Universität Bremen
Fachbereich 3

Robotergestütztes Kalibrierungssystem für Sensoren

Bachelorarbeit

Benny Prange
Matrikel-Nummer 2597237

Betreuer Alexis Maldonado
Erstprüfer Prof. Dr. Michael Beetz
Zweitprüfer Prof. Dr. Doe

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

In der Robotik werden bildgebende Sensoren benutzt, um Informationen über das Umfeld des Roboters zu erhalten. Diese Informationen können genutzt werden, um Hindernisse zu erkennen, die der Roboter meiden soll oder es werden Gegenstände gesucht, die der Roboter greifen und mit ihnen interagieren soll. Es gibt verschiedene Arten von bildgebenden Sensoren, wie z.B. Kameras, die ein zweidimensionales Bild erzeugen und Kameras, die zusätzlich zu dem zweidimensionalen Bild auch Tiefeninformationen erkennen. Andere Sensoren, die ein räumliches Abbild ohne Farbinformationen erstellen, sind z.B. Lidar-, Ultraschall- und Radarsensoren.

Bei all diesen Sensoren ist es wichtig, dass diese korrekt kalibriert sind, damit die gemessenen Daten der Realität möglichst nahe kommen. Stimmen die Messdaten nicht mit der Realität überein, stößt der Greifarm ein Objekt möglicherweise um, anstatt es zu greifen.

Kamerasensoren, die nach dem Lochkameraprinzip arbeiten, werden durch die Kameramatrix definiert. Sie enthält Angaben über den internen Aufbau der Kamera, wie Bildsensorgröße und Brennweite, sowie die Position der Kamera in Relation zu der realen Umgebung. Dadurch lassen sich aus einem aufgenommenen Bild Informationen über die dreidimensionale Welt berechnen.

Das Ziel der Kalibrierung einer Kamera ist es, eine Kameramatrix zu berechnen, die eine möglichst exakte Relation von Kamerabildern zu der realen Umgebung ermöglicht. Diese Kameramatrix kann mithilfe von Kalibrierungsmustern erstellt werden. Dazu werden mehrere Fotos aufgenommen, auf denen das Muster in verschiedenen Entfernungen und Orientierungen zu sehen ist. Schließlich lässt sich aus diesen Daten die Kameramatrix berechnen.

Dieser Vorgang ist sehr zeitaufwändig. Zum einen müssen viele Bilder gemacht werden, um eine möglichst präzise Kalibrierung zu erhalten. Daher werden bis zu 50 Bilder aufgenommen. Zum anderen empfiehlt es sich, Kamera und Kalibiriermuster jeweils auf einem Stativ zu befestigen, um scharfe Bilder zu erzeugen. Damit ist gewährleistet, dass die Erkennung des Kalibriermusters nicht von unscharfen Bildern gestört wird, allerdings wird der Vorgang durch das Verstellen der Stative auch weiter verlängert.

Um die Kalibrierung zu vereinfachen, soll daher ein Roboterarm das Bewegen des Kalibriermusters übernehmen und der ganze Ablauf mit dem Erstellen der Bilder und dem Berechnen der Parameter durch ein Programm gesteuert werden. Der Benutzer platziert die Kamera auf einem Stativ vor dem Roboterarm, an dem das Kalibiriermuster befestigt ist. Anschließend startet er das Programm, welches den Arm an verschiedene Positionen bewegt, sodass die Kamera das Muster aus mehreren Entfernungen und Orientierungen aufnehmen kann. Zum Schluss berechnet das Programm die benötigten Parameter und der Benutzer kann die Kamera mit diesen Parametern einsetzen.

TODO: Kurze Beschreibung der folgenden Kapitel

2 Grundlagen

Wie bereits in der Einleitung erwähnt, sind Kameras ein wichtiger Sensor in der Robotik, indem sie dem Roboter helfen, die Umgebung wahrzunehmen. Abhängig von dem Einsatzgebiet, ist ein genaues Abbild der Umgebung außerordentlich wichtig für die Erfüllung der Aufgabe. Dies ist der Fall, wenn zum Beispiel der Roboter über das Kamerabild einen Gegenstand lokalisieren muss, um ihn anschließend zu greifen, oder er muss in dem Kamerabild Hindernisse erkennen, um einen Weg zum Ziel zu planen.

Der relativ günstige Preis von Lochkameras gegenüber anderen bildgebenden Sensoren hat dazu beigetragen, dass Kameras ein viel genutzter Sensor sind. Da sich aber gezeigt hat, dass die Toleranzen bei der Produktion und der Aufbau der Kameras an sich zu Verzerrungen im Bild führen, wurden Modelle entwickelt, um diese Verzerrungen herauszurechnen und das Bild zu korrigieren. Ein häufig genutztes Modell wird im folgenden beschrieben.

2.1 Kameramodell

In [Zha00] wird ein gängiges Kameramodell beschrieben, welches nun kurz erklärt wird.

Die Beziehung zwischen einem Punkt (u, v) in einem zweidimensionalen Bild und einem Punkt (x, y, z) in der dreidimensionalen Welt wird durch Gleichung 2.1 beschrieben:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} R & T \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.1}$$

mit

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 0 \end{bmatrix} \tag{2.2}$$

K enthält die intrinsischen Parameter der Kamera, die auf Grund der Fertigungstoleranzen für jede Kamera unterschiedlich sind. f definiert die Brennweite der Kamera. Da die Pixel nicht immer quadratisch sind, wird dieser Wert für die x und y-Achse angegeben. c_x und c_y beschreiben das optische Zentrum des Bildes, welches ebenfalls nicht immer genau im Mittelpunktes des Bildes liegt.

R und T definieren die extrinsischen Parameter. R ist eine Rotationsmatrix, die die Rotation zwischen dem Kamera- und Weltkoordinatensystem beschreibt, T ist ein Vektor, der die Transformation zwischen Kamera- und Weltkoordinatensystem beschreibt.

2.2 Verzeichnung

Mit dem Kameramodell kann nun eine Beziehung zwischen einem Punkt im Bild und dem Punkt in der Welt hergestellt werden. Kameras bilden das Bild jedoch nicht immer korrekt ab. Häufig sieht man in Bildern den Effekt, dass gerade Linien in der Welt, wie z.B. Häuserkanten, nicht gerade im Bild aufgenommen werden, sondern das diese gebogen werden. Diesen Effekt nennt man Verzeichnung.

Man kann zwischen zwei verschiedenen Arten der Verzeichnung sprechen. Die Verzeichnung, die gerade Linien gekrümmt darstellt, heißt radiale Verzeichnung. Um diese Verzeichnung herauszurechnen, werden die drei Faktoren k_1, k_2, k_3 benötigt. Sind diese gegeben, lässt sich die Verzeichnung korrigieren. r ist der Abstand vom Bildmittelpunkt zum

Punkt, der korrigiert werden soll. x', y' sind die falsch abgebildeten Punkte, x, y sind die korrigierten Punkte.

$$x = x'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.3}$$

$$y = y'(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.4}$$

Zusätzlich zu der radialen Verzeichnung gibt es tangentiale Verzeichnung. Diese tritt auf, wenn die Kameralinse nicht parallel zum Sensor eingebaut ist. Diese Verzeichnung führt dazu, dass Linien, die im idealen Bild parallel verlaufen, nun einen gemeinsamen Fluchtpunkt besitzen. Um diese Verzeichnung zu korrigieren, werden die Faktoren p_1, p_2 benötigt.

$$x = x' + (2p_1 xy + p_2(r^2 + 2x^2)) \tag{2.5}$$

$$y = y' + (p_1(r^2 + 2y^2) + 2p_2 xy) \tag{2.6}$$

2.3 Kalibrierung

Der Kalibrierungsvorgang dient dazu, die intrinsischen und gegebenenfalls extrinsischen Parameter, sowie die Verzeichnungsparameter zu berechnen. Dies erfolgt in mehreren Schritten.

2.3.1 Kalibrierungsmuster

Man braucht ein Kalibrierungsmuster, von dem im nächsten Schritt einige Aufnahmen gemacht werden. Es gibt verschiedene Kalibrierungsmuster, die benutzt werden können. Eine grundlegende gemeinsame Eigenschaft aller Kalibrierungsmuster ist, dass das Muster aus geometrischen Formen besteht, die gut von Bilderkennungsalgorithmen gefunden werden können. Häufig wird dazu ein Schachbrettmuster benutzt, siehe Abbildung 2.1. In diesem Muster haben die Rechtecke eine Kantenlänge von exakt 3cm. Inge samt sind 6 x 8 Rechtecke in dem Muster vorhanden. Der Kalibrierungsalgorithmus sucht nach den Kanten der Quadrate. Der Schnittpunkt von zwei Kanten, also eine Ecke des Quadrats,

bildet den Kalibrierungspunkt. Dadurch enthält ein Bild mit diesem Kalibrierungsmuster 5 x 7, also 35, Kalibrierungspunkte.

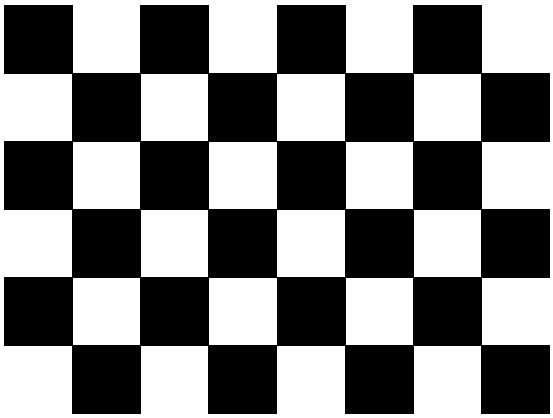

Abbildung 2.1: Schachbrettmuster als Kalibrierungsmuster

Ein anderes Kalibrierungsmuster besteht aus Kreisen, die regelmäßig versetzt zueinander positioniert sind, siehe Abbildung 2.2. In diesem Muster sind die Mittelpunkte der Kreise die Kalibrierungspunkte. In diesem Muster sind 10 x 11 Kreise eingezeichnet, sodass deutlich mehr Informationen für die Kalibrierung in einer Aufnahme vorhanden sind.

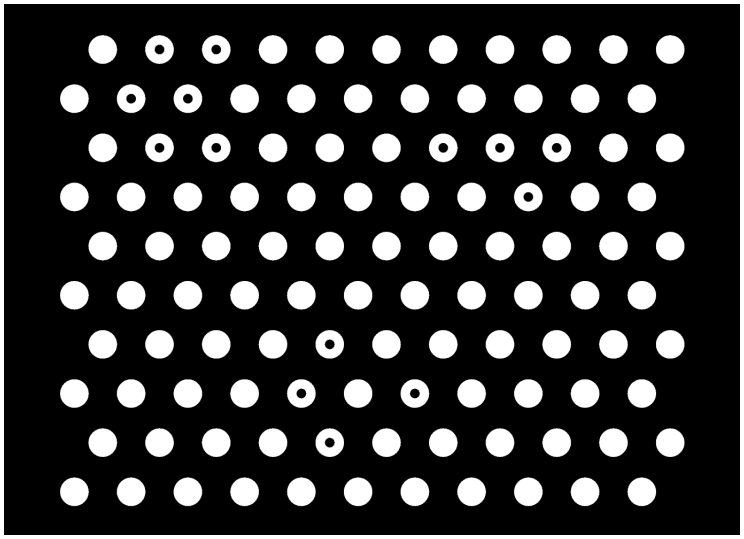

Abbildung 2.2: Punkte als Kalibrierungsmuster

Das Muster ist außerdem speziell für HALCON entwickelt. Während die Kreismuster von OpenCV im ganzen Bild den gleichen Punkt benutzen, sind in diesem Muster einige

Punkte zusätzlich markiert. Dadurch kann HALCON auch dann noch Kalibrierungsinformationen aus einer Aufnahme errechnen, wenn das Kalibrierungsmuster nur teilweise zu sehen ist. Bei dem Schachbrettmuster oder dem Punktmuster von OpenCV muss hingegen immer das ganze Muster erkennbar sein.

Unabhängig davon, welches Muster man einsetzt, muss immer die Größe und Form der ausgedruckten Muster exakt den erwarteten Daten entsprechen. Da manche Drucker versuchen, die Dokumente auf DinA4-Größe anzupassen, muss man für jedes ausgedruckte Muster die Größe und Abstände der geometrischen Objekte überprüfen. Hat das ausgedruckte Muster die korrekte Größe, muss dieses auf eine möglichst ebene Oberfläche geklebt werden. Ein gekrümmtes Muster in einer Aufnahme, die zur Kalibrierung benutzt wird, kann die Ergebnisse verfälschen.

2.3.2 Bildaufnahme

Damit der Kalibrierungsalgorithmus die Parameter berechnen kann, muss er aus mehreren Bildern die Kalibrierungspunkte extrahieren. Dabei wird an die Bilder, die zur Kalibrierung benutzt werden, mehrere Anforderungen gestellt.

Wie bei anderen Fotos auch ist es wichtig, dass das Motiv, also das Kalibrierungsmuster, scharf dargestellt wird und gut belichtet ist. Dabei ist es hilfreich, wenn Kamera und Kalibrierungsmuster auf Stativen befestigt sind.

Das Kalibrierungsmuster sollte außerdem in verschiedenen Distanzen aufgenommen werden. Andernfalls kann es passieren, dass die berechneten Parameter nur auf den Abstand passen, den das Kalibrierungsmuster zur Kamera während der Aufnahmen hatte. Möchte man nun mit diesen Parametern die Kamera benutzen und Objekte mit einem anderen Abstand beobachten, können die Parameter das Bild nun verschlechtern und die Bildinformationen passen nicht mehr zur Realität.

Nicht nur die Entfernung zur Kamera ist wichtig sondern auch die Orientierung des Kalibrierungsmusters. Indem man das Kalibrierungsmuster an jeder Position nach oben und unten sowie nach rechts und links neigt, verbessert man die Qualität der Kalibrierung.

Zuletzt sollte das Kalibrierungsmuster über alle einzelnen Bilder verteilt den ganzen Bildausschnitt füllen. Würden beispielsweise alle Bilder das Kalibrierungsmuster nur in der Bildmitte zeigen, könnten die Parameter so gewählt werden, dass die Bildmitte zwar sehr gut kalibriert ist, aber das Bild an den Bildrändern stark verzeichnet ist.

2.3.3 Berechnung der Parameter

Wurden genügend Bilder aufgenommen, die den Anforderungen entsprechen, können die Parameter berechnet werden. Dazu ist eine initiale Konfiguration der intrinsischen Parameter nötig. Die dafür notwendigen Informationen kann man den Spezifikationen der Kamera oder des Sensors entnehmen. Die Brennweite f wird in der Regel vom Hersteller angegeben. Für das optische Zentrum c_x und c_y kann man die Bildmitte, also die halbe Auflösung des Bildes, annehmen.

Anschließend werden in jedem aufgenommenen Bild die Kalibrierungspunkte aus dem Bild extrahiert und abgespeichert. Als nächstes werden dann mit diesen Informationen die Gleichungssysteme gelöst, die die gesuchten Parameter enthalten. Zusätzlich zu den Parametern berechnen die Algorithmen den durchschnittlichen Fehler in Pixeln. Dieser sollte möglichst klein, am Besten unter 1 sein. Wird das Kalibrierungsmuster nicht in verschiedenen Orientierungen aufgenommen oder es ist nicht im ganzen Bildausschnitt zu sehen, kann dies den Fehler vergrößern.

Es gibt mehrere Frameworks, die man zum Kalibrieren benutzen kann. OpenCV ist eine freie Bibliothek zur Bildverarbeitung und maschinellen Sehen. Es wird kein fertiges Programm angeboten, mit dem der Vorgang direkt gestartet werden kann. Stattdessen existieren Bibliotheken in C++ und Python, in denen der Benutzer Funktionen findet, mit denen er sein eigenes Programm schreiben kann, um eine Kamera zu kalibrieren. Für Benutzer, die sich mit der Materie nicht so gut auskennen, ist dies eine hohe Hürde, Profis können damit aber das Programm auf die eigenen Bedürfnisse anpassen.

Ein anderes Framework ist HALCON. HALCON wird kommerziell vermarktet und ist nicht frei verfügbar. Es werden wie bei OpenCV Bibliotheken für verschiedene Programmiersprachen angeboten, mit denen Benutzer ihre eigenen Programme schreiben können. Zusätzlich existiert mit HDevelop eine integrierte Entwicklungsumgebung. In HDevelop wird eine eigene Programmiersprache benutzt, die weniger tief und detailliert als anderen

Sprachen ist, aber dafür für unerfahrene Benutzer deutlich sprechender und intuitiver zu benutzen ist. Außerdem ist ein Kalibrierungsassistent enthalten, der die aufgenommen Bilder vor der Berechnung der Parameter anhand ihrer Bildschärfe, Belichtung und weiterer Eigenschaften bewertet.

2.3.4 Weitere benutzte Frameworks

ROS

ROS [QCG+09] steht für Robot Operating System. Es ist dazu gedacht, ein Framework für die Programmierung von unterschiedlichen Robotern zur Verfügung zu stellen. Man kann fertige Pakete einbinden, die von anderen Benutzern bereitgestellt werden, zum Beispiel MoveIt!, sieheAbschnitt 2.3.4.

In der Regel startet man mehrere Programme, die jeweils eine bestimmte Aufgabe erfüllen sollen. Es gibt unterschiedliche Möglichkeiten mit denen die Programme untereinander kommunizieren können. ROS stellt Bibliotheken für viele verschiedene Programmiersprachen bereit, damit man eigene Pakete schreiben kann, die sich in das Gesamtsystem integrieren lassen.

MoveIt!

MoveIt! [CSC12] ist ein Framework, dass die Steuerung eines Roboters abstrahiert. MoveIt! stellt Bibliotheken für verschiedene Programmiersprachen bereit, mit denen man die Bewegung des Roboters kontrollieren kann. Möchte man wie in diesem Projekt einen Roboterarm steuern, kann man eine Zielposition und Orientierung für die Spitze des Arms angeben. MoveIt! berechnet für diese Pose dann die Konfiguration der Robotergelenke. Im nächsten Schritt wird dann eine Abfolge von Zwischenkonfigurationen gesucht, damit sich der Roboter von seiner Startkonfiguration zur Zielkonfiguration bewegt, ohne dass er sich selbst oder andere Objekte in seiner Umgebung berührt. Existiert eine solche Abfolge von Konfigurationen, wird diese ausgeführt sodass sich der Roboterarm letztendlich an der gewünschten Position mit der gewünschten Orientierung befindet.

3 Architektur

Für dieses Projekt wurden mehrere Softwareprogramme integriert. In den folgenden Kapiteln wird ein Gesamtüberblick gegeben und die einzelnen Teile werden detailliert beschrieben.

3.1 Gesamtübersicht

In Abbildung 3.1 ist die Architektur graphisch dargestellt. Jedes Element stellt dabei eine ROS-Node dar, also ein einzelnes Programm.

Die beiden dunkelblauen Elemente ur_moder_driver und libuvc_camera steuern direkt die Hardware an. im Fall von ur_modern_driver wird der UR5-Arm gesteuert, libuvc_camera steuert die Webcam.

Das hellblaue Element universal_robot stellt eine abstrahierte Schnittstelle zur Steuerung des Roboterarms zur Verfügung.

Die beiden grünen Elemente MoveArmServer und caltab_detector_node wurden im Rahmen dieser Bachelorarbeit geschrieben. MoveArmServer bietet Funktionen zur Steuerung des Arms, die speziell in diesem Projekt benötigt werden. caltab_detector_node kümmert sich um das Verarbeiten der Bildinformationen.

Das gelbe Element CalibrationController wurde ebenfalls im Rahmen dieser Bachelorarbeit entwickelt. Dieses Programm steuert den gesamten Ablauf, kommuniziert mit den beiden gelben Elementen und sorgt dafür, dass der Roboter sich an die gewünschten Positionen bewegt und die Bilder im richtigen Moment aufgenommen werden. Zum Schluss werden dem Benutzer dann die ermittelten Parameter zurückgegeben.

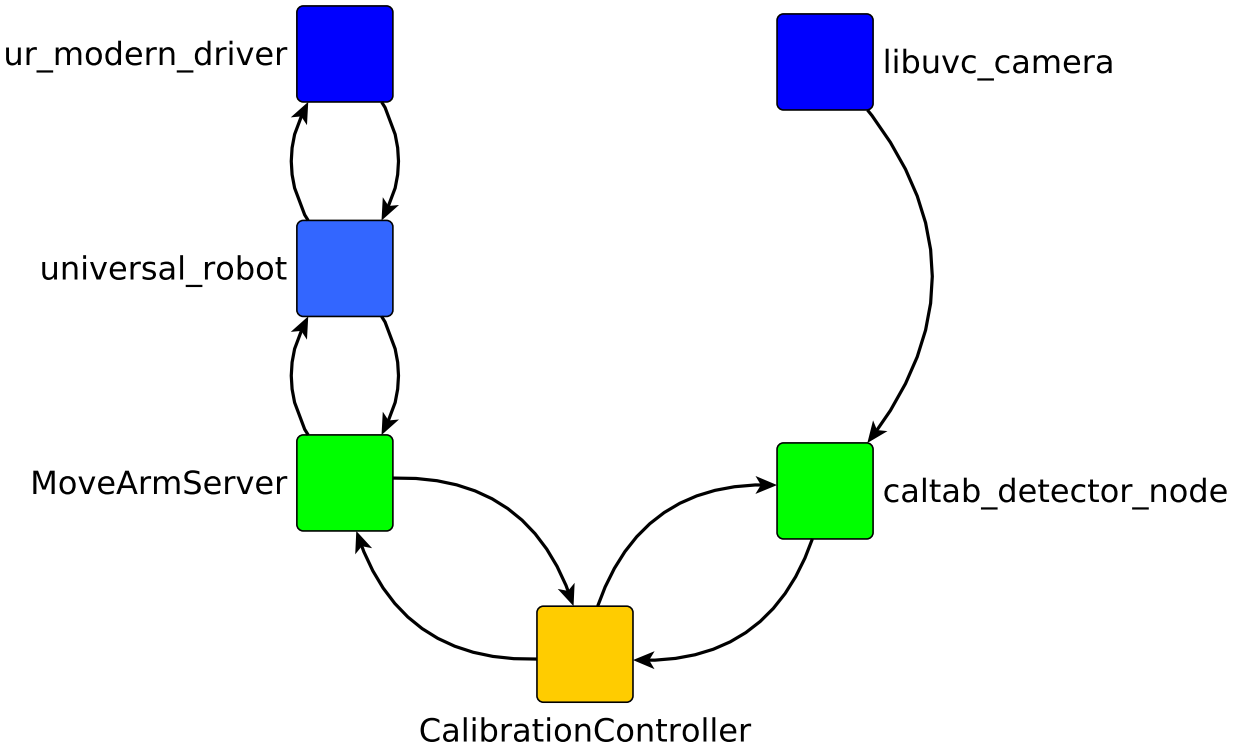

Abbildung 3.1: Übersicht über die Architektur

3.2 ur_modern_driver

Diese Node wurde bereits als Paket in ROS veröffentlicht, siehe [And15]. Sie bietet eine Schnittstelle zwischen dem Roboterarm UR5 und dem ROS Framework. Da dieses Paket nicht für die in diesem Projekt benutzte ROS Version entwickelt wurde, mussten einige wenige Anpassungen am Sourcecode vorgenommen werden, um das Paket in diesem Projekt zu benutzen.

3.3 universal_robot

Diese Node wurde ebenfalls als Paket in ROS veröffentlicht, siehe [EGH+17]. In dieser Node ist MoveIt! integriert, welches zum Bewegen des Roboterarms benutzt wird. Man kann diesem Programm die gewünschte Position und Orientierung des Roboterarms

übergeben, und das Programm führt die erforderliche Bewegung dann durch, sofern die Position und Orientierung für den Arm erreichbar sind.

An diesem Programm wurden einige Anpassungen vorgenommen. Zum einen wurde die in MoveIt! benutzte Bibliothek zur Lösung der inversen Kinematik KDL durch TRAC_IK ausgetauscht. Während der Versuche kam es vor, dass MoveIt! keinen oder nur einen viel zu komplizierten Weg von der Start zur Zielkonfiguration gefunden hat. Dieses Problem konnte teilweise durch diesen Austausch behoben werden.

Zusätzlich wurde das URDF in diesem Paket angepasst. Das URDF enthält geometrische Informationen über den Roboterarm wie z.B. die Länge und Form der einzelnen Glieder. Im Standardfall soll der Endpunkt des Roboterarms an die übergeben Position bewegt werden. In diesem Projekt ist allerdings das Kalibrierungsmuster an dem Endpunkt befestigt. Es ist allerdings nicht mittig an dem Punkt montiert sondern steht zu einer Seite ab. Daher wurde das URDF so verändert, dass nun der Mittelpunkt des Musters der Endpunkt für den Roboterarm ist. Dadurch lässt sich das Muster einfacher an die gewünschten Positionen bewegen.

Diese Node wird im Projekt von dem Programm MoveArmServer angesprochen. Dazu stellt sie über MoveIt! einige Schnittstellen bereit, die in diesem Fall benutzt werden, um die Position und Orientierung des Kalibrierungsmusters zu verändern.

3.4 libuvc_camera

Auch diese Node ist als Paket in ROS verfügbar. Sie dient dazu, das Bild der angeschlossen Webcam über ROS verfügbar zu machen. In dem Script, mit dem diese Node gestartet wird, lassen sich verschiedene Parameter einstellen. In diesem Fall wurde mittels der Vendor-ID und Product-ID die zu benutzende Kamera definiert, die Auflösung wurde auf Full-HD gesetzt und der Fokus wird so eingestellt, dass der Bereich, in dem sich das Kalibrierungsmuster befinden wird, scharf zu sehen ist.

Die Node veröffentlicht die Bilder der Kamera über sogenannte Topics. Das Programm caltab_detector_node empfängt die Bilder und wertet sie aus.

3.5 MoveArmServer

Dieses Programm wurde im Rahmen dieser Bachelorarbeit selber entwickelt. Es dient dazu, dem CalibrationController eine weitere Abstraktionsschicht zum Bewegen des Arms zu bieten.

Dazu wurde ein Actionserver erstellt, der die Position des Kalibrierungsmusters als Auftrag erhält. Zusätzlich kann als weiterer Parameter für den Auftrag die Neigung des Musters angegeben werden. Das Programm berechnet dann für die gewünschte Position zunächst die Orientierung für das Muster damit dieses senkrecht zur Kamera steht. Anschließend wird die Orientierung mit der gewünschten Neigung berechnet. Diese Zielkonfiguration wird dann an MoveIt! übergeben. Abhängig davon, ob MoveIt! den Arm erfolgreich bewegen konnte oder nicht, informiert der MoveArmServer den CalibrationController, ob der Auftrag erfolgreich ausgeführt wurde oder nicht.

3.6 caltab_detector_node

Auch dieses Programm wurde selbst entwickelt. Es benutzt die Bibliothek von HALCON, um in den Bildern, die es von der Kamera empfängt, nach dem Kalibrierungsmuster zu suchen und die Kalibrierung durchzuführen.

Diese Node bietet zwei Actionserver an. Der erste Actionserver erwartet einen Auftrag, um in den aktuellen Bildern nach dem Kalibrierungsmuster zu suchen. Dazu wird ein Parameter übergeben, wie viele Bilder ausgewertet werden sollen. Die Algorithmen von HALCON finden nicht immer im ersten Versuch das Muster, daher lässt sich die Anzahl der Bilder definieren. Die Node meldet dem CalibrationController zurück, ob nach der angegeben Anzahl der Bilder das Muster gefunden wurde oder nicht.

Der zweite Actionserver nimmt den Auftrag zur Durchführung der Kalibrierung an. Dazu werden die Bilder, in denen das Muster zu sehen war, ausgewertet. Danach werden die errechneten intrinsischen Parameter, die Verzeichnungsparameter sowie der ermittelte durchschnittliche Fehler an den CalibrationController zurückgegeben.

3.7 CalibrationController

Diese letzte Node wurde ebenfalls selbst entwickelt. Sie dient dazu, den gesamten Ablauf der Kalibrierung zu steuern, indem sie die Actionserver mit Aufgaben versorgt. Zunächst übergibt der Benutzer dem Programm die Position der Kamera in Relation zur Basis des Roboterarms und die initialen intrinsischen Parameter. Dann wird aus diesen Werten das Sichtfeld der Kamera bestimmt. Mit diesem Sichtfeld werden dann die verschiedenen Distanzen zur Kamera berechnet und anschließend die Positionen für das Kalibrierungsmuster, damit dieses das ganze Sichtfeld der Kamera abdeckt.

Diese Positionen werden der Reihe nach an den MoveArmServer übergeben. Für jede Position wird das Muster für um 10°, 20°, 30° und 40° nach oben, unten, rechts und links geneigt. Hat der Actionserver den Auftrag erfolgreich ausgeführt, wird der caltab_detector_node der Auftrag übergeben, in dem Kamerabild nach dem Muster zu suchen. Ist dieser Auftrag abgeschlossen, wird die nächste Position angesteuert.

Wurden alle Positionen abgearbeitet, wird zum Schluss der Auftrag zur Kalibrierung durchgeführt. Die dadurch erhaltenen Parameter werden dem Benutzer angezeigt und das Programm hat seine Durchführung beendet.

Literaturverzeichnis

[And15] Andersen, Thomas T.: Optimizing the Universal Robots ROS driver. / Technical University of Denmark, Department of Electrical Engineering. Version: 2015. http://orbit.dtu.dk/en/publications/optimizing-the-universal-robots-ros-driver(20dde139-7e87-4552-8658-dbf2cdaab.html. 2015. – Forschungsbericht

[CSC12] Chitta, S.; Sucan, I.; Cousins, S.: MoveIt! [ROS Topics]. In: *IEEE Robotics Automation Magazine* 19 (2012), March, Nr. 1, S. 18–19. http://dx.doi.org/10.1109/MRA.2011.2181749. – DOI 10.1109/MRA.2011.2181749. – ISSN 1070–9932

[EGH+17] Edwards, Shaun; Glaser, Stuart; Hawkins, Kelsey; Meeussen, Wim; Messmer, Felix: *universal_robot*. https://github.com/ros-industrial/universal_robot, 2017

[QCG+09] Quigley, Morgan; Conley, Ken; Gerkey, Brian P.; Faust, Josh; Foote, Tully; Leibs, Jeremy; Wheeler, Rob; Ng, Andrew Y.: ROS: an open-source Robot Operating System. In: *ICRA Workshop on Open Source Software*, 2009

[Zha00] Zhang, Z.: A flexible new technique for camera calibration. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22 (2000), Nov, Nr. 11, S. 1330–1334. http://dx.doi.org/10.1109/34.888718. – DOI 10.1109/34.888718. – ISSN 0162–8828

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe, dass alle Stellen der Arbeit, die wörtlich oder sinngemäß aus anderen Quellen übernommen wurden, als solche kenntlich gemacht und dass die Arbeit in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt wurde.

Ort, Datum Unterschrift