Diplomarbeit

Automatische Roboterkalibrierung für den humanoiden Roboter NAO

Tobias Kastner

Bremen, 8. April 2014

Betreuer: Dr. Thomas Röfer
Dr. Tim Laue
Gutachter: Dr. Thomas Röfer
Prof. Dr.-Ing Udo Frese

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Bremen, den 8. April 2014

Tobias Kastner

Zusammenfassung

Die Einführung des humanoiden Roboters *NAO* als neue Plattform der *Standard Platform League* des *RoboCups* stellt die Teams vor die Herausforderung neue Verfahren für die Fortbewegung und das Schießen mit nur zwei Beinen zu entwickeln. Die Qualität dieser Bewegungen hängt stark von der Kalibrierung der Roboter ab. Fehler, die sich durch unpräzise Montage, Abnutzung oder Beschädigung des Roboters ergeben, werden mit der *Roboterkalibrierung* ausgeglichen. Die bisherigen Verfahren der Kalibrierung des *NAO*s sind sehr zeitintensiv und müssen von Nutzern manuell durchgeführt werden.

In dieser Arbeit wird ein automatisches Verfahren der Roboterkalibrierung untersucht, welches speziell für den *NAO* konzipiert ist. Das Ziel dieser Kalibrierung ist die Verbesserung der Mobilität des *NAO*s, sowie die Entlastung der Nutzer der Roboter.

Der Roboter bewegt ein an den Füßen befestigtes Schachbrett, welches von der Kamera des *NAO*s beobachtet wird. Mit einem Optimierungsverfahren wird ein mathematisches Modell des Roboters angepasst, so dass das Modell korrekte Vorhersagen über die Position des Schachbretts im Kamerabild treffen kann. Ein Roboter gilt als kalibriert, wenn die Position des Schachbretts im Bild mit der Positionsvorhersage des Modells übereinstimmt und die optimierten Parameter des Modells die Mobilität des *NAO*s verbessern.

Für die Umsetzung des Verfahrens wird die Roboterkalibrierung als Problem der kleinsten Quadrate formuliert und mit dem *Levenberg-Marquardt*-Algorithmus gelöst. Darüberhinaus werden die Implementierungen der Schachbretterkennung und der Bewegungssteuerung, sowie die automatische Steuerung der Kalibrierung mit Hilfe eines endlichen Automaten vorgestellt.

Inhaltsverzeichnis

1 Einleitung

Diese Arbeit ist thematisch im Bereich der Robotik angesiedelt und beschäftigt sich mit der automatischen Kalibrierung eines humanoiden Roboters, welcher dem Aufbau des menschlichen Körpers nachempfunden ist. Der Begriff Robotik wird von Thrun u.a.[TBF05] folgendermaßen definiert:

> „Robotics is the science of perceiving and manipulating the physical world through computer-controlled devices. Examples of successful robotic systems include mobile platforms for planetary exploration, industrial robotics arms in assembly lines, cars that travel by themselves, and manipulators that assist surgeons. Robotics systems are situated in the physical world, perceive information on their environments through sensors, and manipulate through physical forces.“

Die *Robotic Industries Association* führt folgende Definition[1] eines Roboters auf:

> „A robot is a reprogrammable, multifunctional manipulator designed to move material, parts, tools or specialized devices through variable programmed motions for the performance of a variety of tasks.“

Laut dieser Definitionen handelt es sich bei einem Roboter um eine programmierbare, technische Apparatur, welche für die Ausübung einer bestimmten Aufgabe in einer festgelegten Umgebung konzipiert ist. Um mit der Umwelt interagieren zu können, verfügen Roboter über Aktoren und optionale Endeffektoren, welche aufgrund von Beobachtungen und Messungen mit geeigneten Sensoren zielführend eingesetzt werden. Ein Aktor kann z. B. ein Arm und ein Endeffektor ein Werkzeug oder eine Hand sein.

Das Zusammenspiel von Sensoren und Aktoren wird von Computerprogrammen gesteuert. Solche Programme verfügen zumeist über interne Modelle der Umgebung und des Roboters selbst, um genauer planen zu können, welche Schritte zur Erfüllung der aktuellen Aufgabe als nächste ausgeführt werden müssen oder um auf unerwartete Ereignisse geeignet reagieren zu können. Für einen humanoiden Roboter ist beispielsweise hilfreich, wenn dieser berechnen kann, wo sich Hände und Füße im Raum relativ zur Roboterbasis befinden, um eventuelle Kollisionen zu vermeiden oder um das Gleichgewicht zu halten. Weicht das tatsächliche Modell des Roboters vom intern verwendeten ab, ist nicht mehr garantiert, dass dieser korrekt funktioniert. Abweichungen ergeben sich zum einen durch nicht berücksichtigte Parameter im internen Modell, zum anderen durch Verschleiß und unpräziser Montage der Sensoren und des Roboters selbst. Im

[1] `http://definitions.uslegal.com/r/robotics/`

Falle von Montagefehlern und Verschleiß müssen die betroffenen Sensoren entsprechend kalibriert werden.

Unter Kalibrierung wird der Prozess der wiederholbar genauen Feststellung der Abweichung eines Messkörpers zu einem anderen Körper, welcher zuvor als das Normal oder der Standard definiert wurde, verstanden. Die Korrektur dieser Abweichung ist laut des *VIM*[08] (International Vocabulary of Metrology) des *BIPM*[2] (International Bureau of Weights and Measures) als ein zusätzlicher Schritt anzusehen und darf nicht mit Kalibrierung verwechselt werden.

Die Problematik von unzureichend kalibrierten Robotern können anhand von zwei einfachen Beispielen angeführt werden. Für einen Industrieroboter ist es nicht akzeptabel, wenn dieser aufgrund von unpräzise verbauten Servomotoren des Roboterarms, wenige Millimeter neben der eigentlichen Naht schweißt. Auf einen humanoiden Roboter übertragen, zeigt sich die Problematik bei der Nachahmung des menschlichen Ganges, falls die geforderten Fußpositionen nicht korrekt angesteuert werden können. Im besten Fall führt dies zu einem instabilen, langsamen Gang oder führt im schlimmsten Falle zu kostspieligen Beschädigungen als Resultat von Stürzen.

Die Fehler der Positionierung von Servomotoren zu korrigieren, ist die Aufgabe der sogenannten *Roboterkalibrierung* (siehe Abschnitt 3.1). Im Kontext dieser Arbeit wird am Beispiel des humanoiden Roboters *NAO* eine Methode zur automatischen *Roboterkalibrierung* untersucht und vorgestellt.

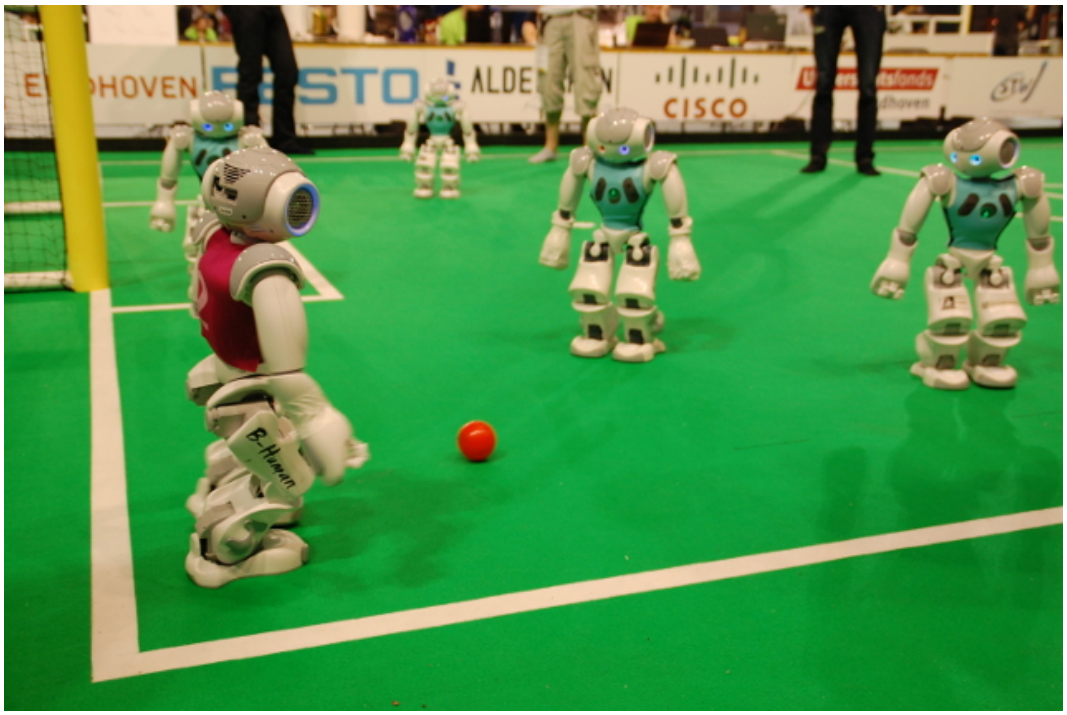
1.1 Motivation

Um die Entwicklung innerhalb der Robotik voranzutreiben, werden diverse Wettbewerbe mit den verschiedensten Szenarien ausgetragen. Die *DARPA Grand Challenge*[3] stellte Entwickler vor die Aufgabe, autonome Fahrzeuge zu entwickeln und diese einen festgelegten Pfad abfahren zu lassen. Der im Jahre 2013 zum ersten Mal ausgetragene *SpaceBot Cup*[4] des *Deutsche Zentrums für Luft- und Raumfahrt* (DLR) beschäftigte die teilnehmenden Teams mit den Problemen von Robotern, die auf anderen Planeten agieren sollen.

Diese Arbeit ist durch den *RoboCup*[Kit+95], eine internationale Initiative zur Förderung und Forschung der künstlichen Intelligenz und Robotik, motiviert. Das selbst verfasste Ziel dieses jährlich stattfindenden Wettbewerbs ist es, den im Jahr 2050 amtierenden Fußballweltmeister mit autonomen Robotern nach offiziellen FIFA-REGELN zu

[2] http://www.bipm.org/
[3] http://archive.darpa.mil/grandchallenge/index.asp
[4] http://www.dlr.de/rd/desktopdefault.aspx/tabid-8101/13875_read-35268/

besiegen. Die *Standard Platform League* (*SPL*) ist eine der vielen verschiedenen Ligen



(a)

Abbildung 1.1: Eine typische Spielszene der *Standard Platform League* (`http://www.informatik.uni-bremen.de/~judy/robocup2013/DSC_0127.JPG`, abgerufen am 25.03.2014)

des Fußball-Wettbewerbs (siehe Abbildung 1.1) des *RoboCup*s und stellt gleichzeitig den Rahmen dieser Arbeit dar.

Teilnehmer dieser Liga entwickeln ihre eigenen Programme für eine vorher festgelegte Roboter-Plattform. Eigenständige Modifikationen des Roboters sind nicht erlaubt um sicherzustellen, dass für alle teilnehmenden Teams identische Bedingungen herrschen. Es handelt sich um einen reinen Programmier-Wettbewerb.

Seit 2008 wird der humanoide Roboter *NAO*[Gou+08] von *Aldebaran Robotics*[5] verwendet, welcher den *AIBO* (ein hundeähnlicher, vierbeiniger Roboter) von *Sony* ablöst. Der bisher erfolgreichste Teilnehmer in der *SPL* mit der *NAO*-Plattform ist das Team *B-Human*[6], ein studentisches Projekt des Fachbereichs Informatik der *Universität Bremen*[7] mit Unterstützung der Forschungsgruppe *Cyber-Physical Systems* des *Deutschen Forschungszentrums für Künstliche Intelligenz GmbH*[8]. In den Jahren 2009, 2010, 2011

[5] `www.aldebaran.com`
[6] `http://www.b-human.de`
[7] `http://www.uni-bremen.de`
[8] `http://www.dfki.de/web/forschung/cps`

und 2013 konnte das Team den *RoboCup*[9] gewinnen und damit den Weltmeistertitel erlangen. Die von *B-Human* entwickelte Software wird in der Regel nach einer Weltmeisterschaft veröffentlicht. Die Veröffentlichung[Röf+12] des Jahres 2012 stellt dabei die Grundlage für diese Arbeit dar.

Stabiles, schnelles Laufen, sowie präzise Schüsse und robuste Lokalisierung sind die Stärken der *B-Human*-Software unter der Voraussetzung, dass der *NAO* korrekt kalibriert ist. Aktuell ist die Kalibrierung ein manuelles Verfahren, welches viel Zeit beansprucht und einiges an Erfahrung der Benutzer voraussetzt, da die Korrekturen der jeweiligen fehlerbehafteten Gelenke über Augenmaß geschätzt werden. Problematisch dabei ist, dass kein eindeutiges Kriterium existiert, welches aussagt, ob ein Roboter korrekt kalibriert ist und dass die Roboter aufgrund von Belastungen während des Spiels hinterher neu kalibriert werden müssen. Da die Kalibrierung eine wichtige Voraussetzung für den Erfolg von B-Human darstellt und das Team mittlerweile über eine beachtliche Anzahl von *NAO*s verfügt, sind viele Mitglieder des Teams während eines Wettkampfes mit der manuellen Kalibrierung beschäftigt.

Die vorliegende Arbeit beschäftigt sich mit der Untersuchung einer automatischen Lösung der Kalibrierung, mit dem Ziel sämtliche *NAO*s für Experimente und Fußballspiele einsatzfähig zu machen und gleichzeitig die Mitglieder von *B-Human* zu entlasten.

1.2 Aufbau der Arbeit

Die folgenden Abschnitte des ersten Kapitels befassen sich mit dem Stand der Technik zur Kalibrierung von Robotern und der Formulierung von Anforderungen an das im Rahmen dieser Arbeit entstandene Verfahren.

Kapitel 2 widmet sich der verwendeten Hardware- und Software-Plattform.

Kapitel 3 beschäftigt sich mit Definition des Begriffs der Roboterkalibrierung und den wichtigsten mathematische Grundlagen. Zu diesen zählen unter anderem die homogene Koordinatentransformation, die Vorwärtskinematik und das Modell der Lochkamera.

Kapitel 4 umfasst die Implementierung der Roboterkalibrierung für den *NAO*. Zu Beginn wird die Kalibrierung als Problem der kleinsten Quadrate definiert. Die weiteren Abschnitte beschäftigen sich mit einer Schachbretterkennung, einer Bewegungssteuerung und der Umsetzung eines Optimieralgorithmus.

Kaptiel 5 beinhaltet die Evaluation des in Kapitel 4 vorgestellten und implementierten Verfahrens.

[9] `http://www.robocup.org`

Im sechsten und letzten Kapitel werden die Tests und Ergebnisse zu einem Fazit zusammengefasst und mögliche Verbesserungen im Ausblick aufgelistet.

1.3 Stand der Technik

Die meisten Arbeiten im Bereich der Roboterkalbrierung beschäftigen sich mit der Verbesserung der Positionierungsgenauigkeit von Industrierobotern, unter anderem als Manipulatoren bekannt. In der Automobilfertigung kommen oft sogenannte Knickarmroboter zum Einsatz und die typischerweise über fünf bis sechs Freiheitsgrade verfügen. Ebenfalls existieren diverse Veröffentlichungen aus der Forschung über Roboter, wie beispielsweise der *Rollin' Justin*[Bäu+11] des *DLR*, der *Robonaut*[Amb+04] oder der *PR2*[Wyr+08] von *Willow Garage*. Gemein ist diesen Arbeiten, dass die untersuchten Roboter über Arme als Aktoren verfügen und mit Hilfe dieser Werkzeuge bedienen, Nahrungsmittel zubereiten oder Bälle fangen.

Die von Nickels u.a.[Nic03], Pradeep u.a.[PKB10], Hubert u.a[HSB12] und Birbach u.a.[OF12] vorgestellten Lösungen zur Verbesserung der Positionierungsgenauigkeit basieren auf der sogenannten *Hand-eye Calibration*[SH09], unterscheiden sich jedoch in der Wahl des Optimierungsverfahrens und des verwendeten Kalibrierkörpers. Beim Verfahhren von Nickels u.a.[Nic03] können unter der Annahme einer *a priori* Kamerakalibrierung entweder nur die Nullagenfehler der Gelenkwinkel oder die kompletten Parameter der *Denavit-Hartenberg-Konvention*[DH55] berechnet werden. Für die Optimierung wird der *Downhill-Simplex*-Algorithmus und als Kalibrierkörper ein kugelförmiges Werkzeug verwendet.

Für die Methode von Pradeep u.a.[PKB10] wird an der Hand des *PR2* ein Schachbrett angebracht. Zusätzlich zur Kinematik werden die Posen der Kameras und Laser-Entfernungsmesser verbessert. Angewandt wird der Optimierungsalgorithmus von Levenberg[Lev44] und Marquardt[Mar63], ein Verfahren zur Optimierung von Problemen, die sich im Sinne der Summe der kleinsten Quadrate definieren lassen.

Im Unterschied zur Kalibrierung des *PR2* werden für die Roboter *Cosero* und *Dynamaid* zwar auch Schachbretter als Kalibrierkörper verwendet, allerdings werden die relevanten Parameter mit der *Maximum-a-posteriori-Methode* (*MAP*) optimiert. Diese kommt laut Hubert u.a[HSB12] mit wenigen Testdaten aus und erzielt ähnlich gute Ergebnisse, wie das erwähnte *Levenberg-Marquardt*-Verfahren.

Die von Birbach u.a.[OF12] dargestellte Lösung kommt ohne einen externen Kalibrierkörper aus und verwendet spezielle Markierungen, die an den Handgelenken des *Rollin' Justin* angebracht sind. Unter der Verwendung des *Levenberg-Marquardt*-Algorithmus, werden die Nullagenfehler, die Elastizitäten der Gelenke, die Kalibrierung der Iner-

tialsensoren und die Posen, sowie die intrinsischen Parameter der beiden Kameras geschätzt.

Die bis hierhin genannten Arbeiten, basieren alle auf Methoden der computergestützten Bildverarbeitung. Um sich einen allgemeineren Überblick über das Thema Roboterkalibrierung zu verschaffen, empfiehlt sich das Studium der Ausarbeitung von Elatta u.a.[Ela+04], eine Zusammenfassung der verschiedenen Kalibrierungs-Methoden, welche auch Lösungen enthält, die nicht auf visuelle Sensoren angewiesen sind.

Im Gegensatz zu den bisher vorgestellten Arbeiten existieren sehr wenige Veröffentlichungen für die Kalibrierung speziell für Roboterbeine, welche beispielsweise für Laufen und Balancieren von großer Bedeutung sind. Roboterbeine agieren zwar auch im dreidimensionalen kartesischen Raum, verfügen aber über die Einschränkung, dass der Fuß mit dem Boden Kontakt hat und die Fußsohle dabei möglichst eben zum Boden sein muss.

Von Yamane u.a.[Yam] wird ein Verfahren vorgestellt, welches nur mit Gelenkwinkelmessugnen und Inertialsensoren auskommt um die Beine des *CMU/Sarcos* zu kalibrieren, ohne dabei auf externe Hilfsmittel angewiesen zu sein. Hierfür werden die Beine manuell, bei ständigen Kontakt der Fußsohlen mit dem Boden, bewegt. An den beiden Oberschenkeln ist jeweils ein Inertialsensor angebracht, welcher die Orientierung des Schenkels bestimmt. Die Orientierungen werden zusammen mit den jeweiligen Gelenkwinkeln mit $500Hz$, während der manuellen Beinbewegungen, aufgezeichnet. Die Nullagenfehler werden dann mit dem Verfahren der konjugierten Gradienten berechnet, indem die Messungen und eine geeignete Kostenfunktion bereitgestellt werden. Die Kostenfunktion setzt sich dabei aus der Abweichung der Messung und der Vorhersage der Orientierung des Oberschenkels, sowie der Höhenvariation von bestimmten Punkten auf der Fußsohle des Roboters zusammen. Der Vorteil dieser Art der Kalibrierung besteht darin, dass keine externen Hilfsmittel benötigt und dass die Messungen mit Bodenkontakt durchgeführt werden. Allerdings müssen die Bewegungen von Nutzern des Roboters manuell durchgeführt werden.

Ein Verfahren um speziell die Beine des *NAO*s zu kalibrieren wird in der Bachelorarbeit von Markowsky[Mar11] vorgestellt. Es wurde untersucht, ob mit Hilfe eines Bergsteiger-Algorithmus die Stromlasten der Beingelenke und die Gewichtsverteilung beider Füße ausgleichbar sind. Es wurde angenommen, dass der *NAO* korrekt kalibriert ist, wenn die Stromlasten der beiden Beine identisch sind. Die verwendete Kostenfunktion besteht aus den Differenzen der Stromlast der Gelenke des linken und rechten Beines, sowie die Differenz der Gewichtsverteilung des linken und rechten Fußes. Nach einer Parameterbestimmung des Optimierers erfolgt eine neue Auswertung der Kostenfunktion, indem der *NAO* für eine kurze Zeit auf der Stelle läuft, damit sich die neuen Parameter auch auf die Gelenke auswirken können. Die Verteilungen werden erst wieder gemessen, wenn der *NAO* wieder ruhig auf der Stelle steht. Diese Methode scheitert allerdings

aufgrund der Sensorik des *NAO*s. Vor allem die Gewichtssensoren an den Fußsohlen sind äußerst Fehleranfällig und funktionieren oft überhaupt nicht. Außerdem zeigen die dargestellten Ergebnisse nicht auf, wie gut der Roboter nach einer solchen Kalibrierung laufen und schießen kann.

Die von *B-Human* verwendete Lösung der Kalibrierung des *NAO*s wird in der Dokumenation[Röf+13b] dargestellt. Es handelt sich dabei um entweder eine komplett manuelle Methode, bei der ein Benutzer die Nullagenfehler via Augenmaß abschätzt oder eine halbautomatische Herangehensweise. Zweitere bedingt, dass die Fußsohlen parallel zu Boden sind, was in einem ersten manuellen Schritt erfolgt. Anschließend wird eine halbautomatische Kamerakalibrierung durchgeführt, welche zusätzlich die Rotation des Oberkörpers schätzt. Die Gelenke werden dann so angepasst, dass diese Rotationen ausgeglichen werden, was dazu führt, dass der *NAO* aufrechter steht. Allerdings kommt es oft vor, diese Schritte mehrfach wiederholt werden müssen, um am Ende eine annehmbare Kalibrierung zu erhalten.

1.4 Ziel und Anforderungen

Die vorgestellten Veröffentlichungen von Nickels u.a.[Nic03], Pradeep u.a.[PKB10], Hubert u.a[HSB12] und Birbach u.a.[OF12] zur Roboterkalibrierung verfügen über eine ähnliche Funktionsweise. Die Roboter bewegen ihre Aktoren durch ihren Arbeitsraum, wobei die Position eines Kalibrierkörpers mit einer Kamera gemessen wird. Die Abweichungen der Messungen von den Vorhersagen eines mathematischen Robotermodells werden durch verschiedene Optimierungsverfahren minimiert. Ziel dieser Arbeit ist, die Entwicklung einer automatischen Roboterkalibrierung für den *NAO*, nach dem Vorbild der soeben beschriebenen Verfahrensweise. Sie muss die folgenden Anforderungen erfüllen:

A1 - Schneller als manuelle Methode

Die automatische Roboterkalibrierung soll im Gesamten weniger Zeit benötigen, als die bisher eingesetzten manuellen und halbautomatischen Methoden von B-Human.

Wird ein *NAO* kalibriert, steht er nicht für Experimente zur Verfügung. Daher ist es erforderlich, dass die automatische Kalibrierung nicht mehr Zeit in Anspruch nimmt als die manuelle Methode.

A2 - Einfache Handhabung

Die automatische Roboterkalibrierung muss ohne komplexe Vorbereitungen durch einen Benutzer auskommen.

Je aufwändiger ein Verfahren ist, umso wahrscheinlicher schleichen sich unbeabsichtigte Fehler ein.

A3 - Robustheit

Die automatische Roboterkalibrierung muss für alle NAO*s in verschiedenen Lokalitäten anwendbar sein und plausible Kalibrierungen erzeugen.*

Diese Anforderung betrifft sämtliche Softwarekomponenten der Roboterkalibrierung: Die Algorithmen der Erkennung des Kalibrierkörpers müssen diesen unter verschiedenen Gelenkstellungen immer korrekt feststellen können. Dabei muss die Bewegungssteuerung die geforderten Gelenkkonfigurationen auch präzise ansteuern können. Die Bildverarbeitung muss äußerst tolerant gegenüber Beleuchtungsveränderungen sein, da die Beleuchtung der verschiedenen Austragungsorte für Roboterfußballspiele oft unterschiedlich ist und von wechselhaften Wetterbedingungen beeinflusst wird. Der verwendete Optimierungsalgorithmus darf für gleiche Eingabewerte keine verschiedenen Ausgaben erzeugen. Außerdem müssen die optimierten Parameter einen plausiblen Wertebereich aufweisen.

A4 -Qualität der Kalibrierung

Die Ergebnisse der automatischen Roboterkalibrierung sollen die gleiche Qualität aufweisen wie die Resultate wie die manuelle Kalibrierung.

Ein *NAO* muss nach erfolgter Kalibrierung in der Lage sein, erfolgreich ein Fußballspiel zu bestreiten. Hierfür muss er stabil laufen können und die Distanz zu den Feldlinien, dem Spielball, den Toren sowie gegnerischen Robotern korrekt berechnen können. Die Durchführung der manuellen Roboterkalibrierung ist, abgesehen von schwer beschädigten Robotern, immer in der Lage einen *NAO* für ein Spiel einsatzfähig zu machen.

A5 - Wiederholbarkeit

Die optimierten Parameter der Roboterkalibrierung sollen für mehrfache Wiederholung gleiche Werte aufweisen.

Andernfalls kann die verwendete Methodik angezweifelt werden.

2 Plattform

In diesem Kapitel werden die Eigenschaften des *NAO* untersucht und die Besonderheiten des verwendeten Software Frameworks von *B-Human* aufgezeigt.

2.1 NAO

Der *NAO* ist ein humanoider Roboter der französischen Firma *Aldebaran-Robotics*, welcher speziell für Forschung und Lehre entwickelt wurde und aus diesem Grund einen relativ günstigen Anschaffungspreis von circa 12.000 € aufweist. Seit seiner Vorstellung im Jahr 2006 wurden neue Versionen dieses Roboters veröffentlicht, wobei diese sich z. B. in der Größe, dem Gewicht, die Anzahl der Freiheitsgrade und der Sensorik unterscheiden. Die meiste Prominenz erlangte der *NAO* durch die Einführung als Standard-Plattform der *SPL* des *Robocups* im Jahre 2008.

Ausstattung

Die zu diesem Zeitpunkt aktuellste Version des *NAO* stellt der *V.4.0 H.25*[Rob] dar, welcher über 25 Freiheitsgrade verfügt. Eine Besonderheit der Konstruktion des *NAO*s stellt das Hüftgelenk dar, welches im Gegensatz zum menschlichen Skelett eine Kopplung der Beine bewirkt. Der Mensch ist in der Lage, seine Beine unabhängig voneinander zu bewegen, d. h. er kann sein Bein zur Seite und nach vorne heben oder es nach innen oder außen drehen, ohne dass das andere Bein dadurch beeinflusst wird. Das Verdrehen der Beine nach innen oder außen wird beim *NAO* durch einen einzigen Motor gesteuert. Dreht das linke Bein nach außen, so dreht sich auch das rechte Bein entsprechend nach außen.

Die Gelenke sind mit Servomotoren realisiert, die in der Lage sind Gelenkwinkel und anliegende Spannung zu messen. Winkel können mit einer Präzision von $0,1°$ gesetzt bzw. gemessen werden und verfügen laut Gouaillier u.a[GCK10] ein Gelenkspiel von $\pm 5°$.

Zusätzlich verfügt der *NAO* über zwei Lautsprecher im Kopf sowie diverse LEDs, die über den gesamten Körper verteilt sind. Zu den Sensoren zählen zwei Kameras, vier Mikrofone, ein zwei-Achsen Gyroskop und ein drei-Achsen Beschleunigungsmesser im Oberkörper, Drucksensoren unter den Fußsohlen und Kontaktsensoren auf dem Kopf und an den Fußspitzen.

Abbildung 2.1: Der *NAO* (`http://www.sedoniatech.com.au/imgs/nao_debout.jpg`, abgerufen am 27.03.2014).

Die beiden Kameras befinden sich im „Mund“ und in der „Stirn“ des Kopfes und nehmen 30 Bilder pro Sekunde auf, bei einer maximalen Auflösung von 1280×960 Pixeln.

Als Prozessor wird ein *Intel ATOM Z530* mit einer Taktung von $1,6\ GHz$ verwendet und verfügt dabei über 1 GB Arbeitsspeicher. Als Betriebssystem kommt ein spezielles Linux für eingebette Systeme zum Einsatz.

2.2 Software

Die Software dieser Arbeit wurde komplett mit der *C++*-Programmiersprache[1] geschrieben, was zum einen der Erzeugung von effizienten und schnellen Code und zum anderen dem verwendeten *B-Human*-Framework[Röf+13b] geschuldet ist, da dieses ebenfalls in *C++* geschrieben ist.

[1] C++98-Standard

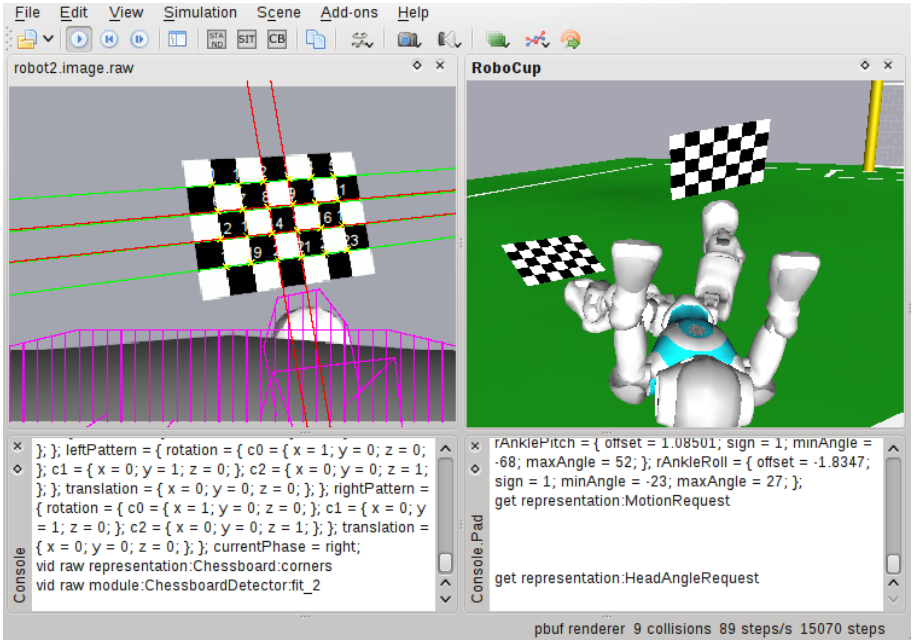


Abbildung 2.2: *SimRobot* im Einsatz: eine simulierte Roboterkalibrierung.

Das von *B-Human* entwickelte Framework ermöglicht eine modulare Entwicklung von Roboter-Kontrolprogrammen. Eine bestimmte Tätigkeit oder Aufgabe, wie z. B. das Erkennen des Spielball, wird als ein *Modul* implementiert. Jedes *Modul* stellt seine Ergebnisse in Form von sogenannten *Repräsentationen* bereit, welche von anderen *Modulen* als Eingabedaten verwendet werden können. Die Ballerkennung benötigt beispielsweise ein Kamerabild, welches von einem anderen *Modul* zuvor als *Repräsentation* bereitgestellt werden muss. Durch die Definition solcher Abhängigkeiten ergibt sich eine Ausführungsreihenfolge der verschiedenen *Module*, welche vom Framework automatisch ermittelt und auf mögliche zirkuläre Abhängigkeiten überprüft wird.

Module werden einer *Kategorie* (*Perception*, *Modeling*, *BehaviorControl*, *MotionControl* oder *Sensing*) zugeordnet, damit man diese logisch gruppieren und, viel wichtiger, einem *Prozess* zuordnen kann. Ein *Prozess* stellt eine Umgebung dar, in dessen Kontext *Module* ausgeführt werden. Ein Unterscheidungsmerkmal von *Prozessen* stellt dabei die Aufrufhäufigkeit der *Module* pro Zeiteinheit dar. Implementiert sind diese als *Threads*. Von diesen Prozessen gibt es insgesamt drei: *Motion*, *Cognition* und *Debug*:

- Der *Cognition*-Prozess kommuniziert via *Video4Linux* mit den Kameras und verarbeitet deren Bilder mit einer Frequenz von 60 Hz. Das Erkennen von Torpfosten, Feldlinien, Bällen und anderer Roboter (*Perception*) wird hier erledigt, woraufhin diese Informationen dazu verwendet werden, um Modelle wie z.B. die Position des Roboters auf dem Feld oder die Geschwindigkeit des Balls (*Mode-*

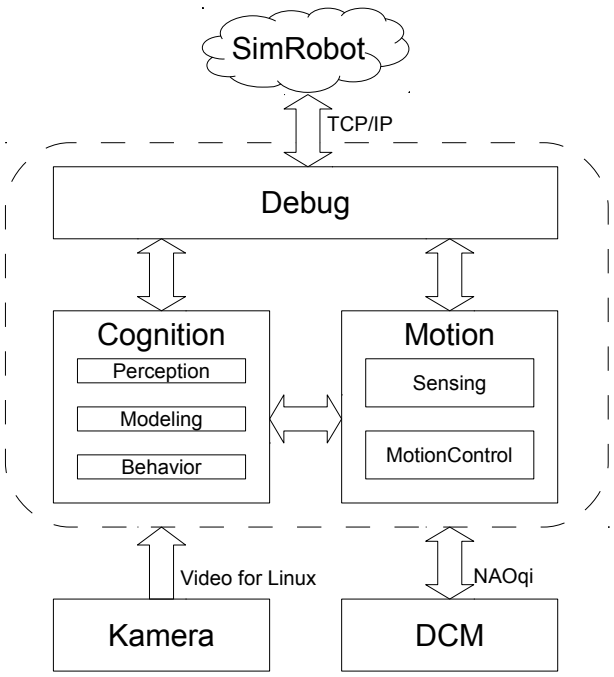


Abbildung 2.3: Die Prozesse *Cognition*, *Motion* und *Debug* in Zusammenarbeit mit *Video4Linux* und *NAOqi*. Die im gestrichelten Rahmen befindlichen Komponenten sind Bestandteil der Software von *B-Human*, welche auf dem *NAO* läuft.

ling) zu berechnen. Diese Modelle werden dann in der Verhaltenssteuerung (*BehaviorControl*) dazu verwendet, intelligente Entscheidungen auf dem Spielfeld zu treffen.

- Die Sensoren (*Sensing*) und Gelenkwinkel des *NAO* werden im *Motion*-Prozess mit 100 Hz ausgelesen bzw. gesetzt. Hierfür wird mit der Software des Herstellers (*NAOqi*) kommuniziert, da diese über exklusiven Zugriff auf die Sensorik und Aktorik des *NAO*s verfügt (*DCM: Device Control Manager*). Um einen Datenaustausch mit dem *Device Control Manager* (*DCM*) ermöglichen zu können, wird über ein eigenes *NAOqi*-Modul auf die Sensoren zugegriffen, dessen Werte in einen *Shared-Memory* geschrieben werden. Im Code des *B-Human* Frameworks gibt es ein entsprechendes Modul, welches ebenfalls Zugriff auf diesen *Shared Memory* hat. Bewegungen wie das Laufen oder das Treten gegen einen Ball werden von *Motion* durchgeführt, wobei die hierfür notwendigen Befehle aus *Cognition* kommen, weswegen bestimmte *Repräsentationen* zwischen den beiden *Prozessen* geteilt werden.
- Der *Debug*-Prozess hat die Aufgabe mit der Simulations-Software *SimRobot* via *TCP/IP* zu kommunizieren, welche auf einem entfernten Rechner läuft. Ein Nutzer kann somit Werte von *Repräsentationen* auslesen und setzen, Datenplots er-

stellen, Kamerabilder mit Hilfszeichnungen versehen und bestimmte Programmzeilen per Kommando aktivieren.

SimRobot

Die bereits erwähnte Software *SimRobot*[LSR06] (siehe Abbildung 2.2) ist in der Lage, beliebig viele *NAO*s zu simulieren. So können *Module* und Algorithmen unter perfekten, simulierten Bedingungen getestet werden, bevor man potentiell fehlerhafte Software auf einen *NAO* installiert und diesen dadurch zu Schaden bringt.

SimRobot ermöglicht es, verschieden Roboterarten in beliebigen Situationen zu simuleren. Um dies zu gewährleisten, gibt es eine zu *XML* ähnliche Auszeichnungssprache.

SimRobot kann *Log*-Dateien abspielen, wobei diese zuvor mit dem *NAO* aufgezeichnet wurden. Eine *Log*-Datei enthält die Daten von beliebigen *Repräsentationen*. Im Zuge dieser Arbeit wurden die Bildverarbeitungsalgorithmen mit Hilfe der Logs entwickelt, indem die Kamerabilder aufgezeichnet wurden.

3 Grundlagen

In diesem Kapitel werden die wichtigsten Grundlagen erklärt. Einleitend wird der Begrif der *Roboterkalibrierung* definiert. Die folgenden Abschnitte beschäftigen sich mit der Festlegung des Koordinatensystems, der homogenen Koordinatentransformation, dem Prinzip der Vorwärtskinematik, dem Modell der Lochkamera und der linearen sowie nichtlinearen Optimierung.

3.1 Roboterkalibrierung

Dieser Abschnitt beschäftigt sich mit der Definition der bereits erwähnten *Roboterkalibrierung*. Dessen Ziel ist laut Wiest[Wie01] die Verbesserung der Positionierung der Effektoren im Sinne von absoluter Positioniergenauigkeit und Wiederholungsgenauigkeit. Absolute Positioniergenauigkeit ist ein Maß dafür, wie gut ein Endeffektor eine gewünschte Position in der Welt erreicht. Wiederholungsgenauigkeit beschreibt, wie groß die Abweichung der Position des Effektors bei mehrfacher Ansteuerung eines gleichen Ziels aus entweder immer gleichen oder verschiedenen Startstellungen ist.

Ungenauigkeiten bei der Ansteuerung entstehen hauptsächlich durch falsche Montage der Motoren oder auch durch Abnutzung und Beschädigungen der Gelenke. In Tabelle 3.1 werden die Hauptfehlerquellen für Roboterkinematiken nach Beyer[Bey05] aufgelistet.

Geometrisch		Nicht-geometrisch	
Nullagenfehler	80 bis 90%	Gelenkelastizitäten	3 bis 8%
Längen- und Winkelfehler	5 bis 10%	Getriebefehler	1 bis 2%
Temperatureinflüsse	0 bis 10%	Stochastische Fehler	1 bis 2%

Tabelle 3.1: Die Fehlereinflüsse auf Roboterkinematiken.

Die Nullagenfehler haben hierbei den größten Einfluss. Nach Wiest[Wie01] wird ein Nullagenfehler als Abweichung der vom Roboterhersteller festgelegten nominellen, mechanischen Achsnullstellung definiert. Laut Beyer[Bey05] sind die elastischen Fehler, trotz eines maximalen Beitrags zum Fehler von 8%, in den meisten Fällen ignorierbar. Diese entstehen z. B. durch schlechte Lagerung und gering steife Materialien der Servos und Achsen.

Von Elatta u.a.[Ela+04] und Wiest[Wie01] wird die Roboterkalibrierung in mindestens vier aufeinander folgende Phasen unterteilt:

In der *Modellierungs*-Phase wird ein geeignetes, parametrierbares mathematisches Modell der Kinematik des jeweiligen Roboters erstellt, um eine möglichst genaue Vorhersage treffen zu können, wie sich der Roboter unter verschiedenen Konfigurationen verhalten wird. Hierfür wird in den meisten Fällen die *Denavit-Hartenberg-Konvention*[DH55] oder die *homogene Koordinatentransformation*[SK08] verwendet.

Während der *Messungs*-Phase werden verschiedene Gelenkstellungen angesteuert, wobei die Position des Endeffektors gemessen wird. Die Konfigurationen müssen so gewählt sein, dass jeder Parameter des Modells Verwendung findet und der Großteil des Arbeitsbereichs des Roboters abgedeckt wird.

Um die nun eventuell auftretenden Ungenauigkeiten auszugleichen, werden in der *Identifikations*-Phase Parameter des Modells und deren Belegung bestimmt, die dafür sorgen, dass Messung und Vorhersage besser zusammenpassen. Dieses Problem lässt sich oft im Sinne eines Problems der kleinsten Quadrate formulieren und mit numerischen Methoden lösen.

Die Anwendung der optimierten Parameter bzw. des aktualisierten Modells in die Steuerungs-Software ist Aufgabe der letzten Phase, der *Kompensation*. Nun sollten die Vorhersagen des Modells mit den Messungen der tatsächlichen Position der Endeffektoren übereinstimmen. Diese Phase ist der in *VIM*[08] erwähnte zusätzliche Schritt zur eigentlichen Kalibrierung.

3.2 Koordinatensystem

Für die automatische Gelenkkalibrierung ist es notwendig zu wissen, wo sich die einzelnen Gelenke im dreidimensionalen, kartesischen Raum relativ zum Ursprung des *NAO*s befinden. Dieser Ursprung befindet sich dabei zwischen den beiden Hüftgelenken im Inneren der Torso-Hülle. Die x-Achse zeigt in Richtung des Torsos bzw. nach vorne, die y-Achse nach links und die z-Achse nach oben. Ein solches Koordinatensystem wird auch als *rechtshändig* und *links drehend* bezeichnet.

3.3 Homogene Koordinatentransformation

Eine *homogene Koordinatentransformation*[SK08] T_B^A, im dreidimensionalen kartesischen Raum, kann als eine $4x4$ Matrix dargestellt werden und beinhaltet eine $3X3$ Rotationsmatrix R_B^A und einen Translationsvektor p_B^A.

$$T_B^A = \left(\begin{array}{c|c} R_B^A & p_B^A \\ \hline 0\ 0\ 0 & 1 \end{array}\right) \tag{3.1}$$

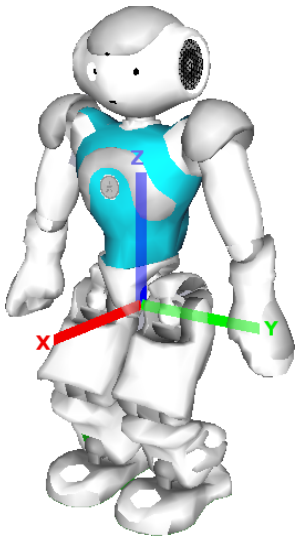


Abbildung 3.1: Die Orientierung und Lage des Koordinatensystems.

Die Matrix T_B^A beschreibt die Position und Orientierung (auch als *Pose* bekannt) des Koordinatensystems B in Koordinaten von Koordinatensystem A. Gleichzeitig kann T_B^A dazu verwendet werden, einen gegebenen Punkt p^B relativ zum Koordinatensystem B in einen Punkt p^A umzuwandeln, so dass p^B in Koordinaten relativ zum Koordinatenursprung von A vorliegt:

$$p^A = T_B^A * p^B. \tag{3.2}$$

Ein Punkt p^A, kann auch in Koordinaten von B umgewandelt werden:

$$p^B = (T_B^A)^{-1} * p^A. \tag{3.3}$$

Die Rotation eines Punktes um die $x - Achse$ um den Winkel α wird mit der folgenden Funktion bewerkstelligt:

$$Rot_x(\alpha) = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{3.4}$$

Für die $y - Achse$:

$$Rot_y(\alpha) = \left(\begin{array}{ccc|c} \cos\alpha & 0 & \sin\alpha & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\alpha & 0 & \cos\alpha & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{3.5}$$

Für die $z - Achse$:

$$Rot_z(\alpha) = \left(\begin{array}{ccc|c} \cos\alpha & -\sin\alpha & 0 & 0 \\ \sin\alpha & \cos\alpha & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{3.6}$$

Um einen Punkt entlang einer Achse zu verschieben, werden folgende Funktionen verwendet:

$$Trans_x(t) = \left(\begin{array}{ccc|c} 1 & 0 & 0 & t \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{3.7}$$

$$Trans_y(t) = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & t \\ 0 & 0 & 1 & 0 \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{3.8}$$

$$Trans_z(t) = \left(\begin{array}{ccc|c} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & t \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \tag{3.9}$$

3.4 Vorwärtsinematik

Die *Vorwärtskinematik* bestimmt anhand von Gelenkwinkeln, wo sich bestimmte Körperteile des *NAO*s relativ zur Roboterbasis befinden. Dies wird mit der Verknüpfung von homogenen Koordinatentransformationen bewerkstelligt, indem vom Roboterursprung ausgehend die jeweiligen Transformationen (Translation und Rotation) zum nächsten Gelenk angewendet werden. Die jeweilige Rotation θ wird durch die gewünschten oder gemessenen Gelenkwinkel bestimmt. Die Translationen werden entsprechend der offiziellen *NAO*-Dokumentation[Rob] angewendet. In Abbildung 3.2 wird der kinematische Baum und in Abbildung 3.3 die Position der unteren Kamera des *NAO*s dargestellt. Die Pose für den linken Fuß ergibt sich wie folgt:

$$\begin{aligned} T^O_{LF}(\theta) = & Trans_y(\frac{lbl}{2}) * Rot_x(\frac{\pi}{4}) * Rot_z(-\theta_{LeftHipYawPitch}) * Rot_x(\frac{-\pi}{4}) * \\ & Rot_x(-\theta_{LeftHipRoll}) * Rot_y(\theta_{LeftHipPitch}) * Trans_z(ull) * \\ & Rot_y(\theta_{LeftKneePitch}) * Trans_z(lll) * Rot_y(\theta_{LeftAnklePitch}) * \\ & Rot_x(-\theta_{LeftAnkleRoll}). \end{aligned} \tag{3.10}$$

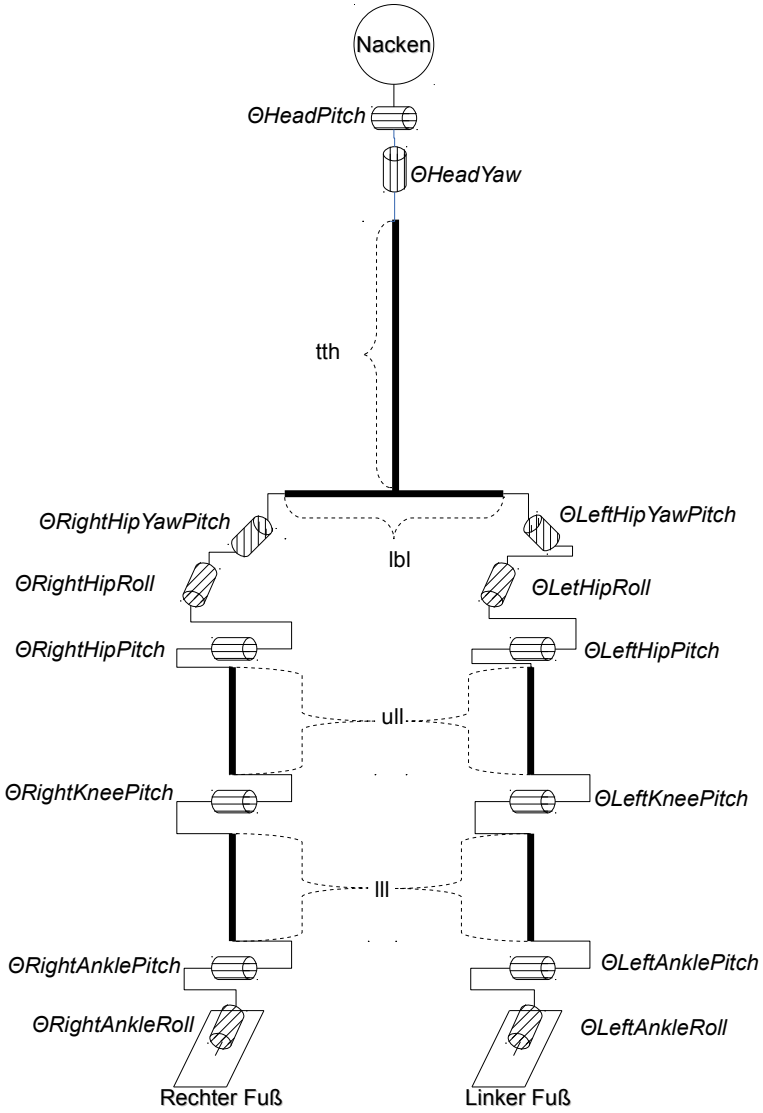


Abbildung 3.2: Der kinematische Baum des *NAO*s ohne Arme: tth = 211.5mm, lbl = 100mm, ull = 100mm und lll = 102.9mm.

Analog zur linken Fußpose wird die des rechten Fußes berechnet:

$$\begin{aligned} T_{RF}^{O}(\theta) =& Trans_y(-\frac{lbl}{2}) * Rot_x(\frac{-\pi}{4}) * Rot_z(\theta_{RightHipYawPitch}) * Rot_x(\frac{\pi}{4}) * \\ & Rot_x(\theta_{RightHipRoll}) * Rot_y(\theta_{RightHipPitch}) * Trans_z(ull) * \\ & Rot_y(\theta_{RightKneePitch}) * Trans_z(lll) * Rot_y(\theta_{RightAnklePitch}) * \\ & Rot_x(\theta_{RightAnkleRoll}). \end{aligned} \tag{3.11}$$

Die Rotation von $\frac{-\pi}{4}$ bzw. $\frac{\pi}{4}$ werden angewendet, da die ersten Hüftgelenke $\theta_{LeftHipYawPitch}$ und $\theta_{RightHipYawPitch}$ jeweils um 45° um die x-Achse verdreht montiert sind.

Die Pose des Nackens ergibt sich durch:

$$T_{N}^{O}(\theta) = Trans_z(tth) * Rot_z(\theta_{HeadYaw}) * Rot_Y(\theta_{HeadPitch}) \tag{3.12}$$

Die untere Kamera des *NAO*s ist im Inneren des Kopfes relativ zum Nacken montiert (siehe Abbildung 3.3):

$$T_{C}^{O}(\theta) = T_{N}^{O}(\theta) * Trans_x(nlcx) * Trans_z(nlcz) * Rot_y(\theta_{lct}) \tag{3.13}$$

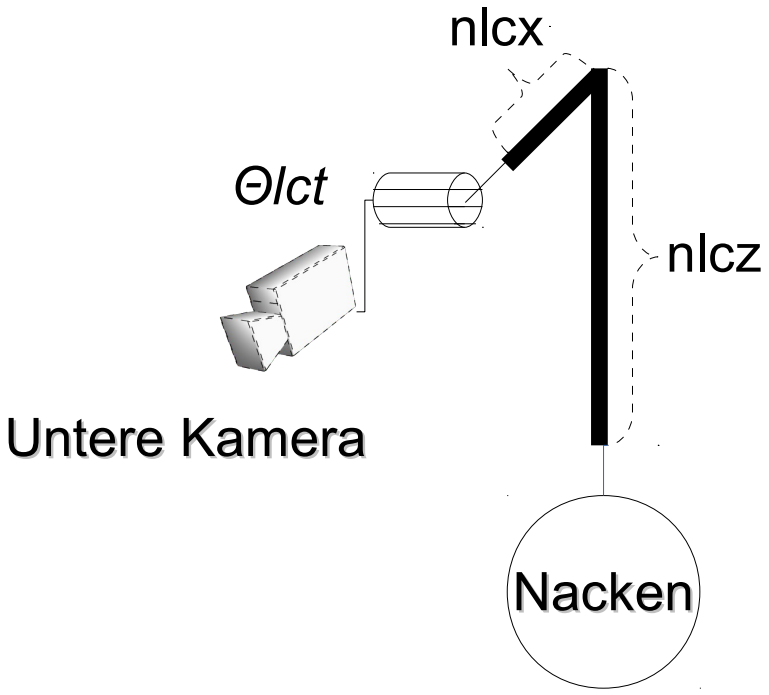


Abbildung 3.3: Die Pose der unteren Kamera relativ zum Nacken des *NAO*s: nlcx = 50.71mm, nlcz = 17.74mm, $\theta lct = 39.7°$

Für die Bezeichnung der Winkel werden dei Begriffe *yaw, pitch* und *roll* verwendet, welche ursprünglich aus der Luftfahrt stammen:

- yaw: Drehung um die z-Achse eines Bezugssystems.
- pitch: Drehung um die y-Achse eines Bezugssystems.
- roll: Drehung um die x-Achse eines Bezugssystems.

3.5 Lochkamera

Eine Lochkamera besteht aus einem geschlossenen Gehäuse. Durch eine sehr kleine Öffnung fallen Lichtstrahlen ein und treffen auf der auf der Rückwand des Gehäuses auf einen Film. Nach ausreichender Belichtungsdauer, ergibt sich ein auf den Kopf gestelltes Abbild der Szenerie vor der Lochkamera.

Diese sehr einfache Kamera wird in der digitalen Bildverarbeitung häufig als Modell für die Abbildung von 3D-Punkten auf ein zweidimensionales Bild verwendet. Da die

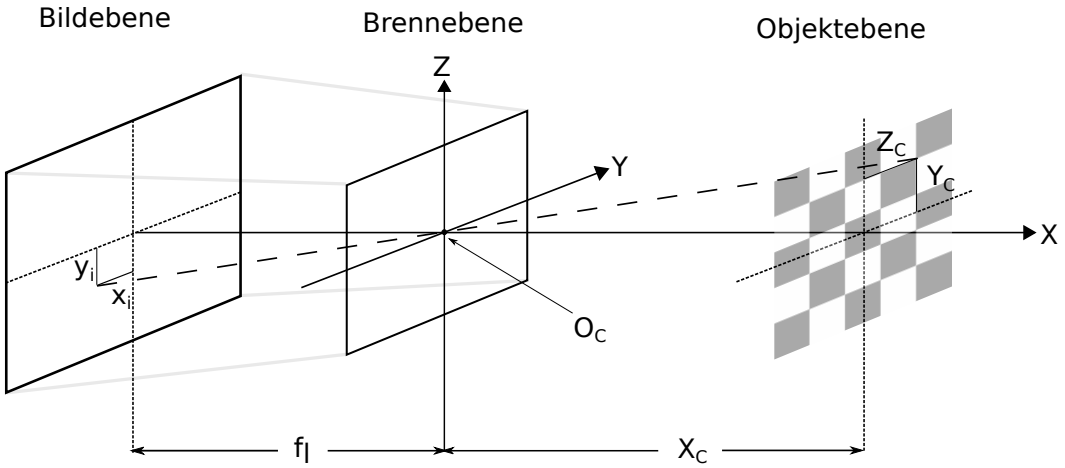


Abbildung 3.4: Die Lochkamera beinhaltet die Bild- und Brennebene. Die Ecke des Schachbrettes, welche parallel zur Brennebene liegt, hat die Koordinate $(X_C, Y_C, Z_C)^T$ relativ zur Roboterbasis. Die zugehörige Projektion auf die Bildebene hat die Bildkoordinate $(x_i, y_i)^T$.

Lichtstrahlen alle durch einen Punkt fallen, wird der Strahlensatz angewandt, um die Projektion eines Weltpunktes auf die Bildebene zu bestimmen.

$$\frac{O_C - (x_i, y_i)^T}{f_l} = \frac{(Y_C, Z_C)^T}{X_C} \tag{3.14}$$

Das optische Zentrum O_c legt fest, wo sich die Öffnungs auf der Brennebene befindet, durch welches die Strahlen fallen. Der Abstand zwischen der Brenn- und Bildebene wird mit der Brennweite f_l angegeben. Diese beiden Parameter sind abhängig von der verwendeten Kamera. $(X_C, Y_C, Z_C)^T$ ist ein dreidimensionaler Punkt relativ zur Kamera und $(x_i, y_i)^T$ ist die entsprechende Bildkoordinate. Die Gleichung 3.14 wird nach der Bildkoordinate aufgelöst:

$$(x_i, y_i)^T = O_C - (Y_C, Z_C)^T \frac{f_l}{X_C}. \tag{3.15}$$

Als Funktion von $\mathbb{R}^3 \to \mathbb{R}^2$ ergibt sich:

$$P(X_C, Y_C, Z_C) = O_C - (Y_C, Z_C)^T \frac{f_l}{X_C}. \tag{3.16}$$

Wichtig für die Korrektheit dieser Formeln ist, dass die Weltpunkte in Kamerakoordinaten vorliegen. Ist das nicht der Fall, müssen diese erst in das Kamerakoordinatensystem überführt werden. In dieser Arbeit sind alle Weltpunkte im Roboterkoordinatensystem angegeben. Für die Umrechnung der Koordinaten wird das Inverse der Kamerapose benötigt:

$$(X_C, Y_C, Z_C)^T = (T_C^O)^{-1} * (X_R, Y_R, Z_R)^T. \tag{3.17}$$

Mit $(X_R, Y_R, Z_R)^T$ ist ein Punkt relativ zur Roboterbasis.

3.6 Methode der kleinsten Quadrate

Gegeben einer Menge von n gemessenen Datenpunkten $((x_i, y_i),\ i = 1, \ldots, n)$, kann mit der Hilfe der *Methode der kleinsten Quadrate* eine Modellfunktion $(f(x, \alpha) = y)$ so parametriert werden, dass die Funktion die gemessenen Daten möglichst gut vorhersagt.

Hierfür werden die Parameter $(\alpha = \alpha_1, \ldots, \alpha_m)$ von $f(x, \alpha)$ so gewählt, dass die Summe (χ^2) der quadrierten *Residuen* (r_i) minimal wird, wobei zu beachten ist, dass mehr Beobachtungen als zu optimierende Parameter vorhanden sein müssen $(n \geq m)$.

$$\chi^2 = \sum_{i=1}^{n} r_i^2,\ r_i = y_i - f(x_i, \alpha) \tag{3.18}$$

Das *Residuum* beschreibt dabei die Differenz der gemessen Werte y für x und der Funktionswerte der Modellfunktion an der Stelle x. Das zu lösende Optimierungsproblem ist dann durch

$$\arg\min_{\alpha} \chi^2 \tag{3.19}$$

gegeben.

Ein Minimum einer Funktion befindet sich an der Stelle, an der die Steigung der Funktion gleich null ist, weswegen Formel 3.19 abgeleitet und gleich null gesetzt wird. Je nach Problemstellung wird zwischen *linearer* und *nichtlinearer Methode der kleinsten Quadrate* unterschieden.

Linear

Die Modellfunktion für *lineare kleinste Quadrate* ist eine *Linearkombination* aus den Parametern α und beliebigen, nicht von α abhängigen Funktionen ϕ mit N unabhängigen Variablen x.

$$f(x, \alpha) = \sum_{j=1}^{m} \alpha_j \phi_j(x_1, \dots x_N) \tag{3.20}$$

Diese Gleichungssysteme lassen sich als Verknüpfungen von Matrizen darstellen:

$$\arg\min_{\alpha} ||y - A\alpha||_2^2. \tag{3.21}$$

Die Matrix A stellt dabei die *Designmatrix* dar und beinhaltet pro Zeile die jeweiligen Terme der Modellfunktion in Abhängigkeit von x.

$$A = \begin{pmatrix} \phi_1(x_{1,1}, x_{1,2}, \dots, x_{1,N}) & \phi_2(x_{1,1}, x_{1,2}, \dots, x_{1,N}) & \dots & \phi_j(x_{1,1}, x_{1,2}, \dots, x_{1,N}) \\ \phi_1(x_{2,1}, x_{2,2}, \dots, x_{2,N}) & \phi_2(x_{2,1}, x_{2,2}, \dots, x_{2,N}) & \dots & \phi_j(x_{2,1}, x_{2,2}, \dots, x_{2,N}) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_1(x_{m,1}, x_{m,2}, \dots, x_{m,N}) & \phi_2(x_{m,1}, x_{m,2}, \dots, x_{m,N}) & \dots & \phi_j(x_{m,1}, x_{m,2}, \dots, x_{m,N}) \end{pmatrix} \tag{3.22}$$

Der Vektor α beinhaltet die gesuchten m Parameter und y die n Messungen.

$$\alpha = \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{pmatrix}, \; y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} \tag{3.23}$$

Die Ableitung von Gleichung 3.21 wird gleich null gesetzt, was die *Normalengleichung* ergibt:

$$A^T A\alpha - A^T y = 0. \tag{3.24}$$

Durch Anwendung von Matrixinversion lässt sich dieses Gleichungssystem lösen:

$$\alpha = (A^T A)^{-1} A^T y. \tag{3.25}$$

Nichtlinear

Gehen die Parameter *nichtlinear* in die Modellfunktion ein, also nicht wie in Gleichung 3.20 dargestellt, findet die *nichtlineare Methode der kleinsten Quadrate* Anwendung. Im Gegensatz zur *linearen* Variante, welche nach Lösung des Gleichungssystems von Gleichung 3.25 die optimale Belegung für α findet, sind im *nichtlinearen* Fall initiale Werte für die Parameter erforderlich, welche iterativ verbessert werden.

Die *Residuums-Funktion* r aus Formel 3.18 wird durch ein angenähertes, lineares Modell ersetzt, mit diesem dann wie bei der *linearen Methode der kleinsten Quadrate* verfahren werden kann. Eine Funktion kann mit Hilfe einer *Taylorreihe* erster Ordnung linear angenähert werden:

$$f(x) = f(a) + f^{'}(a)(x - a). \tag{3.26}$$

Die *Residuums-Funktion* $\hat{r}$ sieht dann für n Messungen wie folgt aus:

$$\hat{r_i} = r_i^k + J_r(\alpha^k)\alpha_\delta; \quad r_i^k = y_i - f(x_i, \alpha^k); \; i = 1, \ldots, n. \tag{3.27}$$

Bei α^k handelt es sich um die Wertebelegung für die Parameter α der $k-ten$ Iteration; α_δ gibt die Schrittweite an, mit dieser sich in Richtung des negativen Gradienten bewegt wird. Der Gradient wird dabei durch $J_r(\alpha_k)$ definiert, die sogenannte *Jacobi-Matrix*, welche die ersten partiellen Ableitungen der Residuumsfunktion r aus Formel 3.19 in Abhängigkeit der Parameter α^k beinhaltet.

$$J_r(\alpha) = \begin{pmatrix} \frac{\partial r_1}{\partial \alpha_1} & \frac{\partial r_1}{\partial \alpha_2} & \cdots & \frac{\partial r_1}{\partial \alpha_m} \\ \frac{\partial r_2}{\partial \alpha_1} & \frac{\partial r_2}{\partial \alpha_2} & \cdots & \frac{\partial r_2}{\partial \alpha_m} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial r_n}{\partial \alpha_1} & \frac{\partial r_n}{\partial \alpha_2} & \cdots & \frac{\partial r_n}{\partial \alpha_m} \end{pmatrix} \tag{3.28}$$

In jeder Iteration wird die Schrittweite α_δ angepasst, welche mit der Startbelegung der Parameter verrechnet wird:

$$\alpha^{k+1} = \alpha^k - \alpha_\delta. \tag{3.29}$$

Das Optimierungsproblem ist dann durch:

$$\arg\min_{\alpha_\delta} \; = ||\hat{r}||_2^2 \tag{3.30}$$

gegeben. Die schrittweise Annäherung an ein Minimum erfolgt wie bei der *linearen Methode*, indem die *Residuenquadratsumme* abgeleitet und gleich null gesetzt wird ($J_r = J_r(\alpha^k)$):

$$(J_r^T J_r)\alpha_\delta - J_r^T r^k = 0. \tag{3.31}$$

Gesucht wird das α_δ, welches die Summe der quadrierten, Resiuden schrittweise mini-

miert:

$$\alpha_\delta = (J_r^T J_r)^{-1} J_r^T r^k. \tag{3.32}$$

Da es sich um ein iteratives Verfahren handelt, müssen Abbruchkriterien definiert werden. Ein einfaches Kriterium stellt eine maximale Anzahl von Iterationen dar oder die Überprüfung, ob sich die Summe der quadrierten Residuen nur noch wenig verändert.

4 Automatische Roboterkalibrierung für den *NAO*

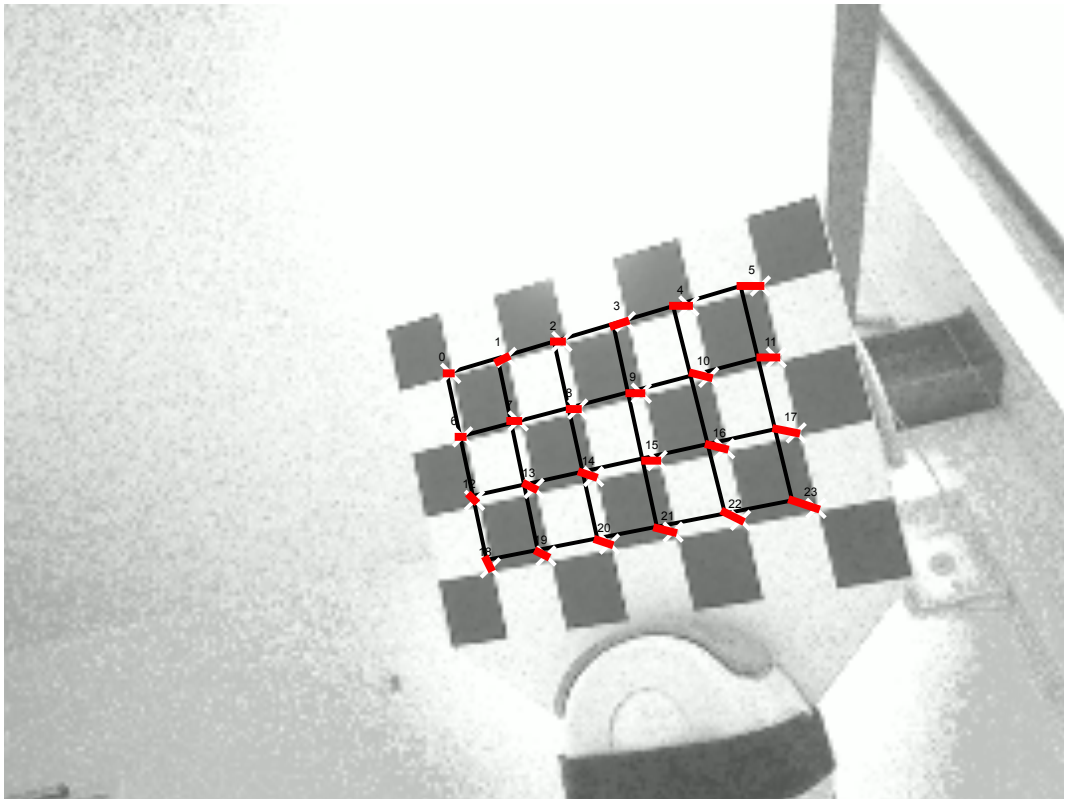


Abbildung 4.1: Die Residuen (rote Linien) der im Bild erkannten Ecken (weiße Kreuze) und den zugehörigen projizierten Ecken durch das Projektionsmodell *proj*. Ziel dieser Arbeit ist, dass die Projektion des Schachbrettes unter möglichst allen Gelenktstellungen Deckungsgleich ist mit dem im Bild zu sehenden Schachbrett.

Dieses Kapitel widmet sich der Umsetzung der automatischen Roboterkalibrierung für den *NAO*. Folgende Probleme müssen gelöst werden, um das in Kapitel 1.4 definierte Ziel zu erreichen:

- Es muss ein Modell des Roboters erstellt werden, welches die wichtigen Parameter der Roboterkalibrierung berücksichtigt.
- Es muss ein geeigneter Kalibrierkörper gefunden werden.
- Der Kalibrierkörper muss mit Methoden der digitalen Bildverarbeitung messbar

sein.

- Es bedarf einer Lösung für die Ansteuerung von verschiedenen Beinstellungen.
- Die Roboterkalibrierung muss als ein Problem der nichtlinearen kleinsten Quadrate definiert werden.
- Es muss ein geeignetes Optimierungsverfahren gefunden und implementiert werden.
- Es ist notwendig, dass der Ablauf der gesamten Kalibrierung durch eine zentrale Softwarekomponente gesteuert wird.

Die Lösung dieser Probleme werden in den folgenden Abschnitten dargestellt. Zuerst wird die Beschaffenheit und Arretierung des Kalibrierkörpers behandelt, gefolgt von der Wahl eines geeigneten Mess- und Vorhersagemodells und der Identifizierung der zu optimierenden Parameter. Als nächstes wird der Ablauf einer Kalibrierung skizziert, beginnend mit der Konfiguration und Vorbereitung des *NAO*s durch einen Benutzer und der darauf folgenden automatischen Steuerung der Kalibrierung. Die weiteren Kapitel behandeln die Bildverarbeitung, die Bewegungssteuerung und den *Levenberg-Marquardt*-Algorithmus.

ex

4.1 Kalibrierkörper

Bei einem *Kalibrierkörper* handelt es sich um einen Gegenstand, der nicht Teil des Roboters ist und dessen Position über eindeutige Merkmale messbar ist. In dieser Arbeit werden Messungen mit einer Kamera vorgenommen, d. h. es werden Bilder vom Kalibrierkörper aufgenommen, wobei die besagten eindeutigen Merkmale mit Methoden der digitalen Bildverarbeitung gut lokalisierbar sein müssen. Von besonderer Wichtigkeit für das gesamte Verfahren ist, dass die Positionen und Dimensionen dieser Merkmale auf dem Körper *a-priori* bekannt sind. Da es sich um einen externen Gegenstand handelt, muss dieser möglichst einfach und wiederholbar präzise am Roboter angebracht und natürlich auch wieder entfernt werden können.

All diese Bedingungen müssen bei der Wahl und Konstruktion eines geeigneten Körpers bedacht werden.

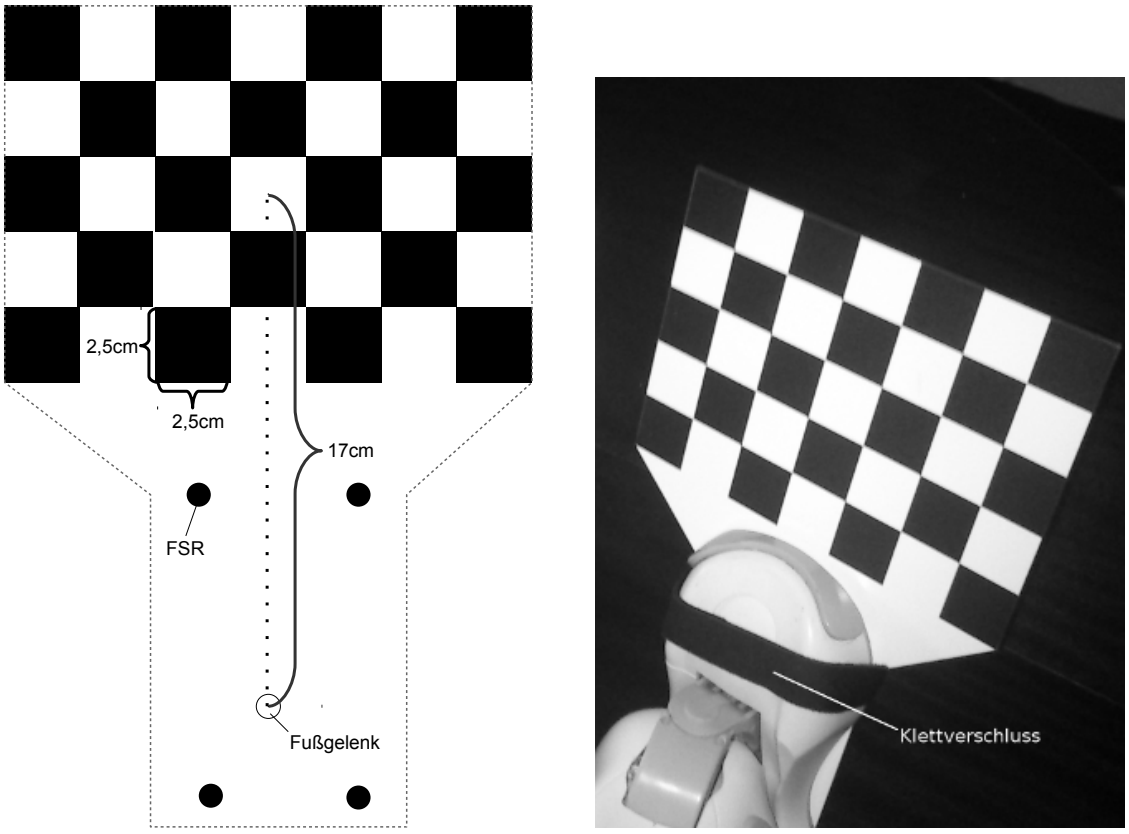
Für die Kalibrierung der extrinsischen und intrinsischen Parameter der Kamera wird von Zhang[ZZ98] ein Schachbrett verwendet. Dieses eignet sich besonders gut für die Bestimmung der intrinsischen Parameter wie der Verzerrungskoeffizient, die Skalierung,

die Brennweite und das optische Zentrum, da das Brett eine planare Ebene bildet. Die markanten Merkmale eines Schachbretts sind die Berührungspunkte der schwarzen und weißen Kacheln. Die Dimensionen der Kacheln sind im Voraus bekannt oder können nachgemessen werden.

Auch in den Veröffentlichungen von Pradeep u.a.[PKB10] und Hubert u.a.[HSB12] werden Schachbretter verwendet um zusätzlich zur Kamerakalibrierung auch die Nullagenfehler der Gelenke berechnen zu können.

Da sich Schachbretter für Kalibrierungsaufgaben bewiesen haben und viele Verfahren für das Erkennen von Schachbrettern existieren, wird für die automatische Roboterkalibrierung des *NAO*s auch ein Schachbrett als Kalibrierkörper verwendet.

In Abbildung 4.2 wird der in dieser Arbeit verwendete Kalibrierkörper, im weiteren Verlauf als *Schachbrettsandale* bezeichnet, dargestellt.



(a) (b)

Abbildung 4.2: Die *Schachbrettsandale* für den rechten Fuß des *NAO*s. (b) Eine arretierte Sandale.

Es handelt sich um ein Schachbrett aus Aluminium-Dibond mit sieben Kacheln pro Zeile und fünf pro Spalte. Eine Kachel hat eine Breite und Länge von $2,5\ cm$. Ins-

gesamt gibt es 24 Berührungspunkte von gleichfarbigen Schachbrettfeldern. Positive Eigenschaften von Aluminium-Dibond stellen die Steifheit, das geringe Gewicht und die Kosten des Materials dar. Außerdem gibt es eine Vielfalt an Herstellern, die sich um den Druck und den Zuschnitt kümmern. Die schwarz gefüllten Kreise stellen die Positionen der Drucksensoren dar, deren Lagen relativ zur Projektion des letzten Fußgelenks auf die Sohlen-Ebene (weiß gefüllter Kreis) bekannt sind, wobei der Abstand der Projektion zur Mitte des Schachbrettes 17 cm beträgt und alle relativen Positionen der Schachbrettecken mit diesen Informationen berechenbar sind.

Da die Sandale an der Fußsohle des *NAO*s angebracht wird, liegen alle Ecken immer auf der Sohlen-Ebene. Ein wichtiges Ziel der Kalibrierung ist, dass die beiden Fußsohlen auf gleicher Höhe zur Roboterbasis liegen und die Sohlen gerade auf dem Boden stehen.

Arretierung

Um die Sandalen an den Sohlen befestigen zu können, werden die Aussparungen an den Drucksensoren ausgenutzt. Diese haben einen Durchmesser von 8 mm und eine ungefähre Höhe von ca. 4 mm. In Abbildung 4.3 sind die Positionen der Sensoren relativ zum Fußgelenk angegeben. An diesen Stellen befinden sich auf der Sandale entsprechend 8 mm breite und 3 mm hohe runde Stifte, die in die Aussparungen passen. Diese Stifte dienen zur korrekten Positionierung der Sandale. Allerdings bieten die Stifte wenig Halt, weswegen noch Klettverschlussbänder zur finalen Fixierung verwendet werden (siehe 4.2 (b)).

4.2 Defintion des Optimierungsproblems

Gemäß der Definition der Roboterkalibrierung aus Abschnitt 3.1 wird die automatische Roboterkalibrierung für den *NAO* ebenfalls in die vier Phasen Modellierung, Messung, Identifikation und Kompensation aufgeteilt:

Modellierung

Die maßgeblichen Komponenten des Robotermodells stellen die in Abschnitt 3.4 vorgestellten Vorwärtskinematiken der Beine und der unteren Kamera, sowie das Lochkameramodell aus Kapitel 3.5 dar. Die Transformationen für den linken und rechten Fuß müssen noch um die Höhe des letzten Fußgelenks zur Sohlenfläche verschoben werden.

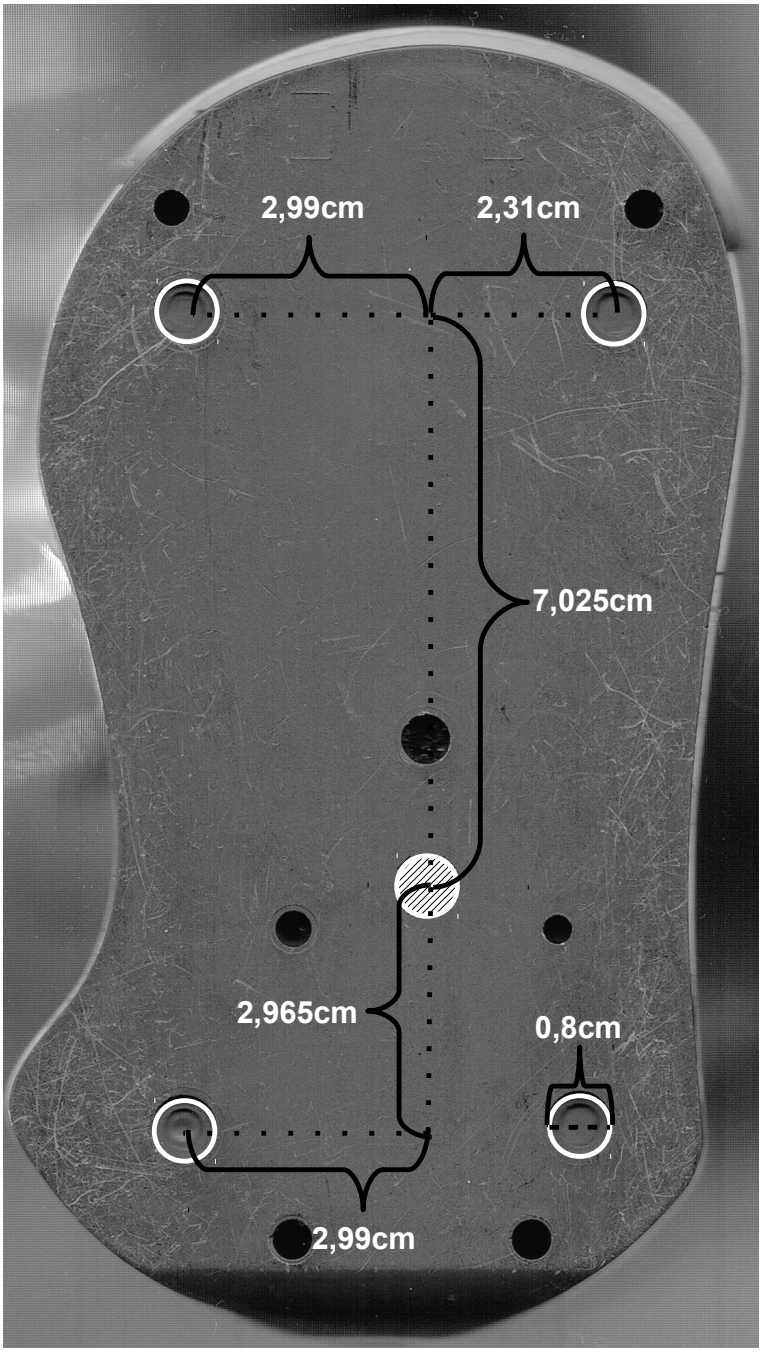


Abbildung 4.3: Die Positionen der Druckensoren (weiße Kreise) relativ zur Projektion des letzten Fußgelenk auf die Fußfläche (weißer ausgefüllter Kreis), sowie der Durchmesser der Aussparungen der Sensoren.

Es folgt die Verschiebung für den linken Fuß (für den rechten analog):

$$T^O_{LS}(\theta) = T^O_{LF}(\theta) * Trans_z(-hlj). \tag{4.1}$$

Die Höhe hlj beträgt 45.19 mm. Nach der Verschiebung beschreibt die Transformation $T^O_{LS}(\theta)$ die Transformation vom Roboterursprung zur linken und $T^O_{RS}(\theta)$ zur rechten Fußsohle.

Durch die Verknüpfung der Modelle der Kinematik und der Kamera lässt sich mit einem kompletten Satz von Gelenkwinkeln ein Punkt des Schachbrettes relativ zum linken Fuß in das Bild der Kamera projizieren:

$$proj(p_{LS}, \theta) = P(T^O_C(q)^{-1} * T^O_{LS}(q) * p_{LS}). \tag{4.2}$$

$$\begin{aligned} q_i = \theta_i,\ i \in \Big\{ & LeftHipYawPitch,\ LeftHipRoll,\ LeftHipPitch, \\ & LeftKneePitch,\ LeftAnklePitch,\ LeftAnkleRoll, \\ & RightHipYawPitch,\ RightHipRoll,\ RightHipPitch, \\ & RightKneePitch,\ RightAnklePitch,\ RightAnkleRoll \Big\} \end{aligned} \tag{4.3}$$

Der Punkt p_{LS} ist die Koordinate einer Ecke des Schachbretts relativ zur Fußsohle. Diese wird mit $T^O_{LS}(q)$ in Roboterkoordinaten und mit $T^O_C(q)^{-1}$ in Kamerakoordinaten umgewandelt. Die gewünschten oder gemessenen Gelenkwinkel werden durch q beschrieben. Letzten Endes wird durch die Anwendung der Projektionsfunktion P die 2D-Koordinate im Kamerabild berechnet.

Messung

Die Position des am Fuß befestigten Schachbrettes wird mit einer Kamera gemessen. Das Ergebnis einer solchen Messung stellen die Bildkoordinaten aller erkannten Ecken des Schachbrettes dar. Genauer handelt es sich bei einer Messung um ein Tripel $(p_{img}, \theta, p_{cb})$, bestehend aus genau einer im Bild erkannten Ecke p_{img}, den zu diesem Zeitpunkt gemessenen Gelenkwinkeln θ und der Koordinate der gesehenen Ecke relativ zur Fußpose p_{cb}. All diese Messungen werden der Menge M hinzugefügt.

$$M = \Big\{ (p_{img_0}, \theta_0, p_{cb_0}), (p_{img_1}, \theta_1, p_{cb_1}), \ldots, (p_{img_n}, \theta_n, p_{cb_n}) \Big\} \tag{4.4}$$

Identifikation

Das durch Formel 4.2 beschriebene Modell des *NAO*s ist unvollständig, da die Nullagenfehler der Gelenke, eine falsch montierte Kamera, sowie Fehler bei der Arretierung der Schachbrettsandale nicht berücksichtigt werden. Die Belegung dieser Parameter gilt es in der *Identifikations*-Phase zu finden. Eine Auflistung aller identifizierten Parameter befindet sich in Tabelle 4.1.

Parameter	**Typ**	**Parameter**	**Typ**	**Parameter**	**Typ**
$\alpha_{CameraYaw}$	Δ°	$\alpha_{CameraPitch}$	Δ°	$\alpha_{CameraRoll}$	Δ°
$\alpha_{LeftHipYawPitch}$	Δ°	$\alpha_{LeftHipRoll}$	Δ°	$\alpha_{LeftHipPitch}$	Δ°
$\alpha_{LeftKneePitch}$	Δ°	$\alpha_{LeftAnklePitch}$	Δ°	$\alpha_{LeftAnkleRoll}$	Δ°
$\alpha_{RightHipYawPitch}$	Δ°	$\alpha_{RightHipRoll}$	Δ°	$\alpha_{RightHipPitch}$	Δ°
$\alpha_{RightKneePitch}$	Δ°	$\alpha_{RightAnklePitch}$	Δ°	$\alpha_{RightAnkleRoll}$	Δ°
$\alpha_{LeftRotationZ}$	Δ°	$\alpha_{LeftTranslationX}$	Δmm	$\alpha_{LeftTranslationY}$	Δmm
$\alpha_{RightRotationZ}$	Δ°	$\alpha_{RightTranslationX}$	Δmm	$\alpha_{RightTranslationY}$	Δmm

Tabelle 4.1: Die Spalte **Typ** gibt an um was für eine Art Fehler es sich beim gegebenen **Parameter** handelt. Bei Δ° handelt es sich um einen Gelenkwinkelversatz und bei Δmm um einen Translationsversatz.

Die Kameras des *NAO*s sind mit Heißkleber in einem speziellen Rahmen befestigt. Es wird angenommen, dass die Position relativ zum Nackengelenk konstant ist, weshalb keine Korrekturparameter für die Translation der Kamera berücksichtigt werden. Allerdings hat sich herausgestellt, dass die Rotation der Kamera von *NAO* zu *NAO* unterschiedlich ist. Diese Rotationsfehler werden mit den drei Parametern $\alpha_{CameraYaw}$, $\alpha_{CameraPitch}$ und $\alpha_{CameraRoll}$ ausgeglichen.

Die Gelenkwinkelversätze der Beine und Füße werden durch die Parameter $\alpha_{LeftHipYawPitch}$ bis $\alpha_{RAnkleRoll}$ definiert, also insgesamt zwölf Stellgrößen.

Es wird angenommen, dass die Arretierung des Schachbretts einen Translationsfehler in x- und y-Richtung, sowie einen Rotationsfehler um die z-Achse erzeugt. Dies ergibt für beide Füße insgesamt weitere sechs Parameter.

Insgesamt wurden **21** Parameter identifiziert, die zusätzlich vom Projektionsmodell berücksichtigt werden müssen. Sieht man von den Korrekturparametern für die Schachbrettsandale ab, werden die gleichen Parameter benötigt, wie auch bei der manuellen Kalibrierung.

Zuerst wird die Parameterliste der Projektionsfunktion *proj* um den Parametervektor

α erweitert (hier für das linke Bein):

$$proj(p_{LS}, \theta, \alpha) = P(T_C^O(q, \alpha)^{-1} * T_{LF}^O(q) * p_{LS\alpha}). \tag{4.5}$$

Die drei Kamerakorrektur-Parameter werden auf $T_C^O(q)$ angewandt:

$$T_C^O(q, \alpha) = T_C^O(q) * Rot_y(\alpha_{CameraPitch}) * Rot_x(\alpha_{CameraRoll}) * Rot_z(\alpha_{CameraYaw}). \tag{4.6}$$

Die einzelnen Winkelwerte der Gelenke der Beine werden um den jeweiligen Nullagenfehler erweitert:

$$q_i = \theta_i + \alpha_i, \ i \text{ aus Gleichung 4.3.} \tag{4.7}$$

Die Positionen der Schachbrettecken p_{LS} werden entsprechend der Parameter für die Korrektur der Schachbrettsandalen angepasst:

$$p_{LS\alpha} = Rot_z(\alpha_{LPatternRotZ}) * Trans_x(\alpha_{LPatternX}) * Trans_y(\alpha_{LPatternY}) * p_{LS}. \tag{4.8}$$

Nun ist das Modell des *NAO*s komplett und kann zur Lösung eines Optimierungsproblems verwendet werden. Wie in Abschnitt 3.6 definiert, handelt es sich um ein nichtlineares Optimierungsproblem, da die genannten Parameter nichtlinear in die Projektionsfunktion eingehen.

Wie von Birbach u.a.[OF12] vorgeschlagen wird die Summe über die quadrierten Abstände zwischen der im Bild lokalisierten Schachbrettecken und den Vorhersagen des Projektionsmodell (siehe Abbildung 4.1) minimiert.

$$R = \left\{ r \middle| m \in M \wedge r = |m[p_{img}] - proj(m[p_{cb}], m[\theta], \alpha)| \right\} \tag{4.9}$$

Die Menge R beinhaltet die Abstände zwischen den Messungen und den zugehörigen Projektionen. Der Zugriff auf die einzelnen Elemente des Tripels $m \in M$ wird mit $m[p_{img}]$ (Bildkoordinate einer Ecke), $m[p_{cb}]$ (3D-Koordinate der Ecke relativ zum Fußgelenk und $m[\theta]$ (die gemessenen Gelenkwinkel) formalisiert. Die 3D-Koordinaten werden anhand der Dimensionen der Schachbrettsandale berechnet (siehe Abschnitt 4.1).

Das zu lösende nichtlineare Optimierungsproblem ist dann durch

$$\arg\min_{\alpha} \sum_{r \in R} r^2 \tag{4.10}$$

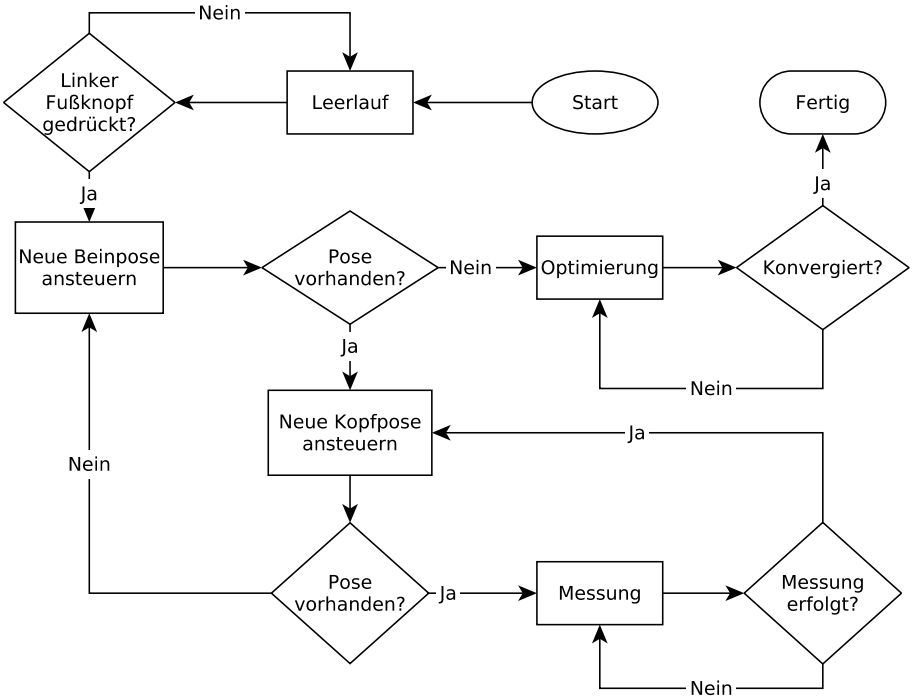
gegeben, wobei die initiale Belegung für jeden Parameter des Vektors α auf null festgelegt ist.

Kompensation

Die Parameter der Kamerakorrektur werden bei der Berechnung der *RobotCameraMatrix*[Röf+13b] innerhalb des *Moduls CameraMatrixProvider* angewandt. Die *RobotCameraMatrix* stellt die Transformation $T_C^O(q, \alpha)$ aus Gleichung 4.6 dar.

Die Parameter für die Nullagenfehler der Gelenke werden auf die zu setzenden Winkel addiert und von den gelesenen abgezogen. Dies geschieht innerhalb des Moduls *NaoProvider*, welches Zugriff auf die Sensorwerte und die Gelenke des *NAO*s hat.

4.3 Steuerung der Kalibrierung



(a)

Abbildung 4.4: Ein Kalibrierungsdurchlauf als Flussdiagramm dargestellt.

Der automatische Roboterkalibrierung lässt sich in zwei Phasen aufteilen:

1. Datensammlung
2. Optimierung

Während der Datensammlung werden, während der *NAO* auf dem Rücken liegt, verschiedene Beinstellungen angefahren, wobei die Schachbrettsandale für jede Beinkonfi-

guration unter drei verschiedenen Kopfstellungen betrachtet wird. Die jeweiligen gemessenen Gelenkwinkel und die Koordinaten der Ecken des Schachbrettes im Kamerabild werden zusammen in einer Liste gespeichert. Eine Messung erfolgt, wenn die jeweilige Kopfstellung erreicht wurde und der Kopf sich nicht mehr bewegt. Insgesamt hat die Schachbretterkennung maximal vier Sekunden Zeit ein Schachbrett zu detektieren. Diese Prozedur wird für das linke und rechte Bein jeweils durchgeführt, so dass am Ende, wenn es keine weiteren Gelenkkonfigurationen mehr gibt, die gesammelten Daten an einen Optimierer übergeben werden, der als Ergebnis möglichst gute Werte für die in Tabelle 4.1 dargestellten Parameter findet.

Die Steuerung der gesamten Prozedur, das *Modul Calibrator*, ist als endlicher Automat realisiert. In Abbildung 4.4 wird das Verfahren als Flussdiagramm und in Abbildung 4.4 als Fotostrecke exemplarisch visualisiert. Auf der Fotostrecke werden verschiedenen Beinstellungen dargestellt. Das Schachbrett wird so aus vielen verschiedenen Perspektiven und Distanzen wahrgenommen.

Auswahl der Beinstellungen

Die Art und Anzahl an Gelenkkonfigurationen ist ein entscheidender Faktor der Roboterkalibrierung. Eine große Anzahl an Gelenkstellungen verspricht eine hohe Genauigkeit der resultierenden Korrektur-Parameter. Allerdings skaliert die Dauer der Kalibrierung mit der Anzahl der Messungen, weswegen die Anzahl von Beinstellungen ein Kompromiss aus Genauigkeit und Zeitaufwand darstellt.

Insgesamt gibt es 24 verschiedene Gelenkkonfigurationen pro Bein. Die Auswahl berücksichtigt, dass das Schachbrett in verschiedenen Distanzen zu Kamera und unter unterschiedlichen Perspektiven beobachtet wird. Außerdem wird jedes Gelenk mindestens einmal um einen großen Winkel gedreht, damit jedes Gelenk seinen Einfluss auf die Position des Schachbrettes relativ zur Roboterbasis betont. Ein weiterer wichtiger Faktor ist, dass das Schachbrett nicht durch Körperteile des *NAO*s verdeckt wird oder so weit rotiert ist, dass die Kacheln nicht mehr sichtbar sind.

Auswahl der Kopfstellungen

Um den Einfluss der Kamera und dessen eventuelle Fehlstellung zu berücksichtigen, wird die Schachbrettsandale pro Beinstellung aus drei verschiedenen Kopfgstellungen observiert:

1. Blick auf die Mitte des Schachbrettes.

2. Blick auf einen Punkt drei Zentimeter links und vier Zentimeter unterhalb der Mitte des Schachbrettes.

3. Blick auf einen Punkt drei Zentimeter rechts und vier Zentimeter unterhalb der Mitte des Schachbrettes.

Die unterschiedlichen Kopfstellungen sollen dafür sorgen, dass der Einfluss der Kamera auf den Projektionsfehler isoliert zur Geltung kommt.

4.4 Kalibrierungsdurchlauf

In den folgenden Zeilen wird erklärt, wie die automatische Kalibrierung für den *NAO* konfiguriert und eingesetzt wird.

Konfiguration

Die automatische Roboterkalibrierung ist nicht Bestandteil der Software von *B-Human*, welche während eines Fußballspiels auf dem *NAO* läuft, da die im Rahmen dieser Arbeit entwickelten *Module* exklusiv für die Kalibrierung gebraucht werden und in einem echten Spiel unter Umständen wertvolle Rechenzeit verschwenden. Außerdem muss laut Röfer u.a[Röf+13a] eine Kalibrierung immer rechtzeitig vor dem Beginn eines Spiels erfolgen, weswegen die automatische Kalibrierung innerhalb einer eigenen *Location* existiert.

Eine *Location* beinhaltet Konfigurationsdateien, welche für ein spezielles Szenario angepasst wurden. Die *Location* dieser Arbeit beinhaltet eine Liste von *Modulen* (siehe Abbildung 4.5), welche ausgeführt werden müssen, damit die automatische Roboterkalibrierung korrekt funktioniert. Unnötige *Module* von *B-Human* wie die Verhaltenssteuerung, die Erkennung von Bällen, Toren und Robotern sind dabei deaktiviert.

Handhabung

Die folgende Auflistung stellt alle Schritte dar, die ein Benutzer tätigen muss, um die Kalibrierung durchzuführen.

1. Installation der Kalibrierungs-Software bzw. *Location* auf dem *NAO*.

2. Arretierung der beiden Schachbrettsandalen am linken und rechten Fuß.

3. Initialisierung der Kalibrierung durch Betätigung des linken Fuß-Knopfes.

4. Warten auf Terminierung der gesamten Prozedur.

5. Überprüfung der optimierten Parameter.

Die Interaktion von Benutzern ist also auf ein Minimum beschränkt, so dass während der Laufzeit des Verfahrens an anderen wichtigeren Aspekten im Kontext eines Roboterfußball-Turniers gearbeitet werden kann.

Die optimierten Parameter müssen vom Benutzer auf Plausibilität überprüft werden. Hierfür werden die Gelenkstellungen der Kalibrierung nochmal angefahren. Die Projektion des Schachbrettes sollte nun an der richtigen Stelle im Bild auftauchen. Des Weiteren muss der Benutzer die Schachbrettsandalen entfernen und den Roboter in einen aufrechten Stand überführen. Die Füße müssen nun auf einer Höhe liegen und die Fußsohlen gerade auf dem Boden aufliegen.

Um die Kameraparameter zu überprüfen, empfiehlt es sich den *NAO* auf eine feste Position auf dem Spielfeld zu stellen und die Feldlinien ins Kamerabild zu projizieren. Die projizierten Linien müssen auf den tatsächlichen Linien im Bild liegen.

4.5 Architektur

Die Abbildung 4.6 zeigt auf, welche Komponenten für die Umsetzung dieser Arbeit entwickelt werden mussten. Die Darstellung ist dabei nicht vollständig, sie zeigt lediglich die wichtigsten Komponenten auf, was der Übersichtlichkeit geschuldet ist, da z. B. im Prozess *Motion* mehrere Module an der Bereitstellung von Gelenkwinkeln beteiligt sind.

Das gesamte Verfahren wurde hierbei in verschiedene *Module* aufgeteilt, wobei jedes eine bestimmte Teilaufgabe löst, wie z. B. das Erkennen von Schachbrettern, die Ausführung von verschiedenen Bewegungen oder die Steuerung des Zusammenspiels aller Module untereinander.

Es folgt eine Auflistung aller entwickelten Softwarekomponenten:

- Der *Calibrator* ist das Herzstück dieser Arbeit, da dieser die Koordination aller anderen wichtigen *Module* steuert und alle bereitgestellten *Repräsentationen* dieser verwendet. Die wichtigsten Daten sind dabei das erkannte Schachbrett (*Chessboard*) und die Winkel der Gelenke des *NAO*s (*FilteredJointData*).

  Um Rechenzeit zu sparen, wird die teure Schachbretterkennung nur aktiviert,

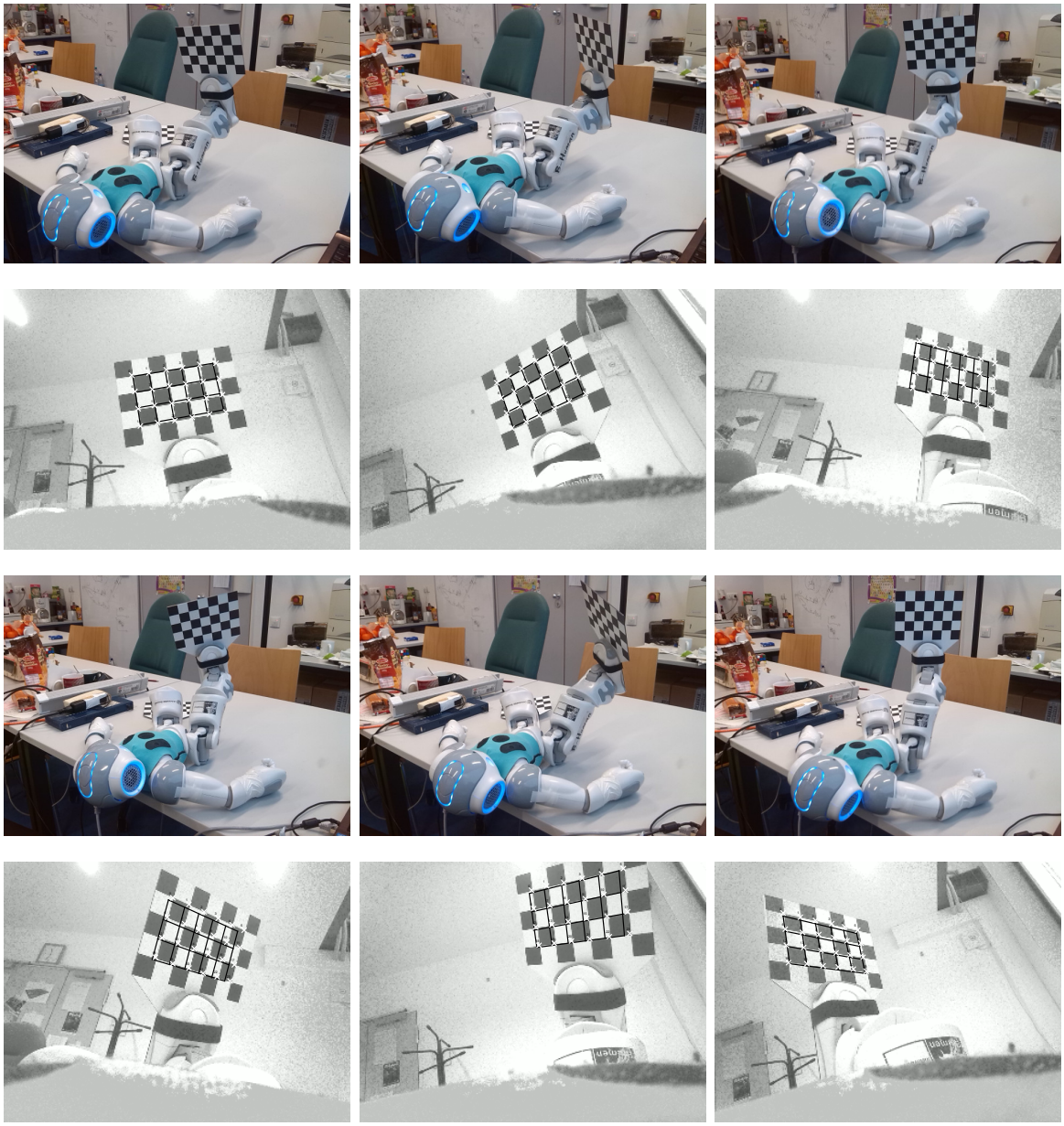


Abbildung 4.5: Eine Fotostrecke, aufgenommen während eines Experiments. Zu einem Bild des *NAO*s gehört das darunter befindliche Grauwert-Bild der unteren Kamera mit der Projektion des Schachbrettes (schwarze Linien) und den detektierten Schachbrettecken (weiße Kreuze).

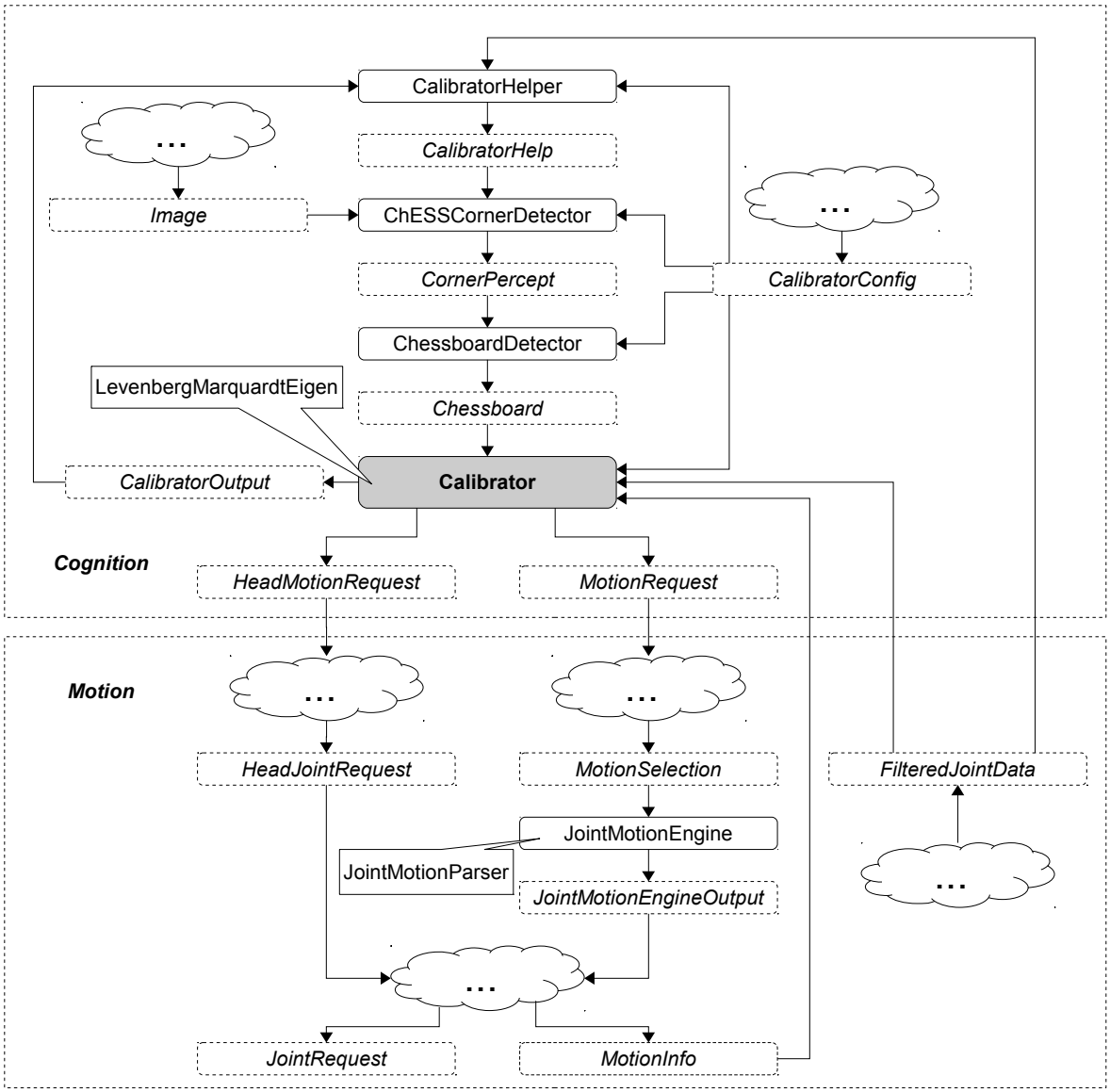


Abbildung 4.6: Die Architektur der automatischen Gelenkkalibrierung als Abhängigkeitsgraph dargestellt. Die bereitgestellten *Repräsentationen* sind in gestrichelten Kästen in kursiver Schrift dargestellt und die *Module* in normaler Schrift innerhalb der durchgängigen Kästen. Die Wolken beinhalten weitere Abhängigkeiten auf dessen Darstellung aus Übersichtlichkeit verzichtet wird. Die *Module* im oberen Kasten laufen innerhalb von *Cognition* und die im unteren Kasten im Kontext von *Motion*.

wenn zuvor eine bestimmte Gelenkstellung des Roboters erreicht wurde. Die bereitgestellte *Repräsentation CalibratorOutput* beinhaltet dabei, neben dem Schalter zur Aktivierung der Schachbretterkennung, die Kalbirierung für die Gelenke und die der Kamera. Der *MotionRequest* wird zur Anforderung von Bewegungen verwendet und der *HeadMotionRequest* bestimmt die Orientierung des Kopfes. Aus diesen beiden Informationen wird schließlich der *JointRequest* erzeugt, der über die zu setzenden Gelenkwinkel verfügt. Informationen zur aktuell ausgeführten Bewegung befinden sich in der *Repräsenation MotionInfo*.

- Im *CalibratorHelper* sind Methoden des *Calibrators* ausgelagert worden, wie z. B. die Berechnung des Bildausschnitts für die *Schachbretterkennung*. Darüberhinaus stellt das *Modul* die Möglichkeit bereit, unter Berücksichtigung einer festgelegten Position auf dem Spielfeld, die Feldlinien in das Kamerabild zu projizieren.

- Die *JointMotionEngine* führt durch den Benutzer definierte Bewegungen aus, indem zwischen festgelegten Gelenkwinkelsätzen interpoliert wird (Siehe Abschnitt 4.7).

- Bewegungen werden in einer Konfigurationsdatei anhand einer einfachen Syntax definiert. Diese Datei wird dann vom *JointMotionParser* eingelesen und in relevante Datenstrukturen der *JointMotionEngine* übersetzt. Der *JointMotionParser* ist Bestandteil der *JointMotionEngine* und kein eigenständiges *Modul*.

- Der erste Schritt der *Schachbretterkennung* erledigt der *ChESSCornerDetector*. Das *Modul* sucht nach Berührungspunkten gleichfarbiger Schachbrettfelder im Kamerabild und trägt diese in die *Repräsenation CornerPercept* ein.

- Der *ChessboardDetector* verwendet die detektierten Berührungspunkte des *CornerPercepts*, um ein Schachbrett zu extrahieren, so dass die interne Lage der Punkte auf dem Brett festgestellt wird.

- *LevenbergMarquardtEigen* ist eine Implementierung des *Levenberg-Marquardt*-Algorithmus, der auf der *Methode der nichtlinearen kleinsten Quadrate* basiert (siehe 4.8). Der Optimierer ist Bestandteil des *Calibrators*.

Die *Repräsentation CalibratorConfig* beinhaltet sämtliche Parameter der beteiligten *Module* der Kalibrierung. Dies ermöglicht einen sofortigen Überblick über alle Einstellungen einer Kalibrierung und es kann besser nachvollzogen werden, welche Parameter zum Zeitpunkt einer Durchführung eingestellt waren.

4.6 Schachbretterkennung

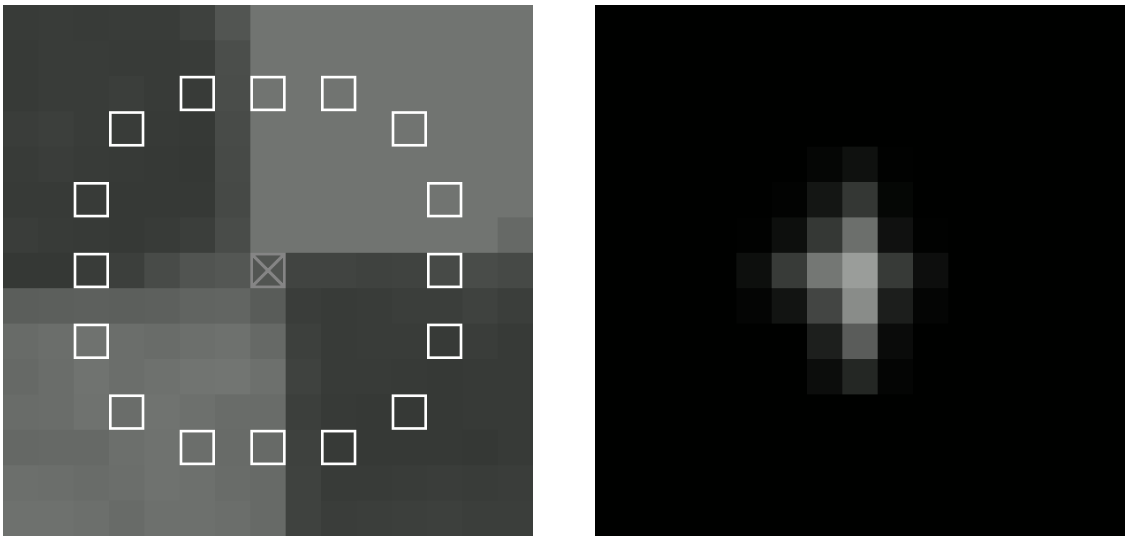
In Abschnitt 4.1 wird als Kalibrierkörper ein Schachbrett und als Messmodell die Ecken bzw. Berührungspunkte gleichfarbiger Kacheln im Kamerabild definiert. Dieser Abschnitt befasst sich mit der Erkennung dieser Ecken und einem Algorithmus zur Zuordnung und Indizierung von Ecken auf einem Schachbrett, so dass die Zuordnung von $m[p_{img}]$ zu $m[p_{cb}]$ aus Gleichung 4.9 korrekt ist. Die Schachbretterkennung muss außerdem sehr robust gegenüber Beleuchtungsveränderungen sein, die z. B. durch Schatten der verschiedenen Beinstellungen erzeugt werden können.

4.6.1 Eckenerkennung

Um die Ecken eines Schachbretts auf einem Kamerabild zu lokalisieren, wird das Verfahren von Bennet u.a[BL13] verwendet. Eine Ecke stellt den Berührungspunkt gleichfarbiger Schachbrettkacheln dar. Mit Hilfe einer speziellen Maske werden pixelweise solche Berührungspunkten auf Grauwertbildern gezielt gesucht. Die Grauwerte dieser Maske werden nach einem bestimmten Muster zusammengerechnet und das resultierende Ergebnis in ein neues Bild eingetragen, welches im Rahmen dieser Arbeit als *Antwortbild* bezeichnet wird. Aus dem *Antwortbild* werden Kandidaten für eine Ecke extrahiert. Die Anzahl von *false postives* wird mit weiteren Verarbeitungsschritten verringert. Ein *false positive* beschreibt in diesem Zusammenhang eine als korrekt erkannte falsche Ecke.

ChESS-Algorithmus

Dieses Verfahren beruht auf der pixelweisen Anwendung einer ringförmigen Maske auf einem Graustufenbild. Die Rechenzeit von Bildverarbeitungsalgorithmen skaliert mit der Anzahl von Rechnungsschritten bzw. zu betrachtenden Pixel pro Ergebnispixel. Ein guter Kompromiss aus Genauigkeit und Rechenaufwand stellt laut Bennet u.a.[BL13] die Maske aus Abbildung 4.7(a) dar, mit einem Radius von fünf Pixel. Das *Antwortbild* für die in Abbildung 4.7(a) dargestellte Ecke ist durch Abbildung 4.7(b) gegeben. Je intensiver bzw. heller ein Pixel dort erscheint, umso wahrscheinlicher handelt es sich um einen Berührungspunkt. Im ersten Schritt des Algorithmus wird die sogenannte *sum_response* für jeden Pixel mittels Formel 4.11 berechnet. Das I_n steht für einen beliebigen Pixel der Maske bzw. für den Index eines Feldes des in Abbildung 4.8 dargestellten Arrays. Der erste Eintrag entspricht dem Grauwert des linken Pixels der oberen drei Maskenpixel aus Abbildung 4.7(a). Die weitere Indizierung verläuft im



(a) (b)

Abbildung 4.7: (a) Die Maske des *ChESS-Algorithmus* (weiße Kästchen) um den aktuell betrachteten Pixel (durchgekreutztes, graues Kästchen). (b) Das Antwortbild von Abbildung 4.7(a) nach der Durchführung des Algorithmus.

Uhrzeigersinn.

$$sum\_response = \sum_{n=0}^{3} |I_n - I_{n+4} + I_{n+8} - I_{n+12}| \tag{4.11}$$

Es werden Pixel betrachtet, die um 90° auseinander liegen. Das Absolute der Differenz dieser Pixel ist im Bereich einer wirklichen Ecke groß, da die Intensitäten der Pixel sehr unterschiedlich sind. Die *sum_response* alleine reicht allerdings nicht, da noch *false positive* auf Kanten im Bild gefunden werden. Dies geschieht dann, wenn die Maske zentriert auf einer solchen Kante liegt. Für diesen Fall wird die *difference_response* (siehe Formel 4.12) gebildet. Hier werden Pixelpaare betrachtet die um 180° auseinander liegen, sich also in der Maske gegenüberstehen.

$$difference\_response = \sum_{n=0}^{7} |I_n - I_{n+8}| \tag{4.12}$$

Die Summe der absoluten Differenzen dieser Pixel fällt bei einer echten Ecke klein aus, da die gegenüberliegenden Pixel von ähnlicher Intensität sind In Abbildung 4.8 wird die Bildung der *sum_response* und der *difference_response* beispielhaft für Grauwerte der Maske aus Abbildung 4.7(a) dargestellt. Um Kanten auszuschließen, wird die *difference_response* einfach von der *sum_response* abgezogen. Ein letzter Schritt schließt weitere *false positives* aus, die noch bei dünnen Linien im Bild entstehen können. Dies ist dann der Fall, wenn der betrachtete Pixel mitten auf einer solchen Linie liegt. Um

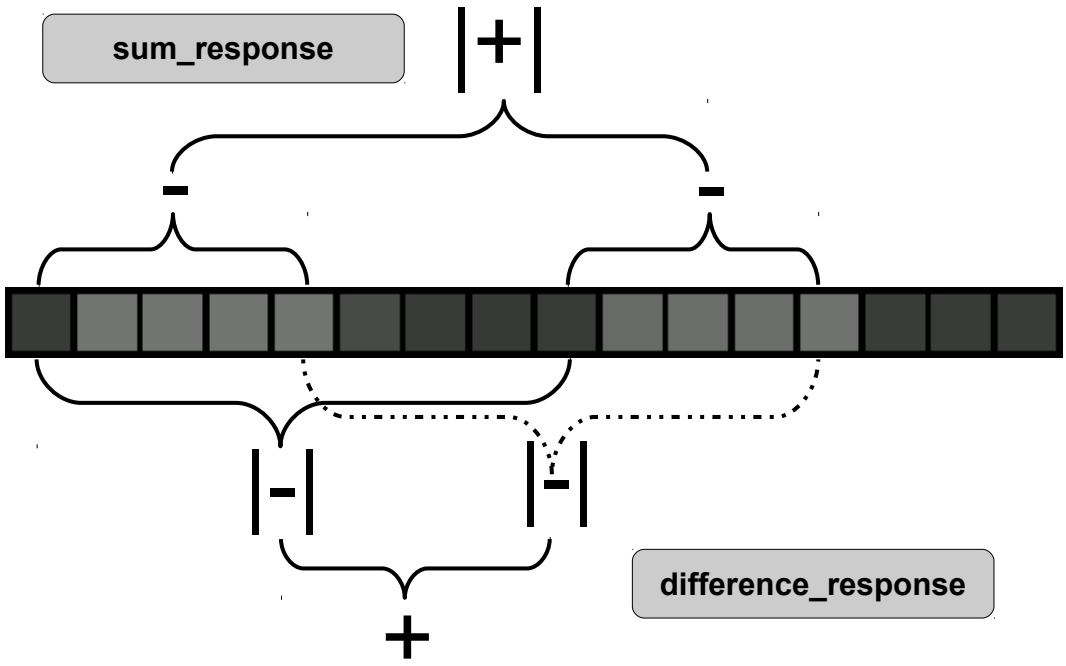


Abbildung 4.8: Darstellung der Berechnung der *sum_response* für $i = 0$ und der *difference_response* für $i = 0$ und $i = 4$.

solche Pixel auszuschließen, werden zwei weitere Werte gebildet: der *local_mean* (siehe Formel 4.14) und der *neighbour_mean* (siehe Formel 4.13) bzw. die *mean_response* (siehe Formel 4.15), welche die absolute Differenz dieser Werte darstellt.

$$neighbour\_mean = \sum_{n=0}^{15} I_n \tag{4.13}$$

$$local\_mean = (\sum_{xx=-1}^{1} \sum_{yy=-1}^{1} I(x + xx, y + yy)) * \frac{16}{9} \tag{4.14}$$

$$mean\_response = |neighbour\_mean - local\_mean| \tag{4.15}$$

Der *neighbour_mean* wird aus der Summe der Grauwerte aller Pixel der ringförmigen Maske bestimmt. Der *local_mean* beschreibt die Intensität der direkten Nachbarschaft des aktuell betrachteten Pixels, also insgesamt neun Pixel. Da die ringförmige Maske über 16 Pixel verfügt, wird der *local_mean* noch mit $\frac{16}{9}$ multipliziert. Dieser Wert wird dann mit der Anzahl der Maskenpixel multipliziert. Das in Formel 4.14 verwendete I stellt in diesem Falle das Grauwertbild dar, wobei x und y die Koordinaten des aktuell betrachteten Pixels sind. Dieser Wert wird, im Falle einer echten Kante, ungefähr so hoch ausfallen wie der *neighbour_mean*, weswegen die absolute Differenz beider niedrig ausfällt. Dies funktioniert, da echte Kameras im Bereich der Berührungspunkte der

Schachbrettfelder Schwierigkeiten mit Kontrast und Rauschen haben. Die Grauwerte nahe einer Ecke sind meist von durchschnittlichem Kontrast, was in Abbildung 4.7(a) ersichtlich wird.

$$response = sum\_response - difference\_response - mean\_response \tag{4.16}$$

Die *response* ist der finale Wert, welcher in das Antwortbild eingetragen wird, wobei Werte kleiner null mit null eingetragen werden.

Algorithmus 4.1 *ChESS*-Algorithmus

Require: I {Ein Schwarz-weiß Bild}
Require: $threshold$ {Der Schwellwert für eine akzeptable Ecke}
Require: R {Das Antwortbild dieses Algorithmus mit gleichen Dimensionen wie I}
Ensure: R enthält das korrekte Antwortbild

$possible\_corners \leftarrow \{\}$
for $x \leftarrow 5$ to $I_{width} - 5$ **do**
  for $y \leftarrow 5$ to $I_{height} - 5$ **do**
    $mask \leftarrow ARRAY[16]$
    $mask[0] \leftarrow I(x+2, y-5)$
    $mask[1] \leftarrow I(x+0, y-5)$
    $\vdots$
    $mask[14] \leftarrow I(x+5, y-2)$
    $mask[15] \leftarrow I(x+4, y-4)$
    $lm \leftarrow \frac{(I(x-1,y)+I(x,y)+I(x+1,y))*16}{3}$
    $sr \leftarrow dr \leftarrow nm \leftarrow 0$
    for $i \in \{0, 1, 2, 3\}$ **do**
      $a \leftarrow mask[i]$
      $b \leftarrow mask[i+4]$
      $c \leftarrow mask[i+8]$
      $d \leftarrow mask[i+12]$
      $sr \leftarrow sr + |a - b + c - d|$
      $dr \leftarrow dr + |a - c| + |b - d|$
      $nm \leftarrow nm + a + b + c + d$
    end for
    $response \leftarrow sr - dr - |nm - lm|$
    if $response > 0$ **then**
      $R(x, y) \leftarrow response$
      if $response > threshold$ **then**
        $possible\_corners \leftarrow \{(x, y, response)\} \cup possible\_corners$ {Eine neue Ecke hinzufügen}
      end if
    else
      $R(x, y) \leftarrow 0$
    end if
  end for
end for
return $possible\_corners$

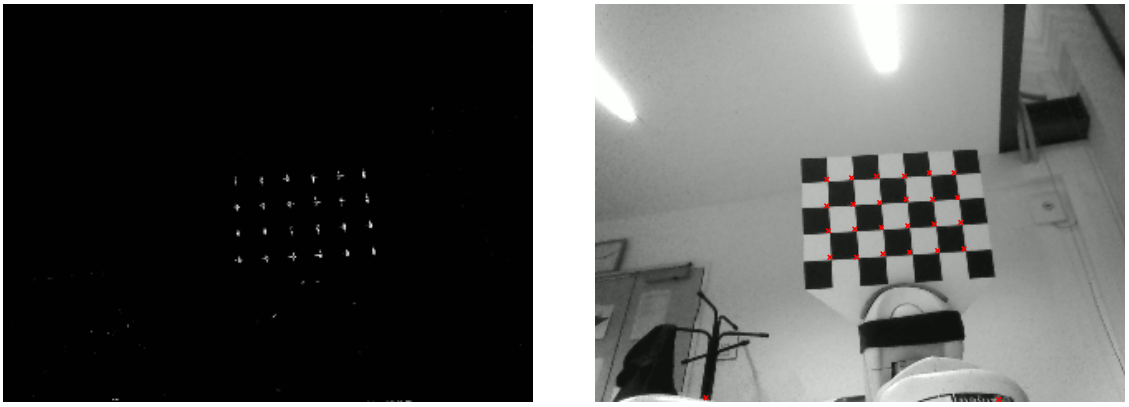
ChESSCornerDetector

Der *ChESSCornerDetector* ist als ein *Modul* implmentiert und setzt den vorgestellten *ChESS*-Algorithmus um. Als Ergebnis dieses Moduls wird die *Repräsentation Corner-Percept* bereitgestellt, welche eine Liste von möglichen Schachbrettecken enthält.

Da der *ChESS*-Algorithmus Grauwert-Pixel benötigt und das Kamerabild des *NAO*s im *YCbCr*-Farbmodell vorliegt, wird der *Y-Kanal* eines Pixels verwendet. Der *ChESS*-Algorithmus erzeugt das *Antwortbild* aus Abbildung 4.9(a). Die Bilder der unteren Kamera werden mit VGA-Auflösung bereitgestellt.

Als ersten Filterungsschritt empfiehlt Bennet u.a.[BL13] beim Erstellen des *Antwortbildes* für jeden Pixel zu prüfen, ob die *response* größer als ein Schwellwert ist. Im positiven Fall wird die Pixelkoordinate mit dem der *response* in einer Liste von möglichen Kandidaten gespeichert. Im Nahbereich einer echten Ecke sammeln sich viele solcher Kandidaten, so dass komplett isolierte Ecken entfernt werden können. Eine Ecke gilt als isoliert, falls es in der direkten Pixel-Nachbarschaft keine oder zu wenige weitere Kandidaten gibt.

Auf die Kandidaten wird eine *Non-maximum-supression*[Can86] angewandt, um die Ecken mit der lokal größten *response* zu finden. Für jede Ecke wird dabei in einer bestimmten Nachbarschaft überprüft, ob sie über die lokal maximale *response* verfügt. Gibt es in einer Nachbarschaft mehrere identische lokale Maxima, werden dessen Pixelkoordinaten gemittelt. Alle nicht-maximalen Ecken werden aus der Liste entfernt.



(a) (b)

Abbildung 4.9: (a) Das finale *Antwortbild* des *ChESS*-Algorithmus. (b) Die finale Auswahl von Ecken, dargestellt durch rote Kreuze.

Im letzten Schritt werden alle Ecken aus der Liste gelöscht, falls die *response* der Ecke im Verhältnis zur global maximalen *response* kleiner als ein parametrierbarer Schwellwert ist. Echte Ecken haben eine ähnlich große *response*, wobei die maximale *response* des *Antwortbilds* zu einer echten Ecke gehören sollte.

Verbesserung der Laufzeit und Robustheit

Um das gesamte Verfahren zu beschleunigen, wird ein Teilausschnitt des Bildes verwendet. Dieser Teilausschnitt kann mit Hilfe der *direkten Kinematik* (siehe Abschnitt 3.4) erzeugt werden, da die Position des Schachbrettes relativ zum Fußgelenk bekannt ist. Nun wird *Bounding-Box* des gesehenen Schachbretts berechnet und dessen Dimensionen etwas vergrößert. Dadurch wird sichergestellt, dass der resultierende Teilausschnitt mehr als das tatsächliche Schachbrett im Bild abdeckt. Da der Ausschnitt deutlich kleiner als das gesamte Bild ist und hauptsächlich nur das Schachbrett beinhaltet, wird die Anzahl an möglichen *false positives* verringert.

Subpixelgenauigkeit

Das vorgestellte Verfahren erzeuget eine Liste von Ecken mit diskreten Bildkoordinaten. Es gibt aber auch Methoden die kontinuierliche Koordinaten berechnen, was als Subpixelgenauigkeit bezeichnet wird. Subpixelgenaue Koordinaten verbessern die Qualität der *Roboterkalibrierung* dieser Arbeit, da die Projektionen aus Formel 4.2 bzw. Formel 4.5 ebenfalls kontinuierliche Koordinaten erzeugen.

Die Implementierung basiert auf der Methode *localmarkerdetector.m* der *Matlab*-Bibliothek *MTKM*[1], welche auch in der Kalibrierung von Birbach u.a.[OF12] angewandt wurde.

Alle gefilterten Ecken werden subpixelgenau verfeinert und in das das *CornerPercept* eingetragen, welches nun eine unsortierte Menge von subpixelgenauen Ecken enthält (siehe Abbildung 4.9(b)).

4.6.2 Schachbrettextraktion

In diesem Abschnitt wird erklärt, wie aus der unsortierten Menge von Ecken des *CornerPercepts* mit der Methode von Wang u.a.[ZX05] ein Schachbrett extrahiert wird. Das Ergebnis des Verfahrens ist eine stets gleich sortierte Menge von Ecken.

Da ein Schachbrett einem Gitter aus waage- und senkrechten Linien entspricht, können die waage- und die senkrechte Fluchtpunkte dieser Linien bestimmt werden. Anhand der Fluchtpunkte können Ecken einer Zeile und einer Spalte auf dem Schachbrettes zugeordnet werden.

[1] `https://openslam.informatik.uni-freiburg.de/data/svn/MTK/trunk/matlab/examples/calibration_common/localmarkerdetector.m`

ChessboardDetector

Das Verfahren von Wang u.a.[ZX05] ist im *Modul* ChessboardDetector implementiert und teilt sich in zwei Phasen auf:

1. Da das *CornerPercept* möglicherweise Ecken beinhaltet, welche außerhalb des Schachbrettes liegen, müssen diese in einem ersten Schritt ausgeschlossen werden.

2. Für jede Ecke wird die korrekte Position auf dem Schachbrett berechnet. Das Ergebnis, die Repräsentation *Chessboard*, wird in Abbildung 4.6.2(b) dargestellt. Hier sind die Ecken durchnummeriert, beginnend mit der oberen linken und dann zeilenweise bis zur unteren rechten Ecke.

Zu Beginn wird zu überprüft, ob ausreichend viele Ecken erkannt wurden. Im Falle der fünf mal sieben Kacheln großen Schachbrettsandale müssen mindestens 24 Ecken erkannt werden.

Im nächsten Schritt der Schwerpunkt der Ecken berechnet.

$$centroid = \frac{\sum_{i=0}^{n} c_i * response_i}{\sum_{i=0}^{n} response_i} \tag{4.17}$$

Optimalerweise befindet sich dieser innerhalb des Schachbrettes, zentriert in einem Schachbrettfeld. Von Wang u.a.[ZX05] wird der folgende Schritt verlangt:

> „Find out four corners, p_{c_1}, p_{c_2} , p_{c_3} and p_{c_4} , which are closest to $p_c(x_c, y_c)$ and belong to the same square.“

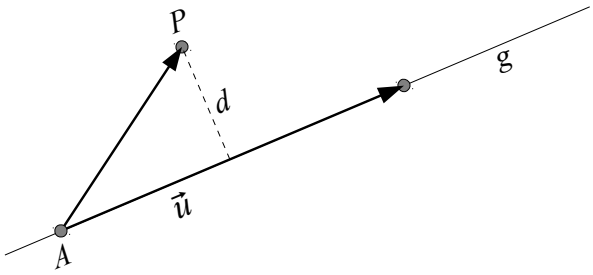
Es sollen also vier Ecken gefunden werden, die zu ein und demselben Schachbrettfeld gehören. Die Berechnung des Schwerpunktes *centroid* weicht von der Defintion von $p_c(x_c, y_c)$ ab, da eine Ecke mit der jeweiligen *response* gewichtet wird.

Die Ecken des *CornerPercepts* werden nach der Distanz zu *centroid* aufsteigend sortiert und die ersten vier Ecken dieser Sortierung ausgewählt. Wie sichergestellt werden soll, dass diese Ecken auch zum selben Schachbrettfeld gehören wird von Wang u.a.[ZX05] nicht angeführt.

Aus den vier Ecken werden nun vier Linien erzeugt (siehe Abbildung 4.6.2 (a)):

1. Waagerechte Linie blau zu grün.

2. Senkrechte Linie grün zu rot.



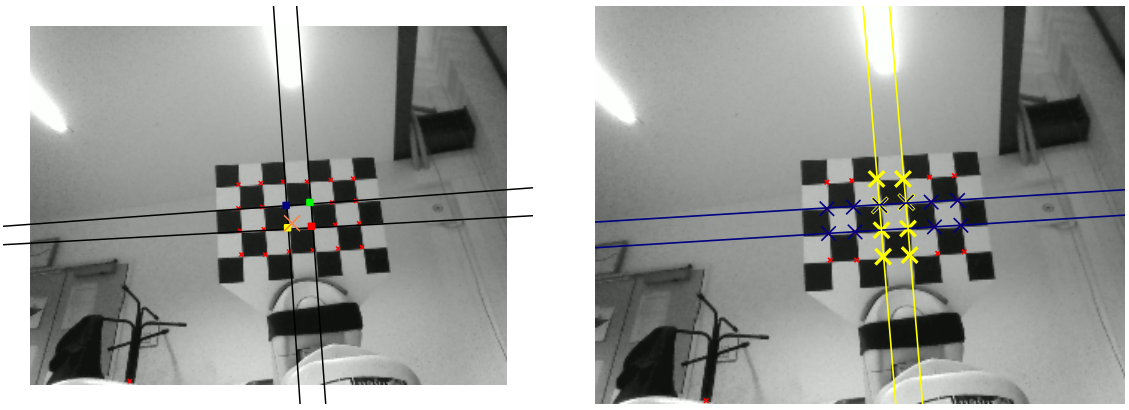
$$d = \frac{|(\overrightarrow{p} - \overrightarrow{a}) \times \overrightarrow{u}|}{|\overrightarrow{u}|} \tag{4.18}$$

Abbildung 4.10: Berechnung des Abstandes eines Punktes von einer Linie nach Formel 4.18.

3. Waagerechte Linie rot zu gelb.

4. Senkrechte Linie gelb zu blau.

Pro Linie werden nun alle Ecken gesucht, welche nah der jeweiligen Linie liegen. Zur Berechnung des Abstandes eines Punktes von einer Linie wird Formel 4.18 verwendet. Die maximal gültige Distanz, die eine Ecke von einer Linie haben darf, wird über einen konfigurierbaren Parameter bestimmt.



(a) (b)

Abbildung 4.11: (a) Darstellung des Schwerpunktes aller roten Ecken als oranges Kreuz; der vier nächsten Ecken als roter, grüner, gelber und blauer Punkt; der aus den vier Punkten erzeugten schwarzen Linien. (b) Die durch lineare Regression verbesserten Linien sowie deren zugeordneten Ecken. Der Fluchtpunkt v_1 befindet sich oberhalb und v_2 links außerhalb des Bildes (Schnittpunkte der gleichfarbigen Linien).

Pro Linie gibt es nun einen Satz Ecken, welche im nächsten Schritt dazu verwendet werden, die Linien dahingehend zu verbessern, dass die Linie möglichst nah an allen zugehörigen Ecken liegt. Dies ist für die folgende Berechnung von Fluchtpunkten nützlich, worauf aber erst im weiteren Verlauf dieses Textes eingegangen wird. Dieser

Schritt stellt eine Ergänzung zum Verfahren von Wang u.a[ZX05] dar.

Die Linien werden mittels der *linearen Methode der kleinsten Quadrate* (siehe Abschnitt 3.6) verbessert. Da hier nur einfache Ausgleichsgeraden anhand von jeweils n Ecken pro Linie berechnet werden, kann eine einfache Ausgleichsrechnung anwenden.

$$f(x) = m * x + b \tag{4.19}$$

Die Funktion f berechnet eine Gerade mit der Steigung m und dem Achsenabschnitt b.

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{4.20}$$

$$\bar{y} = \frac{1}{n} \sum_{i=1}^{n} y_i \tag{4.21}$$

Die Werte $\bar{x}$ und $\bar{y}$ geben dabei den jeweiligen Durchschnitt der Koordinatenkomponenten der Ecken an. Die Steigung m wird dann mittels

$$m = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2}. \tag{4.22}$$

und der Achsenabschnitt b mit

$$b = \bar{y} - m * \bar{x} \tag{4.23}$$

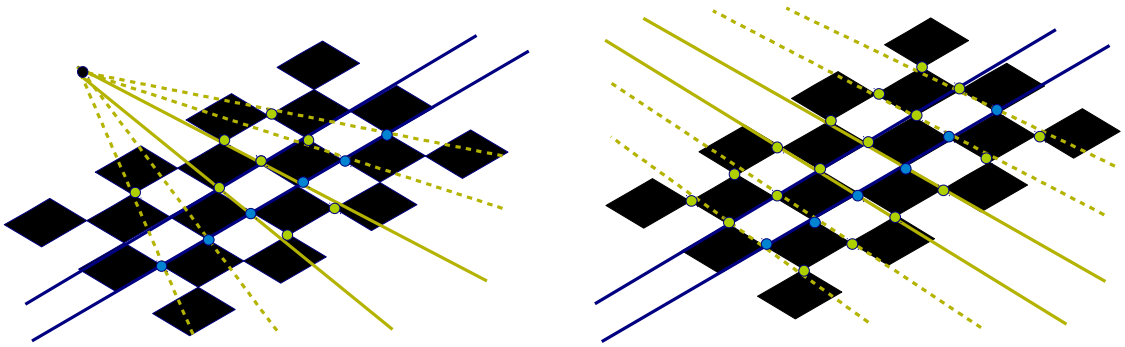
abschließend bestimmt.

In Abbildung 4.6.2 (b) werden die verbesserten Linien dargestellt. Im Vergleich zu den Linien aus Abbildung 4.6.2 (a) wird deutlich, dass die verbesserten Linien näher an den jeweiligen Ecken liegen und dass die beiden blauen waagerechten Linien augenscheinlich parallel zueinander verlaufen.

Aus den verfeinerten Linien werden nun die Fluchtpunkte bestimmt, welche im Verlauf dieses Texts mit v_1 und v_2 bezeichnet werden. Der Fluchtpunkt ergibt sich aus dem Schnittpunkt der beiden senkrechten bzw. waagerechten Linien.

Unter der Annahme, dass v_1 aus den beiden senkrechten Linien bestimmt wurde, wird die waagerechte Linie mit der größten Distanz (siehe Formel 4.18) zu v_1 ausgewählt. Für jede Ecke, die nah an dieser Linie liegen, werden neue Linien zu v_1 erzeugt. Für jede dieser neuen Linien werden alle Ecken gesucht, die nahe an diesen liegen; es erfolgt eine Zuordnung von Ecken zu Linien. Genauso wird mit den senkrechten Linien und v_2 verfahren.

Die vorher erwähnte Problematik der Fluchtpunkten und dessen Distanz zum Schachbrett wird in Abbildung 4.6.2(a) deutlich. Der Schnittpunkt der beiden durchgezogenen gelben Linien ergibt den schwarzen Flucktpunkt links oberhalb des Schachbrettes. Die nun zusätzlich erstellten Linien (gestrichelt gelb) von den Ecken (blaue Kreise) zum Fluchtpunkt, sorgen dafür, dass nicht alle Ecken des Schachbrettes, aufgrund der zu großen Distanz, einer Linie zugeordnet werden können (gelbe Kreise). In Abbildung 4.6.2(b) liegt der Fluchtpunkt außerhalb des Bildes, weit entfernt vom Schachbrett. Es wird ersichtlich, dass alle Ecken (gelbe Kreise) einer Linie zugeordnet werden konnten.



(a) (b)

Abbildung 4.12: (a) Der Fluchtpunkt (schwarzer Kreis) ist zu nah am Schachbrett; nicht alle Ecken (gelbe Kreise) wurden einer Linie zugeordnet. (b) Alle Ecken konnten einer Linie zugeordnet werden, da sich der Fluchtpunkt weit entfernt vom Schachbrett befindet.

Die Linien samt zugehöriger Ecken werden im weiteren Verlauf dieses Textes mit L_{v_1} und L_{v_2} bezeichnet und durch Formel 4.24, am Beispiel von L_{v_1}, formalisiert.

$$L_{v_1} = \left\{ \overbrace{\{(x_{00}, y_{00}), (x_{01}, y_{01}), \ldots\}}^{\textit{Eine Linie, definiert durch zugewiesene Ecken}}, \{\underbrace{(x_{10}, y_{10})}_{\textit{Koordinate einer Ecke}}, (x_{11}, y_{11}), \ldots\}, \ldots \right\} \tag{4.24}$$

In L_{v_1} bzw. L_{v_2} befinden sich alle Linien zum Fluchtpunkt v_1 bzw. v_2. Ein Element aus diesen Mengen beinhaltet die Menge an Ecken, welche in den vorausgegangenen Schritten einer Linie zugeordnet wurden (siehe Abbildung 4.6.2 (b)). Für die hier folgenden Formalismen wird eine Linie und ihre Ecken nur durch eine Menge von Ecken dargestellt (die zugehörige Linie ergibt sich durch die Ecken). Diese Darstellung erleichter die folgenden Formalismen.

Es werden zwei Mengen $S1$ und $S2$ wie folgt gebildet:

$$mean_{L_{v_1}} = \frac{\sum_{i=0}^{|L_{v_1}|} |L_{v_1 i}|}{|L_{v_1}|} \tag{4.25}$$

$$mean_{L_{v_2}} = \frac{\sum_{i=0}^{|L_{v_2}|} |L_{v_2 i}|}{|L_{v_2}|} \tag{4.26}$$

Mit $L_{v_1 i}$ ist ein Element der Menge L_{v_1} gemeint, also wiederum eine Menge mit den zur jeweiligen Linie zugeordneten Ecken. In Formel 4.25 und Formel 4.26 wird die durchschnittliche Anzahl der Ecken pro Linie der Mengen L_{v_1} bzw. L_{v_2} berechnet.

$$S1 = \bigcup \left\{ l \in L_{v_1} \;\middle|\; |l \cap (\bigcup L_{v_2})| \geq mean_{L_{v_1}} \right\} \tag{4.27}$$

$$S2 = \bigcup \left\{ l \in L_{v_2} \;\middle|\; |l \cap (\bigcup L_{v_1})| \geq mean_{L_{v_2}} \right\} \tag{4.28}$$

Durch Formel 4.27 befinden sich beispielsweise in $S1$ alle Ecken einer Linie l, welche auch in $\bigcup L_{v_2}$ enthalten sind, falls die Kardinalität des Schnittes aus dieser Linie mit L_{v_2} größer gleich der durchschnittlichen Anzahl an Ecken pro Linie in L_{v_1} ist.

Die Implementierung für die automatische Roboterkalibrierung für den *NAO* weicht an zwei Stellen von der Methode von Wang u.a.[ZX05] ab:

- In der Publikation werden die Ecken einer Linie nur verwendet, wenn die Anzahl der Ecken *größer* als der Durchschnitt von Ecken pro Linie ist. Dies führt zu Problemen, wenn angenommen wird, dass es keine *false positives* bei der Eckenerkennung gibt. Dann wären pro Linie genau die durchschnittliche Anzahl von Ecken vorhanden, was dazu führt, dass alle Linien und deren Ecken verworfen werden. Aus diesem Grund wird in der Implementierung dieser Arbeit ein *größer gleich* an der besagten Stelle verwendet.

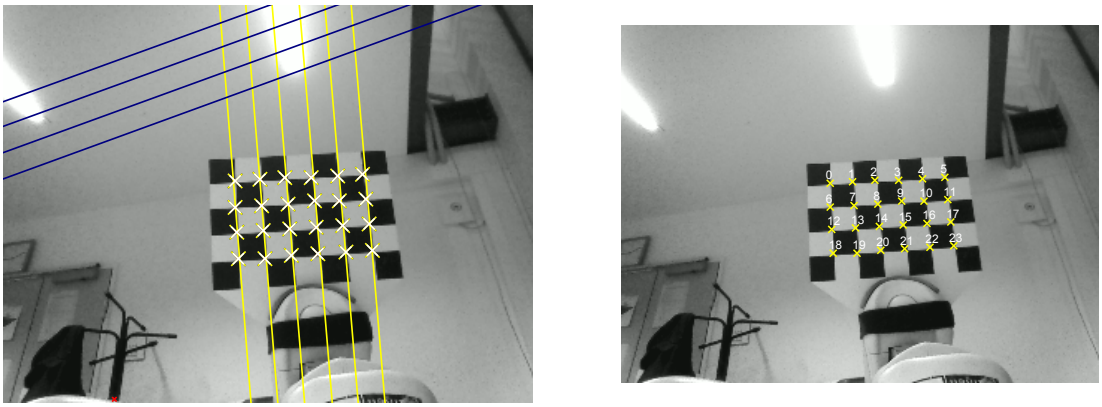
- Der berechnete Durchschnitt von Ecken pro wird auf die erwartete Anzahl von Ecken pro Linie limitiert. Falls durch *false positives* der Durchschnitt von Ecken pro Linie beispielsweise größer wird, als die erwartete Anzahl von Ecken pro Linie, werden auch die Ecken von Linien verworfen, die über die erwartete Anzahl von Ecken verfügen.

Mit diesen Anpassungen werden im weiteren Verlauf nur noch Linien betrachtet, die über ausreichend viele zugeordnete Ecken verfügen.

Aus $S1$ und $S2$ wird mittels Durchschnitt die Menge S gebildet, welche nur noch Ecken enthält, die zum Schachbrett gehören.

$$S = S1 \cap S2 \tag{4.29}$$

Hiermit ist die erste Phase dieses Algorithmus abgeschlossen und es kann mit der Bestimmung der internen Lage der Ecken auf dem Schachbrett fortgefahren werden. Hierfür werden die Schritte der ersten Phase wiederholt. Allerdings wird jetzt nur die



(a) (b)

Abbildung 4.13: (a) Der Durschnitt der Menge $S1$ und $S2$ wird durch die weißen Kreuze dargestellt. Durch geeignete Sortierung der Linien und Ecken nach der Distanz zu den beiden Fluchtpunkten erzeugt die in (b) dargestellte Ordnung. (b) Alle Ecken verfügen über einen eindeutigen Index und sind korrekt Zeile für Zeile durchnummeriert.

Menge L_{v_1} aus S gebildet. Um die interne Lage der einzelnen Ecken auf dem Schachbrett zu bestimmen werden die Linien und deren Ecken, abhängig vom Abstand zu v_1 und v_2 sortiert.

Die Sortierung hängt dann von der Lage der Fluchtpunkte v_1 und v_2 ab. Befindet sich v_1 beispielsweise oberhalb des Schachbrettes im Bild, so werden die Linien aus L_{v_1} aufsteigend nach Abstand zu v_1 sortiert, was die Ordnung der Zeilen ergibt. Für die Spaltenordnung werden die Ecken der Linien aufsteigend nach Abstand zu v_2 sortiert insofern v_2 links vom Schachbrett liegt. Als Ergebnis liegt nun eine sortierte Menge von Ecken vor, so dass die Ecken zeilenweise von links nach rechts korrekt durchnummeriert sind, was durch Abbildung 4.6.2(b) verdeutlicht wird.

Durch diese Ordnung kann eine Zuordnung von Bild- und Weltkoordinate einer Ecke hergestellt werden, da durch die Nummerierung jeder Ecke im Bild klar ist, um welche Ecke es sich in der Welt handelt.

4.7 Bewegungssteuerung

Die *B-Human*-Software verfügt über spezialisierte Bewegungssteuerungen. Die *WalkingEngine* ermöglicht den zweibeinigen Lauf, die *GetUpEngine* stabilisiert das Aufstehen und das *Modul BIKE* erzeugt dynamische, balancierte Schüsse. Weiterführende Informationen zu diesen *Modulen* beinhaltet die Dokumentation[Röf+13b] des Frameworks von *B-Human*

Neben diesen speziellen Bewegungssteuerungen gibt es die sogenannten *SpecialActions*. Mit *SpecialActions* können grundsätzlich alle mit dem *NAO* möglichen Bewegungen erstellt werden. Dafür wird eine Abfolge von Gelenwinkeln definiert, zwischen denen linear interpoliert wird. Für diese Arbeit wurde eine weitere allgemeine Bewegungssteuerung entwickelt, da sich die Umsetzung der Bewegungen mit *SpecialActions* als zu aufwändig herausgestellt hat: die *JointMotionEngine*.

4.7.1 JointMotionEngine

Für die Bewegungsdefinitionen existiert eine bestimmte Syntax, welche im Anhang 7.1 in *EBNF* (Erweiterte Backus-Naur-Form) zu finden ist. Alle Bewegungen werden in einer Konfigurationsdatei definiert, die beim Start der B-Human Software vom *JointMotionParser* (ein *rekursiv absteigender Parser*) auf korrekte Syntax überprüft wird. Nach erfolgtem Parsieren werden Datenstrukturen erzeugt, welche die Winkel und andere Steuerinformationen enthalten. Eine Bewegung der *JointMotionEngine* wird über die *Repräsentation MotionRequest* angefordert.

Eine Bewegung wird durch eine Abfolge von Gelenkwinkel erzeugt. Für jeden Winkelsatz wird angegeben, wieviel Zeit die *JointMotionEngine* hat, diese anzusteuern, wobei dies durch lineare Interpolation geschieht.

Lineare Interpolation

Damit der Wechsel von Gelenkstellungen auch schonend vonstattengeht, wird zwischen zwei Gelenkstellungen linear interpoliert. Hierfür muss lediglich eine Zeit definiert werden, in welcher der Wechsel durchgeführt werden soll.

$$\theta = \frac{duration - \Delta t}{duration} * (\theta_{to} + (\theta_{from} - \theta_{to})) \tag{4.30}$$

Dabei sind θ_{from} die Startwerte und θ_{to} die Zielwerte der Gelenke. Die *duration* gibt die Dauer der Interpolation und Δt die verstrichene Zeit seit Beginn der Interpolation an. Das Ergebnis von Formel 4.30 sind die im aktuellen *Motion*-Zyklus zu setzenden Gelenkwinkel θ.

Eigenschaften

Für diese Arbeit ist es wichtig, dass der Ablauf einer Bewegung kontrollierbar ist. Hierfür werden die jeweiligen Gelenkstellungen speziell ausgezeichnet, so dass nach erfolgter Ansteuerung keine neuen Stellungen angefahren werden. Die Fortführung wird über einen Schalter innerhalb des *MotionRequest* angeregt. Die Möglichkeit der Steuerung ist für notwendig, da verschiedene Beinstellungen angesteuert werden, um unter verschiedenen Kopfstellungen das Schachbrett zu erkennen. Eine neue Beinstellung wird dann angefahren, sobald ein Schachbrett erfolgreich erkannt wurde oder nach Ablauf einer maximalen Wartezeit.

Für die Bewegungen der *SpecialActions* muss jeder Winkelsatz vollständig sein, d. h. es müssen Winkelangaben für alle Servormotoren des *NAO*s definiert werden. Die Syntax der *JointMotionEngine* erlaubt es, auch unvollständige Winkelsätze zu verwenden. Die erforderlichen Gelenke werden benannt und dazu ein entweder absoluter Winkel oder ein relativer Winkelversatz angegeben.

Format

Die Syntax einiger simpler *Bewegungsdefinitionen* wird im folgenden Quelltext 4.1 gezeigt:

Die *JointMotionEngine* bedingt, dass die erste Bewegungsdefinition, der Konfigurationsdatei, *motions* heißen muss, welches als erstes ein *Label* namens *start* besitzt. Jede Bewegungsdefinition muss mit einem initialen *Label* (siehe Zeile 3) beginnen, welche als Einstiegspunkte in die Bewegung dienen. Ein *Label* wird durch eine *Transition* aktiviert. Eine *Transition* besteht aus dem Schlüsselwort *transition*, gefolgt von dem Bezeichner der durch den *MotionRequest* angeforderten Bewegung. Eine Transition wird dann genommen, wenn der Bezeichner nach dem Schlüsselwort *Transition* gleich der angeforderten Bewegung des *MotionRequest* ist. Das Ziel der *Transition* stellen die letzten beiden Angaben dar, wobei die Erste den Namen der Bewegungsdefintion und der Zweite den Namen des *Labels* enthält. Neben dieser *bedingten Transition* (siehe Zeile 4) gibt es auch eine *unbedingte Transition*(siehe Zeile 33), welche genommen wird, sobald diese erreicht wurde. Die Bewegung *motions* verfügt über *bedingte Transitionen* zu den anderen Definitionen.

```
motions                                                                    1
{                                                                          2
  tlabel start                                                             3
  transition test test start                                               4
  transition test2 test2 start                                             5
  transition test3 test3 start                                             6
  transition motions start                                                 7
}                                                                          8
                                                                           9
test                                                                       10
{                                                                          11
  label start                                                              12
  hardness(0 0 0 0 0 0 0 0 0 0 8 8 8 8 8 8 8 8 8 8 8) 0                   13
  (10 -5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0) 20 0                       14
  transition test2 demo                                                    15
}                                                                          16
                                                                           17
test2                                                                      18
{                                                                          19
  label start                                                              20
  (0 0 0 0 0 0 0 0 0 0 0 -9.5 0 0 0 0 0 0 0 0 0 0) 30 0                    21
  label demo                                                               22
  (HeadPitch 10 HeadYaw 0)                                                 23
  transition test test start                                               24
  {RHipPitch -10 RHipRoll -6} 100 20 abdce                                 25
  (HeadPitch 10 HeadYaw 0) 100 0 milestone                                 26
  transition motions start                                                 27
}                                                                          28
                                                                           29
test3 | test2                                                              30
{                                                                          31
  transition test test start                                               32
  transition motions start                                                 33
}                                                                          34
```

Quelltext 4.1: Einige simple *Bewegungsdefinitionen.*

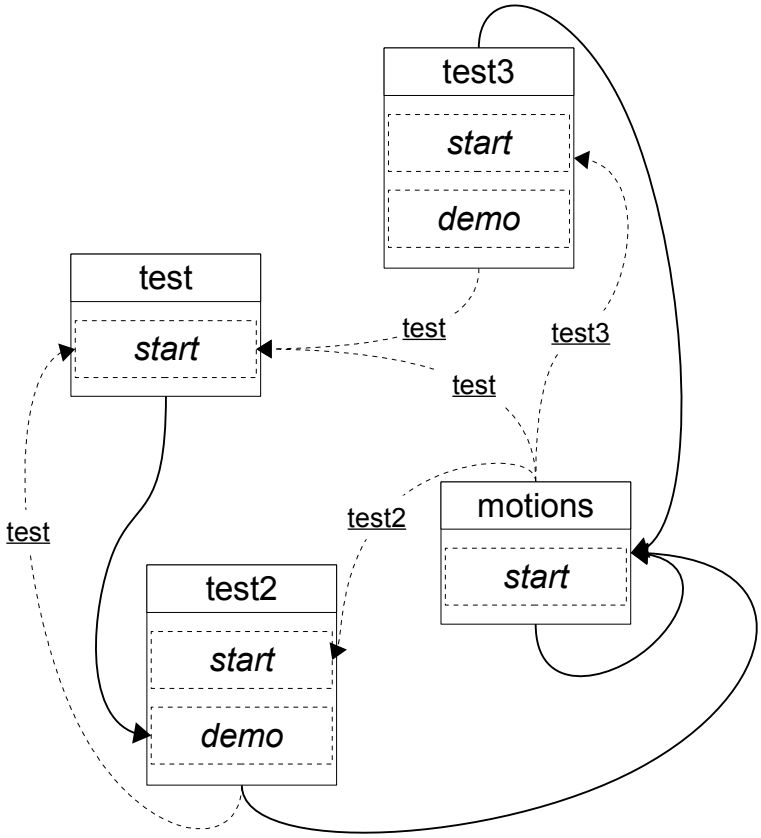


Abbildung 4.14: Darstellung der Transitionen der Bewegungsdefinitionen aus Quelltext 4.1. Durchgängige Kanten stellen unbedingte und gestrichelte bedingte Transitionen dar. Die Beschriftung einer unbedingten Transition steht für die durch den *MotionRequest* angeforderte Bewegung. Die durchgängigen Kästen sind dabei die Bewegungsdefinitionen und die gestrichelten Boxen stehen für *Labels*

.

In Zeile 21 wird ein vollständiger Satz von Gelenkwinkeln angefordert. Die eingeklammerte Liste von Dezimalzahlen stellen die absoluten Winkel in ° dar, beginnend mit den beiden Kopfgelenken gefolgt von zehn Winkeln für die Arme und weiteren zehn Winkeln für die Beine. Die beiden Zahlen neben der schließenden Klammer geben die Dauer der Interpolation und eine maximale Rotationsgeschwindigkeit der Gelenke in Millisekunden an. Sind diese beiden Werte auf null gesetzt, werden die angeforderten Gelenke so schnell wie möglich angesteuert. Es ist auch möglich einzelne Gelnke mit einem absoluten Winkelwert zu versehen, was durch Zeile 23 dargestellt wird.

In Zeile 25 wird die Syntax für einen relativen Versatz von Gelenkwinkeln gezeigt. Diese beginnen mit einer öffnenden geschweiften Klammer. In der besagten Zeile wird das Hüftgelenk um $-10°$ zur Seite und um $-6°$ nach vorne rotiert.

Es besteht die Möglichkeit, Steifigkeiten für die Servomotoren anzugeben. Je weniger steif ein Gelenk ist, um so schneller gibt es externen Krafteinflüssen nach. In Zeile 13 wird ersichtlich, dass solche Angaben mit dem Schlüsselwort *hardness* beginnen.

Das Schlüsselwort *milestone* (siehe Zeile 26), am Ende einer Zeile, gibt an, dass nach Ansteuerung der geforderten Gelenkwinkel keine neuen Winkelsätze angefahren werden; der *NAO* verharrt in der aktuellen Stellung. Die Fortführung der Bewegung wird durch einen Schalter innerhalb des *MotionRequest* gesteuert.

Am Ende der Zeile 25 befindet sich ein spezielles Token, welches eine optionale Zeichenkette darstellt. Mit dieser Zeichenkette lässt sich feststellen, welche Stellung aktuell angesteuert wird. Der letzte Winkelsatz der Bewegungsdefinition der automatischen Roboterkalibrierung verfügt über die Zeichenkette *end*. So ist es möglich zu überprüfen, ob das Ende der Bewegung erreicht wurde.

Für eine gespiegelte Bewegungsdefinition wird die Syntax aus Zeile 30 verwendet. Die Bewegung *test3* ist eine gespiegelte Variante von *test3*, wobei die Transitionen angepasst werden müssen; die Gelenkwinkelangaben werden automatisch kopiert und gespiegelt.

In Abbildung 4.14 werden die Bewegungsdefinitionen und Transitionen aus Quelltext 4.1 als Graph dargestellt.

4.8 Optimierer

In diesem Abschnitt wird zusammengefasst, wie das in Formel 4.10 definierte nichtlineare Optimierungsproblem gelöst werden kann. Die Wahl fiel dabei auf das Verfahren von *Levenberg*[Lev44] und *Marquardt*[Mar63], da dieses in vielen Veröffentlichungen zur Roboterkalibrierung verwendet wird und sich dort bewehrt hat.

Der *Levenberg-Marquardt*-Alogrithmus macht sich jeweils die positiven Eigenschaften des *Gradientenverfahrens* und des *Gauß-Newton*-Algorithmus zu eigen, indem zwischen beiden Verfahren interpoliert wird. Bei beiden Verfahren handelt es sich um iterative Algorithmen, welche ausgehend von einer initialen Parameterbelegung α^0 eine Folge $\alpha^1, \alpha^2, \ldots, \alpha^k$ erzeugen, die zum dem lokalen Minimum einer von α abhängigen Zielfunktion konvergiert. Die zu minimierende Zielfunktion stellt hierbei Formel 3.30 dar.

Gradientenverfahren

Beim *Gradientenverfahren* wird sich mit einer bestimmten Schrittlänge γ, in Richtung des negativen Gradienten J_r^T der Zielfunktion, einem lokalen Minimum angenähert. Hierfür muss in jeder Iteration die Steigung an der aktuellen Position sowie die Schrittlänge neu berechnet werden.

$$\alpha_\delta = \gamma J_r^T r^k \tag{4.31}$$

Das Verfahren konvergiert eher langsam, da die Parameterfolge einen „Zick-Zack“ ähnlichen Verlauf nimmt, was sich vor allem im Nahbereich des Minimus negativ auswirkt. Allerdings ist durch geeignete Anpassung der Schrittweiten γ in jeder Iteration Konvergenz garantiert.

Gauß-Newton-Algorithmus

Der *Gauß-Newton-Algorithmus* ergibt sich aus der Definition eines nichtlinearen Optimierungsproblem der kleinsten Quadrate aus Abschnitt 3.6. Dabei wird das eigentliche nichtlineare Problem durch in lineares ersetzt.

$$\alpha_\delta = (J_r^T J_r)^{-1} J_r^T r^k \tag{4.32}$$

Das Verfahren macht sich zu eigen, dass die Hesse-Matrix (Matrix der zweiten partiellen Ableitungen) aus der Jacobi-Matrix approximiert werden kann ($J_r^T J_r$), so dass die aufwändige Berechnung wegfällt. Im Nahbereich des lokalen Minimus konvergiert der Algorithmus sehr schnell, bei größerer Distanzen allerdings sehr langsam.

Levenberg-Marquardt

Um zwischen den beiden vorgestellten Verfahren umschalten zu können, hat *Kenneth Levenberg* die Formel 4.32, die Normalengleichung, um eine skalierbare Einheitsmatrix erweitert:

$$\alpha_\delta = (J_r^T J_r + \lambda I)^{-1} J_r^T r^k. \tag{4.33}$$

Der Skalierungswert λ muss mit einer initialen Belegung versehen und nach jeder Iteration angepasst werden. Falls die Summe der quadrierten Residuen χ_i^2 kleiner ist als χ_{i-1}^2 der vorherigen Iteration, so wird der Wert von λ verkleinert und die nächste Iteration eingeleitet. Falls in der aktuellen Iteration keine Verkleinerung von χ_i^2 gegenüber χ_{i-1}^2 erreicht wurde, wird λ solange vergrößert, bis χ_i^2 kleiner als χ_{i-1}^2 wird, wobei eine Neuaswertung von J_r nicht erforderlich ist.

Da durch die skalierte Einheitsmatrix I nur die diagonalen Werte von $J_r^T J_r$ verändert

werden, wird bei einem großen Wert für λ aus $J_r^T J_r$ annäherungsweise eine Diagonalmatrix. Das Inverse einer Diagonalmatrix sind dann die inversen Werte auf der Hauptdiagonalen. Das $(J_r^T J_r + \lambda I)^{-1}$ aus Formel 4.33 ist dann die Schrittweite γ aus Formel 4.31 des Gradientenverfahrens.

Falls λ sehr klein ist, handelt es sich bei der aktuellen Iteration um das normale Gauß-Newton-Verfahren, da die Skalierungsmatrix annäherungsweise zu einer Nullmatrix wird.

Marquardt[Mar63] hat den Algorithmus dahingehend angepasst, dass anstatt der Einheitsmatrix I die Diagonalmatrix $diag(J_r^T J_r)$ verwendet wird:

$$\alpha_\delta = (J_r^T J_r + \lambda \; diag(J_r^T J_r))^{-1} J_r^T r^k. \tag{4.34}$$

Hierdurch wird laut Marquardt[Mar63] die Konvergenzgeschwindigkeit verbessert.

4.8.1 Implementierung

Die Umsetzung ist durch die Implementierung [2] des Algorithmus in der Matlab-Bibliothek *MTKM*[3] inspiriert und wird durch Algorithmus 4.2 in Pseudo-Code dargestellt. Für die Matrizen- und Vektorenrechnung wird die freie Matrizen-Bibliothek *Eigen*[4] verwendet, welche auf Geschwindigkeit optimiert ist und über diverse Möglichkeiten für das Lösen von Gleichungssystemen verfügt.

Es gibt insgesamt drei Terminierungskriterien:

- Maximale Anzahl an Iterationen ($iter < num\_of\_iterations$). In der Implementierung wird die Iterationsanzahl nur erhöht, wenn ein Parametersatz gefunden wurde, der χ^2 kleiner werden lässt.
- Differenz der aktuellen und letzten χ^2 ist kleiner als ein Schwellwert ($|\chi^2_{old} - \chi^2| < tol$), wobei dies eine bestimmte Anzahl oft passieren darf ($conv > convergence\_count$).
- Schwellwert für λ wird überschritten. Wenn λ größer als $1e^{16}$ wird, terminiert der Algorithmus.

[2] `https://svn.openslam.org/data/svn/MTK/trunk/matlab/nrlm.m`, abgerufen am 04.12.2013
[3] `http://openslam.org/MTK.html`
[4] `http://eigen.tuxfamily.org`

Jacobi-Matrix

Die Jacobi-Matrix beinhaltet sämtliche erste partielle Ableitungen mehrerer differenzierbarer Funktionen (siehe Formel 3.28). Da eine analytische Differenzierung des Residuums (siehe Formel 4.9) zu aufwändig ist, werden die Ableitungen numerisch durch Berechnung des *symmetrischen Differenzquotient 1. Ordnung* bestimmt:

$$f'(x) = \frac{f(x+\Delta) - f(x-\Delta)}{2\Delta}. \tag{4.35}$$

In der Implementierung der automatischen Roboterkalibrierung wird für Δ das einfach genaue ϵ aus **<limits.h>** der *Standard Template Library* von *C++* verwendet. Dieses ϵ ist die kleinste positive Gleitkommazahl für die gilt: $1 + \epsilon > 1$.

Lösen von Gleichungssystemen

In Zeile 8 von Algorithmus 4.2 wird das Gleichungssystem mit Matrixinversion gelöst. In der Implementierung wird jedoch die von *Eigen* angebotene *Cholesky-Zerlegung* [5] verwendet, da diese im Vergleich zur Matrixinversion numerisch stabiler und effizienter ist. Dabei wird die LDL^T-Zerlegung, anstatt der klassischen LL^T-Zerlegung, verwendet, da diese laut der *Eigen*-Dokumentation effizienter ist.

Fehleranalyse

Um den asymptotischen Standardfehler bzw. die Standardabweichung der optimierten Parameter zu erhalten, wird die unskalierte Hesse-Matrix der finalen Iteration verwendet. Der Standardabweichungen α_σ werden aus der Quadratwurzel der Elemente der Hauptdiagonalen der invertierten Hesse-Matrix berechnet:

$$\alpha_\sigma = \sqrt{diag((J_r^T * J_r)^{-1})}. \tag{4.36}$$

[5] `http://eigen.tuxfamily.org/dox-2.0/TutorialAdvancedLinearAlgebra.html#TutorialAdvCholesky`, abgerufen am 04.12.2013

Algorithmus 4.2 *Levenberg-Marquardt*

Require: $parameters$ {Die zu optimierenden Parameter (Siehe Tabelle 4.1)}
Require: $measurements$ {Die Messungen (Siehe Gleichung 4.4)}
Require: $error$ {Funktion zur Berechnung des Residuums}
Require: tol {Toleranz-Schwellwert}
Require: $delta$ {Siehe Gleichung 4.35}
Require: $num\_of\_iterations$ {Anzahl der maximalen Iterationen}
Require: $convergence\_count$ {Anzahl aufeinanderfolgender Konvergenzen}

1: $iter \leftarrow 0$
2: $\lambda \leftarrow 0.001$
3: $J \leftarrow jacobi(parameters, measurements, error, delta)$
4: $(\chi^2, residuals) \leftarrow residuals(parameters, measurements, error)$
5: $\chi^2_{old} \leftarrow \chi^2$
6: $conv \leftarrow 0$
7: **while** $iter < num\_of\_iterations$ **do**
8: $da \leftarrow ((J^T * J) + (diag(J^T * J) * \lambda))^{-1} * (J^T * residuals)$
9: $parameters_{new} \leftarrow parameters - da$
10: $(\chi^2, residuals_{new}) \leftarrow residuals(parameters_{new}, measurements, error)$
11: **if** $|\chi^2 < \chi^2_{old}| < tol$ **then**
12: $conv \leftarrow conv + 1$
13: **if** $conv > convergence\_count$ **then**
14: **if** $\chi^2 < \chi^2_{old}$ **then**
15: $parameters \leftarrow parameters_{new}$
16: **end if**
17: $break$
18: **end if**
19: **end if**
20: **if** $\chi^2 < \chi^2_{old}$ **then**
21: $\lambda \leftarrow \lambda * 0.1$
22: $\chi^2_{old} \leftarrow \chi^2$
23: $residuals \leftarrow residuals_{new}$
24: $J \leftarrow jacobi(parameters\_new, measurements, error, delta)$
25: $parameters \leftarrow parameters_{new}$
26: $iter \leftarrow iter + 1$
27: **else**
28: $\lambda \leftarrow \lambda * 10$
29: **if** $\lambda > 1e^{16}$ **then**
30: $break$
31: **end if**
32: $\chi^2 \leftarrow \chi^2_{old}$
33: **end if**
34: **end while**
35: **return** $parameters$

5 Ergebnisse

Im vorletzten Kapitel dieser Arbeit wird die Implementierung der Roboterkalibrierung auf den Prüfstand gestellt. Die ersten Testsubjekte sind die Schachbretterkennung und Optimierung, da beide Komponenten maßgeblichen Einfluss auf die Güte der Roboterkalibrierung haben. Die Roboterkalibrierung wird als Machbarkeitsuntersuchung zunächst simuliert und abschließend auf vier *NAO*s getestet.

5.1 Bildverarbeitung

Das Hauptaugenmerk der Tests für die Bildverarbeitung liegt auf der Beleuchtungstoleranz und der Genauigkeit der Eckenerkennung. Die Präzisionstests wurden mit *SimRobot* und die Beleuchtungstests mit einem *NAO* durchgeführt.

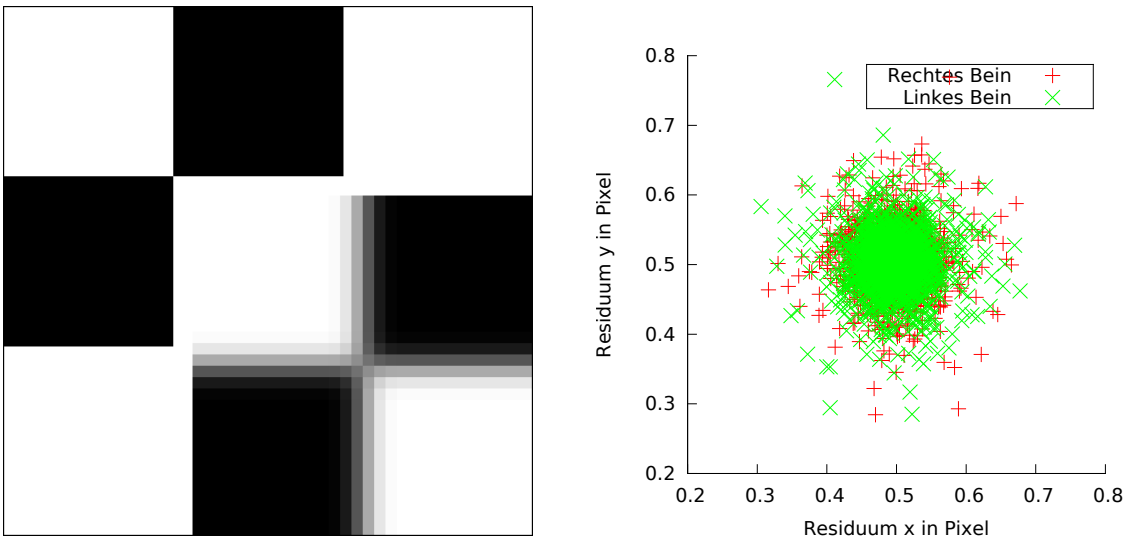
5.1.1 Testfall 1 - Präzision

Als Maß für die Genauigkeit der Eckenerkennung wird der Abstand der Projektion einer Ecke von der erkannten Ecke im Kamerabild herangezogen. Dieser Test wird mit einem simulierten *NAO* durchgeführt, wobei dieser nicht von Fehlern in den Gelenke und der Kamera betroffen ist.

Es wird nun ein normaler Kalibrierungsdurchlauf simuliert, d. h. es werden pro Bein unter 24 verschiedenen Beinstellungen mit jeweils drei verschiedenen Kameraposen ein Schachbrett betrachtet. Bei 24 Ecken pro Schachbrett ergibt das insgesamt 3456 einzelne Messungen. Das Ergebnis wird in Abbildung 5.1 (b) dargestellt. Für die simulierten Tests wurde eine Schachbrett-Textur angefertigt, welche zusätzlich weichgezeichnet wurde, um reale Kamerabilder zu simulieren. Es wurde ein Durchschnittsfilter wie in Abbildung 5.1(a) dargestellt verwendet.

Das quadrierte Mittel ∅ der Residuen beträgt in x-Richtung 0,497457 und in y-Richtung 0,505985 bei einer Standardabweichung σ 0,0384107 Pixel für die x- bzw. 0,0411112 Pixel für die y-Richtung. Die Werte der Standardabweichung sind, im Gegensatz zum Durchschnitt ∅, gering. Aufgrund der symmetrischen Abweichung wird der ∅ als systematischer Fehler gewertet. Für die simulierte Roboterkalibrierung ∅ von den Messungen abgezogen.

Trotz des Wertes für ∅ kann die Eckenerkennung als äußerst präzise angesehen werden,



(a) (b)

Abbildung 5.1: (a) Eine Textur einer Schachbrett-Ecke vor (oben links) und nach dem Weichzeichnen (unten rechts). (b) Die Residuen der visuellen Messung und Projektion (in Pixel): $\varnothing_x$ $0,497457 \pm 0,0384107$ und $\varnothing_y$ $0,505985 \pm 0,0411112$.

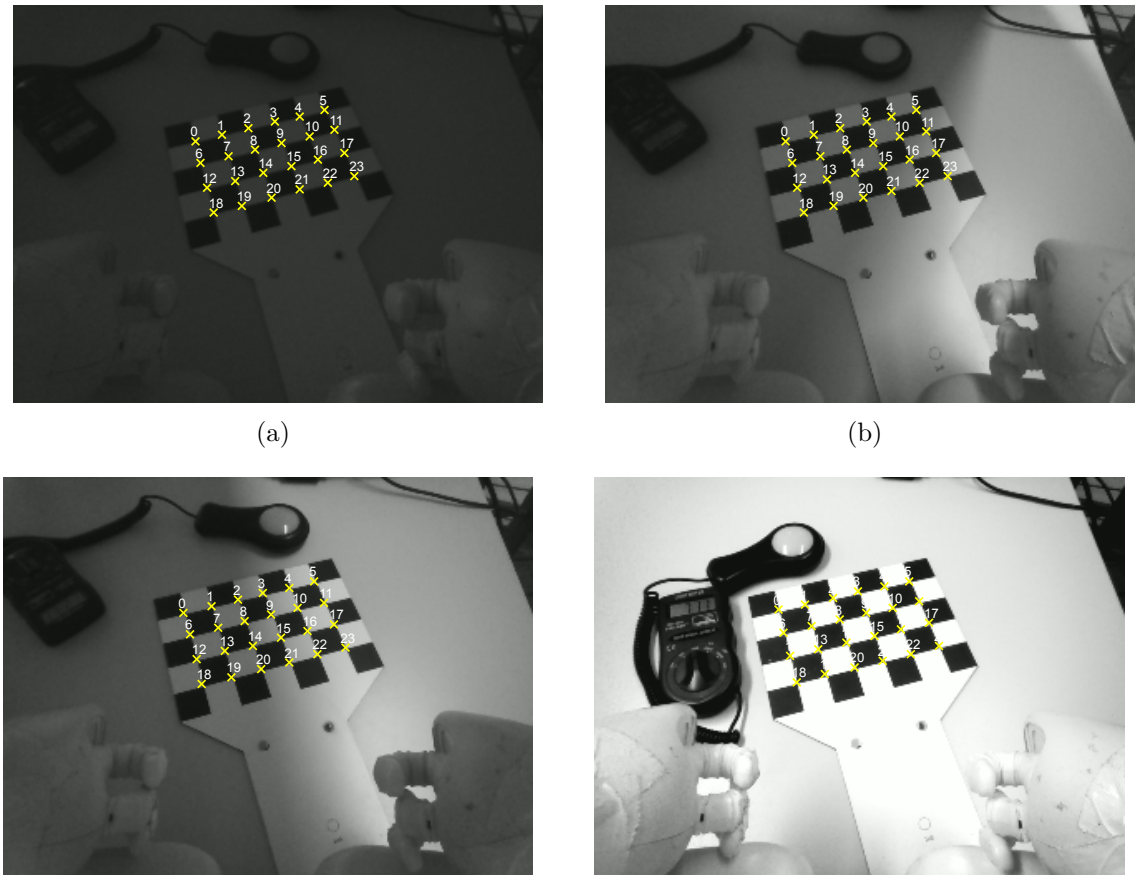
da fast alle Resiuden innerhalb der dreifachen Standardabweichung liegen.

5.1.2 Testfall 2 - Beleuchtungstoleranz

Die Schachbretterkennung muss robust gegen Beleuchtungsveränderungen sein, da aufgrund der Wechsel der Beinstellungen während der Kalibrierung die Schachbretter verschiedenartig beleuchtet werden.

Es wurde überprüft, ob der Schachbretterkenner mit einem festgelegten Parametersatz, unter verschiedenen Beleuchtungsstärken das Schachbrett erkennt (siehe Abbildung 5.1.2). Im ersten Fall war die künstliche Beleuchtung im Labor von *B-Human* ausgeschaltet, so dass nur das schwache natürliche Licht von außerhalb des Labors Einfluss nimmt. In zwei weiteren Fällen war die künstliche Beleuchtung teilweise eingeschaltet, wobei partiell Schatten auf das Schachbrett geworfen wurde. Im letzten Fall war die komplette Raumbeleuchtung eingeschaltet und das gesamte Schachbrett beleuchtet. Die Lichtstärke wurde bei allen Testfällen mit einem Luxmeter gemessen.

Der Abbildung 5.1.2 kann entnommen werden, dass in allen Fällen alle Ecken gefunden und korrekt durchnummeriert wurden. Anhand dieser Ergebnisse wird festgestellt, dass die Schachbretterkennung tolerant gegenüber Beleuchtungsveränderungen ist. Dass diese Ergebnisse auch auf die Prozedur der automatischen Roboterkalibrierung anwendbar



(a) (b)

(c) (d)

Abbildung 5.2: (a) Licht aus (45 Lux). (b) Zwei unterschiedliche Intensitäten (105 Lux im dunklen Bereich). (c) Ebenfalls zwei unterschiedliche Beleuchtungen (290 Lux im hellen Bereich). (d) Alle Lichtquellen sind eingeschaltet (711 Lux).

sind, wird implizit durch die Kalibrierungstests mit *NAO*s aufgezeigt, da bei diesen fast alle Schachbretter erfolgreich erkannt wurden.

5.2 Optimierer

Um die Implementierung des *Levenberg-Marquardt*-Algorithmus zu testen, werden zwei Standard-Optimierungsszenarien betrachtet. Im ersten Szenario soll das globale Minimum der *Rosenbrock*-Funktion gefunden werden (Testfall 3 - Rosenbrock-Funktion). Im zweiten werden die Parameter einer Kurvenfunktion anhand von Messdaten optimiert (Testfall 4 - Ausgleichskurve). Die Tests sollen die Funktionsfähigkeit der Implementierung des *Levenberg-Marquardt*-Algorithmus aufzeigen und stehen in keinem direkten

Zusammenhang zur automatischen Roboterkalibrierung.

5.2.1 Testfall 3 - Rosenbrock-Funktion

Die *Rosenbrock-Funktion* (siehe Abbildung 5.3) eignet sich als Testfall für Optimierungsalgorithmen, da diese über ein globales Minimum an der Stelle $(1, 1)$ verfügt. Dieses Minimum liegt innerhalb eines schmalen, bogenförmigen Tals, welches die eigentliche Herausforderung für die Algorithmen darstellt. Die Funktion hat folgende Gestalt:

$$f(x, y) = (1 - x)^2 + D * (y - x^2)^2. \tag{5.1}$$

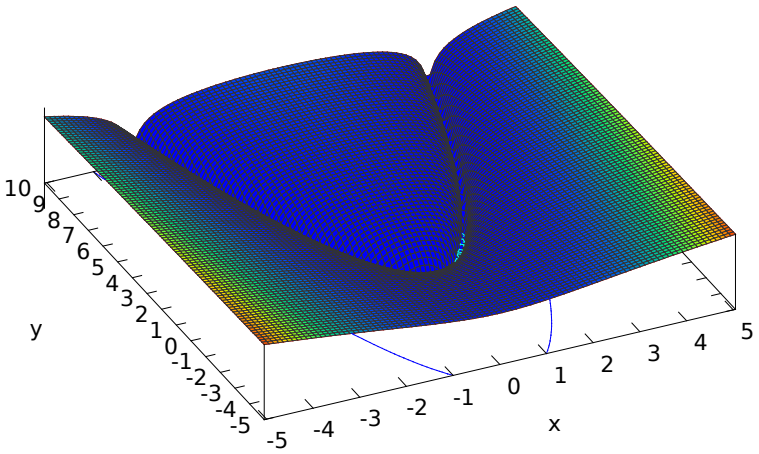


Abbildung 5.3: Die Rosenbrock-Funktion mit $D = 100$ und dem globalen Minimum an der Stelle (1,1).

Die Funktion muss noch in ein nichtlineares Optimierungsproblem umgewandelt werden, damit der *Levenberg-Marquardt*-Algorithmus angewendet werden kann. Von Lampton[Lam97] wird hierzu der Residuenvektor r folgendermaßen aufgestellt:

$$r = \begin{pmatrix} D * (y - x^2) \\ 1 - x \end{pmatrix}. \tag{5.2}$$

Das nun zu lösende Problem ergibt:

$$\arg\min_{x,y} ||r||_2^2. \tag{5.3}$$

In Tabelle 5.2.1 die Testfälle aufgelistet. Die Spalten D, x_0 und y_0 beinhalten jeweils die variierenden Werte für die Kostenfunktion r. Die Startbelegungen der Parameter

x und y werden mit x_0 und y_0 und die optimierten Parameter mit x_I und y_I bezeichnet. Der Faktor D ist pro Testfall konstant und erschwert laut Lampton[Lam97] das Problem dahingehend, dass bei steigenden Werten für D mehr Iterationsschritte benötigt werden. Die Spalte I beinhaltet die Anzahl an Iterationsschritten bis zu finalen Konvergenz.

Obwohl r analytisch ableitbar ist, wird die numerische Approximation der ersten Ableitungen aus Gleichung 4.35 verwendet wird. Die Belegung für Δ (siehe Formel 4.35) ist auf $0,0001$ festgelegt. Außerdem wird ein Toleranzwert von $1e^{-10}$ für die relative Änderung von χ^2 zwischen der aktuellen und vorherigen Iteration festgelegt.

	D	x_0	y_0	x_I	y_I	I
1	10	-1,2	1	1	1	22
2	100	-1,2	1	1	1	134
3	1000	-1,2	1	1	1	715
4	10	-2	0,7	1	1	18
5	100	-2	0,7	1	1	135
6	1000	-2	0,7	1	1	662
7	10	0	0	1	1	13
8	100	0	0	1	1	86
9	1000	0	0	1	1	443

Tabelle 5.1: Testfälle der Rosenbrock-Funktion.

In allen Testfällen konnte das globale Minimum gefunden werden, wobei die Anzahl der Iterationen I mit steigendem Wert für D erwartungsgemäß anstieg.

5.2.2 Testfall 4 - Ausgleichskurve

Bei dem folgenden Testfall soll anhand eines Datensatzes eine Ausgleichskurve bestimmt werden. Es gilt eine plausible Belegung für bestimmte Parameter einer Zielfunktion zu finden, so dass die gemessenen Datenpunkte möglichst nah an den Graphen der Funktionen liegen.

Die Daten dieses Testfalls entspringen einer *Gnuplot*-Dokumentation[1], wobei zu der Bedeutung der dort angegebenen Messungen (siehe Tabelle 5.2.2) keine Aussage existiert. Die folgende Funktion $f(x)$ soll anhand von fünf Parametern so angepasst werden, dass sie für die gemessenen Daten gut erklärt:

$$f(x) = \theta + a * exp(-t/\tau) * sin(2 * \pi * \frac{t}{T} + \phi). \tag{5.4}$$

[1] `http://www.cs.hmc.edu/~vrable/gnuplot/using-gnuplot.html`

x	y	x	y	x	y
0,0	-14,7	12,9	-15,0	26,0	-9,9
1,0	8,6	13,8	-1,6	27,0	5,8
2,1	28,8	14,9	19,5	28,0	14,7
3,1	46,7	15,9	27,5	29,0	21,8
4,2	47,4	17,0	32,6	30,0	29,8
5,2	36,5	17,9	27,5	31,0	21,4
6,2	37,0	18,9	20,2	32,0	24,6
7,2	5,1	20,0	13,8	32,9	25,8
8,2	-11,2	21,0	-1,3	33,8	0,6
9,1	-22,4	22,0	-24,5	34,7	-16,6
10,0	-35,5	23,0	-25,0	35,7	-24,0
11,0	-33,6	24,0	-25,0	36,6	-24,6
12,0	-21,1	25,0	-20,2	37,7	-19,8

Tabelle 5.2: Die Messungen.

Die einzelnen Residuenwerte werden wieder aus der Differenz der Messung und der Funktionsauswertung gebildet:

$$r_i = y_i - f(x_i),\ i \in \{0, \ldots, n\}. \tag{5.5}$$

Das nichtlineare Optimierungsproblem ergibt sich dann folgendermaßen:

$$\underset{a,\tau,\phi,T,\theta}{\arg\min} \sum_{i=0}^{n} r_i^2. \tag{5.6}$$

In Tabelle 5.2.2 werden die Ergebnisse der Optimierung im Vergleich zu *Gnuplot* (Subskript *gnu*), sowie die Belegung der initialen Parameter (Subskript 0) dargestellt. Beide

a_0	τ_0	ϕ_0	T_0	θ_0	χ_0^2	σ_0	-
40	15	-0,5	15	10	19588,9	22,4116	-
a	τ	ϕ	T	θ	χ^2	σ	I
45,5251	57,8277	-0,400942	13,1094	3,02182	819,255	4,58329	10
a_{gnu}	τ_{gnu}	ϕ_{gnu}	T_{gnu}	θ_{gnu}	χ_{gnu}^2	σ_{gnu}	I_{gnu}
45,5316	57,796	-0,401081	13,1091	3,02194	819,256	4,58329	13

Tabelle 5.3: Die Ergebnisse für die Parameterbelegung der Ausgleichskurve.

Parametersätze erzeugen einen identischen Standardabweichung von 4.58329, wobei die Implementierung im Vergleich zu *Gnuplot* einen marginal kleineren χ^2 erzeugte und dafür zehn anstatt dreizehn Iterationen benötigte. Diese Ergebnisse zeigen auf, dass die Implementierung mit der von *Gnuplot* mithalten kann und diese sogar noch minimal

bessere Ergebnisse erzielt. In Abbildung 5.4 wird der Graph der Funktion mit einer initialen Parameterbelegung dargestellt. Zusätzlich sind auch die Messpunkte eingezeichnet. Die optimierten Parameter erzeugen den Graph aus Abbildung 5.5, aus welcher ersichtlich wird, dass der Graph besser zu den Messungen passt.

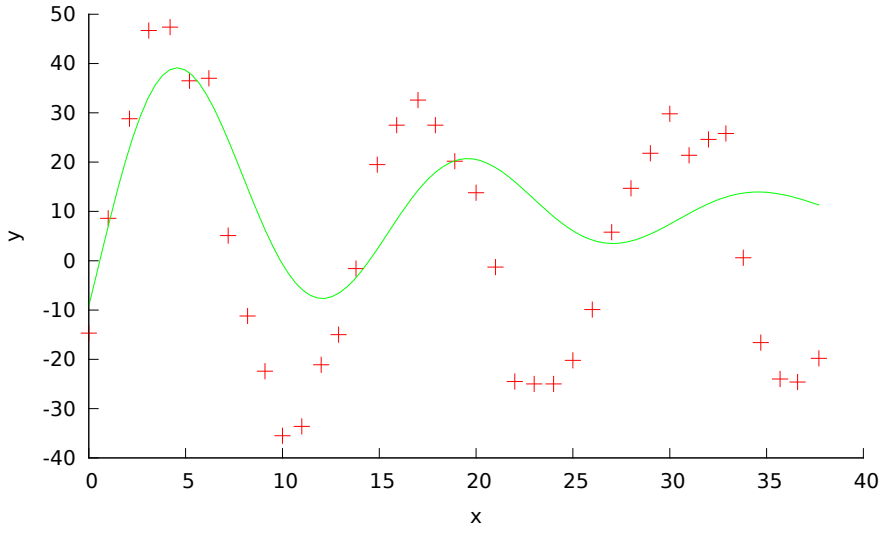


Abbildung 5.4: Die Messungen und der Funktionsgraph mit initialer Parameterbelegung.

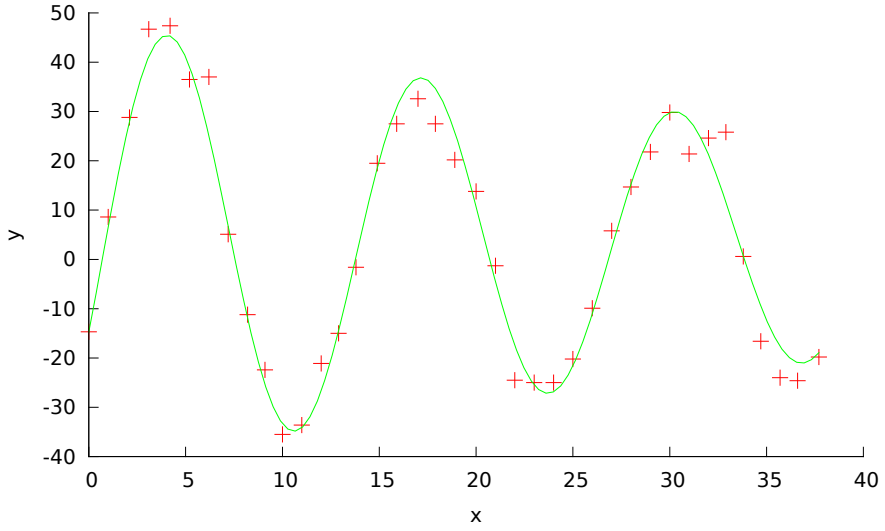


Abbildung 5.5: Der Graph der optimierten Funktion.

5.3 Simulierte Roboterkalibrierung

Als erste Machbarkeitsuntersuchung wurde die Roboterkalibrierung mit Hilfe der Simulationssoftware *SimRobot* durchgeführt. Die Simulation ermöglicht es, Fehler für die Gelenke und die Kamera einzustellen.

Insgesamt wurden drei Testfälle erstellt, wobei für alle eine maximale Anzahl von Iterationen auf 500 festgesetzt wurde. Des weiteren terminiert die Optimierung, wenn der Unterschied von χ^2 zwischen zwei Iterationen insgesamt viermal unter $0,0001$ gefallen ist.

Der erste Testfall (Testfall 5 - Kein Arretierungsfehler) wird mit perfekt positionierten Schachbrettsandalen durchgeführt. Im zweiten Testfall (Testfall 6 - Alle Parameter fehlerbehaftet) sind alle Parameter des Projektionsmodells fehlerbehaftet. Beim abschließenden simulierten Testfall (Testfall 7 - Partielle Fehlerbelegung) sind nur bestimmte Parameter mit Fehlern versehen worden.

Jeder simulierte Testfall verfügt über eine Tabelle, welche den manuell festgelegten Fehler eines Gelenks ϵ, den optimierten Parameter α, die Abweichung von α zu ϵ und die Standardabweichung σ von α beinhaltet. In einer weiteren Tabelle werden die Anzahl getätigter Messungen, die benötigten Iterationen des Optimierers, die Summe der quadrierten Residuen vor und nach der Kalibrierung, sowie das quadrierte Mittel des Resiuden mit optimierten Parametern aufgelistet. In zwei Grafiken werden die Residuen, jeweils für das linke und rechte Bein, nach der Datensammlungs- und Optimierungsphase dargestellt. Es sind maximal 3456 Messungen möglich ($24\ (Ecken) * 24\ (Beinstellungen) * 2(Beine) * 3\ (Kopfstellungen) = 3456$).

	M	I	$\chi_0^2\ [px]$	$\chi_I^2\ [px]$	$\sigma_x\ [px]$	$\sigma_y\ [px]$
Testfall 5	3456	159	105220	14,6177	0,041338	0,0500075
Testfall 6	3456	248	16057,2	10,3446	0,0396921	0,0375641
Testfall 7	3408	302	932488	10,9303	0,0400293	0,0405593

Tabelle 5.4: Testfall 1 - Kein Arretierungsfehler: Getätigte Messungen M, benötigte Iterationen I, Fehler χ_0^2 vor der Optimierung, Fehler χ_I^2 nach der Optimierung und die Standardabweichung σ_x und σ_y der Residuen in x- und y-Richtung.

5.3.1 Testfall 5 - Kein Arretierungsfehler

Im ersten Testfall sind alle Gelenke der Kinematik der Beine sowie die Rotation der unteren Kamera mit individuellen Fehlern versehen. Außer Acht werden hier allerdings Fehler bei der Arretierung des Schachbrettes gelassen. Dieser Test wird als Erster

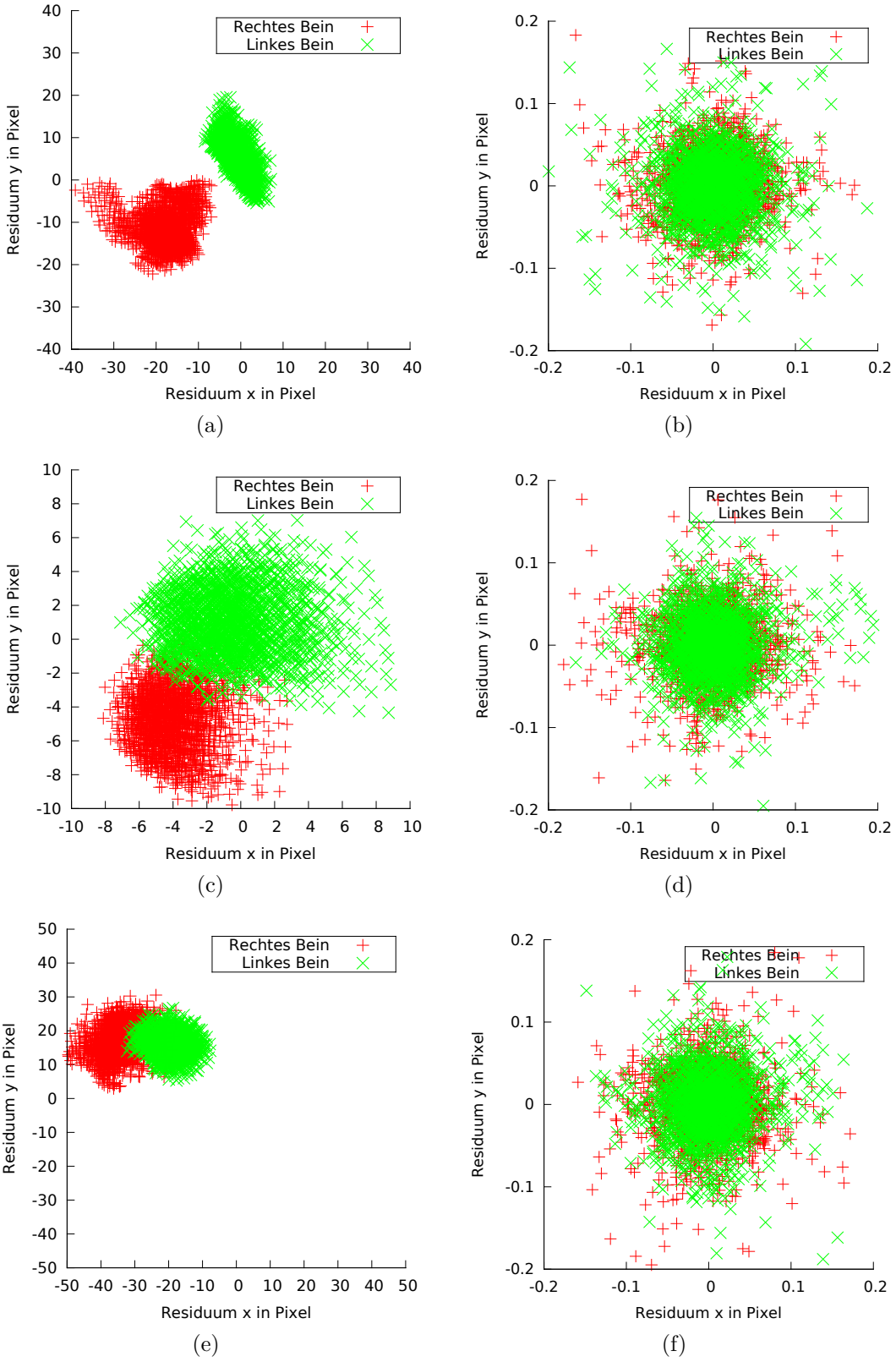


Abbildung 5.6: Verteilung der Residuen vor (links) und nach der Optimierung (rechts): Testfall 5 (a) und (b), Testfall 6 (c) und (d) und Testfall 7 (e) und (f).

durchgeführt, um zu überprüfen, inwiefern die Parameter für die Schachbrettsandale angepasst werden.

In Abbildung 5.6(a) wird die Verteilung der Resiuden direkt nach der Datensammlungsphase dargestellt. Es ist ersichtlich, dass die Fehler der Gelenke für beide Beine eine unterschiedliche Ausprägung aufweisen.

In Tabelle 5.5 werden die optimierten Parameter aufgelistet. Alle gesetzten Fehler wurden sehr genau gefunden, was durch die Spalten $\alpha - \epsilon$ ersichtlich wird. Außerdem wurden alle möglichen Schachbretter erkannt, was die Anzahl der Messungen M von 3456 zeigt.

Dass die optimierten Parameter die Messungen besser vorhersagen, wird durch Abbildung 5.6(b) visualisiert, wobei zu beachten ist, dass der Wertebereich der beiden Koordinatenachsen sehr klein ist. Die Residuen sind nach der Optimierung mit einer sehr geringen Standardabweichung normalverteilt (siehe Tabelle 5.4). Fast alle Residuen liegen im Bereich der dreifachen Standardabweichung.

Die Parameter der Schachbrettsandale wurden um den Bruchteil eines Millimeters angepasst. Insgesamt benötigte der Optimierer 159 Iterationen um zu konvergieren und konnte χ^2 um das 7206-fache minimieren.

5.3.2 Testfall 6 - Alle Parameter fehlerbehaftet

Im zweiten Testfall sind alle möglichen Fehler vorhanden. Die Fehler der Rotation der Kamera sind im Vergleich zum ersten Testfall geringer. Die Schachbrettsandalen sind um wenige Millimeter verschoben und verfügen über einen geringen Rotationsfehler.

Auch in diesem Testfall war es dem Optimierer möglich, alle relevanten Parameter (siehe Tabelle 5.6) mit großer Präzision zu finden. Lediglich bei den translativen Parametern in y-Richtung des Schachbrettes gibt es eine größere Abweichung von jeweils ca. einem Millimeter. Allerdings sind die Kamera- und Kinematikparameter gefunden worden, was die eigentliche Aufgabe des gesamten Verfahrens ist. Der Optimierer konvergierte nach 248 Iterationen und konnte dabei aus den vollen 3456 möglichen Messungen schöpfen.

Die Verteilung der Fehler nach der Optimierung (siehe Abbildung 5.6(d)) ist sehr ähnlich zu der aus dem ersten Testfall, wobei die Standardabweichung des zweiten Testfalls minimal geringer ausfällt. Obwohl χ_0^2 deutlich kleiner ist als sein Pendant im ersten Testfall, brauchte das Verfahren 89 Iterationen mehr, um zu konvergieren (siehe Tabelle 5.4).

5.3.3 Testfall 7 - Partielle Fehlerbelegung

Im letzten simulierten Testfall sind nur neun Komponenten des Projektionsmodells mit einem Fehler versehen worden. Speziell im rechten Bein ist das erste Hüft- sowie das Kniegelenk fehlerbehaftet, wobei der Fehler für das Knie mit 5.1° vergleichsweise groß ausfällt.

Die optimierten Parameter passen wieder sehr präzise zu den manuell eingestellten Fehlern. Dieser Testfall hat die meisten Iterationen (insgesamt 302 Iterationen) in Anspruch genommen und erzeugte den größten χ_0^2 mit 932488 Pixel.

Auch der letzte Testfall erzeugt nach der Optimierung normalverteilte Residuen (siehe Abbildung 5.6(f)) mit sehr geringer Standardabweichung (siehe Tabelle 5.4), ähnlich zu den vorherigen simulierten Testfällen.

Auffällig in diesem Testfall ist, dass nicht alle möglichen Messungen getätigt werden konnten. Dies lag daran, dass das Schachbrett aufgrund der eingestellten Fehler bei zwei Gelenkstellungen vom Oberschenkel des *NAO*s teilweise verdeckt wurde. Bemerkenswert ist außerdem die Abweichung der Translation in y-Richtung des rechten Schachbrettes bei acht Millimetern liegt (siehe Tabelle 5.7).

Parameter	ϵ	α	$\alpha - \epsilon$	σ_α
$\alpha_{CameraYaw}$ [°]	-1,7	-1,7007	-0,0007	0,0099
$\alpha_{CameraPitch}$ [°]	0,7	0,7004	0,0004	0,0084
$\alpha_{CameraRoll}$ [°]	-1,1	-1,1001	-0,0001	0,0117
$\alpha_{LeftHipYawPitch}$ [°]	1,9	1,9019	0,0019	0,0120
$\alpha_{LeftHipRoll}$ [°]	2,1	2,1004	0,0004	0,0130
$\alpha_{LeftHipPitch}$ [°]	1,7	1,6882	-0,0118	0,0146
$\alpha_{LeftKneePitch}$ [°]	-1,1	-1,0790	0,0210	0,0157
$\alpha_{LeftAnklePitch}$ [°]	-3,3	-3,3089	-0,0089	0,0153
$\alpha_{LeftAnkleRoll}$ [°]	-2,7	2,6972	-0,0028	0,0198
$\alpha_{RightHipYawPitch}$ [°]	1,8	1,7978	-0,0022	0,0122
$\alpha_{RightHipRoll}$ [°]	-0,9	-0,9024	-0,0024	0,0136
$\alpha_{RightHipPitch}$ [°]	-0,7	-0,6756	0,0244	0,0142
$\alpha_{RightKneePitch}$ [°]	-1,1	-1,1407	-0,0407	0,0160
$\alpha_{RightAnklePitch}$ [°]	-3,3	-3,2858	0,0142	0,0156
$\alpha_{RightAnkleRoll}$ [°]	-1,7	-1,7001	-0,0001	0,0198
$\alpha_{LeftPatternRotZ}$ [°]	0	0	0	0,0161
$\alpha_{LeftPatterTransX}$ $[mm]$	0	0,0044	0,0044	0,0750
$\alpha_{LeftPatterTransY}$ $[mm]$	0	0,0035	0,0035	0,0826
$\alpha_{RightPatternRotZ}$ [°]	0	0	0	0,0163
$\alpha_{RightPatterTransX}$ $[mm]$	0	0,0006	0,0006	0,0705
$\alpha_{RightPatterTransY}$ $[mm]$	0	-0,0066	-0,0066	0,0845

Tabelle 5.5: Testfall 5 - Kein Arretierungsfehler: Parameter α, Fehler ϵ, Abweichung $\alpha - \epsilon$ und die Standardabweichung der Parameter σ.

Parameter	ϵ	α	$\alpha - \epsilon$	σ_α
$\alpha_{CameraYaw}$ $[^\circ]$	0,1	0,1005	0,0005	0,0098
$\alpha_{CameraPitch}$ $[^\circ]$	0,2	0,2000	0	0,0077
$\alpha_{CameraRoll}$ $[^\circ]$	-0,6	-0,5995	0,0005	0,0115
$\alpha_{LeftHipYawPitch}$ $[^\circ]$	0,3	0,3004	0,0004	0,0120
$\alpha_{LeftHipRoll}$ $[^\circ]$	-1,1	-1,1019	-0,0019	0,0133
$\alpha_{LeftHipPitch}$ $[^\circ]$	0,2	0,1945	-0,0055	0,0131
$\alpha_{LeftKneePitch}$ $[^\circ]$	-1,1	-1,0891	0,0109	0,0145
$\alpha_{LeftAnklePitch}$ $[^\circ]$	1,3	1,2959	-0,0041	0,0142
$\alpha_{LeftAnkleRoll}$ $[^\circ]$	-0,7	-0,6975	0,0025	0,0189
$\alpha_{RightHipYawPitch}$ $[^\circ]$	0,1	0,0984	-0,0016	0,0119
$\alpha_{RightHipRoll}$ $[^\circ]$	0,9	0,8987	-0,0013	0,0127
$\alpha_{RightHipPitch}$ $[^\circ]$	-1,7	-1,6824	0,0176	0,0142
$\alpha_{RightKneePitch}$ $[^\circ]$	1,1	1,0700	-0,0300	0,0161
$\alpha_{RightAnklePitch}$ $[^\circ]$	0,3	0,3114	0,0114	0,0145
$\alpha_{RightAnkleRoll}$ $[^\circ]$	0,2	0,2007	0,0007	0,0197
$\alpha_{LeftPatternRotZ}$ $[^\circ]$	0,3	0,2987	-0,013	0,0147
$\alpha_{LeftPatterTransX}$ $[mm]$	2	2,0041	0,0041	0,0668
$\alpha_{LeftPatterTransY}$ $[mm]$	1	0,1092	-0,8008	0,0783
$\alpha_{RightPatternRotZ}$ $[^\circ]$	0,4	0,4001	0,0001	0,0138
$\alpha_{RightPatterTransX}$ $[mm]$	1	1,0036	0,0036	0,0736
$\alpha_{RightPatterTransY}$ $[mm]$	-1	-2,2004	-1,2004	0,0804

Tabelle 5.6: Testfall 6 - Alle Parameter fehlerbehaftet: Parameter α, Fehler ϵ, Abweichung $\alpha - \epsilon$ und die Standardabweichung der Parameter σ.

Parameter	ϵ	α	$\alpha - \epsilon$	σ_α
$\alpha_{CameraYaw}$ [°]	-3	-3,0003	-0,0003	0,0098
$\alpha_{CameraPitch}$ [°]	0	0,0002	0,0002	0,0072
$\alpha_{CameraRoll}$ [°]	0	0,0003	0,0003	0,0110
$\alpha_{LeftHipYawPitch}$ [°]	3,5	3,5019	0,0019	0,0123
$\alpha_{LeftHipRoll}$ [°]	0	-0,0002	-0,0002	0,0131
$\alpha_{LeftHipPitch}$ [°]	-0,1	-0,0940	0,0060	0,0142
$\alpha_{LeftKneePitch}$ [°]	0,1	0,0845	-0,0155	0,0144
$\alpha_{LeftAnklePitch}$ [°]	0	0,0080	0,0080	0,0138
$\alpha_{LeftAnkleRoll}$ [°]	-3,4	-3,4047	-0,0047	0,0189
$\alpha_{RightHipYawPitch}$ [°]	-0,2	-0,1984	0,0016	0,0123
$\alpha_{RightHipRoll}$ [°]	0	-0,0012	-0,0012	0,0121
$\alpha_{RightHipPitch}$ [°]	0	0,0298	0,0298	0,0130
$\alpha_{RightKneePitch}$ [°]	5,1	5,0423	-0,0577	0,0139
$\alpha_{RightAnklePitch}$ [°]	0	0,0257	0,0257	0,0148
$\alpha_{RightAnkleRoll}$ [°]	0	-0,0063	-0,0063	0,0191
$\alpha_{LeftPatternRotZ}$ [°]	0	0	0	0,0128
$\alpha_{LeftPatterTransX}$ $[mm]$	0	0,0061	0,0061	0,0671
$\alpha_{LeftPatterTransY}$ $[mm]$	0	-0,0005	-0,0005	0,0790
$\alpha_{RightPatternRotZ}$ [°]	2,7	2,7012	0,0012	0,0157
$\alpha_{RightPatterTransX}$ $[mm]$	0	0,1881	0,1881	0,0735
$\alpha_{RightPatterTransY}$ $[mm]$	-5	-13,0143	-8,0143	0,0803

Tabelle 5.7: Testfall 7 - Partielle Fehlerbelegung: Parameter α, Fehler ϵ, Abweichung $\alpha - \epsilon$ und die Standardabweichung der Parameter σ.

5.4 Roboterkalibrierung des *NAO*s

Die folgenden Testfälle befassen sich mit der automatischen Roboterkalibrierung von *NAO*s. Die in Abschnitt 1.4 definierten Anforderungen werden hier auf die Probe gestellt. Neben der Kalibrierung wird getestet, wie viele von den insgesamt 3456 möglichen Messungen getätigt werden konnten, sowie die Dauer einer kompletten Kalibrierung in Sekunden. Die Zeitmessung startet mit dem Beginn der Datensammlung und endet mit der Konvergenz des Optimierers.

Die Abbildungen 5.7 und 5.8 stellen hier, wie bei den simulierten Tests, die Verteilung der Residuen vor und nach der Optimierung dar. Es werden immer alle Parameter aus Tabelle 4.1 verwendet, wobei die Ergebnisse der Optimierung in einer Tabelle mit der jeweiligen asymptotischen Standardabweichung der Parameter aufgelistet sind. In Tabelle 5.8 sind die Anzahl der getätigten Messungen, die benötigten Iterationen bis zur Konvergenz, die Dauer der Kalibrierung, χ^2 vor und nach der Kalibrierung, sowie die Standardabweichung der Residuen nach der Optimierung abgedruckt.

5.4.1 Testfall 8 - Leslie, Kripke, Wil Wheaton und Mrs. Wolowitz

Zum Zeitpunkt der Durchführung der Tests befand sich das Team *B-Human* in den Vorbereitungen für die *RoboCup - German Open 2014*[2]. Für die Tests wurden die Roboter Leslie, Kripke, Wil Wheaton und Mrs. Wolowitz für die Kalibrierung ausgewählt, wobei Leslie als einziger unbeschädigt war.

Insgesamt wurde die Kalibrierung dreimal pro Roboter durchgeführt, um zu überprüfen, ob das Verfahren jedesmal identische, plausible Lösungen findet. Zur Überprüfung wird für jeden Roboter der Durchschnitt und die Standardabweichung der Parameter der drei Durchgänge berechnet.

Die optimierten Parameter für Leslie sind in Tabelle 5.9, für Kripke in Tabelle 5.10, für Wil Wheaton in Tabelle 5.11 und für Mrs. Wolowitz in 5.12 aufgelistet.

In den Abbildungen 5.9 und 5.10 wird gezeigt, wie sich die optimierten Parameter auf die Projektionsfunktion auswirken. Auf den Abbildungen auf der linken Seite wird die Projektion des Schachbrettes in das Kamerabild mit unkalibrierten *NAO*s dargestellt. Im Vergleich dazu wird in den Abbildungen auf der rechten Seite die Projektion mit optimierten Parametern gezeigt. Die Bilder wurden mit einer der 24 Gelenkstellungen aus der Datensammlungsphase aufgenommen. Die optimierten Parameter sorgen dafür, dass das eingeblendete Schachbrett deutlich besser zum Brett im Kamerabild passt.

[2] `http://www.robocupgermanopen.de/`

Leslie	M	I	$t\ [s]$	$\chi_0^2\ [px]$	$\chi_I^2\ [px]$	$\sigma_x\ [px]$	$\sigma_y\ [px]$
α_1	3456	68	611	265058	69282,8	2,72939	3,49119
α_2	3456	91	627	257342	69742,5	2,85152	3,47116
α_3	3456	57	577	265915	68566,4	2,77763	3,36615
Kripke	M	I	$t\ [s]$	$\chi_0^2\ [px]$	$\chi_I^2\ [px]$	$\sigma_x\ [px]$	$\sigma_y\ [px]$
α_1	3456	60	618	171221	85081,2	3,50402	3,51287
α_2	3456	34	531	188865	93573,6	3,60333	3,67955
α_3	3456	71	598	202753	86580,1	3,53368	3,50002
Wil Wheaton	M	I	$t\ [s]$	$\chi_0^2\ [px]$	$\chi_I^2\ [px]$	$\sigma_x\ [px]$	$\sigma_y\ [px]$
α_1	3456	37	548	2,32532e+06	62664,4	2,86071	3,15411
α_2	3456	49	567	2,15496e+06	66153,8	2,9547	3,22668
α_3	3456	51	572	2,08024e+06	65610	3,00121	3,15866
Mrs. Wolowitz	M	I	$t\ [s]$	$\chi_0^2\ [px]$	$\chi_I^2\ [px]$	$\sigma_x\ [px]$	$\sigma_y\ [px]$
α_1	3432	79	651	216965	106589	3,95383	3,92743
α_2	3456	43	580	219343	104099	3,84553	3,91577
α_3	3432	77	640	210118	99029,2	3,73625	3,85942

Tabelle 5.8: Testfall 8 - Leslie: Auflistung der getätigten Messungen M, der benötigten Iterationen I, die Dauer der Kalibrierung S, χ^2 vor und nach der Optimierung und die Standardabweichung der Residuen für die drei aufeinanderfolgenden Kalibrierung α_i, α_2, und α_3.

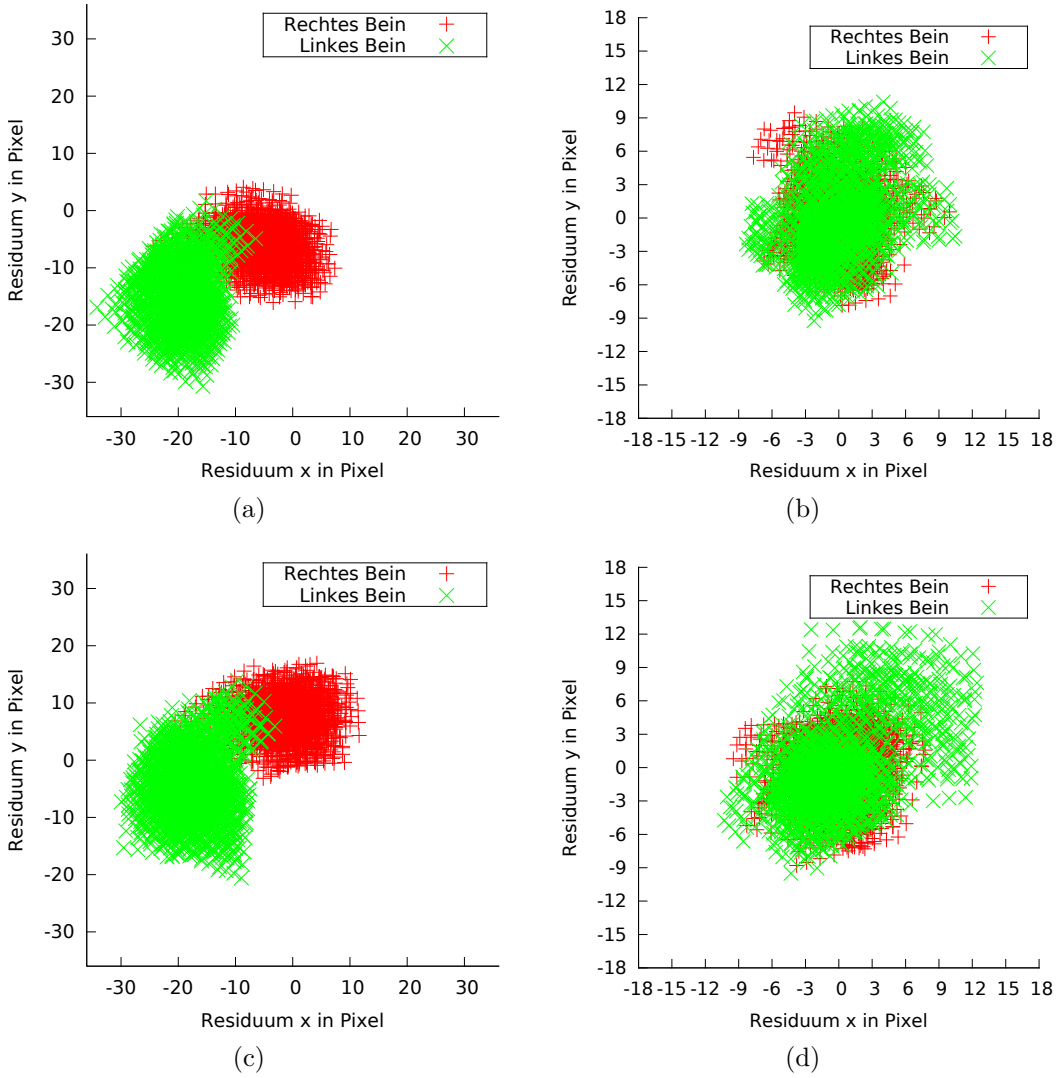


Abbildung 5.7: Testfall 8 - Verteilung der Residuen vor (links) und nach der Optimierung (rechts): Leslie ((a) und (b)) und Kripke ((c) und (d)).

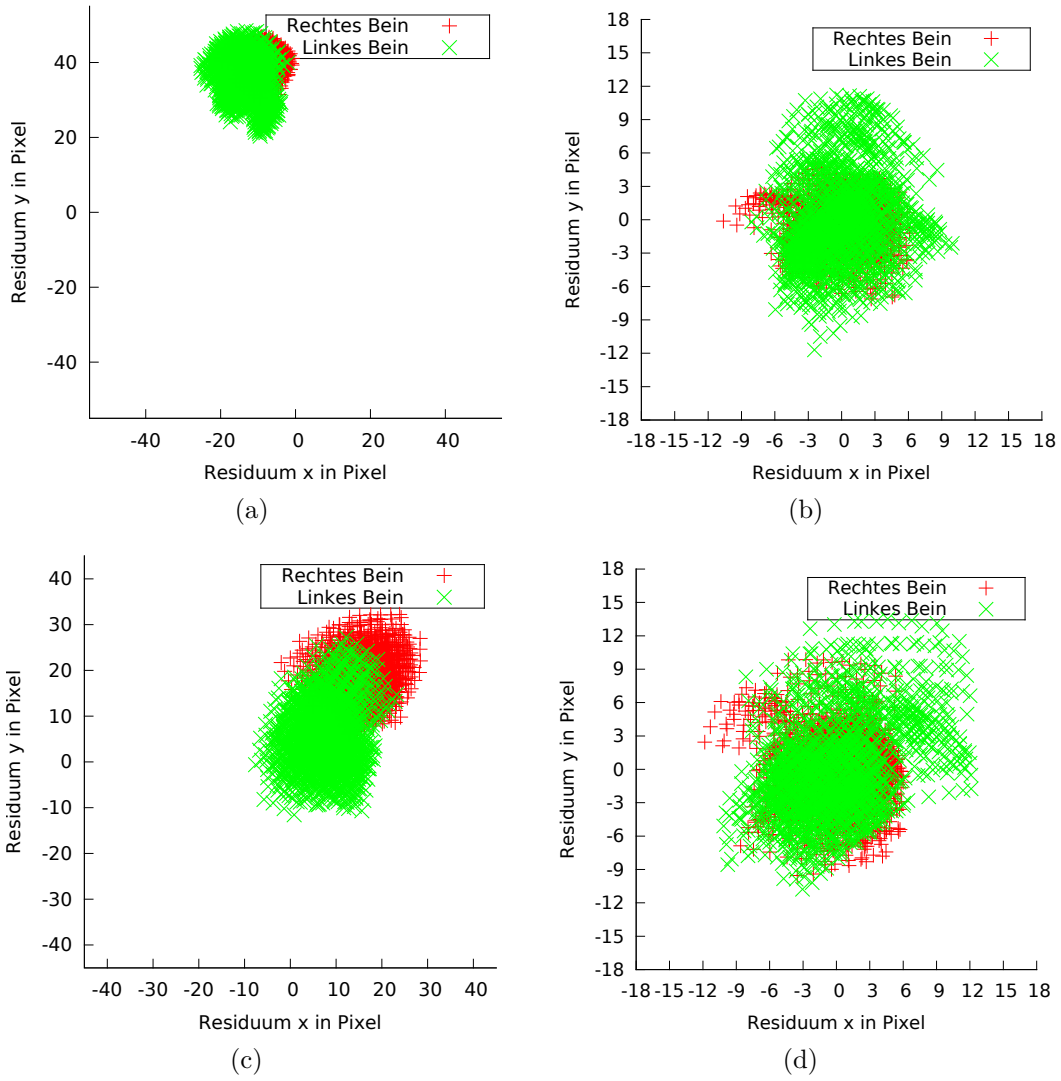


Abbildung 5.8: Testfall 8 - Verteilung der Residuen vor (links) und nach der Optimierung (rechts): Wil Wheaton ((a) und (b)) und Mrs. Wolowitz ((c) und (d)).

Leslie		α_1	α_2	α_3	$\varnothing_{\alpha_1,\alpha_2,\alpha_3}$
$\alpha_{CameraYaw}$ [°]	α	1,1001	1,1428	1,0602	1,1010 ± 0,0413
	σ	0,0096	0,0100	0,0097	
$\alpha_{CameraPitch}$ [°]	α	-2,1339	-2,2333	-2,3429	-2,2367 ± 0,1045
	σ	0,0066	0,0080	0,0077	
$\alpha_{CameraRoll}$ [°]	α	-0,5939	-0,8213	-0,6557	-0,6903 ± 0,1176
	σ	0,0115	0,0121	0,0114	
$\alpha_{LeftHipYawPitch}$ [°]	α	-0,8690	-0,8286	-0,8676	-0,8551 ± 0,0229
	σ	0,0128	0,0127	0,0120	
$\alpha_{LeftHipRoll}$ [°]	α	0,0322	-0,0076	0,0443	0,0230 ± 0,0272
	σ	0,0123	0,0135	0,0124	
$\alpha_{LeftHipPitch}$ [°]	α	-0,1931	-0,4365	-0,0145	-0,2147 ± 0,2118
	σ	0,0127	0,0134	0,0135	
$\alpha_{LeftKneePitch}$ [°]	α	-1,4950	-1,3234	-1,8046	-1,5410 ± 0,2439
	σ	0,0144	0,0147	0,0146	
$\alpha_{LeftAnklePitch}$ [°]	α	3,0142	3,0640	3,2248	3,1010 ± 0,1101
	σ	0,0146	0,0140	0,0146	
$\alpha_{LeftAnkleRoll}$ [°]	α	0,1700	0,4193	0,2086	0,2660 ± 0,1342
	σ	0,0183	0,0196	0,0182	
$\alpha_{RightHipYawPitch}$ [°]	α	1,4010	1,2788	1,6476	1,4425 ± 0,1879
	σ	0,0110	0,0122	0,0115	
$\alpha_{RightHipRoll}$ [°]	α	-0,6942	-0,7306	-0,7280	-0,7176 ± 0,0203
	σ	0,0132	0,0131	0,0124	
$\alpha_{RightHipPitch}$ [°]	α	1,1670	1,2930	1,3059	1,2553 ± 0,0767
	σ	0,0126	0,0131	0,0124	
$\alpha_{RightKneePitch}$ [°]	α	-1,5763	-1,6909	-1,7358	-1,6677 ± 0,0822
	σ	0,0142	0,0139	0,0136	
$\alpha_{RightAnklePitch}$ [°]	α	1,9737	1,8917	1,7711	1,8788 ± 0,1019
	σ	0,0144	0,0139	0,0132	
$\alpha_{RightAnkleRoll}$ [°]	α	-0,0089	0,2481	0,0939	0,1110 ± 0,1294
	σ	0,0183	0,0198	0,0191	
$\alpha_{LeftPatternRotZ}$ [°]	α	-0,6370	-0,4723	-0,6561	-0,5885 ± 0,1011
	σ	0,0145	0,0133	0,0139	
$\alpha_{LeftPatterTransX}$ $[mm]$	α	10,3281	10,3688	10,7984	10,4984 ± 0,2606
	σ	0,0686	0,0644	0,0651	
$\alpha_{LeftPatterTransY}$ $[mm]$	α	-1,5308	-2,0518	-1,5180	-1,7002 ± 0,3046
	σ	0,0768	0,0778	0,0798	
$\alpha_{RightPatternRotZ}$ [°]	α	1,6613	1,8847	2,0053	1,8504 ± 0,1745
	σ	0,0137	0,0147	0,0125	
$\alpha_{RightPatterTransX}$ $[mm]$	α	5,0401	4,5275	4,8732	4,8136 ± 0,2614
	σ	0,0666	0,0701	0,0668	
$\alpha_{RightPatterTransY}$ $[mm]$	α	0,3234	-0,0192	0,0543	0,1195 ± 0,1804
	σ	0,0770	0,0792	0,0739	

Tabelle 5.9: Testfall 8 - Leslie: Parameter α, Standardabweichung σ und Parameterdurchschnitt $\varnothing$ der drei Experimente.

Kripke		α_1	α_2	α_3	$\varnothing_{\alpha_1,\alpha_2,\alpha_3}$
$\alpha_{CameraYaw}$ [°]	α	0,9057	0,9652	0,8670	0,9126 ± 0,0495
	σ	0,0099	0,0099	0,0099	
$\alpha_{CameraPitch}$ [°]	α	-0,2309	-0,3348	-0,4242	-0,3300 ± 0,0967
	σ	0,0079	0,0081	0,0076	
$\alpha_{CameraRoll}$ [°]	α	-1,2924	-1,1958	-1,3045	-1,2642 ± 0,0596
	σ	0,0122	0,0122	0,0126	
$\alpha_{LeftHipYawPitch}$ [°]	α	-1,5297	-1,8063	-1,3865	-1,5742 ± 0,2134
	σ	0,0127	0,0117	0,0127	
$\alpha_{LeftHipRoll}$ [°]	α	0,3594	0,1920	0,2767	0,2760 ± 0,0837
	σ	0,0131	0,0130	0,0130	
$\alpha_{LeftHipPitch}$ [°]	α	-0,3761	0,0366	-0,1213	-0,1536 ± 0,2082
	σ	0,0134	0,0125	0,0129	
$\alpha_{LeftKneePitch}$ [°]	α	-2,6932	-3,1209	-2,9156	-2,9099 ± 0,2139
	σ	0,0156	0,0140	0,0159	
$\alpha_{LeftAnklePitch}$ [°]	α	3,3081	3,2649	3,3033	3,2921 ± 0,0237
	σ	0,0151	0,0146	0,0147	
$\alpha_{LeftAnkleRoll}$ [°]	α	0,7273	0,6524	0,4008	0,5935 ± 0,1710
	σ	0,0192	0,0194	0,0185	
$\alpha_{RightHipYawPitch}$ [°]	α	1,2053	1,1448	1,2920	1,2140 ± 0,0740
	σ	0,0121	0,0117	0,0129	
$\alpha_{RightHipRoll}$ [°]	α	-1,2742	-1,2501	-1,3272	-1,2838 ± 0,0394
	σ	0,0132	0,0135	0,0135	
$\alpha_{RightHipPitch}$ [°]	α	0,1569	-0,3864	-0,1203	-0,1166 ± 0,2717
	σ	0,0132	0,0131	0,0139	
$\alpha_{RightKneePitch}$ [°]	α	-1,7805	-1,3336	-1,6422	-1,5854 ± 0,2288
	σ	0,0148	0,0148	0,0149	
$\alpha_{RightAnklePitch}$ [°]	α	2,7550	2,8195	2,8599	2,8115 ± 0,0529
	σ	0,0141	0,0141	0,0137	
$\alpha_{RightAnkleRoll}$ [°]	α	-0,1573	0,0804	0,0281	-0,0163 ± 0,1249
	σ	0,0191	0,0192	0,0191	
$\alpha_{LeftPatternRotZ}$ [°]	α	-0,1938	-0,3226	-0,7157	-0,4107 ± 0,2719
	σ	0,0177	0,0156	0,0151	
$\alpha_{LeftPatterTransX}$ $[mm]$	α	9,6515	9,6134	8,5990	9,2880 ± 0,5970
	σ	0,0692	0,0648	0,0678	
$\alpha_{LeftPatterTransY}$ $[mm]$	α	-2,3913	-0,3268	-1,1910	-1,3030 ± 1,0368
	σ	0,0851	0,0804	0,0807	
$\alpha_{RightPatternRotZ}$ [°]	α	1,4993	1,2251	1,6024	1,4423 ± 0,1950
	σ	0,0161	0,0155	0,0165	
$\alpha_{RightPatterTransX}$ $[mm]$	α	5,3871	6,5507	6,0325	5,9901 ± 0,5830
	σ	0,0629	0,0625	0,0683	
$\alpha_{RightPatterTransY}$ $[mm]$	α	1,3767	2,1884	1,0779	1,5477 ± 0,5747
	σ	0.0806	0.0803	0.0852	

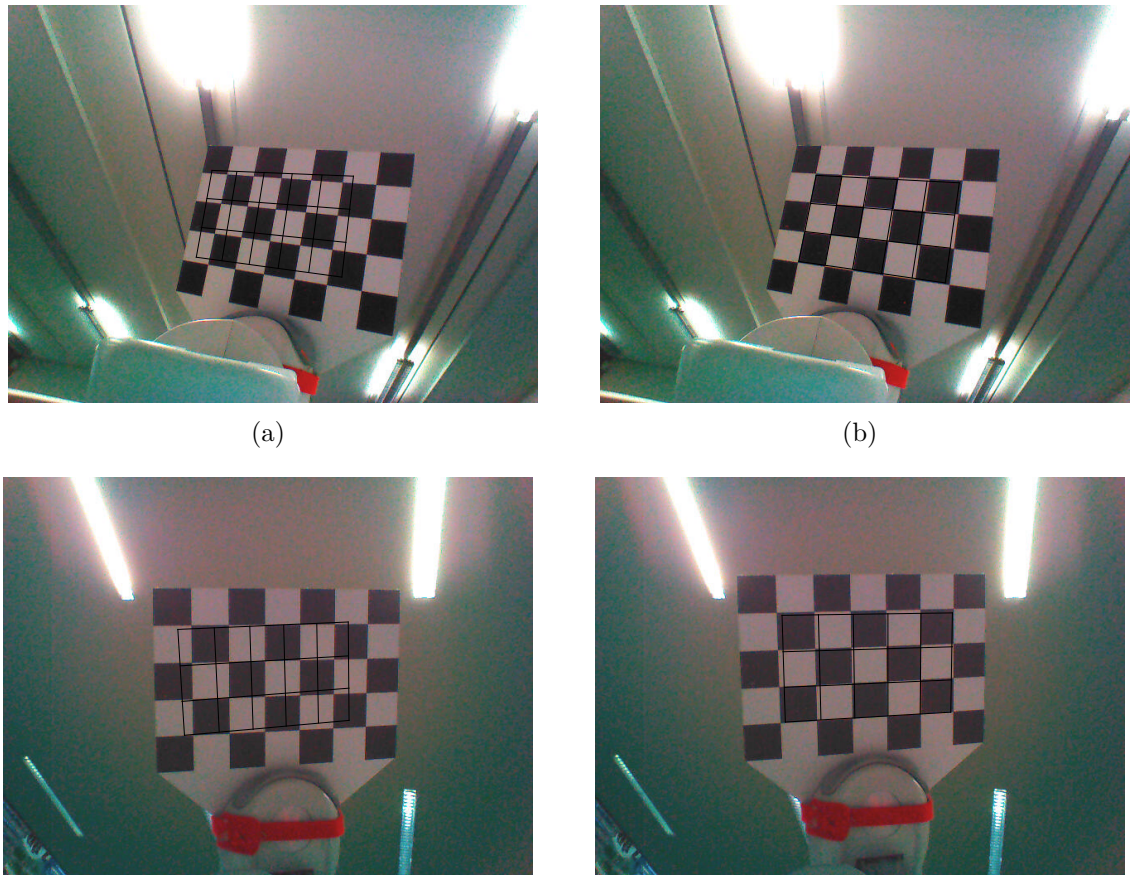
Tabelle 5.10: Testfall 4 - Kripke: Parameter α, Standardabweichung σ und Parameterdurchschnitt $\varnothing$ der drei Experimente.

Wil Wheaton		α_1	α_2	α_3	$\varnothing_{\alpha_1,\alpha_2,\alpha_3}$
$\alpha_{CameraYaw}$ [°]	α	1,1203	1,1515	1,2199	1,1639 ± 0,0509
	σ	0,0098	0,0098	0,0098	
$\alpha_{CameraPitch}$ [°]	α	3,3673	3,4776	3,3383	3,3944 ± 0,0735
	σ	0,0074	0,0073	0,0076	
$\alpha_{CameraRoll}$ [°]	α	-0,2993	-0,2519	-0,3197	-0,2903 ± 0,0348
	σ	0,0119	0,0120	0,0121	
$\alpha_{LeftHipYawPitch}$ [°]	α	-0,3026	-0,3828	-0,2830	-0,3228 ± 0,0529
	σ	0,0120	0,0127	0,0118	
$\alpha_{LeftHipRoll}$ [°]	α	1,2804	1,3287	1,3525	1,3205 ± 0,0367
	σ	0,0142	0,0145	0,0142	
$\alpha_{LeftHipPitch}$ [°]	α	-1,3395	-2,2897	-1,8953	-1,8415 ± 0,4774
	σ	0,0142	0,0141	0,0145	
$\alpha_{LeftKneePitch}$ [°]	α	-1,9254	-1,1052	-1,2619	-1,4308 ± 0,4354
	σ	0,0139	0,0155	0,0154	
$\alpha_{LeftAnklePitch}$ [°]	α	3,0861	3,1153	2,9069	3,0361 ± 0,1128
	σ	0,0149	0,0138	0,0151	
$\alpha_{LeftAnkleRoll}$ [°]	α	0,4651	0,1976	0,2594	0,3074 ± 0,1401
	σ	0,0190	0,0189	0,0187	
$\alpha_{RightHipYawPitch}$ [°]	α	1,3833	1,3099	1,3031	1,3321 ± 0,0445
	σ	0,0127	0,0119	0,0125	
$\alpha_{RightHipRoll}$ [°]	α	0,3614	0,2715	0,3306	0,3212 ± 0,0457
	σ	0,0125	0,0124	0,0125	
$\alpha_{RightHipPitch}$ [°]	α	0,2622	0,0071	-0,0859	0,0611 ± 0,1802
	σ	0,0137	0,0142	0,0140	
$\alpha_{RightKneePitch}$ [°]	α	-0,2769	-0,0312	0,3430	0,0116 ± 0,3122
	σ	0,0151	0,0163	0,0148	
$\alpha_{RightAnklePitch}$ [°]	α	0,3180	0,3244	0,1415	0,2613 ± 0,1038
	σ	0,0145	0,0144	0,0156	
$\alpha_{RightAnkleRoll}$ [°]	α	-0,1519	-0,2315	-0,1308	-0,1714 ± 0,0531
	σ	0,0187	0,0195	0,0189	
$\alpha_{LeftPatternRotZ}$ [°]	α	-0,2357	-0,2867	-0,3140	-0,2788 ± 0,0397
	σ	0,0154	0,0154	0,0158	
$\alpha_{LeftPatterTransX}$ $[mm]$	α	11,1515	12,2896	11,3391	11,5934 ± 0,6102
	σ	0,0685	0,0716	0,0642	
$\alpha_{LeftPatterTransY}$ $[mm]$	α	-2,6969	-2,1145	-1,8457	-2,2190 ± 0,4351
	σ	0,0826	0,0811	0,0806	
$\alpha_{RightPatternRotZ}$ [°]	α	1,5605	1,4791	1,4462	1,4953 ± 0,0588
	σ	0,0765	0,0139	0,0131	
$\alpha_{RightPatterTransX}$ $[mm]$	α	1,7354	1,9467	1,9661	1,8827 ± 0,1280
	σ	0,0783	0,0708	0,0701	
$\alpha_{RightPatterTransY}$ $[mm]$	α	1,4329	1,4569	2,0506	1,6468 ± 0,3499
	σ	0,0154	0,0777	0,0776	

Tabelle 5.11: Testfall 8 - Wil Wheaton: Parameter α, Standardabweichung σ und Parameterdurchschnitt $\varnothing$ der drei Experimente.

$Mrs.\ Wolowitz$		α_1	α_2	α_3	$\varnothing_{\alpha_1,\alpha_2,\alpha_3}$
$\alpha_{CameraYaw}$ [°]	α	-1,5517	-1,3869	-1,4051	$-1,4479 \pm 0,0904$
	σ	0,0097	0,0105	0,0100	
$\alpha_{CameraPitch}$ [°]	α	0,7030	0,7535	0,7404	$0,7323 \pm 0,0262$
	σ	0,0077	0,0082	0,0080	
$\alpha_{CameraRoll}$ [°]	α	-1,8866	-1,9516	-2,0716	$-1,9699 \pm 0,0939$
	σ	0,0125	0,0127	0,0122	
$\alpha_{LeftHipYawPitch}$ [°]	α	-1,5041	-0,9961	-1,1364	$-1,2122 \pm 0,2623$
	σ	0,0132	0,0122	0,0123	
$\alpha_{LeftHipRoll}$ [°]	α	0,3704	0,5332	0,5711	$0,4916 \pm 0,1066$
	σ	0,0131	0,0141	0,0132	
$\alpha_{LeftHipPitch}$ [°]	α	-0,8835	-1,5931	-0,9773	$-1,1513 \pm 0,3855$
	σ	0,0142	0,0144	0,0145	
$\alpha_{LeftKneePitch}$ [°]	α	-2,7205	-2,3143	-2,4804	$-2,5051 \pm 0,2042$
	σ	0,0163	0,0158	0,0146	
$\alpha_{LeftAnklePitch}$ [°]	α	4,5117	4,5368	4,4496	$4,4994 \pm 0,0449$
	σ	0,0147	0,0142	0,0141	
$\alpha_{LeftAnkleRoll}$ [°]	α	0,4253	0,3935	0,2880	$0,3689 \pm 0,0719$
	σ	0,0187	0,0187	0,0187	
$\alpha_{RightHipYawPitch}$ [°]	α	0,6455	0,3344	0,5830	$0,5210 \pm 0,1646$
	σ	0,0124	0,0133	0,0127	
$\alpha_{RightHipRoll}$ [°]	α	0,4542	0,5672	0,4745	$0,4986 \pm 0,0602$
	σ	0,0130	0,0120	0,0133	
$\alpha_{RightHipPitch}$ [°]	α	-0,2202	-0,3103	-0,4852	$-0,3386 \pm 0,1347$
	σ	0,0148	0,0139	0,0143	
$\alpha_{RightKneePitch}$ [°]	α	-1,7719	-1,4092	-1,9460	$-1,7090 \pm 0,2739$
	σ	0,0150	0,0147	0,0154	
$\alpha_{RightAnklePitch}$ [°]	α	3,5064	3,3033	3,7491	$3,5196 \pm 0,2232$
	σ	0,0145	0,0143	0,0143	
$\alpha_{RightAnkleRoll}$ [°]	α	0,1249	0,1936	0,2975	$0,2053 \pm 0,0869$
	σ	0,0189	0,0193	0,0184	
$\alpha_{LeftPatternRotZ}$ [°]	α	-0,2846	-0,3897	-0,1791	$-0,2845 \pm 0,1053$
	σ	0,0157	0,0145	0,0155	
$\alpha_{LeftPatterTransX}$ $[mm]$	α	10,9643	11,3720	10,5954	$10,9772 \pm 0,3885$
	σ	0,0729	0,0687	0,0708	
$\alpha_{LeftPatterTransY}$ $[mm]$	α	-1,1080	-1,6244	-2,1189	$-1,6171 \pm 0,5055$
	σ	0,0816	0,0790	0,0826	
$\alpha_{RightPatternRotZ}$ [°]	α	0,9864	0,7457	1,2318	$0,9880 \pm 0,2431$
	σ	0,0157	0,0153	0,0149	
$\alpha_{RightPatterTransX}$ $[mm]$	α	8,1210	9,8381	9,1038	$9,0210 \pm 0,8615$
	σ	0,0741	0,0662	0,0718	
$\alpha_{RightPatterTransY}$ $[mm]$	α	0,3859	1,3980	0,8397	$0,8745 \pm 0,5069$
	σ	0,0785	0,0889	0,0817	

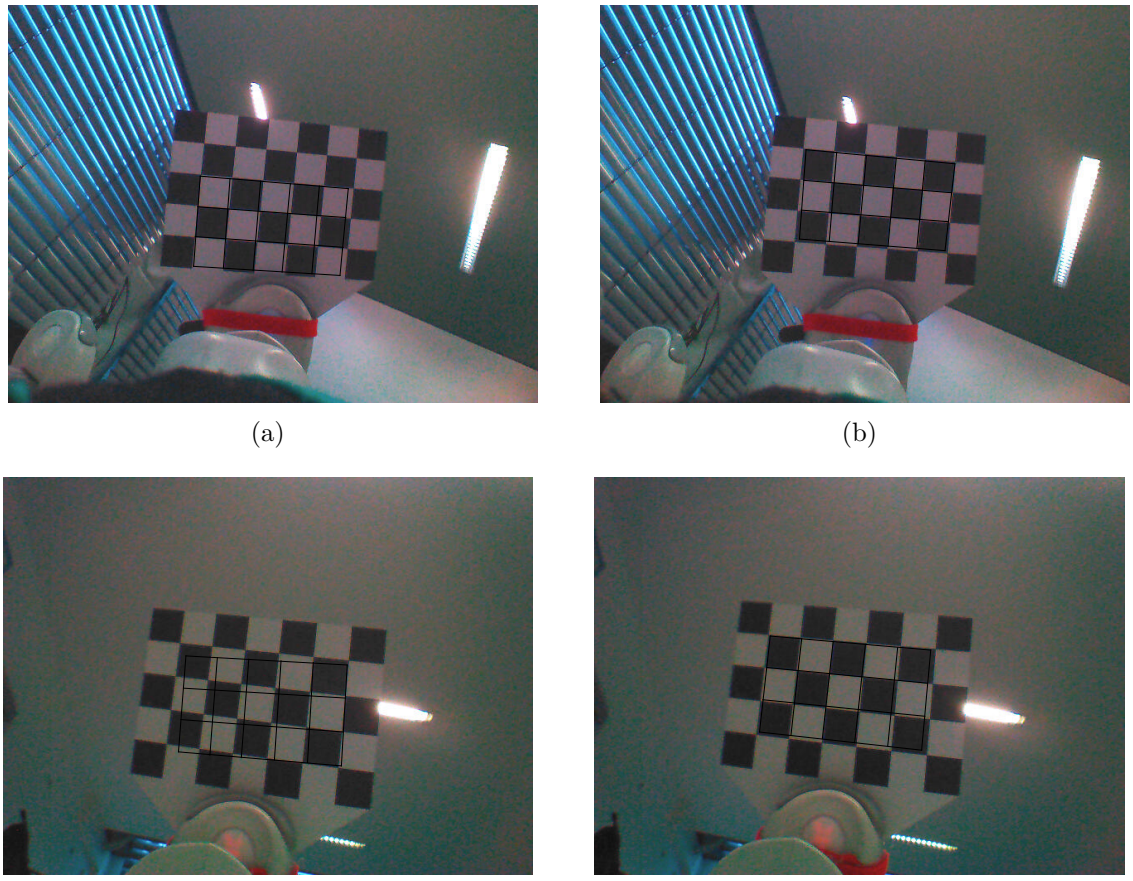
Tabelle 5.12: Testfall 8 - Mrs. Wolowitz: Parameter α, Standardabweichung σ und Parameterdurchschnitt $\varnothing$ der drei Experimente.



(a) (b)

(c) (d)

Abbildung 5.9: Projektion der Schachbretter vor (links) und nach der Kalibrierung (rechts): (a) und (b) für Leslie; (c) und (d) für Kripke.



(a) (b)

(c) (d)

Abbildung 5.10: Projektion der Schachbretter vor (links) und nach der Kalibrierung (rechts): (a) und (b) für Wil Wheaton; (c) und (d) für Mrs. Wolowitz.

5.4.2 Testfall 9 - Fortbewegung

Ziel dieser Arbeit ist es den *NAO* so zu kalibrieren, dass dieser erfolgreich ein Fußballspiel bestreiten kann. Mit der wichtigste Erfolgsfaktor ist daher ein stabiler und schneller Lauf. Da Kripke zum Zeitpunkt dieses Tests schwer beschädigt war, wurden die drei *NAO*s Leslie, Wil Wheaton und Mrs. Wolowitz mit der automatischen Roboterkalibrierung kalibriert und mussten daraufhin für zehn Minuten, der Dauer einer Halbzeit eines Fußballspiels der *SPL*, auf dem sechs mal vier Meter großem Spielfeld dem Spielball hinterherlaufen. Dabei wurde gezählt, wie oft der Roboter dabei umgefallen ist. Unterschieden wurde dabei, ob der Roboter einen Schuss ausgeführt hat oder sich geradeaus oder seitlich fortbewegte. Für die Lauftests wurden die Parameter für das Laufen so eingestellt, dass die *NAO*s eine deutlich höhere Vorwärtsgeschwindigkeit erzielen konnte, als beim vergangenen *RoboCup* 2013 in Eindhoven. Dieser Parametersatz war für alle Roboter identisch. Wie oft ein Roboter umgefallen ist, kann man Tabelle 5.13 entnehmen. Zusätzlich werden noch die Temperaturen des linken und rechten Servomotors der Knie nach Absolvierung der zehn Minuten aufgelistet, da die Kniegelenke während des Laufens am stärksten belastet werden.

		Vorwärts	*Seitwärts*	*Schuss*	*linkes Knie*	*rechtes Knie*
Leslie	0	1	4	0	68°*C*	65°*C*
	I	1	4	2	65°*C*	65°*C*
Wil Wheaton	0	3	5	0	71°*C*	59°*C*
	I	2	4	1	69°*C*	61°*C*
Mrs. Wolowitz	0	*10*	*1*	*1*	48°*C*	44°*C*
	I	8	24	1	79°*C*	66°*C*

Tabelle 5.13: Auflistung wie oft ein Roboter während des Experiments umgefallen ist und der Temperatur der Kniegelenke nach zehn Minuten.

Alle Roboter hatten große Probleme mit dem seitlichen Gang, was vor allem auffällig wurde, während der Ausrichtung nahe des Spielballs zum gegnerischen Tor. Die *NAO*s schaukelten sich dabei so stark auf, dass entweder der Lauf unterbrochen wurde oder ein Umfallen nicht mehr zu verhindern war. Ist ein Roboter nach einem Schuss umgefallen, so handelte es sich bei diesen immer um Seitwärtsschüsse, welche von der Laufsteuerung erzeugt wurden. Auch bei diesen hatten sich Leslie und Wil Wheaton vorher stark aufgeschaukelt und waren am schwanken. Die Problematik beim seitlichen Gang ist seit längerer Zeit bekannt und konnte bisher auch nicht mit von Hand kalibrierten Robotern behoben werden.

Das gerade Laufen war bei Leslie und Wil Wheaton *NAO*s sehr stabil und schnell. Leslie ist dabei innerhalb der ersten Sekunden einmal umgefallen, wobei er sich leicht seitwärts bewegte. Nach diesem einmaligen Umfallen ist er die nächsten 9 Minuten

problemlos mehrfach über die gesamte Länge des Spielfeldes gelaufen. Wil Wheaton ist in der letzten Minute zweimal kurz nacheinander auf der Mittellinie umgekippt. Die Linien sind dabei mit Klebeband auf dem Spielfeld befestigt und bieten eine geringere Reibung als der Teppich. In vielen richtigen Fußballspielen hat sich das Problem mit den Linien ebenfalls gezeigt.

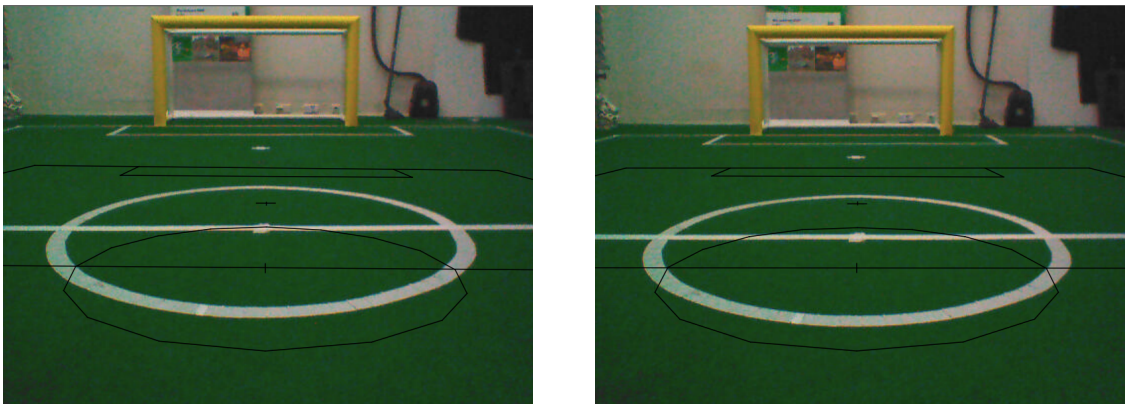
Der Test mit Mrs. Wolowitz war sehr bemerkenswert, da dieser ohne Kalibrierung nicht in der Lage war, sich vorwärts zu bewegen. Dieser Test wurde nach 5:25 Minuten, mit insgesamt zehnfachem Hinfallens beim vorwärtslaufen, abgebrochen. Mit optimierten Parametern konnte ein einigermaßen stabiles Laufen nach vorne erzielt werden. Der Roboter kippte dabei beim Laufen von Kurven achtmal um. Im Nahbereich des Balls schaukelte sich der *NAO* wiederholbar auf und fiel aufgrund des seitwärtigen Ganges insgesamt 24 Mal um. Dass Mrs. Wolowitz ohne Kalibrierung nur einmal beim Seitwärtsgang umgefallen ist, lag daran, dass der Roboter es nur einmal geschafft hatte, sich in den Nahbereich des Balles zu begeben.

Nach den zehn Minuten hatten beide Kniemotoren von Leslie eine Temperatur von $65°C$. Bei Wil Wheaton und Mrs. Wolowitz war das linke Knie deutlich wärmer als das rechte. Die Temperatur wird von *NAOqi* über die gemessenen Stromstärken der Servos geschätzt und nicht gemessen. Die Kalibrierung hat die Temperatur der Knie nicht wesentlich beeinflusst.

5.4.3 Testfall 10 - Feldlinien

Die Bildverarbeitungsalgorithmen von *B-Human* sind in der Lage die Position des orangen Spielballs, der Torpfosten, von anderen Robotern und der Spielfeldlinien im Kamerabild zu bestimmen. Der Algorithmus für die Lokalisierung des *NAO*s auf dem Spielfeld benötigt Informationen über die relative Position dieser Merkmale in Spielfeldkoordinaten. Die Umwandlung von Bildpunkten zu Weltpunkten geschieht über die Kameramatrix, die Pose der Kamera relativ zu einem Punkt auf dem Feld zwischen den Füßen. Die Pose der Kameramatrix hängt dabei besonders von der Kalibrierung der Kamera ab. Ein nicht kalibrierter Roboter steht potentiell schief und verfügt über Rotationsfehler der Kamera, was Auswirkungen auf die Präzision der Koordinatenumrechnung hat.

In Abbildung 5.11 (a) wird die Projektion der Feldlinien ins Kamerabild mit einem unkalibrierten Leslie gezeigt. Die projizierten schwarzen Linien liegen weit unter den echten und sind minimal nach rechts gekippt. In Abbildung 5.11(b) wird die Projektion mit den optimierten Parametern α_3 aus Tabelle 5.9 gezeigt. Die projizierten Linien liegen etwas weiter höher, aber immer noch weit unter den echten Linien. Allerdings erscheinen die schwarzen Linien gerader im Bild. Anhand der gelben Latte wird er-



(a) (b)

Abbildung 5.11: Projektion von Feldlinien in das Kamerabild vor (links) und nach der Optimierung (rechts) von Leslie.

sichtlich, dass der *NAO* etwas aufrechter steht. Mit dieser Kameramatrix würde die Distanz zu einem Ball, der auf dem Anstoßpunkt liegt, um ca. 60 Zentimeter in positiver x-Richtung abweichen. Je weiter ein Objekt entfernt ist, umso schlechter wird die Schätzung der Distanz bzw. Position.

Für die anderen drei *NAO*s war die Situation sehr ähnlich, weswegen auf weitere Abbildungen an dieser Stelle verzichtet wird.

5.5 Evaluation

Die Testfälle für die Bildverarbeitung und der Optimierung sind zufriedenstellend ausgefallen. Die Präzision der Eckenerkennung und die Toleranz gegenüber verschiedener Beleuchtungen konnte nachgewiesen werden. Anhand von zwei Standardproblemen der nichtlinearen Optimierung konnte die korrekte Funktionsweise des *Levenberg-Marquardt*-Algorithmus aufgezeigt werden.

Die Tests der simulierten Roboterkalibrierung sind allesamt erfolgreich ausgefallen. In allen Testfällen konnten die Korrekturparameter für die Beine und die der Kamera gefunden werden. Die Genauigkeit der Parameter ist ausreichend präzise, da die Motoren des *NAO*s nur über eine Auflösung von $0,1°$ verfügen (siehe Abschnitt 2.1). Zusammenfassend kann die simulierte Roboterkalibrierung als erfolgreiche Machbarkeitsuntersuchung des Verfahrens angesehen werden.

Anforderungen

Die folgende Auswertung soll ergeben, ob die formulierten Anforderungen aus Abschnitt 1.4 erfüllt wurden. Hierfür wird in Tabelle 5.14 jeder Anforderung eine Menge von Testfälle mit entsprechender Bewertung zugeordnet. Eine Anforderung wird mit der schlechtesten Wertung eines zugeordneten Testfalls bewertet.

Anforderung	*Testfall*	*Bewertung*
A1 - Schneller als manuelle Methode	Testfall 8 - Leslie, Kripke, Wil Wheaton und Mrs. Wolowitz	✓
A2 - Einfache Handhabung	Testfall 8 - Leslie, Kripke, Wil Wheaton und Mrs. Wolowitz	✓
A3 - Robustheit	Testfall 1 - Präzision	✓
	Testfall 2 - Beleuchtungstoleranz	✓
	Testfall 5 - Kein Arretierungsfehler	✓
	Testfall 6 - Alle Parameter fehlerbehaftet	✓
	Testfall 7 - Partielle Fehlerbelegung	✓
	Testfall 8 - Leslie, Kripke, Wil Wheaton und Mrs. Wolowitz	✓
A4 - Qualität der Kalibrierung	Testfall 8 - Leslie, Kripke, Wil Wheaton und Mrs. Wolowitz	(✓)
	Testfall 9 - Fortbewegung	(✓)
	Testfall 10 - Feldlinien	-
A5 - Wiederholbarkeit	Testfall 5 - Kein Arretierungsfehler	✓
	Testfall 6 - Alle Parameter fehlerbehaftet	✓
	Testfall 7 - Partielle Fehlerbelegung	✓
	Testfall 8 - Leslie, Kripke, Wil Wheaton und Mrs. Wolowitz	(✓)

Tabelle 5.14: Zuordnung von Testfällen zu Anforderungen samt zugehöriger Auswertung. Ein ✓gibt an, dass der Testfall die Anforderung erfüllt, ein (✓) sagt aus, dass die Anforderung nur teilweise umgesetzt wurde und ein - zeigt an, dass die Anforderung nicht eingehalten werden konnte.

A1 - Schneller als manuelle Methode

Die Dauer eine Kalibrierung wurde durch die Tests mit den *NAO*s ermittelt. In den Tests hat eine komplette Kalibrierung im Schnitt 9:53 Minuten benötigt, was deutlich schneller als die manuelle Methode ist. Für eine manuelle Kalibrierung werden im Schnitt 30 Minuten benötigt[3]. Durch das automatische Verfahren ist eine Zeit von ca. zehn Minuten garantiert, welches parallel mit mehreren *NAO*s durchführbar ist, weswegen die Anforderung als erfüllt angesehen wird.

A2 - Einfache Handhabung

Auch die Anforderung der einfachen Handhabung konnte erfüllt werden. Ein unerfahrener Benutzer muss nur die richtige Software auf dem *NAO* installieren, die Schachbrettsandale an den Füßen anbringen und die Prozedur per Knopfdruck initialisieren. Für die nun ca. folgenden zehn Minuten der Kalibrierung kann sich der Benutzer anderen Beschäftigungen widmen.

A3 - Robustheit

Die Beleuchtungstoleranz und Robustheit der Schachbretterkennung konnte anhand von mehreren Testfällen nachgewiesen werden. Eine nahezu einhunderprozentige Erkennungsrate der Tests der Simulation und mit den *NAO*s bestätigt die Robustheit. Die Beleuchtungstoleranz konnte durch den ersten Testfall angenommen und durch die Tests der Kalibrierung mit Leslie, Kripke, Wil Wheaton und Mrs. Wolowitz belegt werden. Die dreimalige Wiederholung der Kalibrierung für jeden *NAO* konnte mit identischer Parametrierung aller beteiligten *Module* ohne Probleme durchgeführt werden. Anhand dieser Ergebnisse wird die Anforderung an die Robustheit als erfüllt angesehen.

A4 - Qualität der Kalibrierung

Die wichtigste Anforderung stellt die Qualität der Kalibrierung dar. Das erste Kriterium für eine erfolgreiche Kalibrierung stellt die Projektion des Schachbretts mit optimierten Parametern in das Kamerabild dar, während die Bewegungen und Gelenkstellungen der Kalibrierung nochmal abgefahren werden. Auf den Abbildungen 5.9 und 5.10 wird gezeigt, dass die Projektion mit optimierten Parametern eine bessere Übereinstimmung mit dem Kamerabild ergibt.

[3]Für diese Zeitangaben gibt es keine empirischen Werte. Sie basieren lediglich auf Erfahrungen und Aussagen der Mitglieder von *B-Human*, die für die Kalibrierung verantwortlich sind.

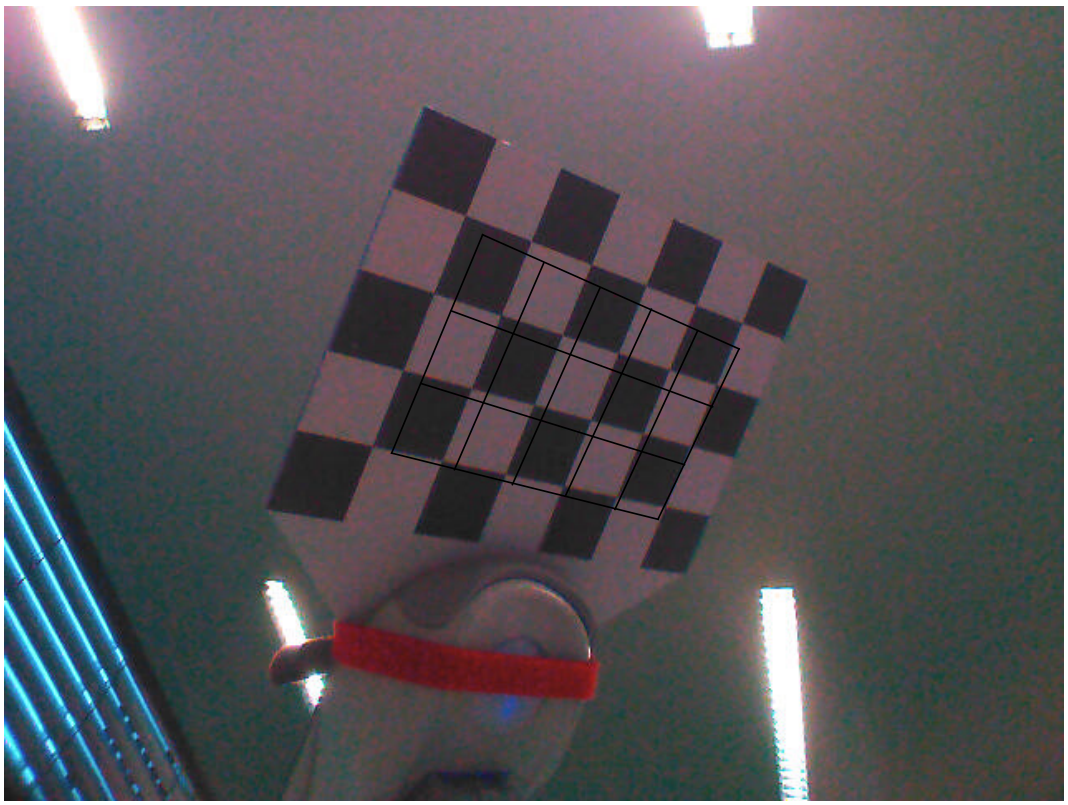
Das nächste Kriterium stellen die Lauftests dar. Ein korrekt kalibrierter Roboter muss in der Lage sein, stabil zu Laufen und präzise zu schießen. Die Ergebnisse dieses Tests haben ergeben, dass die optimierten Parameter das Laufen und das Schießen nicht negativ beeinflusst haben. In einem Fall wurde sogar die Vorwärtsbewegung deutlich verbessert. Die Kalibrierung hat die Temperatur der Knie nicht wesentlich beeinflusst. Die verbesserten Parameter sorgten bei allen *NAO*s dafür, dass diese aufrechter und gerader stehen. Die aufrechte Haltung wird in Abbildung 5.11 deutlich. Obwohl die Lauftests im Großen und Ganzen erfolgreich verliefen, wird die Anforderung an die Qualität als nicht erfüllt angesehen, da der Testfall mit der Projektionen der Feldlinien fehlgeschlagen ist. Erschwerend kommt hinzu, dass die Projektion des Schachbretts nicht für alle Gelenkstellungen gleich ausfällt. In Abbildung 5.12 wird per Hand eine Beinstellung gehalten, welche trotz Kalibrierung eine stark abweichende Schachbrettprojektion erzeugt.

A5 - Wiederholbarkeit

Die Anforderung der Wiederholbarkeit kann nur teilweise erfüllt werden. Die simulierte Roboterkalibrierung kann beliebig oft durchgeführt werden und ist dabei in der Lage die korrekte Lösung mit nahezu gleicher Präzision zu finden. Mit den *NAO*s allerdings, liegt, bei dreimaliger Ausführung, die maximale Abweichung zwischen den optimierten Parametern bei $\pm\ 0,4774°$ (siehe $\alpha_{LeftHipPitch}$ in Tabelle 5.11). Die größeren Abweichungen ergaben sich bei den drei aufeinanderfolgenden pitch-Gelenken der kinematischen Kette der Beine. Der Optimierer war also in der Lage, über verschiedenartige Anpassungen dieser Gelenke die Nullagenfehler auszugleichen. Die meisten Parameter waren mit einer Abweichung von $\pm 0,2°$ ausreichend ähnlich.

5.6 Schwächen

Am Auffäligsten ist das negative Ergebnis des Tests der Projektion der Feldlinien. In den simulierten Fällen wurden die Parameter der Kamera korrekt bestimmt. Bei den *NAO*s hingegen erscheint die Projektion der Linien deutlich unter denen im Bild, als ob die Kamera zu weit um die y-Achse rotiert ist. Die Verteilungen der Residuen nach der Optimierung der *NAO*s weisen, im Vergleich zur Simulation, eine stärkere Standardabweichung von bis zu 4 Pixel auf. In den simulierten Tests konnte eine Abweichung von weniger als einem halben Pixel erreicht werden. Hinzu kommen noch die Gelenkstellungen, für die die Vorhersage des Schachbretts im Kamerabild trotz erfolgter Kalibrierung stark vom zu sehenden Schachbrett abweichen, sowie die teils großen Korrekturwerte für die Translation der Schachbrettsandale von bis zu einem Zentimeter (siehe z.B. Parameter $\alpha_{LeftPatterTransX}$ in Tabelle 5.10).



(a)

Abbildung 5.12: Projektion des Schachbrettes mit einem kalibrierten Wil Wheaton. Es gibt immer noch Gelenkstellungen, bei denen die Projektion des Schachbrettes stark vom Optimum abweicht.

All diese Beobachtungen lassen darauf schließen, dass die zu Grunde liegenden Modelle der Kinematik und der Kamera des *NAO*s unvollständig oder inkorrekt sind. Eine mögliche Fehlerquelle findet sich im Nackengelenk, da für dieses, wie bei der manuellen Methode, keine Gelenkwinkelversätze berücksichtigt werden. Das Nackengelenk gehört allerdings mit zu den anfälligsten Gelenken für Beschädigungen, wenn der *NAO* umfällt.

Das Modell der Kinematik lässt außer Acht, dass die Glieder zwischen den Gelenken, möglichweise von ihrer spezifizerten Länge oder Lage abweichen können. Außerdem wurden die intrinsichen Parameter der Kamera aus der *NAO*-Dokumentation entnommen und als Konstanten verwendent.

Ein weiterer, potentieller Faktor stellen nicht-geometrische Fehler wie Gelenkelastizitäten dar. Diese könnten entstehen, da der *NAO* während der Kalbrierung auf dem Rücken liegt. Das zusätzliche Gewicht der Schachbrettsandalen drückt dabei die Beine stärker in Richtung des Bodens. Laut Birbach u.a.[OF12] wirken sich Elastizitäten auf die Verteilung der Residuen aus. Diese weist dann eine starke Abweichung in y-

Richtung auf. Diese Beobachtung trifft auf die Ergebnisse dieser Arbeit bedingt zu, was durch die Abweichungen der Tabelle 5.14 ersichtlich wird.

Obschon die Vermutung besteht, dass das Modell des *NAO*s für eine perfekte Roboterkalibrierung unvollständig ist, kann man die optimierten Parameter zumindest als verbesserte Ausgangsbasis für die manuelle Kalibrierung verwenden, da diese laut der Testergebnisse, die Einsatzfähigkeit eines *NAO*s im besten Fall verbessert, allerdings nie verschlechtert haben.

6 Schluss

Im letzten Kapitel wird die Arbeit kurz zusammengefasst und ein Fazit formuliert. Im Ausblick werden mögliche Verbesserungen der automatischen Roboterkalibrierung aufgelistet.

6.1 Fazit

Ziel der vorliegenden Arbeit ist es, eine automatische Roboterkalibrierung für den humanoiden Roboter *NAO* zu entwickeln. Das Verfahren sollte sicherstellen, dass der Roboter für ein Fußballspiel eingesetzt werden kann und dabei der Aufwand für menschliche Nutzer minimal ist. Mit der unteren Kamera des Roboters wird unter verschiedenen Beinstellungen, ein an den Fußsohlen befestigtes Schachbrett beobachtet. Ein mathematisches Modell des *NAO*s wird dazu verwendet, um Vorhersagen über die Position der Schachbretter im Kamerabild zu treffen. Mit dem Algorithmus von *Levenberg und Marquardt* wird das Modell solange angepasst, bis es zu den Messungen passt. Zu diesem Zweck wurden im Zuge dieser Arbeit eine Schachbretterkennung, eine Bewegungssteuerung, ein Optimierer und eine Kalibrierungssteuerung entwickelt und getestet.

Mit Hilfe der Simulationssoftware *SimRobot* konnte das Verfahren positiv auf Plausibilität und Machbarkeit geprüft werden und wurde folglich mit den *NAO*s getestet. Dabei ergab sich, dass die in der Simulation gewonnenen Erkenntnisse nicht gänzlich für die *NAO*s gelten. Die Anpassung der Kinematik hat die Mobilität der Roboter nicht negativ beeinflusst und in einigen Fällen leicht verbessert. Allerdings ist das Verfahren nicht in der Lage, mit der Qualität der bisherigen manuellen Lösung mitzuhalten. Es konnte nicht erreicht werden, dass die Umrechnung von Bildpunkten der Kamera zu Weltkoordinaten, welche von der genauen Pose der Kamera abhängt, korrekt funktioniert. Die Evaluation der automatischen Roboterkalibrierung lässt darauf schließen, dass die verwendeten Modelle der Kinematik und der Kamera des *NAO*s unvollständig oder inkorrekt sind. Trotz dieser Vermutung, lässt sich abschließend feststellen, dass die zu Grunde liegende Methodik dazu in der Lage ist, einen Roboter ohne vorherige Kalibrierung in einen einsatzfähigeren Zustand zu bringen, wobei die optimierten Parameter als solide Basis für eine bessere manuelle Kalibrierung verwendet werden können,

6.2 Ausblick

Die folgenden Vorschläge zur Verbesserung der automatischen Roboterkalibrierung basieren auf den in Abbschnit 5.6 aufgezeigten Problemen.

Um die Ergebnisse der Kalibrierung zu verbessern, könnten die Modelle der Kinematik und der Kamera um Längen- und Positionierungsfehler erweitert werden. Für die untere Kamera wurde z. B. angenommen, dass die Position relativ zum Nackengelenk der Dokumentation des *NAO*s entspricht. Außerdem wurde die Kinematik der Beine mit festen Werten für die Länge der einzelnen Glieder berechnet, so dass mögliche vorhandene Abweichungen nicht berücksichtigt wurden.

Eine weitere mögliche Erweiterung des Modells, stellt die Berücksichtigung von Nullagenfehler des Nackengelenks dar. Hier ist allerdings zu befürchten, dass die Parameter der Rotation der Kamera nicht getrennt von den Parametern des Nackens beobachtet werden können, da der Abstand zwischen dem Servomotor des Nackens und der Kamera sehr gering ist. Daher sind die jeweiligen Rotationsachsen sehr nah beieinander, was laut Wiest[Wie01] zu linearen Abhängigkeiten führt, welche nicht eindeutig unterscheidbar sind.

Das Modell könnte zusätzlich noch Elastizitäten der Gelenke berücksichtigen. Birbach u.a.[OF12] und Wiest[Wie01] haben die Elastizitäten als Feder modelliert, welche allerdings von Drehmomenten abhängig ist. Da der *NAO* über keine Drehmomentssensoren in den Servomotoren verfügt, müssten diese geschätzt werden. Für die Berechnung von statischen Drehmomenten gibt es von Wiest[Wie01] einen Algorithmus. Für das Modell müsste dann eine Federkonstante für jedes Gelenk geschätzt werden.

Anstatt auf dem Rücken liegend, könnte der *NAO* die Datensammlung auch im Stehen vollführen. Dies könnte entweder automatisch oder durch einen Nutzer geschehen. Eine aufrechte Haltung könnte den Einfluss von Elastizitäten verringern. Für eine automatische Datensammlung wären Bewegungen auf einem Bein stehend vorstellbar. Das Schachbrett wird dann vom freien Bein bewegt, was allerdings problematisch ist, da eine Position gefunden werden müsste, die entweder allgemein stabil ist oder durch dynamische Balancieren gesteuert wird. Allerdings ist es fraglich, ob ein unkalibrierte Roboter korrekt die Balance halten kann. Eine von Hand geführte Bewegung hingegen, wie von Yamane u.a.[Yam] durchgeführt, könnte den jeweiligen Fuß stabil mit Kontakt zum Boden bewegen. Dabei würden alle Messpunkte auf der Bodenebene liegen. Der Nachteil dieser Vorgehensweise wäre, dass die Bewegungen manuell durchgeführt werden.

Die Kalibrierung der Arme und der oberen Kamera wurden in dieser Arbeit ausgelassen, da die Auswahl von Gelenkstellungen mit denen die Schachbrettsandale in der Kamera

komplett sichtbar wäre, sehr begrenzt ist. Denkbar ist allerdings eine Kalibrierung mit einem Aufkleber einer Schachbrettecke auf den Fäusten des *NAO*s umzusetzen. Diese Kalibrierung würde dann verschiedenen Armstellungen ansteueren, wobei der Aufkleber der Faust in der oberen Kamera sichtbar wäre. Man würde dann ein Modell der Kinematik der Arme verwenden, um die Position der Markierung im Kamerabild vorhersagen zu können.

Abschließend könnte die Aufnahme der intrinsischen Parameter der Kameras in das Modell die Qualität der Kalibrierung verbessern, und nicht, wie in Arbeit als konstant angesehen wurden.

Abbildungsverzeichnis

Tabellenverzeichnis

Literatur

[08] *International vocabulary of metrology - Basic and general concepts and associated terms (VIM) Vocabulaire international de metrologie - Concepts fondamentaux et generaux et termes associes (VIM)*. JCGM. 2008.

[Amb+04] Robert O. Ambrose u. a. "Robonaut: NASA's Space Humanoid." In: *IEEE Intelligent Systems* 15.4 (27. Jan. 2004), S. 57–63. URL: http://dblp.uni-trier.de/db/journals/expert/expert15.html#AmbroseAABBDLMR00.

[Bäu+11] Berthold Bäuml u. a. "Catching flying balls and preparing coffee: Humanoid Rollin'Justin performs dynamic and sensitive tasks." In: *ICRA*. IEEE, 2011, S. 3443–3444. URL: http://dblp.uni-trier.de/db/conf/icra/icra2011.html#BaumlSWBDFFFBGEH11.

[Bey05] L. Beyer. *Genauigkeitssteigerung von Industrierobotern: insbesondere mit Parallelkinematik*. Forschungsberichte aus dem Laboratorium Fertigungstechnik. Shaker, 2005. ISBN: 9783832236816. URL: http://books.google.de/books?id=yNNOXwAACAAJ.

[BL13] Stuart Bennett u. a. "ChESS - Quick and Robust Detection of Chess-board Features". In: *CoRR* abs/1301.5491 (2013).

[Can86] J Canny. "A Computational Approach to Edge Detection". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 8.6 (Juni 1986), S. 679–698. ISSN: 0162-8828. DOI: 10.1109/TPAMI.1986.4767851. URL: http://dx.doi.org/10.1109/TPAMI.1986.4767851.

[DH55] J. Denavit u. a. "A kinematic notation for lower-pair mechanisms based on matrices". In: *Trans. ASME E, Journal of Applied Mechanics* 22 (Juni 1955), S. 215–221.

[Ela+04] A. Y. Elatta u. a. "An overview of robot calibration". In: *IEEE Journal on Robotics and Automation* 3 (ottobre 2004), S. 74–78.

[GCK10] David Gouaillier u. a. "Omni-directional closed-loop walk for NAO." In: *Humanoids*. IEEE, 2010, S. 448–454. ISBN: 978-1-4244-8688-5. URL: http://dblp.uni-trier.de/db/conf/humanoids/humanoids2010.html#GouaillierCK10.

[Gou+08] David Gouaillier u. a. "The NAO humanoid: a combination of performance and affordability". In: *CoRR* abs/0807.3223 (2008). URL: http://dblp.uni-trier.de/db/journals/corr/corr0807.html#abs-0807-3223.

[HSB12] Uwe Hubert u. a. “Bayesian Calibration of the Hand-Eye Kinematics of an Anthropomorphic Robot”. In: *Proceedings of IEEE-RAS International Conference on Humanoid Robots (Humanoids)*. Osaka, Japan, 2012.

[Kit+95] Hiroaki Kitano u. a. *RoboCup: The Robot World Cup Initiative*. 1995.

[Lam97] Michael Lampton. “Damping-undamping Strategies for the Levenberg-Marquardt Nonlinear Least-squares Method”. In: *Comput. Phys.* 11.1 (Jan. 1997), S. 110–115. ISSN: 0894-1866. DOI: 10.1063/1.168600. URL: http://dx.doi.org/10.1063/1.168600.

[Lev44] K. Levenberg. “A method for the solution of certain problems in least squares”. In: *Quart. Applied Math.* 2 (1944), S. 164–168.

[LSR06] Tim Laue u. a. “Simrobot – a general physical robot simulator and its application in RoboCup”. In: *IN: ROBOCUP 2005: ROBOT SOCCER WORLD CUP IX. LECTURE NOTES IN ARTIFICIAL INTELLIGENCE*. Springer, 2006, S. 173–183.

[Mar11] Benjamin Markowsky. “Semiautomatische Kalibrierung von Naogelenken”. Bachelorarbeit. Universität Bremen, 2011.

[Mar63] Donald W. Marquardt. “An algorithm for least-squares estimation of nonlinear parameters”. In: *SIAM Journal on Applied Mathematics* 11.2 (1963), S. 431–441. DOI: 10.1137/0111030. URL: http://dx.doi.org/10.1137/0111030.

[Nic03] K. M. Nickels. *Hand-eye calibration for Robonaut*. Tech. Rep. Johnson Space Center: NASA Summer Faculty Fellowship Program Final Report, 2003.

[OF12] Berthold Bäuml Oliver Birbach u. a. “Automatic and Self-Contained Calibration of a Multi-Sensorial Humanoid’s Upper Body”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. St. Paul, MN, USA, 2012, S. 3103–3108.

[PKB10] Vijay Pradeep u. a. “Calibrating a multi-arm multi-sensor robot: A Bundle Adjustment Approach”. In: *International Symposium on Experimental Robotics (ISER)*. New Delhi, India, Dez. 2010.

[Rob] Aldebaran Robotics. *NAO Humanoid Robot Platform (Datasheet)*. Online: http://www.aldebaran-robotics.com/en/Discover-NAO/Key-Features/hardware-platform.html.

[Röf+12] Thomas Röfer u. a. *B-Human Team Report and Code Release 2012*. Only available online: http://www.b-human.de/wp-content/uploads/2012/11/CodeRelease2012.pdf. 2012.

[Röf+13a] Thomas Röfer u. a. “B-Human 2013: Ensuring Stable Game Performance”. In: *RoboCup 2013: Robot Soccer World Cup XVII*. Lecture Notes in Artificial Intelligence. to appear. Springer, 2013.

[Röf+13b] Thomas Röfer u. a. *B-Human Team Report and Code Release 2013*. Only available online: `http://www.b-human.de/downloads/publications/2013/CodeRelease2013.pdf`. 2013.

[SH09] Klaus H. Strobl u. a. "Optimal Hand-Eye Calibration." In: *IROS*. IEEE, 11. Mai 2009, S. 4647–4653. URL: `http://dblp.uni-trier.de/db/conf/iros/iros2006.html#StroblH06`.

[SK08] Bruno Siciliano u. a., Hrsg. *Springer Handbook of Robotics*. Springer, 2008. DOI: `10.1007/978-3-540-30301-5`. URL: `http://www.libreka.de/9783540239574/`.

[TBF05] Sebastian Thrun u. a. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 0262201623.

[Wie01] U. Wiest. *Kinematische Kalibrierung von Industrierobotern*. Berichte aus der Automatisierungstechnik. Shaker, 2001. ISBN: 9783826586095. URL: `http://books.google.de/books?id=vx0%5C_AwAACAAJ`.

[Wyr+08] Keenan A. Wyrobek u. a. "Towards a personal robotics development platform: Rationale and design of an intrinsically safe personal robot." In: *ICRA*. IEEE, 6. Nov. 2008, S. 2165–2170. URL: `http://dblp.uni-trier.de/db/conf/icra/icra2008.html#WyrobekBLS08`.

[Yam] Katsu Yamane. "Practical kinematic and dynamic calibration methods for force-controlled humanoid robots." In: *Humanoids*. IEEE, S. 269–275. ISBN: 978-1-61284-866-2.

[ZX05] Wei Wu Zhongshi Wang u. a. "Auto-recognition and Auto-location of the Internal Corners of Planar Checkerboard Image". In: *International Conference on Intelligent Computing*. Heifei, China, 2005, S. 473–479.

[ZZ98] Zhengyou Zhang u. a. "A Flexible New Technique for Camera Calibration". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22 (1998), S. 1330–1334.

7 Anhang

7.1 EBNF für *JointMotionEngine*-Syntax

Schlüsselwörter: milestone, transition, hardness, label, HeadYaw, HeadPitch, LShoulderPitch, LShoulderRoll, LElbowYaw, LElbowRoll, RShoulderPitch, RShoulderRoll, RElbowYaw, RElbowRoll, LHipYawPitch, LHipRoll, LHipPitch, LKneePitch, LAnklePitch, LAnkleRoll, RHipYawPitch, RHipRoll, RHipPitch, RKneePitch, RAnklePitch, RAnkleRoll

```
motion_file ::= motions
                {motion_definitions | mirrored_motion}
motions ::= motions
            '{'
            label start
            {lines}
            transition motions start
            '}'
motion_definitions ::= identifier
                       '{'
                       label_line
                       {lines}
                       u_transition
                       '}'
mirrored_motion ::= identifier '|' identifier
                    '{'
                    label_line
                    {u_transition
                  | c_transition
                  | label_line}
                    '}'
lines ::= label_line
        | delta_line
```

```
                |   abs_line
                |   u_transition
                |   c_transition
                |   hardness_line
label_line  ::= ''label '' identifier
abs_line    ::= '('
                 {j_value [',']}
                ')'
                trailer
delta_line  ::= '('
                {joint float [',']}
                ')'
                trailer
u_transition  ::= transition identifier identifier
c_transition  ::= transition identifier identifier identifier
hardness_line ::= '('
                {pos_int [',']}
                ')'
                pos_int
j_value     ::= float
                |   '*'
                |   '#'
joint       ::= ''HeadYaw '' | ''HeadPitch ''
                |   ''LShoulderPitch '' | ''LShoulderRoll ''
                |   ''LElbowYaw '' | ''LElbowRoll ''
                |   ''RShoulderPitch '' | ''RShoulderRoll ''
                |   ''RElbowYaw '' | ''RElbowRoll ''
                |   ''LHipYawPitch '' | ''LHipRoll ''
                |   ''LHipPitch '' | ''LKneePitch ''
                |   ''LAnklePitch '' | ''LAnkleRoll ''
                |   ''RHipYawPitch '' | ''RHipRoll ''
                |   ''RHipPitch '' | ''RKneePitch ''
                |   ''RAnklePitch '' | ''RAnkleRoll ''
trailer     ::= pos_int pos_int [''milestone ''] [identifier]
identifier  ::= letter {letter | digit}
```

```
letter  ::=  'a'...'z' | 'A'...'Z'
digit   ::=  '0'...'9'
pos_int ::=  ['+'] digit{digit}
float   ::=  ['+' | '-'] digit{digit} ['.' {digit}]
```