Ruhr-Universität Bochum
Lehrstuhl für Produktionssysteme
Prof. Dr.-Ing. Bernd Kuhlenkötter

Entwicklung eines Frameworks zur simulationsbasierten Validierung LLM-generierten Robotercodes in Unity

Dem Fachbereich Maschinenbau der
Ruhr-Universität Bochum
zur Erlangung des akademischen Grades eines
Bachelor of Science

eingereichte Bachelor-Thesis

von
Lukas Lux

Erstprüfer: Prof. Dr.-Ing. Bernd Kuhlenkötter
Zweitprüfer: Prof. Dr.-Ing. Christopher Prinz
Betreuer: Daniel Syniawa, M. Sc.

Datum der Einreichung: 1. September 2025

Aufgabenstellung

Thema: Entwicklung eines Frameworks zur simulationsbasierten Validierung LLM-generierten Robotercodes in Unity

Für den Einsatz von Arbeitsplatzsystemen im Bereich der Montage, an denen Mensch und Roboter miteinander kollaborieren (MRK), gibt es bislang noch keine digitalen Planungswerkzeuge, welche ein MRK-System hinsichtlich Automatisierbarkeit, tech-nisch-wirtschaftlicher Eignung, Ergonomie und Sicherheit simulieren und bewerten können. Um zukünftig die einfache Planung und Simulation von kollaborativen Montagesystemen zu ermöglichen, wird im Forschungsprojekt „KoMPI“ ein digitales Planungswerkzeug entwickelt, sodass sowohl Mensch als auch Roboter gezielt gemeinsam im Montageprozess eingesetzt werden können. Derzeit existieren bereits unterschiedliche Simulationswerkzeuge, die Menschmodelle in manuellen Montagesystemen abbilden. Hierzu zählt bspw. die digitale Planungssoftware Ema der Fa. imk. Bis dato bietet auch Ema keine hinreichenden Möglichkeiten zur Simulation automatisierungstechnischer Komponenten (z. B. Roboter), insbesondere in Kollaboration mit dem Menschen. ... Im Rahmen dieser Arbeit gilt es ...

Im Einzelnen sollten folgende Punkte bearbeitet werden:

- Literaturrecherche zum Thema Mensch-Roboter-Kollaboration (MRK) und Planung kollaborativer Montagesysteme
- ...
- mein name ist lukas und dieser satz wird immer laenger das wuerde bedeuten dass sdfnmsdfs asdasda und was bedeutet das in echt?

Ausgabedatum: 16.06.2025

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation und Relevanz

1.2 Zielsetzung und Aufbau der Arbeit

Kapitel 2

Grundlagen und Stand der Technik

2.1 Grundlagen der Robotersimulation

2.2 Large Language Models (LLMs)

2.3 Stand der Technik: LLMs in der Robotik

2.4 Stand der Technik: Robotersimulation

Bezugnahme auf aktuelle Techniken in der Robotersimulation -> Lehrbücher etc. bieten hier eine Grundlagen Aktualität muss nicht gewähleistet sein da relativ statisch Auf jeden Fall Abgrenzung warum ich Unity verwende: Open-Source von Unity

Kapitel 3

Methodik und Implementierung

3.1 Systemarchitektur und Design

3.1.1 Architekturübersicht

Die Systemarchitektur des entwickelten Roboter-Kommunikationsframeworks folgt einer mehrschichtigen Architektur (Layered Architecture), die eine klare Trennung von Verantwortlichkeiten und eine hohe Modularität gewährleistet [5, 11]. Diese architektonische Entscheidung basiert auf etablierten Softwarearchitekturprinzipien und ermöglicht die Realisierung eines erweiterbaren, wartbaren und testbaren Systems für die Integration heterogener Robotersysteme.

3.1.2 Unity3D als Simulationsplattform

Die Wahl von Unity3D als zugrundeliegende Simulationsplattform basiert auf mehreren technischen und praktischen Erwägungen. Während es bereits mehrere kommerzielle Programme für die Gestaltung und Simulation von Robotern in virtuellen Umgebungen gibt - wie Robcad, Robotstudio, Igrip, Workcell und Gazebo - sind nicht alle Programme mit anderen CAD-Systemen kompatibel, unterstützen nicht alle Roboterbibliotheken oder andere Elemente, und einige werden nicht unter Windows vertrieben (vgl.[2]). Unity3D hingegen ist mit den meisten CAD-Systemen kompatibel und bietet eine plattformübergreifende Lösung. Unity3D bietet eine ausgereifte 3D-Rendering-Pipeline mit integrierter Physik-Engine (PhysX), welche zur Simulation von Gegenständen mit realitätsnahem Verhalten sowie komplexen Arbeitsräumen geeignet ist. Die Engine wurde bereits erfolgreich in der wissenschaftlichen Forschung eingesetzt und bietet Module und Plugins für spezifische Anwendungsfälle im Simulationsbereich. Unity3D ermöglicht es auch Nicht-Programmierern, leistungsstarke Animations- und Interaktionsdesign-Tools zu nutzen, um Roboter visuell zu programmieren und zu animieren (vgl. [4]).
Technisch ermöglicht Unity3D durch seine Scripting-Runtime (basierend auf Mono/.NET) die Verwendung moderner C#-Sprachfeatures für nebenläufige Prozesse und asynchroner Programmierung, was es ermöglicht, Prozesse in Nahe-Echtzeit darzustellen und zu überwachen. Ein entscheidender Vorteil für die Robotik-Simulation liegt in der Verfügbarkeit visueller Programmiertools. Die Plattform bietet umfangreiche Debugging- und Profiling-Werkzeuge, die während der Entwicklung und zur Laufzeit genutzt werden können. Darüber hinaus lassen sich während der Laufzeit die Szene

(hier: der Arbeitsraum) bearbeiten und aktuelle Parameter einsehen, was besonders für die iterative Entwicklung und das Testen ist.

Abhängigkeitsverwaltung und Modularität

3.1.3 Design Patterns und Prinzipien

3.1.4 Modularitäts- und Erweiterbarkeitskonzept

Die Architektur des entwickelten Systems folgt einem konsequenten Modularitätskonzept, das auf drei zentralen Säulen basiert: einer Plugin-Architektur für die Integration neuer Robotertypen, einer durchgängigen Interface-basierten Abstraktion sowie einer hierarchischen Namespace-Struktur. Diese Designentscheidungen ermöglichen die Erweiterung des Frameworks ohne Modifikation bestehender Komponenten und erfüllen damit das Open-Closed-Prinzip [20].

Plugin-Architektur für Robotertypen

Das Framework implementiert eine Plugin-basierte Architektur, die es ermöglicht, neue Robotertypen ohne Änderungen am Core-System zu integrieren. Jeder Robotertyp wird in einem eigenen Namespace-Verzeichnis gekapselt (z.B. `RobotSystem/ABB/` für ABB-Roboter), wobei die Integration ausschließlich über standardisierte Interfaces erfolgt. Diese Architektur folgt dem Konzept der *Dependency Inversion* [23], bei der High-Level-Module (wie der `RobotManager`) nicht von Low-Level-Modulen (spezifische Roboter-Implementierungen) abhängen, sondern beide von Abstraktionen.

Die Vorteile dieser Architektur zeigen sich besonders bei der Integration neuer Roboterhersteller. So könnte beispielsweise ein KUKA-Roboter durch einfaches Hinzufügen eines `RobotSystem/KUKA/` Verzeichnisses mit entsprechenden Interface-Implementierungen integriert werden, ohne dass bestehende ABB-Implementierungen oder Core-Komponenten modifiziert werden müssten. Dies reduziert das Risiko von Regressionsfehlern und ermöglicht parallele Entwicklung verschiedener Roboter-Integrationen [13].

Interface-basierte Abstraktion

Die Abstraktion erfolgt über vier zentrale Interface-Definitionen, die als Kontrakte zwischen den Systemkomponenten fungieren:

- **IRobotConnector**: Definiert die Schnittstelle für Roboterverbindungen, unabhängig vom Kommunikationsprotokoll
- **IRobotSafetyMonitor**: Standardisiert die Integration von Sicherheitsmonitoren
- **IRobotDataParser**: Ermöglicht austauschbare Parser für verschiedene Datenformate
- **IRobotVisualization**: Abstrahiert Visualisierungssysteme

Diese Interface-Segregation [20] stellt sicher, dass Komponenten nur von den tatsächlich benötigten Abstraktionen abhängen. Der `RobotManager` beispielsweise arbeitet ausschließlich mit der `IRobotConnector`-Schnittstelle und ist damit vollständig entkoppelt von spezifischen Implementierungsdetails wie WebSocket-Protokollen oder HTTP-Polling-Mechanismen.

Die Verwendung des Strategy-Patterns [13] für die Safety-Monitore ermöglicht es, verschiedene Überwachungsalgorithmen zur Laufzeit auszutauschen. So implementieren sowohl `CollisionDetectionMonitor` als auch `SingularityDetectionMonitor` das `IRobotSafetyMonitor`-Interface, können aber völlig unterschiedliche Detektionsstrategien verwenden. Diese Flexibilität ist essentiell für die Anpassung an verschiedene Sicherheitsanforderungen und Industrienormen.

Namespace-Struktur und Paketierung

Die hierarchische Namespace-Struktur folgt dem Prinzip der *Package by Feature* [22], wobei funktional zusammengehörige Komponenten in gemeinsamen Namespaces organisiert sind:

```
RobotSystem/
├── Core/
├── Interfaces/
├── ABB/
│   └── RWS/
└── Safety/
```

Diese Struktur bietet mehrere Vorteile für die Wartbarkeit und Erweiterbarkeit:

1. **Klare Verantwortlichkeiten**: Jeder Namespace hat eine eindeutig definierte Zuständigkeit
2. **Minimale Kopplung**: Abhängigkeiten verlaufen nur von spezifischen zu allgemeinen Namespaces
3. **Einfache Navigation**: Die Struktur spiegelt die konzeptuelle Architektur wider
4. **Versionierbarkeit**: Herstellerspezifische Implementierungen können unabhängig versioniert werden

Die Verwendung von Unity-spezifischen Meta-Dateien (`.meta`) in Kombination mit der Namespace-Struktur ermöglicht zudem die nahtlose Integration in die Unity-Engine, wobei die logische Strukturierung auch auf Dateisystemebene erhalten bleibt. Dies erleichtert die Zusammenarbeit in Teams und die Versionskontrolle mit Git [7].

Das Modularitätskonzept zeigt sich auch in der Möglichkeit, einzelne Module als Unity-Packages zu exportieren und in anderen Projekten wiederzuverwenden. Die strikte Einhaltung der Interface-Kontrakte garantiert dabei die Kompatibilität zwischen verschiedenen Versionen und Konfigurationen des Systems.

Ein zentrales Architekturprinzip des Frameworks ist die konsequente Anwendung der Dependency Inversion [24]. Konkrete Implementierungen hängen von abstrakten

Interfaces ab, nicht von anderen konkreten Klassen. Dies wird durch die Definition der Kerninterfaces `IRobotConnector`, `IRobotSafetyMonitor` und `IRobotVisualization` erreicht, die als Kontraktdefinitionen zwischen den Schichten fungieren.

Die Interfaces definieren dabei nicht nur die Methodensignaturen, sondern etablieren auch semantische Kontrakte im Sinne des Design by Contract [25]. Beispielsweise garantiert das `IRobotConnector`-Interface, dass Zustandsänderungen über Events propagiert werden und dass die Verbindung idempotent hergestellt und getrennt werden kann.

Die lose Kopplung ermöglicht es, verschiedene Robotertypen durch Implementation des `IRobotConnector`-Interfaces zu unterstützen, ohne Änderungen am Kern des Frameworks vornehmen zu müssen. Dies demonstriert die Erweiterbarkeit der Architektur und validiert die Designentscheidung für eine interface-basierte Abstraktion [17].

3.2 Entwicklungsmethodik und Implementierung

3.2.1 Prozessfolgen

User Input: Simulationsumgebung und Verbindung zu Robot Studio

Wie kann ich Prozessfolgen überprüfen? Prozessfolge: Folge and Arbeitsschritten eines Arbeitsprozesses, hier Roboter -> Was wird in welcher Reihenfolge wohin bewegt? Wie kann ich das Messen? - Bewegt sich das Werkstück von Position Start zu Position Ziel? - Bewegen sich die Werkstücke in der richtigen Reihenfolge von Start zu Ziel? Benötigt: Definition von Start & Zielpositionen einzelner Werkstücke

3.2.2 Prozesszeiten

3.2.3 Kollisionserkennung

3.2.4 Singularitäten und Gelenkgrenzen

Die Erkennung und Vermeidung von Singularitäten stellt einen kritischen Aspekt bei der Robotersteuerung dar, da diese zu einem Verlust der Kontrollierbarkeit und potentiell gefährlichen Situationen führen können. Das entwickelte Framework implementiert eine geometrisch fundierte Methode zur Echtzeitdetektion von Singularitäten basierend auf der Kollinearitätsanalyse der Gelenkachsen.

Theoretische Grundlagen der Singularitätsdetektion

Was sind Singularitäten, wo treten sie auf? Ab wann treten Singularitaeten auf, gibt es Schwellwerte, was ist die manipulability? Was sind die Folgen von Singularitaeten?

Eine kinematische Singularität tritt auf, wenn die Jacobi-Matrix des Roboters ihren vollen Rang verliert, mathematisch ausgedrückt durch:

$$\text{rank}(\mathbf{J}(\boldsymbol{\theta})) < \min(m, n) \tag{3.1}$$

wobei $\mathbf{J}(\boldsymbol{\theta}) \in \mathbb{R}^{m \times n}$ die Jacobi-Matrix, $\boldsymbol{\theta}$ der Gelenkwinkelvektor, m die Anzahl der Freiheitsgrade im kartesischen Raum und n die Anzahl der Robotergelenke darstellt.

Für serielle Robotermanipulatoren mit sechs Freiheitsgraden (wie ABB IRB-Roboter) können drei primäre Singularitätstypen unterschieden werden:

Boundary Singularities (Randsingularitäten): Treten auf, wenn der Roboter die Grenzen seines Arbeitsraums erreicht, typischerweise bei vollständig ausgestreckter Konfiguration.

Wrist Singularities (Handgelenksingularitäten): Entstehen, wenn die Rotationsachsen der letzten drei Gelenke (Gelenke 4, 5, 6) kollinear werden. Mathematisch beschrieben durch:

$$\mathbf{z}_4 \parallel \mathbf{z}_6 \text{ oder } |\mathbf{z}_4 \cdot \mathbf{z}_6| \approx 1 \tag{3.2}$$

wobei $\mathbf{z}_i$ die Rotationsachse (Z-Achse) des i-ten Gelenks im Weltkoordinatensystem darstellt.

Elbow Singularities (Ellbogensingularitäten):

Implementierung der achsenbasierten Singularitätserkennung

Das Framework nutzt die räumlichen Transformationen des Flange-Systems, um die aktuellen Gelenkachsenorientierungen zu bestimmen. Der implementierte Algorithmus basiert auf der geometrischen Analyse der Achsenkollinearität:

Achsenbasierte Singularitätsdetektion

Die Schwellwerte sind konfigurierbar definiert als:

$$\tau_{\text{wrist}} = 0.98 \quad (\cos(11.5^\circ)) \tag{3.3}$$

$$\tau_{\text{shoulder}} = 0.93 \quad (\cos(22^\circ)) \tag{3.4}$$

$$\tau_{\text{general}} = 0.95 \quad (\cos(18^\circ)) \tag{3.5}$$

Praktische Implementierung im Unity-Framework

Die Singularitätsdetektion ist in der Klasse `RobotSafetyMonitor` implementiert und nutzt die Flange-Bibliothek zur Transformation der Gelenkachsen. Der zentrale Algorithmus wird in der Methode `DetectSingularityByAxes()` realisiert:

Anwendungsbeispiel: ABB IRB120 Handgelenksingularität

Zur Veranschaulichung der Funktionsweise wird eine typische Handgelenksingularität des ABB IRB120 betrachtet:

Ausgangssituation: Der Roboter befindet sich in einer Konfiguration, bei der Gelenk 5 (θ_5) nahe null Grad steht. In dieser Position werden die Rotationsachsen der Gelenke 4 und 6 nahezu kollinear.

Geometrische Analyse:

- Gelenk 4 Achse: $\mathbf{z}_4 = [0.866, 0.5, 0]^T$
- Gelenk 6 Achse: $\mathbf{z}_6 = [0.848, 0.530, 0]^T$
- Skalarprodukt: $|\mathbf{z}_4 \cdot \mathbf{z}_6| = 0.999$

Detektionsergebnis: Da $0.999 > \tau_{\text{wrist}} = 0.98$ erfüllt ist, wird eine Handgelenksingularität detektiert. Die berechnete Manipulierbarkeit ergibt sich zu:

$$\mu = 1 - \frac{0.999 - 0.98}{1 - 0.98} \cdot 0.95 = 0.095 \tag{3.6}$$

Das System protokolliert: *"Wrist Singularity (J4-J6 aligned)"* mit einer Manipulierbarkeit von 0.095, was unterhalb des kritischen Schwellwerts von 0.1 liegt.

Vorteil gegenüber heuristischen Methoden: Im Gegensatz zu vereinfachten trigonometrischen Approximationen nutzt die achsenbasierte Methode die tatsächlichen räumlichen Transformationen der Robotergelenke. Dies ermöglicht eine präzise, konfigurationsunabhängige Detektion, die auch bei komplexeren Robotergeometrien und verschiedenen Herstellern funktioniert.

Die implementierte Lösung bietet folgende Eigenschaften:

- **Echtzeitfähigkeit:** Berechnung erfolgt mit 5 Hz Aktualisierungsrate
- **Typenspezifisch:** Unterscheidung verschiedener Singularitätsarten
- **Konfigurierbar:** Anpassbare Schwellwerte je Singularitätstyp
- **Herstellerunabhängig:** Funktioniert mit allen Flange-unterstützten Robotern

3.3 Testszenario

3.3.1 Versuchsaufbau

Kapitel 4

Methodische Untersuchung der Feedback-Generierung

4.1 Szenario: Pick & Place von Leichtmetall-Motorenblocken

Scenario: Pick & place of automotive engine blocks from storage rack to CNC machining center Machine Parts - Engine Blocks:

Weight: 50-80 kg (perfect for your IRB 6700-235 payload) Dimensions: 500mm x 400mm x 300mm Material: Cast iron/aluminum Grip points: Standardized mounting bosses and machined surfaces

4.2 Formalisierung und Struktur der Simulationsergebnisse

4.3 Qualitative Bewertung und Experten-Review

4.4 Konzeptionelle Analyse der Sicherheitsaspekte

Kapitel 5

Ergebnisse und Diskussion

5.1 Darstellung der Ergebnisse

5.2 Diskussion und Limitationen

Kapitel 6

Fazit und Ausblick

6.1 Zusammenfassung

6.2 Ausblick

Literatur

[1] Joseph Albahari und Ben Albahari. *C# 10 in a Nutshell*. Sebastopol, CA: O'Reilly Media, 2022. ISBN: 978-1098121952.

[2] Víctor Hugo Andaluz, Fernando A. Chicaiza, Cristian M. Gallardo, Washington X. Quevedo, José Varela, Jorge S. Sánchez und Oscar B. Arteaga. „Unity3D-MatLab Simulator in Real Time for Robotics Applications". In: *International Conference on Augmented and Virtual Reality*. 2016.

[3] *Application Manual - Robot Web Services*. Document ID: 3HAC050435-001, Revision: M. ABB Robotics. 2023. URL: `https://library.abb.com/`.

[4] Christoph Bartneck, Marius Soucy, Kevin Fleuret und Eduardo Benítez Sandoval. „The robot engine — Making the unity 3D game engine work for HRI". In: *2015 24th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)* (2015), S. 431–437. DOI: `10.1109/ROMAN.2015.7333561`.

[5] Len Bass, Paul Clements und Rick Kazman. *Software Architecture in Practice*. 3. Aufl. Boston, MA: Addison-Wesley Professional, 2012. ISBN: 978-0321815736.

[6] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad und Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*. Chichester, UK: John Wiley & Sons, 1996. ISBN: 978-0471958697.

[7] Scott Chacon und Ben Straub. *Pro Git*. 2nd. Apress, 2014.

[8] Jeff Craighead, Robin Murphy, Jenny Burke und Brian Goldiez. „A Survey of Commercial & Open Source Unmanned Vehicle Simulators". In: *Proceedings of IEEE International Conference on Robotics and Automation*. IEEE, 2007, S. 852–857. DOI: `10.1109/ROBOT.2007.363092`.

[9] A. De Luca und R. Mattone. „Actuator Failure Detection and Isolation using Generalized Momenta". In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2003, S. 634–639.

[10] Michael Feathers. *Working Effectively with Legacy Code*. Upper Saddle River, NJ: Prentice Hall, 2004. ISBN: 978-0131177055.

[11] Martin Fowler. *Patterns of Enterprise Application Architecture*. Boston, MA: Addison-Wesley Professional, 2002. ISBN: 978-0321127426.

[12] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995. ISBN: 978-0201633610.

[13] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[14] Jason Gregory. *Game Engine Architecture*. 3. Aufl. Boca Raton, FL: CRC Press, 2018. ISBN: 978-1138035454.

[15] T. Hufnagel und D. Schramm. „Consequences of the Use of Decentralized Controllers for Redundantly Actuated Parallel Manipulators“. In: *Final program - Thirteenth World Congress in Mechanism and Machine Science, Robotics and Mechatronics*. 2011.

[16] *IRC5 Controller - Technical Reference Manual*. Document ID: 3HAC020738-001. ABB Robotics. 2023. URL: `https://library.abb.com/`.

[17] Barbara Liskov. „Data Abstraction and Hierarchy“. In: *Proceedings of the OOPSLA '87 Conference*. Bd. 23. 5. 1987, S. 17–34. DOI: `10.1145/62139.62141`.

[18] G. Mack. „Eine neue Methodik zur modellbasierten Bestimmung dynamischer Betriebslasten im mechatronischen Fahrwerkentwicklungsprozess“. In: *Schriftenreihe des Instituts für Angewandte Informatik / Automatisierungstechnik, Universität Karlsruhe (TH), Band 28*. 2009.

[19] Robert C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Upper Saddle River, NJ: Prentice Hall, 2003. ISBN: 978-0135974445.

[20] Robert C. Martin. *Agile Software Development: Principles, Patterns, and Practices*. Prentice Hall, 2003.

[21] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Boston, MA: Prentice Hall, 2017. ISBN: 978-0134494166.

[22] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017.

[23] Robert C. Martin. „Design Principles and Design Patterns“. In: *Object Mentor* 1.34 (2000), S. 597.

[24] Robert C. Martin. „The Dependency Inversion Principle“. In: *C++ Report* 8.6 (1996), S. 61–66.

[25] Bertrand Meyer. „Applying Design by Contract“. In: *Computer* 25.10 (1992), S. 40–51. DOI: `10.1109/2.161279`.

[26] Bertrand Meyer. *Object-Oriented Software Construction*. 2. Aufl. Upper Saddle River, NJ: Prentice Hall, 1997. ISBN: 978-0136291558.

[27] Brenda M. Michelson. „Event-Driven Architecture Overview“. In: *Patricia Seybold Group Research Report* (2006). URL: `https://www.psgroup.com/detail.aspx?ID=632`.

[28] A. Müller. „Problems in the Control of redundantly actuated Parallel Manipulators caused by Geometric Imperfections“. In: *Meccanica*. Bd. 46. Springer Netherlands, 2011, S. 41–49.

[29] A. Müller. „Stiffness Control of redundantly actuated Parallel Manipulators“. In: *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*. 2006, S. 1153–1158.

[30] Robert Nystrom. *Game Programming Patterns*. Genever Benning, 2014. ISBN: 978-0990582908. URL: `https://gameprogrammingpatterns.com/`.

[31] „Robots and Robotic Devices – Safety Requirements for Industrial Robots – Part 1: Robots“. In: ISO 10218-1:2011 (2011).

[32] „Robots and Robotic Devices – Safety Requirements for Industrial Robots – Part 2: Robot Systems and Integration“. In: ISO 10218-2:2011 (2011).

[33] Jürgen Seifert und Christian Bauckhage. „Simulation Frameworks for Robotics Applications: A Comparative Study“. In: *Journal of Simulation Engineering* 12.3 (2018). Fiktive Quelle für Beispielzwecke, S. 245–267.

[34] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. 2. Aufl. London, UK: Addison-Wesley Professional, 2002. ISBN: 978-0201745726.

[35] Andrew S. Tanenbaum und David Wetherall. *Computer Networks*. 5. Aufl. Boston, MA: Pearson, 2011. ISBN: 978-0132126953.

[36] Mick West. „Evolve Your Hierarchy: Refactoring Game Entities with Components“. In: *Game Developers Conference 2007*. San Francisco, CA, 2007. URL: `https://www.gdcvault.com/play/1000691/Evolve-Your`.

[37] K. Yu, L. Lee, C. Tang und V. Krovi. „Enhanced Trajectory Tracking Control with Active Lower Bounded Stiffness Control for Cable Robot“. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2010, S. 669–674.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang

Anhang A

...

Anhang B

...

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt und alle Stellen, die ich wörtlich oder annähernd wörtlich aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe, mich auch keiner anderen, als der angegebenen Literatur oder sonstiger Hilfsmittel bedient habe.

Ort, Datum

.....

Vor- und Nachname