Der Digitale Zwilling als Mediator zwischen Mensch und Maschine

Von der Fakultät für Elektrotechnik und Informationstechnik der
Rheinisch-Westfälischen Technischen Hochschule Aachen
zur Erlangung des akademischen Grades eines Doktors der
Ingenieurwissenschaften genehmigte Dissertation

vorgelegt von

Master of Science Torben Cichon
aus Wuppertal

Berichter: Universitätsprofessor Dr.-Ing. Jürgen Roßmann
Universitätsprofessor Dr.-Ing. Jochen Deuse

Tag der mündlichen Prüfung: 7. November 2019

Diese Dissertation ist auf den Internetseiten der Universitätsbibliothek online verfügbar.

Kurzfassung

Immer komplexer werdende mechatronische Gesamtsysteme erschweren es dem Menschen, einen einfachen Zugang zu und ein intuitives Verständnis von diesen inhärent nicht-trivialen Systemen zu erhalten. Deren Komplexität bezieht sich sowohl auf die Struktur des Systems, als auch auf den Entwicklungsprozess und zudem auf die Steuerung/Bedienung des finalen Systems. Unzulänglichkeiten bei der (Nutzer-)Interaktion, Kollaboration und Kooperation sowie der Steuerung, Teleoperation und Autonomie zeigen klar die Grenzen bei der Entwicklung und Nutzung aktueller robotischer Systeme auf. Das heißt, komplexe Systeme von der Grundlagenforschung bis zum Endkunden/Nutzer zu begleiten (*entwickeln*, *überwachen*, *warten* und *betreiben*) ist Gegenstand aktueller Bemühungen in Forschung und Entwicklung. Gerade die Übergänge von *(universitärer) Forschung zu (industrieller) Anwendung* sowie von *Entwickler zu Nutzer* stellen zentrale Herausforderungen dar. Zur Bewältigung ist eine allgemeine Lösung notwendig, die durch innovative Möglichkeiten der Interaktion einen intuitiven Zugang erlaubt und dadurch Entwicklung, Steuerung sowie inhärent sichere Kooperation und Kollaboration mit komplexen Systemen über deren gesamten Lebenszyklus hinweg möglich macht.

Als einer der zentralen Konzepte von Industrie 4.0 zeigt der Digitale Zwilling – die virtuelle digitale Repräsentanz physischer Assets [a] – vielfältige Einsatzmöglichkeiten, insbesondere im Maschinenbau und in der Produktionstechnik. Der Digitale Zwilling, der sämtliche Details vom (mechatronischen) System (Montage, Funktionen, Verwendung und Nutzung) enthält, kann über den gesamten Produktzyklus in unterschiedlichen Ausprägungen Anwendung finden, wie z. B. für optimiertes Maschinendesign, effizientere Inbetriebnahme, kurze Umrüstzeiten, fehlerfreier Betrieb sowie Einblick, Verständnis, Prädiktion und Optimierung der Maschinenleistung. Als Vision kann der Digitale Zwilling nicht nur als Entwicklungstool und Statusmonitor, sondern darüber hinaus als *zentrale Schnittstellentechnologie* zur intuitiven und sicheren, direkten und indirekten Interaktion und Kooperation zwischen Mensch und realem System eingesetzt werden.

In der vorliegenden Dissertation wird daher ein Konzept ausgearbeitet, *den Digitalen Zwilling zum Mediator zwischen Mensch und Maschine* zu etablieren. Der Digitale Zwilling steht im Mittelpunkt, um einen intuitiven Zugang zu dem realen komplexen robotischen System zu ermöglichen. Mit Hilfe eines Virtuellen Testbeds (VTB) – einem Software-Framework zur system-, disziplin- und anwendungsübergreifenden Entwicklung komplexer Systeme – steht eine Laufzeitumgebung für solche Digitale Zwillinge bereit,

um praxisnahe Problemstellungen zu realisieren und neue Ideen, Methoden und Technologien zur Lösung zu entwickeln und zu testen. Die vorliegende Arbeit stellt darauf aufbauend eine konzeptionelle Weiterentwicklung solcher VTBs dar, indem mit und durch den Digitalen Zwilling ein VTB zur Interaktion und Kooperation zwischen Mensch und Maschine lebenszyklusübergreifend eingesetzt werden kann. Neben der Entwicklung des Digitalen Zwillings und dessen Einbettung in das VTB steht insbesondere die Interaktion zwischen Mensch, Digitalem Zwilling und physischem Asset (realem Roboter) im Fokus. Der Digitale Zwilling wird hierbei zum *Mediator* zwischen Mensch und Roboter und nutzt die Fusion von Realität und Virtualität, um eine intuitive und sichere Interaktion des Menschen mit dem Roboter zu ermöglichen. Im Kern der Dissertation werden die drei hierfür notwendigen Konzeptbausteine entwickelt:

(a) die Entwicklung und Integration des Digitalen Zwillings in das VTB sowie die Erweiterung der zugrunde liegenden 3D-Simulation um spezialisierte Frameworks zur Nutzbarmachung und Kopplung der Möglichkeiten des Digitalen Zwillings und des VTBs;
(b) die Anbindung neuartiger Ein- und Ausgabegeräte an das VTB sowie die visuelle Aufbereitung eingehender Daten, um dem Menschen einen intuitiven und direkten Zugang zum Digitalen Zwilling zu verschaffen;
(c) die Implementierung einer Schnittstelle zwischen Digitalem Zwilling und realem Roboter zur standardisierten Kommunikation des VTBs mit externen Hard- und Softwaremodulen.

Dank dieser Bausteine lässt sich der Digitale Zwilling zunächst offline in der Entwicklung des Systems, von datenverarbeitenden Modulen sowie von Nutzerschnittstellen und dem Operator-Training nutzen, anschließend aber auch online zur Überwachung, Interaktion und direkten Steuerung des realen robotischen Systems. Durch die Virtuelle Realität kann sich der Nutzer virtuell an den Einsatzort begeben und wird in seinen Entscheidungen durch sinnvoll aufbereitete Visualisierung wichtiger interner Parameter und möglicher Handlungsoptionen unterstützt. Die Modularität und Flexibilität des Frameworks ermöglicht außerdem die Fusion oder das Wechseln von Realität und Virtualität, sodass das reale System und der Digitale Zwilling sowohl als Datenquelle als auch -senke für das VTB und andere weiterverarbeitende Module eingesetzt werden können. Mögliche Aktionen können dadurch sowohl vor als auch während einer Mission zunächst am Digitalen Zwilling getestet werden, bevor man diese in der Realität umsetzt.

Inhaltsverzeichnis

Verzeichnisse **viii**

Abbildungen . . . viii

Tabellen . . . xii

Akronyme **xiv**

Nomenklatur **xviii**

1 Motivation und Einleitung **1**

2 Stand der Forschung **5**

2.1 Abgrenzung der Arbeits- und Forschungsschwerpunkte . . . 6

2.1.1 Forschungstrends und -ideen: Industrie 4.0, Internet of Things, cyber-physische Systeme . . . 6

2.1.2 Der Lebenszyklus von Maschinen . . . 8

2.1.3 Komplexität in der Entwicklung und Nutzung von Robotern . . . 9

2.1.4 Mediatoren . . . 9

2.2 Assistenzsysteme . . . 11

2.2.1 Assistenz und Arbeitsteilung in der Industrie 4.0 . . . 12

2.2.2 Motorische/physische Assistenz . . . 12

2.2.3 Perzeptions-/Wahrnehmungsassistenz . . . 13

2.2.4 Kognitive Assistenz . . . 13

2.2.5 Assistenzsysteme im Automotive-Bereich . . . 14

2.3 Mediatoren zwischen Mensch und Maschine/Computer . . . 16

2.3.1 Benutzerschnittstelle . . . 16

2.3.2 Natürliche, intuitive und sichere Mensch-Maschine-Interaktion . . . 17

2.3.3 User Interfaces: systemtheoretischer I/O-Ansatz . . . 20

2.3.4 Eingabegeräte . . . 20

2.3.5 Ausgabegeräte . . . 30

2.4 Mediatoren in der Robotik . . . 37

2.4.1 Interaktion, Kollaboration und Kooperation . . . 37

2.4.2 Industrielle Kollaboration von Mensch und Roboter . . . 38

2.4.3 Kooperation von Mensch und mobilem Roboter . . . 41

2.4.4 Hard- und Softwareschnittstellen in der Industrie 4.0 . . . 47

2.5 Von 3D-Simulationssystemen zu virtuellen Experimentierumgebungen für Digitale Zwillinge . . . 52

2.5.1 Simulationssysteme zur Mensch-Roboter-Kooperation . . . 52

2.5.2 Der Digitale Zwilling (DZ)/Digital Twin (DT) . . . 75
2.5.3 Das Virtuelle Testbed (VTB) . . . 80
2.6 Idee und Vision von „Seeing Spaces“ . . . 90
2.7 Diskussion des Forschungsstandes . . . 92

3 Konzept **93**
3.1 Das Gesamtsystem aus Nutzer, Digitalem Zwilling und physischem Asset 95
3.2 Notwendige Bestandteile des Virtuellen Testbeds als Mediator . . . 97
3.2.1 Schnittstelle zwischen Nutzer und Digitalem Zwilling . . . 97
3.2.2 Der Digitale Zwilling im Virtuellen Testbed . . . 98
3.2.3 Schnittstellen zwischen Digitalem Zwilling und physischem Asset . 98
3.3 Anforderungsanalyse . . . 99
3.4 Konzeptionelle Funktionalitäten des Digitalen Zwillings . . . 101
3.4.1 Der Digitale Zwilling als Entwicklungswerkzeug . . . 101
3.4.2 Der Digitale Zwilling als Eingabegerät . . . 102
3.4.3 Der Digitale Zwilling als Schnittstelle . . . 103
3.4.4 Der Digitale Zwilling als Abstraktionsschicht für den Nutzer . . . 104

4 Referenzimplementierung **106**
4.1 Beschreibung des Implementierungsvorhabens . . . 106
4.2 Entwicklung intuitiver Nutzerschnittstellen . . . 108
4.2.1 Mensch-Maschine-Schnittstelle der 3D-Simulation . . . 108
4.2.2 Erweiterung klassischer Eingabegeräte . . . 109
4.2.3 3D-, Positions-, Tracking- und Kamera-basierte Eingabegeräte . . . 111
4.2.4 Peripheriegeräte mit Kraftrückkopplung . . . 116
4.2.5 Parametervisualisierung . . . 121
4.2.6 Immersion und Visualisierungen im Gesamtsystem . . . 131
4.3 Einbettung des Digitalen Zwillings ins Virtuelle Testbed . . . 133
4.3.1 Automatisierte Generierung Digitaler Zwillinge . . . 134
4.3.2 Standardisierung in Soft- und Hardware für die 3D-Simulation . . . 145
4.3.3 Integration notwendiger Funktionalitäten in die 3D-Simulation . . . 152
4.4 Implementierung standardisierter Hardwareschnittstellen . . . 162
4.4.1 Interface zum Robot Operating System (ROS) . . . 162
4.4.2 Der Digitale Zwilling als universelles Interface . . . 171
4.5 Kopplung von Digitalem und Realem Zwilling zum Mediator . . . 176
4.5.1 Erweiterte Anforderungen . . . 176
4.5.2 Echtzeitfähige Starrkörpersimulation . . . 177
4.5.3 Daten-Haltung, -Abgleich und -Austausch . . . 177
4.5.4 Der Mensch als Teil des Systems . . . 178
4.5.5 Flexibles, modulares, nutzerzentriertes Gesamtsystem aus Nutzer, Digitalem Zwilling und Roboter . . . 179

5 Anwendungen **180**

5.1 Teleoperative Steuerung von Robotern in Katastrophenszenarien 182
5.1.1 Modularität in der Entwicklung mobiler Roboter 184
5.1.2 Teleoperation, Telepräsenz und Immersion durch den Digitalen Zwilling 186
5.1.3 Der Digitale Zwilling „in-the-loop“ zur direkten Interaktion und Kontrolle 188
5.1.4 Prädiktion durch den Digitalen Zwilling 194

5.2 Mensch-Roboter-Kooperation im industriellen Anlagenbau 196
5.2.1 Rekonfigurierbare Roboterzelle zur Montage 196
5.2.2 Das Virtuelle Testbed mit VR/AR in der Produktion 197
5.2.3 Direkte, intuitive und sichere Steuerung „softer“ Roboter 198

5.3 Weitere Anwendungen des Digitalen Zwillings im Rahmen von Industrie 4.0 200
5.3.1 Assistenzsysteme zur Entwicklung und Nutzung von Digitalen Zwillingen 200
5.3.2 Das Holodeck: Eine immersive, kooperative Entwicklungs- und Interaktionsplattform 201
5.3.3 Weltraumanwendungen: Konstruktions- und Manöverplanung . . 204
5.3.4 Engineering 4.0: Erweiterung des Büros um die Dimension der Virtualität 206

5.4 Anforderungsrückblick 207

6 Zusammenfassung und Ausblick **210**

7 Anhang **215**

Verzeichnisse

Abbildungen

1.1 Thema dieser Arbeit: der Digitale Zwilling als Mediator 3

2.1 Industrie 4.0 (I4.0): hoch komplexe, heterogene, vernetzte Gesamtsysteme auf Basis von CPS mit dem Menschen im Mittelpunkt 6
2.2 Gartner Hype Cycle 2017 8
2.3 Die drei Determinanten zur Klassifikation von Assistenzsystemen 12
2.4 Modell der menschlichen Informationsverarbeitung 13
2.5 Fahrerassistenzsysteme (FAS) 15
2.6 Bidirektionale Interaktion von Mensch und Maschine 16
2.7 Fragebogen Intuitivität (QUESI) 18
2.8 Virtual Reality (VR): Komplexität, Präsenz und Interaktion 20
2.9 Controllerbasierte Eingabegeräte: eine Übersicht 21
2.10 Thalmic Labs Myo-Armband 21
2.11 Optische Eingabegeräte: eine Übersicht 22
2.12 Kamera bzw. Bildtypen für Systeme mit externer Sensorik 23
2.13 Leap Motion Controller 23
2.14 Haptische Eingabegeräte: eine Übersicht 25
2.15 Force Feedback Interface: virtueller Proxy 27
2.16 Virtuelle Eingabegeräte: eine Übersicht 28
2.17 Integration von Gestenerkennung und Gestensteuerung 29
2.18 Überschneidung von digitalen/virtuellen (V) und realen (R) Inhalten . . 30
2.19 HUDs und HPs 34
2.20 Projektion von Informationen in eine 3D-Szene 35
2.21 MRK im industriellen Kontext 39
2.22 Nutzerschnittstellen verschiedener Mobilitätsklassen 43
2.23 Steuerung quadrupedaler mobiler Roboter 44
2.24 Funktionsprinzip der ROS-Messages 49
2.25 Systemtheorie – die zentralen Bestandteile 53
2.26 Komplexe Systeme 54
2.27 Projektprozesse – eine Übersicht 56
2.28 Einsatz von Simulation 57
2.29 Lebenszyklus robotischer Systeme 58
2.30 SysML für Hard- und Software im Bereich mobiler Roboter 58
2.31 Von „sense-think-cct“ (the robotic paradigm) zu „sense-simulate-act“ . . 60
2.32 VEROSIM-Mikrokernel-Architektur 62
2.33 Nutzerschnittstelle der 3D-Simulationsumgebung 63

2.34 Denavit-Hartenberg-Parameter . . . 65
2.35 Trägheitsellipsoid . . . 68
2.36 Validierungsmöglichkeiten von Modellen . . . 72
2.37 Evaluierung von Robotern . . . 74
2.38 Der Digitale Zwilling: Mediator zur Entwicklung und Führung von Forstmaschinen . . . 76
2.39 Der Digitale Zwilling: Abbildung eines physischen Assets . . . 78
2.40 Von reiner Visualisierung zu Virtuellen Testbeds (VTBs) . . . 80
2.41 Digitaler Zwilling: Einbettung in das VTB am Beispiel eines mobiles Roboters . . . 82
2.42 VTB: Netzwerk interagierender DTs – Automotive . . . 83
2.43 Entwicklung von Systemen mit Hilfe von VTBs . . . 84
2.44 Das VTB: Frameworks, Anwendungen und Nutzen . . . 86
2.45 Knoten (Nodes), Erweiterungen (Extensions) und deren Verbindungen (I/O Board) . . . 88
2.46 „Seeing Spaces“ von Bret Victor . . . 90

3.1 Aus dem Entwicklungsprozess in den Betrieb . . . 93
3.2 Mit dem DT aus dem Entwicklungsprozess in die Anwendung . . . 94
3.3 Konzeptionelle Weiterentwicklung der VTBs mit Hilfe Digitaler Zwillinge 94
3.4 SysML-Graphik des Konzepts . . . 95
3.5 Simplifizierte Darstellung der zentralen Bestandteile des Konzeptes . . . 97
3.6 Virtuelle Modulare Integration von Systemkomponenten . . . 101
3.7 Das VTB als Eingabegerät . . . 103
3.8 Das VTB als Schnittstelle . . . 103
3.9 Das VTB als Abstraktionsschicht für den Nutzer . . . 104

4.1 Piktogramme der Konzeptbestandteile: die drei System-Layer . . . 106
4.2 Piktogramme der Konzeptbestandteile: die drei VTB-Sub-Layer . . . 107
4.3 Piktogramme der Konzeptbestandteile des 3D-Simulationskerns: DT und Datenverarbeitung . . . 107
4.4 Konzept: Nutzerschnittstellen . . . 108
4.5 Tastaturerweiterung: Key Toggler . . . 110
4.6 Integrationskonzept, exemplarisch für das Leap Motion . . . 111
4.7 Microsoft Kinect v2: Sensorintegration . . . 112
4.8 Leap Motion Controller: Sensorintegration . . . 113
4.9 Thalmic Labs Myo-Armband: Sensorintegration . . . 114
4.10 HTC Vive Controller: Integration . . . 115
4.11 Virtuelle Eingabegeräte: Integration . . . 115
4.12 Force Feedback Interface: Konzeptskizze . . . 117
4.13 Sensable Geomagic Touch X: Aufbau . . . 119
4.14 Sensable Geomagic Touch X: physikalischer und nutzbarer Arbeitsraum . 119
4.15 Force-Feedback-Interface: Analyse des PHANToM Device . . . 120
4.16 Exoskelett: ALEx . . . 120
4.17 Exoskelett: UDP 1 kHz-Loop von Arm und Hand Exoskelett . . . 121

4.18 Visualisierung: Live-Visualisierung diverser Rohdaten . . . 122
4.19 Visualisierung: Tracer und Laserscannerebene . . . 123
4.20 Visualisierung: höhencodierte Punktwolkenvisualisierung . . . 124
4.21 Visualisierung: Onlineinspektion von ROS-Nachrichten . . . 124
4.22 Visualisierung: Flexibles HUD . . . 125
4.23 Visualisierung: interaktiver Marker (Gizmo) . . . 126
4.24 Visualisierung: Arbeitsraum und Kontaktkräfte . . . 127
4.25 Visualisierung: Billboards . . . 128
4.26 Visualisierung: Höhenkarte . . . 128
4.27 Visualisierung: abrastern einer Punktwolke . . . 129
4.28 Visualisierung: texturiertes Polygonnetz . . . 130
4.29 Visualisierung: verschiedene Kameraperspektiven . . . 132
4.30 Konzept: Einbettung des Digitalen Zwillings . . . 133
4.31 Konzept: Integration der Digitalen Zwillinge . . . 134
4.32 Automatische DT-Generierung . . . 135
4.33 CAD-Import eines Kaffeebechers mit Oct-Tree-Sphere-Dynamisierung . . . 136
4.34 URDF-Import: Modelltypen Digitaler Zwillinge . . . 137
4.35 URDF-Import: automatische Generierung von Robotern . . . 139
4.36 Repository Digitaler Zwillinge (DT-Repo): Auswahl an Objekten . . . 142
4.37 DT-Repo: Variation verschiedener Systeme und Umgebungen . . . 142
4.38 Disaster-Szenarien nach realem Vorbild . . . 142
4.39 Generierung einer 2.5D-Höhenkarte . . . 144
4.40 Voxelisierung von Punktwolken . . . 144
4.41 Physik-Benchmarking: Energie- und Impulserhaltung . . . 145
4.42 MATLAB: Konsistenzanalyse eines Lineargelenks . . . 149
4.43 Zusammengesetzte Starrkörper . . . 150
4.44 Starrkörpercluster-Visualisierung . . . 151
4.45 Physik zum Anfassen: schiefer Wurf, Kugeltrajektorien, Feder-Dämpfer-System . . . 151
4.46 Physik zum Anfassen: das Pendel . . . 152
4.47 Konzept: interne Datenverarbeitung . . . 152
4.48 Motorradsimulation: Motorrad BMW K1600 GT . . . 154
4.49 Motorradsimulation: Kurvenfahrt . . . 155
4.50 Automobilsimulation . . . 156
4.51 Automobilsimulation: Rennstrecke . . . 157
4.52 Simulation des Bionic Handling Assistant von Festo . . . 158
4.53 Tetraeder als diskretes Volumenteil eines deformierbaren Körpers . . . 159
4.54 Deformierbare Körper: Narrow Phase . . . 160
4.55 Deformierbare Körper: Schema der Kollisionskraftberechnung . . . 160
4.56 Deformierbare Körper: zeitlicher Ablauf einer Kollision . . . 161
4.57 Deformierbare Körper: beliebige deformierbare Strukturen . . . 161
4.58 Konzept: Hardwareschnittstellen . . . 162
4.59 Bestandteile der ROS-Schnittstelle . . . 164
4.60 ROS-Schnittstelle: Plugin-System . . . 165
4.61 ROS-Schnittstelle: Lib- und Plugin-System . . . 166

4.62 UML-Diagramm der ROS-Schnittstelle . . . 167
4.63 Modulare Erweiterung des ROS-Schnittstellen-Konzeptes . . . 168
4.64 ROS-Schnittellenerweiterung . . . 169
4.65 ROS2-Integrationskonzepte . . . 170
4.66 Das Modell des Digitalen Zwillings unterschiedlicher Komplexität . . . 171
4.67 UDP-Schnittstelle . . . 174
4.68 XBotCore-Schnittstelle . . . 175
4.69 Konzept: Live Support . . . 176
4.70 Konzept: Der Mensch als Teil des Systems . . . 178

5.1 Anwendbarkeit des Konzeptes . . . 180
5.2 Das CENTAURO-Projekt . . . 182
5.3 CENTAURO: modulare Zusammenstellung des Konzeptes . . . 183
5.4 CENTAURO: Digitale Zwillinge von Robotern . . . 184
5.5 CENTAURO: VEROSIM CENTAURO-Roboter . . . 185
5.6 CENTAURO: Schunk-Hand . . . 185
5.7 CENTAURO: finaler CENTAURO-Roboter . . . 186
5.8 CENTAURO: Exoskelett – virtuelle Benchmarks . . . 186
5.9 CENTAURO: Exoskelett – Telemanipulation . . . 187
5.10 VTB: Netzwerk interagierender DTs – mobile Robotik . . . 188
5.11 CENTAURO: CENTAURO-Roboter visualisiert im VTB . . . 189
5.12 CENTAURO: Gesamtsystem Evaluationsszenario . . . 190
5.13 CENTAURO: texturiertes Polygonnetz – Kinect . . . 190
5.14 CENTAURO: texturiertes Polygonnetz – Velodyne-Laserscanner . . . 191
5.15 CENTAURO: Kontrollstrategien . . . 191
5.16 CENTAURO: semi-autonome Lokomotion durch den Support Operator . . . 192
5.17 CENTAURO: Nutzungsmodi des Digitalen Zwillings . . . 193
5.18 CENTAURO: „Switch“ in den DT beim Gap-Task . . . 194
5.19 Der Digitale Zwilling in der Industrierobotik . . . 196
5.20 MRK: rekonfigurierbare Roboterzelle . . . 197
5.21 MRK: Peg-in-Hole-Experiment . . . 199
5.22 Haptische Assistenzsysteme . . . 200
5.23 Haptische Gestenerkennung . . . 201
5.24 Holodeck: Trackingmodalitäten . . . 201
5.25 Holodeck: Interaktionsframework . . . 202
5.26 Holodeck: Interaktionmodalitäten, Selektion und Datenbank . . . 203
5.27 Holodeck: Anwendung Haushaltsrobotik (Neato Botvac 85D) . . . 203
5.28 Weltraumanwendungen: ein Überblick . . . 204
5.29 Weltraumanwendungen: Satellitenservicing . . . 205
5.30 Weltraumanwendungen: CENTAURO-Roboter als Explorationsroboter . . . 205
5.31 Engineering 4.0 mobiler Roboter . . . 206

7.1 SysML-Konzeptgraphik: Blockdefinitionsdiagramm (BDD) . . . 216
7.2 SysML-Konzeptgraphik: Internes Blockdiagramm (IBD) (hohes Detaillevel) 217
7.3 SysML-Konzeptgraphik: IBD (mittleres Detaillevel) . . . 218

7.4 SysML-Konzeptgraphik: IBD (geringes Detaillevel) . . . 218
7.5 Thalmic Labs Myo-Armband: EMG-Signale . . . 219
7.6 Thalmic Labs Myo-Armband: Experiment „Pattern Matching“ . . . 221
7.7 Sensable Geomagic Touch X: kinematische Kette des PHANToM Omni . 222
7.8 Sensable Geomagic Touch X: kinematischer Digitaler Zwilling . . . 223
7.9 Sensable Geomagic Touch X: inverse Kinematik . . . 223
7.10 Virtual Reality (VR): Oculus Rift . . . 226
7.11 Virtual Reality (VR): HTC Vive . . . 226
7.12 Assisted Reality (AsR): Vuzix M300 . . . 227
7.13 Augmented Reality (AR): Epson Moverio BT 300 . . . 227
7.14 Mixed Reality (MR): Microsoft HoloLens . . . 228

Tabellen

2.1 Vergleich von Microsoft Kinect v1 und v2 . . . 24

4.1 Jacobi-Parameter zur MATLAB-Evaluation . . . 147
4.2 Relevante physikalische Parameter zur Simulation eines Zweirads . . . 155
4.3 ROS-Schnittstelle: Bewertung von Python-Integrationskonzepten . . . 168

7.1 Myo-Experiment 1: Fehlerrate der Gestenerkennung . . . 219
7.2 Myo-Experiment 2: Geschwindigkeitsabhängigkeit der Fehlerrate . . . 220
7.3 Datenblatt: Sensable Geomagic Touch X . . . 224
7.4 Exoskelett Device Command Struct (EDCS) . . . 225
7.5 Exoskelett Device Data Struct (EDDS) . . . 225
7.6 Datenblatt: Oculus Rift . . . 226
7.7 Datenblatt: HTC Vive . . . 226
7.8 Datenblatt: Vuzix M300 . . . 227
7.9 Datenblatt: Epson Moverio . . . 227
7.10 Datenblatt: Microsoft HoloLens . . . 228

Akronyme

3DUI	3D User Interface
API	Application Programming Interface
AR	Augmented Reality
AsR	Assisted Reality
AsV	Assisted Virtuality
AV	Augmented Virtuality
BDD	Blockdefinitionsdiagramm
BIP	Break in Presence
CPS	Cyber-physisches System
CWM	Central World Model
DDS	Data Distribution Service
DH	Denavit-Hartenberg
Dll	Dynamic Linked Library
DoF	Degree of Freedom
DPS	Data Processing System
DRC	DARPA Robotics Challenge
DT	Digital Twin
DZ	Digitaler Zwilling
EMG	Elektromyogramm
FAS	Fahrerassistenzsysteme
FCV	Free Camera View
FEM	Finite-Elemente-Methode
FF	Force Feedback

FT	Force Torque
FMI	Functional Mockup Interface
FMU	Functional Mockup Unit
FoV	Field of View
FPS	Frames per Second
FPV	First Person View
FSM	Finite State Machine
GUI	Graphical User Interface
HCI	Human-Computer Interaction
HCL	Hardware Communication Layer
HIL	Hardware in-the-loop
HMD	Head-Mounted Display
HMI	Human-Machine Interaction
HMM	Hidden Markov Model
HP	Holographische Projektion
HRC	Human-Robot Collaboration
HUD	Head-Up-Display
I4.0	Industrie 4.0
IBD	Internes Blockdiagramm
IMU	Inertial Mess Einheit
IoT	Internet of Things
KI	Künstliche Intelligenz
M2M	Machine-to-Machine
MBSE	Model-based Systems Engineering
MCI	Mensch-Computer-Interaktion
MDL	Message Definition Language
MMI	Mensch-Maschine-Interaktion
MMK	Mensch-Maschine-Kommunikation
MR	Mixed Reality

MRK	Mensch-Roboter-Kooperation
QOS	Quality of Service
RGB	Rot Grün Blau
ROS	Robot Operating System
RT	Real Twin
RZ	Realer Zwilling
SDK	Software Development Kit
SBC	Simulation-based Control
SBOI	Simulation-based Operator Interfaces
SBS	Simulation-based Support
SBSE	Simulation-based Systems Engineering
SBUI	Simulation-based User Interfaces
SBX	Simulation-based X
SoE	Sense of Embodiment
SWIG	Simplified Wrapper and Interface Generator
SysML	Systems Modeling Language
TCP	Tool Center Point
TOF	Time of Flight
TPV	Third Person View
UCL	User Communication Layer
UDP	User Datagram Protocol
UI	User Interfaces
UML	Unified Modeling Language
URDF	Unified Robot Description Format
VMI	Virtuelle Modulare Integration
VR	Virtual Reality
VRC	Virtual Robotics Challenge
VEROSIM	Virtual Environment and Robotics Simulation System
VSD	VEROSIM Active Simulation Database

VTB Virtuelles Testbed

XML Extensible Markup Language

Nomenklatur

Die in dieser Arbeit verwendete Notation mathematischer Symbole und die verwendeten Formelzeichen sind im Folgenden zusammengefasst.

$s \in \mathbb{R}$	Allgemeiner Skalar
$\boldsymbol{b} \in \mathbb{R}^n$	Allgemeiner Vektor der Dimension $n \in \mathbb{N}$
$b_i \in \mathbb{R}$	i-te Komponente des Vektors $\boldsymbol{b} \in \mathbb{R}^n$ mit $0 \leq i < n \in \mathbb{N}$
$\boldsymbol{b}(t) \in \mathbb{R}^n$	Allgemeiner Vektor als Funktion der Zeit $t \in \mathbb{R}$
$\dot{\boldsymbol{b}} = \frac{\mathrm{d}}{\mathrm{d}t}\boldsymbol{b}(t) \in \mathbb{R}^n$	1. Ableitung eines allgemeinen Vektors nach der Zeit
$\ddot{\boldsymbol{b}} = \frac{\mathrm{d}^2}{\mathrm{d}t^2}\boldsymbol{b}(t) \in \mathbb{R}^n$	2. Ableitung eines allgemeinen Vektors nach der Zeit
$\mathbf{A} \in \mathbb{R}^{n \times m}$	Allgemeine Matrix mit $n \in \mathbb{N}$ Zeilen und $m \in \mathbb{N}$ Spalten
$\mathbf{A}^\mathsf{T} \in \mathbb{R}^{m \times n}$	Transponierte einer Matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$
$\mathbf{A}^{-1} \in \mathbb{R}^{m \times m}$	Inverse einer quadratischen Matrix $\mathbf{A} \in \mathbb{R}^{m \times m}$
$\mathbf{A}^{+} \in \mathbb{R}^{m \times n}$	Pseudoinverse einer beliebigen Matrix $\mathbf{A} \in \mathbb{R}^{n \times m}$
$\boldsymbol{q} \in \mathbb{R}^n$	Gelenkwertvektor über $n \in \mathbb{N}$ Gelenke
$\boldsymbol{p} \in \mathbb{R}^3$	Kartesischer Positionsvektor
$\boldsymbol{l} \in \mathbb{R}^6$	Kartesischer Lagevektor, gegeben durch $\boldsymbol{l} = (\boldsymbol{p}, \phi_x, \phi_y, \phi_z)'$
$\mathbf{J} \in \mathbb{R}^{6 \times n}$	Jacobi-Matrix, definiert durch $\dot{\boldsymbol{l}} = \mathbf{J} \cdot \dot{\boldsymbol{q}}$
$\alpha, \beta, \gamma \in \mathbb{R}$	Konstanten, durch griechische Kleinbuchstaben gekennzeichnet

KAPITEL 1

Motivation und Einleitung

Für die Entwicklung von komplexen robotischen Systemen nimmt die Interaktion und Kooperation von Mensch und Maschine sowohl in der Forschung als auch im alltäglichen Leben einen immer höheren Stellenwert ein. Besonders hervorzuheben ist der Bereich der Mechatronik, einer Disziplin, die durch die Kombination aus Maschinenbau (einschließlich Kamera- und Displaytechnik), Elektronik, Informationstechnik und Softwareentwicklung „intelligente Mechanismen“ schafft. Komplexe mechatronische Systeme sind beispielsweise Autos, die aufgrund ihrer Sensoren und Aktoren Unfälle vermeiden können, unbemannte Flugsysteme, die selbsttätig starten, landen und anderen Flugobjekten ausweichen können, oder aber Roboter: für die „Eroberung“ und Exploration des Weltraums, für die Nutzung in menschenfeindlichen Umgebungen, für die umfangreichen Produktionsabläufe in Fabriken, für die persönliche Betreuung von Menschen bis hin zu minimal-invasiver Chirurgie.

Im Rahmen der vierten industriellen Revolution (Industrie 4.0) werden Prozesse und Systeme immer mehr zu einem Konglomerat aus Soft- und Hardware-Modulen, die zusammen mit dem Nutzer ein effizientes Gesamtsystem ergeben sollen. Diese vernetzten, kommunizierenden, heterogenen Module werden oft auch als „Internet of Things“ bezeichnet. Auf der einen Seite nimmt die Komplexität und Heterogenität dieser Module immer weiter zu, auf der anderen Seite soll der Mensch weiterhin im Mittelpunkt stehen und zentrales Element in der Industrie 4.0 und im Internet der Dinge und Dienste sein. Gerade im Bereich der Robotik sollen Mensch und Roboter ein hybrides Team bilden, um gemeinsam ein Ziel zu erreichen. Hierbei werden in der Industrierobotik „Cobots“ (Collaborative Robots) entwickelt, die durch adaptive Regelung und inhärente Sicherheit die Kollaboration von Mensch und Roboter erst ermöglichen.

In der mobilen Robotik hingegen wird mit Teleoperation und Autonomie ein kooperatives hybrides Team aus Mensch und Roboter erschaffen. Aktuelle Forschung im Bereich der mobilen Robotik wird z. B. von der DARPA im Rahmen wissenschaftlicher (hochdotierter) Wettbewerbe wie der DARPA Robotics Challenge (DRC) [b] gefördert. Hier sollen neue mobile Roboter entwickelt werden, die in Rettungseinsätzen genutzt werden können und dabei verschiedene Aufgaben erfüllen müssen (wie z. B. Auto fahren, Türen öffnen, unebenes Terrain bewältigen usw.). Bei solchen Wettbewerben, wie auch in der finalen unbekannten dynamischen Katastrophenumgebung, ist neben der Hardware und der Steuerung insbesondere die Qualität der Nutzerschnittstelle ausschlaggebend, um

eine effiziente, konstruktive und sichere Mission erfolgreich abzuschließen.

In allen Fällen sollen die Fähigkeiten von Mensch und Maschine möglichst effizient miteinander gekoppelt werden, um das Ziel – die dynamische Fertigung verschiedener Produkte oder eine erfolgreiche (Rettungs-)Mission – zu erreichen. Wenn man nun wirklich den Menschen in der Entwicklung und im Umgang mit diesen komplexen Systemen unterstützen möchte, sind Assistenzsysteme vonnöten, welche dem Nutzer über den gesamten Entwicklungs- und Nutzungsprozess die Arbeit erleichtern. Das heißt, die Interaktion mit diesen komplexen, digitalisierten, vernetzten Systemen, sei es direkt an der Maschine oder aus der Ferne im Sinne der Interaktion mit einem Computer (der dann wiederum mit dem physischen Asset kommuniziert), benötigt einen Paradigmenwechsel in der Mensch-Technik-Interaktion, um die Komplexität zu abstrahieren und den Loop zwischen Menschen und Maschinen zu schließen. In der Entwicklung haben sich dazu heutzutage Simulationen etabliert, die es erst möglich machen, komplexe Sachverhalte und Systeme zu verstehen sowie diese zu entwickeln, analysieren, optimieren etc., jedoch endet die Nutzung von Simulationen mit der Fertigung.

Problemdefinition

Immer komplexer werdende Systeme erschweren dem Menschen einen einfachen Zugang zu diesen bzw. ein intuitives Verständnis des Gesamtsystems. Deren Komplexität sowie Defizite bei der Interaktion, Steuerung, Teleoperation, Autonomie, Kollaboration und Kooperation solcher Systeme zeigen klar die Grenzen bei der Entwicklung und Nutzung aktueller robotischer Systeme auf. Das heißt, komplexe Systeme von der Grundlagenforschung bis zum Endkunden/Nutzer zu begleiten (*entwickeln*, *überwachen*, *warten* und *betreiben*) ist Gegenstand aktueller Bemühungen in Forschung, Entwicklung und Vermarktung. Zur Bewältigung ist eine allgemeine Lösung notwendig, die durch Schnittstellen zwischen Mensch und Maschine sowie innovative Möglichkeiten der Interaktion einen intuitiven Zugang und symbiotisches Miteinander über deren gesamten Lebenszyklus hinweg erlaubt. Für die zuvor beschriebenen komplexen, verteilten, heterogenen, robotischen Systeme ist ein Assistenzsystem bzw. Mediator notwendig, um die Komplexität des Gesamtsystems zu verschleiern, dem Menschen einen intuitiven Zugang zum physischen Asset und schlussendlich ein effizientes, symbiotisches Miteinander von Mensch und Maschine zu ermöglichen.

Ziel dieser Arbeit

Im Rahmen dieser Arbeit soll daher eine Struktur entwickelt werden, die den Menschen in den Mittelpunkt stellt und ihm einen intuitiven Zugang zu komplexen robotischen Systemen über die gesamte Wertschöpfungskette hinweg ermöglicht. Zentraler Bestandteil ist dabei der Digitale Zwilling (Digital Twin (DT)), das virtuelle Abbild eines physischen Assets. Dieser soll nicht nur genutzt werden, um neue Technologien, Maschinen, Konzepte etc. zu entwickeln, sondern vielmehr selbst auch in der Endanwendung als Mediator eingesetzt werden. In dieser Arbeit wird daher das Konzept, die Implementierung und die Anwendung von Digitalen Zwillingen als zentrale Entwickler- und Nutzerschnittstelle über den gesamten Lebenszyklus hinweg aufgezeigt, wie in Abb. 1.1 verdeutlicht.

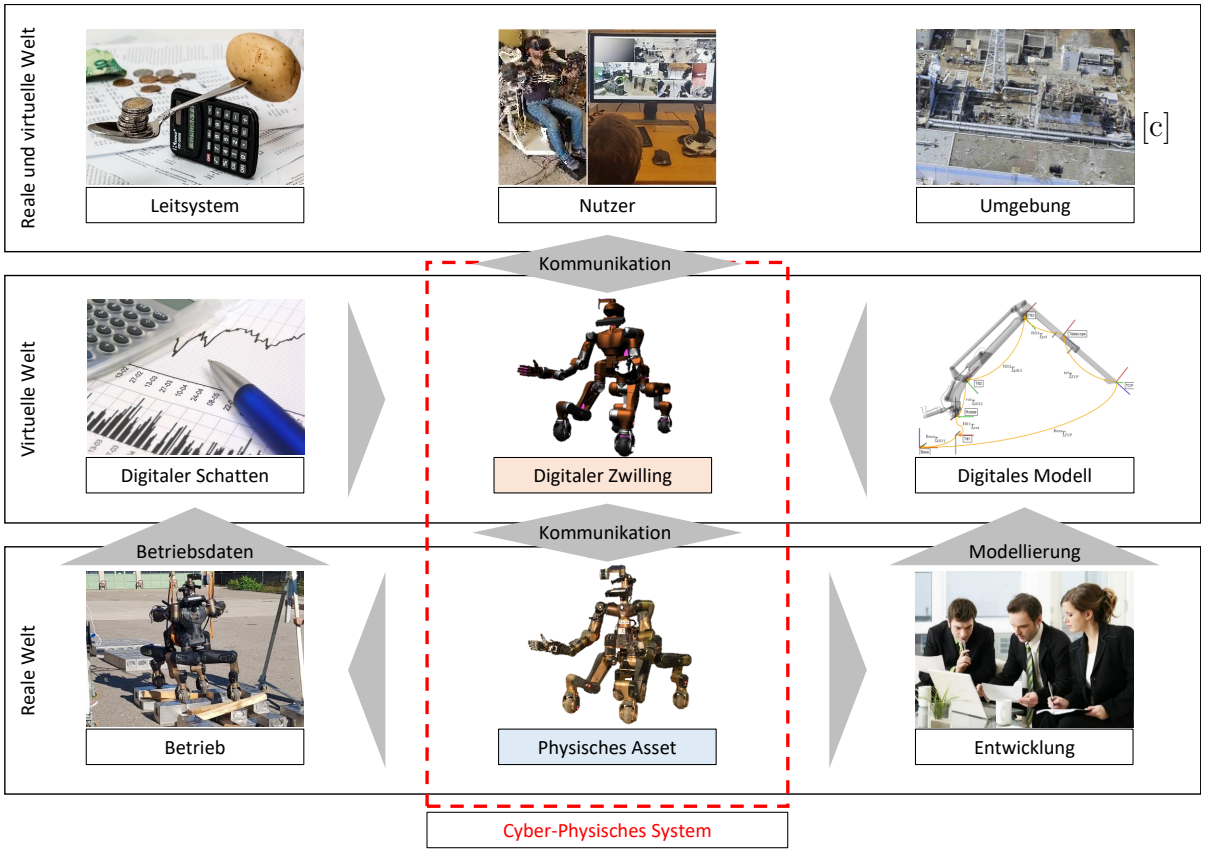


Abb. 1.1: *Thema dieser Arbeit: Der Digitale Zwilling als Mediator zur bidirektionalen Steuerung, Interaktion, Kooperation und Kollaboration sowohl in der Entwicklung als auch im Einsatz robotischer Systeme: (rechts) Entwicklung komplexer Systeme mit Hilfe und auf Basis des Digitalen Zwillings, (mitte/unten) Kombination von realen und virtuellen Komponenten zu einem cyber-physischen System, (mitte/oben) Intuitive Interaktion mit komplexen Systemen durch den Digitalen Zwilling, (links) Steuerung komplexer Systeme mit dem Digitalen Zwilling „in-the-loop“ (basierend auf [15])*

Durch eine Struktur, die den Digitalen Zwilling als lebenszyklusübergreifenden Mediator etabliert, kann dieser vielfältig eingesetzt werden und in unterschiedlichen Funktionen Anwendung finden:

- Der Digitale Zwilling überbrückt Prozesse (Entwicklung zu Betrieb).
- Der Digitale Zwilling kombiniert Virtualität und Realität (virtuelle und reale Welten, Daten oder Zwillinge).
- Der Digitale Zwilling ermöglicht cyber-physische Systeme (CPS), bestehend aus Digitalem Zwilling, physischem Asset und Kommunikation.
- Der Digitale Zwilling assistiert bei der kollaborativen Entwicklung des Assets.
- Der Digitale Zwilling unterstützt die Interaktion des Menschen mit der Maschine durch die Kommunikation von Nutzern zu cyber-physischen Systemen.
- Der Digitale Zwilling fungiert als Assistenzsystem bei der Bedienung komplexer Systeme und dient dem intuitiven Verständnis dieser.

- Der Digitale Zwilling integriert das Gesamtsystem inklusive Asset, Nutzer, Steuerung, Leitsystem, Umgebung etc.

In dieser Arbeit wird dargelegt, *warum* sich der Digitale Zwilling dazu besonders eignet, *wie* der Digitale Zwilling zum Mediator wird, *wann* (bzw. in welchen Entwicklungsphasen) dieser dann *wie* genutzt werden kann und *welche* offenen Fragestellungen damit schlussendlich beantwortbar sind.

Aufbau dieser Arbeit

Hierzu sind zunächst in Abschnitt 2 die notwendigen Grundlagen und der aktuelle Stand der Technik aufgeführt. In Kap. 2.1 werden die in dieser Arbeit zentralen Arbeits- und Forschungsschwerpunkte aufgeführt, die in den darauf folgenden Kapiteln aufbereitet werden. Zunächst werden in Kap. 2.2 allgemeine Mediatoren in Form von Assistenzsystemen vorgestellt. In Kap. 2.3 werden die Mediatoren der Mensch-Maschine-Interaktion (MMI), in Kap. 2.4 die der Mensch-Roboter-Kooperation (MRK) beleuchtet. Daraufhin werden 3D-Simulationssysteme und die Nutzung eines Virtuellen Testbeds (VTB) in Kapitel 2.5 erklärt. Damit sind alle Voraussetzungen geschaffen, um in den folgenden Kapiteln ein solches System zu modifizieren, zu erweitern und anzupassen, um den DT einzubetten und eine effiziente Kopplung aus Mensch und Maschine zu schaffen. Hierzu wird in Abschnitt 3 das Gesamtkonzept erläutert. Dessen Implementierung wird in Abschnitt 4 beschrieben und in den darauf folgenden Kapiteln umgesetzt: In Kapitel 4.2 in Form von Nutzerschnittstellen, in Kap. 4.3 die Einbettung des DTs und weiterer integrierter Funktionalitäten ins VTB, in Kap. 4.4 standardisierte Hardwareschnittstellen und schließlich ergibt all dies in Kap. 4.5 ein Gesamtsystem aus Nutzer, DT und physischem Asset. Anwendungsmöglichkeiten dieses Konzeptes werden dann in Abschnitt 5 in Form von Interaktion, Kooperation und direkter Steuerung robotischer Systeme aufgezeigt. Abschnitt 6 beinhaltet einen zusammenfassenden Überblick dieser Arbeit sowie Hinweise auf mögliche Weiterentwicklungen und zukünftige Forschungszweige und -projekte.

KAPITEL 2

Stand der Forschung

Dieses Kapitel gibt einen Überblick über die theoretischen Grundlagen und den Stand der Technik. Aktuelle Forschungsziele und -inhalte aller Bereiche, die in dieser Arbeit adressiert werden, werden hier eingeführt, erläutert und in den Kontext der gesamten Arbeit eingeordnet.

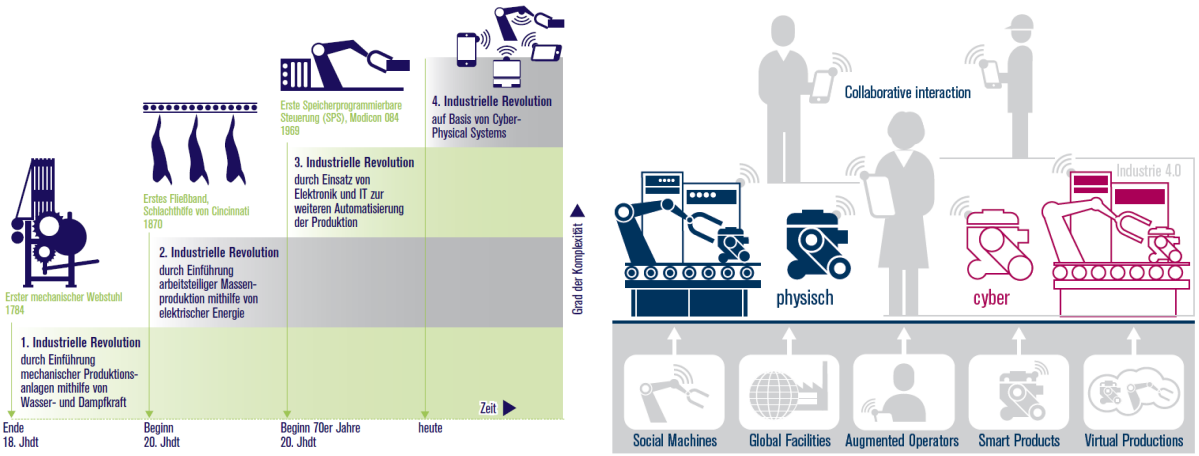
Das Feld der Interaktion, Kooperation und Kollaboration zwischen Mensch und Maschine, Computer und Roboter umspannt weite Felder der Forschung und des täglichen Lebens. Deshalb soll eingangs (Kap. 2.1) abgegrenzt werden, inwiefern Mediatoren zwischen Mensch und Maschine existieren. Danach werden in Kapitel 2.2 Assistenzsysteme allgemein, daraufhin die Methoden der Mensch-Maschine-Interaktion (MMI) (Kap. 2.3) und der Mensch-Roboter-Kooperation (MRK) (Kap. 2.4) beschrieben. Daraufhin wird der Fokus auf die virtuellen Experimentierumgebungen gelegt, die für solche ganzheitlichen MMI- und MRK-Systeme in Frage kommen (Kap. 2.5). Hier werden Kriterien für die Auswahl des 3D-Simulationssystems erläutert, welches als Entwicklungsumgebung die Implementierungsbasis dieser Arbeit bildet.

2.1 Abgrenzung der Arbeits- und Forschungsschwerpunkte

Das Vermitteln zwischen Mensch und Maschine umfasst ein breites Spektrum an Forschungsbereichen. Daher soll hier zunächst eine Abgrenzung stattfinden, welche Arbeits- und Forschungsschwerpunkte in dieser Arbeit behandelt werden. Insbesondere soll hierbei ein Mediator geschaffen werden, der über den gesamten Lebenszyklus einer komplexen Maschine genutzt werden kann, um dem Menschen zu helfen und diesen zu entlasten.

2.1.1 Forschungstrends und -ideen: Industrie 4.0, Internet of Things, cyber-physische Systeme

Im Folgenden sind zentrale Abschnitte aus der „Umsetzungsempfehlung Industrie 4.0“ zusammengetragen, welche sehr gut beschreiben, was die vierte industrielle Revolution ausmacht (siehe auch Abb. 2.1). Die zentralen Stichwörter sind zusätzlich hervorgehoben und werden daraufhin auf die im Rahmen dieser Arbeit zentral relevanten Bestandteile heruntergebrochen.



(a) Industrielle Revolutionen [16]

(b) Cyber-physisches System (CPS) [17]

Abb. 2.1: *Industrie 4.0 (I4.0): hoch komplexe, heterogene, vernetzte Gesamtsysteme auf Basis von CPS mit dem Menschen im Mittelpunkt*

> „Schrauben kommunizieren mit Montagerobotern, selbstständig fahrende Gabelstapler lagern Waren in Hochregale ein, intelligente Maschinen koordinieren selbstständig Fertigungsprozesse. Menschen, Maschinen und Produkte sind direkt miteinander vernetzt: die vierte industrielle Revolution hat begonnen.“[d] „Es entstehen zunehmend autonome Systeme, die umfassend Daten über ihr Umfeld erfassen und in vielfältiger Weise darauf reagieren können. Werden diese Systeme drahtlos untereinander und mit den Daten und Diensten des Internets vernetzt, entsteht das *Internet der Dinge*. [...] Bei diesen so evolutionär entstehenden *Cyber-Physical Systems (CPS)* werden Objekte bis hin zu Alltagsgegenständen durch Programmierbarkeit, Speichervermögen, Sensoren und Kommunikationsfähigkeiten intelligent. Sie können dadurch direkt oder über das Internet durch sogenannte Machine-to-Machine-

> Kommunikation eigenständig Informationen austauschen, Aktionen auslösen und sich wechselseitig steuern. Dies verbessert die Durchführung industrieller Prozesse in der Produktion, dem Engineering für die Industrie, der Materialverwendung und des Lieferketten- und Lebenszyklusmanagements enorm und führt so zu einer neuen Form der Industrialisierung: *Industrie 4.0*. [...] Industrie 4.0 meint im Kern die technische Integration von CPS in die Produktion und die Logistik sowie die Anwendung des Internets der Dinge und Dienste in industriellen Prozessen – einschließlich der sich daraus ergebenden Konsequenzen für die Wertschöpfung, die Geschäftsmodelle sowie die nachgelagerten Dienstleistungen und die Arbeitsorganisation.“ [16] „Die vierte industrielle Revolution führt somit auch zu einem Paradigmenwechsel in der Mensch-Technik- und Mensch-Umgebungs-Interaktion mit neuartigen Formen der kollaborativen Fabrikarbeit. Die Verwirklichung der agilen multiadaptiven Smart Factory erfordert menschenzentrierte sozio-technische Fabrik- und Arbeitssysteme. Die Technik soll dabei die kognitive und physische Leistungsfähigkeit der Beschäftigten durch die richtige Balance von Unterstützung und Herausforderung fördern — insbesondere im Hinblick auf die industriellen Assistenzsysteme, die Mensch-Technik-Kooperation sowie die Aspekte der Qualifizierung und Weiterbildung. [...] Im Mittelpunkt der Interaktionen in der Fabrik stehen die Beschäftigten (Augmented Operators), die ihre Aufgaben durch die (virtuell) erweiterte Sicht auf die reale Fabrik besser wahrnehmen können. Sie erweitern ihre Fähigkeiten stetig und werden so vom Bediener zum kooperierenden Steuerer, Regulierer und Gestalter, der seine Expertise kontinuierlich einbringen kann. [...] Völlig neuartige multimodale Benutzerschnittstellen spielen hierfür eine entscheidende Rolle. Diese Schnittstellen nutzen einerseits den Aufgabenkontext der Beschäftigten für proaktive Assistenzfunktionen und unterstützen den Arbeiter gleichzeitig auch durch adaptiven Privatsphärenschutz — wobei sie gleichzeitig die Sicherheitsaspekte der Smart Factory berücksichtigen.“ [17]

Industrie 4.0 umfasst also die aktuellen Bemühungen aus Forschung und Entwicklung insbesondere im Bereich der „Smart Factory of the Future“. Die Basis dieser Revolution, die CPS, lassen sich aber nicht nur im Bereich der Industrierobotik anwenden, sondern sind vielmehr selbst ein eigener Forschungsbereich.

Auch der Gartner Hype Cycle (siehe Abb. 2.2), der die „Emerging Technologies“ von 2017 – 2027 aufzeigen soll, untermauert die Trends, die mit Industrie 4.0 und den CPS einhergehen. Insbesondere hervorzuheben sind hierbei laut Gartner drei zentrale Trends:

(i) AI Everywhere
(Smart Robots, Smart Workspace, Deep Learning, Autonomous Vehicles etc.),
(ii) Transparently Immersive Experiences
(Augmented Reality, Virtual Reality, Human Augmentation etc.) und
(iii) Digital Platforms
(Digital Twin, IoT Platform, Software-defined Security etc.).

Neben den zuvor erwähnten Trends werden die fortschreitende Digitalisierung und die immer intelligenter werdenden Maschinen dem Nutzer transparent und immersiv durch

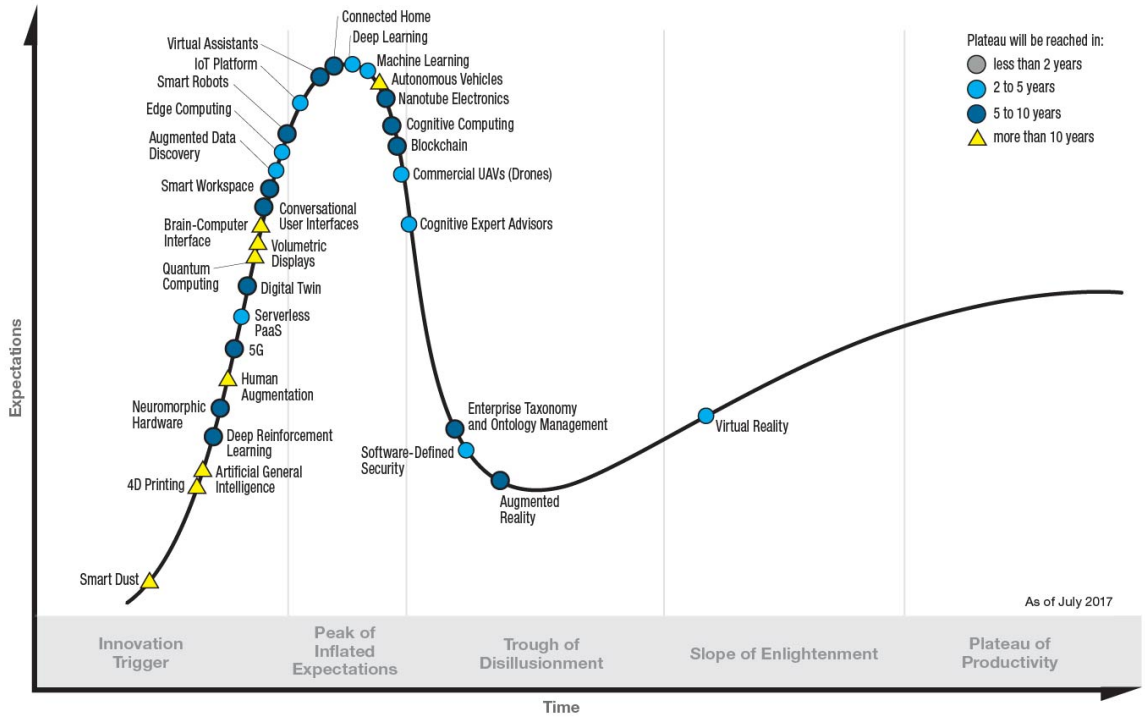


Abb. 2.2: *Gartner Hype Cycle 2017 [e]*

neue Visualisierungstechnologien zugänglich gemacht.

Auf Basis dieser Trends und Technologien lassen sich folgende Arbeits- und Forschungsschwerpunkte dieser Arbeit definieren:

1. Kommunikation und Schnittstellen (CPS, M2M, MMI, MRK),
2. verbessertes Lebenszyklusmanagement,
3. Integration (CPS, IoT),
4. neuartige Kollaboration – der Mensch im Mittelpunkt,
5. agil, multiadaptiv, menschenzentriert, kognitiv, physisch,
6. Augmentierung (AR), Virtualisierung (VR), Immersion, Transparenz,
7. neuartige multimodale Benutzerschnittstellen,
8. Assistenz und Sicherheit.

Diese Forschungstrends sollen zeigen, dass zur Umsetzung dieser ein Mediator notwendig ist, der zwischen dem Menschen und den komplexen Systemen vermittelt, auf Basis der aktuell verfügbaren und sinnvoll einzusetzenden Technologiebausteine.

2.1.2 Der Lebenszyklus von Maschinen

Klassische Assistenz- und Unterstützungssysteme dienen oft dazu, einem Endkunden die Interaktion mit einem komplexen System zu erleichtern. Dies ist aber nur ein Teil des Lebenszyklus von Maschinen, in denen Mediatoren sinnvoll zum Einsatz kommen

können. Insbesondere kann man hier unterscheiden zwischen dem Entwicklungs- und Nutzungsprozess, der nicht nur von unterschiedlichen Personen mit unterschiedlichen Fähigkeiten, sondern auch mit ganz anderen Werkzeugen zur Interaktion oder Assistenz einhergeht.

2.1.3 Komplexität in der Entwicklung und Nutzung von Robotern

Assistenz- und Unterstützungssysteme werden erst nötig, wenn die Arbeitsbelastung des Menschen zu hoch bzw. die betrachtete Maschine zu komplex wird, um sie intuitiv zu verstehen. Diese Komplexität umfasst:

- Komplexität des Systems – die Struktur, Morphologie, Anzahl und Heterogenität verschiedener Module etc.
- Komplexität des Entwicklungsprozesses – das Zusammenspiel verschiedener Entwicklungsdomänen, vieler verschiedener Entwickler an verschiedenen Standorten mit unterschiedlichem Wissensständen.
- Komplexität der Steuerung/Bedienung/Interaktion des finalen Systems – zu viele Freiheitsgrade, zu viele Daten, heterogene undurchschaubare Struktur, unterschiedliche Schnittstellen verschiedener Module, unintuitive Eingabegeräte, Sicherheitsaspekte usw.

In all diesen Ebenen kann der Entwickler bzw. Nutzer durch Assistenzsysteme unterstützt werden, um der Komplexität Herr zu werden. Gerade für den Entwicklungsprozess werden hierfür gerne Simulationen eingesetzt, um komplexe Systeme zu verstehen und entwickeln zu können.

2.1.4 Mediatoren

Schlussendlich ergeben sich also die Forschungsschwerpunkte der Assistenzsysteme, der Mensch-Maschine-Interaktion (MMI), der Human-Computer Interaction (HCI) und der Mensch-Roboter-Kooperation (MRK), welche in den folgenden Kapiteln (Kap. 2.2, Kap. 2.3 und Kap. 2.4) thematisiert werden, bevor in Kap. 2.5 Simulationssysteme vorgestellt werden. Hiermit soll die Forschungsfrage beantwortet werden, welche Mediatoren und Assistenzsysteme es gibt und wie schon heute versucht wird, in Teilen des Lebenszyklus Simulationssysteme als Mediatoren einzusetzen.

Mediatoren zwischen Mensch und Maschine umfassen:

1. Assistenz- und Unterstützungssysteme im Allgemeinen (siehe Kap. 2.2) – also Möglichkeiten, durch Soft- und Hardware dem Menschen die Arbeit an Maschinen zu erleichtern. Dies umfasst:
   a) eine vereinfachte Eingabe,
   b) eine aufbereitete Datenausgabe, meist visueller Natur.
2. Die Interaktion zwischen Mensch und Maschine (siehe Kap. 2.3) – also die Steuerung von Maschinen durch den Menschen. Dies umfasst primär die Schnittstelle von Mensch zu Computer.

3. Die Interaktion, Kooperation und Kollaboration von Mensch und Roboter (siehe Kap. 2.4) – Dies umfasst zusätzlich zu dem zuvor Erwähnten:
   a) neue Technologien, die Roboter inhärent sicher und kollaborativ gestalten (Hardware- und Software-seitig),
   b) die verwendete „Middleware“, die den Übergang zwischen Nutzerhardware und Roboterhardware verbessert und erleichtert.

Zusammengefasst ist dies in Def.[1] 1 erläutert.

Definition 1: Mediator

Mediator wird hierbei im Wortsinn verwendet als „Vermittler“ zwischen Mensch und Maschine. Der Mediator integriert und koordiniert heterogene, verteile Hard- und Softwarekomponenten mit dem Menschen zu intelligenten Gesamtsystemen.

[1] Alle Definitionen zeigen die Sichtweise, die im Rahmen dieser Arbeit eingenommen wird.

2.2 Assistenzsysteme

Um den Menschen zu unterstützen sind *Unterstützungssysteme* ins Leben gerufen worden, die mit Hilfe intelligenter Techniken Mittler- und Assistenzaufgaben in der Mensch-Maschine-Interaktion übernehmen [18]. Der Grundgedanke besteht darin, die Systemschnittstelle selbst intelligenter zu machen, d. h. sie in die Lage zu versetzen, zwischen Benutzer und technischem System zu vermitteln, indem sie Anteile der Lösungsfindung eigenständig erbringt. Für diese Interaktionsform wurde von [19] die Metapher des *indirekten Managements* geprägt. Hierbei handelt es sich um einen kooperativen, zum Menschen komplementär ausgerichteten Ansatz (Human Complementary Approach) [20]), der sowohl das künstliche System als auch den Menschen als einzigartiges System mit besonderen Fähigkeiten und Stärken ansieht. Dies führt zu einer Asymmetrie, bei welcher dem künstlichen System und dem Menschen unterschiedliche Rollen übertragen werden, in denen die Verantwortlichkeiten derart aufgeteilt werden, dass ihre jeweiligen Stärken möglichst gut ausgeschöpft werden.

Geht man noch einen Schritt weiter, kann der kommunikative Umgang zu virtuellen konversationalen Agenten (Embodied Conversational Agents) führen, die menschenähnlich konzipiert sind und durch kognitive Verarbeitung die Auswahl von Aktionen treffen können. Damit verschiebt sich die Sicht, dass Menschen „Benutzer“ einer Anwendung sind, zu der einer Sicht von „Partnerschaft“ mit künstlichen Agenten.

Die zentralen inhaltlichen Aspekte der MMI zur Interaktion mit komplexen Systemen lassen sich wie folgt zusammenfassen:

1. direkte Manipulation,
2. Etablierung multimodaler Schnittstellen (z. B. durch Gesten),
3. VR als neue interaktive Mensch-Maschine-Schnittstelle,
4. Unterstützungssysteme zur Assistenz,
5. indirektes Management oder
6. miteinander kommunizierende Agenten.

Aus implementierungstechnischer Sicht beinhaltet die MMI folgende Aspekte:

1. Ein- und Ausgabe von Informationen und Daten,
2. Design der Nutzerschnittstellen,
3. Design der Interaktion,
4. Evaluierung der Nutzererfahrung.

Ausgehend von der Annahme, dass intuitive Interaktion auf Anwendung von Vorwissen beruht und dieses vor allem in der Interaktion mit realen Objekten erworben wird, interessiert in einem weiteren Ansatz die Frage, ob physisch repräsentierte, greifbare Benutzungsschnittstellen (Tangible User Interfaces [21]) besondere Möglichkeiten zur Unterstützung intuitiver Interaktionen aufweisen. Beobachtungen elementarer Objektmanipulationen an physisch und virtuell repräsentierten Objekten haben signifikante Unterschiede in Geschwindigkeit, Fehlerrate und empfundener pragmatischer Qualität zugunsten physischer Objekte ergeben. Die Forschungsfrage ist hier, wie die einer intuitiven Interaktion förderlichen Eigenschaften physischer Schnittstellen auf die Manipulation

insbesondere komplexer digitaler Daten übertragen werden können, die meist abstrakt und stark vernetzt organisiert sind und sich somit nicht eins-zu-eins physisch abbilden lassen [22].

2.2.1 Assistenz und Arbeitsteilung in der Industrie 4.0

Für eine intelligente Hybridisierung der Fähigkeiten von Mensch und Maschine wurden Assistenzsysteme ins Leben gerufen, um die jeweiligen Schwächen zu kompensieren. Es gibt dabei fünf Arten des Supports: *physisch*, *physiologisch*, *kognitiv*, *organisatorisch* und *überwachend*. Assistenzsysteme können in jeder dieser Domäne unterstützen. Zusätzlich dazu gibt es drei Determinanten, um Assistenzsysteme zu klassifizieren (siehe Abb. 2.3): Die *raum-zeitliche Relation*, die *Form der Kopplung* sowie die *Verortung von Kontrolle*. Die Form der Kopplung wiederum kann man unterscheiden in *haptisch*, *visuell* oder *akustisch*.

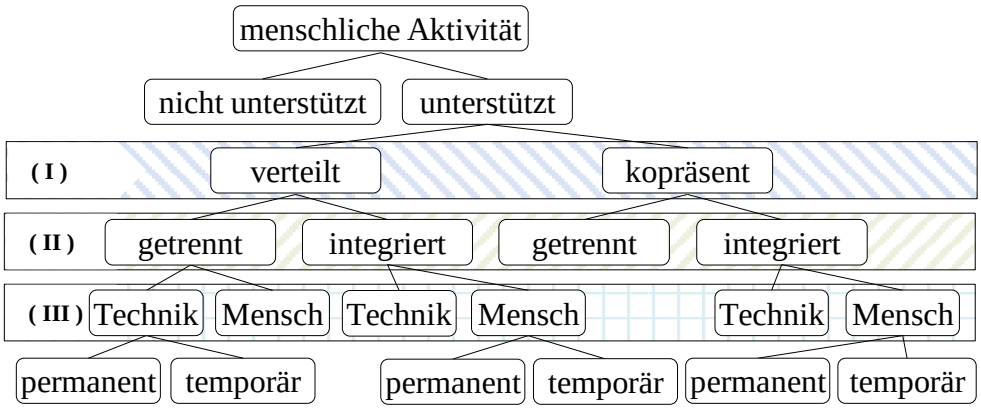


Abb. 2.3: *Die drei Determinanten zur Klassifikation von Assistenzsystemen (basiert auf [23]): (I) raum-zeitliche Beziehung, (II) Form der Kopplung und (III) Kontrollhoheit.*

Im Bereich der cyber-physischen System ist der Mensch ein essenzieller Teil dieses Systems aus realem System und digitalem Abbild. Basierend auf dem Modell der menschlichen Informationsverarbeitung (Abb. 2.4) sollte sich eine Assistenz auf die Bereiche *perception*, *cognition* oder *action* fokussieren. Diese überspannen ein weites Feld aus physiologischen und technischen Disziplinen, die Assistenzsysteme abdecken müssen.

2.2.2 Motorische/physische Assistenz

Eine kopräsente integrierte Lösung für die motorische Unterstützung bzw. die kooperative Kopplung menschlicher Intelligenz mit robotischer Kraft, Genauigkeit und Effizienz sind sogenannte Exoskelette. Direkt mit den Extremitäten des Menschen verbunden, unterstützen diese die Muskeln in diversen Anwendungen. Einen generellen Überblick über diese integrierten Mensch-Roboter-Systeme ist in [26] und [27] zu finden. Speziell für die Beine zeigt [28] Hocken, Beugen, Schwingen, Drehen und Laufen wie auch das Treppensteigen mit Gepäck. Speziell für den Oberkörper werden Exoskelette von [29] oder [30] für

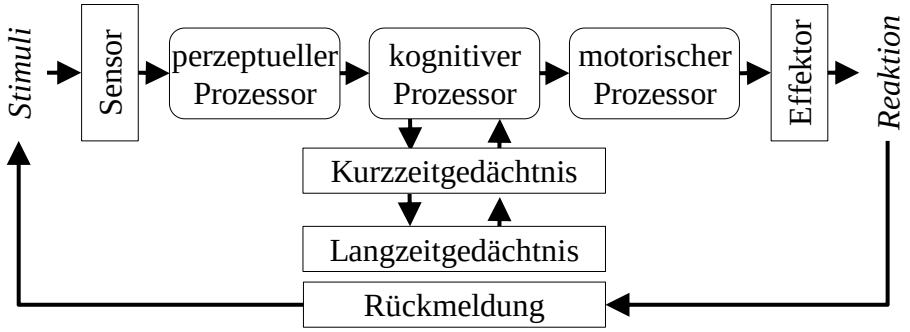


Abb. 2.4: *Modell der menschlichen Informationsverarbeitung (basiert auf [24], [25]): perception, cognition und action – in all diesen Dimensionen kann der Mensch unterstützt werden*

physisch beeinträchtigte Personen sowie bei der Assistenz von manuellen Fügeaufgaben genutzt.

Statt der direkten physischen Kopplung ist es auch möglich, Exoskelette in verteilten und getrennten Mensch-Roboter-Aufgaben einzusetzen. Die Nutzung eines Ganzkörper-Exoskeletts mit Kraftrückkopplung als Eingabegerät für einen mobilen humanoiden Roboter wird in [10] gezeigt.

2.2.3 Perzeptions-/Wahrnehmungsassistenz

Neben der physischen Assistenz gibt es auch mehrere Möglichkeiten, die menschliche Wahrnehmung zu unterstützen. Haptische Assistenz kann erreicht werden, wenn z. B. Eingabegeräte mit Kraftrückkopplung zur Steuerung genutzt werden, wie zuvor erläutert. Auch wenn es fünf menschliche Sinne gibt, werden 80 % aller Informationen aus dem Sehen gewonnen [31]. Da das menschliche Sehen aber stark limitiert ist – z. B. bzgl. Wellenlänge und Beleuchtungsvoraussetzungen – kann dieses sehr gut durch robotisch-technische Kognition unterstützt werden. Zwei erwähnenswerte Beispiele sind die Visualisierung von (normalerweise unsichtbaren) Geräuschen bzw. Schallquellen sowie die Visualisierung infraroter Licht- bzw. Hitzequellen, demonstriert in [25].

2.2.4 Kognitive Assistenz

Um die Kognition eines menschlichen Nutzers zu unterstützen, muss primär die Arbeitslast des Menschen reduziert werden, indem man ihm einen direkten intuitiven Zugang zu allem Notwendigen ermöglicht.

Das heißt, die kognitive Assistenz ist eng verbunden mit der *visuellen Aufbereitung* eingehender Daten. Gerade im Automotive-Bereich gibt es vielfältige Beispiele für solche visuellen Assistenzsysteme wie z. B. den Toter-Winkel-Assistent, Abstandsregelung, Navigationssystem, Verkehrszeichenerkennung, Geschwindigkeitserkennung und -regelung sowie diverse Warnsysteme. All diese können auch auf mobile robotische Systeme und deren Steuerung übertragen werden, beispielsweise durch AR-Metaphern [8].

Neben solchen visuellen Systemen liegt eine weitere kognitive Assistenz des Menschen

in der *Autonomie* des zu steuernden Systems. In teleoperativ gesteuerten Systemen ist es hilfreich, eine Menge an Bewegungsabläufen wie z. B. Greifen oder das Manövrieren von Punkt a zu Punkt b (semi-)autonom durchführen zu lassen. Statt einer permanenten Autonomie können diese Aktionen unter der Aufsicht des menschlichen Nutzers durchgeführt und von diesem überwacht werden wie z. B. in [32] oder [33].

2.2.5 Assistenzsysteme im Automotive-Bereich

Das Führen eines Kraftfahrzeugs beinhaltet maßgeblich, Informationen in Reaktionen, meist eine Steuerungstätigkeit, umzusetzen. Dementsprechend werden Fahrerassistenzsysteme (FAS) entwickelt, die den Prozess der Informationsverarbeitung unterstützen wollen. Bereits von der Wortbedeutung ausgehend ist bei einem „Fahrerassistenzsystem“ auf das Zusammenwirken des Systems mit dem menschlichen Fahrer zu schließen. Diese Arbeitsteilung kann kategorisiert werden in (A) informierende und warnende Funktionen, (B) kontinuierlich automatisierende Funktionen und (C) eingreifende Notfallfunktionen. Beispiele für Fahrerassistenzsysteme sind: bremsenbasierte Assistenzfunktionen (ABS, ASR, MSR, ESP etc.), Bahnführungsassistenz (Sichtverbesserungssysteme, Einparkassistenz, Adaptive Cruise Control, Frontkollisionsschutz, Querführungs-, Fahrstreifen- und Kreuzungsassistenz), Navigationssysteme, Fahrerzustandsüberwachung und Fahrerwarnelemente. Einen generellen Überblick über Fahrerassistenzsysteme bietet [34], wo die Mensch-Maschine-Schnittstelle von Assistenzsystemen in zwei Hauptkategorien unterteilt wird: Bedienelemente und Anzeigen (vgl. Abb. 2.5).

Zusätzlich dazu beschreibt [35] sechs ergonomische Leitsätze bzw. Anforderungen, die bei der Gestaltung von Schnittstellen (Anzeigen und Stellteile) zu berücksichtigen sind:

1. Aufgabenangemessenheit (Funktionszuweisung, Komplexität, Gruppierung, Unterscheidbarkeit, funktioneller Zusammenhang),
2. Selbstbeschreibungsfähigkeit (Informationsverfügbarkeit),
3. Steuerbarkeit (Redundanz, Zugänglichkeit, Bewegungsraum),
4. Erwartungskonformität (Kompatibilität zum Erlernten, Kompatibilität zur Praxis, Konsistenz),
5. Fehlertoleranz (Fehlerkontrolle, Fehlerbehandlungszeit) und
6. Individualisierbarkeit und Lernförderlichkeit (Flexibilität).

Aktuelle Bemühung in der Forschung und Entwicklung von FAS sind z. B. Fahrerabsichtserkennung, kooperative Fahrzeugführung (Zusammenführen der Fähigkeiten, Autorität, Autonomie, Kontrolle und Verantwortung) sowie eine notwendige Standardisierung der Schnittstellen. Diese Standardisierung soll den Informationsaustausch zwischen verschiedenen Domänen sowie die Komplexität reduzieren. Außerdem ist damit die Wiederverwendbarkeit der Schnittstellen und der einzelnen Module gewährleistet. Ein Beispiel für diese Bemühungen ist AUTOSAR (AUTomotive Open System ARchitecture) [36] welche, initiiert von BMW, einen offenen Industriestandard für Softwarearchitekturen im Automotive-Bereich anstrebt.

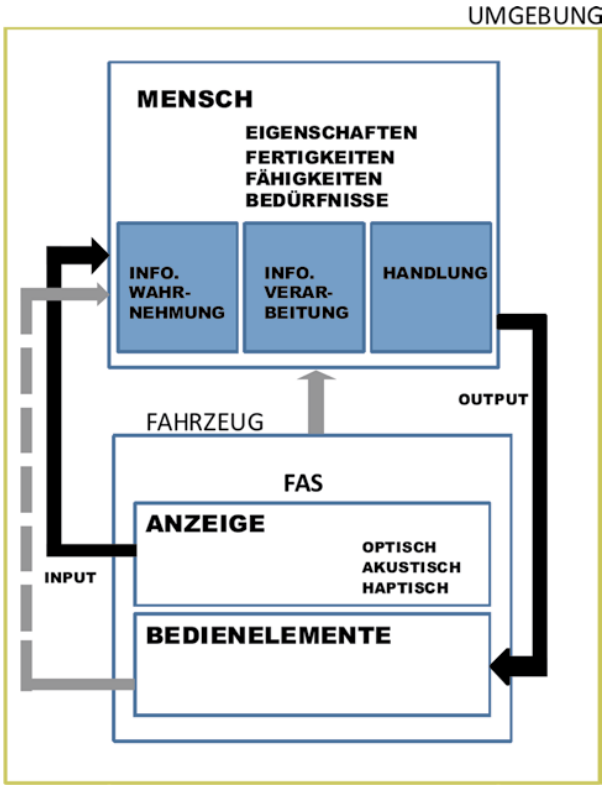


Abb. 2.5: *Fahrerassistenzsysteme (FAS): Arbeitsmodell der Mensch-Maschine-Schnittstelle [34]*

Zusammenfassend unterstützen Assistenzsysteme den Menschen meist kognitiv in den Bereichen (A) der Bedienung und (B) der Wahrnehmung bzw. Perzeption. Zentral gerade bei letzterem die Aufbereitung von Daten, sodass sogar Möglichkeiten entstehen, „Unsichtbares“ sichtbar zu machen.

2.3 Mediatoren zwischen Mensch und Maschine/Computer

Die Mensch-Maschine-Interaktion (MMI) behandelt die Interaktion zwischen Mensch und Maschine. In vielen Fällen ist die Maschine ein Computer bzw. enthält Informations- und Kommunikationstechnologien (IKT) sowie Anwendungs- oder Informationssysteme.

Hierbei richtet sich das Interesse auf die Entwicklung von Methoden und konkreten Technologien, welche sowohl die *Interaktion zwischen dem Nutzer mit einer Maschine* erleichtern als auch die *Benutzungsfreundlichkeit von Computer und Software* verbessern sollen. Neben den Entwicklungen in der Informatik umfasst die MMI natürlich auch Bereiche der Psychologie, Arbeitswissenschaft, Kognitionswissenschaft, Ergonomie, Soziologie und Design. Da dieses Gebiet so viele Teilgebiete umfasst, soll hier zunächst kurz auf den Stand der Technik eingegangen werden und daraufhin die Möglichkeiten und Ideen analysiert werden, die bei der Interaktion mit komplexen Systemen federführend sind oder sein werden.

2.3.1 Benutzerschnittstelle

Zunächst sei definiert, dass im Rahmen dieser Arbeit nicht untersucht wird, wie Menschen mit Computern oder Technologien umgehen, sondern vielmehr die Entwicklung neuer Methoden der Interaktion im Vordergrund stehen (siehe Abb. 2.6).

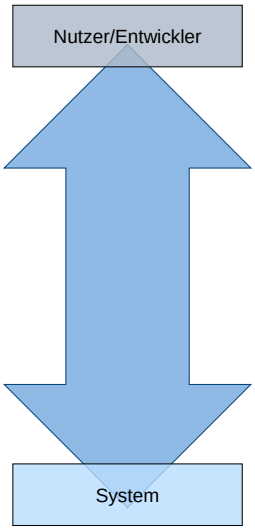


Abb. 2.6: *Bidirektionale Interaktion von Mensch und Maschine*

Somit ist das Hauptaugenmerk auf die Mensch-Maschine-Schnittstelle (Def. 2) bzw. die Benutzerschnittstelle (Def. 3) gerichtet, mit speziellem Fokus auf Effektivität, Effizienz und Nutzerzufriedenheit.

Definition 2: Mensch-Maschine-Schnittstelle (MMS)

Die Schnittstelle zwischen Mensch und Maschine umfasst sowohl die direkte Interaktion des Menschen mit der Hardware, als auch die Benutzerschnittstelle (siehe Def. 3) und zusätzlich die Hardwareschnittstelle und Middleware (siehe Kap. 2.4.4.2).

Definition 3: Benutzerschnittstelle/User Interface (UI)

Die Benutzerschnittstelle (User Interfaces (UI)) umfasst nach [37] bzw. [35] „alle Bestandteile eines interaktiven Systems, die Informationen und Steuerelemente zur Verfügung stellen, die für den Benutzer notwendig sind, um eine bestimmte Arbeitsaufgabe mit dem interaktiven System zu erledigen“.

Historisch gesehen wurden aus den Kommandozeileneingaben (1960er/1970er Jahre) zunächst menüorientierte Arbeisplatzrechner (1980er Jahre), bis die bis heute dominanten graphischen Benutzerschnittstellen (Graphical User Interfaces (GUIs)) Einzug erhielten. Hierbei kann der Nutzer mit Objekten wie auf einer echten Schreibtischoberfläche (englisch „desktop“) interagieren. Mit diesen WIMP-Schnittstellen (Windows Icon Menus Pointing Device) können Fenster, Icons, Menüs und Mauszeiger benutzt werden. Solche Interaktionsschnittstellen – oder auch neuere Entwicklungen wie Multi-Touch – werden nach [38] als *direkte Manipulation* bezeichnet. Diese vordergründig einfache und komfortable (und deshalb auch etablierte) Bedienung wird bei komplizierten Anwendungen auch komplexer, da sämtliche Aktionen auf die Seite des Nutzers verlagert sind. Daher sind intuitiv nutzbare Schnittstellen nötig, die den Menschen von technischen Details entlasten, wie Sprach- und Gesteneingaben oder sogar multimodale Schnittstellen [39]. „Parallel dazu entstand seit Mitte der 1980er Jahre mit 'Virtual Reality' eine neue Form interaktiver Mensch-Maschine-Schnittstellen. Im Vordergrund steht dabei die Echtzeitinteraktion mit rechnergenerierten Modellen oder mit Simulationsergebnissen, aber auch mit Ereignissen und Personen an anderen Orten und zu anderen Zeiten, zuweilen überlagert mit realen Darstellungen (AR oder MR).“ [40]

Eine weitergehende Frage der MMI zielt auf die Übertragbarkeit von Aspekten der natürlichen Interaktion und Kommunikation zwischen Menschen auf Maschinen ab. Bezugspunkte sind hier kognitive Leistungen wie das Verstehen und Produzieren von Sprache, Gestik oder Mimik in dialogischer Interaktion bis hin zur Mensch-Technik-Kooperation und ihrer Realisierung durch künstliche Gesprächspartner sowie „soziale“ Roboter.

2.3.2 Natürliche, intuitive und sichere Mensch-Maschine-Interaktion

Wie zuvor schon eingeleitet, sind die Wörter „natürlich“, „intuitiv“ und „sicher“ im Bereich der MMI von zentraler Bedeutung. Das Design von Nutzerschnittstellen an sich ist ein breit gefächertes Forschungsgebiet. Beispiele dafür sind z. B. sogenannte Dashboards, eine oft webbasierte Visualisierung des aktuellen Status von Maschinen, Unternehmen, Aktienkursen usw. Wie z. B. in [41] beschrieben, gibt es hierfür schon beliebig komplexe Herangehensweisen, um dort Daten zu visualisieren. Des Weiteren spielen die visuelle Aufbereitung in Form, Aufbau, Farben und die Farbwahl eine entscheidende Rolle. Nach [42] gibt es im Bereich der Gestaltung auch diverse Möglichkeiten, Daten räumlich, alphabetisch, zeitlich, kategorisiert oder hierarchisch anzuordnen und dafür spezielle Formen und Farbspektren für bestimmte Zwecke zu nutzen.

Dieser kurze Exkurs soll nur verdeutlichen, dass es sehr viele Parameter gibt, eine visuell ansprechende, intuitiv zugängliche Visualisierung zu erstellen.

Neben diesem – oft statischen – Zugang zu Daten zeigen aktuelle Computerspiele

sehr gut, wie man einem Nutzer dynamisch Daten in sein Sichtfeld projiziert. Aufgrund der weiten Verbreitung von Computern, Computerspielen sowie allgemeiner Consumer-Elektronik sind uns manche Gegenstände oder Interaktionsmetaphern vertraut. Bestes Beispiel hierfür ist die „pinch“-Geste auf Tablet/Smartphone oder PC, um zu zoomen.

Auf der anderen Seite haben sich im wissenschaftlichen Bereich oft extrem unintuitive, komplexe Programme etabliert, die aufgrund ihres Feature-Umfangs von ihren Nutzern in einzelnen Bereichen auch als natürliches Mittel angesehen werden. So sind die Nutzer von beispielsweise *VIM* („Vi IMproved“) [43] [f] unter Linux stets sehr zahlreich, obwohl es auf den ersten Blick ein extrem komplexes, unintuitives, konsolen- und textbasiertes Nutzerinterface anbietet.

Das bedeutet letztendlich, dass es die perfekte Nutzerschnittstelle nicht immer gibt und es oft auf den Nutzer und sein Vorwissen im Applikationsbereich, aber auch seine allgemeine Affinität zu Technik ankommt. Insgesamt sollte man bei der Entwicklung von Werkzeugen zur Nutzerinteraktion und -integration darauf achten, dass der Nutzer nicht unwissend ist, sondern ein gewisses Vorwissen mitbringt. Sinnvolle Visualisierungen wissenschaftlicher Daten oder zusätzliche Informationen können dies merklich verstärken. Es erscheint sinnvoll, sich an aktuellen Technologien sowie an Computerspielen zu orientieren, um den Nutzer an geeigneter Stelle „abzuholen“.

2.3.2.1 Intuitivität

Aufgrund der Tatsache, dass Intuitivität (vgl. Def. 4) stark subjektiv ist, ist es auch schwer, die Güte einer intuitiven Nutzerschnittstelle zu messen. Im Rahmen von [44] und [45] wurde dies versucht. Hierbei wird die intuitive Nutzung (einer Nutzerschnittstelle) als „die unterbewusste Anwendung von Vorwissen, das zu einer effektiven Interaktion führt“ definiert.

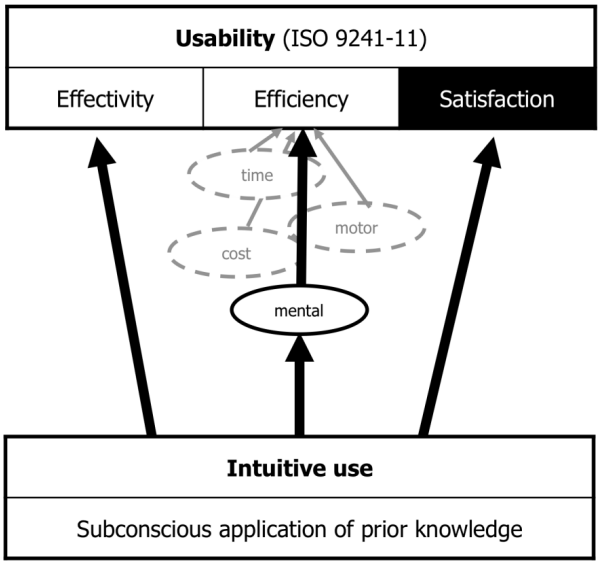


Abb. 2.7: *Fragebogen Intuitivität (Questionnaire for Intuitive Use (QUESI)) [46]*

Definition 4: Intuitivität

„Ein technisches System ist im Rahmen des spezifischen Kontextes einer Aufgabenstellung in dem Maße intuitiv benutzbar, in dem der jeweilige Benutzer mit ihm durch unbewusste Anwendung von Vorwissen effektiv interagieren kann.“ [44]

Zur Bewertung der „Usability“ einer Nutzerschnittstelle (vgl. Abb. 2.7) sind in [46] die subjektiven Folgen (in Form eines Fragebogens) analysiert worden in Form von:

(W) geringer subjektiver mentaler Arbeitsbelastung (workload),
(G) positiver Wahrnehmung der Zielerreichung (goal),
(L) negativer Wahrnehmung der Anstrengung des Lernprozesses (leaning),
(F) hoher Vertrautheit (familiarity),
(E) negative Wahrnehmung der Fehlerrate (error).

Hierbei wird die „intuitive Benutzung“ als „die unbewusste Anwendung von Vorwissen, welches zu einer effektiven und zufriedenstellenden Interaktion bei minimalem Verbrauch kognitiver Ressourcen führt“ beschrieben (siehe auch [47]).

2.3.2.2 Sicherheit

Im Bereich der Sicherheit hängt diese bei der MMI von diversen Faktoren des untersuchten Systems ab wie Geschwindigkeit, Abstand, Leistung, Kraft usw. Für industrielle Anwendungen werden hierbei Gefährdungsbeurteilungen nach [48] sowie [49] erstellt. Diese führen dann für den Betrieb industrieller Anlagen dazu, dass für ein benötigtes Sicherheitslevel spezielle Begrenzungen, Achsüberwachungen o. Ä. installiert werden müssen. Spezialisierte Roboterentwicklungen für den kollaborativen Betrieb nach [50] [51] werden dann meist über Leistungs- oder Kraftbegrenzungen gesichert. Die Applikationsanforderungen für solche kollaborativen Roboter sind in [52] festgehalten.

2.3.2.3 Komplexität und intuitive Interaktion

Im Hinblick auf Interaktionsmöglichkeiten im Allgemeinen gibt es drei Unterscheidungen:

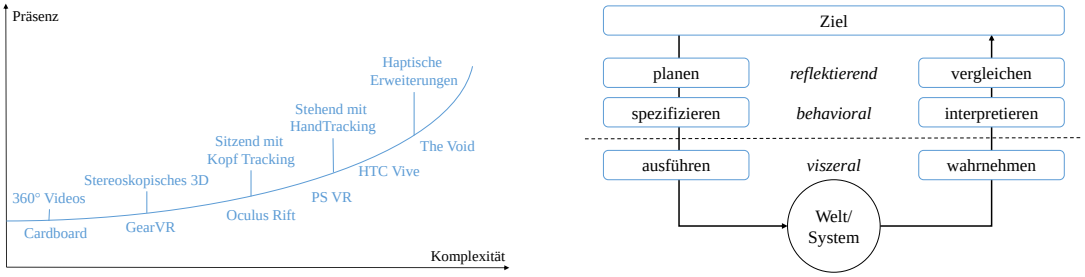
1. direkte Interaktion,
2. metaphorische Interaktion,
3. abstrakte Interaktion.

Wo die direkte Interaktion noch physikalischen Regeln folgt (Drücken eines virtuellen Knopfes), werden auf der metaphorischen und vor allem der abstrakten Ebene neue Logiken eingeführt, welche mal mehr, mal weniger bekannt sind (z. B. auf etwas zeigen zum interagieren). Des Weiteren ist auch die menschliche Lernbereitschaft und Lernkurve im Bereich dieser MMI zu beachten. Der Spagat zwischen Herausforderungen („challenges“) und den eigenen Fähigkeiten („skills“) muss zwischen Angst („anxiety“) und Langeweile („boredom“) liegen.

„Intuitive Aktionen“ sind die, bei denen man als Nutzer automatisch das Richtige tut, getriggert durch Vertrautheit und die Gestaltung der Interaktion (englisch oft als

„affordance“ bezeichnet). Das bedeutet, ein Nutzer kann ein Produkt verwenden, indem er unbewusst Vorwissen anwendet. Je nach Nutzergruppe variiert dieses Vorwissen und somit auch die Intuitivität des Nutzungserlebnisses. Eine Möglichkeit hierbei ist eine *konsistente Gestaltung* der Interaktionsstruktur, wodurch erlerntes Wissen automatisch übertragen werden kann. Diese Konsistenz kann auch Einzug in die Virtualität erhalten, welche reale Interaktionsstrukturen zum Vorbild nimmt. Sogenannte „Natural User Interfaces“ sind eben die Benutzerschnittstellen, die den Nutzer mit virtuellen Elementen genauso interagieren lassen, wie dieser es mit realen tun würde.

Der Psychologe Don Norman (siehe Abb. 2.8b) teilt menschliches Handeln auf die drei Bewusstseinsebenen „visceral“, „behavioral“ und „reflective“ ein und sagt dabei, dass sich eine Aktion umso intuitiver ausführen lassen sollte, je unbewusster sie ist. Aktuelle VR-Systeme zeigen (siehe Abb. 2.8a): Je komplexer die VR-Systeme aufgebaut sind, desto größer ist das Präsenzgefühl der Spieler und somit werden die durchgeführten Aktionen unbewusster.



(a) *Komplexität und Illusion der Präsenz der VR-Systeme [53]*

(b) *Basierend auf Don Normans „The Design of Everyday Things“ [54]*

Abb. 2.8: *Virtual Reality (VR): Komplexität, Präsenz und Interaktion*

Aufgrund der o. g. Herausforderungen, eine „optimale“ und „intuitive“ MMI zu garantieren und zu erstellen, wird im Folgenden gezielter auf die verschiedenen Systeme und Hardware eingegangen, welche jeweils andere Möglichkeiten bieten, eine optimale Schnittstelle zu kreieren.

2.3.3 User Interfaces: systemtheoretischer I/O-Ansatz

In diesem Kapitel wird der systemtheoretische Ansatz der I/O-Geräte angewandt und die unterschiedlichen Klassen von Ein- und Ausgabegeräten vorgestellt. Ergänzende Informationen zu Ein- und Ausgabegeräten sind in Anhang 7 zu finden.

2.3.4 Eingabegeräte

Nun werden aktuelle Interaktionsgeräte vorgestellt. Genauer klassifiziert und eingeordnet werden diese dann bei der jeweiligen Implementierung in Kapitel 4. Klassifiziert werden die Eingabegeräte im Folgenden nach: klassisch (K), controllerbasiert (CB), optisch (O), haptisch (H), virtuell (V) und „neuartig“ (N).

Im Folgenden werden kurz controllerbasierte, optische, haptische und virtuelle Eingabegeräte eingeführt, wohingegen klassische Eingabegeräte wie Tastatur, Maus, SpaceMouse [g] als bekannt vorausgesetzt werden und die Klasse der „neuartigen“ Eingabegeräte alle solche beinhaltet, die in keine der anderen Kategorien einzuordnen sind.

2.3.4.1 Controllerbasierte Eingabegeräte



(a) *Nintendo Wii* ***(b)*** *Thalmic Labs Myo* ***(c)*** *HTC Vive Controller*

Abb. 2.9: *Controllerbasierte Eingabegeräte: eine Übersicht*

Das bekannteste controllerbasierte Eingabegerät stellt die Nintendo Wii-Fernbedienung dar (vgl. Abb. 2.9a). Aufgrund der Tatsache, dass die exakten Spezifikationen offiziell nicht veröffentlicht wurden, ist diese in der Wissenschaft nur reverse-engineered zu gebrauchen, wie z. B. in [55].



(a) *Myo-Armband [h]*

(b) *Myo-Gesten (vlnr): ThumbToPinky (TtP), FingersSpread (FS), WaveIn (WI), WaveOut (WO), Fist (F) [i]*

Abb. 2.10: *Thalmic Labs Myo-Armband*

Das *Myo* „Gesture Control Armband“ von Thalmic Labs (erschienen im Jahr 2014) ist ein Armband, das es Entwicklern erlaubt, Arm- und Handgesten ihres Trägers ohne optische Verbindung zu erkennen (vgl. Abb. 2.9b). Ein zusätzlicher Vibrationsmotor dient als Feedback. Die Neuheit ist dabei, dass das Myo-Armband nicht nur eine eingebaute Inertial-Measurement-Unit-(IMU-)-Sensorik mit neun Freiheitsgraden besitzt und die Bewegung über Gyroskop, Beschleunigungsmesser und Magnetometer misst, sondern auch die Muskelaktivität im Arm (und damit auch in der Hand) in Form von Elektromyogramm-(EMG-)Signalen erkennen kann (vgl. Abb. 2.10). Hierbei werden die Summenaktionspotentiale der Muskeln bei unterschiedlichen Aktionen gemessen, um die Muskelaktivität in Form von Spannungsänderungen aufzuzeichnen. So lassen sich Anwendungen auch durch Arm- oder sogar Fingergesten steuern.

Ein weiteres Beispiel für controllerbasierte Systeme ist die im April 2016 offiziell in den Markt eingeführte HTC Vive (siehe auch Abb. 2.9c). Sie besteht aus einer Brille, zwei Controllern sowie zwei Lighthouse-Sensoren und sie wird vor allem für Spiele eingesetzt und dient somit der Unterhaltung. Die Brille ist hierbei für eine visuelle Simulation einer virtuellen Welt zuständig, die beiden Controller dienen der Simulation von Bewegungen und Aktionen in dieser virtuellen Welt. Sie sind ein weitergehendes Konzept der Nintendo Wii. Die Positionsbestimmung der HTC Vive und somit auch der HTC Vive Controller erfolgt hierbei durch das Trackingverfahren Lighthouse. Dieses verfolgt ein mehrschrittiges Schema aus (1) Infrarot-(IR-)LED-Blitz, (2) horizontalem Abrastern des Raumes mit einem Laserstrahl, (3) einem weiteren IR-LED-Blitz und (4) vertikalem Abrastern des Raumes mit einem Laserstrahl. Die aktiven Sensoren (an Brille und Controllern) berechnen aus der zeitlichen Differenz zwischen Blitz und eintreffendem Laserstrahl ihre eigene Orientierung zu den Basisstationen. Um den ganzen Raum abzudecken, werden dabei häufig zwei Basisstationen benutzt. Des Weiteren werden zusätzlich noch ein Beschleunigungssensor zur besseren Positionsbestimmung sowie ein Gyroskop zur Bestimmung der Lageänderung verwendet.

2.3.4.2 Optische Eingabegeräte



(a) *Leap Motion* ***(b)*** *Microsoft Kinect v2* ***(c)*** *Google Soli [j]*

Abb. 2.11: *Optische Eingabegeräte: eine Übersicht*

Optische Eingabegeräte nutzen externe Sensoren z. B. in Form von Kameras. Dabei werden entweder mehrere normale Kameras oder sogenannte 3D-Kameras eingesetzt (siehe auch Abb. 2.12). Bei den 3D-Kamera-Systemen handelt es sich um Time-of-Flight-(TOF-)Kameras oder um Structered-Light-Kameras, welche sich die physikalischen Eigenschaften von Licht zunutze machen. Diese Kameras emittieren Lichtpulse mit der Lichtpulsdauer t_D und sind mit sogenannten PMD-Sensoren (Photonic Mixing Device) ausgestattet, welche die Entfernung zwischen Kamera und Objekt (D) berechnen.

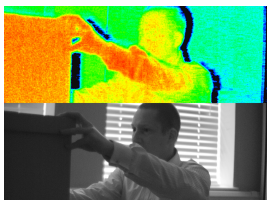
$$t_D = 2 \cdot \frac{D}{c_{Luft}} \tag{2.1}$$

$$\Rightarrow D_{max} = \frac{c_{Luft} \cdot t_0}{2} \tag{2.2}$$

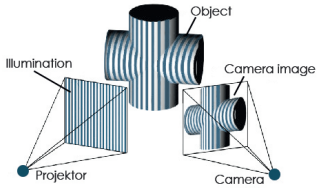
Somit ist der Distanzbereich D_{max} direkt von der Pulslänge t_0 abhängig (Lichtgeschwindigkeit in Luft $c_{Luft} \approx 3 \cdot 10^5 m/s$).

Mittels dieser Kameras ist ein dreidimensionaler Scan des Raumes möglich, mit dem man dann ein Tiefenbild des Raumes erhält. Somit kann dann die Orientierung von

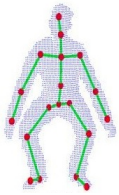
Personen im Raum berechnet werden. Mittels verschiedener Trackingmodalitäten lassen sich so auch Gesten durch verschiedene Verfahren identifizieren. Genauer betrachtet wird die Gestenerkennung in Kap. 2.3.4.5.



(a) Time-of-Flight-Kamera Tiefenbild [k]



(b) 3D-Messung anhand der Deformation strukturierten Lichtes an 3D-Objekten [l]



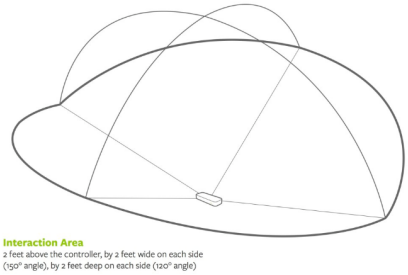
(c) Visualisierung der Struktur aus Gelenkpunkten und Knochen in Form eines menschlichen Skeletts [56]

Abb. 2.12: *Kamera bzw. Bildtypen für Systeme mit externer Sensorik*

Der *Leap Motion Controller* (vgl. Abb. 2.11a) besitzt zwei Infrarot-sensitive Weitwinkelkameras und drei IR-LEDs, die ein Volumen von circa $V \approx 60\text{cm}^3$ rund um den Sensor beleuchten und abtasten. Spezielle Analysesoftware erfasst dabei Hände und Finger mit hoher Genauigkeit.



(a) Explosionsplot



(b) Arbeitsraum

Abb. 2.13: *Leap Motion Controller[m]*

Die Leap Motion besteht im Grunde aus zwei Weitwinkelkameras (längs $\alpha = 150°$ und $\beta = 120°$ quer) und drei infraroten LEDs ($\lambda = 850nm$) (vgl. Abb. 2.13). Über einen Rahmen, der den Sensor mit einem Head-Mounted Display (HMD) verbindet, besteht die Möglichkeit, dass der HMD-Benutzer seine eigenen Hände in der virtuellen Welt sehen kann – ein entscheidendes Kriterium für die geforderte Immersion in virtuellen Welten und VR. Nachteile sind derzeit, dass der Leap Motion Controller empfindlich für aus der Umgebung einfallendes oder reflektierendes Infrarotlicht ist und stets freie Sicht auf die Hände benötigt.

Die *Kinect v2* (vgl. Abb. 2.11b) von Microsoft als Eingabegerät bietet Möglichkeiten des Ganzkörper- oder „Full Body“-Trackings. Neben einem RGB-Bild in 1080p Auflösung kann sie ein IR-Bild mit 512×424 Pixeln aufnehmen und eine Tiefenmaske in der gleichen Auflösung erstellen. Mit einer Wiederholfrequenz von 30 Hz lassen sich damit

automatisiert Skelette (vgl. Abb. 2.12c) für bis zu sechs Personen im Umkreis von 0, 5 bis 4, 5 Metern tracken. Neben der Nutzung als Eingabegerät ist die *Kinect v2* auch ein sehr guter (und vor allem günstiger) Sensor im Bereich der (mobilen) Robotik. Die Microsoft *Kinect for XBox One* oder auch *Kinect v2* ist die Weiterentwicklung der Microsoft *Kinect for XBox 360* oder auch *Kinect v1* (vgl. Tab. 2.1).

Tab. 2.1: *Microsoft Kinect-Daten: Vergleich von Kinect v1 und v2*

Feature	Kinect v1	Kinect v2
RGB-Kamera [pxls]	640 × 480 @ 30 fps	1920 × 1080 @ 30fps
ToF-Kamera [pxls]	320 × 240	512 × 424
d_{max} [m]	4,5	4,5
d_{max}^{near} [m]	0,4	0,5
FOV_h [°]	57	70
FOV_v [°]	43	60
# Skelett-Gelenke	20	26
# Skelette	2	6
USB-Standard	2,0	3,0
Tiltmotor	ja	nein

Die allgemeine Technologie der Kinect wurde erstmals im Juni 2009 auf der E3 unter dem Codenamen Project Natal vorgestellt, im Oktober 2017 wurde die Produktion der Kinect v2 eingestellt. In der Zwischenzeit haben sich die Kinects sowohl in der Consumer-Elektronik als auch in Forschung und Entwicklung als kostengünstige Hardware etabliert und werden in vielen wissenschaftlichen Arbeiten genutzt. Im Folgenden ist immer die Kinect v2 gemeint, auch wenn nur allgemein über die Kinect geredet wird. Die Kinect benutzt zwei Tiefensensoren, von denen einer einen großflächigen Infrarotstrahl in den Raum aussendet und der andere die Infrarotstrahlen empfängt, die von den Objekten im Raum reflektiert werden. Dadurch kann das Gerät die Entfernungen der Objekte im Raum berechnen und ein dreidimensionales Abbild des Nutzers erzeugen. Anschließend werden Punkte am Körper (Kopf, Hals, Unterleib, Knie und Füße) des Nutzers definiert, die während der jeweiligen Anwendung in ständiger Beobachtung stehen. Durch die Verfolgung dieser Punkte wird die Bewegung des Nutzers erfasst und die entsprechende Bewegung in der Anwendung mit der Kinect ausgeführt. Darüber hinaus verfügt die Kinect über ein Mikrofonarray, das mittels Lautstärke die Entfernung der Nutzer zu dem Gerät feststellen und zwischen den verschiedenen Nutzern unterscheiden kann.

Die Kinect wird als Sensor insbesondere auf die Vor- und Nachteile bei Verwendung von strukturiertem Licht gegen eine TOF-Kamera in verschiedenen Forschungsbereichen intensiv getestet [57]. Die Kinect als Sensor (Kamera und Tiefenkamera) wird im Bereich der Kartengenerierung in verschiedensten Applikationen eingesetzt [58] [59] [60] [61], vor allem im Bereich mobiler Systeme. Aber auch als Eingabegerät zum Körper- und Handtracking wird mit der Kinect in verschiedenen Anwendungsbereichen geforscht [62] [63], sehr intensiv im Bereich der medizinischen Rehabilitation.

Das in der Entwicklung befindliche *Soli* von Google ist ein miniaturisierter Sensor, der

Radarwellen nutzt, um Handbewegungen zu erfassen (siehe Abb. 2.11c). Ausgesendete Radarwellen des Sensors werden an Objekten gestreut, sodass das zurückgestreute Signal genutzt werden kann, um auf Charakteristika des Streuobjektes rückschließen zu können. Parameter hierfür sind Signalstärke, Energie, Zeitverzögerung und Frequenzshift.

Gerade optische Eingabegeräten unterstützen die sog. *„come as you are"-Philosophie*, derzufolge der Nutzer ohne weitere Hardware das System bedienen kann. Diese direkte, natürliche und intuitive Art der Interaktion ist immer wünschenswert, jedoch erweist sich zusätzliche Hardware für Ein- und Ausgabe oft als einfacher zu implementieren.

2.3.4.3 Haptische Eingabegeräte

Neben visuellen und akustischen Ausgaben beschäftigen sich Forscher, Startups und große internationale Unternehmen mit dem Einbeziehen weiterer Sinneseindrücke wie insbesondere die Kraftrückkopplung (Force Feedback (FF)).



(a) *Geomagic Touch X* ***(b)*** *Exoskelett* ***(c)*** *Motion Simulator*

Abb. 2.14: *Haptische Eingabegeräte: eine Übersicht*

Die haptische Wahrnehmung umfasst folgende Wahrnehmungsbereiche: haptische Sensitivität (Oberflächensensitivität), Propriozeption (Tiefensensibilität), Kinästhesie (Tiefensensibilität), Viszerozeption (Organtätigkeitswahrnehmung), Nozizeption (Schmerzwahrnehmung) sowie Thermozeption (Temperaturwahrnehmung). Im Bereich haptischer Eingabegeräte beschränkt sich die „Haptik" auf die wörtliche Definition dieses Begriffes, also das „Greifbare" bzw. „den Tastsinn betreffende" Funktionen. Solche haptischen Eingabegeräte werden normalerweise nicht in der 3D-Simulation genutzt. Überwiegend werden sie als dreidimensionales Eingabegerät für Modellierungsaufgaben eingesetzt, nur in sehr wenigen Anwendungen wie chirurgischen Simulatoren wird mit Force Feedback als zentralem Bestandteil des Systems gearbeitet. Die Integration von Force Feedback in ein starrkörperbasiertes Simulationsframework stellt somit eine Neuerung und einen Fortschritt aktueller Technologien dar. Solche haptischen Eingabegeräte können die Immersion in virtuelle Welten weiter voran treiben und verbessern.

Anders als bei der direkten Interaktion mit realen Objekten beeinflusst das haptische Eingabegerät das Benutzerempfinden. Um auftretende Verfälschungen zu minimieren, formulieren Massie und Salisbury in [64] drei Anforderungen an ein haptisches System, um dem Benutzer ein realistisches Gefühl der Umgebung zu geben:

1. Der freie Raum muss sich frei anfühlen.
2. Feste virtuelle Objekte müssen sich fest anfühlen.
3. Virtuelle Bedingungen sollen nicht leicht überschreitbar sein.

2.3.4.3.1 Force Feedback

Force Feedback ist ein Element des haptischen Feedbacks, das Kraft-Feedback, taktiles Feedback und propriozeptives Feedback umfasst [65]. Obwohl die Teleoperation vor allem auf audiovisuellem Feedback basiert, beginnt auch Force Feedback mehr und mehr Anwendungen zu finden. Mehrere Produkte mit Kraftrückkopplung sind im Handel erhältlich, insbesondere das 6 DOF *Geomagic Touch X* (auch PHANToM Device)[n] als das gängigste Modell. Besonders wichtig ist die benötigte hohe Updaterate bzw. Frequenz von

$$f_{FF} > 1\text{kHz} \tag{2.3}$$

Im Gegensatz zu Abtastraten beim graphischen Rendern, die ungefähr bei $f_R \approx 30$ Hz liegen, ist beim haptischen Rendern die Echtzeitfähigkeit unabdingbarer Bestandteil und benötigt den o. g. 1-kHz-Updateloop. Ein allgemeiner Überblick über die Geschichte, die Komplexität und den Nutzen haptischer Schnittstellen ist in [66] zu finden. Aus technischer Sicht sollte die Schnittstelle zwischen Software und (beliebigem) FF-Gerät gleich sein: „FF-Schnittstellen können als Computer-Erweiterungen angesehen werden, die physikalische Kräfte und Drehmomente auf den Benutzer anwenden.“ [65]

Haptische Eingabegeräte beinhalten neben dem dreidimensionalen Inputframe $\mathbf{P}_{in}(\boldsymbol{p}, \mathbf{R})$ auch eine Rückstellkraft $\mathbf{F}_{out}$. Für ein reibungsfreies Update wird hierbei, auf Basis der Grenze des menschlichen Tastsinns, eine Frequenz von $f = 1$ kHz benötigt.

Der *virtuelle Proxy* (engl. „Stellvertreter“) wurde erstmals in [67] eingeführt und stellt eine Abwandlung des Gott-Objektes dar, wie in [68] beschrieben. Diese Methode wird eingesetzt, um Kollisionen zwischen der Geräteposition, die in der Simulation zunächst nur als Punkt repräsentiert wird, und allen anderen Objekten zu detektieren. Dazu nutzt die virtueller-Proxy-Methode zwei Objekte, welche miteinander verknüpft sind: das *haptische Objekt* und den *Proxy*. Das haptische Objekt stellt die Geräteposition in der Simulation dar. Es kann frei im Raum bewegt werden und wird nicht am Eindringen in andere Objekte gehindert. Der Proxy hingegen ist ein dynamisches Objekt. Er stellt den physikalischen Finger in der Simulation dar und wird üblicherweise durch eine masselose Kugel repräsentiert. Das Ziel des Proxys besteht darin, die Distanz zur Geräteposition durch lineare Bewegungen zu minimieren. Anders als das haptische Objekt kann der Proxy nicht durch Oberflächen von anderen dynamischen Körpern dringen und interagiert mit den berührten Objekten.

In Abb. 2.15 ist das Verhalten des Proxys illustriert. Grundsätzlich folgt der Proxy der Bewegung des Eingabegerätes. Im freien Raum fallen die Positionen der beiden Objekte zusammen (Abb. 2.15a). Dringt das haptische Objekt durch die Oberfläche eines anderen Körpers, wird dieses Hindernis erkannt. Da der Proxy nicht in das Objekt eindringen kann, verbleibt dieser an der Oberfläche. Während das haptische Objekt sich innerhalb des Körpers befindet, wird der Proxy auf der Oberfläche bewegt, um den Abstand zum haptischen Objekt zu minimieren (Abb. 2.15b, c). Dazu wird eine Kostenfunktion eingesetzt, die außerdem verhindert, dass ein Objekt durchdrungen wird, selbst wenn sich das

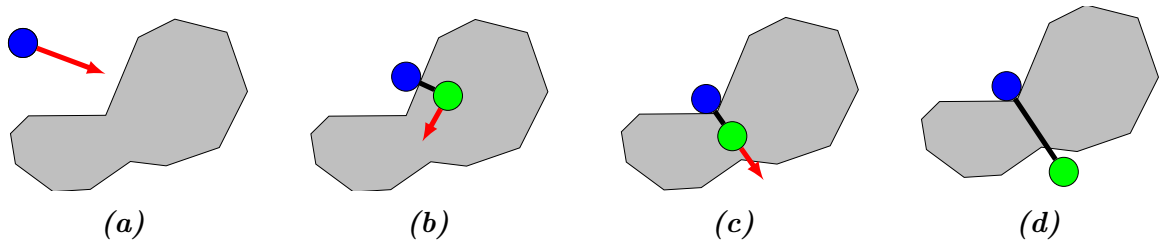


Abb. 2.15: *Beispielhaftes Verhalten des virtuellen Proxys (blau) und des haptischen Objektes (grün), basierend auf [67], zu verschiedenen Zeitpunkten (a) bis (d)*

haptische Objekt auf der anderen Seite des berührten Körpers befindet (Abb. 2.15d).

2.3.4.3.2 FF Devices

Das *Geomagic Touch X* [n] (vgl. Abb. 2.14a) der Firma Sensable (auch bekannt als *Phantom Device*) ist ein stiftbasiertes Eingabegerät, welches auf den drei Hauptachsen auch Force-Feedback-Funktionalitäten mitbringt und gerne als Modellierungswerkzeug im Bereich vom System Design genutzt wird.

Neben solchen kommerziell erhältlichen Produkten sind *Exoskelette* Gegenstand aktueller Forschung (vgl. Abb. 2.14b). Im Folgenden werden ein paar Beispiele solcher Exoskelette aufgezeigt. Das Exoskelett *ALEx*, entwickelt an der Sant'Anna School of Advanced Studies (SSSA) in Pisa (IT), wurde zunächst für die therapeutische Anwendung entworfen, wird jedoch im Rahmen des CENTAURO-Projektes (siehe Kap. 5.1) auch als bilaterales Eingabegerät zum Manövrieren mobiler Roboter genutzt. Reine Hand-Exoskelette zum manuellen Ertasten, Erfühlen und Greifen sind momentan aktuell wie nie zuvor, jedoch sind sie keineswegs neu. Aufbauend auf früheren Entwicklungen, wie Datenhandschuhen wie dem *CyberGlove* [o] oder Mensch-Maschine-Schnittstellen wie dem *ExoHand* [p] von Festo, sind aktuelle Beispiele wie *VRGluv* [q], *CyberGrasp* [r], *Wolverine* [s] oder *Dexmo* [t] mehr auf die Unterhaltungselektronik ausgelegt.

Ganzheitliche FF-Systeme wie z. B. *AxonVR* [u] oder *Virtuix Omni* [v] sind mehr oder weniger auf eine bessere Mobilität in Computerspielen ausgelegt und nicht auf die Verbesserung der Force-Feedback-basierten Eingabegeräte.

Ein besonderes Exemplar solcher haptischer Eingabegeräte mit Kraft- und Bewegungsfeedback ist der *Motion Simulator* des Instituts für Mensch-Maschine-Interaktion (MMI) der RWTH Aachen University (vgl. Abb. 2.14c). Basierend auf einem *KUKA KR 500-2* Industrieroboter sitzt der Nutzer hier in einer Kapsel, die an den Tool Center Point (TCP) des Roboters montiert ist. Mit Hilfe eines hemisphärischen Bildschirms werden Bildinhalte in dieses Force-Feedback-Gerät projiziert.

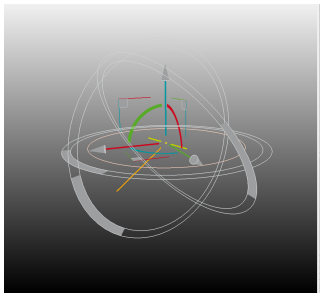
2.3.4.4 Virtuelle Eingabegeräte

Virtuelle Eingabegeräte, oder auch 3D User Interface (3DUI) genannt, sind eine weitere Klasse von Eingabegeräten, die inspiriert sind von allen anderen zuvor beschriebenen Klassen (siehe Def. 5).

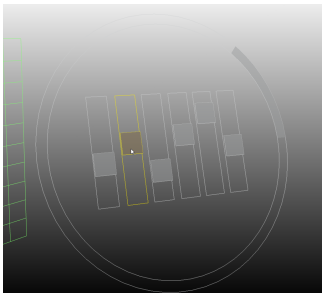
Definition 5: 3D User Interfaces (3DUI)

3DUI umfassen die Klasse der virtuellen Eingabegeräte. Oft mit realen Vorbildern sind dieses rein virtuelle Interaktionsmetaphern wie Knöpfe (Buttons (en)), Schieberegler (Slider (en)) usw., die im Rahmen eines Interaktionskonzeptes bedient werden können.

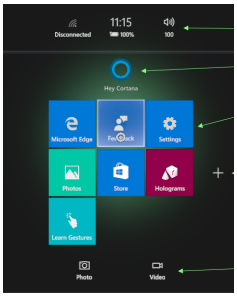
Solche virtuellen Eingabegeräte werden zum einen bei optischen Eingabegeräten mit im Software Development Kit (SDK) ausgeliefert, können aber auch komplett unabhängig vom verwendeten realen Eingabegerät implementiert werden. Beispiele für solch unabhängige virtuelle Eingabegeräte sind in Abb. 2.16a sowie Abb. 2.16b abgebildet, wohingegen in Abb. 2.16c das integrierte Interaktionspanel der *HoloLens* von Microsoft gezeigt wird.



(a) *Interaktiver Marker*



(b) *Button/Slider Board*



(c) *HoloLens „Bloom“[w]*

Abb. 2.16: *Virtuelle Eingabegeräte: eine Übersicht*

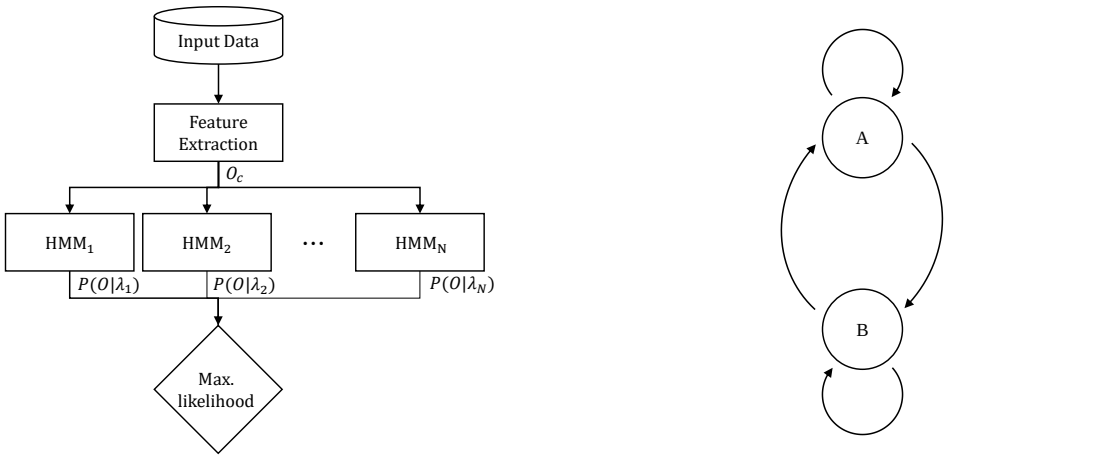
2.3.4.5 Gestenerkennung

Eine Geste ist eine Bewegung (meist des menschlichen Körpers), welche Informationen enthält. Bei der Gestenerkennung geht es daher um die Umwandlung dieser Informationen in nutzbare Informationen. Da dies unabhängig vom Eingabemedium ist, umfasst das Feld der Gestenerkennung alle zuvor genannten Arten von Eingabegeräten (sowohl controllerbasiert, optisch, haptisch, als auch virtuell), insbesondere aber optische. Die Erkennung einer Geste beschäftigt sich mit der *Bewegung* des Eingabegerätes. Im Bezug auf optische Eingabegeräte umfasst dies die Bewegung des Menschen und somit die Position, Richtung und Schnelligkeit der Bewegung eines Fingers, einer Hand, des Kopfes oder des ganzen Körpers. Eine erkannte Geste kann daraufhin genutzt werden, um weitere Aktionen oder Prozesse zu triggern, also zur Gestensteuerung (Kap. 2.3.4.5.1).

Die Verfahren zur Gestenerkennung lassen sich allgemein in *modellbasierte* (pattern based) und *nicht-modellbasierte Verfahren* (non-pattern based) unterteilen.

Bei modellbasierten Verfahren wird der menschliche Körper durch geometrische Formen angenähert und die Körperhaltung geschätzt. Dabei werden die Körperteile als Geradensegmente beschrieben und sind durch Gelenke miteinander verbunden, beispielsweise in Form von Skeletten von Körper und Händen (vgl. Abb. 2.12c). Bei nicht-

modellbasierten Verfahren wird keine Schätzung vorgenommen, sondern es werden einzelne Körperteile, wie Hände und Kopf, lokalisiert und analysiert. Modellbasierte Verfahren sind ein dreistufiger Prozess [69][70]. Nach der *„Data Acquisition“* wird in der *„Feature Extraction“* das Signal im Zeit- oder Frequenzraum analysiert, bevor in der *„Classification“* mit Verfahren von Principal Component Analysen (PCA) [71], Bayes Netzwerken [72], Hidden Markov Modellen (HMM) [73] und Kalman Filtern [74] oder aber auch Neuronalen Netzen [75] sowie Fuzzylogik [76] die Gesten klassifiziert werden. Die nicht-modellbasierten Verfahren hingegen nutzen die Rohdaten, um damit Finite State Machine (FSM) zu entwickeln oder Onset-Analysen durchzuführen [77].



(a) *Gestenerkennung – Hidden Markov Model (HMM)*

(b) *Gestensteuerung – Finite State Machine (FSM)*

Abb. 2.17: *Integration von Gestenerkennung und Gestensteuerung*

2.3.4.5.1 Gestensteuerung

Die drei Phasen der Gestenerkennung (vgl. Kap. 2.3.4.5) *„Data Acquisition“*,*„Feature Extraction“* und *„Classification“* können hier um die Phase des *„Result“* ergänzt werden.

Im Rahmen der Integration der vorgestellten Eingabegeräte sind die Phasen der Gestenerkennung oft schon im jeweiligen SDK integriert und können sofort genutzt werden. Bei den hier vorgestellten optischen Systemen (Kinect und LeapMotion) werden verschiedene Pattern-Matching-Algorithmen benutzt, um die einzelnen Gesten herauszufiltern. Das Resultat einer erkannten Geste stellt dann normalerweise einen binären Ausgang dar, welcher daraufhin als Input bzw. Trigger genutzt werden kann und somit eine Finite-State-Maschine (FSM) bildet (vgl. Abb. 2.17b).

Auch die Klassifizierung mittels HMM auf Basis von Rohdaten ist hierbei möglich. Hierbei wird eine Gestentrajektorie vorgegeben und mit den aus den Datenpunkten interpolierten Vektoren verglichen. Diese Vektoren werden in der „Feature-Extraktion“ uniform verteilten „Feature-Vektoren“ zugeordnet. Ein HMM kann dann für jede Geste ein Modell mit Hilfe des „Baum-Welch-Algorithmus“ trainiert werden. Die Klassifizierung erfolgt dann auf Basis der Ähnlichkeit (P) zwischen Input-Daten (Observation Sequence

O_c) und Modellen (vgl. Abb. 2.17a).

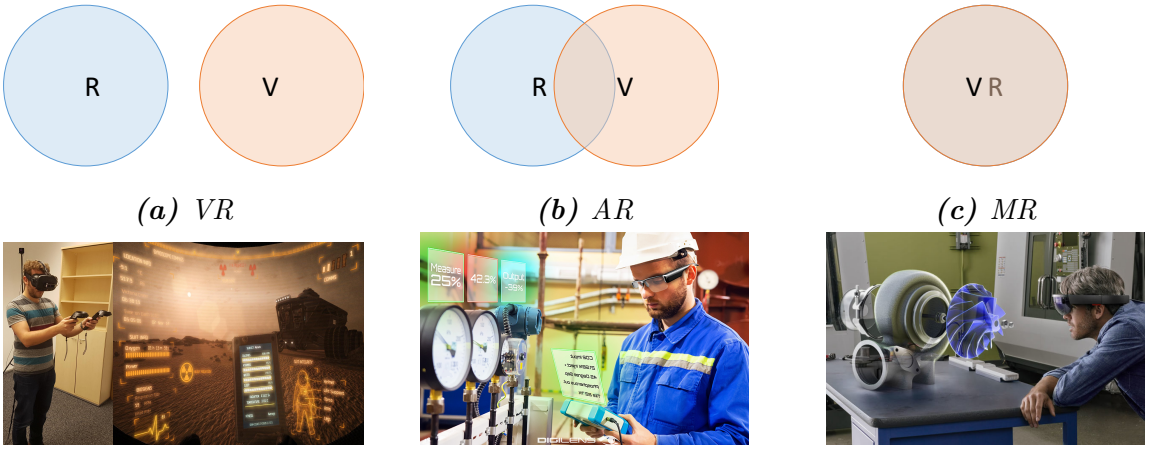
Zusätzlich dazu wird bei der Kopplung von haptischen Eingabegeräten mit Gestenerkennung in Kap. 5.3.1 die haptische Kraftrückkopplung genutzt, um eine Geste zu erkennen, und daraufhin Finite-State-Machines (FSM) für die Gestensteuerung genutzt. FSM eignen sich dabei insbesondere dazu, (mehrere) erkannte Gesten zu koppeln oder zu kombinieren um einen Zustand zu wechseln.

2.3.5 Ausgabegeräte

Klassische Ausgabegeräte wie ein PC-Bildschirm geben dem Nutzer die Möglichkeit, genau das zu sehen, was dieser sehen möchte. Jedoch gibt es sowohl hardware- als auch softwareseitig Möglichkeiten, den Informationsfluss zu verbessern und einen intuitiveren Zugang zu ermöglichen. Dies beinhaltet zum einen neuartige Hardware in Form von (stereoskopischen) Head-Mounted-Displays (HMDs) und zum andern die visuelle Aufbereitung der Daten. Im Folgenden soll daher zunächst auf die Hardware und anschließend auf die Möglichkeiten im Softwarebereich eingegangen werden.

2.3.5.1 Die Virtuelle Realität (VR): VR/AR/MR

Unter Virtueller Realität versteht man die Einbettung (Immersion) des Nutzers in eine vollständig computergenerierte Welt, sodass eine Interaktion mit den dreidimensionalen Inhalten in Echtzeit möglich ist. Hierbei unterscheidet man Augmented Reality (AR), Mixed Reality (MR), und Virtual Reality (VR), welche jeweils ein unterschiedliches Maß an Überschneidung von digitalen und realen Inhalten bedeutet (siehe Abb. 2.18).



(a) *VR* ***(b)*** *AR* ***(c)*** *MR*

(d) *Opake VR-Welt ohne Sicht auf die Realität [x]*

(e) *Erweiterung der Realität mit 2D-Visualisierungen [y]*

(f) *Überlagerung der Realität mit semi-transparenten virtuellen 3D-Objekten [z]*

Abb. 2.18: *Überschneidung von digitalen/virtuellen (V) und realen (R) Inhalten*

Selbstverständlich lassen sich diesne auch passenden Hardwarekomponenten zuordnen, wie es in Anhang. 7 getan wird, jedoch sollte man beachten, dass es auch möglich ist, z. B. AR-Metaphern in VR zu integrieren.

Als günstiges und mittlerweile überaus leistungsfähiges Ausgabegerät bieten sich aktuell Head-Mounted Displays (HMD) unterschiedlicher Hersteller an. Mit zwei direkt vor den Augen platzierten hochauflösenden Displays und passenden Optiken, die das sehr nahe Pixelbild passend verzerren, befindet sich der Benutzer direkt in der virtuellen Welt, statt nur davor. Um die Illusion perfekt zu machen, ist es jedoch notwendig, die Bewegungen des Benutzers exakt zu erfassen und auf die virtuelle Kamera in der 3D-Welt zu transformieren. Das gilt besonders für die Kopfbewegungen – dreht der Nutzer seinen Kopf, muss auch auf dem Bildschirm vor dem Gesicht eine andere Perspektive gerendert werden. Dabei spielt unter anderem die Zeitverzögerung von der erfolgten Nutzerbewegung bis zur Darstellung der neu berechneten Pixelwelten auf dem Display eine wichtige Rolle. Die sogenannte Motion-to-Photon-Latenz sollte weniger als 20 ms betragen, damit sie der Benutzer nicht wahrnimmt.

Eine weitere Ebene virtueller Immersion stellt die Übertragung digitaler Elemente in die reale Welt dar, sodass virtuelle und reale Inhalte im Idealfall ununterscheidbar sind. Dieser Ansatz der Augmented oder Mixed Reality birgt besondere Herausforderungen: Das Erkennen der realen Welt sowie eine konsistente Kombination virtueller und realer Objekte sind dabei offensichtlich schwierig. Weiterhin spielt bei allen diesen Modalitäten die Visualisierung an sich eine zentrale Rolle und das Tracking sowie die Steuerung (z. B. in Form von Bewegungsplattformen) muss betrachtet werden.

2.3.5.1.1 Immersion und Präsenz

Ein weiterer wesentlicher Baustein für VR ist das Gefühl, eigene Entscheidungen zu treffen, frei zu agieren und die eigene Wirkung auf die virtuelle Umgebung möglichst unmittelbar zu spüren. Man könnte auch von „Agency“ sprechen, das sich als „Handlungsfähigkeit“ nur unzureichend ins Deutsche übersetzen lässt. Von abstrakten und komplizierten Benutzerschnittstellen sollte man daher Abstand nehmen. VR hat das Potenzial, sich viel stärker an natürliche Interaktionen anzulehnen und somit viele Elemente nicht mehr erklären zu müssen. Damit wird ein komplett neues Niveau von Benutzerfreundlichkeit möglich. Jedoch sollte man es nicht übertreiben. Gerade relativ belanglose Aktionen sollten dem Nutzer nicht zu viel Aufwand abfordern und auch die Bedienung mit Tastatur, Maus und Gamepad kann manchmal die beste Wahl darstellen.

Gerade im Bereich von VR ist insbesondere die Interaktion in einer dreidimensionalen computergenerierten Umgebung mit virtuellen Objekten möglich. Damit sich dies natürlich anfühlt, müssen die menschlichen Sinne angesprochen werden, um ein *immersives* Gefühl (siehe Def. 6) hervorzurufen. Der Nutzer solch einer VR-Brille taucht in die virtuelle Realität ein und schaut nicht nur auf sie herab, wie dies bei einem Monitor der Fall ist. Das Empfinden, wirklich Teil einer nicht realen Welt zu sein, die Immersion, wird im visuellen Bereich erst mit dieser neuen Technik wirklich möglich.

Definition 6: Immersion

Immersion oder das Eintauchen in eine meist virtuelle Welt beschreibt das Empfinden einer virtuellen Welt durch sensorische Stimuli als real. Zentral ist hierbei die Übereinstimmung (consistency (en)) von realen und virtuellen Daten.

Stimmen die verwendete Technik, die Darstellung und Interaktionsmöglichkeiten überein, so ermöglicht das menschliche Gehirn ein Gefühl der eigenen *Präsenz* (siehe Def. 7) in der Szene, tatsächlich in dieser Welt zu sein und sich in dieser zu bewegen und sie beeinflussen zu können. Der sogenannte Break in Presence (BIP) hingegen beschreibt Momente, in denen Nutzern ihre reale Umgebung begreiflich und das Gefühl der Präsenz in der VR zerstört wird. Folglich sollte dieser vermieden werden.

Definition 7: Präsenz

Präsenz ist das emotionale Gefühl, körperlich und geistig anwesend zu sein. Es ist somit die Fortsetzung bzw. das Ergebnis einer sehr guten Immersion.

Die Präsenz ist somit die Fortsetzung bzw. das Ergebnis einer sehr guten Immersion. Diese verstärkte Präsenz wiederum kann erschaffen werden, indem der eigene Körper mit dem virtuellen Körper übereinstimmt. Hierbei nutzt man das sogenannte „Embodiment“. Das *Embodiment* (siehe Def. 8) ist ein Hilfsmittel, die Immersion zu erhöhen und somit auch die Präsenz zu verstärken. In [78] wird dieser Sense of Embodiment (SoE) wie folgt beschrieben: „SoE toward a body B is the sense that emerges when B's properties are processed as if they were the properties of one's own biological body.“

Definition 8: Embodiment

Embodiment ist die direkte Übertragung des eigenen Körpers auf ein virtuelles (meist menschenähnliches) Objekt. Die „Verleiblichung“ ist im Bereich der VR wörtlich gemeint. Das Embodiment, also das Sehen und Fühlen des eigenen Körpers, ist ein Mittel zur Verstärkung der Präsenz. Diese beinhaltet das Umsehen (sense of location), die Interaktion (sense of agency) und den Körper (sense of body ownership).

Dieses Embodiment wird häufig durch sog. *Avatare* erreicht, also eine Repräsentation des realen Körpers in der virtuellen Welt. Ein Avatar ist dabei eine künstliche virtuelle Figur in Form eines Bildes, eines Icons oder auch einer 3D-Figur. Mit Hilfe von (Motion-)Tracking-Modalitäten können sich solche 3D-Avatare dann auch so bewegen wie der Nutzer. Die Akzeptanz solcher künstlichen Figuren hängt von der realitätsnähe des Avatars ab. Die Akzeptanzkurve steigt jedoch nicht stetig linear mit dem Anthropomorphismus der Figur, sondern erleidet einen Einbruch im sogenannten „Uncanny Valley“ [79] [80]. Dies gilt sowohl für Avatare als auch für humanoide Roboter. Neben dieser Genauigkeit der Abbildung hat auch die Art der Bewegung Einfluss auf die Akzeptanz.

2.3.5.2 Head-Mounted Displays (HMDs)

Die visuelle Ausgabe auf sogenannte Head-Mounted Display (HMD)s ist selbstverständlich auf der einen Seite beschränkt durch die technische Machbarkeit, jedoch auf der

anderen Seite auch durch die Möglichkeiten, Vorlieben und Grenzen des menschlichen Sehens. Selbstverständlich spielen hier die Graphik (Auflösung, Latenz, Augen-Tracking), die Haptik sowie die Audioausgabe eine essentielle Rolle.

Beginnend mit dem Sichtfeld hat der Mensch ein Field of View (FoV) von ca. 220° horizontal und ca. 120° vertikal, wobei sich dieses im Alter auch noch reduziert [81]. Die assoziierte Auflösung der Zellen im Sehorgan (z. B. in der „Fovea Centralis“) kann bei einem Visus von $v = 1' = \frac{1}{60}^\circ = 0.016^\circ$ auf ca. 6 Mio Zapfen (welche jeweils ein Pixel darstellen) zu 120 pxls/° approximiert werden. Des Weiteren ist bekannt, dass für flüssiges Sehen mindestens 24 Frames per Second (FPS) benötigt werden. Daraus folgt dann

$$220^\circ \times 120^\circ \times (120\text{pxls}/^\circ)^2 \times 24\ \text{FPS} \rightarrow 10\ \text{GPxls/s}$$

ein optimaler Pixelfluss von ca. 10 GPxls/s. Dies ist verbunden mit einer sog. „ultra low latency“-Verbindung, bei der die Updaterate mindestens 20 ms sein muss.

Für einen Überblick über passende Hardware sei auf den Anhang 7 verwiesen.

2.3.5.3 Visualisierung: Von opaker Virtualität zu transparenter Überlagerung der Realität

Im Bereich von VR/AR/MR steht vor allem das Verschmelzen von Realität und Virtualität im Vordergrund (siehe auch Abb. 2.18). Im Großen und Ganzen lässt sich dies auf reale und virtuelle Datenquellen reduzieren. Je nach Ausnutzung dieser Daten lässt sich eine rein virtuelle Welt (opak) oder eine Projektion und Verschmelzung von V und R erschaffen (transparent). Dies führt dann zur:

- Augmented Reality (AR) (Realität mit transparenten realen oder virtuellen Daten überlagert),
- Assisted Reality (AsR) (Realität mit opaken realen oder virtuellen Daten überlagert),
- Mixed Reality (MR) (Realität mit variablen transparent/opaken realen oder virtuellen Daten überlagert) sowie
- Augmented/Assisted Virtuality (AV/AsV) (Virtualität mit variablen transparent/opaken realen oder virtuellen Daten überlagert) und
- Virtual Reality (VR) (Virtualität mit opaken realen oder virtuellen Daten überlagert).

2.3.5.4 Visuelle AR-Metaphern

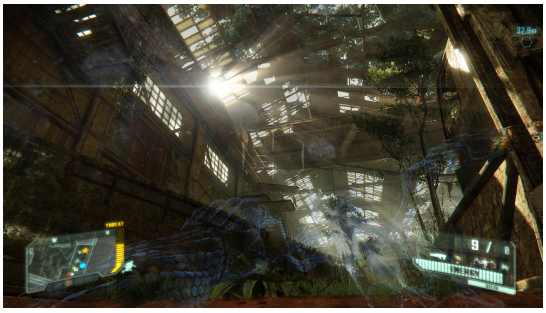
Ein weiterer Gestaltungsaspekt, der im Bereich VR/AR/MR besondere Beachtung finden sollte, sind User Interfaces. In klassischen Videospielen bestehen diese meist aus 2D-Elementen, die in der Peripherie des Sichtfeldes/Monitors auftauchen. Eine gute gestalterische Lösung hierfür ist, die Elemente der Benutzerfläche in die 3D-Welt einzubinden. Zum Beispiel bietet sich bei einer Flugsimulation an, das dreidimensionale Cockpit, in dem ein Spieler sitzt, interaktiv zu gestalten und sämtliche Interface-Elemente über diese Oberfläche bedienen zu lassen.

Aufgrund der Tatsache, dass die menschliche Aufnahmefähigkeit visueller Daten um einiges besser ist als ohne eine visuelle Komponente, macht es Sinn, solche visuellen AR-Metaphern zu nutzen. Diese können auch als *Scientific Visualizations* oder allgemeiner als

Parametervisualisierungen bezeichnet werden. Wichtig zu unterscheiden sind hier zwei Render- bzw. Projektions-Ebenen:

1. Head-Up-Display (HUD), also die Visualisierung als Visier,
2. Holographische Projektion (HP), also die Projektion in die 3D-Szene.

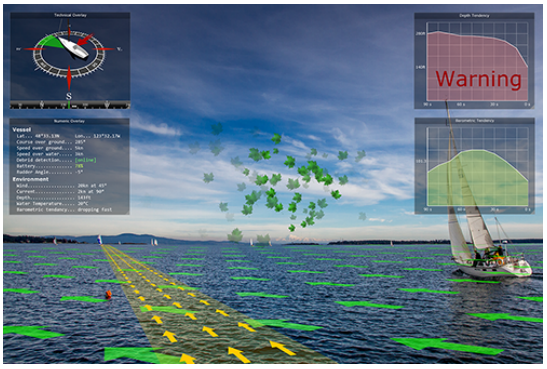
In Abbildung 2.19 kann man anhand einiger Beispiele diese Ebenen erkennen und die Nützlichkeit ableiten. Interne Parameter werden direkter und einfacher zugänglich gemacht, sodass die Entscheidungsfindung in einer Situation erleichtert wird.



(a) *AsV in PC-Spielen[aa]*



(b) *AsR in realer Umgebung[ab]*



(c) *AR mit zusätzlicher HP[ac]*



(d) *AR mit zusätzlicher HP[ad]*

Abb. 2.19: *HUDs (links) sowie HP (rechts) in unterschiedlichen Anwendungsszenarien und unterschiedlichen Ursprungs*

2.3.5.4.1 Head-Up-Display (HUD)

Als Überlagerung (Overlay) direkt im Blickfeld des Nutzers stellt das Head-Up-Display ein Anzeigesystem dar, welches als eine Art Visier-Metapher angesehen werden kann. Nutzen finden Head-Up-Displays primär im Cockpit von (Kampf-)Flugzeugen oder neuerdings auch im Automotive-Bereich Anwendung, damit der Nutzer den Kopf oben lassen kann, wie der Name es schon suggeriert. In solchen HUDs können dann sowohl 2D/3D-Daten als auch 3D-Modelle angezeigt werden. Wichtig hierbei ist die Verankerung aller Visualisierungen am Sichtfeld des Nutzers.

2.3.5.4.2 Holographische Projektion (HP)

Neben den Overlays auf dem Visier des Nutzers erhöht die Projektion der Information in die 3D-Szene die Immersion des Nutzers enorm. Durch die 3D-Projektion werden die Daten um die Informationen von Ort und Zeit erweitert (vgl. Def. 9).

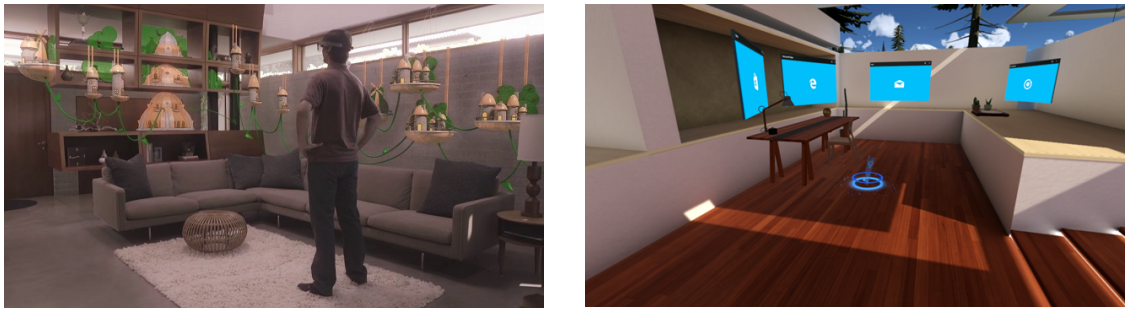
Definition 9: Holographische Projektion (HP)

Die visuelle Projektion von virtuellen Inhalten in eine 3D-Szene bezeichnen wir als holographische Projektion.

Innerhalb der HP ist die Bezeichnung der Art der Projektion von visuellen Informationen nicht klar definiert. Jedoch hat Microsoft als Vorreiter der MR-Technologie hierfür folgende Begriffe eingeführt [ae] (siehe auch Abb. 2.20):

- Immersive View
- 2D View

Wobei der „immersive view“ die 3D-Projektion von Daten und Objekten in die 3D-Szene beschreibt und der „2D View“ virtuelle Bildschirme (welche von Natur aus 2D sind) in die 3D-Szene einfügt. Im Rahmen dieser Arbeit werden diese Begriffe genutzt. Wichtig hierbei ist die ortsfeste Verankerung aller Visualisierung in der 3D-Szene.



(a) *Immersive View* ***(b)*** *2D View*

Abb. 2.20: *Projektions-Views: Microsofts Definition der Projektion von Informationen in eine 3D-Szene [ae]*

2.3.5.5 Kameraperspektiven

Neben der Projektion von Daten sind auch der Blickwinkel bzw. die Perspektive des Nutzers auf die Szene bzw. auf das System entscheidend. Hierbei kann man zwischen drei klassischen „Views“ unterscheiden:

1. First Person View (FPV),
2. Third Person View (TPV) / Over-the-shoulder View,
3. Free Camera View (FCV).

Diese aus der Film- oder Computerspieltechnik bekannten Kameramodi vermitteln unterschiedliche Einblicke in die Szene. Die Ego-Perspektive (oder FPV) gibt einen immersiven Eindruck, da man durch die Augen des virtuellen Alias sieht. Im TPV hingegen hat man das gesamte System und Teile der Umwelt im Blickfeld. Noch unabhängiger ist die frei positionierbare Kamera des FCV, die beliebig platziert und unabhängig vom System bewegt werden kann.

2.3.5.6 Avatare: Steuerungs- und Interaktionsmöglichkeiten

Der Begriff Steuerung beinhaltet hier die Steuerung des Nutzers durch die 3D-Umgebung der Simulation und somit eine Art „Avatar Control“.

Teleportation ist die klassische Steuerung eines Avatars in VR, da sie den inhärenten Vorteil bietet, sich physisch nicht bewegen zu müssen, um an einen anderen virtuellen Ort zu gelangen. Diese wird nativ von der mitgelieferten Software genutzt und ermöglicht eine intuitiv verständliche Manövrierfähigkeit.

Neben Teleportation ist beispielsweise der „Astral Body“ eine durchaus interessante Alternative. Dabei bleibt die Kamera Nutzers in einer 3rd-Person-Perspektive stehen, während er von seiner Position aus seinen eigenen Avatar, wie einen Geist, vorübergehend durch die Welt bewegt. Erst wenn dieser an seinem Ziel angekommen ist, springt die Kamera wieder in die Ich-Perspektive am neuen Standort.

2.4 Mediatoren in der Robotik

Wie schon in Def. 12 beschrieben, beinhaltet der Begriff der *Kooperation* immer eine gewisse Arbeitsteilung, um ein gemeinsames Ziel zu erreichen. Der Begriff der *Kollaboration* dagegen ist die gemeinsame Arbeit eines Teams an einem Teil der Zielsetzung. Kollaboration ist hier dann eine besondere Form der Zusammenarbeit. Hierbei erreicht die Kooperation parallel und die Kollaboration sequentiell gestellte Sub-Ziele.

Im Bereich der Robotik ist somit die Zusammenarbeit zwischen Mensch und Roboter gemeint, welche entweder nacheinander oder parallel durchgeführt wird (vgl. Kap. 2.4.1). Man kann hierbei die *Industrierobotik* und die *mobile Robotik* unterscheiden, da diese in Bezug auf die MRK unterschiedliche Prozesse, Herangehensweisen und Lösungen aufzeigen. Diese wiederum können autonom, semi-autonom oder teleoperiert (manuell) durchgeführt werden.

Im Rahmen der Industrierobotik haben sich vor allem zur Programmierung und zur Interaktion mit robotischen Systemen interessante Forschungsfelder aufgetan. Assistenzsysteme beschränken sich in diesem Fall jedoch primär auf (a) eine intuitivere Programmierung der Roboter oder (b) eine anleitende auditive bzw. visuelle Unterstützung des Benutzers. Weitaus spannender ist der Bereich der mobilen Robotik, in dem das Feld der Teleoperation sowie der direkten Interaktion mit robotischen Systemen Assistenzsysteme dringend notwendig gemacht hat. Hier ergeben sich zwei essentielle Besonderheiten. Zum einen – primär ein aktueller Trend in der Industrierobotik – gibt es eine direkte Interaktion vom Menschen mit robotischen Systemen, welche so „nachgiebig“ ausgelegt sein müssen, dass eine effiziente Kooperation möglich ist (vgl. Kap. 2.4.2). Zum anderen – primär aus dem Bereich der mobilen Robotik – beschäftigt sich die aktuelle Forschung mit der Fragestellung, wie bei der Teleoperation die Aufgabenteilung zwischen Mensch und Roboter aussehen kann (vgl. Kap. 2.4.3). Ein weiterer Fokus in diesem Kapitel ist neben der Schnittstelle zwischen Mensch und Roboter auch die Hardware-Schnittstelle der Roboter an sich (vgl. Kap. 2.4.4).

2.4.1 Interaktion, Kollaboration und Kooperation

Auch wenn die Begriffe Kollaboration und Kooperation oft als Synonym verwendet werden, soll hier eine genaue Trennung erfolgen. Diese Trennung ist sowohl *örtlich* als auch *zeitlich* zu verstehen.

Definition 10: Mensch-Maschine-Interaktion (MMI)

Mensch-Computer-Interaktion (MCI) (häufig auch als HCI bezeichnet) erforscht das Design und die Verwendung von Computer-Technologie an der Schnittstelle zwischen Menschen (Anwendern) und Computern. Ein übergeordnetes Gebiet ist die Mensch-Maschine-Interaktion (MMI) (oder Mensch-Maschine-Kommunikation (MMK), im Englischen Human-Machine Interaction (HMI) genannt), die sich mit ähnlichen Fragestellungen beschäftigt, aber den Interaktionspartner des Menschen zur Maschine verallgemeinert.

Definition 11: Mensch-Roboter-Kollaboration (HRC)

Unter dem Begriff „Kollaboration“ (lat. con- = „mit-“, laborare = „arbeiten“) wird in der Robotik die Zusammenarbeit zwischen Mensch und Roboter verstanden. Die Zusammenarbeit der Human-Robot Collaboration (HRC) beschränkt sich auf einen genau definierten gemeinsamen Arbeitsbereich. Eine Bearbeitung erfolgt dabei *sequenziell*.

Definition 12: Mensch-Roboter-Kooperation (MRK)

Bei der Human-Robot Cooperation bzw. Mensch-Roboter-Kooperation (MRK) arbeiten einzelne Personen, Maschinen oder Teams an unterschiedlichen Teilaufgaben des Endergebnisses. Die Bearbeitung erfolgt dabei *parallel*.

2.4.2 Industrielle Kollaboration von Mensch und Roboter

MRK im Bereich der Industrierobotik holt den „Standard-Roboterarm“ aus seinem „Käfig“. Dieser soll dem Menschen assistieren und mit ihm zusammen eine Aufgabe erledigen. Diese Zusammenarbeit beinhaltet verschiedene Anforderungen an diese Generation von „Cobots“ (Collaborative Robots):

- intuitiv und einfach bedienbar,
- nachgiebig und
- kooperativ.

Eine intuitive und einfache Steuerung bzw. Programmierung wird dabei oft durch das „Teachen“ per Hand erreicht. Teachen (oder auch Teach-in) bezeichnet dabei ein Verfahren zur Programmierung eines Roboters, wobei dieser zu unterschiedlichen Posen gesteuert wird und diese Punkte automatisiert zu einer Trajektorie verbunden werden, die der Roboter abfahren kann. Nachgiebige Roboter wie z. B. FRANKA EMIKA Panda [af] zeichnen sich dadurch aus, dass sie reflexartig auf mögliche Kollisionen reagieren und über ein menschenähnliches haptisches Gefühl verfügen, um mit dem Menschen zusammen arbeiten zu können Die Kooperation kann daraufhin durch geeignete Sensorik, Gesten- und Spracherkennung sowie intuitiv zu handhabende Programmierungstools erstellt werden. Hersteller wie KUKA (KUKA LWR iiwa), Festo (Greifer wie Octopus Gripper, Flex Shape Gripper, Festo-Elefantenrüssel) oder Rethink Robotics (Bionic Cobot) sind Vorreiter auf diesem Gebiet. Durch Material, Sensorik und Regelung werden die oben genannten Anforderungen hier schon weitestgehend erfüllt. Basis für all diese Cobots sind Leichtbauroboter, die im Folgenden für diese Klasse von Industrierobotern stellvertretend stehen sollen.

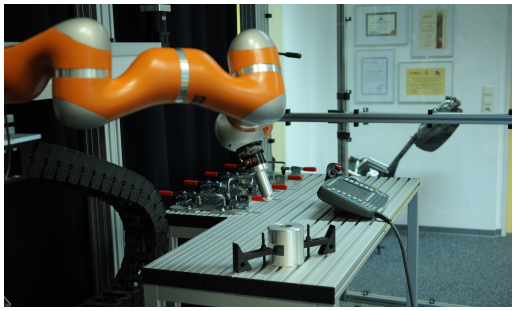
2.4.2.1 Leichtbauroboter

Eine intelligente Mensch-Roboter-Kooperation – also der Robotereinsatz im direkten Umfeld des Menschen – überspannt viele mechatronische Forschungsbereiche wie die Überwachung, Regelung, das Lernen sowie den Aufbau kooperierender Roboter.

Als Basis aller Cobots und „Soft Robots“ stehen Leichtbauroboter stellvertretend für die Klasse starrer robotischer Systeme, die durch Ausnutzen von Sensorik und Aktorik nachgiebig reagieren können (vgl. Abb. 2.21a).



(a) *FRANKA EMIKA Panda Roboter [af]*



(b) *KUKA LWR iiwa: Peg-in-Hole-Aufgabe*

Abb. 2.21: *MRK im industriellen Kontext*

Als klassische Aufgabe in der industriellen Fertigung hat sich das aktive Fügen (engl. *Peg-in-Hole*) als repräsentatives Beispiel für eine manuelle, kooperativ automatisierte MRK sowohl in der Industrie und auch in der Forschung herauskristallisiert (vgl. Abb. 2.21b). Hierbei kann sich der Roboter, als Lösungskonzept einer multisensoriellen Regelung, direkt am Verhalten des Menschen bei dieser Aufgabenstellung orientieren.

Das Beispiel der klassischen Peg-in-Hole-Aufgabe – also das Einführen eines Zapfens in eine Vertiefung – soll hier zur Verdeutlichung der Möglichkeiten der Leichtbauroboter genutzt werden. Roboter werden entworfen und programmiert, um Menschen nachzuahmen. Daher löst eine große Mehrheit der bestehenden Algorithmen die Aufgabe auf der Grundlage des von Menschen verwendeten Ansatzes, der sogenannten „Tilt-Strategie“. Der Verkippungsvorgang beruht darauf, den Zapfen schräg in das Loch einzuführen und dann um ihren Drehpunkt zu drehen, sodass ein vollständiges Einführen des Zapfens möglich ist. Der vollständige, vom Menschen angewandte Algorithmus lässt sich in drei Schritten zusammenfassen. Zuerst wird die ungefähre Lochposition visuell bestimmt. Der gekippte Zapfen wird dann in die ungefähre Lochposition bewegt, um Kontakt mit dem Loch aufzunehmen. Schließlich wird durch dynamisches, haptisches Feedback der Zapfen um seinen Drehpunkt rotiert, so dass er erfolgreich in das Loch eingeführt werden kann.

Ein ähnlicher Ansatz wird von Schaeffer et al. in ihren Arbeiten [82] für einen Roboter angewandt. Sie verwenden Kameras und Bildverarbeitungssoftware, um die Form und Position des Lochs zu bestimmen, die Neigungsstrategie anzuwenden und den Zapfen in das Loch einzusetzen. Die Autoren behaupten, dass durch Drücken und gleichzeitiges Einstellen der entsprechenden kartesischen Nachgiebigkeit der Zapfen automatisch mit dem Loch ausgerichtet wird.

Chatpar et al. [83] schlagen eine „blinde Suche“ der Lochposition vor. Beim Überstreichen des Zapfens auf der Fläche, auf der sich das Loch befindet, werden Geschwindigkeitsänderungen herangezogen, um die Lochposition zu ermitteln. Wiederum wird die Neigungsstrategie verwendet, um den Zapfen final einzuführen, wobei eine kontinuierliche Kraft mit einer geeigneten Nachgiebigkeit entlang einer Ebene verwendet wird. D. E. Whitney [84] konzentriert sich darauf, das Problem in ein geometrisches und ein Kraft-

gleichgewichtsproblem zu zerlegen und letztlich die gesammelten Informationen zusammenzuführen, um die Aufgabe erfolgreich zu erfüllen. Auch hier wird die Neigungsstrategie genutzt, um auf Basis der geometrischen Position des Zapfens und der aufgezeichneten Kontaktkräfte und Drehmomente den Zapfen in das Loch einzuführen. Dies verhindert das Blockieren und Verkeilen des Pegs und minimiert folglich die auftretenden Kräfte beim Einführen.

In [85] wird eine aktive Steifigkeitsregelung genutzt, die Kraftrückkopplung verwendet, um den Zapfen in das Loch einzuführen. Hogan [86] führte zuerst die Idee der Impedanzregelung (als eine Verallgemeinerung der Steifigkeitsregelung) ein. Die Impedanzregelung regelt die Nachgiebigkeit des Roboters. So kann die dynamische Reaktion des TCPs auf Kontakte und Kollisionen festgelegt werden. Eine solche Impedanzregelung kann vereinfacht als ganzheitliche Steuerung von Trägheit (m), Steifigkeit (d) und Dämpfung (k) interpretiert werden – also ein Feder-Dämpfer-Masse-System mit der Kraft $f(t)$.

$$f(t) = d \cdot x(t) + k \cdot \dot{x}(t) + m \cdot \ddot{x}(t) \tag{2.4}$$

Zur Bestimmung der zu kommandierenden Drehmomente (τ) des Roboters kann man dieses Verhalten in generalisierten Koordinaten wie folgt notieren:

$$\mathbf{M}\ddot{\boldsymbol{e}} + \mathbf{D}\dot{\boldsymbol{e}} + \mathbf{S}\boldsymbol{e} = \boldsymbol{F} \tag{2.5}$$

wobei $\boldsymbol{e}$ die Abweichung zw. Soll- und Ist-Pose des TCPs ist, $\boldsymbol{F}$ der Vektor externer Kräfte und Momente und $\mathbf{M}$, $\mathbf{D}$ und $\mathbf{S}$ die Matrizen für Trägheit, Dämpfung und Steifigkeit des Roboters sind. Um die Nachgiebigkeit des Roboters zu erreichen, werden diese Kräfte $\boldsymbol{F}$ über die Jacobi-Matrix $\mathbf{J}$ des TCPs (abhängig von der Gelenkstellung $\boldsymbol{q}$ der N Gelenke) auf die Gelenkmomente projiziert und ergeben mit der Gravitationskompensation ($\boldsymbol{g}(\boldsymbol{q})$) die benötigten Drehmomente $\boldsymbol{\tau}$, die auf die Gelenke des Roboters aufgebracht werden müssen:

$$\boldsymbol{\tau} = \boldsymbol{g}(\boldsymbol{q}) + \mathbf{J}^T(\boldsymbol{q})\boldsymbol{F} \tag{2.6}$$

Ergänzende Informationen hierzu sind in [87] und [88] zu finden.

Für die finale Peg-Einführung ist eine Verringerung der Steifigkeit orthogonal zur Einführrichtung hilfreich, da sich der Zapfen so automatisch an der Lochachse ausrichtet. In Broenink et al. [89], Gershon et al. [90] und Goldenberg [91] sind Beispiele zu finden, die das Einfügen des Zapfens durch verschiedene Variationen dieser Regelung erfolgreich demonstriert haben. Der Vorteil der Tilt-Strategie ist eine noch genauere Bestimmung des Loches. Die Kraftrückkopplung und die Nachgiebigkeit des Manipulators führen die Spitze des Zapfens nahezu automatisch zur Mitte des Loches. Allerdings sind Verklemmen und Verkeilen bei ähnlichen Loch- und Peg-Radien möglich.

Die Peg-in-Hole-Aufgabe zeigt zum einen die Möglichkeiten, die sich durch nachgiebige Roboter ergeben, und zum anderen die Schwierigkeit, menschliche Prozesse auf solche robotischen Systeme zu übertragen.

2.4.2.2 Soft-Robotik

Ein Roboter, der weich ist, impliziert, dass er verformbar, inhärent sicher und somit „kompatibel“ ist, nicht nur in Kontakt mit seiner Umgebung, sondern auch mit menschlichen Mitarbeitern, als Rehabilitationstool [92] in direktem Kontakt mit Patienten oder auch indirekt – in assistierender Funktion – im Haushalt [93]. Das Adjektiv „weich“ bzw. „soft“ in diesem Zusammenhang umfasst sowohl das Material bzw. die Materialzusammensetzung als auch die Steuerung der Roboter (in Form einer weichen Regelung bzw. Nachgiebigkeit). Solche weichen Roboter stellen heute immer noch zahlreiche neue Herausforderungen insbesondere für (i) die *Morphologie*, (ii) das *Material* und (iii) die *Regelung* [94] dar. Zusätzlich dazu sind neue Steuerungs- und Überwachungsansätze sowie Interaktionskonzepte zwischen Mensch und Roboter nötig [95].

Inspiriert durch die Natur und das Verhalten von Tieren, mit Unsicherheiten umzugehen, ist das neu entstehende Gebiet der Soft-Robotik, eine neue, überzeugende und vielversprechende Richtung. Im Vergleich zur traditionellen Robotik sind die Sensorik, Regelung und Steuerung in den mechanischen Eigenschaften sowie in der Strukturmorphologie des Roboters integriert und verteilt, um Unsicherheiten inhärent auszugleichen zu können [96]. Zum Beispiel erfährt der Roboter, der äußeren Störungen ausgesetzt ist, Verformungen und Steifigkeitsschwankungen, wodurch er seine Dynamik und Morphologie an die Umgebung anpassen kann und auf diese Weise eine Aufgabe trotzdem robust ausführen kann.

Trotz dieser Vorteile steht die Entwicklung von weichen Robotern vor verschiedenen Herausforderungen. Unter anderem sind dies das Entwerfen, Auswählen und Platzieren von Aktoren [97], das Erkennen äußerer Störungen und das Steuern der Struktur dieser stark unteraktuierten Systeme in Bezug auf sehr unterschiedliche Umwelt-, Betriebs-, Leistungs- und Sicherheitsanforderungen [98].

Prominente Beispiele der Soft Robotics stellen weiche Unterwasserroboter in Form von Fischen [99], Inspektionsroboter in Schlangenform [100], aber auch Robotergreifer, -Finger oder -Hände dar. Gerade letztere können genutzt werden, um (industrielle oder mobile) starre Roboter zu erweitern. Dies bietet den Vorteil einen robusten (*„robustness“*) Greifprozess filigraner Objekte mit Geschick (*„dexterity“*) zu erreichen unter Ausnutzen der inhärenten Elastizität und Nachgiebigkeit (*„compliance“*) der Finger [101] [102] [103].

2.4.3 Kooperation von Mensch und mobilem Roboter

Es gibt verschiedene Ansätze für die Steuerung von mobilen Robotern. Derzeit versuchen die meisten Ansätze, autonome und kartengenerierende Roboter zu entwickeln. Die MRK mobiler Roboter bezieht sich aber nicht nur auf autonome, sondern auch auf teleoperierte Roboter. Bei der Teleoperation kann das mobile System dann „low-level“-Aufgaben wie die Ruhelage und den stetigen Schwerpunktausgleich übernehmen, sodass der Operator entlastet werden kann und sich auf die „high-level“-Aufgaben wie Inspektion der Umgebung, Lokomotion oder Manipulation fokussieren kann. Für eine solche MRK, gerade bei der Teleoperation, ist dann aber eine optimierte Nutzerschnittstelle unabdingbar.

2.4.3.1 Teleoperation und Autonomie

Gerade im Bereich der mobilen Robotik unterteilt sich die MRK primär in *Teleoperation* und *Autonomie*. Der Planungs- und Denkprozess von Aktionen kann entweder voll automatisiert und somit autonom geschehen (siehe Def. 13) oder man plant autonome Aufgaben, welche aber noch zur Ausführung durch den Menschen bestätigt werden müssen, was auch als *semi-autonom* angesehen wird. Als letzte Möglichkeit hat der Mensch selbst die direkte Kontrolle über den Roboter und *teleoperiert* diesen (vgl. Def. 14).

Definition 13: Autonomie/Semi-Autonomie

Als autonome mobile Roboter werden Roboter bezeichnet, die sich in ihrer Umgebung selbständig bewegen und agieren können. Semi-autonom werden solche Systeme bezeichnet, die zwar selbständig agieren, bei denen jedoch der Nutzer zusätzlich prüfen und eingreifen kann. Der Zweck der Handlung wird also hierbei durch den Menschen vorgegeben.

Definition 14: Teleoperation/Telerobotik

Die Telerobotik ist ein Teilgebiet der Robotik, das sich mit der Steuerung von Robotern aus der Distanz, der Teleoperation, beschäftigt.

In der klassischen Industrierobotik – einer planbaren strukturierten und statischen Umgebung – ist die Nutzung autonomer Aktionen ein effizientes Mittel, um schnell und skalierbar Produktionsstraßen oder sonstige Montageaufgaben durchführen zu lassen. Gerade in der mobilen Robotik treten häufig *unstrukturierte*, *unvorhersehbare*, *nicht planbare*, *dynamische* Szenarien auf, in denen die kognitiven Fähigkeiten des Menschen benötigt werden, um schnell und effizient die richtigen Entscheidungen zu treffen und adäquate Aktionen durchzuführen.

Für solche dynamische Umgebungen werden oft mobile Systeme entworfen, die mit Hilfe einer Vielzahl von „Piloten“ bzw. Anwendern sowohl teleoperiert werden können, als auch semi-autonome Funktionen aufweisen. Ein Beispiel hierfür stellt der Roboter „Momaro“ (mobile manipulation robot) dar, welcher sowohl für Aufgaben der Lokomotion [104] als auch Manipulation [32] eingesetzt werden kann. In den Publikationen wird neben dem Hardwareaufbau des Roboters vor allem auch auf die Schnittstelle zur Teleoperation hingewiesen, die für die Lösung von Aufgaben unerlässlich ist. Weitere Untersuchungen dazu werden im nächsten Kapitel genauer beschrieben.

Klassische Nutzerschnittstellen solch teleoperierter Roboter sind oft joystick- und bildschirmbasiert (siehe Abb. 2.22). Diese können je nach Anwendungsfall und Einsatzszenario in Form und Funktion von höchst portablen Systemen zur Bombenentschärfung bis hin zu sehr robusten Operator-LKWs für Einsätze in Katastrophenszenarien vorkommen. Hierbei spiegeln sich die Funktion, Mobilität und Gewicht oft im Roboter selbst wider.

Neuartige Konzepte der Kollaboration oder Kooperation mit autonomen Systemen sind auch in der aktuellen Forschung zu finden.



(a)



(b)



(c)

Abb. 2.22: *Nutzerschnittstellen verschiedener Mobilitätsklassen, aufgenommen bei der Kerntechnischen Hilfsdienst GmbH (KHG)*

In [105] wird ein multimodales, teleoperiertes Szenario für mobile Roboter beschrieben, das insbesondere die Interaktion mit mehreren mobilen Systemen beinhaltet. Mit Hilfe von Gesten- und Spracherkennung können mehrere Roboter aus naher Distanz teleoperiert werden und semi-autonome Aktionen durchführen.

2.4.3.2 Teleoperative Roboter in Katastrophenszenarien

Die Fähigkeit, die Expertise eines bestimmten Nutzers an einen anderen Ort zu projizieren und die Fähigkeit, Bewegungen und andere Handlungen zu skalieren, haben bereits viele Anwendungen der Teleoperation hervorgebracht, von Roboterchirurgie bis hin zu Roboterplattformen für verschiedene Anwendungen auf dem Boden, unter Wasser oder in der Luft. Ein Überblick über die bilaterale Teleoperation ist in [106] beschrieben, in der die beiden Hauptziele der Teleoperation als „Stabilität“ in Bezug auf die Regelung bezeichnet werden und „Telepräsenz“ in Bezug auf die Transparenz des Robotersystems, der Umwelt und des Operators. [107] gibt einen Überblick über die allgemeine Geschichte der Schnittstellen zur Teleoperation mobiler Roboter sowie Fahrzeuge und fasst die zentralen Aspekte zusammen: „Teleoperationsschnittstellen bieten Werkzeuge und Displays, um die entfernte Umgebung wahrzunehmen, Entscheidungen zu treffen und Befehle zu generieren.“

Katastrophenszenarien wie Fukushima im Jahr 2011 haben die Notwendigkeit, dass teleoperierte Roboter sicher in hoch gefährlichen und ansteckenden Umgebungen handeln können, erneut untermauert. Diese Forderung prägte mehrere hoch dotierte Herausforderungen, insbesondere die DARPA Robotics Challenge (DRC), die sich auf die Entwicklung von Robotertechnologien für Katastrophenreaktionen konzentriert. Um eine stabile Systemsteuerung zu erreichen, fokussieren sich die erfolgreichsten Teams in der DRC nicht nur auf die Entwicklung zuverlässiger Roboterhardware, sondern auch auf den Menschen in-the-loop und dessen Kontrollstation, welche ein paar hundert Meter von dem Roboter auf dem Kurs (bzw. dem Einsatzgebiet) entfernt ist [108] [109] [110]. Zusätzlich dazu stellt die Überwindung von Kommunikationsverlust oder -instabilitäten eine große Herausforderung in der Forschung mobiler Rettungsroboter dar.

2.4.3.2.1 (Teleoperative) Steuerung quadrupedaler mobiler Roboter

Gerade im Bereich der Teleoperation können die zuvor angesprochenen verschiedenen Kamera- oder Kontrollmodi sehr gut variiert werden. Beispiele hierfür zeigt Petter Ogren in seinen Publikationen, in denen er Computerspielelemente auf reale robotische Systeme transferiert, um die Bedienbarkeit dieser zu vereinfachen. So zeigt er in [111] die Unterschiede von „Tank Control“ und „Free Look Control“, deren Anwendung in verschiedenen Computerspielen sowie die Übertragung dieser auf UGVs (Unmanned Ground Vehicles). In [112] wird der Nutzen – der reduzierte Workload für den Operator – beschrieben und evaluiert, der aus diesen Steuerungsmodi herrührt.

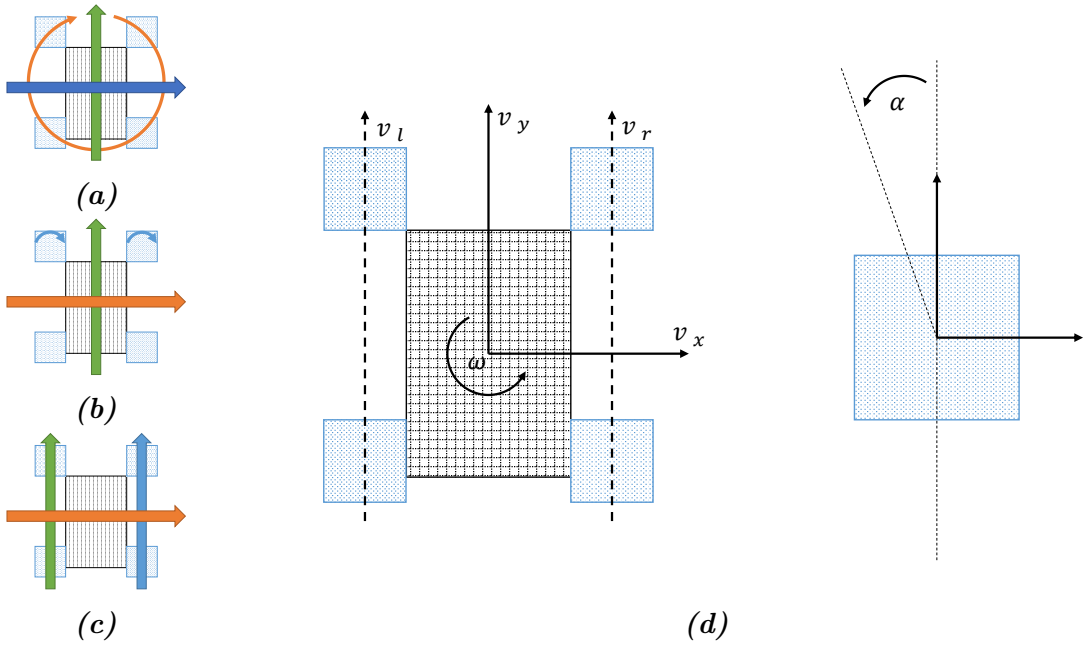


Abb. 2.23: *Schematische Zeichnung quadrupedaler mobiler Roboter (von oben)*

Nachstehend wird die mathematisch-technische Grundlage verschiedener Steuerungsmodi (vgl. Abb. 2.23) präsentiert, wobei folgende Umrechnungen (für die Basis B, die Reifen W und das Weltkoordinatensystem WKS) allgemein gelten:

$$T_W^{rel} = (T_B^{WKS})^{-1} \cdot T_W^{WKS} \tag{2.7}$$

$$T_W^{rel} = (R_W, \boldsymbol{p}_W)^T \tag{2.8}$$

$$\boldsymbol{p}_W = \begin{pmatrix} p_{W_x} \\ p_{W_y} \\ p_{W_z} \end{pmatrix} \tag{2.9}$$

Außerdem werden folgende Geschwindigkeiten benötigt:

$$v_L, v_R, \omega, v_x, v_y \tag{2.10}$$

sowie der Abstand der Ankerpunkte von der Basis d.

Bei (a), der *Entkopplung von Rotation und Translation*, wird das Eingabegerät (z. B.

Gamepad) genutzt, um v_x, v_y sowie ω zu variieren. Hierbei wird mit dem Ackermann-winkel $\omega' = \frac{\omega}{d}$ und den geometrischen Beziehungen der Punkte p die Geschwindigkeit v berechnet, mit:

$$v_{W_x} = v_x + \frac{\omega}{d} \cdot p_{W_x} \tag{2.11}$$

$$v_{W_y} = v_y - \frac{\omega}{d} \cdot p_{W_y} \tag{2.12}$$

$$v = \sqrt{v_{W_x}^2 + v_{W_y}^2} \tag{2.13}$$

Der Lenkwinkel α wird wiederum geometrisch berechnet zu:

$$\alpha = atan(\frac{v_{W_x}}{v_{W_y}}) \tag{2.14}$$

Bei (b), der *autoähnlichen Steuerung*, wird über das Eingabegerät v sowie ein Lenkwert s übergeben, woraus wiederum eine geforderte Drehgeschwindigkeit ω berechnet werden kann:

$$\omega = v \cdot s \tag{2.15}$$

Mit $v_x = 0$ und $v_y = v$ kann daraufhin Gl. 2.12 genutzt werden.

Bei (c), der *Raupensteuerung*, werden über das Eingabegerät v_L, v_R sowie ein v_x übergeben, wobei $v_x = \frac{1}{2} \cdot (v_{x_L} + v_{x_R})$ gemittelt wird. Hieraus kann man wie oben beschrieben die Geschwindigkeiten in Fahrtrichtung (y) und lateral dazu (x) berechnen:

$$v_{W_x} = v_x + \frac{v_R - v_L}{d} \cdot p_{W_x} \tag{2.16}$$

$$v_{W_y} = (v_R + v_L) - \frac{v_R - v_L}{d} \cdot p_{W_y} \tag{2.17}$$

2.4.3.3 Modellbasierte MRK

Nicht nur im Bereich der haptischen Schnittstelle werden Modelle zur Manipulation und Steuerung von Systemen genutzt. Auch bei verfahrenstechnischen Prozessen wird beispielsweise mit der Modellprädiktiven Regelung (MPC) [113] ein Modell des zu regelnden Prozesses verwendet, um dessen zukünftiges Verhalten zu berechnen. Aber auch im Bereich der MRK gibt es vielversprechende Ansätze, sich Modelle nutzbar zu machen.

2.4.3.3.1 Simulation-based Operator Interfaces (SBOI)

Sheridan fasst den Stand der Technik der menschlichen Roboter-Interaktion in [114] zusammen, indem er aktuelle Herausforderungen überprüft: von der Überwachung der Roboter für routinemäßige industrielle Aufgaben bis hin zu teleoperierten Fahrzeugen und Flugzeugen sowie menschlichen robotersozialen Interaktionen. Er kommt zu dem Schluss, dass es notwendig ist, nochmals zu rekapitulieren, wo Menschen am besten in Systeme passen im Vergleich zu künstlicher Intelligenz und Computersteuerung. Als zentrale Methode in der modernen Robotersteuerung sieht Sheridan viel Potenzial in sogenannten

mentalen Modellen, welche alles Wahrgenommene in einem Modell der Umgebung abbilden (siehe Def. 15). Diese werden ständig aktualisiert, so wie es die Menschen zu tun scheinen.

Definition 15: Mentales Modell

Mentale Modelle sind Abbildungen bzw. Modelle von dem, was in der Umgebung wahrgenommen wurde und vor sich geht.

Die Kombination der 3D-Simulation mit mentalen Modellen ist in [8] als konzeptionelle Erweiterung von Simulationstechnologien motiviert. Diese Simulation-based Operator Interfaces (SBOI) können dann für die direkte Steuerung von Robotern aus der Simulation heraus genutzt werden. Außerdem können diese zur Unterstützung teleoperierter Systeme dem Nutzer Assistenzsysteme zur Verfügung stellen. Eine Einführung in Bedienoberflächen für die Simulation, die AR und VR nutzen, ist in [115] zu finden, in der diese nachher auch in sogenannten „Search and Rescue Applications“ angewendet werden.

2.4.3.3.2 Modellbasierte Telemanipulation

Im Bereich der haptischen Telemanipulation stellt die von Niemeyer vorgestellte *Model-Mediated Telemanipulation* [116] ein eindimensionales System der modellbasierten bilateralen Teleoperation vor. Schon in [117] stellten die Autoren ein robotisches Master-Slave-System vor, das direktes haptisches Feedback in der Telemanipulation liefern sollte, obgleich große Verzögerungen („delay“) vorlagen. Möglich ist dieser Abgleich aus Roboterbewegung und Kraftrückkopplung mittels eines (vereinfachten) Modells, um ein intuitives Actio-Reactio-Prinzip des Operators trotz Verzögerung zu gewährleisten. Zentral in diesem Fall ist der Abstraktionsgrad des genutzten Modells: Das Modell abstrahiert eine statische Umgebung und fokussiert sich auf die reine haptische Wechselwirkung sowie die Verringerung des Delays. Die Autoren fassen ihre Arbeit als wichtigen ersten Schritt auf, um Force Feedback und die damit verbundene extrem hohe Updaterate mit teleoperierter Steuerung und Regelung zusammenzubringen.

2.4.3.3.3 Interne Modelle für inhärent sichere Roboter

Einen anderen Ansatz verfolgt [118], wo mobile Roboter mit Hilfe einer integrierten Simulation ihrer selbst die Sicherheit in der MRK erhöhen sollen. Dies basiert auf einer „Konsequenzen-Engine“ [119], welche mögliche Konsequenzen der nächsten Aktion innerhalb eines Simulators evaluiert, sodass der Roboter selbst wählt, welche Aktion er in der realen Welt durchführt. Das interne Modell beinhaltet dann das Robotermodell, die Robotersteuerung und ein Weltmodell. Aktionen werden dann hinsichtlich ihrer Sicherheit ausgewählt, die hierbei definiert ist in Form einer Sicherheitsdistanz ($\approx 0,2$ m) die zwischen Roboter und andern Hindernissen (andere Roboter, Menschen, Objekte) nicht unterschritten werden darf. Die Autoren zeigen insbesondere, dass durch mehrschrittige Simulationen Entscheidungsbäume aufgespannt werden, die nicht nur Aktionen erster Ordnung, sondern auch die damit verbundenen Konsequenzen und Interaktionen zweiter und höherer Ordnung mit einbeziehen.

2.4.4 Hard- und Softwareschnittstellen in der Industrie 4.0

Heterogene Szenarien aus Soft- und Hardware, Kommunikation in Echtzeit sowie die Integration des Menschen in das Gesamtsystem der Industrie 4.0 bestehen aus mehr als nur MMI und MRK. Neben der zuvor beschriebene Schnittstelle des Menschen, insbesondere dem Erfassen der Nutzerintention durch Eingabegeräte sowie die Interaktion und Steuerung robotischer Systeme, gibt es noch weitere Schnittstellen. Gemeint ist die Hardwarekommunikationsschnittstelle (bzw. die Schnittstelle zwischen Maschine und Maschine (M2M)). Diese Schnittstelle ist eine weitere entscheidende, notwendige Komponente, um Mediatoren zwischen Mensch und Hardware zu erschaffen.

2.4.4.1 Cyber-physisches System (CPS)

Neben der zuvor ausführlich beschriebenen Nutzerschnittstelle wird für Mediatoren von CPS (siehe Def. 16) eine Schnittstelle zwischen dem „cyber“ und dem „physischem“ Part benötigt. Diese Schnittstelle wird mit Hilfe von Netzwerkprotokollen oder „Middlewares“ umgesetzt. Hiermit ist es möglich, Netzwerke aus verbundenen Maschinen zu erschaffen.

Definition 16: Cyber-physisches System

Ein System, das reale (physische) Objekte und Prozesse verknüpft mit informationsverarbeitenden (virtuellen) Objekten und Prozessen über offene, teilweise globale und jederzeit miteinander verbundene Informationsnetze, wird als CPS bezeichnet. [a]

2.4.4.2 Netzwerkprotokolle/Middlewares

Zur Kommunikation zwischen einer Vielzahl von Soft- und Hardwaremodulen haben sich verschiedene Kommunikationsstandards herauskristallisiert. Darauf aufbauend erlauben „Middlewares“ dem Entwickler, auf eine High-Level-API zuzugreifen, und Low-Level-Details der Kommunikation werden automatisch von der Middleware behandelt.

Im Kontext der IoT- und I4.0-Anwendungen gibt es eine Vielzahl von M2M-Kommunikationsprotokollen, die Gegenstand aktueller Forschung sind. Hierzu zählen u. a.: OPC UA (Open Platform Communications Unified Architecture) [ag] [120] [121], MQTT (Message Queuing Telemetry Transport) [ah] [122], DDS (Data Distribution Service) [ai], ROS (Robot Operating System) [aj], ROS2 [ak] [123], CoAP (Constrained Application Protocol) [al] [124], XMPP (Extensible Messaging and Presence Protocol) [am] [125], AMQP (Advanced Message Queuing Protocol) [an] [126] usw. Alle davon versuchen eine standardisierte M2M-Kommunikationsschicht zu etablieren. Als sogenannte *message oriented middlewares (MOMs)* basieren alle diese Software-Infrastrukturen darauf, dass sie das Senden und Empfangen von Nachrichten in verteilten Systemen unterstützen bzw. ermöglichen. Details darüber werden daher im Folgenden nur für das in dieser Arbeit primär genutzte ROS gegeben. Die generelle Kommunikationsinfrastruktur aller Middlewares – das Unterstützen des Publish/Subscribe (Pub/Sub) Pattern – wird dabei beschrieben, wohingegen spezifische Eigenheiten der einzelnen Protokolle in den Quellen

zu finden sind. Weiterführende Informationen der Unterschiede zwischen den einzelnen Protokolle bzgl. deren Besonderheiten, Paket-Overhead, Performanz (round trip time), Latenz, Quality of Service und Sicherheit ist in [127] beschrieben.

2.4.4.3 ROS-Softwareframework

Das Robot Operating System (ROS) ist ein Softwareframework mit dem inhärenten Ziel, dass jeder Nutzer möglichst einfach Code für robotische Systeme schreiben und nutzen kann. In der Robotik hat sich ROS als akzeptierte und standardisierte Schnittstelle für ein breites Spektrum an robotischen Applikationen etabliert [128].

Bestehend aus einem Core und vielen Nodes erlaubt es, verteilte Systeme zu erstellen und einfache Kommunikationswege beizubehalten. ROS wurde 2007 mit Hilfe des *Stanford AI Robot projects* vom SAIL (Stanford Artificial Intelligence Laboratory) entwickelt. Durch die weite Verbreitung existieren neben vielen unterstützten Plattformen auch zahlreiche Beispiele und Bibliotheken, die die schnelle Nutzung des Frameworks erlauben. Seit 2008 wird das Projekt hauptsächlich von *Willow Garage*, einem Roboter-Forschungsinstitut, betreut. Dabei bekommt es von mehr als 20 anderen Instituten Unterstützung [129].

ROS ist im eigentlichen Sinn kein Betriebssystem, sondern vielmehr eine Kollektion aus Tools, Libraries und Konventionen, um die Entwicklung komplexer robotischer Systeme zu vereinfachen und zu vereinheitlichen. Die zentrale Idee für ROS als Middleware war nicht, ein Framework für eine spezifische Anwendung zu entwickeln, sondern eine allgemein akzeptierte Basis für alle möglichen Entwicklungen zu schaffen. In [130] fassen die ROS-Gründer die zentralen Ziele von ROS als „peer-to-peer, tools-based, multi-lingual, thin, free and open-source“ und „support our philosophy of modular, tools-based software development“ zusammen. Neben der Modularität steht bei ROS vor allem die Community im Vordergrund, die ROS den heutigen Stellenwert gab. Die zentralen Bestandteile von ROS sind:

1. die Kommunikationsinfrastruktur,
2. die roboterspezifischen Features und
3. die verschiedenen Tools.

Da in der vorliegenden Arbeit die Nutzung von Schnittstellentechnologien von besonderer Bedeutung ist, liegt der Fokus auf der Kommunikationsinfrastruktur. Man bekommt allerdings alle Features sowie den Zugriff auf die verschiedenen Tools durch geeignete Implementierungen nahezu umsonst dazu. Im folgenden Abschnitt wird daher die Nachrichtenübermittlung (das „Message Passing Interface“) ausführlich beleuchtet.

Zum Robot Operating System existieren einige Quellen. Die nützlichste und sinnvollste Ressource stellt das eigene Wiki [aj] dar. Das ist darauf zurückzuführen, dass grundsätzlich die Communityarbeit ein (wenn nicht sogar der) zentrale Eckpfeiler von ROS ist und somit das communitybestimmte Wiki immer die aktuellste Ressource momentaner Forschung und deren Fortschritte darstellt.

2.4.4.3.1 ROS Message Passing

Zunächst soll in diesem Kapitel ROS in der Funktion als Middleware und somit das

grundlegende Konzept der Kommunikation einzelner *Nodes* über *Messages* betrachtet werden.[2]

Es gibt einen globalen ROS-*Master*, bei dem sich die verschiedenen Nodes registrieren müssen (siehe Abb.2.24).

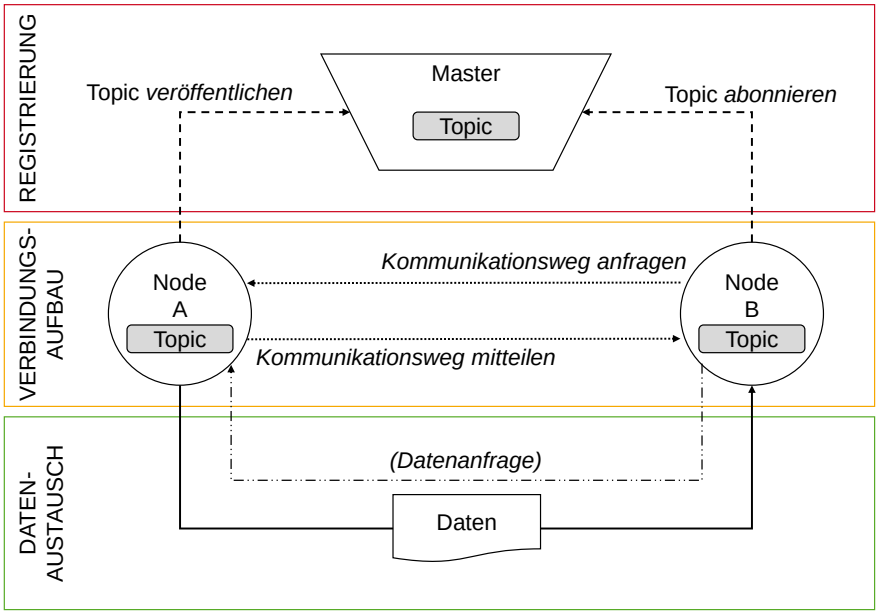


Abb. 2.24: *Funktionsprinzip der ROS-Messages: 1. Registrierung, 2. Verbindungsaufbau, 3. Datenaustausch*

Der Master ist von grundlegender Bedeutung und dient als Meldestelle, an der sich Nodes anmelden, selber neue *Topics* registrieren und sich über angebotene *Messages* und *Services* der anderen Nodes erkundigen können. Der Master verwaltet somit eine Liste aller Nodes und derer angebotener Topics sowie Services und stellt einen globalen *Parameter-Server* zur Verfügung. Da sich zwei Master gegenseitig nicht beeinflussen, können auch mehrere von ihnen parallel aktiv sein, jedoch können sich Nodes immer nur an einem dieser anmelden.

Nodes bilden das zentrale Element des Robot Operating Systems und sind einzelne Programme, die Messages verschicken (*Publisher*) oder abonnieren (*Subscriber*) sowie Services bereitstellen oder in Anspruch nehmen können. Ein Node kann dabei eine einzelne Roboterkomponente (z. B. einen Sensor) oder auch ganze Systeme abbilden. Eine Message stellt bei dieser Kommunikation eine n:m-, ein Service eine 1:1-Kardanität her. Sobald zwei Nodes sich kennen, können diese direkt per „peer-to-peer“ (über Serialisieren, TCP/UDP-Kommunikation, Deserialisieren) miteinander kommunizieren. Für den seltenen Fall, dass Publisher und Subscriber im selben Node existieren, wird der Schritt der Serialisierung und anschließender Deserialisierung durch einen Shared Pointer optimiert.

[2] Diese Nachrichtenübertragung wird im Folgenden, wie auch im ROS-Standard, als „Message Passing“, die Nachrichten als „Messages“ und die Nachrichtentypen als „Message Types“ bezeichnet

Es gibt jedoch eine Grenze der maximalen Anzahl von zu verarbeitenden Nachrichten. Sie wird vom Node durch die Größe seiner *Messagequeue* und der Geschwindigkeit, mit der er die eingegangenen Nachrichten verarbeiten kann, vorgegeben. Die Messagequeue stellt einen Eingangs- bzw. Ausgangspuffer dar, um Schwankungen in der Übertragungsgeschwindigkeit oder der Verarbeitungsgeschwindigkeit auszugleichen. Ist die Anzahl an übertragenen Nachrichten im zeitlichen Mittel größer als das zeitliche Mittel der verarbeiteten Messages, so läuft die Messagequeue über. In diesem Fall werden die ältesten Messages in der Messagequeue gelöscht [ao].

Die Kommunikation der Nodes per Messages ist das zentrale Austauschmedium von Daten. Innerhalb der Message Definition Language (MDL) ist dieses Format klar definiert und gibt an, welche Datentypen es gibt.

- Aufbau der MDL:

```
# standard form defined by MDL:
fieldtype1 fieldname1;
fieldtype2 fieldname2;
```

- Definition des standard message type (*.msg):

```
# msg type example std_msgs/Float64
float64 data;
```

- Definition des standard service type (*.srv):

```
# srv type example
float64 a;
float64 b;
---
float64 result;
```

Durch die Kenntnis des Typs von Publisher und Subscriber können diese die zu übermittelnden Daten serialisieren bzw. deserialisieren. Services haben zusätzlich noch einen Rückgabeparameter (alias „result“) in der MDL angegeben.

Wichtig zu wissen ist, dass ROS durch diverse Client-Bibliotheken in weitere Programme eingebunden werden kann. In dieser Arbeit wird dazu primär auf *roscpp* und sekundär auf *rospy* zurückgegriffen. Zusätzlich bringt ROS ein eigenes Buildsystem (namens catkin) mit, das aber hier nur erwähnt werden soll. Entsprechendes gilt für verschiedene Tools, die den Funktionspool von ROS erweitern können, wie z. B. *rqt*, *rviz* usw.

2.4.4.3.2 ROS2

Das Robot Operating System 2 (ROS2) ist eine Weiterentwicklung von ROS, welche parallel zu dieser existiert. Zum einen möchten die Entwickler hiermit insbesondere die große Community, die weiter an ROS festhalten möchte, nicht verschrecken und zum anderen manche Fehlplanungen von ROS in ROS2 korrigieren. ROS2, insbesondere entwickelt für „multi-robot control“, „Quality of Service (QoS) Settings“, „Real-time capability“, und „Interoperability“ zwischen Betriebssystemen kam im Dezember 2017 auf den Markt. Zentrale Änderung im Bereich des Message-Passing-Systems ist das Nichtvorhandensein

eines ROS-Masters. Im Gegensatz zu ROS basiert ROS2 auf der Data Distribution Service (DDS) Middleware, sodass die einzelnen Knoten direkt miteinander kommunizieren und keinen Verteiler mehr benötigen. DDS bietet dabei Industriestandard und ist auf Echtzeitsysteme [ak] optimiert.

2.5 Von 3D-Simulationssystemen zu virtuellen Experimentierumgebungen für Digitale Zwillinge

In diesem Kapitel wird auf die simulationsbasierte Entwicklung von komplexen mechatronischen Systemen eingegangen.

Hierzu wird in Kap. 2.5.1 zunächst ein Überblick über Simulationen, zugrundeliegende Prozesse, Bestandteile sowie Simulationssystemen zur MRK gegeben. Spezieller Fokus wird hier auf das Simulationssystem VEROSIM gelegt (Kap. 2.5.1.8) für das das Frontend, die Datenbank und das Backend in Form der Starrkörpersimulation (Kinematik und Dynamik), der Sensorsimulation und das Rendering Framework vorgestellt werden.

Eingegliedert wird das Ganze in virtuelle Experimentierumgebungen, welche die Simulationen auf ganzheitlicher Systemebene erlauben. Diese Virtuellen Testbeds (VTBs) werden daraufhin in Kap. 2.5.3 näher betrachtet. Hierzu wird in Kap. 2.5.3.3 gezeigt, wie komplexe Systeme mit diesen entwickelt werden können, warum man VTBs nutzt, wie diese funktionieren und warum diese gerade in der Robotik viele Vorteile bieten. Darauf aufbauend wird in Kap. 2.5.3.4 dargelegt, wie man ein neues VTB erstellt in Form von Modellen, Implementierung, Einbindung oder Kopplungen, Evaluation und finaler Anwendung. Schlussendlich wird in Kap. 2.5.3.5 beschrieben, wie der Transfer vom VTB zum realen System aussieht.

2.5.1 Simulationssysteme zur Mensch-Roboter-Kooperation

Simulationen werden nahezu in jeder Industriesparte, in jedem Komplexitätslevel und von jedem Forschenden genutzt. Leider führt dies auch dazu, dass der Begriff einer Simulation nicht klar definiert, verwaschen bzw. abgenutzt wird.

Häufig ist der Grund für eine Simulation, dass das reale System noch nicht verfügbar ist, aber man schon jetzt Daten sammeln möchte. Jedoch gibt es auch weitere Gründe, warum man eine Simulation nutzen möchte. Einige Beispiele dafür sind:

- Das reale System existiert noch nicht,
- steht nicht zur Verfügung,
- würde zerstört werden,
- wird noch entwickelt und soll optimiert werden oder
- das Experiment ist gefährlich, ethisch bedenklich oder zu teuer.

Gerade bei der Entwicklung komplexer Systeme werden Simulationen genutzt, um das I/O-Verhalten nachzubilden und zu untersuchen.

Ein komplexes System (Def. 18) kann dabei als Modell (Def. 19) abgebildet und in einem Simulator (Def. 21) untersucht werden. Diese zentralen Aspekte der *Systemtheorie*, bzw. der Modellierung und Simulation komplexer Systeme, sind u. a. auch in [131] zu finden (vgl. Abb. 2.25).

2.5.1.1 Begriffsdefinition

Ein System ist nach [132] eine abgegrenzte Anordnung von Komponenten, die miteinander in Beziehung stehen. Das Systemverhalten entsteht dann aus dem Zusammenwir-

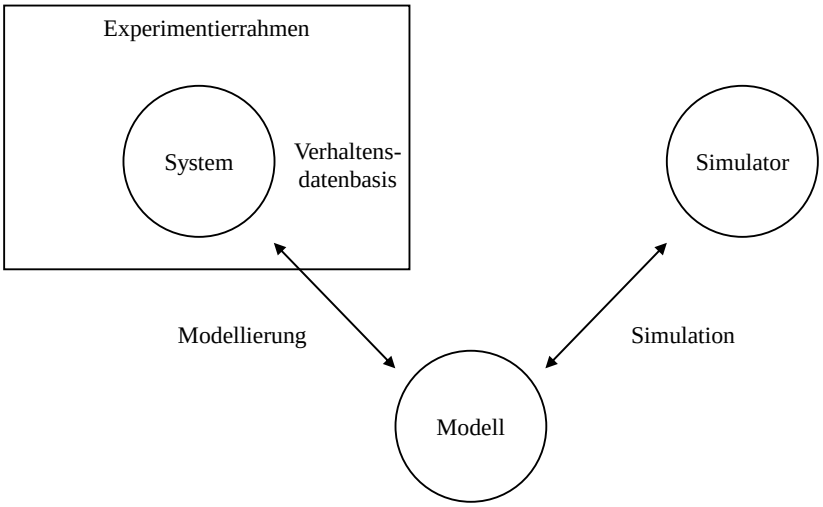


Abb. 2.25: *Systemtheorie – die zentralen Bestandteile: System, Modell und Simulator sowie deren Relationen (basierend auf [131])*

ken dieser Komponenten. Ein System ist dabei festgelegt durch *Systemgrenze*, *Ein- und Ausgangsgrößen des Systems*, *Komponenten, Subsysteme und Systemelemente*, *Ablaufstruktur*, *Aufbaustruktur*, *Zustandsgrößen* und *Zustandsübergänge*. Nach [133] muss die Unterscheidung von System, Komponenten und Umwelt gedanklich konstruiert werden und ist somit in gewissem Maße willkürlich. Für jedes betrachtete System gibt es eine Außenwelt (ein umfassenderes System), mit der es wechselwirkt. Das Verhalten eines solchen *offenen System* kann nur verstanden werden, wenn es in Verbindung mit seiner Umwelt als Teil eines Gesamtsystems gesehen wird.

Definition 17: System

Ein System ist die Menge aller Elemente dieses Systems (seiner Untersysteme), die im gewünschten Sinn miteinander zusammenarbeiten. Jedes System – entweder als reales Objekt oder als begriffliche Abbildung – ist charakterisiert durch die Umgebung, die Verhaltensfunktion und die Struktur.

Ursprünglich aus der Systemtheorie werden triviale Systeme als solche bezeichnet, welche irgendein Eingangs-/Input-Signal[1] x gemäß einer konstanten Funktion $f(x)$ in ein Ausgangs-/Output-Signal[1] y verwandeln:

$$y = f(x)$$

In jedem Fall verhält sich diese immer gleich und ist somit vollständig vorhersehbar. Nicht-triviale Systeme verändern hingegen nach jedem Input (und in Abhängigkeit von diesem) ihren inneren Zustand und damit die Überführungs- oder Wirkungsfunktion:

$$y = f(x, z(x))$$

[1] Es gilt: Eingang/Ausgang = Input/Output = I/O

Definition 18: Komplexes System

Komplexe Systeme sind nicht vollständig aus den Eigenschaften der Komponenten des Systems erklärbar, sondern bestehen aus einer Vielzahl miteinander verbundener und interagierender Teile, Entitäten oder Agenten.

In Abb. 2.26 sind Systeme in Bezug auf die Anzahl an Systemelementen und der Veränderlichkeit im Zeitverlauf klassifiziert.

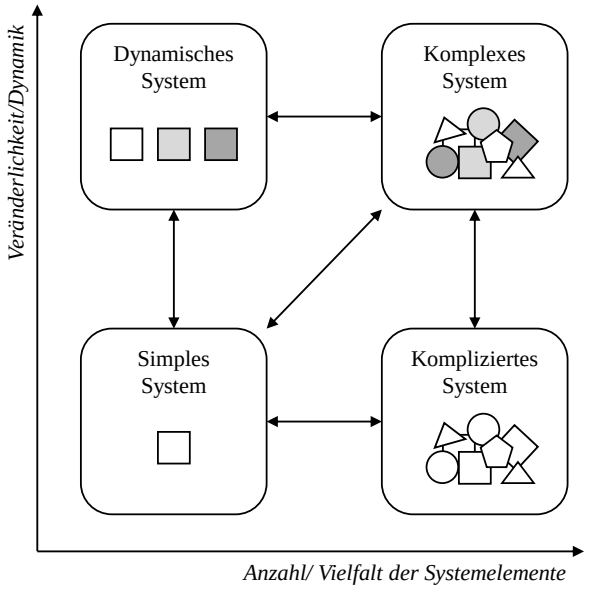


Abb. 2.26: *Komplexe Systeme (angelehnt an [133]): Viele verschiedene, interagierende Elemente führen zu einem zeitlich veränderlichen, offenen Gesamtsystem.*

Im Gegensatz zu einem *simplen System* zeichnet sich ein *kompliziertes System* dadurch aus, dass es zwar aus vielen Elementen besteht, sich jedoch mittels Aufschlüsselung in kleinere, überschaubare Teile relativ einfach auf eine Summe simpler Systeme zurückführen lässt. Es gibt aber auch *dynamische Systeme*, bei denen die Elemente (wenige) verschiedene Zustände annehmen können. Ein *komplexes System* ist fähig, in einer gegebenen Zeitspanne eine große Zahl von verschiedenen Zuständen annehmen zu können. Im Gegensatz zum komplizierten System hat ein komplexes System Merkmale, die seine Einzelteile ursächlich nicht in sich selbst tragen müssen. Das Ganze ist somit mehr als die Summe seiner Bestandteile und man spricht auch von der Emergenz[2] des Systems.

Nach [132] ist ein Modell (vgl. Def. 19) eine vereinfachte Nachbildung eines geplanten oder existierenden Systems mit seinen Prozessen in einem anderen begrifflichen oder gegenständlichen System. Es unterscheidet sich hinsichtlich der untersuchungsrelevanten Eigenschaften nur innerhalb eines vom Untersuchungsziel abhängigen Toleranzrahmens vom Vorbild. Nach [134] repräsentieren Modelle Systeme mit unterschiedlichen Abstraktionsniveaus (Analyse, Entwurf, Implementierung), Systemteilen oder Gewerken

[2] Emergenz bezeichnet die Möglichkeit der Herausbildung von neuen Eigenschaften oder Strukturen eines Systems infolge des Zusammenspiels seiner Elemente.

(Verfahrenstechnik, Mechanik, Elektrotechnik, Automatisierung, Informatik), Belangen (Sicherheit, Leistung, Belastbarkeit) und Aufgaben (Test, Einsatz).

Definition 19: Modell

Unter einem Modell versteht man die (vereinfachte) Nachbildung eines Originalsystems. Das Modell muss dem Originalsystem im Hinblick auf den Zweck seiner Realisierung hinreichend ähnlich sein. [135]

Zumeist bedient man sich einer Simulation, wenn die Analyse oder Lösung eines Systems analytisch zu komplex wird. Daher muss zunächst ein Modell gefunden werden, welches das System beschreibt und mit Hilfe einer Simulation gelöst werden soll. Dieses Modell wird anschließend parametriert, um anschließend simuliert werden zu können. Die Simulation kann als Experiment am Modell gesehen werden, das Rückschlüsse auf das reale Systemverhalten ermöglicht (vgl. auch Def. 20). Die Simulation stellt immer eine Abstraktion des realen Systems dar, zudem muss man an geeigneten Stellen Vereinfachungen vornehmen.

Nach [132] ist eine Simulation das Nachbilden eines Systems mit seinen dynamischen Prozessen in einem *experimentierbaren Modell*, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind. Insbesondere werden Prozesse über die Zeit entwickelt. Im weiteren Sinne wird unter Simulation das *Vorbereiten*, *Durchführen* und *Auswerten* gezielter Experimente mit einem Simulationsmodell verstanden.

Definition 20: Simulation/3D-Simulation

Eine Simulation ist das Nachbilden von Prozessen realer Systeme in einem Modell und das anschließende Durchführen von Experimenten an diesem Modell, bei dem Eingangsgrößen in Ausgangsgrößen transformiert werden.
Die 3D-Simulation beschreibt eine Klasse von Simulationen, die 3D-Modelle und 3D-Rendering-Technologie für eine Computersimulation nutzen.

Um solche komplexen Systeme zu entwickeln oder zu untersuchen, werden oftmals Simulationen eingesetzt. Ein Simulationswerkzeug (Simulator) ist nach VDI3633 [132] ein „Softwareprogramm, mit dem ein Modell zur Nachbildung des dynamischen Verhaltens eines Systems und seiner Prozesse erstellt und ausführbar gemacht werden kann."

Definition 21: Simulationssystem

Ein Simulationssystem ist ein Programm, das für Eingangsdaten und Modell mittels eines geeigneten Algorithmus Berechnungen durchführt und die Ergebnisse abspeichert oder ausgibt.

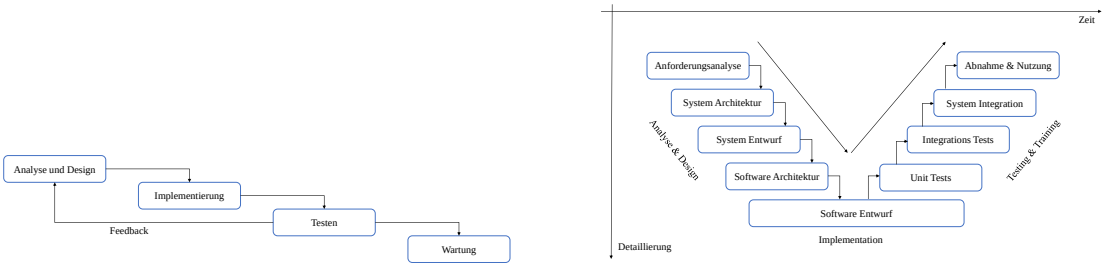
2.5.1.2 Prozesse zum Einsatz von Simulationen

Es gibt verschiedene Prozesse, Abläufe oder Workflows, in denen Simulationstechnologien genutzt werden. Im Ingenieurbereich werden Simulationen vor allem eingesetzt, um aus einer Idee eine Anwendung zu generieren. Die Simulation fokussiert dabei überwiegend

einzelne Simulationsaspekte, ist rein funktional, nicht interaktiv und die Nutzung der Simulation endet spätestens bei der Fertigung des untersuchten Systems. Dieser Prozess sowie Erweiterungen davon sollen im Folgenden thematisiert werden.

2.5.1.2.1 Projektprozesse

In Abb. 2.27 kann man die verschiedenen Ansätze zu Projektprozessen erkennen.



(a) Wasserfallmodell 2.0 *(b) V-Modell*

Abb. 2.27: *Projektprozesse – eine Übersicht*

Diese werden natürlich auch im Projektmanagement sowie der Softwareentwicklung genutzt, um 3D-Simulationssysteme zu entwickeln, jedoch ist es vielmehr die 3D-Simulation als Tool, das neue Möglichkeiten bietet, diese Prozesse zu verbessern. Die 3D-Simulation kann schon früh im Entwicklungsprozess eingesetzt werden, zur

- frühzeitigen Integration (externer) Komponenten,
- Prozessoptimierung durch die Simulation,
- Validierung im Entwicklungsprozess in Form von
  - „Unit Tests“ von (externen) Komponenten in der Simulation,
  - Evaluation von Komponenten im Kontext der finalen Anwendung,
- Aufzeigen einer Vision durch geeignete visuelle Präsentation im Bereich der
  - Projektakquise,
  - Marketing und
  - Visualisierung des Projektstandes.

Die *Vorteile der Simulation* liegen dabei auf der Hand: Nicht vorhandene Hardware kann und wird durch die 3D-Simulation ersetzt. Nach initialem Modellieren „kostet“ jede weitere Simulation nichts und kontinuierliche Modellupdates lassen weitere Rückschlüsse zu. Somit lässt sich also der gesamte Entwicklungsprozess in Form von

- Akquise,
- Entwurf und Analyse,
- Entwicklung,
- Validierung und Verifikation,
- Visualisierung und Interaktion,
- Betrieb und Monitoring,
- Training,

- Marketing und
- Wissenstransfer

abbilden bzw. mit Hilfe der Simulation durchführen. Die drei Säulen zur Entwicklung innerhalb dieses Rahmens sind die aus dem Wasserfallmodell 2.0 (Abb. 2.27a) bekannten „Design“, „Implementierung“ und „Testing“. Hinzu kommen dazu noch die im V-Modell (Abb. 2.27b) genannte „Nutzung“ sowie das aus der Softwareentwicklung bekannte Feedback oder „Recap“. Damit ergibt sich das Gesamtbild zum Einsatz von Simulationen in Abb. 2.28.

2.5.1.2.2 Die Simulation im Lebenszyklus von Maschinen

Man sieht, dass die Simulation nicht nur in der Entwicklung und in Tests Anwendung findet, sondern darüber hinaus auch in der zweiten und dritten Phase, in der die Simulation in-the-loop zur Analyse, Steuerung usw. oder im Nachgang als Wissensdatenbank und Ausgangspunkt möglicher Optimierungen genutzt wird (vgl. Abb. 2.29) und somit über den gesamten Lebenszyklus des robotischen Systems hinweg.

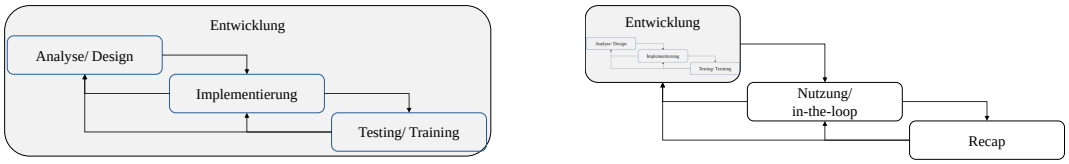


Abb. 2.28: *Einsatz von Simulation*

Den Lebenszyklus von Maschinen im Allgemeinen kann man unterteilen in Entstehung (vor der Nutzung), Betrieb (während der Nutzung) und Verwertung (nach der Nutzung). Der klassische Produktlebenszyklus hingegen beinhaltet die Planung, Entwicklung, Prozessplanung, Herstellung, Vertrieb, Nutzung und Ablöse. In dieser Arbeit wird jedoch der Fokus auf die Entstehung und den Betrieb robotischer Systeme gelegt und der Lebenszyklus nach Def. 22 genutzt (vgl. auch Abb. 2.29).

Definition 22: Lebenszyklus

Der Lebenszyklus robotischer Systeme beinhaltet insb. die Entstehung (Konzeption, Design, Entwicklung, Implementierung, Herstellung und Training) und den Betrieb (Nutzung, Steuerung, Visualisierung, Wartung, Instandhaltung, Umrüstung sowie den Recap zur Analyse und Optimierung).

Die weitere Verwertung (Entsorgung, Recycling und Umfunktionierung) ist dabei kein essentieller Teil des Lebenszyklus im Rahmen dieser Arbeit, jedoch kann die in dieser Arbeit entwickelten Struktur auch hier ein hilfreiches Mittel werden.

2.5.1.3 Model-based Systems Engineering (MBSE) und die Systems Modeling Language (SysML)

Model-based Systems Engineering (MBSE) ist die Entwicklung von Systemen, weg von aktueller, Dokumenten-zentrierter hin zu neuer modellbasierter Entwicklung. MBSE un-

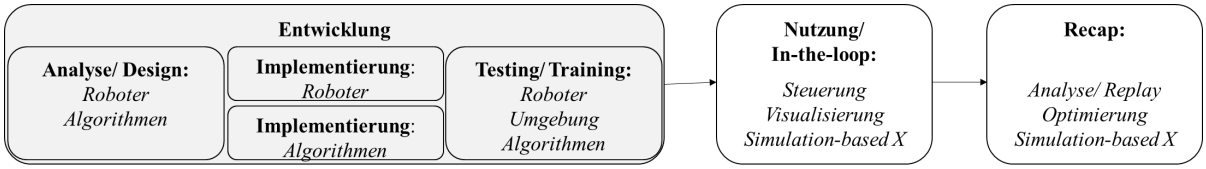
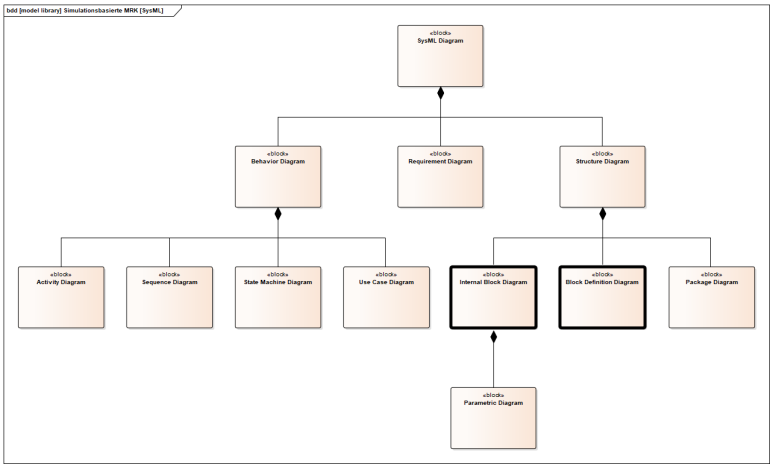
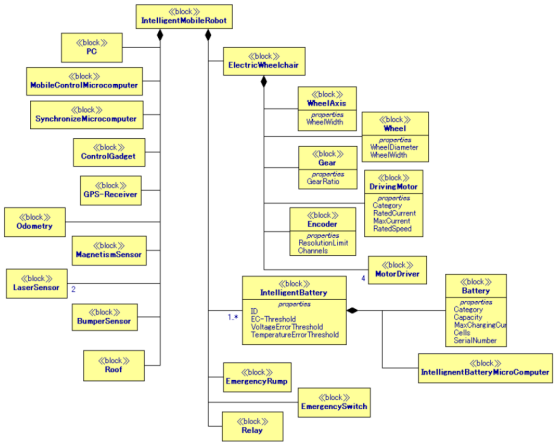


Abb. 2.29: *Lebenszyklus robotischer Systeme (kursiv dargestellt sind mögliche Einsatzmöglichkeiten von Simulationen)*

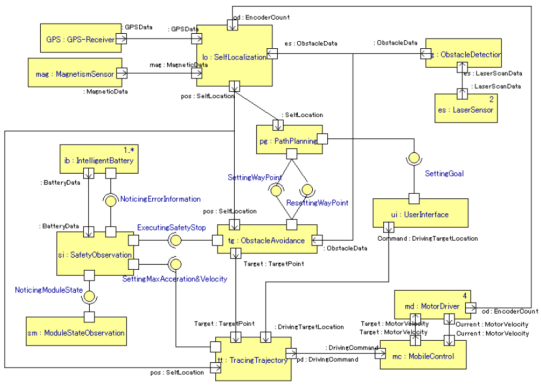
terstützt hierbei durch die Formalisierung in Modellen die verschiedenen Entwicklungsphasen der Anforderungsanalyse, Design, Analyse, Optimierung, Verifikation und Validierung, beginnend in der Designphase, über die gesamte Entwicklung, bis zum finalen Lifecycle-Management.



(a) *Bestandteile von SysML: Anforderungen, Struktur und Verhalten*



(b) *Blockdefinitionsdiagramm (BDD) [136]*



(c) *Internes Blockdiagramm (IBD) [136]*

Abb. 2.30: *SysML für Hard- und Software im Bereich mobiler Roboter*

Zur Beschreibung, Spezifizierung, Analyse, Design und Verifikation komplexer Systeme (in Bezug auf Hardware, Software, Informationen, Personal, Prozeduren, Objekte) hat sich als Standard die graphische Modellierungssprache OMG Systems Modeling Lan-

guage (OMG SysML) [ap] – kurz SysML [137] [138] – herauskristallisiert. Neben der Vereinheitlichung sind die zentralen Ziele von SysML, Komplexität nutzbar zu machen, Risiken zu reduzieren und die Kommunikation zu verbessern. Abbildung 2.30 zeigt die Bestandteile von SysML sowie die beispielhafte Anwendung dieser für Hard- und Software-Entwicklungen eines mobilen robotischen Systems. Als zentrales Tool des MBSE und international akzeptierter Standard soll SysML in dieser Arbeit als Ausgangspunkt der gezeigten strukturellen und konzeptionellen Diagramme dienen. Diese werden mit Hilfe von Enterprise Architect [aq] erstellt.

2.5.1.4 Bestandteile einer 3D-Simulation

Als graphisch basierte Software ist eine 3D-Simulation ähnlich wie ein Computerspiel aufgebaut, legt jedoch einen anderen Fokus auf die verschiedenen Aspekte wie zum Beispiel Realismus, physikalische Plausibilität oder hochauflösende Visualisierung. Jedoch lassen sich in den Bestandteilen aktueller 3D-Simulationssysteme oft Komponenten aus der Unterhaltungsindustrie wiederfinden. Hierbei sind insbesondere sogenannte *Engines* hervorzuheben, welche einen wichtigen, wenn nicht sogar den wichtigsten Bestandteil der 3D-Simulation darstellen.

Der *Main Loop* ist der Kern jeder *Core-Engine*. Für die Synchronisation der zusätzlichen Komponenten ist der *Scheduler* zuständig. Zudem muss jede Engine ein mehr oder weniger ausgefeiltes *Ressourcenmanagement* betreiben, um die Anforderungen der einzelnen Module miteinander in Einklang zu bringen. Technisch gesehen verteilt die Core-Engine die Aufgaben auf CPU und GPU. Ziel ist, möglich voneinander entkoppelte Subkomponenten parallel abzuarbeiten. Gerade im Bereich des Rendering ist dies unumgänglich und im Detail in [139] beschrieben.

Neben der Core-Engine und dem Rendering ist die *Physik-Engine* die dritte essentielle Komponente einer 3D-Simulation. Diese übernimmt die Aufgabe einer realistischen Spielphysik und rechnet meist auf Basis starrer Körper nach Gesetzen der klassischen Mechanik. Beispiele solcher Physik-Engines stellen Bullet [ar], ODE [as], PhysX [at] oder Havok [au] dar. Die Physik-Engine beschäftigt sich mit der physikalischen Korrektheit des Systems und ist im Detail in [140] beschrieben.

2.5.1.5 Übersicht und Klassifizierung verschiedener Simulationssysteme

Simulationssysteme gibt es nahezu unendlich viele. Unterschieden werden können diese anhand ihrer Anwendungen (Maschinenbau, Elektrotechnik, Informationstechnik etc.), dem Abstraktionslevel (d. h., wie viel Know-How ist notwendig, um eine Simulation zu erstellen), dem Detaillierungsgrad usw. Beispiele für derartige Klassifizierungen sind z. B. in [141] zu finden. Im Folgenden liegt der Fokus klar auf Simulationssystemen für robotische Systeme, wobei die Simulatoren hierbei überwiegend eine Starrkörpersimulation als Basis nutzen und für die MRK eine direkte Interaktion mit realen Systemen möglich sein muss.

Auf dieser Basis der „Stages of Use“ werden die nur zur (Roboter-)Entwicklung genutzten Systeme in Kapitel 2.5.1.6 und die zur direkten Interaktion und Kontrolle in Kapitel 2.5.1.7 beschrieben, bevor in Kapitel 2.5.1.8 das Simulationssystem VEROSIM

vorgestellt wird.

2.5.1.6 Simulationssysteme zur Entwicklung von Robotern

Der Fokus liegt hier im Anwendungsgebiet der Robotik, aber auch der allgemeinen Datenanalyse und Auswertung.

2.5.1.6.1 Anwendungsfeld Robotik

Die Robotik ist aufgrund ihrer Komplexität und Vielfältigkeit ein ideales Anwendungsfeld für Simulationen. Einzelne Aspekte oder ganzheitliche Systeme können mit Hilfe von Simulationstechnologien entwickelt werden. Zentraler Bestandteile dieses Anwendungsfeldes stellt der *„sense-think-act“*-Paradigma dar.

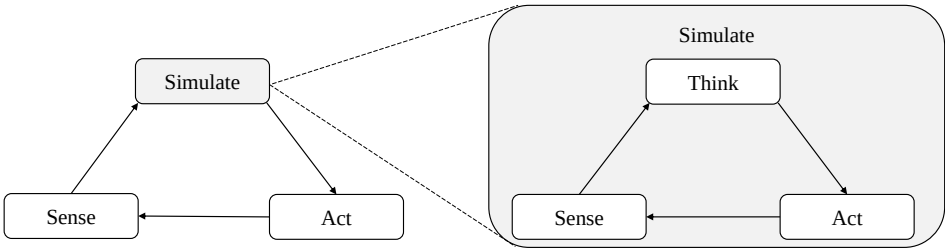


Abb. 2.31: *Von „sense-think-cct“ (the robotic paradigm) zu „sense-simulate-act“*

Dieses Roboter-Paradigma, in Abb. 2.31 dargestellt, zeigt, wie Sensordaten aufgenommen, verarbeitet, im System verteilt und daraus Entscheidungen generiert werden, die anschließend umgesetzt werden. Wie auch schon in [142] kann man das „sense-think-act“-Paradigma durch Ausnutzen der 3D-Simulation in-the-loop auch in „sense-simulate-act“ umwandeln.

2.5.1.6.2 Simulatoren für die Robotik

Gerade im Bereich Robotik existieren mehr oder weniger spezialisierte Simulationssysteme. Erwähnenswerte Software-Tools sind ROBCAD [av], ROBOTEXPERT [aw] sowie PROCESS SIMULATE [ax] von Siemens PLM Software, DELMIA von Dassault Systems [ay] und CIROS [143] (alle genutzt für die Simulation und Entscheidungsunterstützung im Bereich der robotergestützten Automatisierung), EASY-ROB [az] [144] (Robotersimulator bis ins Gelenk-Niveau), V-REP [ba] und GAZEBO [bb] [145] (beide spezialisiert auf Sensoren und bildbasierte Fertigungssysteme), ROBOTRAN (physikbasierte, symbolische Modelliersprache von Starrkörpersystemen [146], meist genutzt für symbolische Regressionsmodelle zur Identifikation von robotertauglichen Systemen wie in [147]) und VEROSIM (3D-Simulationsplattform für eRobotik-Anwendungen [148]). Die Anwendung solcher Simulationstools hat schon zu effizienten und flexiblen Möglichkeiten zur Planung und Optimierung der robotergestützten Fertigung, von Produktionsanlagen, Sensoren usw. geführt.

Abschließend soll hier noch ein weiterer Trend skizziert werden, der auch in den Bereich der Simulatoren gehört. Initiiert durch die Computerspielindustrie haben sich die unterschiedlichen Engines immer weiter auch in der Forschung etabliert. So werden Game- oder

Render-Engines wie Unity [bc], Unreal [bd], Ogre3D [be], CryEngine [bf], oder Blender [bg] mit Physikengines wie Bullet [ar], ODE [as], PhysX [at] oder Havok [au] gekoppelt, um schlussendlich eine visuell ansprechende und physikalisch adäquate Repräsentation eigener Produkte oder Entwicklungen zu erschaffen.

Für weitere Spezialfälle haben sich auch spezielle Simulatoren oder Animationen etabliert. Für deformierbare Strukturen beispielsweise werden entweder die rechenaufwändigen und exakten Finite-Elemente-Methode (von SOLIDWORKS [bh], CATIA [bi], Z88Aurora [bj] usw.), ein numerisches Verfahren zur numerischen Lösung von strukturdynamischen Fragestellungen komplexer Festkörper, oder aber Animationssoftware (wie SideFX Houdini [bk]), welche Sand- oder Wassersimulationen durchführt, aber auch voxelbasierte Systeme wie VoxCAD [bl] zur Simulation zusammengesetzter deformierbarer Körper eingesetzt.

2.5.1.7 Simulationssyteme zur direkten Interaktion mit und Kontrolle von Robotern

Neben den in Kap. 2.5.1.6 präsentierten Simulatoren zur *Entwicklung* von Robotern sollen hier nun Simulatoren zur direkten *Interaktion und Kontrolle* von Robotern vorgestellt werden. Hier gibt es eine große Anzahl von Simulationssystemen, die speziell für Roboter gemacht sind oder gerade in diesem Bereich Anwendung finden. Oft kommen spezialisierte Robotersimulatoren direkt vom Roboterhersteller wie FANUC ROBOGUIDE [bm], ABB RobotStudio [bn] [149] oder KUKA.Sim [bo] [150]. Außerdem gibt es allgemeinere Simulationssysteme wie Msc.Adams [bp] (welches auf reine Starrkörpersimulation fokussiert ist), NI LabVIEW [bq] (ein graphisches Programmiersystem mit Fokus auf Datenerfassung, Datenfluss und Datenverarbeitung), Modelica [br] (das mehr eine Modellierungssprache für physikalische Modelle darstellt) oder Mathworks MATLAB [bs] (welches für die numerische Berechnung mit Hilfe von Matrizen genutzt wird), die alle auch im Bereich der Robotik mit unterschiedlichem Fokus genutzt werden können.

Initiiert als Middleware für Roboter hat auch das Robot Operating System (ROS) einige Simulatoren hervorgebracht, die in der Entwicklerszene rege genutzt werden wie Gazebo [bb] (als klassische Starrkörpersimulator), MORSE (das auch ein Starrkörpersimulationssystem ist und Blender [bg] als 3D-Paket nutzt), Player/Stage [bt] (eine Kombination aus dem Regelungsinterface Player und einem Starrkörperbackend Stage) sowie V-Rep [ba] (eine kommerzielle Entwicklungsumgebung für Roboter).

2.5.1.7.1 3D-Simulationstechnologie zur direkten Steuerung von Robotern

Heutzutage spielt die Simulation bei Kontrollsystemen für teleoperierte Roboterfeldsysteme nur in einigen seltenen und eher begrenzten Fällen eine Rolle. In der Teleoperation wird sie üblicherweise zum Testen und Validieren einzelner Module oder Algorithmen während der Entwicklungsphase verwendet. Die Simulationswerkzeuge konzentrieren sich vor allem auf einzelne Aspekte oder spezifische Anwendungsgebiete wie z. B. ROS Gazebo [145], welches auch in der oben erwähnten DRC angewendet wird. Des Weiteren werden diese Simulationen zwar gerne in der Entwicklung der robotischen Systeme eingesetzt, jedoch endet die Nutzung dieser bei der realen Fertigstellung des Roboters und seiner Benutzung.

2.5.1.8 Das 3D-Simulationssystem VEROSIM

Das Virtual Environment and Robotics Simulation System (VEROSIM) [148] wird am Institut für Mensch-Maschine-Interaktion der RWTH Aachen University als universelle Entwicklungsplattform eingesetzt. Basierend auf *C++* und dem GUI-Toolkit *Qt* zur Entwicklung graphischer Nutzeroberflächen, können hier einzelne Module relativ unkompliziert angepasst und entwickelt werden.

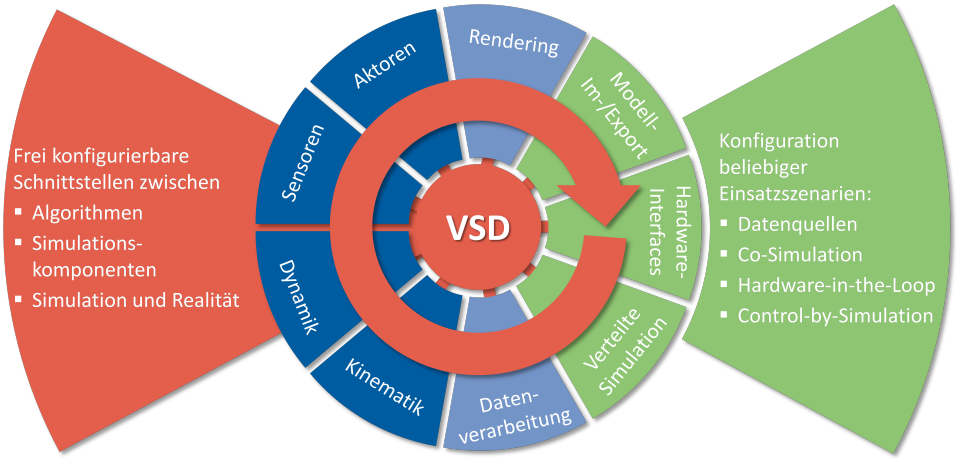


Abb. 2.32: *VEROSIM-Mikrokernel-Architektur zur freien Konfigurier- und Erweiterbarkeit des Simulationssystems auf Basis einer zentralen Datenbank. [151]*

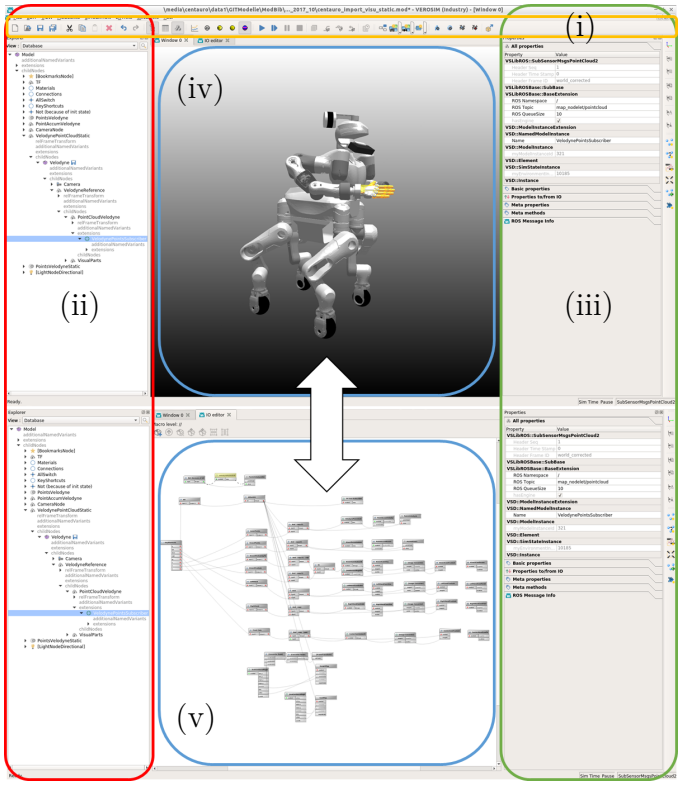
VEROSIM ist ein facettenreiches Simulationssystem, welches das Mikrokernelkonzept verfolgt. Das bedeutet, dass ein kleiner fester Kern die grundlegenden Funktionen (z. B. Kommunikation und Speicherverwaltung) zur Verfügung stellt und alle weiteren Funktionen durch das Hinzufügen von Modulen in Form von Bibliotheken integriert werden. So kann je nach Anwendungsszenario das System um verschiedenste Komponenten erweitert werden (siehe Abb. 2.32). VEROSIM zeichnet sich insbesondere durch eine intrinsische Modularität aus, die für die Realisierung eines VTBs unabdingbar ist. Erweiterungen können als Module leicht in das bestehende Framework integriert werden sowie vorhandene Frameworks ausgebaut werden. Bestehende Schnittstellen können genutzt sowie um weitere ergänzt werden. Des Weiteren kann VEROSIM auch auf verschiedenen Zielplattformen eingesetzt werden wie Windows, Linux oder QNX.

Zentrale Aspekte von VEROSIM umfassen neben der Nutzerschnittstelle die zentrale Datenbank, die Dynamiksimulation, das Sensorframework sowie das graphische 3D-Rendering. Diese Komponenten werden im Folgenden kurz vorgestellt.

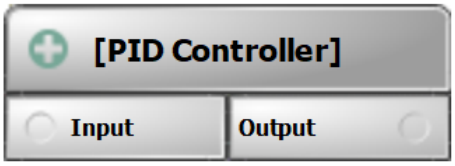
2.5.1.8.1 Frontend

Neben dem Backend – also der Implementierung, Datenhaltung, Datenverwaltung, Prozesskette usw. – spielt vor allem im Bereich der MMI natürlich das Frontend – also die sichtbare Schnittstelle zum Nutzer – eine entscheidende Rolle.

Wie der Abb. 2.33a zu entnehmen ist, besteht die Nutzeroberfläche aus Explorer (links), Eigenschaften/Properties (rechts) sowie dem Renderfenster bzw. dem I/O Board



(a) VEROSIM-Frontend: (i) Toolbar, (ii) Explorer, (iii) Eigenschaften, (iv) Renderfenster, (v) I/O Board



(b) VEROSIM-I/O-Board: Funktionale Gruppen können untereinander verschaltet werden mittels Ein- und Ausgängen unterschiedlicher Datentypen

Abb. 2.33: *Nutzerschnittstelle der 3D-Simulationsumgebung*

in der Mitte. Der Explorer beinhaltet Knoten (engl. Nodes) und Erweiterungen (engl. Extensions), die kaskadiert hierarchisch angeordnet sind. Im gezeigten Beispiel wird auch dem Obermodell mindestens ein Submodell hinzugefügt und im Explorer eingehängt. Im Eigenschaftenfenster (engl. Property Window) können für selektierte Nodes/Extensions die jeweiligen Eigenschaften angezeigt und modifiziert werden. Diese Selektion spiegelt sich auch im I/O Board wieder. Im I/O Board können die Ein- und Ausgänge der verschiedenen Nodes/Extensions miteinander verknüpft werden.

Zum Starten, Pausieren und Stoppen der Simulation gibt es in der Toolbar (oben) die notwendigen Knöpfe, welche natürlich auch erweitert werden können, sowie allgemeine Projektoptionen, die über ein separates Menü eingestellt werden können und allgemeine Einstellungen beinhalten.

2.5.1.8.2 Datenbank

Die VEROSIM Active Simulation Database (VSD) [152] beinhaltet den Kernbestandteil der Mikrokernelarchitektur (vgl. Abb. 2.32). Die VSD ist ein objektorientierter, gerichteter Graph, in dem neben Daten auch Funktionalitäten abgebildet werden können. Des Weiteren erlaubt die VSD mehrere Datenschemata und ist nicht von der Art der Simulation abhängig. Durch Hinzufügen von Softwaremodulen (sog. Extensions) kann die VSD erweitert werden. Neben frei konfigurierbaren externen und internen Schnittstellen lassen

sich sowohl der Simulationstakt anpassen als auch einzelne Module parallelisieren oder sogar auf andere Rechner auslagern. Somit ist es möglich, einzelne Simulationsabläufe zu trennen, parallel berechnen zu lassen und im Rendertakt wieder zusammenzuführen.

Auf dieser Simulationsdatenbank bauen dann unterschiedliche Algorithmen bzw. Softwaremodule auf.

2.5.1.8.3 Starrkörpersimulation

Unter den Simulationsalgorithmen hat die Simulation mechanischer Komponenten in Form einer Starrkörpersimulation zentrale Bedeutung.

Ein starrer Körper ist ein System von diskreten Massepunkten oder dieser wird durch eine kontinuierliche Massenverteilung in einem Objekt angenähert. Die zentrale Eigenschaft eines starren Körpers ist, dass er sich nicht verformen lässt. Auch wenn solche Körper in der Realität nicht existieren, da Verformungen als Resultat von einwirkenden Kräften entstehen, ist die Modellvorstellung eines solchen jedoch sehr nützlich. Für Berechnungen und Simulationen findet dies vielfach Anwendung, da die minimalen realen Verformungen oft vernachlässigbar gering sind oder spezialisierter Verfahren bedürfen. Insgesamt lässt sich die Starrkörpersimulation daher in die Aspekte der *Objektmodellierung*, der *Kinematik* und der *Dynamik* unterteilen. Einfacher gesagt, kann man beliebige Objekte formen, welche sich bewegen, auf externe Kräfte und Momente reagieren, miteinander interagieren sowie Aussagen treffen über die sogenannte inverse Dynamik oder das Berücksichtigen und Einhalten von Zwangsbedingungen.

2.5.1.8.4 Das Modell des starren Körpers

Auf Basis der Theorie von Massenpunkten kann ein ausgedehnter Körper K als Summe aller einzelnen Subvolumeneinheiten (ΔV_i) mit den Massen (Δm_i) angesehen werden. Bei inhomogenen Körpern werden diese Teilkörper mit homogener Massenverteilung mit Hilfe der Massendichte ρ ($\rho = \text{const}$) berechnet.

$$V = \sum_{i=1}^{N} \Delta V_i \tag{2.18}$$

$$M = \sum_{i=1}^{N} \Delta m_i \tag{2.19}$$

$$\Delta m_i = \rho_i \Delta V_i \tag{2.20}$$

Bei zusammengesetzten starren Körpern lässt sich der Massenschwerpunkt S berechnen als

$$r_s = \frac{\sum_{i=1}^{N} r_i \Delta m_i}{\sum_{i=1}^{N} \Delta m_i} \tag{2.21}$$

2.5.1.8.5 Kinematik

Die Bewegung eines solchen starren Körpers in Raum und Zeit, ohne Berücksichtigung von Kräften und Momenten, nennt man *Kinematik*. „Die Bewegung eines ausgedehnten starren Körpers lässt sich immer zusammensetzen aus der Translation des

Schwerpunktes und der Rotation des Körpers um den Schwerpunkt.“ [153] Zur Beschreibung des starren Körpers werden daher sechs Parameter benötigt, drei Ortskoordinaten $p = (x(t), y(t), z(t))$ zur Beschreibung der Translation des Schwerpunktes sowie drei Winkelkoordinaten (ϕ_x, ϕ_y, ϕ_z) zur Beschreibung der Rotation des Körpers bezüglich seines lokalen Schwerpunktkoordinatensystems.

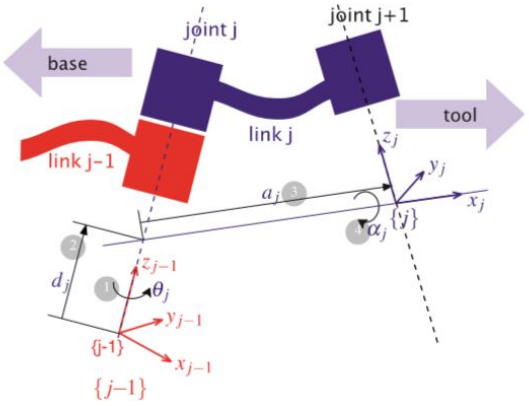


Abb. 2.34: *Denavit-Hartenberg-Parameter* $\zeta_N = K(\mathbf{q}; \theta, \mathbf{d}, \mathbf{a}, \alpha, \sigma)$ *[154]*

Im Bereich der Robotik werden kinematische Ketten oft durch DH-Parameter, d. h. in Denavit-Hartenberg-Konvention [155] angegeben (siehe Abb. 2.34). Hierbei wird diese geometrische Kette aus sogenannten „Links“ (Achsen bzw. Starrkörpern) und „Joints“ (Gelenken) aufgebaut. Ein Starrkörper kann beschrieben werden durch seine Länge $a_j = const$ und Orientierung/Twist $\alpha_j = const$. Gelenke können beschrieben werden durch den Abstand d_j der durch das Gelenk verbundenen Starrkörper-Koordinatensysteme. Der Gelenkwinkel θ_j beschreibt die Rotation eines Starrkörpers relativ zum nächsten, bezogen auf die Gelenkachse ($= z$ Achse). Diese vier Parameter $(\theta_j, d_j, a_j, \alpha_j)$ sind bekannt als DH-Parameter. Eine zusätzliche Konstante σ_j gibt an, ob es sich um ein Drehgelenk ($\sigma_j = 0$) oder um ein Lineargelenk ($\sigma_j = 1$) handelt. Somit ergibt sich die Transformation vom Koordinatensystem $j-1$ zu j als Aneinanderreihung von elementaren Rotationen und Translationen:

$$ {}^{j-1}A_j(\theta_j, d_j, a_j, \alpha_j) = R_z(\theta_j) T_z(d_j) T_x(a_j) R_x(\alpha_j) \tag{2.22} $$

$$ = \begin{pmatrix} \cos\theta_j & -\sin\theta_j \cos\alpha_j & \sin\theta_j \sin\alpha_j & a_j \cos\theta_j \\ \sin\theta_j & \cos\theta_j \cos\alpha_j & -\cos\theta_j \sin\alpha_j & a_j \sin\theta_j \\ 0 & \sin\alpha_j & \cos\alpha_j & d_j \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.23} $$

2.5.1.8.6 Homogene Transformation

Homogene Transformationen sind ein sehr hilfreiches und gebräuchliches Mittel zur Beschreibung von Bewegungen, Posen oder Koordinatensystemwechseln im dreidimensionalen Raum. Dazu vereint eine homogene Transformation die Rotation und anschließende Translation bezüglich eines bestimmten Koordinatensystems. Die Transformationsmatrix

$\mathbf{T}$ wird als Frame bezeichnet und hat folgenden Aufbau:

$$\mathbf{T} = \begin{bmatrix} & \mathbf{R} & & \boldsymbol{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}, \mathbf{R} = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix}, \boldsymbol{p} = \begin{matrix} p_x \\ p_y \\ p_z \end{matrix} \tag{2.24}$$

Diese enthält zum einen die Rotationsmatrix $\mathbf{R}$ – definiert als $\det(R) = 1$ – und zum anderen den Translationsvektor $\boldsymbol{p}$. Die Spaltenvektoren $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$ der Rotationsmatrix $\mathbf{R}$ repräsentieren dabei die Lage der x-, y- und z-Achse des rotierten Koordinatensystems im Ursprungskoordinatensystem.
Der Translationsvektor $\boldsymbol{p}$ definiert die Lage des transformierten Koordinatenursprungs im Ursprungskoordinatensystem. In der Literatur findet man häufig eine Abkürzung der Form $\mathbf{R}(\boldsymbol{v}, \alpha)$, die die Rotationsmatrix für eine Drehung um den Vektor $\boldsymbol{v}$ mit dem Winkel α repräsentiert.

Ein Frame kann auch als Beschreibung einer Pose bezüglich eines bestimmten Koordinatensystems interpretiert werden. In diesem Fall wird eine Rotationsmatrix als eine Orientierungsmatrix aufgefasst, welche die Ausrichtung bezüglich eines Referenzkoordinatensystems beschreibt. Der Translationsvektor gibt die Position in diesem Referenzkoordinatensystem an. Diese Darstellung ist vor allem dann nützlich, wenn neben der Position im Raum auch die Orientierung eines Objektes berücksichtigt und transformiert werden soll.
Die Pose eines Objektes im Raum ist dabei nichts anderes als die Lage des Objektkoordinatensystems relativ zum Weltkoordinatensystem – dieses ist beliebig, aber sinnvoll wählbar. Ist also die Pose eines Objektes in Form eines Frames ${}^{Welt}\mathbf{T}_{Objekt}$ gegeben und die vollzogene Bewegung durch einen weiteren Frame ${}^{Objekt}\mathbf{T}_{ObjektNeu}$ bekannt, kann die neue Pose (definiert durch den Frame ${}^{Welt}\mathbf{T}_{ObjektNeu}$) mit Hilfe einer einfachen Matrixmultiplikation wie folgt berechnet werden:

$${}^{Welt}\mathbf{T}_{ObjektNeu} = {}^{Welt}\mathbf{T}_{Objekt} \cdot {}^{Objekt}\mathbf{T}_{ObjektNeu} \tag{2.25}$$

Interessiert man sich nur für die Position eines Objektes (Punktes) und nicht für seine Ausrichtung, dann reicht die Beschreibung durch homogene Koordinaten anstelle eines ganzen Frames aus. Die homogenen Koordinaten eines Punktes P mit den kartesischen Koordinaten (p_x, p_y, p_z) sind folgendermaßen definiert:

$$\boldsymbol{p}_{homogen} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \tag{2.26}$$

Möchte man hingegen einen Richtungsvektor (z. B. einen Kraftvektor), so geschieht dies über eine reine Rotation ohne Translation. Daher sind die homogenen Koordinaten eines

Richtungsvektors $(v_x, v_y, v_z)^T$ folgendermaßen definiert:

$$\boldsymbol{v}_{homogen} = \begin{bmatrix} v_x \\ v_y \\ v_z \\ 0 \end{bmatrix} \tag{2.27}$$

Die Transformation eines Punktes oder Richtungsvektors, der durch seine homogenen Koordinaten gegeben ist, erfolgt dabei analog zur Transformation einer Pose.

2.5.1.8.7 Dynamik starrer Körper

Die Dynamik starrer Körper beinhaltet alle Bewegungen aufgrund von Kräften (F_i) und Momenten (M_i). Falls die Summe der am Schwerpunkt S einwirkenden Kräfte $F = \sum_{i=1}^{N} F_i$ ungleich Null ist, ändert sich der Impuls $\boldsymbol{p}$ mit der Zeit t und mit dem Impuls auch die Geschwindigkeit $\boldsymbol{v}$.

$$\boldsymbol{F} = m \cdot \boldsymbol{a} = m \cdot \frac{d\boldsymbol{v}}{dt} = \dot{\boldsymbol{p}} \tag{2.28}$$

Bei ausgedehnten Körpern muss u. U. auch der Angriffspunkt P $(P \neq S)$ dieser Kraft mitberücksichtigt werden, da solche Kräfte einen Drehimpuls L hervorrufen können.

$$\boldsymbol{L} = (\boldsymbol{r} \times \boldsymbol{F}) \tag{2.29}$$

Analog zu Geschwindigkeit (v), Impuls (p) und Kraft (F) hat die zeitliche Änderung der Drehgeschwindigkeit (ω) bzw. des Drehimpulses (L) ein Drehmoment (M) zur Folge:

$$\boldsymbol{M} = \frac{d\boldsymbol{L}}{dt} \tag{2.30}$$

Bei genauerer Betrachtung lässt sich diese Analogie auch auf die Energien übertragen. Die kinetische Energie (E_{kin}) wird dann zur Rotationsenergie (E_{rot}), mit

$$E_{\text{kin}} = \frac{1}{2} m v^2 \tag{2.31}$$

$$E_{\text{rot}} = \frac{1}{2} I \omega^2 \tag{2.32}$$

$$\Rightarrow L = I \cdot \omega \tag{2.33}$$

wobei (als Analogon zur Masse) I das Trägheitsmoment darstellt. Das Trägheitsmoment stellt also ein Maß für die Massenverteilung im Körper dar und ist immer auf eine bestimmte Drehachse bezogen.

Wenn sich ein Körper um eine Achse B dreht, die nicht durch S geht, so lässt sich mit Hilfe des *Steinerschen Satzes* das Trägheitsmoment berechnen, wenn man sein Trägheitsmoment I_S bezüglich einer zu B parallelen Achse A mit Abstand a durch den Schwerpunkt S kennt [153]:

$$I_B = I_S + a^2 M \tag{2.34}$$

Anschaulich wird mittels des Trägheitstensors ein Ellipsoid aufgespannt, wobei die Diagonaleinträge die Hauptachsen des Ellipsoids darstellen (vgl. Abb. 2.35).



Abb. 2.35: *Trägheitsellipsoid: Stabilität nur bei kleinstem oder größtem Trägheitsmoment [bu]*

Rotationen um die Hauptträgheitsachsen sind im Prinzip möglich. Jedoch können kleinste Störungen zum „Umkippen“ der Achsen führen. Für $I_a \leq I_b \leq I_c$ ist die Rotation um a und c stabil, jedoch um b instabil.

Bei zusammengesetzten Körpern lässt sich die Gesamtmasse des Starrkörperverbunds aus der Summe der Einzelmassen sehr leicht bestimmen:

$$M = \sum_i m_i \tag{2.35}$$

und der Schwerpunkt als gewichtete Summe der Einzelschwerpunkte gewichtet mit deren Masse, normalisiert mit der Gesamtmasse:

$$\boldsymbol{v}_{CoG} = \frac{1}{M} \cdot \sum_j \frac{1}{m_j} \cdot \boldsymbol{v}^j_{CoG} \tag{2.36}$$

Für den Gesamtträgheitstensor müssen die einzelnen Trägheitstensoren zunächst in Weltkoordinaten umgerechnet werden:

$$\mathbf{I}' = \mathbf{R} \cdot \mathbf{I} \cdot \mathbf{R}^T \tag{2.37}$$

wobei $\mathbf{R}$ die Rotationsmatrix darstellt. Daraufhin können alle Trägheitstensoren mit Hilfe des Steiner'schen Satzes (Gl. 2.34) in den neuen CoG transformiert werden. Schlussendlich werden alle Trägheitstensoren zu einem Gesamtträgheitstensor aufsummiert.

2.5.1.8.8 Kollisionsbehandlung

Im Gegensatz zu frei beweglichen Körpern sowie ruhenden Körpern (die Zwangsbedingungen unterliegen), umfasst die Interaktion mehrerer starrer Körper eine Kollisionsbehandlung in Form von:

1. Kollisionserkennung,
2. Kontaktpunktbestimmung und
3. Kraftberechnung/Kollisionsantwort.

Für eine Simulation mit n Objekten, in der jedes Objekt mit jedem auf Kollision getestet wird, ergeben sich

$$(n-1)+(n-2)+\cdots+1=\frac{n(n-1)}{2}=O(n^2)$$

Operationen. Um den Rechenaufwand zu minimieren, unterteilt man die Kollisionsdetektion in eine „Broad Phase“ und eine „Narrow Phase“. In der Broad Phase werden die Objekte ermittelt, die potentiell kollidieren können und alle anderen ausgeschlossen. In der Narrow Phase wird dann die Kollisionsberechnung durchgeführt, die Kontaktpunktbestimmung, Kontaktzeitbestimmung sowie eine mögliche Eigenkollision betrachtet. Auf Basis von geometrischen *Hüllkörpern* werden sogenannte *AABBs* (Axis-Aligned Bounding-Boxes) genutzt und hierarchisch unterteilt. In der Broad Phase wird der dreidimensionale Raum in kleine Zellen unterteilt und über *räumliches Hashing* Tetraeder, die möglicherweise kollidieren können, in der Hashtabelle vermerkt (siehe auch [156]). In der Narrow Phase wird nun genau ermittelt, welche Objektteile miteinander kollidieren sowie eine präzise Kollisionsdetektion zwischen den Körperteilen, die sich überschneiden, durchgeführt. Daraus werden entstehende Kräfte berechnet, aus denen Zwangsimpulse entstehen, die auf die Starrkörper wirken. Weitere Erläuterungen hierzu finden sich in [140].

2.5.1.8.9 Lagrange-Zwangsbedingungen

Zwangsbedingungen schränken die möglichen Freiheitsgrade von Mehrkörpersystemen ein, wie z. B. durch Kontakte oder Gelenke. In der *inversen Dynamik* werden die für dieses numerische Verhalten notwendigen Kräfte und Momente bestimmt, darauf aufbauend die Bewegungsgleichungen aufgestellt und schließlich (wenn diese sowie alle Kräfte und Momente bekannt sind) die numerische Integration der Bewegungsgleichungen durchgeführt. Abschließend werden die resultierenden Geschwindigkeiten, Positionen und Orientierungen der Körper im System aktualisiert. Die hier verwendete Technik der Lagrange-Zwangsbedingungen ist ein auf Maximalkoordinaten basierter Ansatz (siehe auch [157]).

Die Starrkörpersimulation, die zur Lösung der Bewegungsgleichungen aller involvierten Starrkörper genutzt wird, ist ein Maximalkoordinatenformalismus mittels Lagrange-Faktoren λ_i (mit einer generellen Einführung in [158] und einem Impuls-basierten Starrkörperformalismus in [159]). Im Gegensatz zu anderen Ansätzen führt die Beschreibung von Zwangsbedingungen $C(\boldsymbol{x},t)$ mit Hilfe von Lagrange-Faktoren λ_i bei komplexen Systemen zu einer flexiblen, genauen und stabilen Lösungsform ohne iterative Stabilisierung oder positionsbasierte Projektion. Eine geschwindigkeitsbasierte Zwangsbedingung erhält man durch Ableiten nach der Zeit:

$$\frac{d}{dt}(C(\boldsymbol{x},t))=0 \tag{2.38}$$

$$\Rightarrow \underbrace{\frac{\partial C}{\partial x}}_{J}\dot{\boldsymbol{x}}+\frac{\partial C}{\partial t}=0 \tag{2.39}$$

Unterliegt ein System $m \geq 1$ Zwangsbedingungen, ergibt sich durch deren Ableitungen

eine Jacobimatrix J mit m Zeilen. Grundlage hierbei ist das *d'Alembert-Prinzip*, das darauf beruht, dass Zwangskräfte und -momente in einem mechanischen System keine virtuelle Arbeit leisten. Entsprechend dieses Prinzips resultieren aus den Zwangsbedingungen *Zwangskräfte* $\boldsymbol{f}_c$:

$$\boldsymbol{f}_c = \mathbf{J}_c^T \boldsymbol{\lambda} \tag{2.40}$$

wobei λ die *Lagrange-Multiplikatoren* sind. Mit Hilfe von Newtons Gesetz und d'Alemberts Prinzip folgt für Zwangskräfte:

$$\begin{aligned} \boldsymbol{f} &= m \cdot a \\ \boldsymbol{f} &= \boldsymbol{f}_{\text{ext}} + \boldsymbol{f}_c \\ \Rightarrow M \cdot \ddot{x} &= \boldsymbol{f}_{\text{ext}} + \mathbf{J}_c^T \cdot \boldsymbol{\lambda} \end{aligned} \tag{2.41}$$

Die resultierenden geschwindigkeitsbezogenen Zwangsbedingungen haben die allgemeine Form

$$\mathbf{J}_c \cdot \dot{\boldsymbol{x}} - \boldsymbol{b} = \boldsymbol{a}, \quad \boldsymbol{a} \geq 0 \tag{2.42}$$

Die diskretisierten Bewegungsgleichungen (mit den resultierenden Zwangsimpulsen $\mathbf{J}_c^T \lambda \Delta t$) können daraufhin durch einen Euler-Integrationsschritt von Gl. 2.41 gelöst werden:

$$\begin{aligned} \Rightarrow \dot{x} &= \mathbf{M}^{-1}\mathbf{J}_c^T \cdot \boldsymbol{\lambda} \cdot \Delta t + \mathbf{M}^{-1}\boldsymbol{f}_{\text{ext}} \cdot \Delta t + \dot{x}_0 \\ &= \mathbf{M}^{-1}(\mathbf{J}_c^T \cdot \boldsymbol{\lambda} \cdot \Delta t + \boldsymbol{f}_{\text{ext}} \cdot \Delta t) + \dot{x}_0 \end{aligned} \tag{2.43}$$

$\mathbf{M}$ ist die Masse bzw. Trägheitsmatrix und $\boldsymbol{f}_{ext}$ beinhaltet die externen Kräfte und Momente sowie den gyroskopischen Term der Drehimpulserhaltung.

Um gleichzeitig die Zwangsbedingungen und die Bewegungsgleichungen zu erfüllen, wird (2.43) in (2.42) eingesetzt und man erhält ein „mixed linear complementary problem (MLCP)“ und somit einen Standard, der z. B. mit *Lemkes Algorithmus* gelöst werden kann:

$$\mathbf{J}_c\mathbf{M}^{-1}\mathbf{J}_c^T\boldsymbol{\lambda}\Delta t + \mathbf{J}_c\boldsymbol{f}_{ext}\Delta t + \mathbf{J}_c\dot{x}_0 = \boldsymbol{a} + \boldsymbol{b} \tag{2.44}$$

Das Dynamikframework in VEROSIM wird ausführlich in [140] erläutert.

2.5.1.8.10 Sensorsimulationsframework

Sensoren werden, wie auch Aktoren, im Simulationssystem als Komponenten eingeführt. Für die Kommunikation wird auch hier das I/O Board genutzt, um einen standardisierten Austausch von Dateninput und -output zu erreichen. Die Implementierung verschiedener Sensoren beinhaltet das Sensormodell, Fehlermodelle sowie auch Visualisierungen. Diese werden als abstrakte Zwischenschicht dem Nutzer zur Verfügung gestellt, wobei (abgeleitet davon) die Realisierung und effiziente Implementierung spezifischer Sensoren erfolgt. In dieser konkreten Implementationsschicht können dann entweder Application Programming Interfaces (APIs) realer Sensoren eingebunden oder auch simulierte Sensoren erschaffen werden, die die gleichen Inputs und Sensoroutputs wie das reale Gegenstück annehmen bzw. liefern. Erweiterungen stellen dann die erwähnten Filter, Fehlermodelle

oder Visualisierungen bereit, die nach Bedarf zusätzlich entwickelt und mit Hilfe des I/O Boards verschaltet werden können.

Dieses Sensorframework umfasst u. a. RGB- und TOF-Kameras wie auch Laserscanner (siehe auch [160]). Diese wurden mit Hilfe von realen Sensoren adäquat kalibriert, um eine realitätsnahe Abbildung zu erschaffen und anschließend in diversen Anwendungen genutzt. Im Speziellen wurde auch hier insbesondere auf die Echtzeitfähigkeit der einzelnen Sensorkomponenten geachtet.
Das Sensorframework in VEROSIM wird ausführlich in [161] erläutert.

2.5.1.8.11 Renderingframework

Damit die Simulation ein ernstzunehmender Ersatz zu realen Experimenten sein kann, kombiniert VEROSIM die Starrkörpersimulation mit fotorealistischen virtuellen Welten, welche in Echtzeit berechnet werden. Gravierende Unterschiede in der Renderqualität sind dabei zwischen dem Entertainment- und Engineering-Bereich zu erkennen. Durch die Anwendung und Optimierung aktueller Renderingtechnologien im Bezug auf die 3D-Simulation in [139] wurde ein System geschaffen, das performant die Simulation *konzeptionell* mit dem 3D-Rendering verknüpft.
Das Rendering Framework in VEROSIM wird ausführlich in [139] beschrieben.

2.5.1.9 Validierung der 3D-Simulation

Um die physikalische Genauigkeit von Starrkörpersimulationen zu untersuchen, gibt es unterschiedliche Herangehensweisen. Die Hauptschwierigkeit besteht oft darin, vergleichbare Szenarien in den verschiedenen Tools einzurichten, denn häufig gibt es mehrere Möglichkeiten, die gleiche Situation zu simulieren, was allerdings zu unterschiedlichen Ergebnissen führt, weil ein Simulationsmodell immer Vereinfachungen beinhaltet und diese von Modell zu Modell unterschiedlich sein können. Dabei wird die „Qualität“ der Simulation meistens mit der Rechenzeit, Stabilität oder Recheneffizienz belegt, obwohl die eigentliche Qualität darin liegt, wie die Realität bestmöglich abgebildet wird. Daraus ergibt sich die Idee, diverse *physikalische* Benchmarking-Szenarien zu definieren, um diese

1. mit der analytischen Lösung zu vergleichen (nur möglich in vergleichsweise einfachen Szenarien),
2. mit anderen Simulationswerkzeugen gegenüberzustellen und
3. mit realen Experimenten zu vergleichen.

Hierbei nutzt man validierte physikalische Prozesse als Ausgangspunkt zur Validierung dieser Modelle in der Simulation. Dieser zweistufige Prozess (vgl. Abb. 2.36) ist oftmals einfacher als der direkte Abgleich von Simulationsergebnissen mit dem realen System.

Zunächst auf den Vergleich des Rechenaufwands ausgelegt, untersuchte [162] ein Set an Benchmark-Systemen, welches von [163] um Fügeaufgaben ergänzt wurde. Weitere Benchmarking-Beispiele in diesem Sinne sind in den Arbeiten der ARGESIM-Gruppe [bv] oder in [164] zu finden, alle beschränkt auf CPU-Zeit oder Genauigkeit. Für den vorgestellten zweistufigen Evaluierungsprozess wurden klassische Experimente der Mechanik genutzt, welche sowohl analytisch lösbar, als auch durch reale Experimente schon

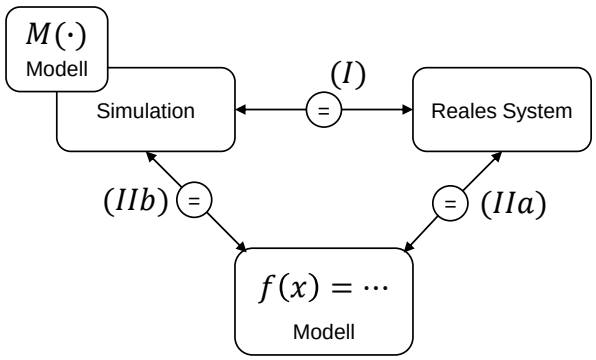


Abb. 2.36: *Validierungsmöglichkeiten: (I) direkte Validierung durch Vergleich mit dem realen System; (II) Nutzen von schon validierten Modellen (IIa) und anschließende Modellierung und Validierung anhand dieser Modelle (in den unterschiedlichen Simulationssystemen)*

im Rahmen klassischer physikalischer Modellbildung behandelt wurden. Beispiele für solche Experimente und daraus resultierten wissenschaftlich belegten und allgemein akzeptierten physikalisch-mathematischen Modellen sind:

- Die Position $z(t)$ einer im Schwerefeld fallenden Kugel ($z(t) = - - \frac{1}{2}gt^2$).
- Auf einer schiefen Ebene wird der kritische Neigungswinkel α gesucht, bei dem ein Körper von statischer zu dynamischer Reibung übergeht ($\alpha = \tan^{-1}(\mu_{stat})$).
- Eine Kugel wird aus verschiedenen Höhen fallen gelassen um die Stoßzahl (Coefficient of Restitution C_R) nach einer Kollision zu ermitteln ($C_R = \sqrt{\frac{h}{H}} = \frac{v_{pre}}{v_{post}}$).
- Untersuchung der Schwingungsdauer (T) und Schwingungsamplitude ($A(t)$) eines ebenen mathematischen Pendels unter Einfluss der Gravitation ($T = 2\pi\sqrt{\frac{l}{g}}(1 + (\frac{\alpha_0}{4})^2 + \cdots)$).

Darauf aufbauend werden in [14] drei Standardexperimente durchgeführt (Schiefe Ebene aus [162], Pendel aus [164] sowie der SCARA-Roboter aus ARGESIM [bv]). Verglichen wird dabei Msc.Adams [165] mit Modelica [br] [166] [167] [168] und VEROSIM. Interessant ist hierbei, dass

- die analytische Lösbarkeit schnell zu komplex wird,
- die Art, ein Modell zu erstellen (also die Modellbildung), enormen Einfluss auf das Simulationsergebnis hat,
- der Prozess der Messung nur in VEROSIM möglich ist,
- die Reibung sehr unterschiedlich modelliert und parametriert wird und
- alle Systeme Vor- und Nachteile aufwiesen, welche durch optimierte Modellierung ausgeglichen werden können.

2.5.1.10 Validierung mit Hilfe der 3D-Simulation

Ebenso wie die im vorherigen Kapitel beschriebene Standardisierung von Evaluationsverfahren von Simulationen gibt es einen solchen Trend auch für reale Roboter. Beachtung

finden hier insbesondere die „KITTI Benchmarks“ und die „DARPA Robotics Challenge (DRC)“ als Beispiele, wie ein erfolgreiches standardisiertes Prüfverfahren für komplexe Systeme aussehen kann. Insbesondere die DARPA Challenge zeigt, schon durch die vorangehende „Virtual Robotics Challenge“ (VRC), den Trend zur Virtualisierung eines solchen Evaluationsverfahrens. Daher wird im Folgenden kurz KITTI und anschließend ausführlicher die DRC beschrieben.

2.5.1.10.1 Kitti Benchmark Suite

Um Bildverarbeitungsalgorithmen zu testen, wurde im Rahmen von [169] und [170] ein Datenset aufgenommen. Ein Fahrzeug mit GPS, Laserscanner und RGB-D-Kameras hat eine bekannte Trajektorie in und um Karlsruhe abgefahren und dabei Daten aufgezeichnet. Auf Basis dieser öffentlich zur Verfügung gestellten Datensätze kann jeder Entwickler eigene Algorithmen testen, kalibrieren und optimieren. Somit haben die Autoren einen Benchmark für die Bildverarbeitung sowie neuronale Netze im Straßenverkehr geschaffen. Anwendung finden diese insbesondere in der Forschung von autonom fahrenden Autos, können aber auch im Bereich der mobilen Robotik genutzt werden.

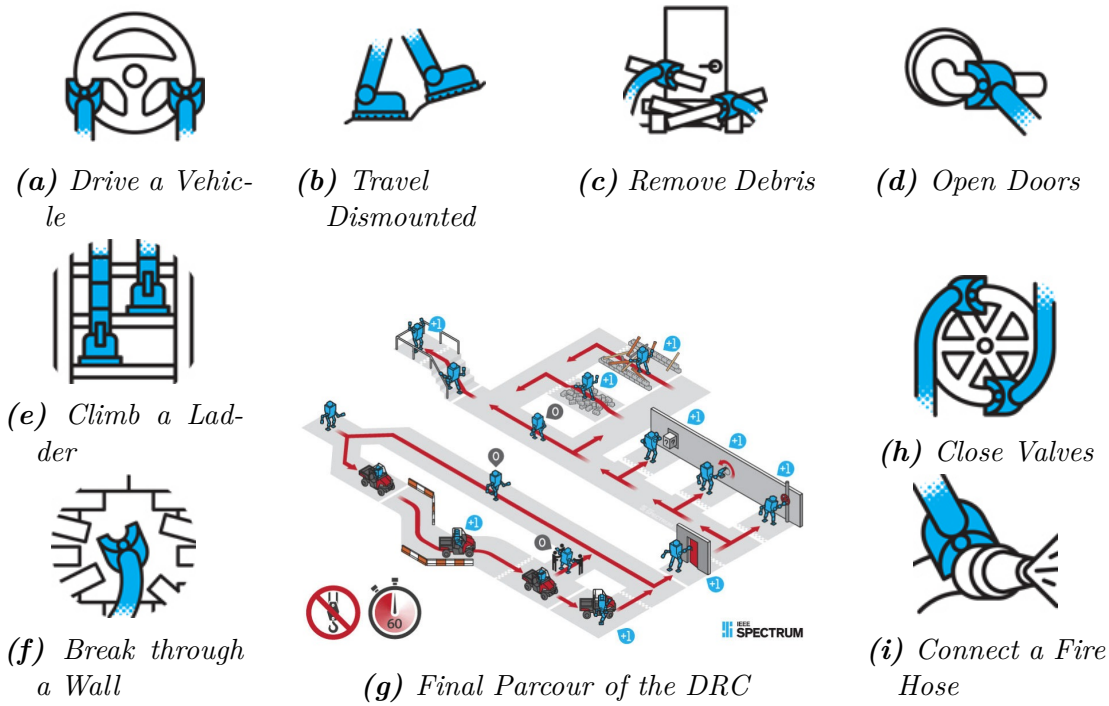
2.5.1.10.2 DARPA Robotics Challenge (DRC)

Als Reaktion auf die spärlichen robotischen Unterstützungsmöglichkeiten im Falle des Fukushima-Unglücks initiierte das US-Verteidigungsministerium im Jahr 2014 eines der ambitioniertesten und größten Forschungs- und Entwicklungsprogramme im Bereich der Robotik. Ziel der DRC war, die Entwicklung der Roboter zu beschleunigen, die Menschen helfen können, sei es bei nuklearen Katastrophen, Feuer, Erdbeben, Chemikalienaustritt oder allen anderen natürlichen oder anthropogenen Katastrophen. Eine „Olympiade“ für Roboter sollte die Fähigkeiten dieser in Bezug auf die verschiedenen Anwendungsszenarien testen, wobei das finale Preisgeld bei $ 2 Mio. lag.

Die Herausforderungen hierbei waren breitgefächert. Zunächst waren Roboter bis dahin meist nur auf eine der geforderten Disziplinen optimiert. So können Hondas Asimo [171] [172] und Südkoreas Hubo [173] laufen oder sogar rennen. Andere wie zum Beispiel der Robonaut der NASA [174] [175] oder Rollin Justin [176] [177] vom DLR können greifen und verschiedene Tools benutzen. PR2 [178] konnte die Umgebung erfassen, kartographieren, fahren und Objekte verschieben. Des Weiteren gab es schon Roboter wie iRobots PackBot [179], welche zur Bombenentschärfung sowohl durch unebenes Terrain manövriert werden, als auch Manipulationsaufgaben ausführen konnten. Das Ziel der DRC war, einen Roboter zu finden, welcher all diese Fähigkeiten vereint, während er in unbekanntem Terrain mit limitierter Kommunikation und Stromversorgung semi-autonom teleoperativ arbeitet. Für das Erreichen dieses Ziels setzte die DARPA ein Zeitfenster von zwei Jahren.

Zunächst als Idee gedacht, finanziell benachteiligte Gruppen teilhaben zu lassen, wurde vorab eine virtuelle DRC, die Virtual Robotics Challenge (VRC), durchgeführt. Im Endeffekt war dies eine Möglichkeit aller Teams, sich gegen starke Konkurrenz vorab zu beweisen und durchzusetzen. Diese Initiative der Virtualisierung trägt auch heute noch Früchte. Die spezifischen Aufgaben der DRC und auch der VRC waren identisch (siehe Abb. 2.37). Jedoch wurde der Umfang in der VRC reduziert und vorgeschrieben, den

ATLAS-Roboter [180] sowie den DARPA Robotics Challenge Simulator (Gazebo [145]) zu nutzen.



(a) Drive a Vehicle *(b) Travel Dismounted* *(c) Remove Debris* *(d) Open Doors*

(e) Climb a Ladder *(h) Close Valves*

(f) Break through a Wall *(g) Final Parcour of the DRC* *(i) Connect a Fire Hose*

Abb. 2.37: *Evaluierung von Robotern: DRC- und VRC-Szenarios und -Aufgaben [bw]*

Insgesamt lag der Fokus insbesondere darauf, auf Unvorhersehbares reagieren zu können. Die DARPA definierte die „key capabilities“, welche die Roboter aufweisen sollten, als:

- „Mobility and dexterity to maneuver in the degraded environments typical of disaster zones“;
- „Ability to manipulate and use a diverse assortment of tools designed for humans“;
- „Ability to be operated by humans who have had little to no robotics training“;
- „Partial autonomy in task-level decision-making based on operator commands and sensor inputs“.

Die DRC testete all diese Fähigkeiten, jedoch primär *mobility*, *manipulation* und *dexterity* [b].

Schlussendlich sah man durch die Ergebnisse der DRC, dass Roboter eine Vielzahl nützlicher Dinge tun können, wie das Benutzen von Tools, Auto fahren oder Treppen steigen – oft auch erfolgreicher, als zu erwarten war. Auf der anderen Seite konnte man die zusätzlichen Herausforderungen erkennen, die für Menschen so einfache Aufgaben, wie eine Tür zu öffnen, für Roboter mit sich bringen. Auch die Art der Roboter unterschied sich stark. Neben bipedalen humanoiden Robotern nahmen auch quadrupedale reifenbasierte Roboter teil sowie ein affenartiger Roboter, welcher als einziger die Eigenschaft besaß, sich nach einem Fall selbst wieder aufrichten zu können. Interessant war, dass vor allem die nicht-humanoiden transformierbaren Roboter mit am besten abschnitten.

Diese oder solche „Challenges“ im Allgemeinen sind ein Motor aktueller technologischer Prozesse und haben die mobile Robotik weit vorangebracht. Des Weiteren wurde durch diese Challenge eine Art Benchmark mit definiert, der genutzt werden kann, um weitere Entwicklungen im Bereich der Robotik vergleichbar zu gestalten. Solche Aufgabensets oder ganze Einsatzszenarien werden auch von weiteren solcher Wettbewerbe (wie z. B. RoboCup Rescue [181] [182] [183]) mitgetragen, erweitert oder modifiziert. Und auch hierbei haben sich Simulationsligen etabliert [184], die seit 2001 genutzt werden, um auch virtuell Desastereinsätze zu proben und zu optimieren.

2.5.2 Der Digitale Zwilling (DZ)/Digital Twin (DT)

Als feste Bestandteile der Industrie 4.0 (I4.0) [16] haben sich neben der weiter fortschreitenden Digitalisierung auch die Software und der Digitale Zwilling etabliert. Eine Übersicht über verfügbare Literatur, Definitionen und Nützlichkeit von DTs ist in [185] gegeben. Auch dort erkennt man, dass der „Digitale Zwilling“ kein eindeutig belegter Begriff ist. Im Rahmen dieser Dissertation wird die Sichtweise verfolgt, dass der DT ein virtuelles Ebenbild einer realen Maschine bzw. eines (physischen) Assets bezeichnet, welcher eine Schlüsselrolle einnimmt, um cyber-physische Systeme zu entwickeln und genutzt wird, um das Internet of Things (IoT) zu formen.

2.5.2.1 (Physisches) Asset/Realer Zwilling (RZ)/Real Twin (RT)

Digitale Zwillinge sind virtuelle digitale Repräsentanten *physischer Assets* bzw. Realer Zwillinge (RT/RZ). Ein physisches Asset (siehe Def. 23) umfasst z. B. Maschinen, Fahrzeuge, Häuser oder Bäume, aber auch z. B. Menschen oder Softwarekomponenten.

Definition 23: Asset

Ein Asset ist eine Entität, die einen wahrgenommenen oder tatsächlichen Wert für eine Organisation hat und der Organisation gehört oder von ihr individuell verwaltet wird. [a]
Nach [186] ist das physische Asset wiederum ein Gegenstand oder Artefakt, das speziell zur Ausführung einer bestimmten Rolle in einem bestimmten System gedacht ist.

2.5.2.2 Digitaler Zwilling (DT)

Die gesamte Datenhaltung und der Datenfluss der Assets untereinander sowie das Abbild des Gesamtprozesses wird hierbei manchmal auch als „Digitaler Schatten“ (Def. 24) bezeichnet.

Definition 24: Digitaler Schatten

Der digitale Schatten als ein Handlungsfeld von Industrie 4.0 ist das hinreichend genaue Abbild der Prozesse in der Produktion, der Entwicklung und angrenzenden Bereiche mit dem Zweck, eine echtzeitfähige Auswertungsbasis aller relevanten Daten zu schaffen. Im Einzelnen gehört dazu die Beschreibung der notwendigen Datenformate, der Datenauswahl und der Datengranularitätsstufe [187].

Daraus ergibt sich dann schlussendlich das Gesamtkonstrukt des Digitalen Zwillings („Digital Twin“ (en)) gemäß Def. 25.

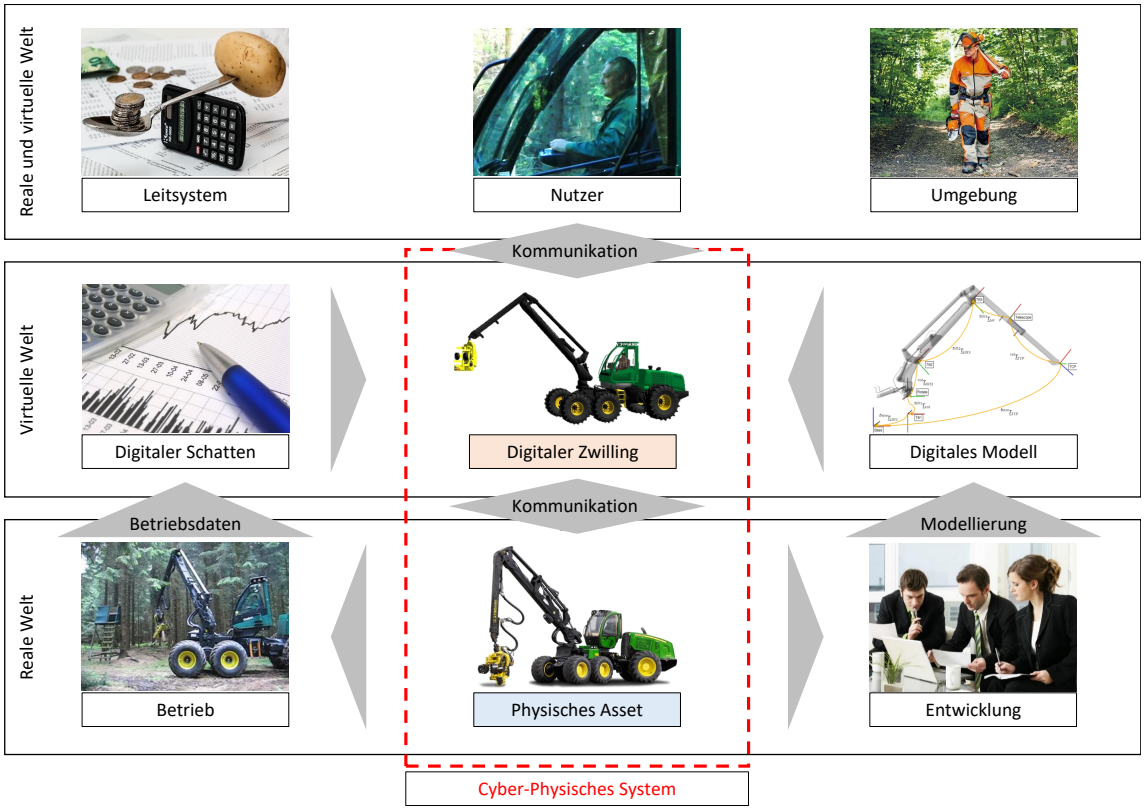


Abb. 2.38: *Der Digitale Zwilling: Mediator zur Entwicklung und Führung von Forstmaschinen (basierend auf [15])*

Wie in Abb. 2.38 zu erkennen, kann der Digitale Zwilling in unterschiedlichen Zusammenhängen Verwendung finden:

1. Entwicklung komplexer Systeme mit Hilfe und auf Basis des Digitalen Zwillings,
2. Optimierung des digitalen Modells mit realen Betriebsdaten,
3. Steuerung komplexer Systeme mit dem Digitalen Zwilling „in-the-loop“,
4. intuitive Interaktion mit komplexen Systemen durch den Digitalen Zwilling,
5. Erschaffen cyber-physischer Systeme durch die Kopplung von Realem und Digitalem Zwilling.

Definition 25: Digitaler Zwilling

Der Digitale Zwilling ist die virtuelle digitale Repräsentanz eines physischen Assets. [a]

Ein Digitaler Zwilling besteht mindestens aus dem Asset-System und kann erweitert werden um folgende Bestandteile:

- Benutzerschnittstelle
- Datenverarbeitungssystem
- Asset (Sensor, **System**, Aktor)
- Kommunikationsinfrastruktur

Aufgrund dieser Allgemeinheit des DTs soll dieser noch genauer in den nächsten 2 Kapiteln beleuchtet werden.

2.5.2.3 Struktur und Funktion des Digitalen Zwillings

Obwohl die Repräsentation des Digitalen Zwillings (DT) oft mit einem Computer-Aided-Design-(CAD-)Modell verwechselt wird, ist der DT in Wirklichkeit wesentlich komplexer. Der Irrtum liegt darin, das Modell des DT mit einem herkömmlichen Simulationsmodell zu verwechseln. Damit ein DT als solcher zählt, braucht er zwingend ein *physisches Gegenstück*, mit dem der DT *interagiert*.

DTs werden immer noch auf der Basis von CAD-Modellen und Animationen sowie Einzeldomänen-Simulatoren entwickelt. Eine ganzheitliche Lösung, die Domänen, Anwendungen und vor allem das virtuelle und reale System verbindet, fehlt noch.

Einfach gesagt, ist der Digitale Zwilling (DT) eine Abbildung einer Maschine (eines physischen Assets) in der Simulation und wird daher auch in [a] als „Simulationsmodell“ definiert, bietet aber viel mehr Möglichkeiten als das, was klassisch unter Simulationsmodell verstanden wird. Um zu begreifen, was ein Digitaler Zwilling genau ist und welche Möglichkeiten dieser bietet, soll hier zunächst auf die definierenden Eigenschaften, daraufhin auf die ergänzenden Eigenschaften und schlussendlich auf die Anwendbarkeit eingegangen werden.

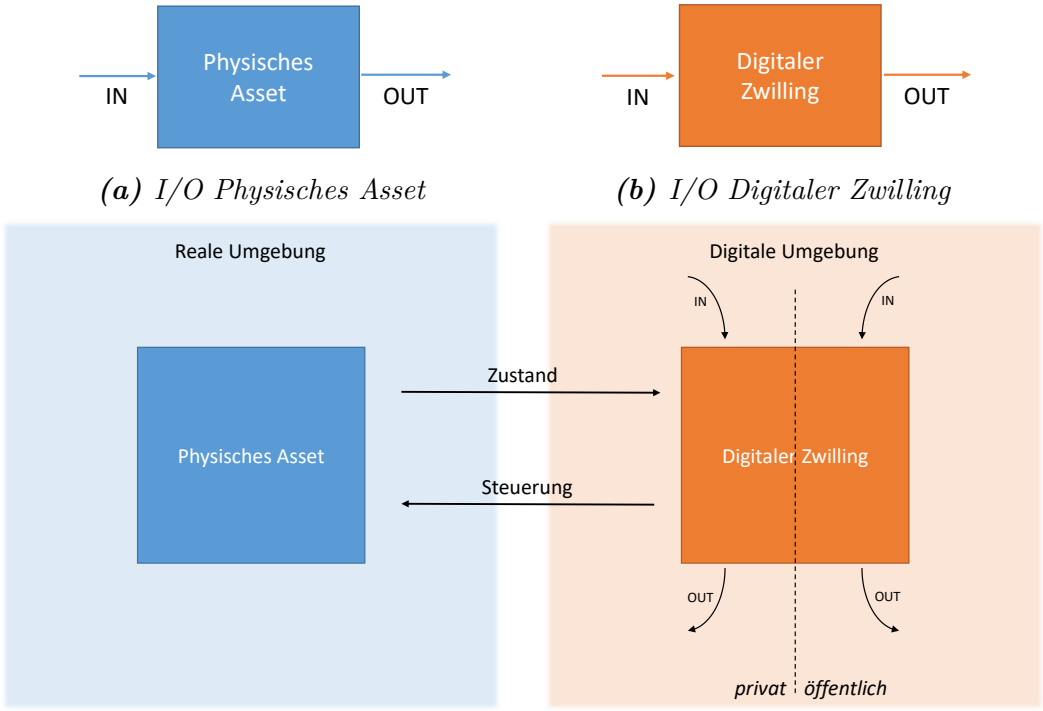
Als definierende Eigenschaften für einen Digitalen Zwilling lassen sich zwei zentrale Punkte festhalten:

- Einen Digitalen Zwilling gibt es nur, wenn es auch einen Realen Zwilling gibt[3] und
- Digitale Zwillinge bilden Reale Zwillinge hinsichtlich Struktur und Verhalten 1-zu-1 ab.

Die Komplexität dieser „Abbildung“ kann natürlich beliebig gewählt werden. Wie auch die Level der Systemspezifikation in der Systemtheorie ist neben der Struktur insbeson-

[3] Bei der Entwicklung Digitaler Zwillinge ist es oft so, dass auch bevor der Reale Zwilling existiert mit Hilfe eines Digitalen Zwillings simuliert und experimentiert wird. Auch diese Vorstufen eines finalen Digitalen Zwillings, welche nur virtuell existieren, werden im Folgenden als DTs bezeichnet, d. h. wenn die Existenz oder die Absicht der Existenz des Assets vorliegt.

dere das I/O-Verhalten des Systems (bzw. von Systemkomponenten) interessant (vgl. Abb. 2.39).



(a) I/O Physisches Asset *(b) I/O Digitaler Zwilling*

(c) Digitaler Zwilling und physisches Asset: technischer Aufbau und Verbindung

Abb. 2.39: *Der Digitale Zwilling: Abbildung eines physischen Assets*

Wie auch das reale System entwickelt sich der Digitale Zwilling weiter und kann zu jedem Zeitpunkt des Lebenszyklus in unterschiedlicher Ausprägung Anwendung finden. Insbesondere folgende modular ergänzbaren Eigenschaften Digitaler Zwillinge machen diesen zu einem mächtigen Werkzeug, welches ebenbürtig zum realen System eingesetzt werden kann und dessen Anwendung Zugang zu einer großen Bandbreite an Möglichkeiten öffnet (basierend auf [bx]):

- Struktur
  - Der Digitale Zwilling kann von simplen Abbildungen bis zu kompletten interagierenden (virtuellen) Welten reichen.
  - Die Abbildung Digitaler Zwillinge funktioniert auch hierarchisch und kaskadierbar.
  - Über Kommunikationsschnittstellen werden Digitale Zwillinge zu Knoten des Internets der Dinge und Dienste (IoT).
  - Digitale Zwillinge untereinander sowie Digitale Zwillinge und physische Assets können verbunden sein bzw. eine Verbindung herstellen.
- Daten
  - Digitale Zwillinge beschreiben das physische Asset umfassend über Metadaten, digitale Modelle und Betriebsdaten.

  - Digitale Zwillinge kennen ihre Historie, ihren aktuellen Zustand und alternative Wege zum Ziel und können so ihr Verhalten prognostizieren.
  - Der Digitale Zwilling trennt zwischen privaten und öffentlichen Daten und Prozessen.
- Funktion
  - Digitale Zwillinge strukturieren und integrieren den (Entwicklungs-)Prozess. Von der Idee über Spezifikation, Analyse, Optimierung, Test, Verifikation & Validierung unterschiedlicher Realisierungen bis zur Inbetriebnahme basieren alle Arbeiten und Entscheidungen auf gemeinsamen Digitalen Zwillingen, die aus unterschiedlichen Perspektiven betrachtet werden.
  - Digitale Zwillinge sind interdisziplinär, domänen-, system-, prozess- und anwendungsübergreifend.
  - Der Digitale Zwilling ist oft mit modellbasiertem Denken verbunden, welches reale Daten mit einem systemtechnischen Modell eines physischen Produkts und dessen gesamtem Lebenszyklus verknüpft.
  - Digitale Zwillinge realisieren intelligente Systeme.

Der Digitale Zwilling ist somit ein Tool auf Systemebene zur Realisierung cyber-physischer Systeme und ist daher Grundbaustein der Industrie 4.0. Diese Sichtweise spiegelt sich auch in Def. 25 wieder.

2.5.2.4 Modell und Simulation des Digitalen Zwillings

Basierend auf dieser Definition Digitaler Zwillinge sowie dem Formalismus aus [188] kann der Übergang vom Simulationsmodell zum DT-Modell wie folgt definiert werden.
Sei ein Modell M mit Eingang $\underline{\mathbf{u}}(t)$ und Ausgang $\underline{\mathbf{y}}(t)$ eine Abbildung in Form von

$$\underline{\mathbf{y}}(t) = M(\underline{\mathbf{u}}(t)) \tag{2.45}$$

so ist das „naive Ziel“ von Simulationen, dass der Ausgang des realen (R) Modells ($\underline{\mathbf{y}}^{\mathrm{R}}(t) = M^{\mathrm{R}}(\underline{\mathbf{u}}^{\mathrm{R}}(t))$) identisch zum simulierten, Digitalen-Zwilling-(DT-)Modell ($\underline{\mathbf{y}}^{\mathrm{DT}}(t) = M^{\mathrm{DT}}(\underline{\mathbf{u}}^{\mathrm{DT}}(t))$) ist und gilt:

$$\left\| \underline{\mathbf{y}}^{\mathrm{R}}(t) - -\underline{\mathbf{y}}^{\mathrm{DT}}(t) \right\| \to 0 \tag{2.46}$$

Jedoch ist dies eben nicht das Hauptziel von DTs. Vielmehr ist es vor allem wichtig, dass Algorithmen (A_i), die Ausgangsdaten des DTs ($\underline{\mathbf{y}}^{\mathrm{DT}}(t)$) nutzen, die gleichen Ergebnisse liefern als wenn sie Daten vom physischen Asset ($\underline{\mathbf{y}}^{\mathrm{R}}(t)$) nutzen würden. Somit gilt für beliebige Algorithmen $\underline{\mathbf{z}}(t) = A(\underline{\mathbf{y}}(t))$, dass

$$\left\| A_i(\underline{\mathbf{y}}^{\mathrm{R}}(t) - A_i(\underline{\mathbf{y}}^{\mathrm{DT}}(t)) \right\| \to 0 \tag{2.47}$$

welches das eigentliche Hauptziel Digitaler Zwillinge darstellt. Analog verhält es sich mit Steuerungsmodulen, welche die gleichen Schnittstellen zu physischem Asset und Digitalem Zwilling nutzen.

2.5.3 Das Virtuelle Testbed (VTB)

Auch wenn es zunächst etwas verwirrend erscheinen mag, wird im Folgenden zunächst in diesem Abschnitt 2.5.3.3 davon ausgegangen, dass es ein solches VTB schon gibt und man dieses nutzt, bevor in Abschnitt 2.5.3.4 auf den Entwurf und die Entwicklung der VTBs selber eingegangen wird. Dieses Vorgehen soll zu einem besseren Zugang und einem intuitiveren Verständnis der Zusammenhänge und Abläufe beitragen.

2.5.3.1 Das VTB als Laufzeitumgebung Digitaler Zwillinge

Das Virtuelle Testbed (VTB) ist Teil der „eRobotics“-Methodologie. Basierend auf den Entwicklungen in [189], [190], [191] und [192] wurde das Feld der eRobotics in Anwendungen im Bereich Industrie, Weltraum und Umwelt ausführlich diskutiert und erfolgreich genutzt. Hierbei beinhaltet das VTB sowohl die Visualisierung/Animation, die Simulation einzelner physikalischer Prozesse, als auch die Modellierung, Simulation und Datenverarbeitung der zu entwickelnden Systeme in der finalen Anwendungsumgebung (siehe Abb. 2.40).



Abb. 2.40: *Von reiner Visualisierung zu Virtuellen Testbeds am Beispiel der Forstmaschinen-, Umwelt- sowie Sensorsimulation*

Anwendung finden VTBs und die eRobotics-Methodik u. a. hier:

- In [151] wird ein genereller Überblick über eRobotics und die Bestandteile der zugrunde liegenden 3D-Simulationsarchitektur gegeben.
- Eine etwas andere Herangehensweise zeigt [139], wobei der Übergang von Visualisierung zu performanter 3D-Simulation im Bereich des Renderings aufgezeigt wird.
- Im Bereich Umwelt zeigt [193] den „Virtuellen Wald“, der sowohl biologische als auch technische Prozesse in Betracht zieht, um Mechanismen in diesem Forschungsfeld zu untersuchen und schlussendlich auch die Waldarbeiter in ihrer Arbeit zu unterstützen.
- Anwendung im Weltraum finden VTBs u. a. in [194]. Hier wird ein mobiler Roboter mit Hilfe der 3D-Simulation entwickelt und zur extraterrestrischen Kraterexploration genutzt.
- In [188] wird eRobotics um die Methodiken der „Simulation-based X“ erweitert und im Rahmen von Weltraumanwendungen eingesetzt, in diesem Fall der Trajek-

torienoptimierung von teleoperierten Laufrobotern oder auch Satelliten-Docking-Szenarien.
- [88] zeigt neue Regelungsalgorithmen industrieller Roboter, um ein intrinsisch sicheres System auf Basis nachgiebiger Gelenke zu erschaffen.
- In [195] werden VTBs eingesetzt, um für klein- und mittelständische Unternehmen konfigurierbare Roboterzellen zu entwickeln und zu überwachen. Zentraler Forschungsgegenstand sind hierbei vor allem die Programmierung und Regelung der Roboter sowie die Simulation, Auswahl und Positionierung sensorischer Systeme.
- Ein Überblick über die Möglichkeiten, die eRobotics-Methodik auch im Rahmen von Industrie 4.0 im Bereich Robotik und Automatisierung einzusetzen, ist in [196] aufgezeigt.

Die Anwendbarkeit beschränkt sich dabei eben nicht auf klassische industrielle Roboteranwendungen (wie z. B. Fertigungsstraßen), sondern beinhaltet vielmehr ein allgemeines Konzept, komplexe Systeme zu verstehen und zu entwickeln.

2.5.3.1.1 Definition des VTBs

Im Rahmen der „Industrie 4.0 (I4.0)“-Bewegung sowie dem Wunsch nach „Internet of Things (IoT)“ bekam der Digitale Zwilling (siehe Def. 25) eine wichtige Bedeutung. Der Digitale Zwilling als Repräsentation des realen Systems kann dann auch im VTB genutzt und eingesetzt werden. So definierte [142] das VTB noch wie folgt: „Ein Virtuelles Testbed (VTB) ist ein Software-System zur system-, disziplin- und anwendungsübergreifenden Entwicklung komplexer Systeme. Es stellt Methoden zum reproduzierbaren Test sowie zur Verifikation und Validierung dieser Systeme unter frei wählbaren Randbedingungen bereit. Hierzu werden Systemmodelle, Simulationsalgorithmen, datenverarbeitende Algorithmen und reale Systeme für die ganzheitliche Betrachtung des dynamischen Gesamtsystems einschließlich aller Wechselwirkungen in gewählter Detaillierung integriert.“

In Kombination mit dem Digitalen Zwilling lässt sich dies sehr gut zu einem Gesamtsystem vereinen (siehe Def. 26). Wie in Abb. 2.41 am Beispiel eines mobilen Roboters gezeigt, fügt sich der Digitale Zwilling in die 3D-Simulation ein, wird erweitert um ein Umgebungsmodell (welches wiederum ein Digitaler Zwilling der realen Umgebung sein kann) und eingebettet ins VTB.

Insgesamt erschaffen wir innerhalb des VTBs mit Hilfe der 3D-Simulation Digitale Zwillinge realer (physischer) Assets. Ein Digitaler Zwilling (z. B. Robotermodell, Umgebungsmodell, Objekte etc.) wird (wie in [132] definiert) daraufhin als experimentierbares Modell genutzt.

Definition 26: Virtuelles Testbed (VTB)

Ein Virtuelles Testbed ist somit ein ganzheitliches System aus 3D-Simulation, Datenhaltung, Schnittstellen und Tools, das erlaubt, Digitale Zwillinge zunächst in der virtuellen Realität zu entwickeln/testen/optimieren und dann gewonnene Erkenntnisse, aber auch Algorithmen und Modelle auf das reale System direkt oder indirekt zu übertragen. Zusammengefasst ist das VTB daher eine *Experimentierumgebung für Digitale Zwillinge*.

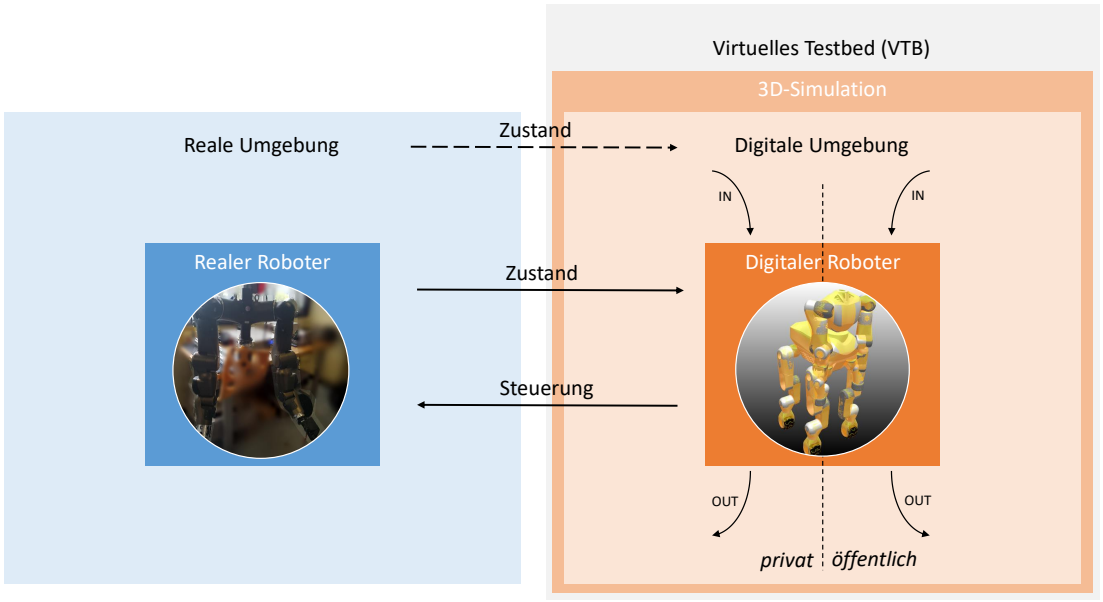


Abb. 2.41: *Digitaler Zwilling: Einbettung in das VTB am Beispiel eines mobiles Roboters (vgl. Abb. 2.39c)*

Somit wird das Virtuelle Testbed zur Laufzeitumgebung für Digitale Zwillinge. In diesem können DTs miteinander und mit den jeweiligen realen Zwillingen interagieren und werden zu *einem Netzwerk aus interagierenden DTs*. Im VTB geschieht dies auf Basis der Multi-Domänen-Simulation (vgl. [142]).

2.5.3.1.2 Das VTB als Netzwerk interagierender Digitaler Zwillinge

Zur Verdeutlichung dieses Netzwerks aus DTs wird hier kurz auf das Beispiel des autonomen Fahrens (vgl. Abb. 2.42) eingegangen. Details zur Umsetzung eines solchen DTs bzw. eines solchen Gesamtszenarios werden später erläutert.

Bei der Entwicklung eines autonom fahrenden Autos geht es um das physische Asset des Autos an sich, aber auch um die verschiedenen Sensoren des Autos, die Software und Algorithmen zur Steuerung, Bildverarbeitung etc. Dieses komplexe System des autonomen Autos kann man auch als Digitalen Zwilling abbilden. Dieser DT beinhaltet alle Bestandteile, die auch das physische Asset beinhaltet. Dies bedeutet auch, dass der DT auch Sub-DTs wie z. B. einen Laserscanner als Sensor beinhaltet, welche selber wiederum ein komplexes System darstellen. Außerdem können standardisierte Schnittstellen (aus dem Automotive-Sektor) genutzt werden, um Daten zwischen DT und physischem Asset auszutauschen. All dies führt zu einem ganzheitlichen Digitalen Zwilling.

Zusätzlich dazu ist es aber auch nötig, weitere Objekte der Szene hinzuzufügen, um ein Gesamtszenario zu definieren. In diesem Fall die Straße, auf der das Auto fahren soll, Ampeln oder Straßenschilder, die von der Sensorik erkannt werden sollen, und auch andere dynamische Objekte, beispielsweise Autos. Durch die Beschreibung Digitaler Zwillinge (vgl. Def. 25) lassen sich auch diese Objekte als Digitale Zwillinge beschreiben. Hierbei



(a) Der DT des Autos im VTB *(b) Google Maps*

Abb. 2.42: *Das VTB als Netzwerk interagierender DTs am Beispiel Automotive. Der (ausgezeichnete) DT des zu untersuchenden Objekts Auto (1) interagiert direkt mit dem DT der Straße (2) und indirekt mit den DTs der Ampel (3) und dem entgegenkommenden DTs des LKW (4) [197]*

stellt sich heraus, dass die Straße einen Grenzfall der DTs darstellt, da diese wirklich ausschließlich als statisches Modell vorliegen muss, ohne jedwede Intelligenz, Kommunikationsschnittstelle, Datenhaltung etc. Die Ampel hingegen kann ein DT sein, der auch als Knoten im Internet der Dinge fungiert und mit seinem realen Gegenstück verbunden sein kann. Aus all dieses DTs entsteht ein Netzwerk, welches im VTB umgesetzt und „zusammengehalten“ wird. Erst im Gesamtszenario lassen sich dann einzelne Aspekte wie z. B. der Rad-Straßen-Kontakt oder das Sensorverhalten, Objekterkennung etc. genauer untersuchen.

Analog zu *„Everything is a file“* unter Linux [198] oder *„Everything is an object“* für Python ist man versucht in diesem Fall zu sagen *„Everything (in a VTB) is a Digital Twin“*.

2.5.3.2 Systementwicklung auf Basis Digitaler Zwillinge

In der 3D-Simulation lassen sich Digitale Zwillinge entwickeln, untersuchen und optimieren. In diesem Zusammenhang ist insbesondere das Simulation-based Systems Engineering (SBSE) ausschlaggebend.

- **Simulation-based Systems Engineering (SBSE):** SBSE umfasst die Entwicklung ganzheitlicher Systeme mit Hilfe der 3D-Simulation. Dies beinhaltet die Idee, das Design, die Programmierung sowie Test, Validierung und Verifikation der einzelnen Komponenten wie auch des gesamten Systems.

Bei diesem Prozess einer ganzheitlichen Entwicklung auf Systemebene werden oft auch Methoden aus der Simulationstechnik eingesetzt. Solche simulationsbasierten Methoden können auch mit Hilfe des Anglizismus Simulation-based X (SBX) beschrieben werden. Die zentralen SBX-Methodiken werden im Folgenden beschrieben:

- **Simulation-based Control (SBC):** SBC beschreibt die direkte Steuerung, Regelung und Kontrolle robotischer Systeme aus der Simulation heraus.

- **Simulation-based User Interfaces (SBUI):** Mit dem Oberbegriff der SBUI ist das Feld der direkten Nutzerinteraktion mit Systemen mittels der Simulation gemeint.
- **Simulation-based Operator Interfaces (SBOI):** SBOI ist ein Spezialfall der SBUI, da auch hier die direkte Interaktion mit dem realen System im Mittelpunkt steht. Primär geht es um eine direkte Kontrolle akut echtzeitrelevanter Operationen mit dem System. Ein Beispiel ist hier die direkte Interaktion mit mobilen Explorationsrobotern.
- **Simulation-based Support (SBS):** SBS zeigt Möglichkeiten, die Simulation in-the-loop zu nutzen, um damit den Nutzer zu unterstützen.

2.5.3.3 Entwicklung komplexer Systeme mit Hilfe von VTBs

Komplexe Systeme werden aufgrund ihrer inhärenten Komplexität oft zunächst als Modelle beschrieben. Jedoch endet die analytische Lösbarkeit oft sehr schnell. Zu diesem Zweck werden in allen Bereichen der Forschung schon verschiedenste Simulationsprogramme eingesetzt. Mit Hilfe von VTBs ist die komponentenweise sowie die ganzheitliche Systembetrachtung möglich. Dies kann je nach Anwendung die Modellierung, die 3D-Simulation sowie auch die (visuelle) Programmierung beinhalten.

2.5.3.3.1 Funktionsweise von VTBs

Zum Verständnis der Funktionsweise Virtueller Testbeds wird im Folgenden anhand verschiedener Beispiele die Nutzung von VTBs und somit die Entwicklung komplexer Systeme mit Hilfe dieser erläutert. Wie auch in der Forschung (siehe auch Abb. 2.43) wird mit einem leeren Blatt Papier, bzw. in der 3D-Simulation einem leeren Raum, begonnen und dieser dann mit Inhalt gefüllt (vgl. Abb. 2.43a).

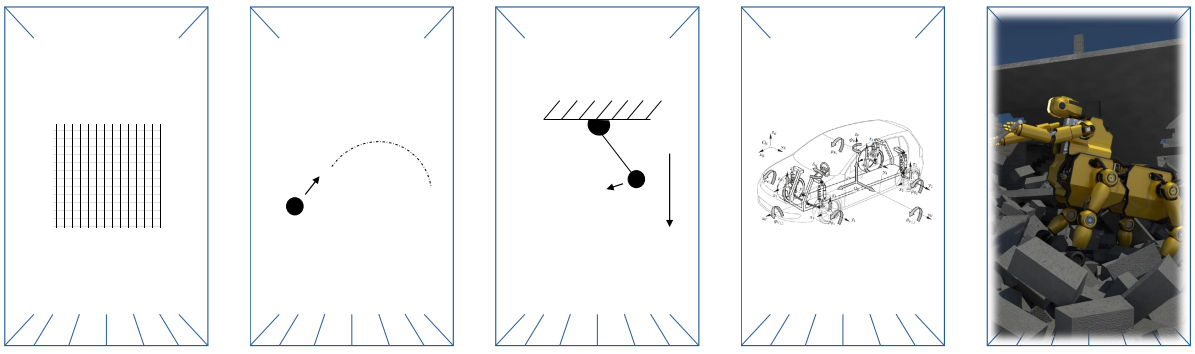


Abb. 2.43: *Entwicklung von Systemen mit Hilfe von VTBs: steigende Komplexität von links nach rechts*

2.5.3.3.2 Physikalisch triviale Komponenten und Systeme

Beginnend mit den *physikalischen Grundexperimenten* kann man direkt die generelle

Funktionsweise erkennen bzw. prognostizieren. Ausgehend von Fragestellungen der klassischen Mechanik in der theoretischen Physik, hier am Beispiel von [200], findet man schnell einfache physikalische Phänomene wie zum Beispiel den *schiefen Wurf* (vgl. Abb. 2.43b) oder ein einfaches *Pendel* (vgl. Abb. 2.43c). Diese können sowohl phänomenologisch durch reale Experimente betrachtet werden als auch in Form mathematischer Modelle. Somit ist es hier möglich, reale Experimente mit analytisch lösbaren Differentialgleichungen (und auch durch gesunden Menschenverstand) zu beschreiben. Der Transfer zu einem VTB erfolgt dann dadurch, dass man nicht einfach die Differentialgleichungen dieses einen Problems zu lösen versucht, sondern einen allgemeineren Lösungsweg wählt, um ähnliche Phänomene ebenfalls lösen zu können. So führt hier eine „simple“ Simulation der allgemeinen Dynamik starrer Körper (Gravitation) dazu, dass man dieses System ohne tieferes Verständnis der Sache betrachten kann. Hierbei dient die Simulation als Abstraktionsschicht, die durch ein geeignetes Integrationsverfahren in der Lage ist, die zugrunde liegenden Gleichungen zu lösen. Selbstverständlich werden hier Annahmen getroffen und somit das Gesamtsystem vereinfacht. Insgesamt ist es jedoch so, dass man bei einem schiefen Wurf nur eine Kugel mit einer vektoriellen Initialgeschwindigkeit modellieren muss und die Simulation damit alle notwendigen Informationen gesammelt hat. Wenn nun die Anforderung wäre, ausschließlich den schiefen Wurf zu betrachten, wäre es sicherlich geraten, dies entweder spezifisch zu modellieren oder sogar analytisch durchzurechnen. Das VTB bietet die Möglichkeit, die implementierte Starrkörperdynamik nun allgemein für beliebige starre Körper zu nutzen. Der Transfer vom schiefen Wurf zum simplem Pendel beinhaltet daher nur das Hinzufügen eines Drehgelenks bzw. die Implementierung der notwendigen Lagrange-Zwangsbedingungen, die mit einem Drehgelenk einhergehen, und nicht die komplette Neuimplementation. Aufbauend auf diesen simplen Beispielen, die später aufgrund ihrer Simplizität und der analytischen Lösbarkeit noch einmal besprochen werden (vgl. Kap. 4.3.2.2), kann man den modularen Aufbau des VTBs weitertreiben und mehrere starre Körper kombinieren, mit weiteren Gelenken ausstatten, interagieren lassen und ggf. um weitere Module ergänzen.

2.5.3.3.3 Weitergehende, komplexere Systeme

Eine (von vielen) Erweiterungen stellt zum Beispiel die *Fahrdynamik* dar (vgl. Abb. 2.43d). Diese ist ein gutes Beispiel dafür, wie man die zuvor erwähnten Gelenke, Starrkörper und Kollisionen aufgrund neuer Anforderungen ergänzt und erweitert. Die Fahrdynamiksimulation erfordert aufgrund von Übersetzungsverhältnis, Rad-Straßen-Kontakt, Differentialgetrieben usw. eine Spezialisierung der VTBs angepasst an die erweiterten Anforderungen. Das heißt, um sinnvolle Erkenntnisse zur Fahrdynamik neuer komplexer Systeme zu erforschen, muss zunächst ein VTB entwickelt werden, in welchem dann *experimentierbare Modelle* entstehen können.

2.5.3.3.4 Robotik – ganzheitliche, komplexe Systeme

Auch wenn eine Fahrdynamiksimulation schon als außerordentlich komplexes System angesehen werden kann, stellt die Robotik ein darüber hinaus gehendes komplexes System dar (vgl. Abb. 2.43e). Hierbei stellen direkte und inverse Kinematik, sowie die Dynamik, Regelung etc. wiederum weitere Anforderungen an ein VTB. Bisher auf die reine Dyna-

miksimulation beschränkt, werden im Bereich der Robotik inhärent die Kinematik, die Sensorik sowie eine Wechselwirkung mit der Umwelt hinzukommen. Ein modularer Aufbau des VTB und somit eine Trennung zwischen Datenbasis, Datenhaltung, Dynamik, Kinematik, Regelung, Sensorik usw. führt dazu, dass auch hier die Erweiterbarkeit im Mittelpunkt steht.

2.5.3.3.5 Modularer Aufbau von VTBs

Alles in allem sind also das Design und die Entwicklung von VTBs selbstverständlich stark anforderungs- bzw. anwendungsabhängig, jedoch können stets vorhandene Module und Komponenten weitergenutzt werden. Anhand der genannten Beispiele wäre es möglich, einen Roboter zu konfigurieren, zu programmieren und zu modellieren, der an einem Auto verschiedene Reparaturaufgaben übernimmt – dies stellt allerdings nur eine Teilmenge der Möglichkeiten dar.

Von dieser anwendungszentrierten Sichtweise soll nun im folgenden Kapitel auf das Fundament eines VTBs und somit seine Struktur eingegangen werden.

2.5.3.4 Design und Entwicklung von VTBs

Zur Entwicklung der zuvor beschriebenen Systeme mit Hilfe von VTBs ist die zentrale Voraussetzung die Existenz eines solchen VTBs. Das Konzept des VTBs beinhaltet eine stetige, modulare Erweiterbarkeit. Dies bedeutet, dass die Nutzung des VTBs weitaus häufiger vorkommt als die Entwicklung eines solchen. Im Falle einer neuen Anwendung, die ein VTB nutzen soll, muss eine solche Entwicklung nur einmal initiiert werden und kann dabei sehr gut auf bisherigen Entwicklungen aufbauen. Somit beinhaltet das Design und die Entwicklung von VTBs vielmehr die Anpassung und Erweiterung vorhandener Testbeds, um eine optimale Forschungssituation für neue Anwendungen zu ermöglichen.

Als Plattform für die Realisierung von VTBs wird das Simulationssystem VEROSIM (siehe Abschnitt 2.5.1.8) gewählt. In Abb. 2.44 sind der Aufbau und die zentralen Bestandteile bzw. Anwendungsgebiete eines VTBs dargestellt.

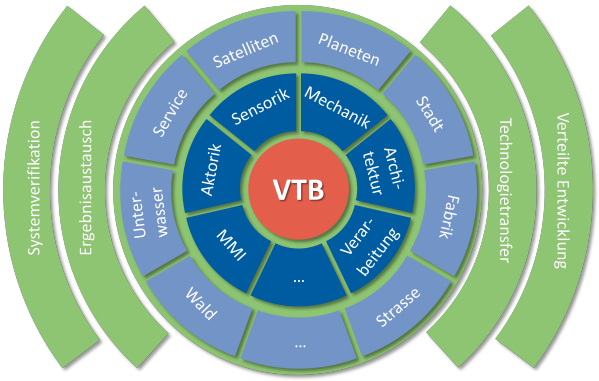


Abb. 2.44: *Das VTB: Frameworks, Anwendungen und Nutzen [201]*

Für das Design und den Entwicklungsprozess von VTBs sind folgende Schritte entscheidend:

1. Definition und Modellbildung für neue Anwendungen oder bisher nicht abgebildete (physikalische) Prozesse,
2. Implementierung der Modelle in Form von Erweiterungen für das VTB,
3. Einbindung/Kopplung der entwickelten Komponenten mit bestehenden,
4. 3D-Modellierung neuer Anwendungsszenarien,
5. Evaluation und Test der entwickelten Module sowie deren Zusammenspiel mit bestehenden Komponenten.

Diese sollen im Folgenden genauer beschrieben werden.

2.5.3.4.1 Definition und Modellbildung

Für neue Anwendungsgebiete oder neue physikalische Phänomene müssen bestehende Simulationsframeworks erweitert oder neue hinzugefügt werden. Hierzu muss zunächst ein mathematisches, physikalisches, systemtheoretisches Modell entwickelt werden, das man abbilden möchte. Gerade im Bereich der Simulationstechnologien ist hierbei insbesondere der Abstraktionsgrad von zentraler Bedeutung. Je nach Anwendung können in diesem Schritt Approximationen getroffen werden, die den Rechenaufwand reduzieren, jedoch in einem vertretbaren Maße *physikalische Plausibilität* über physikalische Korrektheit stellen. Als „Engineering Tool“ muss das VTB hier immer das Ziel haben, sich einem physikalisch korrekten Verhalten bestmöglich anzunähern.

2.5.3.4.2 Implementierung

Die Implementierung der mathematischen Modelle in Form von Erweiterungen für das VTB kann anschließend durch die Integration vorhandener (externer) Softwaremodule oder Eigenimplementationen erfolgen.

2.5.3.4.3 Einbindung/Kopplung

Die Flexibilität der VSD ermöglicht schon inhärent eine modulare Struktur, um neue Frameworks einzubinden und zu koppeln.

Generell ist beim VTB insbesondere die vorherrschende *Baumstruktur* (Eltern-/Kind-Knoten (Parent/Child Nodes)) ein Mittel, um eigene Entwicklungen einzubinden.

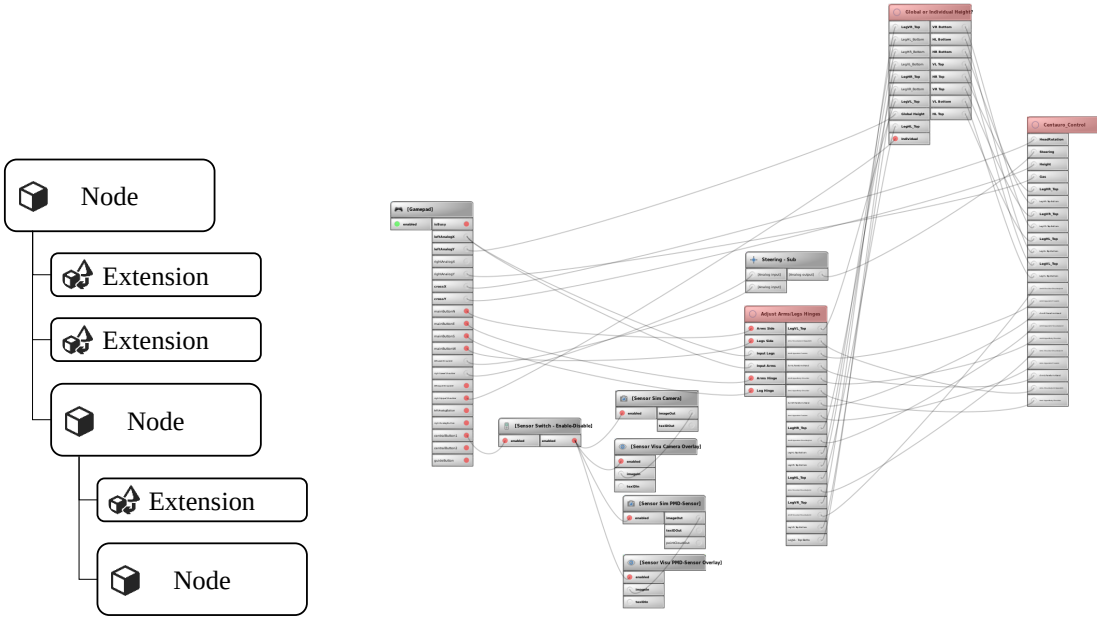
Sogenannte „Extensions“ erweitern dabei die Funktion ihrer Elternknoten (siehe auch Abb. 2.45a). Nodes können dann durch *Extensions* in ihrer Funktionalität erweitert werden – so wird beispielsweise jeder Knoten mit einer `ExtensionRigidBody` zu einem Teil der Starrkörpersimulation.

Des Weiteren stellt das *I/O Board* [202] (siehe Abb. 2.45b) eine Möglichkeit der Kopplung eigener Entwicklungen mit schon bestehenden zur Verfügung. So kann man, basierend auf VSD-Datentypen, beliebige Ein- und Ausgabeparameter übergeben.

Gerade das I/O Board stellt eine komfortable Umgebung zur Verfügung, um verschiedene interne Prozesse oder Module standardisiert miteinander zu verbinden. Es wird im Laufe dieser Arbeit immer wieder verwendet.

2.5.3.4.4 3D-Modellierung

Des Weiteren müssen 3D-Simulationsmodelle erstellt werden, welche die neuen Features nutzen. Je nach Anwendung beinhaltet die Modellierung im VTB unterschiedliche



(a) Knoten (Nodes) und Erweiterungen (Extensions)

(b) VTB Kopplung: interner Datenaustausch mit Hilfe des „I/O Boards“

Abb. 2.45: *Knoten (Nodes), Erweiterungen (Extensions) und deren Verbindungen (I/O Board)*

Aspekte, je nachdem, was implementiert wurde. Im Bereich der dynamischen Objektmodellierung umfasst dieses dann beispielsweise den Geometrieimport, die Dynamisierung, die Umweltgenerierung, das Ausstatten mit Sensorik sowie das modulare Hinzufügen weiterer gewünschter Funktionalitäten.

2.5.3.4.5 Evaluation und Test

Mit Hilfe der erstellten Modelle kann man nun die implementierten (physikalischen) Modelle testen. Des Weiteren bietet das VTB die Möglichkeit, auch reale Referenzszenarien hinzuzuziehen, um das Gesamtsystem zu validieren. Selbstverständlich muss in diesem Schritt auch darauf geachtet werden, dass bisherige Funktionalitäten nicht beeinträchtigt oder gar zerstört werden.

2.5.3.4.6 Anwendung und Erweiterung

Nach erfolgreicher Evaluation kann man die neuen Funktionalitäten in Anwendung bringen und anhand von Kap. 2.5.3.3 komplexe Systeme mit Hilfe von VTBs entwickeln. Die entwickelten Frameworks stehen dann nicht nur für diese eine Anwendung bereit, sondern können direkt von jedem Entwickler und Nutzer anwendungsübergreifend genutzt werden.

2.5.3.5 Transfer von VTBs zu realen Systemen

Der Transfer von VTBs zu realer Hardware kann auf verschiedene Art und Weise geschehen.

2.5.3.5.1 Wissentransfer durch VTBs

Zunächst können gewonnene Erkenntnisse aus dem VTB genutzt werden, um das reale System zu optimieren. Je nach Anwendung schließt dies die Hardware und/oder Software mit ein.

2.5.3.5.2 Simulationsbasierte Regelung/Steuerung

In einem nächsten Schritt kann das VTB auch genutzt werden, um direkt reale Hardware anzusteuern. Dieser Simulation-based Control (SBC) Ansatz beinhaltet entweder (a) die Möglichkeit, die Simulationssoftwaremodule selbst auf der Hardware laufen zu lassen oder (b) die Integration von Hardwarespezifischer Steuerungssoftware in das VTB. Für (b) sind die im Folgenden präsentierten Schnittstellen nötig.

2.5.3.5.3 Schnittstellen

Mit Hilfe von standardisierten Schnittstellen wie zum Beispiel des „Functional Mockup Interfaces“ [by] ist es möglich, Modelle vereinheitlicht auszutauschen. Des Weiteren zeigen solche Standardisierungen über Systemgrenzen hinweg auch die Möglichkeit der Co-Simulation oder der Simulationskopplung.

2.5.3.5.4 Co-Simulation/Simulationskopplung

Unter Co-Simulation versteht man die gleichzeitige Ausführung verschiedener Simulationsprogramme. Dies wird oft genutzt, um spezielle Aspekte (wie zum Beispiel die Strukturdynamik eines Bauteils) in speziellen (dafür optimierten) Softwaresystemen zu analysieren um am Ende die Ergebnisse wieder ineinander zu übertragen. Bei der Simulationskopplung werden die vorher beschriebenen Systeme nicht unabhängig voneinander instantiiert, sondern beeinflussen einander. Passiert beispielsweise eine Kollision in der Starrkörpersimulation, soll daraufhin die dadurch verursachte strukturdynamische Bauteilbeeinflussung in einem FEM-Tool überprüft und die dort herausgefundene Verformung des Bauteils rückgekoppelt werden in die Starrkörpersimulation, damit diese Verformung bei einer weiteren Kollision mit in die Kollisionsbehandlung einfließt.

2.6 Idee und Vision von „Seeing Spaces“

Ein Beispiel für einen sehr interessanten Ansatz einer „intuitiven“ Nutzerschnittstelle – als Weiterentwicklung der UI Bits von [21] – hat Bret Victor im Jahr 2014 auf der EG-Konferenz [203] als seine Vision der nächsten Generation von Werkstätten oder „Maker-Spaces“ präsentiert, einen sogenannten „Seeing Space“. Zentral dabei ist eine kollaborative Entwicklungsumgebung, in der intuitiv alle nötigen Informationen zugänglich gemacht werden, wie in Abb. 2.46 gezeigt.

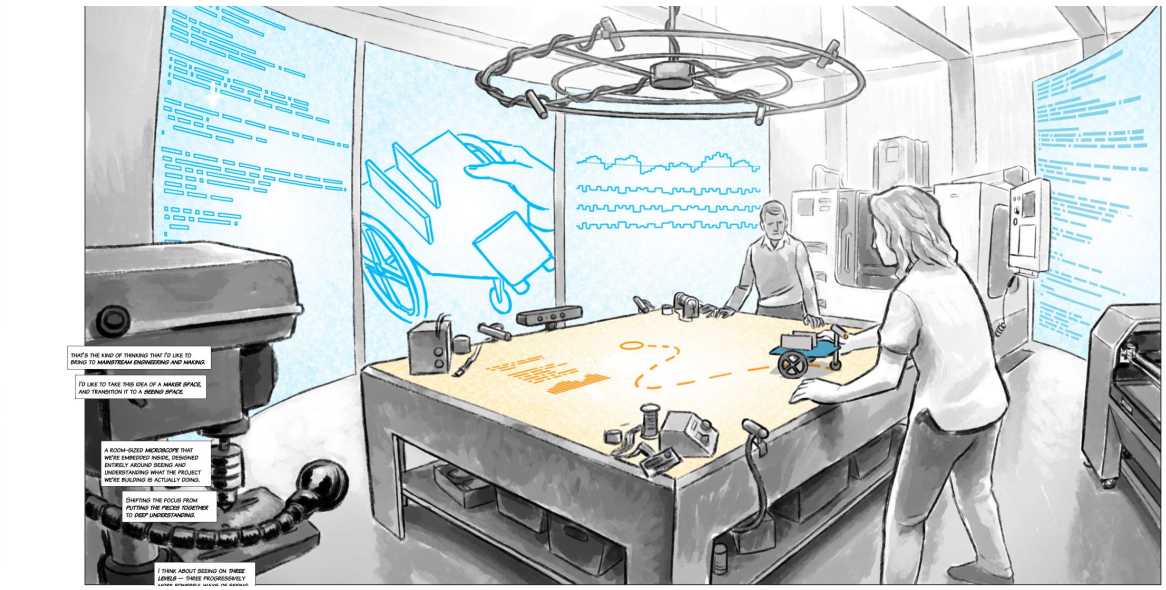


Abb. 2.46: *„Seeing Spaces“ von Bret Victor [bz]*

Neben der Möglichkeit, zusammenzuarbeiten, beinhaltet ein solcher „Seeing Space“ verschiedene Kamera/Tracking-Modalitäten, unterschiedliche Visualisierungen (sowohl auf dem Tisch als auch an der Wand) sowie die von Bret Victor beschriebenen zentralen Bestandteile „seeing inside, seeing across time, seeing across possibilities“. Kurz gesagt, beschreibt er hier die Interaktion des Menschen mit der Maschine mit seinen Händen sowie die Visualisierung aller „wichtigen“ Informationen, die man braucht, um das System (in diesem Fall einen mobilen Roboter) optimal zu fertigen. Leider beschreibt Bret Victor in seinem Vortrag nicht, wie man eine solche Tüftlerwerkstatt erschafft und wann man diese (wie genau) nutzt.

Mit dieser Vision eines „Seeing Space“ sind die Ansätze, einen Mediator zwischen Mensch und robotischer Maschine zu etablieren, sehr gut zu fassen als:

1. intuitive Eingabe (Nutzerschnittstelle) und
2. aufbereitete Ausgabe (meist visuell), die dem Menschen alle nötigen Informationen präsentiert sowie
3. Verbindung und Kopplung von realen Objekten mit direkt sichtbaren (virtuell erschaffenen) Visualisierungen.

Da sich diese mit den Forschungsschwerpunkten dieser Arbeit decken, sollen „Seeing Spaces“ als Motivation, Ausgangspunkt und Vision angesehen werden, welche mit der in

dieser Arbeit entwickelten Struktur umgesetzt und erweitert werden soll. Insbesondere soll der DT im Rahmen dieser Arbeit eben nicht nur in der Entwicklung genutzt werden, sondern der „Seeing Space“ soll auch online, parallel zum physischen Asset, Anwendung finden.

2.7 Diskussion des Forschungsstandes

In diesem Abschnitt 2 wurden Mediatoren verschiedener Anwendungsgebiete, verschiedener Prozesse und in unterschiedlichen Abschnitten des Lebenszyklus von komplexen Systemen präsentiert. Am präsentesten sind Assistenzsysteme aus dem Automotive-Bereich, die durch Bedien- und Anzeigeelemente Hilfestellung geben. Diese geben dem Nutzer alle nötigen Informationen an die Hand oder greifen sogar in geplante Aktionen ein. In der Robotik sind Assistenzsysteme meist über die Regelung implementiert, um eine direkte Interaktion mit dem Menschen zu ermöglichen. Im Bereich der Entwicklung solcher komplexen Systeme haben sich Simulationssysteme etabliert, welche den Entwickler dabei unterstützen, die Komplexität zu abstrahieren und das Systems zu verstehen. Die hier geschaffenen „Hilfsmittel“ sind aber ausschließlich in diesem frühen Entwicklungsprozess nutzbar und verlieren ihren Nutzen, sobald das System produziert bzw. in Betrieb genommen wird. Hier geht daher sehr viel Know-How verloren.

Die Idee ist daher, die 3D-Simulationstechnologie über die Entwicklungsphase hinaus auch im Betrieb bzw. Nutzungsprozess zu etablieren. Gerade im Bereich cyber-physischer Systeme steht mit dem Digitalen Zwilling des betrachteten physischen Assets eine direkte Kopplung von Virtualität und Realität bereit, die zentraler Dreh- und Angelpunkt dieser Idee sein soll. Wie in den vorigen Kapiteln beschrieben, wird diese Assistenz sowohl die Bedienung als auch die Anzeige umfassen, um einen intuitiven Zugang zu dem komplexen System zu schaffen, sodass der Digitale Zwilling als Mediator für den Entwickler und auch für den Nutzer fungiert.

Eine erste Vision wurde im vorigen Kapitel mit den „Seeing Spaces“ präsentiert, welche auch die Kopplung von Realität und Virtualität nutzen, um kooperativ komplexe robotische Systeme zu entwickeln. Mit den gleichen Tools könnte man auch die finale Nutzung (Steuerung, Wartung, Umrüstung etc.) von Robotern unterstützen. Insbesondere die MRK steht hierbei als potentieller Anwendungsfall im Vordergrund, da hier die Entwicklung aller (heterogenen) Komponenten und deren Zusammenspiel sowie eine intuitive und effiziente Nutzerschnittstelle ausschlaggebend ist. Zur Umsetzung dieser kann dann wiederum auf bisherige Entwicklungen der Assistenzsysteme (Kap. 2.2), der MMI (Kap. 2.3) und der MRK (Kap. 2.4) zurückgegriffen und domänenübergreifend nutzbar gemacht werden. Dies kann wiederum mit den aktuellen Trends der Industrie 4.0, cyber-physischer Systemen und den spezifischen Forschungs- und Arbeitsschwerpunkten dieser Arbeit (vgl. Kap. 2.1.1 und insb. die Aufzählung auf S. 8) überein gebracht werden.

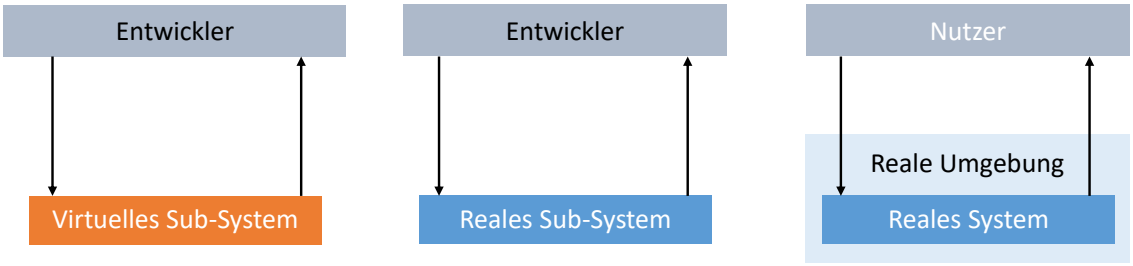
KAPITEL 3

Konzept

Im Rahmen dieser Arbeit soll die Entwicklung von Mensch-Maschine-Schnittstellen für die Entwicklung und Nutzung komplexer Systeme betrachtet werden.

Wo zunächst ein Entwickler ein komplexes System oder Systemkomponenten entwickelt (meist mit 3D-Simulatoren im Virtuellen), soll später ein Endnutzer dieses komplexe System benutzen. Hierbei treten folgende Fragestellungen auf, die auch in Abb. 3.1 bildlich dargestellt sind:

- Wie interagiert der Entwickler mit dem komplexen System im Virtuellen?
- Wie interagiert der Nutzer mit dem komplexen System in der Realität?
- Wie transferiert man Erfahrungen, Wissen etc. aus dem Entwicklungs- in den Nutzungsprozess?

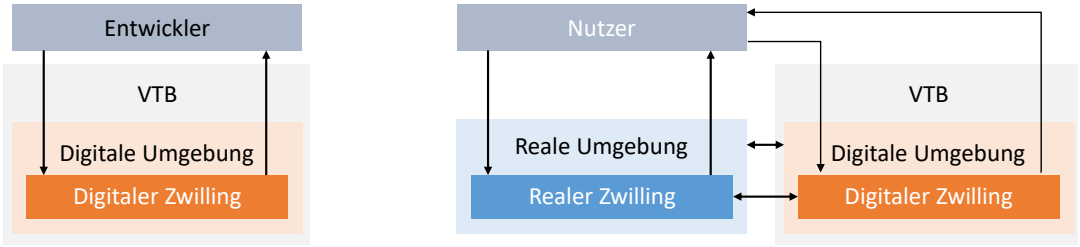


(a) Digitale Entwicklung *(b) Reale Entwicklung* *(c) Reales Endszenario*

Abb. 3.1: *Aus dem Entwicklungsprozess in den Betrieb*

Die Nutzung von Simulationsmöglichkeiten im Entwicklungsprozess von Robotersystemen ist bereits als ein Standardverfahren zur zeit- und kostengünstigen Vorhersage komplexen Systemverhaltens bekannt. Als elementarer Bestandteil in der Entwicklungsphase findet die Simulationstechnologie im finalen System jedoch keine Anwendung. Außerdem birgt ein Simulationssystem oft selbst eine Komplexitätsebene, die erst überschritten werden muss, um sinnvoll damit arbeiten zu können.

Als konzeptionelle Weiterentwicklung der VTBs wird in dieser Arbeit der Digitale Zwilling als Bindeglied zwischen Mensch und Maschine installiert. Dieser lebt und wächst über den gesamten Entwicklungszyklus hinweg, um auch in dem finalen System Anwendung



(a) DT-basierte Entwicklung *(b) DT-basierte Nutzung*

Abb. 3.2: *Mit dem DT aus dem Entwicklungsprozess in die Anwendung (vgl. Abb. 3.3)*

zu finden. Als lebenszyklusübergreifender Mediator schafft der DT einen reibungslosen Übergang zwischen Entwickler und Nutzer.

In der vorliegenden Arbeit wird eine Struktur entwickelt die den DT als zentralen Mediator zwischen Mensch und Maschine platziert. Diese Struktur stellt eine konzeptionelle Weiterentwicklung eines VTBs dar. Neben der notwendigen Integration diverser Module in das VTB steht auch die (bidirektionale) Online-Interaktion sowohl mit dem Digitalen Zwilling als auch dem realen System im Fokus. Somit soll, basierend auf der 3D-Simulation, eine Symbiose aus Mensch, dem Digitalen Zwilling (im VTB) und dem realen System entstehen.

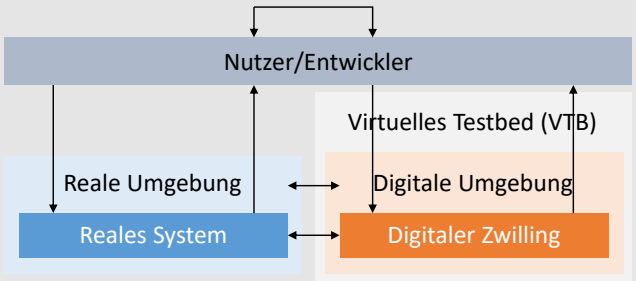


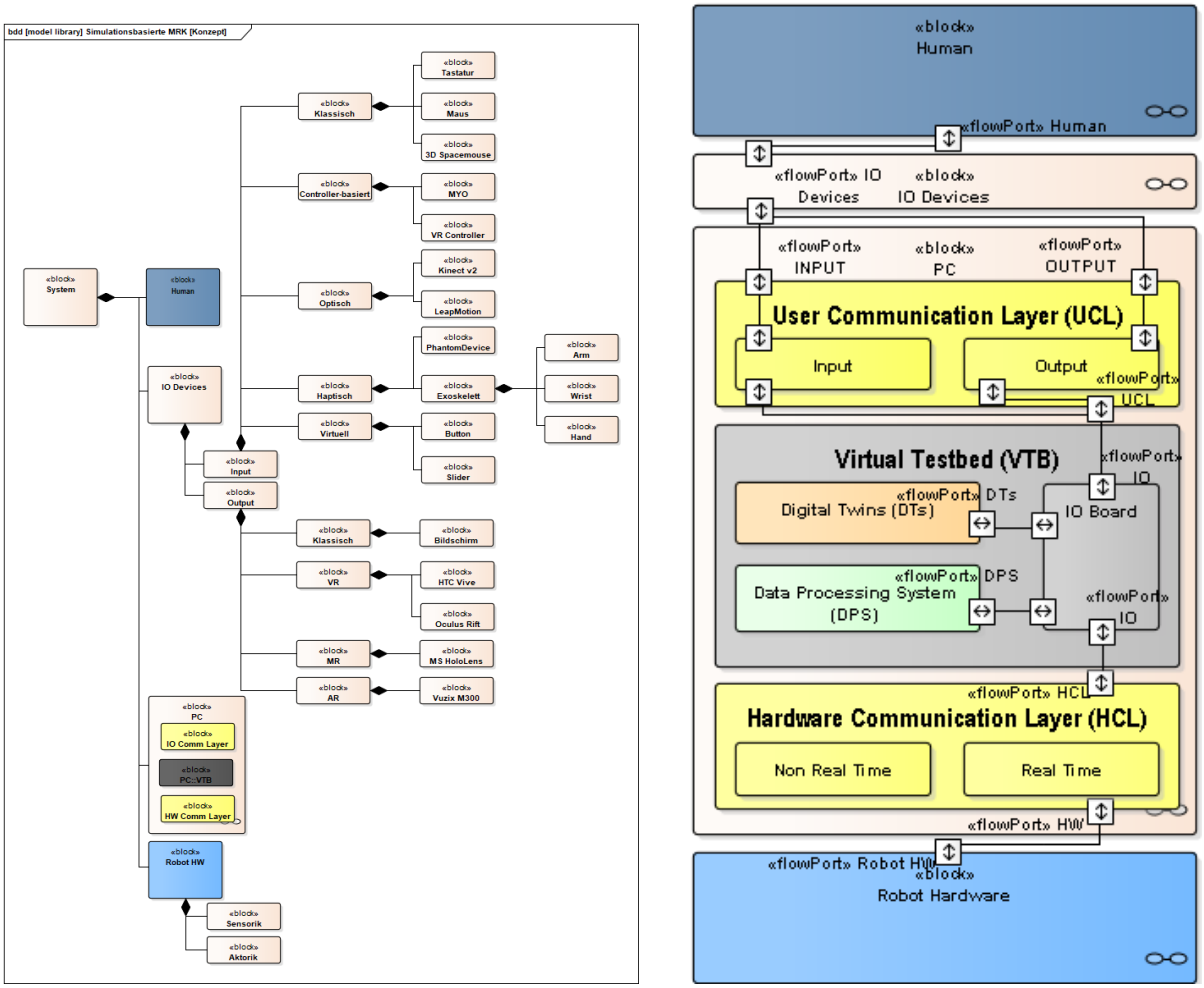
Abb. 3.3: *Konzeptionelle Weiterentwicklung der VTBs mit Hilfe Digitaler Zwillinge*[*]

Der Nutzer bzw. Entwickler stellt dabei einen wesentlichen Bestandteil in diesem System dar, wie in Abb. 3.3 zu sehen ist.

[*] Grafische Codierung in folgenden Grafiken: Das VTB wird in der Farbe Grau, der DT (bzw. virtuelle/ simulierte Komponenten) in Orange, das reale System (bzw. reale Systemkomponenten) in Hellblau und der Nutzer/Entwickler in Dunkelblau dargestellt

3.1 Das Gesamtsystem aus Nutzer, Digitalem Zwilling und physischem Asset

Mit dieser Arbeit soll sowohl dem Entwickler als auch dem Endnutzer komplexer Systeme geholfen werden, indem ein intuitiver Mediator – der Digitale Zwilling im Virtuellen Testbed – eingesetzt wird, wie in Abb. 3.4 (und Abb. 3.5) zu erkennen.



(a) SysML-Konzeptgraphik als Blockdefinitionsdiagramm (BDD)

(b) SysML-Konzeptgraphik als Internes Blockdiagramm (IBD)

Abb. 3.4: *SysML-Graphik des Konzepts: Gesamtbild aus Nutzer, I/O-Devices, inkl. User Communication Layer (UCL), VTB und Hardware Communication Layer (HCL) als Mediator und das reale System (alle eingezeichneten Flüsse sind Datenflüsse; weitere Graphiken in Anhang 7)*

Im Rahmen dieses Konzeptes kann man drei zentrale Schichten/Layer ausmachen:

1. den Menschen: Nutzer, Entwickler oder Operator,
2. das VTB als Mediator,
3. das reale komplexe System, die Roboterhardware.

Das reale robotische System wiederum besteht aus Aktoren und Sensoren, die mit ihrer Umwelt wechselwirken und Daten austauschen. Der Nutzer hat bestimmte Möglichkeiten, mit einem System zu interagieren, wobei hier diese Herausforderung durch die direkte Interaktion mit dem VTB ersetzt wird.

3.2 Notwendige Bestandteile des Virtuellen Testbeds als Mediator

Der Mediator muss also sowohl Eingabe und Ausgabe auf Seiten des Nutzers, als auch auf Seiten der Hardware abdecken. Somit ergeben sich im VTB wiederum drei essentielle (Sub-)Schichten bzw. Layer (siehe Abb. 3.5):

a. User Communication Layer (UCL) (Schnittstelle/Interface Nutzer-VTB),
b. der DT und der 3D-Simulationskern,
c. Hardware Communication Layer (HCL) (Schnittstelle/Interface VTB-Hardware).

Eine solche Struktur soll ermöglichen, mit Hilfe von VTBs und der 3D-Simulation intuitiv mit komplexen Systemen zu interagieren. Im Rahmen dieses Konzeptes wird der Digitale Zwilling zentrales Entwicklungs- und Operationswerkzeug. Des Weiteren wird das VTB zum universellen Eingabegerät und zur universellen Schnittstelle für die Interaktion mit komplexen Systemen. Schlussendlich kann das VTB als weitere Abstraktionsschicht angesehen werden, welche die Komplexität des Systems verbirgt und einen intuitiven sowie sicheren Zugang, ermöglicht. Dies gilt insbesondere für die Entwicklung und direkte Interaktion im Bereich der Mensch-Roboter-Kooperation.

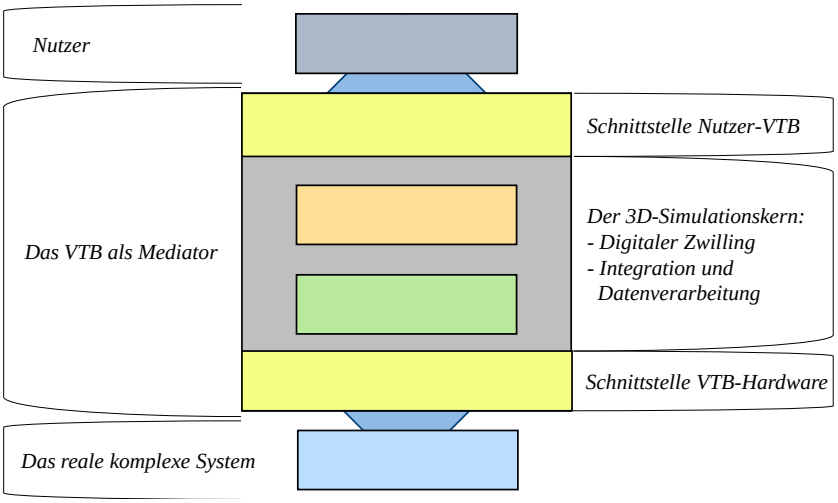


Abb. 3.5: *Simplifizierte Darstellung der zentralen Bestandteile des Konzeptes (vgl. Abb. 3.4)*

3.2.1 Schnittstelle zwischen Nutzer und Digitalem Zwilling

Um mit einem System interagieren zu können benötigt dieses eine Nutzerschnittstelle. Konzeptionell kann diese entweder zwischen Nutzer und einzelnen DTs sein oder generisch zwischen Nutzer und allen DTs in einem VTB. Durch eine Verbindung nicht nur zum DT, sondern vielmehr sowohl zu diesem als auch zum VTB, besteht die Möglichkeit auch das Gesamtszenario zu beeinflussen/zu verändern. Außerdem kann durch eine generische Lösung hier eine vielseitigere und konfigurierbare Lösung geschaffen werden.

Individuelle, nutzerspezifische Anforderungen an diese Schnittstelle können dadurch weiterhin angepasst werden, um einen möglichst intuitiven Zugang zum Digitalen Zwilling zu ermöglichen.

3.2.2 Der Digitale Zwilling im Virtuellen Testbed

Den DT als reines Modell eines realen physischen Assets zu betrachten wird der Komplexität des DTs nicht gerecht. Interessant wird der DT erst in seiner Einsatzumgebung und durch die Einbettung des DT in die virtuelle Experimentierumgebung – das VTB. Schlussendlich wird der DT hierin erlebbar und insbesondere in seinen physikalisch relevanten Parametern evaluierbar. Daten, die der Digitale Zwilling nutzt und produziert, werden in diesem Verbund erzeugt, wodurch auch die gesamte Prozesskette analysiert werden kann. Neben dem DT selbst, dessen Integration und Datenverarbeitung, sind Erweiterungen der umgebenden 3D-Simulation notwendig, um spezielle Anforderungen des DTs adäquat abbilden zu können.

3.2.3 Schnittstellen zwischen Digitalem Zwilling und physischem Asset

Um den DT mit Leben zu füllen, benötigt dieser eine direkte Verbindung zu seinem Realen Zwilling. Erst hierdurch wird ein Datenaustausch zwischen diesen in beide Richtungen möglich. Diese virtuellen und realen Datenströme sind nötig, um im VTB zu experimentieren und zu evaluieren. Damit die Hardwareschnittstelle nicht für jeden DT neu implementiert werden muss, ist auch hier ein möglichst universeller Ansatz nötig, der ein breites Spektrum der Robotik abdecken kann. Der Ansatz wird daher als Schnittstelle zwischen VTB und Realem Zwilling konzipiert.

Im Folgenden wird zunächst in Kap. 3.3 eine Anforderungsanalyse an ein solches Framework beschrieben, bevor die vier Säulen des Konzeptes in Kap. 3.4 aufgezeigt werden.

3.3 Anforderungsanalyse

Bei der Entwicklung einer Struktur zur direkten Interaktion mit komplexen Systemen -– vermittelt und unterstützt durch den DT im VTB als Mediator -– ergibt sich eine Vielzahl von Anforderungen. Diese werden im Folgenden kurz vorgestellt und erläutert, um in Kap. 5.4 wieder aufgegriffen zu werden.

Anforderung *1: Entwickler- und Nutzer-zentriert*

Zentrales Element ist der Mensch, der ein komplettes, intuitiv bedienbares Gesamtsystem benötigt.

Im Sinne der verschiedenen Entwicklungsstadien komplexer Systeme soll das Konzept über den gesamten Entwicklungs- und Wertschöpfungsprozess einsetzbar sein. Dies beinhaltet trotz der inhärenten Komplexität des Systems eine intuitive Interaktionsmöglichkeit.

Anforderung *2: Kollaborationsmöglichkeiten*

Sowohl in der Entwicklung als auch in der finalen Anwendung soll zusammengearbeitet werden.

Bereits in der Entwicklungsphase soll das Konzept die Möglichkeit bieten, gemeinsam ein ganzheitliches System aufzubauen. Entwicklungen und Implementierungen sollen auch in der finalen Anwendung zum Einsatz kommen und ein breites Spektrum von Nutzern ansprechen. Das Konzept soll die Kollaboration auch im Sinne der Mensch-Maschine-Kommunikation (MMK) ermöglichen.

Anforderung *3: Immersion*

Durch die eigene Präsenz soll der Nutzer immersiv in (semi-)virtuelle Welten eintauchen können.

Zur intuitiven Interaktion soll hiermit ein immersives Erlebnis geschaffen werden, um eine weitere intuitiv zugängliche Abstraktionsschicht für den Nutzer zu erschaffen.

Anforderung *4: Intuitiv und Sicher*

Unabhängig von der Anwendung soll ein inhärent sicherer Zugang für den Nutzer ermöglicht werden, der intuitiv und assistierend die menschlichen Fähigkeiten ergänzt und unterstützt.

Zur Unterstützung des Nutzers soll dieser intuitiv und sicher mit den komplexen Systemen interagieren können. Des Weiteren soll durch dieses Konzept eine inhärente Sicherheit des Systems aus Mensch und Maschine erschaffen werden, um Schäden an beiden zu vermeiden.

Anforderung *5: Integration aktueller Technologien*

Aktuelle Technologien sollen im Bereich von Ein- und Ausgabe verwendet werden, einschließlich der Möglichkeit, auch auf zukünftige Entwicklungen schnell reagieren zu können.

Zur Massentauglichkeit und für den vielfachen Einsatz sollen aktuelle (meist kommerziell erhältliche) Technologien eingesetzt sowie standardisierte Schnittstellentechnologien verwendet werden.

Anforderung *6: Modular*

Die Gesamtstruktur soll modular flexibel sein.

Eine modulare Grundstruktur sowie eine leichte Erweiterbarkeit sollen dem Nutzer die Möglichkeit geben, einfach neue Funktionen hinzufügen zu können.

Anforderung *7: Universell einsetzbar, skalierbar*

Die Gesamtstruktur soll unabhängig von Anwendung, Größe oder Einsatzzweck sein.

Universell einsetzbar kann das Konzept jedwede Art von System bedienen und durch eine skalierbare Struktur auch Anwendungen in vielen Bereichen ermöglichen.

Anforderung *8: Inspektionsmöglichkeit und Validierbarkeit*

Zur Überprüfung der Ergebnisse soll die Struktur einfach validierbar und mit anderen Systemen vergleichbar sein.

Nicht nur die Postprozessvalidierung steht im Fokus, sondern auch die Inline-Validierung des Systems als solchem. Eine derartige Validierung kann prä-, post-, aber auch inprozedual stattfinden.

Anforderung *9: Technologietransfer und Kostenreduktion*

Ausnutzen des Konzeptes zum Transfer sowie zur Kostenersparnis.

Die Struktur soll einen Technologietransfer ermöglichen, um so eine Basis für zukünftige Erweiterungen oder neue Systeme zu schaffen. Weiterhin soll die Struktur Möglichkeiten zur Kostenreduktion auf verschiedenen Ebenen (Entwicklungskosten, allgemein laufende Kosten, Wartungskosten, Personalkosten etc.) bieten.

3.4 Konzeptionelle Funktionalitäten des Digitalen Zwillings

Zu den Funktionalitäten des DTs gehören folgende Aspekte, die in den nächsten Kapiteln genauer beleuchtet werden:

1. der Digitale Zwilling als Entwicklungswerkzeug
2. der Digitale Zwilling als Eingabegerät
3. der Digitale Zwilling als Schnittstelle
4. der Digitale Zwilling als Abstraktionsschicht für den Nutzer

Schlussendlich wird das Ziel verfolgt, einen intuitiveren und natürlicheren Zugang zu komplexen Systemen zu ermöglichen, unter Ausnutzung der Fähigkeiten von Mensch, Maschine und Digitalem Zwilling.

3.4.1 Der Digitale Zwilling als Entwicklungswerkzeug

Modelle und Simulationen werden im Entwicklungsprozess schon jetzt rege genutzt. Im Rahmen des vorliegenden Konzeptes sollen diese Nutzung und der Simulationseinsatz erweitert werden. Auf Basis des VTBs und eines DTs ist es möglich, ein reales System virtuell abzubilden, sowohl als Ganzes als auch als Zusammenschluss verschiedener Submodule. Diese Submodule wiederum können zunächst Dummy-Funktionen beinhalten, modular hinzugefügt und mit Leben gefüllt werden. Des Weiteren soll es möglich sein, einzelne Subsysteme modular austauschen zu können. So wird das VTB als Experimentierzone für Ingenieure die Möglichkeit bieten, neue Sensoren, Aktoren oder Datenverarbeitungsalgorithmen quasi spielerisch ausprobieren zu können.

Zudem sollen diese Module sowohl virtuell als auch real vorliegen und natürlich auch miteinander gekoppelt werden können. Somit kann der Digitale Zwilling des Systems ein Konglomerat aus realen und virtuellen Sub-Modulen beinhalten und beliebig aus diesen konfigurierbar sein. Dieses Vorgehen wird im Rahmen dieser Arbeit als Virtuelle Modulare Integration (VMI) (siehe Abb. 3.6) klassifiziert, da die Simulation als Integrationswerkzeug genutzt wird und flexibel das Systemmodell an virtuelle oder reale Submodule anpasst. Diese VMI wurde auch bereits im Bereich des Simulation-based Support (SBS) in [9] angewandt.



(a) Virtuell *(b) Real*

Abb. 3.6: *Virtuelle Modulare Integration von Systemkomponenten: Dieselbe Datenbasis kann schon im Entwicklungsprozess genutzt werden. Der Transfer auf das reale System ist dann simpel, durch Umlegen des Schalters.*

Das VTB bietet hierbei die zentrale Datenbasis (DPS), um diese Integration zu ermöglichen. Des Weiteren ist dieses Konzept auch in den generellen Aufbau des VTBs mit DPS, Rendering Framework, echtzeitfähigem Starrkörperframework sowie dem Sensorframework sehr gut einzubetten. Im Zusammenspiel mit den intern verfügbaren Frameworks sowie extern verfügbaren Bibliotheken kann der Entwicklungsprozess flexibel und zielführend gestaltet werden. Der finale Transfer zur realen Hardware kann natürlich durch eine Kopie des Algorithmus erfolgen, soll aber vielmehr durch geeignete Schnittstellen in und mit dem VTB verschaltet werden. Den Entwickler interessiert es somit im Entwicklungsprozess nicht, ob er mit dem realen oder virtuellen System arbeitet. Insbesondere bei klassischen Entwicklungsprozessen, in denen einzelne Projektpartner für einzelne Submodule zuständig sind, kann das Gesamtsystemmodell im VTB dazu dienen, das zu entwickelnde Modul direkt in den Kontext der finalen Anwendung einzuordnen und zu testen.

3.4.2 Der Digitale Zwilling als Eingabegerät

Dadurch, dass die Simulation quasi automatisch im Entwicklungsprozess eingesetzt wird, entsteht für den Entwickler eine neue Interaktionsschnittstelle mit dem realen System. Durch das VTB werden die Befehle an das robotische System weitergeleitet und sensorisches Feedback kann von der Simulation aufbereitet werden, bevor der Entwickler oder Nutzer dieses sieht. Das VTB wird somit zu einer weiteren Abstraktionsebene zur Interaktion mit realen Robotern sowohl in der Entwicklung als auch in der finalen Operation.

Diese Entwicklung kann nun genutzt werden, um eine nutzerfreundliche Kooperationsmöglichkeit zwischen Mensch, DT und Asset zu erschaffen. Je nach Anwendung, Prozess oder auch Nutzer stellt das VTB Ein- und Ausgabemodalitäten bereit, die einen intuitiven Zugang zu dem robotischen System liefern. Diese Schnittstelle zwischen Mensch und VTB kann auch schon im Entwicklungsprozess des Systems mitentwickelt und optimiert werden und in der finalen Mission Anwendung finden. Das beinhaltet klassische Ein- und Ausgabegeräte genauso wie aktuelle Technologien im Bereich der MMI. Insbesondere sind hier der Bereich von VR-Visualisierungshardware, Kraftrückkopplungssystemen sowie die kontaktlose optische Gestenerfassung zur intuitiven Wechselwirkung mit dem VTB gemeint. Zusätzlich wird durch diese konzeptionelle Idee ermöglicht, alle im VTB vorhandenen Funktionen und Frameworks weiterhin zu nutzen, wenn es für die Anwendung förderlich ist. So sollen dem Nutzer alle (internen) Parameter und Informationen stets intuitiv und sinnvoll zugänglich gemacht werden, um das volle Potential aus diesem symbiotischen System aus Mensch, VTB und Maschine zu erzielen. Hiermit fungiert das VTB sowohl als Abstraktionsschicht, da es die Komplexität des Systems verbirgt, als auch als universelles Eingabegerät, das durch neue Technologien sinnvoll ergänzt werden kann.

Wie in Abb. 3.7 erkennbar, wird das VTB genutzt, um simultan das reale System und den DT zu betreiben und bei Belieben zwischen diesen umzuschalten. Durch das zentrale DPS und die Möglichkeit, die gleichen Eingabegeräte für virtuelle und reale Systeme zu nutzen, kann ein solcher „Switch“ auch in der finalen Mission genutzt werden, um kritische Situationen zunächst in der virtuellen Welt zu evaluieren und Aktionen zu testen, bevor man diese mit dem realen System ausführt. Der DT als Eingabegerät führt

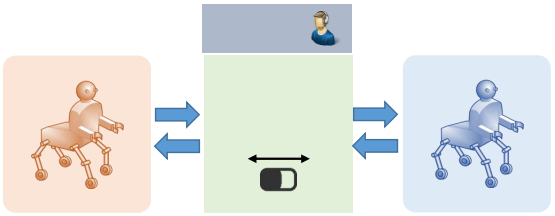
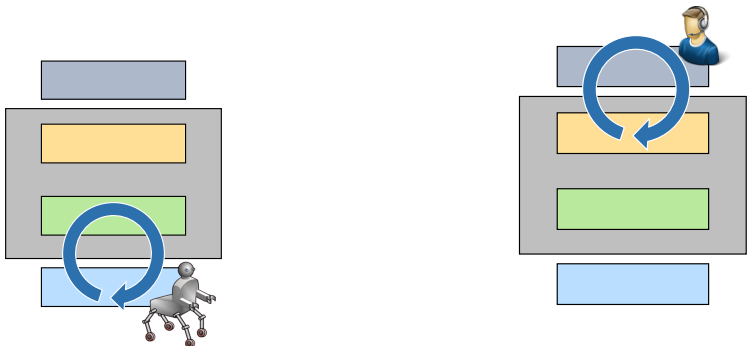


Abb. 3.7: *Das VTB als Abstraktionsebene und Eingabegerät für Nutzer und Entwickler sowie als „Switch“ zwischen realem System und Digitalem Zwilling.*

zu einer sicheren Interaktion mit dem System durch das VTB.

3.4.3 Der Digitale Zwilling als Schnittstelle

Schon während der Entwicklung von komplexen Systemen wird der DT immer mehr das zentrale Integrationswerkzeug. Dadurch, dass das reale System noch nicht verfügbar ist, ist man gezwungen, Algorithmen und Submodule mit Hilfe des DTs im VTB zu entwickeln. Dieser Ansatz, das VTB für die kontinuierliche Forschung, Entwicklung und Optimierung von robotischen Systemen einzusetzen, führt zu einer ganzheitlichen Entwicklung von Hard- und Software. Im Gegensatz zum vorherigen Kapitel liegt der Fokus hier nicht darauf, den DT als Eingabegerät (bzw. Mensch-Maschine-Schnittstelle), sondern als Integrator mit weiteren Schnittstellen zu nutzen.



(a) *Simulation-based Control (SBC)* ***(b)*** *Simulation-based Support (SBS)*

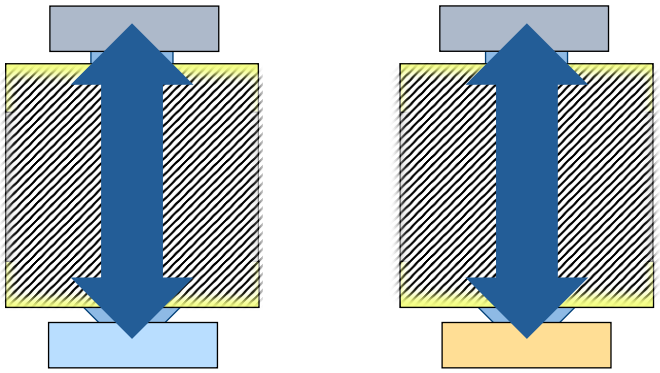
Abb. 3.8: *Das VTB als Schnittstelle, sowohl zum Menschen als auch zum robotischen System.*

In Anwendungen wie dem Motion Simulator [204] als ein Beispiel für SBC (siehe Abb. 3.8a) sieht man, dass diese hoch spezialisierten Systeme oft spezialisierte Schnittstellen erfordern. Im Bereich von SBS dagegen spielt vor allem die Schnittstelle zwischen Operator und Simulationssystem die entscheidende Rolle (siehe Abb. 3.8b). Im Rahmen dieser Arbeit sollen beide Systeme in Einklang gebracht werden. Generell ist es so, dass auch in der Robotik und Simulationstechnik die Standardisierung von Schnittstellen und Datenstrukturen mehr und mehr Anklang finden. Daher ist Teil dieses Konzepts, das

VTB basierend auf diesen Standards zu erweitern, sodass ein kollaborativer Entwicklungsprozess entsteht und im Endeffekt intuitiv steuerbare, intelligente robotische Systeme entstehen. Konkreter bedeutet dies für das VTB (a) die Entwicklung standardisierter Schnittstellen, (b) die Integration notwendiger Funktionalitäten und Frameworks sowie (c) das Einhalten spezieller Anforderungen an das Simulationssystem. Solche Anforderungen können zum Beispiel im Bereich der Datenhaltung, der Prozessgeschwindigkeit oder der Parallelisierung von Prozessen liegen.

3.4.4 Der Digitale Zwilling als Abstraktionsschicht für den Nutzer

Durch die zuvor geschilderten Konzeptbestandteile wird der DT zur Abstraktionsschicht für den Nutzer (vgl. Abb. 3.9). Mittels intuitiver Interaktion mit dem VTB, kombiniert mit vorhandenen und neu integrierten Funktionalitäten, gekoppelt mit dem realen System durch geeignete, standardisierte Schnittstellen, ist es dem Nutzer sofort möglich, mit dem System zu interagieren. Hierbei dient der DT wiederum als Möglichkeit, dieses Interaktionssystem zu testen bzw. auch in der Endanwendung zu nutzen.



(a) *Reales System* ***(b)*** *Digitaler Zwilling*

Abb. 3.9: *Das VTB als Abstraktionsschicht für den Nutzer: Die Komplexität des Systems wird dank und durch 3D-Simulationstechnik versteckt sowohl bei Nutzung des realen Systems (a) als auch des Digitalen Zwillings (b).*

Die Abstraktion als solche kann in folgenden Kategorien erreicht werden:

1. direkte Kontrolle des Roboters mit Hilfe natürlicher Eingabegeräte des VTBs,
2. Visualisierung interner Parameter sowie Online-Simulationsergebnissen,
3. Standardisierung der Schnittstellen für beliebiges Verschalten interner und externer Softwaremodule oder -pakete,
4. Modularität, Flexibilität und Austauschbarkeit von Ein- und Ausgabegeräten, internen und externen Frameworks, simulierten oder realen Endgeräten system-, anwendungs- und plattformübergreifend.

Somit wird das VTB zum „Schweizer Taschenmesser“ mit integrierter 3D-Simulation, Datenhaltung, Frameworks und dem I/O Board als Austauschmedium sowie der Möglichkeit des Dateiaustauschs und Verbindungsmöglichkeit mit externen Frameworks –

schlussendlich mit dem Ziel, einen intuitiveren und natürlicheren Zugang zu komplexen Systemen zu ermöglichen und das Feld der MMI und MRK voran zu treiben.

KAPITEL 4

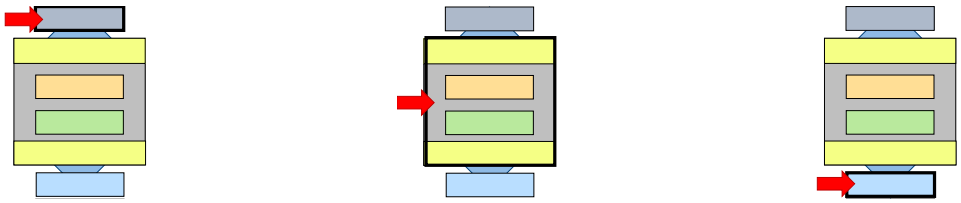
Referenzimplementierung

In diesem Abschnitt wird auf die Umsetzung bzw. Implementierung des Konzeptes eingegangen. Insgesamt handelt es sich um eine Implementierung von Methoden, die szenariounabhängig entwickelt werden. Diese Implementierung wird in Kap. 5 anhand verschiedener Anwendungen präsentiert.

4.1 Beschreibung des Implementierungsvorhabens

Analog zum Konzept umfasst das Implementierungsvorhaben alles, was nötig ist, damit Entwickler und Endnutzer mithilfe und durch den Digitalen Zwilling intuitiv und sicher mit komplexen Systemen interagieren können.

Zentral ist die *Einbettung des Digitalen Zwillings* in das VTB und die zugrundeliegende 3D-Simulation. Wie im Konzept skizziert, wird hierbei ein modularer Ansatz gewählt, um die zentralen Aspekte *Integration*, *Schnittstellen* und *Standardisierung* des DTs in der 3D-Simulation bzw. dem VTB zu erreichen. Umgesetzt wird dieses für das Simulationssystem VEROSIM (vgl. Kap. 2.5.1.8 und Kap. 2.5.3.4).

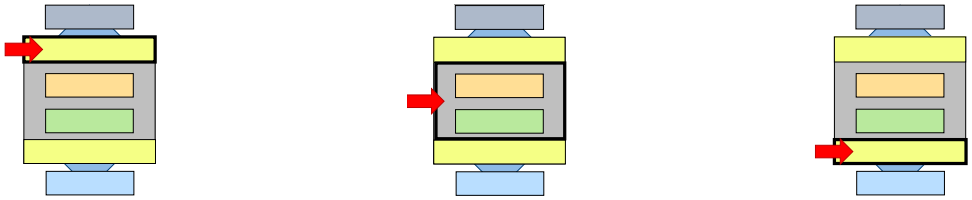


(a) *Nutzer* ***(b)*** *VTB* ***(c)*** *Reales System*

Abb. 4.1: *Piktogramme der Konzeptbestandteile: die drei System-Layer (vgl. Abb. 3.4)*

Daraus folgt eine Einteilung in drei zentrale System-Layer (Nutzer, VTB, reales System, siehe Abb. 4.1) und weitere drei VTB-Sub-Layer (Nutzerschnittstelle, 3D-Simulationskern, Hardwareschnittstelle – siehe Abb. 4.2), wobei der Simulationskern die Einbettung des DTs sowie die Datenverarbeitung beinhaltet (vgl. Abb. 4.3). Diese Einteilung ist auch konzeptionell in Abb. 3.4 und Abb. 3.5 gezeigt.

Diese Einteilung ist in der Implementierung wiederzufinden und wird in den folgenden Kapiteln wieder aufgegriffen. Hierzu wird in Kapitel 4.2 die Implementierung von



(a) Nutzerschnittstelle *(b) VTB-Kern* *(c) Hardwareschnittstelle*

Abb. 4.2: *Piktogramme der Konzeptbestandteile: die drei VTB-Sub-Layer (vgl. Abb. 3.4)*



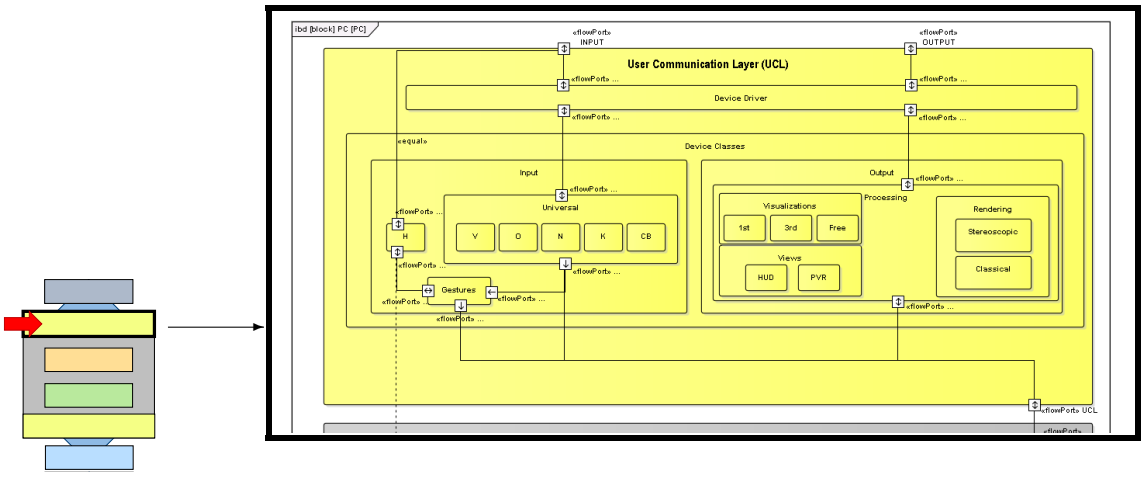
(a) Digitaler Zwilling *(b) Datenverarbeitung*

Abb. 4.3: *Piktogramme der Konzeptbestandteile des 3D-Simulationskerns: DT und Datenverarbeitung (vgl. Abb. 3.4)*

Nutzerschnittstellen, u. a. anhand der Anbindung verschiedener Peripheriegeräte, vorgestellt. In Kapitel 4.3.3 werden notwendige Funktionalitäten in das VTB integriert sowie in Kapitel 4.3.2 die Standardisierung des und durch den DT implementiert. Im darauffolgenden Kapitel 4.4 wird die Schnittstelle zwischen DT und finaler Hardware erläutert, u. a. mit der Anbindung des Robot Operating System (ROS), bevor in Kapitel 4.5 das gesamte System zur Kopplung von DT und physischem Asset mittels VTBs beschrieben wird.

Im Mittelpunkt steht in der gesamten Entwicklung die Modularität aller einzelnen Komponenten. Dies ermöglicht am Ende ein auf die Anwendung zugeschnittenes Nutzerinterface sowie die Ausschöpfung aller Möglichkeiten, die der DT generell bietet.

4.2 Entwicklung intuitiver Nutzerschnittstellen



(a) *(b)*

Abb. 4.4: *Konzept: Nutzerschnittstellen (vgl. Abb. 3.4)*

Die nächste Generation von Computern birgt im Kern zwei grundlegende Neuerungen: Zum einen werden Inhalte dreidimensional erfahren, zum anderen verändern sich die Interfaces, durch welche mit diesen räumlichen Inhalten interagiert wird. Dies ist im Sinne der allgemeinen MMI sowohl auf die klassische Interaktion mit Software auf herkömmlichen PCs als auch auf beliebige Maschinen anzuwenden. Durch Digitalisierung, Vernetzung usw. ist nahezu jede komplexe Maschine schon jetzt als ein PC anzusehen und wird von nun an auch äquivalent betrachtet.

Im Grunde ist unser Körper das natürlichste Interface, über das wir verfügen. Er ermöglicht uns, auf vielfältige Weise mit Menschen und Gegenständen, also der Realität, zu interagieren. Aber nicht für jede Art von digitalem Inhalt wird sich ein solches Interface eignen. Häufig ist eine Abstraktion notwendig. Beispielsweise ist eine Tastatur schneller bedient als ein Buchstabe in die Luft gezeichnet. Insgesamt wird die Auswahl an Interfaces aber steigen.

Dies ist die Ausgangssituation zur Entwicklung neuartiger Schnittstellen, die eine intuitive Interaktion mit komplexen Systemen ermöglichen soll. Hierzu gehört immer sowohl die Eingabe als auch die Ausgabe, d. h., sowohl eine Tastaturanbindung als auch ein guter Texteditor werden benötigt, um dem Nutzer das Verfassen von Texten zu ermöglichen bzw. zu erleichtern.

4.2.1 Mensch-Maschine-Schnittstelle der 3D-Simulation

Im Grunde beinhaltet die Schnittstelle zwischen Mensch und Maschine primär den Seh-, Hör- und Tastsinn des Menschen. Um eine „Sprache“ für Nutzerschnittstellen zu entwickeln, muss man erst einmal darauf schauen, was heute schon intuitiv funktioniert. Zum Beispiel hat sich durch Smartphones eine intuitiv verständliche Pinch-to-Zoom-Geste

etabliert, welche von den meisten Menschen als natürlicher Bestandteil jeder App gesehen wird, auch wenn diese nicht expliziert erklärt wird. Solche Interfaces haben aber eher Seltenheit. Vielmehr setzt man darauf, klassische Menüs weiterhin zweidimensional einzublenden, auf die man dann mit einer dreidimensionalen Maus oder sogar dem Finger tippen kann. Leider birgt gerade diese „natürliche“ Interaktion das zuvor beschriebene Risiko des nicht vorhandenen Feedbacks, das eine intuitive Aktion unnatürlich wirken lässt. Es ist insgesamt sehr schwer, eine allgemeingültige Sprache für intuitive Nutzerschnittstellen zu definieren. Vielmehr ist diese Sprache abhängig von der gewählten Interaktionshardware und der zu interagierenden Software.

Allgemein kann man aber zwischen der visuellen Wahrnehmung und der direkten Interaktion zwei separate Aspekte der Nutzerinteraktion festhalten. In diesen Grenzen der *Eingabe* und *Ausgabe* kann man dann weitere Unterteilungen anhand der genutzten Methodik bzw. Hardware aufzeigen:

1. Interaktion,
    a) klassische Interaktionshardware,
    b) optische Systeme,
    c) haptische Eingabegeräte,
    d) virtuelle Eingabegeräte,
2. Visualisierung,
    a) klassische Bildschirme,
    b) (stereoskopische) Daten-Brillen,
    c) Visualisierungsmöglichkeiten/-metaphern.

Im Folgenden werden anhand dieser Auflistung neue Interaktionsmodalitäten für das VTB implementiert. Zusätzliche Informationen sind im Anhang 7 aufgeführt.

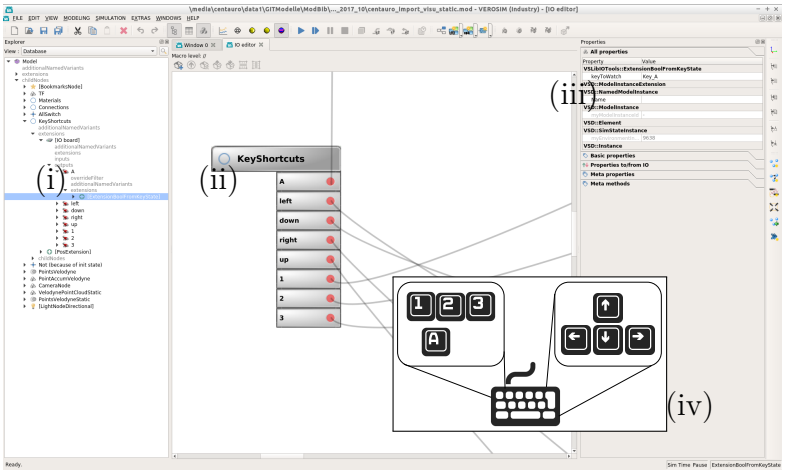
4.2.2 Erweiterung klassischer Eingabegeräte

Gerade im Bereich des klassischen Engineerings sind Tastatur, Maus und Bildschirm immer noch nicht wegzudenken. Weitere Ergänzungen wie eine 3D-SpaceMouse erleichtern zumindest die Arbeit der Designer und Konstrukteure, die sich in dreidimensionalen Räumen bewegen müssen. Mehr aus der Spieleindustrie sind vor allem im Bereich der mobilen Robotik auch Gamepads oder Joysticks als einfache – und für Spieler sehr intuitive – Eingabegeräte hinzugekommen. Erweiterungen dieser klassischen Eingabegeräte lassen sich zum einen durch neue Steuerungskonzepte erreichen, wie sie in Kap. 2.4.3 präsentiert werden, oder durch eine Umdefinition der Funktionalitäten, wie im Folgenden gezeigt.

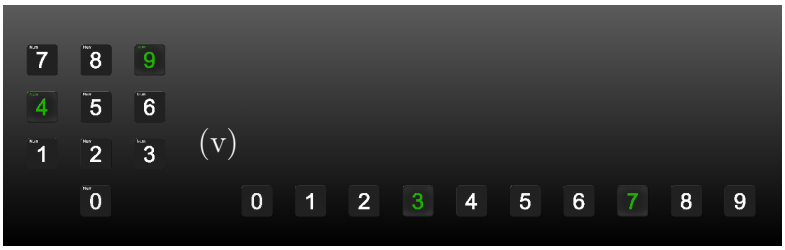
4.2.2.1 Tastaturerweiterung

Schaltpulte im Bereich der Musikindustrie oder Instrumentenpanel wie bei einem Flugsimulator inspirieren zur intuitiven Kombination von Ort und Funktion. So ist es vielen heute schon möglich, blind mit zehn Fingern zu schreiben und ein Pilot weiß nach einer gewissen Eingewöhnungszeit ohne zu schauen, welcher Schalter welche Funktion hat. Im

Bereich des Engineerings bietet es sich demnach an, die Funktion der Tastatur mehrfach zu nutzen. Dies hat den inhärenten Vorteil, dass jeder PC schon über ein Keyboard verfügt und auch die Neuanschaffung eines zweiten erschwinglich ist.



(a) I/O Board



(b) Ausschnitt: Renderfenster-Overlay

Abb. 4.5: *Tastaturerweiterung: Key Toggler mit (i) eingehängter Extension im VSD-Baum, (ii) I/O Board, um Ein- und Ausgänge zu verbinden, (iii) Eigenschaftenfenster zur Auswahl der jeweiligen Taste, (iv) der klassischen Tastatur als Eingabegerät und (v) einem HUD des Numpads und der Zahlen*

Hierzu wurde eine einzige `ExtensionBoolFromKeyState` implementiert, welche eine Erweiterung für ein I/O Board Output darstellt (siehe Abb. 4.5). Nach Auswahl eines Keys (also einer Taste der Tastatur) können mit Hilfe des Qt Signal/Slot Pattern QKeyEvents registriert und auf einen binären Ausgang gelegt werden. Bei erneutem Betätigen wird der Ausgang getoggelt.

So kann die Tastatur während der Modellierung als Tastatur genutzt und nach Simulationsstart zu einem Instrumentenpanel werden, das sofort für jedermann intuitiv zu bedienen ist. Ein weiterer Vorteil der Implementierung ist, dass diese nur bei selektiertem Renderfenster den KeyState empfängt und man somit im Explorer oder Property-Fenster uneingeschränkt seine normale Tastatur nutzen kann. Des Weiteren ist dies bei vielen Optionen im I/O Board ein sehr guter Weg, den Überblick zu behalten bzw. einfach einzelne Funktionen und Datenströme zu- oder abzuschalten.

4.2.3 3D-, Positions-, Tracking- und Kamera-basierte Eingabegeräte

Im Bereich der Motion-Capture-Systeme sind neben der Hardware auch die Verarbeitungsalgorithmen und -methoden von zentraler Bedeutung. Während ein Gyrosensor direkt eine (relative) Position und Orientierung übermittelt, steht bei kamerabasierten Eingabegeräten vor allem die Bildverarbeitung im Mittelpunkt. Darauf aufbauend kann man noch weitere Methoden anschließen, wie z. B. die Gestenerkennung oder eine audiovisuelle Rückkopplung.

Stellvertretend für kamerabasierte Eingabegeräte sollen im Folgenden Microsofts Kinect, Leap Motion und anschließend allgemein die mit solchen Eingabegeräten verbundene Gestenerkennung vorgestellt werden. Des Weiteren wird das Myo-Armband von Thalmic Labs als Spezialfall und die HTC Vive Controller als klassischer Fall controllerbasierter Eingabegeräte präsentiert.

4.2.3.1 Generelles Integrationskonzept

Für eine effiziente Implementierung verschiedener Eingabegeräte wird ein einheitliches Integrationsschema verwendet, das in Abb. 4.6 exemplarisch für den Leap Motion Controller dargestellt ist. Hierbei wird ein generisches UI-Framework entwickelt, welches mit dem gesamten Spektrum und allen Funktionen des VTBs sowie den internen Frameworks interagieren kann. Ein modulares, objektorientiertes Implementationsschema führt zu einem Layer an Eingabegeräten, losgelöst vom Simulationssystem. Per I/O Board lassen sich die Starrkörpersimulation (RBS), die Sensor Simulation (SS), das Rendering (R) oder andere Frameworks ansprechen.

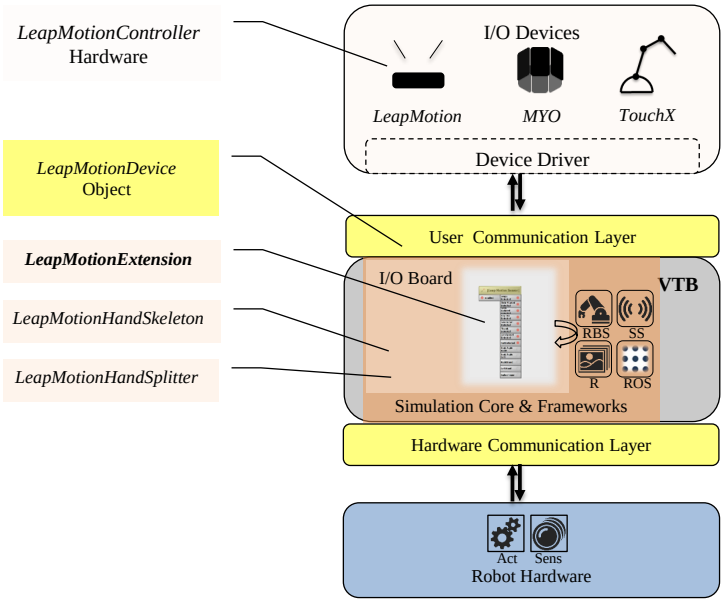


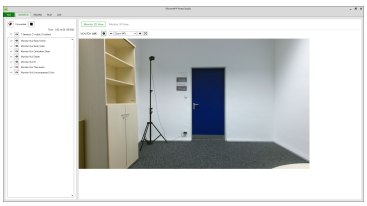
Abb. 4.6: *Integrationskonzept, exemplarisch für das Leap Motion*

Aus Nutzersicht gibt es in der Simulationsumgebung eine `LeapMotionExtension`, die in ein beliebiges 3D-Modell im VTB eingefügt werden kann. Die Extension besitzt

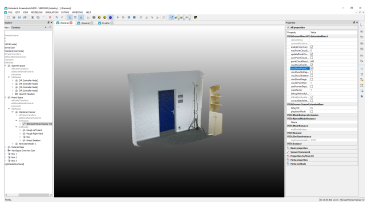
ein Objekt `LeapMotionDevice` im User Communication Layer. Dieses Objekt stellt die Kommunikation mit dem Leap Motion Sensor unter Ausnutzung des LeapMotion SDKs und den Gerätetreibern her. Das `Device` fragt die Daten (wie Position, Frames, Hand-Exoskelett usw.) ab, die dann an die `Extension` weitergegeben werden. Hier können diese in VEROSIM-Datentypen umgewandelt und über das I/O Board verteilt werden. In der Regel werden alle `Device`-Funktionen in einem separaten Thread (mittels QThread oder QConcurrent) ausgeführt. Des Weiteren ist es möglich, eine zusätzliche Visualisierung (`Visu`) zu erstellen, die, basierend auf den `Device`-Daten, beispielsweise ein Hand-Exoskelett (`HandSkeleton` und `HandSplitter`) in das Renderfenster zeichnet und vom Rendering Framework aufgerufen wird. Wie auch Laserscanner oder Kameras betten sich Sensoren, die als Eingabegerät genutzt werden, auch in das zuvor beschriebene Sensorframework ein und erben von der gemeinsamen Basisklasse `VSLibSensor::SensorExtensionBase`.

4.2.3.2 Microsoft Kinect v2

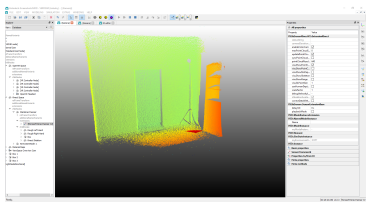
Die Kinect v2 (kurz: Kinect) wurde als Eingabegerät für die Spielekonsole XBox One entwickelt und kann durch Körperbewegungen die Software bedienen.



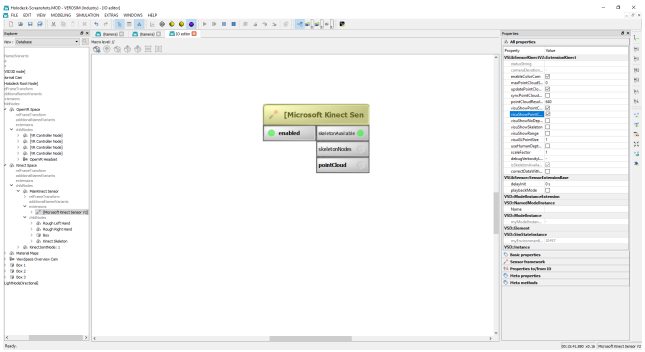
(a) RGB-Bild



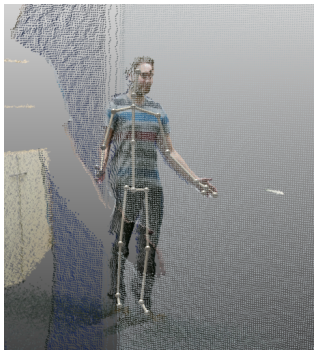
(b) Kolorierte Punktwolke



(c) Tiefenbild



(d) I/O Board, Explorer, Properties



(e) Skelett-Visualisierung

Abb. 4.7: *Microsoft Kinect v2: Sensorintegration*

Durch ein gut strukturiertes SDK kann man die Kinect auch in der Forschung und Entwicklung nutzen, wobei hier zwei Anwendungsformen unterschieden werden müssen:

1. die Kinect als Sensor
2. die Kinect als Eingabegerät

Analog zum zuvor beschriebenen Integrationskonzept wird eine `KinectV2Extension` implementiert, welche in einem separaten Thread auf die Daten des `KinectV2Device`

zugreift und an das I/O Board weiterleitet. Neben den Rohdaten liefert die Kinect dabei auch das Skelett mit, das als neuer Datentyp auch im I/O Board verfügbar gemacht wird (vgl. Abb. 4.7d). Des Weiteren wird eine `KinectV2Visu` entwickelt, die dieses Skelett in die 3D-Umgebung zeichnet (vgl. Abb. 4.7e).

Somit ist die Kinect sowohl als Eingabegerät als auch als Sensor (vgl. Abb. 4.7 a-c) angebunden und kann entsprechend genutzt werden.

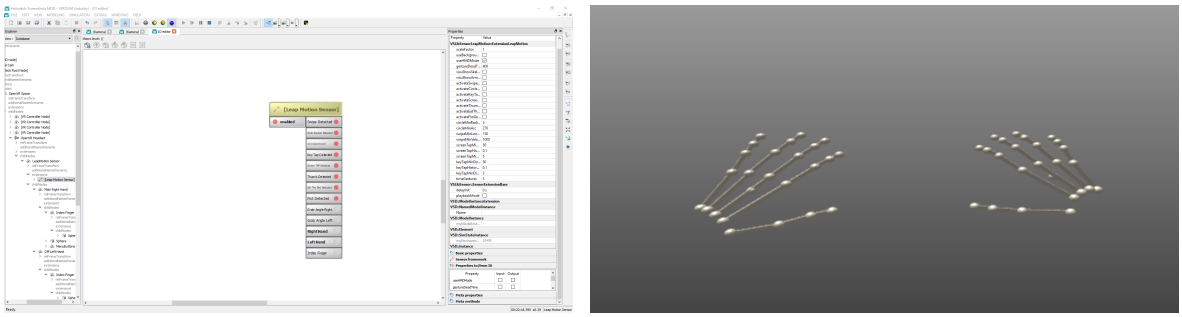
4.2.3.3 Leap Motion Controller

Der Leap Motion Controller (oft auch einfach *Leap Motion* genannt) gehört zu der Klasse der kamerabasierten Eingabegeräte.

Per USB werden die Daten als schwarz-weiß Bild weitergeleitet. Innerhalb per Leap Motion Software wird keine Tiefenkarte erzeugt, sondern das gelieferte Schwarz-Weiß-Bild wird mit Hilfe von diversen Algorithmen überarbeitet. Diese Bearbeitung beinhaltet mehrere Schichten:

- Compensation Layer (Background, Light),
- Tracking Layer (matching, filtering, smoothing),
- Transport Protocol ,
- Application Programming Interface (API).

Nach Kompensation von Hintergrundobjekten (Kopf und Lichtquellen) werden die 3D-Objekte, die gesehen werden, rekonstruiert (d. h. 3D-Rekonstruktionslayer). Daraufhin werden im Trackinglayer die gewonnenen Daten mit Position und Orientierung von bekannten Objekten wie Hände oder Finger gematched. Im Transportlayer werden diese Daten dann via TCP oder WebSocket weitergeleitet. Darauf aufbauend kann die API Objekt-orientiert in C++-Code eingebunden werden.



(a) *I/O Board, Explorer, Properties* ***(b)*** *Skelett-Visualisierung*

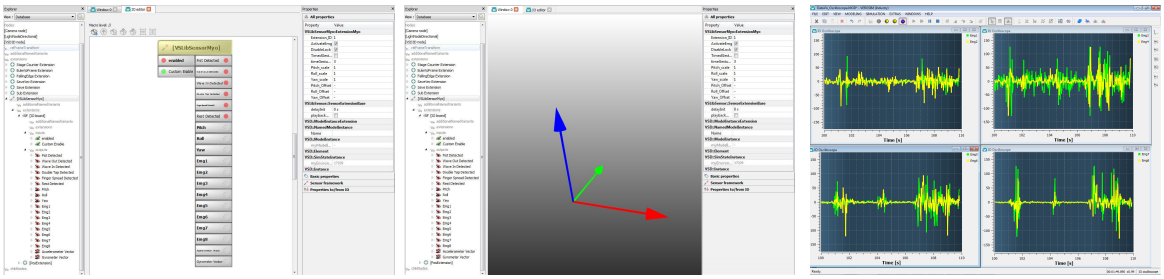
Abb. 4.8: *Leap Motion Controller: Sensorintegration*

Das mitgelieferte SDK beinhaltet bereits eine optimierte Gestenerkennung für Handgesten. Verschiedene Gesten wie z. B. „Thumbs up to continue“, „Pinch with both hands to create a block“, „Grab a block“, „lift both hands to turn gravity off“ oder „change block shape“ sind für Beispielanwendungen vorgesehen.

Analog zur Kinect werden für den Leap Motion Controller `Extension`, `Device` und `Visu` implementiert. Neben der Skelett-Visualisierung (vgl. Abb. 4.8b) werden über das I/O Board auch durch das SDK erkannte Gesten als Boolean ausgegeben (vgl. Abb. 4.8a).

4.2.3.4 Thalmic Labs Myo-Armband

Analog zum beschriebenen Integrationskonzept wird hier das Myo-SDK genutzt und in das interne Sensorframework eingebettet. Ein `MyoDevice` fragt alle Myo-Daten ab, die von einem Myo-Armband kommen und gibt diese an eine `MyoExtension` weiter. Eine zusätzliche `MyoDataCollector`-Klasse kann genutzt werden, um per Signal/Slot-Entwurfsmuster eintreffende Daten verfügbar zu machen. Im I/O Board werden dann erkannte Gesten binär (Boolean), die Orientierung, Accelerometer- und Gyroskop-Daten als Vektoren sowie EMG-Daten als acht Ganzzahlen (Integer) ausgegeben (siehe Abb. 4.9).



(a) I/O Board, Explorer, Properties *(b) Visualisierung* *(c) Oszilloskop der EMG-Signale*

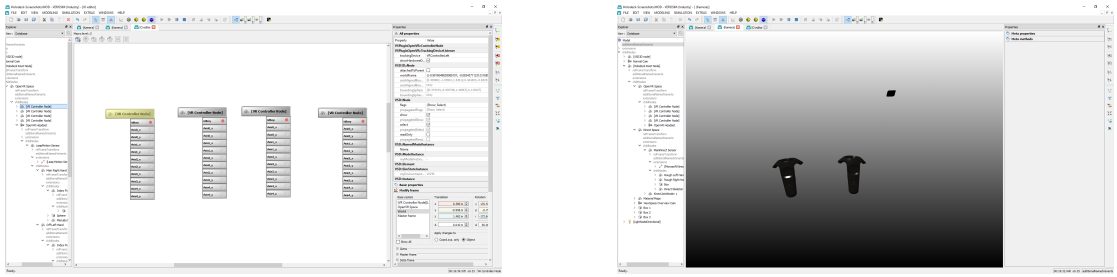
Abb. 4.9: *Thalmic Labs Myo-Armband: Sensorintegration*

Im Vergleich zu optischen Sensoren hat das Myo den Vorteil, weniger von externen Störeinflüssen (wie Licht oder Verdeckung) beeinflusst zu werden. Die Genauigkeit des Myo ist dagegen bei weitem nicht so gut wie die des Leap Motion Sensor, welche bis in den Submillimeter-Bereich geht. Auch das Abgreifen des EMG-Signals ist im Vergleich zur heutigen optischen Technologie nicht so verlässlich. Zur Evaluation wurden Experimente mit mehreren Probanden durchgeführt, welche (1) die Gestenerkennungsrate, (2) die unterschiedliche Erkennungsrate von Gesten durchgeführt mit unterschiedlicher Geschwindigkeit und (3) die Wiederholgenauigkeit verschiedener Muster beinhalten (siehe Anhang 7).

4.2.3.5 HTC Vive Controller

Der Vollständigkeit halber soll hier auf die HTC Vive Controller hingewiesen werden. Da diese zum SDK der jeweiligen stereoskopischen VR-Brille gehören, entspricht die Implementierung hier nur in Teilen dem vorgestellten Integrationskonzept. Mit Hilfe des OpenVR-SDKs [ca] werden Controller automatisch erkannt und visualisiert (siehe Abb. 4.10).

Über das I/O Board wird hier das Trackpad (x, y) (als Fließkommazahlen) und die verschiedenen Button-States (als Boolean) ausgegeben. Die Pose der Controller und der Basisstationen wird zusätzlich als Property angezeigt und lässt sich somit zusätzlich über das I/O Board ausgeben.

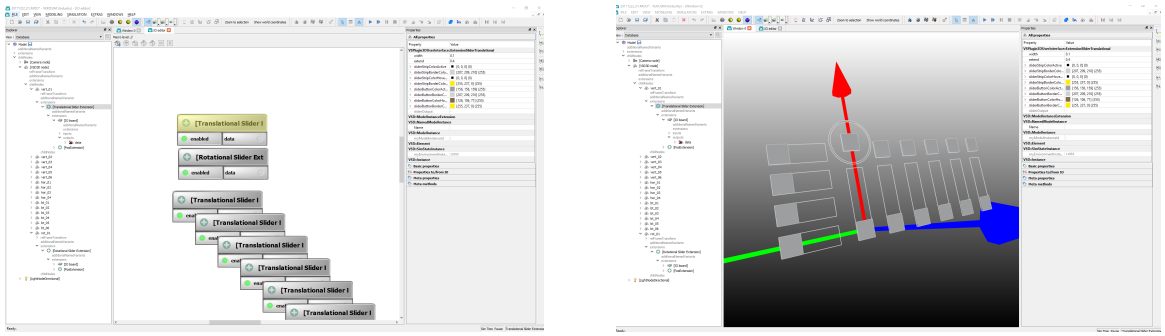


(a) I/O Board, Explorer, Properties *(b) Visualisierung*

Abb. 4.10: *HTC Vive Controller: Integration*

4.2.3.6 Virtuelle Eingabegeräte

Nutzerinteraktionshardware muss nicht immer physische Hardware im klassischen Sinne sein. Auf Basis klassischer OpenGL-Renderfunktionen und einem Mouse Handler werden daher Knöpfe und Slider in die 3D-Szene gerendert, die mit der Maus per Drag und Drop interaktiv bedient werden können (vgl. Abb. 4.11). Per I/O Board kann dann sowohl der Toggle-State (als Boolean) weitergegeben werden als auch eine von 0 bis 100 % skalierte Slidervariable (als Fließkommazahl).



(a) I/O Board, Explorer, Properties *(b) Visualisierung*

Abb. 4.11: *Virtuelle Eingabegeräte: Integration*

4.2.3.7 Interaktionsframework

Zur modularen und effizienten Kopplung der verschiedenen Eingabegeräte mit virtuellen Interaktionsmetaphern ist ein Interaktionsframework nötig, damit man nicht nur mit der klassischen PC-Maus, sondern auch z. B. mit seinen Fingern die Knöpfe und Slider bewegen kann.

Um das Framework zur Interaktion so generisch wie möglich zu halten, wird hierbei ausschließlich auf die euklidische Geometrie zurückgegriffen. Da jeder Knoten einen Frame besitzt und somit mit Position und Orientierung beschrieben wird, lassen sich relative Abstände sehr gut bestimmen. Button liegen stets auf einer virtuellen Ebene (mit Normalen), sind also „PlaneButton“. Durch Ray-Tracing [205] lässt sich dann ein Strahl aus dem Augenmittelpunkt durch die Fingerspitze auf die Buttonebene angeben. So lässt sich

der Button (aus der richtigen Richtung kommend) drücken. Die verschiedener Button-States (*idle*, *hover*, *clicked*, *locked*) lassen sich zusätzlich visualisieren, um dem Nutzer ein Feedback zu geben. Mittels einer Finite State Machine (FSM) können die verschiedenen Button-States getoggelt werden. Damit das Interaktionsframework funktioniert, wird mittels einer `InteractionRoot`-Node eine Kalibration zwischen allen Trackingmodalitäten, die genutzt werden, durchgeführt. In Kap. 5.3.2 wird eine Anwendung gezeigt, in der dies für die HTC Vive (VR-Space), die Kinect v2 (Kinect-Space) und die Leap Motion (Leap-Space) durchgeführt wird, um Kopf-, Körper-, und Handtracking miteinander zu fusionieren und eine fingerbasierte Interaktion zu ermöglichen. Der `InteractionRoot`-Knoten definiert auch die Nullpose des Arbeitsraumes und liegt im Zentrum des VR-Spaces. Mit Hilfe einer kalibrierten Bodenebene und dem Ray-Tracing-Verfahren lässt sich auch eine Teleportation implementieren. Der Schnittpunkt von Strahl und Bodenebene wird per FSM-Trigger zum neuen `InteractionRoot` Node Frame.

Zusätzlich zur Teleportation wurde mit Hilfe einer `MagneticNodeExtension` die Möglichkeit geschaffen, den InteractionRootNode mit Hilfe eines Feder-Dämpfer Systems an beliebig andere Knoten „magnetisch“ anzuziehen. Mit Hilfe des Toggle-Mechanismus aus Kap. 4.2.2.1 lässt sich so ein Set aus vordefinierten, oder dynamisch durch Selektion hinzugefügten Knoten durchschalten.

Abschließend bleibt noch zu sagen, dass nicht nur die Interaktionsmöglichkeiten, sondern auch die Interaktion als solche weitaus komplexer sind als man zunächst annehmen würde. Die Frage, die sich stellt, ist: Was passiert, wenn eine Kraft der Hand ein imaginäres digitales Objekt trifft? Auch das menschliche Bewegungsfeld ist in traditionellen Spielen oft willkürlich eingeschränkt, z. B. durch unüberwindbares Wasser oder Wände, die dem Nutzer unmissverständlich Grenzen aufweisen. In VR führen solche Grenzen dazu, dass Sie durch reale Bewegungen überwunden werden können, aber dies schadet der Immersion extrem. Normalerweise können solche Interaktionen nur durch visuelles Feedback „immersiv“ gestaltet werden. Im nächsten Kapitel 4.2.4 wird ein anderer Lösungsansatz mit Hilfe der Kraftrückkopplung beschrieben.

4.2.4 Peripheriegeräte mit Kraftrückkopplung

Neben den zuvor beschriebenen unidirektionalen Eingabegeräten gibt es auch bidirektionale Ein-/Ausgabegeräte. In Form *haptischer Eingaberäte* machen sich diese das Prinzip der Kraftrückkopplung zu Nutze.

Auch wenn die Integration haptischer Eingabegeräte das gleiche Integrationskonzept verfolgt wie in Kap. 4.2.3 beschrieben, kommt hier die Rückkopplung der Kraft hinzu. Diese Kraft muss entweder vom realen System direkt zurückgegeben werden oder durch die Starrkörpersimulation berechnet werden.

4.2.4.1 Force Feedback Interface (FFI)

FF-Eingabegeräte in der 3D-Simulation erfordern ein durchdachtes Schnittstellenkonzept, sowohl innerhalb der Simulationsumgebung als auch zu den externen Treibern. Hier

wurde ein generisches Interface entwickelt, um (verschiedene haptische) Eingabegeräte mit der Starrkörpersimulation zu vereinen.

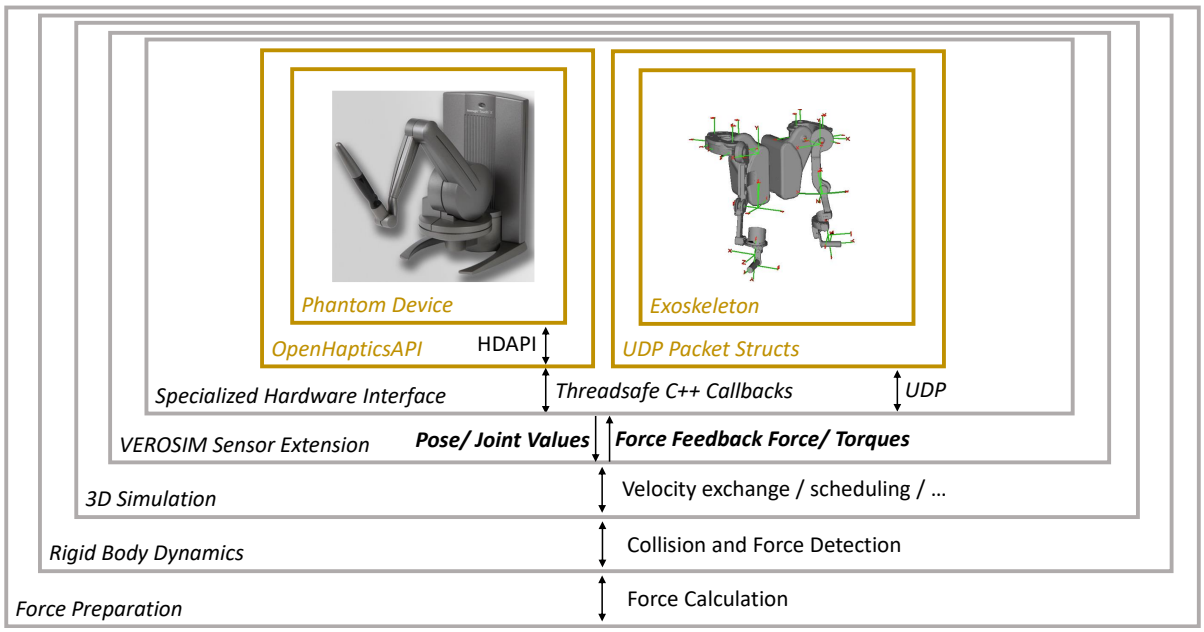


Abb. 4.12: *Modulare FF-Konzeptskizze (basiert auf [206]): Entkopplung von FF-Geräten, deren API und den Frameworks der 3D-Simulation*

Dieses Interface (siehe auch Abb. 4.12) umfasst zum einen eine Kopplung der generischen Sensorextensions mit der Starrkörpersimulation, um effiziente (notwendige) Kraftberechnungen durchführen zu können und zum anderen spezialisierte Hardwareschnittstellen, die die harten Echtzeitbedingungen erfüllen können. Innerhalb der 3D-Simulation ergeben sich dann wiederum drei Layer, welche die Pose oder Gelenkwinkel als Eingabe und eine resultierende Kraft bzw. Momente als Ausgabe liefern:

FF-Interface Layer 1 – Kollisionsdetektion/Eventhandling
Auf Basis der Inputposition und Orientierung folgt das haptische Objekt den Bewegungen des Eingabegeräts. Dieses ermöglicht es, den Eingabeframe des haptischen Eingabegeräts als *haptisches Objekt* zu nutzen (nach Transformation in das Simulationskoordinatensystem), dem ein Starrkörper als *virtueller Proxy* folgt. Dieses dynamische Objekt unterliegt dann automatisch den Gesetzen der Starrkörpersimulation und nutzt die gleiche Kollisionsdetektion und denselben Eventloop. Sobald der dynamische Proxy mit einem anderen Objekt kollidiert, muss die Rückstellkraft berechnet werden.

FF-Interface Layer 2 – Force Feedback Kraftberechnung
Bei einer Kollision kann nun (a) die berechnete Kraft der zwei interagierenden Starrkörper F_{RB}, oder (b) die von (simulierten) Kraft-Momenten-Sensoren (engl. „Force Torque Sensoren (FT)“) aufgezeichnete Kraft F_{FT}, oder (c) ein neuer generalisierter Ansatz namens „Virtual Coupling“ genutzt werden.

Virtual Coupling nutzt ein Feder-Dämpfer-System, beschrieben in [207], [208] oder

[209] mit:

$$F_{SD} = k \cdot x + d \cdot \dot{x} + m \cdot \ddot{x}$$

Hierbei werden eine Varianzanalyse der aktuellen Pose sowie der Zielpose genutzt, um eine (virtuelle) Federrückstellkraft F_{SD} zu berechnen. Diese Variante hat den Vorteil, dass die Kraftdimension unabhängig von den Trägheitstensoren der zwei kollidierenden Körper sind. Wäre dies nicht so, könnten simulierte Kollisionen sehr schnell die Schwellwerte des FF-Devices überschreiten. Daher wird (c) genutzt, um die Richtung und Amplitude der Kraft zu berechnen, wobei die Kollisionsdetektion von der Starrkörpersimulation durchgeführt wird. Das Interface zum haptischen Eingabegerät geschieht, wie im allgemeinen Konzept erläutert, in einem separaten Thread.

FF-Interface Layer 3 – Hardware Schnittstelle
Gerätespezifische Charakteristika sollen unabhängig von dem generischen Interface durch die vom Hersteller bereitgestellten Treiber in spezialisierten VEROSIM Extensions eingebunden werden. Generell müssen hierbei immer Endeffektor- bzw. TCP-Pose oder ein Set aus Gelenkwinkeln oder -momenten übermittelt werden. Zurückgegeben werden dann entweder die Kräfte und Momente am Endeffektor oder Gelenkmomente.

Aufbauend auf diesem Interface wird im Folgenden anhand von einem kommerziellen und einem selbst entwickelten haptischen Eingabegerät gezeigt, wie diese in das VTB integriert werden und mit dieser zusammenarbeiten.

4.2.4.2 Sensables Geomagic Touch X

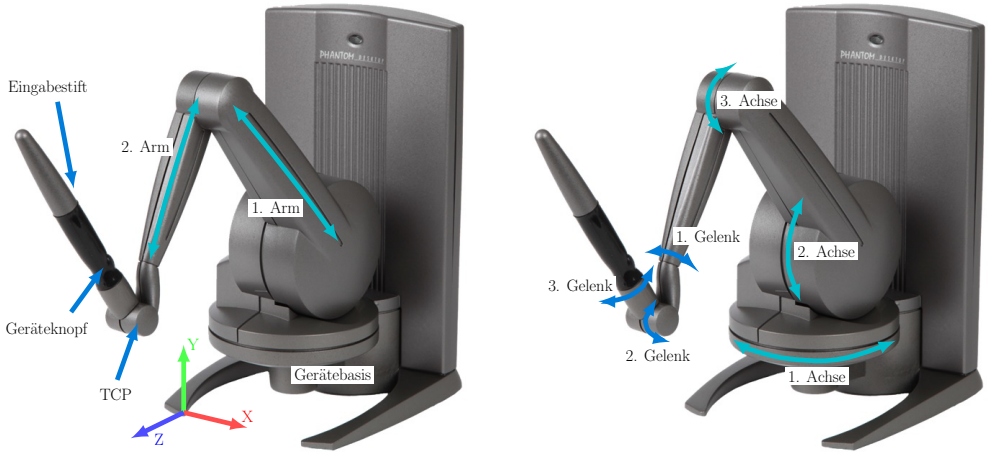
Ein Beispiel für ein kommerziell vertriebenes, haptisches Eingabegerät, das den zuvor definierten Anforderungen genügt, ist das *Geomagic Touch X* – auch *PHANToM Desktop* genannt – des Herstellers *Sensable*.

Das in Abb. 4.13 dargestellte *Geomagic Touch X* [210] verfügt als Eingabegerät über 6 Freiheitsgrade, die zur Darstellung der Position und Orientierung im dreidimensionalen Raum ausgenutzt werden und als Ausgabegerät über ein Force Feedback in den ersten drei Achsen.

Das Phantom Device liefert als Input die Position p^{TouchX} des TCP und erwartet als Rückgabe eine berechnete FF-Kraft F_{FF}^{TouchX}, was mit dem zuvor beschriebenen Konzept (vgl. Abb. 4.12) erreicht wird.

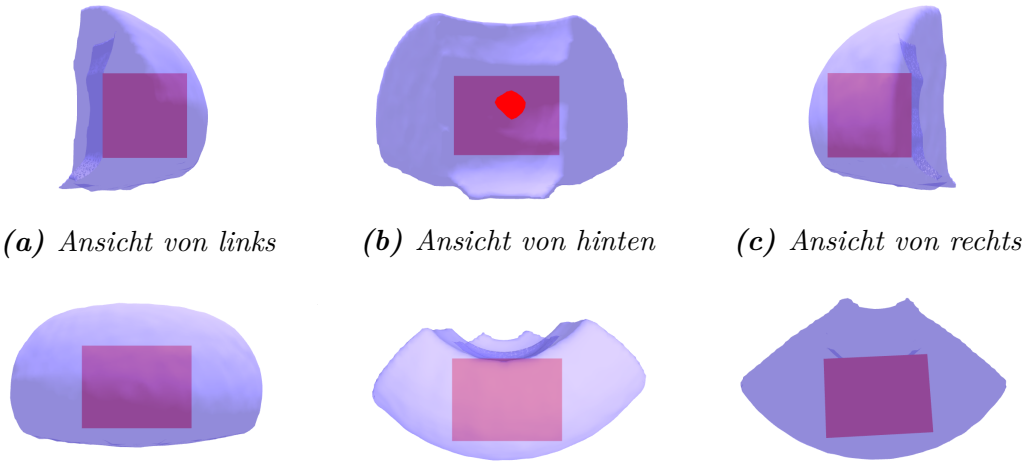
Sensable spezifiziert für den *Geomagic Touch X* sowohl einen physikalischen als auch einen nutzbaren Arbeitsraum, in denen sich der Nutzer bewegen kann. Zur besseren Anschauung wurde dieser in Abb. 4.14 visualisiert. Weitergehende Untersuchungen insbesondere zur Kinematik des Geomagic Touch X sind im Anhang 7 zu finden.

Ein „Proof of Concept“-*Evaluationssetup* ist in Abb. 4.15a und Abb. 4.15b gezeigt, wobei ein simples 3D-Modell genutzt wird, um die Kollisionsdetektion in VEROSIM zu testen und exakte Kraftamplituden und -vektoren in MATLAB zu analysieren.



(a) Bezeichnungen *(b) Kinematischer Aufbau*

Abb. 4.13: *Aufbau des Geomagic Touch X [n]. Die Benutzereingabe geschieht über die drei Achsen und die drei Gelenke, die die Gerätebasis mit dem Eingabestift verbinden. Die Rückkopplungskraft wird über Motoren in den drei Achsen ausgegeben. Der TCP befindet sich am Ende des Eingabestiftes. Das Koordinatensystem in (a) ist die Referenz des SDKs.*



(a) Ansicht von links *(b) Ansicht von hinten* *(c) Ansicht von rechts*

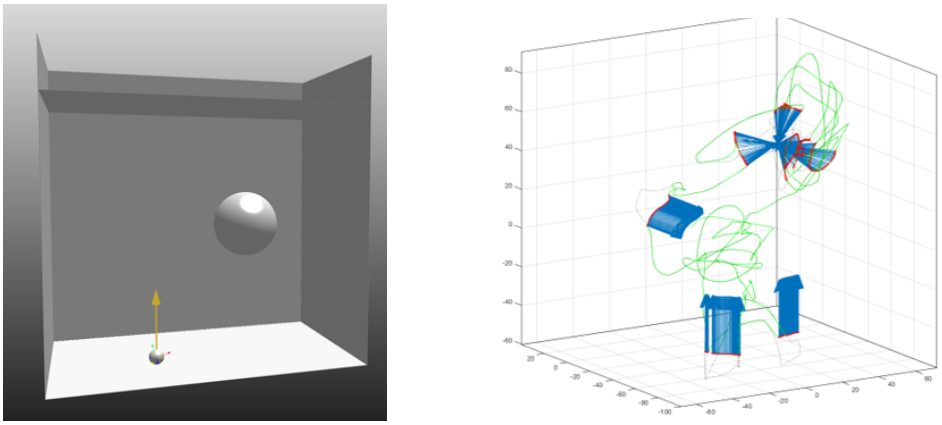
(d) Ansicht von vorne *(e) Ansicht von oben* *(f) Ansicht von unten*

Abb. 4.14: *Vergleich des physikalischen und nutzbaren Arbeitsraumes des Geomagic Touch X aus Sicht des Benutzers.*

4.2.4.3 (Ganzkörper-)Exoskelette

Als ein weiteres, maßgefertigtes, selbst gebautes, haptisches Eingabegerät für die Kontrolle humanoider Roboter sind sogenannte Exoskelette eine interessante Alternative, um eine direkte Abbildung des menschlichen Oberkörpers mit Armen und Händen auf dem Roboter darzustellen.

Meist werden Exoskelette heute im Bereich der Rehabilitationsmedizin eingesetzt,



(a) VEROSIM-Modell *(b) Analyse in MATLAB*

Abb. 4.15: *FF-Interface: Analyse des PHANToM Device: Kopplung von Kollisionsdetektion, Feder-Dämpfer-System als Proxy und gerätespezifischer API. Proof of Concept anhand eines 3D-Modells zur Analyse von Kollisionspunktbestimmung, Kraftamplitude und -richtung.*

kommen aber mehr und mehr auch in den Fokus aktueller Forschung, insbesondere für teleoperativ gesteuerte menschenähnliche Systeme. Ein solches System, entwickelt an der Scuola Superiore Sant'Anna, wird in Abb. 4.16 gezeigt. Das genutzte Exoskelett besteht aus zwei Armen mit jeweils einem Handgelenk und einer Hand. Jeder Arm hat 4 aktive Degrees of Freedom (DoF) (Schulter, Abduktor, Rotation und Flexion) und 2 passive DoFs (Unterarm Pro-/Supination und Gelenkflexion). Jedes Handgelenk hat 3 aktive Rotationsfreiheitsgrade, jede Hand 17 unteraktuierte DoFs (bzw. 5 aktive DoFs – für jeden Finger einen). Insgesamt soll das Exoskelett mehr als 90 % des natürlichen Arbeitsraumes des menschlichen Armes abbilden. Das Exoskelett kann entweder im „Force Mode" Endeffektor-Positionen bzw. ein Set aus Gelenkmomenten übertragen oder im „Compliant Position Mode" vordefinierte Trajektorien mit den assoziierten Steifheitsparametern übertragen.

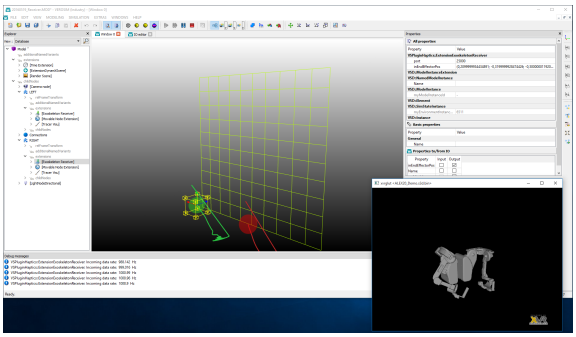


(a) Bild *(b) CAD-Daten*

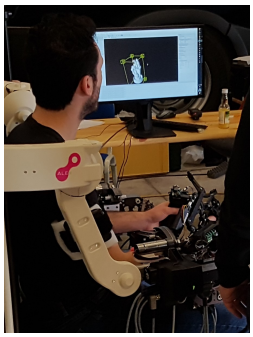
Abb. 4.16: *Exoskelett: Active Leg Exoskeleton (ALEx) [211] [212]*

Auch mit einem Exoskelett ist die Möglichkeit implementiert, entweder die Endeffektorposition oder ein Set aus Gelenkwinkeln/-momenten an das VTB zu übertragen. Da dieses Eingabegerät einzigartig ist und kein eigenes SDK mitliefert, wird hierbei das User Datagram Protocol (UDP) als Austauschmedium verwendet.

Anhand vordefinierter UDP-Pakete lassen sich so „Exoskeleton Device Command Structs“ (EDCS) (Tab. 7.4) und „Exoskeleton Device Data Structs“ (EDDS) (Tab. 7.5) vom Exoskelett an das VTB übermitteln oder an das Exoskelett senden. Hierbei wird das EDDS als Input für den Hostrechner bzw. das VTB genutzt, welches als Output ein EDCS ausgeben kann. Ein solches EDCS kann sowohl im „Force Mode“ (FM) als auch im „Compliant Position Mode“ (CPM) übermittelt werden. Hierbei nutzt der FM die Gelenkmomente τ_i^{out} für jedes Gelenk i und der CPM nutzt ausschließlich die Endeffektorposition p_{ee}^{out}. Das Exoskelett EDDS nutzt meist die Position des Endeffektors als zentralen Input p_{ee}^{in}. Anhand des zuvor beschriebenen Integrationskonzeptes (vgl. Abb. 4.12) wurde eine UDP-Schnittstelle implementiert, welche genauer in Kap. 4.4.2 vorgestellt wird.



(a) Exoskelett-Simulation steuert Endeffektorposen in VEROSIM



(b) Schunk-Hand-Simulation gesteuert per Handexoskelett



(c) Teleoperation mit Hilfe des Ganzkörperexoskeletts

Abb. 4.17: *Exoskelett: UDP 1 kHz-Loop von Arm und Hand Exoskelett*

4.2.5 Wissenschaftliche Aufbereitung und Visualisierung interner Parameter

Die Aufbereitung und Visualisierung eingehender und ausgehender Datenströme als zentrale Elemente der Nutzerinteraktion soll im Folgenden vorgestellt werden. Dies unterteilt sich in:

- Immersion und Visualisierungen als Werkzeug der Nutzerintegration,
- Visualisierung und Inspektion von Daten sowie
- Aufbereitung und Optimierung von Daten.

Allgemein kann man dies als „Parametervisualisierung“ bezeichnen. Hierbei wird, wie in Kap. 2.3.5.4 beschrieben, zwischen HUDs und HP unterschieden.

4.2.5.1 Immersion und Visualisierungen als Werkzeug der Nutzerintegration

Die MMI endet nicht bei der reinen Eingabe, sondern ist vielmehr ein Zusammenspiel aus Ein- und Ausgabe. Dies wurde bereits im vorherigen Kapitel im Rahmen der Kraftrückkopplung einiger Eingabegeräte sichtbar, jedoch ist der Mensch primär auf seine visuellen Eindrücke angewiesen. Wie schon in Kapitel 2.3.5 beschrieben, ist eine visuelle Aufbereitung (großer) Datenmengen sehr wichtig, um intuitiv und schnell als Mensch richtige Entscheidungen zu treffen. Je zugänglicher wichtige Informationen sind, umso fundierter sind die Entscheidungen. Die Unterteilung in die verschiedenen Visualisierungsebenen ist somit auch bei der Implementierung hilfreich und mit folgenden Aspekten sinnvoll:

1. Visualisierungshardware,
2. HUD,
3. HP und
4. Views.

4.2.5.2 Aufbereitung von Rohdaten

Durch die Integration in das Sensorframework ist es möglich, die schon implementierten Visualisierungen zu nutzen. Diese umfassen Bilder, Punktwolken, 3D-Geometrien sowie im Code verankerte Eigenschaften, die ausgegeben werden können.

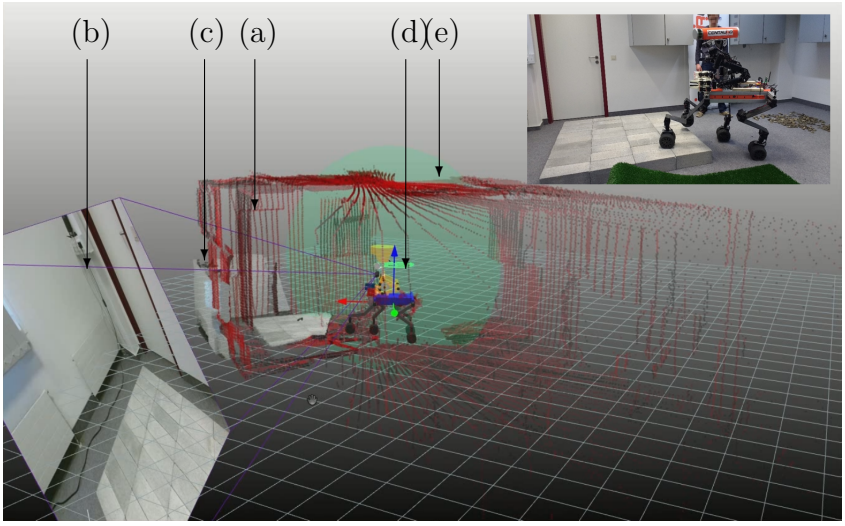


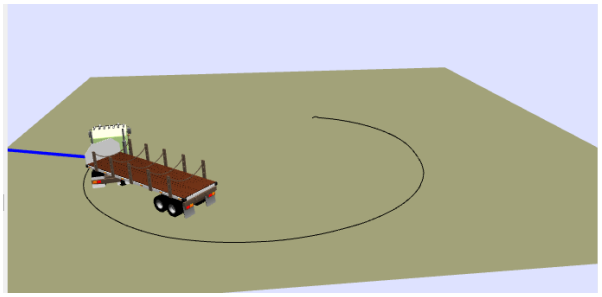
Abb. 4.18: *Live-Visualisierung diverser Rohdaten beim realen „Stepping“-Manöver: (a) Punktwolke des rotierenden Velodyne Scanners, (b) Frustrum Visu des Kinect v2-RGB-Bildes, (c) Kinect v2-kolorierte Punktwolke, (d) Trajektorienvisualisierung („Tracer“) Pfadvisualisierung, (e) Velodyne-Laserscanebenen-Visualisierung*

Wie man in Abb. 4.18 erkennt, wird der DT zusammen mit den verschiedenen Datenvisualisierungen gerendert. Im Rahmen dieser Arbeit wurden diese erweitert bzw. neue Visualisierungen hinzugefügt. Hierbei erkennt man (a) eine Punktwolke eines Velodyne-Laserscanners, die keine Farbinformationen enthält, (b) eine Punktwolke der Kinect v2 (in der diese als klassischer Sensor genutzt wird), (c) die Projektion des Kinect v2-Bildes

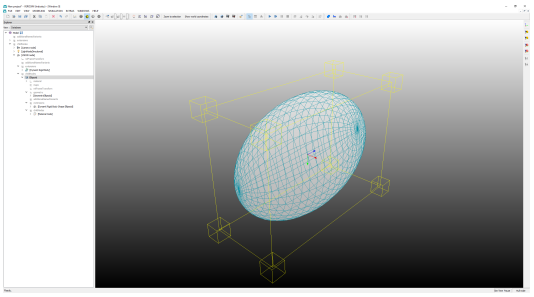
der RGB-Kamera in einer sog. „Frustrum Visu“ an die Stelle, in der das Bild aufgenommen wurde sowie (d) eine Trajektorienvisualisierung („Tracer“) und (e) die drehende Ebene des Laserscanners. Wichtig hierbei ist, dass die Posen (oder Frames) der einzelnen Objekte und Visualisierungen korrekt relativ zueinander bekannt sind, sodass z. B. Bild und Punktwolke der Kinect, aus dem richtigen Blickwinkel betrachtet, aufeinanderpassen.

4.2.5.2.1 Tracer und Laserscannerebene

Ein simples, aber effektives Beispiel ist die Trajektorien- oder *Tracer*-Visualisierung. Hierzu wurde eine `TracerExtension` entwickelt, die die Position einzelner Frames über die Zeit mitverfolgt und diese Spur in das Renderfenster zeichnet (vgl. Abb. 4.19a).



(a) *Tracer Extension: Visualisierung der Trajektorie des vorderen linken Rades einer Zugmaschine*



(b) *Ellipsoid: Geometrie und Starrkörper, parametriert durch die drei Hauptachsen (a,b,c)*

Abb. 4.19: *Visualisierung: Tracer und Laserscannerebene*

Des Weiteren wurde eine elliptische Geometrie genutzt, um die aktuelle Ebene des Laserscanners anzeigen zu können. Hierbei wurde eine neue Geometrie eines Ellipsoids (siehe Abb. 4.19b) entwickelt und der Simulation als Geometrie und Starrkörper hinzugefügt. Diese kann dann sowohl hier, als auch in Kap. 4.3.2.2.2 zur Visualisierung von Trägheitstensoren und in Kap. 4.3.3.1 auch als Starrkörpergeometrie für Reifen verwendet werden.

4.2.5.2.2 Erweiterung der Punktwolkenvisualisierung

Auf Basis der implementierten Punktwolkenvisualisierung wurde diese um einen neuen Visualisierungstyp erweitert.

Dieser lässt das Farbspektrum nach der Höhe der Punkte kodieren, wobei dieses Spektrum (min bis max) auch im Farbspektrum ausgewählt werden kann. Hiermit wird die oft sehr wichtige Höheninformation intuitiv und direkt zugänglich gemacht.

4.2.5.2.3 Inspektion der Nachrichtenübertragung

Im Rahmen der späteren ROS-Anbindung (siehe Kap. 4.4.1) wird auch eine Inspektionsmöglichkeit geschaffen, um sich die übertragenen Nachrichten direkt ansehen zu können.

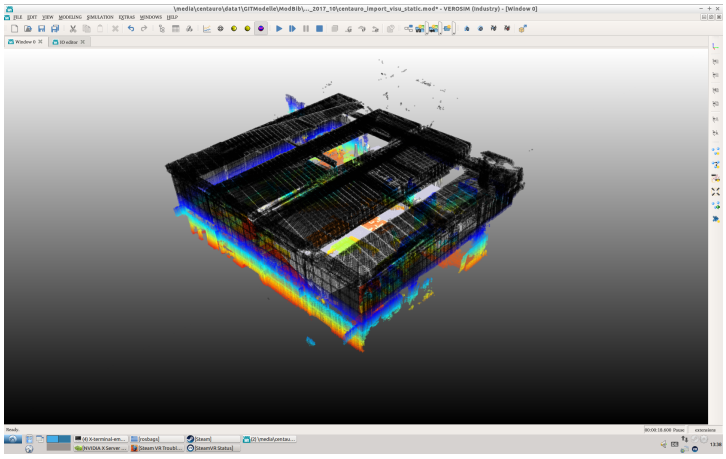


Abb. 4.20: *Visualisierung: höhencodierte Punktwolkenvisualisierung*

Ähnlich zu dem Debugging-Tool *rqt_topic* [cb] wird der Inhalt jedes Nachrichtentyps in Form eines Textstreams (korrekt formatiert) präsentiert (vgl. Abb. 4.21).

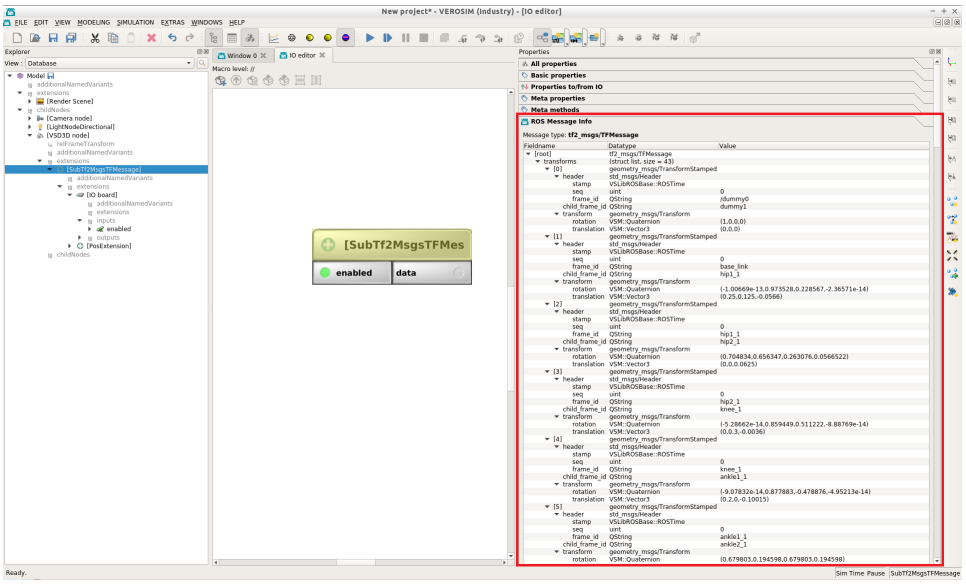


Abb. 4.21: *VSPluginROSGUI: Onlineinspektion von ROS-Nachrichten am Beispiel eines TF-Trees*

4.2.5.3 2D-Visualisierung in Form eines Head-Up-Displays (HUD)

Die Basis der hier beschriebenen Nutzerschnittstellen bilden die im Rahmen von [213] entwickelten GUI-Elemente. Hier wurde ein Videostream mit variabel anzuordnen GUI-Elementen überlagert, welche wichtige Informationen ins Sichtfeld des Operators von mobilen teleoperativ gesteuerten Robotern projiziert.

Diese Visiere oder HUDs werden im Sinne eines GUI-Frameworks erweitert und optimiert. Aufbauend auf einer gemeinsamen Basisklasse `BaseExtension` werden allen

Elementen folgende Eigenschaften zugeordnet:

- Position $p(x, y)$ [%]
- Größe $A(x, y)$ [pxls]
- Transparenz α [%]
- Renderebene R_F^i
- Renderfenster R_W^j

Neben den selbsterklärenden Eigenschaften bieten die Renderebene und -fenster die Möglichkeit, die Visualisierung auf einer oder mehreren Renderfenstern sowie in den verschiedenen Renderebenen (Vordergrund, Hintergrund etc.) anzuzeigen.

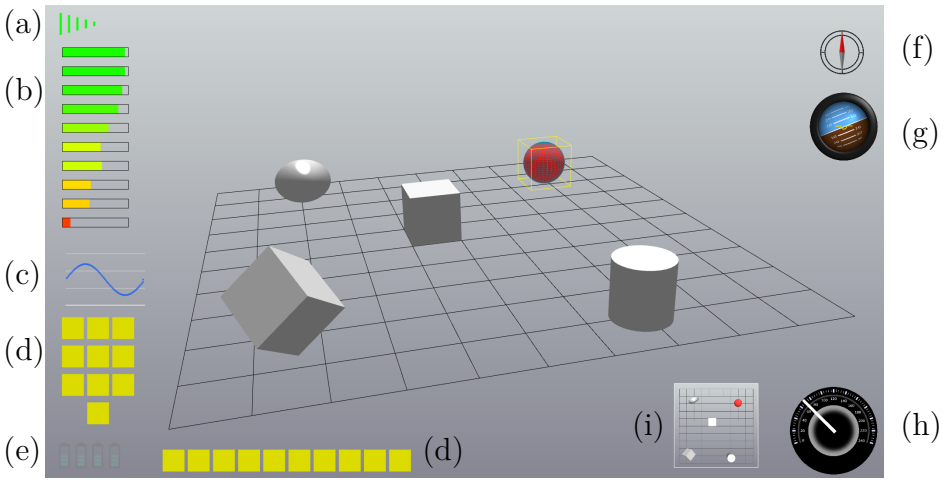


Abb. 4.22: *Flexibles HUD: 3D-Szene mit Head-Up-Display-Visualisierungen, (a) vertikale und (b) horizontale Balken, (c) Plots, (d) Texturen wie z. B. Nummernblock-Visualisierung, (e) Batteriefüllstand, (f) Kompass, (g) Gyroskop bzw. künstlicher Horizont, (h) Tachometer oder (i) Minimap*

Modularität steht auch hier im Vordergrund. Es sind Parametervisualisierung, die einem Zweck zugeordnet werden kann – wie z. B. die prozentuale Füllung eines Balkens. Jeder Nutzer kann im Nachhinein sein persönliches Cockpit zusammenstellen. Diese werden später, stereoskopisch gerendert, auch in acsVR-Brillen genutzt (Kap. 4.2.6). Wie in Abb. 4.22 zu sehen, sind neben klassischen Balkendiagrammen oder Plots auch Visualisierungen, bekannt aus anderen Domänen (wie dem Automotive-, Flugzeug- oder Spieleindustriebereich), implementiert, wie z. B. Tacho, künstlicher Horizont, Kompass oder Minimap. Kombiniert mit der `KeytogglerExtension` (vgl. Kap. 4.2.2.1) kann man auch aktuelle Tastatureingaben visuell hervorheben.

4.2.5.4 Immersive 3D-Visualisierung in Form von Holographischer Projektion

Die Holographische Projektion (HP) beinhaltet alle Visualisierungsformen, die den Daten auch parallel einen Ort bzw. eine Pose zuordnen. Allgemein beinhaltet diese HP:

1. Modellierung,

2. Interaktion und
3. Datenprojektion.

Ein Beispiel hierfür ist die bereits gezeigte *Tracer*-Visualisierung, welche die 3D-Trajektorie in eine 3D-Umgebung zeichnet, wie z. B. in Abb. 4.19a und Abb. 4.18.

4.2.5.4.1 Interaktive Marker („Gizmo“)

Neben neuartigen Eingabegeräten können auch visuelle Hilfestellungen für die Modellierung hilfreich sein. Ein neues, sehr sinnvolles Feature stellen interaktive Marker wie das *„Gizmo“* (siehe Abb. 4.23) dar.

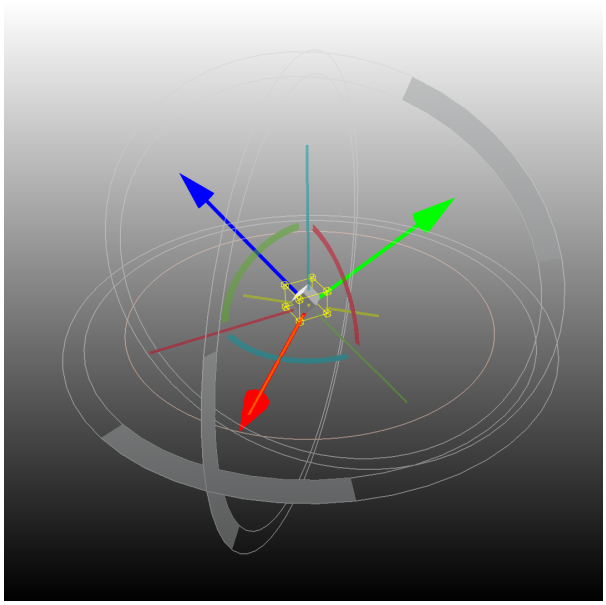


Abb. 4.23: *Interaktiver Marker (Gizmo): Koordinatensystem, Roll-Pitch-Yaw-Rotationssystem, $\psi_x/\psi_y/\psi_z$-Rotationssystem*

Die VSD baut automatisch einen Transformationsbaum auf, wobei das Koordinatensystem des Kindsknotens immer relativ zum Elternknoten vorliegt, aber auch in Weltkoordinaten umgerechnet werden kann. Wo vorher noch ein Transformationswidget für x, y, z (T_{XYZ}) sowie roll, pitch, yaw (T_{RPY}) genutzt werden musste, können nun mit dem Gizmo Position und Orientierung beliebig per Drag and Drop verändert werden.

$$T_{XYZ} = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}, T_{RPY} = \begin{bmatrix} & \mathbf{R}_{RPY} & & 0 \\ & & & 0 \\ & & & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, T_{DH} = (\theta, d, a, \alpha)$$

$$\text{mit } \mathbf{R}_{RPY} = Rot(roll, z)Rot(pitch, y)Rot(yaw, x)$$

Der Vollständigkeit halber ist es natürlich auch möglich, Transformationen in Form von DH-Parametern (T_{DH}) anzugeben (um relative Lagen zwischen Achsen in kinematischen Ketten zu definieren).

Das Objektkoordinatensystem wird beim Gizmo opak angezeigt. Die x-Achse (rot) definiert dann die RPY-Ebenen, welche in Grau als äußere Ringe des Gizmos dargestellt werden. Die inneren transparenten Achsen sind für die Translation und Rotation um XYZ im Weltkoordinatensystem.

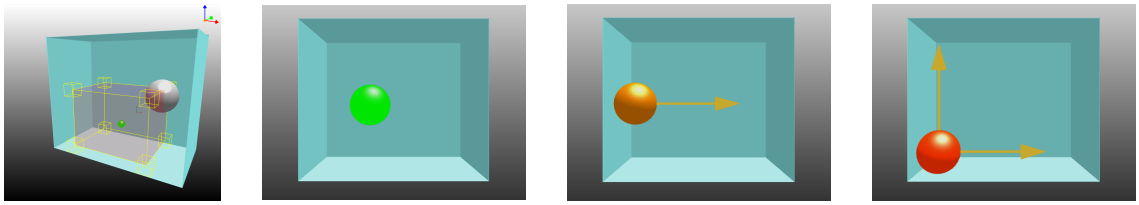
Dieser interaktive Marker erleichtert nicht nur die Modellierarbeit. In Kombination mit einer inversen Roboterkinematik kann so z. B. der TCP eines (kinematisch ausmodellierten) Industrieroboters per Drag and Drop verschoben werden. Somit führt diese Visualisierungsmetapher einer Objektpose zu einem intuitiveren Zugang zu 3D-Objekten und Gesamtsystemen.

4.2.5.4.2 Arbeitsraumvisualisierung

Eine weitere wichtige Visualisierung ist die des *Arbeitsraumes*. Sowohl der Arbeitsraum von Systemen als auch der Arbeitsraum von Eingabegeräten kann mit Hilfe von parametrierten Geometrien visualisiert werden. Exemplarisch für das PHANToM Device ist dies in Abb. 4.24a zu sehen (welches auf Abb. 4.14 basiert).

4.2.5.4.3 Kontaktkraftvisualisierung

Eine weitere Interaktionsmetapher ist die *Kontaktkraftvisualisierung*. Mit Hilfe der Starrkörperdynamik kann sowohl der Kollisionszeitpunkt als auch die Kollisionskraft visualisiert werden. Bei haptischen Eingabegeräten hat man ein solches Feedback im Sinne der Kraft oder durch Vibration direkt, jedoch kann man auch beliebige andere Eingabegeräte durch diese Visualisierung mit Feedback versorgen. Der Kollisionszeitpunkt wird durch eine Farbveränderung, die Kollisionskraft durch einen Pfeil mit Richtung und Amplitude proportional zur wirkenden Kraft gezeichnet (Abb. 4.24b, c, d). Die Kraftamplitude kann parallel mit der Farbwahl gekoppelt werden.



(a) *(b)* *(c)* *(d)*

Abb. 4.24: *Arbeitsraum und Kontaktkraftvisualisierungen (Vektoren und Farbänderungen)*

4.2.5.4.4 Billboards

Als letzter und wichtigster Bestandteil der HP sind *Billboards* (zweidimensionale Flächen/„schwarze Bretter“, die sich immer dem Betrachter zuwenden) zu nennen. In [214] wurde für den Anwendungszweck autonomer Logistik und Lagersysteme ein Billboard als Statusmonitor für die einzelnen mobilen Transportsysteme genutzt. Darauf aufbauend wurde das bestehende Billboard-Framework erweitert, um basierend auf Name und Pose Objekte einzufügen. Des Weiteren können diese um Textur-basierte Icons erweitert

werden, die vordefinierte Statusinformationen enthalten. Diese Billboards haben die Eigenschaft, sich automatisch in die aktuelle Blickrichtung auszurichten und im gegebenen Abstand ein- bzw. auszublenden. Mit Hilfe dieser Technologie ist es möglich, „Notizzettel“ in der 3D-Szene zu platzieren, die zusätzliche Informationen über das referenzierte Objekt bereithalten. Ein Beispiel dafür ist in Abb. 4.25 zu sehen.

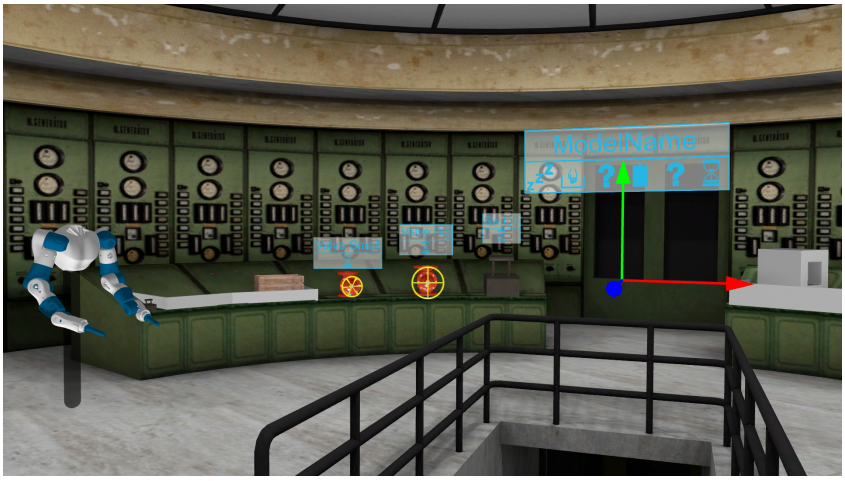


Abb. 4.25: *Billboards: Kombination aus Pose, Name, Texturen*

4.2.5.4.5 Visualisierung von Höhenkarten und Kostenfunktionen

In den Bereich der HP fallen auch 3D-Karten. So wird aus Sensordaten ein Gitter generiert, welches neben x/y-Koordinaten auch Höhenwerte (z) enthält. Auf Basis dieser wird eine neue dreidimensionale Geometrie erzeugt (`RBShapeHeightField`), welche auch in das Starrkörperframework eingebunden werden kann. Diese nutzt als standardisierten Eingabeparameter eine `cv::Mat` . Des Weiteren kann als Projektion auf diese Höhenkarte eine dreidimensionale Kostenfunktion ($c = f(x, y)$) gelegt werden, um beispielsweise eine Lokomotionsplanung durchführen zu können, wie in Abb. 4.26 gezeigt.

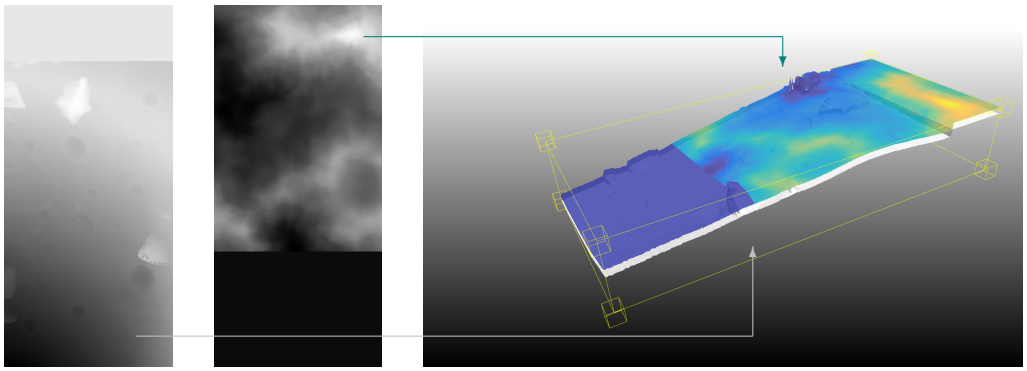


Abb. 4.26: *Projektion einer Kostenfunktion auf eine Höhenkarte*

4.2.5.5 Texturierte Polygonnetze

Als Kombination von flachen Texturen, wie sie beim HUD genutzt werden, und der Projektion dieser in die Szene wie bei der HP, sind texturierte Polygonnetze (siehe z. B. [215]) eine Möglichkeit, Kamerabilder mit hochauflösenden Punktwolken zu einem System verschmelzen zu lassen. Hierbei handelt es sich nicht um eine planare 2D-Textur, sondern um ein dreidimensionales Polygonnetz (auch oft als „Mesh“ bezeichnet) mit Textur. Um das texturierte Polygonnetz zu generieren, werden folgende Daten benötigt, die über das I/O Board zur Verfügung gestellt werden:

- Punkte, Weite und Höhe der Punktwolke,
- Kamerabild und
- Projektionsmatrix der Kamera.

Der erste Schritt, welcher auf die Daten angewendet wird, ist die Sortierung der Punkte (siehe Abb. 4.28a). Bei der Kinect v2 z. B. übermittelt der Tiefensensor die Koordinaten einzelner Raumpunkte reihenweise und indiziert sie dementsprechend. Punkte, die nebeneinander im Array liegen, sind daher auch räumlich höchstwahrscheinlich nebeneinander (Zeilenwechsel ausgelassen) angeordnet. Um ein Polygonnetz zu rendern, müssen die einzelnen Punkte zu Dreiecken verbunden sein. Dazu werden drei Punkte, die ein Dreieck bilden, nebeneinander in ein Array gelegt (vgl. Abb. 4.27).

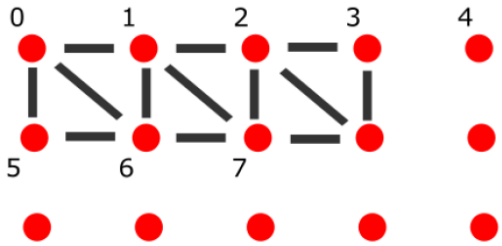
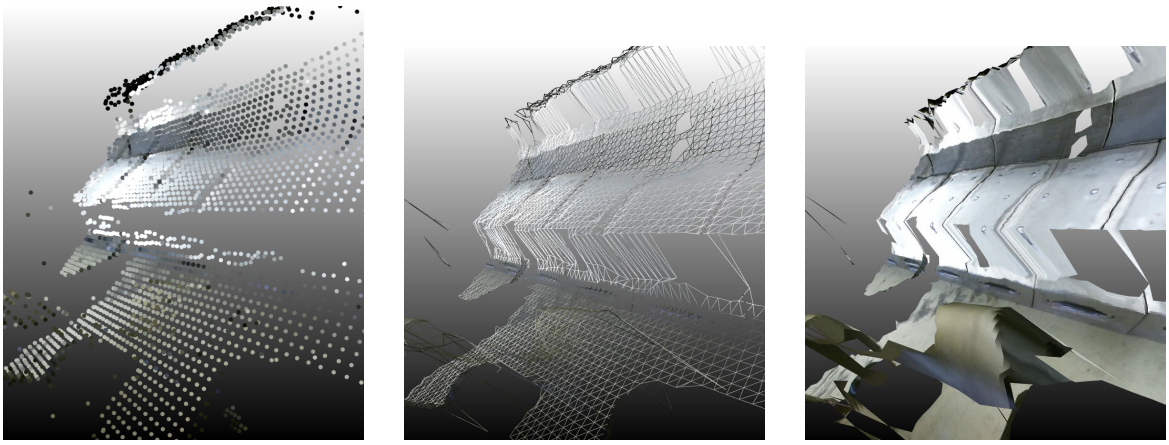


Abb. 4.27: *Abrasterung einer Punktwolke: Zur Dreiecksgenerierung wird hier das Array gefüllt mit: 0,5,6, 0,6,1, 1,6,7 usw.*

Beim Zusammensetzen der Dreiecke werden Punkte dupliziert, wodurch sich die Datenmenge vervielfachen kann. Bei dem verwendeten Verfahren wird jedes Dreieck aussortiert, wenn einer der Punkte undefiniert ist. Dies kann zu Lücken führen, die meistens jedoch nicht störend sind. Das nun sortierte Array kann bereits genutzt werden, um ein Drahtgittermodell zu erzeugen (siehe Abb. 4.28b).

Im letzten Schritt muss noch die Textur auf das Mesh gelegt werden (vgl. Abb. 4.28c). Es stellt sich dabei die Frage, welcher Raumpunkt mit welchem Pixel verbunden ist bzw. wo sich ein gegebener Raumpunkt im Kamerabild wiederfinden würde. Eine homogene Transformation vom Raumkoordinatensystem in das Kamerakoordinatensystem liefert die übergebene Projektionsmatrix. Werden die erhaltenen Raumpunkte mit dieser multipliziert, so erhält man deren Darstellung im homogenen Kamerakoordinatensystem:



(a) Punktwolke

(b) Drahtgitterrendering: Einzelne Punkte lassen sich gut erkennen, wie auch Lücken im Modell.

(c) Texturiertes Polygonnetz: Fusion der Kinect v2-Punktwolke mit Bilddaten

Abb. 4.28: *Texturiertes Polygonnetz*

$$\begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix} \rightarrow \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix}, \qquad T_{cam} \cdot \begin{pmatrix} x_p \\ y_p \\ z_p \\ 1 \end{pmatrix} = \begin{pmatrix} x'_c \\ y'_c \\ z'_c \\ w'_c \end{pmatrix} \tag{4.1}$$

Anschließend werden die Raumpunkte durch Division der zusätzlichen Dimension in das zweidimensionale kartesische Kamerakoordinatensystem transformiert:

$$\begin{pmatrix} x'_c \\ y'_c \\ z'_c \\ w'_c \end{pmatrix} \rightarrow \begin{pmatrix} x'_c/w'_c \\ y'_c/w'_c \\ z'_c/w'_c \end{pmatrix} = \begin{pmatrix} x_c \\ y_c \\ z_c \end{pmatrix} \tag{4.2}$$

Liegt der erhaltene Punkt innerhalb der Größe des Bildes (Kinect liefert Full HD, Breite: 1920, Höhe 1080) so befindet er sich auch innerhalb des Bildes. Da das Bild als 2048×2048 OpenGL-Textur vorliegt, müssen noch die UV-Koordinaten bestimmt werden. UV-Koordinaten sind eine (willkürlich gewählte) 2D-Repräsentation $p(u, v)$ eines 3D-Modells $p(x, y, z)$. Dazu wird der erhaltene Punkt in beiden Dimensionen durch 2048 dividiert. Die erhaltenen UV-Koordinaten werden gespeichert und letztendlich in einem Array zusammen mit den Raumpunkten dem Shader übergeben. Der Shader ist so minimal wie möglich gehalten und vollzieht mangels vorhandener Normalen keine Lichtbrechung. Hierbei ist zu beachten, dass zusätzlich Schatten aus der Simulation zu sehen sein können. Bei realen Bildern sind die Schatten oft schon in der Bilddatei enthalten und sollten nicht zusätzlich durch die Simulation hinzugefügt werden.

4.2.6 Immersion und Visualisierungen im Gesamtsystem

Um die Immersion des Nutzers bei seiner Interaktion zu verstärken, werden sowohl Hardwareintegration als auch Softwareimplementierungen durchgeführt. Basierend auf Kap. 2.3.5 werden diese im Folgenden vorgestellt.

4.2.6.1 Stereoskopisches Rendering

Der Übergang von monoskopischem Rendering auf stereoskopisches Rendering stellt implementierungstechnisch eine Herausforderung dar. Möchte man eine Ansicht vom Bildschirm auf ein HMD wechseln, muss die Szene aus unterschiedlichen Perspektiven gerendert werden, sodass zwei Views und zwei Projektionsmatrizen nötig sind. Mit Hilfe der OpenVR-API, die von Valve entwickelt wurde und mit der Laufzeitumgebung SteamVR funktioniert [cc], können HMDs wie die Oculus Rift oder die HTC Vive an Drittanbieterapplikationen angebunden werden. Zur Integration in ein plattformunabhängiges System müssen hierbei natürlich auf Windows-Applikationen optimierte Systeme auch unter Linux performant funktionieren. Neben der generellen Anbindung des OpenVR-SDKs müssen auch einige nutzungsspezifische Erweiterungen hinzugefügt werden.

4.2.6.1.1 Nullpose

Ein zentrales Element hierbei ist die *Nullpose* des HMDs. Basierend auf der Eingangspose des HMDs muss dieses zunächst ins Koordinatensystem der Simulationsumgebung transformiert werden. Des Weiteren wird ein Offset in Position und Orientierung zugelassen, da sich der Nutzer ansonsten physisch umpositionieren muss.

4.2.6.1.2 Fortbewegung

Ein weiteres Element ist die Art der Fortbewegung. Hierbei haben sich zwei Optionen herauskristallisiert:

1. Physische Bewegung, skaliert auf die Anwendung, sowie
2. Teleportation.

Durch *physische Bewegung* ist es möglich, ein bestimmtes Areal ungehindert abzuschreiten. Limitiert wird dies durch sog. „Chaperone-Wände“, die von der HTC Vive bei der Einrichtung aus Sicherheitsgründen aufgespannt werden. Außerdem wird die Skalierung einzelner Aktionen – der Nutzer geht physisch 1 m nach vorne – auf das Anwendungsszenario angepasst bzw. skaliert.

Zur *Teleportation* wird einfach der 3D-Knoten der Kamera verschoben. Durch die Modularität der Simulationsumgebung kann dies wiederum auf verschiedene Arten getriggert und durchgeführt werden. Ein sinnvolles Vorgehen ist die Selektion eines Zielknotens, beispielsweise durch die VR-Controller und anschließendes „magnetisches“ Anziehen an eben diesen Knoten. Daraufhin verschiebt sich natürlich auch der verfügbare Arbeitsraum mit an diese Stelle.

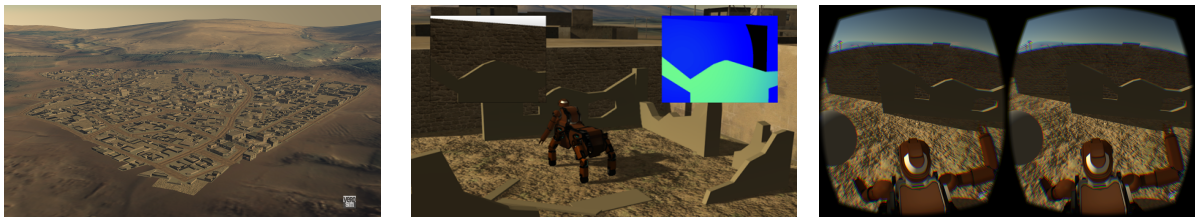
4.2.6.1.3 Visualisierungen stereoskopisch rendern

Im Bereich des Renderings müssen vor allem die in Kap. 4.2.5 vorgestellten HUD-Visualisierungen auch stereoskopisch gerendert werden. Damit ein stereoskopischer Eindruck entsteht, wird vor jedem Rendervorgang erfasst, auf welchen Bildschirm/Auge gerendert wird. Je nach Auge verschiebt sich das Projektionszentrum der Kamera um den Augenabstand ($d_{Auge} \approx 8$ cm) nach rechts oder verharrt im Ursprung. Die entsprechende Projektionsmatrix erhält man vom Kameraknoten. Diese muss sogar pro Auge angegeben werden, da manche HMDs asymmetrische Frustums verwenden.

Des Weiteren wird eine stereoskopische Skybox hinzugefügt, die das Raumgefühl und die Immersion des Nutzers in der Szene unterstützen soll.

4.2.6.2 Kameraperspektiven

Als Seiteneffekt der zuvor beschriebenen Entwicklungen können beliebige Blickwinkel im VTB eingenommen werden (vgl. Kap. 2.3.5.5).



(a) *Free-flying-Kamera* ***(b)*** *Third-Person-Perspektive* ***(c)*** *Ego-Perspektive*

Abb. 4.29: *Verschiedene Kameraperspektiven zur (a) Exploration, (b) Manöverplanung, (c) Manipulation*

So ist es möglich (a) sich mit einer Free-flying-Kamera einen Überblick über die Szenerie zu verschaffen, bevor man (b) in der Third-Person-Perspektive Lokomotionsaufgaben plant und durchführt und schließlich (c) feinfühlige Manipulationsaufgaben aus der Ego-Perspektive angeht. Durch Ausnutzen der Magnete und diverser Toggle-Mechanismen (Kap. 4.2.2.1) kann man unverzögert zwischen diesen hin- und herschalten und sich mit dem Gizmo (Kap. 4.2.5.4) neue Frames erschaffen, die als neue Nullpose genutzt werden können. Natürlich kann man sich auch durch physisches Ablaufen der gesamten Szene einen guten Überblick verschaffen.

Hiermit ist es möglich, eine sogenannte „Bubble of Cognition“, also eine Unterscheidung von relevanten, hoch aufgelösten Daten in naher Umgebung $r < R$, sowie niedrig aufgelöste oder gar ausgeblendete Daten für $r > R$ zu definieren. Dies ist insbesondere für die Latenz und Simulationsgeschwindigkeit ein Performance-steigerndes Mittel.

4.3 Einbettung des Digitalen Zwillings ins Virtuelle Testbed

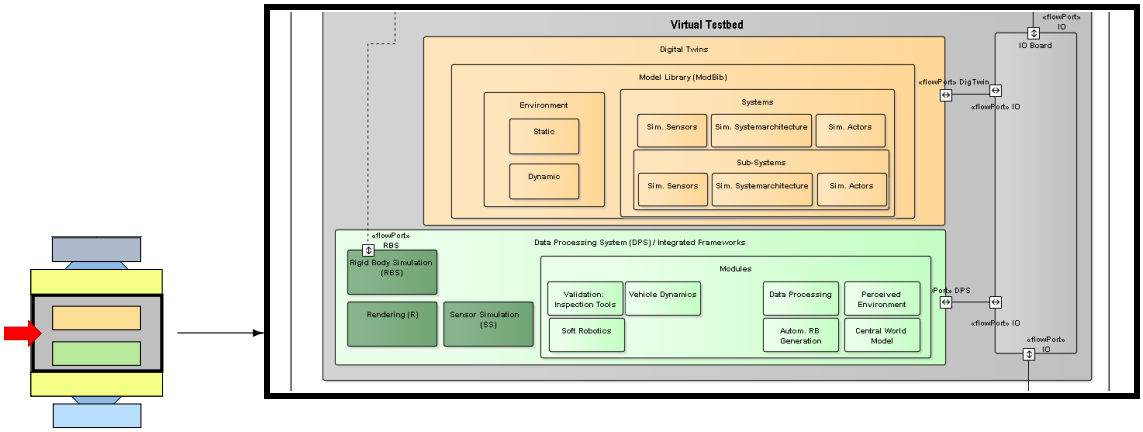


Abb. 4.30: *Konzept: Einbettung des Digitalen Zwillings (vgl. Abb. 3.4)*

Die Einbettung des DTs ins VTB umfasst, auf Basis des Modells (vgl. Kap. 2.5.2.3 und Gl. 2.47) ein *„experimentierbares“ Modell* zu generieren, welches im VTB mit anderen DTs interagieren kann und über die Grenzen des VTBs ausgetauscht werden kann. Zusätzlich gehört die direkte Verbindung mit dem physischen Asset (vgl. Kap. 4.4) und die intuitive Anbindung zum Nutzer (vgl. Kap. 4.2) dazu. In unterschiedlichen Ausprägungen bzw. Repräsentationen – visuell, funktionell, strukturell etc. – kann der DT dann genutzt werden.

Zunächst wird in Kap. 4.3.1 das Erstellen und der Zugriff auf Digitale Zwillinge im VTB gezeigt. Hierbei geht es insbesondere um das dynamische Generieren solcher DTs zur Laufzeit sowie der standardisierte Austausch dieser.

Für einen sinnvollen und effizienten Umgang mit Digitalen Zwillingen müssen diese aber auch *validierbar* sein (Kap. 4.3.2). Dies gilt insbesondere für einzelne und/oder kombinierte physikalische Parameter (Kap. 4.3.2.2). Zusätzlich kann man aber auch den DT selbst nutzen, um das zu untersuchende Gesamtsystem zu validieren und z. B. in Form diverser Benchmarking-Szenarien zu evaluieren (Kap. 4.3.1.3). Gerade bei diesen Gesamtszenarien erkennt man das Netzwerk aus interagierenden Digitalen Zwillingen besonders gut.

Während die Starrkörpersimulation, das Rendering Framework und eine Sensorsimulation essentielle Komponenten des VTBs darstellen, erfordern manche Digitalen Zwillinge *spezialisierte Frameworks* bzw. Funktionalitäten, die die umgebende Simulation aufweisen muss. Daher werden in Kap. 4.3.3 spezialisierte Frameworks der Fahrphysik und der Soft-Robotik aufgezeigt.

4.3.1 Automatisierte Generierung Digitaler Zwillinge

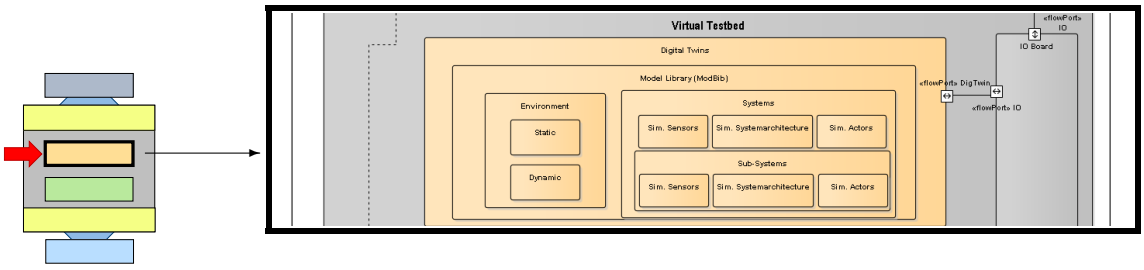


Abb. 4.31: *Konzept: Integrierte Funktionalitäten – Digitale Zwillinge (vgl. auch Abb. 3.4)*

Um einen realitätsgetreuen Digitalen Zwilling als Vorlage für das reale System entwickeln zu können, ist eine einheitliche Blaupause nötig. Eine solche Blaupause ist im Falle der Systementwicklung und Simulationstechnologie oftmals eine Datei, häufig im Extensible-Markup-Language-(XML-)Format, die das Robotermodell beschreibt. Im Folgenden wird URDF, der ROS-Dateistandard vor allem für robotische Systeme, sowie *FMI/FMU*, der Dateiaustausch und Co-Simulationsstandard im Automotive-Bereich, betrachtet werden. Bei allen diesen Formaten steht die *Standardisierung* zum Austausch im Vordergrund. Notwendig dafür ist ein möglichst *automatisierter Importprozess* der jeweiligen Dateitypen. Im Folgenden wird dies aus der Sicht der Simulationsumgebung VEROSIM beschrieben, welche den EGEMOL6-Standard für allgemeine Systeme, Umweltmodelle sowie Daten nutzt.

Das Generieren von DTs aus Geometriedaten erfordert oft viele manuelle Arbeitsschritte. Um diesen Prozess zu automatisieren, gibt es verschiedene Herangehensweisen, welche sich auch im finalen Nutzen wiederspiegeln. In diesem Kapitel werden daher statische und dynamische Verfahren beleuchtet, wie in Abb. 4.32 dargestellt.

Auf der einen Seite stellt der initiale Import *statischer* Daten immer ein Problem dar, da es kein standardisiertes Austauschformat gibt. Dies Daten müssen daraufhin noch funktional mit kinematischen oder dynamischen Eigenschaften ausgestattet werden. In Kap. 4.3.1.1 und 4.3.1.2 wird daher der (teil-)automatisierte Import von CAD-Daten beschrieben, welcher insbesondere für statische Fälle sinnvoll ist. Aus all diesen statisch importierten DTs lässt sich daraufhin ein Repository anlegen, auf das man jederzeit Zugriff hat (vgl. Kap. 4.3.1.3).

Neben statisch importierten DTs soll auch die Möglichkeit der dynamischen „online“-Objektgenerierung gegeben sein. Diese beinhaltet den Übergang von Punktwolken zu einer Polygonsuppe, Geometrien und schlussendlich *dynamischen* DTs (siehe Kap. 4.3.1.4 und 4.3.1.5).

4.3.1.1 (Teil-)automatisierter CAD-Datenimport

Starrkörpersimulationen werden immer häufiger auf Grundlage von CAD-Daten durchgeführt. Hierbei wird zunächst die CAD-Software genutzt, um Teile, Baugruppen oder gan-

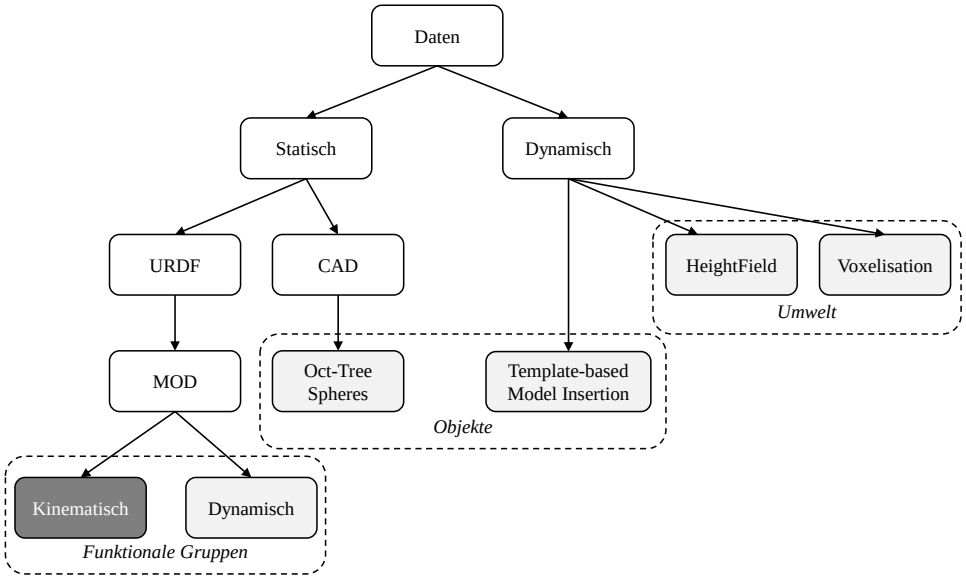
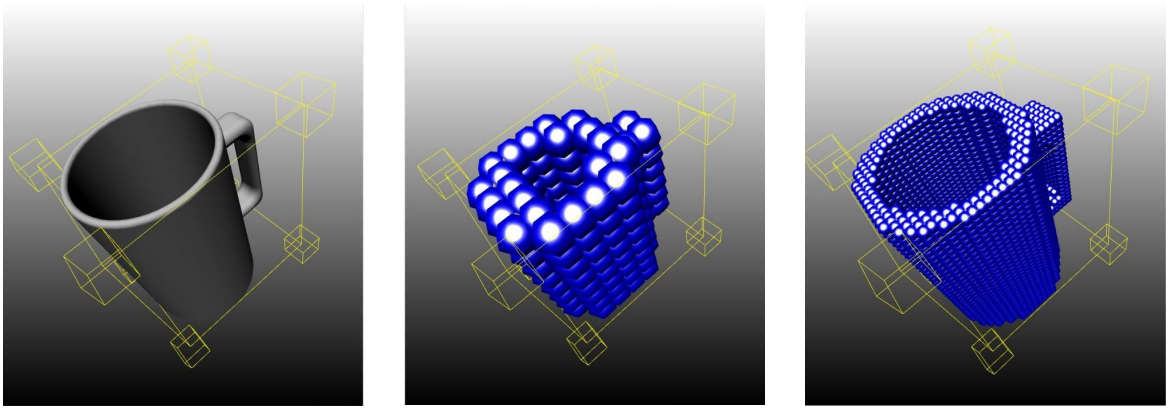


Abb. 4.32: *Automatische DT-Generierung: statischer und dynamischer Objektimport*

ze Systeme und anschließend die Starrkörpersimulation zu designen, um diese zu testen, einzusetzen oder in einen größeren Kontext einzuordnen. Zur automatischen Überführung der CAD-Daten in das Simulationsprogramm muss eine Parametrisierung vorgenommen werden, die oft kompliziert und zeitintensiv ist. Auf Basis der CAD-Daten werden von den einzelnen Bauteilen die Anzahl stetiger Flächen erkannt und davon ausgehend den verschiedenen *primitiven Geometrien* zugeordnet. Nach dieser Geometrieerkennung wird die CAD-Geometrie um eine des assoziierten geometrischen Primitiven erweitert. Anhand der Bounding Box der CAD-Geometrie wächst die des Primitiven in diese hinein. Nach dem Approximationsschritt wird die Massenverteilung entweder gleichverteilt angenommen oder nach dem Volumen der Subkörper skaliert. Die importierten CAD-Daten, übermittelt im STEP-Datenformat [216], beinhalten ausschließlich geometrische Informationen und sind in der Baugruppe untereinander nicht durch Gelenke verbunden. Diese dynamischen Gelenke werden daher nur semi-automatisch erstellt und müssen per Hand nachjustiert werden.

Alternativ zu diesem Verfahren gibt es auch noch das Verfahren der *Oct-Tree Spheres*, in dem Geometrien mit vielen Kugeln angenähert werden. Bei den Oktalbäumen wird ein hierarchisches und rekursives Oktalbaumschema verwendet. Anfangs wird der Körper durch einen ausreichend großen Würfel beschrieben. Mittels rekursiver Schritte wird der Würfel in immer kleinere Würfeln aufgeteilt. In jedem rekursiven Schritt wird der Würfel in Teilwürfel halber Kantenlänge unterteilt, bis die Würfel entweder ganz innerhalb oder außerhalb des Körpers liegen bzw. bis die minimale Kantenlänge erreicht ist (siehe Abb. 4.33).

Auf Basis dieses dynamischen Modells kann nun z. B. auch automatisch ein kinematisches Robotermodell erzeugt werden. Im Gegensatz zum dynamischen Modell, in dem die einzelnen Teile unabhängig von den anderen Bauteilen modelliert und am Ende durch Zwangsbedingungen zusammengehalten werden, benötigt ein kinematischer Roboter ei-



(a) CAD Geometrie *(b) Oct-Tree Tiefe 3* *(c) Oct-Tree Tiefe 5*

Abb. 4.33: *CAD-Import eines Kaffeebechers mit Oct-Tree-Sphere-Dynamisierung*

ne „offene Kette“ als grundlegende Struktur, also jeder Körper darf maximal ein Gelenk und somit nur einen Nachfolger aufweisen. Diese Voraussetzung stimmt für nahezu jeden Industrie-Arm-Roboter. Ausgehend von den Gelenken wird so eine Kette aufgebaut, die in der Parent-Child-Beziehung den kinematischen Roboter aufbaut.

4.3.1.2 (Teil-)automatisierter URDF-Datenimport

Mit Hilfe des in Kap. 4.4 beschriebenen URDF-Importers lassen sich URDF-Modelle und die zugehörigen Geometrien importieren, um das sogenannte *VisuModell* zu erhalten. Hierauf aufbauend kann dann zum einen ein dynamisches Modell (*DynModell*) und zum anderen ein kinematisches Modell (*KinModell*) automatisch generiert werden. Beim dynamischen Modell wird der hierarchische Baum „geflattened“ und mit Starrkörperprimitiven oder vereinfachten Meshgeometrien zur Kollisionsbehandlung ausgestattet. Beim kinematischen Modell wird die hierarchische Struktur erhalten und durch DH-Parameter ergänzt.

Die Modellierung im ROS-Netzwerk basiert auf dem sog. URDF-Format [cd][ce]. Dieses XML-basierte Format repräsentiert immer ein komplettes Robotermodell (jedoch nicht die Szene, in die es platziert wird). Hauptbestandteile eines Roboters sind sog. *links*, verbunden durch sog. *joints*. Basis der URDF-Definition ist der Joint. Jeder Joint muss genau einen parent-Link und einen child-Link haben. Der Joint (origin) beschreibt dabei die Frametransformation vom parent- zum child-Link. Zusätzliche Spezifikationen wie z. B. eine definierte Drehachse können optional in jedem Joint angegeben werden. Ein Link besteht zunächst nur aus seinem Namen. Zusätzlich kann dieser ein visuelles Element, ein Kollisionselement oder ein Trägheitselement erhalten. Praktisch kann somit ein Link eine komplexe visuelle „Hülle“ haben, ein aus Standardgeometrien zusammengesetztes Kollisionselement, und dieses kann noch genauer durch spezifische Trägheitseigenschaften parametriert werden.

Für das ROS-basierte Robotersystem müssen reale Roboter, URDF-Modell und 3D-Simulation immer konsistent sein (vgl. Abb. 4.59). Deshalb wird ein URDF-Importer in VEROSIM implementiert. Der hier implementierte URDF-Importer automatisiert die

Modellierung von Starrkörpergeometrien.

Das URDF-Modell (vgl. Abb. 4.35a) spannt einen Baum mit voneinander abhängigen Knoten auf. Darauf aufbauend lässt sich zum einen ein rein visuelles, zum anderen ein *dynamisches* oder auch ein *kinematisches* (Roboter-)Modell generieren, welche strukturell in Abb. 4.34 und visuell in Abb. 4.35 dargestellt sind.

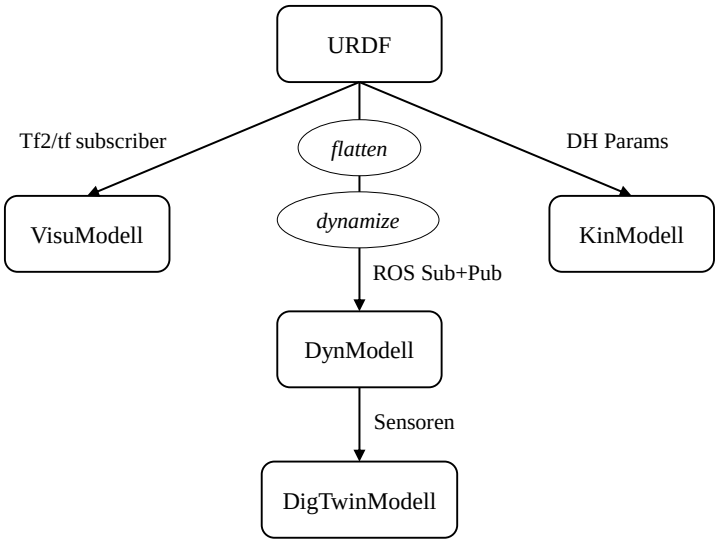


Abb. 4.34: *URDF-Import: Modelltypen Digitaler Zwillinge*

4.3.1.2.1 DT zur Visualisierung (VisuModel)

Die Struktur des Importers ist in Algorithmus 1 (S. 138) beschrieben. Hier werden zunächst Hashes aller Elemente angelegt, um alle Links (per ID und Name) sowie alle Joints (per ID, Name, ParentLink und ChildLink) direkt finden zu können. Daraufhin werden rekursiv, ausgehend vom Modellknoten, die Links und die damit verbundenen Joints durchgegangen. Hierbei werden für die Links Visuals, Collision und Inertial tags im XML umgewandelt in URDFLinks sowie VEROSIM-Geometrien wie Box, Zylinder usw. Bei den Joints werden analoge URDFJoints angelegt. Insgesamt ergibt sich so ein *hierarchischer VSD-Baum*, der alle URDF-Links und URDF-Joints sowie die geometrischen Informationen der Links enthält. Hieraus entsteht ein DT, der mit Hilfe eines *tf2_msgs/tf* Subscribers sofort zur Visualisierung des aktuellen Roboterstatus genutzt werden kann und deshalb im Folgenden als *VisuModel* bezeichnet wird.

4.3.1.2.2 Kinematischer DT (KinModel)

Aus dem URDF-Modell lassen sich neben kartesischen Transformationen zusätzlich die Denavit-Hardenberg-Parameter (vgl. auch Kap. 2.5.1.8) auslesen und als `DHParamTransform` (also Frametransformationen) hinzufügen und parametrieren. Diese beinhalten sogenannte `KinematicMapping` Extensions, welche wiederum in einer zentralen

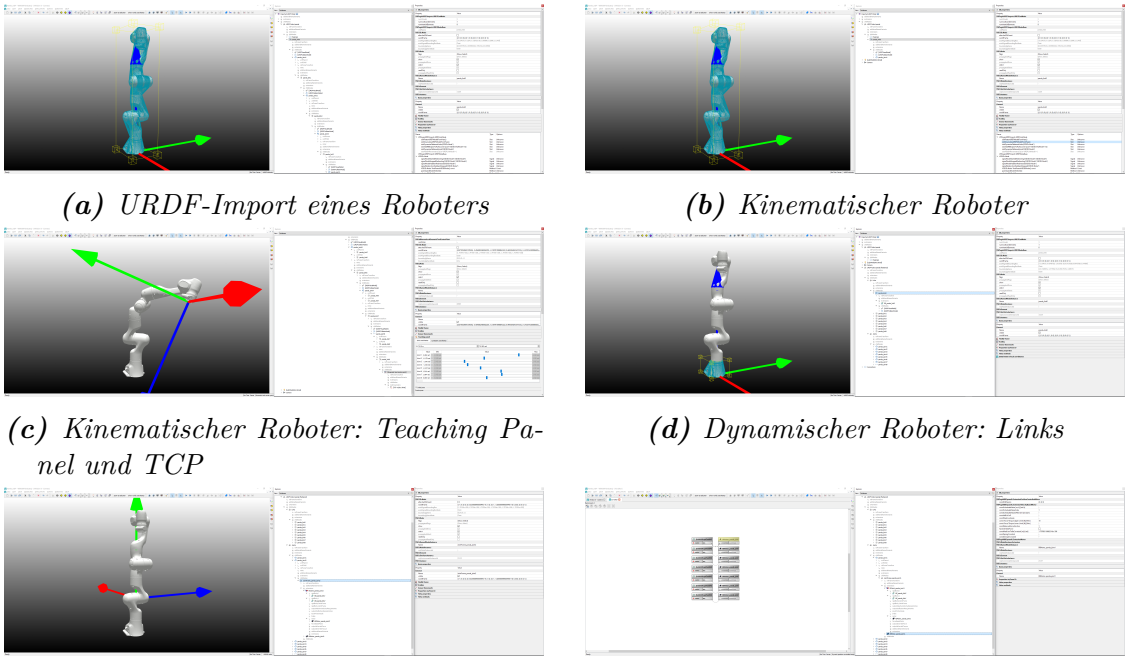
Algorithm 1 VisuModel: URDF-Baumgenerierung und (Geometrie-)Datenimport

```
1:  function readModel
2:      ↷ buildElementLookups(URDFRootElement)
3:      ↷ processLinkNode(URDFRootLink)

4:      procedure buildElementLookups(rootQDomElement)
5:          for all childQDomElements ∈ rootQDomElement do
6:              ⟲ AllLinks                                  ▷ : Liste
7:              ⟲ LinksByName                               ▷ : Hashes
8:              ⟲ AllJoints
9:              ⟲ JointsByName
10:             ⟲ JointsByParentLink
11:             ⟲ JointsByChildLink
12:         end for
13:     end procedure

14:     procedure processLinkNode(LinkNode)
15:         create new VSD instance ⇒ URDFLink
16:         getVisual() → handleVisual() → box, cylinder, ..., mesh
17:         getCollision() → handleVisual()
18:         getInertial() → handleInertial()
19:         ↷ processJointNode(JointsByParentLink)
20:     end procedure

21:     procedure processJointNode(parentJointNode)
22:         get all Joints pointing to this link
23:         create new VSD instance ⇒ URDFJoint
24:         ↶ processLinkNode(childLinkNode)
25:     end procedure

26:     return Visu Modell: Hierarchischer VSD-Baum mit allen „URDF Links“ und
        „URDF Joints“
27: end function
```

`KinematicMappingRoot` referenziert werden. Hiermit ist es dann möglich, die einzelnen Gelenkwinkel einzustellen. Eine Besonderheit stellt außerdem der Tool Center Point (TCP) dar, welcher mit einer `KinematicTCP` Node explizit ausgezeichnet werden muss. Hieraus entsteht das *KinModel* (vgl. Abb. 4.35b), welches durch das Teaching Panel (vgl. Abb. 4.35c) direkt bewegt werden kann.

4.3.1.2.3 Dynamischer DT (DynModel)

Kompatibilitätsprobleme treten dann auf, wenn man die hierarchische Baumstruktur mit der „flachen“ Dynamikstruktur [140] übereinbringen will. Hierfür werden Funktionen zum



(a) URDF-Import eines Roboters

(b) Kinematischer Roboter

(c) Kinematischer Roboter: Teaching Panel und TCP

(d) Dynamischer Roboter: Links

(e) Dynamischer Roboter: Joints

(f) Dynamischer Roboter: I/O Board

Abb. 4.35: *URDF-Import zur automatischen Generierung von dynamischen und kinematischen Robotern*

„Flatten“ und zum „Dynamisieren“ benötigt (siehe Algorithmus 2). Simpel gesagt, wird der hierarchisch importierte Baum in einen flachen Baum konvertiert, in dem alle Joints nebeneinanderliegen. Daraufhin können die Knoten automatisiert in Starrkörper/Links umgewandelt und parametriert werden (vgl. Abb. 4.35d). Verbunden durch Joints, analog zum URDF-Modell, entsteht daraus ein ganzheitliches Robotermodell (vgl. Abb. 4.35e). Zusätzlich werden Motoren erstellt, welche im I/O Board mit dem ROS-Framework gekoppelt werden, um den Roboter so direkt kontrollieren zu können (vgl. Abb. 4.35f). Hieraus entsteht ein sogenanntes dynamisches Modell (*DynModel*).

4.3.1.2.4 Modell des Digitalen Zwilling (DigTwinModel)

Um aus diesem dynamischen Modell einen vollständigen Digitalen Zwilling (*DigTwinModel*) zu erstellen, müssen insbesondere noch simulierte Sensoren hinzugefügt werden. Hierfür gibt es ein Skript, welches auf Basis des Repositories Digitaler Zwillinge (vgl. auch Kap. 4.3.2) bestimmte Submodelle (wie Laserscanner, RGB-Kameras usw.) an die korrespondierenden Frames aus der URDF-Datei platziert.

4.3.1.2.5 Functional Mockup Interface (FMI)

Der Vollständigkeit halber sei hier auch auf das Functional Mockup Interface (FMI) [217] [218] und dessen Functional Mockup Unit (FMU) hingewiesen. Initiiert von der Daimler AG hat sich das FMI im Automotive-Bereich als Austausch- und Co- Simulationsmedium etabliert. Auch hier wird ein XML-basiertes Modell erzeugt, welches ausgetauscht werden kann. Zusätzlich lässt sich das Modell noch mit einem Solver koppeln und die-

Algorithm 2 DynModel: Dynamisieren eines importierten URDF-Baumes

```
1:  rootURDFLinkNode
2:  function slotFlattenURDFModelFromHere
3:      allURDFLinks = collectChildNodesRecursively<URDFLinkNode>
4:      allURDFJoints = collectChildNodesRecursively<URDFJointNode>
5:      move all Links remembering WorldFrame → allVSDLinks
6:      move all Joints remembering WorldFrame → allVSDJoints
7:      ↷ slotDynamizeFlattenedLinks(allVSDLinks)
8:      ↷ slotDynamizeFlattenedJoints(allVSDJoints)

9:      procedure slotDynamizeFlattenedLinks(allVSDLinks)
10:         for all URDFCollisionNodes do
11:             VSDCollisionHulls = collectChildNodesRecursively<HullNode>
12:             ↷ slotAddRBShapesToHulls(VSDCollisionHulls)
13:         end for
14:         if !(URDFCollisionNodes) then
15:             for all URDFVisualNodes do
16:                 VSDVisualHulls = collectChildNodesRecursively<HullNode>
17:                 ↷ slotAddRBShapesToHulls(VSDVisualHulls)
18:             end for
19:         end if
20:         for all URDFInertialNodes do
21:             updateIntertiaTensor
22:         end for
23:     end procedure

24:     procedure slotDynamizeFlattenedJoints(allVSDJoints)
25:         for all allVSDJoints do
26:             createRBJointX by URDFJointX
27:              (X = {Fixed, Continuous, Revolute, Prismatic, Mimic})
28:             reference parentRB and childRB in RBJointX
29:             create and connect MotorY
30:              (Y = {PositionControlled, VelocityBased, ForceControlled })
31:             update Properties
32:         end for
33:     end procedure

34:     return Dynamisches Modell: Starrkörper, Gelenke, Motoren usw.
35: end function
```

se FMU zur Co-Simulation nutzen. Eine Integration des FMI sowie eine Erweiterung auf Hardware-in-the-loop-(HIL-)Verfahren mittels der 3D-Simulation VEROSIM wird in [142] beschrieben.

4.3.1.3 Repository Digitaler Zwillinge

Anhand der zuvor geschilderten Benchmark-Szenarien kann man auch mit Hilfe der 3D-Simulation solche Evaluationsverfahren erstellen. Im Gegensatz zu den Simulationsligen des Robocups oder der VRC wird hier der VTB-Ansatz genutzt. Dabei werden zwei Arten von virtuellen Benchmark-Szenarien betrachtet:

1. Rein virtuelle Szenarien und
2. Abbildung realer Szenarien.

wobei sich Szenarien im Allgemeinen unterteilen in:

1. Umgebung,
2. Objekte/Aufgaben und
3. zu evaluierendes System (Roboter).

Für solche simulationsbasierten Benchmark-Szenarien ist der erste wichtige Schritt eine Integration eines *Set von Digitalen Zwillingen* analog zu Abb. 4.31.

Da bei der Abbildung realer Szenarien der Digitale Zwilling eine Abbildung des realen Systems darstellt, muss man sich überlegen, wie dieses reale System aufgebaut ist. Analog zum Digitalen Zwilling findet man hier eine Zusammensetzung aus Sensoren, Aktoren und System-Soft- und Hardware.

Hierfür wird ein *Repository Digitaler Zwillinge (DT-Repo)* angelegt, also eine Sammlung an DT-Modellen, auf die einfach im VTB zugegriffen werden kann, und die als Sub-Modelle in ein Gesamtszenario oder Gesamtmodell eingefügt werden können.

Für rein virtuelle Szenarien können hier beliebige Aufgaben und Objekte kombiniert werden, um verschiedene (robotische) Systeme zu testen und auf spezielle Aufgaben vorzubereiten oder diese zu optimieren. Die Objekte besitzen natürlich eine dreidimensionale Geometrie sowie eine Starrkörpergeometrie, welche häufig aus geometrischen Primitiven (Kugel, Ellipse, Zylinder, Quader usw.) zusammengesetzt ist. Somit können diese Objekte sofort gegriffen/befahren/usw. werden. Es können beispielsweise verschiedene Hindernisse für Lokomotionsaufgaben genutzt werden und diverse Objekte Manipulationsskills testen. Ein exemplarischer Ausschnitt aus dem DT-Repo ist in Abb. 4.36 zu sehen.

Diese Aufgaben können natürlich auch in unterschiedlichen *Umgebungen* mit verschiedenen *Systemen* durchgeführt werden (siehe Abb. 4.37).

Um sich auf reale Benchmark-Szenarien vorzubereiten, macht es natürlich auch Sinn, exakt diese in der Simulation nachzubilden und zu trainieren. Auch sind Ergänzungen mit Objekten des DT-Repos möglich oder neue DTs entstehen nach realem Vorbild. Umgebungsmodelle jedoch müssen meist aus realen Sensordaten nachgebildet werden (siehe Abb. 4.38).

Das führt zu einem modularen und variablen Benchmarking von Systemen in realer Umgebung, die realistische Aufgaben durchführen müssen. Koppelt man dies mit den Nutzerschnittstellen und Visualisierungen aus Kap. 4.2 sowie den Hardware-Schnittstellen in Kap. 4.4, kann hiermit schon ein Gesamtszenario aus Operator und Digitalem Zwilling geprobt werden. Dieser bewegt sich dann mit den gleichen Eingabegeräten wie das finale System durch ein virtuelles Benchmark-Szenario. Parallel produziert der DT sensorischen Output, der wiederum für Postprocessing-Algorithmen genutzt werden kann,

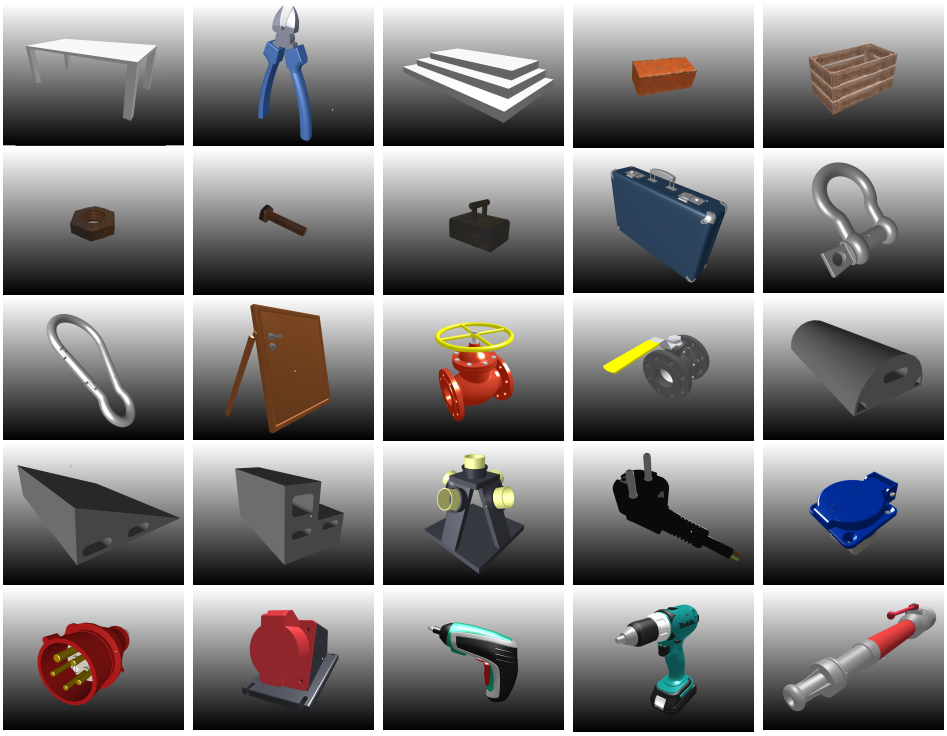
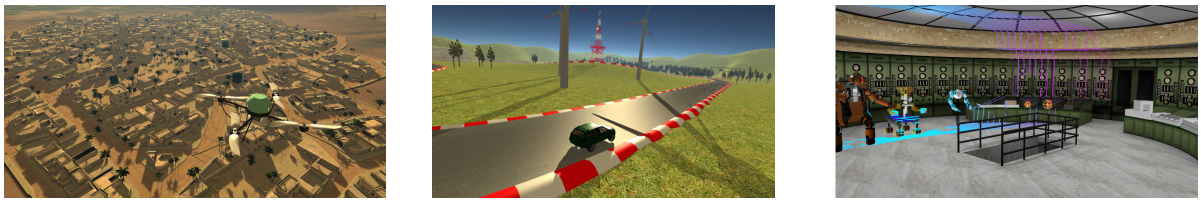


Abb. 4.36: *Repository Digitaler Zwillinge (DT-Repo): Auswahl an Objekten insb. für Lokomotion und Manipulationsaufgaben, welche modular zusammen ein Gesamtszenario ergeben können*



(a) *Quadcopter in einem arabischen Dorf*

(b) *Autonomes Auto auf der Rennstrecke*

(c) *Mobile Roboter im Atomkraftwerkskontrollraum*

Abb. 4.37: *DT-Repo: Variation verschiedener Systeme und Umgebungen*



(a) ***(b)*** ***(c)*** ***(d)***

Abb. 4.38: *Disaster-Szenarien nach realem Vorbild, modelliert (a, b) oder mit Hilfe eines Laserscanners aufgenommen (c, d)*

welche auch im finalen Setup zum Einsatz kommen sollen. Es muss abschließend noch der DT durch den realen Roboter ausgetauscht werden und man hat sogar schon eine virtuelle Evaluation an semi-realen Benchmark-Szenarien durchgeführt.

Damit der Übergang zwischen Digitalem Zwilling und realem System nahtlos ist, bedarf es noch der Integration weiterer Funktionalitäten in die 3D-Simulationsumgebung, welche im folgenden Kapitel vorgestellt werden.

4.3.1.4 Semantische Objekterkennung und Online-Objektgenerierung

Objekte, die zur Laufzeit der Simulationsumgebung hinzugefügt werden müssen, liegen oft nicht als STEP-Dateien vor. Mit Hilfe (externer oder interner) Bibliotheken können Objekte in der realen Szene erkannt werden. Beispielsweise können neuronale Netze zur Objekterkennung genutzt werden und anhand registrierter Punktwolken zusätzlich eine Posenschätzung des Objektes durchgeführt werden, wie in [219]. Hierbei werden grundsätzlich Objekte – wie ein Hammer – in der Umgebung detektiert. Wenn zusätzlich noch örtliche Informationen vorhanden sind, so kann man ein Tupel definieren, das die semantische Information des Objektnamens (String s) mit der Pose $\mathbf{p}$ des Objektes koppelt:

$$(s, \mathbf{p})_i$$

Um dieses Tupel in der 3D-Simulation zu nutzen, erstellt man einen neuen Knoten mit dem Frame $\mathbf{f} = \mathbf{p}$ und nennt ihn anhand des Strings s. Somit ist in der 3D-Szene ein Frame vorhanden, der die Pose des Objektes erkennen lässt. Mittels der sogenannten *„Template-based Model Insertion“* wird nun an diesem Frame das passende Objekt eingefügt. Hierfür wird anhand des Namens das Repository der DTs (siehe Kap. 4.4.2) durchsucht. Sollte man einen DT finden, welcher diesem key-String (exakt) entspricht, wird er an dieser Pose eingefügt. Manuell kann man dann die Dimensionen (meist uniform skaliert) anpassen.

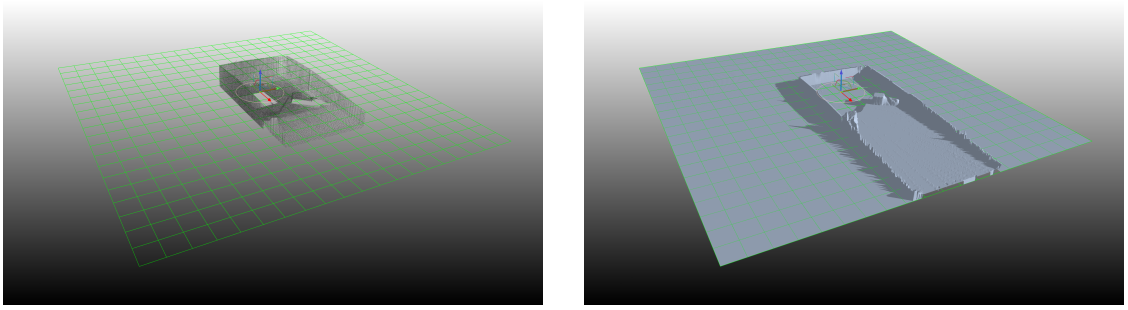
4.3.1.5 Online-Umweltgenerierung

Reale dynamische Umgebungen bestehen überwiegend aus Objekten, dem Boden, Terrain usw. So können mit Bibliotheken (extern oder intern) Punktwolken mittels SLAM (Simulataneous Localization and Mapping) in ein Grid mit Höhendaten umgewandelt werden. SLAM-Algorithmen fassen die Daten aus verschiedenen Sensoren (z. B. LIDAR, IMU, Kameras) zusammen, um die Positionen der Sensoren und die Karte ihrer Umgebung zu erstellen. Die Lösungsmethoden von SLAM enthalten meist den Partikel-Filter (sequenzielle Monte-Carlo-Filter) und den Extended-Kalman-Filter (EKF) [220].

4.3.1.5.1 Höhenkarte

Im Falle von (dem in in Kap. 4.4 beschriebenen) ROS werden die Daten danach mittels `nav_msgs/OccupancyGrid` oder dem projektspezifischen Nachrichtentyp `centauro_msgs/SingleDataMap` übertragen. Beide diskretisieren die Welt in starre unabhängige Zellen, welche zusammen ein Gitter (englisch „Grid“) aufspannen, das pro Zelle mit einem Wert gefüllt ist. Je nach Anwendung entspricht dieser Wert der (absoluten

oder relativen) Höhe oder beinhaltet beispielsweise eine Kostenfunktion für Lokomotionsaufgaben. Somit entspricht dieses Grid im Endeffekt einer Karte (englisch „Map“) der Umwelt (die adäquate Visualisierung wurde in Kap. 4.2.5.4 beschrieben). Mit Hilfe der eigens dafür angelegten Starrkörpergeometrie `HeightField`, welche als Datentyp auf OpenCVs `cv::Mat` aufbaut, können die einzelnen Punkte nicht nur gerendert werden, es wird vielmehr ein Mesh aus diesen erzeugt und in die Starrkörpersimulation eingehangen (siehe Abb. 4.39). Im Rahmen dieser Arbeit wurde hierbei die Starrkörpergeometrie des HeightFields aus [142] genutzt und auf zur Laufzeit veränderliche Datentypen erweitert und optimiert.

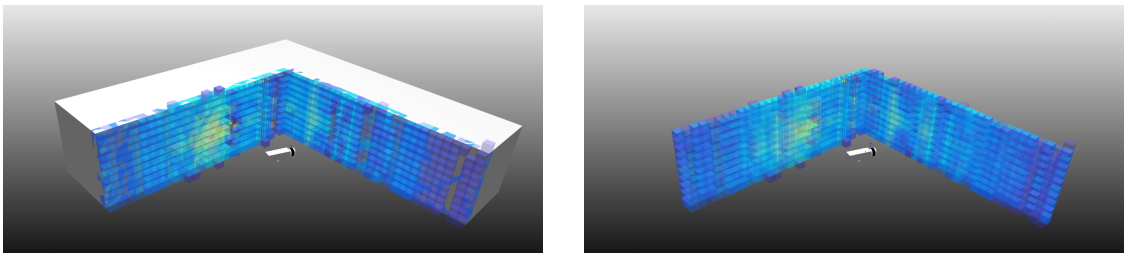


(a) *Punktwolke* ***(b)*** *HeightMap-Starrkörpergeometrie*

Abb. 4.39: *Generierung einer 2.5D-Höhenkarte*

4.3.1.5.2 Voxelisierung

Neben solchen 2.5D-Höhenkarten lassen sich diskretisierte Punktwolken aber auch auf andere Art und Weise in 3D-Geometrien übertragen. Ein Beispiel dafür stellt die *Voxelisation* dar (siehe Abb. 4.40). Hierbei wird für jeden Punkt in der Punktwolke ein Voxel, also ein Einheitskubus, erstellt, der direkt in die Starrkörpersimulation integriert werden kann. Alternativ können die Punkte natürlich auch zu einem Mesh verbunden werden (vgl. auch Kap. 4.2.5.5).



(a) ***(b)***

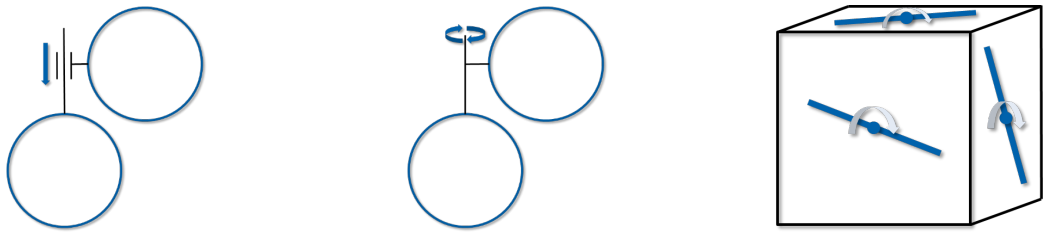
Abb. 4.40: *Voxelisierung von Punktwolken (eines rotierenden Velodyne-Laserscanners)*

4.3.2 Standardisierung in Soft- und Hardware für die 3D-Simulation

In diesem Kapitel geht es um die Standardisierung von Simulationsergebnissen und realen Systemen. Im Detail handelt es sich dabei um die *Vergleichbarkeit* von Simulationsergebnissen sowie Benchmarking-Szenarien in, durch und mit dem DT zur Evaluation von simulierten und realen Systemen.

4.3.2.1 Validierung der 3D-Simulation

Aufbauend auf Kap. 2.5.1.9 hat sich der Wunsch nach einer umfangreicheren Evaluation einzelner physikalischer Sachverhalte innerhalb von VEROSIM im Vergleich mit der analytischen Lösung ergeben. Im Speziellen wird ein in Kap. 4.3.2.2.1 erläutertes Werkzeug (zur Inspektion physikalischer Prozesse anhand der Jacobi-Matrix in MATLAB) genutzt, um die Energie- und Impulserhaltung von verschiedenen Gelenktypen zu analysieren.



(a) Lineargelenk *(b) Drehgelenk* *(c) Rotierende Box*

Abb. 4.41: *Physik-Benchmarking: Energie- und Impulserhaltung von verschiedenen Gelenktypen*

Zwei Kugeln gleicher Masse werden über ein Lineargelenk (Abb. 4.41a) oder ein Drehgelenk (Abb. 4.41b) miteinander verbunden, wobei der Gelenkpunkt einen Offset in z-Richtung hat. Nun wird (visuell) das Verhalten der Kugeln vor und nach der Kollision beobachtet, zunächst vom Lineargelenk. Schon visuell fällt auf, dass die Kugeln, bevor sie kollidieren, ihre Orientierung ändern. Nach der Kollision nimmt die Gesamtenergie des Systems stetig zu, die kollidierenden Kugeln driften voneinander weg und erzeugen ein weiteres Drehmoment. Mit Hilfe des Inspektionstools kann so herausgefunden werden, dass im Code die z-Achsen des Gelenks jeder Kugel nicht aufeinander liegen und somit ein Drehmoment erzeugen, was korrigiert werden kann.

Das andere Inspektionsbeispiel, das Drehgelenk, wird über einen geschwindigkeitsgeregelten Motor angetrieben. Erwartet wird eine uniforme Rotation um die Drehachse, jedoch nimmt auch hier die Drehgeschwindigkeit zu. Dies kann ebenfalls mit Hilfe des Inspektionstools analysiert und im Code durch eine Veränderung der Achsausrichtung behoben werden.

Im letzten Beispiel, der rotierenden Box (Abb. 4.41c), soll ein initiales Motor-Drehmoment mehrerer Drehgelenke nach einer definierten Zeit ausgeschaltet werden und die freischwebende Box sollte aufhören, sich zu drehen.

Nach den zuvor beschriebenen Anpassungen ist dies auch der Fall, jedoch konnte in VEROSIM und in Msc.Adams ein System erzeugt werden, welches sich immer weiter dreht und so die physikalischen Erhaltungssätze verletzte.

Auf Basis dieser Physik-Benchmarks (und der Validierung des Tools) wird im Folgenden ein allgemeiner Ansatz für Benchmarking bzw. Validierung mit Hilfe des Tools präsentiert.

4.3.2.2 Erweiterung der Simulationstechnologie physikalischer Prozesse

Im Allgemeinen basiert die Dynamik auf der eingangs erwähnten geschwindigkeitsbasierten Lagrange-Dynamik (vgl. Kap. 2.5.1.8). Um diese sinnvoll zu erweitern, muss sie erst einmal im Detail verstanden werden. Um jedwede Softwarevalidierung durchführen zu können, muss zunächst die grundlegende Physik dahinter validiert werden können. Hierfür wurden verschiedene Konzepte und Ansätze gewählt, welche im Folgenden vorgestellt werden.

4.3.2.2.1 Inspektion physikalischer Prozesse

Auf Basis der Evaluierung physikalischer Prozesse ist die Inspektion der Lagrange-Dynamik von essentieller Bedeutung. Hierfür können natürlich codebasierte Analysen durchgeführt werden, jedoch ist ein ganzheitlicher Ansatz sinnvoller. Mit Hilfe von MATLAB als zentralem Auswertungswerkzeug mathematischer Probleme soll der Nutzer im laufenden Betrieb der Simulation die Physik dahinter inspizieren und notfalls debuggen können. Dazu werden die aktuelle Jacobi-Matrix und alle assoziierten Parameter in jedem Simulationsschritt in einer Datei gespeichert und können anschließend mit einem automatisierten MATLAB Script untersucht werden.

Für die Evaluation der Jacobi-Matrizen eines Lineargelenks ist ein tieferes Verständnis dieser notwendig. Basierend auf den Erläuterungen in Kap. 2.5.1.8.9 soll hier noch ergänzt werden, wie genau man die Jacobi-Matrix untersucht.

Die Jacobi-Matrix $\mathbf{M}$ hat m Zeilen, wobei m die Anzahl an Zwangsbedingungen bzw. die Anzahl reduzierter Freiheitsgrade des Systems darstellt. In jedem Zeitschritt wird die Jacobimatrix Zeile für Zeile (bzw. Zwangsbedingung für Zwangsbedingung) neu aufgebaut. Für einen Starrkörper sind die Zwangsbedingungen:

$$\mathbf{J}_i \cdot v_i = c \tag{4.3}$$

$$v_i = (v_{ix}, v_{iy}, v_{iz}, w_{ix}, w_{iy}, w_{iz})^T \tag{4.4}$$

Bei zwei Starrkörpern:

$$\mathbf{J}_1 \cdot v_1 + \mathbf{J}_2 \cdot v_2 = c \tag{4.5}$$

$$v = (v_x^1, v_y^1, v_z^1, w_x^1, w_y^1, w_z^1, v_x^2, v_y^2, v_z^2, w_x^2, w_y^2, w_z^2)^T \tag{4.6}$$

Das bedeutet also, bei n Starrkörpern entsteht ein Geschwindigkeitsvektor mit $6n$ Einträgen. Die passende Jacobi-Matrix hat dann m Zeilen und $6n$ Spalten. Das heißt z. B. die erste Zeile der Jacobi-Matrix entspricht dem ersten Constraint und wirkt auf alle Geschwindigkeitsvektoreinträge (also auch alle Starrköper) mit $J_1 v_1 = c_1$ usw. In der Jacobi-Matrix werden zunächst die bilateralen Zwangsbedingungen (equality constraints) und danach erst die unilateralen (complementary constraints) eingetragen. Basierend auf

Gleichung 2.43 folgt:

$$\dot{x} = \mathbf{M}^{-1}\mathbf{J}_c^T\boldsymbol{\lambda}\Delta t + \mathbf{M}^{-1}\boldsymbol{f}_{ext}\Delta t + \dot{x}_0 \tag{4.7}$$

Mit Geschwindigkeit-basierten Zwangsbedingungen ergibt sich:

$$\begin{aligned} \dot{C} &= 0 \\ \frac{d}{dt}C(x) &= \underbrace{\nabla C(x)}_{=\underline{J}(x)} \cdot \dot{x} = 0 \\ \Rightarrow \underline{J} \cdot \dot{x} &= 0 \end{aligned} \tag{4.8}$$

Kombination von Gleichung 2.43 und 4.8 führt zu:

$$\underbrace{\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T}_{A} \cdot \boldsymbol{\lambda} \cdot \Delta t = \underbrace{-\mathbf{J}(\mathbf{M}^{-1}\boldsymbol{f}_{\text{ext}} \cdot \Delta t + \underbrace{\dot{x}_0}_{v_{\text{old}}})}_{b'} \tag{4.9}$$

Mit den Zwangsbedingungen folgert:

$$b = \mathbf{J}(v_{\text{old}} + \mathbf{M}^{-1}\boldsymbol{f}_{\text{ext}} \cdot \Delta t) + C + b_{\text{speed}} \tag{4.10}$$

In jedem Zeitschritt der Dynamiksimulation werden zunächst b bzw. A berechnet. Anschließend wird das LCP vom Solver gelöst, um λ zu bestimmen. Dies wiederum wird genutzt, um die neue Geschwindigkeit $\dot{x}$ und die Orientierung upzudaten. Deshalb werden in MATLAB die Parameter aus Tab. 4.1 analysiert.

Tab. 4.1: *Constraints-Debug-Struct: Jacobi-Parameter zur MATLAB-Evaluation*

Type	Name	Explanation
Double	newTime	current time
MatrixNxM	J	Jacobian
VectorN	b	b
VectorN	l	λ
MatrixNxM	matA	matrix A
VectorN	m	masses
VectorN	pos_old	position in last time step
VectorNDynamic	λ_{low}	limit for friction calculus
VectorNDynamic	λ_{high}	limit for friction calculus
VectorNDynamic	C	constraints right side
VectorN	JvOld	$J \cdot v_{old}$
VextorN	v_old	velocity in last time step
VectorN	v_new	velocity in current time step

Mit Hilfe dieser Extension lassen sich die Lagrangemultiplier der Dynamik analysieren. Angelehnt an das Observer Pattern und die Vorgehensweise bei der Starrkörpersimulation erzeugt die `ExtensionDebugLagrangeMFile` einen `RBDClusterLagrangeDebugObserver`

(LDO), welcher sich in die `RBDScene` einhängt und von der an den `RBDClusterLagrangeMultiplier` weitergegeben wird. Dieser updatet den LDO und bei Simulationsstopp wird das Ganze in einen MATLAB m-file geschrieben.

Jeder Eintrag ist ein sogenanntes Constraints-Debug-Struct (siehe Tab. 4.1). Diese können dann in MATLAB weiter untersucht werden. Das Struct ist im LDO zu finden. Zur Evaluation wird nun der Gesamtimpuls und Gesamtdrehimpuls berechnet. Der Gesamtimpuls setzt sich zusammen aus Einzelimpulsen. Der Gesamtdrehimpuls enthält zusätzlich noch einen Term der per Kreuzprodukt aus der Position (pos_old) berechnet wird. Dies wird in der Literatur als „shifting rule“ bezeichnet. Wenn die Massen der Körper identisch sind, ist eine Betrachtung der Geschwindigkeiten ausreichend. Das heißt, wir definieren die Einträge in der Jacobi-Matrix als Geschwindigkeitskomponenten. Am Ende kann die Summe

$$\sum_i J(v_i(x,y,z), w_i(x,y,z)) = 0$$

pro Zeitschritt überprüft werden, indem in jeder Jacobi-Matrix jede Spalte (Constraint) betrachtet wird. In jeder Spalte addiert man dann Geschwindigkeiten und Drehgeschwindigkeiten der verschiedenen Starrkörper miteinander und erhält eine resultierende Matrix (cons[1]), welche pro Zeitschritt eine Zeile beinhaltet:

$$(v_x^1, v_y^1, v_z^1, w_x^1, w_y^1, w_z^1, v_x^2, v_y^2, v_z^2, w_x^2, w_y^2, \cdots)$$

Das heißt, für jedes Constraint gibt es sechs fortlaufende Elemente. Diese Matrix hat dann bei m Constraints $6m$ Spalten, die in 3er Gruppen gelesen werden müssen. Über die Zeit entsteht so eine gesamte Matrix, die auch visuell dargestellt werden kann. Dies kann dann mit dem Idealfall analysiert werden, sodass gilt:

$$v_{ges} = w_{ges} = 0$$

Somit sind die Voraussetzungen geschaffen, die aufgenommenen Parameter aus Tab. 4.1 zu analysieren. Des Weiteren können der Gesamtdrehimpuls sowie die Einzel-Constrainsts berechnet und aufsummiert werden. Am Ende sollte die Gesamtsumme aller Elemente gleich Null sein, um physikalisch korrektes Verhalten zu zeigen. Sollte dies nicht der Fall sein, kann zunächst in MATLAB und anschließend im C++-Code der Fehler gesucht werden.

Das Endresultat der dynamischen Analyse eines Lineargelenks ist in Abb. 4.42 dargestellt. Ein Neuausrichtung des Ankerpunktes des Lineargelenks führte vom vorherigen Verhalten (oben) zu einem korrigierten Verhalten (unten). Die notwendige Korrektur stellte sich in diesem Fall als Orientierungs-mismatch heraus. Die Korrektur zweier Zeilen der Jacobi-Matrix, also die Berücksichtigung der momentanen Auslenkung, trug zur Verbesserung der Impulserhaltung bei. Die resultierende Kurve Abb. 4.42 beschreibt die „consistency of the constraints (cons)“ (also die Impulserhaltung) über die Zeit. Die Schwankung um die Ideale Null ist nach der Neuausrichtung des Ankerpunktes um den Faktor 10^4 gestaucht.

Insgesamt kann dieses Validierungsverfahren in verschiedenen Testszenarios oder Si-

[1] consistency of constraints

mulationsframeworks genutzt werden. Dadurch ist es sehr leicht möglich, Energie- und Impulserhaltung zu untersuchen und zentrale Fehlerquellen zu identifizieren.

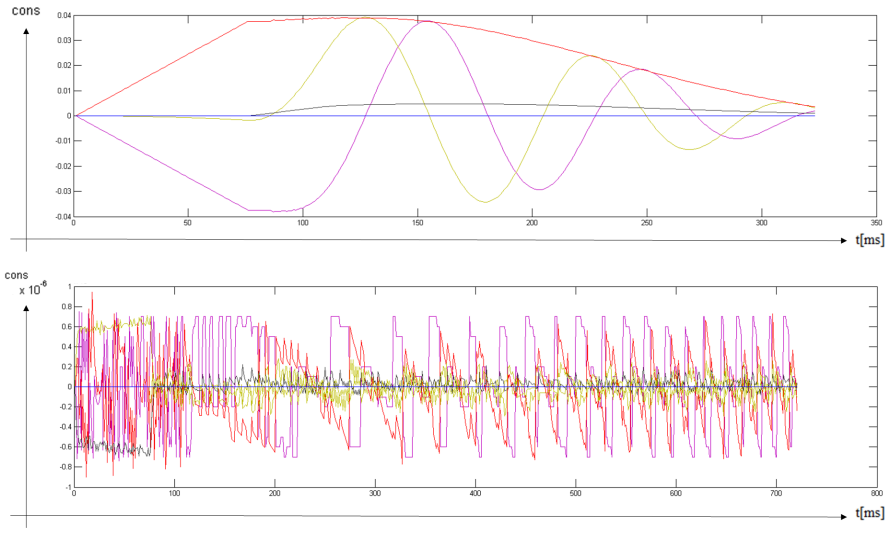


Abb. 4.42: *MATLAB: Konsistenzanalyse eines Lineargelenks bevor (oben) und nach (unten) der Neuausrichtung des Ankerpunktes.*

4.3.2.2.2 Physikalische Inline-Validierung

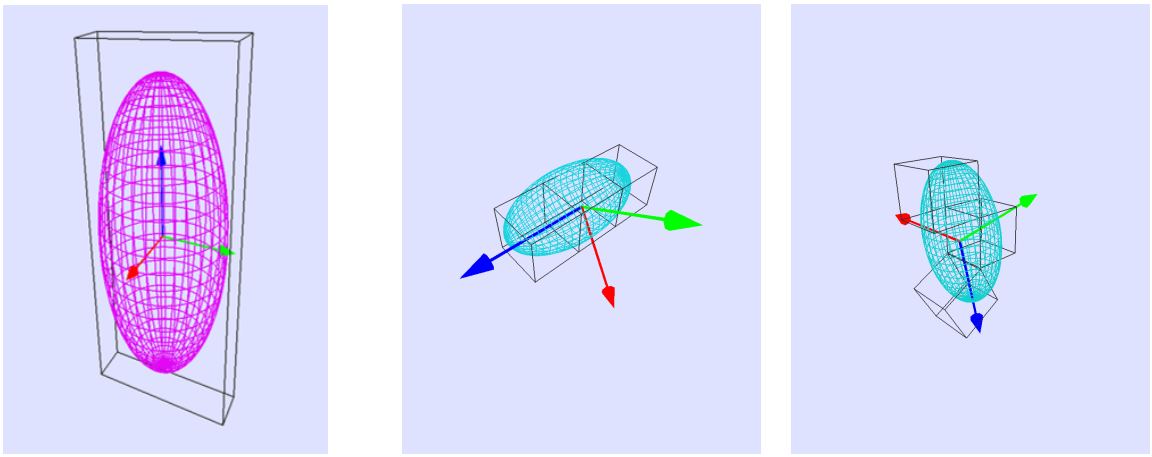
Neben Energie- und Impulserhaltung sind bei Starrkörpern vor allem die Masse, der Schwerpunkt und somit das Trägheitsmoment für ihr Verhalten und ihren Krafteinfluss ausschlaggebend.

Starrkörperverbund

Während es bei simplen Geometrien noch trivial ist, wie das Trägheitsmoment aussieht, ist es bei zusammengesetzten komplexen Geometrien, die sich bewegen, oft schwer, durch genaues Hinsehen den Schwerpunkt und das Trägheitsmoment zu erahnen. Insbesondere bei Greifprozessen, in denen neben der eigenen Geometrie noch eine „Fremde“ hinzukommt, ist dies besonders entscheidend, auch für die Regelung des Roboters. Daher wird in der Simulation die Möglichkeit geschaffen, faktisch nicht zusammengehörige Objekte zu einem virtuellen *„Composite Body“* zusammenzufassen. Für die Berechnung dieses zusammengesetzten Körpers sind folgende Parameter aller beteiligten Starrkörper nötig: Masse m, Schwerpunkt CoG und kombinierter Trägheitstensor $\mathbf{I}$.

Masse und Trägheitstensor werden wie in Kap. 2.5.1.8.7 beschrieben berechnet. Damit der Trägheitstensor während eines Greifvorgangs inspiziert werden kann, ist eine Visualisierung implementiert. Trägheitstensoren spannen ein sogenanntes Trägheitsellipsoid auf (siehe Abb. 4.43). Die diagonalisierte Matrix (I) wird mit den Eigenvektoren multipliziert, um auf die Hauptträgheitsachsen zu kommen $(I \cdot EV)$ und drei orthogonale Vektoren zu erhalten. Diese drei Hauptachsen des Ellipsoids sind identisch zu den Hauptträgheitsachsen des Gesamtträgheitstensors. Die Längen der Achsen ergeben sich aus $1/\sqrt{I}$. Durch Platzieren des Trägheitsellipsoids im neuen Gesamtschwerpunkt

bekommt der Nutzer sofort einen guten Einblick in die zugrundeliegende Physik des Starrkörperverbunds. Die Achsen des Ellipsoids können dabei per Hand noch gemeinsam um einen Faktor gestreckt oder gestaucht werden. Rotationen um die Achse mit dem größten und mit dem kleinsten Trägheitsmoment sind stabil. Rotationen um die Achse mit dem mittleren Trägheitsmoment sind nicht stabil und kleine Störungen führen zum Torkeln.



(a) *Quader mit zwei stabilen und einer instabilen Trägheitsachse*

(b) *Bewegliche Boxen mit zwei Drehgelenken*

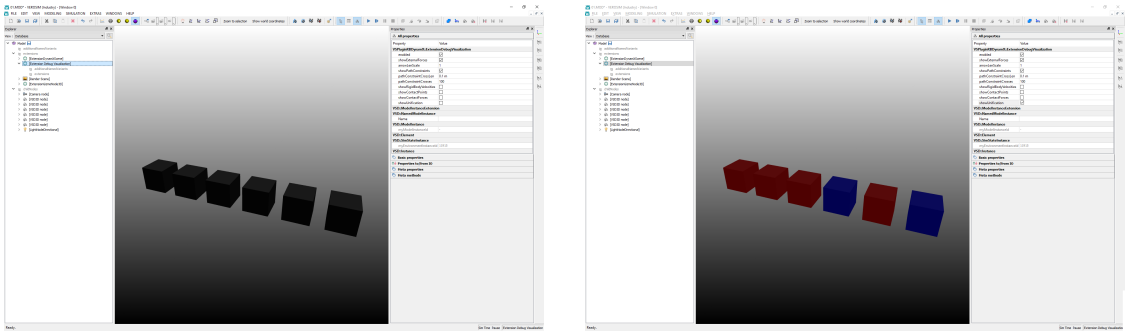
Abb. 4.43: *Starrkörperverbund: zusammengesetzte Starrkörper mit Visualisierung des Trägheitstensors*

Starrkörpercluster-Visualisierung
Zusätzlich zur visuellen Inspektion des Trägheitstensors ist es nun möglich, verbundene Starrkörpergruppen anzeigen zu lassen. Zu Debug- und Analysezwecken bzw. als Modellierhilfe können mit Hilfe einer `ExtensionDebugVisu` alle Geometrien einer Starrkörpergruppe mit einer Farbe übermalt werden, sodass man sofort sieht, welche Starrkörpercluster vorhanden sind (vgl. Abb. 4.44). Auch die Verschmelzung verschiedener Gruppen durch ein starres Gelenk lässt sich so anzeigen.

4.3.2.2.3 Physik zum Anfassen

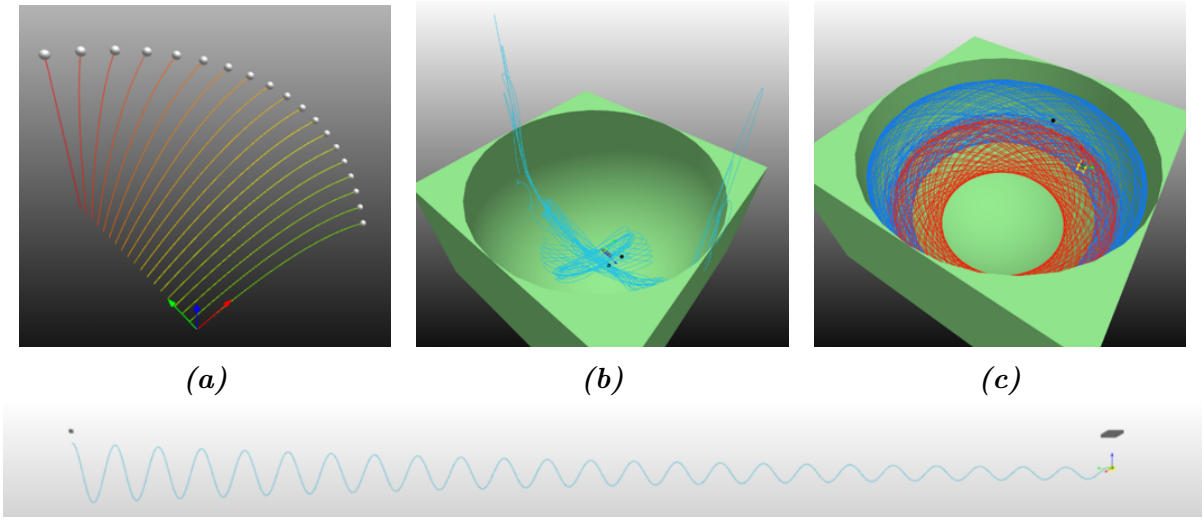
Durch die 3D-Visualisierung ist ein sehr direkter und intuitiver Zugang zu physikalischen Parametern möglich, wie das zuvor erwähnte Trägheitsellipsoid eines Körpers. Eine Simulation ist natürlich nur die Approximation des Gesamtsystems, jedoch kann man schnell und einfach physikalische Plausibilität prüfen, indem man bekannte physikalische Phänomene in der Simulation modelliert und anschließend simuliert. Der Detailgrad einer solchen Evaluation ist dabei dem Nutzer überlassen. Im Folgenden sollen ein paar Beispiele gegeben werden, welche den in Kap. 4.3.2 beschriebenen Physik-Benchmarks ähneln.

In Abb. 4.45 sieht man die Energie beim schiefen Wurf einer Kugel (a), die chaotischen Trajektorien zweier Kugeln in einer Schale (b, c blau vs. rot) sowie die harmonische



(a) Starrkörpercluster nicht sichtbar *(b)* Starrkörpercluster sichtbar

Abb. 4.44: *Starrkörpercluster-Visualisierung: visuelle Überlagerung der zusammengehörigen Körper*

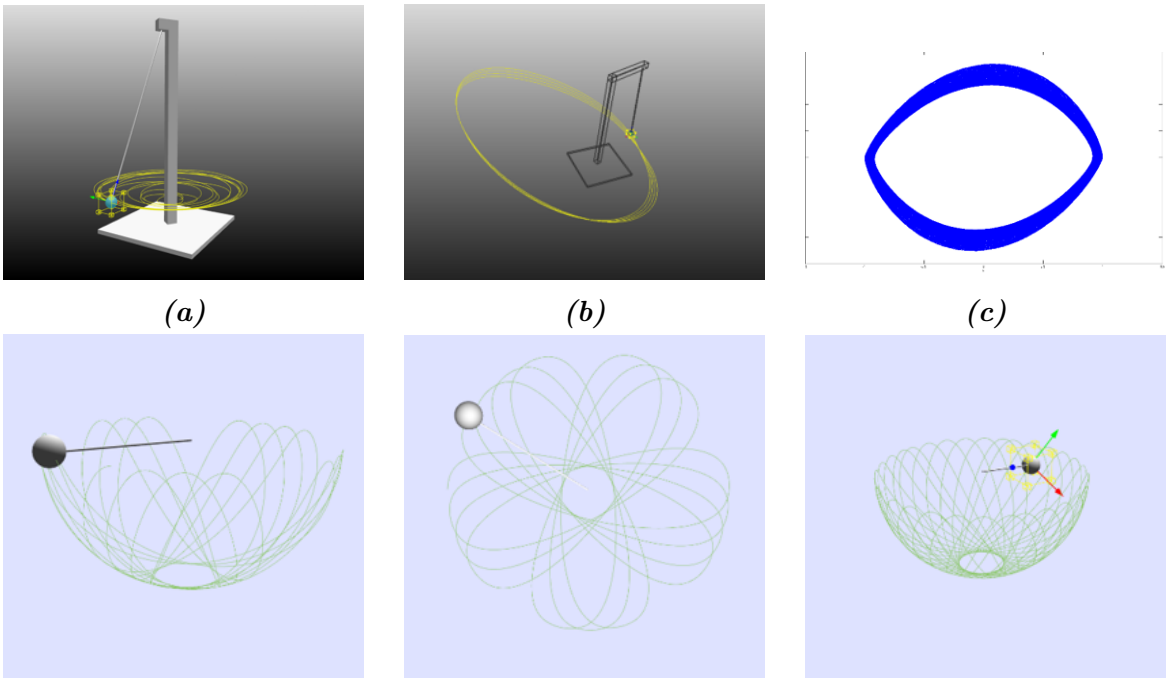


(a) *(b)* *(c)*

(d)

Abb. 4.45: *Physik zum Anfassen: Schiefer Wurf (a); Kugeltrajektorien (b, c); Feder-Dämpfer-System (d)*

Schwingung einer Feder über die Zeit (d) (und im Raum). Abb. 4.46 zeigt ein starres Pendel. In (a, b, c) erkennt man die verschiedenen Trajektorien des Pendels, wodurch man auch den Satz von Liouville „Das von benachbarten Trajektorien im Phasenraum eingeschlossene Volumen ist konstant“ sowohl graphisch als auch in MATLAB später numerisch analysieren kann. In (d, e, f) hingegen kann man Lissajous-Figuren erkennen. Dies sind Kurvengraphen, die durch die Überlagerung zweier harmonischer, rechtwinklig zueinander stehender Schwingungen entstehen, benannt nach dem französischen Physiker Jules Antoine Lissajous (1822 – 1880). Mit der 3D-Simulation und den hier vorgestellten visuellen Metaphern – hierbei in Form der `TracerExtension` – kann man sehr gut physikalische Phänomene in der Simulation erkennen, verstehen und erklären.

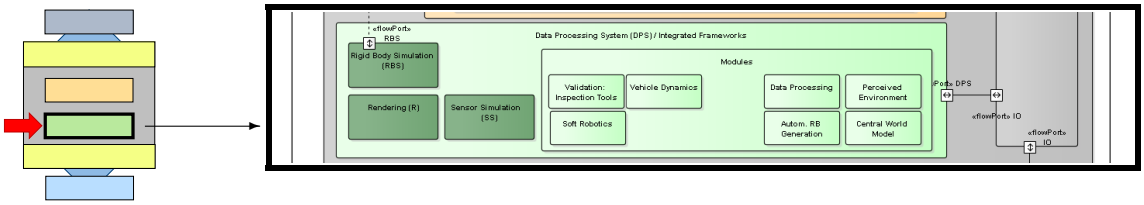


(a) *(b)* *(c)*

(d) *(e)* *(f)*

Abb. 4.46: *Physik zum Anfassen: das Pendel*

4.3.3 Integration notwendiger Funktionalitäten in das umgebende 3D-Simulationssystem



(a) *(b)*

Abb. 4.47: *Konzept: integrierte Funktionalitäten – Datenverarbeitung (vgl. Abb. 3.4)*

Wie schon zu Anfang erwähnt, umfassen Simulationen immer komplexere Prozesse. Begonnen mit einfachen physikalischen Phänomenen wie einem Pendel, geht es über beispielsweise Automobile zu Robotern. Genau dies erfordert die Integration weiterer Funktionalitäten in das umgebende Simulationssystem. So ist es oft nicht ausreichend, den Rad-Straßen-Kontakt durch einfache starre Kugeln zu approximieren, da gerade dieser ein essentieller Bestandteil der Stabilität eines Motorrads ist. Im Folgenden werden daher Erweiterungen vorgestellt und in den Kontext der Anwendung eingeordnet. Diese umfassen zum einen die Evaluation und Inspektion physikalischer Prozesse in der Simulation und zum anderen die Erweiterung alter oder Spezialisierung neuer Frameworks, welche von zentraler Bedeutung für die physikalisch korrekte Simulation in diversen Anwendun-

gen sind.

Selbstverständlich gibt es eine Vielzahl von Möglichkeiten, um ein 3D-Simulationssystem zu erweitern. Im Rahmen der Anforderungen stehen hier aber primär eine Erweiterung der Möglichkeiten des Starrkörperframeworks sowie die Automatisierung im Rahmen der Modellierung im Fokus.

Im Folgenden sollen daher die Integration spezialisierter Frameworks im Bereich Automotive (Kap. 4.3.3.1) sowie Soft Robotics (Kap. 4.3.3.2) behandelt werden. Zusätzlich implementierte Funktionalitäten im Bereich des Renderings (insbesondere in Form von diversen Datenvisualisierungen) sowie der Sensordatenverarbeitung sind dagegen im UCL und somit in Kap. 4.2 bereits beschrieben.

4.3.3.1 Fahrphysik in der 3D-Simulation

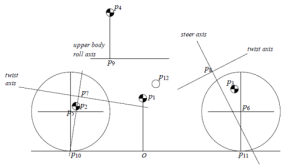
Für die korrekte Abbildung Digitaler Zwillinge im Automotive-Bereich sind Erweiterungen der Physiksimulation notwendig. Insbesondere geht es dabei um die korrekte Abbildung der verschiedenen Bestandteile eines *Autos* – Getriebe, Übersetzung usw. – sowie das regelungstechnische System eines *Motorrades* und der dafür so wichtige Rad-Straßen-Kontakt.

4.3.3.1.1 Motorradsimulation

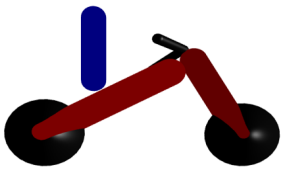
Zweiräder an sich sind sehr interessante dynamische Systeme. Beim Fahren dieser treten viele verschiedene dynamische Phänomene (gyroskopischer Effekt, Präzession, Nachlauf, Massenverteilung etc.) gleichzeitig auf und der Mensch tritt selber als Regler in Aktion. Obwohl schon seit über 100 Jahren an den Gründen für die Stabilität von Zweirädern geforscht wird, konnte dafür bisher noch keine nicht widerlegbare Begründung gefunden werden (vgl. [221]).

Regelungstechnisch kann das Motorrad als sequenzielle Weiterentwicklung des inversen Pendels gesehen werden. Das inverse Pendel wiederum kann als das simpelste robotische System bestehend aus einem starren Körper und einem Rotationsgelenk gesehen werden und ist daher der Benchmark für nichtlineare Regelungssysteme [222]. Das Prinzip des inversen Pendels muss hierbei zur Anwendung bei Zweirädern auf die geometrischen Besonderheiten angepasst werden. Im Vergleich zum Segway wird ein Zweirad nicht in Fahrtrichtung, sondern senkrecht dazu geneigt. Um nun, wie bei der Stabilisierung eines inversen Pendels erforderlich, den Auflagepunkt des Zweirades unter den Schwerpunkt zu bewegen, kann keine geradlinige Bewegung in Fahrtrichtung erfolgen. Bei Zweirädern wird das Fahrzeug durch Lenken in die Richtung, in die es geneigt ist, stabilisiert. Durch den Lenkeinschlag wird der Auflagepunkt des Vorderrades in die Richtung des Fallens verschoben.

Auf Basis der Starrkörpersimulation wird ein DT analog zu Abb. 4.48a und Abb. 4.48b in VEROSIM integriert. Charakteristische Merkmale sind der Radstand und der Winkel zwischen der Vertikalen und der Lenkachse. Durch die Implementation können die *Stabilisierung der Geradeausfahrt* und die *aktive Kurvenfahrt* daraufhin automatisiert durchgeführt werden. Bei der Anfahrt auf eine Kurve wird nun vereinfachend von einer aufrechten Fahrlage ausgegangen. Beim Durchfahren einer Rechtskurve wirkt die Fliehkraft nach außen. Dementsprechend würde bei einer direkten Bahnkurve nach rechts das



(a) Sharp-Modell [223]



(b) Approximation inkl. Fahrer



(c) 3D-Geometrie



(d) Finales 3D-Starrkörpermodell

Abb. 4.48: *Motorradsimulation: Motorrad BMW K1600 GT*

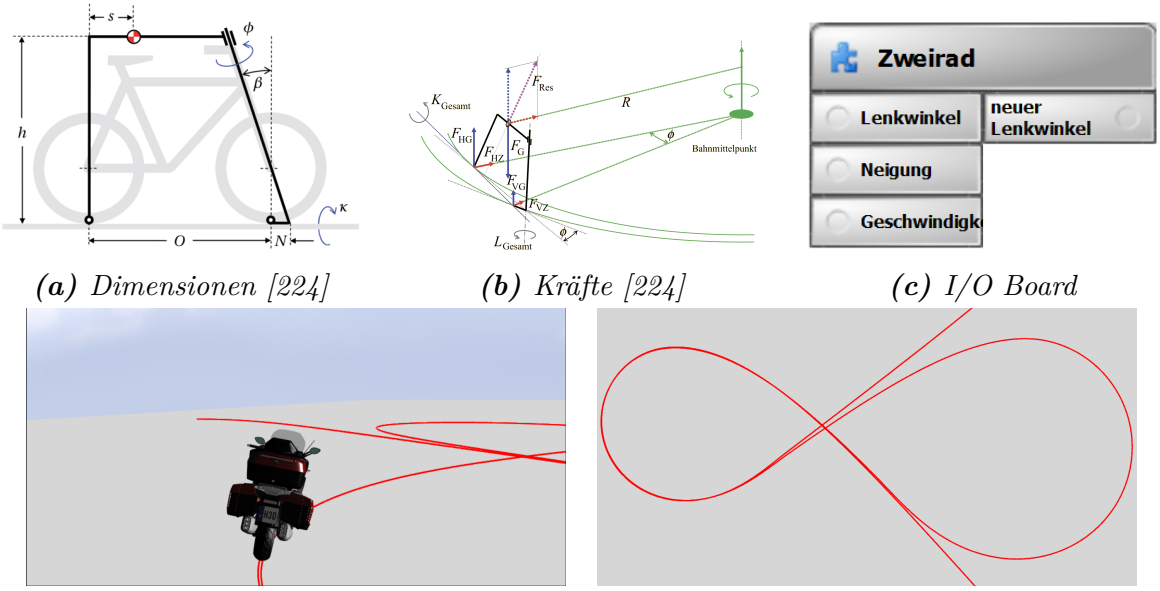
Zweirad nach links kippen. Somit ist es erforderlich, für eine Rechtskurve zuerst nach links zu lenken, damit das Zweirad nach rechts geneigt wird. Dies wird als Gegenlenken bezeichnet. Aus Gründen der Stabilität muss anschließend so weit nach rechts gelenkt werden, dass eine stationäre Lage erreicht werden kann. Diese zeichnet sich dadurch aus, dass die resultierende Kraft aus Flieh- und Gewichtskraft genau in der Symmetrieachse des Zweirades liegt. In diesem Zustand wirkt kein weiteres Kippmoment, sodass bei konstanter Geschwindigkeit, unverändertem Lenkeinschlag und gleichbleibender Position des Massenschwerpunktes eine Kreisfahrt beschrieben wird (siehe Abb. 4.49 und Tab. 4.2). In der Realität erhöhen Motorradfahrer nach dem Scheitelpunkt der Kurve die Geschwindigkeit, um so das Aufrichten der Maschine zu erleichtern.

Da die Simulation des Kurvenfahrens wenig intuitiv und die Reaktion des Modells auf einen kleinen Lenkeinschlag nicht kontrollierbar sind, wird ein *Assistenzsystem* in Form einer Extension für VEROSIM implementiert, welche das Zweirad bei der Geradeausfahrt stabilisiert und den Fahrer insbesondere beim Kurvenfahren unterstützt. Im ersten Schritt werden grundlegende Berechnungen durchgeführt. Mit den oben beschriebenen Eingangsgrößen lassen sich nach [225] der Kurvenradius und die benötigte ideale Schräglage für die gewünschte Kurve sehr gut approximieren:

$$R = \frac{O \cdot \cos(\phi)}{\cos(\beta) \cdot \phi} \tag{4.11}$$

$$\text{und } \kappa = \arctan\left(\frac{v^2}{g \cdot R}\right) \tag{4.12}$$

Per Definition wird die Neigung nach rechts mit einem positiven und die Neigung nach



(a) Dimensionen [224] *(b) Kräfte [224]* *(c) I/O Board*

(d) Durchfahren einer Acht

Abb. 4.49: *Motorradsimulation Kurvenfahrt: (1) detektiere Beginn einer Kurve, (2) erreiche die nötige Schräglage, (3) halte Schräglage, (4) Aufrichten*

Tab. 4.2: *Relevante physikalische Parameter zur Simulation eines Zweirads*

Bezeichnung	Variable
Höhe des Schwerpunktes vom Boden	h
horizontale Verschiebung des Schwerpunktes von der Hinterradachse	s
Radstand	O
Nachlauf	N
Winkel der Lenkachse	β
Steuerkopfwinkel	$90° - -\beta$
Lenkwinkel	ϕ
Neigung der Hinterradebene normal zur Fahrtrichtung	κ
Gewichtskraft	F_G
Zentrifugalkraft	F_Z
Reibungskraft	F_R

links mit einem negativen Neigungswinkel beschrieben. Damit stimmen die Vorzeichen, bezogen auf die Richtung, mit dem für die Lenkerstellung überein. Auch hier ist der Lenkwinkel nach rechts positiv definiert. Außerdem werden der vom Nutzer eingegebene Lenkwinkel zu Beginn der Kurve und die Richtung der Kurve gespeichert.

Im zweiten Schritt wird die Gegenkurve gefahren. Dazu wird mit einem PD-Regler jeweils ein neuer Wert für die Lenkerstellung generiert, bis ein angenäherter Wert der aktuellen an die zuvor berechnete ideale Neigung erreicht ist.

Schritt drei dient dazu, das Zweirad in der stabilen Schräglage zu halten. Ein PD-

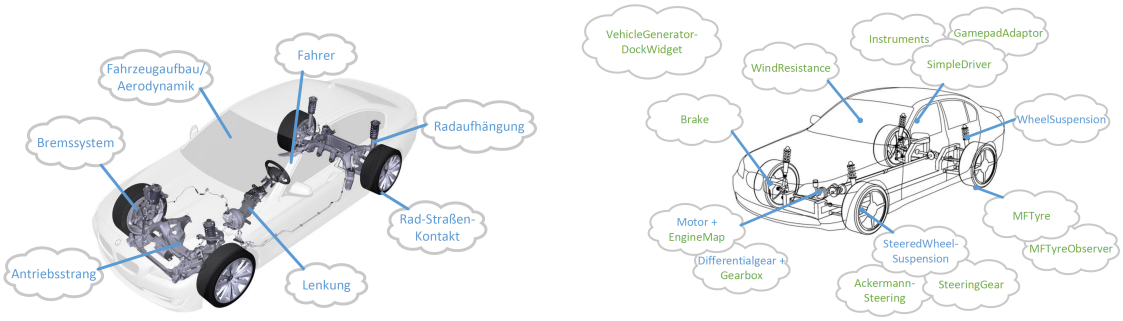
Regler vergleicht dafür fortlaufend die aktuelle mit der idealen Neigung und berechnet daraus eine neue Stellung des Lenkers. Dies wird solange wiederholt, bis vom Fahrer signalisiert wird, dass die Kurve beendet werden soll.

An dieser Stelle beginnt der vierte Schritt. Hier wird das Zweirad analog zu der Stabilisierung der Geradeausfahrt wieder aufgerichtet. Wird ein Grenzwinkel der Neigung unterschritten, so gilt diese Kurve als durchfahren und es besteht die Möglichkeit, geradeaus weiter zu fahren oder eine neue Kurve zu beginnen.

Im Ablauf dieses Konzeptes ist die Schwierigkeit zu lösen, dass Kurven mit einem geringeren Radius durchfahren werden können. Für diesen Fall sind die Übergänge zwischen Schritt zwei und drei sowie zwischen den Schritten drei und vier kritisch. Gerade der eben angesprochene erste Übergang bedingt eine spezielle Regelung, damit die Lenkerstellung nicht zu abrupt geändert wird, sondern das Zweirad mit einer sanften Bewegung in die Schräglage bewegt und dort auch gehalten wird. In der Realität findet bei einer Kurve mit engem Radius ebenfalls keine abrupte Änderung der Lenkerstellung statt. Die Änderung der Lenkerstellung erfolgt fließend. Geringe Abweichungen können vom Fahrer leicht korrigiert werden. Wird in der Realität eine Kurve mit sehr geringem Radius durchfahren, so ist es nur mit permanenten (und nichtlinearen) schnellen Korrekturbewegungen möglich. Damit ergibt sich keine stabile Fahrlage.

4.3.3.1.2 Automobilsimulation

Gerade im Automotive-Bereich werden vielfach Simulationen eingesetzt, um Fahr- und Bremsverhalten, Motorsteuerung, Windwiderstand, Nutzerfreundlichkeit usw. zu untersuchen. Im Rahmen von [226] wurde VEROSIM um das Gebiet der Fahrdynamiksimulation erweitert. Hierbei wurde insbesondere auf Echtzeitfähigkeit und Modularität Wert gelegt. Zusammenfassend dargestellt sind die Entwicklungen in Abb. 4.50.



(a) Bestandteile *(b) Implementierte Module*

Abb. 4.50: *Automobilsimulation*

Im Rahmen dieser Arbeit sind hierbei insbesondere der `VehicleGenerator`, `MFTire`, `SimpleDriver` sowie `Instruments` hervorzuheben.

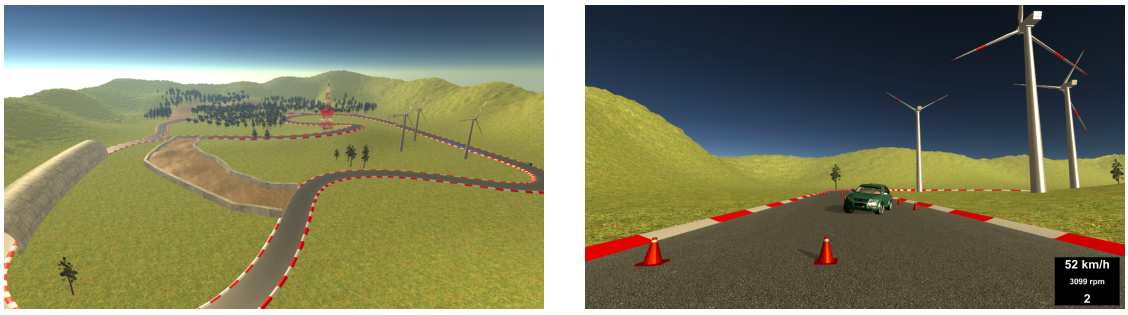
Für den Rad-Straßen-Kontakt wurde das MFTyre Packet[cf] implementiert, das auf Paceijkas „Magic Formula“ [227] basiert:

$$F(x) = D \cdot \sin\left(C \cdot \arctan\left(B \cdot x - -E \cdot B \cdot x + E \cdot \arctan\left(B \cdot x\right)\right)\right) \tag{4.13}$$

wobei B, C, D und E Fittingparameter und F die Kraft bzw. das Moment, ist welches aus dem Schlupf x herrührt. Mit dieser Gleichung wird bestimmt, welche externen Kräfte und Momente am Kontaktpunkt für den nächsten Simulationszeitschritt hinzufügt werden. Als Starrkörper für Reifen wird die zuvor implementierte Ellipsoidgeometrie (siehe Kap. 4.2.5) genutzt, wobei das Starrkörpermaterial stetig angepasst wird. Im nächsten Kapitel wird dieses Reifenmodell noch erweitert.

Der „Vehicle Generator“ automatisiert die Generierung von Fahrzeugmodellen. Nach Eingabe der wichtigsten Parameter zu Chassis, Reifen, Lenkung, Getriebe, Bremsen und Geometrie wird automatisch ein passendes Starrkörpermodell erschaffen, das alle entwickelten Fahrdynamik-Bestandteile enthält.

Weitere Hilfen für den Nutzer stellen die Instrumente und der „SimpleDriver“ dar. Die Instrumente beinhalten simple Visualisierungen für den aktuellen Gang, die Geschwindigkeit und die Umdrehungen. Der SimpleDriver dagegen kann anhand eines beliebigen Pfades Radlenkwinkel und Gaspedalposition so anpassen, dass das Auto mit Hilfe eines PID-Reglers auf diesem Weg bleibt. Durch eine Verallgemeinerung der Assistenzsysteme können diese in beliebigen Anwendungen genutzt werden (wie z. B. in Abb. 4.51).



(a) *Rennstrecke* ***(b)*** *Parkour mit Fahrzeuginstrumenten*

Abb. 4.51: *Automobilsimulation: Rennstrecke*

4.3.3.2 Soft-Robotik-Simulation

„Soft Robotics“ umfasst den gesamten Forschungsbereich weicher Roboter. Dieser beinhaltet neben deformierbaren, weichen Strukturen natürlich auch eine weiche Regelung im Sinne der Nachgiebigkeit. Die Simulation von Soft Robots ist eine Herausforderung bzgl. (1) der enormen Anzahl an Freiheitsgraden, (2) dem Übergang von gewöhnlichen DGLs zu partiellen DGLs [96], (3) nichtlinearen Effekten in Material und Geometrie [228] [229] sowie (4) der Parametrisierung solcher Systeme. Des Weiteren haben softe Roboter keine diskreten Gelenke und Strukturen. Sie können sich elastisch verformen. Dafür werden neue regelungstechnische Herangehensweisen benötigt [230]. Simulatoren für solche Soft Robots sind meist auf FEM-Basis implementiert und sehr zeitaufwändig. Andere Möglichkeiten sind VoxCAD [bl], welches ein voxelbasiertes Feder-Dämpfer-Modell nutzt, um zusammengesetzte Strukturen aus mehreren 3D-Boxen zusammenzusetzen, oder der Soft Cell Simulator [231], welcher in 2D Verbindungen auf zellulärer Ebene ermöglicht, um multizelluläre Roboter entwickeln zu können. Einzelne Ansätze zur Integration in ein Starrkörper-Simulationssystem werden in [142] präsentiert und hier erweitert.

4.3.3.2.1 Starrkörperbasierte Deformation – Soft-Robotik-Simulation

Obwohl es widersprüchlich erscheint, ist es möglich, deformierbare Körper durch Starrkörper zu simulieren. Als Beispiel wird der „Bionic Handling Assistant“ von Festo betrachtet, dessen Simulation in Abb. 4.52 gezeigt ist. Der simulierte Körper besteht aus vielen kleinen Starrkörpern. Das Verhalten eines Soft Robots wird durch das Einfügen angepasster parametrierter Feder- und Dämpfungsglieder simuliert [232].

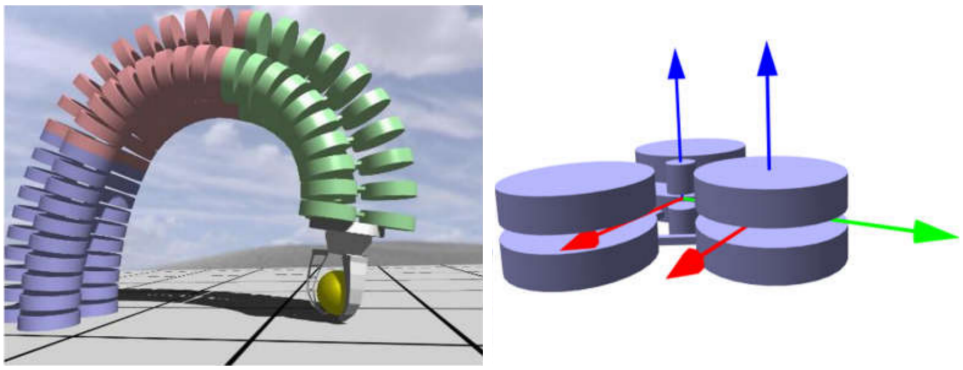


Abb. 4.52: *Simulation des Bionic Handling Assistant von Festo [232].*

Dieses Modell von deformierbaren Körpern hat den Vorteil, dass die in VEROSIM implementierte Starrkörpersimulation genutzt werden kann. Somit ist es ohne Erweiterung der Kollisionsbehandlung möglich, echtzeitfähige deformierbare Körper zu erstellen. Durch geeignete Einstellung der Parameter der Starrkörper und Gelenke ist es möglich, so eine realistische Simulation vieler Arten von deformierbaren Körpern darzustellen.

Ein Nachteil dieser Methode ist, dass nicht beliebige deformierbare Körper und somit nicht alle Soft Robots simuliert werden können, ohne andere Kriterien zu verletzen. Um die Deformierbarkeit des Soft Robots zu simulieren, muss der Körper als eine große Menge von sehr kleine Starrkörpern moduliert werden, die mit entsprechenden Gelenken, Federn und Dämpfern miteinander verbunden sind. Dies ist grundsätzlich möglich, aber jeder dieser Starrkörper wird als Körper mit sechs Freiheitsgraden angesehen, die ggf. durch ein Gelenk eingeschränkt werden. Bei dieser Darstellung ist durch die große Menge an kleinen Starrkörpern die Echtzeitfähigkeit gefährdet. Aus diesem Grund ist das Modell für die Simulation einiger Soft Robots ausreichend, aber es bietet keine vollständige Integration deformierbarer Körper.

In [142] ist die in VEROSIM implementierte Erweiterung des Modells beschrieben. Um mit Luft gefüllte deformierbare Körper wie Bälle zu simulieren, wird ein Modell mit einem volumenorientierten Polyeder benutzt. Aus diesem Grund wird bei diesen Modellen das Volumen der deformierbaren Körper durch eine Potentialfunktion bestimmt:

$$E_V(\boldsymbol{p}_0, \ldots, \boldsymbol{p}_n) = \frac{1}{2} k_V \left(\frac{V(\boldsymbol{p}_0, \ldots, \boldsymbol{p}_n) - V_0}{V_0} \right)^2 \tag{4.14}$$

Um diese Gleichung zu lösen, werden Volumen mit Tetraedern gebildet, die durch ein Oberflächendreieck des Polyeders und einen festen Punkt erstellt werden (siehe Gleichung 4.15 und Abb. 4.53). Hier ist als fester Punkt der Koordinatenursprung festgelegt.

$t.\boldsymbol{p}_{i/j/k}$ sind die Ecken des Oberflächendreiecks t im positiven Umlaufsinn um die nach außen positiv orientierte Normale des Dreiecks. Auf die gleiche Art können die volumenerhaltenen Kräfte berechnet werden:

$$V(\boldsymbol{p}_0, \ldots, \boldsymbol{p}_n) = \sum_t \frac{1}{6} t.\boldsymbol{p}_i (t.\boldsymbol{p}_j \times t.\boldsymbol{p}_k) \tag{4.15}$$

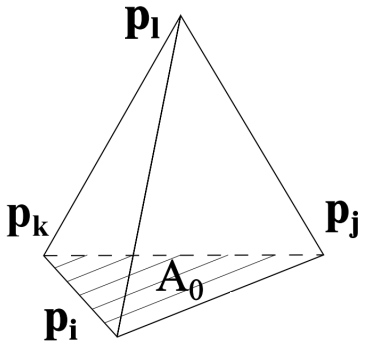


Abb. 4.53: *Tetraeder als diskretes Volumenteil eines deformierbaren Körpers*

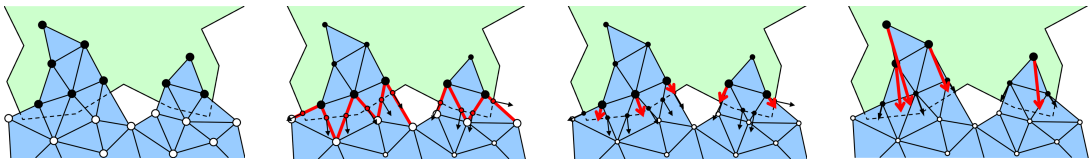
Dieser Ansatz ist für eine Integration deformierbarer Körper in VEROSIM sehr gut geeignet. Durch die Tatsache, dass dieser Ansatz bereits in VEROSIM existiert, ist zum einen kein Implementierungsaufwand vorhanden, und zum anderen die Echtzeitfähigkeit bereits im Vorhinein validiert worden. Deshalb wird für die in VEROSIM weiterführende Integration deformierbarer Körper dieses Modell verwendet.

Neben dem Modell unterteilen sich die Soft Body Dynamics noch in:

1. Modell,
2. Kollisionsdetektion,
   a) Broad Phase,
   b) Narrow Phase,
3. Kollisionsantwort.

In der *Broad Phase* wird dabei anhand der Hüllkörper überprüft, welche Objekte miteinander kollidieren können. Hierzu wird das räumliche Hashing wie in der Starrkörperdynamik (Kap. 2.5.1.8) genutzt. In der *Narrow Phase* wird anschließend die Kollision genauer untersucht.

Zunächst werden alle Punkte in kollidierend/nicht kollidierend unterteilt (Abb. 4.54a), anschließend die Grenz- und Schnittpunkte bestimmt sowie die Normalen zugeordnet (Abb. 4.54b). Die Penetration zunächst der Grenzpunkte (Abb. 4.54c) und anschließend der restlichen Punkte (Abb. 4.54d) wird in Form der Gewichtung ($G(\boldsymbol{x}_i, \boldsymbol{p})$) (wie groß der Einfluss eines nicht kollidierenden Punktes auf einen Grenzpunkt ist), Tiefe ($d(\boldsymbol{p})$)



(a) *(b)* *(c)* *(d)*

Abb. 4.54: *Deformierbare Körper: Narrow Phase, basierend auf [233]*

und Richtung ($\boldsymbol{r}(\boldsymbol{p})$) bestimmt.

$$G(\boldsymbol{x}_i, \boldsymbol{p}) = \frac{1}{\|\boldsymbol{x}_i - \boldsymbol{p}\|^2} \tag{4.16}$$

$$d(\boldsymbol{p}) = \frac{\sum_{i=1}^{k} \Big(G(\boldsymbol{x}_i, \boldsymbol{p})(\boldsymbol{x}_i - \boldsymbol{p})\boldsymbol{n}_i \Big)}{\sum_{i=1}^{k} G(\boldsymbol{x}_i, \boldsymbol{p})} \tag{4.17}$$

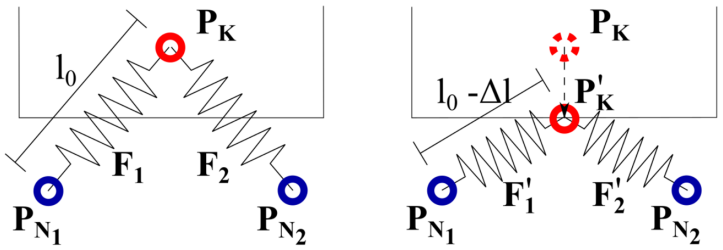
$$\boldsymbol{r'}(\boldsymbol{p}) = \frac{\sum_{i=1}^{k} \Big(G(\boldsymbol{x_i}, \boldsymbol{p})\boldsymbol{n}_i \Big)}{\sum_{i=1}^{k} G(\boldsymbol{x_i}, \boldsymbol{p})} \tag{4.18}$$

$$\boldsymbol{r}(\boldsymbol{p}) = \frac{\boldsymbol{r'}(\boldsymbol{p})}{\|\boldsymbol{r'}(\boldsymbol{p})\|} \tag{4.19}$$

Die Kollisionskraft (Abb. 4.55) wird hier folgendermaßen berechnet:

$$\boldsymbol{F}_{K_{Kol}} = \underbrace{\boldsymbol{F}_K^{'}}_{\boldsymbol{F}_K' = \boldsymbol{F}_1' + \boldsymbol{F}_2'} - \underbrace{\boldsymbol{F}_K}_{\boldsymbol{F}_K = \boldsymbol{F}_1 + \boldsymbol{F}_2} \tag{4.20}$$

$$\text{mit } \boldsymbol{F}_1^{'} = k \cdot ((l - \Delta l) - l_0)(\boldsymbol{P}_K^{'} - \boldsymbol{P}_{N_1})/((l - \Delta l) - l_0) \tag{4.21}$$

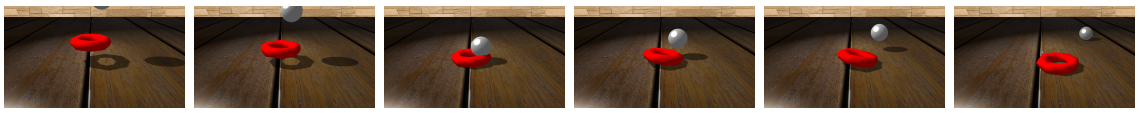


(a) *(b)*

Abb. 4.55: *Deformierbare Körper: Schema der Kollisionskraftberechnung*

Die berechnete Kollisionskraft eines kollidierenden Punktes wird auf den Starrkörper und dessen Grundform umgerechnet. Hiermit ist es nun möglich, sowohl RigidBody-RigidBody, SoftBody-SoftBody als auch RigidBody-SoftBody (sowie SoftBody-RigidBody)

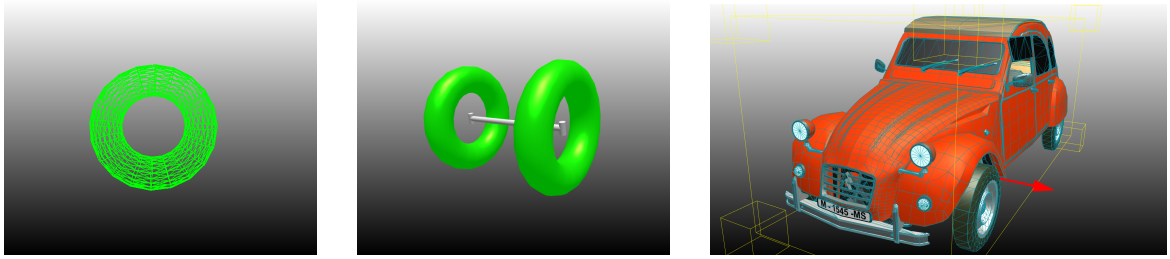
Kollisionen durchzuführen. Exemplarisch ist in Abb. 4.56 der zeitliche Ablauf einer solchen gemischten Kollision zu sehen.



(a) *(b)* *(c)* *(d)* *(e)* *(f)*

Abb. 4.56: *Deformierbare Körper: zeitlicher Ablauf einer Kollision aus Starrkörper-Kugel, deformierbarem SoftBody-Ring und Starrkörper-Tischoberfläche (RigidBody-SoftBody-Kollision)*

Anhand dieser SoftBody-Integration können nun auch deformierbare Reifen im Starrkörpermodell genutzt werden. Auch wenn das beschriebene MFTyre-Modell (aus Kap. 4.3.3.1) hierbei um einiges genauer ist, benötigt Letzteres viele Parameter und liefert in der Simulation am Reifen kein visuelles Feedback. Das heißt, verformbar ist der Reifen nur, wenn er auch in die Starrkörperdynamik integriert ist (siehe Abb. 4.57).



(a) *Reifen* ***(b)*** *Drehachse* ***(c)*** *„Wireframe“-Selektion einer Ente in VEROSIM*

Abb. 4.57: *Beliebige deformierbare Strukturen*

4.3.3.2.2 Beliebige deformierbare Strukturen

Eine der zukünftigen Erweiterungen dieser Soft Body Dynamics stellt ein realistisches Schadensmodell dar.

Inspiriert von dem Soft-Body-Physics-Simulator „Rigs of Rods“ [cg] werden Geometrien nicht mehr starr, sondern durch Federn verbunden. Dadurch ist es möglich, Deformationen darzustellen. Wie in Abb. 4.57c könnten alle Kanten eines Modells durch ein Feder-Dämpfer-Modell (vgl. [234]) eine realistische Verformung nachbilden.

4.4 Implementierung standardisierter Hardwareschnittstellen

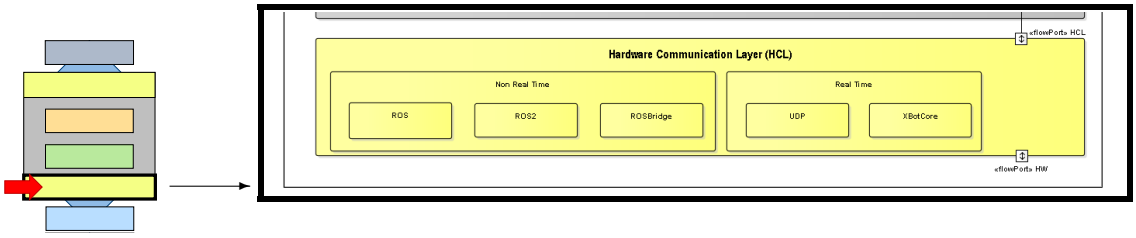


Abb. 4.58: *Konzept: Hardwareschnittstellen (vgl. Abb. 3.4)*

Zur sinnvollen Interaktion mit realen Systemen aus dem VTB heraus – bzw. durch den DT – muss ein ganzheitliches System geschaffen werden, welches insbesondere auch allgemein akzeptierte *Hardwareschnittstellen* mit sich bringt. Im Rahmen robotischer Systeme ist in diesem Fall das Robot Operating System (ROS) [235] die meistgenutzte Middleware und wurde daher auch in Kapitel 4.4.1 inhaltlich betrachtet. Die Integration umfasst nicht nur die Inklusion von ROS-spezifischen Client-Bibliotheken, sondern vor allem sowohl ein plattformunabhängiges Simulationssystem (insb. für Linux und Windows) als auch einen standardisierten Daten- und Modellaustausch.

Als Endergebnis entsteht dabei ein System aus Modellen, Algorithmen, Hardwaremodulen sowie internen und externen Softwareframeworks, in dem der DT als universelle Schnittstelle das „Herzstück“ dieses Systems bildet. Die spezifische Umsetzung dieses universellen Interfaces wird in Kapitel 4.4.2 detailliert beschrieben, in dem der gesamte Entwicklungsprozess von der Idee über Modell, Umweltgenerierung, Interaktion und direkte Kommunikation betrachtet wird.

4.4.1 Interface zum Robot Operating System (ROS)

Wie in Kap. 2.4.4.3 beschrieben, ist ROS weit mehr als nur ein Kommunikationsprotokoll cyber-physischer Systeme, jedoch soll im Folgenden eben diese zentrale Eigenschaft der Middleware – die Nachrichtenübertragung (das „message passing“) – umgesetzt werden.

4.4.1.1 Anforderungen an das ROS-Framework

Um dem in Kap. 3 eingeführten Konzept gerecht zu werden, muss ein entsprechendes Framework in der Lage sein, sich während der Laufzeit flexibel an das Modell und damit an die Simulation anzupassen. Während des Modellierungsschrittes ergibt sich daraus die notwendige Bedingung, dass das Modell jederzeit geändert werden kann. Dadurch folgen Ansprüche an die Abbildung der verschiedenen Datentypen aus VEROSIM auf die Datentypen des ROS, die mit einer Änderung des Simulationsmodells verbunden sind.

Eine grundlegende Anforderung dabei ist, dass VEROSIM (beziehungsweise Teilkomponenten davon) zusammen mit ROS unter einer Plattform laufen müssen. ROS wird in

erster Linie im Bereich der Hard- und Softwareentwicklung robotischer Systeme unter Linux genutzt, wohingegen VEROSIM sich auch an Anwender unterschiedlicher Disziplinen richtet und primär als Modellierungs- und Simulationstool unter Windows genutzt wird. Da der volle Funktionsumfang von ROS nur unter Linux nutzbar ist, ist eine Aufgabe im Rahmen dieser Arbeit, VEROSIM auch unter Linux verwenden zu können. Da ROS und VEROSIM kein gemeinsames, plattformübergreifendes Austauschformat besitzen, muss stattdessen auf die bereitgestellten APIs zurückgegriffen werden.

Weiterhin muss das Framework in der Lage sein, die ROS-Elemente innerhalb von VEROSIM darzustellen, sodass der Nutzer mit ihnen interagieren kann. Das erfordert neben der Einbindung in die VSD eine graphische Repräsentation innerhalb der Simulationsumgebung. Für eine leichte Bedienung ist das bestehende Benutzungskonzept der Simulation beizubehalten. Die Nutzung des Frameworks soll sich für den Nutzer nativ und intuitiv anfühlen.

Ergänzend zu den bisherigen Anforderungen an das Framework muss es modular aufgebaut sein. So kann es an unterschiedliche Szenarien angepasst und bei Bedarf erweitert werden.

Ein wichtiges Ziel des Frameworks ist, den Tätigkeitsschwerpunkt des Entwicklers von der Programmierung auf die Modellierung zu verschieben. Weitere zu berücksichtigende Faktoren, die das Konzept des Frameworks prägen, sind die sich aus dem verwendeten Systemen VEROSIM und ROS ergebenen Randbedingungen. Mit Randbedingungen sind hier die Faktoren gemeint, welche die Erstellung des Frameworks maßgeblich beeinflussen. Dazu gehören neben dem eingesetzten Betriebssystem, unter dem die Softwaresysteme laufen, die verwendeten Programmiersprachen und die damit verbundenen Eigenheiten.

4.4.1.2 Integrationsvorgehen

Insgesamt ergeben sich drei zentrale Anforderungen an die ROS-Schnittstelle:

1. Kohärenz der Datentypen,
2. einheitliches Schnittstellenkonzept
   - flexible Kommunikation,
   - Änderungen zur Laufzeit ermöglichen,
   - Plattformunabhängigkeit,
3. schneller, automatisierter, konsistenter Modell- und Dateiaustausch.

In Abb. 4.59 ist eine Übersicht über die zentralen Aspekte der ROS-Integration in ein 3D-Simulationssystem dargestellt.

Der generelle Workflow besteht daraus, aus der ROS-basierten „message type definition“ eine passende Extension in der Simulation (im Sinne eines Subscriber oder eines Publishers) zu kreieren, der Umwandlung der Datentypen und schlussendlich der Weitergabe der Ergebnisse mittels des I/O Boards. Auf Basis eines einheitlichen Modells kann daraufhin definiert werden, welche Extensions genutzt werden müssen.

4.4.1.2.1 Kohärenz der Datentypen

Die *Menge A* ist die Menge aller VEROSIM-I/O-Datentypen, wobei x einen dieser Datentypen repräsentiert und B die Menge aller ROS-Nachrichtentypen ist und y eine konkrete

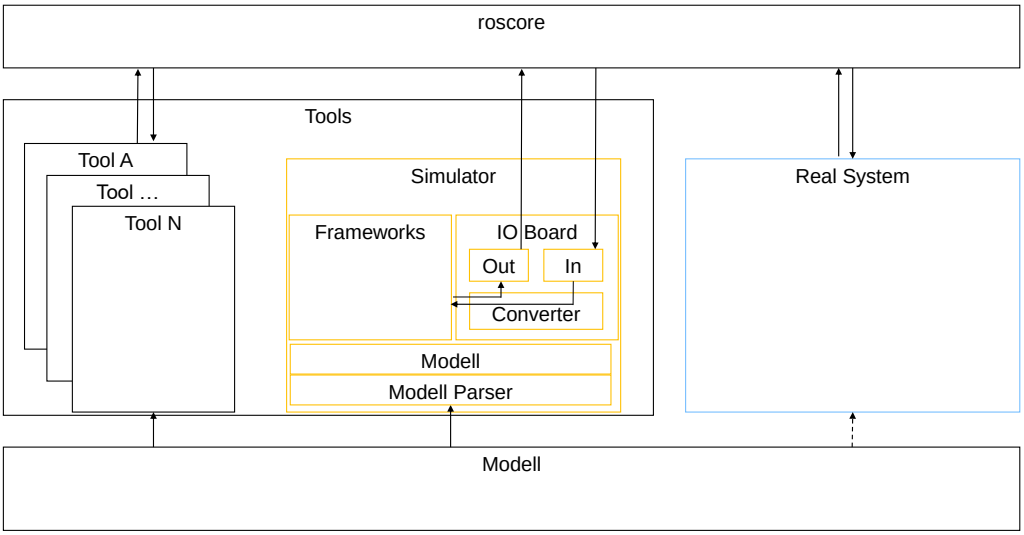


Abb. 4.59: *Bestandteile der ROS-Schnittstelle: Der Simulator als ein Tool mit Modellimporter, interner I/O-Board-Verschaltung und externer ROS-Kommunikationsmöglichkeit*

ROS-Nachricht darstellt. Die Menge der ROS-Nachrichten ist dabei größer als die der VEROSIM-I/O-Typen:

$$B > A$$

Die Aufgabe besteht darin, eine Abbildung f zu finden, welche die *Inputs* des I/O-Frameworks auf ROS-Nachrichten abbildet:

$$f\colon A \to B,\ \forall\, x \mapsto y$$

Um die *Outputs* des I/O-Frameworks mit den ROS-Nachrichten zu verknüpfen, benötigt man eine inverse Abbildung g, analog zu der Input-Abbildung:

$$f^{-1} \equiv g\colon B \to A,\ \forall\, y \mapsto x$$

Da jedem VEROSIM-I/O-Datentyp mindestens ein Äquivalent aus der Menge der ROS-Nachrichten zugewiesen werden kann, handelt es sich hier um eine surjektive, totale Abbildung. Anders herum können nicht alle ROS-Nachrichtentypen genau einem I/O-Datentyp zugewiesen werden, sondern müssen durch eine Kombination von Abbildungen zusammengesetzt werden. Die Abbildung der Datentypen kann dann (1) manuell, (2) Template-basiert oder (3) dynamisch erfolgen.

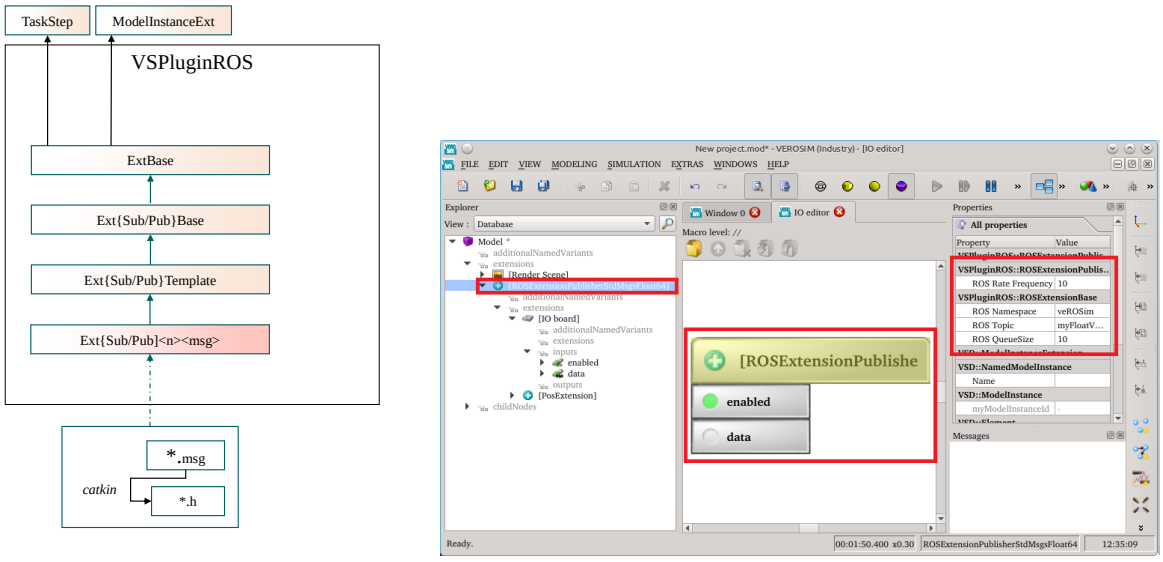
Die Implementierungsarbeit der ROS-Schnittstelle im Simulator (als kontinuierliche Weiterentwicklung von [8]) unterteilt sich in die (a) Templatisierung und Automatisierung der Nachrichtenübermittlung, (b) Entkopplung dieser vom umgebenden Betriebssystem, (c) Erweiterung auf laufzeitunabhängige Implementierungsansätze, (d) Erweite-

rung auf ROS2 sowie (e) Interkommunikationsstruktur der unterschiedlichen Systeme.

4.4.1.3 Templatisierung und Automatisierung der Nachrichtenübermittlung

Wie eingangs erwähnt, ist eine 1:1-Abbildung der ROS-Nachrichtentypen als eigene VEROSIM-Extensions sehr sinnvoll. Jeder neue Nachrichtentyp besteht aus einem Tupel aus „Message Definition File“ und einem automatisch generierten C++-Header. Darauf aufbauend kann man auf die Datentypen der Nachricht zugreifen.

In Abb. 4.60a ist die generelle templatebasierte Struktur des ROS-Plugins zu sehen. Alle Gemeinsamkeiten aus Publisher und Subscriber wie z. B. die Properties „namespace“, „topic“, „queuesize“ werden in eine Basisklasse (`ROSExtensionBase`) ausgelagert (siehe Abb. 4.60b). Spezifische Eigenschaften für Publisher und Subscriber werden in der jeweiligen `Pub/SubBase` bearbeitet (wie z. B. die Publisherfrequenz). Um mit dem ROS-Master zu kommunizieren, benötigen Publisher und Subscriber ein *NodeHandle*. Im Gegensatz zu Publishern, die im gegebenen Takt Nachrichten publizieren, benötigen Subscriber einen *Callback*, welcher ein Update der Daten pollt. Mit Hilfe von Publisher- und Subscriber-Templateklassen lassen sich dann beliebige neue Nachrichtentypen als Extensions hinzufügen, in denen nur noch die Datentypenabbildung und möglicherweise weitere Post-Processing-Schritte geschehen müssen.



(a) *Pseudo-UML-Diagramm (vgl. Abb. 4.62)*

(b) *I/O Board, Explorer, Properties*

Abb. 4.60: *ROS-Schnittstelle: Plugin-System*

Implementiert sind u. a. *std_msgs*, *sensor_msgs*, *nav_msgs* usw., wobei zusätzlich auch projektspezifische Nachrichtentypen integriert wurden, welche in Kap. 4.4.2 beschrieben werden.

4.4.1.4 Entkopplung vom umgebenden Betriebssystem und Drittanbieteranwendungen

Um alle ROS-Funktionalitäten von VEROSIM-Funktionalitäten zu entkoppeln, wird zusätzlich zum Plugin noch eine Lib implementiert. Verbunden durch ein „Abstract Factory Pattern“ [236] wird eine `VSLibROSExtension` in VEROSIM genutzt, diese sucht sich eine passende sogenannte Engine im `VSPluginROS`. Wenn es eine solche Engine gibt – man also unter Linux arbeitet und ROS vorhanden ist – werden die Daten von ROS über das Plugin an die Lib weitergegeben. Dieses Weitergeben erfolgt mit einem sogenannten „VEROSIM DynStruct“ (`VSDS`), welches eine Abbildung der ROS-Nachrichtentypen auf ein dynamisches Struct darstellt. Als Hash von QVariants dient dies als dynamisches, hierarchisches, universelles Austauschmedium. Analog zur MDL des jeweiligen Nachrichtentyps werden die Daten in Standard-, Qt- oder VEROSIM-Datentypen umgewandelt.

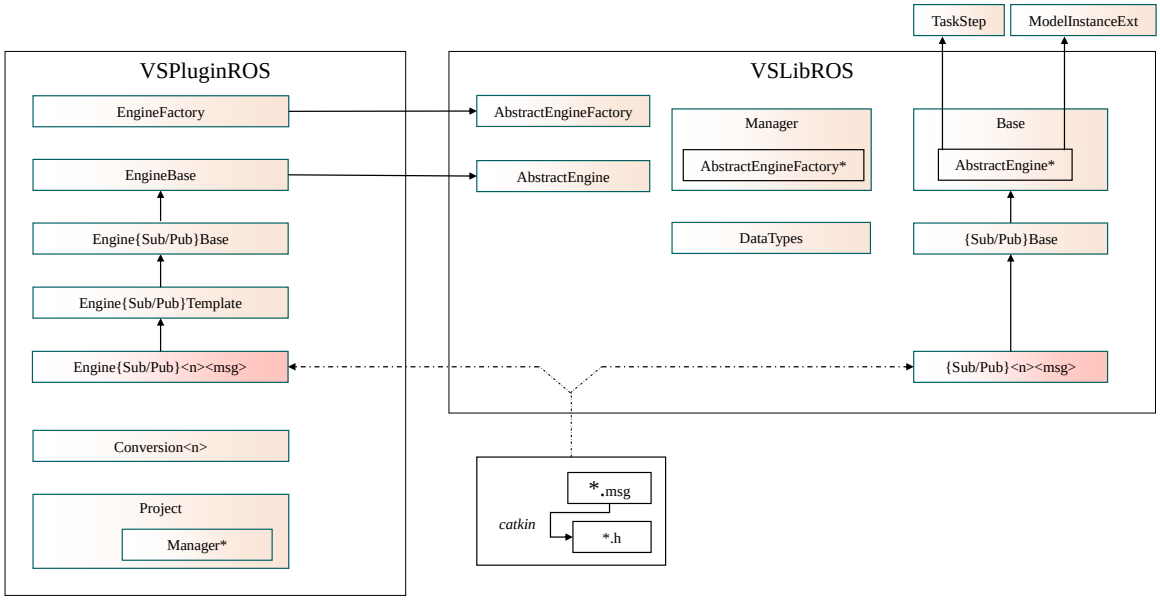


Abb. 4.61: *Lib- und Plugin-System der ROS-Schnittstelle: Pseudo-UML-Diagramm von Lib + Plugin + VSDS (vgl. Abb. 4.62)*

Durch die Entkopplung der zwei Welten (ROS und VEROSIM) ergibt sich der Gesamtfluss von:

$$\underbrace{\text{ROS} \xrightarrow{ROSMsg} \text{ROSEngine}}_{ROS-World} \xrightarrow{DynStruct} \underbrace{\text{BaseExtension} \xrightarrow{IO} \ldots}_{VEROSIM-World}$$

Dies hat den inhärenten Vorteil, dass auch Drittanbieterbibliotheken wie die OpenCV (Open Source Computer Vision Library) oder die PCL (Point Cloud Library) von ROS und VEROSIM in unterschiedlichen Versionen nutzen können. Das Plugin verwendet in dem Fall die gleiche Version wie ROS, die Lib wie VEROSIM. Zusätzlich hat dies den

Vorteil, dass alle Funktionen auch unter Windows funktionieren, auch wenn keine ROS-Engine vorhanden ist.

Dieses Konzept kann in Zukunft erweitert werden, um andere Engines (wie z. B. in dieser Arbeit die ROS2-Engine (vgl. Kap. 4.4.1.6) oder im Allgemeinen andere Middlewares (siehe Kap. 2.4.4)) mit der gleichen Struktur anzubinden.

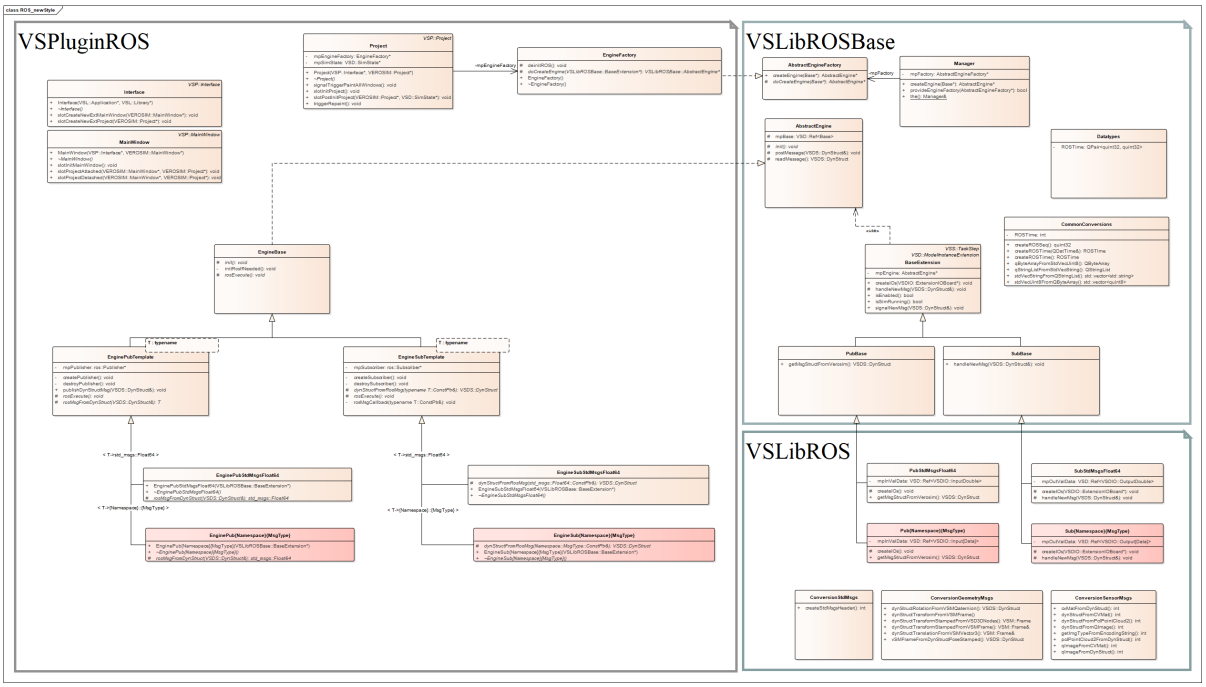
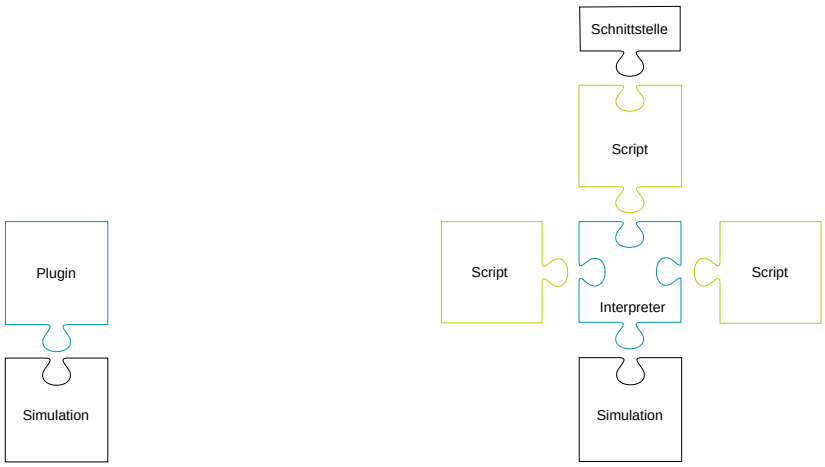


Abb. 4.62: *Lib- und Plugin-System der ROS-Schnittstelle: UML-Diagramm von VSLibROS, VSLibROSBase und VSPluginROS*

4.4.1.5 Laufzeitunabhängige Erweiterung des Frameworks

Über das dynamische Laden von Dynamic Linked Libraries (Dlls) ist es bereits möglich, verschiedene Module wie Puzzleteile in die 3D-Simulation einzubinden (vgl. Abb. 4.63). Jedoch muss dafür immer eine vordefinierte Anbindung zwischen 3D-Simulation und Dll bestehen. Außerdem bedarf es einer Entwicklungsumgebung, welche es erlaubt, das benötigte Modul zu kompilieren. Damit die 3D-Simulation auf verschiedene, teilweise neue Schnittstellen vollständig dynamisch zugreifen kann, ohne auf eine Entwicklungsumgebung zurückgreifen zu müssen, bedarf es eines neuen Ansatzes.

Als eine Lösungsmöglichkeit kann man eine weitere Programmiersprache in die 3D-Simulation integrieren, die es erlaubt, Änderungen am Code zur Laufzeit vorzunehmen. Als Programmiersprache mit Laufzeitänderungsmöglichkeiten bieten sich vor allem Interpretersprachen an. Diese brauchen zum Ausführen, im Gegensatz zu Just-in-time-Kompilierung, keine Vorlaufzeit und es ist nicht erforderlich, einen Compiler in die 3D-Simulation einzubetten. Als Interpretersprache wurde Python [ch] [ci] gewählt, da ROS auf dieser basiert und mit Paketen wie NumPy [cj], SciPy [ck] oder scikit-learn [cl]



(a) Aktuell *(b) Erweiterung*

Abb. 4.63: *Modulare Erweiterung des ROS-Schnittstellen-Konzeptes vorhandener (schwarz), laufzeitkonstanter (türkis) und laufzeitvariabler (grün) Bausteine.*

weitere nützliche Funktionen im Bereich der Algebra, Lösung von Gleichungssystemen, diskreten Optimierungsproblemen, numerischer Integration bis hin zu Machine Learning automatisch hinzukommen.

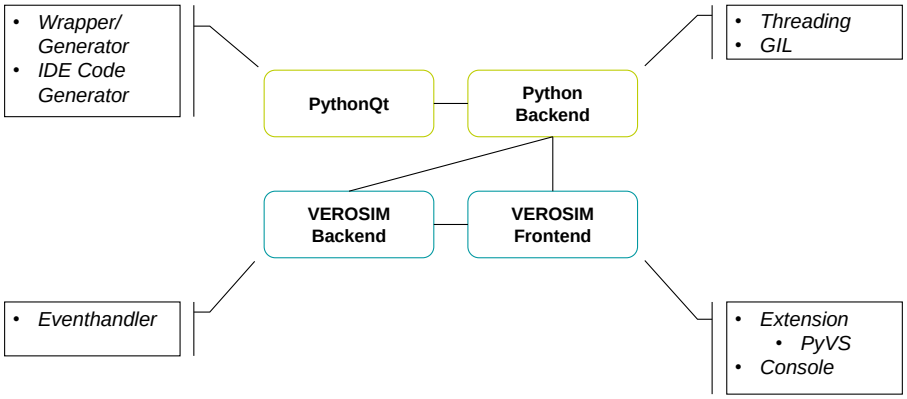
Das Gesamtkonzept ist in Abb. 4.64 dargestellt. Basierend auf PythonQt werden *Generater* geschrieben, die alle notwendigen VEROSIM-Funktionalitäten in Python zugänglich machen sowie ein *Wrapper*, der Python-Funktionen mit Qt-Signalen und -Slots verbinden kann. Zusätzlich werden im *Python-Backend* das Threading und der CPython Global Interpreter Lock (GIL) [237] behandelt. VEROSIM-seitig übernimmt dann das *VEROSIM-Backend* insbesondere das Eventhandling. Somit hat der Nutzer die Möglichkeit, im *VEROSIM-Frontend* entweder mittels einer Extension oder innerhalb einer Python-Konsole zur Laufzeit Python-Code wie aus der *rospy*-Bibliothek auszuführen und zusätzlich auf interne Parameter zuzugreifen.

Im Gegensatz zu Ansätzen wie Simplified Wrapper and Interface Generator (SWIG) [cm], Boost.Python [cn], PySide [co] und PyQt [cp] ist PythonQt [cq] als Implementierungsansatz besonders gut geeignet, um in eine größere C++/Qt-basierte Applikation eingebunden zu werden und wurde deshalb eingesetzt (siehe auch Tab. 4.3).

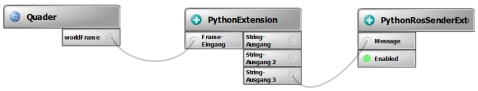
Tab. 4.3: *ROS-Schnittstelle: Bewertung von Python-Integrationskonzepten*

Voraussetzungen	Python API	SWIG	Boost.Python	PySide & PyQt	PythonQt
C++	✗	✓	✓	✓	✓
Qt	✗	✗	✗	✓	✓
Datentypen	✗	✓	✓	✓	✓
Einbindung	✗	✗	✗	✗	✓

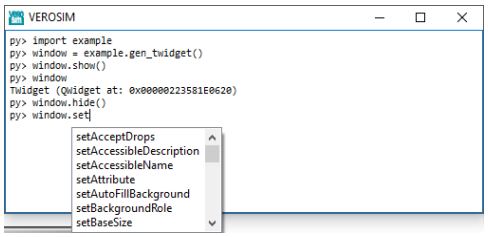
Als Ergebnis entsteht also eine Kette aus VEROSIM-Frontend, VEROSIM-Backend,



(a) Python-Konzept: PythonQt Wrapper, Generator und Backend, welches mit dem VEROSIM-Backend Eventhandling und Threading übernimmt, wodurch im Frontend Extensions oder eine Konsole genutzt werden können.



(b) VEROSIM-Extension: PythonExtension, welche NumPy nutzt, wie PythonRosSenderExt, welcher den z-Wert der Quaderposition publiziert.



(c) VEROSIM-Konsole: direkte Eingabemöglichkeit von Python-Code in der C++-Umgebung mit Autovervollständigung

Abb. 4.64: *ROS-Schnittellenerweiterung: dynamische Python-Implementierung*

Python-Backend und PythonQt, die sich wie folgt zusammenfügt (vgl. auch Abb. 4.64):

VEROSIM → `VSPluginPython` → PythonQt → `PyVS` → *Python Script* → ROS,

wobei `PyVS` die Methodenregistrierung, den Threading-Zugriff sowie den Generator-Code zur einfacheren Autovervollständigung enthält und zusätzlich im ROS-Kontext das dynamische Konvertieren von VEROSIM zu ROS-Datentypen (und umgekehrt) übernimmt.

Während ROS hauptsächlich für Linux entwickelt wird, ist es möglich, die für Python vorhandene Anbindung ebenfalls unter Windows zu nutzen. Hierzu bedarf es jedoch eines Linux-Systems, auf welchem der *roscore* läuft. Auf diesem wird der nötige Python-Code aus den Definitionsdateien erzeugt, die die in ROS versendeten Nachrichten definieren [238]. Daraufhin können zur Laufzeit neue Python-basierte ROS-Knoten innerhalb der Simulation erzeugt und im ROS-Netzwerk genutzt werden.

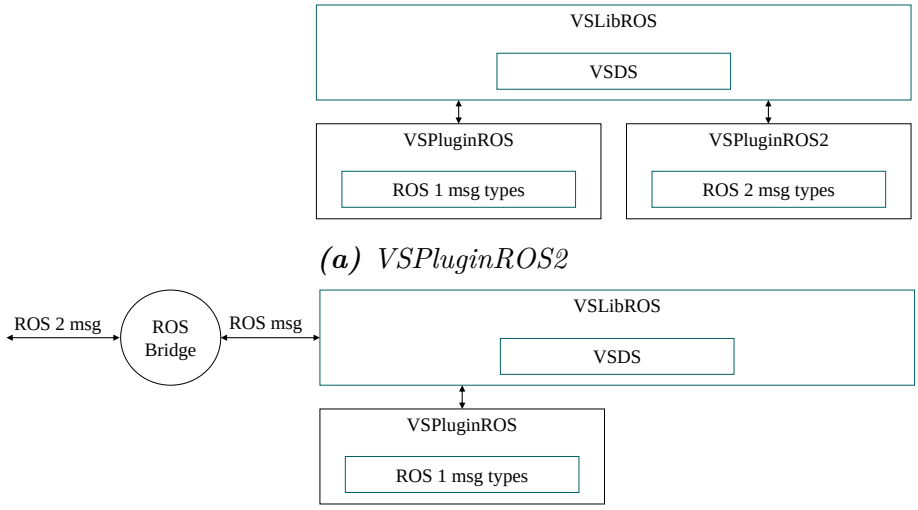
4.4.1.6 Erweiterung des Frameworks um ROS2

Wie auch bei der eigentlichen ROS-Entwicklung wird das VEROSIM-ROS-Interface an sich nicht verändert, bleibt uneingeschränkt weiter existent und voll funktionsfähig. Um dies auch auf ROS2 zu erweitern, sind zwei Ansätze sinnvoll:

1. eine Integration von ROS2 in VEROSIM analog zur ROS-Integration,
2. Nutzung einer sogenannten „Bridge“(`ros1_bridge`), die zwischen ROS und ROS2 vermittelt [cr].

Um bisherige Entwicklungen auch in ROS2 nutzen zu können, aber auch die Performanz einer Komplettintegration zu sichern, werden beide Ansätze umgesetzt. Vollständig analog zu Abb. 4.65a wird eine neue Engine hinzugefügt. Aufbauend auf dem standardisierten DynStruct können so viele Funktionen, die für ROS-Nachrichtentypen implementiert wurden, auch für ROS2 genutzt werden. Für solche, die eine neue Signatur haben, müssen – wie auch bei ROS – sowohl Plugin als auch Lib neu angelegt werden.

Dadurch, dass ROS nur unter Linux läuft, wird die Bridge primär genutzt, um Daten von ROS unter Linux zu ROS2 unter Windows zu transportieren (siehe Abb. 4.65b). Diese Bridge ist ein eigenständiger Knoten, der alle Nachrichten umwandelt. Dies führt jedoch zu einem erheblichen Overhead, einer erhöhten Latenz und somit einer reduzierten Performance des Gesamtsystems [239].



(a) *VSPluginROS2*

(b) *ROS-Bridge*

Abb. 4.65: *ROS2-Integrationskonzepte (vgl. auch Abb. 4.62)*

Neben der Möglichkeit, dies unter Linux und Windows (und auch MacOS) zu nutzen, muss vor allem die Quality-of-Service-(QOS-)Einstellmöglichkeit vorhanden sein. Hierbei kann bei einzelnen Nachrichten umgeschaltet werden, je nachdem, ob ein Verhalten wie UDP („Best effort“ – wenn ein Paket verloren geht, wird einfach das nächste genommen) oder TCP („Reliable“ – verlorene Pakete werden nochmals gesendet) eingehalten werden soll.

4.4.1.7 Modularität der ROS-Schnittstelle

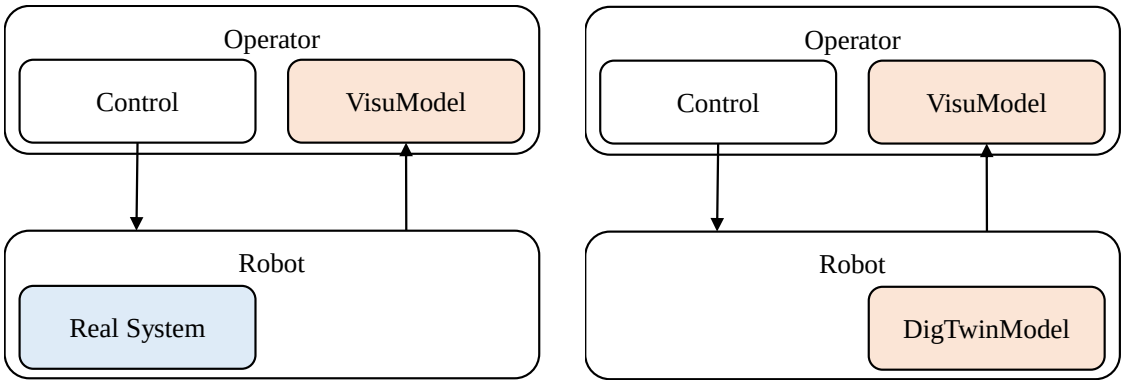
Insgesamt bietet die ROS-Schnittstelle den Zugang zum gesamten Netzwerk aus vorhandenen und zukünftigen Entwicklungen der ROS-Community. Zusätzlich ist mit dem ROS2-Interface eine Möglichkeit geschaffen worden, zukunftssicher beide Versionen von ROS zu unterstützen. Insbesondere ist in diesem Zusammenhang die Plattformunabhängigkeit herauszustellen. Während ROS nur unter Linux verwendet werden kann, ist mit dem ROSPython Framework nun die direkte Interaktion mit einem ROS-Netzwerk von Windows aus (und sogar Änderungen zur Laufzeit) möglich. Durch die Erweiterung auf ROS2 lässt sich das ROS2-Interface nativ auf Windows nutzen. Außerdem kann das ROS-Interface unter Linux mit Hilfe der ROS-Bridge auch mit ROS2 kommunizieren. Somit bietet die ROS-Integration mit (i) ROS, (ii) ROSPython, (iii (a)) ROS2 und (iii (b)) ROSBridge eine ganzheitliche Abdeckung verschiedener Plattformen.

4.4.2 Der Digitale Zwilling als universelles Interface

Wie in den vorherigen Kapiteln ausgeführt, ist ein umfassendes ROS-Interface implementiert. Dies beinhaltet das Einbinden von ROS unter Linux (C++-basiert) sowie ROS unter Windows (C++- und Python-basiert) genauso wie den kompletten Modellierungsworkflow bzgl. URDF-Dateien (vgl. Kap. 4.3.1.2). Aus diesem Workflow entstehen die folgenden Modi, analog zu den präsentierten URDF-Modellen (vgl. Abb. 4.34):

1. Visualisierung (*VisuModel*)
2. Digitaler Zwilling (*DigTwinModel*)

Diese Modelltypen sind schematisch in Abb. 4.66 dargestellt.



(a) *Direkte Steuerung des realen Systems* ***(b)*** *Direkte Steuerung des DTs*

Abb. 4.66: *Das Modell des Digitalen Zwillings unterschiedlicher Komplexität: VisuModel (reine Datensenke und Visualisierungsmodell) und DigTwinModel (Datenquelle für Sensordaten sowie Datensenke für Steuerungsbefehle)*

Das *VisuModel* visualisiert alle Rohdaten sowie alle verarbeiteten Daten, die aus dem komplexen System herauskommen. Hierbei steht vor allem die geeignete Aufbereitung dieser Daten zur Visualisierung im Vordergrund. Das *DigTwinModel* dagegen ist der Digitale Zwilling des realen Assets und muss sich nach außen so verhalten wie das reale

System. Das heißt auf der einen Seite, dass es mit denselben Eingabegeräten gesteuert werden kann, und auf der anderen Seite auch Daten produziert, die von weiteren „Processing Modules“ verarbeitet werden können, wie die Daten des realen Systems.

4.4.2.1 Ausnutzen von ROS als Middleware

Zentraler Nutzen von ROS ist die Nachrichtenübertragung („message passing system“) und somit die Integration von Publishern und Subscribern sowie die Verbindung mit dem *roscore*. Der im vorherigen Kapitel schon ausführlich thematisierte Modellierungsworkflow führt dazu, dass automatisiert Änderungen in den URDF-Modellen in ein VEROSIM-Modell überführt werden können, in dem dann alle weiteren Schritte sowie die Nutzung des Digitalen Zwillings erfolgen. Zunächst werden spezielle Nachrichtentypen, ihre Funktionsweise und ihr Nutzen vorgestellt.

4.4.2.1.1 Simulationszeit

Die Zeit und das Timing stellen sowohl im echten Leben als auch in der Programmierwelt immer gewisse Probleme dar. Normalerweise kann man sich auf die sogenannte „wall-time“, also die Zeit, die die Wanduhr im Labor anzeigt, einigen und als globales Maß bestimmen. In Simulationen, verteilten Systemen o. Ä. ist es aber oft sinnvoll bzw. nötig, eine simulierte Uhr zu beschleunigen, zu verlangsamen oder definierte Zeitschritte durchzugehen. Die interne Simulationszeit wird daher als „simulation time“ definiert und erlaubt genau die zuvor genannten Schritte. In ROS ist es möglich, diese Simulationszeit mit Hilfe der `rosgraph_msgs/clock` topic zu publishen oder zu subscriben. Eine sinnvolle selbstdefinierte Simulationszeit hat als einziges Kriterium, dass sie streng monoton steigend sein muss. Somit ist gewährleistet, jeden Zeitpunkt aller Nutzer mit der Simulationszeit zu synchronisieren. In Echtzeitsystemen (wie z. B. QNX) bringt dies natürlich weitere Herausforderungen mit sich, die aber nicht im Rahmen dieser Arbeit behandelt werden.

4.4.2.1.2 Robotersteuerung

Zur Kontrolle eines realen Roboters ist es ratsam, Standarddatentypen zu nutzen. Hierbei wird der Soll/Ist-Vergleich der einzelnen Motorstellwerte oft mit Hilfe der JointState msg überprüft, wohingegen die Befehle auf Motorebene einzeln (mit vielen Joint Commands) gesendet werden. Das heißt beispielsweise, es gibt N Joint Commands für N Joints, wohingegen es nur 1 JointState msg gibt, welche rückkoppelt, wie der aktuelle Status aller Joints ist. Diese kann dann direkt für das VisuModel genutzt werden. Für den Digitaler-Zwilling-Modus muss es daher, analog zum realen System, N Float64 Subscriber (für die N Joint Commands) und 1 JointState Publisher geben.

- `std_msgs/{Bool/Float64/Int32/String/*}`
- `sensor_msgs/JointState`

4.4.2.1.3 TF (Transform)

Das tf package [cs] ermöglicht die Überwachung mehrerer Koordinatensysteme (Frames)

über die Zeit. Eine detaillierte Übersicht zur „transform library“ wird in [240] gegeben. Der mit einem Szenegraphen gleichzusetzende „tf tree“ repräsentiert in einer Baumstruktur die verschiedenen Frames und deren Verbindung. Somit sind alle Knoten abhängig vom Koordinatensystem des jeweiligen Parent-Frames.

- `tf2_msgs/TFMessage`
- `geometry_msgs/TransformStamped`

Für den TF-Subscriber wird beim ersten Start das aktuelle Modell geparst, um einen Hash aus einzelnen Frames und ihren Namen herzustellen. Die TF-Nachricht beinhaltet einen Vektor von Transformationen (TransformStamped), welche stringbasiert Transformationen von Parent- zu Child-Nodes beinhalten. Diese Parent- und Child-Nodes müssen gefunden und die Transformation angewendet werden. Auf Basis des erstellten Hashes können dann in jedem Takt die einzelnen VSD-Transformationen aktualisiert werden.

4.4.2.1.4 Sensorik und Visualisierungen

Die Visualisierung hängt sehr stark mit der Wahl des Sensorkopfes des realen Roboters zusammen. Allgemein lässt sich sagen, dass mit den folgenden Sensor-Nachrichtentypen viele aktuelle Sensorköpfe abgedeckt werden können:

- `sensor_msgs/PointCloud2` [ct] [cu]
- `sensor_msgs/Image`
- `sensor_msgs/CompressedImage`
- `sensor_msgs/CameraInfo`
- `sensor_msgs/LaserScan`

Wie zuvor erwähnt, gehören Sensoren nur im Rahmen ihrer zentralen Frames mit in das URDF-Modell und werden anschließend für das DigTwinModel per Skript aus Untermodellen hinzugefügt und per I/O Board mit den jeweiligen Visualisierungen verbunden.

4.4.2.1.5 Eigene spezialisierte ROS-Extensions

Im Rahmen von Projekten können innerhalb dieses Frameworks auf Basis des Tupels aus msg-Definition und -Header beliebige spezialisierte Nachrichtentypen eingebunden werden. Hierfür sind aus dem CENTAURO-Projekt [cv] (vgl. auch Kap. 5.1) folgende Beispiele anzuführen:

- `centauro_msgs/SingleDataMap`
- `centauro_msgs/ModelPose`
- `centauro_msgs/LocomotionGoal`
- `centauro_msgs/LocomotionPose`
- `centauro_msgs/LocomotionPlan`
- `centauro_msgs/TexturedPolygonMesh`
- `centauro_msgs/XCMJointState`
- `centauro_msgs/XCMJointCommand`

4.4.2.1.6 Zusammenfassung

Es wurde ein ganzheitliches System entwickelt, welches ROS-Input nutzen kann, um sinnvolle Visualisierungen zu erstellen, aber auch unter Ausnutzung aller internen Funktionalitäten ROS-Output erstellen kann. Diese Modularität führt dazu, dass ROS nicht nur für die hier beschriebene Sensordatenumwandlung zur Verfügung steht, sondern vielmehr eine allgemein gültige Schnittstelle darstellt. Hierzu muss die passende Extension (mit dem passenden ROS-Nachrichtentyp) hinzugefügt und per I/O Board modular zu verbunden werden.

4.4.2.2 Weitere Schnittstellen

Durch ROS als Schnittstelle zu robotischen Systemen und Sensoren in Kombination mit internen Funktionalitäten von VEROSIM sowie der Möglichkeit, mittels Python weitere Bibliotheken wie Numpy/CyPy hinzuzuziehen, ist der DT durch das VTB sehr gut vernetzt. Zusätzlich ist die Möglichkeit geschaffen, plattformübergreifend unter Linux und Windows arbeiten zu können.

4.4.2.2.1 User Datagram Protocol (UDP)

Gerade im Bereich der spezialisierten haptischen Eingabegeräte (vgl. Kap. 4.2.4 sowie Abb. 4.16) muss eine Kommunikationsfrequenz von $f \leq 1$ kHz erreicht werden. Hierzu wird ein generelles UDP-basiertes Interface entwickelt, welches so auf Threads ausgelagert wird, dass eine solche Updaterate eingehalten werden kann.

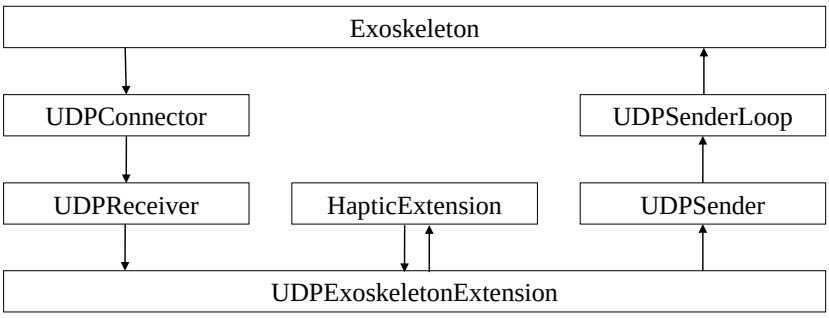


Abb. 4.67: *UDP-Schnittstelle am Beispiel des Exoskeletts*

Im Anwendungsfall des Exoskeletts werden drei Aspekte abgedeckt. Zum einen die Verflechtung mit dem haptischen Rendering, zum anderen das Empfangen von EDDS sowie auch das Senden von ECDS. Zum Senden dieser ECDS wird ein `UDPSender` entwickelt, zum Empfangen der EDDS ein `UDPReceiver`. Diese zwei werden zusammen mit der `HapticExtension` in einer kombinierten `ExoskeletonExtension` referenziert. In diesem Kapitel geht es aber um die reine UDP-Verbindung, die gemäß Abb. 4.67 modular und auch losgelöst vom Exoskelett genutzt werden kann. Im Falle des UDPReceivers wird ein `UDPConnector` implementiert, welcher die UDP-Verbindung herstellt und bei einem Update mittels Qt-Signalen diese an den Receiver weitergibt. Beim UDPSender wird dagegen ein `UDPSenderLoop` implementiert. Dieser kann vom UDPSender ein- bzw.

ausgeschaltet werden und kann mittels einer Setter-Funktion der zu sendende Paketinhalt geändert werden. Der SenderLoop stellt derweil eine stetige UDP-Verbindung her und sendet UDP-Pakete in einem separaten Thread mittels `QTimer::singleShot()`.

4.4.2.2.2 XBotCore

Für den Fall, dass robotische Systeme ein eigenes Betriebssystem nutzen, kann es auch möglich sein, dieses als eigenständige Bibliothek im VTB abzubilden bzw. einzubinden. Im Rahmen des CENTAURO-Projektes wurde hier beispielsweise XBotCore [241] genutzt. Die XBotCore-Bibliothek lässt den Nutzer mittels Dependency Injection neue Features zum System hinzufügen. Insbesondere kann man den realen Roboter durch einen simulierten ersetzen, wenn man die Interfaces von XBotCore implementiert: `XBot::IXBotJoint`, `XBot::IXBotIMU`, `XBot::IXBotFT`. Mittels dieser Interfaces ist es möglich, ein 1:1-Mapping zwischen Joints, Links, Force-Torque-(FT-)Sensoren und Inertial-Mess-Einheiten (IMU) anzubieten. Exemplarisch ist in Abb. 4.68 zu sehen, wie der *Communication Handler* von XBotCore (rechts) mit dem im linken XBotCore-integrierten VEROSIM per ROS kommuniziert und im zweiten VEROSIM (Mitte) der vom ersten VEROSIM publizierte TF-Baum simpel visualisiert werden kann. Hierbei wird eine implementierte Funktion genutzt werden, um schnell einen dreidimensionalen Überblick über den TF-Baum zu erhalten. Diese Funktion generiert an jedem Frame eine Box mit unterschiedlicher Seitenlänge in x-, y- und z-Richtung, sodass eindeutig die Frame-Orientierung visualisiert werden kann.

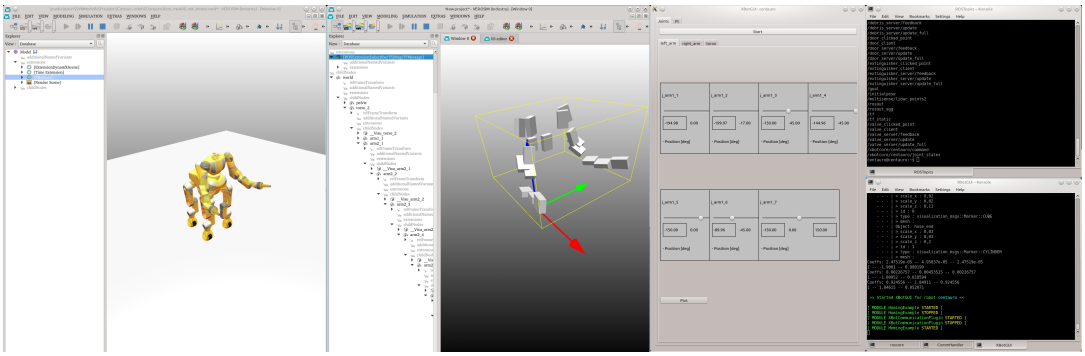
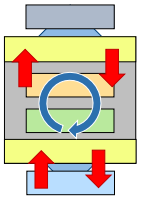


Abb. 4.68: *XBotCore-Schnittstelle: Kommunikationsinfrastruktur aus XBotCore Communication Handler, einer XBotCore-GUI zur selektiven Gelenkwertmodifikation, einem VEROSIM zur TF-Baum-Visualisierung (VisuModel) sowie einem VEROSIM mit Digitalem Zwilling des CENTAURO-Roboters (DigTwinModel)*

4.5 Kopplung von Digitalem und Realem Zwilling zum Mediator



(a)

Abb. 4.69: *Konzept: Live Support (vgl. Abb. 3.4)*

Auf Basis der in Kap. 4.2 entwickelten Nutzerschnittstellen, den neuen integrierten Funktionalitäten in Kap. 4.3 sowie den Hardwareschnittstellen in Kap. 4.4 ist es hier möglich, das Konzept aus Kap. 3 als ein ganzheitliches System aus Mensch, DT und realer Hardware in einer Struktur umzusetzen. Hierbei sind vor allem eine Kopplung, eine Fusion und ein Abgleich von Realität und Virtualität herzustellen, um weiterhin das Ziel zu verfolgen, komplexe Systeme intuitiv zugänglich zu machen. Des Weiteren kann der DT in der Funktion des Mediators als Assistenzsystem zum Support sowie zur Prädiktion eingesetzt werden. Daher sind im Folgenden die zentralen Punkte:

1. VR – Abgleich, Fusion sowie Kopplung von Realität und Virtualität,
2. der DT als Assistenzsystem,
3. der DT zur Prädiktion.

Im Rahmen des „Live-Support“ verbindet das VTB den DT mit externen Bibliotheken und Algorithmen primär im ROS-Kontext. Dies führt zu einem modularen, flexiblen und überaus robusten Gesamtsystem, das den Nutzer komplexe Systeme intuitiv erleben lässt. Hierbei helfen optimierten Nutzerschnittstellen, die mit immersiven und intuitiven Steuerungsmodalitäten und sinnvollen Datenvisualisierungen sowohl die direkte Steuerung aus der Ich-Perspektive als auch die Konsultation, Überwachung und Steuerung mit Hilfe des Gesamtsystems im VTB im Blick ermöglichen. Die Nutzung des Konzeptes des DTs als Mediator des Live-Supports und der Prädiktion bedingt jedoch weitere Anforderungen an das zugrundeliegende Simulationssystem. Der Live-Support soll die Möglichkeit liefern, jederzeit zwischen den VisuModel und dem DigTwinModel wechseln zu können. So können Aktionen in der Simulation sicher ausprobiert werden, bevor diese in der Realität durchgeführt werden.

4.5.1 Erweiterte Anforderungen

Die direkte Kopplung von Realität und Virtueller Realität stellt weitere Anforderungen an das umgebende Simulationssystem.

Anforderung *10: Echtzeitfähigkeit*

Eine Steuerung echtzeitkritischer Systeme muss möglich sein.

Auch wenn hier keine harten Echtzeitkriterien greifen, ist z. B. im Bereich der haptischen Interaktion eine Updaterate von 1 kHz nötig. Wie in der Model-Mediated-Telemanipulation (vgl. Kap. 2.4.3.3) könnte natürlich die Simulation auch genutzt werden, um als virtueller Proxy zu fungieren. Die generelle Anforderung ist jedoch, dass die direkte Steuerung auch in Echtzeit funktioniert.

Anforderung *11: Datenabgleich*

Eine zentrale Datenbasis muss als Grundlage für reale und virtuelle Daten vorhanden sein.

Bei der Kopplung/Fusion von Realität und Virtueller Realität muss sowohl eine geeignete Datenhaltung eingerichtet werden, um reale Daten und virtuelle Daten zu halten, als auch ein Datenabgleich möglich sein. Somit ist die Übereinstimmung („consistency“ (en)) der verschiedenen Daten essentiell.

Anforderung *12: Interaktion/Interaktivität*

Laufzeitunabhängige Modularität der Nutzerschnittstelle.

Bei großen Datenmengen, die akquiriert und verarbeitet werden, müssen diese dem Nutzer wiederum zugänglich gemacht werden. Dies erfordert ein höchstes Maß an Modularität, Geschwindigkeit und Übersicht, um ein geeignetes Nutzerinterface zu schaffen, welches das Abstraktionslevel anwenderspezifisch live anpasst.

4.5.2 Echtzeitfähige Starrkörpersimulation

Ein Defizit stellt bisher die Echtzeitfähigkeit der präsentierten Komponenten dar. So ist es zwar möglich, Teile von VEROSIM auf echtzeitfähigen Betriebssystemen laufen zu lassen, aber dies geht einher mit Einschränkungen der visuellen Nutzerschnittstelle und/oder dem Aufwand der Inbetriebnahme verschiedener Modelle. Beispiele dafür sind insbesondere im Bereich des Simulation-based Control (SBC) zu finden wie z. B. in [204]. Hier werden Teile von VEROSIM auf QNX ausgelagert und im Rahmen eines Mockups (aus mehreren KUKA-Leichtbaurobotern) oder auch zur Steuerung eines Motion-Simulators eingesetzt. Echtzeitbetriebssysteme wie beispielsweise QNX erlauben dies, aber ROS ist von Natur aus nicht echtzeitfähig. Oft ist auch eine Echtzeitfähigkeit der Simulation nicht unbedingt notwendig, sondern nur ein echtzeitfähiges Interface. Beispiele hierfür sind in Kap. 5.1 aufgeführt, wo zum einen für ein Exoskelett das 1 kHz-fähige UDP-Interface genutzt und zum andern mit XBotCore ein echtzeitfähiges Roboterbetriebssystem in die Simulation integriert ist (vgl. Kap. 4.4.2.2).

4.5.3 Daten-Haltung, -Abgleich und -Austausch

Zur zentralen Datenhaltung und -verwaltung wird das Konzept des zentralen Weltmodells (englisch „Central World Model“(CWM)) eingeführt und genutzt. Das Konzept des CWM wurde erstmals in [189] als zentraler Knotenpunkt eingeführt, in dem alle unterschiedlichen Systeme Daten speichern und anfordern können. Als gemeinsame Datenbasis

sowie für einen einheitlichen Datenfluss zuständig, ist das CWM Austauschmedium aller Datenverarbeitungskomponenten. Im VTB wird das CWM primär durch die VSD repräsentiert. Hier ist schon der Digitale Zwilling mit allen Aktoren und Sensoren sowie ein Umweltmodell enthalten. Zusätzlich dazu kann ROS als aktives Austauschmedium genutzt werden, wobei die VSD bzw. das VTB im Allgemeinen sowohl als Senke oder als Quelle für das ROS-Netzwerk genutzt werden kann (vgl. vor allem Abb. 4.66). So ist es möglich, im Falle des Visu-Modus auf alle ROS-Nachrichten subscribed zu werden und der DT nimmt ausschließlich Daten auf. Auf der anderen Seite kommen im DigTwin-Modus alle Daten aus dem DT. Natürlich sind auch Mischformen möglich, die DT-Daten mit realen Daten überlagern. Somit umfasst das Konstrukt des CWM sowohl das Datenmanagementsystem des VTB als auch alles, was im ROS-Netzwerk verschickt wird. Das CWM entspricht somit einem Wissenscontainer und lebt von der Standardisierung der internen und externen Schnittstellen des VTBs.

Die Granularität der Daten kann dabei modular an die verschiedenen Szenarien angepasst werden. Somit lassen sich „Regions of Interest“ oder „Bubble of Knowledge“ definieren, in denen eine feine Granularität vorherrschen muss. So ist es bei filigranen Manipulationsaufgaben wichtig, die eigenen Hände hochaufgelöst zu betrachten, jedoch können zur Performance-Stabilisierung die Umgebung und/oder großflächige Laserscans für diese Zeit ausgeschaltet oder ausgeblendet werden. Gegenteilig verhält es sich bei Lokomotionsaufgaben, da dort die hochauflösenden Informationen nicht benötigt werden, dafür aber die Umgebung bestmöglich analysiert werden muss.

4.5.4 Der Mensch als Teil des Systems

Betrachtet man den Nutzer als Teil dieses Systems, kann auch dieser „modular aufgebaut“ sein. So gibt es zum einen *unterschiedliche Anwender* wie Entwickler, Nutzer, Operator oder Besucher und Kunden, welche wiederum unterschiedliches Vorwissen mitbringen.

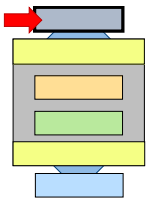


Abb. 4.70: *Konzept: Der Mensch als Teil des Systems (vgl. Abb. 3.4)*

Auch wenn man den Nutzer nicht „optimieren“ kann, so kann durch das genutzte Konzept ein System zusammengestellt werden, welches *individuell* auf den Menschen optimiert ist, z. B. durch geeignete Auswahl der Ein- und Ausgabegeräte, Kameramodi, Visualisierungen usw. Auch der *Grad der Autonomie* spielt eine erhebliche Rolle. Neben der direkten Teleoperation durch den Nutzer kann das System auch autonom agieren. Durch interne und externe Frameworks können solche semi-autonome oder autonome Aktionen durchgeführt werden und dem Menschen die Arbeit erleichtern.

4.5.5 Flexibles, modulares, nutzerzentriertes Gesamtsystem aus Nutzer, Digitalem Zwilling und Roboter

Im Rahmen dieser Arbeit ist ein flexibles und modulares nutzerzentriertes System um die VTB-Architektur erschaffen worden, um intuitiv und sicher mit komplexen Systemen zu interagieren. Die Modularität besteht hierbei darin, dass man nicht nur einzelne Datenverarbeitungsmodule extern oder intern zu- und abschalten kann, sondern sie spiegelt sich in allen folgenden aufgeführten Bestandteilen des Konzeptes wider.

1. Roboter (R): Real R_R, Virtuell R_V, Mischformen $R_{R/V}$, versch. Roboter R^i,
2. Umgebung/Environment (E): Real E_R, Virtuell E_V, „Perceived“ E_P,
3. Daten (D): Real D_R^{In}, D_R^{Out}, Virtuell D_V^{In}, D_V^{Out},
4. Datenverarbeitung (DV): intern, extern,
5. Visualisierung (V)
   a) Blickwinkel/Views (V^w): 1st Person V_{1st}^w, 3rd Person V_{3rd}^w, Free V_{Fr}^w, DataView V_{DV}^w, 2D/3D V_{2D}^w, V_{3D}^w, FrameView V_{FV}^w,
   b) Visualisierungstyp/-style (V^s): HUD V_{HUD}^s, Projected V_{Pr}^s,
6. Steuerung/Control (C): Direct C_D, Autonomous C_A, Semi-Autonomous C_{SA},
7. Eingabegerät/Inputdevice (In): Gamepad, Joystick, Spacemouse, ROS-Control, Exoskeleton, Leap Motion, Kinect, Myo, PHANToM (In_i).

Diese Modularität lässt sich sehr gut anhand des Konzeptes und der unterschiedlichen Schnittstellen verdeutlichen. Der DT kann dann zum Beispiel in einem teleoperativen Szenario in Form von $T(R_R, E_R, D_R^{In}, D_R^{Out})$ verwendet werden. Auch im Bereich der Simulation-based Control (SBC) kann dieses Konzept angewendet werden, da hier $T(R_V, E_V, D_R^{In}, D_R^{Out})$ zur direkten Steuerung aus der Simulation heraus genutzt werden kann.

Neben diesen kooperativen Tätigkeiten ist es möglich, dieses Konzept auch im Rahmen der MRK zu nutzen, um (a) mit mobilen robotischen Systemen entweder indirekt teleoperiert oder direkt „Seite an Seite“ zu agieren und um (b) im industriellen Kontext mit weichen Robotern zusammen zu arbeiten, als seien sie menschliche Kollegen. Für (a) würde dann das mobile System im Rahmen von Sense-Think-Act bzw. Sense-Simulate-Act Aktionen vorausplanen und dann mit dem menschlichen Bediener per Sprache oder Gesten Interaktionen abstimmen. Für (b) kann man von einem gemeinsamen Arbeitsbereich von Roboter und Mensch sprechen, wenn diese intuitiv und natürlich zusammenarbeiten. Hierbei kann der DT „in-the-loop“ sowohl zur Arbeitsraumbegrenzung sowie zur Planung, als auch zur direkten Visualisierung durch Projektoren oder Bildschirmen zu einem interaktiven Miteinander führen.

KAPITEL 5

Anwendungen

Die Kopplung einer MRK mit intuitiven und sicheren MMI mittels des DTs im VTB öffnet ein breites Spektrum an Anwendungen, komplexe Systeme zu verstehen, zu entwickeln, zu gestalten, erlebbar zu machen, zu nutzen, zu optimieren und vor allem deren Zusammenarbeit zu ermöglichen. Der DT wird zum Mediator in allen Prozessschritten. Das vorliegende Konzept und die darin beinhalteten Methodiken lassen sich spwohl lebenszyklusübergreifend in den verschiedenen Prozessstadien als auch prozessübergreifend und prozessformend einsetzen.

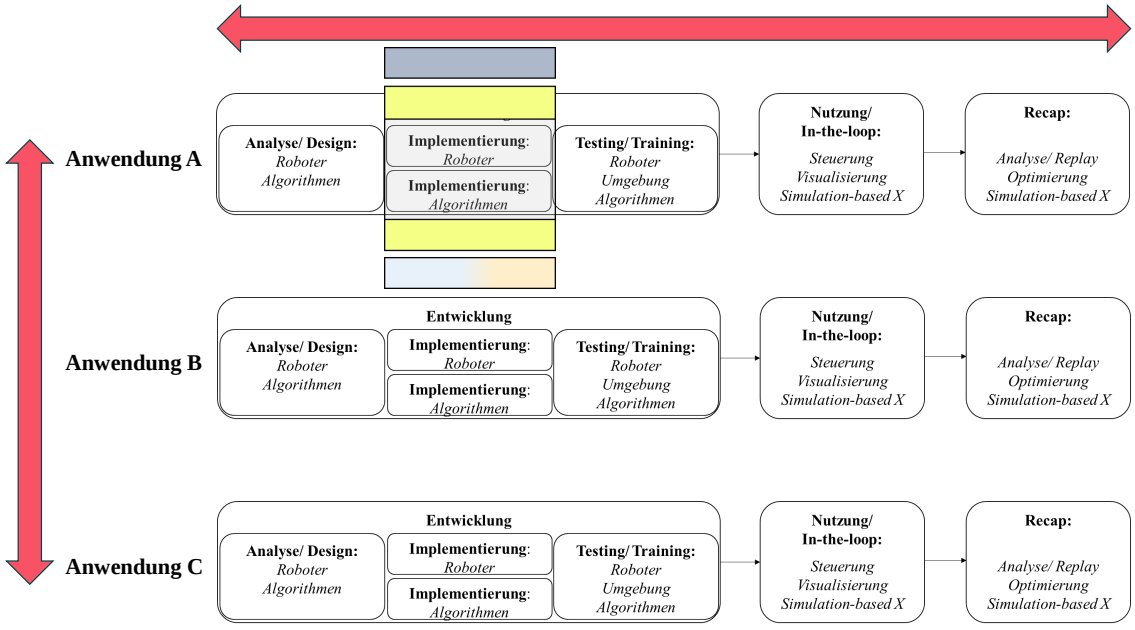


Abb. 5.1: *Anwendbarkeit des Konzeptes: Das Konzept kann (i) über unterschiedliche Anwendungen hinweg sowie (ii) über die verschiedenen Projektphasen genutzt werden (vgl. Abb. 2.29). Je nach Anwendung, Anwender, Prozess, System usw. kann (a) individuell kombiniert werden, welche Submodule des Konzeptes eingesetzt werden und (b) ob mit realen Systemkomponenten oder Digitalen Zwillingen interagiert werden soll.*

Anhand von Abb. 5.1 erkennt man, dass dieses Konzept sich über alle Projektphasen, Anwendungen und Nutzerwünsche erstreckt. Je nach

- Wissensstand des Nutzers,
- favorisiertem Ein-/Ausgabegerät,
- benötigten internen Funktionen,
- gewählter Hardwareschnittstelle
- sowie der Unterscheidung von realem System und Digitalem Zwilling

stellt der Nutzer sich das bestmögliche System zusammen.

So wird in Kap. 5.1 ein mobiler Roboter entwickelt, welcher insbesondere in Katastrophenfällen zum Einsatz kommen soll. Hierbei ist eine effektive Interaktion von Operator und teleoperativ gesteuertem Roboter unerlässlich, um in Extremsituationen den richtigen Zugang, Zugriff und Entscheidungshorizont zu haben, um als Team ein möglichst gutes Ergebnis zu erzielen.

Weitaus planbarer ist dagegen die Optimierung einer rekonfigurierbaren Roboterzelle in Kap. 5.2 einzuordnen. Hierbei stehen vielmehr der Entwicklungs-, Optimierungs- und schlussendlich Nutzungs- und Überwachungsprozess im Mittelpunkt – eine ganzheitliche Herangehensweise, die auch den Nutzer in das System mit einbezieht.

Die Interaktion und Kooperation mit Systemen spielt dann in Kap. 5.3 die vorherrschende Rolle, da hier die neuen Methodiken der VR/AR in den Entwicklungs- und Nutzungsprozess mit eingegliedert werden und so ein neues Engineering-Denken entstehen kann. Mit diesem „Engineering 4.0“ ist es dann möglich, neue Assistenzsysteme oder Interaktionsmetaphern zu entwickeln, die wiederum in diversen Anwendungen eingesetzt werden können.

5.1 Teleoperative Steuerung von Robotern in Katastrophenszenarien

Katastrophen wie in Fukushima zeigen sehr deutlich die Notwendigkeit robotischer Einsatzkräfte für Anwendungen, in denen Menschen nicht eingesetzt werden können. Leider zeigt der Einsatz von mobilen Robotern dort auch deutlich die Grenzen der aktuellen Systeme. Gerade in den kritischen ersten 24 Stunden stellt sich oft die Frage, warum sich hier Menschen in Gefahr begeben müssen und dies nicht von Robotern übernommen werden kann. Gründe dafür sind die merklich eingeschränkte Mobilität und Manipulationsfähigkeiten der mobilen Systeme sowie ihre Komplexität. Ihr Einsatz erfordert umfassendes Training im Umgang und ist in der Regel sehr unintuitiv. Autonome Aktionen sind dagegen für die unstrukturierte Umgebung und das hohe Maß an Ungewissheit in einem solchen Katastrophenszenario nur bedingt möglich, sodass die Nutzung der Fähigkeiten eines Menschen in-the-loop unabdingbar ist. Daraus ergibt sich das Ziel des CENTAURO-Projektes [cv], ein Mensch-Roboter-System zu entwickeln, bei dem der Operator mit seinem gesamten Körper telepräsent einen zentaurenähnlichen Roboter steuern kann. Dieser soll robust manövriert werden, geschickt und fingerfertig manipulieren und unter rauen, ungewissen Bedingungen der Katastrophenszenarien agieren können (siehe Abb. 5.2).



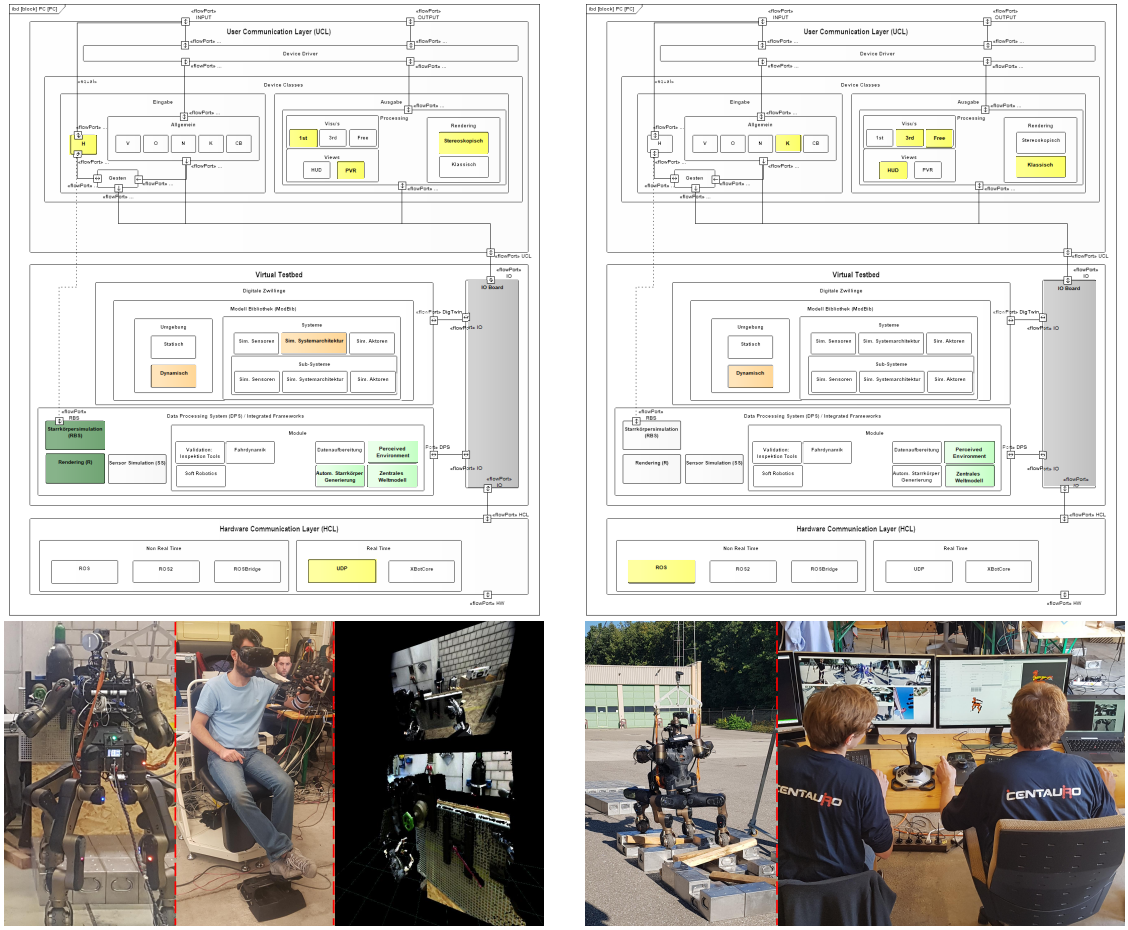
(a) Digitaler Zwilling *(b) Physisches Asset* *(c) Projektidee*

Abb. 5.2: *Das CENTAURO-Projekt: Entwicklung mobiler Roboter mit intuitiven Nutzerschnittstellen sowie Teleoperation mit Hilfe des DTs*

Neben der reinen mechanischen Auslegung dieses Projektes ist insbesondere eine effiziente Kopplung von Operator und mobilem Roboter für den Erfolg einer finalen Mission ausschlaggebend. Die generelle Idee ist hierbei, dass es zwei Operatoren geben wird, wobei der eine (Main Operator) teleoperativ präsent vor allem den Oberkörper mittels eines Force-Feedback-Exoskeletts steuern kann und der zweite (Support Operator) sowohl den Main Operator unterstützt, als auch semi-autonome Bewegungen des Roboters plant und ausführen lässt wie beispielsweise die Lokomotion auf ebenem Terrain von Punkt A zu Punkt B. Die zugehörige modulare Zusammenstellung des DTs ist in Abb. 5.3 dargestellt, um in der finalen Mission entweder mit Exoskelett, HTC Vive und direkter UDP-Verbindung oder mit klassischen I/O-Geräten und ROS den CENTAURO-Roboter

zu steuern.

Im Rahmen dieses Projektes kann der DT als Mediator über den gesamten Lebenszyklus hinweg sehr gut zur Anwendung kommen. Dies beginnt in Kap. 5.1.1 mit der inkrementellen und modularen Entwicklung des Systems in Form von Hard- und Software sowie mit den Mensch-Roboter-Schnittstellen. In Kap. 5.1.2 stehen anschließend der Nutzer bzw. der Operator im Mittelpunkt, wobei die verschiedenen Interaktionsmodalitäten, sowohl im Bereich Teleoperation als auch Autonomie dargestellt, und diskutiert werden. Im dritten Abschnitt (Kap. 5.1.3) wird auf die direkte Interaktion und Kontrolle des Roboters durch den DT eingegangen, bevor es in Kap. 5.1.4 um das Benutzen des DTs im Einsatz geht.

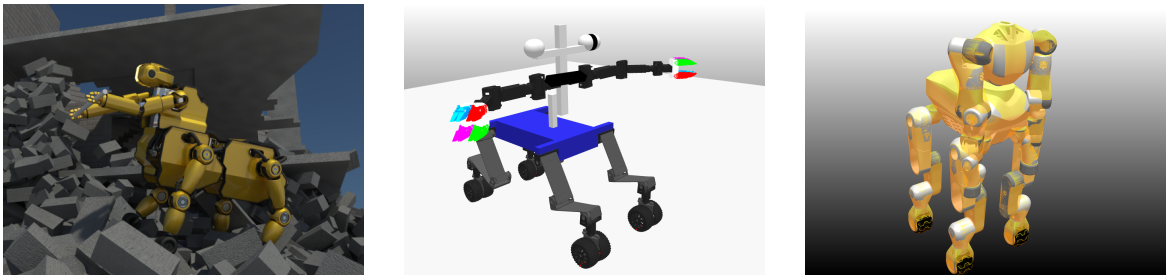


(a) *1st-Person-Operator* ***(b)*** *3rd-Person-Operator*

Abb. 5.3: *Beispielhafte modulare Zusammenstellung des Konzeptes für den Einsatz des CENTAURO-Roboters in Form der Eingabegeräte, genutzten Framework-Module, Hardwareanbindung usw. (vgl. Abb. 3.4)*

5.1.1 Modularität in der Entwicklung mobiler Roboter

Als klassisches Nutzungsmuster des DTs kann man diesen im VTB im Entwicklungsprozess nutzen. Hierbei dient das VTB als „Spielplatz“ für Ingenieure robotischer Systeme. Man kann verschiedene virtuelle Robotermodelle erstellen und überlegen, welche für welche Anwendungsszenarien am besten geeignet sind.



(a) Designstudie *(b) Momaro* *(c) Centauro v1*

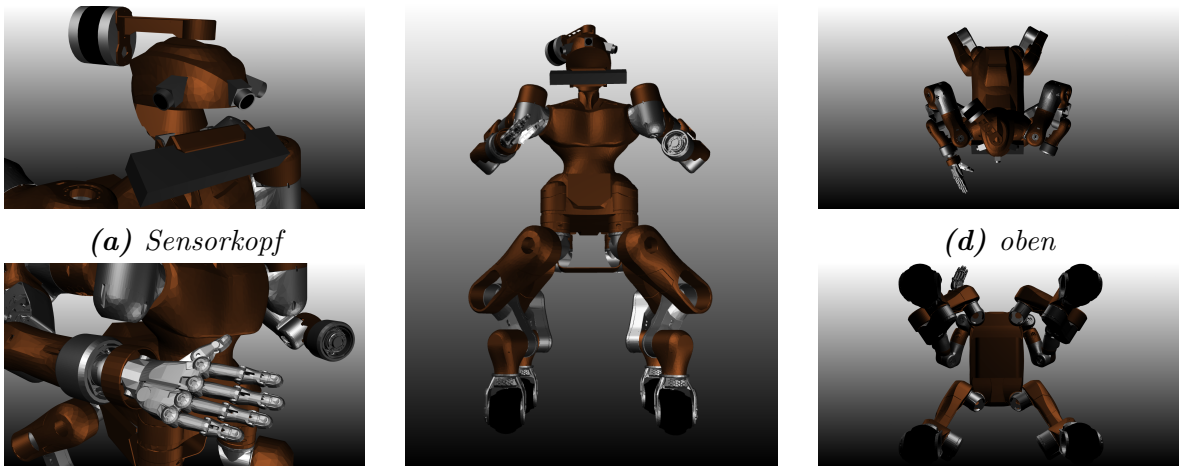
Abb. 5.4: *Digitale Zwillinge von Robotern: Von komplett virtuellen Designstudien über vorhandene reale Systeme zu kompletten Neuentwicklungen lassen sich diese modular zusammenstellen, kombinieren und optimieren.*

Auch wenn die virtuellen Roboter zu diesem Entwicklungszeitpunkt noch keine Digitalen Zwillinge darstellen, da sie bis hierhin keinen Bezug zu realen Komponenten haben, können sie genutzt werden, um das reale System zu entwickeln. Im Laufe der weiteren Entwicklung wird der virtuelle Roboter durch Anpassung und Integration der notwendigen Funktionalitäten immer mehr zum Digitalen Zwilling des realen Systems. Zunächst wird aber dieses prototypische virtuelle Modell genutzt, um modular die einzelnen Komponenten (vgl. Virtual Modular Integration in Kap. 3.4.1) wie z. B. die Image Processing Pipeline oder die Exoskelett-Steuerung zu entwickeln. So entsteht als Endergebnis ein Roboter, der in diesem Fall primär aus der Basis (inkl. Beine), dem Oberkörper (inkl. Arme), der linken und rechten Hand sowie dem (Sensor-)Kopf besteht (vgl. Abb. 5.5).

Alle Sub-Modelle werden dann im Repository Digitaler Zwillinge (Kap. 4.3.2) gesammelt und organisiert. Ein Beispiel für so ein Submodul stellt die Schunk-Hand [cw] dar, welche mit einem Hand-Exoskelett per UDP oder per ROS oder auch per Leap Motion gesteuert werden kann (sieht Abb. 5.6).

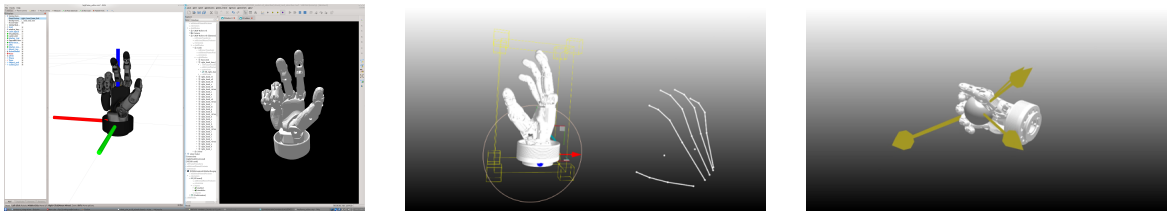
Zunächst wird mit Hilfe des VTB sowie dessen ROS-Interface eine zentrale Datenbasis in Form des CWM (Kap. 4.5.3) aufgebaut, die alle notwendigen Daten beinhaltet. Durch ROS als Middleware (Kap. 4.4.1) können hier sofort externe Module an das virtuelle Robotermodell angedockt und genutzt werden. Durch mehrere Iterationsschleifen kann das virtuelle Modell immer weiter dem realen Modell angepasst werden und anders herum. Außerdem können Erkenntnisse, die mit Hilfe des DTs gewonnen wurden, in die Entwicklung und Auswahl entsprechender Hardwaremodule einfließen. Hierbei können virtuelle Benchmarkszenarien helfen (Kap. 4.3.1.3), die Fähigkeiten „eines solchen Roboters“ in sehr frühem Entwicklungsstadium zu prüfen.

Das CENTAURO-Modell kann dabei mit Hilfe des URDF-Importers (Kap. 4.3.1.2) erneuert oder erweitert werden. Dies ist somit insgesamt ein sehr agiler Prozess, der als



(a) Sensorkopf *(b) Hände* *(c) vorne* *(d) oben* *(e) unten*

Abb. 5.5: *VEROSIM CENTAURO-Roboter: modularer Aufbau des Roboters aus (i) Basis (bestehend aus Rumpf, Oberkörper und zwei Armen), (ii) Sensorkopf, (iii) Schunk Hand (links) sowie Universaladapter für z. B. Soft Hand (rechts)*



(a) ROS-Interface *(b) Leap-Motion-Steuerung* *(c) Greifkraftanalyse*

Abb. 5.6: *Schunk-Hand*

Gesamtziel einen realen Roboter und seinen Digitalen Zwilling hat. Des Weiteren werden in diesem Entwicklungsschritt auch schon die unterschiedlichen Operationsmodi definiert (VisuMode und DigTwinMode (Kap. 4.4.2 sowie Abb. 4.66)) und mittels ROS verbunden.

Mit Hilfe dieser SBSE-Entwicklung wurde schlussendlich ein finales Gesamtsystem entwickelt (siehe Abb. 5.7).

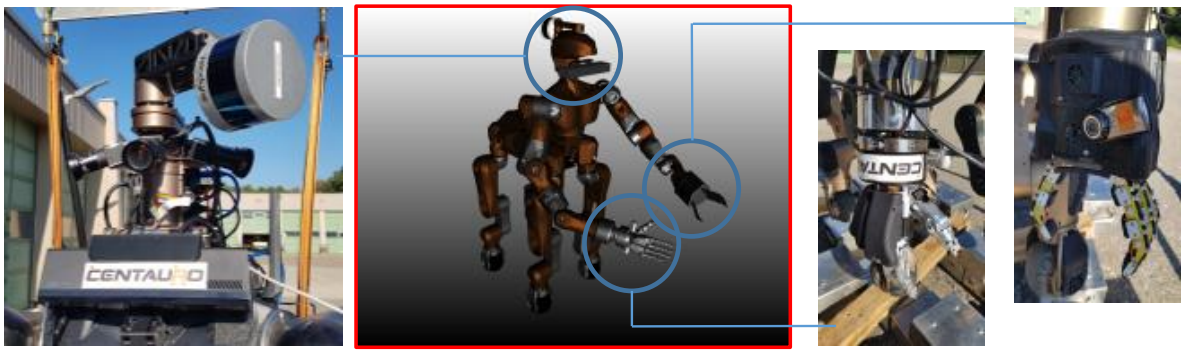


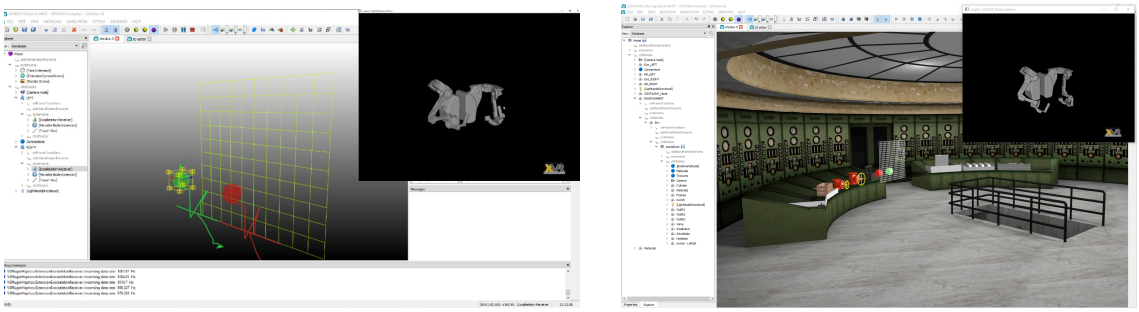
Abb. 5.7: *Finaler CENTAURO-Roboter: basierend auf dem Robotermodell sowie dem realen Sensorkopf und variablem Hand-Setup aus Schunk- und HERI-Hand*

5.1.2 Teleoperation, Telepräsenz und Immersion durch den Digitalen Zwilling

Der Digitale Zwilling als lebenszyklusübergreifender Mediator umfasst neben der im vorigen Kapitel gezeigten Entwicklung auch die Nutzungsphase. Hierzu gehört insbesondere das FF-Exoskelett, welches zunächst in Kap. 5.1.2.1 mit Hilfe des DTs entwickelt wurde und daraufhin in Kap. 5.1.2.2 zur Situation Awareness und in Kap. 5.1.2.3 auch zum Einsatz kommt.

5.1.2.1 FF-Entwicklung mit Hilfe des DTs

Mit Hilfe eines Digitalen Zwillings des realen Systems kann man die Teleoperation, Telepräsenz und Immersion in das VTB übertragen. Alleine schon die korrekte Visualisierung des Digitalen Zwillings als 1:1-Kopie des realen Systems führt beim Nutzer zu einer direkten Verbundenheit und Akzeptanz. Die Teleoperation kann daraufhin mit der Integration des Force-Feedbacks sowie der Anbindung des Exoskelettes an das VTB erfolgen, um dies zunächst in der Virtualität zu testen (Abb. 5.8) und schlussendlich auch in der direkten Teleoperation zu nutzen (Abb. 5.9).



(a) *Beidhändige Interaktion mit Force Feedback*

(b) *Teleoperation in Katastrophenumgebung*

Abb. 5.8: *Exoskelett: Entwicklung und Nutzung eines FF-Exoskeletts im VTB in verschiedenen realitätsnahen Manipulations-Benchmarkszenarien*

5.1.2.2 Situation Awareness durch den DT in VR

Die Integration stereoskopischer VR-Brillen hilft dabei, eine solche immersive Erfahrung zu ermöglichen. Dies führt zu einer sog. „Situation Awareness“.

Aufgrund des modularen Aufbaus des UCLs können die vorhandenen Eingabegeräte von allen Projektpartnern genutzt werden. So macht es beispielsweise Sinn, für die Entwicklung von Greifszenarien der Schunk-Hand auf das Leap-Motion-Eingabegerät (Kap. 2.3.4.2) zurückzugreifen, wenn kein Hand-Exoskelett verfügbar ist. Andererseits kann es zur Entwicklung des reinen Force-Feedbacks sinnvoller sein, dies vorerst mit dem PHANToM Device (Kap. 2.3.4.3) zu testen, bevor das gesamte Exoskelett genutzt wird.

5.1.2.3 Der Digitale Zwilling zur Teleoperation

Bei der direkten Teleoperation mit Hilfe des Digitalen Zwillings des Systems kann sich der Operator telepräsent an den Einsatzort begeben und durch die Augen des Roboters Aufgaben erledigen. Exemplarisch ist in Abb. 5.9 die Manipulationsaufgabe des zweihändigen Greifens eines Akkuschraubers gezeigt. Hierbei ist der Main Operator im Exoskelett direkt mit dem Roboter verbunden und wird visuell durch den Digitalen Zwilling unterstützt. Dies umfasst hierbei eine an den speziellen Operator angepasste Anordnung von Visualisierungen der Robotersensordaten. In Abb. 5.9 sieht man das stereoskopische Rendering eines texturierten Polygonnetz der Kinect v2 sowie ein RGB-Bild in der oberen Peripherie des Blickfeldes und ein RGB-Bild der Handkamera an der rechten Seite des Blickfeldes. Bei Bedarf kann der Support-Operator zusätzlich z. B. das Visu-Modell einblenden, damit man noch einen besseren Überblick über die aktuelle Roboterpose bekommt. Auch weitere HUD- oder HP-Visualisierungen können dynamisch zu- oder abgeschaltet werden. Die Sicht des Support Operators ist in den folgenden Kapiteln beschrieben.

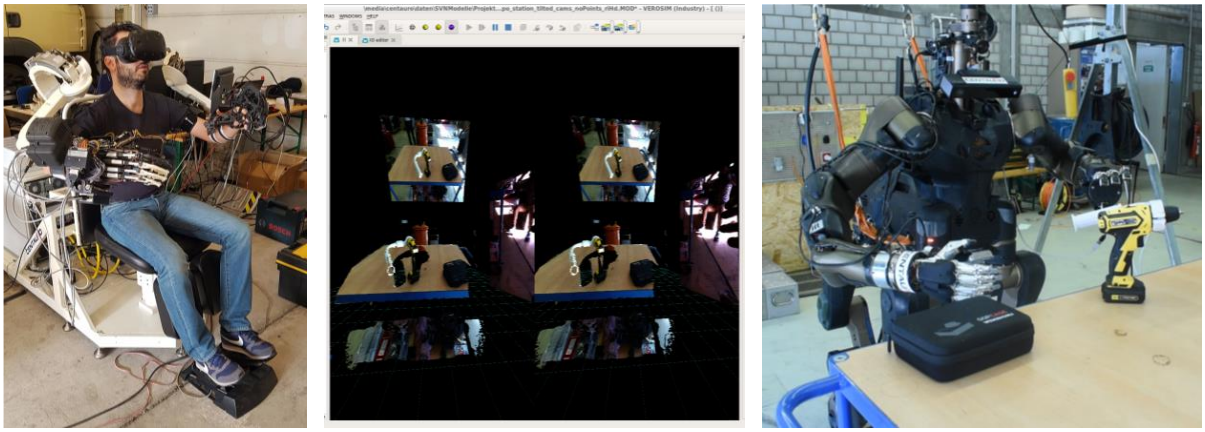


Abb. 5.9: *Exoskelett: Telemanipulation mit Kraftrückkopplung. Live-Daten werden im VTB stereoskopisch als texturiertes Polygonmesh in die HTC Vive gerendert.*

5.1.3 Der Digitale Zwilling „in-the-loop“ zur direkten Interaktion und Kontrolle

Nutzt man dann den DT in-the-loop, kann dieser vor allem im VisuMode genutzt werden, um einen intuitiven Zugang zum System zu bekommen. Dabei helfen vor allem sinnvolle Visualisierungen der Rohdaten oder vorverarbeiteter Daten. Diese können dann entweder im Sinne eines HUD oder projiziert in die 3D-Umgebung als HP (Kap. 4.2.5.1) angezeigt werden. Hierbei stehen insbesondere die Objekterkennung sowie die semantische Einordnung der Daten im Vordergrund. Zusätzlich zur Visualisierung ist die Steuerung des mobilen Systems von besonderer Bedeutung.

Dies ist auch möglich, wenn die DTs *direkt* mit ihren physischen Assets verknüpft sind, wie am Beispiel eines mobilen Roboters in Abb. 5.10 zu sehen ist.

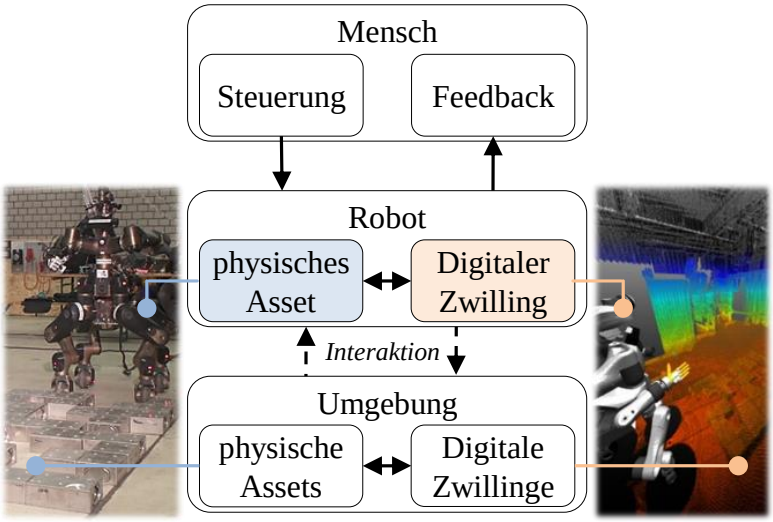
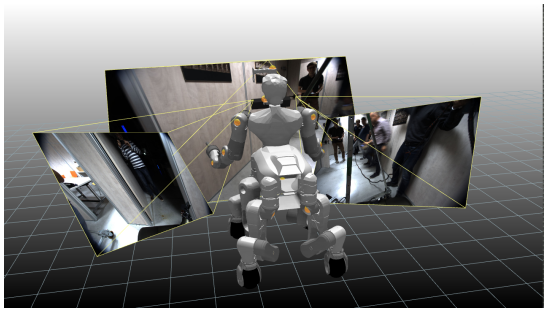


Abb. 5.10: *Das VTB als Netzwerk interagierender DTs am Beispiel mobiler Roboter. Der DT des zu untersuchenden Roboters interagiert mit DTs von Objekten der Umgebung und ist im direkten Austausch mit seinem physischen Asset [5]*

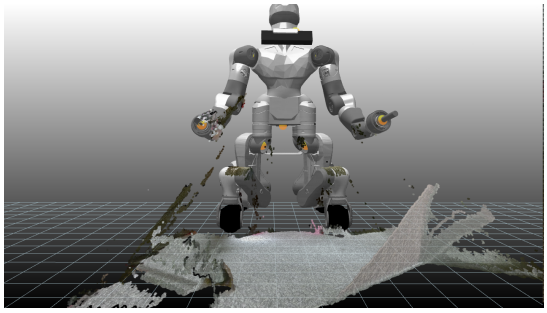
5.1.3.1 Visualisierung

Zur effizienten Interaktion mit dem robotischen System müssen alle notwendigen Systemparameter für den Nutzer zugänglich sein. So ist es für den klassischen Nutzer möglich, alle Video- und Bilddaten modular im VTB zusammenzustellen, diese mit Diagrammen und weiteren Informationen zu überlagern, in die 3D-Umgebung zu projizieren und auch den aktuellen Roboterstatus am Digitalen Zwilling abzubilden.

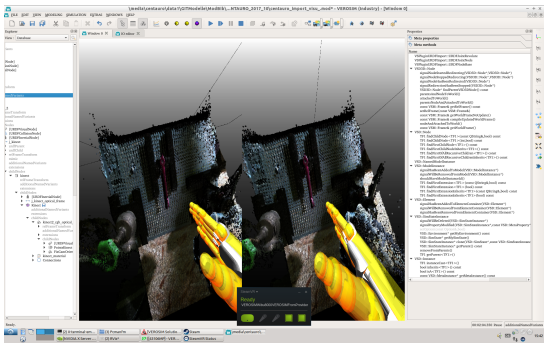
Geht man nun einen Schritt weiter, hat der Nutzer auch in VR Zugriff auf all diese Informationen. Mit dem direkten Blick „aus dem Kopf“ des Roboters in VR hat der Nutzer einen intuitiven Zugang zu bevorstehenden Manipulationsaufgaben und kann sogar während dieser den Blickwinkel digital verändern, ohne dass sich der reale Kopf des Roboters bewegen muss. Wie in Abb. 5.11 zu erkennen, ist es so möglich, zunächst

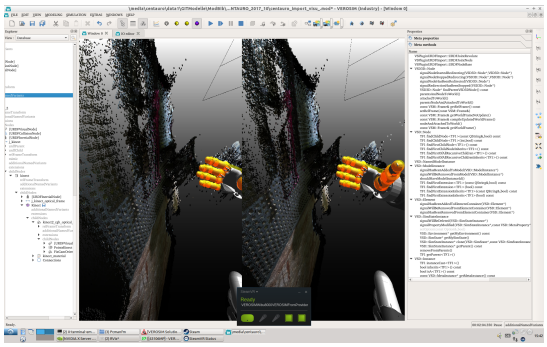


(a) 3D-Modell und Kamerafrustrumvisualisierung von drei RGB-Kameras

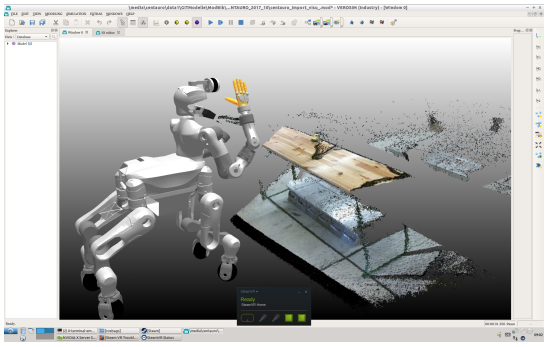


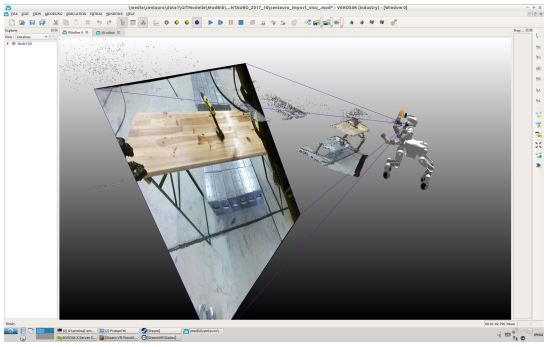
(b) 3D-Modell und Kinect v2 Punktwolkenvisualisierung





(c) 1st-Person-View auf die Punktwolke während eines Greifprozesses aus versch. Perspektiven





(d) 3rd-Person-View auf 3D-Modell, Punktwolke und RGB-Bild beim Greifen eines Akkuschraubers

Abb. 5.11: *CENTAURO-Roboter visualisiert im VTB*

aus der 3rd-Person-Perspektive die Navigationsaufgaben zu verfolgen (a,b) und sobald Manipulationsaufgaben anstehen, in die 1st-Person-Ansicht zu wechseln (c). Zusätzlich kann es sehr hilfreich sein, das Blickfeld an der jeweils gestellten Aufgabe zu optimieren (d).

Während alle bisherigen Ausführungen im Sitzen durchführbar waren, ist es jetzt möglich physisch aufzustehen, um die Immersion noch zu vergrößern (vgl. Abb. 5.12). Der Operator kann sich somit im VTB einen Eindruck über den teleoperierten Roboter verschaffen, indem er (virtuell) um diesen (bzw. den DT) herumläuft und sich alle Daten an die Position des Erscheinens oder in sein Sichtfeld projizieren lässt.

Auch hier werden neben Punktwolken, Bild-, Roh- sowie vorverarbeitete high-level-

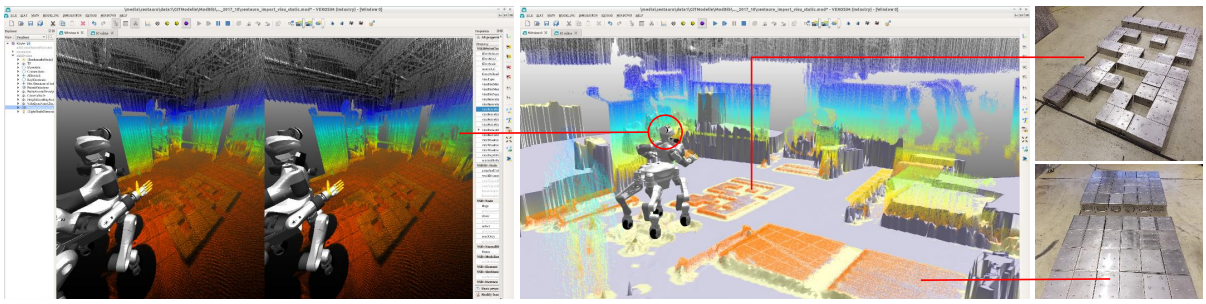
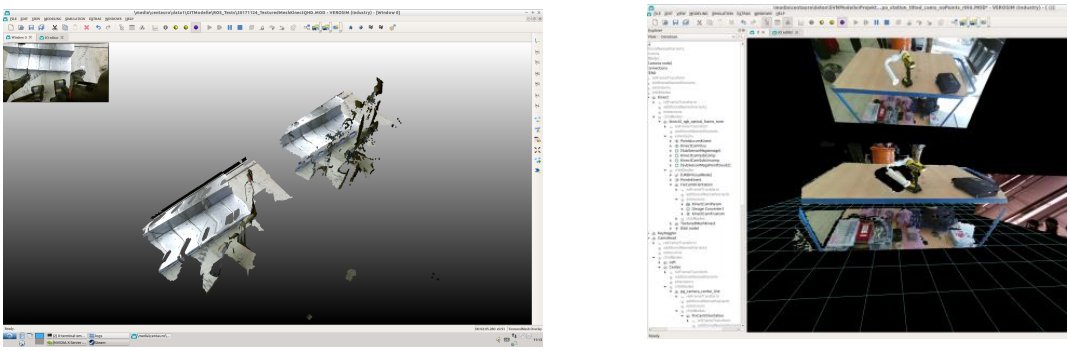


Abb. 5.12: *Gesamtsystem Evaluationsszenario, bestehend aus verschiedenen Aufgaben sowie die Visualisierung im VTB (in Ego- oder 3rd-Person-Perspektive) und die direkte Interaktion mit dem realen Roboter durch den DT*

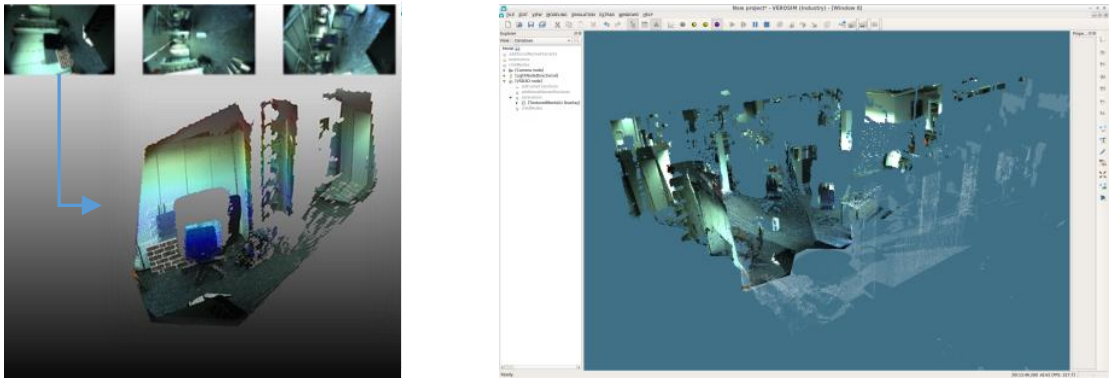
Daten wie das texturierte Polygonnetz aus Kap. 4.2.5 genutzt (präsentiert in Abb. 5.13 für die Kinect und in Abb. 5.14 für den Velodyne-Laserscanner).



(a) *Punktwolke (oben rechts) und Bild (oben links) ergeben Mesh (mitte)*

(b) *Hochaufgelöstes Kinectmesh für Manipulationsaufgaben im HMD gerendert*

Abb. 5.13: *Texturiertes Polygonnetz: Kinect*



(a) *(b)*

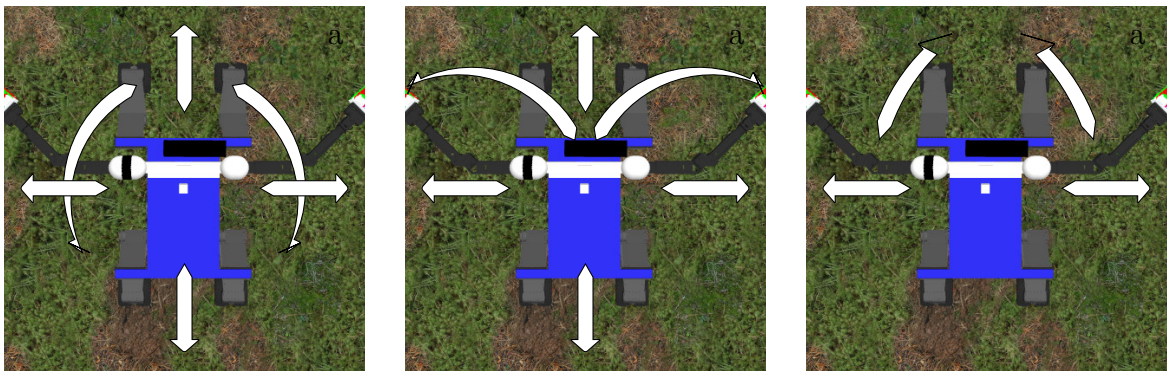
Abb. 5.14: *Velodyne-Punktwolke und drei RGB-Kamerabilder – wobei eins davon zusammengemeshed und mit der zugehörigen Sub-Punktwolke gezeigt wird (a); daraufhin werden alle drei RGB-Kameras mit der gesamten Punktwolke gemeshed, fehlende Bilddaten (hinter dem Roboter) werden mit einer uniform grauen Textur gemeshed (b).*

5.1.3.2 Direkte Kontrolle mobiler Roboter

Der mobile Roboter kann mit Hilfe von verschiedenen Control Setups in und durch den DT auf unterschiedliche Weisen neben der direkten Ansteuerung einzelner Gelenke (*Joint-Control*) gesteuert werden.

5.1.3.2.1 Kontrollstrategien

Präsentiert in Kap. 2.4.3.2 wurden hier für quadrupedale Roboter die verschiedenen Kontrollstrategien in Abb. 5.15 implementiert.



(a) *Decoupled Rot-Trans* ***(b)*** *Car-like steering* ***(c)*** *Caterpillar*

Abb. 5.15: *Verschiedene Kontrollstrategien (von oben)*

Für solche quadrupedalen Robotersysteme (siehe auch Abb. 2.23) sind verschiedene Steuerungsmodi implementiert, die entweder (a) die klassische/simple Rotation und Translation entkoppeln, (b) wie bei einem Auto Geschwindigkeit und Lenkwinkel nutzen

oder (c) die Raupenfahrzeug-Steuerung mit der Erweiterung um direkte laterale Manövrierbarkeit.

5.1.3.2.2 Semi-Autonomie

Insbesondere im Bereich der Lokomotion bietet es sich an, Aufgaben mit semi-autonomen sog. „Motion-Primitives“ zu lösen. Beispiele hierfür sind zum einen die Fahrt zu einem bestimmten Punkt, zum anderen das Übersteigen eines Spalts. Mit Hilfe externer Planungstools, welche Daten über die ROS-Schnittstelle austauschen, konnte somit folgender Workflow etabliert werden (vgl. Abb. 5.16): Auf Basis der Sensordaten wird eine Höhenkarte erzeugt, welche im VTB angezeigt wird. Zusätzlich wird diese Karte genutzt, um in einem externen Tool eine Kostenfunktion zu berechnen, welche auch an das VTB geschickt und dort als Überlagerung der Höhenkarte angezeigt werden kann. Anhand dieser Karte kann eine Zielpose definiert werden, wohin der Roboter fahren soll. Diese wird über ROS (*centauro_msgs/locomotiongoal*) an ein externes Planungstool geschickt. Mögliche Pfade werden daraufhin an das VTB zurückgeschickt, dort angezeigt und es ist nun möglich, einen dieser auszuwählen und diese Auswahl zurück an den Planer zu schicken. Dieser schickt die nötigen Steuerungsbefehle an den Roboter. Bei der Bewegung des Roboters kann dieser dann im VTB mit Hilfe des DTs überwacht werden. Zusätzliche Visualisierung der Kamerabilder dient zusätzlich der Überwachung.

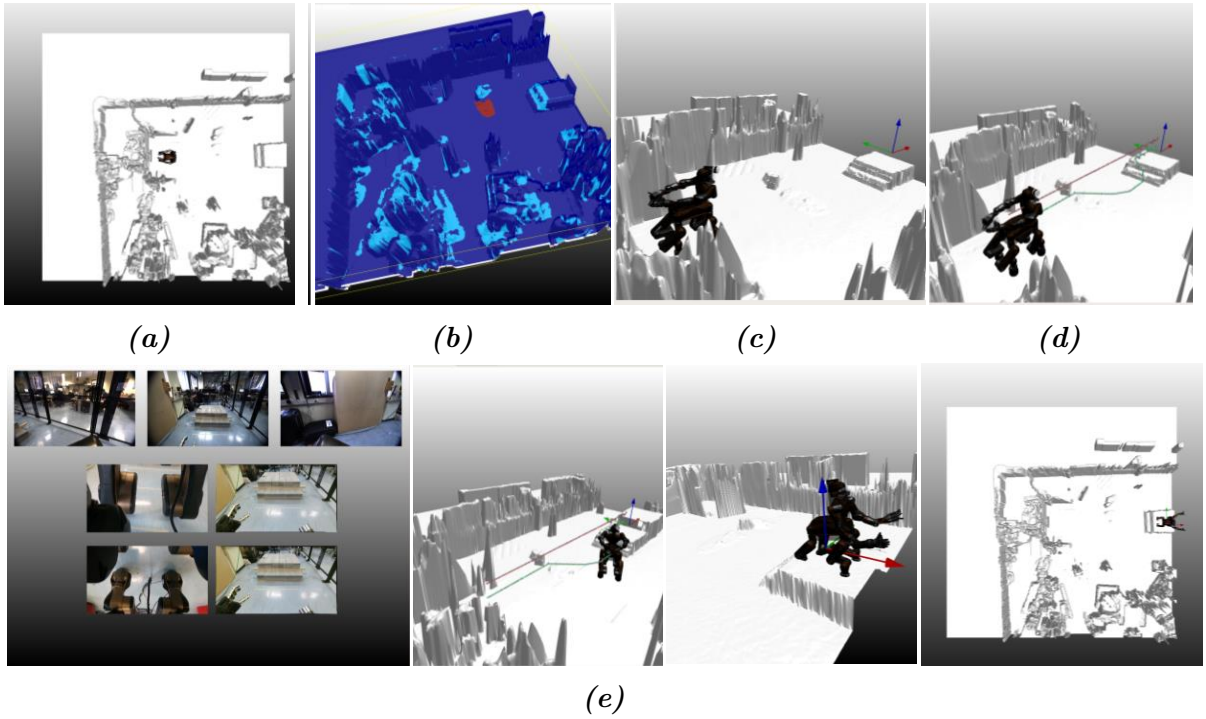


Abb. 5.16: *Semi-autonome Lokomotion durch den Support Operator: (a) Höhenkarte, (b) Kostenfunktion, (c) Zielpose definieren, (d) versch. Pfadplanung und Auswahl eines Pfades, (e) Lokomotion mit Live-Überwachung anhand des 3D-Modells und Bilder sowie erfolgreiches Abschließen der Lokomotion auf der Plattform*

5.1.3.3 Nutzungsmodi

Zusätzlich dazu ergeben sich durch das einfache Austauschen von realem Roboter und Digitalem Zwilling weitere Nutzungsmodi, die im Rahmen des CENTAURO-Projektes genutzt werden. Insbesondere interessant ist der „Switch“ aus einem realen Szenario in den DT – der sog. „Prediction Mode“ welcher im folgenden Kapitel ausführlich beschrieben ist. Hierzu dienen die in Kap. 4.3.1.4 und 4.3.1.5 gezeigte Umwandlung von wahrgenommen Umgebungen und Objekten in Digitale Zwillinge. Somit lässt sich online das reale Szenario automatisch nachbilden und experimentierbar erfahren.

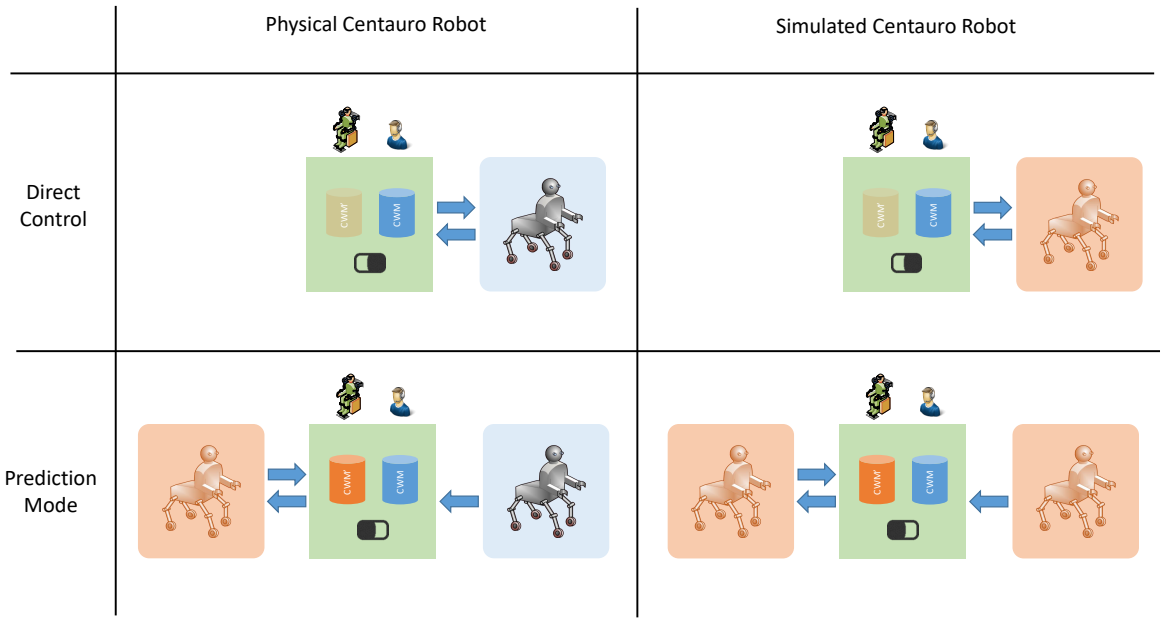
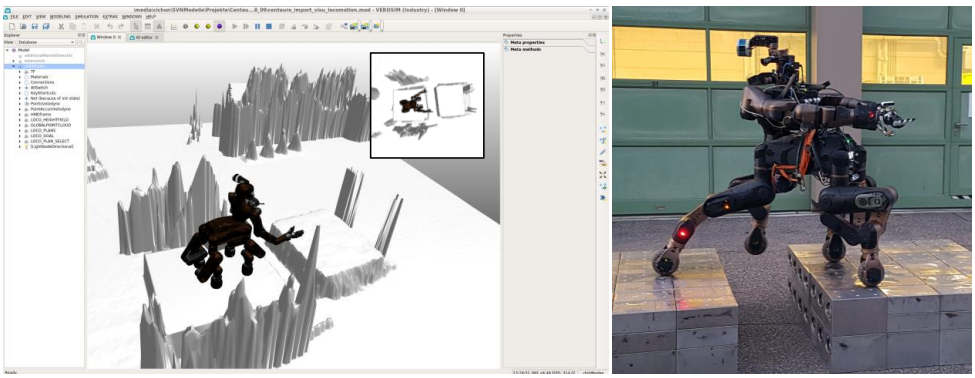


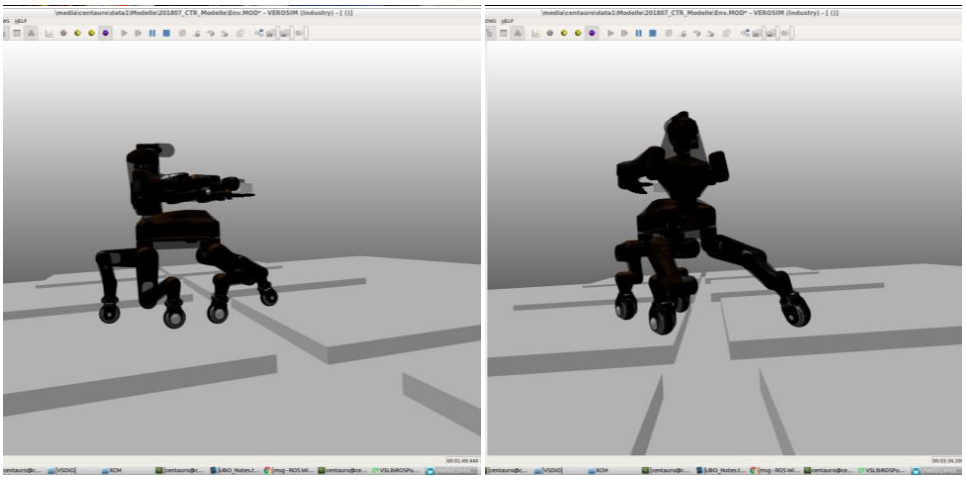
Abb. 5.17: *Nutzungsmodi des Digitalen Zwillings: (lo) direkte Steuerung des realen Systems, (ro) direkte Steuerung des Digitalen Zwillings, (lu) Prädiktionsmodus real, direkte Steuerung des realen Systems mit Möglichkeit des Wechsels zum virtuellen, (ru) Prädiktionsmodus virtuell, direkte Steuerung des virtuellen Systems mit Möglichkeit des Wechsels in ein weiteres virtuelles.*

5.1.4 Prädiktion durch den Digitalen Zwilling

Durch die Möglichkeit, den Digitalen Zwilling in den zwei Modi (VisuMode, DigTwinMode) zu nutzen, kann hierbei auch in der Mission zwischen der direkten Kontrolle des realen Systems, visualisiert im VTB, und der direkten Kontrolle des Digitalen Zwillings umgeschaltet werden. Dies führt im Allgemeinen zur Möglichkeit der Prädiktion. Der Digitale Zwilling in der wahrgenommenen Umgebung kann in der Starrkörpersimulation mit dem Exoskelett gesteuert werden. Man kann diverse Lösungswege der aktuellen Aufgabe erst am DT ausprobieren, bevor man diese in der Realität durchführt. Solche „What if...?"-Fragen können sehr gut im VTB beantwortet werden. So können diverse Bahnplanungen überprüft, getestet und visualisiert werden, der Arbeitsraum überprüft oder einfach die Reichweite, Schrittgröße o. Ä. analysiert werden (vgl. Abb. 5.18). Besonders interessant ist, dass diese Informationen und Daten wiederum auch vom Support Operator generiert und daraufhin in das Sichtfeld des Main Operators gepusht werden können, sodass dieser den besten Weg oder die sinnvollste Greiftrajektorie in seinen Arbeitsraum projiziert bekommt. Der Main Operator selber kann durch die direkte Steuerung des realen Systems durch das Exoskelett nur schwer solche Befehle geben.



(a)



(b)

Abb. 5.18: *Der „Switch" in den DT: Online wird eine Höhenkarte generiert (a) und daraufhin können im VTB verschiedene Ansätze des Steppings ausprobiert werden (b).*

Neben dieser „Online-Prädiktion“ kann das hier präsentierte Konzept natürlich auch zur „Offline-Prädiktion“ genutzt werden. Durch die Abbildung realer Benchmark-Szenarien (Kap. 4.3.1.3) kann das System schon in jedem Entwicklungsstadium auf die finalen Einsätze hin getestet und optimiert werden. Des Weiteren lässt sich damit trainieren und testen, ob die Nutzerschnittstelle schon optimal eingestellt ist. Hierbei spielt dann wiederum der Modus virtueller Prädiktion (Abb. 5.17 (ur)) eine entscheidende Rolle, da man hierbei das VTB mit einem zweiten VTB koppelt.

Insgesamt betrachtet, ist im Rahmen des Projektes CENTAURO nicht nur ein Roboter, sondern vielmehr ein System aus Operator, DT und mobilem Roboter entstanden, das sich in erster Linie durch die Immersion und Interaktion der Operator auszeichnet.

5.2 Mensch-Roboter-Kooperation im industriellen Anlagenbau

Auch in der Industrie bzw. der industriellen Anlagenentwicklung geht der Trend mehr und mehr dahin, flexibel auf Änderungen in der Produktionskette reagieren zu können, den Mitarbeiter durch robotische Systeme zu unterstützen und im Allgemeinen ein symbiotisches Miteinander aus Montageapparatur, Roboter und dem Werker zu erreichen. Auch hierfür kann der DT genutzt werden, zum einen zur Systementwicklung (Anlagenplanung, Ablaufoptimierung, Konzeption usw.) und zur Prozessüberwachung (Monitoring des Prozesses), zum anderen auch zur direkten Steuerung dieser genannten Vorgänge in Form von SBC.

5.2.1 Rekonfigurierbare Roboterzelle zur Montage

Um schnell auf Produkt- oder Prozessänderungen in der Produktion reagieren zu können, wird im Rahmen des Projektes ReconCell [cx] eine rekonfigurierbare Roboterzelle für klein- und mittelständische Unternehmen entwickelt. Dabei wird hier auch der VTB-Ansatz gewählt.

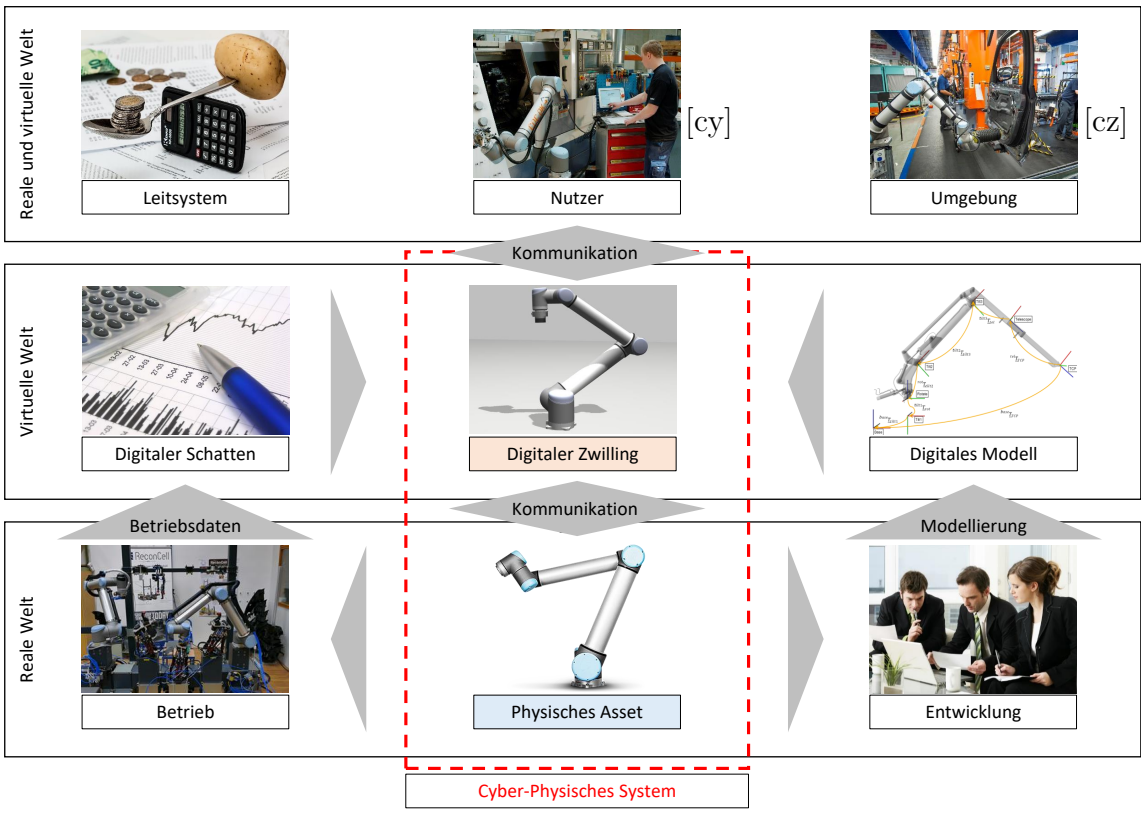


Abb. 5.19: *Der Digitale Zwilling in der Industrierobotik*

Hinzu kommt, dass das im Rahmen dieser Arbeit entwickelte ROS-Framework es ermöglicht, das VTB auch mit dem realen System (den Universal Robots [da] Armen) zu verbinden. Dazu wird ein Digitaler Zwilling des Gesamtsystems entwickelt, an dem auch Arbeitsschritte visuell programmiert werden können (vgl. Abb. 5.20). Zusätzlich kann

man in einem VisuMode den aktuellen Status des realen Systems überwachen und in Notsituationen eingreifen.

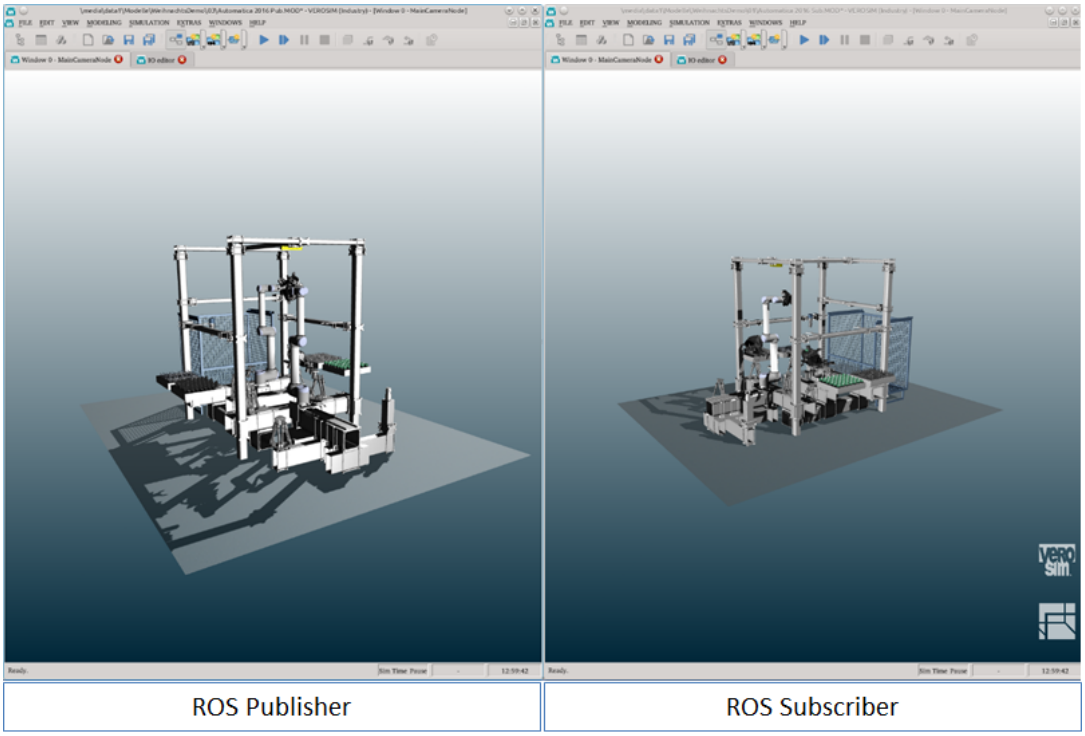


Abb. 5.20: *MRK: rekonfigurierbare Roboterzelle, entwickelt im Rahmen des ReconCell-Projektes. Hier im Anwendungsmodus der virtuellen Prädiktion (vgl. Abb. 5.17). Das reale System wird durch einen Digitalen Zwilling (links) repräsentiert, welcher durch einen zweiten Digitalen Zwilling (rechts) überwacht werden kann, verbunden durch ROS.*

Interessant hierbei ist, dass der Digitale Zwilling auch für die Optimierung der Zellkonstruktion genutzt werden kann. So wird in [242] eben diese ReconCell insofern optimiert, als die verwendete Sensorik dort positioniert wird, wo die beste Sicht vorhanden ist. Es können also verschiedene Optionen durchdacht und am Ende eine optimale Lösung gefunden werden.

5.2.2 Das Virtuelle Testbed mit VR/AR in der Produktion

Neben der reinen Konstruktion, Entwicklung und Optimierung von industriellen Anlagen stehen auch hier natürlich der Mensch und die Interaktion zwischen Mensch und Maschine klar im Mittelpunkt. Gerade hier kann der DT wiederum als Mediator fungieren, wenn man die Funktionalitäten der VR, AR oder MR ausnutzt. So ist es möglich, sich schon vor Fertigung der Zelle in dieser virtuell zu bewegen und mit dieser zu interagieren. Mit Hilfe des interaktiven Markers (Kap. 4.2.5.1 und Abb. 4.23) beispielsweise steht eine sehr intuitive Metapher zur Verfügung, mit der man den TCP beliebiger Roboter bewegen kann. Des Weiteren werden erste Schritte unternommen, um dieses Konzept auf die AR-

Brillen zu erweitern. Hierfür werden bisher die Vuzix M300 und ein 360-Grad-Video aus der Simulation genutzt, da der Übergang zur Android-basierten AR-Brille etwas mehr Umstellungen beinhaltet. Jedoch ist es so möglich, vordefinierte Simulationsinhalte auf die AR-Brille zu übertragen und sich mit Hilfe dieser in der Anlage besser zurecht zu finden. Auch im Bereich MR wurden Vorbereitungen getroffen, um auch dies mit demselben DT zu koppeln (siehe [243]), indem Daten aus dem DT selektiv auf die MR-Hardware gestreamt werden können.

Schlussendlich ist die Idee, dass im Entwicklungsprozess primär VR genutzt wird, da dadurch die Konstruktion und die Entwicklung erlebbar gemacht werden, man VR weiterhin zum Monitoring nutzt, der Werker aber mit Hilfe von AR oder MR und demselben Digitalen Zwilling in der realen Anlage angelernt werden kann.

5.2.3 Direkte, intuitive und sichere Steuerung „softer“ Roboter

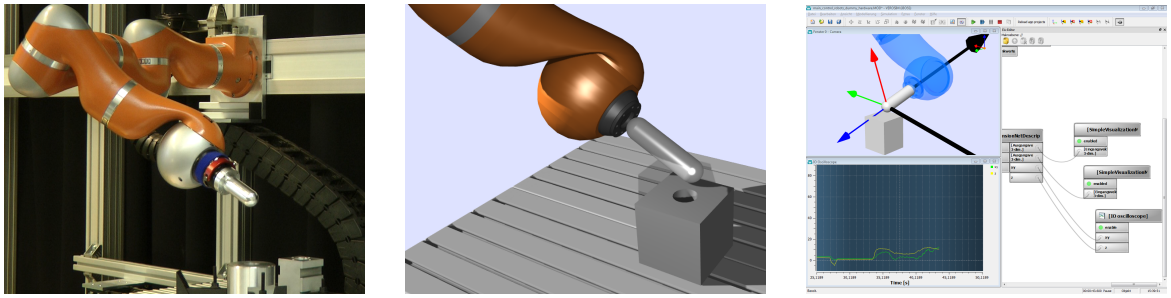
Auch wenn klassische Roboterzellen durch die zuvor genannten Mittel rekonfigurierbarer, modularer und flexibler werden, ist es momentan immer noch so, dass Manipulations- und Montageaufgaben hoch präziser Industrieroboter in Roboterzellen durchgeführt werden. Um Industrieroboter aus ihren Käfigen zu befreien, müssen diese Systeme inhärent sicher und ungefährlich für menschliches Personal sein. Nur so, interagierend mit der Umgebung und den menschlichen Arbeitskräften, können die Produktivität und Flexibilität erhöht und daraus folgend auch Kosten reduziert werden, wie z. B. in [244] aufgezeigt. Dies fördert den Wunsch nach „soften“ Robotern, die im Rahmen der MRK sicher und konstruktiv mit Menschen zusammenarbeiten wie z. B. in Füge- oder Montageaufgaben [245]. Diese Soft Robots bringen dann die Eigenschaft mit, sich adaptiv an ihre Umwelt bzw. extern angreifende Kräfte anzupassen.

5.2.3.1 Peg-in-Hole

Dieses Verhalten kann und sollte auch im Rahmen des klassischen Peg-in-Hole-Problems (vgl. Kap. 2.4.2.1) genutzt werden. Beim Einfügen eines Zapfens in ein Loch führen minimale positionelle Ungenauigkeiten aufgrund von Material, Reibung usw. zu Kräften und Momenten, die ein Soft Robot ausgleichen kann.

Für einen effizienten und sicheren Soft Robot ist ein kontinuierlicher Entwicklungsprozess für einen solchen Regelalgorithmus erforderlich. Dieser Prozess sollte die Entwicklung, das Testen, die Optimierung und die Validierung mit Hilfe des DTs innerhalb des VTBs sowie die Übertragung und den Endbetrieb auf einem echten Roboter umfassen. Folglich ist eine schrittweise Implementierungs- und Optimierungsphase wünschenswert, gefolgt von einer nahtlosen Übertragung und einer endgültigen Integration. Deshalb ist ein ganzheitlicher Entwicklungszyklus von der Idee bis zum endgültigen, realen Roboterexperiment sinnvoll (am Beispiel von Peg-in-Hole-Aufgaben):

1. theoretische Entwicklung eines Kontrollalgorithmus basierend auf Petri-Netzen,
2. mit einem VTB für anwendungsorientiertes Testen (wie in einem echten Mock-up-System),



(a) *(b)* *(c)*

Abb. 5.21: *MRK: Peg-in-Hole-Experiment: reales System mit überlagertem virtuellen Digitalen Zwilling*

3. Optimierung des Algorithmus anhand des DTs,
4. Bereitstellung auf das echte Robotersetup.

Dieser ganzheitliche Ansatz der Steuerungstechnik ermöglicht es, die Steuerungsalgorithmen nahtlos zu entwickeln, zu optimieren, zu testen und zu implementieren, ohne dass ein zusätzlicher Implementierungsaufwand auf dem realen System entsteht. Der verwendete Ansatz beruht auf dem „Erfühlen der Geometrie“ und einem Aufbau mit einem KUKA-Leichtbauroboter. Hierbei wurde auf Basis der Lochgeometrie eine ungefähre Einführungspose definiert, die durch eine Selbstzentrierung des Zapfens verbessert wurde, sodass schließlich das Peg-in-Hole durchgeführt werden konnte. Der konkret implementierte Algorithmus kann in [11] detailliert nachgelesen werden.

Somit kann das hier vorgestellte Konzept wiederum Anwendung finden, um ein inhärent intelligenteres und sicheres robotisches System für Montageaufgaben zu schaffen.

5.2.3.2 Weiterführende Konzepte in der industriellen Robotik

Neben einer inhärent soften Regelung gibt es natürlich noch weitere Methodiken, die in diesem Konzept verankert sind, um eine intuitive und sichere Interaktion mit solchen komplexen Systemen zu gewährleisten.

Eine Erweiterung des zuvor genannten Peg-in-Hole-Problems kann z. B. auch mit Hilfe der hier vorgestellten Force-Feedback-Eingabegeräte verknüpft zu einer kooperativ durchzuführenden Aufgabe werden. Mittels des PHANToM Device (Kap. 4.2.4) kann die Endeffektor-Pose vorgegeben und die auf das Peg wirkenden Kräfte können auf das Gerät projiziert werden, sodass der Nutzer eine direkte Rückkopplung in dieser Fügeaufgabe bekommt.

Eine ähnliche Aufgabe, aber eine andere Herangehensweise stellen virtuelle Sicherheitsbarrieren dar. So können in parallel ausgeführten Simulationen verschiedene Robotertrajektorien zunächst am DT getestet werden, um zu erkennen, ob kritische Arbeitsräume überschritten werden, in denen sich evtl. Menschen aufhalten oder aufhalten könnten. Solche „Active Constraints“ oder „Virtual Fixtures“ können dann genutzt werden, um das gesamte System sicherer zu gestalten.

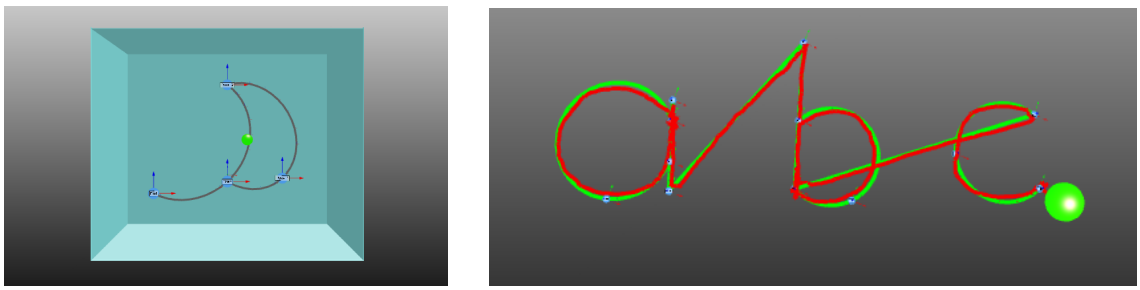
5.3 Weitere Anwendungen des Digitalen Zwillings im Rahmen von Industrie 4.0

Den Einsatz von DTs bei der Entwicklung neuer (robotischer) Systeme sollte man im entsprechenden Entwicklungsprozess fest etablieren. Zur Kopplung der Entwicklungen über den gesamten Wertschöpfungsprozess hinaus haben sich zum einen die visuelle Aufbereitung in einer VR-Umgebung, zum anderen eine Schnittstelle zum Robot Operating System (ROS) als nützliche Ergänzung erwiesen.

5.3.1 Assistenzsysteme zur Entwicklung und Nutzung von Digitalen Zwillingen

Assistenz als solche kann auf verschiedene Arten geschehen. Gerade im Bereich der visuellen Assistenz sind bereits viele Aspekte in dieser Arbeit herausgestellt worden, die Parametervisualisierungen in Form von HUDs (Kap. 4.2.5.3) oder Holographische Projektion (HP) (Kap. 4.2.5.4).

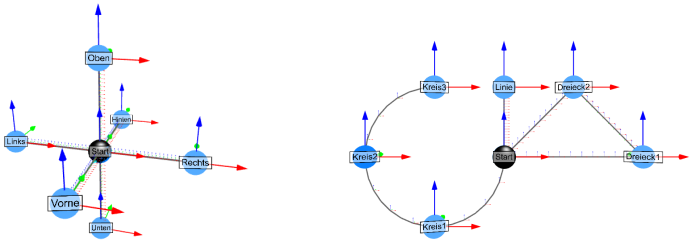
Neben solchen visuellen Assistenzsystemen kann mit Hilfe der in dieser Arbeit präsentierten Kraftrückkopplung durch das VTB Assistenz erzeugt werden. Hierbei werden Assistenzkräfte auf haptische Eingabegeräte gelegt, die das Verhalten des Proxys verändern. Der Nutzer wird dann in seinen Bewegungen durch die Simulation und die generierten Assistenzkräfte „geführt“. Hierbei ist exemplarisch (a) die Navigation zu einem Punkt und (b) die Navigation auf einem Pfad modelliert worden. Dem Nutzer kann so durch die Simulation und anziehende oder abstoßende Kräfte assistiert werden. In Abb. 5.22 sieht man vordefinierte dreidimensionale Pfade, die haptisch unterstützt abgefahren werden können.



(a) *Pfadverfolgung* ***(b)*** *Schreiben*

Abb. 5.22: *Haptische Assistenzsysteme*

Gekoppelt werden kann ein haptisches Assistenzsystem auch mit der Gestenerkennung. Hierbei bieten sich zunächst simple Gesten wie eine Bewegung in die verschiedenen Raumrichtungen an, bevor komplexere Gesten durchgeführt werden. Realisiert wurde dies hier mit dem PHANToM Device. Eine Geste gilt als erfolgreich erkannt, wenn der gesamte Pfad vollständig absolviert wurde (siehe Abb. 5.23).

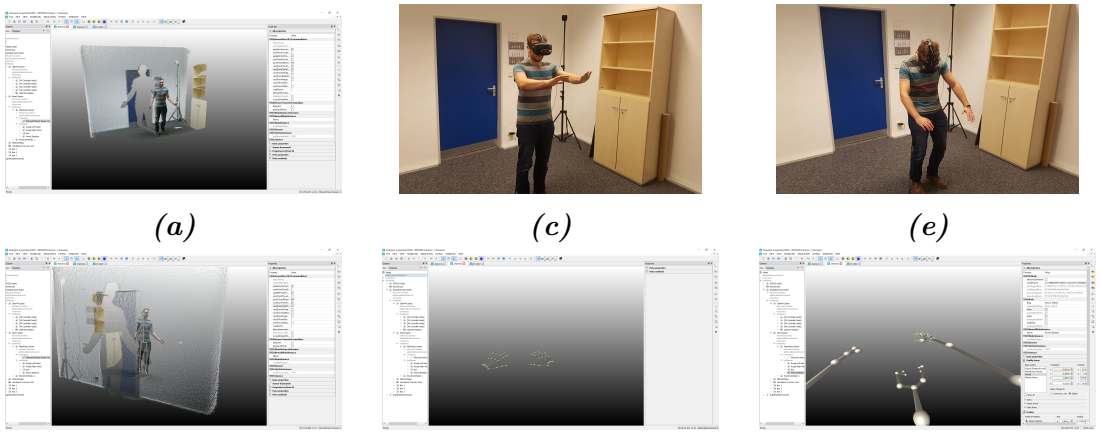


(a) simpel *(b) komplexer*

Abb. 5.23: *Haptische Gestenerkennung*

5.3.2 Das Holodeck: Eine immersive, kooperative Entwicklungs- und Interaktionsplattform

Um diese Ansätze weiter zu verfolgen und zu optimieren, wurde in dieser Arbeit eine ganzheitliche Entwicklungsumgebung in Form eines sog. Holodecks geschaffen, das sowohl in der Entwicklungsphase als *virtuelle Experimentierumgebung* als auch in der Fertigungs- und Nutzungsphase als *interaktive Test-, Evaluation- und Validierungs-Plattform* dienen soll. Durch eine Fusion von realen und virtuellen Daten, gepaart mit intuitiven Nutzerschnittstellen, gekoppelt mit dem realen System über ROS, kann dieses Holodeck genutzt werden, um mit und durch den DT neue Nutzerschnittstellen zu erschaffen, prädiktiv verschiedene Aktionen zu testen oder auch ganze Systeme durch die VR zu steuern und zu optimieren. Hierbei stehen neben einer natürlichen Interaktion auch die visuelle Aufbereitung und Fusion von (Sensor-)Daten im Fokus, um dem Nutzer das nötige Gefühl der Immersion und Präsenz in dieser Mixed Reality zu verschaffen.



(a) *(c)* *(e)*

(b) *(d)* *(f)*

Abb. 5.24: *Holodeck Trackingmodalitäten: Headtracking (HTC Vive) + Skelett (Kinect v2) + Handtracking (Leap Motion) → Immersion, Embodiment, Interaktion*

In diesem Rahmen werden verschiedene Modalitäten gekoppelt:

- stereoskopische Visualisierung,

- Tracking-Kombination und Fusion des Kopf-Trackings (der HTC Vive), Hand-Tracking (der Leap Motion) und Körper-Tracking (der Kinect v2) (vgl. Abb. 5.24),
- Kommunikation mit robotischer Hardware mit Hilfe der ROS-Middleware,
- Interaktionsframework innerhalb des stereoskopisch gerenderten Holodecks (vgl. Abb. 5.26).

Durch eine modell- und anwendungsunabhängige Implementierung durch eine `HolodeckExtension`, in der die Kalibration der verschiedenen Sensoren zueinander durchgeführt wird und in der auch alle Visualisierungen (temporär) dem Modell hinzugefügt werden, entsteht eine *interaktive Entwicklungs- und Interaktionsplattform* für nahezu alle VTB-basierten Anwendungen. Neben der Kombination vieler im Rahmen dieses Konzeptes entwickelter Eingabe- und Trackingmethoden steht hier vor allem die Fusion von Realität und Virtualität im Vordergrund.

5.3.2.1 Avatare und Interaktion

Zusätzlich ist die Möglichkeit des „Avataring“ geschaffen. Dies beinhaltet die Visualisierung aller im System befindlichen Teilnehmer in Form eines Kinect-Skeletts sowie die Möglichkeit, mit diesem Avatar durch die Simulation zu laufen, ohne diese zu beeinflussen. Auch die Chaperone-Linien der Umgebung zeigen zum einen die Grenzflächen des erlaubten VR-Raumes und zum anderen eine Gitterstruktur realer Objekte. So lässt man die Realität in die Virtualität ineinander übergehen und muss nur für die Konsistenz dieser zwei Welten sorgen.

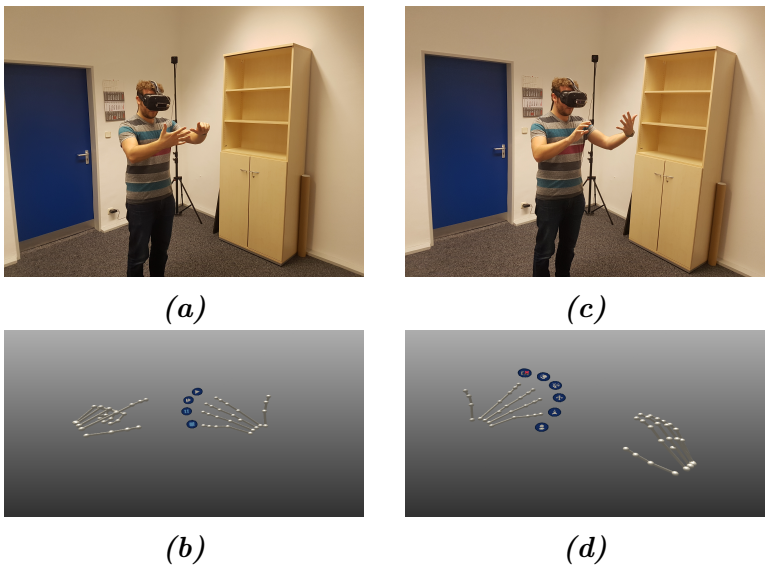
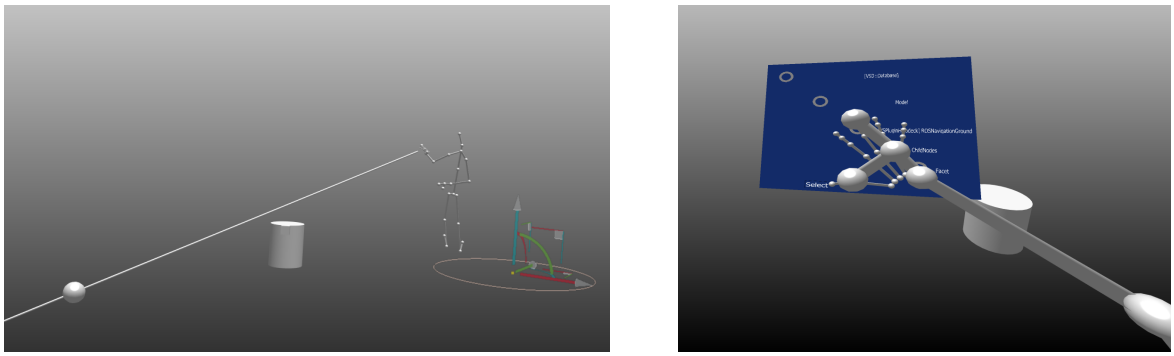


Abb. 5.25: *Holodeck: Interaktionsframework*

Für die Interaktion mit den DTs und dem VTB werden dynamische Menüs an den Händen angezeigt. In Abb. 5.25 sieht man, dass die rechte Hand mit den Simulationssteuerelementen (c, d) und die linke Hand mit dynamischen Modellelementen (a, b) verknüpft ist. Durch die Geste des Drehens der jeweiligen Hand (Handfläche nach innen

drehen) erscheint das jeweilige Menü, das mit dem Zeigefinger der anderen Hand bedient werden kann. So lässt sich die Simulation starten, stoppen, der eigene Avatar durch Teleportation durch das VTB bewegen, oder mit der anderen Hand ein selektiertes Objekt verschieben, skalieren usw., je nachdem, welchen Objekttyp man gerade selektiert (vgl. Abb. 5.25).

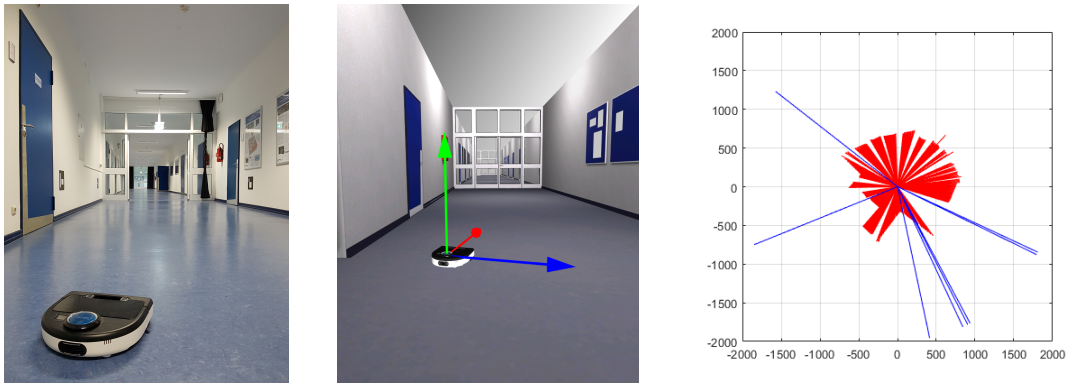


(a) *(b)*

Abb. 5.26: *Holodeck-Interaktionmodalitäten: Selektion sowie Teleportation durch Ray-Tracing (sowie Schnittpunktberechnung mit Objekten oder dem Boden) und direkter Zugriff auf die Datenbank (VSD) in stereoskopisch gerenderten Fenstern, die auch mit den eigenen Händen bedient werden können.*

Dadurch, dass dieses Holodeck nicht speziell für einen Anwendungsfall implementiert wurde, sondern allgemein im Sinne dieses Konzeptes genutzt werden kann, kann man beliebige andere Digitale Zwillinge hier untersuchen, entwickeln, optimieren, überwachen oder steuern.

5.3.2.2 Haushaltsrobotik: mobile Staubsaugroboter



(a) RT *(b) DT* *(c) Sensordatenanalyse*

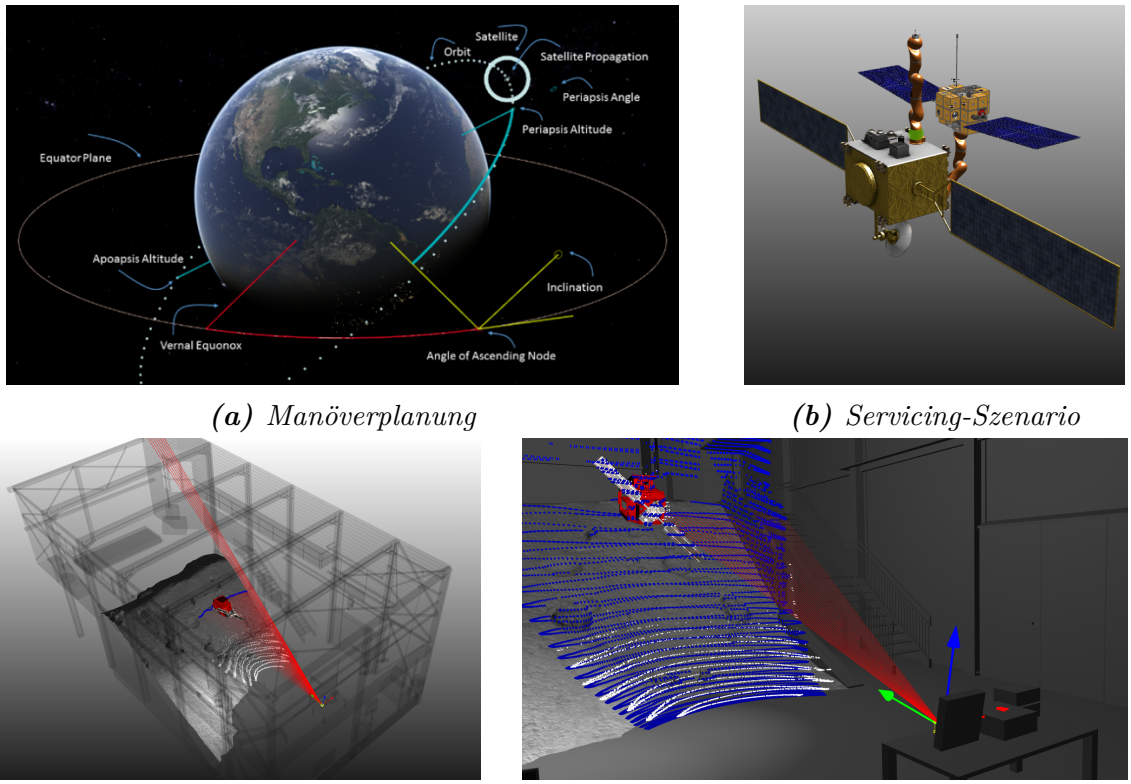
Abb. 5.27: *Holodeck: Anwendung Haushaltsrobotik (Neato Botvac 85D)*

So ist es möglich, einen mobilen Roboter wie den Staubsaugroboter Neato BotVac 85D

mittels eines Wi-Fi-Mods [db] mit ROS zu verbinden [dc] und diesen daraufhin im Holodeck zu nutzen, alle Daten bereitzustellen und mit diesem zu interagieren.

5.3.3 Weltraumanwendungen: Konstruktions- und Manöverplanung

Als ein weiteres Beispiel seien hier noch Weltraumanwendungen genannt, die oft aufgrund ihrer Einmaligkeit (Losgröße von eins) nahezu ausschließlich auf Simulationen zurückgreifen müssen. Hierbei können dann in einem solchen Holodeck eine Weltraummission erlebbar gemacht werden und physikalische Parameter besser analysiert werden. Des Weiteren können Visualisierungen genutzt werden, um intuitiven Zugang zu extrem komplexen Trajektorien und Manöverplanung zu erhalten (siehe Abb. 5.28a), geeignete Sensoren anhand real und virtuell existierender Mock-ups auszuwählen (Abb. 5.28c) oder eine Starrkörpersimulation geplanter Missionen durchzuführen (Abb. 5.28b).



(a) Manöverplanung *(b) Servicing-Szenario*

(c) Sensorsimulation

Abb. 5.28: *Weltraumanwendungen: ein Überblick*

Gerade hier lassen sich auch die zuvor beschriebenen Inspektionstools der Dynamiksimulation nutzen, um Rückschlüsse auf die Erfolgswahrscheinlichkeit der Mission ziehen zu können. Beispielsweise ist es möglich, das in Kap. 4.3.2.2.2 präsentierte Tool zur Analyse des Trägheitstensors auch auf Servicing-Szenarien anwenden. Hiermit kann, wie in Abb. 5.29 dargestellt, mögliches „Trudeln“ des Satelliten aufgrund von Bewegungen des Servicing-Roboters untersucht werden.

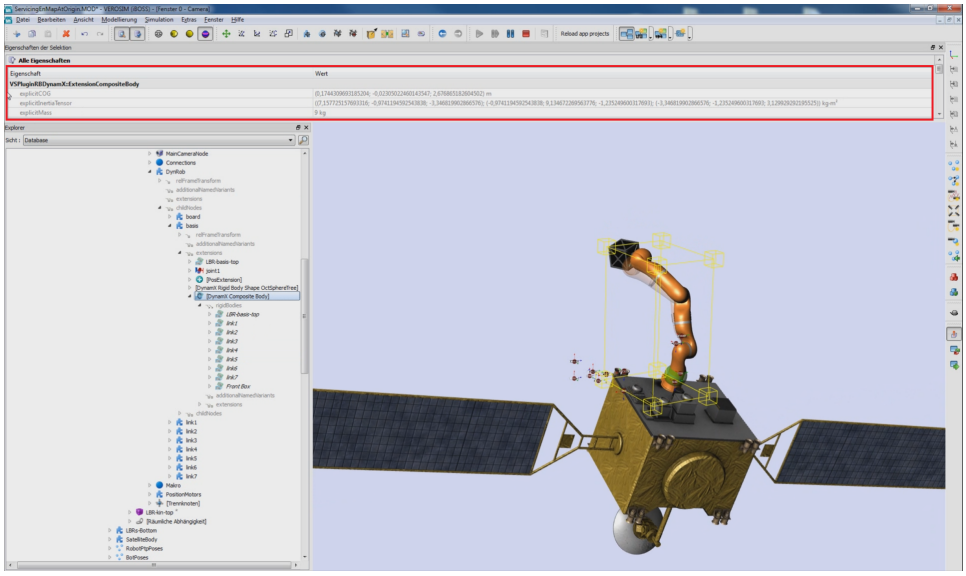


Abb. 5.29: *Weltraumanwendungen: Satellitenservicing. Untersuchung der Gesamtmasse, des Schwerpunkts und des Trägheitstensors dynamisch zusammengesetzter Starrkörper während des Servicings (vgl. Kap. 4.3.2.2.2 und Abb. 4.43)*

Durch die Anwendungsunabhängigkeit des Konzeptes ist es auch möglich, anwendungsübergreifende Szenarien im VTB auszuprobieren. Ein Beispiel dafür stellt die Nutzung des CENTAURO-Roboters als mobilem Explorationsroboter in Weltraumanwendungen dar (vgl. Abb. 5.30).

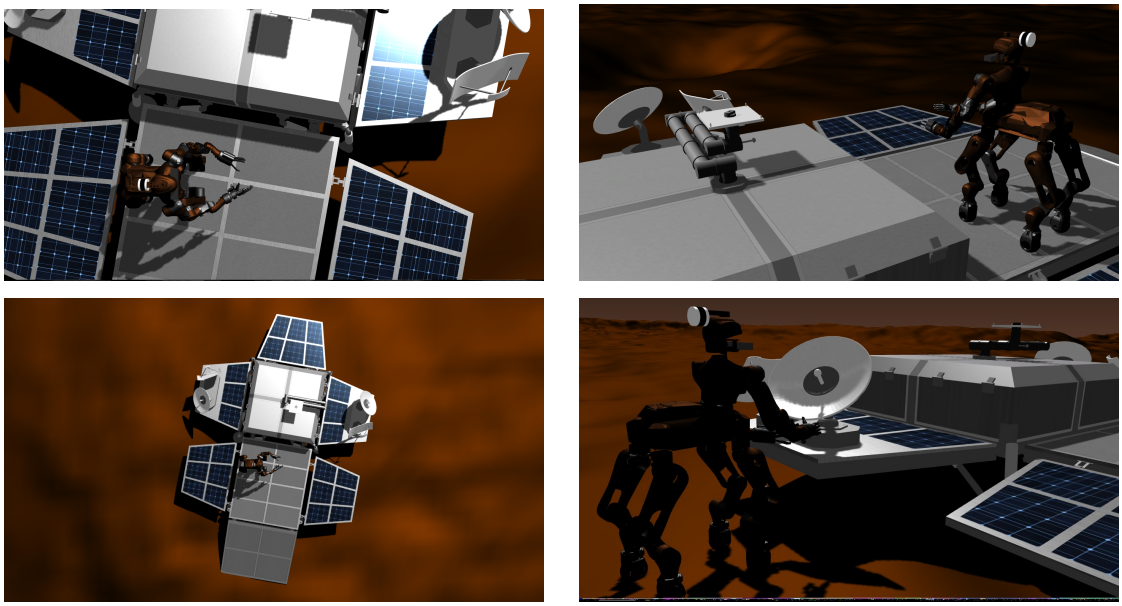
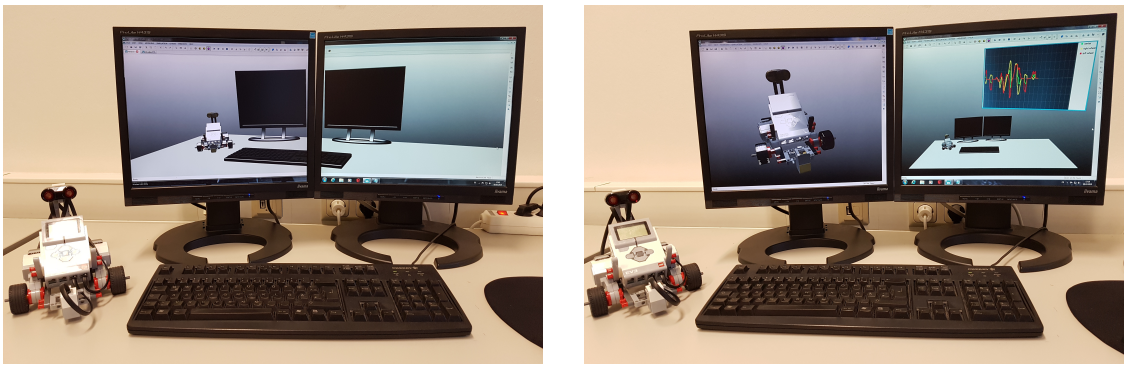


Abb. 5.30: *Anwendungsübergreifende Nutzung des DTs: Virtelle Erprobung eines Einsatzes des CENTAURO-Roboters als extraterrestrischem Explorationsroboter*

5.3.4 Engineering 4.0: Erweiterung des Büros um die Dimension der Virtualität

Der aktuelle Trend der Realisierung Digitaler Zwillinge für Prozesse, Maschinen auch für vollständige robotische Systeme erfordert die vernetzte und symbiotische Zusammenarbeit von Entwickler, realer Maschine und Modellierungs- und Programmierumgebung. Gerade in der Entwicklung von komplexen Systemen wie mobilen Robotern ist ein tiefes und grundsätzliches Verständnis der einzelnen Komponenten sowie des Gesamtsystems von hoher Bedeutung. Hierfür reichte es bisher oft aus, verschiedene Aspekte in Simulationen am PC auszuprobieren und daraufhin am realen System zu integrieren oder zu verwerfen. Viel wichtiger ist jedoch das Verständnis, warum etwas besser funktioniert als anderes. Der Entwickler benötigt also stets Einblick in das System und die Daten, um intuitiv und direkt richtige Entscheidungen treffen zu können.



(a) *(b)*

Abb. 5.31: *Engineering 4.0 mobiler Roboter am Beispiel des LEGO EV3 Mindstorm-Roboters*

Mit dem DT als Mediator kann auch der Arbeitsplatz eines Ingenieurs durch die Dimension der Virtualität erweitert und mit dieser fusioniert werden. Hierzu wird mit „off-the-shelf“ VR-Hardware, einem VTB sowie einem ROS-basierten realen mobilen Roboter das „VR-Büro“ bzw. der „Schreibtisch der Zukunft“ entwickelt. Insbesondere zum Erproben oder für die Lehre bieten sich die LEGO EV3-Roboter an, um spielend die Robotik zu erleben. Auch hier findet das entwickelte Konzept Anwendung, bei dem ein Digitaler Zwilling des EV3-Roboters gepaart mit einem Digitalen Zwilling des eigenen Arbeitsplatzes im VTB erstellt wird (vgl. Abb. 5.31). Hier kann dann Datenauswertung, Programmierung oder auch Optimierung des LEGO-Roboters anhand des Digitalen Zwillings passieren. Interessant sind dann insbesondere Konstellationen, in denen simulierte Sensordaten den realen Roboter zu Aktionen zwingen oder reale Sensordaten dem Digitalen Zwilling Objekte in den Weg stellen, die dann automatisiert in der Simulation erscheinen.

5.4 Anforderungsrückblick

Auf Basis der genannten Anwendungen folgt hier eine Gegenüberstellung der Anforderungen und Bewertung der geschaffenen Lösungen mit der Anwendung des Konzeptes, das in dieser Dissertation vorgestellt wurde.

Anf. 1: Entwickler- und nutzerzentriert

Der Einsatz des Konzeptes von der ersten Idee über die Entwicklung, das Training, die Fertigung und im Nutzungsprozess positioniert den Menschen als zentrales Element. *Über den gesamten Lebens- und Schaffungsprozess* wird mit Hilfe des Konzeptes eine optimale nutzerzentrierte Schnittstelle zu komplexen Systemen geschaffen.

Anf. 2: Kollaborationsmöglichkeit

Ebenfalls werden die Möglichkeiten zur Kollaboration und Kooperation mit robotischen Systemen geschaffen, welche durch den DT sicher, intuitiv und interaktiv gestaltet werden können. Durch die multimodale Integration diverser Eingabegeräte ist es möglich, mit mehreren Personen an einem System direkt zu arbeiten. Des Weiteren bietet das Repository Digitaler Zwillinge und das zentrale Weltmodell zentrale Knotenpunkte für das Sammeln und Austauschen von Daten.

Anf. 3: Immersion

Immersion und Präsenz werden durch die unterschiedlichen Ein- und Ausgabemodalitäten geschaffen. Durch das Embodiment der optischen Trackingsysteme, kombiniert mit wissenschaftlichen Parametervisualisierungen und Eingabegeräten mit Force Feedback, verstärkt sich das Gefühl der Immersion in VR. Gerade die verschiedenen Visualisierungen erschaffen ein intuitiv verständliches und immersives „Erleben“ des digitalen Systems.

Anf. 4: Kognitiv, intuitiv und sicher

Der DT als Mediator und Abstraktionsebene für den Nutzer lässt das robotische System inhärent sicherer werden. Durch Gestenerkennung sowie virtuelle Eingabegeräte und Interaktionsframeworks wird ein intuitiv verständlicher Zugang zu diesen Systemen ermöglicht, sodass die menschliche Kognition genutzt und eingesetzt werden kann, um kritische Situationen zu erkennen und auftretende Probleme optimal zu lösen.

Anf. 5: Integration aktueller Technologien

Neben dem VTB an sich werden im Rahmen des präsentierten Konzeptes aktuelle Technologien insbesondere aus dem Bereich VR eingeführt. Durch die Modularität können diese auch in Zukunft leicht erweitert werden. Zusätzlich dazu werden mit den präsentierten optischen und haptischen Eingabegeräten auch in diesen Forschungsbereichen aktuelle Forschungsfragen und Technologien genutzt, um einen intuitiven Zugang zu komplexen Systemen zu ermöglichen.

Anf. 6: Modular

Das genutzte Simulationssystem VEROSIM ist dank der Mikrokernel-Architektur schon von Natur aus modular aufgebaut. Darauf aufbauend wird innerhalb dieser Arbeit insbesondere mit Plugins und dem I/O Board ein modulares kombinatorisches Zusammenspiel verschiedener Module ermöglicht. Das wird zusätzlich im Rahmen der ROS-Schnittstelle mit einer Vielzahl von externen Bibliotheken, Programmen und Entwicklern zugänglich gemacht, sodass man interne und externe Frameworks schnell, einfach und modular kombinieren kann. Dank des Repository Digitaler Zwillinge ist eine Modularität der DTs möglich, sodass Roboter aus Sub-Modellen erstellt werden können und die Simulation die optimale Zusammenstellung aus z. B. Basis, Beinen, Oberkörper, Armen, Händen und Sensoren ermöglicht. Durch die Einbettung des DTs ins VTB sind alle Entwicklungen nie spezifisch für einen DT, sondern lassen sich direkt für weitere DTs anwenden.

Anf. 7: Universell einsetzbar, skalierbar

Auch die universelle Einsetzbarkeit und Skalierbarkeit wird durch die Schnittstellentechnologien von ROS, UDP sowie die Integrationsmöglichkeit echtzeitkritischer Module (wie XBotCore) abgedeckt. Das spiegelt sich in der Standardisierung der Schnittstellen und durch den implementierten URDF-Importer wider. Die Universalität zeichnet sich ferner durch die Kombination starrer und weicher Körper im Rahmen der Soft-Robotik-Simulation aus. Auch anhand der aufgezeigten Anwendungen sieht man die Möglichkeiten, das Konzept in verschiedenen Anwendungen (von mobilen LEGO-Robotern über Katastrophenrobotik bis zu Weltraumszenarien) zu nutzen.

Anf. 8: Inspektionsmöglichkeit und Validierbarkeit

Die geforderte Inspektion und Validierbarkeit sind zum einen durch das direkte Monitoring von realen Systemen in deren Einsatz durch die ROS-Schnittstelle und die implementierten Visualisierungen in VR, AR oder MR möglich, zum anderen dadurch, die physikalischen Grundlagen der Digitalen Zwillinge eingehend zu untersuchen bis auf die analytische Ebene. Eine Validierung ist schlussendlich möglich, indem man reale und virtuelle Benchmark-Szenarien nutzt, welche auch im DT-Repo vorhanden sind.

Anf. 9: Technologietransfer und Kostenreduktion

Ein Technologietransfer aus der Entwicklungs- in die Nutzungsphase wird mit diesem Konzept erreicht, da die gleichen Simulationstechnologien sowie Ein- und Ausgabegeräte genutzt werden können. Des Weiteren lassen sich interne und externe Funktionalitäten mittels ROS koppeln, sodass schon vorhandene ROS-Module nahtlos weiter genutzt werden können und nicht neu implementiert werden müssen. Die Plattformunabhängigkeit und die implementierten Ansätze zu Python und ROS2 führen zusätzlich zu einer Reduktion des Implementations-Overheads, da je nach Entwickler, Bedürfnissen oder Anforderungen das gewünschte System genutzt werden kann. Durch die Implementierungsstruktur der ROS-Schnittstelle ist auch die Möglichkeit geschaffen, weitere Kommunikationsprotokolle/Middlewares wie MQTT oder OPC UA analog als „Engines“ zu integrieren.

Anf. 10: Echtzeitfähigkeit

Auch wenn im Rahmen dieser Arbeit nicht zwingend „harte“ Echtzeitfähigkeit eingehalten wird, so ist es möglich, echtzeitkritische Dlls direkt in das VTB zu integrieren (vgl. XBotCore). Des Weiteren wurde im Rahmen des haptischen Renderings eine Updaterate von $f_{FF} > 1$ kHz erreicht und gezeigt, dass die Integration von Force Feedback möglich und sinnvoll ist.

Anf. 11: Datenabgleich

Der DT wird im Rahmen dieser Arbeit als Datenquelle und -senke genutzt, um mit dem realen System direkt oder indirekt mittels ROS als Middleware zu interagieren. Der Datenabgleich erfolgt mittels des zentralen Weltmodells, welches sowohl für das reale System als auch für den Digitalen Zwilling genutzt und gefüllt wird.

Anf. 12: Interaktion und Interaktivität

Die vielleicht wichtigste Anforderung stellt (neben der lebenszyklusübergreifenden Nutzung) die Interaktion und Interaktivität dar. Diese bezieht sich hierbei auf die Interaktion des Nutzers *mit* dem DT sowie die Interaktion des Nutzers mit dem realen System *durch* den DT. Die Kombination aus Nutzerschnittstellen, integrierten Frameworks, Hardwareschnittstellen und deren modulare Kombination zu einem Gesamtsystem lassen jeden Nutzer interaktiv – egal ob teleoperiert, semi-autonom oder autonom – mit komplexen Systemen interagieren.

KAPITEL 6

Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit werden die Methodik, das Konzept, die Umsetzung und die Anwendung des Digitalen Zwillings als lebenszyklusübergreifendem Mediator zwischen Mensch und Maschine erarbeitet. Der DT, eingebettet ins Virtuelle Testbed (VTB), als Bindeglied, Mediator und Abstraktionsebene zwischen Nutzer und (robotischem) System, ist hierbei zentraler Baustein, um ein symbiotisches Miteinander von Mensch und Maschine zu erschaffen. Mit Hilfe der Digitalen Zwillinge kann dabei eine Kopplung und Fusion von Realität und Virtualität helfen, einen intuitiven Zugang zum finalen System über alle Prozessschritte hinweg und darüber hinaus zu entwickeln.

Mit und durch die geschaffene Struktur (vgl. Abb. 3.4) ist es nun möglich:

- modular und anwendungsunabhängig komplexe Systeme visionär zu gestalten, zu entwickeln und zu integrieren,
- den Digitalen Zwilling lebenszyklusübergreifend als zentrale Schnittstelle und Mediator zu etablieren und effizient in allen Entwicklungs- und Nutzungsphasen einzusetzen,
- durch die Visualisierung (in VR, AR oder MR) sowie (klassische, controllerbasierte, optische und insb. haptische) Eingabegeräte, Systeme intuitiv bedienbar bzw. erlebbar werden zu lassen,
- durch die ROS-Schnittstelle eine direkte, standardisierte und unkomplizierte Kommunikation mit realer Hardware und/oder externen Softwaremodulen zu schaffen und durch den URDF-Datenstandard den Modellaustausch vereinfachen,
- durch die integrierten Funktionalitäten ins VTB Einblick in Daten und physikalische Prozesse (von realen und virtuellen Systemen) zu ermöglichen und durch geeignete Visualisierungen den Nutzer sinnvoll und intuitiv zu unterstützen,
- Entwickler- und Nutzerschnittstellen zu entwickeln und zu nutzen – in jeder Entwicklungsphase und anwendungsübergreifend – sowie
- eine optimale Kopplung aus Realität und Virtualität zu schaffen – im Bereich der Eingabegeräte, Datenquellen, Integration usw.

Um alle genannten Funktionalitäten zu ermöglichen, wird eine flexible und nutzeroptimierte Ein- und Ausgabe bereitgestellt, welche neben diversen Eingabegeräten auch

unterschiedliche Visualisierungen sowie Kraftrückkopplung beinhaltet. Mit dem DT-Interface wird eine zentrale Hardwareschnittstellentechnologie bereitgestellt, um mit einer Vielzahl von robotischen Systemen direkt zu interagieren bzw. für diese als Datenquelle oder -senke zu dienen. Die Einbettung des DTs in das VTB führt dazu, dass alle Entwicklungen im Rahmen dieser Arbeit nicht spezifisch für einen einzigen DT, sondern generell für DTs an sich implementiert wurden. Notwendige Erweiterungen innerhalb des VTBs führen dazu, dass insbesondere ein standardisierter Daten- und Modellaustausch sowie die Evaluation und Inspektion (meist physikalischer) Prozesse intuitiv zugänglich gemacht werden. Schlussendlich können alle diese Funktionalitäten im Digitalen Zwilling zusammengeführt und so ganzheitliche Systeme aus Entwickler/Nutzer, DT und physischem Asset geschaffen werden, jeweils unter Zuhilfenahme der Input/Output-Funktionalität und der modularen Kombination der unterschiedlichen Komponenten.

Insbesondere am Beispiel der Katastrophenrobotik (Kap. 5.1) konnten diese Aspekte erfolgreich angewandt werden. Gerade hier spielt eine intuitive und effiziente Nutzerschnittstelle eine zentrale Rolle. Um in kritischen Situationen die richtigen Entscheidungen treffen zu können, hat der Operator durch den DT alle notwendigen Daten im Blick und kann alle Funktionalitäten des DTs auch in-the-loop nutzen.

Je nachdem, aus welcher Richtung man sich der Problematik nähert, ergeben sich viele verschiedene Möglichkeiten, den DT nun als (lebenszyklusübergreifenden) Mediator einzusetzen.

Der DT als Mediator in der Entwicklung komplexer (robotischer) Systeme

Durch die geschaffene Struktur ist es möglich, in VR intuitiv innerhalb eines Holodecks/„Seeing Space“ neue Ideen und Systeme kollaborativ zu erleben und zu entwickeln. Hier können cyber-physische Systeme erschaffen werden, die als Knoten im Internet der Dinge die Industrie 4.0 formen. Mit Hilfe des VTBs und dessen Schnittstellen kann unabhängig von finalen Systemkomponenten entwickelt werden. Kooperativ können so Komponenten, Algorithmen (nur auf Basis von Daten des DTs), oder ganzheitliche Systeme entwickelt, optimiert und validiert werden. Insbesondere intuitive Nutzerschnittstellen zur Steuerung und Visualisierung des Assets können schon frühzeitig mit und für den DT entwickelt werden und sowohl dem Entwickler als auch später dem Nutzer als Assistenzsystem dienen.

Der DT als „Enabler“ intelligenter Gesamtsysteme aus Nutzer, DT und physischem Asset

Durch die geschaffene Struktur ist es möglich, den DT (i) als Assistenzsystem, (ii) zum Monitoring und (iii) zur Prädiktion einzusetzen.

Zur Unterstützung des Nutzers (bzw. als Assistenzsystem) kann direkt mit dem realen System durch den DT kommuniziert werden, entweder durch direkte (nutzerzentrierte) teleoperative Steuerung oder durch (semi-)autonome Aktionen. Durch den DT wird dabei mit dem virtuellen System die reale Umwelt erlebt und dabei ist der Zwilling keine Einbahnstraße, sondern visuelles wie haptisches Feedback hilft, die richtigen Entscheidungen zu treffen. Außerdem können zur Entscheidungsunterstützung virtuelle und reale

Daten kombiniert oder fusioniert werden. Die Nutzerschnittstelle wird hierbei mit und durch den DT gestaltet und kann stets optimal an den jeweiligen Nutzer oder Operator angepasst werden.

Primär im industriellen Szenario sind Themen wie Predictive Maintenance, Structural Health Monitoring, oder einfach die Statusüberwachung von Maschinen (Monitoring) interessant. Mit der entwickelten Struktur kann eine laufende Maschine durch den DT erlebt werden und das im fließenden Übergang von Realität zu Virtualität mittels VR, AR oder MR, entweder aus der Ferne oder direkt bei der Interaktion mit dem physischen Asset. Mehr aus der mobilen Robotik bekannt, kann der DT zur Szenario-Interpretation genutzt werden, um online einen besseren Überblick zu erhalten oder auch nach der Mission Rückschlüsse auf die nächste ziehen zu können.

Einen besonderen Stellenwert in der Entwicklung und im Betrieb komplexer Systeme hat die Prädiktion. Nun ist es möglich, online in das VTB wechseln und mit dem DT die „What if...?"-Fragen zu stellen, den DT intern (also auf der Roboterhardware) oder extern als mentales Modell nutzen. Des Weiteren können die heterogenen, verteilten, Komponenten-basierten Systeme auf Basis eines zentralen semantischen Weltmodells intelligent zu einem Gesamtsystem kombiniert werden (z. B. zur Schwarmintelligenz).

Der DT als Mediator in der Planung und im Operator-Training

Durch oder anhand des DTs lässt sich vorab die Missions- und Aufgabenplanung durchführen. Zum einen kann die Wahl der Ein- und Ausgabegeräte – also die Nutzerschnittstelle – vorab festgelegt und für die Anwendung und für den Operator optimiert werden. Mit denselben oder anderen zur Verfügung stehenden Eingabegeräten kann man den DT steuern und schon vorab mit dem DT-Repository Benchmarks am DT durchführen. Zum anderen können Algorithmen, Motion Primitives etc. vom DT direkt (aus dem Entwicklungsprozess in den laufenden Roboterbetrieb) auf den RT projiziert werden.

Der DT als Ausgangspunkt/Baustein von KI

Durch die standardisierten Schnittstellen des DT kann dieser nun als Datenquelle und -senke für beliebige externe und interne Datenverarbeitungsmodule genutzt werden. Dies kann das Training von KI-Algorithmen sowie im VTB automatisiert gelabelte Daten umfassen, um so neuronale Netze zu trainieren. Mit dem DT im VTB stehen somit Möglichkeiten zum Deep Learning bereit. Ein Beispiel ist das Transfer Learning, wo man zunächst ausschließlich mit Hilfe von Daten des DTs lernt und am Ende auf reale Systeme oder andere Situationen transferiert. Durch die Wiederholbarkeit sowie die Varianz verschiedener Szenarien des DT können generierte Daten auch zu Big Data Analytics genutzt werden.

Ausblick

Wie schon zuvor beschrieben, bietet die entwickelte Struktur des DT im VTB als Mediator viele Möglichkeiten, die schon jetzt genutzt werden können. Insbesondere die Kopplung, Fusion und Überbrückung von Virtualität und Realität, DT und RT, Input und

Output sowie VR, AR und MR öffnet viele interessante Ansätze für die Nutzung des DTs als Mediator.

Gerade die Bereiche des Deep Learnings, KI/AI und Big Data wurden bisher nur touchiert, können aber in Zukunft einen großen Mehrwert bieten. Durch die geschaffenen Schnittstellen (insb. ROS, ROS2) sind Daten des DTs sofort zugänglich und werden standardisiert ausgetauscht. Auch weitere Anwendungen im Bereich von Cobots, Manipulatoren oder mobiler unbemannter Boden-, Luft- oder Wasser-Roboter (UGVs, UAVs, UUVs, AUVs) können ohne großen Aufwand in die Struktur integriert werden. Durch die modular gewählte Implementierung sind Erweiterungen am VTB automatisch für alle Typen Digitaler Zwillinge sowie auch mit den Ein- und Ausgabemodalitäten kompatibel und können flexibel genutzt und eingesetzt werden.

Zur Weiterentwicklung der Struktur bietet es sich an, zusätzliche Schnittstellentechnologien des IoT zu integrieren (z. B. MQTT, OPC UA etc.). Hier wurden durch die modulare Implementierung alle Vorbereitungen getroffen, weitere Middlewares hinzuzufügen und den DT flexibel an diese anzubinden. Auch weitere Entwicklungen im Bereich der Ein- und Ausgabe können in die Referenzimplementierung ohne viel Aufwand integriert werden.

KAPITEL 7

Anhang

Im Folgenden sind ergänzende Informationen zu finden. Diese umfassen weitere Konzeptgraphiken bzw. die Konzeptgraphiken in unterschiedlichen Granularitätsstufen (Kap. 7) sowie zusätzliche Informationen zu Ein- (Kap. 2.3.4) und Ausgabegeräten (Kap. 2.3.4).

7.1 SysML-Konzeptgraphiken

Im Folgenden sind die Konzeptgraphiken (vgl. Abb. 7.1) in unterschiedlicher Granularität aufgeführt:

- BDD des Gesamtsystems (Vollbildversion von Abb. 3.4a)
- IBD des entwickelten Systems in hohem Detailgrad (Abb. 7.2)
- IBD des entwickelten Systems in mittlerem Detailgrad (Abb. 7.3)
- IBD des entwickelten Systems in geringem Detailgrad (Abb. 7.4), welches in Abb. 3.4b integriert wurde.

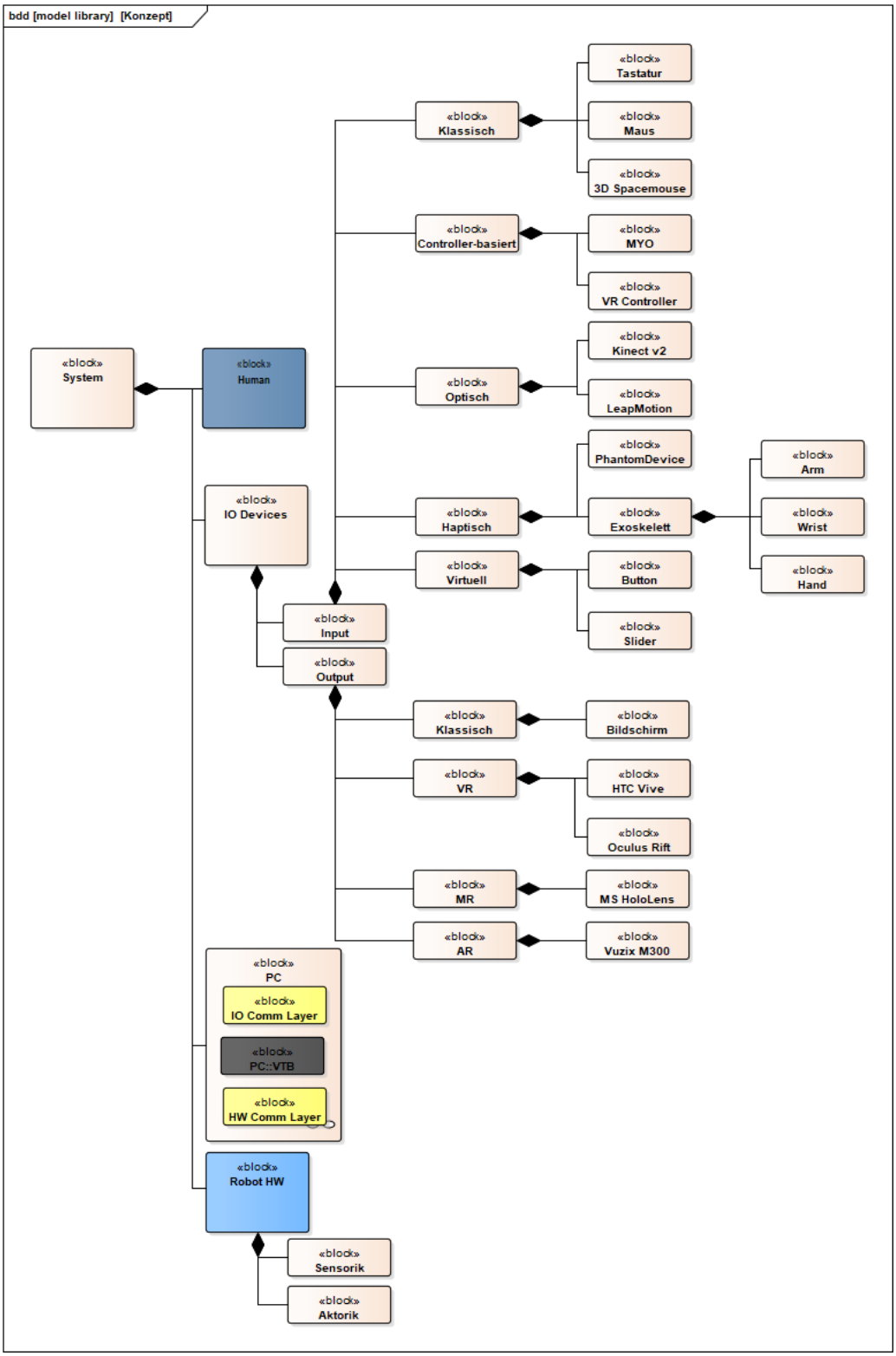


Abb. 7.1: *SysML-Konzeptgraphik als Blockdefinitionsdiagramm (BDD)*

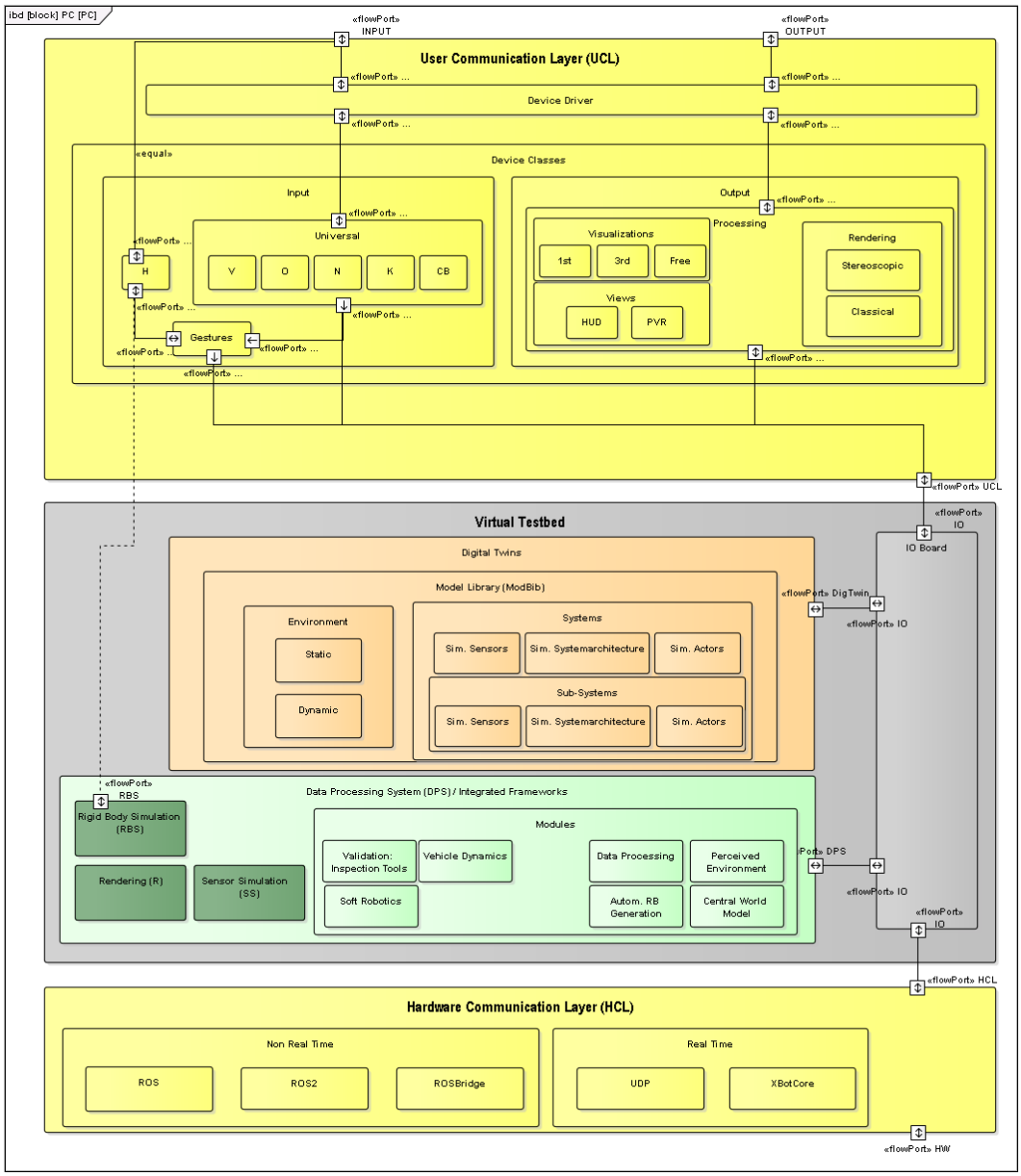


Abb. 7.2: *SysML-Graphik des PCs bzw. des implementierten Frameworks als Internes Blockdiagramm (IBD) (hohes Detaillevel)*

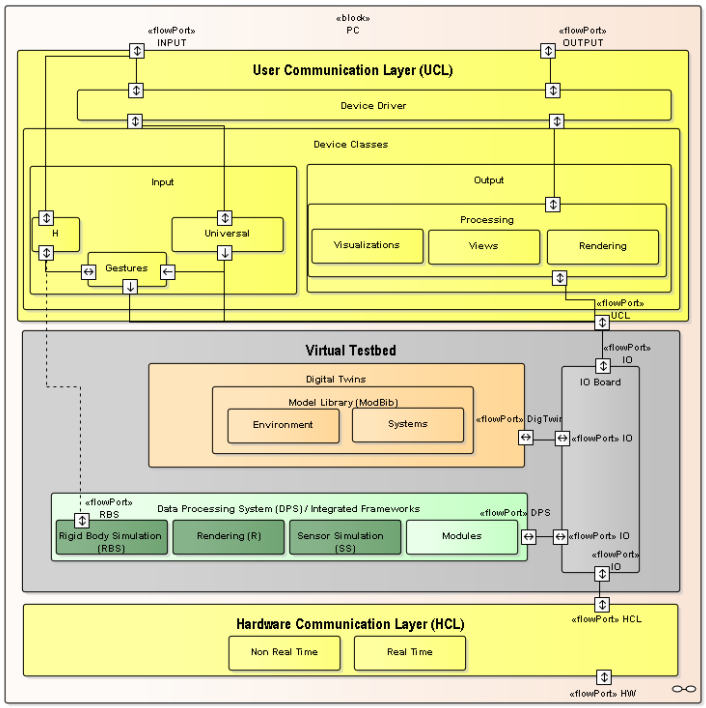


Abb. 7.3: *SysML-Graphik des PCs bzw. des implementierten Frameworks als IBD (mittleres Detaillevel)*

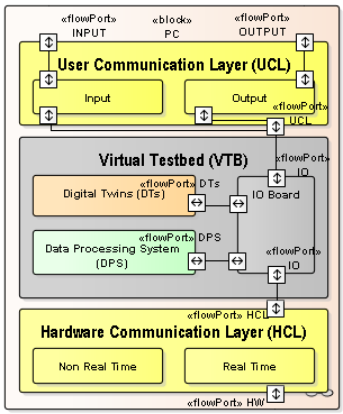


Abb. 7.4: *SysML-Graphik des PCs bzw. des implementierten Frameworks als IBD (geringes Detaillevel)*

7.2 Ein- und Ausgabegeräte

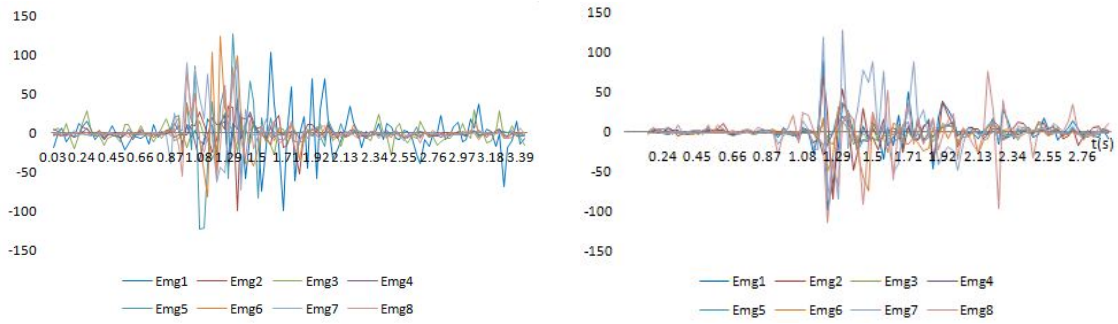
Neben den in Kap. 2.3 genannten Informationen zu Ein- und Ausgabegeräten sind im Folgenden zusätzliche Informationen aufgeführt.

7.2.1 Eingabegeräte

Zusätzlich zu den Informationen in Kap. 2.3.4 sind hier eine Probandenstudie des Myo-Eingabegerätes sowie die physikalischen Grundlagen des Geomagic Touch X zu finden.

7.2.1.1 Myo-Evaluation

Ergänzung zum controllerbasierten Eingabegerät Myo aus Kap. 2.3.4.1 und Kap. 4.2.3.4.



(a) EMG-Signal (F): Proband A *(b) EMG-Signal (F): Proband B*

Abb. 7.5: *Thalmic Labs Myo-Armband: EMG-Signale*

Zur Evaluation wurden Experimente mit mehreren Probanden durchgeführt, welche (1) die Gestenerkennungsrate, (2) die unterschiedliche Erkennungsrate von Gesten durchgeführt mit unterschiedlicher Geschwindigkeit und (3) die Wiederholgenauigkeit verschiedener Patterns beinhalten. Zusätzlich wurden allgemeine Untersuchungen der EMG-Signale durchgeführt (vgl. Abb. 7.5).

Tab. 7.1: *Myo-Experiment 1: Fehlerrate der Gestenerkennung (vgl. Abb. 2.10b)*

	(F)	(FS)	(WI)	(WO)	(TtP)
Proband 1	0	1/20	0	2/20	1/20
Proband 2	0	0	0	2/20	10/20
Proband 3	5/20	3/20	1/20	1/20	
Proband 4	0	3/20	1/20	0	1/20
Overall	6,25 %	8,75 %	2,50 %	6,25 %	15,00 %

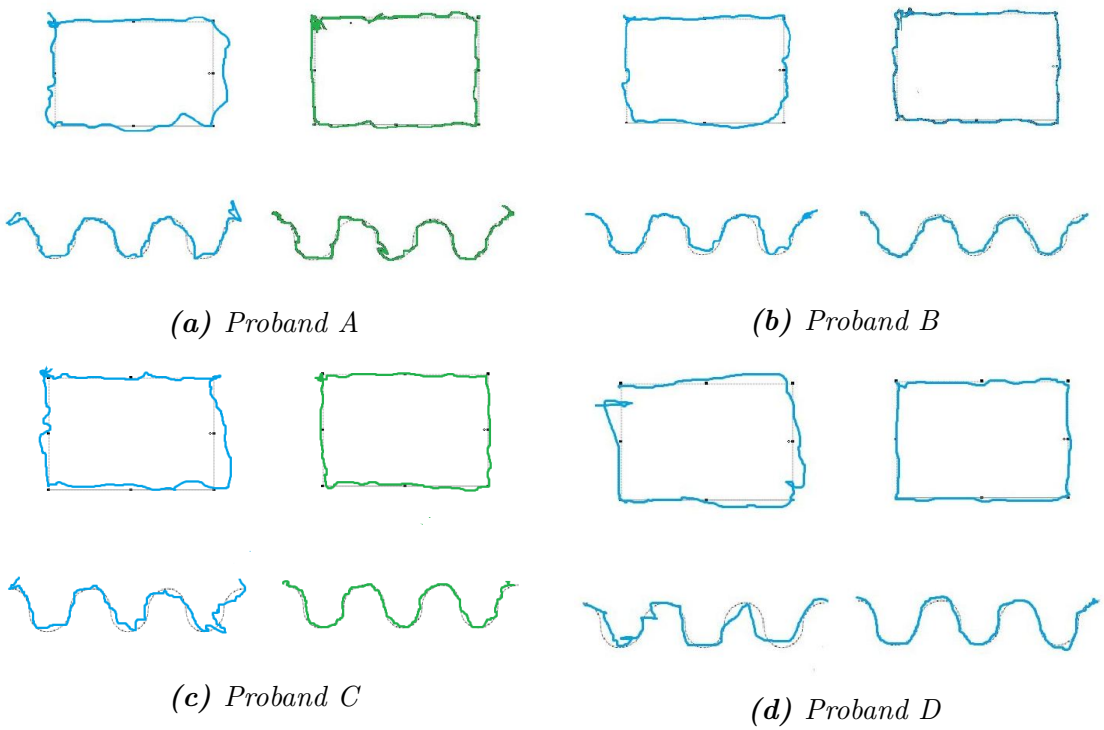
Trotz der Tatsache, dass die EMG-Signale stark probandenabhängig sind (vgl. Abb. 7.5a und 7.5b), konnte hier für die vier meistgenutzten Gesten eine Erkennungsrate von 95 % erreicht werden (ohne die TtP-Geste). Des Weiteren ist hier zu sehen, dass einzelne Probanden (Proband 3) insgesamt Probleme hatten bzw. gar nicht fähig waren, die geforderten Gesten überhaupt durchzuführen. Daher wurde Proband 3 für weitere Untersuchungen nicht hinzugezogen.

Tab. 7.2: *Myo-Experiment 2: Geschwindigkeitsabhängigkeit der Fehlerrate*

	Errors	Rec. Err.	Undetected	Common error
Langsam				
Proband 1	4/20	1	3	FS undetected
Proband 2	1/20	0	1	FS undetected
Proband 4	2/20	0	2	FS undetected
Overall	11,66 %	1	6	
Mittel				
Proband 1	4/20	0	4	FS or F undetected
Proband 2	3/20	0	3	FS undetected
Proband 4	5/20	2	3	F undetected TtP instead of FS TtP instead of FS
Overall	20 %	2	10	
Schnell				
Proband 1	5/20	2	3	WI or F undetected FS instead of WI FS instead of WI FS or F undetected
Proband 2	6/20	0	6	FS or F undetected
Proband 4	8/20	3	5	FS undetected; TtP instead of FS and F TtP instead of FS and F
Overall	31,66 %	5	14	

Wie man in Tab. 7.2 erkennen kann, ist die Erkennungsrate von $R_{slow} \approx 88$ % über $R_{medium} \approx 80$ % zu $R_{fast} \approx 68$ % degradiert.

Im Bereich der Positionsgenauigkeit wurden ein Rechteck sowie eine Sinuskurve nachgezeichnet (vgl. Abb. 7.6). In Abb. 7.6a sieht man links jeweils den ersten und rechts den letzten von drei Versuchen des ersten Probanden. Insgesamt waren alle Probanden mit ihren Ergebnissen nach drei Versuchen zufrieden, bemängelten lediglich den benötigten Bewegungsradius zum Nachzeichnen der Pfade.



(a) Proband A *(b) Proband B*

(c) Proband C *(d) Proband D*

Abb. 7.6: *Myo-Experiment 3: „Pattern Matching“ verschiedener Probanden vom jeweils ersten (links) zum letzten Versuch (rechts)*

7.2.1.2 Geomagic Touch X

Im Folgenden wird das haptische Eingabegerät Geomagic Touch X genauer vorgestellt (vgl. Kap. 4.2.4).

Das *Geomagic Touch X* besitzt eine Positionsgenauigkeit von $0,023$ mm (> 1100 dpi) über digitale Encoder in den ersten drei Gelenken. Die rotatorischen Komponenten – Roll, Pitch und Yaw – werden über lineare Potentiometer mit einem Fehler von ± 3 % gemessen. Der Nutzer erfährt bei der freien Bewegung des Stiftes eine Reibungskraft von $0,06$ Newton bei einem Gewicht von 45 Gramm am Stiftende.

Die *Vorwärtskinematik* beschreibt die Berechnung der Position im Zielkoordinatensystem aus den Gelenkwinkeln:

$$\boldsymbol{p} = f(\boldsymbol{q}) \tag{7.1}$$

mit dem Positionsvektor

$$\boldsymbol{p} = (x, y, z)^T \in \mathbb{R}^3$$

und den Gelenkwinkeln

$$\boldsymbol{q} = (\theta_1, \theta_2, \theta_3)^T \in \mathbb{R}^3 \ .$$

Die kinematische Kette des ähnlichen *PHANToM Omni Haptic Devices* mit den dazugehörigen Gelenkwinkeln und Abmessungen ist in Abb. 7.7 dargestellt. Der grundlegende kinematische Aufbau kann direkt auf das *Geomagic Touch X* übertragen werden, da sie

sich nicht unterscheiden. Damit ergibt sich für die Vorwärtskinematik:

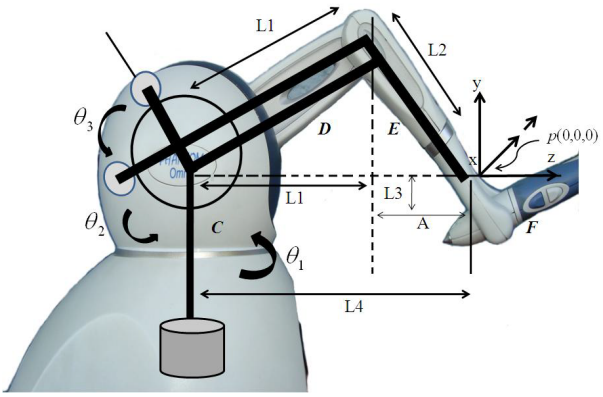


Abb. 7.7: *Kinematische Kette des PHANToM Omni Haptic Devices [246]*

$$\boldsymbol{p} = \begin{pmatrix} -s_1(L_1c_2 + L_2s_3) \\ L_3 - -L_2c_3 + L_1s_2 \\ -L4 + c_1(L_1c_2 + L_2s_3) \end{pmatrix} \tag{7.2}$$

mit $c_i = cos(\theta_i)$ und $s_i = cos(\theta_i)$. In dem Gleichungssystem und der Zeichnung beschreiben $L_1 = L_2 = 133,35$ mm die Längen der beiden Arme. $L_3 = 133,35$ mm bzw. $L_4 = 33,35$ mm sind Größen, um den Gerätenullpunkt zu definieren. Um diese Längen zu erhalten, wurden die Gelenkwinkel und Gerätepositionen, die vom Gerätetreiber geliefert werden, bei einer willkürlichen Bewegung aufgenommen und in die hier beschriebene Vorwärtskinematik eingesetzt. Durch den Vergleich der Gleichungen für die jeweiligen Raumrichtungen in mehreren Punkten konnte auf die einzelnen Längen geschlossen werden. Zur Validierung der Ergebnisse wurde die Vorwärtskinematik im späteren Verlauf der Arbeit in Form eines DTs des PHANToM Devices (basierend auf Tab. 7.3) in VEROSIM integriert (vgl. Abb. 7.8).

Bei der *Rückwärtskinematik* werden die Gelenkwinkel $\boldsymbol{q}$ aus der Geräteposition $\boldsymbol{p}$ berechnet:

$$\boldsymbol{q} = f^{-1}(\boldsymbol{p}) \tag{7.3}$$

Aus der Betrachtung der x- und z-Komponente der Vorwärtskinematik folgt direkt:

$$\theta_1 = - - atan2(x, z + L_4) \tag{7.4}$$

Für die Berechnung von θ_2 und θ_3 wird Abb. 7.9 verwendet mit folgenden Parametern:

$$\begin{aligned} R &= \sqrt{x^2 + (z + L_4)^2} \\ r &= \sqrt{x^2 + (z + L_4)^2 + (y - -L_3)^2} \\ \beta &= atan2(y - L_3, R) \end{aligned} \tag{7.5}$$



Abb. 7.8: *Realer und kinematischer Digitaler Zwilling des Sensable Geomagic Touch X*

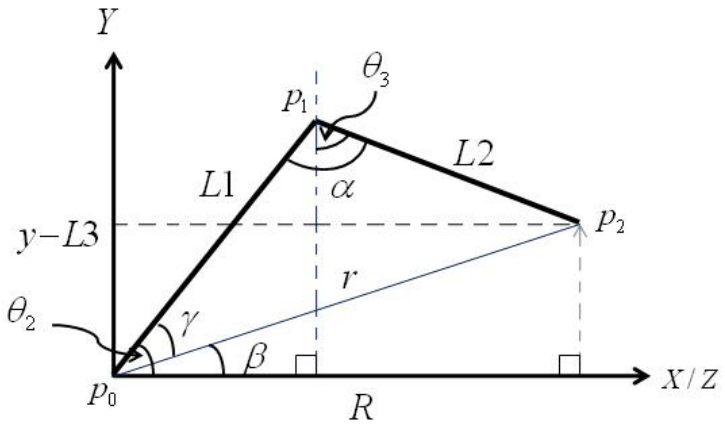


Abb. 7.9: *Inverse Kinematik des Geomagic Touch X [246]*

Durch Anwendung des Kosinussatzes für das Dreieck $\Delta P_0P_1P_2$ folgt:

$$\begin{aligned} {L_2}^2 &= {L_1}^2 + r^2 + 2L_1 r cos(\gamma) \\ \gamma &= cos^{-1}(\tfrac{{L_1}^2+r^2--{L_2}^2}{2L_1 r})\theta_2 = \gamma + \beta \end{aligned} \tag{7.6}$$

Zur Berechnung von θ_3 wird der Kosinussatz für den Winkel α angewendet:

$$\begin{aligned} r^2 &= {L_1}^2 + {L_2}^2 + 2L_1 L_2 cos(\alpha) \\ \alpha &= cos^{-1}(\tfrac{{L_1}^2+{L_2}^2--r^2}{2L_1 L_2}) \end{aligned} \tag{7.7}$$

Der Winkel α ist positiv für den physikalischen Arbeitsraum des Eingabegeräte, woraus folgt:

$$\theta_3 = \theta_2 + \alpha - -\frac{\pi}{2} \tag{7.8}$$

Zur Bestimmung der *Achsengeschwindigkeiten* aus vorgegebenen kartesischen Geschwin-

digkeiten kann die Jakobi-Matrix $\boldsymbol{J}$ benutzt werden:

$$\dot{\boldsymbol{p}} = \boldsymbol{J} \cdot \dot{\boldsymbol{q}} \tag{7.9}$$

Im Falle der Invertierbarkeit der Jakobi-Matrix gilt:

$$\dot{\boldsymbol{q}} = \boldsymbol{J}^{-1} \cdot \dot{\boldsymbol{p}} \tag{7.10}$$

Für die hier betrachtete Kinematik ist die Matrix definiert als:

$$\boldsymbol{J} = \begin{pmatrix} -c_1(c_2L_1 + s_3L_2) & s_1s_2L_1 & -s_1c_3L_2 \\ 0 & c_2L_1 & s_3L_2 \\ -s_1(c_2L_1 + s_3L_2) & -c_1s_2L_1 & c_1c_3L_2 \end{pmatrix} \tag{7.11}$$

Tab. 7.3: *Datenblatt: Sensable Geomagic Touch X [dd]*

Name	Specs
Freiheitsgrade	6
Positionserkennung in	X, Y, Z (digitaler Encoder) R, P, Y (Potentiometer)
Auflösung	0,023 mm (> 1100 dpi)
Kraftrückkopplung	X, Y, Z
maximale Kraft	7,9 N
kontinuierliche Kraft	1,75 N
minimal erreichbare Position:	
- physikalisch	-210 mm (X), -100 mm (Y), -115 mm (Z)
- nutzbar für Force Feedback	-80 mm (X), -50 mm (Y), -50 mm (Z)
maximal erreichbare Position:	
- physikalisch	210 mm (X), 165 mm (Y), 130 mm (Z)
- nutzbar für Force Feedback	80 mm (X), 70 mm (Y), 70 mm (Z)
physikalischer Arbeitsraum	420 mm (X) × 265 mm (Y) × 245 mm (Z)
Arbeitsraum für Force Feedback	160 mm (X) × 120 mm (Y) × 120 mm (Z)
Backdrive Friction	< 0,06 N
Masse am TCP	45 g
Gewicht des Eingabegerätes	~2,9 kg
Schnittstellen	Parallel Port, FireWire
unterstützte Plattformen	Intel, AMD

7.2.1.3 Exoskelett: UDP-Datenstrukturen

Im Folgenden sind Ergänzungen zum Exoskelett-UDP-Interface aus Kap. 4.2.4.3 zu finden.

Tab. 7.4: *Exoskelett Device Command Struct (EDCS)*

#	name	symbol	byte	unit
1	end-effector force	F_{ee}^{out}	3	N
2	joint torques	τ_i^{out}	4	Nm
3	end-effector position	p_{ee}^{out}	3	m
4	end-effector impedance		3	
	stiffness	k_{ee}		N/m
	viscosity	b_{ee}		Ns/m
	inertia	j_{ee}		kg
5	end-effector speed	$\dot{p}_{ee}^{out}$	1	m/s
6	joint-reference positions	q_{ref}	4	rad
7	joint impedance		12	
	stiffness	$k_{i,1..4}$		Nm/rad
	viscosity	$b_{i,1..4}$		Nm s/rad
	inertia	$j_{i,1..4}$		kg m2
8	joint reference speed	q_{ref}	4	rad/s
9	time	t_{exo}^{out}	1	s

Tab. 7.5: *Exoskelett Device Data Struct (EDDS)*

#	name	symbol	byte	unit
1	config ID		1	
2	motor position	p_{motor}	4	rad
3	joint position	q_i	6	rad
4	joint speed	$\dot{q}_i$	6	rad/s
5	joint position for control	q_i^{ctr}	6	rad
6	joint speed for control	$\dot{q}_i^{ctr}$	6	rad/s
7	joint torque	τ_i^{in}	4	Nm
8	motor torque	τ_{motor}^{in}	4	Nm
9	end-effector position	p_{ee}^{in}	3	m
10	end-effector speed	$\dot{p}_{ee}^{in}$	3	m/s
11	end-effector force	F_{ee}^{in}	3	N
...				
17	received packets		1	Hz
18	time	t_{exo}^{in}	1	s

7.2.2 Ausgabegeräte

Im Folgenden werden einige aktuelle HMDs vorgestellt, welchen momentan im Bereich VR/AR/MR besondere Beachtung geschenkt wird (vgl. auch Kap. 2.3.5). Die Reihenfolge der Vorstellung erfolgt *von opak zu transparent* und somit von VR über AR zu MR (analog zu Abb. 2.18).

7.2.2.1 VR-Brillen

Die unterschiedlichen zurzeit verfügbaren stereoskopischen Headsets oder Brillen werden im Folgenden (VR, vgl. Abb. 7.10 und Tab. 7.6, Abb. 7.11 und Tab. 7.7; AR, vgl. Abb. 7.12 und Tab. 7.8; MR, vgl. Abb. 7.14 und Tab. 7.10) vorgestellt.



Abb. 7.10: *Virtual Reality (VR): Oculus Rift[de]*

Name	Specs
FoV [deg]	110
Res [pxls]	2160 x 1200
M [g]	470
CPU	–
RAM	–
Tracking [m]	5 x 11
Price [€]	450

Tab. 7.6: *Rift-Datenblatt*

Die schon sehr fortschrittlichen und permanenten VR-Brillen haben nahezu identische Spezifikationen. Herauszustellen ist hier das sogenannte „Roomscale VR“, das mit den „LightHouses“ von HTC direkt und sehr gut ein stehendes VR-Erlebnis möglich macht und bei Oculus durch mehrere Tracker auch realisiert werden kann, wobei der Fokus hier jedoch auf einem sitzenden Nutzer liegt. Gerade im Bereich der Forschung ist deswegen die HTC Vive öfter vertreten. Des Weiteren gibt es im Bereich der Software mit *OpenVR*[ca] ein Software Framework, welches von beiden (und auch weiteren) VR-Brillen genutzt werden kann.



Abb. 7.11: *VR: HTC Vive[df]*

Name	Specs
FoV [deg]	110
Res [pxls]	2160 x 1200
M [g]	468
CPU	–
RAM	–
Tracking [m]	1,2 x 2,0 (min) 5,0 x 5,0 (max)
Price [€]	700

Tab. 7.7: *Vive-Datenblatt*

7.2.2.2 AR-Brillen

AR-Brillen wie die *Google Glass*, *Epson Moverio* oder die *Vuzix M300* bieten Möglichkeiten, die Realität mit virtuellen Daten zu überlagern. Hierbei gibt es rein hardwaretechnisch zwei Ansätze. Im Falle der Google Glass oder Vuzix M300 wird mit dem „Bild im Bild“-Modus ein Bildschirm am Rande des Sichtfeldes des Nutzers platziert, welcher alle Informationen anzeigen kann. Manchmal wird diese Form der AR auch als Assisted Reality (AsR) bezeichnet. Im Gegensatz dazu bietet die Epson Moverio ein ganzheitlich transparentes Sichtfeld, das als richtiges AR angesehen wird.



Abb. 7.12: *Assisted Reality (AsR): Vuzix M300[dg]*

Name	Specs
FoV [deg]	20
Res [pxls]	640 x 360
M [g]	110
CPU	Intel Atom
RAM [GB]	2
Camera	720p Vid, 13 Mpx Cam
Price [€]	1700

Tab. 7.8: *M300-Datenblatt*



Abb. 7.13: *AR: Epson Moverio BT 300[dh]*

Name	Specs
FoV [deg]	23
Res [pxls]	1280 x 720
M [g]	69
CPU	Intel Atom X5
RAM [GB]	2
Camera	720p Vid, 5 Mpx Cam
Price [€]	850

Tab. 7.9: *Moverio-Datenblatt*

7.2.2.3 MR-Brillen

Mixed-Reality-Brillen beschränken sich momentan noch primär auf die von Microsoft entwickelte *HoloLens* (vgl. Abb. 7.14).

Die HoloLens von Microsoft zeichnet sich auf der einen Seite durch das integrierte Tracking (in Form von zwei integrierten Kinect) sowie den PC-unabhängigen Betrieb aus. Auf der anderen Seite weist die HoloLens ein hohes Gewicht, den dadurch reduzierten Tragekomfort, einen hohen Preis und den momentan noch reduzierten Software-Support auf.



Abb. 7.14: *MR: Microsoft HoloLens[di]*

Name	Specs
FoV [deg]	90-110
Res [pxls]	720p
M [g]	578
CPU	Intel Atom X5
RAM [GB]	2
Camera	4 RGBD Cams
Price [€]	3300

Tab. 7.10: *HoloLens-Datenblatt*

Schlussendlich soll mit Hilfe einer Vereinheitlichung des VR/AR/MR-Standard im *OpenXR*-Konsortium die Möglichkeit (softwareseitig) geschaffen werden, Programme für alle Headsets einfach herstellen zu können. Bisher sind nur VR-Brillen durch den OpenVR-Standard vereinheitlicht.

Literaturverzeichnis

[1] A. Atanasyan, D. Kobelt, M. Goppold et al.: *The FeDiNAR Project: Using Augmented Reality to Turn Mistakes into Learning Opportunities*. In: *Augmented Reality in Education: Theory, Technology, Practice. Springer Nature*. (Artikel in Vorbereitung). 2020.

[2] M. Goppold, S. Tackenberg, A. Atanasyan et al.: *Systemkonzept und Modellierung beruflicher Handlungen im FeDiNAR-AR-Lernsystem*. In: *Gemeinschaften in Neuen Medien (GeNeMe)*. 2019.

[3] T. Klamt, D. Rodriguez, L. Baccelliere et al.: *Flexible Disaster Response of Tomorrow–Final Presentation and Evaluation of the CENTAURO System*. In: *IEEE Robotics and Automation Magazine (RAM), Special Issue on Humanoid Robot Applications in Real World Scenarios* (2019).

[4] T. Klamt, M. Schwarz, C. Lenz et al.: *Remote mobile manipulation with the centauro robot: Full-body telepresence and autonomous operator assistance*. In: *Journal of Field Robotics* (2019).

[5] T. Cichon und J. Rossmann: *Digital Twins: Assist and Support Cooperation in Human-Robot Teams*. In: *International Conference on Control, Automation, Robotics and Vision (ICARCV)*. 2018.

[6] T. Cichon und J. Rossmann: *Robotic Teleoperation: Mediated and Supported by Virtual Testbeds*. In: *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. 2017.

[7] T. Cichon und J. Rossmann: *Simulation-based User Interfaces for Digital Twins: Pre-, In-, or Post-Operational Analysis and Exploration of Virtual Testbeds*. In: *Proceedings of the 31st European Simulation and Modelling Conference (ESM), October 25–27, IST, Lisbon, Portugal*. IEEE. 2017.

[8] T. Cichon, C. Schlette und J. Rossmann: *Towards a 3D simulation-based operator interface for teleoperated robots in disaster scenarios*. In: *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. 2016, S. 264–269.

[9] T. Cichon, M. Priggemeyer und J. Rossmann: *Simulation-based Control and Simulation-based Support in eRobotics Applications*. In: *Applied Mechanics & Materials* 840 (2016).

[10] T. Cichon, C. Loconsole, D. Buongiorno et al.: *Combining an exoskeleton with 3D simulation in-the-loop*. In: *9th International Workshop on Human Friendly Robotics*. 2016, S. 31–34.

[11] G. Grinshpun, T. Cichon, D. Dipika et al.: *From Virtual Testbeds to Real Lightweight Robots: Development and deployment of control algorithms for soft robots, with particular reference to*. In: *Proceedings of the 47st International Symposium on Robotics (ISR)*. 2016, S. 1–7.

[12] L. Atorf, T. Cichon und J. Rossmann: *Flexible Data Logging, Management, and Analysis of Simulation Results of Complex Systems for eRobotics Applications*. In: *29th European Simulation and Modelling (ESM) Conference Proceedings*. 2015.

[13] J. Rossmann, M. Schluse, M. Rast et al.: *Simulation Technology for Soft Robotics Applications*. In: *Soft Robotics*. Springer, 2015, S. 100–119.

[14] T. Cichon, M. Rast und J. Rossmann: *Simulation tools for robotics: survey and comparison*. In: *28th European Simulation and Modelling (ESM) Conference Proceedings*. 2014.

[15] M. Schluse, M. Priggemeyer, L. Atorf et al.: *Experimentable Digital Twins—Streamlining Simulation-Based Systems Engineering for Industry 4.0*. In: *IEEE Transactions on Industrial Informatics* 14.4 (2018), S. 1722–1731.

[16] H. Kagermann, W. Wahlster und J. Helbig: *Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0 – Abschlussbericht des Arbeitskreises Industrie 4.0*. In: (2012).

[17] H. Kagermann, W. Wahlster und J. Helbig: *Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0 – Abschlussbericht des Arbeitskreises Industrie 4.0 – Vorabversion*. In: (2012).

[18] J. Sullivan, J. W. Sullivan, S. W. Tyler et al.: *Intelligent user interfaces*. 1994.

[19] B. Laurel und S. J. Mountford: *The art of human-computer interface design*. Addison-Wesley Longman Publishing Co., Inc., 1990.

[20] L. G. Terveen: *Overview of human-computer collaboration*. In: *Knowledge-Based Systems* 8.2–3 (1995), S. 67–81.

[21] H. Ishii und B. Ullmer: *Tangible bits: towards seamless interfaces between people, bits and atoms*. In: *Proceedings of the ACM SIGCHI Conference on Human factors in computing systems*. ACM. 1997, S. 234–241.

[22] J. H. Israel: *Tangible Interfaces in Virtual Environments*. In: *GI Jahrestagung (2)*. 2006, S. 392–395.

[23] R. Weidner, T. Redlich und J. P. Wulfsberg: *Technische Unterstützungssysteme*. Springer-Verlag, 2015.

[24] S. K. Card: *The psychology of human-computer interaction*. CRC Press, 2017.

[25] R. Weidner und F. Steinicke: *Human Enhancement Technology*. In: *Technische Unterstützungssysteme, die die Menschen wirklich wollen* (2016), S. 365.

[26] M. P. de Looze, T. Bosch, F. Krause et al.: *Exoskeletons for industrial application and their potential effects on physical work load*. In: *Ergonomics* 59.5 (2016), S. 671–681.

[27] H. Lee, W. Kim, J. Han et al.: *The technical trend of the exoskeleton robot system for human power assistance*. In: *International Journal of Precision Engineering and Manufacturing* 13.8 (2012), S. 1491–1497.

[28] H. Kazerooni: *Exoskeletons for human power augmentation*. In: *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. IEEE. 2005, S. 3459–3464.

[29] K. Kiguchi, M. H. Rahman, M. Sasaki et al.: *Development of a 3DOF mobile exoskeleton robot for human upper-limb motion assist*. In: *Robotics and Autonomous systems* 56.8 (2008), S. 678–691.

[30] R. Weidner, N. Kong und J. P. Wulfsberg: *Human Hybrid Robot: a new concept for supporting manual assembly tasks*. In: *Production Engineering* 7.6 (2013), S. 675–684.

[31] D. Stokes und S. Biggs: *The dominance of the visual*. In: *Perception and its Modalities* (2014).

[32] M. Schwarz, T. Rodehutskors, D. Droeschel et al.: *NimbRo Rescue: Solving Disaster-response Tasks with the Mobile Manipulation Robot Momaro*. In: *Journal of Field Robotics* 34.2 (2017), S. 400–425.

[33] M. Schwarz, J. Stückler und S. Behnke: *Mobile teleoperation interfaces with adjustable autonomy for personal service robots*. In: *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction*. ACM. 2014, S. 288–289.

[34] H. Winner, S. Hakuli, F. Lotz et al.: *Handbuch Fahrerassistenzsysteme: Grundlagen, Komponenten und Systeme für aktive Sicherheit und Komfort (3., überarbeitete und ergänzte Auflage Aufl.)* 2015.

[35] Deutsches Institut für Normung (DIN): *DIN EN ISO 9241-110: Übersicht über die Grundsätze der Dialoggestaltung*. Norm. 2008.

[36] S. Fürst, J. Mössinger, S. Bunzel et al.: *AUTOSAR–A Worldwide Standard is on the Road*. In: *14th International VDI Congress Electronic Systems for Vehicles, Baden-Baden*. Bd. 62. 2009, S. 5.

[37] Deutsches Institut für Normung (DIN): *DIN EN ISO 9241-10: Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten*. Norm. 1996.

[38] B. Shneiderman: *User Interface Design – Effektive Interaktion zwischen Mensch und Maschine [Designing the User Interface-Strategies for Effective Human-Computer Interaction, 1987]*. 2001.

[39] V. Hedicke: *Multimodalität in Mensch-Maschine-Schnittstellen*. In: *Mensch-Maschine-Systemtechnik* 2 (2000), S. 203–232.

[40] A. Stephan und S. Walter: *Handbuch Kognitionswissenschaft*. Springer-Verlag, 2013.

[41] S. Few: *Information dashboard design*. 2006.

[42] S. Rendgen, J. Wiedemann, P. Ciuccarelli et al.: *Information graphics*. Taschen Cologne, Germany, 2012.

[43] S. Oualline: *Vi iMproved*. New Riders Publishing, 2001.

[44] C. Mohs, J. Hurtienne, M. C. Kindsmüller et al.: *IUUI–Intuitive Use of User Interfaces: Auf dem Weg zu einer wissenschaftlichen Basis für das Schlagwort „Intuitivität"*. In: *MMI-Interaktiv* 11 (2006), S. 75–84.

[45] A. Naumann, J. Hurtienne, J. H. Israel et al.: *Intuitive use of user interfaces: defining a vague concept*. In: *International Conference on Engineering Psychology and Cognitive Ergonomics*. Springer. 2007, S. 128–136.

[46] J. Hurtienne und A. Naumann: *QUESI—A questionnaire for measuring the subjective consequences of intuitive use*. In: *Interdisciplinary College* 536 (2010).

[47] J. Hurtienne: *Image schemas and design for intuitive use*. Diss. 2011.

[48] Deutsches Institut für Normung (DIN): *DIN EN ISO 13849-1: Sicherheit von Maschinen – Sicherheitsbezogene Teile von Steuerungen*. In: DIN EN ISO 13849-1 (2016).

[49] Deutsches Institut für Normung (DIN): *DIN EN ISO 14121-1: Sicherheit von Maschinen – Risikobeurteilung – Teil 1: Leitsätze*. In: DIN EN ISO 14121-1 (2005).

[50] Deutsches Institut für Normung (DIN): *DIN EN ISO 10218-1: Industrieroboter – Sicherheitsanforderungen – Teil 1: Roboter*. In: DIN EN ISO 10218-1 (2012).

[51] D. G. Unfallversicherung: *BG/BGIA-Empfehlungen für die Gefährdungsbeurteilung nachMaschinenrichtlinie – Gestaltung von Arbeitsplätzen mit kollaborierenden Robotern*. In: BG/BGIA (2009).

[52] Deutsches Institut für Normung (DIN): *DIN EN ISO 15066: Roboter und Robotikgeräte – Kollaborierende Roboter*. In: DIN EN ISO 15066 (2017).

[53] T. Bedenk: *Virtual Reality*. In: *c't Wissen* 17 (2010), S. 82–88.

[54] D. Norman: *The design of everyday things: Revised and expanded edition*. Basic Books (AZ), 2013.

[55] J. C. Lee: *Hacking the nintendo wii remote*. In: *IEEE pervasive computing* 7.3 (2008).

[56] M. Ye, X. Wang, R. Yang et al.: *Accurate 3d pose estimation from a single depth image*. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE. 2011, S. 731–738.

[57] H. Sarbolandi, D. Lefloch und A. Kolb: *Kinect range sensing: Structured-light versus time-of-flight kinect*. In: *Computer Vision and Image Understanding* 139 (2015), S. 1–20.

[58] K. Khoshelham und S. O. Elberink: *Accuracy and resolution of kinect depth data for indoor mapping applications*. In: *Sensors* 12.2 (2012), S. 1437–1454.

[59] J. Han, L. Shao, D. Xu et al.: *Enhanced computer vision with microsoft kinect sensor: A review*. In: *IEEE transactions on cybernetics* 43.5 (2013), S. 1318–1334.

[60] A. Oliver, S. Kang, B. C. Wünsche et al.: *Using the Kinect as a navigation sensor for mobile robotics*. In: *Proceedings of the 27th conference on image and vision computing New Zealand*. ACM. 2012, S. 509–514.

[61] M. R. Andersen, T. Jensen, P. Lisouski et al.: *Kinect depth sensor evaluation for computer vision applications*. In: *Electrical and Computer Engineering Technical Report ECE-TR-6* (2012).

[62] Z. Ren, J. Meng, J. Yuan et al.: *Robust hand gesture recognition with kinect sensor*. In: *Proceedings of the 19th ACM international conference on Multimedia*. ACM. 2011, S. 759–760.

[63] Y.-J. Chang, S.-F. Chen und J.-D. Huang: *A Kinect-based system for physical rehabilitation: A pilot study for young adults with motor disabilities*. In: *Research in developmental disabilities* 32.6 (2011), S. 2566–2570.

[64] T. H. Massie und J. K. Salisbury: *The PHANToM haptic interface: A device for probing virtual objects*. In: *Proceedings of the ASME Dynamic Systems and Control Division*. 1994, S. 295–301.

[65] G. C. Burdea: *Force and touch feedback for virtual reality*. In: (1996).

[66] G. Burdea: *Haptic feedback for virtual reality, keynote address of Proceedings of International Workshop on Virtual prototyping*. In: *Laval: France* (1999).

[67] D. C. Ruspini, K. Kolarov und O. Khatib: *The Haptic Display of Complex Graphical Environments*. In: *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, S. 345–352.

[68] C. B. Zilles und J. K. Salisbury: *A constraint-based god-object method for haptic display*. In: *Intelligent Robots and Systems 95. 'Human Robot Interaction and Cooperative Robots', Proceedings. 1995 IEEE/RSJ International Conference on*. Bd. 3. 1995, 146–151 vol.3.

[69] Y. Wu und T. S. Huang: *Vision-based gesture recognition: A review*. In: *International Gesture Workshop*. Springer. 1999, S. 103–115.

[70] S. Mitra und T. Acharya: *Gesture recognition: A survey*. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37.3 (2007), S. 311–324.

[71] S. D. Gupta, S. Kundu, R. Pandey et al.: *Hand gesture recognition and classification by discriminant and principal component analysis using machine learning techniques*. In: *Hand* 1.9 (2012).

[72] H.-I. Suk, B.-K. Sin und S.-W. Lee: *Hand gesture recognition based on dynamic Bayesian network framework*. In: *Pattern recognition* 43.9 (2010), S. 3059–3072.

[73] A. D. Wilson und A. F. Bobick: *Parametric hidden markov models for gesture recognition*. In: *IEEE transactions on pattern analysis and machine intelligence* 21.9 (1999), S. 884–900.

[74] P. Li, T. Zhang und B. Ma: *Unscented Kalman filter for visual curve tracking.* In: *Image and vision computing* 22.2 (2004), S. 157–164.

[75] K. Murakami und H. Taguchi: *Gesture recognition using recurrent neural networks.* In: *Proceedings of the SIGCHI conference on Human factors in computing systems.* ACM. 1991, S. 237–242.

[76] R. Verma und A. Dev: *Vision based hand gesture recognition using finite state machines and fuzzy logic.* In: *Ultra Modern Telecommunications & Workshops, 2009. ICUMT'09. International Conference on.* IEEE. 2009, S. 1–6.

[77] X. Zhang, X. Chen, Y. Li et al.: *A framework for hand gesture recognition based on accelerometer and EMG sensors.* In: *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 41.6 (2011), S. 1064–1076.

[78] K. Kilteni, R. Groten und M. Slater: *The sense of embodiment in virtual reality.* In: *Presence: Teleoperators and Virtual Environments* 21.4 (2012), S. 373–387.

[79] M. Mori: *The uncanny valley.* In: *Energy* 7.4 (1970), S. 33–35.

[80] M. Mori, K. F. MacDorman und N. Kageki: *The uncanny valley [from the field].* In: *IEEE Robotics & Automation Magazine* 19.2 (2012), S. 98–100.

[81] S. Silbernagl: *Taschenatlas der Physiologie.* Georg Thieme Verlag, 2007.

[82] A. Stemmer, G. Schreiber, K. Arbter et al.: *Robust assembly of complex shaped planar parts using vision and force.* In: *Multisensor Fusion and Integration for Intelligent Systems, 2006 IEEE International Conference on.* IEEE. 2006, S. 493–500.

[83] S. R. Chhatpar und M. S. Branicky: *Search strategies for peg-in-hole assemblies with position uncertainty.* In: *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on.* Bd. 3. IEEE. 2001, S. 1465–1470.

[84] D. E. Whitney: *Quasi-static assembly of compliantly supported rigid parts.* In: *Journal of Dynamic Systems, Measurement, and Control* 104.1 (1982), S. 65–77.

[85] J. K. Salisbury: *Active stiffness control of a manipulator in Cartesian coordinates.* In: *Decision and Control including the Symposium on Adaptive Processes, 1980 19th IEEE Conference on.* Bd. 19. IEEE. 1980, S. 95–100.

[86] N. Hogan: *Impedance control: An approach to manipulation.* In: *American Control Conference, 1984.* IEEE. 1984, S. 304–313.

[87] J. Rossmann, E. G. Kaigom, L. Atorf et al.: *A Virtual Testbed for Human-Robot Interaction.* In: *Computer Modelling and Simulation (UKSim), 2013 UKSim 15th International Conference on.* IEEE. 2013, S. 277–282.

[88] E. G. Kaigom und J. Rossmann: *A new erobotics approach: Simulation of adaptable joint admittance control.* In: *Mechatronics and Automation (ICMA), 2013 IEEE International Conference on.* IEEE. 2013, S. 550–555.

[89] J. F. Broenink und M. L. Tiernego: *Peg-in-Hole assembly using Impedance Control with a 6 DOF Robot*. In: *Simulation in Industry, Proceedings of the 8 th European Simulation Symposium, Oct 24–26, 1996, Genoa, Italy*. 1996, S. 504–509.

[90] D. Gershon und V. Baruch: *Impedance control with autonomous mode switching*. In: *Advanced Robotics, 1991. 'Robots in Unstructured Environments', 91 ICAR., Fifth International Conference on*. IEEE. 1991, S. 482–486.

[91] A. Goldenberg: *Implementation of force and impedance control in robot manipulators*. In: *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*. IEEE. 1988, S. 1626–1632.

[92] P. Polygerinos, Z. Wang, K. C. Galloway et al.: *Soft robotic glove for combined assistance and at-home rehabilitation*. In: *Robotics and Autonomous Systems* 73 (2015), S. 135–143.

[93] P. Polygerinos, N. Correll, S. A. Morin et al.: *Soft robotics: Review of fluid-driven intrinsically soft devices; manufacturing, sensing, control, and applications in human-robot interaction*. In: *Advanced Engineering Materials* 19.12 (2017), S. 1700016.

[94] J. Rieffel, D. Knox, S. Smith et al.: *Growing and evolving soft robots*. In: *Artificial life* 20.1 (2014), S. 143–162.

[95] C. Majidi: *Soft robotics: a perspective—current trends and prospects for the future*. In: *Soft Robotics* 1.1 (2014), S. 5–11.

[96] D. Trivedi, C. D. Rahn, W. M. Kier et al.: *Soft robotics: Biological inspiration, state of the art, and future research*. In: *Applied Bionics and Biomechanics* 5.3 (2008), S. 99–117.

[97] R. Pfeifer, H. G. Marques und F. Iida: *Soft robotics: the next generation of intelligent machines*. In: *Int. Conf. on Artificial Intelligence*. AAAI Press, 2013, S. 5–11.

[98] D. Rus und M. T. Tolley: *Design, fabrication and control of soft robots*. In: *Nature* 521.7553 (2015), S. 467.

[99] A. D. Marchese, C. D. Onal und D. Rus: *Autonomous soft robotic fish capable of escape maneuvers using fluidic elastomer actuators*. In: *Soft Robotics* 1.1 (2014), S. 75–87.

[100] A. Crespi und A. J. Ijspeert: *AmphiBot II: An amphibious snake robot that crawls and swims using a central pattern generator*. In: *Proceedings of the 9th international conference on climbing and walking robots (CLAWAR 2006)*. BIOROB-CONF-2006-001. 2006, S. 19–27.

[101] M. Manti, T. Hassan, G. Passetti et al.: *A bioinspired soft robotic gripper for adaptable and effective grasping*. In: *Soft Robotics* 2.3 (2015), S. 107–116.

[102] L. Birglen, T. Laliberté und C. M. Gosselin: *Underactuated robotic hands*. Bd. 40. Springer, 2007.

[103] R. Deimel und O. Brock: *A novel type of compliant and underactuated robotic hand for dexterous grasping*. In: *The International Journal of Robotics Research* 35.1–3 (2016), S. 161–185.

[104] M. Schwarz, T. Rodehutskors, M. Schreiber et al.: *Hybrid driving-stepping locomotion with the wheeled-legged robot momaro*. In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, S. 5589–5595.

[105] B. Gromov, L. M. Gambardella und G. A. Di Caro: *Wearable multi-modal interface for human multi-robot interaction*. In: *Safety, Security, and Rescue Robotics (SSRR), 2016 IEEE International Symposium on*. IEEE. 2016, S. 240–245.

[106] P. F. Hokayem und M. W. Spong: *Bilateral teleoperation: An historical survey*. In: *Automatica* 42.12 (2006), S. 2035–2057.

[107] T. Fong und C. Thorpe: *Vehicle teleoperation interfaces*. In: *Autonomous robots* 11.1 (2001), S. 9–18.

[108] M. DeDonato, V. Dimitrov, R. Du et al.: *Human-in-the-loop Control of a Humanoid Robot for Disaster Response: A Report from the DARPA Robotics Challenge Trials*. In: *Journal of Field Robotics* 32.2 (2015), S. 275–292.

[109] M. Schwarz, T. Rodehutskors, M. Schreiber et al.: *Hybrid Driving-Stepping Locomotion with the Wheeled-legged Robot Momaro*. In: *IEEE Robotics and Automation Letters (RA-L)* 2.1 (2016), S. 49–55.

[110] J. Lim, I. Shim, O. Sim et al.: *Robotic software system for the disaster circumstances: System of team KAIST in the DARPA Robotics Challenge Finals*. In: *Humanoid Robots (Humanoids), 2015 IEEE-RAS 15th International Conference on*. IEEE. 2015, S. 1161–1166.

[111] P. Ögren, P. Svenmarck, P. Lif et al.: *Design and implementation of a new teleoperation control mode for differential drive UGVs*. In: *Autonomous Robots* 37.1 (2014), S. 71–79.

[112] F. Båberg, S. Caccamo, N. Smets et al.: *Free look UGV teleoperation control tested in game environment: Enhanced performance and reduced workload*. In: *Safety, Security, and Rescue Robotics (SSRR), 2016 IEEE International Symposium on*. IEEE. 2016, S. 312–319.

[113] E. F. Camacho und C. B. Alba: *Model predictive control*. Springer Science & Business Media, 2013.

[114] T. B. Sheridan: *Human–Robot Interaction Status and Challenges*. In: *Human Factors: The Journal of the Human Factors and Ergonomics Society* 58.4 (2016), S. 525–532.

[115] J. Rossmann, A. Kupetz und R. Wischnewski: *A Universal Approach for the Intuitive Control of Mobile Robots using an AR/VR-based Interface*. In: *Proc. World Acadamy of Science, Engineering and Technology (WASET)* 67 (2010).

[116] P. Mitra und G. Niemeyer: *Model-mediated telemanipulation*. In: *The International Journal of Robotics Research* 27.2 (2008), S. 253–262.

[117] P. Mitra und G. Niemeyer: *Dynamic proxy objects in haptic simulations*. In: *Robotics, Automation and Mechatronics, 2004 IEEE Conference on*. Bd. 2. IEEE. 2004, S. 1054–1059.

[118] C. Blum, A. F. Winfield und V. V. Hafner: *Simulation-based internal models for safer robots*. In: *Frontiers in Robotics and AI* 4 (2018), S. 74.

[119] A. F. Winfield: *Robots with internal models: a route to self-aware and hence safer robots*. In: *The computer after me: Awareness and self-awareness in autonomic systems*. World Scientific, 2015, S. 237–252.

[120] S.-H. Leitner und W. Mahnke: *OPC UA–service-oriented architecture for industrial applications*. In: *ABB Corporate Research Center* (2006).

[121] J. Pfrommer, A. Ebner, S. Ravikumar et al.: *Open source OPC UA PubSub over TSN for realtime industrial communication*. In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*. Bd. 1. IEEE. 2018, S. 1087–1090.

[122] U. Hunkeler, H. L. Truong und A. Stanford-Clark: *MQTT-S—A publish/subscribe protocol for Wireless Sensor Networks*. In: *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*. IEEE. 2008, S. 791–798.

[123] Y. Maruyama, S. Kato und T. Azumi: *Exploring the performance of ROS2*. In: *Proceedings of the 13th International Conference on Embedded Software*. ACM. 2016, S. 5.

[124] D. Thangavel, X. Ma, A. Valera et al.: *Performance evaluation of MQTT and CoAP via a common middleware*. In: *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*. IEEE. 2014, S. 1–6.

[125] S. Bendel, T. Springer, D. Schuster et al.: *A service infrastructure for the Internet of Things based on XMPP*. In: *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on*. IEEE. 2013, S. 385–388.

[126] J. E. Luzuriaga, M. Perez, P. Boronat et al.: *A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks*. In: *Consumer Communications and Networking Conference (CCNC), 2015 12th Annual IEEE*. IEEE. 2015, S. 931–936.

[127] S. Profanter, A. Tekat, K. Dorofeev et al.: *OPC UA versus ROS, DDS, and MQTT: Performance Evaluation of Industry 4.0 Protocols*. In: *Proceedings of the IEEE International Conference on Industrial Technology (ICIT)*. 2019.

[128] S. Cousins, B. Gerkey, K. Conley et al.: *Sharing software with ros [ros topics]*. In: *Robotics & Automation Magazine, IEEE* 17.2 (2010), S. 12–14.

[129] E. Fernandez, L. S. Crespo, A. Mahtani et al.: *Learning ROS for Robotics Programming*. Packt Publishing Ltd, 2015.

[130] M. Quigley, K. Conley, B. Gerkey et al.: *ROS: an open-source Robot Operating System*. In: *ICRA workshop on open source software*. Bd. 3. 3.2. Kobe, Japan. 2009, S. 5.

[131] B. P. Zeigler, T. G. Kim und H. Praehofer: *Theory of modeling and simulation*. Academic press, 2000.

[132] *VDI 3633: Simulationen von Logistik-, Materialfluss- und Produktionssystemen*. Norm. 2014.

[133] H. Ulrich und G. J. Probst: *Anleitung zum ganzheitlichen Denken und Handeln: ein Brevier für Führungskräfte*. Haupt, 1995.

[134] P. K. der Forschungsunion Wirtschaft–Wissenschaft: *Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0*. In: *Abschlussbericht des Arbeitskreises Industrie* 4 (2013), S. 24–29.

[135] T. Sauerbier: *Theorie und Praxis von Simulationssystemen*. In: *Vieweg & Sohn Verlaggesellschaft, Braunschweig/Wiesbaden* (1999), S. 5.

[136] M. A. A. Rahman, K. Mayama, T. Takasu et al.: *Model-driven development of intelligent mobile robot using systems modeling language (SysML)*. In: *Mobile Robots-Control Architectures, Bio-Interfacing, Navigation, Multi Robot Motion Planning and Operator Training*. InTech, 2011.

[137] International Organization for Standardization: *ISO/IEC 19514: Information technology – Object management group systems modeling language (OMG SysML)*. In: ISO/IEC 19514 (2017).

[138] S. C. Spangelo, J. Cutler, L. Anderson et al.: *Model based systems engineering (MBSE) applied to Radio Aurora Explorer (RAX) CubeSat mission operational scenarios*. In: *Aerospace Conference, 2013 IEEE*. IEEE. 2013, S. 1–18.

[139] N. Hempe: *Bridging the Gap Between Rendering and Simulation Frameworks: Concepts, Approaches and Applications for Modern Multi-Domain VR Simulation Systems*. Diss. Springer: RWTH Aachen University, 2016.

[140] T. J. Jung: *Methoden der Mehrkörperdynamiksimulation als Grundlage relitätsnaher Virtueller Welten*. Diss. Dr.Hut: RWTH Aachen University, 2011.

[141] Y. Mizuchi und T. Inamura: *Cloud-based Multimodal Human-Robot Interaktion Simulator Utilizing ROS and Unity Frameworks*. In: IEEE. 2017.

[142] M. Rast: *Domänenübergreifende Modellierung und Simulation als Grundlage für virtuelle Testbeds*. Diss. Apprimus: RWTH Aachen University, 2014.

[143] J. Rossmann, R. Wischnewski und O. Stern: *A Comprehensive 3-D Simulation System for the Virtual Production*. In: *Proceedings of the 8th International Industrial Simulation Conference (ISC)*. ISBN 978-90-77381-5-57. 2010, S. 109–116.

[144] E. Roos, A. Behrens und S. Anton: *RDS-Realistic Dynamic Simulation of Robots*. In: *Int. Symp. on Industrial Robots*. Bd. 28. Int. Fed. of Robotics & Robotic Industries. 1997, S. 17–27.

[145] N. Koenig und A. Howard: *Design and use paradigms for gazebo, an open-source multi-robot simulator*. In: *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*. Bd. 3. IEEE. 2004, S. 2149–2154.

[146] P. Fisette und J. Samin: *ROBOTRAN: Symbolic Generation of Multi-Body System Dynamic Equations*. In: *Advanced Multibody System Dynamics*. Springer, 1993, S. 373–378.

[147] J.-U. Dolinsky: *The development of a genetic programming method for kinematic robot calibration*. Diss. Liverpool John Moores University, 2001.

[148] J. Rossmann, M. Schluse, C. Schlette et al.: *A New Approach to 3D Simulation Technology as Enabling Technology for eROBOTICS*. In: *Int. Simulation Tools Conf. & EXPO (SIMEX)*. 2013, S. 39–46.

[149] C. Connolly: *Technology and applications of ABB RobotStudio*. In: *Industrial Robot: An International Journal* 36.6 (2009), S. 540–545.

[150] K. Vollmann: *A new approach to robot simulation tools with parametric components*. In: *Industrial Technology, 2002. IEEE ICIT'02. 2002 IEEE International Conference on*. Bd. 2. IEEE. 2002, S. 881–885.

[151] J. Rossmann, M. Schluse, C. Schlette et al.: *A New Approach to 3D Simulation Technology as Enabling Technology for eRobotics*. In: *International Simulation Tools Conference & EXPO 2013, SIMEX'2013*. 1st. Brussels, 2013, S. 39–46.

[152] J. Rossmann, M. Schluse, C. Schlette et al.: *A modular system architecture for the distributed simulation and control of assembly lines based on active databases*. In: *International Conference on System Science and Simulation in Engineering (ICOSSSE), Iwate (Japan)*. Bd. 4. 6. 2010.

[153] W. Demtröder: *Experimentalphysik 1: Mechanik und Wärme*. In: 7 (2006).

[154] P. Corke: *Robotics, vision and control: fundamental algorithms in MATLAB*. Bd. 73. Springer, 2011.

[155] R. S. Hartenberg und J. Denavit: *Kinematic synthesis of linkages*. McGraw-Hill, 1964.

[156] M. Teschner, B. Heidelberger, M. Müller et al.: *Optimized Spatial Hashing for Collision Detection of Deformable Objects*. In: *Vmv*. Bd. 3. 2003, S. 47–54.

[157] D. E. Stewart und J. C. Trinkle: *An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction*. In: *International Journal for Numerical Methods in Engineering* 39.15 (1996), S. 2673–2691.

[158] R. Bellman: *Dynamic programming and Lagrange multipliers*. In: *Proceedings of the National Academy of Sciences* 42.10 (1956), S. 767–769.

[159] D. Stewart und J. C. Trinkle: *An Implicit Time-Stepping Scheme for Rigid Body Dynamics with Coulomb Friction*. In: *IEEE International Conference on Robots and Automation*. 2000.

[160] J. Rossmann, N. Hempe, M. Emde et al.: *A real-time optical sensor simulation framework for development and testing of industrial and mobile robot applications*. In: *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*. VDE. 2012, S. 1–6.

[161] J. Rossmann, M. Emde, B. Sondermann et al.: *Advanced sensor simulation in virtual testbeds: A cost-efficient way to develop and verify space applications*. In: *AIAA SPACE 2011 Conference & Exposition*. 2011, S. 7153.

[162] A. Boeing: *Evaluation of real-time physics simulation systems*. In: *Graphite* (2007).

[163] J. Hummel, R. Wolff, T. Stein et al.: *An evaluation of open source physics engines for use in virtual reality assembly simulations*. In: *Advances in Visual Computing*. Springer, 2012, S. 346–357.

[164] M. Gonzalez, U. Lugris, R. Gutierrez et al.: *Benchmarking of MBS Simulaton Software*. In: *ASME* (2005).

[165] M. Adams: *MSC Adams/View Help*. MSC. 2013.

[166] P. Fritzson: *Principles of object-oriented modeling and simulation with Modelica 2.1*. John Wiley & Sons, 2010.

[167] H. Elmqvist, F. E. Cellier und M. Otter: *Object-Oriented Modeling Of Hybrid Systems*. In: 1993, S. 31–41.

[168] M. M. Tiller: *Introduction to Physical Modeling with Modelica*. Springer, 2001.

[169] A. Geiger, P. Lenz und R. Urtasun: *Are we ready for autonomous driving? the kitti vision benchmark suite*. In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, S. 3354–3361.

[170] A. Geiger, P. Lenz, C. Stiller et al.: *Vision meets robotics: The KITTI dataset*. In: *The International Journal of Robotics Research* 32.11 (2013), S. 1231–1237.

[171] S. Collins, A. Ruina, R. Tedrake et al.: *Efficient bipedal robots based on passive-dynamic walkers*. In: *Science* 307.5712 (2005), S. 1082–1085.

[172] J. Chestnutt, M. Lau, G. Cheung et al.: *Footstep planning for the honda asimo humanoid*. In: *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*. IEEE. 2005, S. 629–634.

[173] I.-W. Park, J.-Y. Kim, J. Lee et al.: *Mechanical design of humanoid robot platform KHR-3 (KAIST humanoid robot 3: HUBO)*. In: *Humanoid Robots, 2005 5th IEEE-RAS International Conference on*. IEEE. 2005, S. 321–326.

[174] R. O. Ambrose, H. Aldridge, R. S. Askew et al.: *Robonaut: NASA's space humanoid*. In: *IEEE Intelligent Systems and Their Applications* 15.4 (2000), S. 57–63.

[175] M. A. Diftler, C. Culbert, R. O. Ambrose et al.: *Evolution of the NASA/DARPA robonaut control system*. In: *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*. Bd. 2. IEEE. 2003, S. 2543–2548.

[176] C. Borst, T. Wimbock, F. Schmidt et al.: *Rollin'Justin-Mobile platform with variable base.* In: *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on.* IEEE. 2009, S. 1597–1598.

[177] M. Fuchs, C. Borst, P. R. Giordano et al.: *Rollin'Justin-Design considerations and realization of a mobile platform for a humanoid upper body.* In: *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on.* IEEE. 2009, S. 4131–4137.

[178] S. Cousins: *Ros on the pr2 [ros topics].* In: *IEEE Robotics & Automation Magazine* 17.3 (2010), S. 23–25.

[179] B. M. Yamauchi: *PackBot: A versatile platform for military robotics.* In: *Unmanned Ground Vehicle Technology VI.* Bd. 5422. International Society for Optics und Photonics. 2004, S. 228–238.

[180] S. Kuindersma, R. Deits, M. Fallon et al.: *Optimization-based locomotion planning, estimation, and control design for the atlas humanoid robot.* In: *Autonomous Robots* 40.3 (2016), S. 429–455.

[181] H. Kitano, S. Tadokoro, I. Noda et al.: *Robocup rescue: Search and rescue in large-scale disasters as a domain for autonomous agents research.* In: *Systems, Man, and Cybernetics, 1999. IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on.* Bd. 6. IEEE. 1999, S. 739–743.

[182] S. Tadokoro, H. Kitano, T. Takahashi et al.: *The robocup-rescue project: A robotic approach to the disaster mitigation problem.* In: *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on.* Bd. 4. IEEE. 2000, S. 4089–4094.

[183] H. Kitano und S. Tadokoro: *Robocup rescue: A grand challenge for multiagent and intelligent systems.* In: *AI magazine* 22.1 (2001), S. 39.

[184] C. Skinner und S. Ramchurn: *The robocup rescue simulation platform.* In: *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: Volume 1.* International Foundation for Autonomous Agents und Multiagent Systems. 2010, S. 1647–1648.

[185] E. Negri, L. Fumagalli und M. Macchi: *A Review of the Roles of Digital Twin in CPS-based Production Systems.* In: *Procedia Manufacturing* 11 (2017), S. 939–948.

[186] E. VDI: *Statusreport.* In: *Industrie 4.0 – Technical Assets – Grundlegende Begriffe, Konzepte, Lebenszyklen und Verwaltung* (2015).

[187] T. Bauernhansl: *WGP-Standpunkt Industrie 4.0.* WGP, Wissenschaftliche Gesellschaft für Produktionstechnik, 2016.

[188] L. Atorf, M. Schluse und J. Rossmann: *Simulation-based Optimization, Reasoning, and Control: The eRobotics Approach Towards Intelligent Robots.* In: *Int. Symp. on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS).* to be published. 2014.

[189] E. Freund, M. Müller und J. Rossmann: *Data storage and flow control in automation systems by means of an active database*. In: *Computational intelligence for modelling, control & automation*. Bd. 99. 1999.

[190] E. Freund und J. Rossmann: *Systems approach to robotics and automation*. In: *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*. Bd. 1. IEEE. 1995, S. 3–14.

[191] E. Freund, J. Rossmann und M. Schluse: *Projective Virtual Reality in space applications: A telerobotic ground station for a space mission*. In: *Intelligent Systems and Smart Manufacturing*. International Society for Optics und Photonics. 2000, S. 279–290.

[192] E. Freund, J. Rossmann und M. Schluse: *Virtual Reality for intelligent and interactive operating-, training- and visualization systems*. In: *Proc. SPIE*. Bd. 4196. 2000, S. 268.

[193] J. Rossmann, M. Schluse und C. Schlette: *The virtual forest: Robotics and simulation technology as the basis for new approaches to the biological and the technical production in the forest*. In: *Proceedings of the 13th World Multi-Conference Conference on Systems, Cybernetics and Informatics, WMSCI*. 2009.

[194] J. Rossmann, O. Stern und M. Rast: *Virtual Crater – Entwicklung einer virtuellen Simulations- und Demonstrationsumgebung zur planetarischen Exploration mit Fokus auf extraterrestrische Krater*. Technischer Abschlussbericht. RIF Institut für Forschung und Transfer e.V., 2013.

[195] C. Schlette, E. G. Kaigom, D. Losch et al.: *3D simulation-based user interfaces for a highly-reconfigurable industrial assembly cell*. In: *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*. IEEE. 2016, S. 1–6.

[196] J. Rossmann: *eRobotics Meets the Internet of Things: Modern Tools for Today's Challenges in Robotics and Automation*. In: *Developments of E-Systems Engineering (DeSE), 2015 International Conference on*. IEEE. 2015, S. 318–323.

[197] L. Atorf und J. Rossmann: *Interactive Analysis and Visualization of Digital Twins in High-Dimensional State Spaces*. In: *International Conference on Control, Automation, Robotics and Vision*. (not published yet). ICARCV. 2018.

[198] M. Garrels: *Introduction To Linux: A Beginner's Guide*. Fultus Corporation, 2004.

[199] R. B. Dieter Schramm Manfred Hiller: *Modellbildung und Simulation der Dynamik von Kraftfahrzeugen*. Bd. 2. Springer Vieweg, 2013.

[200] T. Fliessbach und H. Walliser: *Arbeitsbuch zur Theoretischen Physik: Repetitorium und Übungsbuch*. Springer-Verlag, 2011.

[201] J. Rossmann, M. Schluse und M. Rast: *Simulation-Driven Engineering für mobile Systeme. Virtuelle Welten als Entwicklungsplattform für mobile Robotik-Von der Planetenexploration bis in den Wald*. In: *at-Automatisierungstechnik Methoden und Anwendungen der Steuerungs-, Regelungs-und Informationstechnik* 61.4 (2013), S. 278–289.

[202] B. Sondermann, M. Emde, M. Rast et al.: *Simulation-based engineering with hybrid testbeds*. In: *Modelling Symposium (EMS), 2013 European*. Hrsg. von D. Al-Dabass, A. Orsoni und Z. Xie. IEEE. 2013, S. 471–476.

[203] B. Victor: *Seeing spaces*. In: *Talk at EG conference*. 2014.

[204] J. Rossmann, M. Dimartino, M. Priggemeyer et al.: *Practical applications of simulation-based control*. In: *Advanced Intelligent Mechatronics (AIM), 2016 IEEE International Conference on*. IEEE. 2016, S. 1376–1381.

[205] A. S. Glassner: *An introduction to ray tracing*. Elsevier, 1989.

[206] A. Mohammadi, M. Tavakoli und A. Jazayeri: *PHANSIM: a simulink toolkit for the sensable PHANToM haptic devices*. In: *Proceedings of the 23rd CANCAM, Canada* 11 (2011), S. 787–790.

[207] L. S.-H. Chan und K.-S. Choi: *Integrating PhysX and OpenHaptics: Efficient force feedback generation using physics engine and haptic devices*. In: *Pervasive Computing (JCPC), 2009 Joint Conferences on*. IEEE. 2009, S. 853–858.

[208] G. Sankaranarayanan und B. Hannaford: *Comparison of performance of virtual coupling schemes for haptic collaboration using real and emulated internet connections*. In: *Proceedings of the 1st international conference on Robot communication and coordination*. IEEE Press. 2007, S. 31.

[209] K. Salisbury, F. Conti und F. Barbagli: *Haptic rendering: introductory concepts*. In: *Computer Graphics and Applications, IEEE* 24.2 (2004), S. 24–32.

[210] Y. Jiang, C. Yang, X. Wang et al.: *Kinematics modeling of Geomagic Touch X haptic device based on adaptive parameter identification*. In: *Real-time Computing and Robotics (RCAR), IEEE International Conference on*. IEEE. 2016, S. 295–300.

[211] E. Ruffaldi, M. Barsotti, D. Leonardis et al.: *Evaluating Virtual Embodiment with the ALEx Exoskeleton*. In: *International Conference on Human Haptic Sensing and Touch Enabled Computer Applications*. Springer. 2014, S. 133–140.

[212] E. Pirondini, M. Coscia, S. Marcheschi et al.: *Evaluation of the effects of the Arm Light Exoskeleton on movement execution and muscle activities: a pilot study on healthy subjects*. In: *Journal of neuroengineering and rehabilitation* 13.1 (2016), S. 1.

[213] J. Rossmann, A. Kupetz und R. Wischnewski: *An AR/VR based approach towards the intuitive control of mobile rescue robots*. In: *International Journal of Computer, Electrical, Automation, Control and Information Engineering* 4.11 (2010), S. 411–418.

[214] K. Eilers und J. Rossmann: *Modeling an AGV based facility logistics system to measure and visualize performance availability in a VR environment*. In: *Proceedings of the 2014 Winter Simulation Conference*. IEEE Press. 2014, S. 367–375.

[215] D. Holz und S. Behnke: *Fast range image segmentation and smoothing using approximate surface reconstruction and region growing*. In: *Intelligent autonomous systems 12*. Springer, 2013, S. 61–73.

[216] M. J. Pratt: *A new ISO 10303 (STEP) resource for modeling parameterization and constraints*. In: *Journal of Computing and Information Science in Engineering* 4.4 (2004), S. 339–351.

[217] T. Blochwitz, M. Otter, M. Arnold et al.: *The functional mockup interface for tool independent exchange of simulation models*. In: *8th International Modelica Conference, Dresden*. 2011, S. 20–22.

[218] T. Blochwitz, M. Otter, J. Akesson et al.: *Functional mockup interface 2.0: The standard for tool independent exchange of simulation models*. In: *Proceedings of the 9th International MODELICA Conference; September 3–5; 2012; Munich; Germany*. 076. Linköping University Electronic Press. 2012, S. 173–184.

[219] M. Schwarz, A. Milan, A. S. Periyasamy et al.: *RGB-D object detection and semantic segmentation for autonomous manipulation in clutter*. In: *International Journal of Robotics Research* (2017).

[220] A. H. Jazwinski: *Stochastic processes and filtering theory*. Courier Corporation, 2007.

[221] J. D. G. Kooijman, J. P. Meijaard, J. M. Papadopoulos et al.: *A Bicycle Can Be Self-Stable Without Gyroscopic or Caster Effects*. In: *Science* 332.6027 (2011). Aufruf: 17.01.2015, S. 369.

[222] O. Boubaker: *The inverted pendulum: A fundamental benchmark in control theory and robotics*. In: *Education and e-Learning Innovations (ICEELI), 2012 international conference on*. IEEE. 2012, S. 1–6.

[223] S. Evangelou: *Control and stability analysis of two-wheeled road vehicles*. Diss. University of London London, 2004.

[224] W. Suhr und H.-J. Schlichting: *Gleichgewicht auf zwei Rädern. Physik des Fahrradfahrens*. In: *Physik in unserer Zeit* 38.5 (2007), S. 238–241.

[225] V. Cossalter: *Motorcycle dynamics*. Lulu.com, 2006.

[226] J. Thieling: *Modellierung, Simulation und Integration von Fahrzeugdynamiken in ein Starrkörpersimulationssystem*. Magisterarb. RWTH Aachen University, 2016.

[227] E. Bakker, L. Nyborg und H. B. Pacejka: *Tyre modelling for use in vehicle dynamics studies*. Techn. Ber. SAE Technical Paper, 1987.

[228] H. Lipson: *Challenges and Opportunities for Design, Simulation, and Fabrication of Soft Robots*. In: *Soft Robotics* 1.P (2013), S. 21–27.

[229] J. Hiller und H. Lipson: *Dynamic Simulation of Soft Multimaterial 3D-Printed Objects*. In: *Soft Robotics* 1.1 (2014), S. 88–101.

[230] G. Robinson und J. Davies: *Continuum Robots – A State of the Art*. In: *Int. Conf. on Robotics and Automation (ICRA)*. IEEE. 1999.

[231] J. M. Germann, A. Maesani, D. Floreano et al.: *Soft Cell Simulator: A tool to study Soft Multi-Cellular Robots*. In: *Proceedings of the IEEE Int. Conference on Robotics and Biomimetics*. EPFL-CONF-196284. IEEE. 2013, S. 1300–1305.

[232] A. Verl, A. Albu-Schäffer, O. Brock et al.: *Soft Robotics*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015.

[233] Keiser, Bruno Heidelberger Matthias Teschner Richard und M. M. M. Gross: *Consistent penetration depth estimation for deformable collision response*. In: *Vision, Modeling, and Visualization 2004: Proceedings, November 16-18, 2004, Standford, USA*. 2004, S. 339.

[234] S. Haddadin: *Towards safe robots: approaching Asimov's 1st law*. Diss. RWTH Aachen University, 2011.

[235] M. Quigley, K. Conley, B. Gerkey et al.: *ROS: an open-source Robot Operating System*. In: *ICRA workshop on open source software*. Bd. 3. 3.2. 2009, S. 5.

[236] E. Freeman, E. Robson, B. Bates et al.: *Head First Design Patterns: A Brain-Friendly Guide*. O'Reilly Media, Inc., 2004.

[237] D. Beazley: *Understanding the python gil*. In: *PyCON Python Conference. Atlanta, Georgia*. 2010.

[238] L. Joseph: *Mastering ROS for robotics programming*. Packt Publishing Ltd, 2015.

[239] Y. Maruyama, S. Kato und T. Azumi: *Exploring the Performance of ROS2*. In: *Proceedings of the 13th International Conference on Embedded Software*. EMSOFT '16. Pittsburgh, Pennsylvania: ACM, 2016, 5:1–5:10.

[240] T. Foote: *tf: The transform library*. In: *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*. Open-Source Software workshop. 2013, S. 1–6.

[241] L. Muratore, A. Laurenzi, E. M. Hoffman et al.: *XBotCore: A Real-Time Cross-Robot Software Platform*. In: *Robotic Computing (IRC), IEEE International Conference on*. IEEE. 2017, S. 77–80.

[242] L. Atorf, C. Schorn, J. Rossmann et al.: *A framework for simulation-based optimization demonstrated on reconfigurable robot workcells*. In: *Systems Engineering Symposium (ISSE), 2017 IEEE International*. IEEE. 2017, S. 1–6.

[243] M. Alfrink: *Erweiterte Realität mit Hilfe von simulationsbasierten digitalen Zwillingen zur verbesserten Interaktion mit Industrierobotern*. Magisterarb. RWTH Aachen University, 2018.

[244] R. Bischoff, J. Kurth, G. Schreiber et al.: *Aus der Forschung zum Industrieprodukt: Die Entwicklung des KUKA Leichtbauroboters*. In: *at-Automatisierungstechnik Methoden und Anwendungen der Steuerungs-, Regelungs-und Informationstechnik* 58.12 (2010), S. 670–680.

[245] H. Park, J.-H. Bae, J.-H. Park et al.: *Intuitive peg-in-hole assembly strategy with a compliant manipulator*. In: *Robotics (ISR), 2013 44th International Symposium on*. 2013, S. 1–5.

[246] A. J. Silva, O. A. D. Ramirez, V. P. Vega et al.: *PHANToM OMNI Haptic Device: Kinematic and Manipulability*. In: *Proceedings of the 2009 Electronics, Robotics and Automotive Mechanics Conference (Cerma 2009)*. CERMA '09. Washington, DC, USA: IEEE Computer Society, 2009, S. 193–198.

Eigene Veröffentlichungen

[1] A. Atanasyan, D. Kobelt, M. Goppold et al.: *The FeDiNAR Project: Using Augmented Reality to Turn Mistakes into Learning Opportunities*. In: *Augmented Reality in Education: Theory, Technology, Practice. Springer Nature*. (Artikel in Vorbereitung). 2020.

[2] M. Goppold, S. Tackenberg, A. Atanasyan et al.: *Systemkonzept und Modellierung beruflicher Handlungen im FeDiNAR-AR-Lernsystem*. In: *Gemeinschaften in Neuen Medien (GeNeMe)*. 2019.

[3] T. Klamt, D. Rodriguez, L. Baccelliere et al.: *Flexible Disaster Response of Tomorrow–Final Presentation and Evaluation of the CENTAURO System*. In: *IEEE Robotics and Automation Magazine (RAM), Special Issue on Humanoid Robot Applications in Real World Scenarios* (2019).

[4] T. Klamt, M. Schwarz, C. Lenz et al.: *Remote mobile manipulation with the centauro robot: Full-body telepresence and autonomous operator assistance*. In: *Journal of Field Robotics* (2019).

[5] T. Cichon und J. Rossmann: *Digital Twins: Assist and Support Cooperation in Human-Robot Teams*. In: *International Conference on Control, Automation, Robotics and Vision (ICARCV)*. 2018.

[6] T. Cichon und J. Rossmann: *Robotic Teleoperation: Mediated and Supported by Virtual Testbeds*. In: *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. 2017.

[7] T. Cichon und J. Rossmann: *Simulation-based User Interfaces for Digital Twins: Pre-, In-, or Post-Operational Analysis and Exploration of Virtual Testbeds*. In: *Proceedings of the 31st European Simulation and Modelling Conference (ESM), October 25–27, IST, Lisbon, Portugal*. IEEE. 2017.

[8] T. Cichon, C. Schlette und J. Rossmann: *Towards a 3D simulation-based operator interface for teleoperated robots in disaster scenarios*. In: *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. 2016, S. 264–269.

[9] T. Cichon, M. Priggemeyer und J. Rossmann: *Simulation-based Control and Simulation-based Support in eRobotics Applications*. In: *Applied Mechanics & Materials* 840 (2016).

[10] T. Cichon, C. Loconsole, D. Buongiorno et al.: *Combining an exoskeleton with 3D simulation in-the-loop*. In: *9th International Workshop on Human Friendly Robotics*. 2016, S. 31–34.

[11] G. Grinshpun, T. Cichon, D. Dipika et al.: *From Virtual Testbeds to Real Lightweight Robots: Development and deployment of control algorithms for soft robots, with particular reference to*. In: *Proceedings of the 47st International Symposium on Robotics (ISR)*. 2016, S. 1–7.

[12] L. Atorf, T. Cichon und J. Rossmann: *Flexible Data Logging, Management, and Analysis of Simulation Results of Complex Systems for eRobotics Applications*. In: *29th European Simulation and Modelling (ESM) Conference Proceedings*. 2015.

[13] J. Rossmann, M. Schluse, M. Rast et al.: *Simulation Technology for Soft Robotics Applications*. In: *Soft Robotics*. Springer, 2015, S. 100–119.

[14] T. Cichon, M. Rast und J. Rossmann: *Simulation tools for robotics: survey and comparison*. In: *28th European Simulation and Modelling (ESM) Conference Proceedings*. 2014.

Webseitenverzeichnis

[a] *BMBF Platform Industrie 4.0: Glossar*. https://www.plattform-i40.de/I40/Navigation/DE/Service/Glossar/glossar.html. (aufgerufen am 31.08.2018).

[b] *DARPA Robotics Challenge 2015*. http://www.theroboticschallenge.org/. [Online; accessed 2015-09-24].

[c] *Fokushima Daichii Hi-Res Fotos*. http://cryptome.org/eyeball/daiichi-npp/daiichi-photos.htm. (aufgerufen am 06.02.2019).

[d] *BMBF Platform Industrie 4.0*. https://www.plattform-i40.de/. (aufgerufen am 07.02.2019).

[e] *Top Trends in the Gartner Hype Cycle for Emerging Technologies, 2017*. https://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cycle-for-emerging-technologies-2017/. (aufgerufen am 28.12.2017).

[f] *Vim – the ubiquitous text editor*. https://www.vim.org/. (aufgerufen am 01.11.2018).

[g] *3D Connexion Space Mouse Pro*. https://www.3dconnexion.de/products/spacemouse/spacemousepro.html. (aufgerufen am 28.12.2017).

[h] *Youtube-Video: Myo-Armband*. https://www.youtube.com/watch?v=te1RBQQlHz4. (aufgerufen am 02.12.2017). 2015.

[i] *Myo-Armband: Online-Diagnostik-Tool*. http://diagnostics.myo.com/. (aufgerufen am 12.02.2018).

[j] *Google Soli*. https://atap.google.com/soli/. (aufgerufen am 15.12.2017).

[k] *Time of Flight-Kamera Tiefenbild*. http://www.all-electronics.de/3d-kamera-mit-time-of-flight/. (aufgerufen am 02.12.2017).

[l] *Optische 3D-Messung von großen Bauteilen*. http://www.inspect-online.com/topstories/control/optische-3d-messung-von-grossen-bauteilen. (aufgerufen am 02.12.2017).

[m] A. Colgan: *LeapMotion Blog: How does the Leap Motion Controller work*. http://blog.leapmotion.com/hardware-to-software-how-does-the-leap-motion-controller-work/. (aufgerufen am 02.12.2017). 2014.

[n] *Sensable Geomagic Touch X*. http://www.geomagic.com/en/products/phantom-desktop/overview. (aufgerufen am 03.07.2018).

[o] *Cyber Glove*. http://www.cyberglovesystems.com/cyberglove-iii/. (aufgerufen am 18.03.2018).

[p] *Festo Exo-Hand*. https://www.festo.com/group/de/cms/10233.htm. (aufgerufen am 18.03.2018).

[q] *VR Gluv*. https://www.vrgluv.com/. (aufgerufen am 18.03.2018).

[r] *Cyber Grasp*. http://www.cyberglovesystems.com/cybergrasp/. (aufgerufen am 18.03.2018).

[s] *Wolverine*. http://shape.stanford.edu/research/wolverine/. (aufgerufen am 18.03.2018).

[t] *Dexmo*. http://www.dextarobotics.com/. (aufgerufen am 18.03.2018).

[u] *Axon VR*. http://axonvr.com/. (aufgerufen am 28.12.2017).

[v] *Virtuix Omni*. http://www.virtuix.com/. (aufgerufen am 28.12.2017).

[w] *Microsoft Hololens Bloom*. https://developer.microsoft.com/en-us/windows/mixed-reality/navigating_the_windows_mixed_reality_home. (aufgerufen am 15.12.2017).

[x] *Take On Mars*. http://mars.takeonthegame.com/. (aufgerufen am 04.02.2019).

[y] *Wearable Technologies: AR-Brillen*. https://www.wearable-technologies.com/2019/01/ces-2019-digilens-unveils-augmented-reality-smartglasses/. (aufgerufen am 04.02.2019).

[z] *Realite-Virtuelle: HoloLens*. https://www.realite-virtuelle.com/mise-a-jour-hololens-nouveautes. (aufgerufen am 04.02.2019).

[aa] *Crytek Crysis: Cockpit view*. https://blogs-images.forbes.com/erikkain/files/2013/03/Crysis-3-Invisible.jpg. (aufgerufen am 12.02.2018).

[ab] *VR Fitness Insider: The future of AR in fitness*. https://www.vrfitnessinsider.com/the-future-of-augmented-reality-in-fitness/. (aufgerufen am 12.02.2018).

[ac] *Sailboat Control featuring a 3D Augmented Reality Interface*. http://tomorrowlab.cs.uvic.ca/. (aufgerufen am 12.02.2018).

[ad] *Fully Automated Combat Robots: Tactical Augmented Reality*. http://peakoil.com/forums/post1362052.html. (aufgerufen am 12.02.2018).

[ae] *Microsoft App Views*. https://docs.microsoft.com/en-us/windows/mixed-reality/app-views. (aufgerufen am 23.10.2018).

[af] *FRANKA Emika Panda Robot*. https://www.franka.de/. (aufgerufen am 15.12.2017).

[ag] *OPC Foundation*. https://opcfoundation.org/. (aufgerufen am 27.01.2019).

[ah] *MQTT – Message Queuing Telemetry Transport*. http://mqtt.org/. (aufgerufen am 27.01.2019).

[ai] *OMG DDS – OMG Data-Distribution Service for Real-Time Systems*. https://www.omg.org/omg-dds-portal/. (aufgerufen am 27.01.2019).

[aj] *ROS Wiki*. http://wiki.ros.org/. (aufgerufen am 30.10.2018).

[ak] *ROS2 Design – Webseite*. http://design.ros2.org/articles/ros_on_dds.html. (aufgerufen am 03.12.2017).

[al] *CoAP – Constrained Application Protocol*. http://coap.technology/. (aufgerufen am 27.01.2019).

[am] *XMPP – Extensible Messaging and Presence Protocol*. https://xmpp.org/. (aufgerufen am 27.01.2019).

[an] *AMQP – Advanced Message Queuing Protocol*. https://xmpp.org/. (aufgerufen am 27.01.2019).

[ao] *ROS Message Queue*. http://wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers. (aufgerufen am 03.12.2017).

[ap] *OMG Systems Modeling Language (OMG SysML)*. http://www.omgsysml.org/. (aufgerufen am 23.03.2018).

[aq] *Sparx Systems: Enterprise Architect*. www.sparxsystems.de. (aufgerufen am 23.03.2018).

[ar] *Bullet Physik-Engine*. http://bulletphysics.org/wordpress/. (aufgerufen am 29.12.2017).

[as] *ODE Physik-Engine*. http://www.ode.org/. (aufgerufen am 29.12.2017).

[at] *Nvidia PhysX Physik-Engine*. http://www.nvidia.de/object/nvidia-physx-de.html. (aufgerufen am 29.12.2017).

[au] *Havok Physik-Engine*. https://www.havok.com/. (aufgerufen am 29.12.2017).

[av] *ROBCAD*. https://www.plm.automation.siemens.com/de/products/tecnomatix/manufacturing-simulation/robotics/robcad.shtml. (aufgerufen am 19.01.2018).

[aw] *ROBOTEXPERT*. https://www.plm.automation.siemens.com/de/products/tecnomatix/manufacturing-simulation/robotics/robotexpert.shtml. (aufgerufen am 19.01.2018).

[ax] *Siemens PLM: PROCESSSIMULATE*. https://www.plm.automation.siemens.com/de/products/tecnomatix/manufacturing-simulation/robotics/process-simulate.shtml. (aufgerufen am 19.01.2018).

[ay] *Dassault Systemes: Delmia*. https://www.3ds.com/de/produkte-und-services/delmia/. (aufgerufen am 19.01.2018).

[az] *EASY-ROB*. http://www.easy-rob.com/easy-rob-intro.html. (aufgerufen am 19.01.2018).

[ba] *V-REP*. http://www.coppeliarobotics.com/. (aufgerufen am 19.01.2018).

[bb] *GAZEBO*. http://gazebosim.org/. (aufgerufen am 19.01.2018).

[bc] *Unity Engine*. https://unity3d.com/de. (aufgerufen am 19.01.2018).

[bd] *Unreal Engine*. https://www.unrealengine.com/. (aufgerufen am 19.01.2018).

[be] *OGRE 3D Engine*. https://www.ogre3d.org/. (aufgerufen am 19.01.2018).

[bf] *CryEngine*. https://www.cryengine.com/. (aufgerufen am 19.01.2018).

[bg] *Blender*. https://www.blender.org/. (aufgerufen am 19.01.2018).

[bh] *SOLIDWORKS*. http://www.solidworks.de/. (aufgerufen am 13.01.2018).

[bi] *CATIA*. https://www.3ds.com/de/catia/. (aufgerufen am 13.01.2018).

[bj] *Z88 Aurora*. https://z88.de/. (aufgerufen am 13.01.2018).

[bk] *SideFX Houdini*. https://www.sidefx.com/products/houdini-apprentice/. (aufgerufen am 13.01.2018).

[bl] *VOXCAD*. https://www.creativemachineslab.com/voxcad.html. (aufgerufen am 13.01.2018).

[bm] *FANUC ROBOGUIDE*. http://www.fanuc.eu/de/de/roboter/zubeh%C3%B6r/roboguide. (aufgerufen am 13.01.2018).

[bn] *ABB RobotStudio*. http://new.abb.com/products/robotics/de/robotstudio. (aufgerufen am 13.01.2018).

[bo] *KUKA.Sim*. https://www.kuka.com/de-de/produkte-leistungen/robotersysteme/software/planung-projektierung-service-sicherheit/kuka_sim. (aufgerufen am 13.01.2018).

[bp] *Msc.Adams*. http://www.mscsoftware.com/de/product/adams. (aufgerufen am 13.01.2018).

[bq] *National Instruments: LabVIEW*. http://www.ni.com/de-de/shop/labview.html. (aufgerufen am 13.01.2018).

[br] *Modelica*. https://www.modelica.org/. (aufgerufen am 13.01.2018).

[bs] *Mathworks MATLAB*. https://de.mathworks.com/products/matlab.html. (aufgerufen am 13.01.2018).

[bt] *Player/Stage*. http://playerstage.sourceforge.net/doc/Stage-3.2.1/. (aufgerufen am 13.01.2018).

[bu] *Science Blog: Trägheitsellipsoid*. http://scienceblogs.de/mathlog/2012/05/18/topologie-von-flachen-ccxx-1/. (aufgerufen am 12.02.2018).

[bv] *ARGESIM*. www.argesim.org. (aufgerufen am 04.12.2017).

[bw] *DARPA DRC und VRC: Showdown der Rescue Robotik*. http://spectrum.ieee.org/robotics/humanoids/darpas-rescuerobot-showdown. (aufgerufen am 04.12.2017).

[bx] *Der Digitale Zwilling am Institut für Mensch-Maschine-Interaktion (MMI), RWTH Aachen University*. https://www.mmi.rwth-aachen.de/forschung/digitaler-zwilling/. (aufgerufen am 26.02.2019).

[by] *Functional Mockup Interface (FMI)*. http://fmi-standard.org/. (aufgerufen am 29.12.2017).

[bz] B. Victor: *Seeing Spaces*. http://worrydream.com/SeeingSpaces/. (aufgerufen am 15.12.2017).

[ca] *OpenVR*. https://github.com/ValveSoftware/openvr/wiki/API-Documentation. (aufgerufen am 15.12.2017).

[cb] *ROS RQT Topic*. http://wiki.ros.org/rqt_topic. (aufgerufen am 17.12.2017).

[cc] *Valve Steam VR*. http://steamvr.com. (aufgerufen am 03.12.2017).

[cd] *ROS URDF-Definition*. http://wiki.ros.org/urdf. (aufgerufen am 03.03.2018).

[ce] *ROS URDF-Tutorial*. http://wiki.ros.org/urdf/Tutorials. (aufgerufen am 03.03.2018).

[cf] *MFTyre Packet*. https://www.tassinternational.com/delft-tyre-mf-tyremf-swift. (aufgerufen am 05.12.2017).

[cg] *Rigs of Rods*. https://rigsofrods.org/. (aufgerufen am 05.12.2017).

[ch] *Python*. www.python.org. (aufgerufen am 19.01.2018).

[ci] *Pythonbook*. http://www.spronck.net/pythonbook/. (aufgerufen am 19.01.2018).

[cj] *Numpy*. http://www.numpy.org/. (aufgerufen am 17.04.2018).

[ck] *Scipy*. http://www.scipy.org/. (aufgerufen am 17.04.2018).

[cl] *scikit-learn*. http://scikit-learn.org/stable/. (aufgerufen am 17.04.2018).

[cm] *SWIG*. http://www.swig.org/. (aufgerufen am 19.01.2018).

[cn] *Boost.Python*. http://www.boost.org/doc/libs/1_61_0/libs/python/doc/html/index.html. (aufgerufen am 19.01.2018).

[co] *PySide*. https://wiki.qt.io/PySide. (aufgerufen am 19.01.2018).

[cp] *PyQt*. https://wiki.python.org/moin/PyQt. (aufgerufen am 19.01.2018).

[cq] *PythonQt*. http://pythonqt.sourceforge.net/. (aufgerufen am 19.01.2018).

[cr] *ROS2 Design – Github*. https://github.com/ros2/ros1_bridge/blob/master/doc/index.rst. (aufgerufen am 18.04.2018).

[cs] *ROS TF package*. http://wiki.ros.org/tf. (aufgerufen am 17.04.2018).

[ct] *ROS PointCloud2*. http://wiki.ros.org/pcl/Overview. (aufgerufen am 17.04.2018).

[cu] *ROS PointCloud2-Tutorials*. http://wiki.ros.org/pcl/Tutorials. (aufgerufen am 17.04.2018).

[cv] *EU Horizon2020 project CENTAURO – Robust Mobility and Dexterous Manipulation in Disaster Response by Fullbody Telepresence in a Centaur-like Robot*. https://www.centauro-project.eu/. (aufgerufen am 28.12.2017).

[cw] *Schunk Five Finger Hand (SVH)*. https://schunk.com/de_de/greifsysteme/highlights/svh/. (aufgerufen am 03.02.2019).

[cx] *ReconCell*. www.reconcell.eu. (aufgerufen am 09.12.2017).

[cy] *Interempresas: Universal Robots in einer flexiblen Fertigungszelle*. http://www.interempresas.net/Robotica/Articulos/195161-Todas-ventajas-de-robotica-colaborativa-en-stand-de-Universal-Robots-en-Equiplast-2017.html. (aufgerufen am 06.02.2019).

[cz] *Robohub: Universal Robots in der kooperativen Montage*. https://robohub.org/42-companies-empowering-robots-and-humans-to-work-side-by-side/. (aufgerufen am 06.02.2019).

[da] *Universal Robots (UR)*. https://www.universal-robots.com/de/. (aufgerufen am 19.01.2018).

[db] *Tatsch-IT Blog: Neato Botvac Wifi Mod*. http://www.tatsch.it/neato-botvac-80-wifi-mod/. (aufgerufen am 09.12.2017).

[dc] *Tatsch-IT Blog: Neato Botvac ROS*. http://www.tatsch.it/use-the-neato-botvac-with-ros/. (aufgerufen am 09.12.2017).

[dd] *Geomagic Touch X: Datenblatt*. http://www.geomagic.com/en/products/phantom-desktop/specifications/. (aufgerufen am 17.04.2018).

[de] *Oculus Rift*. https://www.oculus.com/rift/. (aufgerufen am 28.12.2017).

[df] *HTC Vive*. https://www.vive.com/de/. (aufgerufen am 28.12.2017).

[dg] *Vuzix M300*. https://www.vuzix.com/Products/m300-smart-glasses. (aufgerufen am 28.12.2017).

[dh] *Epson Moverio BT 300*. https://www.epson.de/products/see-through-mobile-viewer/moverio-bt-300. (aufgerufen am 15.12.2017).

[di] *Microsoft Hololens*. https://www.microsoft.com/de-de/hololens. (aufgerufen am 28.12.2017).