).

[23] DIN 66314-1:1997-08, Schnittstelle zwischen Programmierung und Robotersteuerung - IRDATA - Teil 1: Allgemeiner Aufbau, Satztypen und Übertragung (zurückgezogen).

[24] ISO/TR 10562:1995-05, Manipulating industrial robots - Intermediate Code for Robots (ICR) (withdrawn).

[25] DIN 66312-1:1993-06, Industrieroboter - Programmiersprache - Industrial Robot Language (IRL) (zurückgezogen).

[26] DIN 66312-1:1996-09, Industrieroboter - Programmiersprache - Industrial Robot Language (IRL) (zurückgezogen).

[27] JIS B 8439:1992-02, Industrial robots - Programming language SLIM.

[28] Fraunhofer IPK, „RRS-Interface Specification: Version 1.3“ [Spezifikation], Berlin, 1997.

[29] Keibel, A., Konzeption und Realisierung eines integrierten Moduls zur Simulation und Steuerung von Kinematiksystemen [Dissertation], Dortmund: Technische Universität Dortmund, 2003.

[30] Bernhardt, R., Schreck, G. und Willnow, C., „The Virtual Robot Controller (VRC) Interface“ in Proceedings of the Automation & Transportation Technology Simulation and Virtual Reality Conference, ISATA, Dublin, Ireland, 2000.

[31] Weber, H., Konzeption und Realisierung einer hardwareunabhängigen Robotersteuerung mit offener Programmierschnittstelle [Dissertation], Dortmund: Technische Universität Dortmund, 1992.

[32] van der Valk, U., Konzeption und Realisierung einer PC-basierten Robotersteuerung [Dissertation], Düren: Shaker, 1995.

[33] Roshardt, R., Design und Implementierung einer offenen, modularen Robotersteuerung mit Oberon [Dissertation], Zürich: Eidgenössische Technische Hochschule Zürich, 1997.

[34] Gerkey, B. P., Vaughan, R. T., Stoy, K., Howard, A., Sukhatme, G. S. und Mataric, M. J., „Most valuable player: a robot device server for distributed control“ in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems. IROS: Expanding the Societal Role of Robotics in the the Next Millennium, Wailea, Hawaii, USA, 2001, S. 1226–1231.

[35] Gerkey, B. P., Vaughan, R. T. und Howard, A., „The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems“ in Proceedings of the International Conference on Advanced Robotics, ICAR, Coimbra, Portugal, 2003, S. 317–323.

[36] Vaughan, R. T. und Gerkey, B. P., „Reusable Robot Software and the Player/Stage Project“ in Springer tracts in advanced robotics, Software Engineering for Experimental Robotics, D. Brugali, Hg., Berlin, Heidelberg: Springer, 2007, S. 267–289.

[37] Wenz, M., Automatische Konfiguration der Bewegungssteuerung von Industrierobotern [Dissertation], Berlin: Logos, 2008.

[38] The Orocos Project: Smarter control in robotics & automation, [Website] http://www.orocos.org/ (Zugriff am: 28. Dezember 2018).

[39] Johns, K. und Taylor, T., Professional Microsoft Robotics Developer Studio 1. Aufl. Indianapolis: Wiley, 2008.

[40] Cepeda, J. S., Chaimowicz, L. und Soto, R., „Exploring Microsoft Robotics Studio as a Mechanism for Service-Oriented Robotics“ in Proceedings of the Latin American Robotics Symposium and Intelligent Robotics Meeting, LARS, Sao Bernardo do Campo, Brazil, 2010, S. 7–11.

[41] Quigley, M., Gerkey, B. P. und Conley, K., „ROS: an open-source Robot Operating System“.

[42] Thomas, D., Changes between ROS 1 and ROS 2, [Website] https://design.ros2.org/articles/changes.html (Zugriff am: 21. August 2019).

[43] Ackerman, E., Interview: Scott Hassan on Willow Garage and the Future of Suitable Technologies, 2013 August, [Website] http://spectrum.ieee.org/automaton/robotics/home-robots/interview-scott-hassan-on-willow-garage-and-the-future-of-suitable-tech (Zugriff am: 25. November 2018).

[44] D'Onfro, J., How a billionaire who wrote Google's original code created a robot revolution, 2016 Februar, [Website] http://www.businessinsider.de/a-look-back-at-willow-garage-2016-2 (Zugriff am: 25. November 2018).

[45] Open Source Robotics Foundation, ROS History - ROS moves to the Open Source Robotics Foundation, [Website] http://www.ros.org/history/ (Zugriff am: 25. November 2019).

[46] Open Source Robotics Foundation, Open Robotic, [Website] https://www.openrobotics.org/ (Zugriff am: 28. November 2019).

[47] Tellez, R., Top 10 ROS-based robotics companies to know in 2019, 2019 Juli, [Website] https://www.therobotreport.com/top-10-ros-based-robotics-companies-2019/ (Zugriff am: 21. August 2019).

[48] Michael, A., Thomas, J., Kühbeck, T., Seidl, B., Friedl, M. und Scheickl, O., „Automated Driving with ROS at BMW“ in ROSCon 2015, Hamburg, 2015.

[49] Pluta, W., BMW richtet Entwicklungszentrum für autonomes Fahren ein - Golem.de, 2016 Dezember, [Website] https://www.golem.de/news/auto-bmw-richtet-entwicklungszentrum-fuer-autonomes-fahren-ein-1612-125182.html (Zugriff am: 23. Mai 2020).

[50] Poll, D., In drei Schritten zum perfekten Transportroboter - produktion.de, 2018 August, [Website] https://www.produktion.de/trends-innovationen/in-drei-schritten-zum-perfekten-transportroboter-112.html (Zugriff am: 23. Mai 2020).

[51] ROS Industrial Consortium America, ROS-Industrial - About, [Website] http://rosindustrial.org/about/description/ (Zugriff am: 25. November 2018).

[52] Weisshardt, F., Bubeck, A. und Reiser, U., „Open-Source-Softwareplattform ROS-Industrial: Offenheit schafft Effizienz“, DIGITAL ENGINEERING Magazin, Nr. 06, S. 60–61, 2014.

[53] Rickert, M. und Gaschler, A., „Robotics library: An object-oriented approach to robot applications“ in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, Vancouver, BC, Canada, 2017, S. 733–740.

[54] Elkady, A. und Sobh, T., „Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography // Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography“, Journal of Robotics, Nr. 5, S. 1–15, 2012.

[55] Zug, S., Poltrock, T., Penzlin, F., Walter, C. und Hochgeschwender, N., „Analyse und Vergleich von Frameworks für die Implementierung von Robotikanwendungen“ [Technischer Report], Otto-von-Guericke-Universität, Magdeburg, 2013.

[56] Plank, G., Reintsema, D., Grunwald, G., Otter, M., Kurze, M., Löhning, M, Reiner, M., Zimmermann, U., Schreiber, G., Weiss, Bischoff, R., Fellhauer, B., Notheis, T. und Barklage, T., „PAPAS: Plug and Play Antriebs- und Steuerungskonzepte für die Produktion von morgen“ [Abschlussbericht], 2006.

[57] Gauß, M., „Schnittstellenstandardisierung für Industrieroboter und komplexe Sensoren: XIRP: XML Interface for Robots and Peripherals“. Nürnberg, Nov. 24 2005.

[58] VDMA 66430-1:2006-07, XML-basiertes Kommunikationsprotokoll für Industrieroboter und prozessorgestützte Peripheriegeräte (XIRP) - Teil 1: Allgemeine Vereinbarungen.

[59] Herzinger, T., „ReApp – Wiederverwendbare Roboterapplikationen für flexible Roboteranlagen basierend auf Industrial ROS (BMW-Beitrag)“ [Abschlussbericht], 2017.

[60] Taschew, M., „ReApp: Reusable Robot Apps based on ROS-Industrial“, IAS-13 Workshop: ROS-Industrial in European Research Projects. Padua, Italien, Jul. 15 2014.

[61] Awad, R., Reiser, U., Ahmed, N. und Zander, S., „Wiederverwendbare Roboterapplikationen: Flexiblere Automatsisierung“, DIGITAL ENGINEERING Magazin, Nr. 02, S. 19–21, 2015.

[62] Nilsson, K., Johansson, R., Robertsson, A., Bischoff, R., Brogårdh, T. und Hägele, M., „Productive Robots and the SMErobot Project“ in Book of Abstracts of Third Swedish Workshop on Autonomous Robotics, SWAR, Stockholm, Schweden, 2005.

[63] Perzylo, A., Rickert, M., Kahl, B., Somani, N., Lehmann, C., Kuss, A., Profanter, S., Beck, A. B., Haage, M., Rath Hansen, M., Nibe, M. T., Roa, M. A., Sornmo, O., Gestegard Robertz, S., Thomas, U., Veiga, G., Topp, E. A., Kesslar, I. und Danzer, M., „SMErobotics: Smart Robots for Flexible Manufacturing“, IEEE Robotics & Automation Magazine, Jg. 26, Nr. 1, S. 78–90, 2019.

[64] Naumann, M., Bengl, M. und Verl, A., „Automatic generation of robot applications using a knowledge integration framework“ in Proceedings of the ISR 2010 (41st International Symposium on Robotics) and ROBOTIK 2010 (6th German Conference on Robotics), München, 2010.

[65] Angerer, A., Object-oriented Software for Industrial Robots [Dissertation], Augsburg: Universität Augsburg, 2014.

[66] Finkemeyer, B., Robotersteuerungsarchitektur auf der Basis von Aktionsprimitiven [Dissertation], Aachen: Shaker, 2004.

[67] Friedrich, T., Technologieorientiertes Programmier- und Steuerungssystem für Industrieroboter [Dissertation], Berlin: Technische Universität Berlin, 2010.

[68] Maletzki, G., Rapid Control Prototyping komplexer und flexibler Robotersteuerungen auf Basis des SBC-Ansatzes [Dissertation], Rostock: Universität Rostock, 2013.

[69] Lambrecht, J., Natürlich-räumliche Industrieroboterprogrammierung auf Basis makerloser Gestenerkennung und mobiler Augmented Reality [Dissertation], Berlin: Technische Universität Berlin, 2014.

[70] Lambrecht, J., Chemnitz, M. und Krüger, J., „Control layer for multi-vendor industrial robot interaction providing integration of supervisory process control and multifunctional control

units“ in Proceedings of the IEEE Conference on Technologies for Practical Robot Applications, TePRA, Woburn, MA, USA, 2011, S. 115–120.

[71] Proctor, F. M., Balakirsky, S. B., Kootbally, Z., Kramer, T. R., Schlenoff, C. I. und Shackleford, W. P., „The Canonical Robot Command Language (CRCL)“, The Industrial robot, Jg. 43, Nr. 5, S. 495–502, 2016.

[72] Kroh, R., Robotersteuerungen - Integrierter CNC-Kern ermöglicht Direktverarbeitung, 2012 Mai, [Website] http://www.maschinenmarkt.vogel.de/integrierter-cnc-kern-ermoeglicht-direktverarbeitung-von-g-code-a-363057/ (Zugriff am: 29. Dezember 2019).

[73] DIN 66025-1:1983-01, Programmaufbau für numerisch gesteuerte Arbeitsmaschinen; Allgemeines.

[74] KUKA Roboter GmbH, „KUKA.CNC for maximum robot performance in machinen processes“ [Produktbroschüre], Gersthofen.

[75] KUKA Roboter GmbH, „KUKA Sunrise.OS 1.7/KUKA Sunrise.Workbench 1.7: Bedien- und Programmieranleitung“ [Handbuch], Augsburg, 2015.

[76] Dassault Systemes Deutschland GmbH, Delmia Robotics, [Website] https://www.3ds.com/de/produkte-und-services/delmia/produkte/delmia-3dexperience/der-nutzen-fuer-die-digitale-fertigung/robotics/ (Zugriff am: 21. Mai 2020).

[77] Siemens Industry Software Inc., Siemens PLM - Process Simulate Robotics: Robotik- und Automatisierungssimulation, [Website] https://www.plm.automation.siemens.com/global/de/products/manufacturing-planning/robotics-automation-simulation.html (Zugriff am: 21. Mai 2020).

[78] ArtiMinds Robotics GmbH, „ArtiMinds Robot Programming Suite“ [Produktbroschüre], 2019.

[79] drag and bot GmbH, „drag&bot - robot programming framework“ [Produktbroschüre], Stuttgart, 2019. Zugriff am: 24. Mai 2020.

[80] Hüthig GmbH, Einfache Roboterprogrammierung via Drag-and-drop: Produktbericht, 2018 November, [Website] https://www.neue-verpackung.de/63148/einfache-roboterprogrammierung-via-drag-and-drop/ (Zugriff am: 26. Februar 2021).

[81] KEBA AG, „KeMotion“ [Produktbroschüre], [Website] https://www.keba.com/file/downloads/industrial-automation/brochures/KeMotion_Prospekt_DE.pdf. Zugriff am: 24. Mai 2020.

[82] Henrich Publikationen GmbH, Automation - Echte Robomantik, 2019 September, [Website] https://www.automationnet.de/echte-robomantik-87686 (Zugriff am: 24. Mai 2020).

[83] Müller, B., Die Zukunft der Fertigung: Gemeinsame Sprache für Roboter und Maschine, [Website] https://new.siemens.com/global/de/unternehmen/stories/forschung-technologien/folder-topics/die-zukunft-der-fertigung-run-my-robot.html (Zugriff am: 24. Mai 2020).

[84] Comau Deutschland GmbH, „openROBOTICS: Bedienen Sie den Roboter mit Ihrer eigenen Steuerung“ [Produktbroschüre], Köln, 2016, [Website] https://www.comau.com/Download/our-competences/robotics/Automation_Products/Comau%20openROBOTICS%20Brochure%20GER%20(2016-11).pdf. Zugriff am: 9. April 2020.

[85] Stäubli International AG, „uniVAL Drive: "Ready to plug" robot solutions for generic multi-axis controllers“ [Produktbroschüre], Faverges, 2012.

[86] KUKA Roboter GmbH, „S7 Library: KUKA.PLC mxAutomation 2.0“ [Handbuch], Augsburg, 2014.

[87] Kreil, C., 2013 - Auszeichnung für iRobFeeder, 2013 Februar, [Website] http://blog.profactor.at/2013/02/26/auszeichnung-fur-irobfeeder-2/ (Zugriff am: 29. Dezember 2018).

[88] DigiMetrix GmbH, 2016 - Digimetrix - add Sense to Robotics, [Website] http://www.digimetrix.com/de/ (Zugriff am: 29. Dezember 2018).

[89] CAN in Automation e. V., „CiA 402 series: CANopen device profile for drives and motion control“ [Technische Spezifikation], 2016.

[90] PROFIBUS Nutzerorganisation e. V., „PROFIdrive Systembeschreibung: Technologie und Anwendung“ [Produktbroschüre], Karlsruhe, 2012.

[91] PLCopen, „Function blocks for motion control: PLCopen - Technical Committee 2 – Task Force“ [Technische Spezifikation], Gorinchem, 2011.

[92] OPC Foundation, „OPC Unified Architecture: Interoperabilität für Industrie 4.0 und das Internet der Dinge“ [Produktbroschüre], Verl, 2016, [Website] https://opcfoundation.org/wp-content/uploads/2016/05/OPC-UA-Interoperability-For-Industrie4-and-IoT-DE-v5.pdf. Zugriff am: 25. November 2016.

[93] VDMA 40010-1:2018-12, OPC UA Companion Specification Robotik – Teil 1: Zustandsdatenerfassung, Asset Management, vorausschauende Wartung, vertikale Integration.

[94] Dorofeev, K., Cheng, C.-H., Ferreira, P., Profanter, S. und Zoitl, A., „Device Adapter Concept towards Enabling Plug&Produce Production Environments“ in Proceedings of the 22nd IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, Limassol, Zypern, 2017.

[95] Kaspar, M., Bock, J., Kogan, Y., Venet, P., Weser, M. und Zimmermann, U. E., „Tool and Technology Independent Function Interfaces by Using a Generic OPC UA Representation“ in Proceedings of the 23rd IEEE International Conference on Emerging Technologies and Factory Automation, ETFA,, Turin, Italien, 2018, S. 1183–1186.

[96] Profanter, S., Breitkreuz, A., Rickert, M. und Knoll, A., „A Hardware-Agnostic OPC UA Skill Model for Robot Manipulators and Tools“ in Proceedings of the 24th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, Zaragoza, Spanien, 2019.

[97] Rosch, R., Audi will das Fließband abschaffen, 2016 November, [Website] https://www.manager-magazin.de/unternehmen/autoindustrie/audi-will-das-fliessband-abschaffen-a-1123058-2.html (Zugriff am: 23. August 2019).

[98] Käfer, S., Losgröße 1 durch Additive Fertigung bei BMW, 2019 Juni, [Website] https://www.maschinenmarkt.vogel.de/losgroesse-1-durch-additive-fertigung-bei-bmw-a-838488/ (Zugriff am: 23. August 2019).

[99] Berlin, C., Autoindustrie sieht Losgröße 1 skeptisch, 2017 November, [Website] https://www.automotiveit.eu/autoindustrie-sieht-losgroesse-1-skeptisch/news/id-0059470 (Zugriff am: 23. August 2019).

[100] Bix, J., Mobile Robotik in der bandsynchronen Montage zur flexiblen Mensch-Roboter-Interaktion [Dissertation], Stuttgart: Fraunhofer Verlag, 2019.

[101] Kossmann, M., Sicherheit in der Mensch-Roboter-Interaktion durch einen biofidelen Bewertungsansatz [Dissertation], München: Technische Universität München, 2019.

[102] Dinse, J., Quantitative Betriebsmittelbedarfsplanung für die getaktete Fließfertigung [Dissertation], Stuttgart: Fraunhofer Verlag, 2016.

[103] Ihme, J., Logistik im Automobilbau: Logistikkomponenten und Logistiksysteme im Fahrzeugbau // Logistikkomponenten und Logistiksysteme im Fahrzeugbau, München: Hanser, 2006.

[104] DIN 62264-1:2014-07, Integration von Unternehmensführungs- und Leitsystemen - Teil 1: Modelle und Terminologie (IEC 62264-1:2013).

[105] Ramer, C., Arbeitsraumüberwachung und autonome Bahnplanung für ein sicheres und flexibles Roboter-Assistenzsystem in der Fertigung [Dissertation], Erlangen: FAU University Press.

[106] Mareczek, J., „Pfad- und Bahnplanung“ in Grundlagen der Roboter-Manipulatoren – Band 2, J. Mareczek, Hg., Berlin: Springer, 2020, S. 1–34.

[107] Open Source Robotics Foundation, ROS Wiki - Industrial - Supported Hardware (Zugriff am: 24. März 2021).

[108] KUKA Roboter GmbH, „KUKA.RoboterSensorInterface 3.2: Für KUKA Sysem Software 8.3“ [Handbuch], Augsburg, 2013.

[109] Rickert, M., Robotics Library, [Website] https://www.roboticslibrary.org/ (Zugriff am: 23. Mai 2020).

[110] Werner, J., Methode zur roboterbasierten förderbandsynchronen Fließmontage am Beispiel der Automobilindustrie [Dissertation], München: Herbert Utz Verlag, 2009.

[111] ISO 9283:1998-04, Industrieroboter - Leistungskenngrößen und zugehörige Prüfmethoden (ISO 9283:1998).

[112] Open Source Automation Development Lab (OSADL) eG, Realtime Linux: OSADL - Open Source Automation Development Lab, [Website] https://www.osadl.org/Realtime-Linux.projects-realtime-linux.0.html (Zugriff am: 11. Januar 2021).

[113] VDI 2143 Blatt 1:1980-10, Bewegungsgesetzte für Kurvengetriebe - Theoretische Grundlagen.

[114] RoboDK Inc., Simulator for industrial robots and offline programming - RoboDK, [Website] https://robodk.com/ (Zugriff am: 21. Mai 2020).

[115] Hexagon AB, RoboDyn: Industrial Robot Calibration and ISO Performance Testing System, [Website] https://www.hexagonmi.com/en-US/products/robotics-and-automation/robodyn (Zugriff am: 3. Juni 2020).

[116] Wu, K., Krewet, C. und Kuhlenkötter, B., „Dynamic performance of industrial robot in corner path with CNC controller“, Robotics and Computer-Integrated Manufacturing, Jg. 54, 9–12, S. 156–161, 2017.

[117] Naumann, M., Wegener, K. und Schraft, R. D., „Control Architecture for Robot Cells to Enable Plug'n'Produce“ in Proceedings of the IEEE International Conference on Robotics and Automation, ICRA, Rom, Italien, 2007, S. 287–292.

[118] Taschew, M., “Verfahren zur automatischen Konfiguration eines externen Steuerungssystems zur Steuerung und/oder Regelung eines Robotersystems,” EP 3 356 894 A1, European Patent Office 10 2015 218 699.7, Aug 8, 2018.

[119] Taschew, M., “Method for the Automatic Configuration of an External Control System for the Open-Loop And/Or Closed-Loop Control of a Robot System,” US 10,786,898 B2, United States Patent Office 15 / 830 057, Sep 29, 2020.

[120] Taschew, M., “用于控制和/或调节机器人系统的外部控制系统的自动配置方法,” CN 107710082 B, State Intellectual Property Office China CN 107710082 B, Jan 26, 2021.

[121] Taschew, M., “Verfahren zur automatischen Konfiguration eines externen Steuerungssystems zur Steuerung und/oder Regelung eines Robotersystems,” DE 10 2015 218 699 A1, Deutsches Patent- und Markenamt 10 2015 218 699.7, Mrz 30, 2017.

[122] Taschew, M., “Verfahren zur automatischen Konfiguration eines externen Steuerungssystems zur Regelung und/oder Steuerung eines Robotersystems,” DE 10 2015 218 697 A1, Deutsches Patent- und Markenamt 10 2015 218 697.0, Mrz 30, 2017.

[123] Gaschler, A., Springer, M., Rickert, M. und Knoll, A., „Intuitive robot tasks with augmented reality and virtual obstacles“ in Proceedings of the IEEE International Conference on Robotics and Automation, ICRA, Hong Kong, China, 2014, S. 6026–6031.

[124] Lozano-Pérez, T., „Robot Programming“ [Technischer Bericht], MIT - Massachusetts Institute of Technology, Cambridge A.I. Memo No. 698, 1982.

[125] Biggs, G. und MacDonald, B., „A Survey of Robot Programming Systems“ in Proceedings of the Australasian Conference on Robotics and Automation, ACRA, Brisbane, Australien, 2003.

[126] Vogl, W., Eine interaktive räumliche Benutzerschnittstelle für die Programmierung von Industrierobotern [Dissertation], München: Herbert Utz Verlag, 2009.

[127] Weck, M. und Brecher, C., Werkzeugmaschinen: Automatisierung von Maschinen und Anlagen, 6. Aufl. Berlin: Springer, 2006.

[128] Kugelmann, D., Aufgabenorientierte Offline-Programmierung von Industrierobotern [Dissertation], München: Herbert Utz Verlag, 1999.

[129] Gruhn, V., Pieper, D. und Röttgers, C., MDA®: Effektives Software-Engineering mit UML2® und EclipseTM, Berlin: Springer, 2007.

[130] Universal Robots A/S, „The URScript Programming Language: API Reference - Version 3.5.4“ [Handbuch], Odense, Dänemark, 2018.

[131] KUKA AG, KUKA Sunrise.OS, [Website] https://www.kuka.com/de-de/produkte-leistungen/robotersysteme/software/systemsoftware/sunriseos (Zugriff am: 25. April 2020).

[132] Chakravarty, S., World’s Top 10 Industrial Robot Manufacturers _ Market Research Reports® Inc_, [Website] https://www.marketresearchreports.com/blog/2019/05/08/world%E2%80%99s-top-10-industrial-robot-manufacturers (Zugriff am: 7. Juni 2020).

[133] automationnet - Heinrich Publikationen GmbH, Weltrangliste Robotik, [Website] https://www.automationnet.de/weltrangliste-robotik (Zugriff am: 7. Juni 2020).

[134] ABB Asea Brown Boveri Ltd, „Technisches Referenzhandbuch: Systemparameter“ [Handbuch], Västerås, Schweden, 2013.

[135] ABB Asea Brown Boveri Ltd, „Technisches Referenzhandbuch: RAPID Instruktionen, Funktionen und Datentypen“ [Handbuch], Västerås, Schweden, 2013.

[136] KUKA Roboter GmbH, „KUKA System Software 8.3: Bedien- und Programmieranleitung für Systemintegratoren“ [Handbuch], Augsburg, 2014.

[137] COMAU S.p.A., „PDL2: Programming Language Manual“ [Handbuch], Grugliasco, Italien, 2012.

[138] COMAU S.p.A., „C5G - Controller Unit: Motion Programming“ [Handbuch] , Grugliasco, Italien, 2012.

[139] FANUC America Corporation, „FANUC Robotics SYSTEM R-30iA and R-30iB Controller: KAREL Reference Manual“ [Handbuch], Rochester Hills, Michigan, USA, 2013.

[140] FANUC America Corporation, „FANUC America Corporation SYSTEM R-30iB: Setup and Operations Manual - HandlingTool and MATE HandlingTool“ [Handbuch], Rochester Hills, Michigan, USA, 2014.

[141] FANUC Deutschland GmbH, „FANUC Roboterserie - R30iB: Bedienungshandbuch - Punktschweißfunktion“ [Handbuch], Neuhausen a. d. F., 2013.

[142] FANUC Deutschland GmbH, „FANUC Roboterserie - R30iB: Bedienungshandbuch - Grundlegende Bedienvorgänge“ [Handbuch], Neuhausen a. d. F., 2013.

[143] Haun, M., Handbuch Robotik: Programmieren und Einsatz intelligenter Roboter 1. Aufl. Berlin: Springer, 2007.

[144] DIN EN 61131-3:2014-06, Speicherprogrammierbare Steuerungen - Teil 3: Programmiersprachen (IEC 61131-3:2013).

[145] John., K. H. und Tiegelkamp, M., SPS-Programmierung mit IEC 61131-3: Konzepte und Programmiersprachen, Anforderungen an Programmiersysteme, Entscheidungshilfen, 4. Aufl., Dordrecht: Springer, 2009.

[146] ISO 6983-1:2009-12, Automationssysteme und Integration - Steuerung von Maschinen - Programmformat und Definition von Adresswörtern - Teil 1: Datenformat für Positionierung, Linearbewegungen und Bahnsteuerungen.

[147] Suk-hwan, S. und Seung-jun, S., “Transformation Method of G-Code into Step-NC Part Program,” US 8,041,445 B2, United States Patent Office, Okt 18, 2011.

[148] Warfield, B., CNC Post Processors & CAM for Different G-Code Dialects, [Website] https://www.cnccookbook.com/cnc-post-processor-cam/ (Zugriff am: 23. Juni 2019).

[149] Venkatesh, S., Odendahl, D., Xu, X., Michaloski, J., Proctor, F. und Kramer, T., „Validating Portability of STEP-NC Tool Center Programming“ in Proceedings of the ASME Design Engineering Technical Conferences and Computers and Information in Engineering Conference, Long Beach, California, USA, 2005, S. 285–290.

[150] ISO 14649-10:2004-12, Automationssysteme und Integration - Steuerungen - Datenmodell für CNC - Teil 10: Allgemeine Prozeßdaten.

[151] ISO 10303-238:2020-11, Industrial automation systems and integration - Product data representation and exchange - Part 238: Application protocol: Model based integrated manufacturing.

[152] Othman, M. A., Minhat, M. und Jamaludin, Z., „An overview on STEP-NC compliant controller development“ in Proceedings of the 4th International Conference on Mechanical Engineering Research, ICMER, Pahang, Malaysia, 2017.

[153] Solvang, B., Refsahl, L. K. und Sziebig, G., „STEP-NC Based Industrial Robot CAM System“, IFAC Proceedings Volumes, Jg. 42, Nr. 16, S. 245–250, 2009.

[154] P. A. Henning und H. Vogelsang, Hg., Taschenbuch Programmiersprachen: Mit zahlr. Tabellen, 2. Aufl. Leipzig: Carl Hanser Verlag, 2007.

[155] Fernández, M., Programming Languages and Operational Semantics: A Concise Overview, London: Springer, 2014.

[156] Lischka, P. und Anton, S., „EASY-ROB Produktbeschreibung“ [Produktbroschüre], Hofheim am Taunus, 2019.

[157] KUKA Roboter GmbH, „Industrierobotik - Mittlere Traglast“ [Produktbroschüre], Gersthofen, 2017, [Website] https://www.kuka.com/-/media/kuka-downloads/imported/9cb8e311bfd744b4b0eab25ca883f6d3/kuka_pb_mittlere_tl_de.pdf. Zugriff am: 1. Februar 2021.

[158] ABB Asea Brown Boveri Ltd, Plastic Software - ABB Robotics - Application Software Solutions for Robots | ABB Robotics, [Website] https://new.abb.com/products/robotics/application-software/plastic (Zugriff am: 1. Februar 2021).

[159] FANUC Deutschland GmbH, FANUC iPendant Touch, [Website] https://www.fanuc.eu/de/de/roboter/zubeh%C3%B6r/robot-controller-and-connectivity/ipendant-touch (Zugriff am: 1. Februar 2021).

[160] COMAU S.p.A., Comau - Programmierhandgerät, [Website] https://www.comau.com/de/unsere-kompetenzen/robotik/programmierhandgerat (Zugriff am: 12. August 2020).

[161] Fowler, M., Language Workbenches: The Killer-App for Domain Specific Languages?, 2015 Juni, [Website] https://martinfowler.com/articles/languageWorkbench.html (Zugriff am: 10. Juni 2020).

[162] Causevic, A. (Taschew M.), Konzept und prototypische Implementierung einer visuellen Programmierumgebung für Roboterprogrammmodifikationen auf Shop Floor Ebene, [Bachelorarbeit], HS Ravensburg-Weingarten, unveröffentlicht, 2015

[163] Aykut, T. (Taschew M.), Konzeption und Prototypische Implementierung eines Entwicklungs- und Interpretersystems zur domänenspezifischen und herstellerunabhängigen Programmierung von Industrierobotern, [Masterarbeit], TU-München, unveröffentlicht, 2016

[164] Deutsche Gesetzliche Unfallversicherung (DGUV), „DGUV Information 209-074 „Industrieroboter““ Berlin, 2015.

[165] Larsen, L., Auf dem Weg zu einer flexiblen Produktion: Automatische und kollisionsfreie Bahnplanung für kooperierende Industrieroboter [Dissertation], Augsburg: Universität Augsburg, 2019.

[166] Ozminski, O. (Taschew M.), Vergleich von industriellen Roboterprogrammiersprachen im automobilen Karosseriebau für Standardisierungsansätze sowie Erstellung eines Analysetools, [Masterarbeit], HS München, unveröffentlicht, 2015