Selbstkalibrierung der Hand-Kamera-Kinematik eines anthropomorphen Roboters

Diplomarbeit

Harald Andreas Uwe Hubert

Institut für Informatik VI
Autonome Intelligente Systeme
Rheinische Friedrich-Wilhelms-Universität Bonn

Betreut durch Prof. Dr. Sven Behnke

29. März 2012

Erklärung

Entsprechend DPO 1998 §19 Abs. 7 versichere ich hiermit, dass die Arbeit von mir selbstständig durchgeführt wurde, ich nur die angegebenen Quellen und Hilfsmittel verwendet und Zitate kenntlich gemacht habe.

Bonn, den 29. März 2012 Harald Andreas Uwe Hubert

Abstract

Präzise Steuerung ist wichtig bei der Interaktion eines Roboters mit seiner Umwelt. Hierfür wird ein genaues kinematisches Modell des Roboters, insbesondere der Arme, benötigt. Für Haushaltsroboter ist es nicht praktikabel, das Modell nach jeder Veränderung des Roboters manuell anzupassen. Gesucht ist eine Methode, mit dem sich der Roboter selber in beliebiger Umgebung mit seinen Sensoren kalibrieren kann. Es wird ein Verfahren vorgestellt, bei dem der Roboter seine Kinematik auf Basis eines ungefähren Modells exploriert. Der Roboter bewegt dabei seinen Endeffektor und nimmt ihn mit einer Kamera auf. Das kinematische Modell wird dann in Denavit-Hartenberg Form gebracht und mit den Daten mit Hilfe der Maximum-a-posteriori-Methode optimiert. Die Optimierungen werden dann in das ursprüngliche Modell zurück übertragen. Um eine kinematische Kette mit neun Freiheitsgraden zu kalibrieren, kommt das Verfahren mit wenigen hundert Messungen und geringem Rechenaufwand aus. Dieses Vorgehen wird dann um lokales Lernen erweitert. Dabei werden mehrere Modelle erzeugt, wobei jedes für einen Teilbereich des Konfigurationsraums gültig ist.

Inhaltsverzeichnis

1 Einleitung **1**

2 Verwandte Arbeiten **5**

3 Grundlagen **8**
3.1 Homogene Vektoren und Matrizen 8
3.2 Kinematik 9
3.3 URDF-Kinematik 10
3.4 Denavit-Hartenberg Konvention 12
3.5 KDL-Kinematik 15
3.6 Statistik 17
3.7 MAP 18
3.8 Rprop 18
3.9 Rotationen 19

4 Systembeschreibung **20**
4.1 Cosero und Dynamaid 20
4.2 ROS 22
4.3 NimbRo-Robot-Control 25
4.4 URDF 25
4.5 KDL 26
4.6 OpenCV 28
4.7 pr2_calibration 29
4.8 DH-Converter von Jörg Stückler 30

5 Ansatz **32**
5.1 Erstellung einer Kalibrierung 32
5.2 MAP 33
5.3 Berechnung des Gradienten der FK 34
5.4 Gradientenverfahren 35
5.5 Vergleich zwischen URDF-Ketten und DH-Ketten 35
5.6 Erstellung einer URDF-Kette aus einer DH-Kette 36
5.7 Modifikation einer URDF-Kette mit den Differenzen von DH-Ketten . . . 38
5.8 Gewinnung einer URDF-Kette aus einem URDF-Baum 39
5.9 Modifikation eines URDF-Baums anhand einer URDF-Kette 40
5.10 Erstellung einer URDF-Kette aus einer KDL-Kette 41
5.11 Alternative Formulierung als Offlineverfahren 42

5.12 Austausch des a priori Modells . . . 44
5.13 Mehrere Modelle - Lokal gewichtetes Lernen . . . 44

6 Implementierung **45**
6.1 Exploration . . . 46
6.2 Kalibrierung und Analyse . . . 53
6.3 Evaluation . . . 64
6.4 Nachrichten . . . 67
6.5 Betrieb der Hardware . . . 67
6.6 Simulation . . . 67
6.7 Tests . . . 68
6.8 KDL-Wrapper . . . 68
6.9 Modultester . . . 68
6.10 Hilfsmodule . . . 69
6.11 DH-Konverter . . . 69
6.12 Halterung . . . 69

7 Experimente **71**
7.1 Referenzwerte für die Posenbestimmung . . . 71
7.1.1 Manuell . . . 71
7.1.2 Simulation Cosero . . . 72
7.1.3 Simulation PR2 . . . 73
7.2 10000 im Gelenkraum gleichverteilte Stellungen . . . 75
7.2.1 Keine Abweichungen, 100 Iterationen, Varianz 0,01 . . . 76
7.2.2 Keine Abweichungen, 100 Iterationen, Varianz 1,0 . . . 78
7.2.3 Keine Abweichungen, 200 Iterationen, Austausch alle 70, Varianz 0,1 . . . 79
7.2.4 Abweichungen im Ellenbogen, 100 Iterationen, Varianz 0,01 . . . 80
7.2.5 Abweichungen im Ellenbogen, 100 Iterationen, Varianz 0,1 . . . 82
7.2.6 Abweichungen im Ellenbogen, 200 Iterationen, Varianz 0,1 . . . 83
7.2.7 Abweichungen im Ellenbogen, 200 Iterationen, Austausch alle 70, Varianz 0,01 . . . 84
7.2.8 Abweichungen im Ellenbogen, 200 Iterationen, Austausch alle 70, Varianz 0,1 . . . 85
7.2.9 Abweichungen in allen Gelenken, 200 Iterationen, Austausch alle 70, Varianz 0,1 . . . 86
7.2.10 Abweichungen in allen Gelenken, 100 Iterationen, Varianz 1,0 . . . 88
7.3 370 im Gelenkraum gleichverteilte Stellungen . . . 89
7.3.1 Keine Abweichungen, 200 Iterationen, Varianz 0,01 . . . 90
7.3.2 Keine Abweichungen, 200 Iterationen, Varianz 1,0 . . . 91
7.3.3 Abweichungen in allen Gelenken, 200 Iterationen, Varianz 1,0 . . . 92
7.3.4 Abweichungen in allen Gelenken, 200 Iterationen, Austausch alle 70, Varianz 0,1 . . . 94

7.3.5 Abweichungen in allen Gelenken, Modell 2, 200 Iterationen, Austausch alle 70, Varianz 0,1 . . . 95
7.3.6 Abweichungen in allen Gelenken, Modell 3, 200 Iterationen, Austausch alle 70, Varianz 0,1 . . . 97
7.4 370 in den erlaubten Grenzen gleichverteilte Stellungen . . . 98
7.4.1 Keine Abweichungen, 200 Iterationen, Varianz 0,1 . . . 99
7.4.2 Keine Abweichungen, 340 Iterationen, Austausch alle 70, Varianz 0,5 . . . 101
7.4.3 Abweichungen in allen Gelenken, 200 Iterationen, Austausch alle 70, Varianz 0,1 . . . 102
7.4.4 Abweichungen in allen Gelenken, 340 Iterationen, Austausch alle 70, Varianz 0,5 . . . 103
7.5 370 Stellungen aus Vorgaben . . . 104
7.5.1 Keine Abweichungen, 200 Iterationen, Austausch alle 70, Varianz 0,1 . . . 105
7.5.2 Keine Abweichungen, 340 Iterationen, Austausch alle 70, Varianz 0,5 . . . 106
7.5.3 Abweichungen in allen Gelenken, 200 Iterationen, Austausch alle 70, Varianz 0,1 . . . 107
7.5.4 Abweichungen in allen Gelenken, 340 Iterationen, Austausch alle 70, Varianz 0,5 . . . 108
7.5.5 Abweichungen in allen Gelenken, 480 Iterationen, Austausch alle 70, Varianz 1,5 . . . 109
7.6 Simulation von Laser und Schachbrett . . . 110
7.7 Datenaufnahme Dynamaid . . . 114
7.7.1 Selbstexploration Dynamaid, 327 Stellungen aus 71 Basisstellungen 115
7.7.2 Einlernen von 128 Stellungen bei Dynamaid . . . 116
7.7.3 Selbstexploration Dynamaid, 597 Stellungen aus 128 Basisstellungen 118
7.7.4 Selbstexploration Dynamaid, 924 Stellungen aus 199 Basisstellungen 119
7.7.5 Kalibrierung Dynamaid, Training 597 vs. Test 327 . . . 120
7.7.6 Kalibrierung Dynamaid, Training 597 vs. Test 128 . . . 120
7.7.7 Kalibrierung Dynamaid, Training 924 vs. Test 128 . . . 120
7.8 Anpassung der Lernparameter . . . 121
7.8.1 Getriebebeiwert geringer . . . 121
7.8.2 DH-Parameter geringer . . . 121
7.8.3 θ-Parameter höher . . . 122
7.8.4 d-, a- und α-Parameter höher . . . 122
7.8.5 d-, a- und α-Parameter geringer . . . 122
7.8.6 Angepasste Parameter, 597 Stellungen . . . 123
7.8.7 Angepasste Parameter, 327 Stellungen . . . 124
7.8.8 Angepasste Parameter, 924 Stellungen . . . 124
7.9 Messung der Wiederholgenauigkeit von Dynamaid . . . 125
7.10 Dynamaid richtet Laser auf Schachbrett . . . 125

7.11 Lokal gewichtetes Lernen . . . 126
7.11.1 924 Stellungen, Varianz 0,3 . . . 127
7.11.2 327 Stellungen, Varianz 0,3 . . . 130
7.11.3 327 Stellungen, Varianz 0,1 . . . 133

8 Auswertung **134**
8.1 Referenzwerte für die Posenbestimmung . . . 134
8.2 10000 im Gelenkraum gleichverteilte Stellungen . . . 134
8.3 370 im Gelenkraum gleichverteilte Stellungen . . . 136
8.4 370 in den erlaubten Grenzen gleichverteilte Stellungen . . . 136
8.5 370 Stellungen aus Vorgaben . . . 137
8.6 Vergleich der Trainingsreihen . . . 137
8.7 Simulation von Laser und Schachbrett . . . 137
8.8 Datenaufnahme Dynamaid . . . 137
8.9 Anpassung der Lernparameter . . . 138
8.10 Messung der Wiederholgenauigkeit von Dynamaid . . . 139
8.11 Dynamaid richtet Laser auf Schachbrett . . . 139
8.12 Lokal gewichtetes Lernen . . . 140

9 Zusammenfassung und Ausblick **141**

A Messungen **143**
A.1 Monoskopische Posenbestimmung . . . 143
A.2 Laser auf Schachbrett . . . 143

Glossar **147**

Tabellenverzeichnis **149**

Abbildungsverzeichnis **150**

Literaturverzeichnis **154**

1 Einleitung

Die Haushaltsroboter Dynamaid und Cosero wurden am Institut für Autonome Intelligente Systeme der Universität Bonn entwickelt [Nim12]. Ihre Aufgabe ist die Verrichtung von Tätigkeiten im häuslichen Umfeld. Damit diese Roboter effizient mit ihrer Umwelt interagieren können, ist eine präzise Koordination der Arme und Sensoren notwendig.

Ein übliches Verfahren zur präzisen Steuerung von Robotern ist Visual Servoing [HHC96; Cor94; KGD+03]. Dabei werden Gelenke anhand von optischen Sensordaten geregelt. Bei diesem Verfahren muss die Lage zwischen Endeffektor und Ziel aus den Kamerabildern bestimmt werden. Dies erfordert eine umfassende visuelle Abdeckung des Arbeitsraums, wie etwa durch Kameras in den Armen [pr212a], da es selbst unter günstigen Bedingungen bei zentral montierten Sensoren regelmäßig zu Überdeckungen kommt. In Abbildung 1.1 sehen wir wie der Roboter Cosero einen Löffel greift. Hier

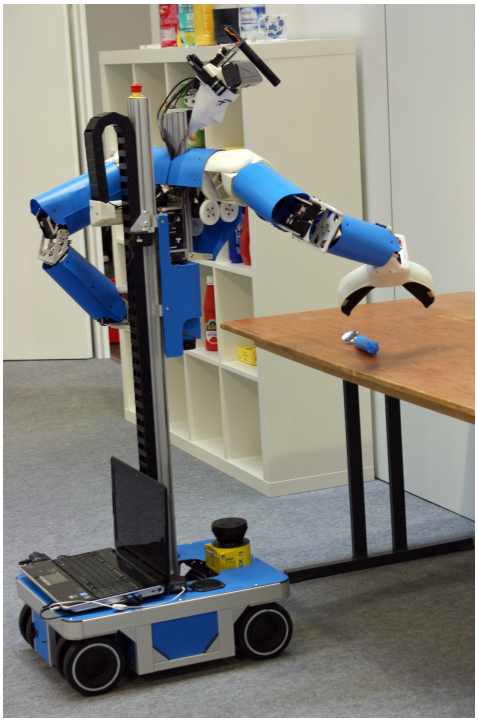


Abbildung 1.1: Cosero greift einen Löffel

kommt es in der Endphase der Greifbewegung zur Überdeckung des Löffels durch den Greifer im Bild der Kopfkameras. Ähnliche Probleme treten auch bei anderen Sensoren auf. Um die Menge an Sensoren klein und so das Hardwaredesign schlank zu halten, ist

Visual Servoing nicht hinreichend. Es ist ein exaktes kinematisches Modell eines Roboters notwendig. Dies erlaubt Bewegungen im Voraus zu planen und sie ohne ständigen Sichtkontakt bis zum Ziel durchzuführen.

Durch die Leichtbauweise der Roboter, die wechselnden Beanspruchungen und durch Umbauten verliert ein einmal erstelltes Modell seine Genauigkeit. Es ist nicht praktikabel das Modell nach jeder Veränderung des Roboters manuell anzupassen, da es schwierig ist, den Roboter präzise auszumessen. So erweist sich beispielsweise die genaue Bestimmung der Nullstellung der Servos als problematisch. Zudem ist das Vermessen mit einem hohen zeitlichen Aufwand verbunden.

Gesucht ist eine Methode, mit der sich der Roboter selber in beliebiger Umgebung mit seinen Sensoren kalibrieren kann. Dabei soll von einem ungenauen Modell des Roboters ausgegangen werden. Dieses wird dann angepasst, wobei die Anzahl, Art und Reihenfolge der Gelenke unverändert bleibt. Eine Schwierigkeit liegt in der hohen Zahl der Freiheitsgrade der zu kalibrierenden kinematischen Ketten. Vom Kopf bis zu einem Endeffektor haben die hier beschriebenen Roboter neun Freiheitsgrade.

Für die Selbstexploration stehen nur die sensorischen Fähigkeiten des Roboters zur Verfügung. Für die Kalibrierung müssen mindestens die Zustände der Gelenke und die Pose des jeweiligen Endeffektors erhoben werden. Die Gelenkstellungen stehen bei den vorgestellten Robotern als Messwerte aus den Servos zur Verfügung. Die visuelle Wahrnehmung bietet sich als Bezugsrahmen an, da die Endeffektoren der Arme Gegenstände bewegen sollen, welche auch visuell lokalisiert werden. Geeignet sind dafür die Kopfkameras, diese können auf einen großen Teil des Arbeitsraumes der Endeffektoren ausgerichtet werden. Mit der Hilfe eines Positionsmarkers am Endeffektor kann dessen Pose monoskopisch bestimmt werden [Ope12b]. Optische Positionsmarker haben den Vorteil, dass sie relativ einfach angebracht werden können.

Ein Verfahren, welches auf Kameras basiert, ist auf ein breites Spektrum von Robotern anwendbar, da diese weit verbreitete Sensoren sind. Hierzu zählen sowohl Mono- und Stereokameras wie auch Kameras mit Tiefensensor, wie etwa die Kinect [Kin12]. Für eine weitere Steigerung der Generalisierbarkeit können auch die Gelenkstellungen mit Hilfe der Steuerkommandos abgeschätzt werden, sofern jene nicht direkt messbar sind.

In Abbildung 1.2 wird das Bild einer Kamera im Kopf von Dynamaid gezeigt. Zu sehen ist der rechte Greifer, auf dem ein Schachbrettmuster als Positionsmarker befestigt ist. In das Bild sind von der Visualisierungssoftware *Rviz* zusätzliche Informationen hineinprojiziert worden. Einerseits ist die leicht nach links versetzte Projektion des Modells des Greifers zu sehen. Die breiten bunten Balken repräsentieren dabei das Koordinatensystem des Endeffektors so, wie es die Vorwärtskinematik (VK) voraussagt. Rechts sehen wir ein zweites Koordinatensystem aus dünnen Strichen, welches die monoskopisch erkannte Pose des Schachbrettmusters zeigt. Wir sehen hier deutlich die Abweichung zwischen dem realen Roboter und seinem Modell.

Ein Ziel der vorliegenden Arbeit ist die **Entwicklung eines effizienten Verfahrens zur Kalibrierung der Kinematik mit Hilfe visueller Sensordaten basierend auf der Maximum-a-posteriori-Methode (MAP)** [Bis06]. Hierbei wird das kinematische Modell, welches der Denavit-Hartenberg (DH)-Konvention [SHV04] entspricht, mit Bilddaten und Gelenkstellungen aus einer Selbstexploration optimiert. Dabei wird

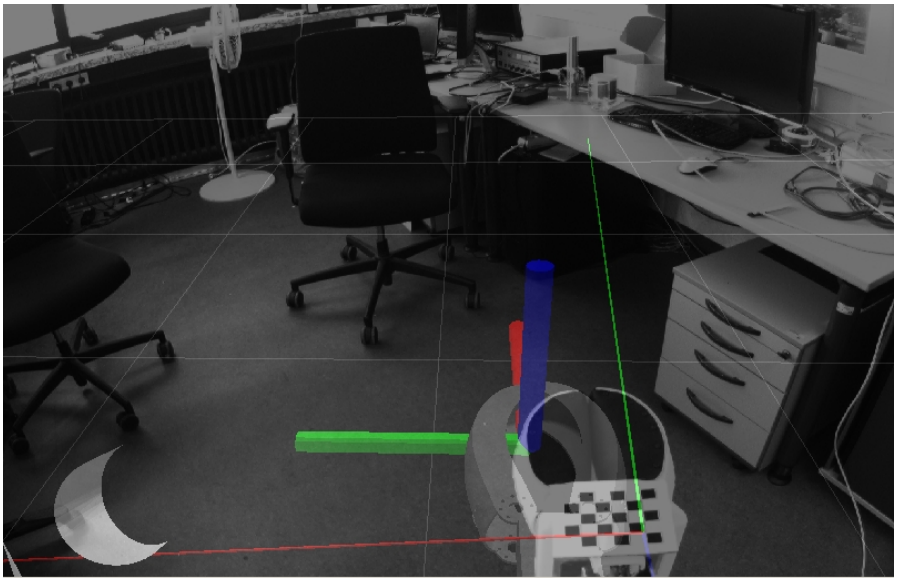


Abbildung 1.2: Endeffektor im Videobild

eine kinematische Kette mit neun Freiheitsgraden mit einigen hundert Roboterstellungen kalibriert.

Eine weitere Aufgabe ist die **Implementierung des Verfahrens in der gegebenen Systemumgebung**. Dazu wird eine Exploration implementiert, bei der der Roboter vorgegebene Konfigurationen der Arme und des Kopfes anfährt. Dabei nimmt er jeweils die Endeffektorstellung anhand eines Markers auf. Für die Kalibrierung mit MAP wird das Modell des Roboters erst in eine der DH Konvention entsprechende Form gebracht, so dass es frei von Redundanzen ist. Nach der Optimierung wird das Gelernte dann in das ursprüngliche Modell zurück übertragen. Das Erlernen der kinematischen Parameter ermöglicht die Weiterverwendung der Komponenten, welche VK und inverse Kinematik (IK) berechnen.

Ein weiteres Ziel ist die **Evaluation des Verfahrens**. Dabei wird mit künstlich erzeugten Daten die Güte des Verfahrens überprüft. Die Einbindung in das bestehende Umfeld wird in Simulationen getestet. Zusätzlich werden mit einem Roboter Daten erhoben, mit denen das Verfahren anhand realer Messungen evaluiert wird. Mit den erhobenen Daten wird eine Kalibrierung durchgefühhrt, deren Güte in Experimenten zur Wiederhol- und Positioniergenauigkeit überprüft wird.

Im weiteren wird lokalisiertes Lernen evaluiert. Dabei wird der Konfigurationsraum in Teilräume aufgeteilt, für die dann jeweils ein Modell gelernt wird.

Das Dokument ist im Weiteren wie folgt aufgebaut. Ähnlichkeiten und Unterschiede zu anderen Arbeiten werden in Kapitel 2 behandelt. Die theoretischen Grundlagen werden in Kapitel 3 betrachtet. Das Kapitel 4 beschreibt die Hard- und Softwareumgebung. Der gewählte Ansatz wird in Kapitel 5 vorgestellt. Die Implementierung der Lösung wird in Kapitel 6 beschrieben. Eine Evaluation der Arbeit befindet sich in Kapitel 7.

Die Bewertung der Ergebnisse findet in Kapitel 8 statt. Zusammenfassung und Ausblick befinden sich in Kapitel 9. Rohdaten aus Experimenten werden in Anhang A aufgeführt.

2 Verwandte Arbeiten

De Angulo und Torras verfeinern in [AT97] den Ansatz von Ritter et al. die IK mit einer Self-Organizing-Map zu lernen. Es wird nicht wie bei Ritter die IK von Grund auf neu gelernt, sondern sie verwendeten die im Roboterarm fest eingebaute und nicht zu umgehende IK. Diese wird mit modifizierten Arbeitsraum-Koordinaten weiter betrieben, um Beschädigungen und Abnutzung des Armes auszugleichen.

In [AT05] und [AT08] führen die gleichen Autoren Verfahren ein, um beim nun vollständigen Lernen der IK die Anzahl der zu messenden Posen drastisch zu reduzieren. Statt mehrerer tausend sind nurnoch einige hundert Posenmessungen notwendig, um einen Arm mit sechs Freiheitsgraden zu kalibrieren. Die erste Arbeit bezieht sich auf Roboterarme bei denen die letzten drei Freiheitsgrade um ein gemeinsames Zentrum rotieren. Die zweite Arbeit kommt ohne diese Annahme aus, benötigt aber eine zusätzliche Marke am mittleren Segment des Roboterarmes.

D'Souza et al. beschreiben in [DVS01] ein weiteres Verfahren zum Lernen der IK des Endeffektors bei dem der Arbeitsraum in einzelne rezeptive Felder aufgeteilt wird. In diesen Feldern werden die kinematischen Funktionen jeweils mit linearer Regression iterativ lokal angepasst. Unter der Vorgabe eines Optimierungskriteriums, hier die Nähe zu einer *natürlichen* Haltung des Roboters, erlaubt das Verfahren eine optimale Auflösung der vorkommenden Redundanzen der IK.

In [UAA+09] stellen Ulbricht et al. ein Verfahren zum Erlernen der VK des Endeffektors vor. Sie parametrisieren mit den Gelenkwinkeln Bézierflächen, welche die Posen des Endeffektors beschreiben. Dies ermöglicht eine Linearisierung des Problems und das Erlernen der VK mit Hilfe von 3^f Posenmessungen, dabei ist f die Anzahl der Freiheitsgrade. Die Parameter des Modells, also die Stützstellen der Bézierflächen, können hierbei auf zwei Arten berechnet werden. Bei dem einem Verfahren wird die Moore-Penrose-Pseudoinverse verwendet, um die Parameter zu bestimmen, dies sogar exakt, sofern kein Rauschen vorliegt. Das vorgestellte Gradientenabstiegsverfahren bietet andererseits die Möglichkeit des Online-Lernens.

Das von Schulz et al. in [SOSB09] vorgestellte Verfahren kommt nur mit der Position des Endeffektors aus, um dessen Kinematik zu lernen. Die Position des Endeffektors wird dabei mit externen Sensoren bestimmt. Durch das Ansteuern zufällig gewählter Konfigurationen wird eine Datenbasis aus Gelenkstellungen und zugehörigen Endeffektorpositionen gebildet. Wenn eine Endeffektorposition erreicht werden soll, so wird im Weltkoordinatensystem nach den nächsten bekannten Nachbarn dieses Punktes in der Datenbasis gesucht. Aus diesen weden in ihrer Gelenkkonfiguration zueinander ähnliche Stellungen ausgewählt, zwischen denen dann linear interpoliert wird. Das Verfahren kommt dabei ohne a priori Modell aus, liefert aber keine Informationen über Zwischensegmente.

Die bisher vorgestellten Methoden erlauben keine direkte Bestimmung der DH-Parameter. Die nun folgenden Vorgehensweisen erlauben hingegen ihre Bestimmung. Bei Hersch et al. [HSB08] werden Transformationen zwischen den Referenzkoordinatensystemen der Segmente der kinematischen Kette gelernt. Das verwendete Gradientenverfahren konvergiert langsam und nicht in allen Fällen und ist auf genaue Gelenkwinkel angewiesen. Das alleinige Verfolgen des Endeffektors führt mitunter zu falschen Ergebnissen beim Auflösen von Mehrdeutigkeiten in der Kinematik. Dies macht hier die Verwendung von Marken an mehreren Segmenten notwendig, da die Posen der Zwischensegmente sonst nicht richtig erkannt werden.

Sturm et al. verwenden in [SPB09] Bayessche Netze, um die allgemeine Struktur der Kinematik zu lernen. Sie verwenden Gaußsche Prozesse, um die Transformationen zwischen den Koordinatensystemen von jeweils zwei Segmenten zu lernen. Hierbei werden nur wenige Posenmessungen benötigt bis sich eine Vorhersagegenauigkeit einstellt, welche die Messgenauigkeit übersteigt. Die Nachteile des Verfahrens sind, dass eine externe Kamera benötigt wird und das an allen Segmenten der kinematischen Kette Marken befestigt werden müssen. Da die Transformationen in Bezug auf Steuersignale gelernt werden, ist es möglich Gelenkübersetzungen und Offsets zu lernen.

Hart und Scassellati verwenden in [HS11] mehrere Stufen, um die Parameter der verwendeten Stereokamera und der Kinematik zu lernen. Zuerst schätzen sie mit Hilfe der kalibrierten Kameras die DH-Parameter ab. Hierfür darf die Kette nur aus Drehgelenken bestehen, deren Anzahl vorher bekannt sein muss. Bei der von ihnen vorgestellten *Circle Point Analysis* werden durch mehrere Bewegungen in jeweils einem Gelenk die Rotationsachse und bis auf Gelenkversatz und Getriebeübersetzung alle weiteren Parameter bestimmt. Die verbleibenden zwei Parameter werden in einem nachfolgenden Schritt bestimmt. Diese Ergebnisse dienen dann als Grundlage für einen nicht linearen Optimierungsschritt der Kinematik, der sich mit dieser Eingabe nun nicht in lokalen Minima verfangen soll. Es zeigt sich, dass die Genauigkeit deutlich steigt, wenn dabei auch die Kamerakalibrierung optimiert wird. Dieses gemeinsame Lernen von Stereokamera- und Kinematikkalibrierung führt zu einem extern gemessenen durchschnittlichen Positionsfehler von 2,29 mm bei lediglich 200 Samples. Die Längenparameter der kinematischen Kette werden gegenüber den realen Segmenten mit einer Genauigkeit von 1,1 mm bestimmt. Die getesteten kinematischen Ketten hatten dabei lediglich vier Freiheitsgrade und Segmente von maximal 130 mm Länge. Die Kalibrierung bei der Werkzeugnutzung, hier der Spitze eines Schraubenziehers, führte zu einer durchschnittlichen Positioniergenauigkeit von 7,18 mm.

Die Vorgehensweise bei der *Circle Point Analysis* ähnelt dabei dem von Vicentini et al. [VPMMT11]. Dort werden allerding nur zwei feste Transformationen gelernt, dies sind die Greifer-Sensor- beziehungsweise Greifer-Marker-Transformation X und die Roboter-Welt-Transformationen Z. In beiden Ansätzen werden mit Hilfe passender Bewegungen Mannigfaltigkeiten, hier Kreise, bestimmt, aus denen dann die gesuchten Parameter berechnet werden. Durch diesen Zwischenschritt soll das Rauschen der Messwerte weniger die Ergebnisse verfälschen.

Zhao stellt in [Zha11] ebenfalls ein Verfahren vor, um X und Z zu bestimmen. Er verwendet dabei konvexe Optimierung, welche ohne die Vorgabe von Startwerten auskommt.

Im Umfeld von ROS wurden Implementierungen zur Bestimmung der Greifer-Kamera-Transformation X unter anderem von Mohanarajah [Moh12] vorgenommen. Hier wurde der Entwurf von Park und Martin [PM94] umgesetzt. Eine weitere Implementierung findet sich in den *camera_pose_toolkits* [cam12b].

Pradeep et al. kalibrieren in [PKB10] optische Sensorkoordinatensysteme (X) und Versätze in den Gelenkstellungen des Roboters *PR2*. Die Parameter schätzen sie hierbei mit der Maximum-Likelihood-Methode ab. Der Roboter führt die Kalibrierung selbstständig mit Hilfe von einem im Greifer gehaltenen Marker durch und nimmt diesen mit seinen Sensoren in einem Explorationslauf auf. Die Datenerfassung entspricht hierbei dem in der vorliegenden Arbeit vorgestellten Vorgehen.

Roy und Thrun stellen in [RT99] ein probabilistisches Verfahren zur automatischen Kalibrierung der Odometrie mit Sensordaten vor. Der beschriebene inkrementelle Maximum-Likelihood-Algorithmus verwendet Sensor- und Bewegungsmodelle, um die systematischen Fehler der Odometrie zu berechnen. Diese gehen, mit dem zurückgelegten Weg gewichtet, als Korrekturglieder in die Posenbestimmung ein. Das System ist dabei in der Lage auf Veränderungen zu reagieren, da die gemessenen Daten, in Abhängigkeit von ihrem Alter, exponentiell an Bedeutung verlieren.

3 Grundlagen

Dieses Kapitel bereitet die theoretische Basis für den restlichen Teils der Arbeit. Die vorgestellten Verfahren berühren unter anderem die Themengebiete der Algebra, der Kinematik und der statistische Optimierung.

3.1 Homogene Vektoren und Matrizen

Im Folgenden werden Vektoren, Skalare und Funktionen mit solchen Werten im Bildraum klein geschrieben, zum Beispiel v, θ, x oder $f : \mathbb{R} \mapsto \mathbb{R}$. Matrizen und matrixwertige Funktionen werden groß geschrieben, z.B. M und $A(x)$.

Im Weiteren bezeichnet v_i die i. Komponente des Vektors v und $M_{i,j}$ das Element in der Reihe i und Spalte j der Matrix M. Seien $F^i : \mathbb{R} \to \mathbb{R}^{m \times m}, 1 \leq i \leq n$ matrixwertige Funktionen mit skalaren Parametern und v ein Vektor der Länge n, dann definieren wir die Funktion $F^i(v) := F^i(v_i)$.

Für die hier betrachtete Kinematik sind zwei geometrische Transformationen von Koordinatensystemen in $\mathbb{R}^3$ von zentraler Bedeutung, die Translation und die Rotation. Die Translation lässt sich durch eine Vektoraddition in $\mathbb{R}^3$ darstellen, die Rotation durch eine Multiplikation einer Matrix aus $\mathbb{R}^{3\times3}$ mit einem Vektor aus $\mathbb{R}^3$.

Die nacheinander folgende Ausführung von Rotation und Translation $w = M \cdot u + v$ sieht komponentenweise wie folgt aus:

$$\begin{aligned}\begin{pmatrix} w_1 \\ w_2 \\ w_3 \end{pmatrix} &= \begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} \\ m_{2,1} & m_{2,2} & m_{2,3} \\ m_{3,1} & m_{3,2} & m_{3,3} \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} + \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \\ &= \begin{pmatrix} m_{1,1} \cdot u_1 + m_{1,2} \cdot u_2 + m_{1,3} \cdot u_3 + v_1 \\ m_{2,1} \cdot u_1 + m_{2,2} \cdot u_2 + m_{2,3} \cdot u_3 + v_2 \\ m_{3,1} \cdot u_1 + m_{3,2} \cdot u_2 + m_{3,3} \cdot u_3 + v_3 \end{pmatrix}\end{aligned}$$

Mit Hilfe einer vierten Dimension kann man dies als eine Multiplikation einer homogenen Matrix aus $\mathbb{R}^{4\times4}$ mit einem homogenen Vektor aus $\mathbb{R}^4$ schreiben, siehe Spong et al. [SHV04, Kap 2.6]. Dabei gehen wir immer davon aus, dass die letzte Zeile von homogenen Vektoren und Matrizen 1 beziehungsweise (0 0 0 1) ist. Aus der obigen Gleichung

wird dann $w' = M' \cdot u'$ mit der komponentenweisen Darstellung

$$\begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ 1 \end{pmatrix} = \begin{pmatrix} m_{1,1} & m_{1,2} & m_{1,3} & v_1 \\ m_{2,1} & m_{2,2} & m_{2,3} & v_2 \\ m_{3,1} & m_{3,2} & m_{3,3} & v_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ 1 \end{pmatrix}$$
$$= \begin{pmatrix} m_{1,1} \cdot u_1 + m_{1,2} \cdot u_2 + m_{1,3} \cdot u_3 + v_1 \\ m_{2,1} \cdot u_1 + m_{2,2} \cdot u_2 + m_{2,3} \cdot u_3 + v_2 \\ m_{3,1} \cdot u_1 + m_{3,2} \cdot u_2 + m_{3,3} \cdot u_3 + v_3 \\ 1 \end{pmatrix}.$$

Man sieht die Übereinstimmung in den ersten drei Dimensionen. Die Transformationsmatrizen sind aus $SO(4)$ [Fis95] und somit invertierbar. Es ist leicht zu sehen, dass die lezte Zeile von Vektoren und Matrizen bei Multiplikation und Invertierung unverändert 1, respektive (0 0 0 1), bleibt. Wir definieren E als die 4×4-Einheitsmatrix. Die Lage eines Koordinatensystems A im Bezug auf Koordinatensystem B lässt sich durch die Matrix der T_A^B angeben, die die Punkte aus A in B abbildet. Sofern nichts anderes erwähnt wird, sprechen wir im Folgenden immer von homogenen Vektoren und Matrizen.

3.2 Kinematik

Im Folgenden betachten wir kinematische Ketten und Bäume, diese bestehen aus mit Gelenken verbundenen starren Segmenten. Im Allgemeinen kann es sich sowohl um prismatische Gelenke als auch um Drehgelenke handeln. In Abbildung 3.1 sehen wir eine Kette mit drei Segmenten und zwei Gelenken, Gelenk 1 ist prismatisch und Gelenk 2 rotiert.

Sofern nichts anderes angegeben wird, beschäftigen wir uns im Folgenden mit Drehgelenken. Die meisten Sachverhalte sind aber auf prismatische Gelenke übertragbar. Eine kinematische Kette führt von ihrem Wurzelsegment zum Endeffektorsegment. Im Folgenden verwenden wir die Bezeichnung Eltersegment sowohl als Beschreibung für den Vorgänger eines Gelenks als auch den Vorgänger eines Segments. Analoges gilt für Kindsegmente. Das Endeffektorsegment ist also Nachfahre des Wurzelsegments. Sei n die Anzahl der Gelenke einer Kette, dann ist die Anzahl der Segmente $n + 1$. Die Numerierung der Gelenke geht von 1 bis n, die der Segmente von 0 bis n. Das Gelenk i verbindet sein Eltersegment $i - 1$ mit seinem Kindsegment i. Zusammenhängende Teile von kinematischen Ketten und aneinander gehängte kinematische Ketten sind wieder kinematische Ketten. Bäume fassen mehrere, miteinander verbundene kinematische Ketten zusammen.

Die Stellung der Gelenke einer Kette wird mit dem Zustandsvektor θ beschrieben. Dabei ist θ_i der Zustand des Gelenks i. Weiter definieren wir θ^0 als den n-dimensionalen Nullvektor $\theta^0{:=}(0, \ldots, 0) \in \mathbb{R}^n$. Dieser beschreibt die Nullstellung der Kette, in der kein Gelenk bewegt wird. Dieser beinhaltet auch Fixierte gelenke, deren Stellung immer Null ist. Die Anzahl der beweglichen Gelenke, also die Zahl der Freiheitsgrade der Kette, bezeichnen wir mit f. Die Getriebeübersetzung eines Gelenks wird durch einen Beiwert

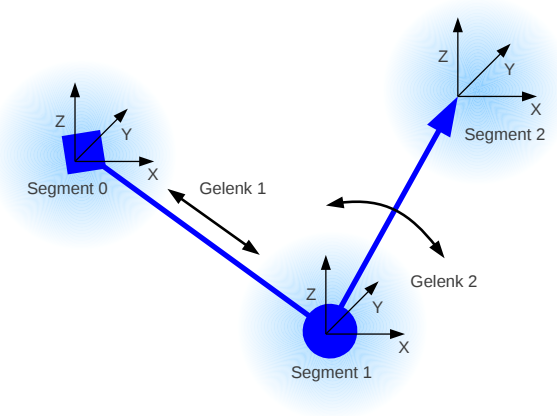


Abbildung 3.1: Kinematische Kette

modelliert, mit dem θ^i multipliziert wird. Ein Stellversatz wird durch darauf folgende Addition mit einem konstanten Wert erreicht.

Mit jedem Segment ist ein Koordinatensystem verbunden. Im weiteren werden Segmente und die ihnen zugeordneten Koordinatensysteme synonym verwendet. Von jedem Kindkoordinatensystem j führt bei gegebenem θ eine Transformation T_j^i ins Elterkoordinatensystem i. Man sieht leicht, dass die Invertierung der Transformation, respektive der Matrix, den Wechsel vom Elterkoordinatensystem ins Kindkoordinatensystem beschreibt. Wenn es im Folgenden nicht anders beschrieben wird, so führen Transformationen vom Koordinatensystem des Kindsegments ins Eltersegment. Um von einem Segmentkoordinatensystem in das eines Vorfahren zu transformieren, muss man nur die entsprechenden Einzeltransformationen konkatenieren. Das Weltkoordinatensystem soll einen festen Bezugsrahmen für die Betrachtung beweglicher Komponenten bieten. Es ist im Folgenden das Koordinatensystem von dem aus eine kinematische Kette startet, also das des Wurzelsegments. Mit T_{Kette} bezeichnen wir die Transformation vom Endeffektor- ins Weltkoordinatensystem einer kinematischen Kette. Im Folgenden werden verschiedene Modellierungen kinematischer Ketten vorgestellt. Diese unterscheiden sich unter anderem in der jeweiligen Lage der Ursprünge der Segmentkoordinatensysteme. Unser Interesse gilt in dieser Arbeit der Kinematik der Position, im Gegensatz zu der der Geschwindigkeit, welche hier keine Beachtung findet.

3.3 URDF-Kinematik

Die Kinematik eines Roboters wird im Robot Operating System (ROS) im Unified Robot Description Format (URDF) beschrieben, siehe [urd12] und Kapitel 4.4. URDF-

Parameter eines Gelenks beschreiben dessen Lage relativ zum Elterkoordinatensystem. Zusammen mit der Achsstellung und dem Gelenkzustand ergibt sich daraus die relative Lage des Koordinatensystems des Kindsegments. Bei der Beschreibung der URDF-Parametrisierung ist es sinnvoll, für jedes Gelenk drei Koordinatensysteme zu betrachten, das des Eltersegments, das des Kindsegments und dazwischen eines, welches dem Gelenk zugeordnet ist. Die URDF-Parameter eines Gelenks i bestehen aus den Tripeln u^i, r^i und gegebenenfalls a^i. In Abbildung 4.7 ist das Eltersegment $i-1$, Gelenk- und Kindkoordinatensysteme i sind identisch, das Gelenk befindet sich in Nullstellung. Die Koordinatensysteme sind grau dargestellt. Der Vektor u^i (rot) beschreibt den Versatz

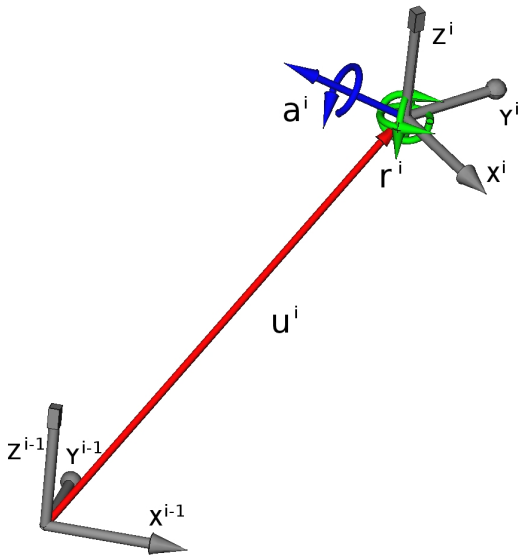


Abbildung 3.2: URDF Parameter eines Gelenks

des Ursprungs des Gelenkkoordinatensystems relativ zum Ursprung des Elterkoordinatensystems. Die relative Neigung des Gelenkkoordinatensystems wird durch das Tripel r^i dargestellt (grün). Dieses besteht aus Roll-, Nick-, und Gierwinkeln in Radiant. Aus Versatz und Neigung lässt sich die konstante Transformationsmatrix O^i bestimmen. Wenn das Gelenk beweglich ist, gibt a^i den Achsvektor des Gelenks relativ zum Gelenkkoordinatensystem an (blau). Zusammen mit dem Zustand des Gelenks θ_i ergibt sich hieraus das Koordinatensystem des Kindsegments. Die Achs-Transformationsfunktion des Gelenks A^i ist durch a^i und den Typ des Gelenks bestimmt und von θ_i abhängig. Es gilt $A^i(0) = E, 1 \leq i \leq n$. Ist das Gelenk steif, so sind Gelenk- und Kindkoordinatensystem immer identisch. Konkateniert man die gegebenen Transformationen, erhält man die Welt-Endeffektor-Transformation

$$T_{URDF}(\theta) = O^1 \cdot A^1(\theta) \cdots O^n \cdot A^n(\theta)\,.$$

Es werden noch weitere Parametrisierungen vorgestellt. Um einen allgemeinen Bezugsrahmen zu haben betrachten wir die URDF-Koordinatensysteme als die natürlichen

Koordinatensysteme eines Roboters; der Ursprung eines Segments entspricht dem der URDF-Parametrisierung.

Ein Stellversatz von Gelenk i wird durch passende Wahl von r^i modelliert. Eine Getriebeübersetzung wird durch Multiplikation von θ_i mit einer Konstanten erreicht. Siehe hierzu Kapitel 4.4

3.4 Denavit-Hartenberg Konvention

Der Vorteil der Beschreibung einer kinematischen Kette in Denavit-Hartenberg-Parametern [SHV04, Kap. 3] ist, dass diese frei von Redundanzen ist. Jedes Gelenk i lässt sich mit einem vierdimensionalen Vektor $d^i = (d^i_\theta \; d^i_d \; d^i_a \; d^i_\alpha)$, beschreiben. Die Parametrisierung geht davon aus, dass die Bewegungsachse i in der Z-Achse des Segmentkoordinatensystems $i-1$ liegt. DH-Segmentkoordinatensysteme entsprechen also häufig nicht den Segmentkoordinatnsystemen der URDF-Parametrisierung. Ein Beispiel ist in Abbildung 3.3 zu sehen. Um vom Koordinatensystem $i-1$ nach i zu kommen, benötigt man eine Drehung von d^i_θ um die Z-Achse (rot), eine Translation von d^i_d entlang der Z-Achse (gelb), eine Translation von d^i_a entlang der neuen X-Achse (grün) und eine Drehung von d^i_α um diese X-Achse (blau).

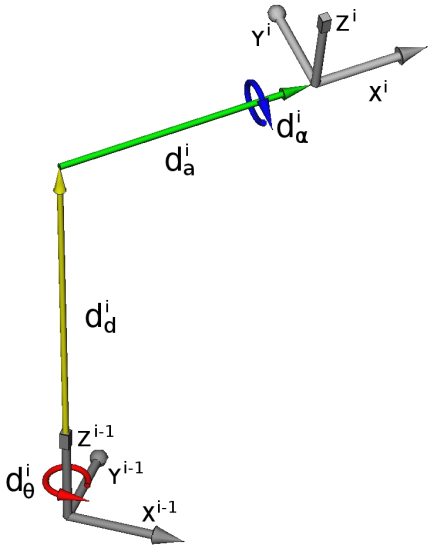


Abbildung 3.3: DH Parameter eines Gelenks

Die Transformationsmatrizen hierfür sehen wie folgt aus:

$$D_\theta(d^i_\theta) = \begin{pmatrix} \cos(d^i_\theta) & -\sin(d^i_\theta) & 0 & 0 \\ \sin(d^i_\theta) & \cos(d^i_\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$D_d(d_d^i) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_d^i \\ 0 & 0 & 0 & 1 \end{pmatrix},$$

$$D_a(d_a^i) = \begin{pmatrix} 1 & 0 & 0 & d_a^i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

und

$$D_\alpha(d_\alpha^i) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(d_\alpha^i) & -\sin(d_\alpha^i) & 0 \\ 0 & \sin(d_\alpha^i) & \cos(d_\alpha^i) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Die Transformationsmatrix D^i ergibt sich aus der Konkatenation dieser Matrizen, sie ist der Funktionswert der Konkatenation D der Funktionen:

$$\begin{aligned} D^i &= D(d^i) \\ &= D(d_\theta^i, d_d^i, d_a^i, d_\alpha^i) \\ &= \begin{pmatrix} \cos(d_\theta^i) & -\sin(d_\theta^i)\cos(d_\alpha^i) & \sin(d_\theta^i)\sin(d_\alpha^i) & d_a^i\cos(d_\theta^i) \\ \sin(d_\theta^i) & \cos(d_\theta^i)\cos(d_\alpha^i) & -\cos(d_\theta^i)\sin(d_\alpha^i) & d_a^i\sin(d_\theta^i) \\ 0 & \sin(d_\alpha^i) & \cos(d_\alpha^i) & d_d^i \\ 0 & 0 & 0 & 1 \end{pmatrix}. \end{aligned}$$

Eine Rotation um den Winkel θ_i in einem Drehgelenk i findet über die Substitution von d_θ^i durch $d_\theta^i + \theta_i$ statt. Translationen um θ_i werden durch die Ersetzung von d_d^i durch $d_d^i + \theta_i$ ermöglicht. Für die Gelenktransformationsfunktion $D'^i : \mathbb{R} \to \mathbb{R}^{4\times4}$ gilt dann $D'^i(\theta_i) = D(d_\theta^i + \theta_i, d_d^i, d_a^i, d_\alpha^i)$ für Drehgelenke und $D'^i(\theta_i) = D(d_\theta^i, d_d^i + \theta_i, d_a^i, d_\alpha^i)$ für prismatische Gelenke. Verallgemeinert gilt

$$\begin{aligned} &D'^i(\theta_i) \\ &\quad = D(d_\theta^i + \theta_i', d_d^i + \theta_i'', d_a^i, d_\alpha^i) \\ &\quad = \begin{pmatrix} \cos(d_\theta^i + \theta_i') & -\sin(d_\theta^i + \theta_i')\cos(d_\alpha^i) & \sin(d_\theta^i + \theta_i')\sin(d_\alpha^i) & d_a^i\cos(d_\theta^i + \theta_i') \\ \sin(d_\theta^i + \theta_i') & \cos(d_\theta^i + \theta_i')\cos(d_\alpha^i) & -\cos(d_\theta^i + \theta_i')\sin(d_\alpha^i) & d_a^i\sin(d_\theta^i + \theta_i') \\ 0 & \sin(d_\alpha^i) & \cos(d_\alpha^i) & d_d^i + \theta_i'' \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{aligned}$$

mit

$$(\theta_i', \theta_i'') = \begin{cases} (\theta_i, 0) & \text{wenn } i \text{ Drehgelenk und} \\ (0, \theta_i) & \text{wenn } i \text{ prismatisches Gelenk ist.} \end{cases}$$

Wir zerlegen nun D'^i in zwei Teile, eine Transformationsfunktion Z^i, die die Bewegung in Bezug auf die Z-Achse repräsentiert und die schon genannte konstante Transformation D^i. Mit

$$Z^i(\theta_i) = \begin{pmatrix} \cos(\theta'_i) & -\sin(\theta'_i) & 0 & 0 \\ \sin(\theta'_i) & \cos(\theta'_i) & 0 & 0 \\ 0 & 0 & 1 & \theta''_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

gilt dann $D'^i(\theta_i) = Z^i(\theta_i) \cdot D^i$, wie die folgende Gleichung belegt:

$$\begin{aligned}
&Z^i(\theta_i) \cdot D^i \\
&= \begin{pmatrix} \cos(\theta'_i) & -\sin(\theta'_i) & 0 & 0 \\ \sin(\theta'_i) & \cos(\theta'_i) & 0 & 0 \\ 0 & 0 & 1 & \theta''_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
&\quad \cdot \begin{pmatrix} \cos(d^i_\theta) & -\sin(d^i_\theta)\cos(d^i_\alpha) & \sin(d^i_\theta)\sin(d^i_\alpha) & d^i_a\cos(d^i_\theta) \\ \sin(d^i_\theta) & \cos(d^i_\theta)\cos(d^i_\alpha) & -\cos(d^i_\theta)\sin(d^i_\alpha) & d^i_a\sin(d^i_\theta) \\ 0 & \sin(d^i_\alpha) & \cos(d^i_\alpha) & d^i_d \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} \cos(\theta'_i)\cos(d^i_\theta) & -(\cos(\theta'_i)\sin(d^i_\theta) & (\cos(\theta'_i)\sin(d^i_\theta) & (\cos(\theta'_i)\cos(d^i_\theta) \\ -\sin(\theta'_i)\sin(d^i_\theta) & +\sin(\theta'_i)\cos(d^i_\theta))\cos(d^i_\alpha) & +\sin(\theta'_i)\cos(d^i_\theta))\sin(d^i_\alpha) & -\sin(\theta'_i)\sin(d^i_\theta))d^i_a \\ \sin(\theta'_i)\cos(d^i_\theta) & (-\sin(\theta'_i)\sin(d^i_\theta) & (\sin(\theta'_i)\sin(d^i_\theta) & (\sin(\theta'_i)\cos(d^i_\theta) \\ +\cos(\theta'_i)\sin(d^i_\theta) & +\cos(\theta'_i)\cos(d^i_\theta))\cos(d^i_\alpha) & -\cos(\theta'_i)\cos(d^i_\theta))\sin(d^i_\alpha) & +\cos(\theta'_i)\sin(d^i_\theta))d^i_a \\ 0 & \sin(d^i_\alpha) & \cos(d^i_\alpha) & d^i_d + \theta''_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} \cos(d^i_\theta + \theta'_i) & -\sin(d^i_\theta + \theta'_i)\cos(d^i_\alpha) & \sin(d^i_\theta + \theta'_i)\sin(d^i_\alpha) & d^i_a\cos(d^i_\theta + \theta'_i) \\ \sin(d^i_\theta + \theta'_i) & \cos(d^i_\theta + \theta'_i)\cos(d^i_\alpha) & -\cos(d^i_\theta + \theta'_i)\sin(d^i_\alpha) & d^i_a\sin(d^i_\theta + \theta'_i) \\ 0 & \sin(d^i_\alpha) & \cos(d^i_\alpha) & d^i_d + \theta''_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \\
&= D'^i(\theta_i)
\end{aligned}$$

Man beachte $Z^i(0) = E$.

Um zwischen dem Koordinatensystem des ersten Gelenks und dem Weltkoordinatensystem, also dem URDF-Koordinatensystem des 0. Segments, zu wechseln, verwenden wir die Transformation B. Die Transformation zwischen dem i. DH-Koordinatensystem und dem i. URDF-Koordinatensystem bezeichen wir mit K^i. Es gilt also

$$O^1 \cdot A^1(\theta) \cdots O^i \cdot A^i(\theta) = B \cdot Z^1(\theta) \cdot D^1 \cdots Z^i(\theta) \cdot D^i \cdot K^i$$

für $1 \leq i \leq n$ und

$$\begin{aligned}
T^{DH}(\theta) &= B \cdot Z^1(\theta) \cdot D^1 \cdots Z^n(\theta) \cdot D^n \cdot K^n \\
&= T^{URDF}(\theta)\,.
\end{aligned}$$

Eine Stellversatz von Gelenk i wird durch passende Wahl von d^i_α modelliert. Eine Getriebeübersetzung wird durch Multiplikation mit einer Konstanten erreicht. Einen Algorithmus zur Erstellung von DH-Parametern einer kinematischen Kette findet sich bei Spong et al. [SHV04, Kapitel 3.2.3], eine Implementation wird in Kapitel 4.8 und 6.11 beschrieben.

3.5 KDL-Kinematik

Die Kinematics and Dynamics Library (KDL) benutzt ihre eigene Parametrisierung der Kinematik, siehe hierfür auch [KDL12a], [KDL12b] und Kapitel 4.5. Einem Segment einer kinematischen Kette in KDL ist ein Gelenk zugeordnet, für das das Segment das Eltersegment ist. Ein Beispiel ist in Abbildung 3.4 zu sehen. Das Gelenk hat einen

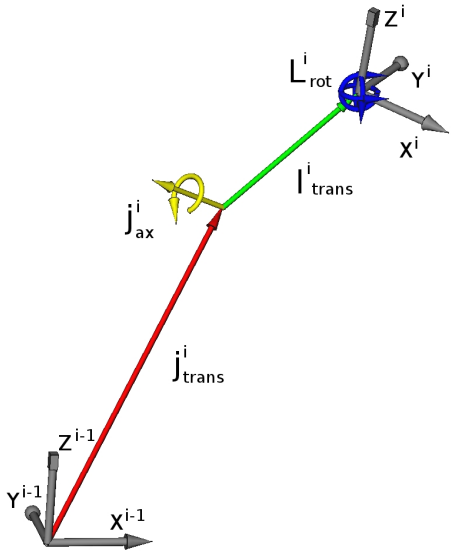


Abbildung 3.4: KDL Parameter eines Segments

Versatz j^i_{trans} innerhalb des Segments (rot) und eine dem Gelenktyp und der Achse j^i_{ax} (gelb) entsprechende Transformationsfunktion. Diese berücksichtigt auch eine Getriebeübersetzung und einen Stellversatz (j^i_{ax} repräsentiert die Achse in Nullstellung). Zusammen ergibt dies für Gelenk i die Transformationsfunktion $J^i(\theta) = J^i_{trans} \cdot J^i_{ax}(\theta)$. Daran schliesst sich eine feste Translation l^i_{trans} (grün) und eine Rotation L^i_{rot} (blau) an, welche in der Matrix $L^i = L^i_{trans} \cdot L^i_{rot}$ Ausdruck finden. Dies verbindet das Gelenk mit seinem Kindsegment.

Eine Kette hat die Gleichung

$$T_{KDL}(\theta) = J^1(\theta) \cdot L^1 \cdots J^n(\theta) \cdot L^n \,.$$

Auch hier gilt

$$T_{KDL}(\theta) = T_{URDF}(\theta) \,.$$

Wenn aus kinematischen Bäumen Ketten extrahiert werden, tritt der Fall auf, dass Teilketten invertiert werden müssen. In Abbildung 3.5 ist ein Beispiel für einen Baum zu sehen. Die Wurzel des Baumes ist die Basis. Der Torso ist wiederum die Wurzel

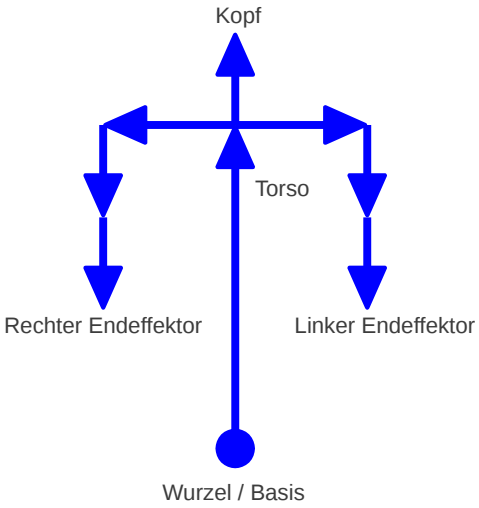


Abbildung 3.5: KDL Baum

für die Ketten zu den Endeffektoren und zum Kopf. Die Kette vom Kopf zum rechten Endeffektor besteht aus der invertierten Kette vom Torso zum Kopf und der Kette vom Torso zum rechten Endeffektor. Sei der Kopf nun Kettensegment 0, der Torso Segment w und der rechte Endeffektor Segment n. Dann gilt für die Transformation vom rechten Endeffektor zum Kopf

$$\begin{aligned} T_{KDL}(\theta) &= (J'^{w}(\theta) \cdot L'^{w} \cdots J'^{1}(\theta) \cdot L'^{1})^{-1} \cdot J^{w+1}(\theta) \cdot L^{w+1} \cdots J^{n}(\theta) \cdot L^{n} \\ &= (J'^{1}(\theta) \cdot L'^{1})^{-1} \cdots (J'^{w}(\theta) \cdot L'^{w})^{-1} \cdot J^{w+1}(\theta) \cdot L^{w+1} \cdots J^{n}(\theta) \cdot L^{n} \\ &= J^{1}(\theta) \cdot L^{1} \cdots J^{w}(\theta) \cdot L^{w} \cdot J^{w+1}(\theta) \cdot L^{w+1} \cdots J^{n}(\theta) \cdot L^{n}\,. \end{aligned}$$

Dabei ist

$$J^{i}(\theta) \cdot L^{i} = (L'^{i})^{-1} \cdot (J'^{i}(\theta))^{-1}$$

für $1 \leq i \leq w$. Aufgespalten in Einzeloperationen ergibt dies

$$\begin{aligned} J^{i}_{trans} \cdot J^{i}_{ax}(\theta) \cdot L^{i} &= (L'^{i}_{trans} \cdot L'^{i}_{rot})^{-1} \cdot (J'^{i}_{trans} \cdot J'^{i}_{ax}(\theta))^{-1} \\ &= (L'^{i}_{rot})^{-1} \cdot (L'^{i}_{trans})^{-1} \cdot (J'^{i}_{ax}(\theta))^{-1} \cdot (J'^{i}_{trans})^{-1} \\ &= \overbrace{(L'^{i}_{rot})^{-1} \cdot (L'^{i}_{trans})^{-1} \cdot L'^{i}_{rot}}^{J^{i}_{trans}} \\ &\quad \cdot \overbrace{(L'^{i}_{rot})^{-1} \cdot (J'^{i}_{ax}(\theta))^{-1} \cdot L'^{i}_{rot}}^{J^{i}_{ax}(\theta)} \cdot \overbrace{(L'^{i}_{rot})^{-1} \cdot (J'^{i}_{trans})^{-1}}^{L^{i}}\,. \end{aligned}$$

Also können

$$J^i_{trans} = ({L'}^i_{rot})^{-1} \cdot ({L'}^i_{trans})^{-1} \cdot {L'}^i_{rot}\,,$$

$$J^i_{ax}(\theta) = ({L'}^i_{rot})^{-1} \cdot ({J'}^i_{ax}(\theta))^{-1} \cdot {L'}^i_{rot}$$

und

$$L^i = ({J'}^i_{trans} \cdot {L'}^i_{rot})^{-1}$$

gesezt werden. Daraus folgen

$$j^i_{ax} = ({L'}^i_{rot})^{-1} \cdot -{j'}^i_{ax}$$

und

$$j^i_{trans} = ({L'}^i_{rot})^{-1} \cdot -{l'}^i_{trans}\,.$$

Vergleiche hierzu auch [KDL12b, chain.h].

3.6 Statistik

Es folgen einige Rechenregeln und Formeln zur Statistik, vergleiche hierzu etwa Thrun et al. [TBF06]. Sei X eine Zufallsvariable und x ein Wert, den diese annehmen kann, dann ist

$$p(x) = p(X = x)$$

die Wahrscheinlichkeit, dass X den Wert x annimmt. p ist hier eine Dichtefunktion. Weiter ist

$$p(x, y) = p(X = x \text{ und } Y = y)$$

die Wahrscheinlichkeit des gleichzeitigen Eintretens von $X = x$ und $Y = y$. Die bedingte Wahrscheinlichkeit $p(x|y)$ folgt den Rechenregeln

$$p(x|y) = \frac{p(x, y)}{p(y)}\,,$$

$$\begin{aligned} p(x|y) &= \frac{p(y|x)\,p(x)}{p(y)} \\ &= \frac{p(y|x)\,p(x)}{\int_{x' \in X} p(y|x')\,p(x')\,dx'} \end{aligned}$$

und

$$p(x|y, z) = \frac{p(y|x, z)\,p(x|z)}{p(y|z)}\,. \tag{3.1}$$

Als Dichtefunktion seien die Normalverteilung $\mathcal{N}(\mu, \sigma)$ mit

$$p(x) = \left(2\pi\sigma^2\right)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right)$$

und die mehrdimensionale Normalverteilung $\mathcal{N}(\mu, \Sigma)$ mit

$$p(x) = \frac{1}{(2\pi)^{\frac{D}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

genannt. Wobei σ die Varianz, Σ die $D \times D$ Kovarianzmatrix, $|\Sigma|$ deren Determinante und μ der Extrempunkt der Gaussverteilung sind.

3.7 MAP

Wir verwenden im folgenden die MAP, um von einer statistischen Population einen nicht beobachtbaren Parameter x anhand eines beobachtbaren Parameters y zu schätzen. Dabei gehen wir von einer a priori Verteilung $p(x)$ aus, siehe hierzu Bishop [Bis06]. Die Maximierung der Funktion $x \mapsto p(x|y)$ führt zu dem Schätzwert

$$\begin{aligned}\hat{x}_{\mathrm{MAP}}(y) &= \underset{x}{arg\,max}\ \ p(x|y) \\ &= \underset{x}{arg\,max}\ \ \frac{p(y|x)\,p(x)}{\int_{x' \in X} p(y|x')\,p(x')\,dx'} \\ &= \underset{x}{arg\,max}\ \ p(y|x)\,p(x)\,.\end{aligned}$$

Bei diesem Wert für x ist unter gegebenem y die Wahrscheinlichkeitsdichte maximal. Unter Verwendung von Gleichung 3.1 folgt die Gleichung

$$\hat{x}_{\mathrm{MAP}}(y, z) = \underset{x}{arg\,max}\ \ p(y|x, z)\,p(x|z)\,.$$

Diese wird im folgenden Verwendung finden.

3.8 Rprop

Wir verwenden Resilient Backpropagation, siehe Riedmiller [Rie94], zur Bestimmung der Lernschritte beim Gradientenverfahren. Sei F eine Funktion mit dem Parametervektor w und $\frac{\partial F}{\partial w_i}^t$ der Gradient in der t. Iteration, dann definieren wir den Vektor der Parameteränderungen Δw_i^t wie folgt:

$$\Delta w_i^t = \begin{cases} -\Delta_i^t & \text{für } \frac{\partial F}{\partial w_i}^t > 0, \\ \Delta_i^t & \text{für } \frac{\partial F}{\partial w_i}^t < 0 \text{ und} \\ 0 & \text{sonst.} \end{cases}$$

Dabei gilt für die Lernrate

$$\Delta_i^t = \begin{cases} \min\{\Delta_{\max}, \eta^+ * \Delta_i^{t-1}\} & \text{für } \frac{\partial F}{\partial w_i}^{t-1} * \frac{\partial F}{\partial w_i}^{t} > 0 \\ \max\{\Delta_{\min}, \eta^- * \Delta_i^{t-1}\} & \text{für } \frac{\partial F}{\partial w_i}^{t-1} * \frac{\partial F}{\partial w_i}^{t} < 0 \\ \Delta_i^{t-1} & \text{sonst,} \end{cases}$$

mit $0 < \eta^- < 1 < \eta^+$. Bei einem Vorzeichenwechsel ($\frac{\partial F}{\partial w_i}^{t-1} * \frac{\partial F}{\partial w_i}^{t} < 0$) findet im aktuellen Schritt t keine Anpassung der Parameter und im folgenden Schritt $(t+1)$ keine Anpassung der Lernrate statt. Entsprechend den Empfehlungen aus der Literatur wird $\eta^- = 0.5$, $\eta^+ = 1.2$, $\Delta_{\min} = 10^{-6}$ und $\Delta_{\max} = 50$ gewählt.

3.9 Rotationen

Die KDL [KDL12a], bietet Methoden zur Umwandlung zwischen Rotationsmatrizen in Quaternionen und der Darstellungen als Roll-Nick-Gier-Winkel. Der KDL-Quelltext dieser Methoden aus der Klasse *Rotation* [KDL12b, frames.hpp] wurde als Vorlage für die Berechnungen in der vorliegenden Arbeit genommen.

4 Systembeschreibung

In diesem Kapitel wird das technische Umfeld beschrieben. Wir beginnen mit der Beschreibung der Hardware und gehen dann auf die allgemeine Softwareumgebung ein. Im Anschluss werden spezifische Komponenten beschrieben, welche für diese Arbeit eine herausragende Rolle spielen.

4.1 Cosero und Dynamaid

Abbildung 4.1: Dynamaid

Dynamaid und Cosero sind die beiden Roboter des NimbRo@Home Projekts am Institut für Autonome Intelligente Systeme der Universität Bonn [Nim12]. Sie haben anthropomorphe Oberkörper und mobile Basen, siehe Abbildung 4.1 und 4.2. Im Folgenden werden die Grundzüge des aktuellen Aufbaus der Roboter beschrieben, vergleiche hierzu auch Stückler et al. [SSB09]. Der Entwurf der Roboter hat das Ziel, dass sich diese möglichst gut in den Alltag von Menschen einfügen können. So wurde auf geringes Gewicht, freundliches Aussehen und hohe Mobilität Wert gelegt. Die auf Aluminiumprofilen basierende Leichtbauweise ermöglicht es einer einzelnen Person die Roboter zu tragen. Zudem werden keine starken Aktuatoren benötigt, was zum sicheren Betrieb beiträgt. Der Oberkörper der Roboter wirkt menschlich, was durch eine weiße Gesichtsmaske für den Kopf noch unterstützt wird. Die Basis hingegen besteht aus einem quaderförmigen

Abbildung 4.2: Cosero

Chassis mit circa 600 x 400 mm Grundfläche. Sie beinhaltet die zwei Mikrocontroller und die austauschbaren Lithium-Polymer-Akkus.

An den Ecken der Basis sind vier omnidirektionale Differentialantriebe befestigt. Jedes der acht Räder kann einzeln gesteuert werden und jedes Radpaar kann für jede Fahrtrichtung gedreht werden. Da die Gieraktuatoren nicht voll rotieren können, müssen die Roboter anhalten und die Radpaare um 180° drehen, wenn deren Anschlag erreicht ist. Auf die Basis ist ein senkrechter Pfosten montiert an dem sich der Aufzug für den Torso befindet. Der Torso ist über ein Giergelenk am Aufzug befestigt. Am Torso ist wiederum der Kopf mit einem Gier- und einem Nickgelenk angebracht. An den beiden Seiten des Torso befinden sich anthropomorphe Arme mit Roll-, Nick- und Giergelenken in der Schulter und einem Nickgegelenk im Ellenbogen. Diese werden durch Dynamixel EX-106 Aktuatoren [Dyn12] mit einer Auflösung von jeweils 251°/4096 angetrieben. Im Handgelenk befindet sich ein Dynamixel RX-28 im Rollgelenk und Dynamixel RX-64 in den Nick- und Giergelenken, jeweils mit einer Auflösung von 300°/1024. Die Hände bestehen aus zwei einzeln steuerbaren Fingern. Insgesamt ist die Anatomie den Abmessungen, der Beweglichkeit und Reichweite eines Menschen nachempfunden. Die maximale Nutzlast der Greifer beträgt bei Dynamaid circa ein Kilogramm und bei Cosero das Doppelte. Für die Finger existieren zwei Konfigurationen, eine mit großflächigen, gepolsterten Greifflächen, siehe Abbildung 4.2, die andere bestehend aus adaptiven Fin-Ray-Elementen der Firma Festo [Fes12]. Am Torso ist je ein roll- und ein nickbarer Hokuyo-Laserscanner

angebracht. Die Steuerung der Aktuatoren wird mit hohen Takten und kurzen Latenzen durch einen der Mikrocontroller übernommen. An diesen werden auch die Aktuatorzustände zurückgeliefert.

Die Basis trägt einen SICK Laserscanner mit ungefähr 30 m Reichweite und 270° Öffnungswinkel nach vorne in horizontaler Ausrichtung in einer Höhe von circa 250 mm. An der Vorderseite, unter dem Chassis ist ein Hokuyo-Laserscanner angebracht, der in etwa 25 mm Höhe horizontal den Nahbereich vor dem Roboter und zwischen den Rädern hindurch den Bereich hinter dem Roboter abdeckt. Die beiden Laserscanner im Torso sind ebenfalls für den vorderen Nahbereich ausgelegt. Der Nahbereich ersteckt sich über eine Distanz von ungefähr zwei Metern. Wegen des hohen Datenaufkommens ist für die Kommunikation mit den Laserscannern ein zweiter Mikrocontroller zuständig. Im Kopf befindet sich ein Richtmikrofon und momentan deaktivierte Stereokameras. Daneben ist eine Kinect-Kamera [Kin12] am Kopf befestigt, diese hat eine Bildauflösung von 640×480 Punkten.

Sowohl die Kinect als auch die Mikrocontroller kommunizieren über USB mit einem leistungsfähigen Laptop, welcher auf der Basis Platz findet. Dieser ist mit einem i7-Prozessor und mehreren Gigabyte RAM ausgestattet und ermöglicht die Verarbeitung der anfallenden Daten. Auf diesem Rechner läuft die im folgenden beschriebene Software während des Betriebs des Roboters.

4.2 ROS

Die Roboter werden mit dem Robot Operating System [ROS12a] des Instituts Willow Garage [Wil12] betrieben. Dieses stark modularisierte, quelloffene Framework für den Betrieb von Robotern ist einfach zu erweitern. ROS erfüllt mehrere Aufgaben, es bietet Gerätesteuerungen und Hardwareabstraktionen, Softwarebibliotheken und Applikationen für häufig wiederkehrende Aufgaben, Interprozesskommunikation über Netzwerke mit standardisierten Datenformaten und Verwaltungswerkzeuge, um Quelltext und Binärdateien zu verteilen, verwalten, übersetzen und darin zu navigieren.

Die Softwarekomponenten sind in Paketen organisiert, welche wiederum zu Aufgabenfeldern, den Stacks, zusammengefasst werden. Diese werden von verschiedenen Anbietern bereitgestellt. Programme laufen als verteilte, miteinander kommunizierende Knoten unter einer zentralen Kontrolle, welche für die Koordination der Datenströme zuständig ist.

Es gibt zwei Kommunikationsformen in ROS, Topics und Services. Topics sind unidirektionale Broadcastnachrichtenkanäle und Services sind Dienste mit blockierender bidirektionaler Kommunikation. Hinzu kommen ROS-Actions, welche keine eigene Kommunikationsart darstellen. Es handelt sich bei ihnen um eine Protokollkonvention basierend auf Topics, mit der asynchrone Serviceaufrufe umgesetzt werden können. In Abbildung 4.3 sieht man eine Übersicht der Topics während einer simulieten Exploration.

Jede Kante repräsentiert eine Sender-Empfänger-Beziehung zwischen den Knoten. Ein Topic kann in dieser Darstellung also durch mehrere Kanten dargestellt werden, oder durch keine, wenn es keinen Sender oder Empfänger hat.

Die ROS ermöglicht es in mehreren Sprachen, unter anderem C++, Python und Java,

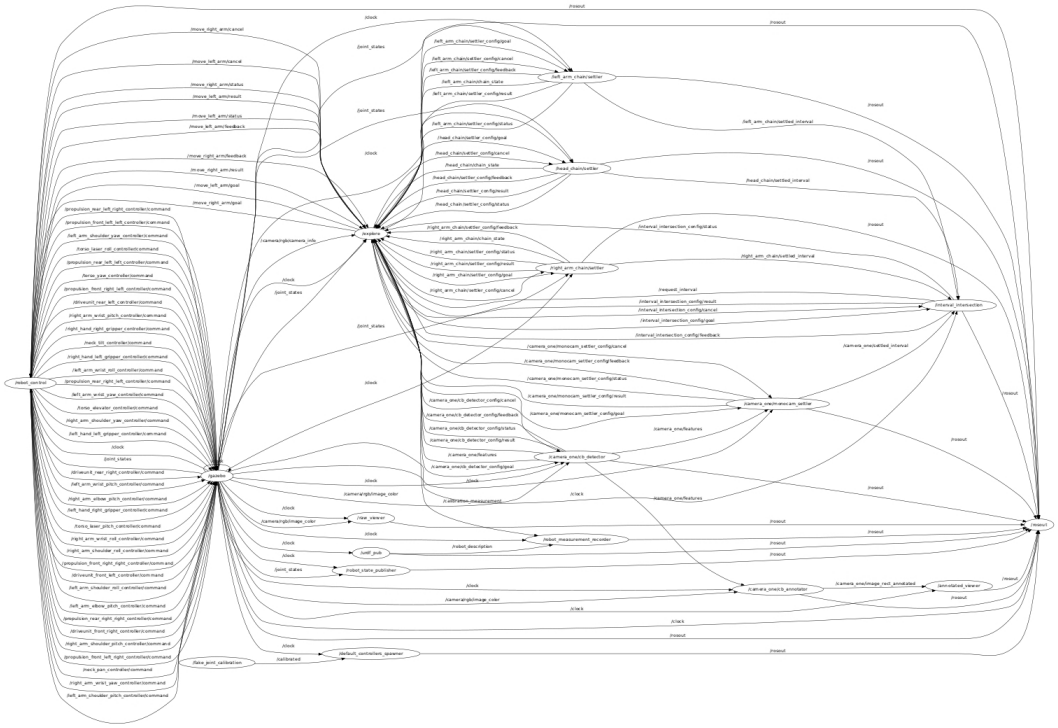


Abbildung 4.3: ROS-Topics

leicht Kommunikationsadapter für die Interprozesskommunikation zu erstellen und zu benutzen. In Kombination mit der einfachen Möglichkeit die Richtung von Datenströmen zu manipulieren erlaubt dies, entsprechend dem Toolbox-Prinzip von Unix, Kommunikationsknoten zu schreiben welche nur sehr eingeschränkte Aufgaben übernehmen aber gut kombiniert werden können. Nachrichten beinhalten immer einen Zeitstempel und ein Referenzkoordinatensystem. Auf diese Weise ist zum Beispiel gewährleistet, dass einem Kamerabild Aufnahmezeitpunkt und Blickrichtung zugeordnet werden können.

Die über Topics publizierten Nachrichten können mit dem Programm *Rosbag* aufgezeichnet und wieder abgespielt werden. Dies ermöglicht wiederholtes Betrachten und Bearbeiten von einmal aufgenommenen Daten und kann zur Fehlersuche benutzt werden.

Mit *Roslaunch* und den dafür erstellten Konfigurationsdateien, lassen sich Gruppen von Knoten koordiniert initialisieren und starten. So können Kommunikationskanäle und Parameter zueinander passend eingestellt werden.

Konfigurationseinstellungen für Knoten können auf einem zentralen Parameterserver abgelegt werden. Für die Adressierung dieser Daten und bei der Kommunikation stehen Mechanismen zur Namensraumverwaltung zur Verfügung, welche das Gruppieren, mehrfaches Instanziieren und hierarchisches Anordnen von Knoten erlauben. Es sei an dieser Stelle auf die ausführliche ROS-Dokumentation verwiesen [ROS12a].

Im Folgenden werden die für die Arbeit relevanten ROS Komponenten beschrieben. Das ROS-Paket *tf* [tf12] ist die zentrale Komponente im Umgang mit Referenzkoordi-

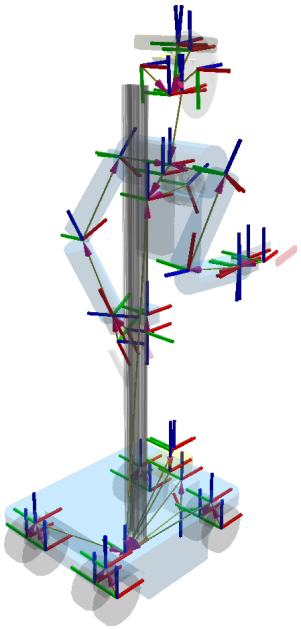


Abbildung 4.4: TFs von Cosero

natensystemen, siehe Abbildung 4.4. Es ermöglicht Transformationen von Koordinaten zwischen zeitlich veränderlichen Referenzkoordinatensystemen. Jeder Knoten, welcher Informationen über Koordinatensysteme benötigt oder bereitstellt, tauscht diese mit dieser Komponente aus.

Der *robot_state_publisher* berechnet mit Hilfe von URDF-Beschreibungen (siehe Kapitel 4.4) und Gelenkstellungen, welche auch vom NimbRo-Robot-Control (siehe Kapitel 4.3) publiziert werden, die Referenzkoordinatensysteme der Segmente und kommuniziert diese an die tf-Komponente.

ROS verfügt mit rviz [rvi12] über ein mächtiges Visualisierungswerkzeug. In ihm können unter anderem Koordinatensysteme, kinematische Modelle, virtuelle Robotermodelle mit Kollisionseigenschaften, Kamerabilder mit Einblendungen von virtuellen Elementen und Punktwolken dargestellt werden. In Abbildung 4.5 sieht man das Modell Coseros mit eingeblendeter Kinematik mit Koordinatensystemen und Kamerabild. In Abbildung 1.2 ist ein Kamerabildzu sehen, in dem mit Rviz Koordinatensysteme und das virtuelle Robotermodell eingeblendet wurden.

Mit dem 3D-Physiksimulator Gazebo [gaz12] besteht die Möglichkeit die Kinematik und Dynamik eines Roboters und seiner Umgebung nachzuahmen. In Abbildung 4.6 sieht man die gleiche Szene wie in Abbildung 4.5. Der Simulator kann Kamerabilder, Laserscanns und Punktwolken von Tiefensensorenkameras generieren.

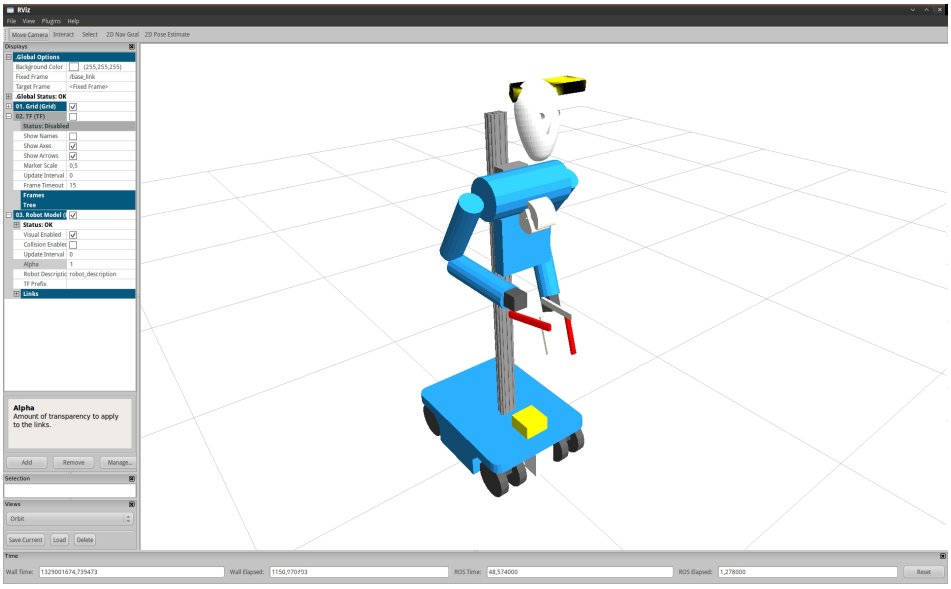


Abbildung 4.5: Rviz

4.3 NimbRo-Robot-Control

Das NimbRo-Robot-Control wurde ursprünglich nicht für ROS entwickelt und übernimmt umfangreiche Aufgaben bei dem Betrieb der Roboter. Zu den Funktionen gehört die Kommunikation mit den Mikrocontrollern, die VK und die IK der Roboter, das Ausführen von Bewegungsprimitiven, das Publizieren der Gelenkstellungen, die Bereitstellung von Referenzkoordinatensystemen und das Abarbeiten von kinematischen Trajektorien.

Für die Bewegung des Kopfes nimmt das Robot-Control unter dem Servicenamen */robot_control/bodyparts_command_service* Aufrufe vom Typ *nimbro_msgs.srv.BodyPartsCommandRequest* entgegen.

Um die Arme zu steuern existieren die ROS-Aktionen *move_left_arm* und *move_right_arm*, welche Nachrichten des Typs *control_msgs.msg.FollowJointTrajectoryAction* empfangen.

Die aktuellen Stellungen der Gelenke werden auf dem Topic */robot_control/joint_states* als *sensor_msgs.msg.JointState* Nachrichten publiziert.

4.4 URDF

Das Unified Robot Description Format [urd12] ist ein XML-Dialkekt, in dem in ROS Robotermodelle beschrieben werden. Neben den Grenzen für die Gelenkstellungen werden mit ihr alle statischen Parameter des kinematischen Baums dargestellt. Dies schließt Kollisionseigenschaften und die Visualisierung im Simulator und in rviz ein. Die URDF-Beschreibung für den in Abbildung 4.7 gezeigten Roboterarm mit einem Rotationsgelenk,

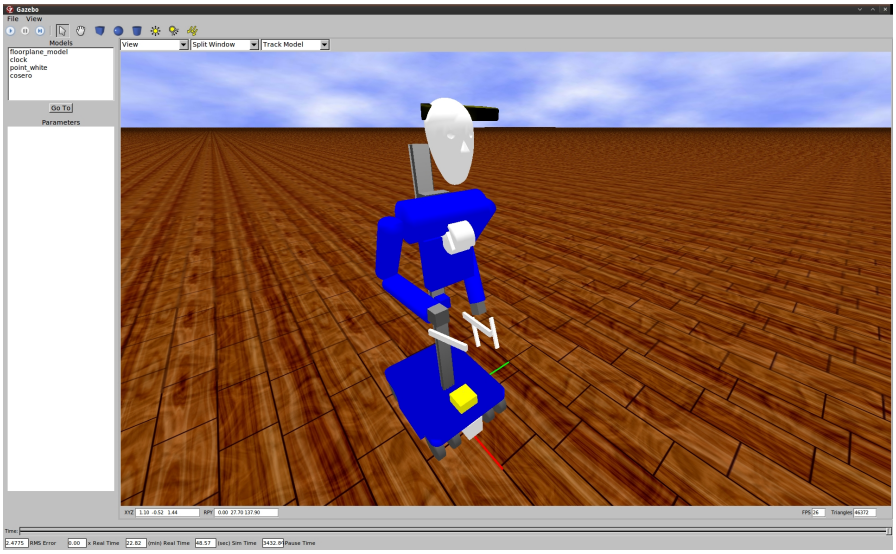


Abbildung 4.6: Gazebo

einem prismatischen und einem starren Gelenk findet sich in Listing 4.1.

Das XML-Element *robot* beinhaltet die Beschreibung des kinematischen Baumes eines Roboters. Dieser besteht aus *links*, welche die Segmente repräsentieren und aus *joints*, welche die Gelenke darstellen. Elter- und Kindbeziehung werden darin durch *parent* und *child* Einträge ausgedrückt. Es gibt immer nur ein Element, die Wurzel, welches keinen Vorfahren hat. Segmente haben neben ihrem Namen noch weitere Eigenschaften, wie etwa Trägheit, Masse, Kollisionskörper und visuelle Darstellung, diese spielen aber für diese Arbeit keine Rolle. Gelenke haben die Eigenschaft *type*, dabei spielen bei dieser Arbeit nur fixe (*fixed*) und rotierende (*continuous* und *revolute*) Gelenke eine Rolle, wobei die letzteren gleich behandelt werden. Das Element *limit* beschreibt die Grenzen der Beweglichkeit und *safety_controller* das gewünschte Verhalten der Gelenksteuerung in deren Nähe. Die Einträge für *origin* und *axis* werden in Kapitel 3.3 genauer betrachtet. Das Element *transmission* beschreibt die Eigenschaften des Antriebs des Gelenks. Es hat das Unterelement *mechanicalReduction*, in ihm wird das Getriebeverhältnis des jeweiligen Aktuators angegeben. Die Beschreibung wird zur Laufzeit auf einen zentralen Parameter-Server geladen. Von dort wird sie von den betreffenden Knoten gelesen.

Die verwendeten URDF-Modelle der Roboter Cosero und Dynamaid sind von der AG Behnke [Nim12] erstellt worden. Das Modell des PR2 stammt von [Wil12]. In Abbildung 4.8 sehen wir einen Plot des URDF-Modells des Roboters Cosero. Man sieht die verschiedenen Äste für Basis, Torso, Arme und Kopf.

4.5 KDL

Die Kinematics and Dynamics Library, siehe [KDL12a] und [KDL12b], im ROS-Paket *kdl* ist die zentrale Bibliothek bei der Berechnung von VK und IK von Position und

Listing 4.1: URDF

```
<?xml version="1.0"?>
<robot name="Arm">
  <link name="Seg0" />
  <link name="Seg1" />
  <joint name="Jnt1" type="revolute">
    <parent link="Seg0" />
    <child link="Seg1" />
    <origin xyz="1 1 0" rpy="0 0 0" />
    <axis xyz="0 0 1" />
    <limit effort="1000" lower="-0.6" upper="0.6" velocity="1" />
    <safety_controller k_position="20" k_velocity="4" soft_lower_limit
       ="-.5" soft_upper_limit=".5" />
  </joint>
  <link name="Seg2" />
  <joint name="Jnt2" type="prismatic">
    <parent link="Seg1" />
    <child link="Seg2" />
    <origin xyz="2 0 0" rpy="0 0 0.785" />
    <axis xyz="1 0 0" />
    <limit effort="1000" lower="-0.7" upper="0.2" velocity="0.5" />
    <safety_controller k_position="10" k_velocity="10"
       soft_lower_limit="-0.6" soft_upper_limit="0.1" />
  </joint>
  <link name="Seg3" />
  <joint name="Jnt3" type="fixed">
    <parent link="Seg2" />
    <child link="Seg3" />
    <origin xyz="1.4 0 0" rpy="0 0 0.785" />
  </joint>
  <transmission type="SimpleTransmission" name="Jnt1_trans">
    <actuator name="Jnt1_act" />
    <joint name="Jnt1" />
    <mechanicalReduction>1.1</mechanicalReduction>
  </transmission>
</robot>
```

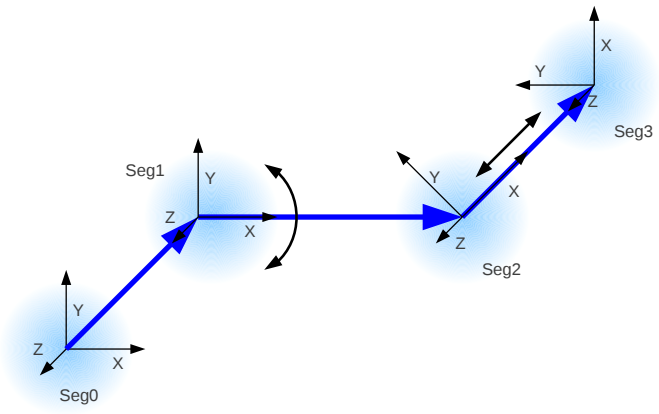


Abbildung 4.7: URDF-Kette

Geschwindigkeit. Zudem werden in ihr Berechnungen zur Dynamik vorgenommen. Kinematische Bäume, Ketten, Segmente, Gelenke und Koordinatensysteme werden durch die Klassen *Tree*, *Chain*, *Segment*, *Joint* und *Frame* modelliert. Mit der Vorgabe eines Start- und eines Endsegments kann aus einem Baum die passende Kette generiert werden.

Die Berechnung der Kinematik übernehmen Solverklassen. Die VK der Position einer Kette kann mit Hilfe der Klasse *ChainFkSolverPos_recursive* berechnet werden. Die IK der Position kann von *ChainIkSolverPos_NR* bestimmt werden, oder von *ChainIkSolverPos_NR_JL*, sofern Gelenklimits beachtet werden sollen. Die letzten beiden Klassen benötigen für ihre Arbeit die Hilfe der ersteren und von einer weiteren welche die IK der Geschwindigkeit löst, dies kann etwa *ChainIkSolverVel_pinv* sein. Eine Beschreibung der Positionskinematik findet sich in Kapitel 3.5.

Die KDL steht als Quelltext zur Verfügung und wird bei der Benutzung als dynamische Bibliothek eingebunden. Zudem besteht die Möglichkeit die KDL mit Hilfe des Wrapperwerkzeugs *SIP* [SIP12] in Python zu verwenden.

ROS bietet mit dem Paket *kdl_parser* [kdl12c] die Möglichkeit URDF einzulesen und direkt in einen KDL-Baum zu wandeln. Der umgekehrte Weg steht hier zur Verfügung.

4.6 OpenCV

OpenCV ist eine Programmbibliothek zur Echtzeitbildverarbeitung und -darstellung [Ope12a]. Sie ermöglicht die Erkennung von Schachbrettmustern und die Bestimmung deren Pose mit Hilfe von monoskopischen Kamerabildern und Kamerakalibrierungsinformationen [Ope12b].

Die Funktion *FindChessboardCorners* bestimmt die Position von Kreuzungspunkten

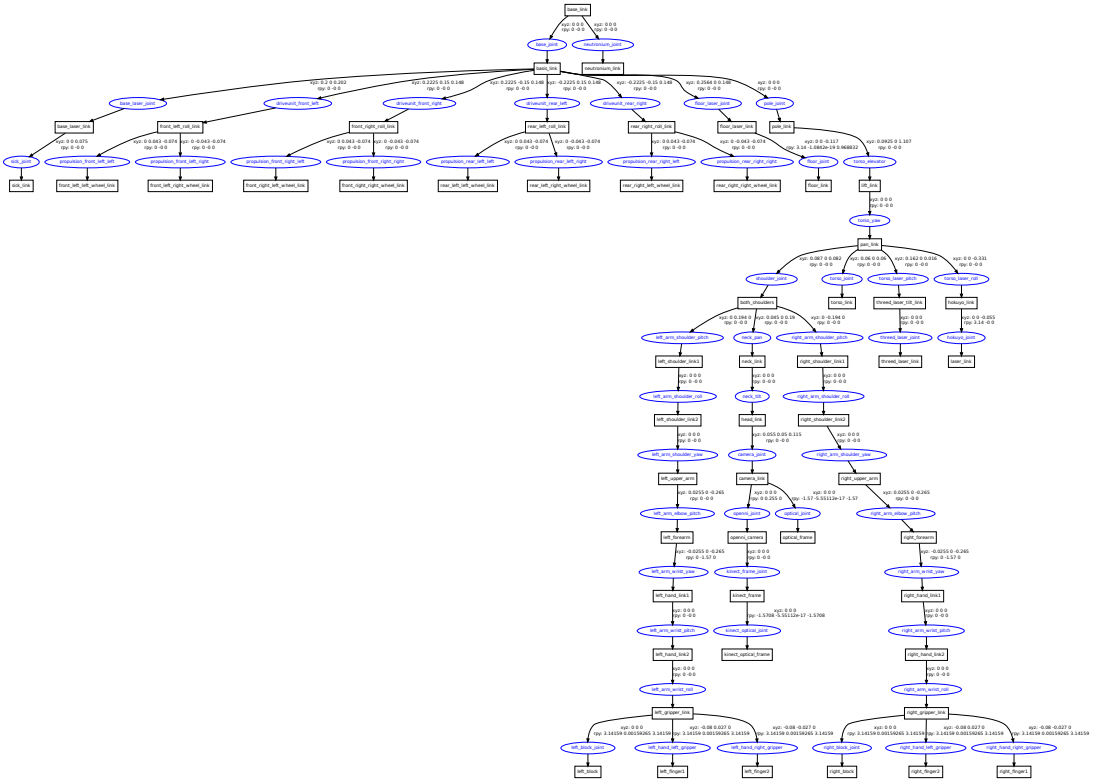


Abbildung 4.8: URDF-Baum von Cosero

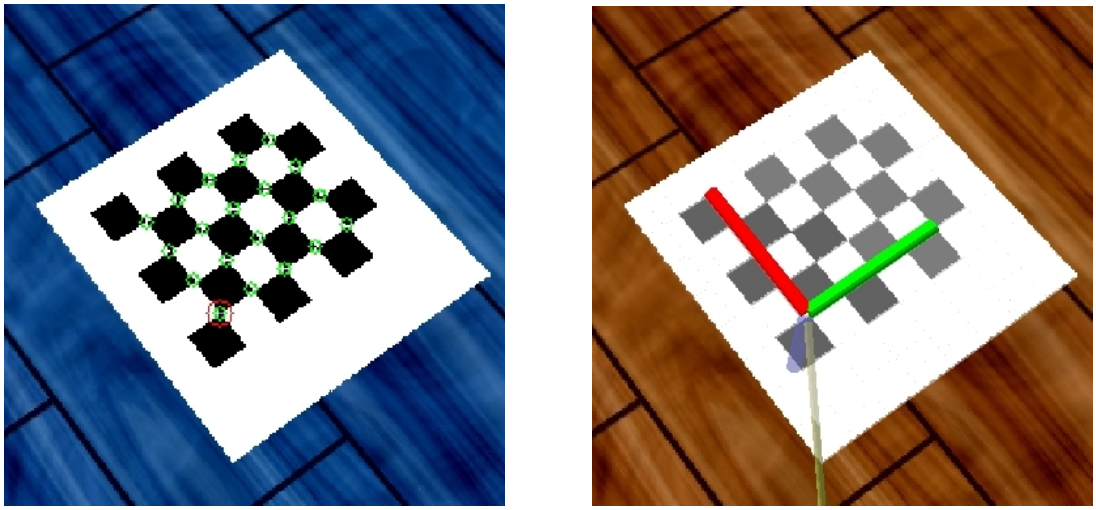
von Schachbrettfeldern in Kamerabildern und *FindCornerSubPix* erweitert die Genauigkeit auf Subpixel. Mit *FindExtrinsicCameraParams2* kann dann die Pose eines Objekts aus von ihm bekannten 3D Punkten, den Kreuzungspunkten, und im Bild gefundenen 2D Punkten, den Kreuzungspunkten, approximiert werden. Hier fließt auch die Kamerakalibrierung mit ein, welche auch mit OpenCV erstellt werden kann. Ein Beispiel hierfür ist in den Abbildungen 4.9 zu sehen.

Mit dem ROS-Paket camera_calibration [cam12a] kann die Kalibrierung einer Kamera vorgenommen werden. Hier werden OpenCV und Schachbrettmuster verwendet, um die Eigenschaften der Kamera zu erfassen. Diese Funktionalität steht auch als Pythonbibliothek zur verfügung.

OpenCV steht als Quelltext zur Verfügung und wird zur Benutzung als dynamische Bibliothek eingebunden. Es besteht die Möglichkeit OpenCV in Python zu verwenden.

4.7 pr2_calibration

Der Stack pr2_calibration [pr212b] beinhaltet die Pakete für die Kalibrierung des Roboters PR2 [pr212a]. Die Kinematik wird hier mit Daten von Kameras und Laserscannern gelernt und in das URDF-Modell des Roboters eingetragen. Der Versuch, die Software im



(a) Kreuzungspunkte (b) Pose

Abbildung 4.9: OpenCV Schachbretterkennung

Simulator für Vergleichszwecke zu betreiben, ließ sich im verfügbaren Zeitrahmen nicht umsetzen. Dennoch können einige Pakete des Stacks für die Datenaufnahme verwendet werden oder dienen als Vorlage für die Datenaufnahme der vorliegenden Arbeit.

Das Paket *image_cb_detector* beinhaltet einen Knoten, welcher Schachbrettmuster in Bildtopics erkennt und die erkannten Kreuzungspunkte als Topic publiziert. Ein Knoten aus dem Paket *laser_cb_detector* wird verwendet, um in Bilddaten diese Punkte zu markieren und die annotierten Bilder zur Veranschaulichung zu publizieren. Das Paket *monocam_settler* enthält ein Programm, welches die Zeitintervalle publiziert, in denen sich die Positionen der erkannten Punkte nicht außerhalb vorgegebener Abweichungen ändert. Im Pakete *joint_states_settler* residiert ein Knoten, der selbiges für Gelenkstellungen übernimmt. Im *interval_intersection* Paket gibt es einen Knoten, der auf verschiedene Settlertopics hört und deren Schnittmenge publiziert. Diese Pakete dienen der Erkennung stabiler Konfigurationen des Roboters, um so mit Messungen arbeiten zu können, welche nicht verwackelt sind.

4.8 DH-Converter von Jörg Stückler

Dieses Programm berechnet Denavit-Hartenberg-Parameter einer gegebenen Kinematik. Es implementiert den von Sponge et al. in [SHV04, Kapitel 3.2.3] beschriebenen Algorithmus. Die Eingabe ist eine Liste von Aufpunkten und Achsvektoren der Gelenke der kinematischen Kette in Nullstellung im Weltkoordinatensystem. In Abbildung 4.10 sind die Ortsvektoren u^i der Aufpunkte rot und die Achsvektoren a^i blau zu sehen.

Die Ausgabe ist eine Transformationsmatrix vom ersten DH-Segment in das Weltkoordinatensystem und eine Liste von DH-Parametern. Für Berechnungen wird die GNU

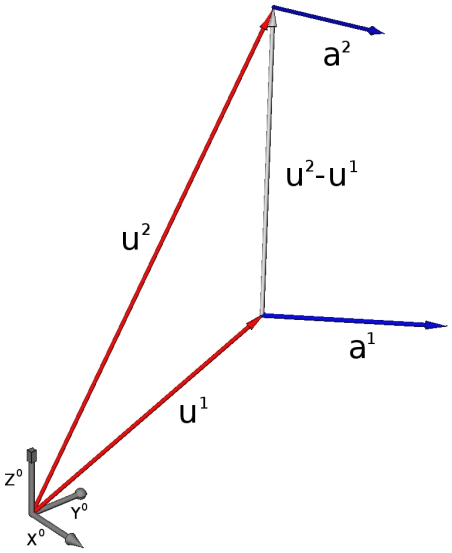


Abbildung 4.10: DH Konverter

Scientific Library [Gnu12] verwendet. Der Quelltext für das Programm wurde von Jörg Stückler am Institut entwickelt und für die vorliegende Arbeit zur Verfügung gestellt. Für die weitere Verwendung musste dieser angepasst werden. Siehe hierzu Kapitel 6.11.

5 Ansatz

In diesem Kapitel wird der verwendete Ansatz beschrieben. Zuerst wird der generelle Ablauf bei der Erstellung einer Kalibrierung dargestellt, danach wird auf die einzelnen Schritte eingegangen. Im Anschluss werden Variationen des Verfahrens vorgestellt.

5.1 Erstellung einer Kalibrierung

Wir kennen ein mehr oder weniger genaues Modell der Kinematik des Roboters im Voraus. Dies erlaubt es uns, den Roboter so zu bewegen, dass die Endeffektoren der Arme mit den eingebauten Kameras erfasst werden können. Zudem können wir mit Hilfe des Modells eine a priori Verteilung für die zu schätzenden kinematischen Parameter angeben. Diese basiert auf dem vorgegebenen Modell, seinen geschätzten Varianzen und der Annahme, dass es sich um eine Normalverteilung handelt.

Bei einer Selbstexploration fährt der Roboter mit dem zu kalibrierenden Arm eine Menge von Stellungen an. Dabei befindet sich in seinem Greifer eine Halterung mit einem Schachbrett, deren Abmessungen bekannt sind. Währenddessen beobachtet der Roboter mit einer oder mehreren Kameras das Schachbrett und bestimmt die Kreuzungspunkte des Schachbretts im Kamerabild. Die Menge an Stellungen deckt den Konfigurationsraum möglichst gleichmäßig ab und beinhaltet Variationen im Nullraum der Kinematik des Endeffektors. Dies sind sich unterscheidende Armstellungen mit der gleichen Endeffektorpose. Bei der Exploration nimmt der Roboter neben den Schachbrett-Posen auch die gemessenen Gelenkstellungen auf.

Mit den Kreuzungspunkten des Schachbretts in den Kamerabildern werden dann die Kameras kalibriert, sofern dies nicht schon vorher geschehen ist, siehe Kapitel 4.6. Danach werden die Schachbrettposen in den Kamerakoordinatensystemen bestimmt.

Aus dem in URDF vorgegebenen Baummodell erstellen wir die URDF-Kette von der Kamera zum Referenzkoordinatensystem des Schachbrettmusters, siehe Kapitel 5.8 und 5.10. Daraus wird dann die DH-Kette gewonnen, siehe Kapitel 4.8. Eine Gegenüberstellung von DH und URDF-Ketten befindet sich in Kapitel 5.5.

Die DH-Kette liefert dann die Mittelwerte der a priori Verteilung für MAP, wobei die Schachbrettposen aus der Exploration verwendet werden, siehe Kapitel 5.2. Die Ausgabe des MAP-Schrittes sind nicht die gesuchten Maxima, sondern Gradienten. Mit diesen führen wir dann ein Gradientenverfahren durch, um die gesuchten Werte zu finden, siehe Kapitel 5.4. Daraus resultiert eine a posteriori Verteilung, deren Mittelwerte wir als Parameter einer neuen DH-Kette verwenden.

Dann vergleichen wir die a priori und die a posteriori DH-Kette miteinander und manipulieren die a priori URDF-Kette so, dass sie diesen Änderungen entspricht, siehe

Kapitel 5.7. Die dabei entstehende Kette ist die a posteriori URDF-Kette. Ein einfacheres Verfahren zur Wandlung von DH-Ketten nach URDF zeigen wir zur Veranschaulichung in Kapitel 5.6. Mit der aposteriori URDF-Kette modifizieren wir dann den UDRF-Baum, siehe Kapitel 5.9. Um den letzten Punkt besser zu veranschaulichen, zeigen wir in Kapitel 5.8 wie URDF-Ketten aus einem Baum gewonnen werden.

5.2 MAP

Wir verwenden die Maximum-a-posteriori-Methode aus Kapitel 3.7, um die unbekannten kinematischen Parameter zu schätzen, es gilt

$$\hat{m}_{\text{MAP}}(z,x) = \underset{m}{arg\,max}\;\; p(z|m,x)\, p(m|x)\,.$$

Dabei ist m der Vektor der zu schätzenden kinematischen Parameter, z ist der Vektor der gemessenen Schachbrettpose und x sind die gemessenen Gelenkstellungen. Es gilt $p(m|x) = p(m)$, da m von x unabhängig ist.

Wir nehmen an, dass m normalverteilt ist, $m \propto \mathcal{N}(\mu_m, \Sigma_m)$, also die Dichtefunktion

$$p(m) = \eta\, exp(-\tfrac{1}{2}(m-\mu_m)^T {\Sigma_m}^{-1}(m-\mu_m))$$

hat. η stellt einen Normierungsfaktor wie bei Thrun et al. [TBF06] dar, er nimmt an den Stellen, an denen er auftritt den Wert an, der das Integral der Funktion auf eins normiert. Für die Schachbrettposen machen wir die Annahme

$$\begin{aligned} p(z|m,x) &\approx f(m,x) + \epsilon \\ &= \eta\, exp(-\tfrac{1}{2}(z-f(m,x))^T {\Sigma_z}^{-1}(z-f(m,x))\,, \end{aligned}$$

wobei $\epsilon \propto \mathcal{N}(0, \Sigma_z)$ gilt. Die Funktion $f(m,x)$ repräsentiert dabei die VK, wobei x der Vektor der bekannten kinematischen Parameter ist. Es folgt

$$\hat{m}_{\text{MAP}}(z,x) \approx \underset{m}{arg\,max}\;\; \eta\, exp(-L)$$

mit

$$L = \tfrac{1}{2}(m-\mu_m)^T {\Sigma_m}^{-1}(m-\mu_m) + \tfrac{1}{2}(z-f(m,x))^T {\Sigma_z}^{-1}(z-f(m,x))\,.$$

Die erste und zweite Ableitung von L sind

$$\frac{\partial L}{\partial m} = {\Sigma_m}^{-1}(m-\mu_m) - (\nabla_m f(m,x))^T\, \Sigma_z^{-1}(z-f(m,x))$$

und

$$\frac{\partial^2 L}{\partial m^2} = {\Sigma_m}^{-1} - \left(\nabla_m^2 f(m,x)\right)^T \Sigma_z^{-1}(z-f(m,x) + (\nabla_m f(m,x))^T\, \Sigma_z^{-1} \nabla_m f(m,x))\,.$$

Hierbei wurde ausgenutzt, dass $\Sigma = \Sigma^T$ ist und somit

$$A = \tfrac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)$$

die Ableitung

$$\begin{aligned}
\frac{\partial A}{\partial x} &= \frac{1}{2}\left(\left(\frac{\partial(x-\mu)}{\partial x}\right)^T \Sigma^{-1}(x-\mu) + (x-\mu)^T\Sigma^{-1}\frac{\partial(x-\mu)}{\partial x}\right) \\
&= \frac{1}{2}\left(E^T\Sigma^{-1}(x-\mu) + (x-\mu)^T\Sigma^{-1}E\right) \\
&= \frac{1}{2}\left(\Sigma^{-1}(x-\mu) + (x-\mu)^T\Sigma^{-1}\right) \\
&= \frac{1}{2}\left(\Sigma^{-1}(x-\mu) + \left(\Sigma^{-1}\right)^T(x-\mu)\right) \\
&= \frac{1}{2}\left(\Sigma^{-1}(x-\mu) + \left(\Sigma^{-1}\right)(x-\mu)\right) \\
&= \Sigma^{-1}(x-\mu)
\end{aligned}$$

hat. Mit den Ableitungen können wir nun Gradientenverfahren durchführen, um die gesuchten Parameter zu finden. Siehe hierzu Kapitel 5.4.

5.3 Berechnung des Gradienten der FK

In Kapitel 5.2 muss der Gradient $\nabla_m f(m,x)$ der VK der Schachbrettpose berechnet werden. Die Pose läst sich als 4×4 Transformationsmatrix $P = F(m,x)$ und als siebendimensionaler Vektor $p = f(m,x) = q(F(m,x))$, zusammengesetzt aus einer dreidimensionaler Pose und einem Quaternion, beschreiben. Dabei ist q eine Funktion, welche die Matrixdarstellung in die Vektordarstellung überführt. Es gilt

$$\nabla_m f(m,x) = \nabla_P q(P)\cdot\nabla_m F(m,x)$$

mit

$$\nabla_m F(m,x) = \begin{pmatrix} \frac{\partial F(m,x)}{\partial m_1} \\ \vdots \\ \frac{\partial F(m,x)}{\partial m_r} \end{pmatrix}.$$

Weiter ist $F(m,x) = T_m^1(x)\cdots T_m^n(x)$, wobei $T_m^i(x)$ die i. Transformation der kinematischen Kette ist. Ein Element m_j des Parametervektors m beeinflusst dabei nur maximal eine Transformation $T_m^i(x)$. Es gilt also

$$\begin{aligned}
\frac{\partial F(m,x)}{\partial m_i} &= \frac{\partial\, T_m^1(x)\cdots T_m^n(x)}{\partial m_i} \\
&= T_m^1(x)\cdots T_m^{i-1}(x)\cdot\frac{\partial T_m^i(x)}{\partial m_i}\cdot T_m^{i+1}(x)\cdots T_m^n(x)\,.
\end{aligned}$$

Siehe hierzu auch Magnus et al. [MN02, Kapitel 5.15].

5.4 Gradientenverfahren

Es wird für jedes Paar aus Gelenkstellungen und Schachbrettposen mit dem Verfahren aus Kapitel 5.2 ein Gradient berechnet. Wir verwenden diese Vektoren als Grundlage für Anpassungschritte für die Parameter. Wir summieren diese Gradienten auf und berechnen daraus mit Rprop die Anpassungen für den Parametervektor, siehe Kapitel 3.8. Alternativ hierzu wäre es auch möglich mit der Conjugate-Gradient-Methode, dem Newton-Verfahren, dem Expectation-Maximization-Algorithmus oder anderen Optimierungsverfahren nach Maxima zu suchen.

5.5 Vergleich zwischen URDF-Ketten und DH-Ketten

In diesem Abschnitt werden die Transformationsmatrizen K^i bestimmt, die von DH-Segmentkoordinatensystemen zu URDF-Segmentkoordinatensystemen führen. Für zueinander äquivalente DH und URDF Ketten gilt für alle θ die Gleichung

$$T_{URDF}(\theta) = T_{DH}(\theta)\,,$$

also

$$\begin{aligned} T_{URDF}(\theta) &= O^1 \cdot A^1(\theta) \cdots O^n \cdot A^n(\theta) \\ &= B \cdot Z^1(\theta) \cdot D^1 \cdots Z^n(\theta) \cdot D^n \cdot K^n \\ &= T_{DH}(\theta)\,. \end{aligned} \tag{5.1}$$

O. B. d. A. setzen wir

$$D^n = E\,. \tag{5.2}$$

Wenn $D^n \neq E$ gelten soll, so müssen die Ketten um ein Segment verlängert werden, dann muss $D^{n+1} = E$ gelten.
Für θ^0 gilt:

$$O^1 \cdots O^n = B \cdot D^1 \cdots D^n \cdot K^n\,.$$

Wir werden im Folgenden Teilketten betrachten, die von der Wurzel aus starten. Für diese gilt

$$O^1 \cdot A^1(\theta) \cdots O^i \cdot A^i(\theta) = B \cdot Z^1(\theta) \cdot D^1 \cdots Z^i(\theta) \cdot D^i \cdot K^i\,, \tag{5.3}$$

mit $1 \leq i \leq n$. Insbesondere gilt für θ^0:

$$O^1 \cdots O^i = B \cdot D^1 \cdots D^i \cdot K^i\,.$$

Daraus ergibt sich mit $K^0 := B^{-1}$

$$K^i = \begin{cases} B^{-1} & \text{für } i = 0 \text{ und} \\ (D^i)^{-1} \cdots (D^1)^{-1} \cdot B^{-1} \cdot O^1 \cdots O^i & \text{für } 1 \leq i \leq n. \end{cases}\,. \tag{5.4}$$

Also

$$K^i = (D^i)^{-1} \cdot K^{i-1} \cdot O^i \tag{5.5}$$

für $1 < i \leq n$.

Wir zeigen nun den Zusammenhang zwischen $A^i(\theta)$ und $Z^n(\theta)$ für $1 \leq i \leq n$.
Da alle Matrizen vollen Rang haben folgt aus

$$\begin{aligned} O^1 \cdot A^1(\theta) \cdots O^{i-1} \cdot A^{i-1}(\theta) \cdot O^i \cdot A^i(\theta) &= B \cdot Z^1(\theta) \cdot D^1 \cdots Z^{i-1}(\theta) \cdot D^{i-1} \cdot Z^i(\theta) \cdot D^i \cdot K^i \\ &= B \cdot Z^1(\theta) \cdot D^1 \cdots Z^{i-1}(\theta) \cdot D^{i-1} \cdot K^{i-1} \cdot O^i \cdot A^i(\theta) \end{aligned}$$

die Gleichung

$$K^{i-1} \cdot O^i \cdot A^i(\theta) = Z^i(\theta) \cdot D^i \cdot K^i$$

und daraus

$$A^i(\theta) = (K^{i-1} \cdot O^i)^{-1} \cdot Z^i(\theta) \cdot D^i \cdot K^i \,. \tag{5.6}$$

A^i und Z^i sind bis auf die Koordinatensysteme die gleichen Transformationsfunktionen. Es gilt

$$K^{i-1} \cdot O^i = D^i \cdot K^i \tag{5.7}$$

und somit

$$A^i(\theta) = (D^i \cdot K^i)^{-1} \cdot Z^i(\theta) \cdot D^i \cdot K^i \,. \tag{5.8}$$

Für $i = n$ gilt

$$A^n(\theta) = (K^n)^{-1} \cdot Z^n(\theta) \cdot K^n \,. \tag{5.9}$$

Es gilt also

$$a^i = (D^i \cdot K^i)^{-1} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{5.10}$$

für $1 \leq i \leq n$.

5.6 Erstellung einer URDF-Kette aus einer DH-Kette

In diesem Abschnitt wird gezeigt, wie aus einer DH-Kette eine äquivalente URDF-Kette erstellt wird. Beide Ketten haben die gleichen Segmentposen für alle θ. Wir suchen für

$1 \leq i \leq n$ URDF-Transformationen O^i und A^i, welche gegebenen DH-Transformationen B, K^n,D^i und Z^i entsprechen, also für die die Gleichung 5.1 gilt:

$$O^1 \cdot A^1(\theta) \cdots O^n \cdot A^n(\theta) = \overbrace{B \cdot Z^1(\theta)}^{O^1 \cdot A^1(\theta)} \cdot \overbrace{D^1 \cdot Z^2(\theta)}^{O^2 \cdot A^2(\theta)} \cdot \ldots \cdot \overbrace{D^{n-2} \cdot Z^{n-1}(\theta)}^{O^{n-1} \cdot A^{n-1}(\theta)} \cdot \overbrace{D^{n-1} \cdot Z^n(\theta) \cdot \overbrace{D^n}^{E} \cdot K^n}^{O^n \cdot A^n(\theta)}.$$

Insbesondere gilt für θ^0 die Gleichung

$$O^1 \cdots O^n = \overbrace{B}^{O^1} \cdot \overbrace{D^1}^{O^2} \cdot \ldots \cdot \overbrace{D^{n-2}}^{O^{n-1}} \cdot \overbrace{D^{n-1} \cdot \overbrace{D^n}^{E} \cdot K^n}^{O^n}.$$

Dies ermöglicht uns die Definition

$$O^i = \begin{cases} B & \text{für } i = 1, \\ D^{i-1} & \text{für } 1 < i < n \text{ und} \\ D^{n-1} \cdot K^n & \text{für } i = n. \end{cases}$$

Gleichung 5.5 induziert für $1 \leq i < n$

$$K^i = (D^i)^{-1}$$

da für $i = 1$

$$\begin{aligned} K^1 &= (D^1)^{-1} \cdot K^0 \cdot B \\ &= (D^1)^{-1} \end{aligned}$$

und für $i \to i + 1$

$$\begin{aligned} K^i &= (D^i)^{-1} \cdot K^{i-1} \cdot D^{i-1} \\ &= (D^i)^{-1} \cdot (D^{i-1})^{-1} \cdot D^{i-1} \\ &= (D^i)^{-1} \end{aligned}$$

gilt. Daraus ergibt für $i = n$ wieder

$$\begin{aligned} K^n &= (D^n)^{-1} \cdot K^{n-1} \cdot O^n \\ &= (D^n)^{-1} \cdot (D^{n-1})^{-1} \cdot D^{n-1} \cdot K^n \\ &= K^n. \end{aligned}$$

Nun ergibt sich aus Gleichung 5.8 für $i = 1$

$$\begin{aligned} A^1(\theta) &= (O^1)^{-1} \cdot (K^0)^{-1} \cdot Z^1(\theta) \cdot D^1 \cdot K^1 \\ &= B^{-1} \cdot B \cdot Z^1(\theta) \cdot D^1 \cdot (D^1)^{-1} \\ &= Z^1(\theta), \end{aligned}$$

und für $1 < i < n$

$$\begin{aligned} A^i(\theta) &= (O^i)^{-1} \cdot (K^{i-1})^{-1} \cdot Z^i(\theta) \cdot D^i \cdot K^i \\ &= (D^{i-1})^{-1} \cdot D^{i-1} \cdot Z^i(\theta) \cdot D^i \cdot (D^i)^{-1} \\ &= Z^i(\theta)\,. \end{aligned}$$

Der Fall $i = n$ ist schon in Gleichung 5.9 behandelt worden. Zusammengefasst gilt

$$A^i(\theta) = \begin{cases} Z^i(\theta) & \text{für } 1 \leq i < n \text{ und} \\ (K^n)^{-1} \cdot Z^i(\theta) \cdot K^n & \text{für } i = n. \end{cases}$$

Daraus folgt

$$a^i = \begin{cases} (0\ 0\ 1\ 1)^T & \text{für } 1 \leq i < n \text{ und} \\ (K^n)^{-1} \cdot (0\ 0\ 1\ 0)^T + (0\ 0\ 0\ 1)^T & \text{für } i = n. \end{cases}$$

Dieses Verfahren hat den Nachteil, dass DH-Segmentkoordinatensysteme, um eine feste Transformation zu den natürlichen Segmentkoordinatensystemen verschoben sind. In Kapitel 5.7 zeigen wir ein Verfahren, welches dieses Problem umgeht.

5.7 Modifikation einer URDF-Kette mit den Differenzen von DH-Ketten

Gegeben seien eine URDF-Kette T_{URDF} und zwei DH-Ketten T_{DH} und T'_{DH}. Die Ketten T_{URDF} und T_{DH} sind zueinander äquivalent und die Kette T'_{DH} ist durch Deformationen aus T_{DH} entstanden. Wir suchen nun eine Kette T'_{URDF} welche aus T_{URDF} durch die gleichen Deformationen entsteht und zu T'_{DH} äquivalent ist. Es gilt also für alle θ, dass

$$\begin{aligned} T_{URDF}(\theta) &= O^1 \cdot A^1(\theta) \cdots O^n \cdot A^i(\theta) \\ &= B \cdot Z^1(\theta) \cdot D^1 \cdots Z^n(\theta) \cdot D^n \cdot K^n \\ &= T_{DH}(\theta)\,. \end{aligned}$$

Die Deformationen seien als C^i modelliert

$$\begin{aligned} T'_{DH}(\theta) &= B' \cdot Z^1(\theta) \cdot D^{1\prime} \cdots Z^n(\theta) \cdot D^{n\prime} \cdot K^n \\ &= C^1 \cdot B \cdot Z^1(\theta) \cdot C^2 \cdot D^1 \cdots Z^n(\theta) \cdot C^{n+1} \cdot D^n \cdot K^n\,. \end{aligned}$$

Gesucht sind nun H^i mit

$$\begin{aligned} T'_{URDF}(\theta) &= O^{1\prime} \cdot A^1(\theta) \cdots O^{n\prime} \cdot A^n(\theta) \\ &= H^1 \cdot O^1 \cdot A^1(\theta) \cdots H^n \cdot O^n \cdot A^n(\theta) \\ &= T'_{DH}(\theta)\,. \end{aligned}$$

Wegen Gleichung 5.2 muss gelten, dass $E = D^{n\prime} = D^n = D^n \cdot C^{n+1} = C^{n+1}$ und somit folgt mit Gleichungen 5.8 und 5.9

$$\begin{aligned} & C^1 \cdot B \cdot Z^1(\theta) \cdot C^2 \cdot D^1 \cdots Z^n(\theta) \cdot C^{n+1} \cdot D^n \cdot K^n \\ & \quad = H^1 \cdot O^1 \cdot (D^1 \cdot K^1)^{-1} \cdot Z^1(\theta) \cdot D^1 \cdot K^1 \\ & \quad \cdots H^n \cdot O^n \cdot (K^n)^{-1} \cdot (D^n)^{-1} \cdot Z^n(\theta) \cdot D^n \cdot K^n \,. \end{aligned}$$

Durch Vergleichen ergibt sich für $i = 1$

$$H^1 = C^1 \tag{5.11}$$

und für $1 < i \leq n$

$$Z^{i-1}(\theta) \cdot C^i \cdot D^{i-1} \cdot Z^i(\theta) = Z^{i-1}(\theta) \cdot D^{i-1} \cdot K^{i-1} \cdot H^i \cdot O^i \cdot (D^i \cdot K^i)^{-1} \cdot Z^i(\theta) \,.$$

So folgt für θ^0

$$C^i \cdot D^{i-1} = D^{i-1} \cdot K^{i-1} \cdot H^i \cdot O^i \cdot (D^i \cdot K^i)^{-1}$$

und daraus

$$H^i \cdot O^i = (D^{i-1} \cdot K^{i-1})^{-1} \cdot C^i \cdot D^{i-1} \cdot D^i \cdot K^i \,,$$

beziehungsweise

$$O^{i\prime} = (D^{i-1} \cdot K^{i-1})^{-1} \cdot D^{i-1\prime} \cdot D^i \cdot K^i \,.$$

So gilt mit Hilfe von Gleichung 5.11

$$O^{i\prime} = \begin{cases} B' \cdot B^{-1} \cdot O^1 & \text{für } i = 1, \\ (D^{i-1} \cdot K^{i-1})^{-1} \cdot D^{i-1\prime} \cdot D^i \cdot K^i & \text{für } 1 < i \leq n. \end{cases}$$

Die Funktionen A^i und somit die Achsvektoren a^i bleiben unverändert.

5.8 Gewinnung einer URDF-Kette aus einem URDF-Baum

Zur Veranschaulichung beschreiben wir hier die Gewinnung einer URDF-Kette aus einem URDF-Baum, welche dem Vorgehen für die KDL-Kinematik in Kapitel 3.5 ähnelt. Man beachte hierzu Abbildung 5.1. Die Transformationen O'^i und A'^i sind durch den Baum vorgegeben. Gesucht sind Transformationen O^i und A^i einer Kette, die von Baumsegment 0 nach n führt. Eine mögliche Kette ist

$$\begin{aligned} T_{URDF}(\theta) &= (O'^w \cdot A'^w(\theta) \cdots O'^1 \cdot A'^1(\theta))^{-1} \cdot O'^{w+1} \cdot A'^{w+1}(\theta) \cdots O'^n \cdot A'^n(\theta) \\ &= \overbrace{E}^{O^1} \cdot \overbrace{(A'^1(\theta))^{-1}}^{A^1} \cdot \overbrace{(O'^1)^{-1}}^{O^2} \cdots \overbrace{(A'^w(\theta))^{-1}}^{A^w} \\ &\quad \cdot \overbrace{(O'^w)^{-1} \cdot O'^{w+1}}^{O^{w+1}} \cdot \overbrace{A'^{w+1}(\theta)}^{A^{w+1}} \cdots \overbrace{O'^n}^{O^n} \cdot \overbrace{A'^n(\theta)}^{A^n} \\ &= O^1 \cdot A^1(\theta) \cdots O^w \cdot A^w(\theta) \cdot O^{w+1} \cdot A^{w+1}(\theta) \cdots O^n \cdot A^n(\theta) \,. \end{aligned}$$

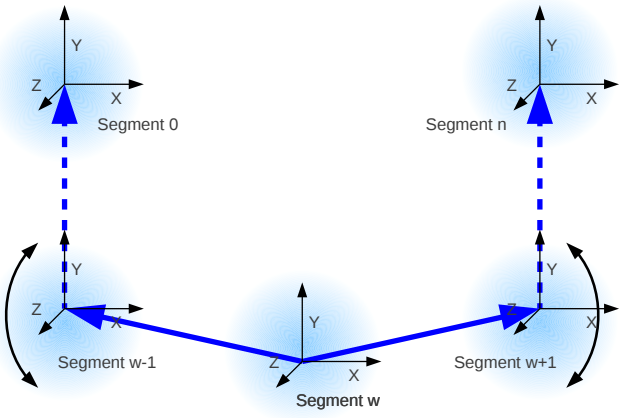


Abbildung 5.1: URDF-Baum

Dies ist nicht die einzige Möglichkeit, es kann auch $O^1 \neq E$ gelten, O^1 kann eine beliebige Rotationsmatrix sein. Dies hat aber Auswirkungen auf andere Matrizen. Wir sehen bei der Wurzel, dass wir Informationen über den Baum verlieren $(O'^w)^{-1} \cdot O'^{w+1} = O^{w+1}$. Diesen Informationsverlust kann man auf den translationalen Anteil O'^w_{trans} von O'^w beschränken, indem man O^1 passend wählt. Diesen Weg beschreiten wir implizit. Wir lassen das URDF vom *kdl_parser* einlesen. Dann gewinnen wir mit Hilfe der hierfür in der KDL vorhandenen Methoden aus dem KDL-Baum eine Kette, siehe Kapitel 4.5. Diese wandeln wir dann in eine URDF-Kette, siehe Kapitel 5.10 und dann in eine DH-Kette, siehe Kapitel 4.8.

5.9 Modifikation eines URDF-Baums anhand einer URDF-Kette

Beim Zurückschreiben der gelernten Kinematik stoßen wir auf das inverse Problem zu dem aus Kapitel 5.8. Die Transformationen O^i und A^i sind durch die Kette vorgegeben. Gesucht sind Transformationen O'^i und A'^i zweier Ketten, die von Wurzelsegment w nach 0 und nach n führen. Es gilt

$$\begin{aligned} T_{URDF}(\theta) &= O^1 \cdot A^1(\theta) \cdots O^w \cdot A^w(\theta) \cdot O^{w+1} \cdot A^{w+1}(\theta) \cdots O^n \cdot A^n(\theta) \\ &= ((A^w(\theta))^{-1} \cdot (O^w)^{-1} \cdots (A^1(\theta))^{-1} \cdot (O^1)^{-1})^{-1} \cdot O^{w+1} \cdot A^{w+1}(\theta) \cdots O^n \cdot A^n(\theta) \\ &= (O'^w \cdot A'^w(\theta) \cdots O'^1 \cdot A'^1(\theta))^{-1} \cdot O'^{w+1} \cdot A'^{w+1}(\theta) \cdots O'^n \cdot A'^n(\theta)\,. \end{aligned}$$

Wir sehen leicht, dass $A'^i = A^i$ und $O'^i = O^i$ für $w < i \leq n$. Betrachten wir nun

$$(A^w(\theta))^{-1} \cdot (O^w)^{-1} \cdots (A^1(\theta))^{-1} \cdot (O^1)^{-1} = O'^w \cdot A'^w(\theta) \cdots O'^1 \cdot A'^1(\theta)\,.$$

Mit $O^i = O^i_{trans} \cdot O^i_{rot}$, wobei O^i_{trans} eine Translation und O^i_{rot} eine Rotation beschreibt, schreiben wir

$$\begin{aligned}
&(A^w(\theta))^{-1} \cdot (O^w_{rot})^{-1} \cdot (O^w_{trans})^{-1} \cdots (A^1(\theta))^{-1} \cdot (O^1_{rot})^{-1} \cdot (O^1_{trans})^{-1} \\
&= \overbrace{(O^w_{rot})^{-1}}^{O'^w} \cdot \overbrace{O^w_{rot} \cdot (A^w(\theta))^{-1} \cdot (O^w_{rot})^{-1}}^{A'^w(\theta)} \cdot \overbrace{(O^w_{trans})^{-1} \cdot (O^{w-1}_{rot})^{-1}}^{O'^{w-1}} \cdots \overbrace{(O^2_{trans})^{-1} \cdot (O^1_{rot})^{-1}}^{O'^1} \\
&\quad \cdot \overbrace{O^1_{rot} \cdot (A^1(\theta))^{-1} \cdot (O^1_{rot})^{-1}}^{A'^1} \cdot \overbrace{(O^1_{trans})^{-1}}^{E} \\
&\quad = O'^w \cdot A'^w(\theta) \cdots O'^1 \cdot A'^1(\theta)\,.
\end{aligned}$$

Man beachte, dass die Transformationsfunktionen A^i nur *rotierbar* und nicht *verschiebbar* sind, da sie durch einen einzelnen Achsenparameter bestimmt werden. Man sieht, dass $O^1_{trans} = E$ gelten muss. Wie wir in Kapitel 5.8 gesehen haben, muss noch der Verlust der Information über die Translationen am Wurzelsegment beachtet werden. Eine Möglichkeit hierfür ist O^w_{trans} beizubehalten. Die berechneten Transformationen können dann so in den Baum übertragen werden.

5.10 Erstellung einer URDF-Kette aus einer KDL-Kette

In diesem Abschnitt wird gezeigt, wie aus einer KDL-Kette eine äquivalente URDF-Kette erstellt wird. Beide Ketten haben die gleichen Segmentposen für alle θ. Wir suchen für $1 \leq i \leq n$ URDF-Transformationen O^i und A^i welche gegebenen KDL-Transformationen J^i_{trans}, J^i_{ax}, L^i_{trans} und L^i_{rot} entsprechen. Für diese gilt

$$\begin{aligned}
T_{KDL} &= J^1_{trans} \cdot J^1_{ax}(\theta) \cdot L^1_{trans} \cdot L^1_{rot} \cdots J^n_{trans} \cdot J^n_{ax}(\theta) \cdot L^n_{trans} \cdot L^n_{rot} \\
&= O^1 \cdot A^1(\theta) \cdots O^n \cdot A^n(\theta) \\
&= T_{URDF}\,.
\end{aligned}$$

Man kann wie folgt zusammenfassen

$$\begin{aligned}
O^1 \cdot A^1(\theta) \cdots O^n \cdot A^n(\theta) = {} & \overbrace{J^1_{trans} \cdot J^1_{ax}(\theta)}^{O^1 \cdot A^1(\theta)} \cdot \overbrace{L^1_{trans} \cdot L^1_{rot} \cdot J^2_{trans} \cdot J^2_{ax}(\theta)}^{O^2 \cdot A^2(\theta)} \\
& \cdots \overbrace{L^{n-2}_{trans} \cdot L^{n-2}_{rot} \cdot J^{n-1}_{trans} \cdot J^{n-1}_{ax}(\theta)}^{O^{n-1} \cdot A^{n-1}(\theta)} \\
& \cdot \overbrace{L^{n-1}_{trans} \cdot L^{n-1}_{rot} \cdot J^n_{trans} \cdot J^n_{ax}(\theta) \cdot L^n_{trans} \cdot L^n_{rot}}^{O^n \cdot A^n(\theta)}\,.
\end{aligned}$$

O. B. d. A nehmen wir $L^n_{trans} = L^n_{rot} = I$ an. Durch Umstellen ergibt sich

$$O^i = \begin{cases} J^1_{trans} \cdot L^1_{rot} & \text{für } i = 1, \\ (L^{n-1}_{rot})^{-1} \cdot L^{n-1}_{trans} \cdot L^{i-1}_{rot} \cdot J^i_{trans} \cdot L^i_{rot} & \text{für } 1 < i \leq n. \end{cases} \tag{5.12}$$

Und weiter

$$A^i(\theta) = (L^i_{rot})^{-1} \cdot J^i_{ax}(\theta) \cdot L^i_{rot}\,.$$

Es gilt also

$$a^{i\prime} = (L^i_{rot})^{-1} \cdot {j^i_{ax}}' ,$$

mit

$$a^i = a^{i\prime} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \tag{5.13}$$

und

$$j^i_{ax} = {j^i_{ax}}' + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} . \tag{5.14}$$

Wir sehen an Gleichung 5.12, dass für $i > 1$ die Werte L^{n-1}_{trans} und J^i_{trans} in O^i aufgehen. Wir verlieren also Informationen über die Kinematik.

5.11 Alternative Formulierung als Offlineverfahren

Neben einem Gradientenverfahren, welches im weiteren Verlauf verwendet wird, wollen wir die alternative Möglichkeit vorstellen, die Parameter durch das Lösen eines linearen Gleichungssystems in einem Offlineverfahren zu bestimmen. Die Formulierung ähnelt dann *GraphSLAM* [TBF06] ohne Ungenauigkeiten bei der Ausführung der Steuerkommandos. Es gilt dann

$$p(m|x_{1:t}, z_{1:t}) = \eta\, p(z_{1:t}|x_{1:t}, m)\, p(m|x_{1:t}) .$$

Dabei ist $m \propto \mathcal{N}(\mu_m, \Sigma_m)$ und es gilt $p(m|x_{1:t}) = p(m)$, da m von $x_{1:t}$ unabhängig ist. Hierbei ist $z_{1:t} = (z_1, z_2, ..., z_t)$, wobei z_t der Wert der Variablen z zum Zeitpunkt t ist. Logarithmiert ergibt sich

$$log\, p(m|x_{1:t}, z_{1:t}) = const. + log\, p(m) + log\, p(z_{1:t}|m) .$$

Unter der Annahme, dass die Messungen statistisch unabhängig sind folgt

$$p(z_{1:t}|x_{1:t}, m) = \prod_{t=1}^{T} p(z_t|x_{1:t}, m) .$$

Weiter nehmen wir an, dass

$$p(z_t|x_t, m) = f(x_t, m) + \epsilon_t ,$$

wobei $\epsilon \propto \mathcal{N}(0, \Sigma_{z_t})$ gilt. Es folgt nun

$$\begin{aligned} log\, p(m|x_{1:t}, z_{1:t}) = \;& const. + \frac{1}{2}\,(m-\mu_m)^T {\Sigma_m}^{-1}(m-\mu_m) \\ & + \frac{1}{2}\sum_{t=1}^{T}(z_t - f(x_t,m))^T {\Sigma_{z_t}}^{-1}(z_t - f(x_t,m))\,. \end{aligned}$$

Wir linearisieren f (Taylor Entwicklung)

$$f(x_t, m) \approx f(x_t, \mu_m) + f'(x_t, \mu_m)(m - \mu_m)$$

und definieren

$$F_t := f'(x_t, \mu_m)\,.$$

Dies ergibt

$$\begin{aligned} log\, p(m|x_{1:t}, z_{1:t}) = \;& const. + \frac{1}{2}\,(m-\mu_m)^T {\Sigma_m}^{-1}(m-\mu_m) \\ & + \frac{1}{2}\sum_{t=1}^{T}(z_t - f(x_t,\mu_m) - F_t(m-\mu_m))^T {\Sigma_{z_t}}^{-1} \\ & (z_t - f(x_t,\mu_m) - F_t(m-\mu_m)) \\ = \;& const. + \frac{1}{2}\,{\mu_m}^T {\Sigma_m}^{-1}\mu_m \\ & - m^T {\Sigma_m}^{-1}\mu_m \\ & + \frac{1}{2}\,m^T {\Sigma_m}^{-1} m \\ & + \frac{1}{2}\sum_{t=1}^{T}(z_t - f(x_t,\mu_m) + F_t\mu_m)^T {\Sigma_{z_t}}^{-1} \\ & (z_t - f(x_t,\mu_m) + F_t\mu_m) \\ & - \sum_{t=1}^{T} m^T\, F_t^T {\Sigma_{z_t}}^{-1}\,(z_t - f(x_t,\mu_m) + F_t\mu_m) \\ & + \frac{1}{2}\sum_{t=1}^{T} m^T\, F_t^T {\Sigma_{z_t}}^{-1} F_t\, m \\ =:\; & const. + \frac{1}{2}\,m^T\Omega m - m^T\xi\,. \end{aligned}$$

Die in m quadratischen Terme ergeben

$$\Omega = \frac{1}{2}\,m^T {\Sigma_m}^{-1} m + \frac{1}{2}\sum_{t=1}^{T} m^T\, F_t^T {\Sigma_{z_t}}^{-1} F_t\, m\,.$$

Die in m linearen Terme ergeben

$$\xi = m^T {\Sigma_m}^{-1}\mu_m + \sum_{t=1}^{T} m^T\, F_t^T {\Sigma_{z_t}}^{-1}\,(z_t - f(x_t,\mu_m) + F_t\mu_m)\,.$$

Daraus folgen die a postiori Parameter

$$\Sigma_{\hat{m}} = \Omega^{-1}$$

und

$$\begin{aligned}\hat{m} &= \Sigma_{\hat{m}}\, z \\ &= \Omega^{-1}\xi\,.\end{aligned}$$

Die Parameter lassen sich nun durch Lösung eines linearen Gleichungssystems bestimmen.

5.12 Austausch des a priori Modells

Es besteht die Möglichkeit, den Lernvorgang nicht immer mit dem gleichen a priori Modell durchzuführen. Wenn man optimistisch davon ausgeht, dass das Modell mit fortschreitendem Lernen besser wird, so kann man das a priori Modell regelmässig durch das a posteriori Modell ersetzen. Dies kann zu schnelleren Lernfortschritten bei der VK des Endeffektors führen, ist aber auch mit stärkeren Abweichungen bei den kinematischen Parametern verbunden.

5.13 Mehrere Modelle - Lokal gewichtetes Lernen

Es besteht die Möglichkeit, für verschiedene Bereiche des Konfigurationsraums unterschiedliche kinematische Modelle zu lernen. Um dies zu erreichen, gewichten wir die einzelnen Messwerte beim Lernen anhand ihres Abstands zu einem jeweiligen Zentrum des Teilraums. Die Gewichtung erfolgt beim Aufsummieren der Einzelvektoren, siehe Kapitel 5.4. Diese gehen nun mit $p(x)$ gewichtet in die Summe ein. Wobei p die wie in Kapitel 3.6 beschriebene mehrdimensionale Normalverteilung ist. Dabei ist x die Konfiguration der Gelenke und μ das Zentrum des betreffenden Teilraums. Die Kovarianzmatrix Σ wird entspechend der Größe des Teilraums gewählt.

Eine Möglichkeit besteht darin, jede Lernkonfiguration zu einem Zentrum zu machen. Es wird dann für die IK das Modell genommen, dessen Zentrum im Arbeitsraum der Zielpose am nächsten liegt.

6 Implementierung

Die Schritte der Kalibrierung sind auf mehrere ROS-Pakete im Stack *hubert_calib* verteilt. Das Paket *hubert_calib_exploration* ist für die Datenerhebung zuständig. Die eigentliche Kalibrierungssoftware befindet sich in *hubert_calib_calibration*. Programme zur Evaluation und Bibliotheken sind auf weitere Pakete verteilt. In den Paketen befinden sich die entwickelten Applikationen und Startdateien und Konfigurationen, um diese zu initialisieren. Die vorhandene Software kann für die Roboter Cosero und Dynamaid benutzt werden. Ausführungen zu einem der beiden Roboter gelten im Allgemeinen auch für den anderen. Für den PR2 stehen nur die Datenaufnahme im Simulator, die Kalibrierung und die Datenauswertung zur Verfügung.

Im Anschluss an die Beschreibung der Software wird die für die Arbeit entwickelte Halterung für das Schachbrettmuster und den Laserpointer beschrieben.

Es können mehrere Kameras verwendet werden, sofern diese auf dem gleichen Starrkörper mit bekannten Abständen und Verdrehungen zu dessen Koordinatensystem montiert sind. Auf das Vorgehen bei der Kalibrierung hat es keinen Einfluss ob eine oder mehrere Kameras verwendet werden.

Skripte zur direkten Ausführung sind in den Paketen unter *scripts* abgelegt und Konfigurationsdateien unter *config*. Quelltexte sind entsprechend dem ROS-Standard unter *src* und Pythonbibliotheken unter *src/<PAKETNAME>* abgelegt. Startdateien befinden sich unter *launch*, ausführbare Binärdateien unter *bin*, Binärbiliotheken unter *lib* und Headerdateien in *include*. Ein Dateiname ist in den meisten Fällen innerhalb eines Pakets eindeutig.

Die Implementierung erfolgte hauptsächlich in Python. Es gab auch Anpassungen existierender Software in C++. Für die Verwendung von C++-Laufzeitbibliotheken, wie bei KDL und DH-Konverter notwendig, wurden mit C++ und SIP [SIP12] Wrapper erstellt. Damit die Berechnungen in Python ausreichend schnell durchgeführt werden können wird *NumPy* [Num12] verwendet. Bei der Visualisierung der Daten wird *Matplotlib* [Mat12] verwendet. Für die Benutzerschnittstellen werden TkInter [TkI12] und WxPython [WxP12] benutzt.

Alle Programme werden mit Konfigurationsdateien im *YAML*-Format [YAM12] initialisiert, deren Pfade werden auf der Kommandozeile angegeben, siehe hierzu Tabelle 6.1. Einzelne Schlüssel-Wert-Paare können auch auf der Kommandozeile gesetzt oder überschrieben werden. Beide Optionen können mehrfach angegeben werden, dabei werden Konfigurationen zusammengeführt. Bei gleichen Feldern gilt die letzte Definition.

Tabelle 6.1: Allgemeine Kommandozeilenparameter

Parameter	**Bedeutung**
-c *Feld.Feld:Wert*	Feld-Wert-Paare.
-d *Pfad*	Der Pfad zu einer YAML-Konfigurationsdatei.

6.1 Exploration

Im Paket *hubert_calib_exploration* befindet sich die Software zur Aufnahme der Kalibrierungsdaten. Die Datenaufnahme basiert auf dem Paket *pr2_calib*, siehe Kapitel 4.7. Zusätzlich zu der Exploration können mit diesem Paket auch die Stellungen aufgenommen werden, welche bei der Exploration angefahren werden. Es besteht die Möglichkeit, die Datenaufnahme mit Hilfe von Gazebo zu simulieren oder synthetische Testdaten mit vorgegeben Fehlern zu erzeugen. In Abbildung 6.1 ist die Simulation eines Explorationslaufs zu sehen.

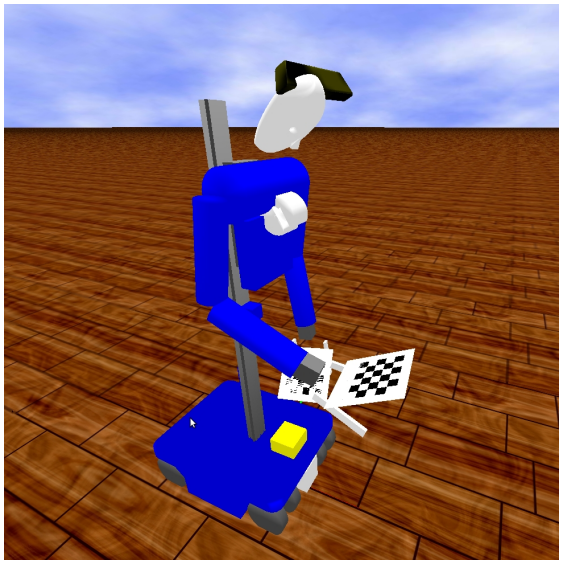


Abbildung 6.1: Simulierte Exploration

Aus dem Stack *pr2_calib*, siehe Kapitel 4.7, werden aus den Paketen *image_cb_detector* und *laser_cb_detector* Dienste zur Erkennung und Kennzeichnung von Schachbrettmustern in Kamerabildern verwendet. Es werden auch Knoten aus den Paketen *monocam_settler*, *joint_states_settler* und *interval_intersection* verwendet. Diese werden verwendet, um zu ermitteln, wann sich Gelenkstellungen und Kamerabild nach einer Bewegung beruhigt haben. Die Skripte und Konfigurationen aus *pr2_calibration_executive* dienten hierbei als Vorlage bei der Verwendung dieser Dienste. Die Starter für die Dienste befinden sich für Cosero unter *cosero_camera.launch*, *cosero_services.launch* und *cosero_viewers.launch*.

Die Aufnahme der Roboterstellungen für die spätere Exploration erfolgt am realen Roboter. Dieser nimmt die Gelenkstellungen der neutral gestellten Aktuatoren des jeweiligen Armes auf und richtet seinen Kopf so aus, dass die Kamera die erwartete Position des Schachbrettmusters aufnimmt. Sobald sich das Schachbrett im Blickfeld befindet, werden die Gelenkkonfigurationen des Nackens und des Armes in einer Konfigurationsdatei gespeichert. Dieser Vorgang wird beliebig häufig wiederholt. Es ist dabei darauf zu achten, dass der Konfigurationsraum der Arme möglichst gut abgedeckt wird und Bewegungen im Nullraum der Endeffektorkinematik stattfinden, also die gleiche Endeffektorstellung bei verschiedenen Armstellungen erreicht wird.

Um die Aufnahme der Stellungen für den rechten Arm zu starten, muss die Startdatei *cosero_teach_joints_real_right.launch* ausgeführt werden. Diese startet den Roboter mit *cosero_real_passive.launch* aus Kapitel 6.5 und die notwendigen Knoten aus *pr2_calib*. Dann wird der Knoten *look_to_frame.py* gestartet, welcher mit Hilfe des Robotcontrols den Kopf steuert. Das Programm *teach_joints.py* übernimmt die Datenaufnahme. Es speichert die Gelenkstellungen in eine Konfigurationsdatei, nachdem es darauf gewartet hat, dass das Schachbrett zu sehen ist. Die Kommandozeilenparameter sind in Tabelle 6.2 aufgeführt.

Tabelle 6.2: Kommandozeilenparameter der Gelenkstellungsaufzeichnung

Parameter	**Bedeutung**
Parameter 1	Der Pfad der Vorlagedatei für die erstellten Gelenkkonfigurationen.
Parameter 2	Der Pfad des Verzeichnisses, in dem die Gelenkkonfigurationen abgelegt werden.

Das Hilfsprogramm *tool_manipulate_config.py* erstellt durch Spiegelung aus der Konfiguration für den rechten Arm eine Konfiguration für den Linken, und umgekehrt. Dabei werden die Stellparameter der seitenabhängigen Gelenke negiert.

Bei der Selbstexploration fährt der Roboter die aufgenommenen Stellungen an. Hierbei wird der Nacken mit den vorgegebenen Gelenkstellungen gesteuert und nicht wie zuvor automatisch auf die Hand ausgerichtet. Die Kopfstellungen werden so variiert, dass neben der ursprünglichen Stellung auch vier weitere mit leicht veränderter Blickrichtung, mit der ursprünglichen in der Mitte, aufgenommen werden. Die Werte hierfür werden vor dem Start konfiguriert. Die Datenaufnahme kann so konfiguriert werden, dass alle Gelenkstellungen in einem engen Bereich um die ursprüngliche Stellung zufällig gewählt werden. Alternativ hierzu kann der Kopf auch automatisch ausgerichtet werden, dabei werden die Kopfstellungen aber nicht variiert. Nach dem Anfahren jeder Stellung wird dann darauf gewartet, dass sich die Gelenkstelllungen und das Kamerabild stabilisieren. Geschieht dies nicht, oder das Schachbrett wird nicht erkannt, wird nach einem vorher konfigurierbaren Zeitraum mit der nächsten Stellung fortgefahren. Die Daten werden als Topics publiziert und in einer Bag-Datei aufgenommen. Zusätzlich wird in dieser Datei noch die aktuelle URDF-Roboterbeschreibung gespeichert.

Um den Explorationsvorgang mit vorgegebenen Stellungen für den rechten Arm von Cosero durchzuführen, startet man *cosero_exploration_real_right.launch*, dabei startet

dann mit Hilfe von *cosero_real.launch* aus Kapitel 6.5 der Roboter. Zu den Datenaufnahmeknoten aus *pr2_calib* wird auch ein Knoten zum Publizieren des URDF *urdf_pub.py* und *rosbag* zur Datenaufzeichnung gestartet. Das Programm *explore.py* steuert dann die Datenaufnahme und publiziert die Schachbrettinformationen und Gelenkstellungen.

Um die Datenaufnahme im Simulator zu testen verwendet man den Starter *cosero_exploration_sim_right.launch*. Dieser startet mit Hilfe von *cosero_services.launch* aus Kapitel 6.6 die benötigten Knoten. Der Ablauf ist dann der Gleiche wie beim realen Roboter. Es ist hierbei möglich, zwei verschiedene URDF-Modelle zu verwenden. Eines, welches der Simulator benutzt, welches also das reale Modell repräsentiert, und eines mit dem die restlichen Komponenten arbeiten, welches also das a priori Modell repräsentiert. Die Kommandozeilenparameter sind in Tabelle 6.3 aufgeführt.

Tabelle 6.3: Kommandozeilenparameter der Exploration

Parameter	**Bedeutung**
Parameter 1	Der Pfad des Verzeichnisses, in dem die Gelenkkonfigurationen abgelegt sind.

Um Testdaten für die Kalibrierung zu erzeugen, wird das Programm *synthesize.py* verwendet. Es generiert aus der VK eines gegebenen Modells Schachbrettposen und speichert sie mit den dazugehörenden Gelenkstellungen in einer Datei. Diese wird dann zum Kalibrieren verwendet. Die Gelenkstellungen und die Schachbrettpose können mit einem konfigurierbaren, normalverteilten Fehler verrauscht werden. Die verwendeten Stellungen können aus den gleichen Konfigurationsdateien ausgelesen werden wie bei der Exploration. Zudem ist eine völlig randomisierte, beliebig große Auswahl an Stellungen möglich. Die Kommandozeilenparameter sind in Tabelle 6.4 aufgeführt.

Tabelle 6.4: Kommandozeilenparameter der Synthese

Parameter	**Bedeutung**
Parameter 1	Der Pfad des Verzeichnisses, in dem die Gelenkkonfigurationen abgelegt sind.
-u *Pfad*	Der Pfad zu einer URDF-Datei für das Modell.
-p *ROS-Parametername*	Der Propertyname eines URDF-Eintrags für das Modell. Alternativ zu *-u*. Wenn '-' angegeben wird, so wird '*robot_description*' verwendet.

Die Konfigurationsdateien der Exploration, Gelenkstellungsaufzeichnung und Synthese entsprechen dem in Tabelle 6.5 beschriebenen Format, vergleiche hierzu auch [pr212b].

Tabelle 6.5: Konfiguration der Exploration, Gelenkstellungsaufzeichnung und Synthese

Feld	**Bedeutung**
cameras.*	Der Bereich für die Konfiguration der Kameras.
cameras.*Kamera*.*	Die Konfiguration der Kamera '*Kamera*'.
cameras.*Kamera*. cam_info_name	Das Suffix des Kamerainfotopics. Der Topic ist also *ns* + '/' + *cam_info_name*.
cameras.*Kamera*. cb_detector_config	Namensraum des *SimpleActionClient* des Detektors.
cameras.*Kamera*.configs.*	Hier wird mindestens ein Schachbrettmuster konfiguriert.
cameras.*Kamera*.configs. *Muster*.*	Die Konfiguration des Musters '*Muster*'.
cameras.*Kamera*.configs. *Muster*.cb_detector.*	Hier wird der Schachbrettmusterdetektor *image_cb_detector* für dieses Muster konfiguriert.
cameras.*Kamera*.configs.*Muster*.cb_detector. height_scaling	Vertikaler Vergrößerungsfaktor zur Verbesserung der Mustererkennung.
cameras.*Kamera*.configs. *Muster*.cb_detector.num_x	Anzahl der Reihen der Kreuzungspunke.
cameras.*Kamera*.configs. *Muster*.cb_detector.num_y	Anzahl der Spalten der Kreuzungspunke.
cameras.*Kamera*.configs. *Muster*.cb_detector.subpixel_window	Konfiguration für *FindCornerSubPix()* in OpenCV [Ope12a].
cameras.*Kamera*.configs. *Muster*.cb_detector.subpixel_zero_zone	Konfiguration für *FindCornerSubPix()* in OpenCV [Ope12a].
cameras.*Kamera*.configs.*Muster*.cb_detector. width_scaling	Horizontaler Vergrößerungsfaktor zur Verbesserung der Mustererkennung.
cameras.*Kamera*.configs. *Muster*.settler.*	Hier wird der Settler *monocam_settler* für das Muster konfiguriert.
cameras.*Kamera*.configs. *Muster*.settler.cache_size	Die Größe des Nachrichtenpuffers.
cameras.*Kamera*.configs.*Muster*.settler.ignore_failures	Wenn *True*, dann werden fehlerhafte Meldungen gelöscht.
cameras.*Kamera*.configs. *Muster*.settler.max_step	Maximaler Zeitabstand in einem Intervall.
...	

Tabelle 6.5: Konfiguration der Exploration, Gelenkstellungsaufzeichnung und Synthese (Fortsetzung)

Feld	**Bedeutung**
cameras.*Kamera*.configs.*Muster*.settler.tolerance	Die maximale Toleranz.
cameras.*Kamera*.ns	Namensraum des Kameratopics.
cameras.*Kamera*.settler_config	Namensraum des *SimpleActionClient* des Settlers.
chains.*	Hier wird die Datenaufnahme der Gelenkstellungen konfiguriert.
chains.*Kette*.*	Konfiguration für die Kette '*Kette*'.
chains.*Kette*.configs.*	Es können hier mehrere Aufnahmegenauigkeiten konfiguriert werden.
chains.*Kette*.configs.*Toleranz*.*	Die Konfiguration mit dem Namen '*Toleranz*'.
chains.*Kette*.configs.*Toleranz*.settler.*	Die Konfiguration für den Settler *joint_states_settler*.
chains.*Kette*.configs.*Toleranz*.settler.cache_size	Die Größe des Nachrichtenpuffers.
chains.*Kette*.configs.*Toleranz*.settler.joint_names[]	Liste mit den Gelenknamen.
chains.*Kette*.configs.*Toleranz*.settler.max_step	Maximaler Zeitabstand in einem Intervall.
chains.*Kette*.configs.*Toleranz*.settler.tolerances[]	Liste mit den akzeptierten Toleranzen der Gelenke.
chains.*Kette*.settler_config	Der Namensraum des *SimpleActionClient* des Settlers.
controllers.*	Hier werden die Aktuatoren konfiguriert.
controllers.*Kette*.*	Die Konfiguration für die Kette '*Kette*'.
controllers.*Kette*.joint_names[]	Liste mit den Gelenknamen.
controllers.*Kette*.topic	Der Name des Topics für die Steuerung.
controllers.*Kette*.type	Für die Steuerung des Kopfes mit dem Robotcontrol *nimbro_msgs.msg.BodyPartsCommand*, für die Steuerung der Arme mit dem Robotcontrol durch Trajektorien *control_msgs.msg.FollowJointTrajectoryAction*, für die Steuerung durch Zielstellungen *nimbro_msgs.srv.ArmMotionPrimitiveRequest*, für die Steuerung der Arme des PR2 *trajectory_msgs.msg.JointTrajectory* und *None* falls abgeschaltet.
...	

Tabelle 6.5: Konfiguration der Exploration, Gelenkstellungsaufzeichnung und Synthese (Fortsetzung)

Feld	**Bedeutung**
cooldown_executions	Anzahl an angefahrenen Stellungen, bevor der Roboter die Stellung *convenient_pose.yaml* anfährt, diese wird sonst nicht verwendet.
cooldown_seconds	Dauer in Sekunden, die der Roboter in der Stellung aus *convenient_pose.yaml* verweilt.
exploration_timeout	Dauer in Sekunden, in der das Schachbrettmuster sicher erkannt sein muss.
explorer.*	Der Bereich der Konfiguration für das Explorieren.
explorer.multi.*	Hier werden die Variationen der Stellungen konfiguriert.
explorer.multi.displacements.*	Diese Gelenke werden um den angegebenen Betrag vor und zurück bewegt. Die Menge der zusätzlichen Stellungen entsteht aus dem kartesischen Produkt, enthält aber mindestens ein Element.
explorer.multi.displacements.*Gelenk*	Der Betrag für Gelenk '*Gelenk*'.
explorer.multi.joint_lim	Alle Gelenkstellungen der Kette werden mit einer Gleichverteilung in $[-joint_lim; joint_lim]$ verrauscht.
explorer.multi.rand_seed	Der Seed-Wert für den Zufallsgenerator der Normalverteilung.
interactive	Wenn *True* in Python, dann wird bei misslungenen Messungen nachgefragt ob diese wiederholt werden sollen.
synth.*	Die Konfiguration für die synthetische Generierung von Messwerten.
synth.ef_std_ori	Die Standardabweichung für das normalverteilte Verrauschen der Endeffektororientierung in *rad*.
synth.ef_std_pos	Die Standardabweichung für das normalverteilte Verrauschen der Endeffektorposition in *mm*.
synth.joint_std	Die Standardabweichung für das normalverteilte Verrauschen der Gelenkstellungen in *rad*.
synth.random_count	Anzahl der generierten Stellungen, wenn diese zufällig erzeugt werden, also *type* den Wert '*random*' hat.
synth.root_link	Das Wurzelsegment der zu generierenden Kette.
synth.seed	Der Seed-Wert für den Zufallsgenerator der Normalverteilung.
synth.tip_link	Das Endeffektorsegment der zu generierenden Kette.
...	

Tabelle 6.5: Konfiguration der Exploration, Gelenkstellungsaufzeichnung und Synthese (Fortsetzung)

Feld	**Bedeutung**
synth.type	Wenn die Stellungen aus Dateien stammen so ist der Wert '*files*', wenn sie in $[-\pi;\pi]^f$ gleichverteilt generiert werden ist der Wert '*random*' und '*allowed*' wenn sie im Bereich der erlaubten Gelenkstellungen gleichverteilt generiert werden.

Die Felder der Konfigurationsdatei für die Datenaufzeichnungstellungen sind in Tabelle 6.6 beschrieben, vergleiche hierzu auch [pr212b].

Tabelle 6.6: Konfiguration einer Datenaufzeichnungstellung

Feld	**Bedeutung**
camera_measurements[]	Die Liste mit den Konfigurationen der Kameras.
camera_measurements[i].*	Die i. Konfiguration.
camera_measurements[i]. cam_id	Der Name der Kamera.
camera_measurements[i]. config	Der Name des verwendeten Musters.
joint_commands[]	Hier werden die Aktuatoren konfiguriert.
joint_commands[i].*	Die i. Konfiguration.
joint_commands[i].controller	Der Name der Kette.
joint_commands[i].segments[]	Die Liste der Segmente der abzufahrenden Trajektorie.
joint_commands[i].segments[j].*	Das j. Segment.
joint_commands[i].segments[j].duration	Zeitraum in Sekunden zwischen den Segmenten.
joint_commands[i].segments[j].positions[]	Die Gelenkstellungen.
joint_measurements[]	Die Liste der Konfigurationen der Gelenkaufzeichnung.
joint_measurements[i].*	Die i. Kette.
joint_measurements[i]. chain_id	Der Name der Kette.
joint_measurements[i].config	Der Name der gewünschten Genauigkeit.

6.2 Kalibrierung und Analyse

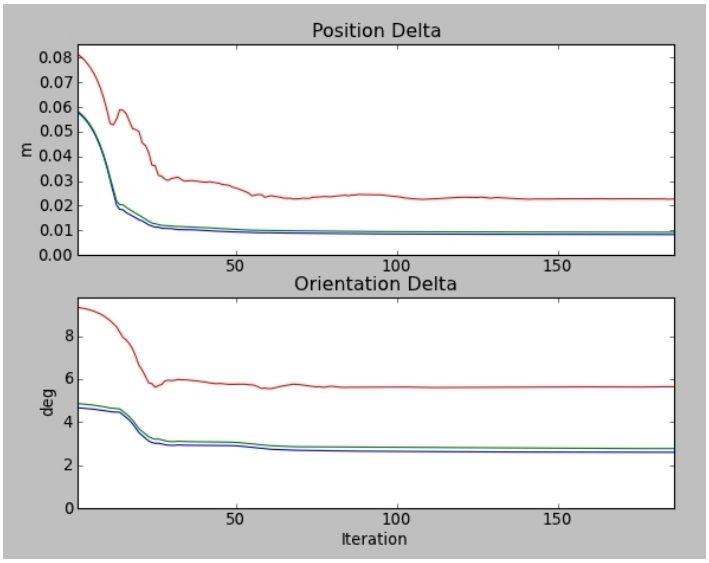
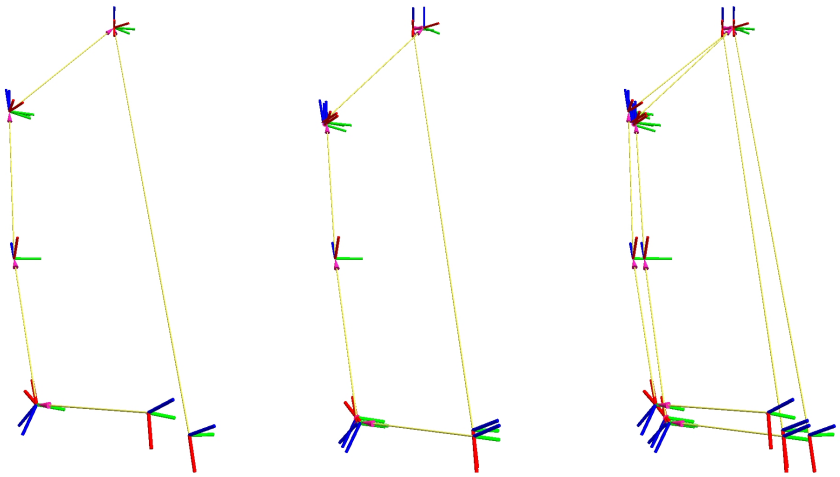


Abbildung 6.2: Fehlerentwicklung beim Lernen
rot: Maximum, *blau*: Mittelwert, *grün*: Effektivwert

Im Paket *hubert_calib_calibration* sind die Komponenten, welche die Kalibrierung und Analyse durchführen. Das Programm *calibrate.py* startet eine Kalibrierung, diese bekommt die Daten der Exploration als Eingabe. Dies sind die Positionen der Kreuzungspunkte des Schachbretts im Kamerabild als Pixelkoordinaten und das a priori URDF-Modell. Die Ausgabe ist ein URDF-Modell, welches die a posteriori Annahme nach dem Gradientenverfahren darstellt. Zusätzlich wird eine Logdatei geschrieben.

Vor der Kalibrierung der Kinematik gibt es einen optionalen Schritt in der die Kameras kalibriert werden. Hierfür werden mit den Schachbrettkoordinaten unter Zuhilfenahme von Bibliotheken aus [cam12a] mit OpenCV Kalibrierungen für die Kameras erzeugt. Diese werden in den nachfolgenden Schritten verwendet und protokolliert. Wenn eine gültige Kalibrierung der Kameras vorliegt, so ist dieser Schritt nicht notwendig.

Das URDF-Modell wird mit der Hilfe von *hubert_kdl_wrapper*, siehe Kapitel 6.8, als KDL-Baum eingelesen. Aus diesem wird dann die KDL-Kette von der Kamera bis zum Endeffektor ausgelesen. Die KDL-Kette wird dann in eine URDF-Kette gewandelt. Der Vorteil, den Umweg über KDL-Bäume und Ketten zu gehen, liegt darin, dass die Wandlungen von URDF bis zu KDL-Ketten standardmäßig in ROS verwendet wird und somit gut getestet ist. URDF-Darstellungen werden im Quelltext auch ORA-Ketten (Origin-Rotation-Axis) genannt, da jedes Gelenk durch diese drei Parametervektoren beschrieben wird. An die URDF-Kette wird dann noch eine fixe Transformation angehängt, um vom Koordinatensystem des Schachbretts in das des Endeffektors zu gelangen. Aus dieser Kette wird dann mit Hilfe vom *nimbro_dh_converter*, Kapitel 6.11 eine DH-Kette gewonnen.



(a) a priori (b) a posteriori (c) beide

Abbildung 6.3: 3D Ansicht der Ketten beim größten Fehler der Orientierung

Aus den Schachbrettkoordinaten wird dann mit OpenCV, siehe Kapitel 4.6, deren Pose im Kamerakoordinatensystem bestimmt. Bei der Verwendung mehrerer Kameras werden die Schachbrettposen in das Koordinatensystem des Starrkörpers, auf dem die Kameras montiert sind, transformiert.

Die DH-Kette wird dann mit MAP und Rprop optimiert, siehe Kapitel 5.2 und 5.4. Dazu wird für jedes Paar aus Schachbrettpose und Gelenkstellungen der Gradient mit MAP berechnet. Diese Gradienten werden dann aufsummiert und fließen dann in Rprop ein. Auf Computern mit mehreren CPU-Kernen werden die Gradienten parallel in mehreren Prozessen berechnet. Nach jeder Iteration werden die Parameter bereinigt. Winkel werden dabei auf das Intervall $(-\pi, \pi]$ projiziert. Zudem wird sichergestellt, dass konstante Parameter unverändert bleiben. So wird das letzte Gelenk nicht verändert, siehe Kapitel 5.5, und es kann hier eingestellt werden, ob Getriebebeiwerte gelernt werden sollen. Diese Schritte werden dann wiederholt, bis sich die gewünschte Genauigkeit einstellt, keine Veränderung mehr stattfindet oder eine vorgegebene Anzahl Iterationen erreicht ist.

Mit dem jeweils aktuellen a posteriori DH-Modell wird entsprechend Kapitel 5.7 aus den a priori Modellen ein a posteriori URDF-Modell erzeugt. Die Abweichungen des gelernten URDF-Modells zu den Trainingsdaten werden während des Lernvorgangs visualisiert, siehe Abbildung 6.2. Aus jedem Satz Gelenkstellungen der Trainingsdaten wird mit dem aktuellen Modell die erwartete Endeffektorpose berechnet und mit der zugehörigen Endeffektorpose verglichen. Es werden die a priori und a posteriori URDF-Modelle in den Stellungen mit den größten Abweichungen sowohl in der Position als auch in der Orientierung visualisiert. Dabei wird auch die Pose des Schachbretts im Kamerakoordinatensystem dargestellt. Visualisiert werden Abweichungen zwischen Messungen und den erwarteten Posen und die Entwicklung der Modelle. Die Darstellung geschieht

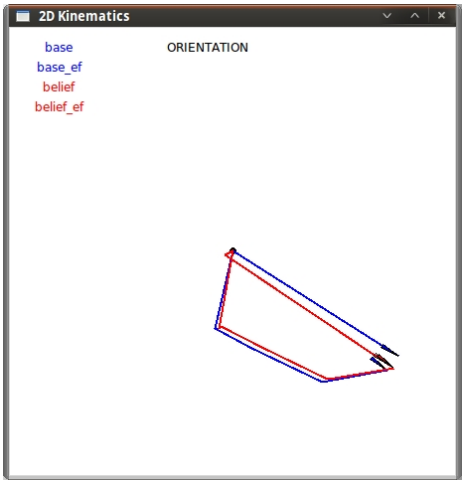


Abbildung 6.4: 2D Aufsicht der Ketten beim größten Fehler der Orientierung

durch publizierte TF-Transformationen, welche in *rviz* dargestellt werden können, siehe Abbildung 6.3.

Während des Kalibrierens wird in einem Fenster die 2D-Aufsicht auf den Roboter dargestellt, siehe Abbildung 6.4. Wenn der Kalibrierungsprozess so konfiguriert ist, dass das a priori Modell während der Iterationen regelmässig durch das a posteriori Modell ausgetauscht wird, werden hier drei Modelle dargestellt, das ursprüngliche Modell, das aktuelle a priori Modell und das a posteriori Modell. Tastatureingaben in dieses Fenster steuern das Programm, siehe Tabelle 6.7.

Tabelle 6.7: Tastatureingaben in der Kalibrierung

Eingabe	**Bedeutung**
q	Bricht die Iterationen ab und gibt die Daten aus.
Leerzeichen	Wechselt zwichen dem Ausführen der Berechnungen und Pause.

Die Ausgabe besteht aus einer URDF-Datei, einer Logdatei und einer Datei mit der optimierten Kette in maschinenlesbarer Form. Für die URDF-Datei kann auch eine Vorlage eingelesen werden, in der das a posteriori Modell eingetragen wird. In diese werden dann die veränderten Gelenkstellungen eingefügt, vergleiche hierzu Kapitel 5.9. Die Anpassung findet in den URDF-Elementen *origin* und *mechanicalReduction* statt, vergleiche hierzu Kapitel 4.4. Die Einträge *axis* bleiben wie im a priori Modell, vergleiche hierzu Kapitel 5.7. Wird diese Datei nicht angegeben, so wird der URDF-Text genommen, aus der das a priori Modell gewonnen wurde. Auch diese Quelle kann getrennt angegeben werden. Vorgabe ist das Modell aus der Exploration.

Das Programm *analyse.py*, siehe Abbildung 6.5, führt Analysen auf den Daten durch und stellt deren Ergebnisse dar. Der Menüpunkt *plot_errors_3d* st stellt den Fehler eines Modells in Abhängigkeit zu der VK des Endeffektors im Arbeitsraum dar, siehe

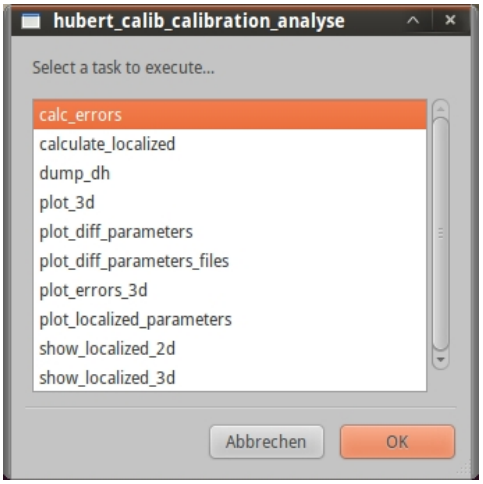


Abbildung 6.5: analyse.py

Abbildung 6.6. Der Ursprung befindet sich hier im Torso.

Mit *plot_3d* werden die Schachbrettpositionen anhand der VK im Arbeitsraum visualisiert, siehe Abbildung 6.7.

Der Menüpunkt *calc_errors* gibt die Fehler des Referenzmodells und des a posteriori Modells aus. Die Eingabe hierfür ist eine Menge von Endeffektorposen mit zugehörigen Gelenkstellungen. Aus jeder Konfiguration wird mit dem jeweiligen Modell die erwartete Endeffektorpose berechnet. Diese wird dann mit der gegebenen Endeffektorpose verglichen. Dabei werden die Postitions- und Orientierungsdifferenzen berechnet.

Bei der Auswahl von *plot_diff_parameters* werden die Unterschiede zwischen zwei DH-Ketten visualisiert, siehe Abbildung 6.8. Zudem werden die Unterschiede in Textform dargestellt. Die roten und blauen Balken repräsentieren das a posteriori Modell. Die grünen und gelben Balken zeigen die Werte des Referenzmodells. In den linken Bildern sind θ und α für alle Gelenke dargestellt, in den mittleren d und a und in den rechten der Getriebebeiwert. Die Wurzel der Kette ist in den Abbildungen links dargestellt. Im oberen Teil sind die absoluten Werte der DH-Parameter für jedes Gelenk aufgetragen. Man sieht hier, dass die Parameter des letzten Gelenks konstant Null bleiben, vergleiche hierzu Kapitel 5.5. Die schmalen Balken repräsentieren das Referenzmodell, grün für θ, d und Getriebebeiwert und gelb für a und α. Die breiten Balken repräsentieren das a posteriori Modell, rot für θ, d und Beiwert und blau für a und α. Im unteren Teil zeigen die Balken die Abweichung zwischen a posteriori und Referenzmodell. Dabei werden die Referenzwerte von den a posteriori Werten abgezogen. Zu beachten ist, dass die Angaben für θ und α in Grad, die für d und a in Millimetern und die für den Getriebebeiwert in Prozent sind. Die dargestellten Werte werden auch als Text ausgegeben.

Der Menüpunkt *calculate_localized* berechnet lokal gewichtete Modelle und speichert diese, siehe hierzu Kapitel 5.13. Der Punkt *plot_localized_parameters*, Abbildung 6.9, zeigt die Verteilung der Werte bei lokalisiertem Lernen, verglichen mit einem Referenzmodell. Die Darstellung unterscheidet sich zu der von *calc_errors* in den folgenden Punk-

ten. Die roten und blauen Balken repräsentieren die aus allen lokalisierten Modellen gewonnenen Kenngrößen. Die breiten Balken repräsentieren den jeweiligen Mittelwert der lokalisierten Modelle, rot für θ, d und Getriebebeiwert und blau für a und α. Im unteren Teil zeigen die schmalen Balken die maximale Abweichung vom Mittelwert und die breiten Balken die Standardabweichung der lokalisierten Modelle.

Die Auswahl von *show_localized_2d*, Abbildung 6.10, visualisiert die mittleren und die maximalen Fehler, also Abstände zu den gemessenen Posen der lokalisierten Modelle. Die werden gegen den kartesischen Abstand der Gelenkstellungen des Zentrums zu der evaluierten Stellung aufgetragen. Mit *show_localized_3d*, Abbildung 6.11, werden Fehler für jedes Zentrum einzeln gezeigt.

Für dieses Paket wurden Unittests erstellt, welche von *rostest* verwendet werden können, siehe Kapitel 6.9. Diese stellen sicher, dass die Semantik von KDL-, URDF-/ORA- und DH-Kinematik erfüllt ist, dass also zwei äquivalente Ketten bei gleichen Gelenkstellungen die gleiche Endeffektorpose erzeugen. Es wird getestet, ob die Umwandlungen zwischen den Typen korrekt verläuft. Das Einlesen und Erzeugen von URDF wird getestet. Zudem wird die korrekte Verwendung der KDL und kinematischer Berechnungen geprüft.

Neben der Angabe der Konfigurationsdateien, siehe Tabelle 6.9, existieren weitere Kommandozeilenparameter für die Programme, siehe Tabelle 6.8.

Tabelle 6.8: Kommandozeilenparameter der Kalibrierung und Analyse

Parameter	**Bedeutung**
Parameter 1	Pfad zu der Datei mit den Explorationsdaten. Dies kann eine *.bag-* oder *.pkl*-Datei sein. Aus dieser Datei stammt auch das a priori URDF-Modell.
-u *Pfad*	Pfad zu einer URDF-Datei für ein alternatives a priori Modell.
-p *ROS-Parametername*	Der Propertyname eines URDF-Eintrags für ein alternatives a priori Modell. Wenn '-' angegeben wird, so wird '*robot_description*' verwendet.

Tabelle 6.9: Konfiguration der Kalibrierung und Analyse

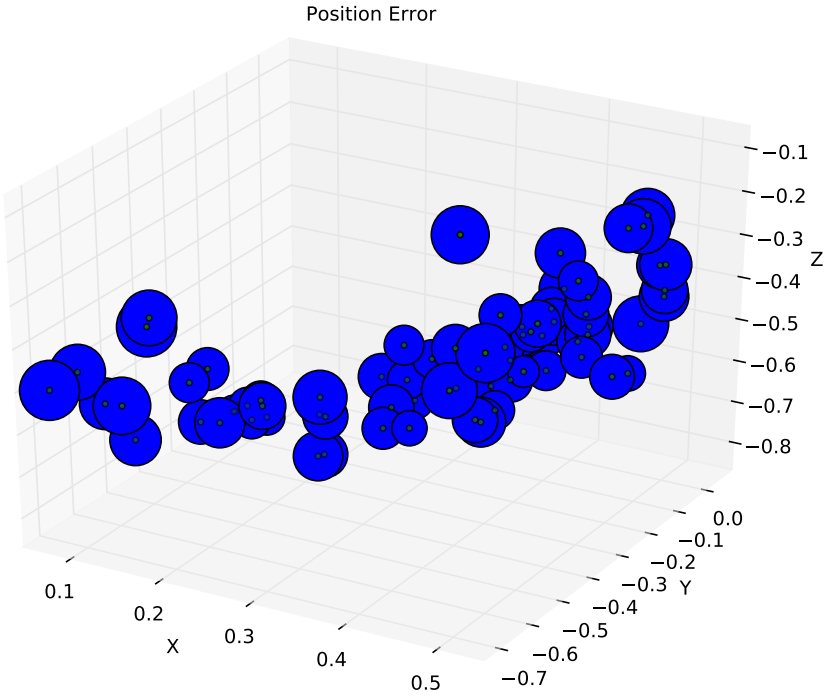
Feld	**Bedeutung**
analyse.*	Bereich für die Analyse
analyse.belief_chain	Die Kette mit der verglichen wird. Dies ist eine *.pkl*-Datei.
analyse.localized.*	Bereich für lokalisierte Modelle
analyse.localized.data_file	Die Ausgabedatei, wenn *null*, dann wird nachgefragt.
analyse.localized.iterations	Anzahl der Iterationen für jedes Modell.
...	

Tabelle 6.9: Konfiguration der Kalibrierung und Analyse (Fortsetzung)

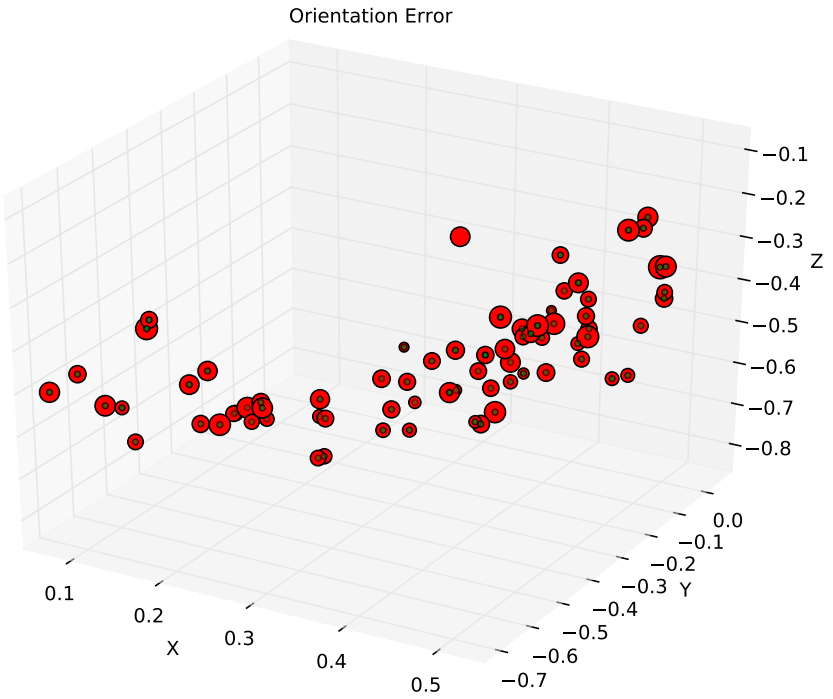
Feld	**Bedeutung**
analyse.localized.variance	Die Varianz bei der Gewichtung.
analyse.outdir	Das Ausgabeverzeichnis, wenn *data_file* nicht gesetzt ist.
analyse.run	Wenn gesetzt, dann wird der Analyseschritt durchgeführt, der dem Wert entspricht.
calibrate.*	Bereich für die Kalibrierung.
calibrate.learn_gear_ratio	Wenn *True* in Python, dann wird Getriebebeiwert gelernt.
calibrate.max_iter	Maximale Anzahl an Iterationsschritten.
calibrate.outdir	Ausgabeverzeichnis für die Kalibrierung.
calibrate.template_urdf	Alternative URDF-Datei die als Vorlage verwendet wird, in die die Kalibrierung eingetragen wird.
calibrate.template_urdf_prop	Alternative URDF-Property die als Vorlage verwendet wird, in die die Kalibrierung eingetragen wird.
calibrate_cameras	Wenn *True* in Python, dann werden die Kameras kalibriert.
camera_info.*	Der Bereich für die Kameras.
camera_info.*Kamera*.*	Der Bereich für Kamera '*Kamera*'.
camera_info.*Kamera*.D[]	Der Kalibrierungsvektor D.
camera_info.*Kamera*.K[]	Der Kalibrierungsvektor K.
camera_info.*Kamera*.P[]	Der Kalibrierungsvektor P.
camera_info.*Kamera*.R[]	Der Kalibrierungsvektor R.
camera_info.*Kamera*.override_bag	Wenn *True* in Python, dann wird die hier angegebene oder die berechnete Kalibrierung verwendet, sonst die aus dem während der Exploration aufgezeichneten Topic .
camera_links.*	Der Bereich für die Referenzrahmen der Kameras.
camera_links.*Kamera*	Referenzrahmen der Kamera '*Kamera*'.
cb_offset.*	Der Bereich für das Schachbrettmusterkoordinatensystem in URDF-Notation.
cb_offset.joint_name	Der Name des Gelenks, welches *tip_link* mit *link_name* verbindet.
cb_offset.link_name	Der Name des Segments. Wenn dieser *tip_link* entspricht, so wird das Koordinatensystem aus dem URDF-Modell gelesen und die anderen hier angegebenen Werte werden ignoriert.
cb_offset.origin[]	Der URDF-Versatz.
cb_offset.rpy[]	Die URDF-Verdrehung.
...	

Tabelle 6.9: Konfiguration der Kalibrierung und Analyse (Fortsetzung)

Feld	**Bedeutung**
dh_drop_rates[]	Der Startwert der Verlangsamungsrate für Gradientenverfahren, die diesen benutzen.
dh_learn_rates[]	Der Startwert für die Lernrate der Gradientenverfahren.
dh_sigma_m[]	Die Varianzen der DH-Parameter für MAP: θ_1, d_1, a_1, α_1, $Getriebebeiwert_1$, θ_2,...
model_scale	Die Kantenlänge eines jeden Schachbrettfelds.
read_gear_ratios	Wenn undefiniert oder *True* in Python, dann wird Getriebebeiwert aus dem URDF-Modell ausgelesen.
sensor_root_link	Der gemeinsame Bezugsrahmen der Kameras.
simple_outdirs	Wenn *False* in Python, dann werden Ausgabeverzeichnisnahmen benutzt, welche Datumseinträge beinhalten.
solver_class	Die Klasse des Gradientenverfahrens, siehe *solver.py*, standarmässig Rprop.
tip_link	Das URDF-Segment des Endeffektors.
update_prior	Anzahl der Iterationen, nachdem das a priori Modell durch das a posteriori ausgetauscht wird, 0 zum Deaktivieren.
vec7[]	Die Varianzen der Schachbrettpose für MAP, diese besteht aus Position unt Quaternion: x, y, z, X, Y, Z, W.
world_frame	Das Segment der Kette, welches bei der Visualisierung unbewegt bleibt.



(a) Position



(b) Orientierung

Abbildung 6.6: Fehler in 3D

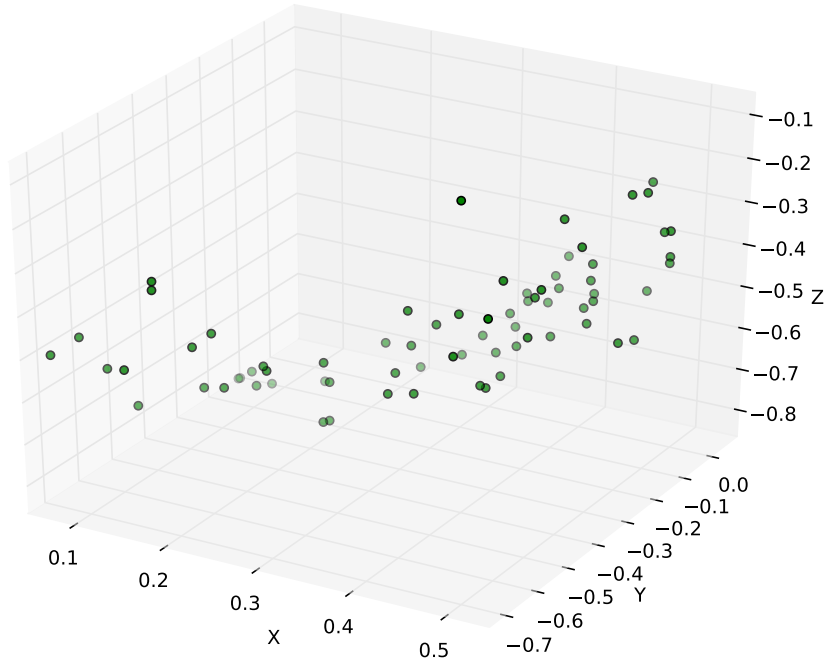


Abbildung 6.7: Endeffektorpositionen in 3D

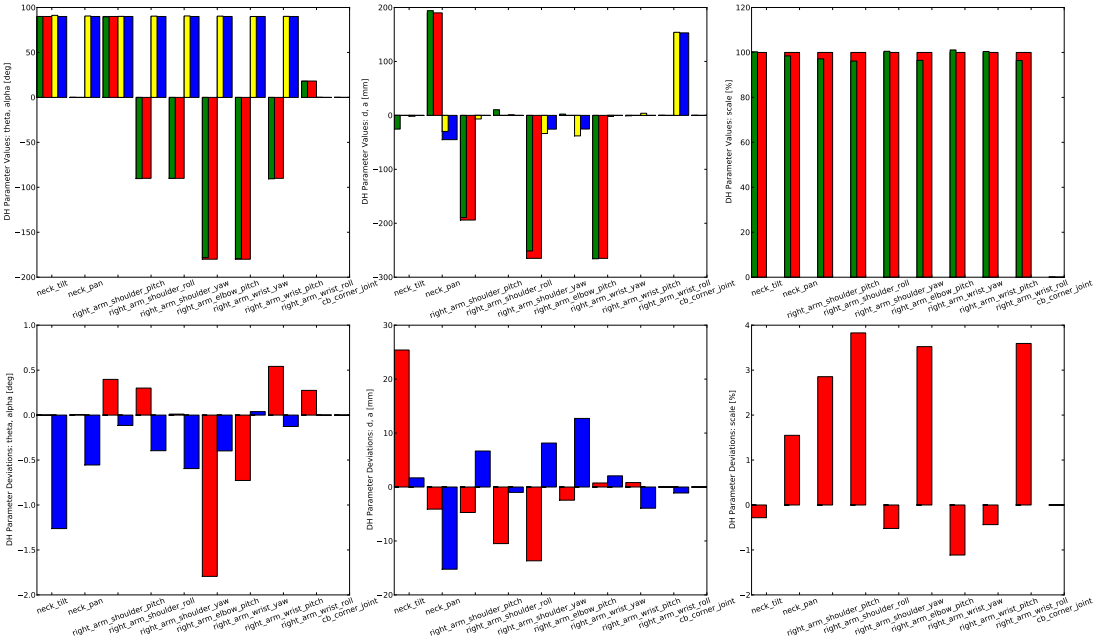


Abbildung 6.8: Vergleich zweier Ketten

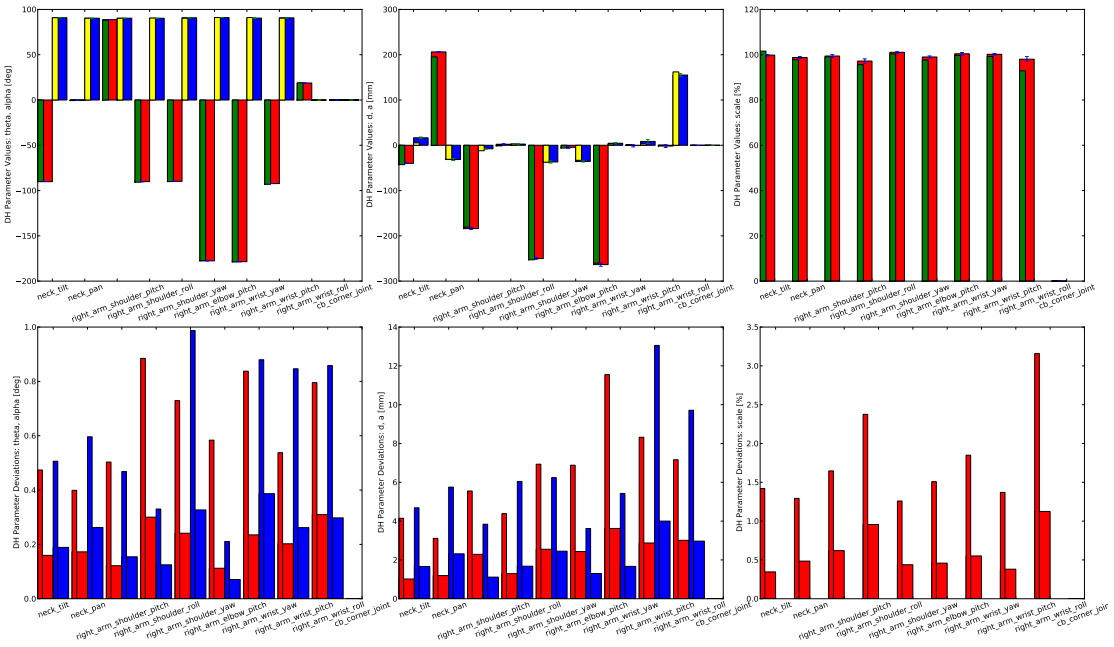


Abbildung 6.9: Parameterverteilung bei lokalisiertem Lernen

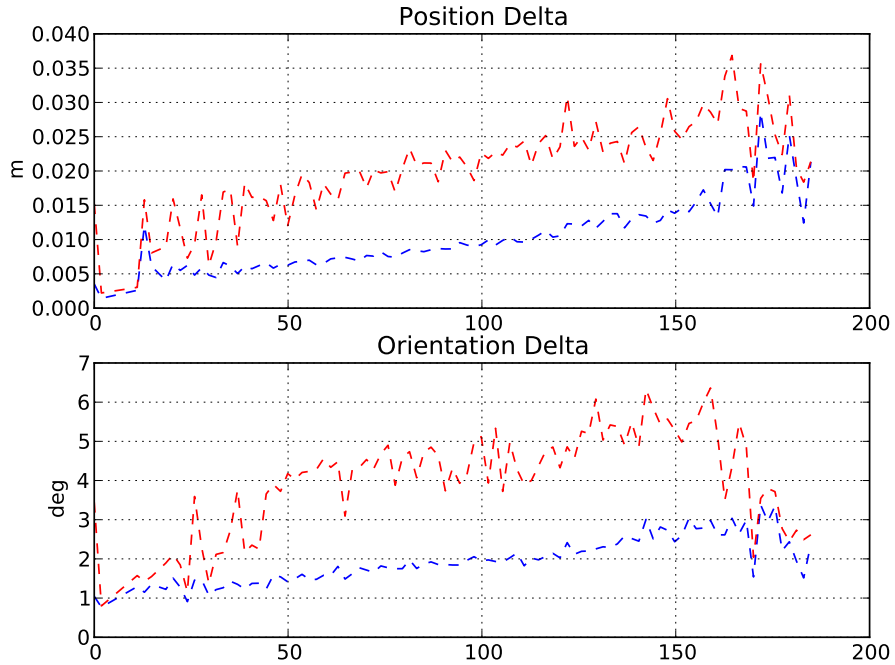
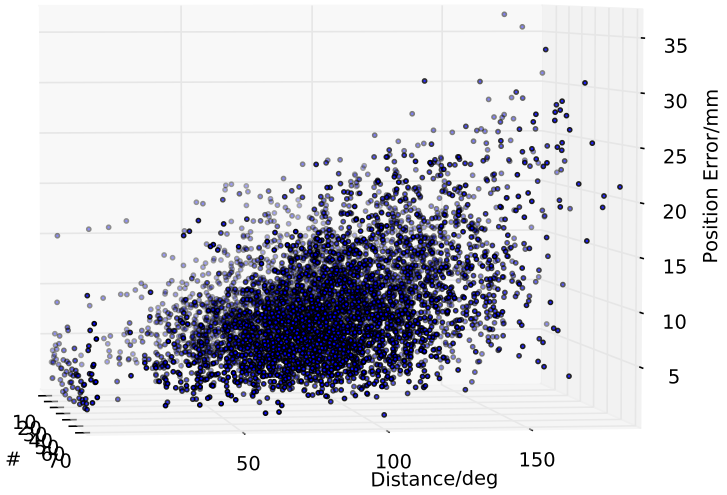
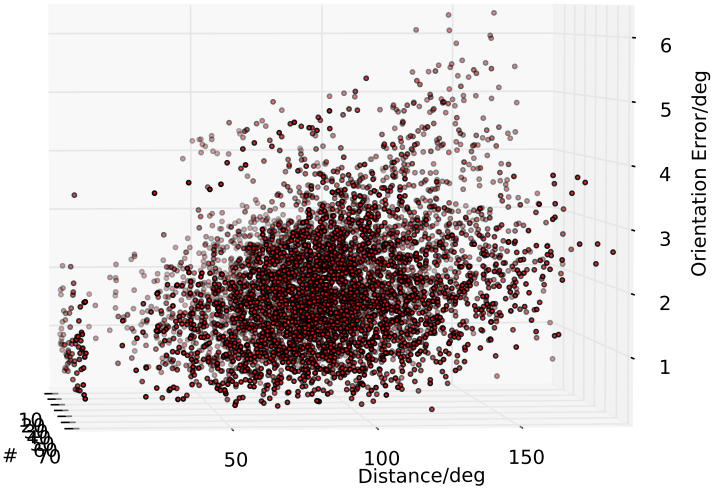


Abbildung 6.10: Fehler bei lokalisiertem Lernen (blau - Mittelwert, rot - Maximum)



(a) Position



(b) Orientierung

Abbildung 6.11: Fehler bei lokalisiertem Lernen

6.3 Evaluation

Im Paket *hubert_calib_evaluation* befinden sich Komponenten zur Evaluation dieser Arbeit. Das mit dieser Applikation durchgeführte Experiment besteht aus dem Ausrichten eines an der Hand angebrachten Lasers. Dieser wird auf beliebig vorgegebene Kreuzungspunkte eines Testschachbretts greichtet, so dass er senkrecht auf das Brett fallen soll. Es wird dann die Abweichung zwischen Laserpunkt und Kreuzungspunkt manuell gemessen. Das Programm *eval.py* erzeugt eine Benutzeroberfläche, mit der ein Test gesteuert wird, siehe Abbildung 6.12. Mit der Oberfläche kann die Nachverfolgung des

Abbildung 6.12: eval.py

Schachbretts gestartet und gestoppt werden und die IK berechnet werden. Es können die Kreuzungspunkte des Schachbretts, welche dann mit dem Laser angefahren werden, ausgewählt werden. Zudem hat man die Möglichkeit, drei fest vorgegebene Armstellungen anzusteuern, die Ruhestellung, eine erhobene Stellung seitwärts vom Rumpf und eine erhöhte Stellung vor dem Roboter.

Mit *cosero_evaluation.launch* wird ein Evaluationslauf auf dem realen Cosero gestartet. Um die Simulation dieses Tests zu starten wird, *cosero_evaluation_sim.launch* verwendet. Dieses Paket benutzt die gleiche Infrastruktur wie die Exploration aus Kapitel 6.1. Das Format der Konfigurationsdatei ist in Tabelle 6.10 beschrieben.

Tabelle 6.10: Konfiguration der Evaluation

Feld	**Bedeutung**
arm_controler.*	Die Konfiguration des Arms.
arm_controler.board_pose[]	Einelementige Trajektorie zum Anfahren einer Endeffektorstellung über dem Tisch.
...	

Tabelle 6.10: Konfiguration der Evaluation (Fortsetzung)

Feld	**Bedeutung**
arm_controler. board_pose[0].*	Die Zielkonfiguration.
arm_controler.board_pose[0]. duration	Erlaubte Zeitdauer.
arm_controler.board_pose[0]. positions[]	Die gewünschten Gelenkstellungen.
arm_controler.convini- ent_pose[]	Einelementige Trajektorie zum Anfahren der Ruhestellung.
arm_controler.convini- ent_pose[0].*	Die Zielkonfiguration.
arm_controler.convini- ent_pose[0].duration	Erlaubte Zeitdauer.
arm_controler.convini- ent_pose[0].positions[]	Die gewünschten Gelenkstellungen.
arm_controler.joint_names[]	Die Namen der Gelenke.
arm_controler.side_pose[]	Einelementige Trajektorie zum Anfahren einer Armstellung seitwärts vom Roboter.
arm_controler.side_pose[0].*	Die Zielkonfiguration.
arm_controler.side_pose[0]. duration	Erlaubte Zeitdauer.
arm_controler.side_pose[0]. positions[]	Die gewünschten Gelenkstellungen.
arm_controler.topic	Der Name des Topics für die Steuerung.
arm_controler.type	Für die Steuerung des Kopfes mit dem Robotcontrol *nimbro_msgs.msg.BodyPartsCommand*, für die Steuerung der Arme mit dem Robotcontrol *control_msgs.msg.FollowJointTrajectoryAction*, für die Steuerung der Arme des PR2 *trajectory_msgs.msg.JointTrajectory* und *None*, falls abgeschaltet.
camera.*	Die Konfiguration der Kamera.
camera.cam_info_topic	Der Name des Kamerainfotopics.
camera.camera_info.*	Die Konfiguration der Kamera.
camera.camera_info.D[]	Der Kalibrierungsvektor D.
camera.camera_info.K[]	Der Kalibrierungsvektor K.
camera.camera_info.P[]	Der Kalibrierungsvektor P.
camera.camera_info.R[]	Der Kalibrierungsvektor R.
...	

Tabelle 6.10: Konfiguration der Evaluation (Fortsetzung)

Feld	**Bedeutung**
camera.camera_info.override_bag	Wenn *False* in Python, dann wird die Kalibrierung aus dem Kamerainfotopics genommen.
camera.cb_detector.*	Hier wird der Schachbrettmusterdetektor *image_cb_detector* für dieses Muster konfiguriert.
camera.cb_detector.config	Namensraum des *SimpleActionClient* des Detektors.
camera.cb_detector.height_scaling	Vertikaler Vergrößerungsfaktor zur Verbesserung der Mustererkennung.
camera.cb_detector.num_x	Anzahl der Reihen der Kreuzungspunke.
camera.cb_detector.num_y	Anzahl der Spalten der Kreuzungspunke.
camera.cb_detector.subpixel_window	Konfiguration für *FindCornerSubPix()* in OpenCV [Ope12a].
camera.cb_detector.subpixel_zero_zone	Konfiguration für *FindCornerSubPix()* in OpenCV [Ope12a].
camera.cb_detector.width_scaling	Horizontaler Vergrößerungsfaktor zur Verbesserung der Mustererkennung.
camera.features_topic	Das Topic mit den erkannten Kreuzungspunkten.
camera.link	Der Name des Kamerakoordinatensystems.
camera.settler.*	Hier wird der Settler *monocam_settler* für das Muster konfiguriert.
camera.settler.cache_size	Die Größe des Nachrichtenpuffers.
camera.settler.config	Namensraum des *SimpleActionClient* des Detektors.
camera.settler.ignore_failures	Wenn *True*, dann werden fehlerhafte Meldungen gelöscht.
camera.settler.max_step	Maximaler Zeitabstand in einem Intervall.
camera.settler.tolerance	Die maximale Toleranz.
camera.settler_topic	Das Topic des Settlers.
eval.ora	Alternative URDF-Datei.
joint_states_topic	Der Name des Topics, auf dem die Gelenkstellungen publiziert weden.
model_scale	Die Kantenlänge eines jeden Schachbrettfelds.
timeout	Zeitraum, nachdem eine erfolglose Aktion abgebrochen wird.
tip_link	Das Segment des Arbeitspunkts des Lasers.
world_frame	Der Startpunkt für die Kette des Arms.

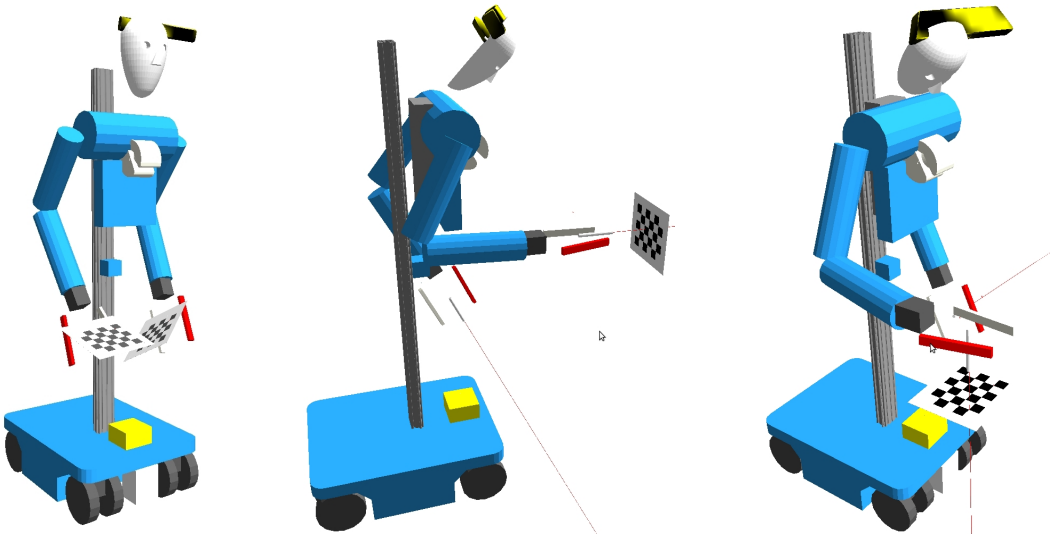
6.4 Nachrichten

Das Paket *hubert_calib_msgs* enthält unter *msg* lediglich die Beschreibungen der *hubert_calib_msgs/CalibrationMeasurement* Nachrichten, welche zur Publikation der Kalibrierungsrohdaten verwendet werden. Sie dient als Ersatz für die Nachricht *calibration_msgs/RobotMeasurement* aus dem Stack *pr2_calibration* [pr212b]. Jede Nachricht enthält ein Array für die Speicherung der Gelenkzustände und eines für die zugehörigen Schachbrettpunkte mitsamt der Kalibrierungsinformationen der Kameras.

6.5 Betrieb der Hardware

Das Paket *hubert_calib_real* bündelt Dateien, die für den Betrieb der realen Roboter benötigt werden. Die Startdatei *cosero_real.launch* startet das *Robotcontrol* wobei alle Gelenke aktiviert sind. Die Gelenke der Arme sind in *cosero_real_passive.launch* auf neutral geschaltet. In beiden Startdateien wird der *robot_state_publisher* und der Knoten für die Kinect-Kamera gestartet. Um die Kinect-Kamera zu kalibrieren, kann die Startdatei *openni_calibrate.launch* verwendet werden.

6.6 Simulation



(a) Exploration (b) Evaluation horizontal (c) Evaluation vertikal

Abbildung 6.13: Robotermodell

Das Paket *hubert_calib_simulation* beinhaltet sowohl Dateien für die Simulation der Roboter in Gazebo als auch URDF-Modelle und Texturen. Die Startdatei *cosero_services.launch* startet die Simulation, das *Robotcontrol* und den *robot_state_publisher*. Die

Datei *pr2_simulation.launch* übernimmt die gleiche Aufgabe für die Initialisierung des PR2.

Unter *urdf* befinden sich die Robotermodelle. Für Dynamaid und Cosero gibt es Modelle für die Evaluation und die simulierte Datenaufnahme. In *cosero/cosero_exp.urdf.xacro* und *dynamaid/dynamaid_exp.urdf.xacro* wurden Schachbretter an den Greifern angebracht. In Abbildung 6.13(a) ist dies für Cosero dargestellt.

In den Modellen *dynamaid/checkerboard_eval_h.urdf.xacro* und *cosero/checkerboard_eval_h.urdf.xacro* sind Laserpointer in den Greifern horizontal angebracht. Außerdem ist vor dem Roboter ein Schachbrett vertikal plaziert, siehe Abbildung 6.13(b) für Cosero. In den Modellen *dynamaid/checkerboard_eval_v.urdf.xacro* und *cosero/checkerboard_eval_v.urdf.xacro* sind Laserpointer vertikal montiert, und das Schachbrett horizontal, siehe Abbildung 6.13(c) für Cosero. Die Laserpointer sind relativ zum Koordinatensystem des Schachbretts der Exploration angebracht. Dies ist in *pr2/pr2_exp.urdf.xacro* auch für den PR2 vorgenommen worden.

Im Verzeichnis *textures* befinden sich die Druckvorlagen für die Schachbrettmuster. Die Datei *cb6x5_25mm.pdf* enthält ein 6×5 á 25 mm Muster, *cb21x11_10mm.pdf* enthält eines mit 21×11 á 10 mm.

6.7 Tests

In *hubert_calib_tests* befinden sich die Komponenten für die in Kapitel 7 beschriebenen Tests. Diese werden dort im Einzelnen erläutert. Zu finden sind hier URDF-Modelle, Startscripte, Protokolle und Ergebnisse von Explorationsläufen.

6.8 KDL-Wrapper

Die KDL stellt nicht ihre komplette Schnittstelle für Python zur Verfügung. Deswegen musste die Pythonschnittstelle erweitert werden. Hierfür wurden die SIP-Quellen [SIP12] der KDL in das Paket *hubert_kdl_wrapper* kopiert und angepasst. So wurde der Zugriff auf kinematische Bäume möglich. Zudem wurde hier die Möglichkeit geschaffen, in Python den *kdl_parser* [kdl12c] zu verwenden. Für die Pythonschnittstelle wurden Unittests erstellt, welche von *rostest* verwendet werden können, siehe Kapitel 6.9.

6.9 Modultester

Im Paket *hubert_unit* befindet sich in *pyunit.py* eine Modifikation der gleichnamigen Datei aus dem Paket *rosunit*. Sie dient dem Starten von Unittestsuites durch rostest [ros12b]. Die Tests werden als Schritt beim Bauen der Pakete ausgeführt.

6.10 Hilfsmodule

Das Paket *hubert_utils* nimmt allgemein verwendete Komponenten auf. In *math_utils.py* sind häufig verwendete mathematische Funktionen und Konstanten untergebracht. Das Modul *config.py* enthält eine Klasse für den erleichterten Zugriff auf Konfigurationsdaten. In *store.py* ist eine Klasse zur einfachen Datenspeicherung enthalten. Und in *misc.py* befinden sich Funktionen aus verschiedenen, schwach vertretenen Bereichen.

6.11 DH-Konverter

Das Paket *nimbro_dh_converter* beinhaltet den modifizierten DH-Converter von Jörg Stückler, siehe hierzu auch Kapitel 4.8. Dieser wurde von der GNU Scientific Library [Gnu12] auf die Bibliothek Eigen [Eig12] umgestellt, da diese zum Standardumfang von ROS gehört. Der Konverter steht jetzt als dynamische Bibliothek anderen Programmen zur Verfügung. Mit dem Wrappergenerator SIP [SIP12] wurde ein Zugang für Pythonprogramme geschaffen. Zur Absicherung der Funktionalität der C++ Schnittstelle wurden Unittests mit dem Google C++ Testing Framework [GTe12] erstellt, welche von *rostest* aufgerufen werden können. Für die Pythonschnittstelle wurden ebenfalls Unittests erstellt, welche von *rostest* verwendet werden können, siehe Kapitel 6.9.

6.12 Halterung

Für die Exploration und die Evaluation wurde am Institut eine Halterung für das Schachbrett und für Laserpointer erstellt. Siehe hierzu Abbildung 6.14. Die Basis der Halterung

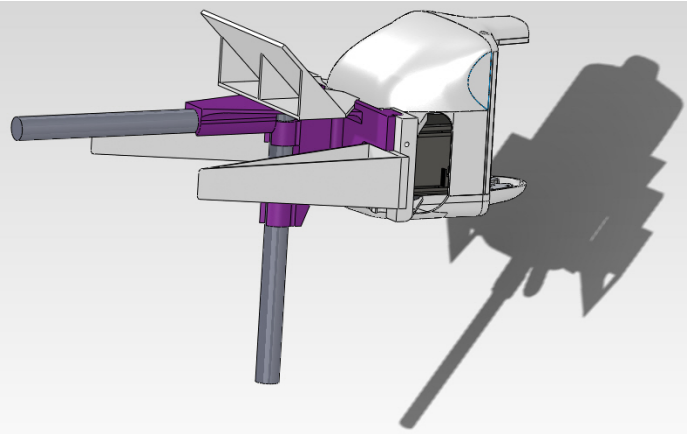


Abbildung 6.14: Halterung

ist lila eingezeichnet, sie bietet die Möglichkeit der horizontalen und vertikalen Befestigung von Laserpointern. Auf der Basis ist die grau eingezeichnete Halterung für die Tafel mit dem Schachbrettmuster zu sehen. Die montierte Tafel steht im 45° Winkel ab.

Die vertikale Konfiguration wird nicht verwendet. Die manuell gemessene Lage zwischen Schachbrettmuster und horizontal angebrachtem Laserpointer, beziehungsweise dessen Strahl, ist in den URDF-Modellen eingetragen.

7 Experimente

In diesem Kapitel wird das entwickelte Verfahren und die Implementation evaluiert. Zuerst werden Referenzwerte für Messfehler abgeschätzt. Danach werden mit diesen Fehlern künstliche Trainings- und Testdatensätze erzeugt, mit denen dann die Kalibrierung getestet wird. Im Anschluss wird die Ausrichtung eines Laserstrahls auf ein Schachbrett simuliert. Dabei werden fehlerfreie, fehlerbehaftete und optimierte Modelle getestet. Danach werden Messreihen von den realen Robotern betrachtet. Darauf folgt die Ermittlung der Wiederholgenauigkeit der realen Roboter und die Ausrichtung eines realen Lasers auf ein Schachbrett. Danach betrachten wir das lokalisierte Lernen.

Im Folgenden werden häufig die Differenzen zwischen gegebenenen Posen und erwarteten Posen berechnet. Die Eingabe hierfür ist eine Menge von gemesenen oder anderwärtig vorgegebenen Endeffektorposen und den zugehörigen Gelenkstellungen. Aus jeder Glenkkonfiguration wird mit dem jeweiligen Modell die erwartete Endeffektorpose berechnet. Diese wird mit der gegebenen Endeffektorpose verglichen. Dabei werden die Postitions- und Orientierungsdifferenzen berechnet. Von diesen wird dann der Mittelwert, das quadratische Mittel (Effektivwert) und das Maximum bestimmt.

Alle Testdaten liegen im ROS-Paket *hubert_calib_tests*. Wenn bei Dateien kein Paket expliziet erwähnt wird, so liegen sie dort.

7.1 Referenzwerte für die Posenbestimmung

In diesem Abschnitt werden Referenzwerte für die Abschätzung der Messfehler bei der Posenbestimmung von Schachbrettmustern gewonnen.

7.1.1 Manuell

Um Kennzahlen für die monoskopische Posenbestimmung mit OpenCV zu erlangen wird ein Schachbrettmuster in verschiedenen Posen auf einem Stativ fixiert. Die Entfernungen der Linse zum Referenzkoordinatensystem des Schachbretts werden mit einem Bandmaß manuell und monoskopisch mit OpenCV gemessen. Es wird ein Schachbrettmuster mit 6×5 Feldern mit einer Größe von 10×10 mm und eine Logitech Pro9000 Kamera mit einer Auflösung von 1600×1200 Pixeln verwendet. Vor Beginn der Messungen wird die Kamera kalibriert. Die Ergebnisse der Messungen sind in Tabelle A.2 aufgeführt. Daraus ergeben sich die folgenden Kennzahlen.

Mittelwert [mm]	**Effektivwert [mm]**	**Maximum [mm]**
9,6470588235	9,9760959974	38

7.1.2 Simulation Cosero

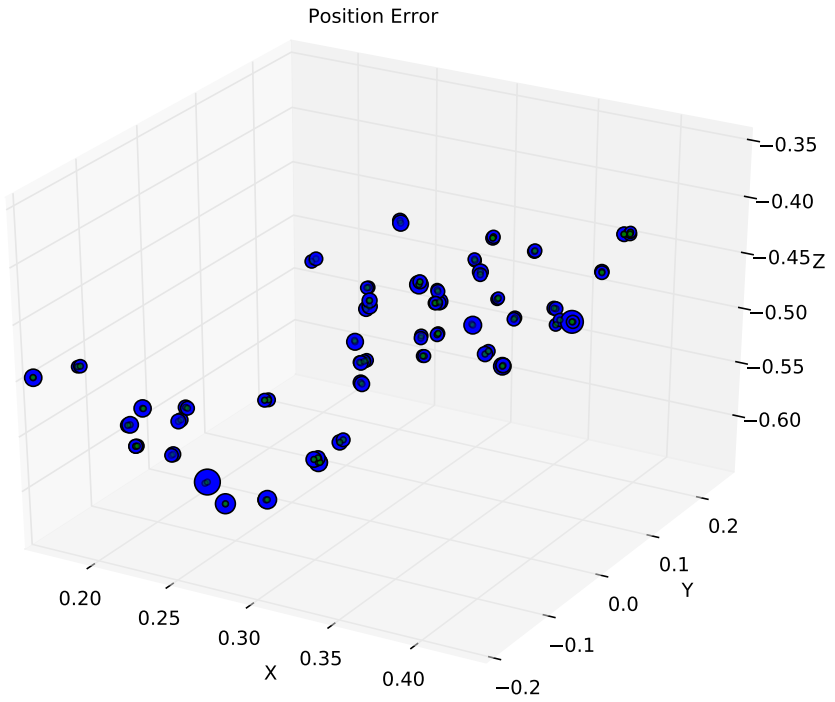


Abbildung 7.1: Fehler in der Simulation bei Cosero, Position

Um Referenzwerte für die Standardabweichung der Position und der Orientierung des Schachbretts in Bezug auf die VK zu bestimmen, wird für den Roboter Cosero eine Simulation durchgeführt. Die simulierte Kamera hat eine Auflösung von 640×480 Punkten und einen Öffnungswinkel von 60°. Der Kopf wird in dem Versuch automatisch auf das Schachbrett ausgerichtet. Die Anzahl der gemessenen Schachbrettposen beträgt 96. Da nur eine Kamera verwendet wird, entspricht dies der Anzahl der Armstellungen. Die Simulation wird mit *hubert_calib_exploration cosero_exploration_sim_right.launch* gestartet. Die gesammelten Daten werden in *bags/cosero_calibration_measurements-sim-2012-03-04.bag* abgelegt. Die Analyse der Daten wird mit *test000_cosero_analyse.launch* gestartet.

Man sieht in den Abbildungen 7.1 und 7.2, dass die Fehler gleichmässig im Raum verteilt sind. Der Programmpunkt *calc_errors* der Analyse berechnet die folgenden Kennzahlen. Vergleiche hierzu auch Kapitel 6.2.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	2,098937	2,233236	6,648053
Orientierung [°]	0,567729	0,752712	2,476247

Um den Einfluss einer nicht optimalen Kamerakalibrierung zu betrachten, werden die beschriebenen Berechnungen abermals mit vorgeschalteter Kalibrierung der Kamera

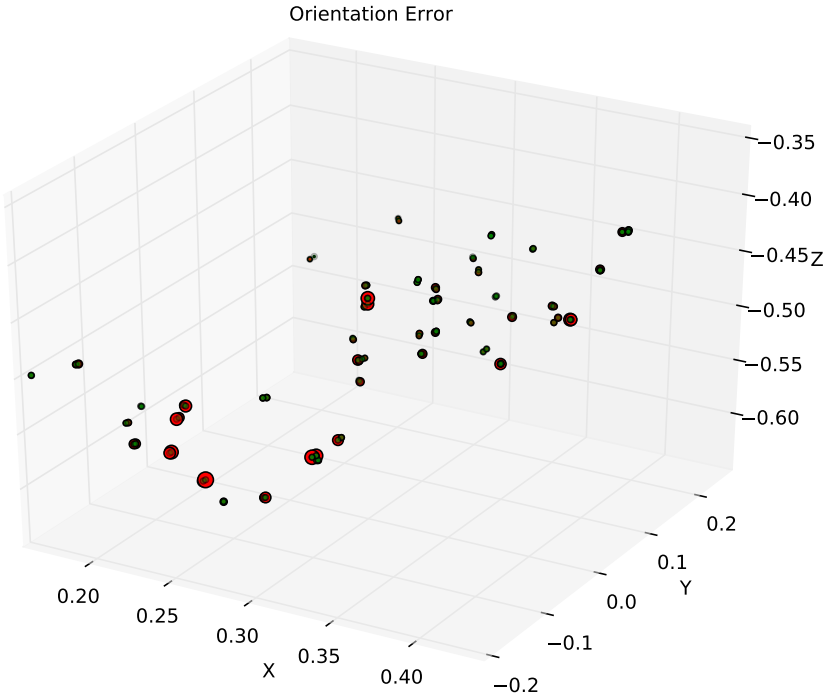


Abbildung 7.2: Fehler in der Simulation bei Cosero, Orientierung

durchgeführt, siehe Kapitel 6.2. In den verwendeten Aufnahmen decken die Schachbrettmuster nicht das ganze Sichtfeld der Kamera ab, was aber für eine gute Kalibrierung erforderlich wäre. Es ergeben sich die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	12,403802	12,454166	15,671717
Orientierung [°]	1,068344	1,161103	3,216307

Die Positionsgeauigkeit verschlechtert sich um über 10 mm, der Fehler versechsfacht sich. Die Orientierungsgenauigkeit leidet deutlich weniger, sie reduziert sich ungefähr auf die Hälfte.

7.1.3 Simulation PR2

Der Versuch aus Kapitel 7.1.2 wird für den PR2 wiederholt. Beim PR2 werden fünf Kameras simuliert, eine Monokamera mit 2448 × 2050 Punkten und einem Öffnungswinkel von 45° und je zwei Stereokameras mit je 640 × 480 Punkten Auflösung und Öffnungswinkeln von 45° beziehungsweise 90°. Es werden bei 119 Armstellungen 368 Schachbrettposen aufgenommen. Jeweils 100 für die Normalwinkel-Stereokameras, 79 für die Weitwinkel-Stereokameras und 10 für die Monokamera. Gestartet wird die Exploration mit *hubert_calib_exploration pr2_exploration_sim.launch* und die Analyse mit *test000_pr2_analyse.launch*, die Daten werden in *bags/pr2_calibration_measurements-*

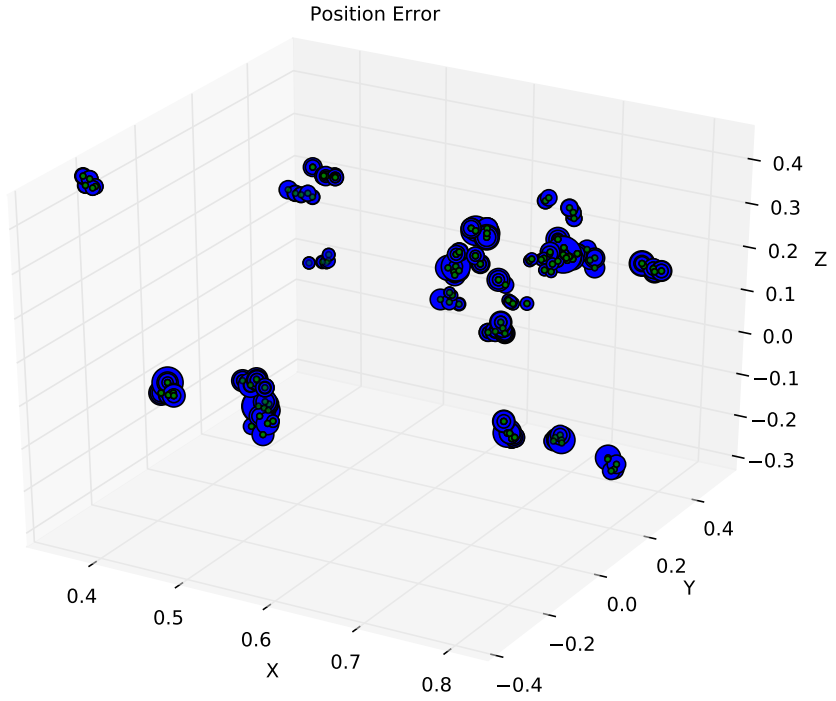


Abbildung 7.3: Fehler in der Simulation bei PR2, Position

2012-03-04.bag abgelegt.

Man sieht in den Abbildungen 7.3 und 7.4, dass die Fehler ungleichmässig im Raum verteilt sind. Der Programmpunkt *calc_errors* berechnet die folgenden Kennzahlen.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	2,830495	3,308671	13,442749
Orientierung [°]	1,221261	2,219403	19,456839

Um den Einfluss einer nicht optimalen Kamerakalibrierung auch hier zu betrachten, werden die beschriebenen Berechnungen abermals durchgeführt. Auch hier decken die Schachbrettmuster wieder nicht ganz die Sichtfelder der Kameras ab. Es ergeben sich folgende Kennzahlen.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,830347	12,341572	22,291258
Orientierung [°]	1,425033	2,324893	19,321906

Die Positionsgenauigkeit verschlechtert sich um ca. 9 mm, dieser Fehler vervierfacht sich. Die Orientierungsgenauigkeit leidet hier fast nicht.

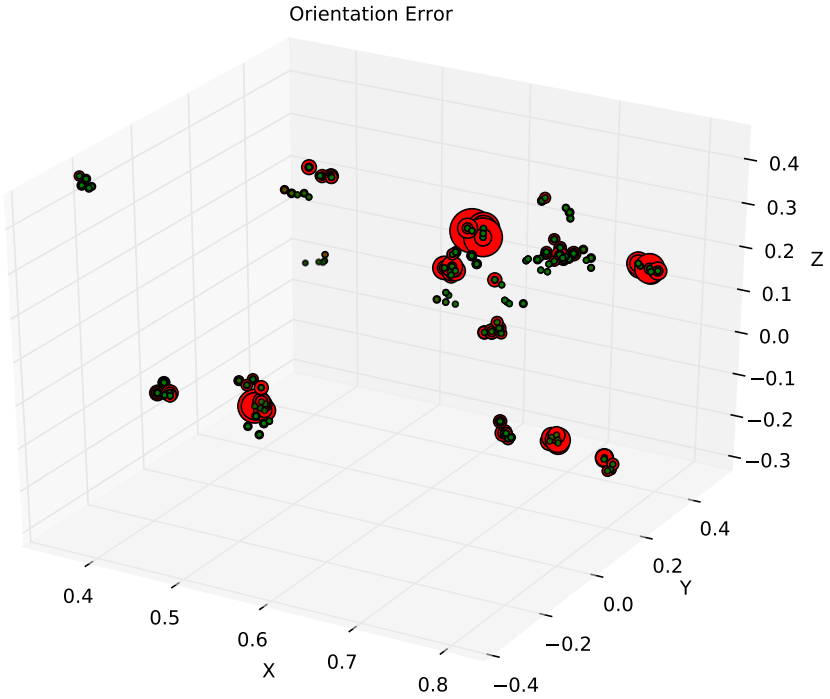


Abbildung 7.4: Fehler in der Simulation bei PR2, Orientierung

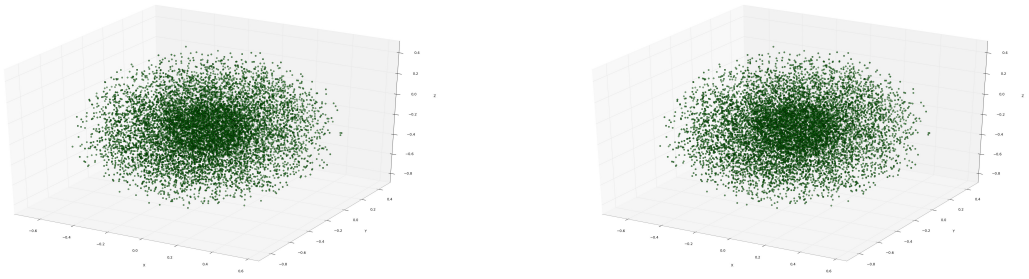
7.2 10000 im Gelenkraum gleichverteilte Stellungen

Im folgenden werden mit Trainingsdaten Kalibrierungen durchgeführt. Für die Modelle werden dann für einen Testdatensatz die Fehler berechnet. Zudem wird das Modell mit dem *realen* Modell verglichen. Für eine Beschreibung der darstellungen sieh auch Kapitel 6.2.

Zunächst wird ein fehlerfreier synthetischer Testdatensatz mit 10000 im Gelenkraum gleichverteilten Stellungen erzeugt. Dabei liegen die Gelenkstellungen jeweils im Intervall $[-\pi; \pi]$. Für einen groben Überblick über die Testpositionen siehe Abbildung 7.5(a). Hierfür werden mit *test001_synth.launch* aus *hubert_calib_simulation/urdf/cosero/cosero_exp.urdf.xacro* Daten generiert, die unter *test0001/data/synth-10000-noerr.pkl* abgelegt werden. Die Analyse, gestartet durch *test001_analyse_test.launch calc_errors*, bestätigt das Fehlen eines Fehlers gegenüber der erwarteten Pose der VK.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	0,000000	0,000000	0,000000
Orientierung [°]	0,000000	0,000000	0,000000

Ein Trainingsdatensatz mit 10000 gleichverteilten Stellungen wird erzeugt. Die Daten werden unter *test0001/data/synth-10000-err.pkl* abgelegt. Die Parameter für die Synthese sind dabei wie folgt, siehe auch Tabelle 6.5.



(a) Testpositionen (b) Trainingspositionen

Abbildung 7.5: 10000 Stellungen

Feld	**Wert**
synth.joint_std	0,005
synth.ef_std_pos	0,012
synth.ef_std_ori	0,04
synth.type	random
synth.random_count	10000

Die Fehler sind so gewählt worden, dass sie den zuvor bestimmten Referenzwerten entsprechen. Bei den Gelenkstellungen wurden Fehler in der Größenordnung der Aktuatorgenauigkeit gewählt. Für einen Überblick über die Positionen siehe Abbildung 7.5(b). Der Aufruf von *test001_analyse_test.launch calc_errors* ergibt die folgende Fehler, gegenüber der erwarteten Pose der VK.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	10,689365	12,738049	47,805202
Orientierung [°]	2,061540	2,444868	9,517290

Die Mittelwerte und Effektivwerte entsprechen den Bereichen aus den Kapiteln 7.1.1, 7.1.2 und 7.1.3. Es werden nun verschiedenartig abweichende Modelle mit diesen Daten optimiert und ihre Konvergenz gegen das reale Modell geprüft. In den Experimenten werden die folgenden Parameter für die Kalibrierung verwendet.

Feld	**Wert**
dh_learn_rates	Alle Werte 0,0001.
vec7	[0,001, 0,001, 0,001, 0,002, 0,002, 0,002, 0,002]

Die angenommenen Varianzen der Pose, *vec7*, entsprechen der Größenordnung der Fehler der Posenbestimmung.

7.2.1 Keine Abweichungen, 100 Iterationen, Varianz 0,01

Um den Einfluss der Fehler in den Messwerten auf das Ergebnis zu bestimmen, wird mit dem Trainingsdatensatz und einer nicht manipulierten Kette eine Kalibrierung durchge-

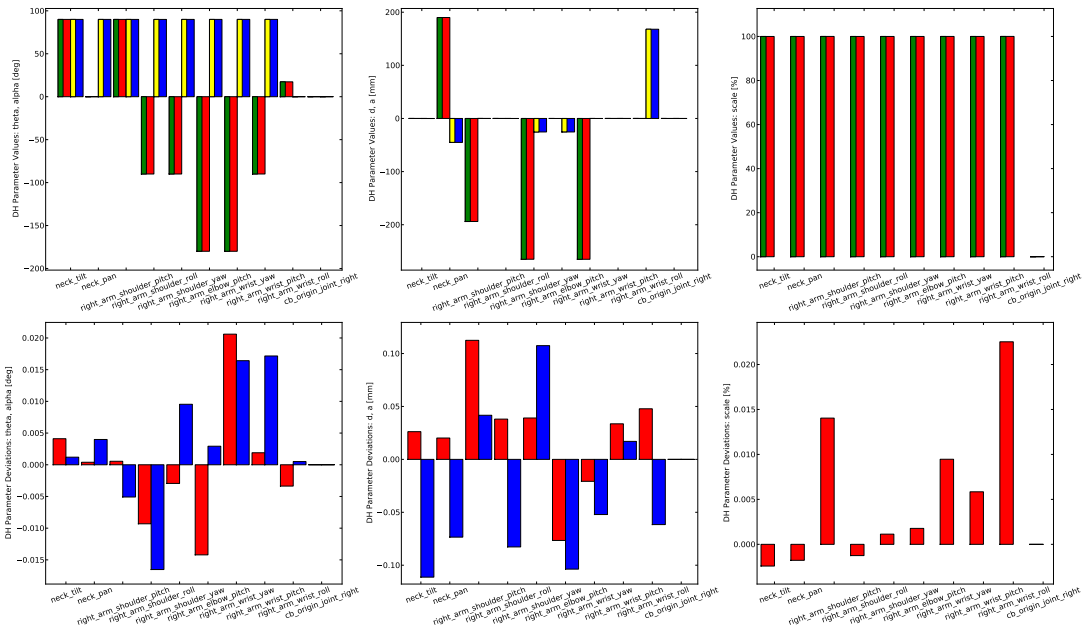


Abbildung 7.6: Keine Abweichungen, 100 Iterationen, Varianz 0,01

führt. Gestartet wird diese mit *test001_calib.launch*. Die Parameter für die Kalibrierung sind dabei wie folgt, siehe auch Tabelle 6.9.

Feld	**Wert**
calibrate.max_iter	100
update_prior	0
dh_sigma_m	Alle Werte 0,01.

Während der Kalibrierung wird die Entwicklung der Fehler angezeigt, wie in Abbildung 6.2 beschrieben. Hier zeigt sich keine sichtbare Entwicklung.

Die Anwendung von *test001_analyse_test.launch calc_errors* auf das resultierende Modell und den Testdatensatz ergibt die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	0,285121	0,307294	0,726871
Orientierung [°]	0,047275	0,051359	0,124763

Die Anwendung von *test001_analyse_training.launch calc_errors* auf das resultierende Modell und den Trainingsdatensatz ergibt die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	10,687756	12,733698	47,985081
Orientierung [°]	2,060477	2,443674	9,516370

Das Ergebnis des Starts von *test001_analyse_training.launch* und der Anwendung vom Programmpunkt *plot_diff_parameters* auf das a priori und das a posteriori Modell ist in Abbildung 7.6 zu sehen.

Es hat beim Kalibrieren keine signifikanten Änderungen am Modell gegeben.

7.2.2 Keine Abweichungen, 100 Iterationen, Varianz 1,0

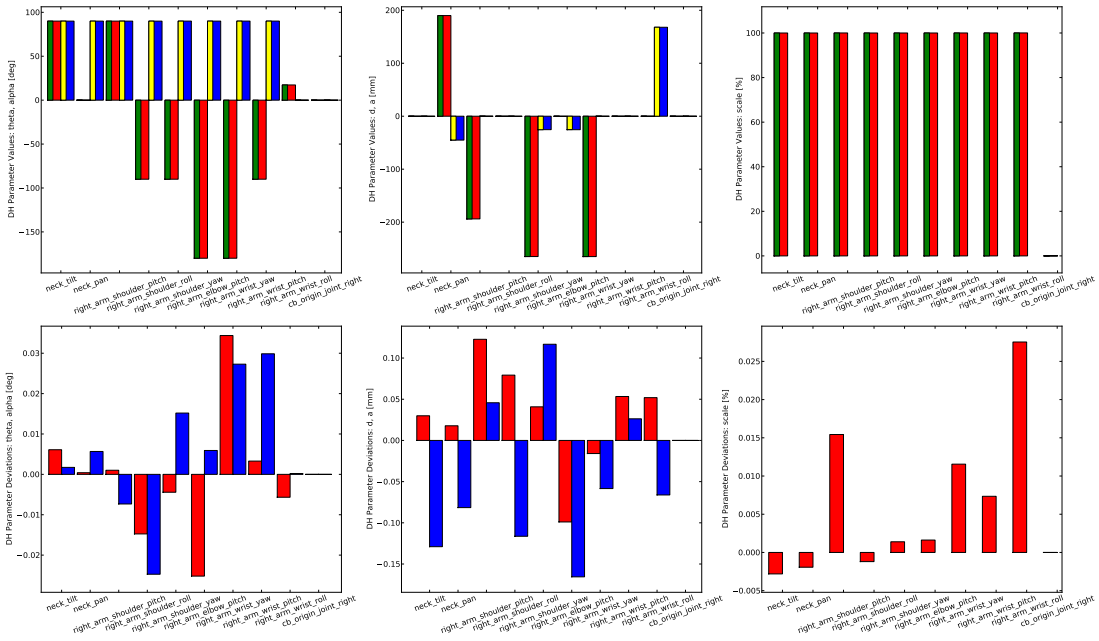


Abbildung 7.7: Keine Abweichungen - real vs. a posteriori, 100 Iterationen, Varianz 1,0

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.2.1. Jedoch werden folgende Änderungen an den Parametern der Kalibrierung vorgenommen.

Feld	**Wert**
calibrate.max_iter	100
update_prior	0
dh_sigma_m	Alle Werte 1,0.

Im Verlauf der Kalibrierung zeigt sich keine sichtbare Änderung der Fehler. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	0,348473	0,375803	0,916210
Orientierung [°]	0,071591	0,077589	0,180684

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	10,688618	12,733732	48,019326
Orientierung [°]	2,060476	2,443456	9,508678

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.7 zu sehen.

7.2.3 Keine Abweichungen, 200 Iterationen, Austausch alle 70, Varianz 0,1

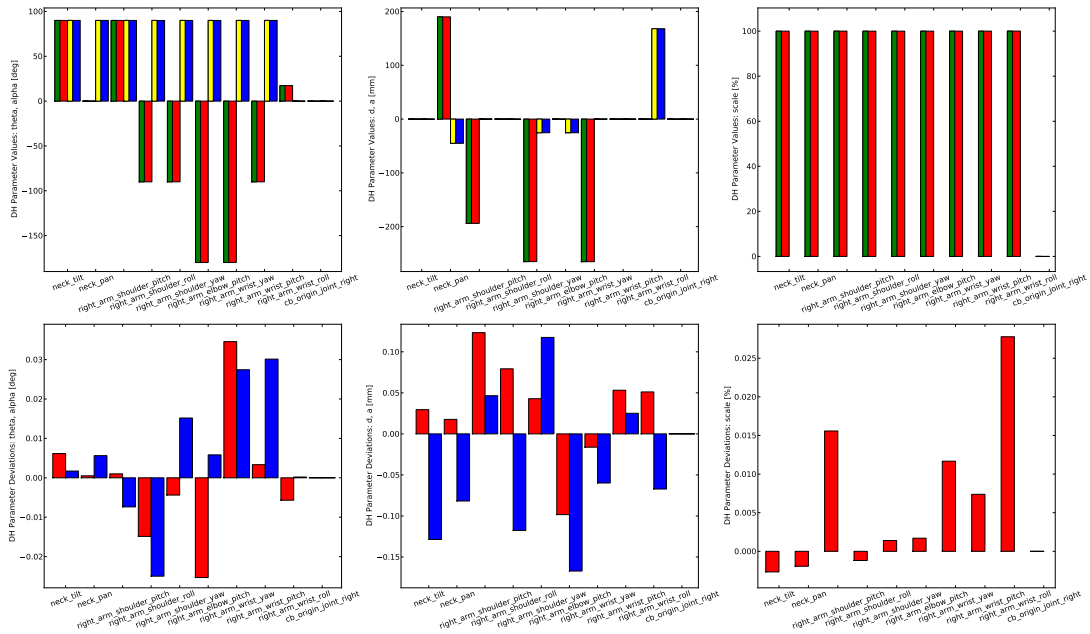


Abbildung 7.8: Keine Abweichungen - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.2.1. Jedoch werden folgende Änderungen an den Parametern der Kalibrierung vorgenommen.

Feld	**Wert**
calibrate.max_iter	200
update_prior	70
dh_sigma_m	Alle Werte 0,1.

Im Verlauf der Kalibrierung zeigt sich keine sichtbare Änderung der Fehler. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	0,350847	0,378342	0,923593
Orientierung [°]	0,072047	0,078088	0,182031

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	10,688661	12,733747	48,022219
Orientierung [°]	2,060480	2,443455	9,508615

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.8 zu sehen.

7.2.4 Abweichungen im Ellenbogen, 100 Iterationen, Varianz 0,01

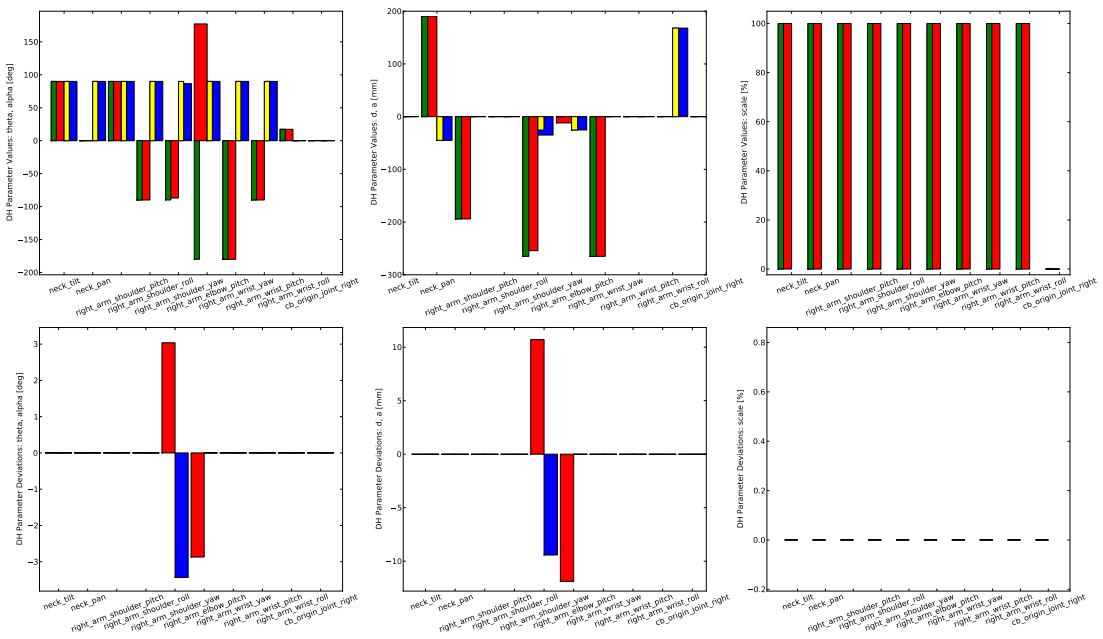


Abbildung 7.9: Abweichender Ellenbogen - real vs. a priori

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.2.1. Das a priori Modell *test0001/data/cosero_prior.urdf* gleicht bis auf die folgenden Änderungen dem *realen* Modell.

Gelenk	**Modell**	**xyz**	**rpy**
right_arm_elbow_pitch	real	0,0255 0 -0,265	0 0 0
	a priori	0,0355 -0,01 -0,255	0,06 -0,05 0,05

Die Auswirkungen der Manipulation der URDF-Parameter auf die DH-Parameter zeigen sich in Abbildung 7.9.

Das a priori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	28,881671	29,702053	46,205716
Orientierung [°]	5,358911	5,358911	5,358911

Das a priori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	30,937413	32,397023	88,056156
Orientierung [°]	5,724499	5,884556	12,635979

Die Kalibrierungsparameter sind wie folgt.

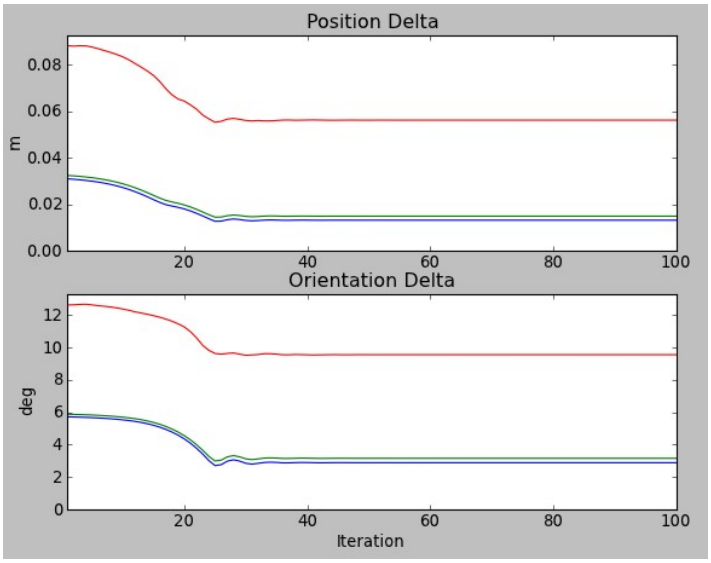


Abbildung 7.10: Fehlerentwicklung Ellenbogen
rot: Maximum, *blau*: Mittelwert, *grün*: Effektivwert

Feld	**Wert**
calibrate.max_iter	100
update_prior	0
dh_sigma_m	Alle Werte 0,01.

Lange vor dem Ende der Kalibrierung erreichen die Fehler ein Plateau, siehe 7.10. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	7,180191	7,662859	14,148244
Orientierung [°]	2,010043	2,010687	2,147314

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	13,173733	14,907680	56,113534
Orientierung [°]	2,888188	3,158824	9,565877

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.11 zu sehen.

Die Abstände zwischen realem Modell und dem optimierten Modell sind deutlich geringer als die Abstände zwischen dem manipulierten Ausgangsmodell und dem Realen. Bei den Parametern d und a haben sich Verschiebungen innerhalb der betroffenen Gelenke ergeben. In den anderen Gelenken sind keine signifikanten Abweichungen hinzu gekommen.

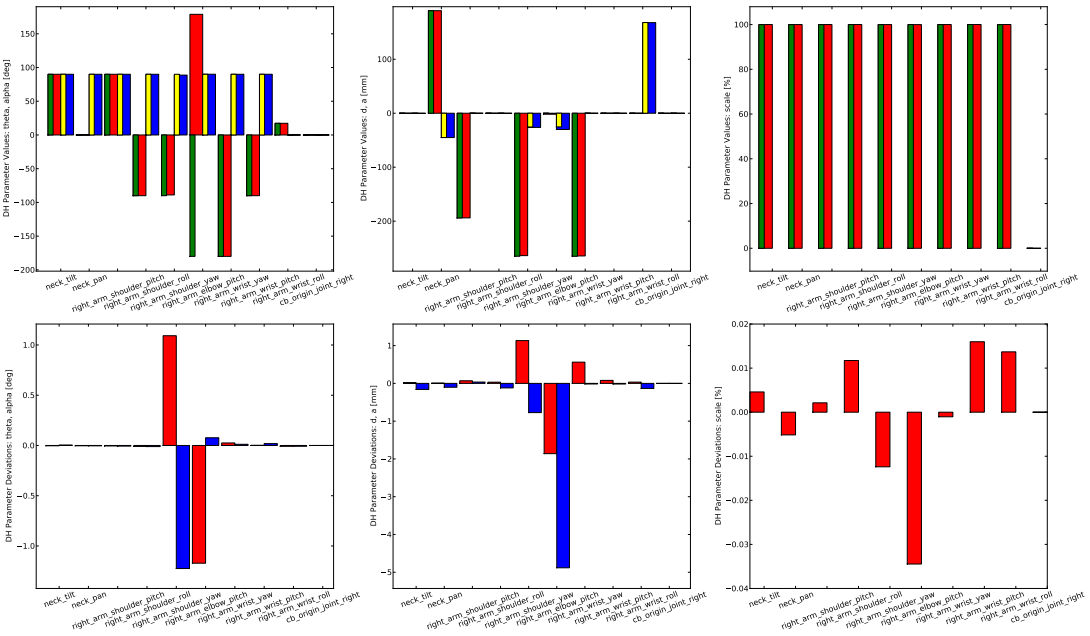


Abbildung 7.11: Abweichender Ellenbogen - real vs. a posteriori, 100 Iterationen, Varianz 0,01

7.2.5 Abweichungen im Ellenbogen, 100 Iterationen, Varianz 0,1

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.2.4. Es werden folgende Änderungen an den Parametern der Kalibrierung vorgenommen.

Feld	**Wert**
calibrate.max_iter	100
update_prior	0
dh_sigma_m	*Alle Werte 0,1.*

Die Fehler erreichen deutlich vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	1,093912	1,163700	2,419922
Orientierung [°]	0,315179	0,317649	0,452992

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	10,757153	12,789597	49,002551
Orientierung [°]	2,084708	2,461700	9,477694

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.12 zu sehen.

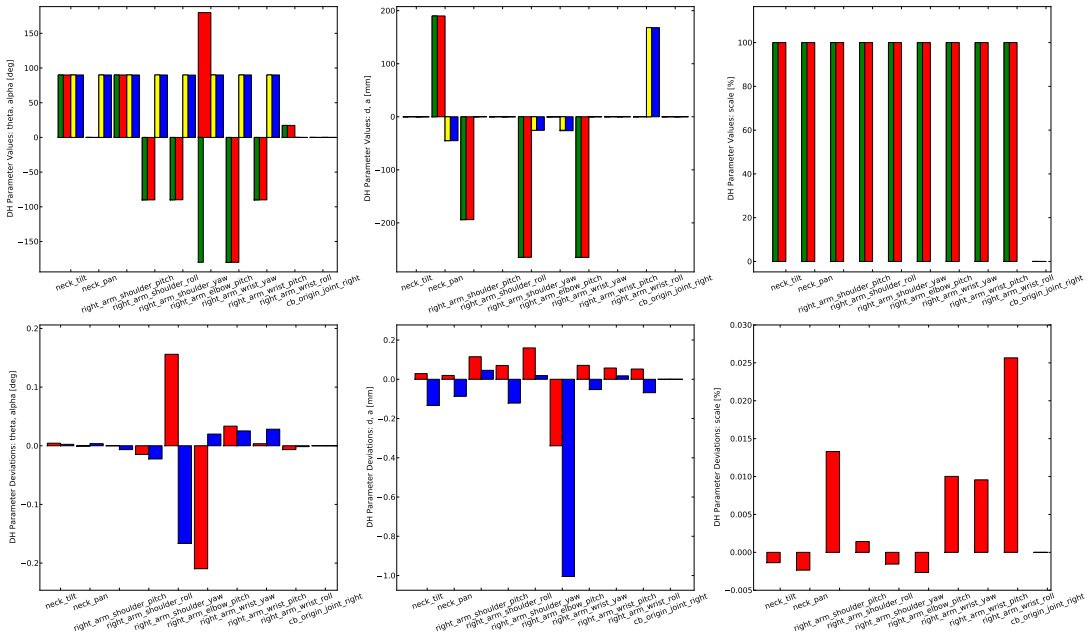


Abbildung 7.12: Abweichender Ellenbogen - real vs. a posteriori, 100 Iterationen, Varianz 0,1

7.2.6 Abweichungen im Ellenbogen, 200 Iterationen, Varianz 0,1

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.2.4. Es werden folgende Änderungen an den Parametern der Kalibrierung vorgenommen.

Feld	**Wert**
calibrate.max_iter	200
update_prior	0
dh_sigma_m	Alle Werte 0,1.

Die Fehler erreichen deutlich vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	1,093925	1,163713	2,419396
Orientierung [°]	0,315182	0,317652	0,452957

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	10,757149	12,789598	49,001750
Orientierung [°]	2,084708	2,461700	9,477688

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.13 zu sehen.

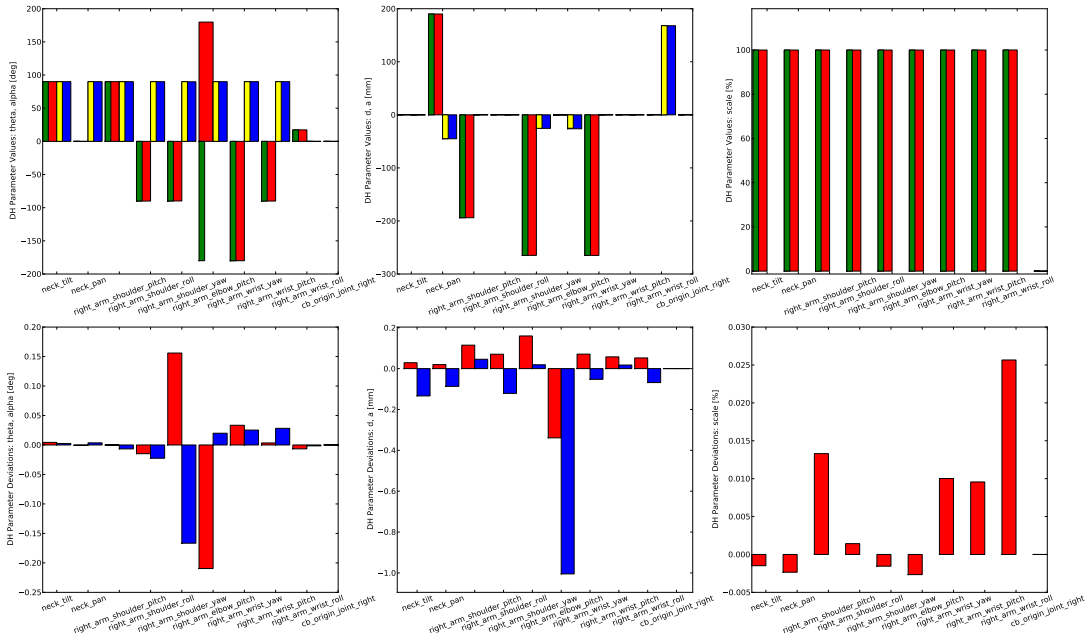


Abbildung 7.13: Abweichender Ellenbogen - real vs. a posteriori, 200 Iterationen, Varianz 0,1

7.2.7 Abweichungen im Ellenbogen, 200 Iterationen, Austausch alle 70, Varianz 0,01

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.2.4. Es werden folgende Änderungen an den Parametern der Kalibrierung vorgenommen.

Feld	**Wert**
calibrate.max_iter	200
update_prior	70
dh_sigma_m	Alle Werte 0,01.

Die Fehler erreichen deutlich vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	1,040486	1,092178	2,190297
Orientierung [°]	0,311811	0,314266	0,451238

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	10,746159	12,781349	48,791412
Orientierung [°]	2,083626	2,460840	9,449294

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.14 zu sehen.

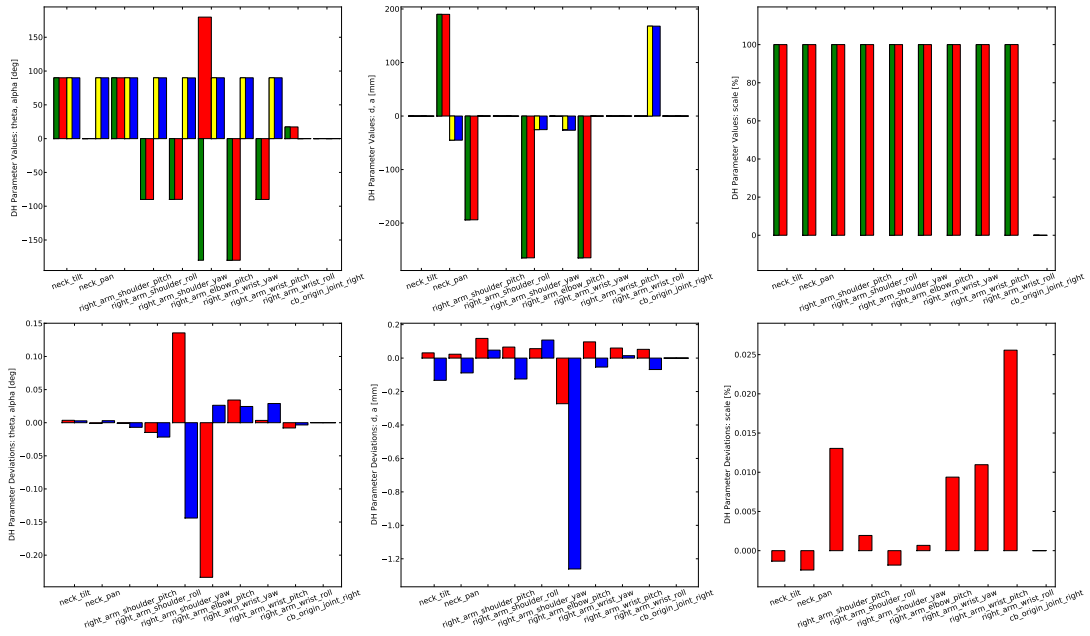


Abbildung 7.14: Abweichender Ellenbogen - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,01

7.2.8 Abweichungen im Ellenbogen, 200 Iterationen, Austausch alle 70, Varianz 0,1

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.2.4. Es werden folgende Änderungen an den Parametern der Kalibrierung vorgenommen.

Feld	**Wert**
calibrate.max_iter	200
update_prior	70
dh_sigma_m	*Alle Werte 0,1.*

Die Fehler erreichen deutlich vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	0,350483	0,377922	0,917810
Orientierung [°]	0,072213	0,078237	0,181880

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	10,688658	12,733760	48,026942
Orientierung [°]	2,060482	2,443452	9,507945

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.15 zu sehen.

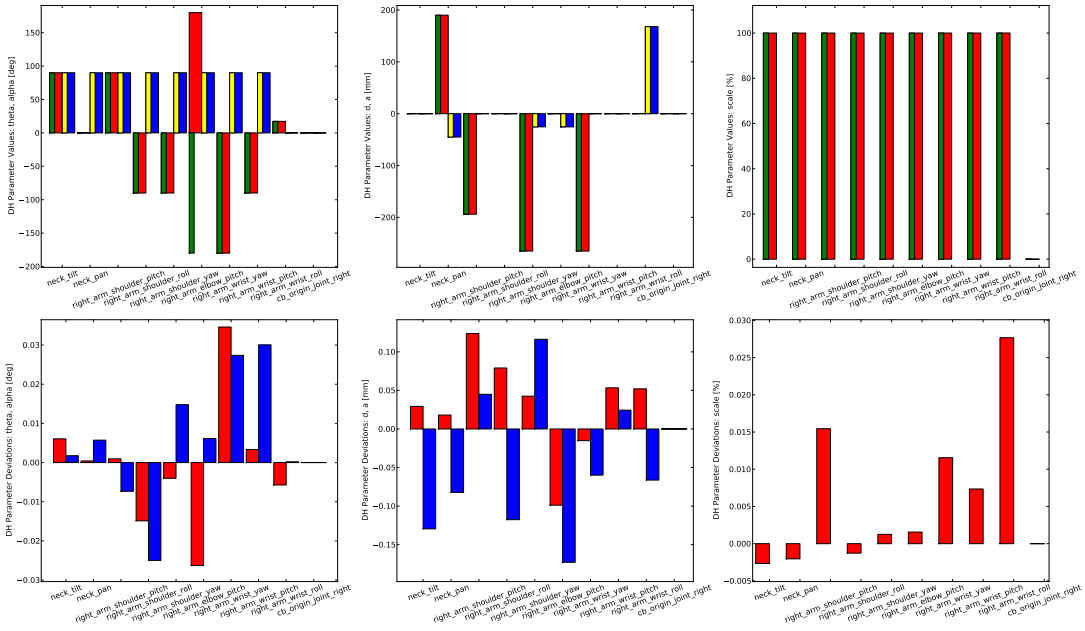


Abbildung 7.15: Abweichender Ellenbogen - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1

7.2.9 Abweichungen in allen Gelenken, 200 Iterationen, Austausch alle 70, Varianz 0,1

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.2.4. Im a priori Modell werden aber nun alle Gelenke manipuliert. Auf alle DH-Parameter werden gleichverteilte Zufallszahlen addiert. Bei den Parametern θ und α liegen die Zufallszahlen in $[-0,09\,rad; 0,09\,rad]$, bei den Parametern d und a in $[-0,015\,m; 0,015\,m]$ und bei den Getriebebeiwerten in $[-0,08; 0,08]$. Einen Vergleich zwischen realem Modell und dem a priori Modell sieht man in Abbildung 7.16. Das Verrauschen wird mit dem Start von *test001_analyse_test.launch* und der Auswahl von *dump_dh* vorgenommen. Der Verwendete Seed ist 6456451. Die *reale* DH-Kette wird in *test001/data/dh.yaml* und die modifizierte Kette in *test001/data/dh-rand.yaml* und *test001/data/rand.urdf* abgelegt.

Das a priori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	74,761667	84,131307	274,546243
Orientierung [°]	15,720674	16,960645	41,692955

Das a priori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	75,728438	85,093917	279,977069
Orientierung [°]	15,879228	17,133095	44,223091

Die Parameter der Kalibrierung sind wie folgt.

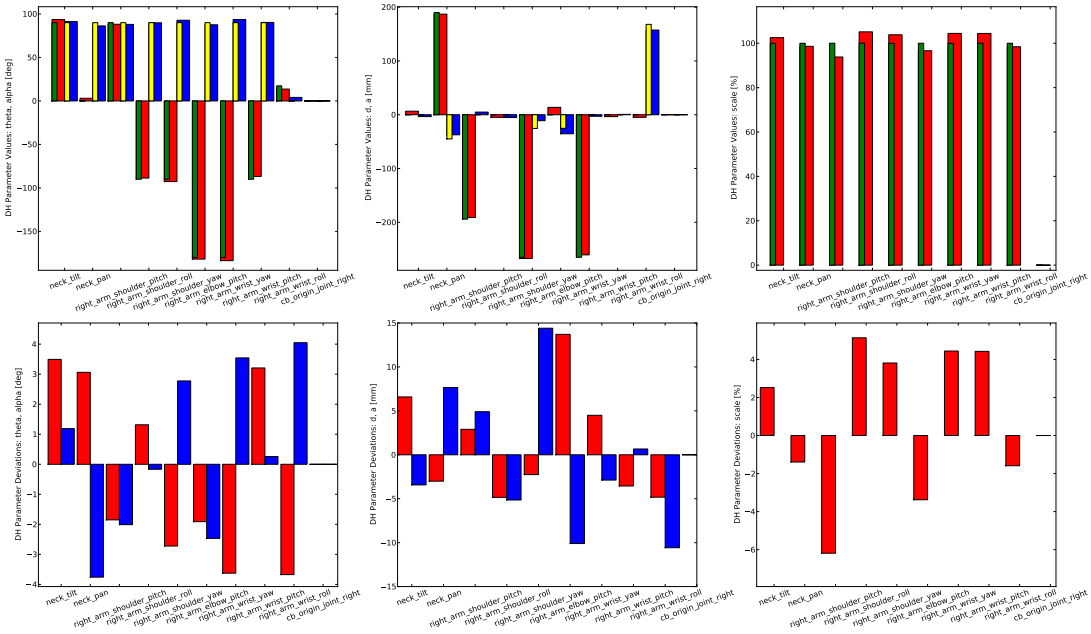


Abbildung 7.16: Alle Gelenke abweichend - real vs. a priori

Feld	**Wert**
calibrate.max_iter	200
update_prior	70
dh_sigma_m	Alle Werte 0,1.

Die Fehler erreichen deutlich vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	0,351073	0,378582	0,919798
Orientierung [°]	0,072274	0,078316	0,181641

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	10,688620	12,733720	48,020639
Orientierung [°]	2,060500	2,443465	9,510542

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.17 zu sehen.

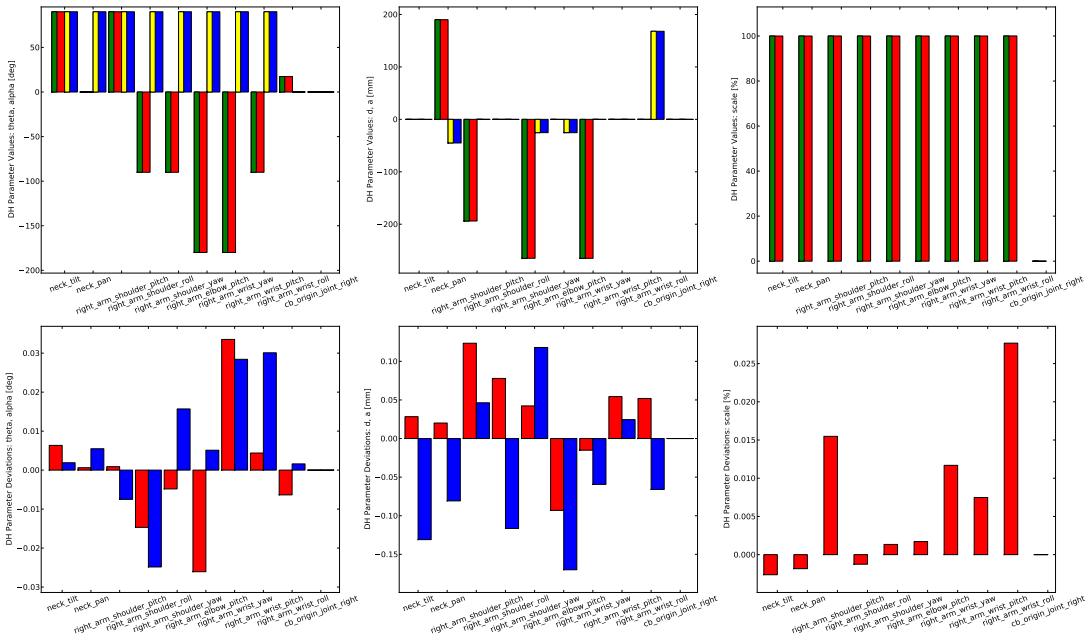


Abbildung 7.17: Alle Gelenke abweichend - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1

7.2.10 Abweichungen in allen Gelenken, 100 Iterationen, Varianz 1,0

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.2.9. Es werden die folgenden Änderungen an den Parametern der Kalibrierung vorgenommen.

Feld	**Wert**
calibrate.max_iter	100
update_prior	*0*
dh_sigma_m	Alle Werte 1,0.

Die Fehler erreichen deutlich vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	0,432143	0,471852	1,265368
Orientierung [°]	0,103435	0,111545	0,264446

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	10,691133	12,736078	47,840895
Orientierung [°]	2,062270	2,444676	9,564318

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.18 zu sehen.

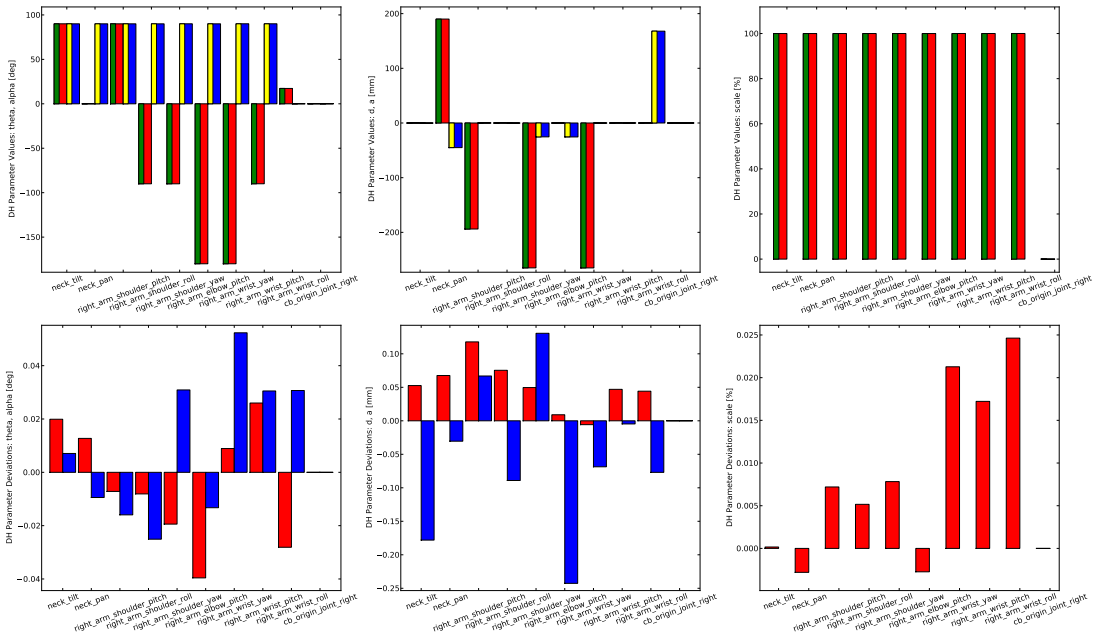


Abbildung 7.18: Alle Gelenke abweichend - real vs. a posteriori, 100 Iterationen, Varianz 1,0

7.3 370 im Gelenkraum gleichverteilte Stellungen

Der Versuch aus Kapitel 7.2 wird mit einem Trainingsdatensatz mit 370 gleichverteilten Stellungen wiederholt. Dabei liegen die Gelenkstellungen im Intervall $[-\pi; \pi]^f$. Für einen groben Überblick über die Trainingspositionen siehe Abbildung 7.19. Die Daten werden unter *test0001/data/synth-370rand.pkl* abgelegt. Die Parameter für die Synthese sind dabei wie folgt, siehe auch Tabelle 6.5.

Feld	**Wert**
synth.joint_std	0,005
synth.ef_std_pos	0,012
synth.ef_std_ori	0,04
synth.type	random
synth.random_count	*370*

Für einen Überblick siehe Abbildung 7.19. Der Aufruf von *test001_analyse_test.launch calc_errors* ergibt die folgenden Fehler, gegenüber der erwarteten Pose der VK.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,149624	13,233071	36,184759
Orientierung [°]	2,010245	2,392308	6,858273

Die Mittelwerte und Effektivwerte entsprechen den Bereichen aus den Kapiteln 7.1.1, 7.1.2 und 7.1.3.

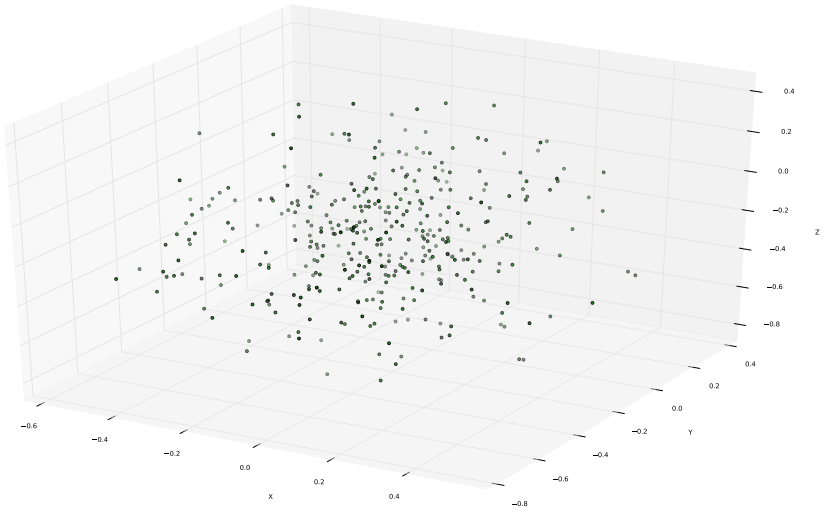


Abbildung 7.19: Trainingspositionen

7.3.1 Keine Abweichungen, 200 Iterationen, Varianz 0,01

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.2.1, jetzt mit 370 gleichverteilten Stellungen. Die Parameter der Kalibrierung sind wie folgt.

Feld	**Wert**
calibrate.max_iter	200
update_prior	0
dh_sigma_m	Alle Werte 0,01.

Zu Beginn der Kalibrierung zeigen sich in der Anzeige leichte Änderungen in den maximalen Fehlern. Die Fehler erreichen lange vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	1,946467	2,094808	5,462166
Orientierung [°]	0,175111	0,188834	0,423208

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,013340	13,017785	37,758774
Orientierung [°]	2,007818	2,382267	6,696448

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.20 zu sehen.

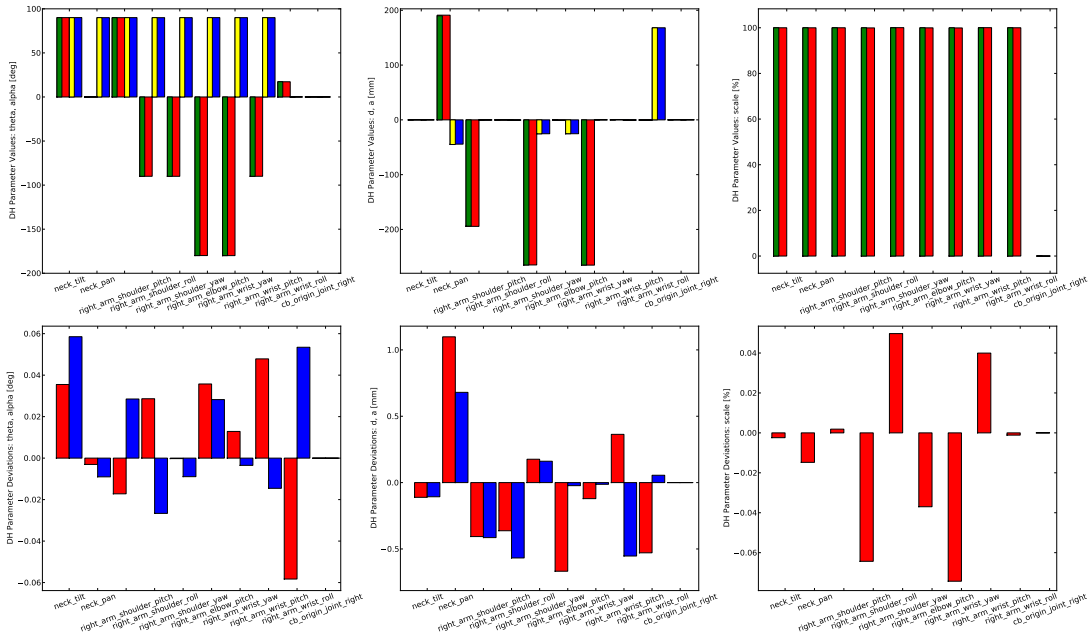


Abbildung 7.20: Keine Abweichungen - real vs. a posteriori, 200 Iterationen, Varianz 0,01

7.3.2 Keine Abweichungen, 200 Iterationen, Varianz 1,0

In diesem Experiment wird der gleiche Aufbau verwendet, wie in Kapitel 7.3.1. Es werden folgende Änderungen an den Parametern der Kalibrierung vorgenommen.

Feld	**Wert**
calibrate.max_iter	200
update_prior	0
dh_sigma_m	Alle Werte 1,0.

Zu Beginn der Kalibrierung zeigen sich in der Anzeige leichte Änderungen in den maximalen Fehlern. Die Fehler erreichen kurz nach dem Start der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	2,225767	2,404145	6,426652
Orientierung [°]	0,253860	0,273607	0,606447

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,014372	13,003848	38,489917
Orientierung [°]	2,013457	2,382365	6,598726

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.21 zu sehen.

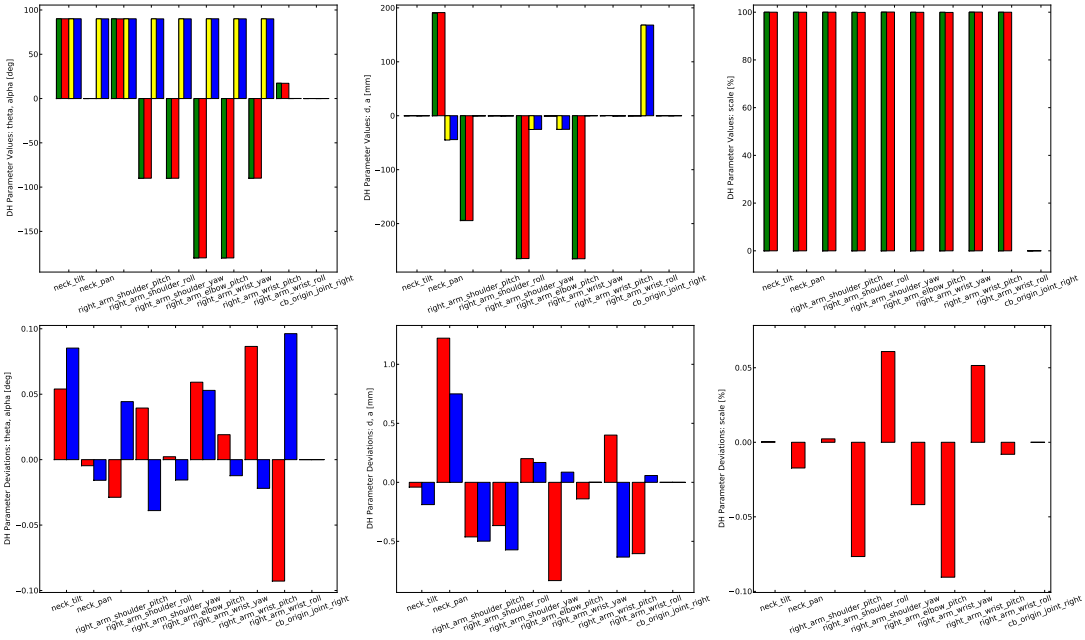


Abbildung 7.21: Keine Abweichungen - real vs. a posteriori, 200 Iterationen, Varianz 1,0

7.3.3 Abweichungen in allen Gelenken, 200 Iterationen, Varianz 1,0

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.2.9, jetzt mit 370 gleichverteilten Stellungen. Es werden folgende Parameter für die Kalibrierung verwendet.

Feld	**Wert**
calibrate.max_iter	200
update_prior	0
dh_sigma_m	Alle Werte 1,0.

Die Fehler erreichen deutlich vor dem Ende der Kalibrierung ein Plateau, siehe Abbildung 7.22. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	2,267817	2,449167	6,581861
Orientierung [°]	0,274568	0,295174	0,665414

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,013402	12,999971	39,017865
Orientierung [°]	2,016098	2,385186	6,571097

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.23 zu sehen.

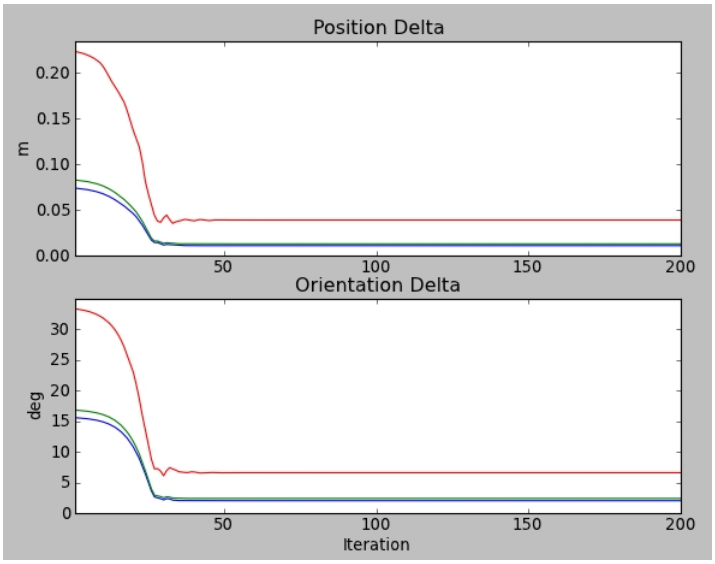


Abbildung 7.22: Fehlerentwicklung - alle Gelenke
rot: Maximum, *blau*: Mittelwert, *grün*: Effektivwert

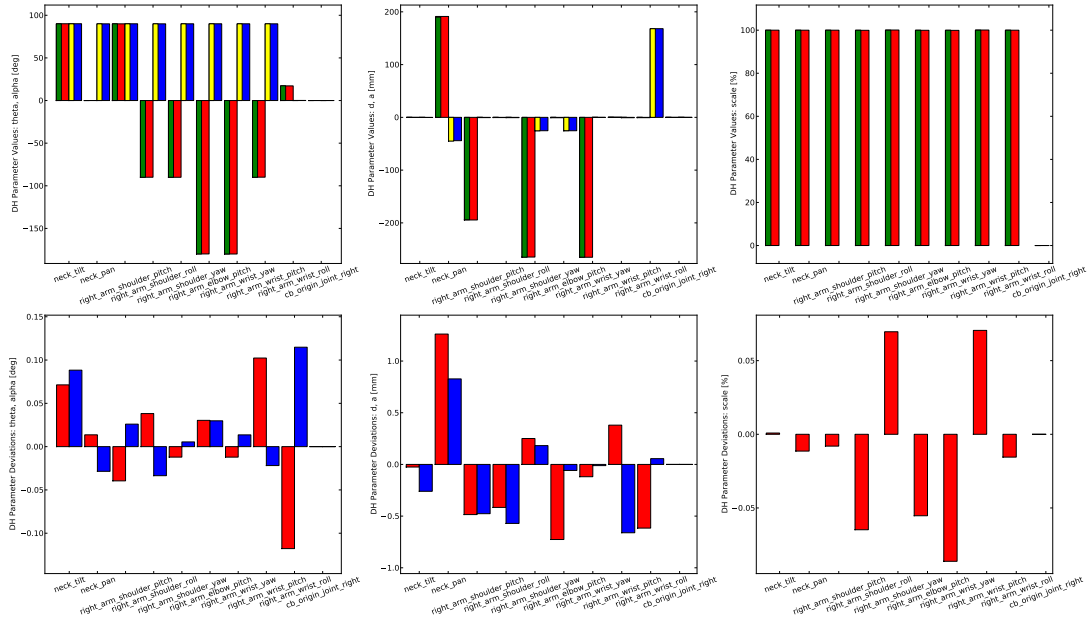


Abbildung 7.23: Alle Gelenke abweichend - real vs. a posteriori, 200 Iterationen, Varianz 1,0

7.3.4 Abweichungen in allen Gelenken, 200 Iterationen, Austausch alle 70, Varianz 0,1

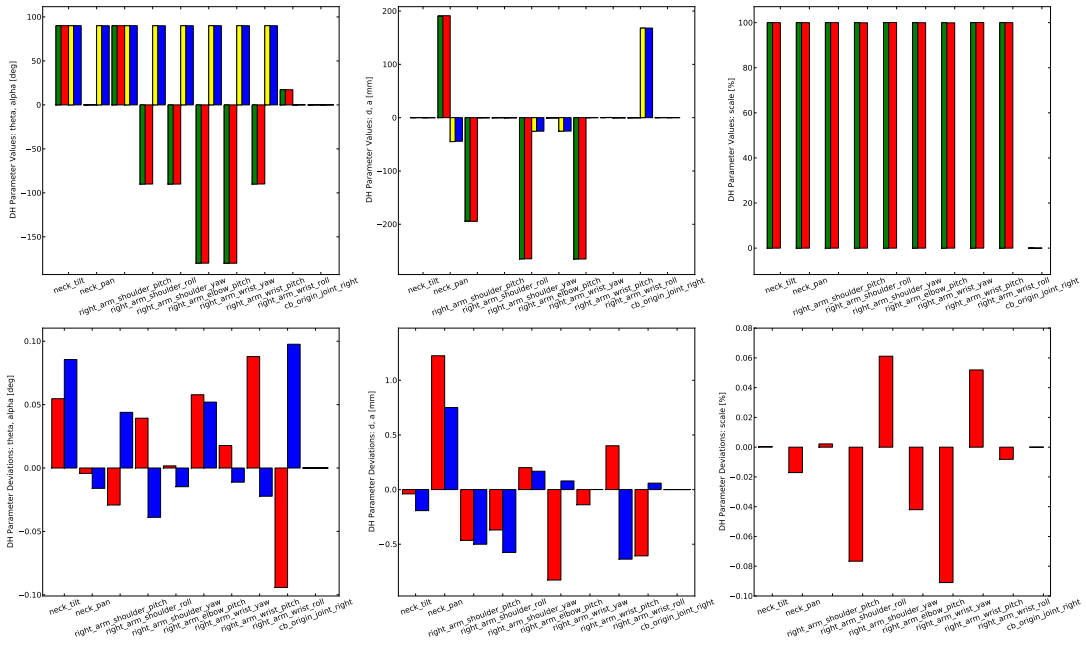


Abbildung 7.24: Alle Gelenke abweichend - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.3.3. Es werden folgende Änderungen an den Parametern der Kalibrierung vorgenommen.

Feld	**Wert**
calibrate.max_iter	200
update_prior	*70*
dh_sigma_m	Alle Werte 0,1.

Die Fehler erreichen deutlich vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	2,230981	2,409830	6,448042
Orientierung [°]	0,255059	0,274867	0,610530

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,014429	13,003573	38,516875
Orientierung [°]	2,013526	2,382421	6,596480

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.24 zu sehen.

7.3.5 Abweichungen in allen Gelenken, Modell 2, 200 Iterationen, Austausch alle 70, Varianz 0,1

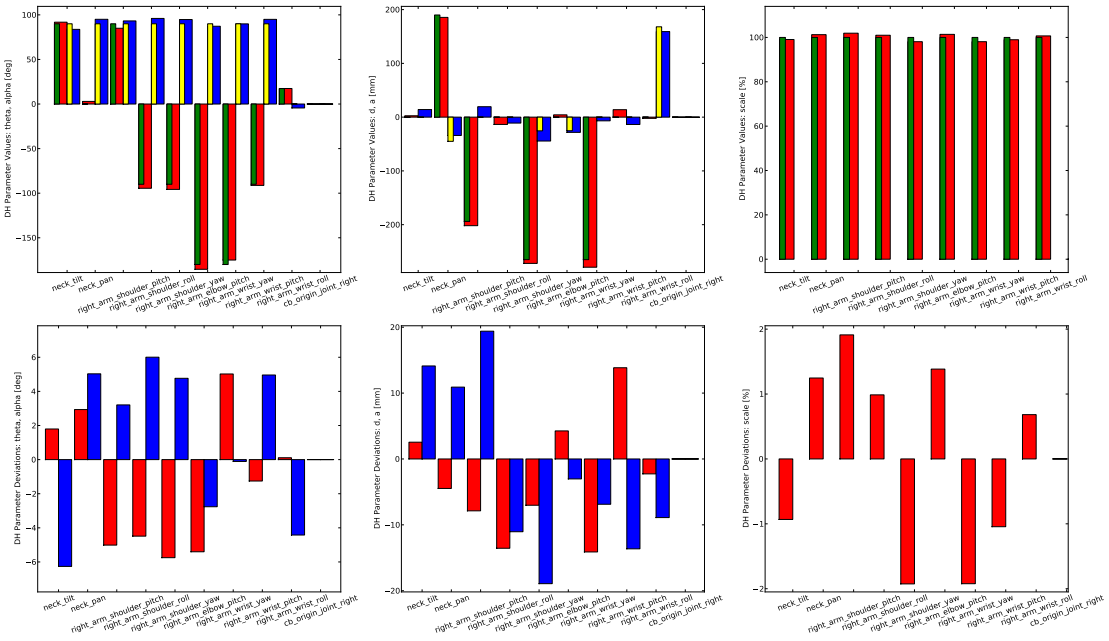


Abbildung 7.25: Alle Gelenke abweichend, Modell 2 - real vs. a priori

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.3.4. Im a priori Modell werden die Gelenke aber anders manipuliert. Auf alle DH-Parameter werden wieder gleichverteilte Zufallszahlen addiert. Bei den Parametern θ und α liegen die Zufallszahlen in $[-0,11\,rad; 0,11\,rad]$, bei den Parametern d und a in $[-0,02\,m; 0,02\,m]$ und bei den Getriebebeiwerten in $[-0,02; 0,02]$. Einen Vergleich zwischen realem Modell und dem a priori Modell sieht man in Abbildung 7.25. Das Verrauschen wird mit dem Start von *test001_analyse_test.launch* und der Auswahl von *dump_dh* vorgenommen. Der Verwendete Seed ist 2243516872431. Die *reale* DH-Kette ist die gleiche wie zuvor. Die modifizierte Kette wird in *test001/data/dh-rand2.yaml* und *test001/data/rand2.urdf* abgelegt.

Das a priori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	90,457608	102,989580	325,587514
Orientierung [°]	17,136890	18,542053	47,437128

Das a priori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	87,504677	99,542215	274,869106
Orientierung [°]	16,927629	18,439079	43,088662

belief

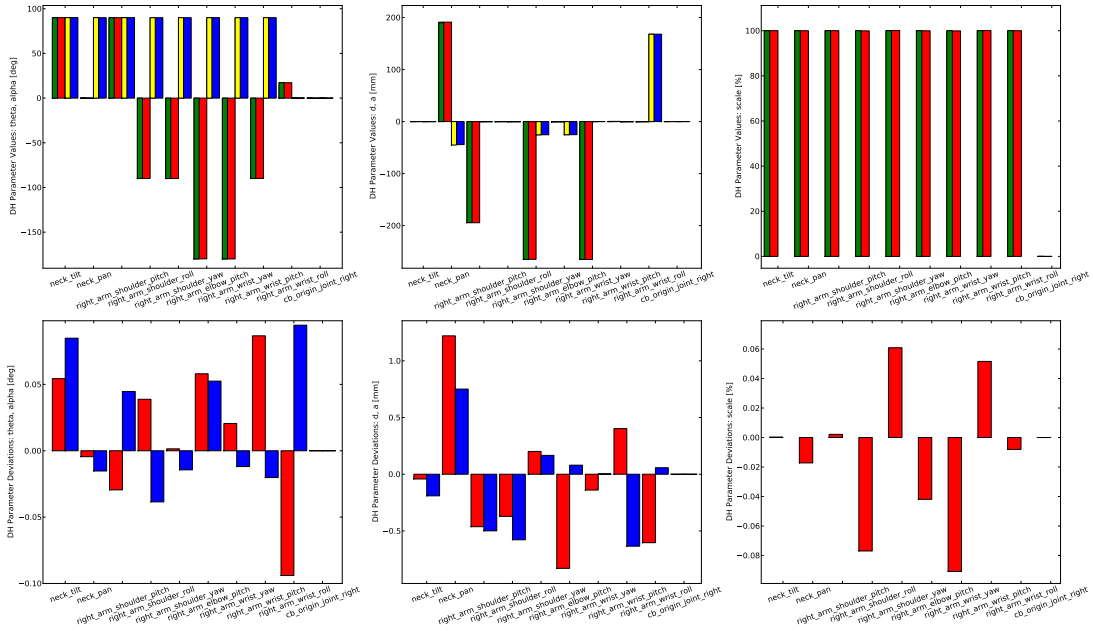


Abbildung 7.26: Alle Gelenke abweichend, Modell 2 - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1

Die Parameter der Kalibrierung sind wie folgt.

Feld	**Wert**
calibrate.max_iter	200
update_prior	70
dh_sigma_m	Alle Werte 0,1.

Die Fehler erreichen deutlich vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	2,227910	2,406603	6,432049
Orientierung [°]	0,253417	0,273166	0,607509

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,014391	13,003808	38,493546
Orientierung [°]	2,013432	2,382387	6,599922

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.26 zu sehen.

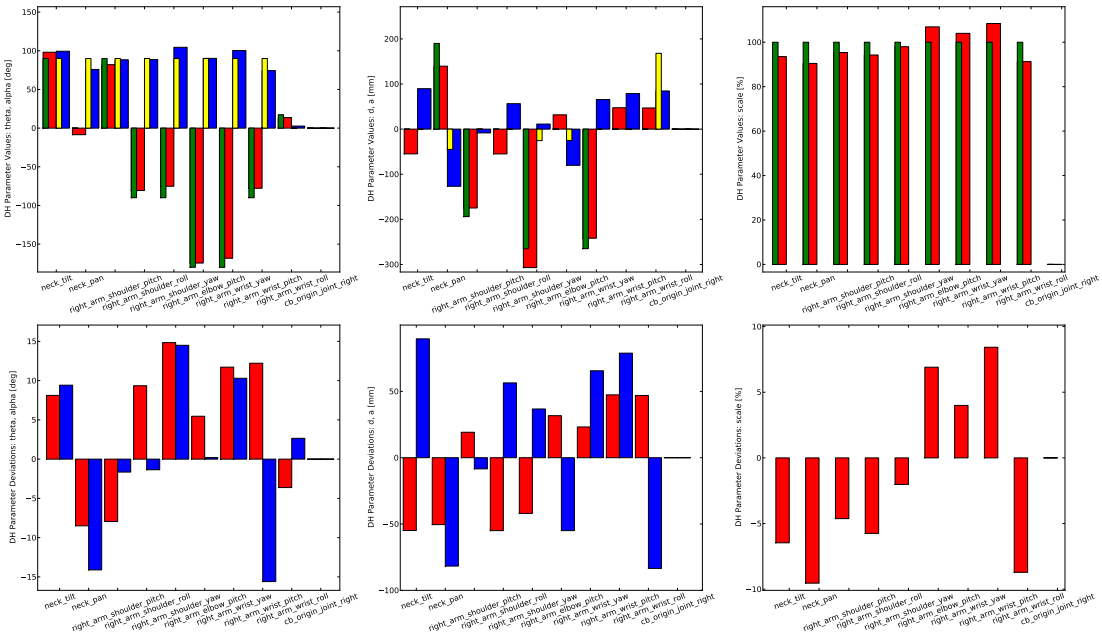


Abbildung 7.27: Alle Gelenke abweichend, Modell 3 - real vs. a priori

7.3.6 Abweichungen in allen Gelenken, Modell 3, 200 Iterationen, Austausch alle 70, Varianz 0,1

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.3.4. Im a priori Modell werden die Gelenke aber anders manipuliert. Auf alle DH-Parameter werden wieder gleichverteilte Zufallszahlen addiert. Bei den Parametern θ und α liegen die Zufallszahlen in $[-0,3\,rad; 0,3\,rad]$, bei den Parametern d und a in $[-0,1\,m; 0,1\,m]$ und bei den Getriebebeiwerten in $[-0,1; 0,1]$. Einen Vergleich zwischen realem Modell und dem a priori Modell sieht man in Abbildung 7.27. Das Verrauschen wird mit dem Start von *test001_analyse_test.launch* und der Auswahl von *dump_dh* vorgenommen. Der Verwendete Seed ist 9634455354. Die *reale* DH-Kette ist die gleiche wie zuvor. Die modifizierte Kette wird in *test001/data/dh-rand3.yaml* und *test001/data/rand3.urdf* abgelegt.

Das a priori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	296,652521	323,128603	883,475427
Orientierung [°]	42,262239	45,334694	108,875145

Das a priori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	291,875190	314,322378	608,409357
Orientierung [°]	41,913823	44,470462	97,344940

Die Parameter der Kalibrierung sind wie folgt.

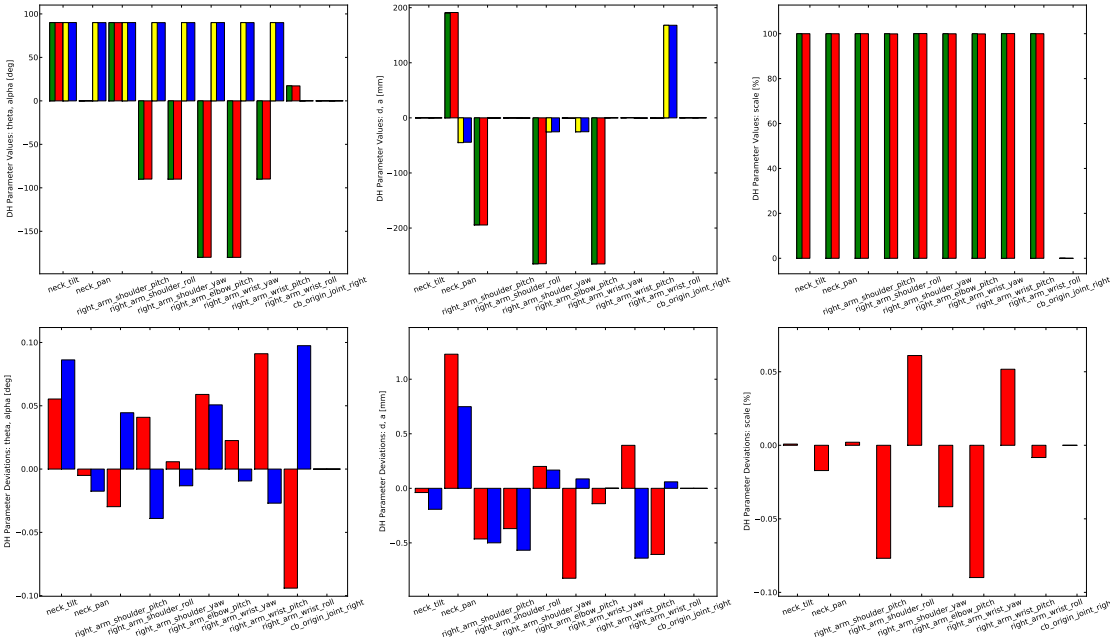


Abbildung 7.28: Alle Gelenke abweichend, Modell 3 - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1

Feld	**Wert**
calibrate.max_iter	200
update_prior	70
dh_sigma_m	Alle Werte 0,1.

Die Fehler erreichen deutlich vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	2,234168	2,413601	6,473157
Orientierung [°]	0,256967	0,277032	0,619385

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,014205	13,003508	38,537735
Orientierung [°]	2,013724	2,382384	6,601060

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.28 zu sehen.

7.4 370 in den erlaubten Grenzen gleichverteilte Stellungen

Der Versuch aus Kapitel 7.2 wird mit einem Trainingsdatensatz mit 370 in den zulässigen den Gelenkgrenzen gleichverteilten Stellungen wiederholt. Dabei liegen die Gelenkstel-

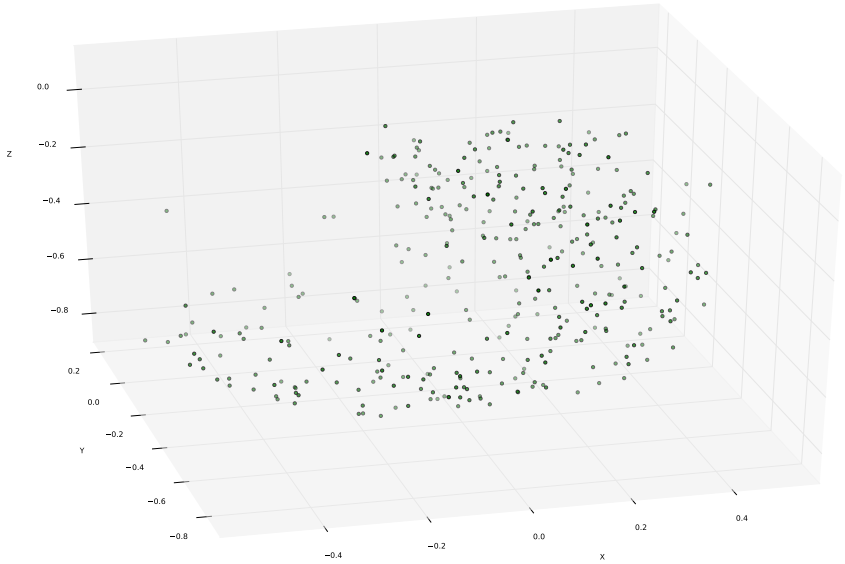


Abbildung 7.29: Trainingspositionen

lungen im Bereich $[Untergrenze^1; Obergrenze^1] \times \ldots \times [Untergrenze^f; Obergrenze^f]$. Für einen groben Überblick über die Trainingspositionen siehe Abbildung 7.29. Die Daten werden unter *test0001/data/synth-370allowed.pkl* abgelegt. Die Parameter für die Synthese sind dabei wie folgt, siehe auch Tabelle 6.5.

Feld	**Wert**
synth.joint_std	0,005
synth.ef_std_pos	0,012
synth.ef_std_ori	0,04
synth.type	*allowed*
synth.random_count	370

Für einen Überblick siehe Abbildung 7.29. Der Aufruf von *test001_analyse_test.launch calc_errors* ergibt die folgenden Fehler, gegenüber der erwarteten Pose der VK.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,395389	13,538551	42,336542
Orientierung [°]	2,034591	2,392917	6,738983

Die Mittelwerte und Effektivwerte entsprechen den Bereichen aus den Kapiteln 7.1.1, 7.1.2 und 7.1.3.

7.4.1 Keine Abweichungen, 200 Iterationen, Varianz 0,1

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.2.1, mit anderen Trainingsdaten. Es werden folgende Parameter für die Kalibrierung verwendet.

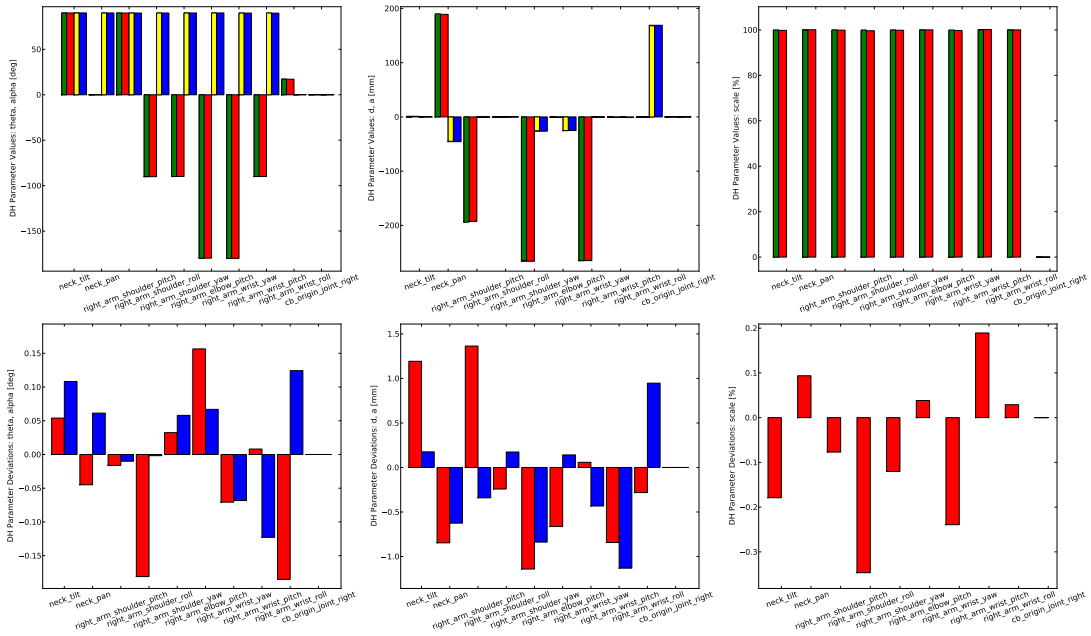


Abbildung 7.30: Keine Abweichungen - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1

Feld	**Wert**
calibrate.max_iter	200
update_prior	70
dh_sigma_m	Alle Werte 0,1.

Zu Beginn der Kalibrierung zeigen sich in der Anzeige Änderungen in den Fehlern. Die Fehler erreichen lange vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	3,950137	4,456183	14,825892
Orientierung [°]	0,626088	0,680037	1,662102

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,387384	13,541087	42,115277
Orientierung [°]	2,027918	2,389312	6,750858

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.30 zu sehen.

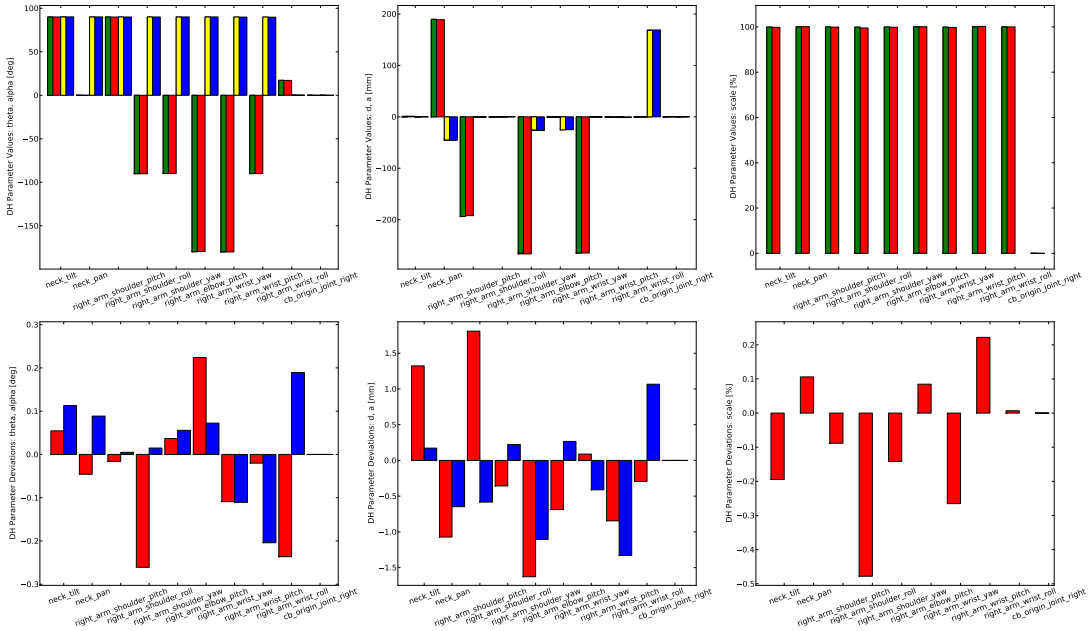


Abbildung 7.31: Keine Abweichungen - real vs. a posteriori, 340 Iterationen, Austausch alle 70, Varianz 0,5

7.4.2 Keine Abweichungen, 340 Iterationen, Austausch alle 70, Varianz 0,5

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.4.1. Zudem werden folgende Änderungen an den Parametern der Kalibrierung vorgenommen.

Feld	**Wert**
calibrate.max_iter	340
update_prior	70
dh_sigma_m	Alle Werte 0,5.

Zu Beginn der Kalibrierung zeigen sich in der Anzeige Änderungen in den Fehlern. Die Fehler erreichen lange vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	4,930417	5,644573	19,083632
Orientierung [°]	0,811225	0,884861	2,254950

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,395807	13,533629	42,075963
Orientierung [°]	2,031435	2,388998	6,795816

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.31 zu sehen.

7.4.3 Abweichungen in allen Gelenken, 200 Iterationen, Austausch alle 70, Varianz 0,1

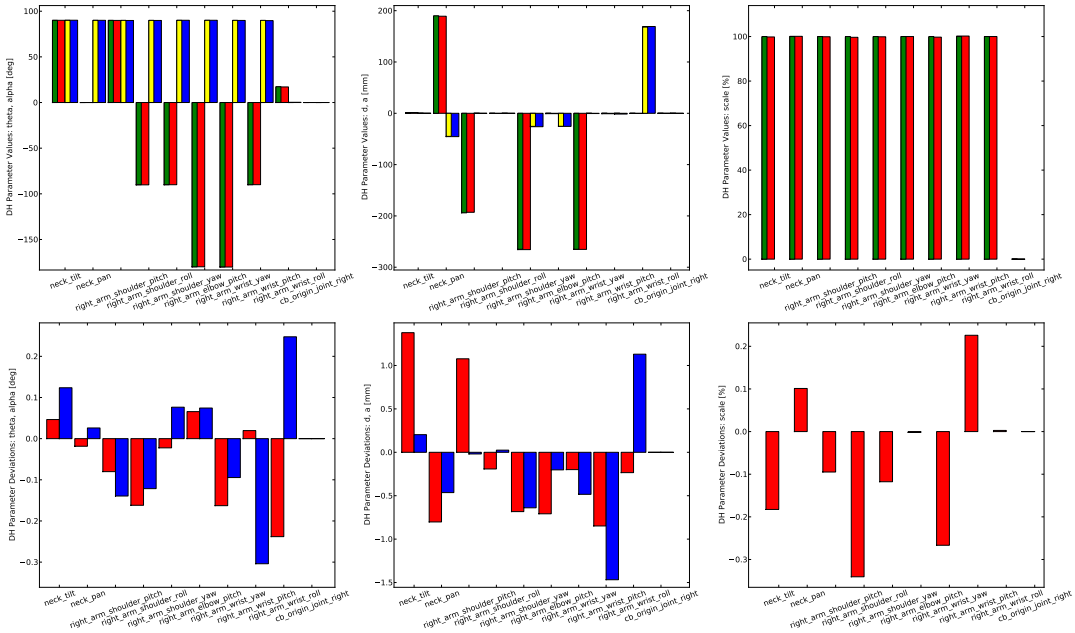


Abbildung 7.32: Alle Gelenke abweichend - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.2.9, mit anderen Trainingsdaten. Es werden folgende Parameter für die Kalibrierung verwendet.

Feld	**Wert**
calibrate.max_iter	200
update_prior	70
dh_sigma_m	Alle Werte 0,1.

Die Fehler erreichen deutlich vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	4,182313	4,680005	14,795198
Orientierung [°]	0,762828	0,826092	1,940915

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,395389	13,538551	42,336542
Orientierung [°]	2,034591	2,392917	6,738983

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.32 zu sehen.

7.4.4 Abweichungen in allen Gelenken, 340 Iterationen, Austausch alle 70, Varianz 0,5

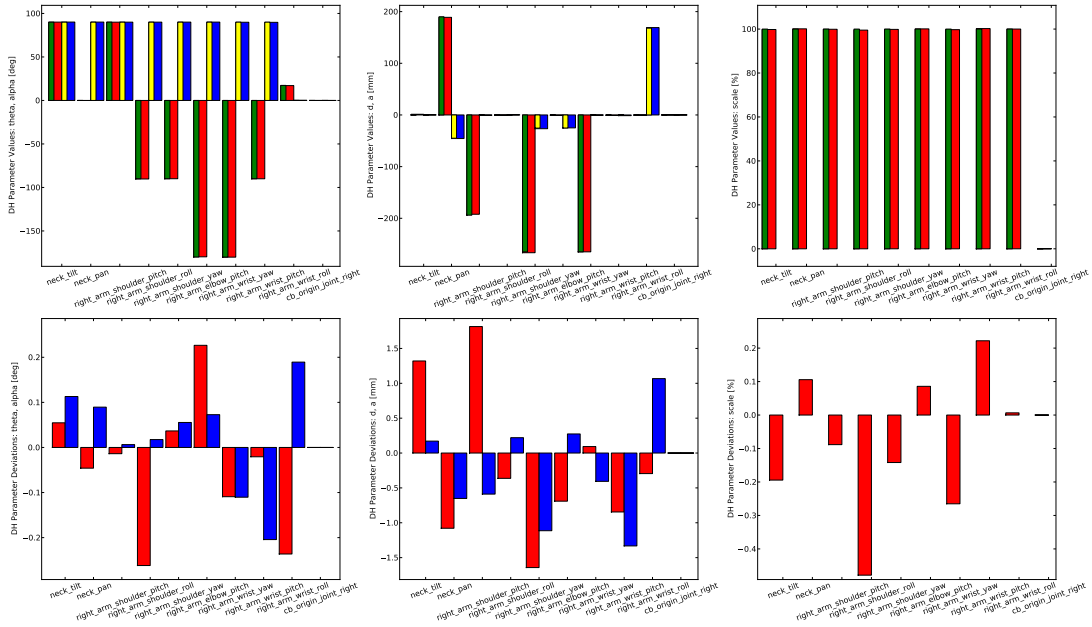


Abbildung 7.33: Alle Gelenke abweichend - real vs. a posteriori, 340 Iterationen, Austausch alle 70, Varianz 0,5

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.4.3, mit anderen Trainingsdaten. Es werden folgende Änderungen an den Parametern der Kalibrierung vorgenommen.

Feld	**Wert**
calibrate.max_iter	340
update_prior	70
dh_sigma_m	Alle Werte 0,5.

Die Fehler erreichen deutlich vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	4,940358	5,655661	19,103404
Orientierung [°]	0,812596	0,886428	2,260912

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,395691	13,533706	42,078350
Orientierung [°]	2,031428	2,388976	6,796236

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori

Modell ist in Abbildung 7.33 zu sehen.

7.5 370 Stellungen aus Vorgaben

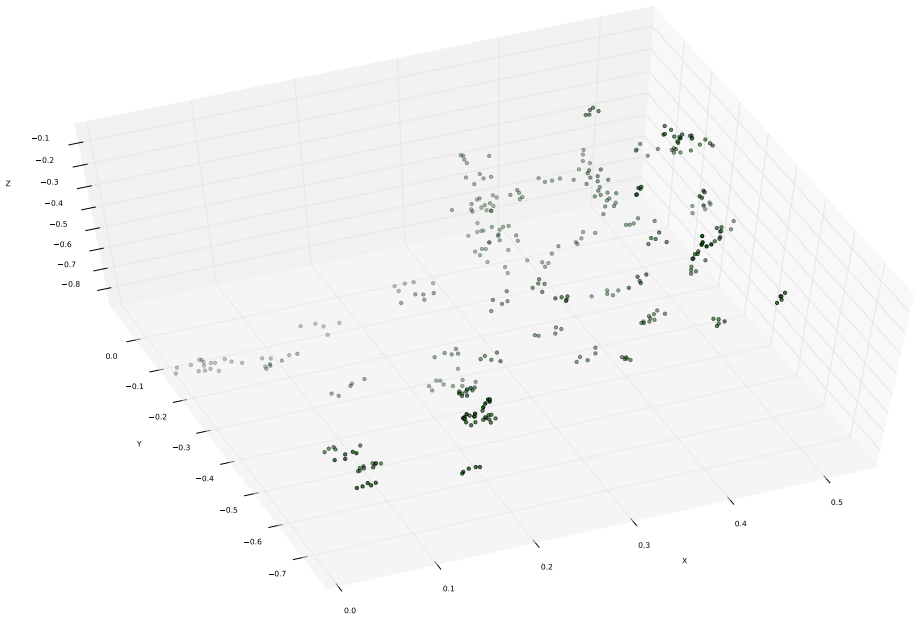


Abbildung 7.34: Trainingspositionen

Der Versuch aus Kapitel 7.2 wird mit einem Trainingsdatensatz mit 370 Stellungen wiederholt. Die Stellungen werden aus 74 vorgegebenen Stellungen, die jeweils fünf mal leicht variiert werden, erzeugt. Für einen groben Überblick über die Trainingspositionen siehe Abbildung 7.34. Die Daten werden unter *test0001/data/synth-370files.pkl* abgelegt. Die Parameter für die Synthese sind dabei wie folgt, siehe auch Tabelle 6.5.

Feld	**Wert**
synth.joint_std	0,005
synth.ef_std_pos	0,012
synth.ef_std_ori	0,04
synth.type	*files*
synth.random_count	370
explorer.multi.joint_lim	0,06
explorer.multi.displacements.neck_pan	0,3
explorer.multi.displacements.neck_tilt	0,2

Für einen Überblick siehe Abbildung 7.34. Der Aufruf von *test001_analyse_test.launch calc_errors* ergibt die folgenden Fehler, gegenüber der erwarteten Pose der VK.

Abweichungen	Mittelwert	Effektivwert	Maximum
Position [mm]	11,739886	13,708629	37,145837
Orientierung [°]	2,170228	2,591481	10,636556

Die Mittelwerte und Effektivwerte entsprechen den Bereichen aus den Kapiteln 7.1.1, 7.1.2 und 7.1.3.

7.5.1 Keine Abweichungen, 200 Iterationen, Austausch alle 70, Varianz 0,1

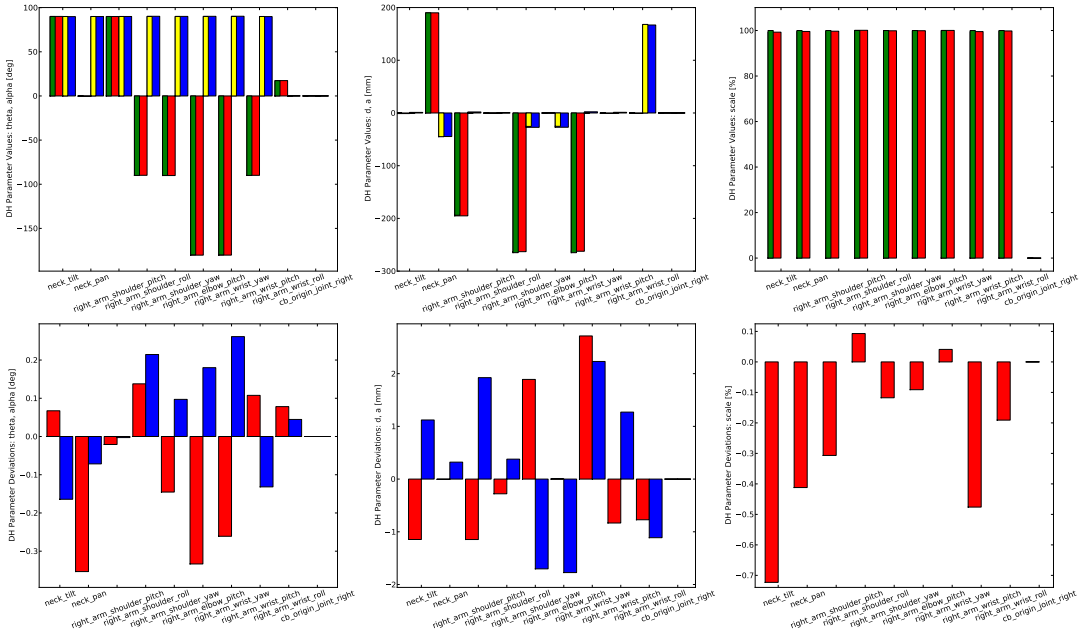


Abbildung 7.35: Keine Abweichungen - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.2.1, mit anderen Trainingsdaten. Es werden folgende Parameter für die Kalibrierung verwendet.

Feld	Wert
calibrate.max_iter	200
update_prior	0
dh_sigma_m	Alle Werte 0,1.

Zu Beginn der Kalibrierung zeigen sich in der Anzeige Änderungen in den Fehlern. Die Fehler erreichen lange vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	Mittelwert	Effektivwert	Maximum
Position [mm]	8,145581	9,079416	29,745218
Orientierung [°]	1,231497	1,329265	3,227830

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,543717	13,497457	38,023185
Orientierung [°]	2,147930	2,561529	10,228348

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.35 zu sehen.

7.5.2 Keine Abweichungen, 340 Iterationen, Austausch alle 70, Varianz 0,5

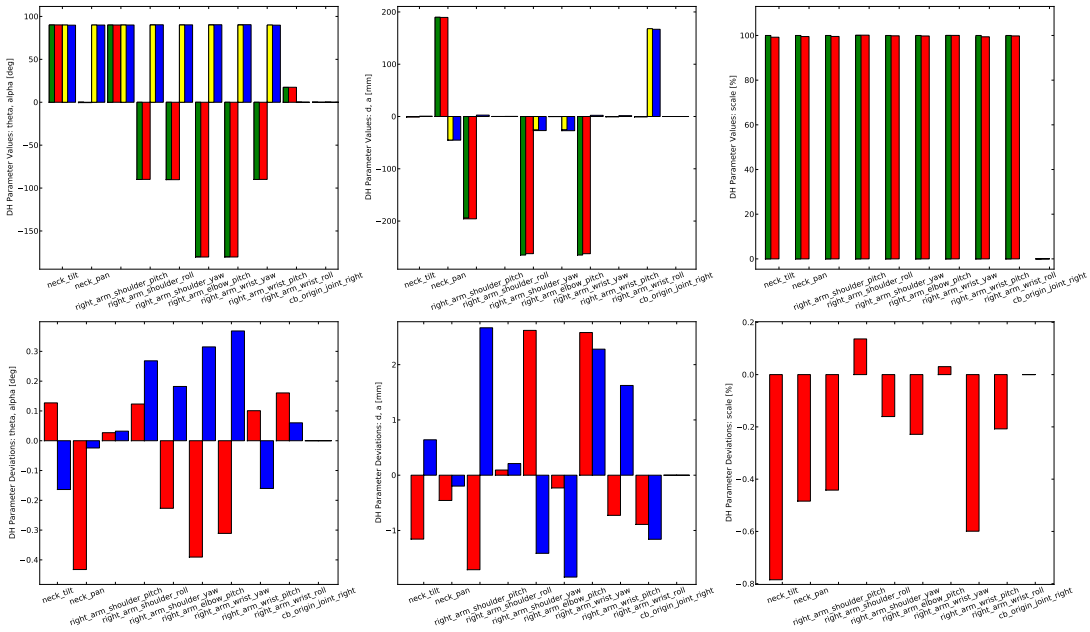


Abbildung 7.36: Keine Abweichungen - real vs. a posteriori, 340 Iterationen, Austausch alle 70, Varianz 0,5

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.5.1. Zudem werden folgende Änderungen an den Parametern der Kalibrierung vorgenommen.

Feld	**Wert**
calibrate.max_iter	340
update_prior	70
dh_sigma_m	Alle Werte 0,5.

Zu Beginn der Kalibrierung zeigen sich in der Anzeige Änderungen in den Fehlern. Die Fehler erreichen lange vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	9,345034	10,405019	35,542956
Orientierung [°]	1,506950	1,629534	4,202998

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,545804	13,498919	38,060718
Orientierung [°]	2,146389	2,560262	10,208519

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.36 zu sehen.

7.5.3 Abweichungen in allen Gelenken, 200 Iterationen, Austausch alle 70, Varianz 0,1

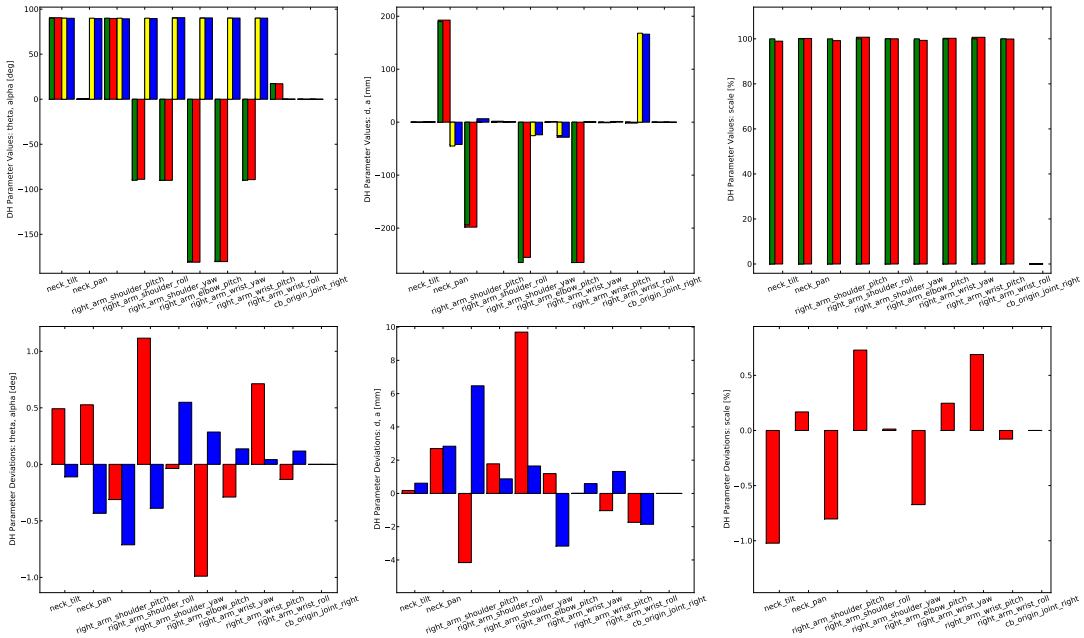


Abbildung 7.37: Alle Gelenke abweichend - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.2.9, mit anderen Trainingsdaten. Es werden folgende Parameter für die Kalibrierung verwendet.

Feld	**Wert**
calibrate.max_iter	200
update_prior	70
dh_sigma_m	Alle Werte 0,1.

Die Fehler erreichen deutlich vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	Mittelwert	Effektivwert	Maximum
Position [mm]	17,076915	18,878428	70,559609
Orientierung [°]	2,639419	2,878597	7,148730

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	Mittelwert	Effektivwert	Maximum
Position [mm]	11,635477	13,588566	36,907193
Orientierung [°]	2,191711	2,597597	10,055237

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.37 zu sehen.

7.5.4 Abweichungen in allen Gelenken, 340 Iterationen, Austausch alle 70, Varianz 0,5

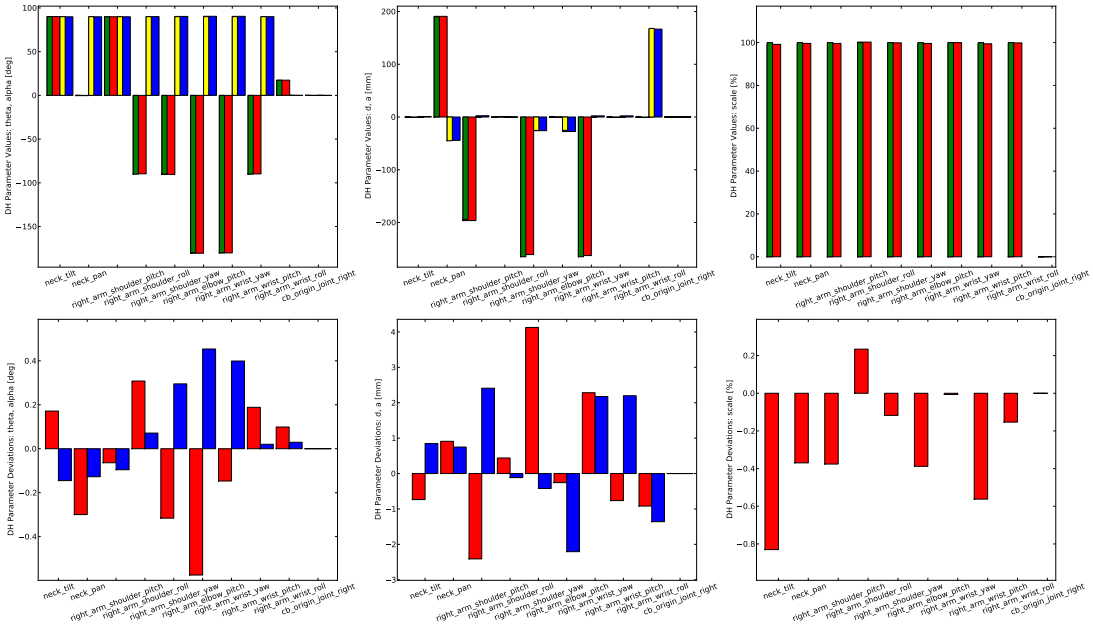


Abbildung 7.38: Alle Gelenke abweichend - real vs. a posteriori, 340 Iterationen, Austausch alle 70, Varianz 0,5

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.5.3. Es werden folgende Änderungen an den Parametern der Kalibrierung vorgenommen.

Feld	Wert
calibrate.max_iter	340
update_prior	70
dh_sigma_m	Alle Werte 0,5.

Die Fehler erreichen deutlich vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	9,957787	11,025092	38,963513
Orientierung [°]	1,567695	1,705560	4,302423

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,552226	13,504186	38,075522
Orientierung [°]	2,148291	2,561702	10,219215

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.38 zu sehen.

7.5.5 Abweichungen in allen Gelenken, 480 Iterationen, Austausch alle 70, Varianz 1,5

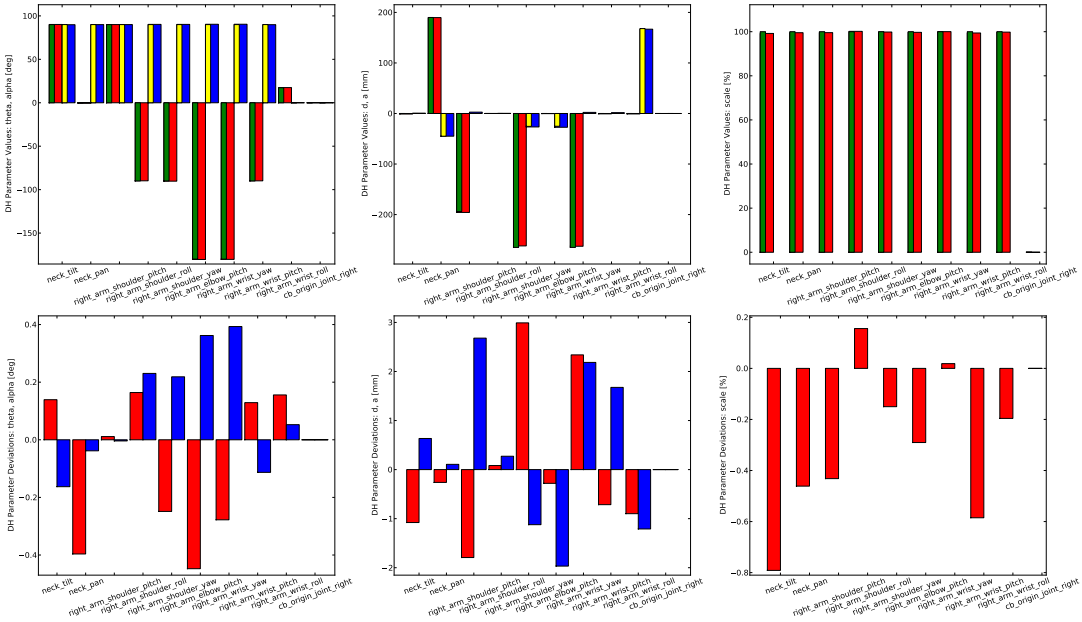


Abbildung 7.39: Alle Gelenke abweichend - real vs. a posteriori, 480 Iterationen, Austausch alle 70, Varianz 1,5

In diesem Experiment wird der gleiche Aufbau verwendet wie in Kapitel 7.5.3. Es werden folgende Änderungen an den Parametern der Kalibrierung vorgenommen.

Feld	**Wert**
calibrate.max_iter	480
update_prior	70
dh_sigma_m	Alle Werte 1,5.

Die Fehler erreichen deutlich vor dem Ende der Kalibrierung ein Plateau. Das a posteriori Modell hat gegenüber den Testdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	9,329362	10,377473	35,736001
Orientierung [°]	1,519568	1,645654	4,279894

Das a posteriori Modell hat gegenüber den Trainingsdaten die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,545468	13,498988	38,037682
Orientierung [°]	2,146558	2,560384	10,204006

Der Vergleich der DH-Parametrisierungen vom realen Modell und dem a posteriori Modell ist in Abbildung 7.39 zu sehen.

7.6 Simulation von Laser und Schachbrett

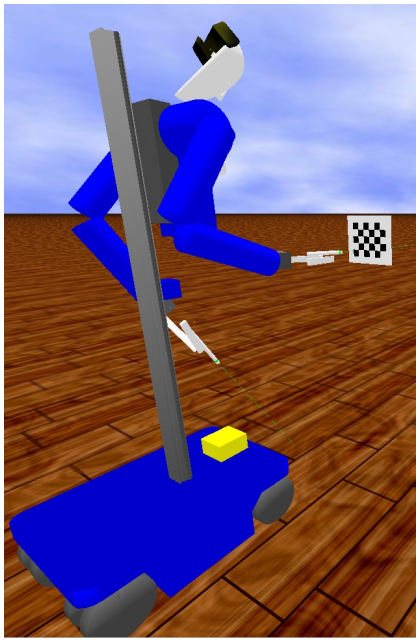
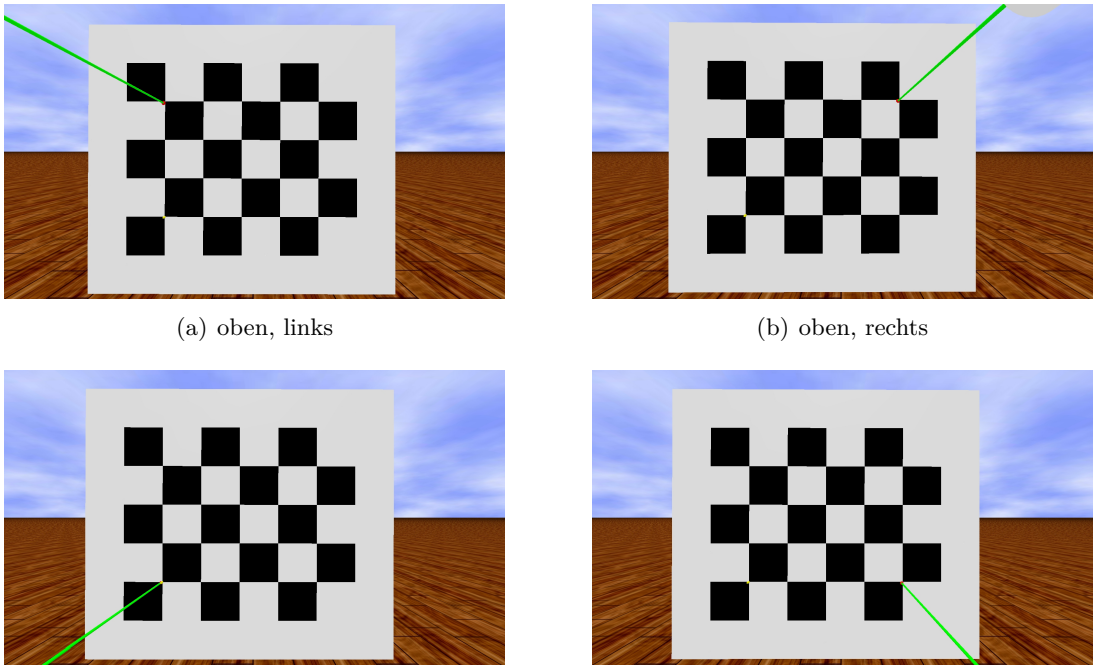


Abbildung 7.40: Simulation von Laser und Schachbrett

Wir betrachten nun die Integration des kalibrierten Modells in die bestehende Softwareinfrastruktur. Hierfür wird die aus Kapitel 6.3 bekannte Applikation *eval.py* verwendet. Diese sucht mit der Kamera in der Roboterumgebung nach einem Schachbrett und bestimmt dessen Pose. Hieraus lassen sich leicht die Kreuzungspunkte des Schachbretts bestimmen. Das Ziel ist es nun, den Strahl eines im Greifer befindlichen Lasers auf einen ausgewählten Kreuzungspunkt zu richten. Hierfür wird ein Arbeitspunkt im



(a) oben, links

(b) oben, rechts

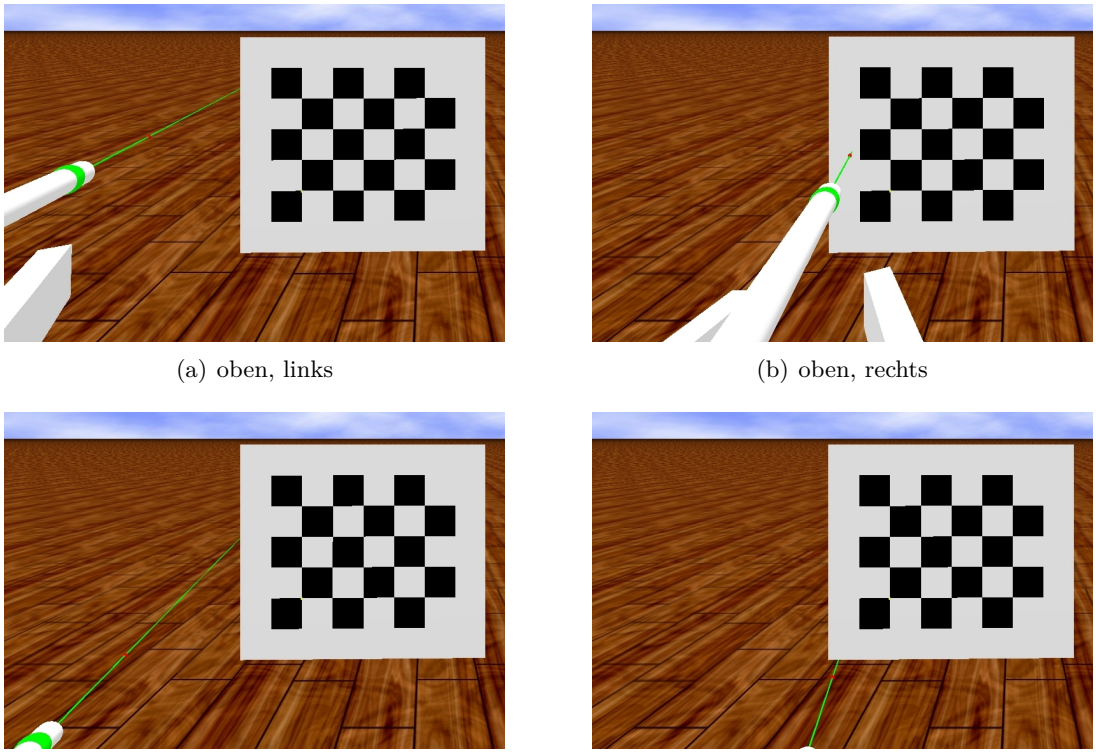
(c) unten, links

(d) unten, rechts

Abbildung 7.41: Reales Modell

Laserstrahl, welcher einige Zentimeter vor der Linse liegt, an die Position des Punktes gebracht. Die quadratischen Felder des Schachbretts haben eine Kantenlänge von 25 mm. In der Simulation ist der Laserstrahl grün und hat einen Durchmesser von einem Millimeter. Der Arbeitspunkt wird als roter Ball von drei Millimetern visualisiert. Mit Hilfe der KDL wird aus dem gegebenen URDF-Modell des Roboters die IK dafür berechnet. Die resultierenden Gelenkstellungen werden dann an das Robotcontrol geschickt, siehe Kapitel 4.3. Dieses steuert dann den Roboter in die gewünschte Gelenkkonfiguration. Der Aufbau der Simulation ist in Abbildung 7.40 zu sehen. Bei der Simulation werden jeweils zwei URDF-Modelle verwendet. Das eine stellt das reale Modell dar, hierfür wird in diesem Experiment immer *hubert_calib_simulation)/urdf/cosero/cosero_eval_h.urdf.xacro* verwendet. Dieses Modell wird von Gazebo benutzt, in ihm ist auch das Schachbrett modelliert. Das andere Modell wird von allen anderen Komponenten benutzt. Es werden im folgenden die vier äusseren Kreutzungspunkte des Schachbretts angesteuert.

Zur Überprüfung der grundlegenden Funktion des Aufbaus führen wir als erstes einen Lauf mit dem realen Modell durch, dies ist in *hubert_calib_simulation)/urdf/cosero/cosero_real_h.urdf.xacro*. Diese Datei entspricht der oben angegebenen, sie enthält aber kein Schachbrett. Die Ergebnisse dieses Laufs sind in Abbildung 7.41 zu sehen. Man sieht, dass der Strahl jeweils nur wenige Millimeter vom gewünschten Punkt das Brett trifft. Bei genauerer Betrachtung zeigt sich sogar, dass die Markierung die Ebene des



(a) oben, links (b) oben, rechts

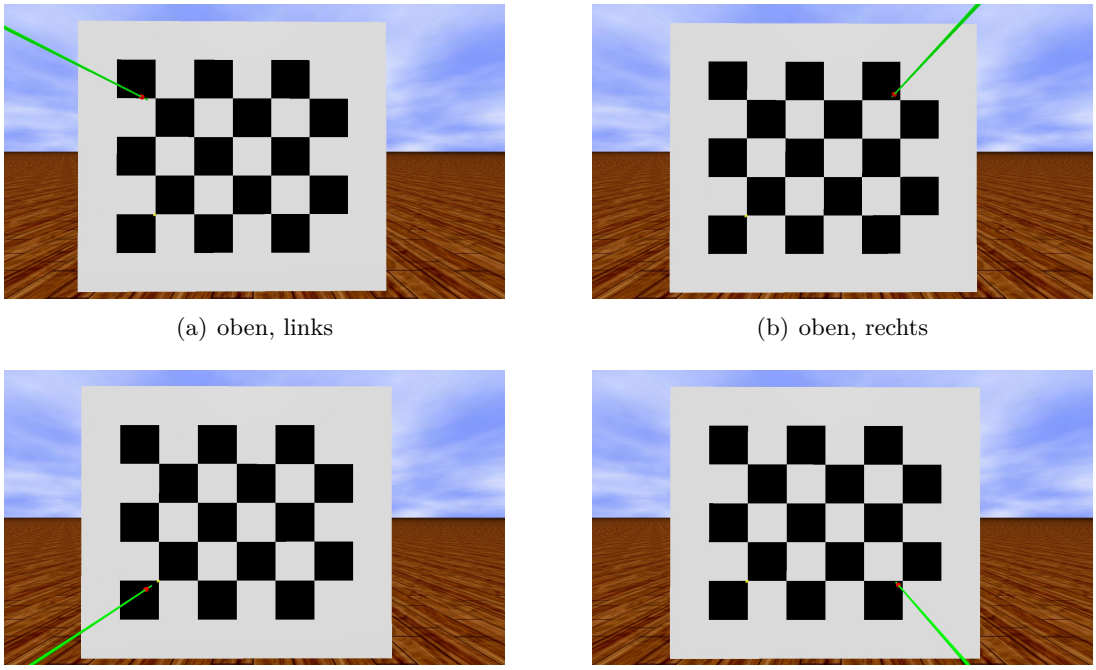
(c) unten, links (d) unten, rechts

Abbildung 7.42: A priori Modell

Schachbretts schneidet. Diese Simulation wird mit *test003_eval_real.launch* gestartet.

Nun verwenden wir die in Kapitel 7.2.9 beschriebenen Abweichungen des Roboters. Hierbei weichen alle DH-Parameter der kinematischen Kette von der Kamera bis zum Greifer von den realen Werten ab. Ein Modell mit diesen Abweichungen wird für die Simulation mit *test003_eval_analyse.launch* erzeugt. Ansonsten entspricht es dem vorherigen Modell. Das Ergebnis der Simulation mit diesem Modell sehen wir in Abbildung 7.42. Lediglich in einem Fall trifft der Laserstrahl überhaupt das Schachbrett, und dort beträgt die Abweichung über 13 cm. Diese Simulation wird mit *test003_eval_prior.launch* gestartet.

Jetzt wird mit *test003_calib.launch* die in Kapitel 7.5.3 beschriebene Kalibrierung durchgeführt. Die Simulation mit dem a posteriori Modell wird mit *test003_eval_belief.launch* gestartet. Die Ergebnisse der Simulation sieht man in Abblildung 7.43. Der Laserstrahl trifft deutlich näher als eine halbe Feldkantenlänge vom Ziel entfernt auf das Brett auf. Um die Tiefe abzuschätzen wurde der Lauf mit zwei blauen Markierungen vor und hinter dem Arbeitspunkt des Lasers wiederholt. Diese haben zum Arbeitspunkt jeweils einen Abstand von einem Zentimeter und einen Durchmesser von drei Millimetern. Die Abbildung 7.44 steht exemplarisch für alle Ecken. Die zweite blaue Markierung ist nicht zu sehen, dies bedeutet, dass der Abstand des Arbeitspunktes zum Brett kleiner



(a) oben, links

(b) oben, rechts

(c) unten, links

(d) unten, rechts

Abbildung 7.43: A posteriori Modell

als neun Millimetern ist.

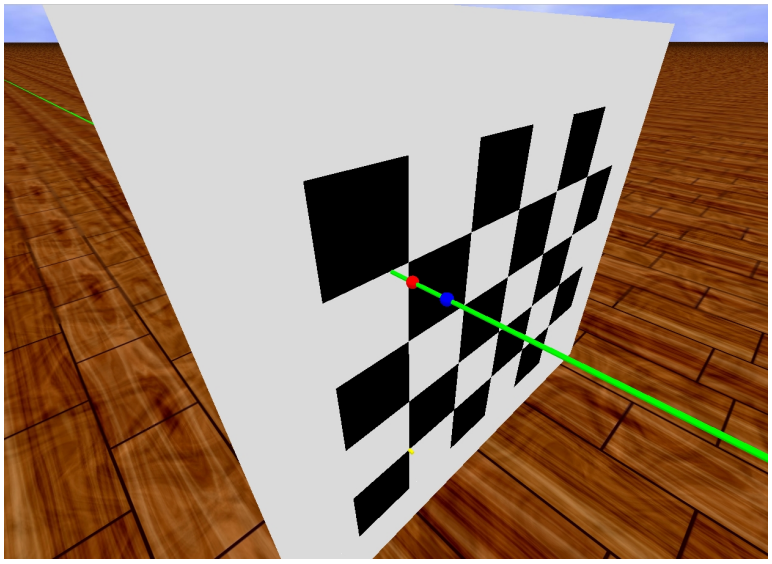


Abbildung 7.44: A posteriori Modell mit Tiefe

7.7 Datenaufnahme Dynamaid

In diesem Abschnitt werden Kalibrierungen mit am realen Roboter erhobenen Daten betrachtet. Hierfür werden beim Einlernen der Stellungen und bei Selbstexplorationen des Roboters Daten aufgenommen. Die Kalibrierungen werden mit den gleichen Einstellungen wie in Kapitel 7.5.3 vorgenommen.

Feld	**Wert**
calibrate.max_iter	200
update_prior	70
dh_sigma_m	Alle Werte 0,1.

Wenn sich in den folgenden Experimenten die Fehler bei der Kalibrierung nicht stabilisieren, so wird ein größerer Wert für die Anzahl der Iterationen gewählt. Dieser wird dann angegeben.

Als a priori Modell wird *hubert_calib_simulation/urdf/cosero/cosero_real_h.urdf.xacro* verwendet. Die Analyse und die Kalibrierung werden mit den jeweils passend konfigurierten Startern *test004_analyse.launch* und *test004_calib.launch* durchgeführt.

Die Selbstexplorationen werden mit *test004_explore.launch* gestartet, welches für die jeweilige Datenaufnahme konfiguriert wird. Dabei werden die folgenden Parameter verwendet.

Feld	**Wert**
explorer.multi.joint_lim	0,06
explorer.multi.displacements.neck_pan	0,3
explorer.multi.displacements.neck_tilt	0,2

7.7.1 Selbstexploration Dynamaid, 327 Stellungen aus 71 Basisstellungen

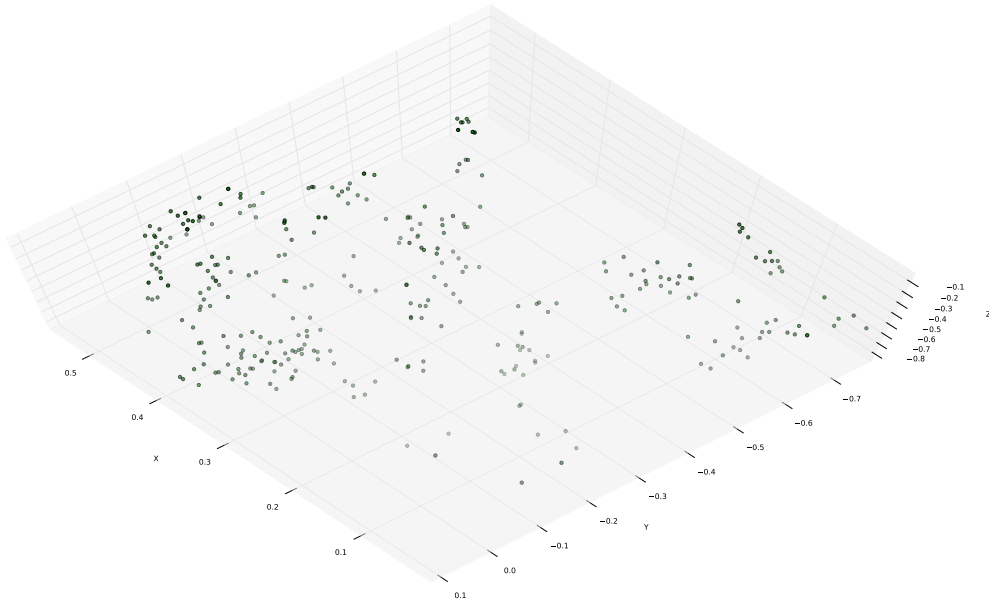


Abbildung 7.45: Selbstexploration Dynamaid, 327 Stellungen

Von vorhergehenden Versuchen mit Cosero werden Armstellungen übernommen, diese werden in *hubert_calib_exploration/config/dynamaid/right_arm* abgelegt. Mit diesen wird eine Exploration durchgeführt. Hierbei muss der Roboter bei einigen Stellungen manuell unterstützt werden, da die Aktuatoren an ihre Leistungsgrenze stoßen. Die aufgenommenen Daten werden in *bags/dynamaid_calibration_measurements_2012-03-15-17-11-49.bag* abgelegt. Die zugehörigen Schachbrettpositionen, welche sich aus der verwendeten VK ergeben, sind in Abbildung 7.45 zu sehen.

Die Berechnung der Fehler mit dem a priori Modell ergibt die folgenden Werte.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	45,577107	48,667993	85,989284
Orientierung [°]	8,296811	8,508713	13,481433

Eine Kalibrierung mit den Daten führt zu dem folgenden Ergebnis.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	11,316152	12,494196	37,224643
Orientierung [°]	2,422549	2,857265	8,082689

Das resultierende Modell wird in Abbildung 7.46 mit dem a priori Modell verglichen.

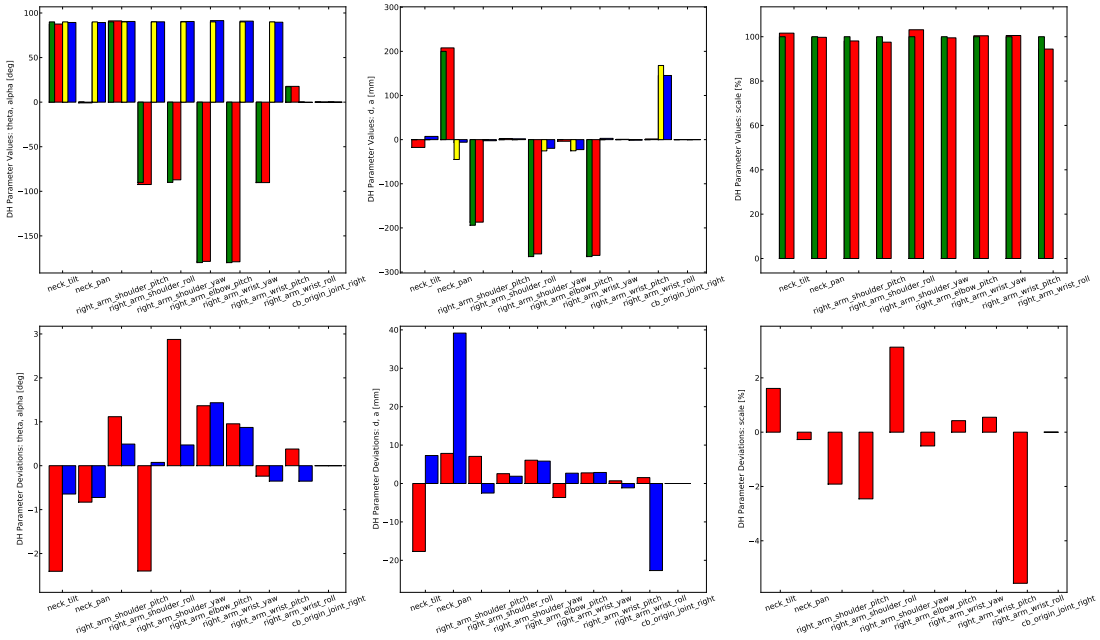


Abbildung 7.46: A posteriori Modell, Dynamaid, 327 Stellungen

7.7.2 Einlernen von 128 Stellungen bei Dynamaid

Für die spätere Selbstexploration werden Stellungen für den rechten Arm von Dynamaid eingelernt. Dabei richtet sich der Kopf automatisch auf die erwartete Position der Hand aus, welche manuell in Position gehalten wird. Wenn das im Greifer befindliche Schachbrett erkannt wird, werden die aktuellen Gelenkstellungen in eine Konfigurationsdatei übertragen. Zudem werden diese mit der erkannten Pose für einen Kalibrierungslauf aufgezeichnet. Die zugehörigen Schachbrettpositionen, welche sich aus der verwendeten VK ergeben, sind in Abbildung 7.47 zu sehen. Die Datenaufnahme wird mit *test004_-teach.launch* gestartet. Die Daten und Konfigurationen sind in *bags/dynamaid_calibration_measurements-manuell-2012-03-12.bag* und *config/dynamaid/right_arm_2012-03-12* abgelegt.

Die Berechnung der Fehler mit dem a priori Modell ergibt die folgenden Werte.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	52,037665	55,453636	97,405950
Orientierung [°]	8,031319	8,368831	15,463057

Eine Kalibrierung mit den Daten führt zu dem folgenden Ergebnis.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	10,023172	10,837082	22,848177
Orientierung [°]	3,185815	3,761789	11,587096

Das resultierende Modell wird in Abbildung 7.48 mit dem a priori Modell verglichen.

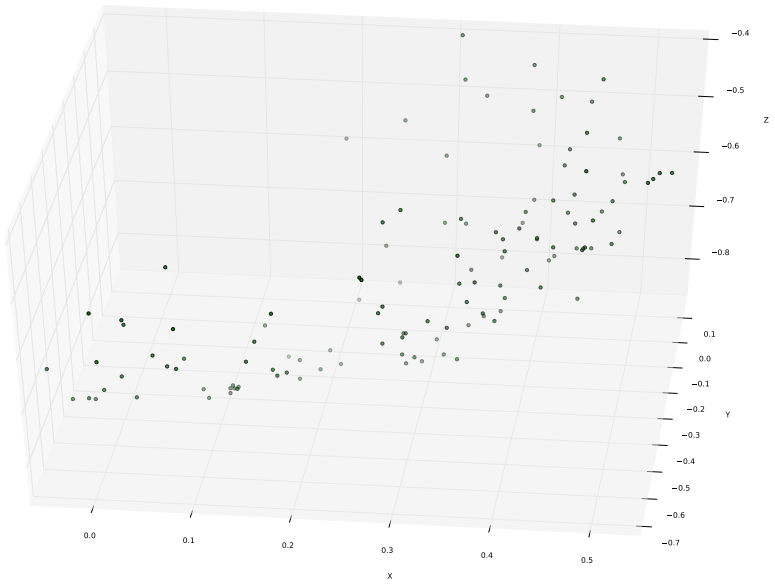


Abbildung 7.47: Einlernen von 128 Stellungen bei Dynamaid

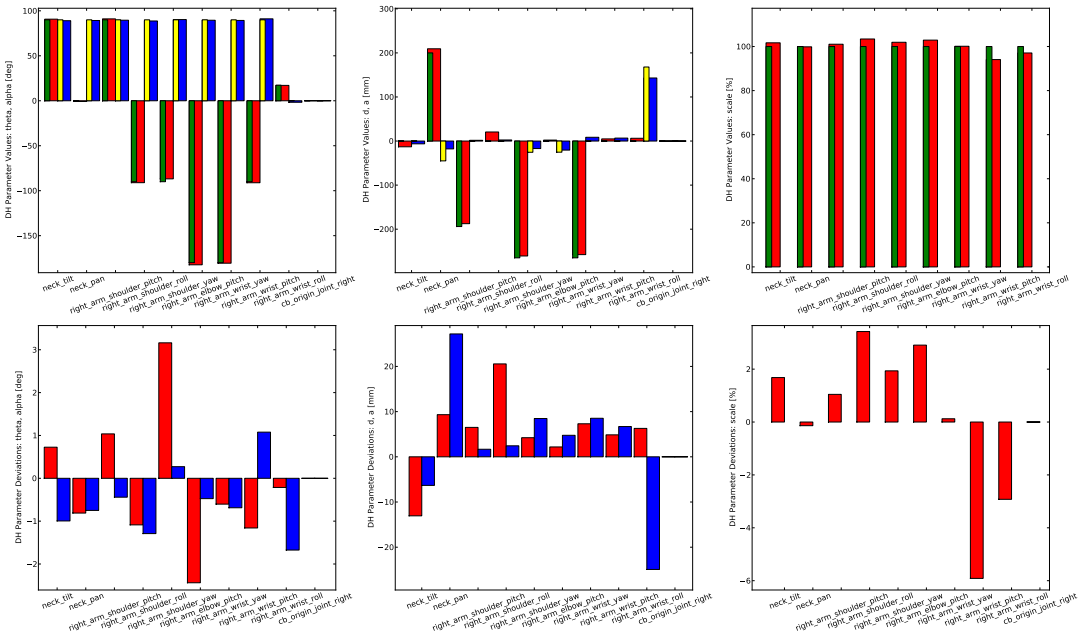


Abbildung 7.48: A posteriori Modell, Dynamaid, 128 Stellungen

7.7.3 Selbstexploration Dynamaid, 597 Stellungen aus 128 Basisstellungen

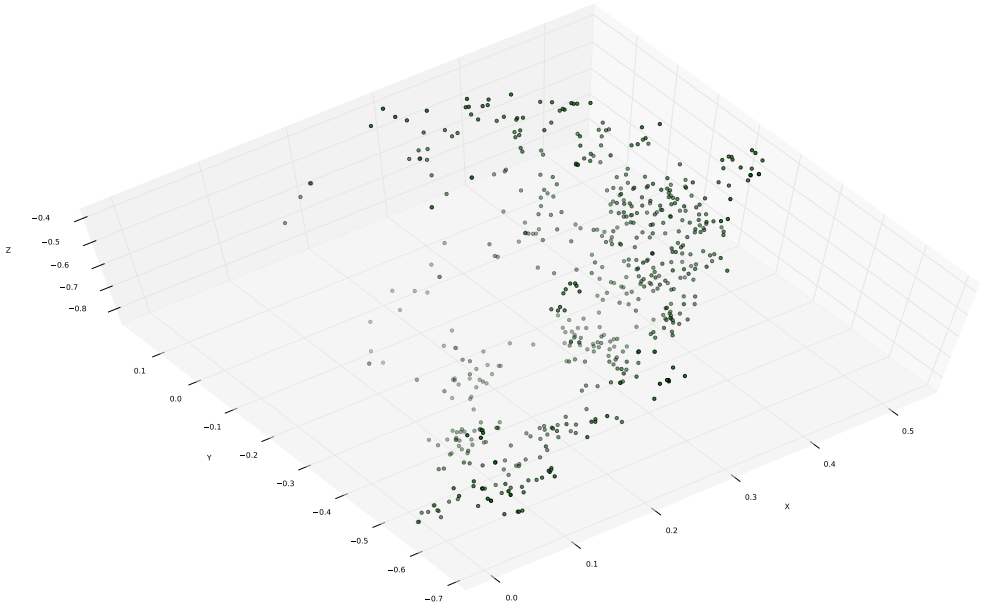


Abbildung 7.49: Selbstexploration Dynamaid, 597 Stellungen

Auf Basis von den in Kapitel 7.7.2 eingelernten Stellungen wird eine Exploration durchgeführt. Die aufgenommenen Daten befinden sich in *bags/dynamaid_calibration_-measurements_2012-03-15-17-06-38.bag*. Die zugehörigen Schachbrettpositionen, welche sich aus der verwendeten VK ergeben, sind in Abbildung 7.49 zu sehen.

Die Berechnung der Fehler mit dem a priori Modell ergibt die folgenden Werte.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	36,452048	39,113167	73,935698
Orientierung [°]	7,371626	7,637075	14,891346

Eine Kalibrierung mit den Daten führt zu dem folgenden Ergebnis.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	13,578112	14,514539	30,468489
Orientierung [°]	2,886163	3,347451	10,323044

Das resultierende Modell wird in Abbildung 7.50 mit dem a priori Modell verglichen.

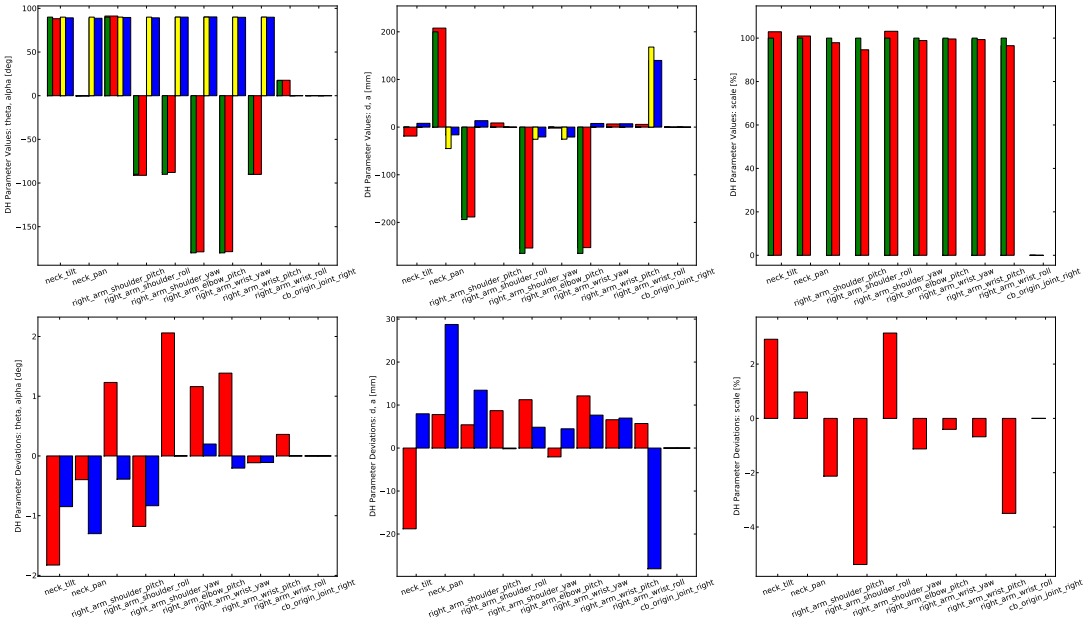


Abbildung 7.50: A posteriori Modell, Dynamaid, 597 Stellungen

7.7.4 Selbstexploration Dynamaid, 924 Stellungen aus 199 Basisstellungen

Wir fassen nun die Messungen aus Kapitel 7.7.1 und 7.7.3 zusammen. Das Resultat befindet sich in *bags/dynamaid_calibration_measurements_2012-03-17-13-46-56.bag*.

Die Berechnung der Fehler mit dem a priori Modell ergibt die folgenden Werte.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	39,681371	42,739495	85,989284
Orientierung [°]	7,699045	7,956469	14,891346

Eine Kalibrierung mit den Daten führt zu dem folgenden Ergebnis.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	13,299094	14,411484	39,756954
Orientierung [°]	2,774397	3,246576	10,939594

Das resultierende Modell wird in Abbildung 7.51 mit dem a priori Modell verglichen.

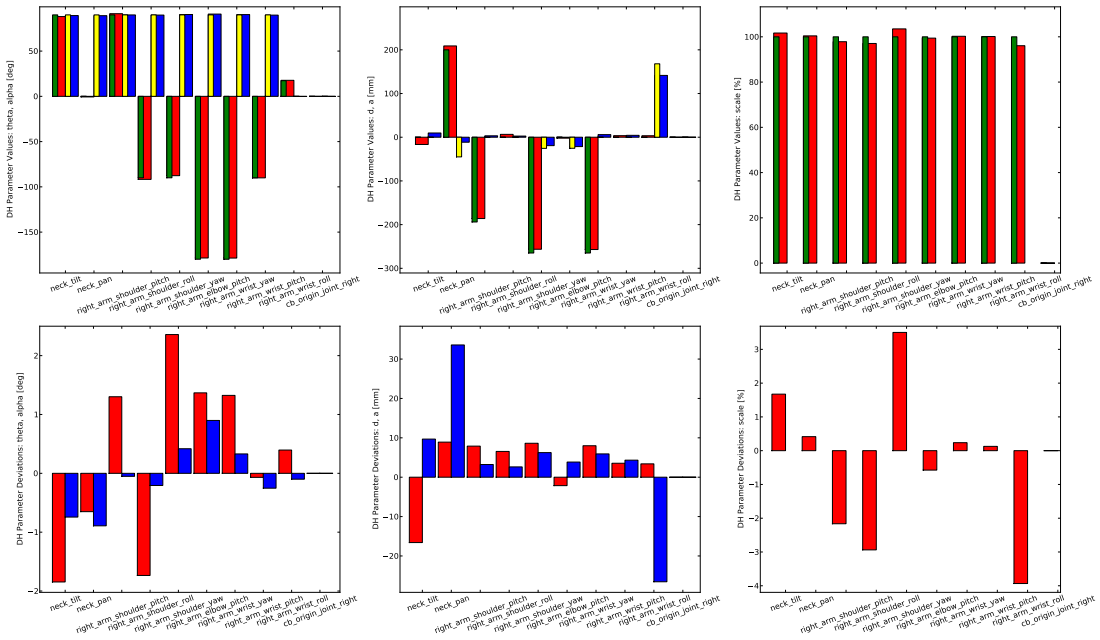


Abbildung 7.51: A posteriori Modell, Dynamaid, 924 Stellungen

7.7.5 Kalibrierung Dynamaid, Training 597 vs. Test 327

Nun evaluieren wir die Kalibrierung aus Kapitel 7.7.3 mit den Daten aus Kapitel 7.7.1 als Testdaten. Das Ergebnis ist wie folgt.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	14,940074	17,824887	59,378062
Orientierung [°]	2,900649	3,327872	9,382119

7.7.6 Kalibrierung Dynamaid, Training 597 vs. Test 128

Nun evaluieren wir die Kalibrierung aus Kapitel 7.7.3 mit den Daten aus Kapitel 7.7.2 als Testdaten. Das Ergebnis ist wie folgt.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	56,287490	57,219723	77,381293
Orientierung [°]	8,075103	8,510256	15,842854

7.7.7 Kalibrierung Dynamaid, Training 924 vs. Test 128

Nun evaluieren wir die Kalibrierung aus Kapitel 7.7.4 mit den Daten aus Kapitel 7.7.2 als Testdaten. Das Ergebnis ist wie folgt.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	56,887895	57,685915	76,693965
Orientierung [°]	7,952548	8,406646	16,068920

7.8 Anpassung der Lernparameter

Um den Einfluss des Auswendiglernens (Overfitting) zu betrachten, werden im folgenden die Varianzen der DH-Parameter variiert. Die restlichen Parameter sind wie in Kapitel 7.7. Wenn sich in den folgenden Experimenten die Fehler bei der Kalibrierung nicht stabilisieren, so wird ein größerer Wert für die Anzahl der Iterationen gewählt. Dieser wird dann angegeben. Als Trainingsdatensatz wird der aus Kapitel 7.7.7 verwendet und als Testdatensatz der aus Kapitel 7.7.1.

Ein der Anpassung zu Grunde liegender Ansatz besteht darin, dass bei den Getriebebeiwerten relativ geringe Schwankungen zu erwarten sind. Ein anderer Ansatz besteht darin, dass von θ-Parametern größere Schwankungen zu erwarten sind, da in sie der Stellversatz der Aktuatoren einfließt.

7.8.1 Getriebebeiwert geringer

Der Getriebeiwert wird weniger angepasst. Die Varianzen sind für alle Gelenke wie folgt.

θ	Beiwert	d	a	α
0,1	0,001	0,1	0,1	0,1

Gegenüber den Trainingsdaten erzeugt das a posteriori Modell die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	14,004544	14,982302	29,617160
Orientierung [°]	3,177575	3,597811	10,451492

Gegenüber den Testdaten erzeugt das a posteriori Modell die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	14,213435	16,283753	46,974655
Orientierung [°]	3,409701	3,733852	9,932341

7.8.2 DH-Parameter geringer

Die DH-Parameter werden weniger angepasst. Die Varianzen sind für alle Gelenke wie folgt.

θ	Beiwert	d	a	α
0,01	0,001	0,01	0,01	0,01

Gegenüber den Trainingsdaten erzeugt das a posteriori Modell die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	14,899609	15,903786	31,850501
Orientierung [°]	3,228723	3,645250	10,533954

Gegenüber den Testdaten erzeugt das a posteriori Modell die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	15,327001	17,097793	45,628946
Orientierung [°]	3,558682	3,888926	9,648511

7.8.3 θ-Parameter höher

Die θ-Parameter werden mehr angepasst. Die Varianzen sind für alle Gelenke wie folgt.

θ	Beiwert	d	a	α
0,1	0,001	0,01	0,01	0,01

Gegenüber den Trainingsdaten erzeugt das a posteriori Modell die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	14,764541	15,777534	31,553463
Orientierung [°]	3,235647	3,653842	10,424586

Gegenüber den Testdaten erzeugt das a posteriori Modell die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	14,561693	16,070994	39,018620
Orientierung [°]	3,446110	3,753992	9,843991

7.8.4 d-, a- und α-Parameter höher

Die d-, a- und α-Parameter werden mehr angepasst. Die Varianzen sind für alle Gelenke wie folgt.

θ	Beiwert	d	a	α
0,1	0,001	0,04	0,04	0,04

Gegenüber den Trainingsdaten erzeugt das a posteriori Modell die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	14,202450	15,176114	29,033884
Orientierung [°]	3,189796	3,609571	10,468866

Gegenüber den Testdaten erzeugt das a posteriori Modell die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	13,931455	15,780299	43,542928
Orientierung [°]	3,408170	3,728033	9,889042

7.8.5 d-, a- und α-Parameter geringer

Die d-, a- und α-Parameter werden weniger angepasst. Die Varianzen sind für alle Gelenke wie folgt.

θ	Beiwert	d	a	α
0,1	0,001	0,007	0,007	0,007

Gegenüber den Trainingsdaten erzeugt das a posteriori Modell die folgenden Fehler.

Abweichungen	Mittelwert	Effektivwert	Maximum
Position [mm]	14,964564	16,011707	32,859533
Orientierung [°]	3,253000	3,670771	10,389169

Gegenüber den Testdaten erzeugt das a posteriori Modell die folgenden Fehler.

Abweichungen	Mittelwert	Effektivwert	Maximum
Position [mm]	15,082534	16,596453	39,025728
Orientierung [°]	3,479624	3,787165	9,746059

7.8.6 Angepasste Parameter, 597 Stellungen

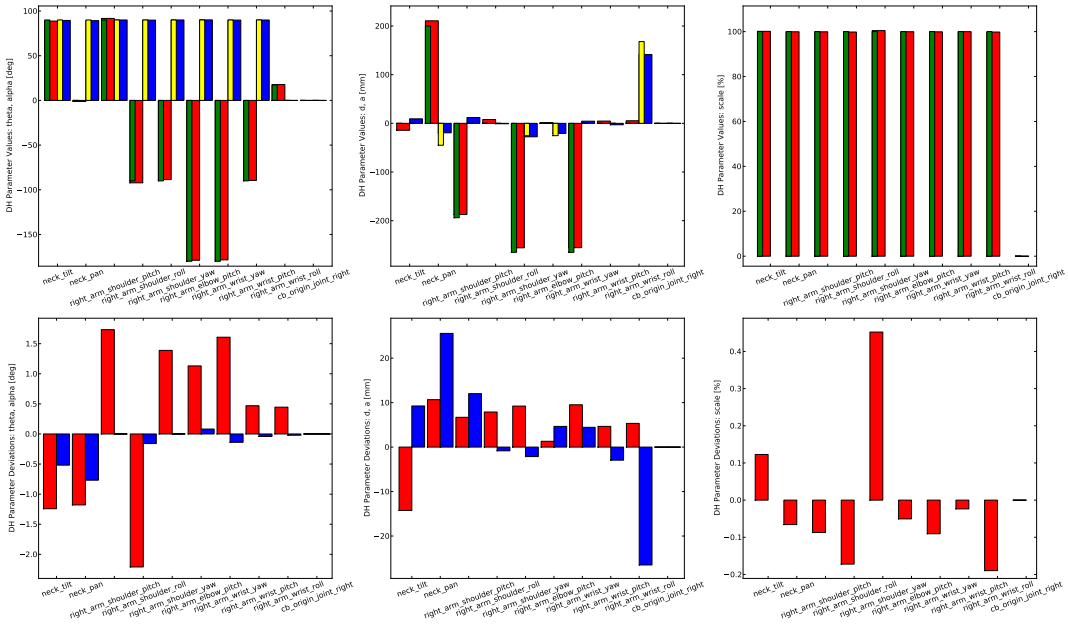


Abbildung 7.52: Kalibrierung, angepasst, 597 Stellungen

Wir betrachten nun das a Posteriori Modell für die Einstellungen aus Kapitel 7.8.3.

θ	Beiwert	d	a	α
0,1	0,001	0,01	0,01	0,01

Der Vergleich zwischen a priori Modell und a posteriori Modell ist in Abbildung 7.52 zu sehen. Ein Vergleich mit den manuell aufgenommenen Daten aus Kapitel 7.7.2 führt zu den folgenden Werten.

Abweichungen	Mittelwert	Effektivwert	Maximum
Position [mm]	57,409315	58,067345	75,327111
Orientierung [°]	8,209980	8,614706	16,005869

7.8.7 Angepasste Parameter, 327 Stellungen

Wir betrachten nun das a Posteriori Modell für die Einstellungen aus Kapitel 7.8.3 und den Trainingsdaten aus Kapitel 7.7.1.

θ	Beiwert	d	a	α
0,1	0,001	0,01	0,01	0,01

Gegenüber den Trainingsdaten erzeugt das a posteriori Modell die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	12,239680	13,462900	32.513120
Orientierung [°]	2,931553	3,314147	8,925494

7.8.8 Angepasste Parameter, 924 Stellungen

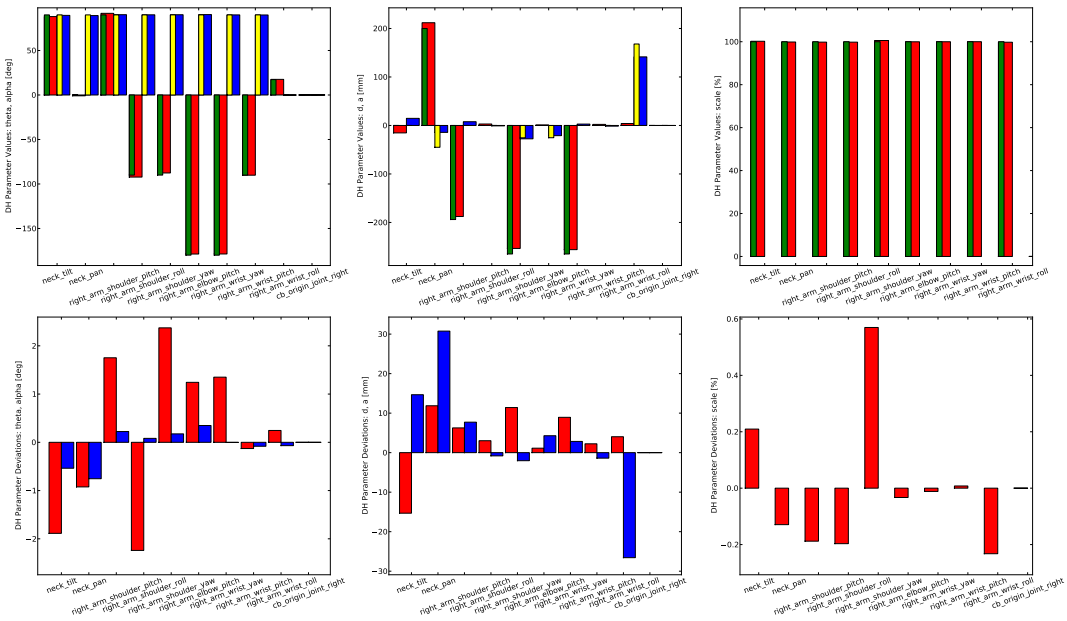


Abbildung 7.53: Kalibrierung, angepasst, 924 Stellungen

Wir betrachten nun das a Posteriori Modell für die Einstellungen aus Kapitel 7.8.3 und den Trainingsdaten aus Kapitel 7.7.4.

θ	Beiwert	d	a	α
0,1	0,001	0,01	0,01	0,01

Gegenüber den Trainingsdaten erzeugt das a posteriori Modell die folgenden Fehler.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	14,101509	15,248387	36,562398
Orientierung [°]	3,191344	3,579241	10,207867

Der Vergleich zwischen a priori Modell und a posteriori Modell ist in Abbildung 7.53 zu sehen.

Ein Vergleich mit den manuell aufgenommenen Daten aus Kapitel 7.7.2 führt zu den folgenden Werten.

Abweichungen	**Mittelwert**	**Effektivwert**	**Maximum**
Position [mm]	57,748145	58,436343	75,482142
Orientierung [°]	8,120134	8,525657	16,016191

7.9 Messung der Wiederholgenauigkeit von Dynamaid

Wir messen nun, wie gut der reale Roboter Dynamaid mit der wiederholten Ansteuerung der gleichen Gelenkstellungen die gleiche Endeffektorpose erreicht. Hierfür wird das Experiment entsprechend dem aus Kapitel7.6 durchgeführt. Das Schachbrett wird vor dem Roboter platziert und dessen Pose erfasst. Dann wird an Stelle des Schachbretts ein Blatt Papier befestigt. Mit Hilfe des a priori Modells wird dann die IK für die vier äußersten Kreuzungspunkte berechnet, so dass der Laserstrahl diese treffen soll. Das verwendete Modell liegt in *hubert_calib_simulation/urdf/dynamaid/dynamaid_real_h.urdf.xacro*. Diese Konfigurationen werden dann wiederholt angesteuert. Zwischendurch werden Endeffektorposen seitwärts oder vor dem Roboter angesteuert. Der Versuch wird mit *test002_eval_prior.launch* durchgeführt.

Hierbei zeigen sich beim Anfahren einer Konfiguration die folgenden Effekte. Es wird nicht immer der gleiche Punkt getroffen. Es werden aber immer nahezu die selben Punkte getroffen, zwischen denen mitunter auch hin und her gesprungen wird, während die Armstellung gehalten wird. Die maximalen Abweichungen beim Treffen dieser Punkte liegen bei zwei Millimetern. Die Abstände zwischen den äussersten dieser Punkten liegen in Wirkrichtung der Gravitation bei sieben bis elf Millimetern und quer zur Gravitation bei zwei bis vier Millimetern.

Auch im folgenden Kapitel 7.10 werden Untersuchungen zur Wiederholgenauigkeit vorgenommen.

7.10 Dynamaid richtet Laser auf Schachbrett

Der Aufbau des folgenden Versuchs entspricht dem aus Kapitel 7.6. Es werden Schachbrettmuster vor und neben dem Roboter in unterschiedlichen Posen aufgehängt. Für jede dieser Schachbrettposen wird das a priori Modell und das a posteriori Modell evaluiert. Dafür wird das Schachbrett mit der Kamera monoskopisch lokalisiert und der Laser nacheinander auf die äußeren Kreuzungspunkte gerichtet. Die Ausrichtung des Lasers auf die vier Punkte wird für jedes Modell zweimal durchgeführt. Die Lokalisierung wird ein oder zweimal durchgeführt, zu Beginn und eventuell nach dem ersten Ansteuern der vier Ecken. Die Stelle an der der Laser das Schachbrett trifft wird manuell markiert. Die Ansteuerung der Punkte findet entweder aus der Ruheposition oder von einem der anderen Punkte aus statt.

Es werden die Modelle aus Kapitel 7.8.8 verwendet. Um die Kalibrierung zu starten wird *test002_calib.launch* verwendet, die Evaluation des a priori Modells wird mit *test002_eval_prior.launch* gestartet und die des a posteriori Modells mit *test002_eval_belief.launch*.

Generell zeigt sich, dass der mit Hilfe des a priori Modells angefahrene Punkt meist links unterhalb von dem Punkt liegt, der durch das a posteriori Modell angefahren wird. In allen Fällen gilt, dass das Dreiek, welches durch diese beiden angefahrenen Punkte und das Ziel definiert wird, seine längeste Seite in der Verbindungslinie zwischen den beiden angefahrenen Punkten hat.

Die einzelnen Abstände sind in Tabelle A.2 aufgeführt. Für die unteren Ecken vom 6. Lauf findet sich keine Lösung für die IK des a posteriori Modells, diese Ecken werden bei der Berechnung der Kennzahlen ganz ignoriert. Es ergeben sich die folgenden Kennzahlen.

Modell	**Mittelwert [mm]**	**Effektivwert [mm]**	**Maximum [mm]**	**Standard-abweichung [mm]**
a priori	34,4210526316	36,5365714406	60	12,3334376477
a posteriori	22,2368421053	24,463721798	40	10,2656299317

Die Werte zur Wiederholgenauigkeit sind in den Tabellen A.3 und A.4 aufgeführt. Hirtbei wird der Abstand der Punkte für ein Modell gemessen. In Tabelle A.3 sind die werte für die Einmalige Posenbestimmung zu finden. Der Roboter soll hier zweimal exakt die gleichen Gelenkstellungen anfahren.

Die Fälle mit zwischenzeitlicher Neubestimmung der Schachbrettpose sind in Tabelle A.4 angegeben. Hier fliesst der Fehler bei der Posenbestimmung in das Ergebnis mit ein. Die resultierenden Kennzahlen sind wie folgt.

Posen-bestim-mungen	**Mittelwert [mm]**	**Effektivwert [mm]**	**Maximum [mm]**
Eine	3,4838709677	4,075892936	8
Zwei	5,5625	6,552671211	12

7.11 Lokal gewichtetes Lernen

Nun untersuchen wir das lokalisierte Lernen der Kinematik. Dabei werden die einzelnen Paare aus Schachprettposen und Gelenkstellungen nach ihrem Abstand zum Zentrum eines Teilraums im Gelenkraum mit einer Gaussfunktion gewichtet. Siehe hierfür auch Kapitel 5.13.

Für unsere Untersuchung wählen wir jede Konfiguration als Zentrum aus und erstellen für dieses ein a posteriori Modell. Danach berechnen wir für jedes Modell und jede Stellung den karthesischen Abstand der Stellung zu dem Zentrum, für das das Modell gelernt wurde, und die Fehler zwischen erwarteter und gemessener Schachbrettpose.

Zudem wird die Entwicklung der kinematischen Parameter betrachtet. Dabei werden

diese mit einer ungewichteten Kalibrierung verglichen. Für eine Beschreibung der Darstellungen vergleiche Kapitel 6.2. Es wird der Starter *test004_analyse.launch* verwendet.

7.11.1 924 Stellungen, Varianz 0,3

Wir führen den beschriebenen Aufbau mit den Daten und Parametern aus Kapitel 7.8.8 durch. Zussätzlich setzen wir die folgenden Parameter.

Feld	**Wert**
analyse.localized.iterations	70
analyse.localized.variance	0,3

Die Entwicklung der mittleren und maximalen Positions- und Orientierungsfehler sind in Abbildung 7.54 zu sehen. Die Mittelwerte sind zudem vergrößert in Abbildung 7.55 dargestellt. Die kinematischen Parameter werden in Abbildung 7.56 gezeigt. Die Fehler der einzelnen Modelle sind in Abbildung 7.57 dargestellt.

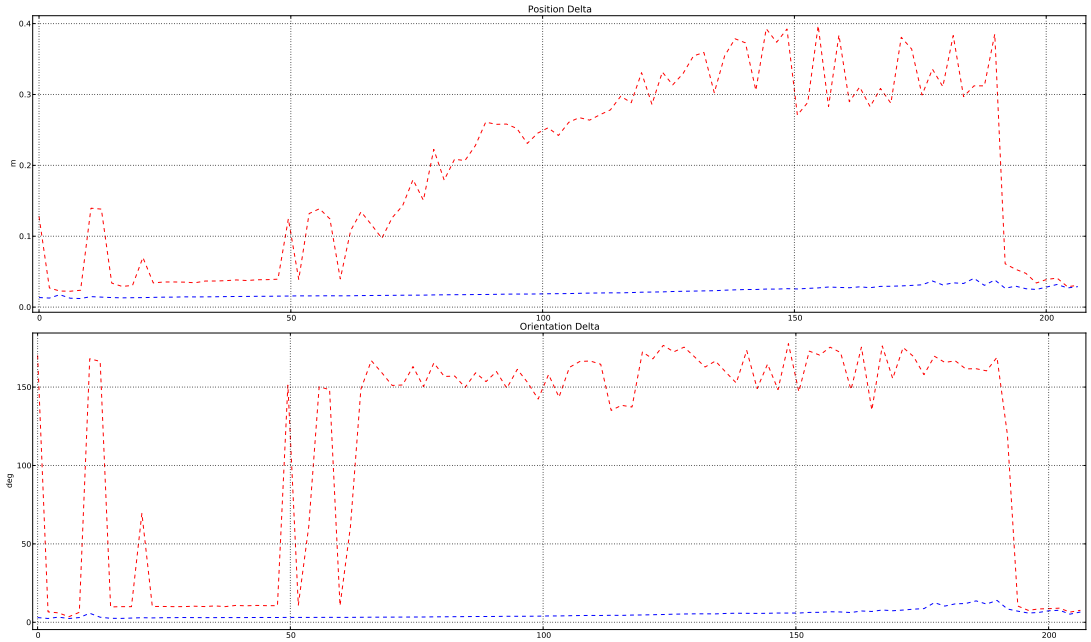


Abbildung 7.54: Fehler beim lokalisierten Lernen, Dynamaid, 924 Stellungen, Varianz 0,3

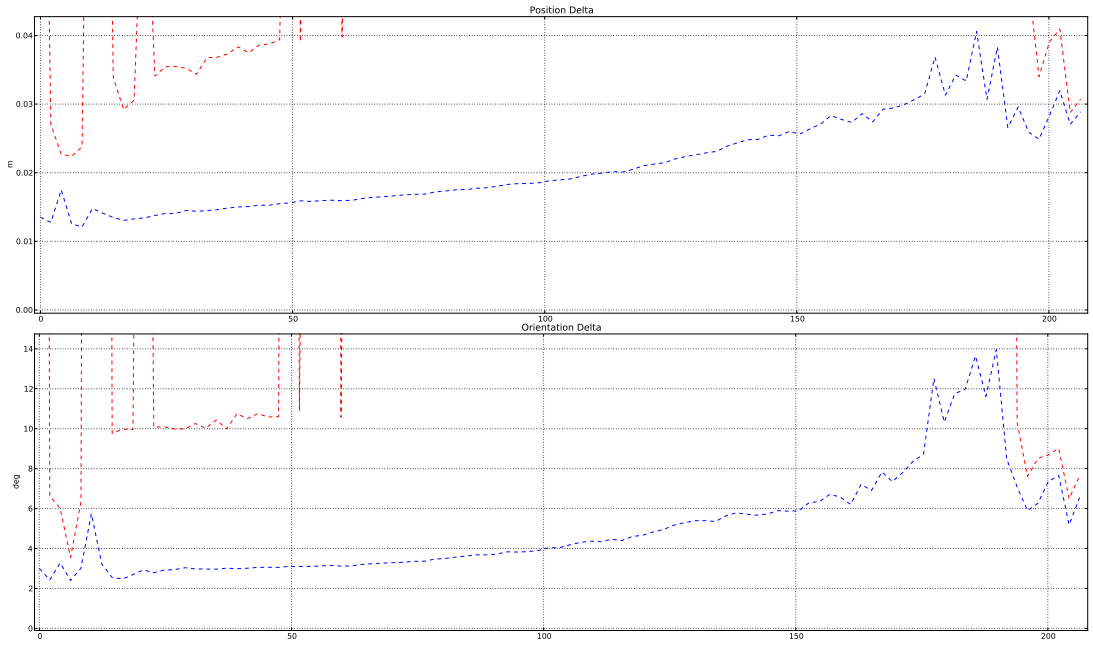


Abbildung 7.55: Mittelwerte der Fehler beim lokalisierten Lernen, Dynamaid, 924 Stellungen, Varianz 0,3

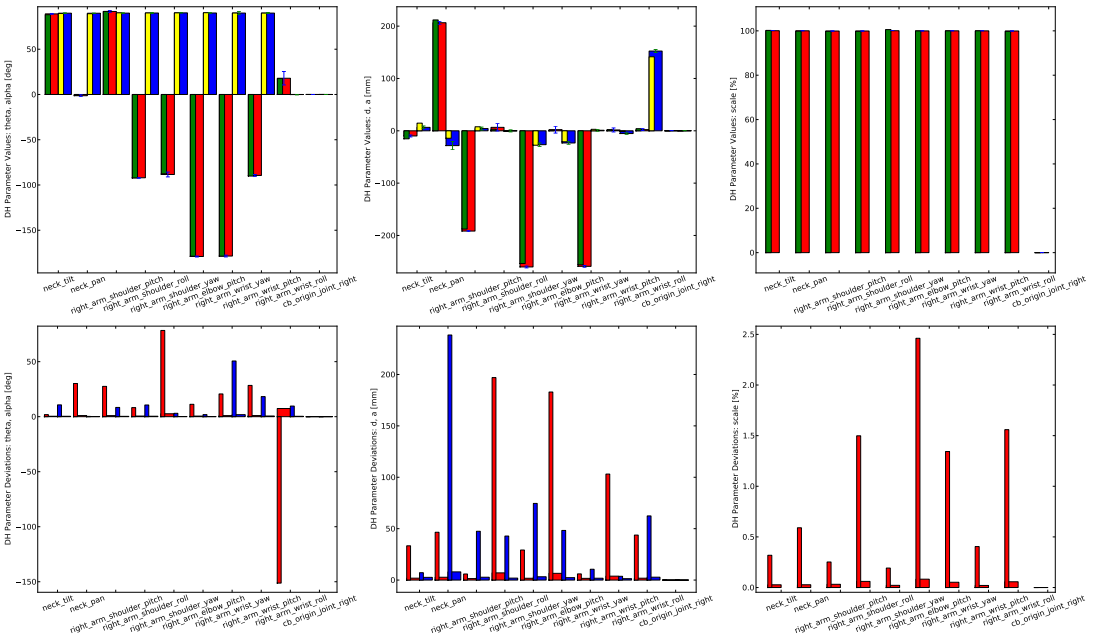
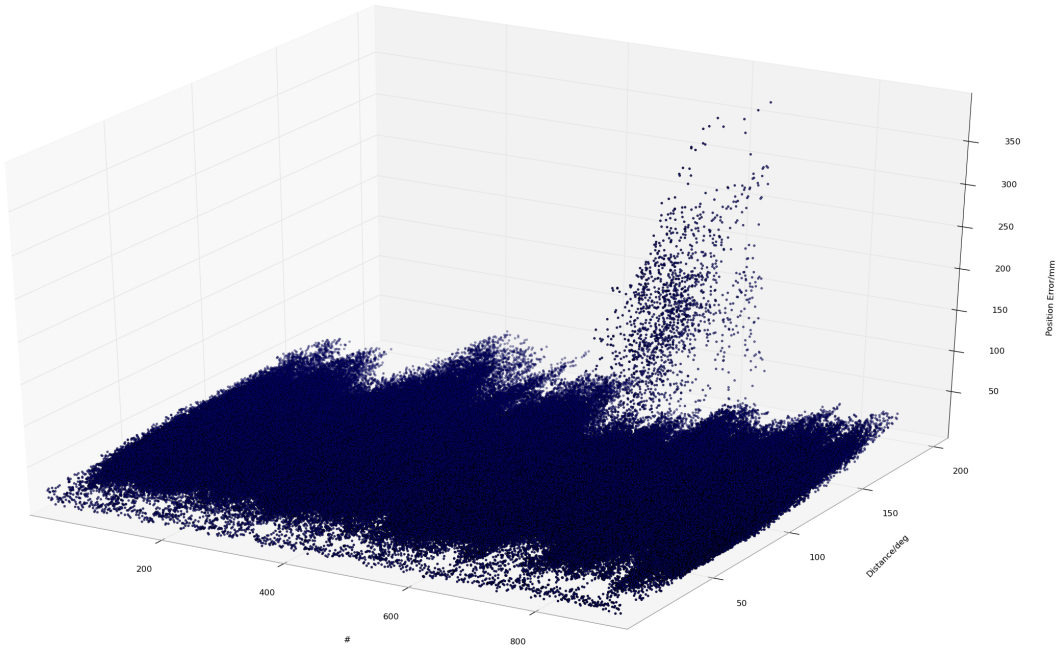
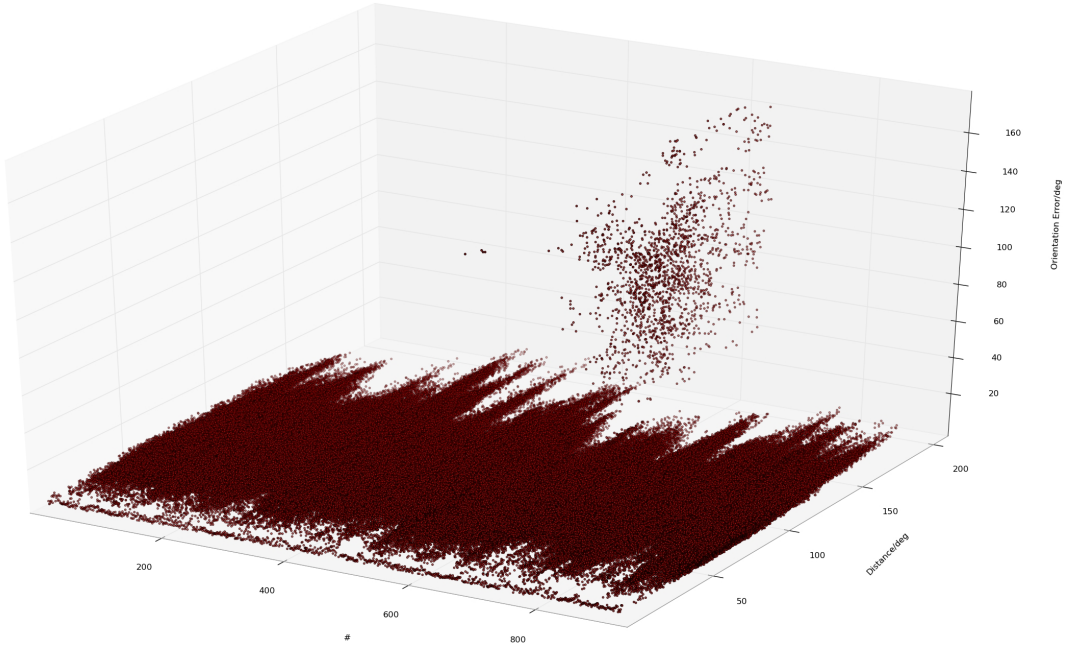


Abbildung 7.56: Parameter beim lokalisierten Lernen, Dynamaid, 924 Stellungen, Varianz 0,3



(a) Position



(b) Orientierung

Abbildung 7.57: Einzelne Fehler bei lokalisiertem Lernen, 924 Stellungen, Varianz 0,3

7.11.2 327 Stellungen, Varianz 0,3

Der gleiche Aufbau wie in Kapitel 7.11.1 wird mit den Daten aus Kapitel 7.7.1 durchgeführt.

Die Entwicklung der mittleren und maximalen Positions- und Orientierungsfehler sind in Abbildung 7.58 zu sehen. Die Mittelwerte sind zudem vergrößert in Abbildung 7.59 dargestellt. Die kinematischen Parameter werden in Abbildung 7.60 gezeigt. Die Fehler der einzelnen Modelle sind in Abbildung 7.61 dargestellt.

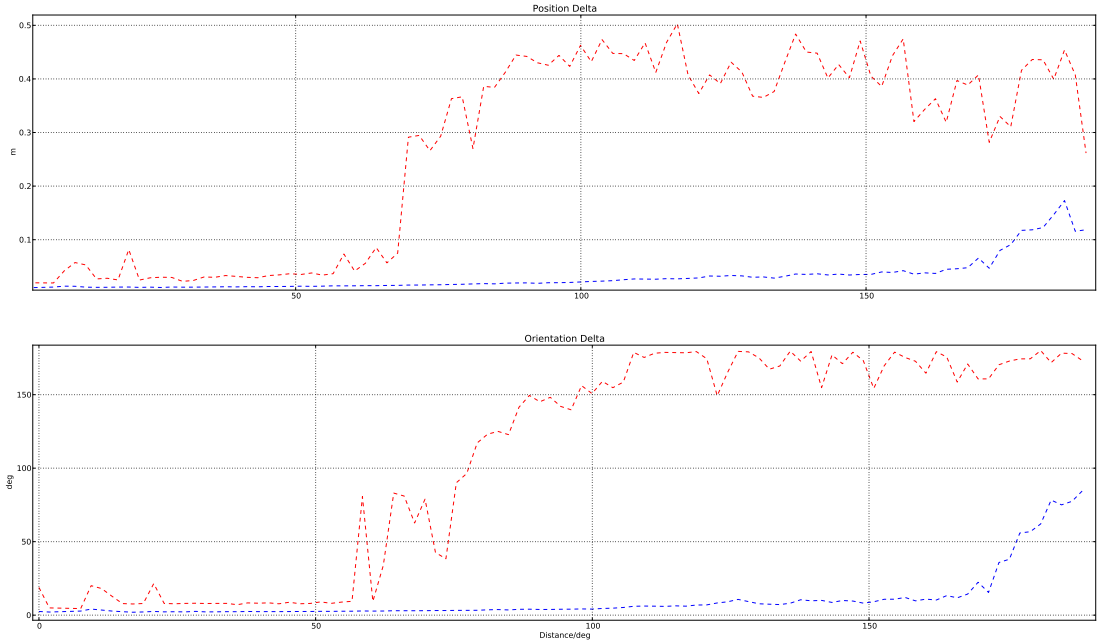


Abbildung 7.58: Fehler beim lokalisierten Lernen, Dynamaid, 327 Stellungen, Varianz 0,3

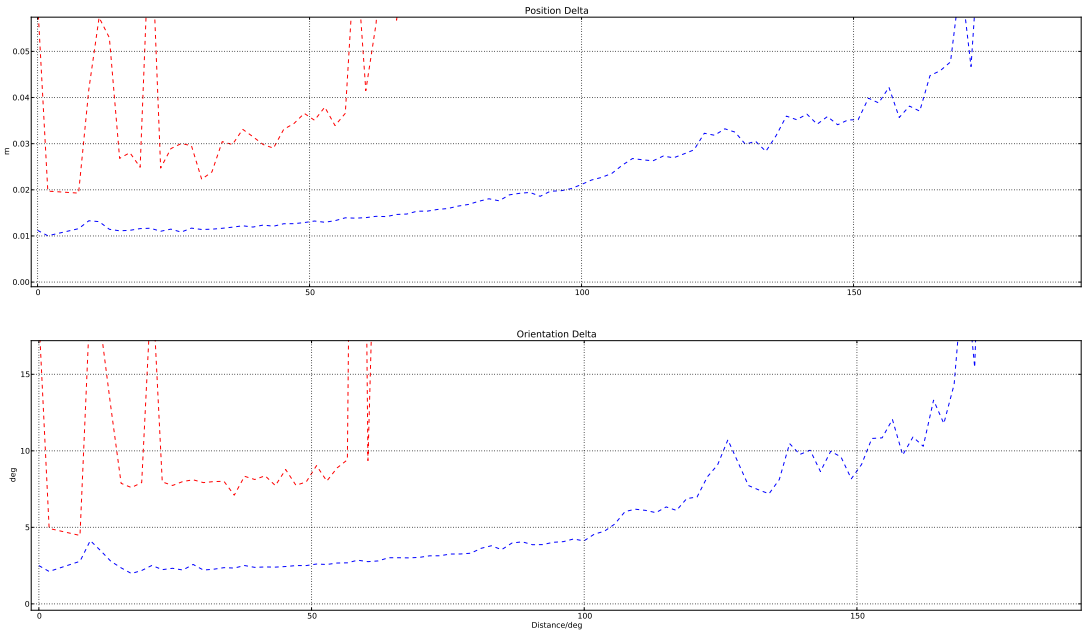


Abbildung 7.59: Mittelwerte der Fehler beim lokalisierten Lernen, Dynamaid, 327 Stellungen, Varianz 0,3

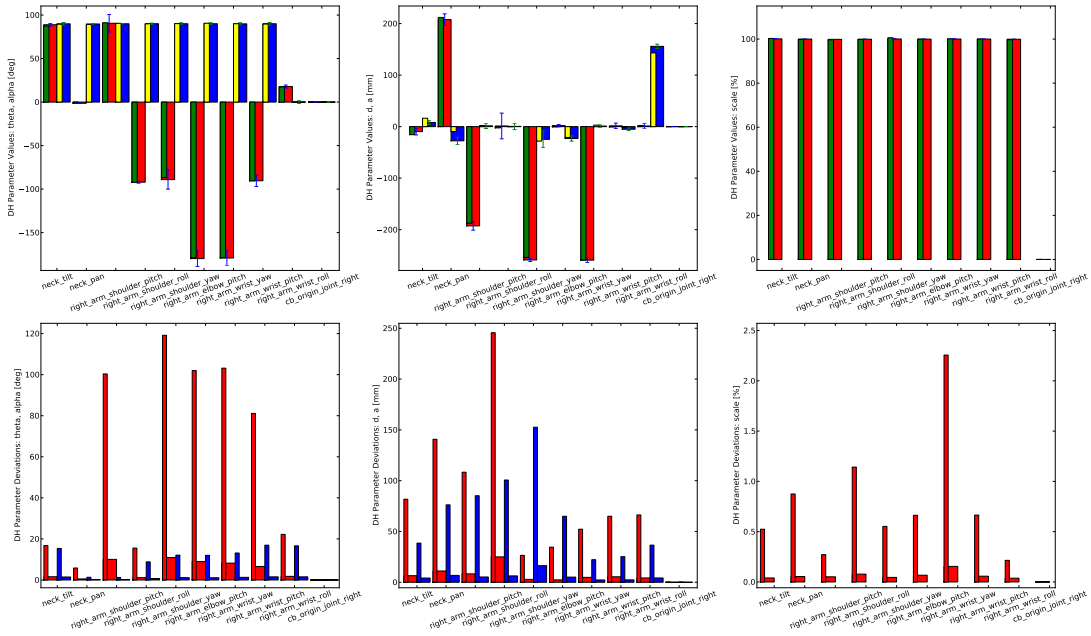
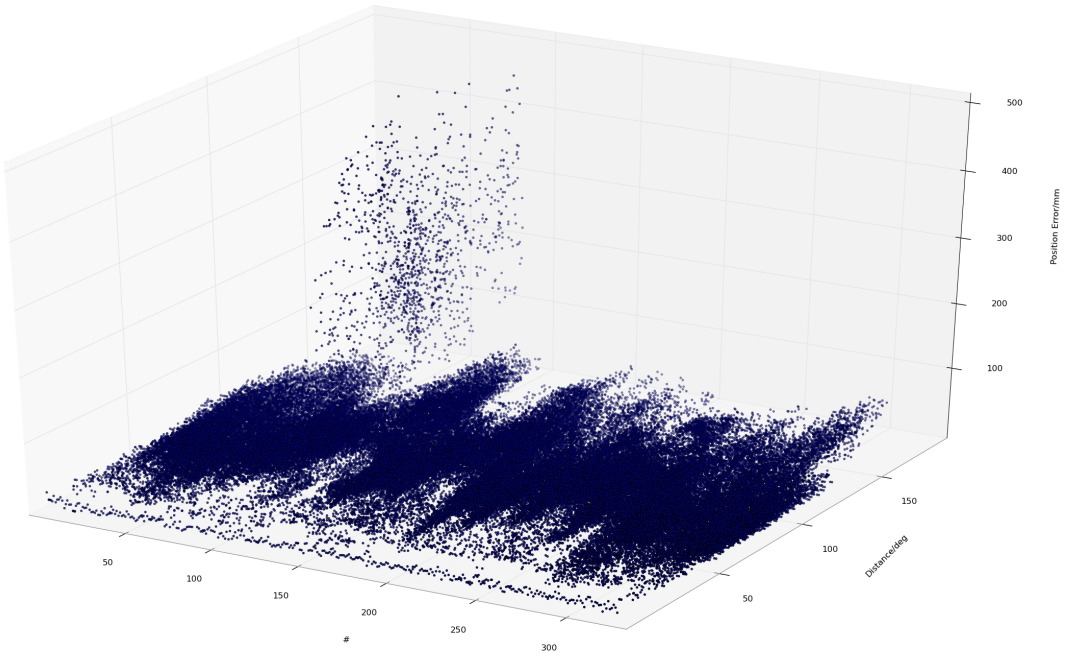
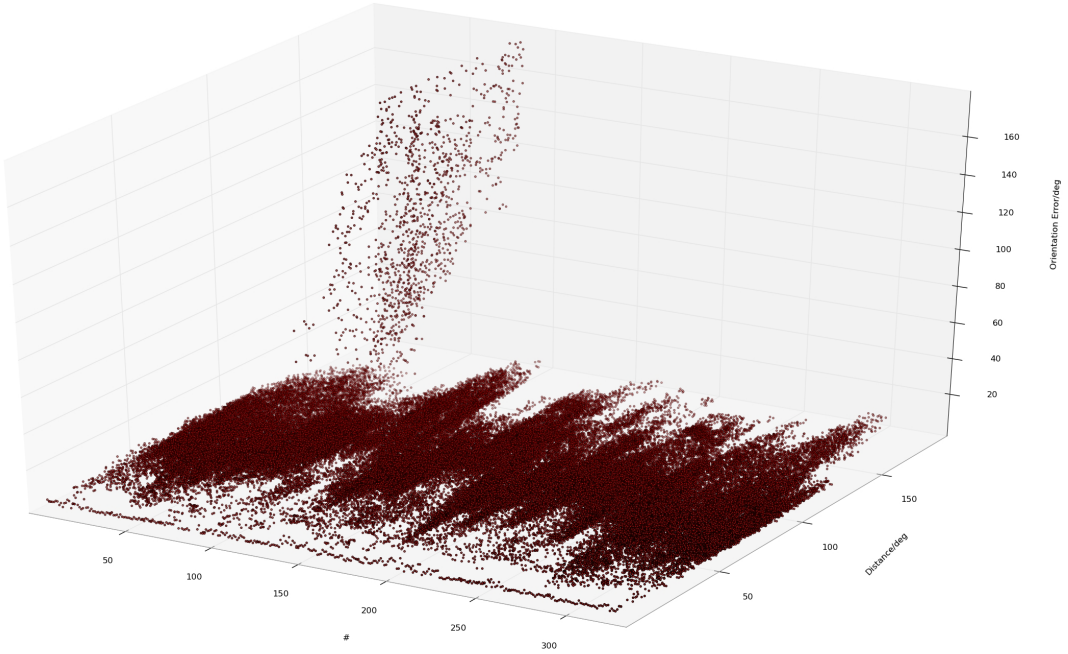


Abbildung 7.60: Parameter beim lokalisierten Lernen, Dynamaid, 327 Stellungen, Varianz 0,3



(a) Position



(b) Orientierung

Abbildung 7.61: Einzelne Fehler bei lokalisiertem Lernen, 327 Stellungen, Varianz 0,3

7.11.3 327 Stellungen, Varianz 0,1

Wir wiederholen den Versuch aus Kapitel 7.11.2 mit geänderter Varianz der Abstandsfunktion.

Feld	Wert
analyse.localized.variance	0,1

Die Entwicklung der mittleren Positions- und Orientierungsfehler sind in Abbildung 7.62 zu sehen.

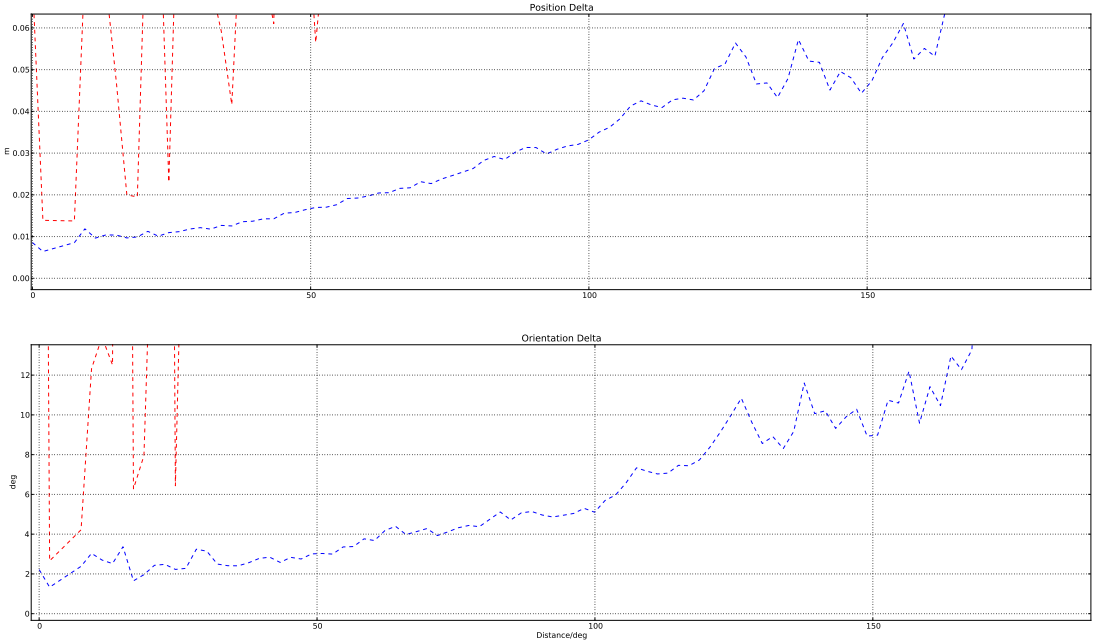


Abbildung 7.62: Mittelwerte der Fehler beim lokalisierten Lernen, Dynamaid, 327 Stellungen, Varianz 0,1

8 Auswertung

In diesem Kapitel werden die Ergebnisse aus Kapitel 7 in Beziehung gesetzt und Schlussfolgerungen abgeleitet.

8.1 Referenzwerte für die Posenbestimmung

In Kapitel 7.1 befassen wir uns mit der Abschätzung der Genauigkeit bei der visuellen Posenbestimmung. In Kapitel 7.1.1 wird ein Schachbrett mit 10×10 mm Feldern mit einer Kamera mit 1600×1200 Pixeln verwendet. Bei gleicher Brennweite entspricht das Verhältnis der Schachbrettfelder und der Pixel, dem der verwendeten Kinect, 25×25 mm Felder bei 640×480 Pixeln. Die Posenbestimmung arbeitet somit bei beiden Kameras mit vergleichbaren Bilddaten. Es ist dennoch davon auszugehen, dass die Kinect verlässlichere Posenbestimmungen zulässt, da sich das Schachbrettmuster über einen größeren Anteil des Bildes erstreckt. Zudem ist zu erwarten, dass die Kinect bessere Ergebnisse liefert, da ihr Konstruktionsziel die Erfassung von Punktwolken ist. Es ist somit gerechtfertigt, die gemessenen Werte als Untergrenze für die Genauigkeit zu betrachten.

Die in den Kapiteln 7.1.2 und 7.1.3 im Simulator gemessenen Werte bei vorgegebenen Kameraparametern sind hingegen als Obergrenze der Genauigkeit zu betrachten, da hier keine Fehler in der Kamerakalibrierung vorkommen. Wobei es im Fall des PR2 zu stärkeren Schwankungen in der Genauigkeit kommt. Die Limitierung besteht hier in der Auflösung der Kamera.

Die Posenbestimmung des Schachbretts kann als Referenz für die Genauigkeit der Objektlokalisierung im Allgemeinen verwendet werden. Es ist nicht zu erwarten, dass mit einzelnen Kamerabildern der gegebenen Auflösung deutlich präziesere Bestimmungen der Pose eines Objekts vorgenommen werden.

8.2 10000 im Gelenkraum gleichverteilte Stellungen

In Kapitel 7.2 befassen wir uns mit der Kalibrierung der Kinematik mit Hilfe einer großen Anzahl an Messdaten. In den Kapiteln 7.2.1, 7.2.2 und 7.2.3 ist der Einfluss des Rauschens auf ein unverfälschtes Modell zu sehen. Es zeigt sich, dass die Verfälschungen der kinematischen Parameter die gleiche Qualität haben, für jeden Parameter gehen sie in die gleiche Richtung. Es bleiben auch die Proportionen zwischen den Fehlern der verschiedenen Parameter erhalten. Somit zeigt sich, dass bei größerer angenommener Varianz der Parameter, die Abweichungen der a posteriori Modelle vom realen Modell stärker werden. Dies bestätigt die Erwartungen an die Funktion des Algorithmus, dass der Einfluss der Messungen und Messfehler steigt. Weiter ist zu sehen, dass die sehr hoch

gewählte Varianz in Kapitel 7.2.2 zu sehr ähnlichen Ergebnissen führt wie eine deutlich geringere Varianz in Kombination mit dem regelmäßigen Austausch des a priori Modells durch das a posteriori Modell, wie in Kapitel 7.2.3. Ein signifikante Verringerung der Abweichungen zum Trainingsdatensatz ist nicht fest zu stellen. Auch dies entspricht den Erwartungen. Es zeigt sich sogar eine kleine Verschlechterung bei den maximalen Positionsfehlern gegenüber dem Trainingsdatensatz. Gegenüber dem Testdatensatz zeigen sich Verschlechterungen, wobei alle drei Kalibrierungen zu ähnlichen Fehlern führen. Selbst bei den sehr gering gewählten Lernparamatern aus Kapitel 7.2.1 beträgt der maximale Positionsfehler circa 80% von diesen Fehlern bei den höheren Lernintensitäten. Bei den Orientierungen werden hier circa 60% erreicht. Insgesamt sind aber sowohl die Abweichungen in den kinematischen Parametern als auch die Differenzen zwischen erwarteter Pose und gegebener Pose gering. Letztere liegen deutlich unter den Abschätzungen für die Fehler der Posenbestimmung aus Kapitel 7.1.

In den Kapiteln 7.2.4, 7.2.5,7.2.6, 7.2.7 und 7.2.8 betrachten wir den Fall, dass ein einzelnes Gelenk in seinen Parametern manipuliert wird. Bei einer angenommenen Varianz aller Parameter von $0,01$, Kapitel 7.2.4, findet eine deutliche Verbesserung bei den Abweichungen statt, diese liegen aber noch deutlich über den minimalen Werten aus 7.1. Es ist anzunehmen, dass diese Varianz für eine gute Anpassung im Allgemeinen zu klein ist. Die Anpassung ist bei Varianzen von $0,1$ deutlich besser. Wobei man im Vergleich zwischen Kapitel 7.2.5 und Kapitel 7.2.6 sieht, dass eine Steigerung der Iterationen nicht zu einem Verlassen des gefundenen lokalen Minimums führt. Alle Werte bleiben nahezu gleich.

In Kapitel 7.2.7 zeigt sich zwar eine recht gute Voraussage der Endeffektorpose, doch bleiben dennoch deutliche Abweichungen im kinematischen Modell erhalten. Am besten ist die Anpassung in Kapitel 7.2.8. Hier sind fast keine Unterschiede zur gleich parametrisierten Kalibrierung des unveränderten Modells aus Kapitel 7.2.3 zu finden. Die Parameter in den Tabellen 7.8 und 7.15 sind nahezu gleich. Auch die Abweichungen zwischen den Posen sind sehr ähnlich. Diese Kalibrierungen konvergieren anscheinend zum selben Modell. Es ist nicht anzunehmen, dass mit dem beschriebenen Ansatz und den gegebenen Daten ein besseres a posteriori Modell erzeugbar ist. Hier wird die Kalibrierung durch die Messdaten dominiert.

In Kapitel 7.2.9 und 7.2.10 wird ein a priori Modell untersucht, in dem alle Parameter zufällig verändert wurden. Es zeigt sich in diesen Kapiteln der gleiche Effekt wie zuvor in Kapitel 7.2.7, die Kalibrierung konvergiert gegen die gleichen Posenfehler und kinematischen Parameter wie beim unveränderten Modell. Es zeigt sich aber, dass der Lernerfolg größer ist, wenn das a priori Modell ausgetauscht wird und die Varianz kleiner angenommen wird, als wenn eine große Varianz der kinematischen Parameter angenommen wird und das a priori Modell beibehalten wird. Wobei in beiden Fällen die Vorhersagegenauigkeit der Kinematik deutlich besser ist, als die Werte aus Kapitel 7.1. Die in den Kapiteln 7.2.3, 7.2.8 und 7.2.9 verwendeten Kalibrierungsparameter erscheinen als gute Wahl für die gegebenen Daten.

8.3 370 im Gelenkraum gleichverteilte Stellungen

In Kapitel 7.3 untersuchen wir die Kalibrierung mit einer leicht zu erhebenden Menge an Daten. Hier wiederholen sich die Erkenntnisse aus Kapitel 8.2. Bei nicht manipulierten Modellen führen unterschiedliche Lernparameter zu Entwicklungen in die gleiche Richtung. Bei den hier verwendeten Trainingsdaten zeigt sich zudem eine deutliche Verschlechterung des a posteriori Modells gegenüber dem realen. Die Werte differieren in Kapitel 7.3.2 deutlich stärker als in Kapitel 7.2.2. Der Positionsfehler übersteigt das sechsfache, der Orientierungsfehler das dreifache der ursprünglichen Werte. Bei den kinematischen Paramtern werden die Fehler der Winkel und Beiwerte verdoppelt und die der Abstände versiebenfacht. Ein ähnliches Bild zeigt sich bei den manipulierten a priori Modellen.

Es zeigt sich, das auch hier alle vier a priori Modelle gegen das gleiche Modell konvergieren. Dies trotz der Tatsache, dass das a priori Modell aus Kapitel 7.3.6 massive Abweichungen zum realen Modell hat, welche zu maximalen Abweichungen zu den Testdaten von fast 90 cm und 110 ° führen.

Auch ist der Einfluss einer sehr großen angenommenen Varianz zu sehen, die Fehler im Vergleich zu den Testdaten sind in Kapitel 7.3.3 minimal höher als in den Kapiteln 7.3.4, 7.3.5 und 7.3.6. Dies kann als Indiez für Overfitting angesehen werden.

Insgesamt ergeben sich hier im Mittel noch vertretbare Posenabweichungen in der Größenordnung von Kapitel 7.1. Die maximalen Fehler liegen allerdings schon über dem Doppelten.

8.4 370 in den erlaubten Grenzen gleichverteilte Stellungen

In Kapitel 7.4 sieht man deutlich die Auswirkungen von Overfitting. Dabei wird das Modell zu stark an ungünstig verteilte Daten angepasst. In den Kapiteln 7.4.1, 7.4.2 und 7.4.4 kann man sehen, dass sich die kinematischen Parameter in die gleiche Richtung entwickeln. Die Letzteren sind sogar fast gleich, mit recht großen Fehlern gegenüber den Testposen.

Die kinematischen Parameter der a posteriori Modelle aus den Kapiteln 7.4.1 und 7.4.3 weisen hingegen sichtbare Differenzen auf. Im Gegenzug haben sie aber kleinere Fehler gegenüber den Testdaten als die beiden anderen Modelle. Da das a posteriori Modell, welches aus dem unmanipulierten Modell aus Kapitel 7.4.2 entstanden ist einen größeren Fehler aufweist, als das aus Kapitel 7.4.3 liegt hier eindeutig ein Fall von Overfitting vor. Bei einer angenommenen Varianz der Parameter von 0,5 werden die Fehler in den Daten auswendig gelernt.

Die vorliegenden Positionsfehler übertreffen hier schon die Minimalwerte der Posenerkennung aus 7.1 deutlich. Was noch schwerer wiegt ist die Verschlechterung des maximalen Positionsfehlers gegenüber den Testdaten auf fast 15 mm.

8.5 370 Stellungen aus Vorgaben

In Kapitel 7.5 werden die Resultate nochmals deutlich schlechter. Die Kalibrierung mit dem unverfälschten Modell als a priori Modell in Kapitel 7.5.1 führt zu doppelt so großen Fehlern in den Testposen und den kinematischen Parametern wie die Kalibrierung mit dem manipulierten a priori Modell in Kapitel 7.4.3. Die Ausnahme stellen hier die Winkelparameter θ und α dar. Beide Kalibrierungen sind mit den gleichen Einstellungen vorgenommen worden. Das aus dem verfälschten a priori Modell erzeugte a posteriori Modell, welches mit diesen Einstellungen erzeugt wird, siehe Kapitel 7.5.3, zeigt noch deutlich größere Fehler.

Der Ausweg ist hier die Annahme einer größeren Varianz der kinematischen Parameter. Es zeigt sich in den Kapiteln 7.5.1, 7.5.2, 7.5.4 und 7.5.5, dass auch hier die a posteriori Modelle gegen das gleiche, durch die Messfehler bestimmte, Modell konvergieren.

Es zeigt sich hier, dass bei ungünstig gewählten Messstellungen Kalibrierungsparameter die im Allgemeinen zu starkem Overfitting führen, vergleiche hierzu Kapitel 8.4, zu besseren Ergebnissen führen. Diese liegen dann bei maximalen Posenfehlern gegenüber den Testdaten von fast 36 mm und 5°.

8.6 Vergleich der Trainingsreihen

In den Kapiteln 7.2, 7.3, 7.4 und 7.5 zeigt sich der starke Einfluss der Messstellungen. Bei Daten gleicher Qualität, alle Trainingsdatensätze haben die gleichen Streuung, steigert die Verringerung der Datenmenge auf 3.7 % die Fehler auf das bis zu Siebenfache. Die daraus resultierenden Fehler sind aber akzeptabel im Verhältnis zu der Messgenauigkeit der Posenbestimmung.

Die Auswahl der Messstellungen stellt einen wichtigen Einflussfaktor dar, bei gleicher Anzahl differiert die resultierende Genauigkeit stark. Aus den Experimenten lässt sich schliessen, dass eine möglichst gleichmässige Verteilung über die möglichen Gelenkwinkel die beste Strategie darstellt.

8.7 Simulation von Laser und Schachbrett

In Kapitel 7.6 zeigt sich, dass die Kalibrierung zu anwendbaren Verbesserungen in der Positionssteuerung des Greifers führen. Die aus den Simulationenen gewonnenen Eindrücke decken sich mit den Erwartungen die an die mittlere Genauigkeit der verwendeten Modelle gestellt werden können.

8.8 Datenaufnahme Dynamaid

In Kapitel 7.7.1 kommt es zu mittleren Posenfehlern von fast fünf Zentimetern und neun Grad. Die maximalen Fehler betragen fast neun Zentimeter und über 13°. Es ist durchaus möglich, dass die manuelle Unterstützung des Roboters hier zu größeren Messfehlern führt.

Dies ist erst recht in Kapitel 7.7.2 zu befürchten. Ein Fehler, welcher zu unzuverlässigen Messungen führen kann ist, dass bei manuellem Bewegen des Roboters Gelenklimits überschritten werden. Dies führt nicht zu einer Beschädigung des Roboters, aber zu falschen Werten, da Messungen der Gelenkstellungen ausserhalb der Grenzen vom Robotcontrol auf diese Grenzen gekappt werden. Diese Datenaufnahme dient somit der Überprüfung der Praktikabilität der passiven Datenaufnahme im Umfeld der gegebenen Softwareinfrastruktur. Es zeigen sich auch hier die größten Abweichungen zwischen a priori Modell und den Daten. Die Winkelfehler entsprechen zwar den aktiv aufgenommenen Läufen, aber die Positionsfehler der Endeffektorpose sind bis über einen Zentimeter schlechter.

Die beste Passung zwischen a priori Modell und Daten findet sich in den völlig eigenständig aufgenommenen Daten aus Kapitel 7.7.3. Die kinematischen Parameter des a posteriori Modells dieses Kapitels zeigen eine sehr ähnliche Entwicklung wie die aus Kapitel 7.7.1. Diese zeigt sich auch in der Kombination der beiden Läufe in Kapitel 7.7.4. Eine Übereinstimmung zeigt sich zudem in Kapitel 7.7.5, das a posteriori Modell aus Kapitel 7.7.3 hat einen deutlich geringeren Fehler gegenüber den Daten aus Kapitel 7.7.1 als das a priori Modell. Dies wiederspricht der Vermutung, dass die manuelle Unterstützung des Roboters zu starken Fehlern führt.

Für die Daten aus Kapitel 7.7.2 ist das Gegenteil der Fall. Bis auf die maximalen Positionsfehler steigen in den Kapiteln 7.7.6 und 7.7.7 sogar alle Abweichungen. Dies ist bemerkenswert, da die Messstellungen aus 7.7.2 die Grundlage für die Messungen aus Kapitel 7.7.3 darstellen. Trotz der Möglichkeit in Kapitel 7.7.2 ein a posteriori Modell zu finden, welches zu recht geringen Abweichungen von den Daten führt, sind diese Daten wohl als unbrauchbar anzusehen.

Für die Daten aus Kapitel 7.7.1 und 7.7.3 zeigen sich ähnliche Fehlergrößen, wie sie in Kapitel 7.5 zu sehen sind.

8.9 Anpassung der Lernparameter

Kapitel 7.8 zeigt, dass die Kalibrierungsparameter so gewählt werden können, dass die Fehler gegenüber den Testdaten den Fehlern gegenüber den Trainingsdaten nahe kommen und dabei auch im Bereich der Werte der Kalibrierungen aus Kapitel 7.7 bleiben. Hierbei ist abzuwägen wie sehr die Maximalwerte berücksichtigt werden. In Kapitel 7.8.3 sind zwar die meisten Fehler leicht größer als in Kapitel 7.8.4, aber die Maximalwerte bei den Testdaten sind geringer. Dies gilt sogar für den maximalen Orientierungsfehler bei den Trainingsdaten. Wobei die Unterschiede bei den Orientierungen aber sehr gering sind. Somit werden im Weiteren die Kalibrierparameter aus Kapitel 7.8.3 verwendet.

In den Kapiteln 7.8.6 und 7.8.8 zeigt sich die gleiche Entwicklung der Parameter wie in Kapitel 7.7 für diese Daten. Auch hier sind die manuell erzeugten Daten aus Kapitel 7.7.2 mit den Restlichen nicht in Einklang zu bringen.

Insgesamt reduziert die Kalibrierung die Fehler auf ungefähr ein Drittel des bisher verwendeten Modells. Vergleiche hierzu die Fehler des a priori Modells gegenüber den Daten aus Kapitel 7.7.1 und die des a posteriori Modells aus Kapitel 7.8.3 gegenüber

den gleichen Daten.

8.10 Messung der Wiederholgenauigkeit von Dynamaid

In den Kapiteln 7.9 und 7.10 werden Messungen zur Wiederholgenauigkeit vorgenommen. Es zeigen sich dabei für einen Roboter in Leichtbauweise gute bis sehr gute Werte. Da es zu Sprüngen beim Halten von Armstellungen kommt ist anzunehmen, dass durch Anpassungen in der Steuersoftware weitere Verbesserungen möglich sind. Wie in Kapitel 7.9 beschrieben, gibt es einzelne Punkte, die mit einem Fehler von circa zwei Millimetern wieder getroffen werden. Dies stellt die Hälfte der in Kapitel 7.10 gefundenen Effektivwerte dar.

Durch die Neuerfassung der Schachbrettpose verschlechtert sich die durchschnittliche Wiederholgenauigkeit um circa zwei Millimeter. Dies entspricht in etwa der in Kapitel 7.1 gefundenen Genauigkeit der monoskopischen Posenbestimmung.

8.11 Dynamaid richtet Laser auf Schachbrett

In Kapitel 7.10 wird die Kalibrierung in einer realen Anwendung getestet. Im direkten Vergleich des a priori Modells mit dem a posteriori Modells zeigt sich eine Verbesserung um ein Drittel. Dies ist eine deutliche Verbesserung gegenüber dem bisherigen, für den Einsatz des Roboters verwendeten a priori Modell.

Der Vergleich der Genauigkeit der Ansteuerung von Endeffektorposen und der Ausrichtung eines Lasers auf ein Schachbrett unterliegt einigen Einschränkungen. Der Laserstrahl verlängert die kinematische Kette, dadurch führen Winkelfehler zu größeren Positionsfehlern. Dies sollte sich hier aber nicht zu stark auswirken, da der Strahl schon nach circa 20 cm Weg auf das Schachbrett trifft. Zudem ist das Experiment so ausgelegt, dass der Strahl senkrecht auf das Brett fallen soll. Dies minimiert den Einfluss von Winkelfehlern auf die Positionsgenauigkeit. Nimmt man an, dass der Positionierungsfehler in allen Raumrichtungen gleich groß ist, führt dieses Vorgehen sogar dazu, dass der gemessene, zweidimensionale Fehler kleiner ist als der wahre dreidimensionale Fehler. Zudem können Ungenauigkeiten beim Ausmessen des Laserstrahlengangs zur Verfälschung der Ergebnisse führen.

Trotz der genannten Unwägbarkeiten liegen die Ergebnisse in den Bereichen von Kapitel 7.7 und 7.8. Die Fehler des a priori Modells im Laser-Experiment entsprechen denen aus Kapitel 7.7.3. Er ist sogar deutlich besser als die Positionsfehler des Modells in Kapitel 7.7.1. Das a posteriori Modell führt im Laser-Experiment zu größeren Mittel- und Effektivwerten der Positionsfehler als in den Kapiteln 7.8.3 und 7.8.8.

Hierfür gibt es mehrere Erklärungsansätze. Wenn die beiden Modelle den Laser in entgegengesetzte Richtungen fehlplatzieren führt eine Ungenauigkeit beim Einmessen des Laserstrahlengangs zu der fälschlichen Verbesserung des einen Modells und der Verschlechterung des anderen. Die höhere Standardabweichung des Fehlers spricht hier gegen das a priori Modell und für das a posteriori Modell.

Eine weitere Erklärung bietet die Auswahl der Trainingsstellungen. Bei der Erfassung der Trainingsdaten wird versucht den Konfigurationsraum der Gelenke möglichst gleichmäßig abzudecken, vergleiche hierzu Kapitel 8.6. Es besteht die Möglichkeit, dass eine zu kleine Menge an Trainingsdaten im Bereich des Konfigurationsraums liegen, der für die Laser-Experimente verwendet wird. So wäre in diesem Bereich der durchschnittliche Fehler größer als im gesamten Raum, bei Beibehaltung des maximalen Fehlers.

8.12 Lokal gewichtetes Lernen

Es zeigt sich in Kapitel 7.11, dass es lokalisierte a posteriori Modelle gibt, welche massiv vom Durchschnitt der lokalisierten Modelle abweichen. Dies führt auch zu sehr großen maximalen Posenfehlern gegenüber dem Trainingsdatensatz. Dabei bietet die aktuelle Umsetzung kaum Vorteile gegenüber einem generellen, nicht lokalisierten, a posteriori Modell. Selbst in direkter Nähe zum Zentrum der Lokalisierung entsprechen die mittleren Fehler den Mittelwerten der Fehler der generellen Modelle aus den Kapiteln 7.8.8 und 7.8.7.

Es zeigt sich in den Abbildungen 7.57 und 7.61, dass nur sehr wenige Modelle für die sehr großen maximalen Fehler verantwortlich sind. Diese großen Abweichungen können aus einer ungleichmässigen Verteilung der Stellungen im Konfigurationsraum resultieren. Wenn es in der Nähe eines Zentrums nur wenige weitere Stellungen gibt, so gewinnen diese einen zu großen Einfluss auf das erzeugte Modell.

Die Ergebnisse zeigen dennoch einen Ansatzpunkt für weitere Entwicklungen. Sowohl die mittleren Fehler der Position als auch die der Orientierung steigen mit größerer Entfernung zum Ursprung. Dieser Effekt steigert sich bei einer kleiner gewählten Varianz der Abstandsfunktion, vergleiche hierzu Abblildungen 7.59 und 7.62. Es ist aber eventuell auf die Vewendbarkeit eines Modells für die Planung der Trajektorie zum Arbeitspunkt zu achten. Zudem ist die Verwendung mehrerer Modelle mit einem erheblichen Mehraufwand in der Anwendung verbunden, da in ROS bisher nur die Verwendung eines Modells vorgesehen ist.

9 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde die **Entwicklung eines effizienten Verfahrens zur Kalibrierung der Kinematik mit Hilfe visueller Sensordaten basierend auf der MAP** beschrieben. Dabei wurde gezeigt, wie ein DH-Modell mit MAP und Paaren aus Gelenkstellungen und Endeffektorposen optimiert wird. Hierbei wird ein Gradientenabstieg mit Rprop durchgeführt.

Später wurde in Experimenten gezeigt, dass die Kalibrierung mit nur 370 zufällig gewählten Trainingsstellungen zu Modellen führt, welche mittlere Endeffektorgenauigkeiten von unter drei Millimetern und einem drittel Grad erreichen. Dabei wichen die a priori Modelle zum Teil stark vom realen Modell ab. Es zeigt sich, dass sich das Verfahren gut eignet, um aus sehr ungenauen a priori Modellen mit wenigen Trainingsdaten schnell a posteriori Modelle zu erzeugen, die eine Positioniergenauigkeit von wenigen Zentimetern erreichen.

Im Weiteren wurde die **Implementierung des Verfahrens in der gegebenen Systemumgebung** beschrieben. Es wurde eine Exploration implementiert, bei der der Roboter vorgegebene Konfigurationen der Arme und des Kopfes anfährt. Dabei wird eine Kamera im Kopf auf die jeweilige Endeffektorposition gerichtet und die Endeffektorpose anhand eines Markers monoskopisch bestimmt. Im Anschluss wurde die Umsetzung der Kalibrierung beschrieben. Hierbei wird das URDF-Modell des Roboters vor der Kalibrierung in das DH-Modell für die Kalibrierung gewandelt. Nach der Kalibrierung wird das Gelernte dann in das ursprüngliche Modell zurück übertragen. So wurde die Verwendung des optimierten Modells durch vorhandene Komponenten ermöglicht.

Die **Evaluation des Verfahrens** fand in mehreren Schritten statt. Zuerst wurde die Präzision der verwendeten Sensorik evaluiert. Mit diesen Daten wurden künstliche Test- und Trainingsdaten erzeugt, mit denen Kalibrierungen durchgeführt und evaluiert wurden. Hierbei zeigte sich auch die Bedeutung der Wahl der Trainingsstellungen. Die Verwendbarkeit eines a posteriori Modells im bestehenden Umfeld wurde dann in Simulationen bestätigt. Danach wurden mit Dynamaid reale Test- und Trainingsdaten erfasst und mit diesen das Verfahren evaluiert und angepasst. Die Ergebnisse dieser Untersuchung bestätigten die Resultate der Experimente mit den künstlichen Daten. Hierbei reduziert die Kalibrierung die Fehler auf ein Drittel. Im Anschluss wurden mit Dynamaid Experimente zur Wiederhol- und Positioniergenauigkeit durchgeführt. Es zeigte sich dabei eine gute Wiederholgenauigkeit der verwendeten Hardware. Die gemessenen Positioniergenauigkeiten bestätigen die vorhergehenden Ergebnisse. Dabei wurde allerdings nicht die gleiche Genauigkeit wie in den vorhergehenden Experimenten erreicht. Dennoch wurde die Genauigkeit gegenüber dem bisher für den Betrieb verwendeten Modell um ein Drittel verbessert.

Im Anschluss wurde lokalisiertes Lernen evaluiert. Dabei wird der Konfigurationsraum

in Teilräume aufgeteilt, für die dann jeweils ein Modell gelernt wird. Dies führte nicht zu verwendbaren Kalibrierungen, zeigte aber Perspektiven für weitere Untersuchungen auf. Die Ergebnisse der lokalisierten Kalibrierung zeigen, dass lokale Abhängigkeiten existieren, die Resultate sprechen aber gegen eine Anwendung lokalisierter Kalibrierungen in der aktuellen Form. Zudem ist es weder in der KDL noch in ROS möglich verschiedene Modelle für unterschiedliche Teilräume zu verwenden. Dieser Teil von ROS müsste in Zukunft noch erweitert werden.

Das Verfahren ist momentan auf die Verwendung einer Kamera ausgelegt. Wenn mehrere Kameras benutzt werden, so wird die Annahme gemacht, dass ein gemeinsamer Bezugsrahmen bekannt ist. Um dies zu gewährleisten, sind vorher Schritte zur Bestimmung der entsprechenden Transformationen durchzuführen, siehe [Zha11; PM94; cam12b; PKB10]. Für den Mehrkamerabetrieb wäre eine Automatisierung dieser Schritte eine sinnvolle Erweiterung.

In der vorliegenden Arbeit wurden nur Drehgelenke betrachtet. Die Erweiterung auf prismatische Gelenke sollte aber ohne Schwierigkeiten umsetzbar sein.

In der Exploration liegt das größte Entwicklungspotential für die Genauigkeit, da die Experimente zeigen, dass die Wahl der Trainingsstellungen die Ergebnisse deutlich beeinflusst. Das Explorationsverfahren könnte so gestaltet werden, dass der Roboter aktiv in Bereichen Daten aufnimmt, in denen die größten Unsicherheiten liegen. Stichworte hierzu sind *Information Gain* [TBF06] und *Gausssche Prozesse*. Ergänzend hierzu könnten Daten während des Betriebs der Roboter aufgenommen werden und die Modelle online angepasst werden.

A Messungen

In diesem Kapitel befinden sich die Tabellen mit den Daten der manuellen Messungen.

A.1 Monoskopische Posenbestimmung

Für eine Beschreibung siehe Kapitel 7.1.1.

Tabelle A.1: Manuelle Messung des Abstands

manuell	**monoskopisch**	Δ
95	100	5
100	100	0
147	150	3
148	150	2
202	201	1
336	340	4
366	383	17
404	401	3
495	510	15
502	540	38
590	587	3
595	576	19
667	665	2
817	803	14
877	872	5
1024	1000	24
1180	1171	9

Alle Werte in Millimetern.

A.2 Laser auf Schachbrett

Für eine Beschreibung siehe Kapitel 7.10. Der Eintrag ¬IK bedeutet, dass es für die Position keine Lösung für die IK gibt.

Tabelle A.2: Abstand zum Ziel

Lauf	**Ecke**	**A priori Modell**		**A posteriori Modell**	
1	OL	36	40	10	13
	OR	50	52	5	5
	UL	31	36	16	18
	UR	45	48	8	10
2	OL	36	41	30	30
	OR	43	55	22	27
	UL	39	47	37	40
	UR	38	42	36	36
3	OL	27	29	34	34
	OR	39	39	16	21
	UL	17	20	32	37
	UR	25	28	19	26
4	OL	17	17	15	17
	OR	48	48	19	20
	UL	7	9	20	21
	UR	38	39	15	15
5	OL	24	25	12	17
	OR	60	60	15	17
	UL	20	20	31	35
	UR	55	57	13	17
6	OL	53	57	36	38
	OR	47	49	36	36
	UL	27	31	¬IK	¬IK
	UR	38	41	¬IK	¬IK
7	OL	17	19	19	19
	OR	30	30	15	15
	UL	23	23	20	25
	UR	34	34	13	15
8	OL	48	48	33	36
	OR	41	41	32	34
	UL	26	30	37	40
	UR	34	38	29	33
9	OL	21	22	9	9
	OR	31	33	5	8
	UL	20	26	12	12
	UR	27	27	7	7
10	OL	23	26	16	18
	OR	32	38	22	23
...					

Tabelle A.2: Abstand zum Ziel (Fortsetzung)

Lauf	**Ecke**	**A priori Modell**		**A posteriori Modell**	
	UL	26	29	31	34
	UR	32	34	26	29

Alle Werte in Millimetern.

Tabelle A.3: Wiederholungen ohne neue Posenbestimmung

Lauf	**Ecke**	**A priori Modell**	**A posteriori Modell**
1	OL	4	4
	OR	4	2
	UL	4	7
	UR	5	6
2	OL		0
	OR		5
	UL		3
	UR		2
3	OL	2	
	OR	2	
	UL	3	
	UR	3	
4	OL	3	2
	OR	3	2
	UL	5	4
	UR	3	0
5	OL	3	6
	OR	5	2
	UL	0	5
	UR	5	4
6	OL	5	4
	OR	8	0
	UL	4	¬IK
	UR	5	¬IK
7	OL	6	1
	OR	5	0
	UL	0	5
	UR	1	2
9	OL	1	0
	OR	6	7
...			

Tabelle A.3: Wiederholungen ohne neue Posenbestimmung (Fortsetzung)

Lauf	**Ecke**	**A priori Modell**	**A posteriori Modell**
	UL	7	5
	UR	0	0
10	OL	3	6
	OR	6	3
	UL	5	3
	UR	5	5

Alle Werte in Millimetern.

Tabelle A.4: Wiederholungen mit neuer Posenbestimmung

Lauf	**Ecke**	**A priori Modell**	**A posteriori Modell**
2	OL	5	
	OR	12	
	UL	10	
	UR	4	
3	OL		0
	OR		7
	UL		5
	UR		7
8	OL	12	4
	OR	0	3
	UL	4	3
	UR	6	7

Alle Werte in Millimetern.

Glossar

A^i Die Achs-Transformationsfunktion vom Kindsegment des i. Gelenks einer URFF-Kette in das Gelenkkordinatensystem des Gelenks. Bestimmt durch a^i und den Typ des Gelenks und abhängig von θ_i. 11

a^i Der Achsvektor des i. Gelenks in Bezug auf dessen Koordinatensystem. 11

B Transformation vom Koordinatensystem des ersten Gelenks zum Ursprung der DH-Kette. 14

D^i Die Transformationsmatrix des i. Gelenks einer DH-Kette; ergibt sich direkt aus dessen DH-Parametern. 13

d^i Die Parameter des i. Gelenks einer DH-Kette; $d^i = (d^i_\theta\ d^i_d\ d^i_a\ d^i_\alpha)$. 12

E Die 4×4 Einheitsmatrix. 9

Effektivwert Quadratisches Mittel, $\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}$. 53, 71, 81, 93, 126

f Anzahl der bewegten Gelenke einer kinematischen Kette. 5, 9

$F^i(v)$ $:= F^i(v_i)$. 8

K^i Die Transformation vom Koordinatensystem des i. DH-Segment zum i. URDF-Koordinatensystem. 14

$M_{i,j}$ Das Element in der Reihe i und Spalte j der Matrix M. 8

n Anzahl der Gelenke einer kinematischen Kette. 9

O^i Die Transformationsmatrix vom Gelenkkordinatensystem des i. Gelenks einer URFF-Kette in sein Eltersement $(i-1)$, bestimmt durch u^i und r^i. 11

r^i Die Roll-Nick-Gier-Winkel in rad des Ursprungs des i. Gelenks einer URFF-Kette in bezug auf das $i-1$. Segment. 11, 12

θ Der Zustandsvektor der Gelenke einer kinematischen Kette. 9

θ^0 $\theta^0 := (0, \ldots, 0) \in \mathbb{R}^n$. 9

T_j^i Die Koordinatentransformation von Segment j nach Segment i. 10

T_{Kette} Die Transformation vom Endeffektor- ins Weltkoordinatensystem einer kinematischen Kette. 10

u^i Die Koordinaten des Ursprungs des i. Gelenk einer URFF-Kette in Bezug auf das $i-1$. Segment. 11

v_i Die i. Komponente eines Vektors v. 8

Weltkoordinatensystem Fester Bezugsrahmen für die Betrachtung beweglicher Komponenten; meist der Beginn einer kinematischen Kette. 10

Z^i Die Z-Achsen-Transformationsfunktion des i. Gelenks einer DH-Kette, bestimmt durch den Typ des Gelenks und abhängig von θ_i. 14

Tabellenverzeichnis

6.1 Allgemeine Kommandozeilenparameter 46
6.2 Kommandozeilenparameter der Gelenkstellungsaufzeichnung 47
6.3 Kommandozeilenparameter der Exploration 48
6.4 Kommandozeilenparameter der Synthese 48
6.5 Konfiguration der Exploration, Gelenkstellungsaufzeichnung und Synthese . . 49
6.6 Konfiguration einer Datenaufzeichnungstellung 52
6.7 Tastatureingaben in der Kalibrierung 55
6.8 Kommandozeilenparameter der Kalibrierung und Analyse 57
6.9 Konfiguration der Kalibrierung und Analyse 57
6.10 Konfiguration der Evaluation 64

A.1 Manuelle Messung des Abstands 143
A.2 Abstand zum Ziel 144
A.3 Wiederholungen ohne neue Posenbestimmung 145
A.4 Wiederholungen mit neuer Posenbestimmung 146

Abbildungsverzeichnis

1.1 Cosero greift einen Löffel, Foto: AG Behnke/Uni Bonn 1
1.2 Endeffektor im Videobild, erstellt mit Rviz 3

3.1 Kinematische Kette, erstellt mit LibreOffice 10
3.2 URDF Gelenk, erstellt mit FreeCAD und GIMP 11
3.3 DH Gelenk, erstellt mit FreeCAD und GIMP 12
3.4 KDL Segment, erstellt mit FreeCAD und GIMP 15
3.5 KDL Baum, erstellt mit LibreOffice 16

4.1 Dynamaid, Foto: AG Behnke/Uni Bonn 20
4.2 Cosero, Foto: AG Behnke/Uni Bonn 21
4.3 ROS-Topics, erstsellt mit rxgraph 23
4.4 TFs von Cosero, erstellt mit Rviz 24
4.5 Rviz, erstellt mit Rviz 25
4.6 Gazebo, erstellt mit Gazebo 26
4.7 URDF-Kette erstellt mit LibreOffice 28
4.8 URDF-Baum von Cosero, erstellt mit urdf_parser-urdf_to_graphiz 29
4.9 (b) Kreuzungspunkte, erstellt mit Pr2_calib, Rviz und Gazebo, (c) Pose, erstellt mit Rviz und Gazebo 30
4.10 DH Konverter, erstellt mit FreeCAD und Gimp 31

5.1 URDF-Baum, erstellt mit LibreOffice 40

6.1 Simulierte Exploration, erstellt mit Gazebo 46
6.2 Fehlerentwicklung beim Lernen, erstellt mit Matplotlib 53
6.3 3D Ansicht der Ketten beim größten Fehler der Orientierung, erstellt mit Rviz 54
6.4 2D Aufsicht der Ketten beim größten Fehler der Orientierung 55
6.5 analzse.py 56
6.6 Fehler in 3D, erstellt mit Matplotlib 60
6.7 Endeffektorpositionen in 3D, erstellt mit Matplotlib 61
6.8 Vergleich zweier Ketten, erstellt mit Matplotlib 61
6.9 Parameterverteilung bei lokalisiertem Lernen, erstellt mit Matplotlib 62
6.10 Fehler bei lokalisiertem Lernen, erstellt mit Matplotlib 62
6.11 Fehler bei lokalisiertem Lernen, erstellt mit Matplotlib 63
6.12 eval.py 64
6.13 Robotermodell Evaluation, erstellt mit Rviz 67
6.14 Halterung, Bild: Michael Schreiber/Uni Bonn 69

7.1 Fehler in der Simulation bei Cosero, Position, erstellt mit Matplotlib 72
7.2 Fehler in der Simulation bei Cosero, Orientierung, erstellt mit Matplotlib . . 73
7.3 Fehler in der Simulation bei PR2, Position, erstellt mit Matplotlib 74
7.4 Fehler in der Simulation bei PR2, Orientierung, erstellt mit Matplotlib 75
7.5 10000 Stellungen, erstellt mit Matplotlib 76
7.6 Keine Abweichungen, 100 Iterationen, Varianz 0,01, erstellt mit Matplotlib . 77
7.7 Keine Abweichungen - real vs. a posteriori, 100 Iterationen, Varianz 1,0, erstellt mit Matplotlib 78
7.8 Keine Abweichungen - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1, erstellt mit Matplotlib 79
7.9 Abweichender Ellenbogen - real vs. a priori, erstellt mit Matplotlib 80
7.10 Fehlerentwicklung Ellenbogen, erstellt mit Matplotlib 81
7.11 Abweichender Ellenbogen - real vs. a posteriori, 100 Iterationen, Varianz 0,01, erstellt mit Matplotlib 82
7.12 Abweichender Ellenbogen - real vs. a posteriori, 100 Iterationen, Varianz 0,1, erstellt mit Matplotlib 83
7.13 Abweichender Ellenbogen - real vs. a posteriori, 200 Iterationen, Varianz 0,1, erstellt mit Matplotlib 84
7.14 Abweichender Ellenbogen - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,01, erstellt mit Matplotlib 85
7.15 Abweichender Ellenbogen - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1, erstellt mit Matplotlib 86
7.16 Alle Gelenke abweichend - real vs. a priori, erstellt mit Matplotlib 87
7.17 Alle Gelenke abweichend - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1, erstellt mit Matplotlib 88
7.18 Alle Gelenke abweichend - real vs. a posteriori, 100 Iterationen, Varianz 1,0, erstellt mit Matplotlib 89
7.19 Trainingspositionen, erstellt mit Matplotlib 90
7.20 Keine Abweichungen - real vs. a posteriori, 200 Iterationen, Varianz 0,01, erstellt mit Matplotlib 91
7.21 Keine Abweichungen - real vs. a posteriori, 200 Iterationen, Varianz 1,0, erstellt mit Matplotlib 92
7.22 Fehlerentwicklung - alle Gelenke, erstellt mit Matplotlib 93
7.23 Alle Gelenke abweichend - real vs. a posteriori, 200 Iterationen, Varianz 1,0, erstellt mit Matplotlib 93
7.24 Alle Gelenke abweichend - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1, erstellt mit Matplotlib 94
7.25 Alle Gelenke abweichend, Modell 2 - real vs. a priori, erstellt mit Matplotlib . 95
7.26 Alle Gelenke abweichend, Modell 2 - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1, erstellt mit Matplotlib 96
7.27 Alle Gelenke abweichend, Modell 3 - real vs. a priori, erstellt mit Matplotlib . 97
7.28 Alle Gelenke abweichend, Modell 3 - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1, erstellt mit Matplotlib 98
7.29 Trainingspositionen, erstellt mit Matplotlib 99

7.30 Keine Abweichungen - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1, erstellt mit Matplotlib . . . 100
7.31 Keine Abweichungen - real vs. a posteriori, 340 Iterationen, Austausch alle 70, Varianz 0,5, erstellt mit Matplotlib . . . 101
7.32 Alle Gelenke abweichend - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1, erstellt mit Matplotlib . . . 102
7.33 Alle Gelenke abweichend - real vs. a posteriori, 340 Iterationen, Austausch alle 70, Varianz 0,5, erstellt mit Matplotlib . . . 103
7.34 Trainingspositionen, erstellt mit Matplotlib . . . 104
7.35 Keine Abweichungen - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1, erstellt mit Matplotlib . . . 105
7.36 Keine Abweichungen - real vs. a posteriori, 340 Iterationen, Austausch alle 70, Varianz 0,5, erstellt mit Matplotlib . . . 106
7.37 Alle Gelenke abweichend - real vs. a posteriori, 200 Iterationen, Austausch alle 70, Varianz 0,1, erstellt mit Matplotlib . . . 107
7.38 Alle Gelenke abweichend - real vs. a posteriori, 340 Iterationen, Austausch alle 70, Varianz 0,5, erstellt mit Matplotlib . . . 108
7.39 Alle Gelenke abweichend - real vs. a posteriori, 480 Iterationen, Austausch alle 70, Varianz 1,5, erstellt mit Matplotlib . . . 109
7.40 Simulation von Laser und Schachbrett, erstellt mit Gazebo . . . 110
7.41 Reales Modell, erstellt mit Gazebo . . . 111
7.42 A priori Modell, erstellt mit Gazebo . . . 112
7.43 A posteriori Modell, erstellt mit Gazebo . . . 113
7.44 A posteriori Modell mit Tiefe, erstellt mit Gazebo . . . 114
7.45 Selbstexploration Dynamaid, 327 Stellungen, erstellt mit Matplotlib . . . 115
7.46 A posteriori Modell, Dynamaid, 327 Stellungen, erstellt mit Matplotlib . . . 116
7.47 Einlernen von 128 Stellungen bei Dynamaid, erstellt mit Matplotlib . . . 117
7.48 A posteriori Modell, Dynamaid, 128 Stellungen, erstellt mit Matplotlib . . . 117
7.49 Selbstexploration Dynamaid, 597 Stellungen, erstellt mit Matplotlib . . . 118
7.50 A posteriori Modell, Dynamaid, 597 Stellungen, erstellt mit Matplotlib . . . 119
7.51 A posteriori Modell, Dynamaid, 924 Stellungen, erstellt mit Matplotlib . . . 120
7.52 Kalibrierung, angepasst, 597 Stellungen, erstellt mit Matplotlib . . . 123
7.53 Kalibrierung, angepasst, 924 Stellungen, erstellt mit Matplotlib . . . 124
7.54 Fehler beim lokalisierten Lernen, Dynamaid, 924 Stellungen, Varianz 0,3, erstellt mit Matplotlib . . . 127
7.55 Mittelwerte der Fehler beim lokalisierten Lernen, Dynamaid, 924 Stellungen, Varianz 0,3, erstellt mit Matplotlib . . . 128
7.56 Parameter beim lokalisierten Lernen, Dynamaid, 924 Stellungen, Varianz 0,3, erstellt mit Matplotlib . . . 128
7.57 Einzelne Fehler bei lokalisiertem Lernen, 924 Stellungen, Varianz 0,3, erstellt mit Matplotlib . . . 129
7.58 Fehler beim lokalisierten Lernen, Dynamaid, 327 Stellungen, Varianz 0,3, erstellt mit Matplotlib . . . 130

7.59 Mittelwerte der Fehler beim lokalisierten Lernen, Dynamaid, 327 Stellungen, Varianz 0,3, erstellt mit Matplotlib 131
7.60 Parameter beim lokalisierten Lernen, Dynamaid, 327 Stellungen, Varianz 0,3, erstellt mit Matplotlib 131
7.61 Einzelne Fehler bei lokalisiertem Lernen, 327 Stellungen, Varianz 0,3, erstellt mit Matplotlib 132
7.62 Mittelwerte der Fehler beim lokalisierten Lernen, Dynamaid, 327 Stellungen, Varianz 0,1, erstellt mit Matplotlib 133

Literaturverzeichnis

[AT97] Angulo, V.R. de ; Torras, C.: Self-calibration of a space robot. In: *Neural Networks, IEEE Transactions on* 8 (1997), Juli, Nr. 4, S. 951–963. http://dx.doi.org/10.1109/72.595895. – DOI 10.1109/72.595895. – ISSN 1045–9227

[AT05] Angulo, V.R. de ; Torras, C.: Speeding up the learning of robot kinematics through function decomposition. In: *Neural Networks, IEEE Transactions on* 16 (2005), nov., Nr. 6, S. 1504–1512. http://dx.doi.org/10.1109/TNN.2005.852970. – DOI 10.1109/TNN.2005.852970. – ISSN 1045–9227

[AT08] Angulo, V.R. de ; Torras, C.: Learning Inverse Kinematics: Reduced Sampling Through Decomposition Into Virtual Robots. In: *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on* 38 (2008), dec., Nr. 6, S. 1571–1577. http://dx.doi.org/10.1109/TSMCB.2008.928232. – DOI 10.1109/TSMCB.2008.928232. – ISSN 1083–4419

[Bis06] Bishop, Christopher M.: *Pattern Recognition And Machine Learning*. Springer Science+Buisiness Media, LLC, 2006. – ISBN 978–0–387–31073–2

[cam12a] *camera_calibration - ROS Wiki*. http://www.ros.org/wiki/camera_calibration. Version: Feb. 2012

[cam12b] *camera_pose_toolkits - ROS Wiki*. http://www.ros.org/wiki/camera_pose_toolkits. Version: Feb. 2012

[Cor94] Corke, Peter I.: Visual Control Of Robot Manipulators – A Review. In: *Visual Servoing*, World Scientific, 1994, S. 1–31

[DVS01] D'Souza, A. ; Vijayakumar, S. ; Schaal, S.: Learning inverse kinematics. In: *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on* Bd. 1, 2001, S. 298–303 vol.1

[Dyn12] *Dynamixel - Homepage*. http://www.robotis.com/xe/dynamixel_en. Version: Feb. 2012

[Eig12] *Eigen Homepage*. http://eigen.tuxfamily.org/index.php?title=Main_Page. Version: Feb. 2012

[Fes12] *Festo Homepage*. http://www.festo.de. Version: Feb. 2012

[Fis95] Fischer, Gerd: *Lineare Algebra*. 10. Vieweg, 1995. – ISBN 3–528–67217–X

[gaz12] *gazebo - ROS Wiki*. http://www.ros.org/wiki/gazebo. Version: Feb. 2012

[Gnu12] *GSL - GNU Scientific Library*. http://www.gnu.org/software/gsl/. Version: Feb. 2012

[GTe12] *Google C++ Testing Framework*. http://code.google.com/p/googletest/. Version: Feb. 2012

[HHC96] Hutchinson, Seth ; Hager, Greg ; Corke, Peter: A Tutorial on Visual Servo Control. In: *IEEE Transactions on Robotics and Automation* 12 (1996), S. 651–670

[HS11] Hart, J.W. ; Scassellati, B.: A robotic model of the Ecological Self. In: *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, 2011. – ISSN 2164–0572, S. 682 –688

[HSB08] Hersch, M. ; Sauser, E. ; Billard, A.: Online learning of the body schema. In: *International Journal of Humanoid Robotics* 5 (2008), Nr. 2, S. 161–181. http://dx.doi.org/10.1142/S0219843608001376. – DOI 10.1142/S0219843608001376

[KDL12a] *KDL Homepage*. http://www.orocos.org/kdl/. Version: Feb. 2012

[KDL12b] *KDL Subversion Repository*. http://svn.mech.kuleuven.be/repos/orocos/trunk/kdl. Version: Feb. 2012

[kdl12c] *kdl_parser - ROS Wiki*. http://www.ros.org/wiki/kdl_parser. Version: Feb. 2012

[KGD+03] Krupa, A. ; Gangloff, J. ; Doignon, C. ; Mathelin, M.F. de ; Morel, G. ; Leroy, J. ; Soler, L. ; Marescaux, J.: Autonomous 3-D positioning of surgical instruments in robotized laparoscopic surgery using visual servoing. In: *Robotics and Automation, IEEE Transactions on* 19 (2003), Nr. 5, S. 842–853. – ISSN 1042–296X

[Kin12] *Kinect for Windows - Homepage*. http://www.microsoft.com/en-us/kinectforwindows/. Version: Feb. 2012

[Mat12] *Matplotlib - Homepage*. http://matplotlib.sourceforge.net/. Version: Feb. 2012

[MN02] Magnus, Jan R. ; Neudecker, Heinz: *Matrix Differential Calculus with Applications in Statistics and Econometrics*. 2. John Wiley and Sons, 2002. – ISBN 0–471–98633–X

[Moh12] *handEyeCalibration Git Repository.* https://github.com/IDSCETHZurich/IDSC-tools/tree/master/handEyeCalibration. Version: Feb. 2012

[Nim12] *Nimbro@Home Homepage.* http://www.nimbro.net/@Home/. Version: Feb. 2012

[Num12] *NumPy - Homepage.* http://numpy.scipy.org/. Version: Feb. 2012

[Ope12a] *OpenCV - Wiki.* http://opencv.willowgarage.com/wiki/. Version: Feb. 2012

[Ope12b] *Camera Calibration and 3d Reconstruction — opencv v2.1 documentation.* http://opencv.willowgarage.com/documentation/python/calib3d_camera_calibration_and_3d_reconstruction.html. Version: Feb. 2012

[PKB10] Pradeep, Vijay ; Konolige, Kurt ; Berger, Eric: Calibrating a multi-arm multi-sensor robot: A Bundle Adjustment Approach. In: *International Symposium on Experimental Robotics (ISER)*. New Delhi, India, 12 2010

[PM94] Park, F.C. ; Martin, B.J.: Robot sensor calibration: solving AX=XB on the Euclidean group. In: *Robotics and Automation, IEEE Transactions on* 10 (1994), oct, Nr. 5, S. 717 –721. http://dx.doi.org/10.1109/70.326576. – DOI 10.1109/70.326576. – ISSN 1042–296X

[pr212a] *PR2 Homepage.* http://www.willowgarage.com/pages/pr2/overview. Version: Feb. 2012

[pr212b] *pr2_calibration - ROS Wiki.* http://www.ros.org/wiki/pr2_calibration. Version: Feb. 2012

[Rie94] Riedmiller, Martin: *Rprop - Description and Implementation Details.* http://citeseerx.ist.psu.edu/viewdoc/download;?doi=10.1.1.21.3428&rep=rep1&type=pdf. Version: 1994

[ROS12a] *ROS Wiki.* http://www.ros.org/wiki/. Version: Feb. 2012

[ros12b] *rostest - ROS Wiki.* http://www.ros.org/wiki/rostest. Version: Feb. 2012

[RT99] Roy, N. ; Thrun, S.: Online self-calibration for mobile robots. In: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on* Bd. 3, 1999, S. 2292–2297 vol.3

[rvi12] *rviz - ROS Wiki.* http://www.ros.org/wiki/rviz. Version: Feb. 2012

[SHV04] Spong, Mark W. ; Hutchinson, Seth ; Vidyasagar, M.: *Robot Dynamics and Control*. http://smpp.northwestern.edu/savedLiterature/Spong_Textbook.pdf. Version: 2, 2004

[SIP12] *SIP - Riverbank Computing*. http://riverbankcomputing.co.uk/software/sip/intro. Version: Feb. 2012

[SOSB09] Schulz, Hannes ; Ott, Lionel ; Sturm, Juergen ; Burgard, Wolfram: Learning Kinematics from Direct Self-Observation using Nearest-Neighbor Methods. In: *German Workshop on Robotics 2009, Proc. of the*. Braunschweig, Germany, 2009

[SPB09] Sturm, Jürgen ; Plagemann, Christian ; Burgard, Wolfram: Body schema learning for robotic manipulators from visual self-perception. In: *Journal of Physiology-Paris* 103 (2009), Nr. 3-5, 220–231. http://dx.doi.org/DOI:10.1016/j.jphysparis.2009.08.005. – DOI DOI: 10.1016/j.jphysparis.2009.08.005. – ISSN 0928–4257. – Neurorobotics

[SSB09] Stückler, Jörg ; Schreiber, Michael ; Behnke, Sven: Dynamaid, an Anthropomorphic Robot for Research on Domestic Service Applications. In: *In Proceedings of the 4th European Conference on Mobile Robots (ECMR)*. Dubrovnik, Croatia, September 2009

[TBF06] Thrun, Sebastian ; Burgard, Wolfram ; Fox, Dieter: *Probabilistic Robotics*. Cambridge, Mass. : MIT Press, 2006 (Intelligent robotics and autonomous agents). – ISBN 978–0–262–20162–9

[tf12] *tf - ROS Wiki*. http://www.ros.org/wiki/tf. Version: Feb. 2012

[TkI12] *TkInter - Homepage*. http://wiki.python.org/moin/TkInter. Version: Feb. 2012

[UAA+09] Ulbrich, S. ; Angulo, V.R. de ; Asfour, T. ; Torras, C. ; Dillmann, R.: Rapid learning of humanoid body schemas with Kinematic Bézier Maps. In: *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*, 2009, S. 431–438

[urd12] *urdf - ROS Wiki*. http://www.ros.org/wiki/urdf. Version: Feb. 2012

[VPMMT11] Vicentini, Federico ; Pedrocchi, Nicola ; Malosio, Matteo ; Molinari Tosatti, Lorenzo: High-accuracy hand-eye calibration from motion on manifolds. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, 2011. – ISSN 2153–0858, S. 3327 –3334

[Wil12] *Willow Garage Homepage*. http://www.willowgarage.com/. Version: Feb. 2012

[WxP12] *WxPython - Homepage*. `http://wxpython.org//`. Version: Feb. 2012

[YAM12] *YAML - Homepage*. `http://yaml.org/`. Version: Feb. 2012

[Zha11] Zhao, Zijian: Hand-eye calibration using convex optimization. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011. – ISSN 1050–4729, S. 2947 –2952