Institut für Informatik VII
Robotik und Telematik

Masterarbeit

Erweiterte Verfeinerung der Detektierung von AprilTags für hochpräzise Andockmanöver von mobilen Robotern

Jan Richter

November 2021

Erstgutachter:	Prof. Dr. Andreas Nüchter
Zweitgutachter:	Prof. Dr. Klaus Schilling
Betreuer:	M.Sc. David Bohling

Danksagungen

Besonderer Dank gilt Herrn Prof. Dr. Andreas Nüchter, der die Bearbeitung des Themas ermöglichte. Des Weiteren möchte ich mich bei M.Sc. David Bohling bedanken, der mich durch Gerätebereitstellung sowie Ratschlägen unterstützt hat. Außerdem bin ich den Korrekturlesern B.Sc. Alexander Ehrlich und B.Sc. Darius Deubert zu Dank verpflichtet, durch die diese Arbeit orthografisch, grammatikalisch und inhaltlich enorm verbessert wurde.

Zusammenfassung

Die Entwicklung von Modulen und Routinen in eine bestehende, serverbasierte Kontrollstation zur Abwicklung von Anfahrts- und Andockmanövern von beliebigen mobilen Systemen ist die zentrale Leistung dieser Arbeit. Ein effizienter Materialfluss in einer digitalisierten Fabrikumgebung setzt voraus, dass Waren automatisch auf Transportroboter geladen und entladen werden. Zu diesem Zweck ist eine genaue Positionierung des Roboters an eine Verladestation notwendig. Dies ist auch bei der Annäherung an eine Andockstation erforderlich, um beispielsweise die Energiespeicher des Roboters automatisch aufzuladen. Abgesehen von diesen Anwendungen werden Roboter darüber hinaus immer mehr für Montagearbeiten in Produktionsstätten eingesetzt. Die Möglichkeit einen Transportroboter als mobile Montageplattform an Assemblerstationen mit Manipulatoren einzusetzen, wird in dieser Arbeit untersucht. Ausschlaggebend hierfür ist die Fähigkeit eine subzentimetergenaue Position relativ zur Station wiederholbar einzunehmen und zusätzlich die präzise Endposition an die Station über Schnittstellen zu übermitteln. Die entscheidende Komponente ist dabei die Schätzung der Pose des mobilen Systems, die mittels Kameraaufnahmen von künstlichen Markern, angebracht an den Stationen, bestimmt wird. Da die Präzision einer visuellen Lokalisierung stark von der Detektionsgenauigkeit der Marker abhängt, wird ein Optimierungsproblem formuliert, um mittels Kantenmodellfunktion die Kanten der Marker anzunähern und somit deren Schnittpunkte auf der Bildebene noch genauer zu bestimmen. Schließlich wird eine planare kontinuierliche Trajektorienplanung für mobile Roboter mit Ackermann Lenkung implementiert, um die Segmente eines möglichen Pfades zwischen zwei beliebigen Posen zu bestimmen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

In den letzten Jahren hat die Robotik dank des technologischen Fortschritts immer neue Bereiche, wie Lagerhallen, Produktionsstätten, bis hin zu privaten Haushalten, erobert. Teil- oder vollautonome mobile Roboter, die Menschen unterstützen, Güter transportieren und andere autonome Funktionen ausführen, finden in fast allen Branchen Anwendung. Eine wichtige Voraussetzung zur Erfüllung eines beliebigen Arbeitsablaufs ist jedoch, verschiedenste Stationen präzise anzufahren. Um beispielsweise bei der Warenbeförderung in einer digitalisierten Fabrikumgebung von Nutzen zu sein, muss ein effizienter Materialfluss durch automatisches Be- und Entladen von Transportrobotern ermöglicht werden. Mögliche Stationen in einem solchen Szenario beinhalten Be- und Entladestationen, designierte Wartestationen oder Dockingstationen zum Aufladen der Batterien. Sollen zudem Waren präzise auf dem Transportroboter abgelegt werden oder sogar Bauteile von Assemblerstationen auf dem Roboter montiert werden, bedarf es nicht nur einer genauen und wiederholbaren Anfahrt, sondern auch einer hochpräzisen Positionserfassung in Relation zu der Station. Dies ermöglicht Industrierobotern mit Manipulator Arbeiten auf einer mobilen Roboterplattform im Submillimeterbereich durchzuführen.

Am Zentrum für Telematik e.V. (ZfT) wird seit langem an einer Kontrollstation für autonome mobile Systeme gearbeitet, die über eine intuitive browserbasierte Benutzeroberfläche das Verwalten beliebig vieler Roboter ermöglichen soll. Jedoch gibt es noch keine Interaktionsmöglichkeit zwischen den autonomen Systemen und Stationen. Die in dieser Arbeit implementierten Module sollen die Grundlage hierfür bilden, indem sie das Problem der Annäherung lösen und eine erste Schnittstelle zum Informationsaustausch bereitstellen.

Andere Projekte des ZfT beinhalten die innerbetriebliche Montage von Kleinstsatelliten. Die einzelnen Komponenten und Subsysteme von Satelliten werden modular nach dem Baukastenprinzip zusammengesetzt und über standardisierte Schnittstellen verbunden. Dabei bedarf es bei vielen Fertigungsschritten einem Höchstmaß an Präzision, wie beispielsweise bei der Platzierung der Solarpaneele an den Rahmen des Satelliten. Unter anderem aus diesem Grund plant das

ZfT durch fortgeschrittene, roboterunterstützte Produktionstechniken Teile dieser Arbeit von Industrierobotern durchführen zu lassen und schließlich komplexe zusammenhängende Fertigungsschritte bis hin zur vollautomatisierten Herstellung zu realisieren [1]. Die Ergebnisse dieser Arbeit sollen zur Verwirklichung dieses Vorhabens beitragen.

1.2 Problembeschreibung

Wie zuverlässig eine Position wiederholbar und präzise von einem Vehikel mit visueller Lokalisierung eingenommen werden kann, hängt vorwiegend von der Qualität der visuellen Erfassung und der Genauigkeit der vehikelinternen Fortbewegungsmethode ab.

Jedoch kann die vehikelseitige Abarbeitung einer Trajektorie auch noch so gut sein - wenn das Kamerasystem eine ungenaue Pose schätzt, wird die angestrebte Position garantiert nicht erreicht. Allerdings handelt es sich hierbei um eine Koppelnavigationsmethode (Dead Reckoning oder Odometrie) und unterliegt daher ganz eigenen Fehlerquellen. Wenn beispielsweise die Wegstrecke anhand der Anzahl der Umdrehungen eines Rades gemessen wird, stellt jede Abweichung zwischen der tatsächlichen und der angenommenen zurückgelegten Strecke pro Umdrehung, etwa aufgrund von Schlupf oder Oberflächenunregelmäßigkeiten, eine Fehlerquelle dar. Da jede Schätzung der Position relativ zur vorigen ist, kumulieren sich die Fehler im Laufe der Zeit. Um das Aufaddieren der Abweichungen so gering wie möglich zu halten, werden nur kurze Pfade von wenigen Metern geplant und um den Schlupf zu verringern, werden die Übergänge zwischen den Pfadsegmenten mit Klothoiden (Kurven mit stetiger Krümmung) angenähert. Dies wird später in Detail im Abschnitt 2.6 erläutert.

Die unabdingbare Voraussetzung ist jedoch die hohe Qualität der visuellen Erfassung der künstlichen planaren Marker (Fiducials) und nimmt daher den Großteil dieser Arbeit ein. Der Grund dafür ist, dass sich Abweichungen bei der Kalibrierung des Kamerasystems durch alle nachkommenden Abläufe ziehen. Je genauer die Positionsbestimmung der Merkmale auf der Bildebene der Kameraaufnahmen, desto genauer die Kalibrierung des Kamerasystems und folglich desto genauer die Posenschätzung - einerseits durch die Güte der Kalibrierung, andererseits durch die präziser erfassten Merkmale der Marker.

1.3 Stand der Technik

Relative Positionierung ist seit vielen Jahren Gegenstand der Forschung und wurde in zahlreichen Studien mit unterschiedlichen Ansätzen untersucht. De Ponte Müller [2] gibt einen umfangreichen Überblick über verschiedene Sensoren zur Schätzung der relativen Position von autonomen Fahrzeugen. Dazu gehören egozentrische oder nicht-kooperative Ansätze wie Radar, Lidar sowie Monokulare-, Stereo- und Time-of-Flight-Kamerasysteme.

Ein sehr simples, aber robustes Positionierungssystem wird von der Roboterplattform Kobuki [3] verwendet. Drei Infrarotstrahler unterteilen das Gebiet vor einer Station in eine linke, rechte und mittlere Region. Der Roboter, der sich in einer der Regionen befinden kann, ist mit drei

Infrarotempfängern - links, rechts und an der Front - ausgestattet. Befindet sich der Roboter in der zentralen Region, muss er einfach dem Signal des zentralen Senders folgen, bis er die Station erreicht. Befindet sich der Roboter in der linken Region, muss er sich gegen den Uhrzeigersinn drehen, bis der rechte Sensor das Signal der linken Region erkennt. In dieser Position blickt der Roboter auf den zentralen Bereich, auf den er sich zubewegt, bis der rechte Sensor das Signal des zentralen Infrarotstrahlers erkennt. Danach muss er sich nur noch im Uhrzeigersinn drehen, bis der frontale IR-Sensor den zentralen Infrarotstrahler erkennt und er ihm wieder folgen kann. Der Ablauf für die rechte Region verläuft analog. Ein auf ähnliche Weise funktionierendes automatisches Aufladesystem wurde von Doumbia et al. [4] auf der ICCAR 2019 vorgestellt.
In Quilez et al. [5] werden ebenfalls IR Sensoren verwendet, jedoch nur zu Distanzmessung und Hinderniserkennung. Zur relativen Positionierung verwenden sie QR-Codes, um dann je nach Entfernung zur Station verschiedene Annäherungsstrategien durchzuführen. Hier wird also bereits ein visueller Ansatz mit künstlichen Markern untersucht, jedoch soll sich nicht ausschließlich auf die visuelle Mustererkennung verlassen werden, um potenzielle Fehler durch die Kamerakalibrierung zu vermeiden.

Eine rein visuelle, auf künstliche Markererkennung basierende Methode stellt Alijani in [6] vor, bei dem in Echtzeit die relative Position des Roboters bezüglich AR-Tags - künstliche Marker ähnlich der QR-Codes - berechnet wird und entsprechende Kommandos an die Aktuatoren übergeben werden, um eine Zielregion mit 4 cm Durchmesser zu erreichen.
Auch Mateos [7] beschränkt sich in seiner Arbeit auf visuell detektierte künstliche Marker, um schwimmende Roboterschwärme aneinander andocken zu lassen. Die von ihm verwendeten Marker nennen sich AprilTags und werden noch in Abschnitt 2.2.1 genauer behandelt. Für eine erfolgreiche Verbindung ist eine relative Gierwinkelgenauigkeit von $\pm 27.5\,°$, dank einer trichterförmigen Führung, ausreichend, bei einer lateralen Abweichung von ± 4 cm. Besonders interessant ist die Verwendung der sogenannten “AprilTags3D”-Methode, bei der zwei angewinkelte LCD-Displays AprilTags anzeigen, anstelle von rein planaren Markern, wodurch die Detektierungsrate erhöht und der Gierwinkelfehler verringert wird. Zudem kann durch die Displays dynamisch auf verschiedene Szenarien reagiert werden.

Noch spielen visuell detektierte künstliche Marker (Fiducials) in Andockmethoden eine untergeordnete Rolle, da robustere und einfacher umzusetzende alternative Sensoren zur Schätzung der relativen Position existieren. Daher gibt es in dem Gebiet noch nicht viele Ansätze zur Verbesserung der Detektierungsgenauigkeit im Allgemeinen. Jedoch ist die präzise Erfassung von Fiducials in vielen Bereichen von Interesse wie z.B. der Lokalisierung, Kalibrierung oder als Passermarker für automatisierte Fertigungsverfahren. Bereits 2004 haben Abawi et al. [8] den Einfluss der relativen Orientierung der künstlichen Marker zur Kamera untersucht und hierin die Hauptursache für systematische Fehlschätzungen und erhöhte Standardabweichungen identifiziert. So stellen Abbas et al. [9] direkt drei Methoden vor, die diesen Einfluss bei einer AprilTag basierten Zustandsschätzung reduzieren. Zuerst verfolgt ein speziell angefertigtes Gierachsen-Gimbal das Zentrum des Tags in Echtzeit. Zusätzlich erfolgt eine trigonometrische Korrektur des Gierwinkels, um die Kamera auf das Zentrum des Tags auszurichten und letztlich wenden sie ein probabilistisches Monte-Carlo-Sensorfehlermodell für AprilTags an.
Selbstverständlich tragen die Entwickler der AprilTag-Bibliothek ebenfalls Methoden zur Verbesserung der Detektierung bei. Wang und Olson stellen in [10] eine verbesserte Kantenerkennung

vor, um aus diesen eine bessere Eckpunkt-Positionsapproximierung zu erhalten. Auf ähnliche Weise soll auch in dieser Arbeit die Detektierungsgenauigkeit erhöht werden, aufgrund dessen auf die Methode von Wang und Olson in Abschnitt 2.2.3 noch genauer eingegangen wird.

Kapitel 2

Konzept

In diesem Kapitel werden die grundlegenden Methoden und mathematischen Hintergründe der Module zur visuellen Positionsbestimmung und der Trajektorienplanung behandelt. Wie zuvor erwähnt nimmt die Optimierung der visuellen Erfassung der Fiducials zur exakten Positionsbestimmung den Großteil dieser Arbeit ein, was in diesem Kapitel, das sich bis auf die letzten zwei Abschnitte ausschließlich mit diesem Thema beschäftigt, besonders deutlich wird.

2.1 Posenschätzung

Eines der elementarsten Probleme in Computer Vision ist die Schätzung der Kamerapose in Bezug auf ein Objekt, um beispielsweise im Bereich der Augmented Reality realistisches 3D-Rendering zu betreiben oder im Bereich der Robotik die Pose eines Objekts zu bestimmen und dieses dann zu manipulieren.

In diesem Abschnitt werden die Grundlagen erläutert, die, beginnend mit der mathematischen Beschreibung des Kameramodels, die Berechnung perspektivischer Projektionen ermöglicht, woraus sich schließlich die Pose, bestehend aus Position und Orientierung, der Kamera schätzen lässt. Ein Großteil des wissenschaftlichen Beitrags dieser Arbeit beruht auf der Maximierung der Präzision dieser Schätzung, da sich alle Ungenauigkeiten bei der Lokalisierung der Kamera unumstößlich auf die restlichen Berechnungen des Andockmanövers auswirken.

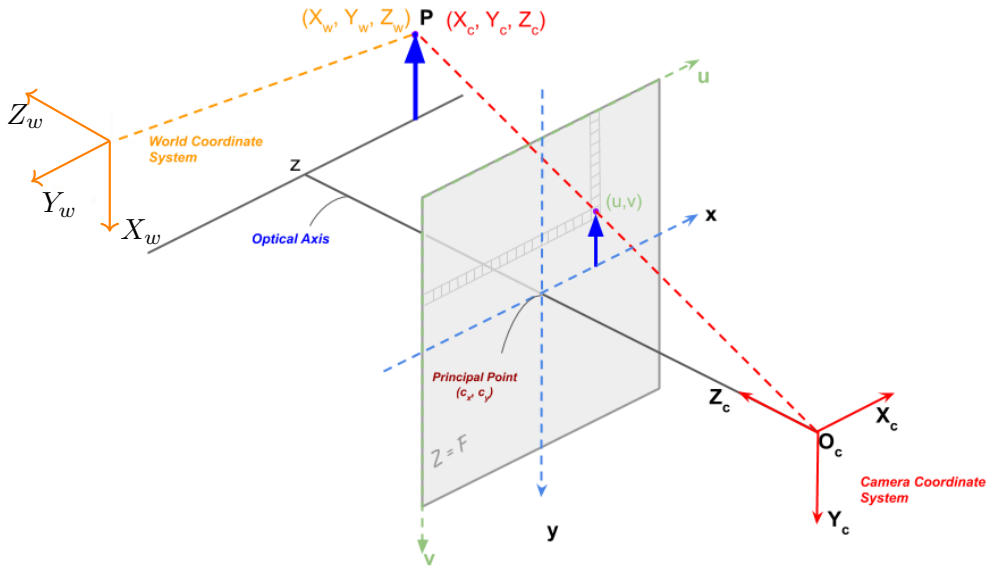


Abbildung 2.1: Illustration eines Lochkameramodells nach [12].

2.1.1 Kameraprojektion

Beim Abbilden einer dreidimensionalen Szene wird diese auf eine zweidimensionale Ebene, der Bildebene, projiziert. Bei diesem Prozess geht also eine Dimension verloren, wodurch es nicht mehr ohne weiteres möglich ist, Aussagen über die Tiefe von Punkten im Bild zu treffen. Allerdings lässt sich feststellen, dass sich ein Punkt (X_w, Y_w, Z_w) in der Szene, der korrespondierende Punkt (u, v) in der Bildebene und das Projektionszentrum $\mathcal{O}_c$ der Kamera auf einer Linie befinden. Mithilfe homogener Koordinaten[1] und einer Projektionsmatrix werden so korrespondierende Punkte, bis auf die Skalierung, in Beziehung gebracht.

Zusammenfassend lässt sich die Beziehung einer projektiven Kamera auf einen Punkt im Raum in Form einer linearen Abbildung homogener Koordinaten wie folgt ausdrücken:

$$s \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K_{3\times3}[R|t]_{3\times4} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \tag{2.1}$$

In Abbildung 2.1 wird diese Beziehung durch die rote gestrichelte Linie illustriert, auf der jeder Punkt durch Variation des Skalierungsparameters s erreichbar ist. K ist die Kameramatrix und $[R|t]$ die Konkatenation der Rotationsmatrix R und des Translationsvektors t, die die

[1] Für weitere Informationen über homogene Koordinaten wird auf das Buch von Hartley et al. [11] verwiesen.

Änderung der Koordinaten vom Welt- zum Kamerakoordinatensystem beschreiben. Diese Art der Kameraprojektion, durch den Ursprung des Kamerakoordinatensystems $\mathcal{O}_c$, wird allgemein "zentrale Projektion" genannt und wird unter anderem von dem Lochkameramodell und dem Dünne-Linse-Modell verwendet. Dabei ist der variable Skalierungsparameter s der projektiven Transformation nicht Teil des Kameramodells.

Die Kameramatrix K projiziert 3D Punkte im Kamerakoordinatensystem auf 2D Punkte in Pixelkoordinaten auf der Bildebene:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = K_{3\times3} \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} \tag{2.2}$$

Sie enthält die fünf intrinsischen Parameter der Kamera, die beim Kalibrieren ermittelt werden.

$$K = \begin{bmatrix} f_x & \gamma & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

Die jeweils in Pixel angegebene Brennweite $f_{x/y}$, der Bildmittelpunkt (c_x, c_y) und der Skew-Koeffizient γ zwischen der x- und y-Achse. Im Folgenden wird zur Vereinfachung $\gamma = 0$ angenommen, da der Skew in vielen Fällen null beträgt oder vernachlässigbar gering ist.

Die Bestimmung der intrinsischen Parameter ist von äußerster Bedeutung, da sich Ungenauigkeiten hier durch alle weiteren Berechnungen fortsetzen. Die Unabhängigkeit der Kameramatrix von äußeren Parametern ermöglicht jedoch, dass diese nur einmal bestimmt werden muss, solange sich die Brennweite beziehungsweise der Zoom des Objektivs nicht verändert.

Durch homogene Transformation bildet die Rotations- und Translationsmatrix $[R|t]$ 3D Punkte im Kamerakoordinatensystem auf 3D Punkte im Weltkoordinatensystem ab:

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{pmatrix} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \tag{2.4}$$

Kombiniert ergibt die projizierende und homogene Transformation die projektive Transformation $\mathcal{P}$, die 3D Punkte aus der Szene auf die Bildebene abbildet. Für $Z_c \neq 0$ ist die Gleichung 2.1 äquivalent zu

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} f_x \frac{X_c}{Z_c} + c_x \\ f_y \frac{Y_c}{Z_c} + c_y \end{pmatrix} \quad \text{mit} \quad \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} = [R|t] \begin{pmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{pmatrix} \tag{2.5}$$

Dieses Lochkameramodell reicht jedoch nicht aus, um reale Objektive zu beschreiben. Konventionelle refraktive Objektive verwenden meist eine Vielzahl verschiedenster Linsen, um chromatischen Aberrationen entgegenzuwirken, die oft, aufgrund von Produktionsfehlern oder gewollt (bei z.B. Weitwinkelobjektiven), mit gewissen radialen und tangentialen Verzerrungen (k und p respektive) behaftet sind.

Daher wird das obige Modell wie folgt erweitert:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} f_x x'' + c_x \\ f_y y'' + c_y \end{pmatrix} \tag{2.6}$$

$$\begin{pmatrix} x'' \\ y'' \end{pmatrix} = \begin{pmatrix} x' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + 2p_1 x' y' + p_2(r^2+2x'^2) \\ y' \frac{1+k_1 r^2+k_2 r^4+k_3 r^6}{1+k_4 r^2+k_5 r^4+k_6 r^6} + p_1(r^2+2x'^2) + 2p_2 x' y' \end{pmatrix} \tag{2.7}$$

mit

$$r^2 = x'^2 + y'^2 \quad \text{und} \quad \begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \frac{X_c}{Z_c} \\ \frac{Y_c}{Z_c} \end{pmatrix}$$

2.1.2 Perspective-n-Point Problem

Im vorigen Unterabschnitt wurde bereits das Problem der Projektion von korrespondierenden 3D und 2D Punkten erläutert. Im allgemeinsten Fall lässt sich mit sechs Punktpaaren mittels direkter linearer Transformation (DLT) [13] die sechs Freiheitsgrade der Pose und die fünf intrinsischen Kalibrierungsparameter ermitteln. Um also die 11 Unbekannten der Matrix $\mathcal{P} = K[R|t]$ aus Gleichung 2.1 zu bestimmen, wird mit mindestens sechs verschiedenen, korrespondierenden Punktpaaren ein Gleichungssystem mit 12 Gleichungen erstellt und gelöst. Für ein Punktpaar sieht dies folgendermaßen aus:

$$\begin{bmatrix} -X & -Y & -Z & -1 & 0 & 0 & 0 & 0 & uX & uY & uZ & u \\ 0 & 0 & 0 & 0 & -X & -Y & -Z & -1 & vX & vY & vZ & v \end{bmatrix} vec(\mathcal{P})_{12\times 1} = 0$$

$$M_{2\times 12}\ vec(\mathcal{P})_{12\times 1} = 0 \tag{2.8}$$

Die Projektionsmatrix $\mathcal{P}$ wird anschließend aus der resultierenden Matrix M mit mindestens 6 Punktpaaren mittels Singulärwertzerlegung [14] extrahiert.

Es gibt jedoch eine Vielzahl von Algorithmen, die die Genauigkeit der DLT verbessern. Einer der Geläufigsten geht von bekannten Kalibrierungsparametern aus und nennt sich Perspective-n-Point (PnP) Algorithmus, nach dem erstmals von Fischler und Bolles 1981 beschriebenen Perspective-n-Point Problem in [15]. OpenCV - eine freie Programmbibliothek für Bildverarbeitung und Computer Vision - liefert vier verschiedene Ansätze das PnP Problem zu lösen, welche die Rotationsmatrix R und den Translationsvektor t zurückliefern und zusammen mit den Kalibrierungsparametern wiederum die Projektionsmatrix $\mathcal{P}$ bilden. [16]

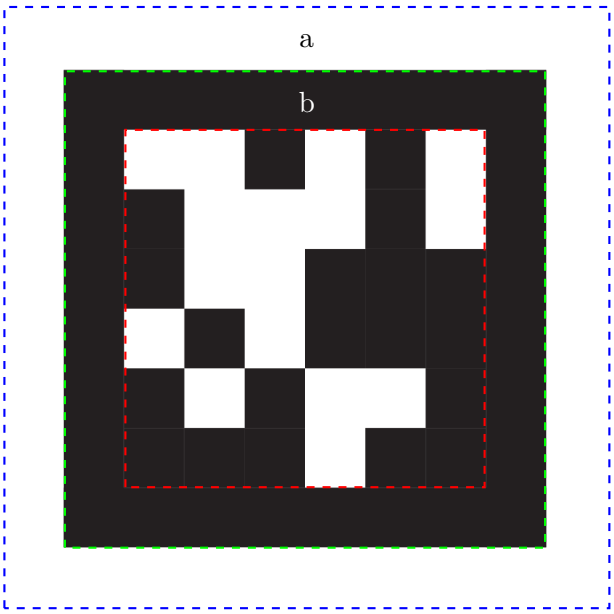


Abbildung 2.2: Darstellung des AprilTags ID 0 der Familie 36h11, die in dieser Arbeit verwendet werden. AprilTags bestehen aus einem obligatorischen weißen (a) und schwarzen (b) Rahmen und einer variablen Menge an Datenbits (innerhalb des rot gestrichelten Rahmens). [17]

2.2 AprilTag

Während die Verwendung von natürlich vorkommenden Merkmalen ein zentraler Schwerpunkt der maschinellen Wahrnehmung ist, spielen künstliche Merkmale (Fiducials) eine wichtige Rolle bei der Erstellung einer Ground Truth oder von kontrollierbaren Experimenten und vereinfachen die Entwicklung einer Vielzahl von Computer-Vision-Systemen wie Augmented Reality, Robotik und Kamerakalibrierung. [18]

Für die hochpräzise Schätzung der Pose der Kamera ist es nötig 3D Punkte aus der Umgebung und deren 2D Projektion auf der Bildebene mit Subpixelgenauigkeit zu ermitteln, um mit diesen Punktpaaren und dem im vorigen Unterabschnitt beschriebenen Perspective-n-Point Algorithmus die Rotationsmatrix und den Translationsvektor zu bestimmen. Die Punkte natürlicher Merkmale auf der Bildebene zu lokalisieren, ist oft weder zuverlässig noch präzise. Zudem ist die anschließende Registrierung in ein gemeinsames Weltkoordinatensystem aufwändig und mit Unsicherheiten behaftet. Aus diesen Gründen werden häufig künstliche Merkmale (Fiducials) in die Szene eingebracht. Eine Möglichkeit dies umzusetzen ist die Verwendung von künstlichen Markern, wie dem bereits erwähnten QR-Code, AR-Tag und AprilTag. In diesem Abschnitt wird genauer auf das AprilTag eingegangen und die in dieser Arbeit verwendete Methode zur automatischen Erfassung und Verfeinerung von Punktpaaren vorgestellt.

2.2.1 AprilTag Marker

AprilTags sind ähnlich zu QR-Codes. Sie wurden allerdings speziell zur robusten Identifikation auf weite Entfernungen und zur Berechnung der genauen 3D Position und Ausrichtung der Tags relativ zur Kamera entwickelt. Wie bei QR-Codes bestehen die "Pixel" aus Bits. Abbildung 2.2 veranschaulicht den Aufbau von AprilTags. Sie sind von einem obligatorischen weißen und schwarzen Rahmen umschlossen, welcher jeweils ein Bit breit ist. Innen enthalten sie eine variable Menge an Datenbits. Anders als QR-Codes speichern sie keine benutzerdefinierten Informationen, sondern lediglich eine ID, die sie identifizierbar macht und einer vordefinierten Familie zuweist. Geläufige Familien sind 16h5, 25h9, 36h10, und 36h11. Die Zahl hinter dem "h" bezeichnet den Hamming-Abstand - ein Maß für die Unterschiedlichkeit zwischen zwei Markern der Familie. Je höher, desto robuster gegenüber falsch-positiven Erkennungen, wie beispielsweise Verwechslungen nicht existierender Marker in zufälligen Bildtexturen oder in Bildrauschen. Aus diesem Grund wird die 36h11 Familie bevorzugt und auch in dieser Arbeit verwendet. Sie besitzen ein 6x6-Bitmuster und decken somit die Zahlen von 0 bis 586 ab.

2.2.2 Markererkennung

Zur Erkennung und Dekodierung der Marker wird die vom APRIL Robotics Laboratory, University of Michigan entwickelte Software "AprilTag3" [19] verwendet.

Die Schritte der Erkennungssoftware sind dabei folgende [10]:

- Eingangsbild mittels adaptiver Schwellenwertbildung binarisieren.
- Die verbundenen schwarzen und weißen Regionen in zusammenhängende Komponenten segmentieren.
- Schließlich die Rechtecke (Quads) in diesen Regionen fitten und schlechte Quad-Fits oder nicht dekodierbare Tags verwerfen.

Der grüne gestrichelte Rahmen in Abbildung 2.2 bildet dabei den äußeren Quad, dessen Eckpunkte für die Berechnung der 3D Position und Orientierung verwendet werden. AprilTag3 liefert die Koordinaten der vier Eckpunkte und des Zentrums. Nützliche optionale Funktionen sind die Berechnung der Pose des Tags und ein "Edge Refinement" von Wang et al. [10], auf das gleich noch genauer eingegangen wird. Der Ursprung der Pose bezüglich des rechtshändigen Koordinatensystems des Markers befindet sich im Zentrum und zeigt mit der z-Achse in die Bildebene.

Die 2D Punkte der Ecken werden ausgehend von der unteren linken Ecke c_{ul} des grünen Rahmens in Abbildung 2.2 im Gegenuhrzeigersinn folgendermaßen definiert:

$$C = \begin{bmatrix} c_{ul} \\ c_{ur} \\ c_{or} \\ c_{ol} \end{bmatrix} \tag{2.9}$$

Wie zuvor erwähnt, wird in der Arbeit von Wang et al. [10] eine optionale Kantenverfeinerung (Edge Refinement) vorgestellt, die die Posenschätzung durch noch präziser ermittelte Eckpunkte weiter verbessert. Für Anwendungen in denen die Pose einzelner isolierter AprilTags bestimmt werden sollen, ist die von AprilTag3 zur Verfügung gestellte Posenschätzung ideal. Jedoch wird in dieser Arbeit ein Verbund von AprilTags (Pattern) verwendet, um die Basis, also den lateralen Abstand zwischen den künstlichen Merkmalen, zu vergrößern. Hierfür bedarf es der Schätzung einer gemeinsamen Pose aller detektierten Punkte mittels des in Unterabschnitt 2.1.2 vorgestellten PnP-Algorithmus. Das hier verwendete Pattern wird in Unterabschnitt 4.2.4 vorgestellt. Unabhängig davon ist das Edge Refinement von AprilTag3 eine enorme Verbesserung zu vergleichbaren Methoden der Subpixeldetektierung von Fiducials (z.B. des Schachbrettmusters von OpenCV). Da die erweiterte Verfeinerung dieser Arbeit auf die Methode von Wang et al. aufbaut, wird diese kurz vorgestellt.

2.2.3 Edge Refinement

Bei dem Edge Refinement wird entlang der bereits gefundenen Kanten an gleichmäßig verteilten Stellen der Gradient entlang der Normalen berechnet. Da die Marker einen dunklen inneren und einen hellen äußeren Rahmen haben, kann der Verlauf der Kante anhand der Positionen mit größtem Gradienten optimiert werden. [10]

Der grobe Ablauf ist folgender:

- Normale zur bereits gefundenen Kante berechnen.
- Punkte nahe der Kante entlang der Normalen mit großem Gradienten ermitteln.
- Mittels Least-Squares-Fit eine neue Linie, durch die zuvor ermittelten Punkte gehend, bestimmen.
- Aus den Schnittpunkten benachbarter Linien die neuen Eckpunkte zurückliefern.

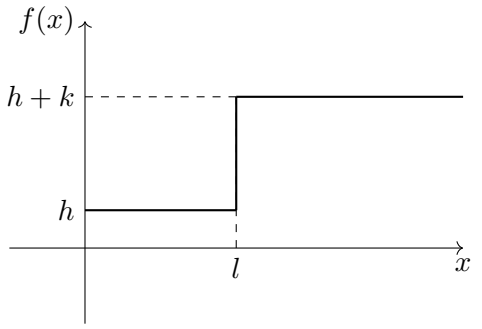


Abbildung 2.3: Darstellung der Stufenfunktion 2.10.

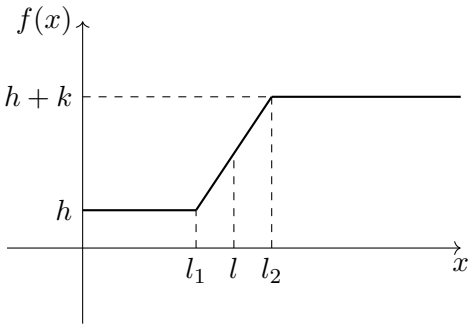


Abbildung 2.4: Darstellung der Rampenfunktion 2.11.

2.2.4 Erweiterte Kantenerkennung

In Hagara et al. [20] werden vier Methoden für Kantenerkennung mit Subpixelgenauigkeit verglichen, mit dem Ergebnis, dass die Methode AEF (Approximation with Erf Function) von M. Hagara und P. Kulla [21] in scharfen sowie unscharfen Bildern die höchste Präzision erzielt, zugleich jedoch auch am langsamsten ist. Da die Laufzeit für die Anwendung in dieser Arbeit jedoch vernachlässigbar ist, soll das Edge Refinement von Wang et al. [10] durch eine nicht lineare Least Square Optimierung (siehe Abschnitt 2.4) an eine Kantenfunktion gefittet werden. Im Folgenden wird das Kantenmodell und im nächsten Unterunterabschnitt das Optimierungsproblem vorgestellt.

Kantenmodell

Das Kantenmodell soll die Kante im Bild mathematisch bestmöglich beschreiben, sodass die Summe der Fehler (Residuen[2]) zwischen den von der Kantenfunktion berechneten Werten und den tatsächlichen Werten im Bild ein stabiles Minimum erreicht. Im einfachsten Fall wird dies mit der in Abbildung 2.3 dargestellte Stufenfunktion realisiert, die folgendermaßen definiert ist:

$$f(x) = \begin{cases} h, & x < l \\ h + k, & x \geq l \end{cases} \tag{2.10}$$

Mit der Kantenposition l, dem Parameter h, der die niedrigste Intensität im Evaluierungsbereich der Kante angibt, und dem Kontrastwert k. Bei einem 8-Bit Graustufenbild beispielsweise steigt die Intensität von Schwarz über alle Graustufen bis hin zu Weiß, was den Integerwerten von 0 bis 255 entspricht. Der Kontrast der Kante berechnet sich dann aus der Differenz der höchsten Intensität und h. Die Anwendung dieses Modells ist jedoch selten sinnvoll, da selbst

[2]Ein Residuum ist die Differenz zwischen einem beobachteten Wert (Messwert) und dem von einem Modell gelieferten angepassten Wert der Modellfunktion.

computergenerierte Bilder ohne Unschärfe ab einer geringen Abweichung von der Horizontalen oder Vertikalen einen mehrstufigen Kontrastübergang entwickeln.

Ein flexibleres Kantenmodell zeigt Abbildung 2.4 und wird durch drei Fälle beschrieben:

$$f(x) = \begin{cases} h, & x < l_1 \\ h + \frac{k(x-l_1)}{l_2-l_1}, & l_1 \leq x < l_2 \\ h + k, & l_2 \leq x \end{cases} \tag{2.11}$$

Die Kante springt jetzt nicht mehr vom niedrigsten zum höchsten Intensitätswert, sondern hat einen stetigen Übergang der Funktionswerte zentriert um l. Jedoch ist die Anwendung dieses Modells ebenfalls lediglich auf idealisierte Kanten sinnvoll. Aus natürlichen Bildern gewonnene Kanten sind in der Regel keineswegs ideale Stufen-/Rampenkanten. Stattdessen sind sie normalerweise von einem oder mehreren der folgenden Effekte betroffen:

- Fokale Unschärfe, verursacht durch eine endliche Tiefenschärfe und Punktspreizfunktion.
- Penumbrale Unschärfe, verursacht durch Schatten einer ausgedehnten Lichtquelle.
- Schattenwurf eines glatten Objekts.

In einer Reihe von Arbeiten [21–23] wird eine S-Funktion, dargestellt in Abbildung 2.5, als einfachste Erweiterung des idealen Stufenkantenmodells zur Modellierung der Effekte der Kantenunschärfe in praktischen Anwendungen verwendet. So wird ein eindimensionales Bild, das genau eine bei l platzierte Kante hat, modelliert als [23]:

$$f(x) = \frac{k}{2}\left[\operatorname{erf}\left(\frac{x-l}{\sqrt{2}\sigma}\right) + 1\right] + h \tag{2.12}$$

mit dem Kantenunschärfefaktor σ und der Fehlerfunktion [24]:

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z e^{-t^2} dt \tag{2.13}$$

In dieser Arbeit wird ebenfalls eine S-Funktion zur Modellierung verwendet, jedoch wird die Gleichung 2.12 und die Fehlerfunktion 2.13 noch etwas abgeändert:

Die Position l der Kante wird auf null gesetzt, damit die Funktion ihre höchste Steigung genau bei $x = 0$, also genau auf der Kante, besitzt und die Fehlerfunktion 2.13 wird durch den Hyperbeltangens (Tangens hyperbolicus) ersetzt, da sich dieser leicht implementieren und schnell berechnen lässt [25].
Daraus resultiert die neue Modellfunktion 2.14.

$$f(x) = \frac{k}{2}\left[\tanh\left(\frac{x}{\sqrt{2}\sigma}\right) + 1\right] + h \tag{2.14}$$

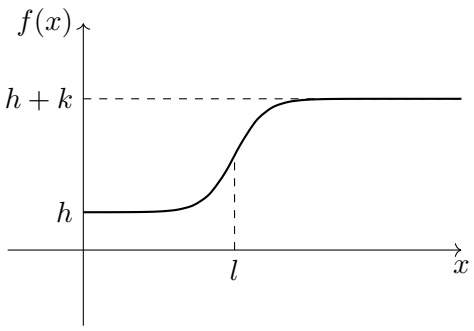


Abbildung 2.5: Darstellung der S-Funktion 2.14.

Mit den Kantenmodellen lassen sich also Position, Kontrast und, abgesehen von der Stufenfunktion, Unschärfe von Kanten in eindimensionalen Bildern beziehungsweise Arrays bestimmen. Es ist zwar möglich ein zweidimensionales Bild in Reihen und Spalten aufzuteilen und so den Verlauf der Kante im Bild zu approximieren, jedoch ist bekannt, dass die gesuchte Kante eines AprilTags in einem unverzerrten Bild geradlinig verläuft. Somit sind die Positionen der Kanten in den einzelnen Reihen und Spalten nicht unabhängig. Um die Kantenmodellparameter zu finden - die die Kante entlang der Linie, die bestmöglich durch alle zusammenhängenden Kanten der eindimensionalen Bilder verläuft, am besten darstellt - wird die nichtlineare Optimierung, beschrieben im Unterabschnitt 2.4.2, angewendet. In Abbildung 2.7 wird die optimale Gerade entlang der Kante durch die rot eingezeichnete Linie dargestellt. Die gesuchten Parameter θ sind die der S-Funktion 2.14 und die der durch die Kante verlaufenden Geraden:

$$\theta = \{\vec{n}_0, d, h, k, \sigma\} \tag{2.15}$$

Der Normalenvektor $\vec{n}_0$ und der Abstand d beschreiben zusammen mit einem beliebigen Ortsvektor die Hessesche Normalform, die im nächsten Abschnitt 2.3 vorgestellt wird. Die Residuen sind die Differenzen der Intensität zwischen eines jeden Pixels im Bild und der geschätzten Intensität des Kantenmodells bezüglich des Pixels Abstand zur Kante. Dies ist nochmal in Abbildung 2.6 veranschaulicht.

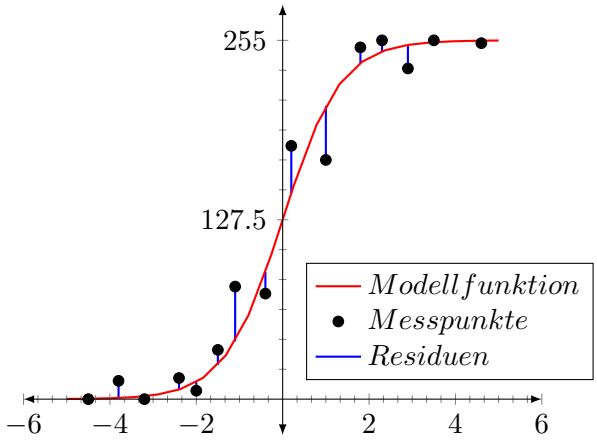


Abbildung 2.6: Messpunkte und deren Abstand von einer nach der Methode der kleinsten Quadrate bestimmten Funktion.

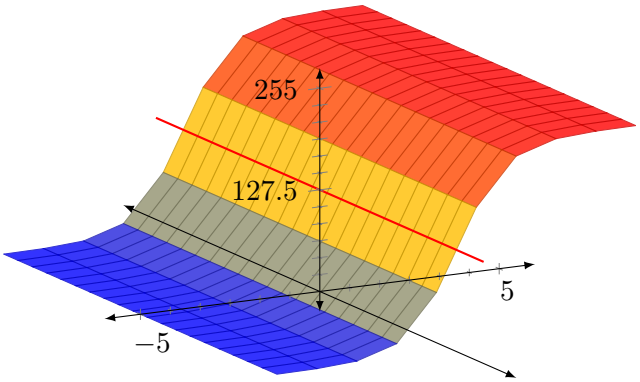


Abbildung 2.7: 3D Illustration der Modellfunktion.

2.3 Hessesche Normalform

Die Hessesche Normalform, benannt nach dem deutschen Mathematiker Otto Hesse, ist eine Gleichung aus der analytischen Geometrie, die eine Gerade in $\mathbb{R}^2$, eine Ebene in $\mathbb{R}^3$ oder eine Hyperebene in $\mathbb{R}^n$ beschreibt.

$$\vec{x} \cdot \vec{n}_0 - d = 0 \tag{2.16}$$

In ihrer einfachsten Form, als Gerade in der euklidischen Ebene, setzt sie sich zusammen aus einem normierten Normalenvektor $\vec{n}_0$ der Gerade, sowie ihren Abstand $d \geq 0$ vom Koordinatenursprung (siehe Abbildung 2.8). Der Abstand d entspricht gerade der Länge der Orthogonalprojektion des Vektors $\vec{x}$ auf die Ursprungsgerade mit Richtungsvektor $\vec{n}_0$.

Sie wird häufig zur Abstandsberechnung $d(P, g)$ eines beliebigen Punkts P zur Geraden g verwendet (siehe Abbildung 2.9).

$$d(P, g) = \vec{p} \cdot \vec{n}_0 - d \tag{2.17}$$

Mit dem Ortsvektor $\vec{p}$ kann die Hessesche Normalform auch in Koordinatenform geschrieben werden.

$$d(P, g) = p_x n_x + p_y n_y - d \tag{2.18}$$

Ist $d(P, g) > 0$ befindet sich der Punkt P auf der Seite der Gerade g in die die Normale $\vec{n}_0$ zeigt, ansonsten auf der anderen Seite.

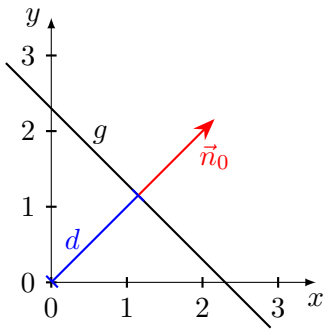


Abbildung 2.8: Darstellung der Hessesche Normalform mit Gerade g, Normalenvektor $\vec{n}_0$ und Koordinatenursprungsabstand d.

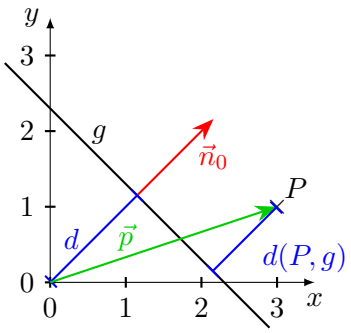


Abbildung 2.9: Der kürzeste Abstand zwischen einem Punkt P und einer Geraden g ist definiert als die Länge des Linienabschnitts $d(P,g)$, der senkrecht zur Geraden liegt und durch den Punkt geht.

Ausgehend von zwei verschiedenen Punkten (x_1, y_1) und (x_2, y_2) wird die Hessesche Normalform wie folgt berechnet

$$\begin{aligned} \frac{y-y_1}{x-x_1} &= \frac{y_2-y_1}{x_2-x_1} = \frac{\Delta y}{\Delta x} \\ (y-y_1)\Delta x &= (x-x_1)\Delta y \\ x\Delta y - y\Delta x - x_1\Delta y + y_1\Delta x &= 0 \\ \frac{x\Delta y - y\Delta x - (x_1\Delta y - y_1\Delta x)}{\sqrt{\Delta x^2 + \Delta y^2}} &= 0 \\ x n_x + y n_y - d &= 0 \end{aligned} \tag{2.19}$$

und der Schnittpunkt (x_i, y_i) aus zwei verschiedenen Hessesche Normalformen ergibt sich dann aus:

$$x_i = \frac{n_{y,1}d_2 - n_{y,2}d_1}{n_{x,1}n_{y,2} - n_{x,2}n_{y,1}} \qquad \text{und} \qquad y_i = \frac{n_{x,2}d_1 - n_{x,1}d_2}{n_{x,1}n_{y,2} - n_{x,2}n_{y,1}} \tag{2.20}$$

Ist die Determinante (der Nenner) von x_i und y_i ungleich null, so sind die Geraden linear unabhängig und es existiert ein Schnittpunkt.

2.4 Optimierungsproblem

Im einfachsten Fall besteht ein Optimierungsproblem darin, eine reelle Funktion zu maximieren oder zu minimieren, indem systematisch Eingabewerte aus einer erlaubten Menge gewählt und der Wert der Funktion berechnet wird oder durch Auffinden der Nullstellen der ersten Ableitung einer differenzierbaren Funktion.

In dieser Arbeit ist vor allem die mathematische Optimierungsmethode der Ausgleichungsrechnung oder Curve-Fitting von Interesse, mit deren Hilfe für eine Reihe von Messdaten die unbekannten Parameter einer vorgegebenen Funktion bestimmt oder approximiert werden.

2.4.1 Methode der kleinsten Quadrate

Im Allgemeinen wird die Berechnung mit der Methode der kleinsten Quadrate oder Least-Squares durchgeführt. Sie ist ein Standardansatz in der Regressionsanalyse, um die Lösung überbestimmter Systeme[3] durch Minimierung der Summe der Quadrate der Residuen aus den Ergebnissen jeder einzelnen Gleichung zu approximieren.

Ein wichtiges Anwendungsgebiet für Least-Squares ist Data-Fitting. Abbildung 2.6 zeigt eine Modellfunktion $y = f(x)$, die an Messpunkte X angepasst ist. Aufgrund von Störungen werden die Wertepaare in X die Gleichung $y_i = f(x_i)$ im Allgemeinen nicht exakt erfüllen, sondern etwas abweichen, d.h.

$$\begin{gathered} y_i = f(x_i) + \epsilon_i \qquad \forall i \in \{1, 2, \ldots, N\} \\ (\epsilon_i : \text{Approximationsfehler bzw. Residuen}) \end{gathered} \tag{2.21}$$

Ziel ist es die Parameter θ der Regressionsfunktion $y = f(x, \theta)$ nun so anzupassen, dass die Summe der Quadrate der Residuen ϵ_i möglichst klein wird. Die Kostenfunktion F muss demnach minimal werden:

$$F \coloneqq \frac{1}{2} \sum_{i=1}^{N} \epsilon_i^2 = \frac{1}{2} \sum_{i=1}^{N} (y_i - f(x_i))^2 \rightarrow min \tag{2.22}$$

Least-Squares-Probleme fallen in zwei Kategorien: lineare und nichtlineare, je nachdem, ob die Residuen in allen Unbekannten linear sind oder nicht. Die Lösung des linearen Least-Squares-Problems hat eine geschlossene Form. Das nichtlineare Problem wird in der Regel durch iterative Verfeinerung gelöst, was mit dem Aufkommen leistungsfähiger Rechner immer mehr an Bedeutung gewinnt.

[3] Überbestimmte Systeme sind Gleichungssysteme, in denen es mehr Gleichungen als Unbekannte gibt.

2.4.2 Nichtlineare Optimierung

Das Nullsetzen der Gradienten ermöglicht das Bestimmen der Maxima oder Minima von der Kostenfunktion F 2.22. Die Anzahl der Gradientengleichungen ist dabei von der Anzahl der Modellparameter M abhängig.

$$\frac{\partial F}{\partial \theta_j} = \sum_{i=1}^{N} \epsilon_i \frac{\partial \epsilon_i}{\partial \theta_j} = 0 \qquad \forall j \in \{1, 2, \ldots, M\} \tag{2.23}$$

Bei einem nichtlinearen System ist das Lösen der Gradientengleichungen $\frac{\partial F}{\partial \theta_j}$ in geschlossener Form nicht möglich. Deshalb wird für jeden der gesuchten Parameter θ ein Startwert definiert und durch sukzessive Approximation iterativ bestimmt. In jedem Iterationsschritt k werden die Startparameter neu definiert

$$\theta_j \approx \theta_j^{k+1} = \theta_j^k + \Delta\theta_j \tag{2.24}$$

und das Modell durch eine Taylorentwicklung von θ^k bis zur 1. Ordnung linearisiert

$$f(x_i, \theta) \approx f(x_i, \theta^k) + \sum_{j=1} \frac{\partial f(x_i, \theta^k)}{\partial \theta_j}(\theta_j - \theta_j^k) = f(x_j, \theta^k) + \sum_{j=1} J_{i,j}\Delta\theta_j \tag{2.25}$$

Die Jakobimatrix J enthält unabhängige Variablen sowie die Modellparameter und ändert sich nach jeder Iteration. Somit ist das linearisierte Modell mit $\frac{\partial \epsilon_i}{\partial \theta_j} = -J_{i,j}$ gegeben durch

$$-\sum_{i=1}^{N} J_{i,j}(y_i - \sum_{l=1}^{M} J_{i,l}\Delta\theta_l) = 0 \qquad \forall j \in \{1, 2, \ldots, M\}. \tag{2.26}$$

Umgestellt in die Normalengleichung

$$\sum_{i=1}^{N}\sum_{l=1}^{M} J_{i,j}J_{i,l}\Delta\theta_l = \sum_{i=1}^{N} J_{i,j}y_i \qquad \forall j \in \{1, 2, \ldots, M\} \tag{2.27}$$

kann schließlich die Matrixgleichung

$$(J^T J)\Delta\theta = J^T y \tag{2.28}$$

formuliert werden.

Ceres - eine Open-Source-Bibliothek in C++ zur Modellierung und Lösung großer, komplizierter Optimierungsprobleme - bietet eine Reihe von Möglichkeiten zur Lösung von Gleichung 2.28 [27]. Für kleine Probleme mit wenigen hundert Modellparametern θ, wenigen tausend Residuen ϵ und relativ dichter Jakobimatrix J ist "Dense QR" die Methode der Wahl [28]. Sei $J = QR$ die QR-Zerlegung von J, mit der orthonormalen Matrix Q und der oberen Dreiecksmatrix R, dann ist die Lösung der Gleichung 2.28 gegeben durch

$$\Delta\theta^* = R^{-1}Q^T y \tag{2.29}$$

Ceres verwendet die QR-Zerlegungroutinen von Eigen, einer Bibliothek in C++ für lineare Algebra. [27]

2.5 Bildverarbeitung

Die Bildverarbeitung liefert die mathematischen und algorithmischen Grundlagen, welche bei der Implementierung von Grafiksoftware zur Bildbearbeitung, in Computer Vision und vielen anderen Bereichen der Informatik und Ingenieurwissenschaften Verwendung finden. Hierbei werden aus Bilddaten entweder wiederum Bilddaten erzeugt, wie beispielsweise bei digitalen Bildtransformationen und Bildverbesserungen oder es werden Informationen extrahiert, wie beispielsweise bei der Erstellung eines Histogramms oder der Objekterkennung/-verfolgung.

Die adaptive Schwellenwertbildung, die von AprilTag3 auf das Eingangsbild angewendet wird (siehe Unterabschnitt 2.2.2), ist ebenfalls eine Bildverarbeitung und liefert ein Ausgangsbild, das in Abhängigkeit vom Kontrastlevel segmentiert ist und dadurch die Dekodierung der Quads des AprilTags verbessert.

Die in Unterabschnitt 2.2.4 vorgestellte erweiterte Kantenerkennung profitiert ebenfalls von einer Vorverarbeitung des Eingangsbildes. Die zwei in dieser Arbeit verwendeten Methoden werden daher im Folgenden kurz vorgestellt.

2.5.1 GaussianBlur

Der GaussianBlur-Filter ist ein Tiefpassfilter, der hochfrequente Intensitätsänderungen im Bild glättet. Die Glättung erfolgt durch eine Faltungsoperation, indem ein Kernel (2D Matrix) über das Bild gezogen wird. Auf Grundlage der konstanten Matrixeinträge des Kernels und der Intensität der darunterliegenden Pixel im Originalbild, wird jeder Pixelwert neu berechnet. Die Namensgebung beruht auf der zugrundeliegenden Funktion, der zweidimensionalen Gauß-Verteilung 2.30, die dem Werteverlauf der Einträge des Kernels die typische Glockenformverteilung zuweist.

$$G_r(p) = \frac{1}{2\pi\sigma^2} e^{-\frac{(px-cx)^2+(py-cy)^2}{2\sigma^2}} \tag{2.30}$$

Abbildung 2.10 illustriert dies anhand eines 5×5-Kernels. Die neue Intensität I des zentralen Pixels c ist dann die Summe $I_c = \sum_{p \in P} G_r(p)$.

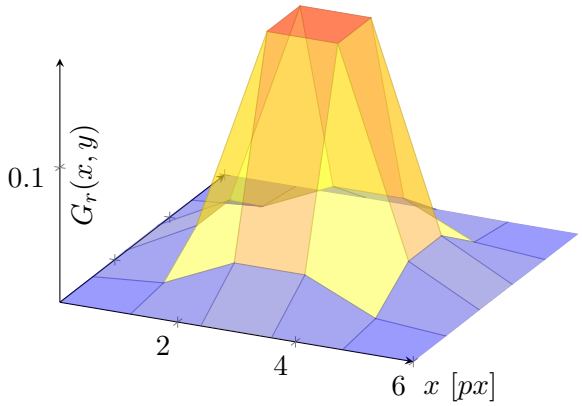


Abbildung 2.10: 5×5-Faltungskernel eines GaussianBlur-Filters.

2.5.2 Bilateralfilter

Im Vergleich zum Gauß-Filter bewahrt der bilaterale Filter die Kanteninformationen des Bildes besser. Das Prinzip besteht darin, dass eine gaußsche Funktion, die sich auf den räumlichen Abstand bezieht, mit einer gaußschen Funktion multipliziert wird, die sich auf den Kontrast bezieht. Die Funktion 2.30 wird also lediglich mit der Funktion 2.31 multipliziert, die sich auf die Differenz der Intensitäten I bezieht.

$$G_k(p) = \frac{1}{2\pi\sigma^2} e^{-\frac{(I_p - I_c)^2}{2\sigma^2}} \tag{2.31}$$

Die Abbildungen 2.11 zeigen vereinfacht die Erstellung des endgültigen Faltungskernels bezüglich eines Pixels c, der sich dicht an einer perfekt stufenförmigen Kante befindet. Der Pixel c wird also primär von benachbarten Pixeln mit ähnlicher Intensität beeinflusst, jedoch kaum von dem Beispielpunkt p, wodurch der Kontrast der Kante erhalten bleibt. Die endgültige Berechnung der neuen Intensität des Pixels c ist dann die Summe:

$$I_c' = \frac{1}{W_c} \sum_{p \in P} G_r(p) * G_k(p) * I_p \tag{2.32}$$

mit dem Normalisierungsterm:

$$W_c = \sum_{p \in P} G_r(p) * G_k(p) \tag{2.33}$$

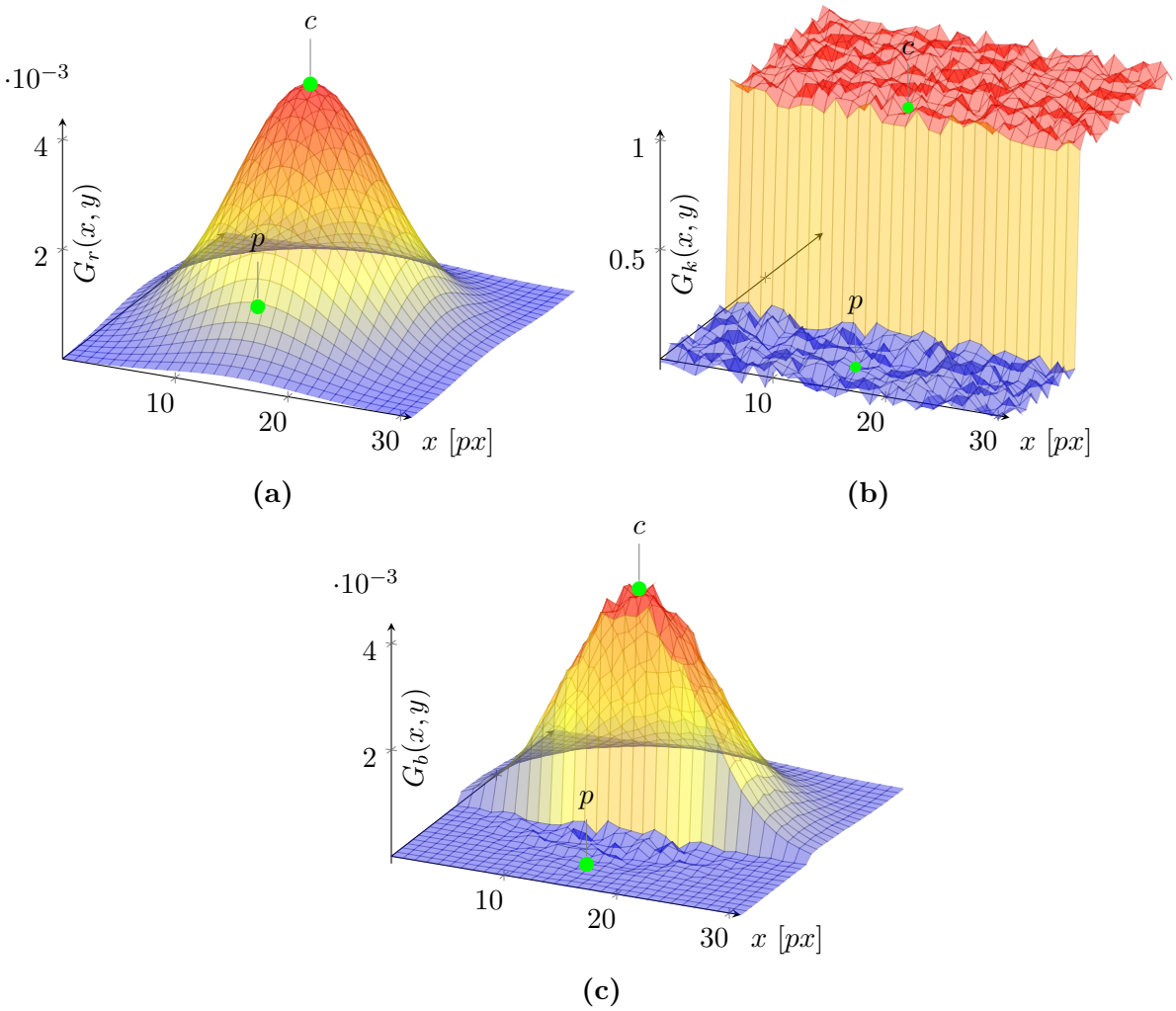


Abbildung 2.11: **(a)** 30×30-Kernel einer Gauß-Verteilung bezüglich der Distanz zum zentralen Pixel und **(b)** die Kontrastdifferenz bezüglich des zentralen Pixels. **(c)** zeigt den endgültigen Faltungskernel des Bilateralfilters.

2.6 Trajektorienplanung

Die Trajektorienplanung ist eine grundlegende Teilaufgabe der Automatisierung von Vehikeln. Mithilfe von karten- oder sensorbasierten Daten wird eine Trajektorie generiert, die als Sollwert für die Steuerung des Vehikels dient. Dabei sind Aspekte wie Realisierbarkeit und Kollisionsfreiheit der Trajektorie zu beachten.

Ein elementares Trajektorienplanungsproblem besteht darin, einen kontinuierlichen Pfad zu berechnen, der eine Startpose S und eine Zielpose Z miteinander verbindet. In [29] wird von J. A. Reeds und L. A. Shepp eine Methode vorgestellt, die genau dieses Problem löst. Der Reeds-Shepp-Pfad ist definiert als der kürzeste Pfad eines Reeds-Shepp-Vehikles, das sich vorwärts und rückwärts mit einem beschränkten Wendekreis fortbewegen kann. Die Methode berechnet planare Bahnen, die aus Liniensegmenten bestehen, die mit tangentialen Kreisbögen mit minimalem Radius verbunden sind. Insgesamt umfasst die Pfadmenge 48 unterschiedliche Pfade, die maximal aus 5 Segmenten bestehen.

Die Krümmung dieses Bahntyps ist jedoch diskontinuierlich. Diskontinuitäten treten an den Übergängen zwischen Segmenten und Bögen sowie zwischen Bögen mit entgegengesetzter Drehrichtung auf. Da die Krümmung mit der Ausrichtung der Vorderräder zusammenhängt, muss ein reales Auto, wenn es eine solche Bahn exakt abfahren will, an jeder Krümmungsunterbrechung anhalten, um seine Vorderräder neu auszurichten. Krümmungskontinuität ist daher eine wünschenswerte Eigenschaft.

Eine Möglichkeit Krümmungskontinuität zu erreichen ist die Annäherung der Reeds-Shepp-Segmente durch Klothoiden. Ihr Krümmungsverlauf nimmt linear zu, wodurch sich anstatt eines abrupten Rucks ein allmählicher Beschleunigungsübergang von Geradeausfahrt in die Kreisfahrt ergibt. Klothoiden werden beispielsweise als Übergangsbogen bei Kurven im Straßenbau sowie im Eisenbahnbau eingesetzt und lassen sich folgendermaßen recht effizient berechnen:

$$T = \frac{L}{2R} \tag{2.34}$$

$$x = \int_0^L \cos(T) dL \tag{2.35}$$

$$y = \int_0^L \sin(T) dL \tag{2.36}$$

L ist hierbei die Länge des Klothoidenabschnitts, R der Krümmungsradius im Endpunkt und T der Schnittwinkel der Normalen im Anfangs- und Endpunkt der Klothoide. Die Koordinaten (x, y) mit Ursprung am Anfangspunkt der Klothoide, beziehen sich auf den Endpunkt der Klothoide, wobei die Tangente im Anfangspunkt mit "unendlichem" Krümmungsradius die x-Achse bildet. In Abbildung 2.12 ist der Übergangsbogen der Klothoide von einem geraden Segment auf einen Kreisbogen rot gestrichelt eingezeichnet.

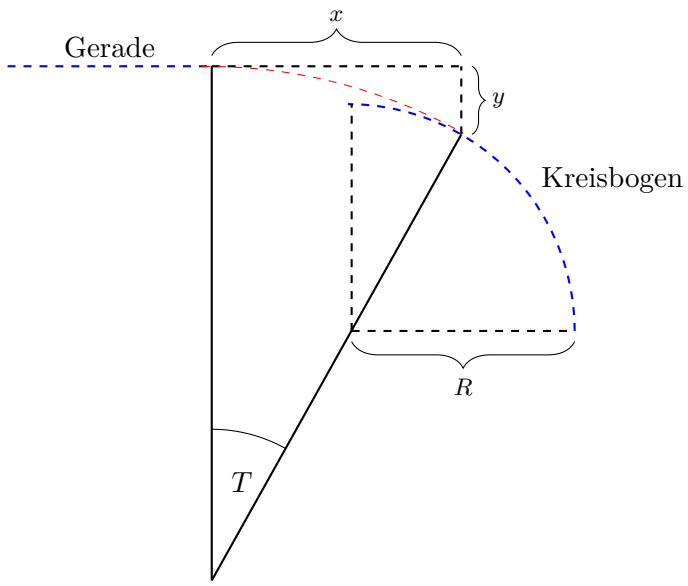


Abbildung 2.12: Darstellung einer Klothoide (rot) mit Länge L zwischen einer Gerade und einem Kreisbogen (blau).

2.7 Hand-Eye-Kalibrierung

Für Anwendungen bei denen eine Kamera in ein Robotersystem integriert ist, muss sie in Bezug auf das Referenzsystem des Roboters kalibriert werden, damit beispielsweise ein Roboterendeffektor in der Lage ist einen Gegenstand aufzunehmen, der von der Kamera erfasst wird. Das Verfahren zur Bestimmung dieser Transformation wird "Hand-Eye-Kalibrierung" genannt. Es existieren zwei Methoden der Hand-Eye-Kalibrierung:

- **Eye-to-Hand:** Wird verwendet, wenn die Kamera stationär neben einem Roboter montiert ist.
- **Eye-in-Hand:** Wird verwendet, wenn die Kamera am Roboter montiert ist.

Der Ablauf der Hand-Eye-Kalibrierung ist folgender:

1. Roboter an eine einmalige Position bewegen.
2. Pose des Roboters aufzeichnen.
3. Pose der Kamera bestimmen.
4. Schritte 1-3 mindestens drei Mal wiederholen.
5. Hand-Eye-Transformation berechnen.

Durch das Bewegen des Roboters an verschiedene Positionen ergeben sich viele unterschiedliche Posen der Kamera und des Roboters bezüglich des Weltkoordinatensystems, die im folgenden mit

A_i^w für den Roboter und B_i^w für die Kamera bezeichnet werden. Der Zusammenhang zwischen den Posen ergibt sich zu

$$A_i^w = A_0^w A_i^0 \tag{2.37}$$
$$B_i^w = B_0^w B_i^0 \tag{2.38}$$

und die Transformation zwischen der Kamera und dem Roboter

$$A_0^w = B_0^w X \tag{2.39}$$
$$A_i^w = B_i^w X \tag{2.40}$$

woraus sich die Gleichung

$$A_i^0 X = X B_i^0 \tag{2.41}$$

bilden lässt. Ausgeschrieben

$$\begin{bmatrix} R_{A_i} & t_{A_i} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_X & t_X \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_X & t_X \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_{B_i} & t_{B_i} \\ 0 & 1 \end{bmatrix} \tag{2.42}$$

werden daraus die zwei zu lösenden Gleichungen

$$R_{A_i} R_X = R_X R_{B_i} \tag{2.43}$$
$$(R_{A_i} - I) t_X = R_X t_{B_i} - t_{A_i} \tag{2.44}$$

formuliert, aus denen iterativ mit der Methode von Tsai et al. [37] erst R_X und schließlich, durch Einsetzen von R_X in 2.44, auch t_X bestimmt wird.

Abbildung 2.13 illustriert den Ablauf anhand von drei unterschiedlichen Posen im Vorfeld des AprilTag-Patterns. Das Automated Guided Vehicle (AGV) stellt hierbei den Roboter dar mit einer Kamera an der Vorderseite. Somit handelt es sich bei dieser Darstellung um eine Eye-in-Hand-Kalibrierung. Die Indizes i und j bezeichnen nacheinander eingenommene Posen, die sich alle auf die Ausgangspose unten rechts mit hochgestellter null beziehen. Da die Kamerapose photogrammetrisch anhand des AprilTag-Patterns bestimmt wird, muss jede Pose auf das Pattern ausgerichtet sein. Die Roboterpose wird durch ein externes Messsystem “iGPS” aufgezeichnet, das später im Unterabschnitt 4.2.5 vorgestellt wird. Das Ergebnis der Kalibrierung ist die gestrichelt dargestellte Transformation X zwischen der Kamera und dem iGPS-Sensor.

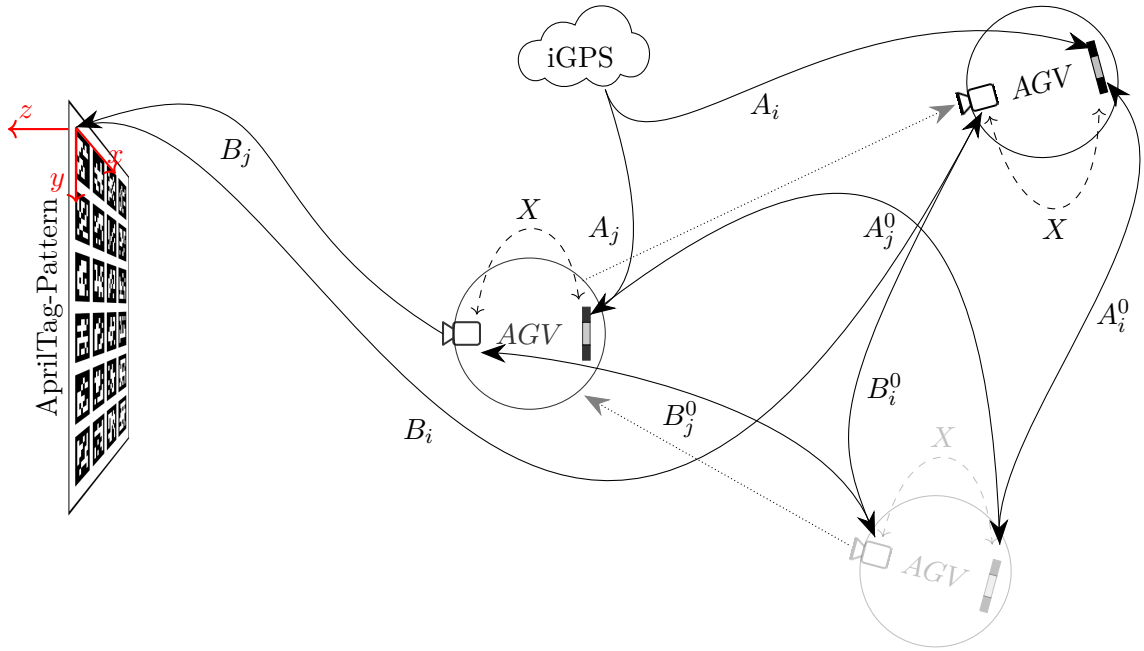


Abbildung 2.13: Veranschaulichung der Hand-Eye-Kalibrierung und Transformationsrelationen.

Kapitel 3

Implementierung

In diesem Kapitel werden alle für die Andockroutine notwendigen Module und Bibliotheken ausführlich beschrieben. Der grobe Ablauf des Andockvorgangs wird in Abbildung 3.1 dargestellt. In dem Programmablaufdiagramm ist gut zu erkennen, dass der Ablauf aus sich wiederholenden Routinen besteht, die in Anfahrt, Validierung und Akquirierung von Posestellungen gegliedert sind. Der Aufbau der wichtigsten Module ist in dem UML-Diagramm 3.2 zu sehen. Die Module müssen die Kamerapose ermitteln, einen Trajektorienplan erstellen und diesen in Segmenten kontrolliert ausführen. In den folgenden Abschnitten wird nochmals an angemessenen Stellen auf die Abbildungen verwiesen.

3.1 Softwarestruktur

Aufbauend auf bereits bestehende Strukturen werden die einzelnen Module innerhalb der Projekte im UML-Diagramm 3.2 weitgehend unabhängig aufgebaut. Zu diesem Zweck findet die Kommunikation innerhalb der Kontrollstation und der Roboter (Agenten) primär über eine Publisher-Subscriber-Architektur statt. Das Andocken soll in Zukunft nur eine von vielen Aktionen sein, die über die Infrastruktur abgewickelt werden.

Da die bestehende Softwarestruktur englische Bezeichnungen verwendet, wird dies auch für die neuen Module fortgeführt. Der Übersicht halber und um Verwechslungen vorzubeugen werden die englischen Bezeichnungen der Projekte, Module und Klassen im Folgenden auch im Text verwendet und durch entsprechende Formatierung zusätzlich hervorgehoben. So wird beispielsweise das Projekt “Kontrollstation” nachfolgend **ControlStation** und die Klasse für die Ablaufplanung `Scheduler` genannt.

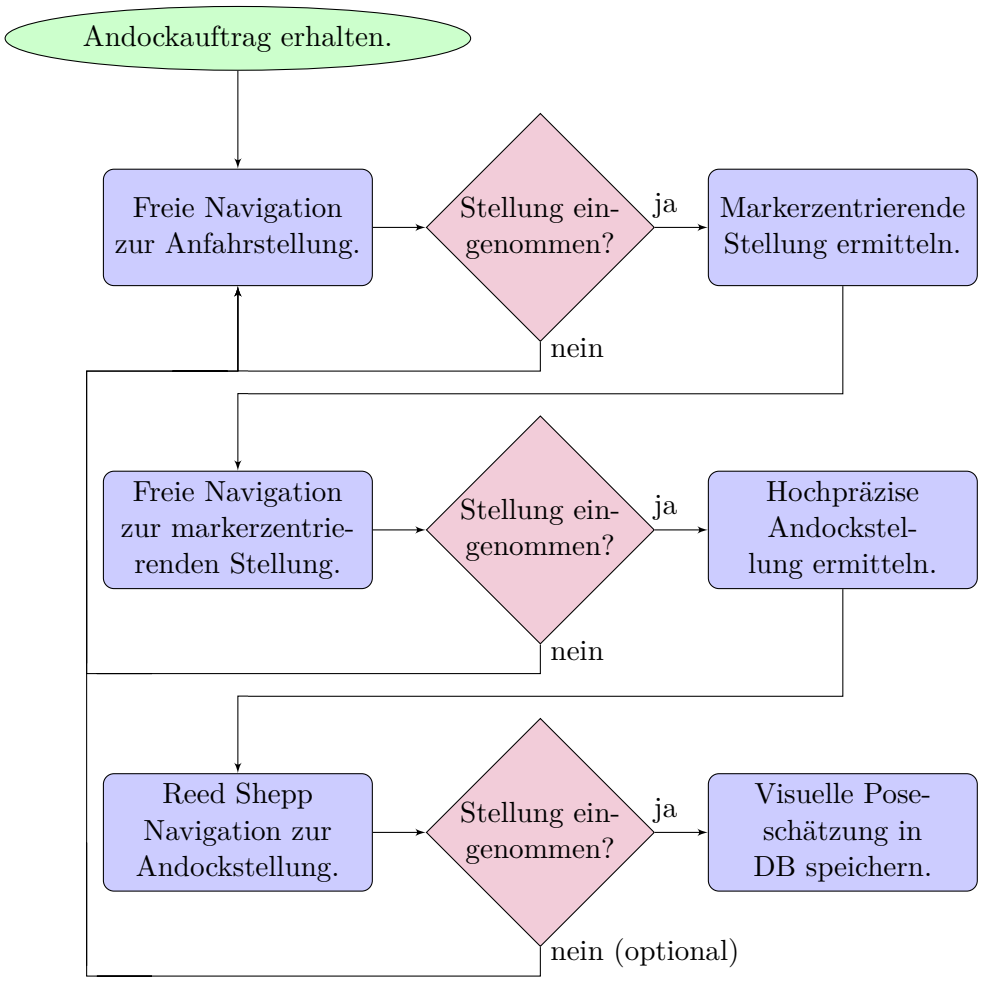


Abbildung 3.1: Programmablaufdiagramm der Andockroutine.

3.1.1 ControlStation

Von der Befehlsannahme vom Nutzer über die Benutzeroberfläche, der Bestimmung aller notwendigen Ausführungsparameter, bis hin zur Übermittlung eines Plans an die entsprechenden Agenten, wird alles in der **ControlStation** abgewickelt. Um ein Grundgerüst hierfür zu bieten wurde bereits ein Backend in Python implementiert, das über SocketIO mit einem hauptsächlich in JavaScript geschriebenen Frontend kommuniziert. Eine Übersicht der verwendeten Bibliotheken wird in Abbildung 3.3 dargestellt. Die Grundfunktionalität des Frontends stammt aus der Leaflet-Bibliothek, die speziell für Web-Mapping-Anwendungen entwickelt wurde. Das Backend besteht aus mehreren Modulen, die über PyPubSub, einer Publisher-Subscriber-Infrastruktur, kommunizieren.

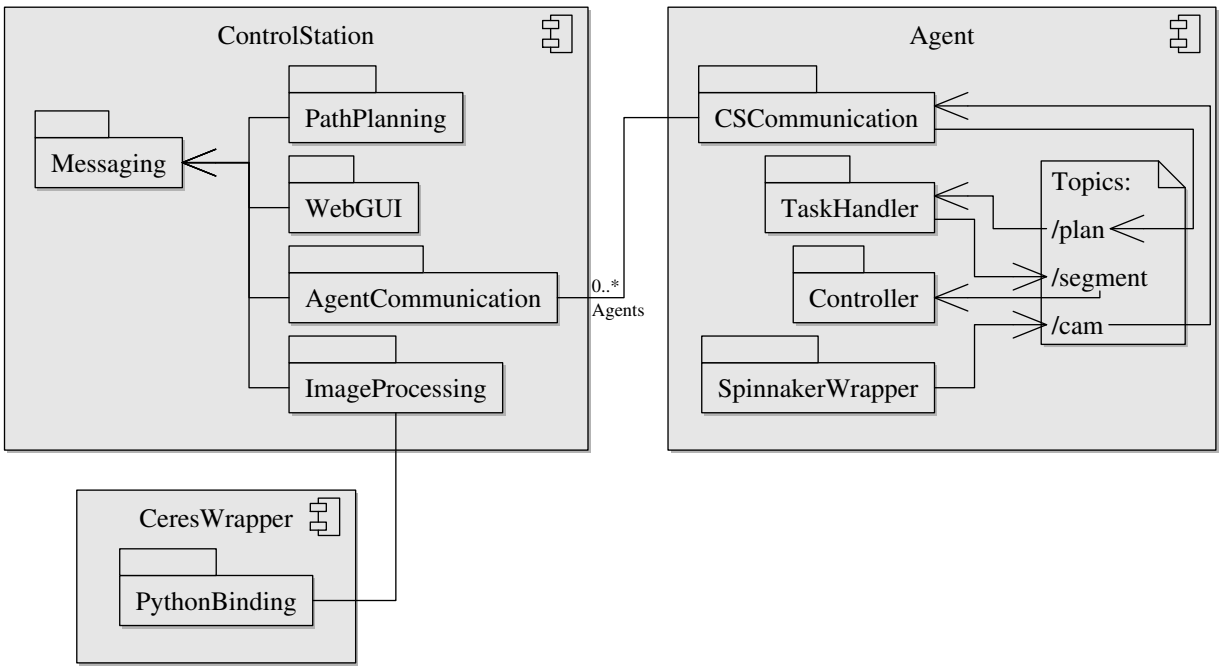


Abbildung 3.2: Übersicht der Projekte **ControlStation**, **Agent** und **CeresWrapper** sowie deren Module und Kommunikation.

Die für diese Arbeit notwendigen Module sind:

- **WebGUI:** Aufgebaut wie ein typisches Flask-Webframework. Wickelt die gesamte Kommunikation mit dem Frontend ab und kümmert sich um die Ablaufplanung (Scheduling).

- **AgentCommunication:** Standardisierte VDA 5050-Kommunikationsschnittstelle unter Verwendung des Machine-to-Machine Netzwerkprotokolls MQTT. Wickelt die gesamte Kommunikation mit allen Arten von fahrerlosen Transportfahrzeugen, die diese Schnittstelle unterstützen, ab. Weiterführend ist ebenfalls die Kommunikation mit Stationen über dieses Modul vorgesehen.

- **PathPlanning:** Implementiert die in Abschnitt 2.6 vorgestellte Trajektorienplanung.

- **ImageProcessing:** Framework für beliebige Bildverarbeitungsaufgaben. Verwaltet die Kameraparameter der Agenten und kalibriert diese bei Bedarf.

Das Modul **ImageProcessing** beinhaltet den Großteil der speziell für diese Arbeit implementierten Methoden, auf die genauer in Abschnitt 3.2 eingegangen wird. Die Verwaltung der Agentenaufgaben (Scheduling) befindet sich derzeit im Backend-Scheduling-Event des **WebGUI**-Moduls. Laufende Pläne werden hier mit dem Status der Agenten verglichen, aktualisiert und gegebenenfalls neu gestartet. Auch die Abarbeitung von Aktionen, wie beispielsweise das Andockmanöver dieser Arbeit, wird im `Scheduler` initialisiert.

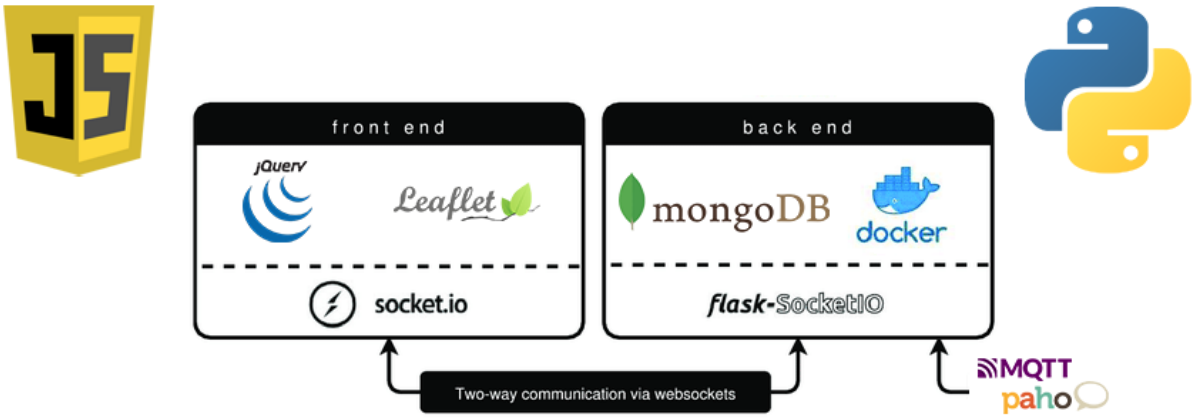


Abbildung 3.3: Grafische Darstellung der Bibliotheken und Programmiersprachen der **ControlStation**.

Der Ablauf eines Andockauftrags wird im Diagramm 3.1 beschrieben. Die Betriebssoftware der Agenten besitzt bereits eine "freie Navigation", mit der sie in der Lage sind, selbstständig zu einer vorgegebenen Pose - Anfahrstellung im Fall der ersten Aktion im Diagramm - zu navigieren. Die Anfahrstellung wird je nach Situation und Robotertyp frei gewählt. Beispielsweise benötigt der in dieser Arbeit verwendete Roboter eine gerade Strecke von ca. 2 m, um sein Fahrwerk vollständig auszurichten. Ab da wird so lange zwischen Anfahrstellung und markerzentrierender Stellung gewechselt, bis sich die AprilTags ausreichend gut im Zentrum des Kamerabildes befinden. Dies klappt jedoch für gewöhnlich beim ersten Versuch. Der Grund für die markerzentrierende Stellung ist die erhöhte Schätzgenauigkeit der Kamerapose bei einer zentrierten Ausrichtung, was zu einem späteren Zeitpunkt im Unterabschnitt 4.3.3 noch genauer erläutert wird. In der weiteren Folge beziehen die rechten Aktionen des Diagramms 3.1 Informationen zur Pose der Kamera vom **ImageProcessing** und die untere linke Aktion (Reed Shepp Navigation zur Andockstellung) nutzt das **PathPlanning**-Modul, um eine Trajektorie von der markerzentrierenden Stellung zur Andockstellung zu berechnen. Im Abschnitt 3.5 wird genauer auf die Implementierung der Trajektorienplanung eingegangen.

3.1.2 Agent

Die Betriebssoftware für Agenten ist bereits funktionsfähig und wird lediglich in den Modulen aus dem UML-Diagramm 3.2 angepasst, um externe Pläne zu empfangen und auszuführen. Das Projekt ist zum größten Teil in C++ geschrieben und verwendet das Robot Operating System (ROS), wodurch die einzelnen Module (ROS-Nodes), wie auch bei der **ControlStation**, weitgehend unabhängige Prozesse sind.

Die für diese Arbeit angepassten Module sind:

- **CSCommunication:** Standardisierte, clientseitige VDA 5050-Kommunikationsschnittstelle unter Verwendung des MQTT-Protokolls. Versendet periodisch Status, Position, Kamerabilder uvm. an die **ControlStation** und verwaltet eingehende Pläne und Aktionen.

- **TaskHandler:** Prüft eingehende Pläne auf Richtigkeit und versendet die einzelnen Segmente der Reihe nach, bei gleichzeitiger Überwachung der Segmentanforderungen (z.B. Endposition) und möglicherweise auftretenden Fehlern und Warnungen (z.B. durch Hindernisse).

- **Controller:** Führt die Segmente aus, prüft die Umgebung auf Hindernisse und ob sich der Agent auf der richtigen Trajektorie befindet. Der verwendete **Controller** trägt maßgeblich zur Güte der Pfadabfahrgenauigkeit bei und wird daher in Unterabschnitt 3.5.2 noch genauer behandelt und optimiert.

- **SpinnakerWrapper:** Framework für beliebige Kameramodelle der Firma FLIR, das deren Spinnaker SDK in eine anwendungsfreundliche ROS-Node verpackt und die Kameraaufnahmen als ROS-Topic zur Verfügung stellt.

Die Kommunikationswege zwischen den vier Modulen sind zusätzlich mit den Topic-Bezeichnungen in dem UML-Diagramm 3.2 dargestellt.

Der Einfachheit halber werden Agent-Informationen, Bildaufnahmen usw. ohne angefordert zu werden periodisch an die **ControlStation** versendet. Schlägt ein Plan fehl und verändert der Agent daraus resultierend seine Position für eine gewisse Zeit nicht, wird ein neuer Plan von der **ControlStation** generiert und an den Agenten übertragen.

3.1.3 CeresWrapper

Das dritte Projekt im UML-Diagramm 3.2 verwendet Pybind11, um das in C++ implementierte Ceres in ein Python-Interface zu verpacken. Pybind11 ermöglicht den Austausch beliebiger benutzerdefinierter Datenstrukturen per Wert, Referenz oder Pointer zwischen C++ und Python. Dadurch kann, ohne nennenswerten Performanzverlust, auf mächtige Bibliotheken, wie Ceres und Eigen, zugegriffen werden. Für diese Arbeit wird Ceres verwendet, um die nichtlineare Optimierung aus Unterabschnitt 2.4.2 zu lösen und Eigen liefert die dafür benötigten QR-Zerlegungsroutinen und eine breite Palette an Werkzeugen für Hyperebenenberechnungen.

3.2 Approximierung mittels Kostenminimierung

In diesem Abschnitt werden die Methoden und Algorithmen vorgestellt, um die Kantenmodellparameter aus Unterabschnitt 2.2.4 mittels der nichtlinearen Optimierung, beschrieben in Unterabschnitt 2.4.2, zu approximieren. In Abbildung 2.7 wird die optimale Gerade entlang einer Kante durch die rot eingezeichnete Linie dargestellt, dessen Parameter θ der S-Funktion 2.14 hier nochmal kurz aufgeführt werden:

$$\theta = \{\vec{n}_0, d, h, k, \sigma\} \tag{3.1}$$

Der Normalenvektor $\vec{n}_0$ und der Abstand d beschreiben zusammen mit einem beliebigen Ortsvektor die Hessesche Normalform aus Abschnitt 2.3. Wie in Unterabschnitt 2.4.2 angesprochen, wird Ceres und die "Dense QR"-Methode zur Lösung der Gleichung 2.29 verwendet. Hierfür muss bereits vor Lösungsbeginn die Anzahl der Residuen ϵ bekannt sein.

Der grobe Ablauf umfasst folgende Punkte:

- Isolieren des detektierten Markers als "Region of Interest" (ROI) vom Rest des Bildes (siehe Abbildung 3.4).
- Erstellen einer binären Maske zwischen den bereits von AprilTag3 gefundenen Eckpunkten (siehe Abbildung 3.5), um nur relevante Pixel für die Residuenberechnung zu verwenden.
- Bestimmen der Parameter n_0 und d für jede Kante unter Verwendung von Algorithmus 1.
- Übermitteln der ROI, der Maske und die mit Anfangswerten initialisierten Parameter θ an den **CeresWrapper**, wobei h mit `min(roi)` und k mit `(max(roi) - h) / 2` initialisiert wird.
- Bestimmen der Schnittpunkte aus den optimierten Hyperebenenparametern n_0 und d unter Zuhilfenahme von Algorithmus 2.

Die letztendlich von dem Optimierungsalgorithmus berücksichtigten Regionen sind nochmals in Abbildung 3.6 hervorgehobenen. Gut zu erkennen ist, dass die Bereiche um die Eckpunkte nicht mit inbegriffen sind, da diese von einem unerwünschten Bildrauschen belegt sind. Die Enden der benachbarten Kanten werden somit ebenfalls ausgeschlossen, da diese zu einem verfälschten Verlauf der Kante führen.

Der in C++ implementierte Ceres-Solver wird mithilfe des **CeresWrappers** eingebunden. Auf C++ Seite verwendet der Wrapper Pybind11, um direkt auf NumPy[1] Datenstrukturreferenzen zuzugreifen. Da sich die Position und Form der Maske während der Optimierung nicht verändert, werden zu Beginn die Pixelkoordinaten aller Residuen und deren Intensitätswert in der ROI erfasst. Dadurch werden alle weiteren Optimierungsschritte auf die hervorgehobenen Regionen in Abbildung 3.6 beschränkt.

[1] NumPy ist eine Programmbibliothek für die Programmiersprache Python, die eine einfache Handhabung von großen mehrdimensionalen Datenstrukturen (Arrays) ermöglicht.

Algorithmus 1: Konstruktion von vier Hyperebenen nach Gleichung 2.19.

Input : Eine Liste C der vier Eckpunkte und Ursprungspunkt o der ROI.
Output: Eine Liste N der Normaleneinheitsvektoren und eine Liste A der Abstände zum Ursprung.

$D \leftarrow \emptyset$
$C' \leftarrow C - o$
for $i \leftarrow 0$ **to** 3 **do**
    $j \leftarrow (i+1) \bmod 4$
    $D \leftarrow D \cup (C'[j] - C'[i])$
    $D[i] \leftarrow [-D[i][1], D[i][0])]$
$N \leftarrow$ `Normalize(`D`)`
$A \leftarrow$ `DotProduct(-`D`,` N`)`
return N, A

Algorithmus 2: Berechnung der Schnittpunkte von vier Hyperebenen nach Gleichung 2.20.

Input : Eine Liste N der Normaleneinheitsvektoren und eine Liste A der Abstände zum Ursprung.
Output: Eine Liste C der Schnittpunkte.

$C \leftarrow \emptyset$
for $i \leftarrow 0$ **to** 3 **do**
    $j \leftarrow (i+1) \bmod 4$
    $det_{inv} \leftarrow (N[i][0] * N[j][1] - N[i][1] * N[j][0])^{-1}$
    $C \leftarrow C' \cup [det_{inv} * (N[i][1] * A[j] - N[j][1] * A[i]), det_{inv} * (N[j][0] * A[i] - N[i][0] * A[j])]$
return C

Algorithmus 3: `operator()` Funktion des `EdgeCostFunctor`.

Input : Die Parameter θ der S-Funktion 2.14 und die Liste R der Residuen, jeweils als Pointer. Eine Liste M der Pixelkoordinaten innerhalb der Maske und die ROI werden bereits bei der Initialisierung des `EdgeCostFunctor` in 4 bereitgestellt.
Output: Aktualisierte Einträge der Liste R.

$n, d, h, k, \sigma \leftarrow \theta$
$j \leftarrow 0$
$l \leftarrow$ `Hyperplane(`n`,` a`)`
for $i \leftarrow 0$ `;` $i < 2 *$ `Size(`R`)` `;` $i \mathrel{+}= 2$ **do**
    $R[j++] \leftarrow$ `GetValueFromSFunction(SignedDistance(`l`, M[i], M[i+1])`, h, k, σ`)`
        $-ROI[M[i], M[i+1]]$
return C

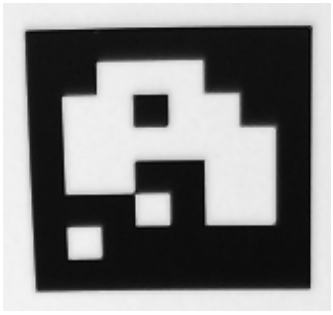


Abbildung 3.4: Als Region of Interest (ROI) bezeichneter Ausschnitt eines detektierten AprilTags.

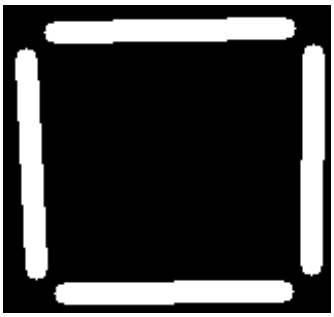


Abbildung 3.5: Die Maske aus der die für die Optimierung berücksichtigten Pixelkoordinaten entnommen werden.

Abbildung 3.6: Veranschaulichung der Maske als Unschärfefilter über der ROI.

Die numerisch differenzierte Kostenfunktion wird dann wie folgt initialisiert:

```
1 new ceres::NumericDiffCostFunction<EdgeCostFunctor, ceres::FORWARD, ceres::DYNAMIC,
2     2, 1, 1, 1, 1>(
3         new EdgeCostFunctor(roi, pts_in_mask),
4         ceres::TAKE_OWNERSHIP,
5         size_of_residuals,
6         ceres::NumericDiffOptions()
7     );
```

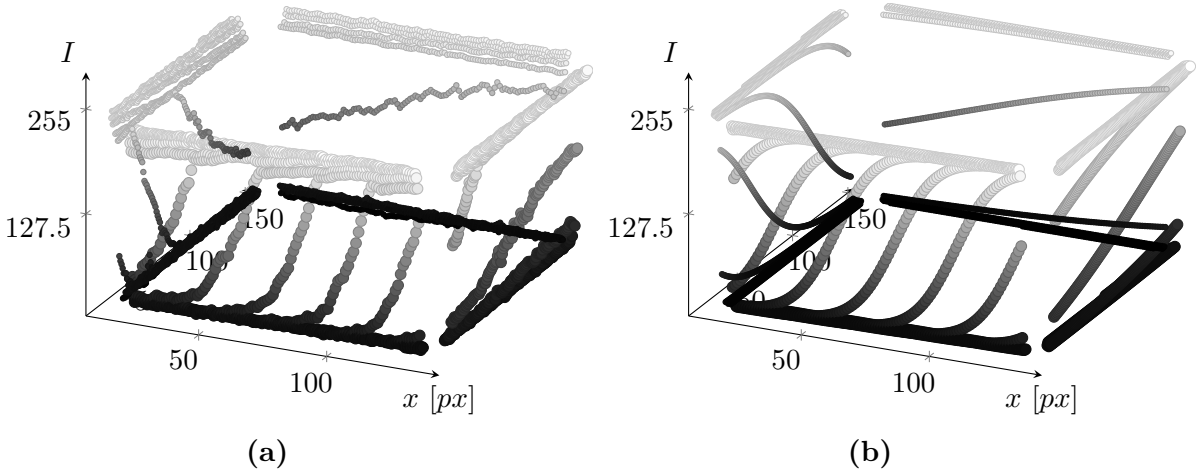
Algorithmus 4: Initialisierung der Ceres-Kostenfunktion in C++.

Der `EdgeCostFunctor` muss eine `operator()` Funktion implementieren (siehe Algorithmus 3), dessen Argumente die Pointer zu den zu optimierenden Parametern θ und zu den Fehlerdifferenzen (Residuals) sind. Dem `EdgeCostFunctor` wird in Zeile 3 des Codeausschnitts 4 die ROI und eine Liste der Koordinaten der Pixel innerhalb der Maske übergeben. Die Anzahl und Dimensionen der Parameter θ müssen Ceres in Zeile 2 mitgeteilt werden. Angefangen mit dem zweidimensionalen Normalenvektor $\vec{n_0}$, dem Abstand d, dem untersten Schwellwert der Intensität h, dem Kontrast k und des Unschärfegrads σ. Die `operator()` Funktion 3 wird iterativ so lange von Ceres mit neu geschätzten Werten für die Parameter θ aufgerufen, bis die zurückgelieferten Fehlerdifferenzen auf ein Minimum konvergiert sind oder 100 Iterationen überschritten werden.

In jedem Optimierungsschritt wird die Liste R der Residuen in Zeile 5 der `operator()` Funktion aktualisiert. Hierfür wird der Funktion 5 `GetValueFromSFunction` die Distanz des derzeit betrachteten Pixels zur Hyperebene `l` (`SignedDistance`), σ, k und h übergeben, woraus nach Gleichung 2.14 die von dem Kantenmodell erwartete Intensität berechnet wird. Die Differenz der erwarteten und der tatsächlichen Intensität in der ROI (Zeile 6) ergibt den zu reduzierenden Fehler.

```
template<typename T>
inline T GetValueFromSFunction(
T dist, T blur_degree, T contrast_over_two, T min_intensity
) const {
return contrast_over_two * (tanh(blur_degree * dist) + 1) + min_intensity;
}
```

Algorithmus 5: Methode zur Berechnung der zu erwartenden Intensität nach Gleichung 2.14.



(a) **(b)**

Abbildung 3.7: Darstellung der Pixelintensität innerhalb der Maske in der ROI **(a)** und des Kantenmodells **(b)**.

Ist eines der Abbruchbedingungen des Ceres-Solvers erfüllt und wurde ein Minimum gefunden, sieht das Ergebnis des Markers in Abbildung 3.4 wie die Abbildungen in 3.7 aus. Die Intensität der Pixel in der Maske **(a)** sind nahezu identisch zu den geschätzten Intensitäten der Modellfunktion **(b)**.

Eines der Ergebnisse der Experimente im nächsten Kapitel ist, dass künstlich induzierte Unschärfe und ein adaptives Weichzeichnen der ROI die Eckpunkterfassung zusätzlich verbessert - besonders bei synthetisch generierten Bildern, bei denen die Kanten eher der Stufenfunktion aus Abbildung 2.3 ähneln. Aus diesem Grund wurde eine weitere Methode entwickelt, dessen Implementierung im Folgenden beschrieben wird.

3.3 Filter-Methode

Dem Modul **ImageProcessing** wird eine neue Klasse `Method` hinzugefügt, die für jede der im Folgenden beschriebenen Methoden eine Instanz erstellt und dessen Ergebnisse und Parameter verwaltet. Die erweiterte Eckpunkterfassung wird dann auf jede der erstellten `Method`-Instanzen angewendet und zum Schluss wird die Instanz mit dem geringsten Reprojektionsfehler für weitere Berechnungen übernommen. Das bedeutet, dass die Anzahl an Ausführungen der erweiterten Eckpunkterfassung mit Anzahl der `Method`-Instanzen steigt.

Die Idee der Filter-Methode ist schlicht, jedoch ineffizient. Die Laufzeit spielt in dem Andockszenario im Gegensatz zur Präzision allerdings eine untergeordnete Rolle und auch bei Offlineberechnungen, wie der Kamerakalibrierung, kann diese Methode nach Bedarf problemlos mit in den Prozess eingebunden werden.

Zahlreiche Simulationen und Tests haben ergeben, dass je nach Unschärfegrad des Ausgangsbildes und Orientierung der Marker im Bild, unterschiedliche Vorverarbeitungen die Eckpunkterfassung weiter verbessern. Die Auswertung der Versuche in Unterabschnitt 4.3.1 kommt zu dem Ergebnis, dass die OpenCV-Methoden Bilateral- und GaussianBlur-Filter (siehe Unterabschnitt 2.5.1 und 2.5.2 respektive) in verschiedenen Kernelgrößen, bei einmaliger Anwendung vor der Kantenapproximierung aus dem vorigen Abschnitt, den Positionsfehler und die Standardabweichung deutlich verringert.

3.4 Koordinatentransformation

Nachdem die Parameter der Kanten nun bestmöglich bestimmt sind, werden die Schnittpunkte der adjazenten Kanten mittels Gleichung 2.20 berechnet und zusammen mit den korrespondierenden 3D Punkten dem Perspective-n-Point Algorithmus (siehe Unterabschnitt 2.1.2) übergeben, der den Rotationsvektor $\vec{r}_c$ und Translationsvektor $\vec{t}_c$ im Kamerakoordinatensystem, dargestellt in Abbildung 2.1, zurückliefert. Für weiterführende Berechnungen zur Bestimmung der Trajektorie bedarf es jedoch noch gewissen Transformationen. Angefangen mit der Überführung der extrinsischen Parameter vom Kamerakoordinatensystem ins Weltkoordinatensystem. Hierfür muss zunächst der Rotationsvektor $\vec{r}_c$ in eine Rotationsmatrix R_c konvertiert werden. Eine effiziente Methode dies zu erreichen ist die nach Olinde Rodrigues benannte Rotationsgleichung:

$$\begin{aligned} &\varphi \leftarrow norm(\vec{r}_c) \\ &\vec{r} \leftarrow \vec{r}_c/\varphi \\ &R_c = \cos(\varphi)I + (1-\cos(\varphi))rr^T + \sin(\varphi)\begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix} \end{aligned} \tag{3.2}$$

Hieraus lässt sich nun durch Konkatenation die bereits aus Unterabschnitt 2.1.1 bekannte Rotations- und Translationsmatrix $[R_c|t_c]$ bilden, die abschließend noch invertiert werden muss,

wodurch sie in das Weltkoordinatensystem überführt wird. Da die Inverse einer Rotationsmatrix gleich ihrer Transponierten ist, wird dies am effizientesten folgendermaßen berechnet:

$$[R_w|t_w] = [R_c|t_c]^{-1} = [R_c^T| - R_c^T t_c] \tag{3.3}$$

$[R_w|t_w]$ beschreibt die Pose der Kamera relativ zum Weltkoordinatenursprung. Die Definition des Weltkoordinatensystems relativ zu den detektierten AprilTags ist beliebig. Wie in Unterabschnitt 2.2.1 erwähnt definiert die Bibliothek AprilTag3 das Koordinatensystem im Zentrum des Tags mit der y-Achse nach unten und die z-Achse in die Bildebene zeigend. In dieser Arbeit wird jedoch ein Verbund von AprilTags (Pattern, siehe Unterabschnitt 4.2.4) verwendet, sodass ein gemeinsames Koordinatensystem für alle Tags definiert werden muss. Die Standardausrichtung des Patterns ist aufrecht und das Tag mit der niedrigsten ID ist oben links, in dessen obere linke Ecke auch das Weltkoordinatensystem verschoben wird. Da das Pattern für eine größere horizontale Basis noch um $-90°$ um dessen z-Achse gedreht wird, bedeutet die Transformation im Detail:

$$\begin{aligned} X_c &\rightarrow -Y_w \\ Y_c &\rightarrow \ \ X_w \\ Z_c &\rightarrow \ \ Z_w \end{aligned}$$

Die relativen Achsenorientierungen des Kamera- und Weltkoordiantensystems zeigt Abbildung 2.1 und die genaue Ursprungsposition am Pattern zeigt Abbildung 2.13.

Im nächsten Abschnitt werden die Transformationsbezugssysteme des Robot Operating Systems (ROS) vorgestellt. Eines der Bezugssysteme ist der `base_frame`, das lokal am Roboter als Referenzpunkt für Sensoren und andere Bauteile dient. Die exakte Bestimmung der Position der Kamera zur Basis des Roboters wird für gewöhnlich durch eine Hand-Eye-Kalibrierung (siehe Abschnitt 2.7) ermittelt, bei der korrespondierende Posen der Kamera und des Roboters mittels der Standardmethode von Tsai Lenz [30] in eine räumliche Beziehung gebracht werden. Jedoch ist die einzige verfügbare Möglichkeit, die Basis des Roboters zu ermitteln, die Odometrie und diese ist auch bei sehr behutsamen Bewegungen zu ungenau, um die starre Transformation zur Kamera zu bestimmen. Daher wird die Position der Kamera so genau wie möglich händisch vermessen und einfachheitshalber zentriert auf der in Fahrtrichtung verlaufenden Achse des Roboters angebracht. Abbildung 4.8 zeigt den Montagepunkt der Kamera am Roboter.

Transformationen, wie 2.4 und 3.3, heißen passiv- oder Alias-Transformation und ändern nur die Koordinaten der Punkte, während die Punkte selbst unverändert bleiben. Sie werden linksseitig an die zu transformierende Pose multipliziert. Für die starre Transformation bezüglich des `base_frame` ist es jedoch sinnvoller, das Koordinatensystem beizubehalten und nur die Pose selbst zu transformieren. Eine solche Transformation heißt aktiv- oder Alibi-Transformation und wird rechtsseitig an die zu transformierende Pose multipliziert.

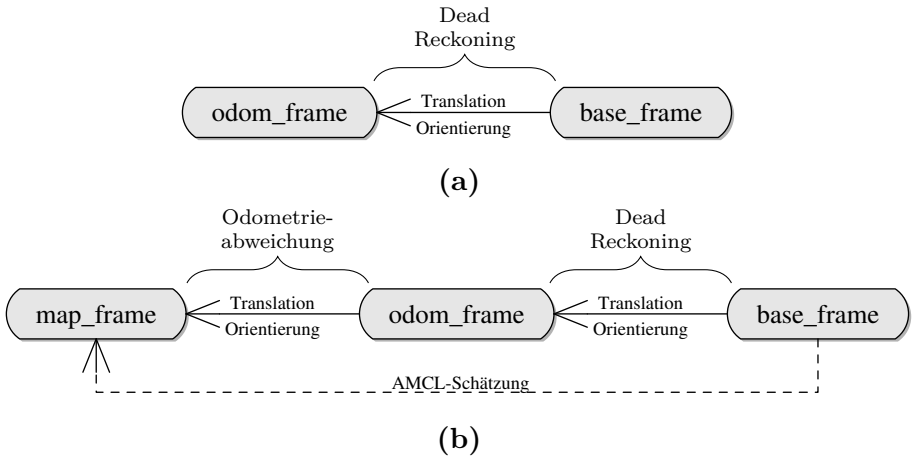


Abbildung 3.8: Beziehung der ROS-Transformationen bei **(a)** Odometrie- und **(b)** AMCL-Map-Lokalisierung. [31]

3.5 Trajektorienplanung

Das Andocken beinhaltet für gewöhnlich ein Rangieren der Agenten in der Nähe von Hindernissen beziehungsweise der Stationen. Das Befahren dieser Bereiche wird bei Verwendung der freien Navigation der Agentbetriebssoftware verhindert, da die Position des Agenten in diesem Betriebsmodus nicht immer optimal bestimmt wird. Das liegt unter anderem an der adaptiven Monte Carlo Lokalisierung (AMCL), die bei Erhalt neuer Sensorinformationen zu diskreten Sprüngen in der Posenschätzung führt. Im Folgenden wird kurz auf AMCL und ROS-Transformationen eingegangen.

3.5.1 AMCL und Transformationsbezugssysteme in ROS

AMCL versucht die Laserscans mit einer bekannten 2D Karte abzugleichen und so festzustellen, ob die auf der Odometrie (Dead Reckoning) basierende Posenschätzung abweicht. Diese Abweichung, bezeichnet als Odometrieabweichung in Abbildung 3.8 **(b)**, wird dann kompensiert, indem eine Transformation zwischen dem `map_frame` und dem `odom_frame` durchgeführt wird, so dass am Ende die Transformation `map_frame`→`base_frame` der tatsächlichen Pose des Agenten in der Welt entspricht.

`map_frame`:
Die als `map_frame` bezeichnete Transformation ist ein in der Welt statisches Transformationsbezugssystem, dessen z-Achse nach oben zeigt. Die Position einer mobilen Plattform relativ zum `map_frame` driftet im Laufe der Zeit nicht, wodurch das Bezugssystem als langfristige globale Referenz nützlich ist. Jedoch ist es nicht kontinuierlich, d.h. die Position einer mobilen Plattform im `map_frame` ändert sich in diskreten Sprüngen. Im normalen Betrieb berechnet ein Lokalisierungsalgorithmus die Roboterposition auf der Grundlage von Sensorbeobachtungen ständig neu, wodurch ein Drift vermieden wird. [32]

`odom_frame`:
Das Odometrie-Transformationsbezugssystem ist ebenfalls statisch in der Welt. Die Position einer mobilen Plattform ist im Laufe der Zeit jedoch von unbegrenztem Drift betroffen. Dieser Drift macht den `odom_frame` als langfristige globale Referenz unbrauchbar. Die Pose eines Roboters ist jedoch garantiert kontinuierlich, d.h. die Pose einer mobilen Plattform im `odom_frame` verläuft immer gleichmäßig und ohne diskrete Sprünge. Für die Berechnung der Odometrie existieren verschiedene Methoden, z.B. durch Inkrementalgeber bei der Rad-Odometrie, durch visuelle Odometrie oder mittels einer Trägheitsmesseinheit. [32]

`base_frame`:
Der `base_frame` ist starr mit der Roboterbasis verbunden. Die Position und Ausrichtung an der Basis ist beliebig und bietet einen klaren Bezugspunkt für am Roboter angebrachte Hardware, wie Kameras, Laserscanner, etc. [32]

Zusammengefasst ist der `map_frame` mit AMCL als Lokalisierungsalgorithmus, aufgrund der Diskontinuität, nur als langfristige globale Referenz nützlich. Der `odom_frame` hat dieses Problem nicht und ist somit bei kurzen, lokalen Aktionen besser geeignet. Aus diesem Grund wird nach Einnahme der markerzentrierenden Stellung (siehe Diagramm 3.1) AMCL deaktiviert, die Trajektorie ermittelt und lediglich mit Odometrie-Lokalisierung, einer Bezugssystemrelation nach Abbildung 3.8 **(a)**, zur Andockstellung gefahren.

3.5.2 Optimierung des Controllers

Der Fehlende Bezug zum `map_frame` eröffnet neue Probleme und bedarf gewissen Anpassungen des im **Controller** verwendeten Reglers. Zudem generiert die neu implementierte Trajektorienplanung mit ihrer Klothoidenannäherung Segmente mit sehr kleinen Zeitintervallen, womit die derzeitige Umsetzung der Regelschleifenrate nicht zurechtkommt.

Regelgrößen und Dämpfung

Der einfachste Ansatz für den Entwurf eines Nachlaufreglers besteht in der näherungsweisen Linearisierung der Fehlerdynamik um den Referenzpfad. Der hier verwendete Regler basiert ebenfalls auf eine approximierte Linearisierung, beschrieben in [33]. Die Reglerausgaben für die lineare Geschwindigkeit v und die Winkelgeschwindigkeit w ergeben sich aus dem Fehlervektor $\vec{e}$ und den Eingaben

$$u_1 = -k_1 e_x \tag{3.4}$$
$$u_2 = -sign(v) k_2 e_y - k_3 e_z \tag{3.5}$$

mit den Reglerverstärkungen

$$k_1 = k_3 = a, \quad k_2 = \frac{a^2 - w_d^2}{v_d^2} \tag{3.6}$$

mit Dämpfungskoeffizient a und den Regelgrößen v_d, w_d zu

$$v = v_d \cos(e_z) - u_1 \tag{3.7}$$
$$w = w_d - u_2 \tag{3.8}$$

Im Standardbetrieb der freien Navigation sind die Regelgrößen und der Dämpfungskoeffizient konstant, so dass ein gewünschtes Gleichgewicht aus Abfahrgenauigkeit und Geschwindigkeit erreicht wird. Da die freie Navigation AMCL verwendet, sind Abweichungen der Odometrielokalisierung ein untergeordnetes Problem und mögliche Pfadabweichungen, die durch leichtes Schwingen zu Pfadverlängerungen führen, werden ebenfalls kompensiert. Wie zuvor erwähnt ist jedoch für die Zeit des Dockingvorgangs AMCL deaktiviert, wodurch jeglicher Schlupf der Räder minimal gehalten werden muss. Zudem weiß der Regler in dieser Zeit nicht, wo er sich im `map_frame` befindet und ob der vorgegebene Pfad tatsächlich abgefahren wurde. Aus diesen Gründen wird zum einen die Regelgröße v_d auf ein Minimum begrenzt, wodurch der Roboter den Pfad langsam abfährt und zum anderen wird der Dämpfungskoeffizient a angepasst, um ein Schwingen des Reglers zu verhindern. Die optimalen Werte werden im Abschnitt 4.4 experimentell ermittelt.

Regelschleifenrate

Die Wiederholungsrate der Regelschleife des **Controller**-Moduls wird mittels ROS-Rate umgesetzt. ROS-Rate ist eine leistungsstarke ROS-Funktion, die standardmäßig für Regelkreise benutzt wird - sei es zum Lesen eines Sensors, zur Steuerung eines Motors usw. Die Frequenz wird in Herz angegeben und legt die maximale Wiederholungsrate fest. Ist die Dauer der Ausführung der Schleife kürzer, wartet ROS die restliche Zeit. Ist sie länger, wird sofort mit dem nächsten Schleifendurchlauf begonnen. Abbildung 3.9 illustriert dies mit einer Rate von 50 Hz, was der Standardfrequenz des implementierten **Controller**-Moduls bei freier Navigation entspricht.

Nun bezieht sich die Ausführungszeit des Codes der Schleife jedoch nur auf Berechnungen zur Hinderniserkennung, Reglerausgaben, Pfadabweichungen usw., aber nicht auf die gewünschte Ausführungszeit der Pfadsegmente. Diese verteilen sich für gewöhnlich über mehrere Schleifendurchläufe, da sie selten kürzer sind als 20 ms (50 Hz). Das bedeutet, dass für die volle Dauer jeder Schleife die zuletzt berechneten Stellgrößen ausgeführt werden. Ist die Segmentzeit geringfügig kürzer als ein Vielfaches der Rate, wird im ungünstigsten Fall das momentane Segment bis zu 20 ms zu lang ausgeführt, was bei der Standardgeschwindigkeit des Roboterbetriebssystems von 0.3 m/s einer Abweichung von 6 mm entspricht. Je nach Länge des Pfades und Art des Segments (Kreis oder Gerade) führt dies zu beträchtlichen Abweichungen der Zielpose.

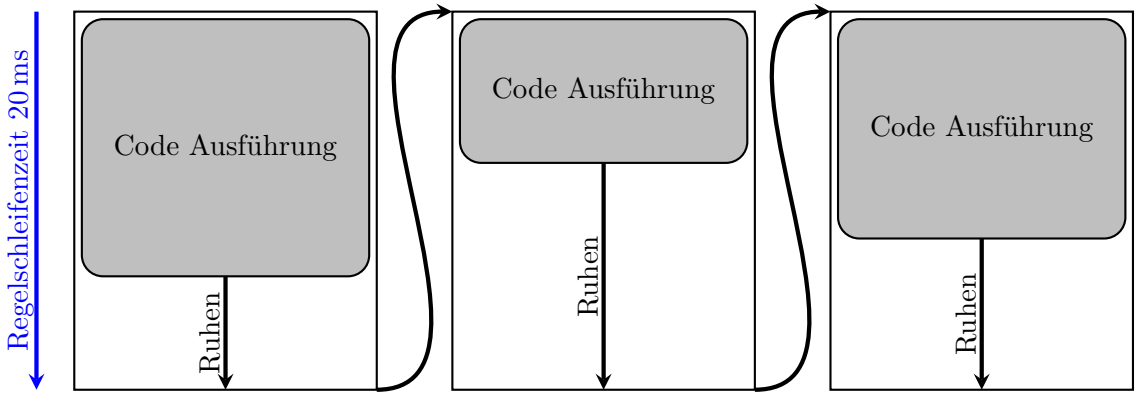


Abbildung 3.9: Illustration der Funktionsweise der ROS-Rate.

Bei den Pfaden der für diese Arbeit implementierten Trajektorienplanung treten Fälle dieser Art in jeder Kurve auf, da die Ausführungszeiten der angenäherten Klothoidensegmente im Übergang zwischen geraden und kreisförmigen Abschnitten kürzer sind als 20 ms und somit jedes Mal für mindestens 20 ms ausgeführt werden, was besonders bei Kreissegmenten, durch ein zu weites Eindrehen, zu Abweichungen vom vorgegebenen Pfad führt.

Eine Maßnahme zur Reduzierung dieser Abweichungen ist die Verminderung der Geschwindigkeit, wodurch auch andere unerwünschte Effekte, wie z.B. Schlupf, minimiert werden. Eine weitere Maßnahme ist die Erhöhung der Wiederholungsrate. Um jedoch das System nicht unnötig zu belasten, wird eine dynamische Rate implementiert, die bei Segmentausführungszeiten von über 20 ms eine Rate nahe der 50 Hz ermittelt, die leicht über dem Vielfachen der Ausführungszeit liegt. Für Segmente unterhalb der 20 ms wird für diese kurze Zeit die Rate auf das Reziproke der Ausführungszeit erhöht.

$$
\begin{aligned}
i &= \frac{t_s}{t_r} \\
r_{dyn} &= \begin{cases} \dfrac{\lfloor i \rfloor}{i t_r} - \Delta t & \text{, wenn } i \geq 1 \\ \dfrac{1}{t_s} & \text{, sonst} \end{cases}
\end{aligned}
\tag{3.9}
$$

Aus der Segmentausführungszeit t_s und der Standardausführungszeit t_r in Sekunden ergibt sich die Anzahl der Schleifeniterationen i. Ist $i \geq 1$ wird die genaue Rate in Hertz bestimmt, die ein Vielfaches von t_s ergibt. Zusätzlich wird ein sehr kurzes Δt subtrahiert, um ein ungewolltes Überschreiten von t_s zu verhindern.

3.5.3 Pfadgenerierung

AMCL ist deaktiviert, die markerzentrierende Stellung bestätigt und das Modul **ImageProcessing** liefert die Pose der hochpräzisen Andockstellung an den `Scheduler` im **WebGUI**-Modul. Von dort wird die Zielpose zusammen mit der Startpose, also der aktuellen Pose des Agenten, und einer eindeutigen Topic-Bezeichnung an das **PathPlanning**-Modul übergeben. Hier wird durch Aufruf des Algorithmus A.1 der in Abschnitt 3.5 beschriebene Prozess gestartet und ein für Agenten interpretierbar segmentierter Pfad zurückgegeben. Dieser wird dann über das zugewiesene Topic zurück an den `Scheduler` übermittelt, wo eine Callback-Methode an gleicher Stelle die Abarbeitung des Manövers fortsetzt.

Es ist möglich, dass die Generierung des Pfades fehlschlägt. In so einem Fall muss eine neue Position vom Agenten eingenommen und der Prozess wiederholt werden. Der Grund für eine erfolglose Generierung ist jedoch gut verstanden und wird, bei Vermeidung der als rote Posen in Abbildung 3.10 dargestellten Anfahrstellungen, nicht eintreten. Aufgrund der Distanz und der relativen Orientierung zwischen zwei Posen besteht die Möglichkeit, dass die Annäherung der Kreissegmente des Reed-Shepp-Pfades an Klothoide in Algorithmus A.3 nicht gelingt. Das liegt daran, dass die derzeitige Implementierung keine Fahrtrichtungswechsel zulässt und keine gültige Klothoide bei direkt aufeinanderfolgenden entgegengesetzten Kreissegmenten berechnet wird. Bestehen also alle möglichen Reed-Shepp-Pfade nur aus Trajektorien, die diese beiden Einschränkungen verletzen, schlägt die Pfadgenerierung fehl. Welche Anfahrtposen relativ zur Ausrichtung der Zielpose nun optimal sind, wird in Abbildung 3.10 verdeutlicht. In dem Szenario **(a)** soll sich der Agent an dem Punkt (0, 0) in Richtung Osten (positive x-Achse in der Abbildung) positionieren. Die AprilTag-Marker, ebenfalls am Punkt (0, 0), zeigen mit der eigenen z-Achse Richtung Norden (positive y-Achse in der Abbildung). Da die Detektierung der Marker bei einem sehr flachen Winkel versagt oder unpräzise ist, sind die gegen 0 und gegen $-\pi$ strebenden Bereiche des relativen Anfahrstellungswinkels in orange beziehungsweise rot eingefärbt. Alle Posen, die mit einem roten Punkt markiert sind, sind nun eben diese Anfahrposen, von denen es nicht möglich ist, mit approximiert kontinuierlich gekrümmten Kreissegmenten die Zielpose zu erreichen. Im Normalfall sind Zielpose und Marker jedoch nicht an gleicher Position, daher zeigt **(b)** ein Szenario, bei dem die AprilTag-Marker weiterhin im Punkt (0, 0) mit gleicher Orientierung positioniert sind, jedoch die Zielpose in den Punkt (5, 5) verschoben ist. Das Wählen einer validen Anfahrposition ist dadurch bereits bedeutend schwerer, daher wird durch geeignete Visualisierungen in der **WebGUI** 3.11 die Bestimmung aller nötigen Parameter erleichtert. Eine kurze Erläuterung der Benutzeroberfläche wird im nächsten Abschnitt behandelt.

Weiterhin werden die Pfade in Zeile 14 des Algorithmus A.1 nach ihrer Segmentanzahl aufsteigend sortiert, damit Pfade mit niedriger Segmentanzahl bevorzugt werden, um unnötige Übergänge zu verhindern. In Zeile 18 werden, wie zuvor bereits erwähnt, alle Pfade mit Segmenten, dessen Fahrtrichtung von der des ersten Segments abweicht, übersprungen, da auch dies einen unnötigen Übergang darstellt und die derzeitige Implementierung der Bestimmung des Krümmungsmittelpunktes (siehe Algorithmus A.4) einen Richtungswechsel nicht unterstützt.

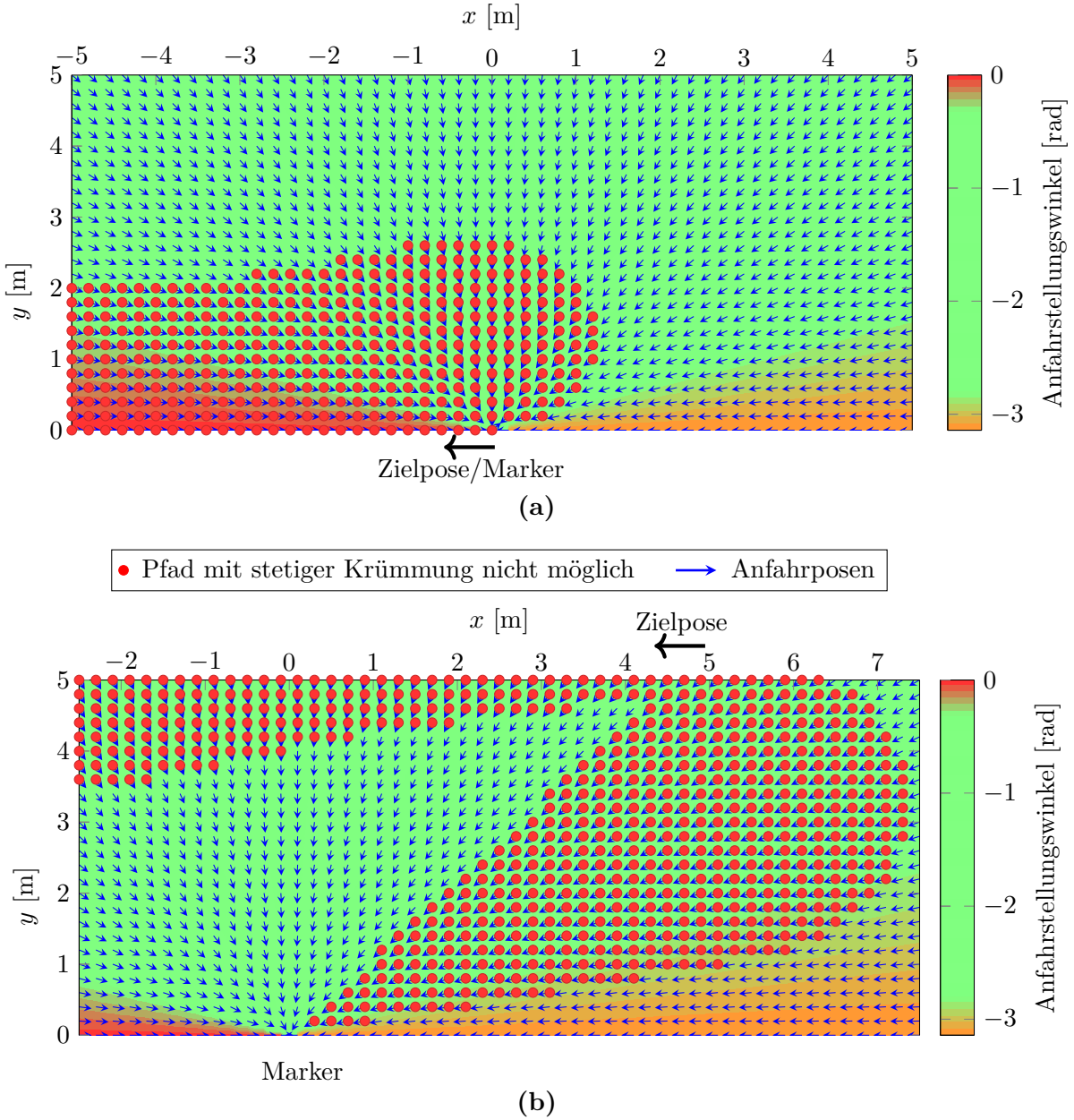


Abbildung 3.10: Darstellung der Anfahrposen, von denen eine Trajektorienplanung mit kontinuierlicher Krümmung möglich ist. In **(a)** soll sich der Agent bei der Zielpose im Punkt (0, 0) in Richtung positiver x-Achse positionieren. Die AprilTag-Marker, ebenfalls im Punkt (0, 0), zeigen mit der markereigenen z-Achse Richtung positiver y-Achse. **(b)** zeigt ein Szenario in dem Zielpose (5, 5) und Marker (0, 0) nicht an gleicher Position sind.

3.6 Benutzeroberfläche

In diesem Abschnitt wird kurz die Benutzeroberfläche in Bezug auf die Implementierungen, speziell für das Dockingmanöver, erläutert.

Abbildung 3.11 illustriert die einzelnen Schritte der Auftragserstellung in dem Graphical User Interface (GUI) der **ControlStation**. **(a)** zeigt eine unbeschnittene Ansicht der GUI mit geöffneten Task Panel links, in dem Aufträge verwaltet werden. Der erste Ausschnitt **(b)** hebt die Optionen des Distrikts und der Andockaktion mit gewünschten Parametern für die relative Andockstellung hervor und zeigt die aktualisierte Visualisierung der Andockumgebung. Zur Definition der Andockstellung werden drei Parameter benötigt,

- **Distanz** [m]**:** Euklidischer Abstand zwischen der Andockposition und der Station.
- **Richtung** [°]**:** Relativer Winkel zur Station, der zusammen mit der Distanz die Polarkoordinaten der Andockposition ergeben. Eine Richtung von 0° entspricht Osten bei aus der Bildebene zeigender Drehachse.
- **Orientierung** [°]**:** Blickrichtung des Roboters in der Andockstellung. Eine Orientierung von 0° entspricht einer Blickrichtung auf die Station bei aus der Bildebene zeigender Drehachse.

In dem letzten Ausschnitt **(c)** wird die Andockumgebung, mit der markerzentrierenden Stellung außerhalb der rot gekreuzten Zone und eingezeichneter validen Trajektorie, gezeigt.

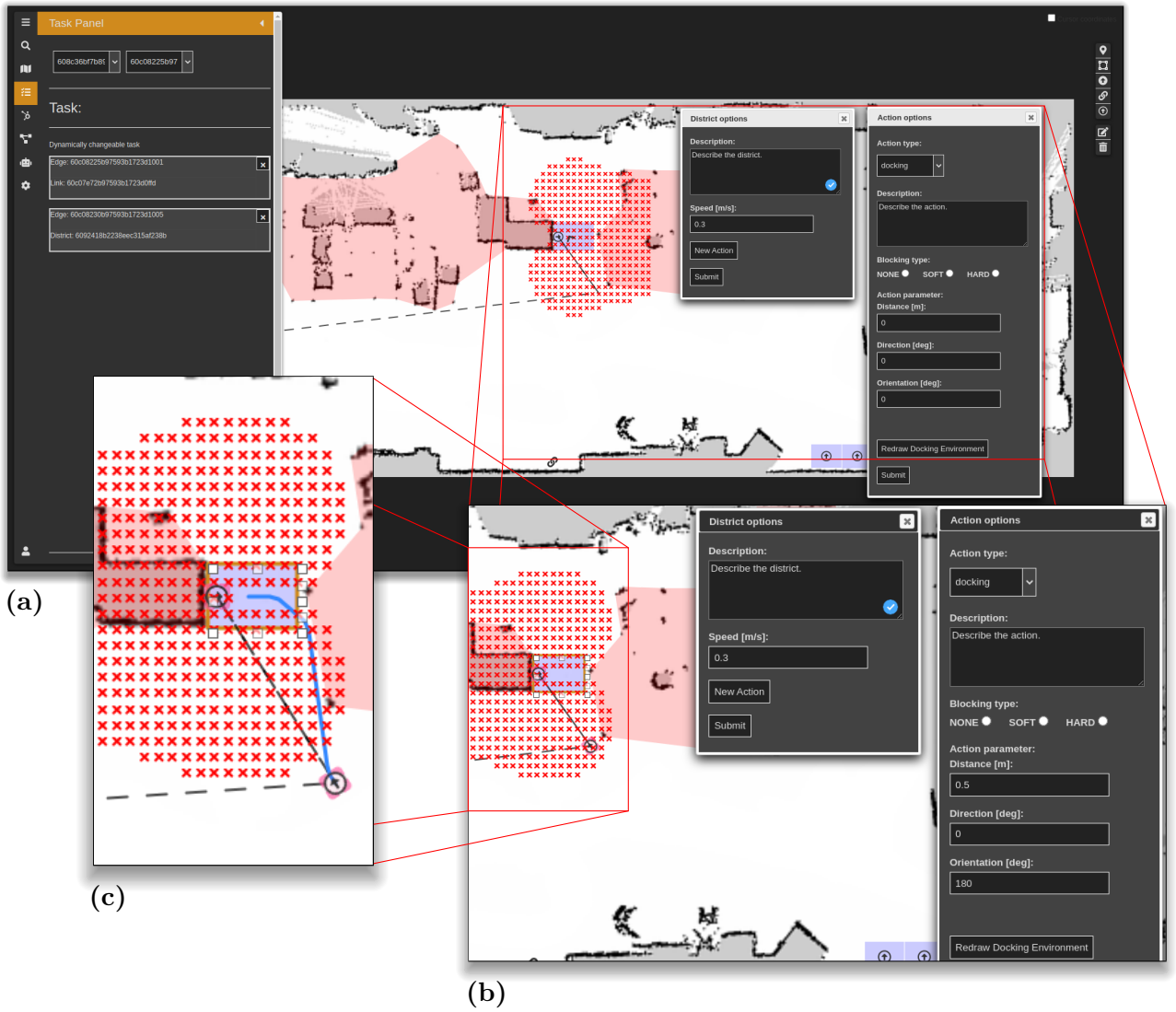


Abbildung 3.11: **(a)** Benutzeroberfläche (GUI) der **ControlStation** mit geöffneten Task Panel links, in dem Aufträge verwaltet werden. **(b)** Ausschnitt der Optionen des Distrikts und der Andockaktion mit gewünschten Parametern für die relative Andockstellung und aktualisierter Visualisierung der Andockumgebung. **(c)** Ausschnitt der Andockumgebung mit der markerzentrierenden Stellung außerhalb der rot gekreuzten Zone und eingezeichneter validen Trajektorie.

Kapitel 4

Evaluierung

In diesem Kapitel werden die einzelnen Module weitestgehend isoliert und abschließend als Gesamtheit evaluiert. Zu diesem Zweck werden zunächst Simulationen durchgeführt, um die Korrektheit der implementierten Methoden, unabhängig von der verwendeten Hardware, zu testen. Primär wird auf diese Weise die erweiterte Kantenerkennung, die Trajektorienplanung und Abfahrgenauigkeit der geplanten Trajektorie überprüft.

4.1 Methodik

Die stochastische Analyse der generierten Daten in den folgenden Experimenten beschränkt sich - in den Fällen mit statistisch aussagekräftiger Anzahl an Messpunkten - auf die Bildung des Mittelwertes

$$\mu = \frac{1}{N} \sum_{i=1}^{N} x_i, \tag{4.1}$$

der Standardabweichung

$$\sigma = \sqrt{\frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu)^2} \tag{4.2}$$

und des Standardfehlers des Mittelwertes

$$\sigma_\mu = \frac{\sigma}{\sqrt{N}} \tag{4.3}$$

Ein Messergebnis wird dann kompakt in der Form $x = \mu \pm \sigma_\mu$ angegeben.

Ohne weitere Vorinformationen wird als statistisches Modell für die zufällige Abweichung der Messwerte vom wahren Wert die übliche Annahme einer gaußschen Normalverteilung mit einer Standardabweichung σ angenommen.

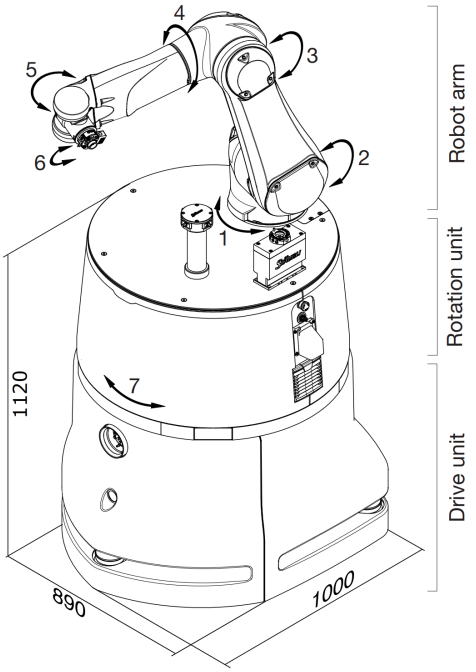


Abbildung 4.1: Roboterplattform HelMo von Stäubli und WFT. Abmessungen sind in Millimeter.[1]

Abbildung 4.2: Blackfly® S BFS-PGE-16S2C-CS von FLIR.[2]

Abbildung 4.3: A4Z2812CS-MPIR von Computar.[3]

4.2 Hardware

Zur Entwicklung der Algorithmen und Durchführung der Experimente sind eine Vielzahl von Gerätschaften nötig, die in diesem Abschnitt kurz aufgeführt werden.

4.2.1 Roboter

Das Kernstück der Experimente bildet die Roboterplattform HelMo, die 2015 von Stäubli und WFT entwickelt wurde. Durch den modularen Aufbau und der robusten, gefederten, differenziellen Antriebseinheit ist der Einsatz von HelMo in nahezu allen Industriebereichen wie z.B. der Logistik oder Montage möglich. Mithilfe von drei integrierten Laserscannern überwacht der Roboter permanent seine Umgebung, wodurch ein selbstständiges manövrieren und navigieren ermöglicht wird. In Abbildung 4.1 sind die Systemkomponenten und Abmessungen der 2018er Version von HelMo dargestellt, die sich, bis auf den angebrachten Roboterarm und Abdeckungen, nur geringfügig von der für diese Arbeit verwendeten Version unterscheidet.

[1] https://www.industry-plaza.com/medias/5/6/5/010561565.pdf
[2] https://www.worthylambtech.com/globalassets/imported-assets/image/blackflys-g3-gige.png
[3] https://www.vision-dimension.com/media/image/da/8e/78/A4Z2812CS-MPIR.png

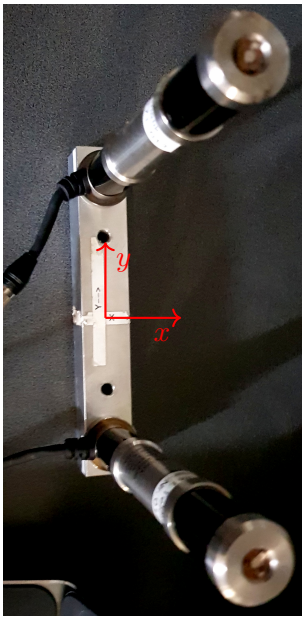


Abbildung 4.4: Koordinatensystem des Sensorrahmens mit der z-Achse aus der Bildebene herauskommend.

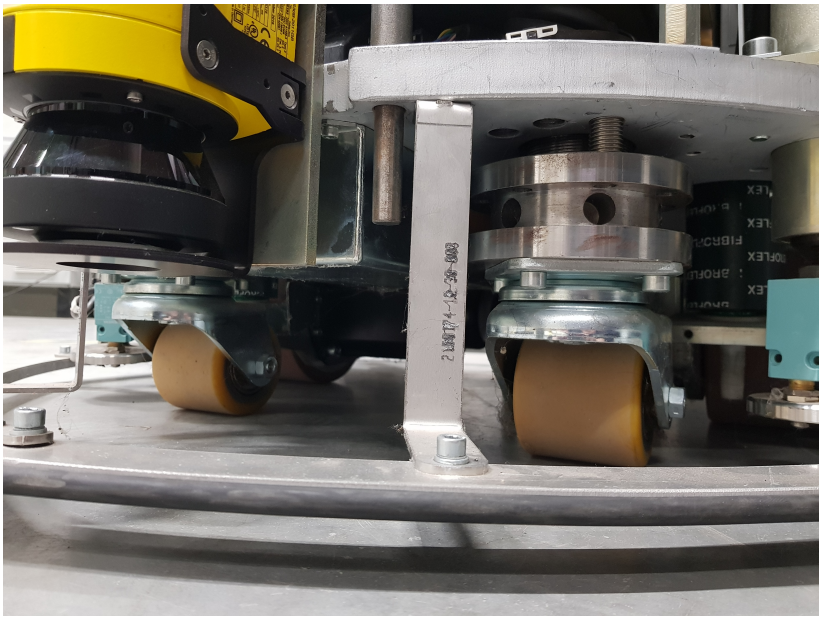


Abbildung 4.5: Differenzielle Antriebseinheit mit zwei Castor-Rollen in Fahrtrichtung des HelMo.

Das Gewicht von HelMo beträgt in etwa 710 kg und übt einen durchschnittlichen Druck von 10 N/mm^2 auf den Boden aus, wodurch mit keinem hohen Schlupf der breiten Hartgummireifen zu rechnen ist. [34] Abbildung 4.5 zeigt die differenzielle Antriebseinheit links und rechts weiter hinten im Bild und zwei der drei robusten Lenkrollen (Castor-Rollen).

4.2.2 Kamera

Am Zentrum für Telematik e.V. wird in laufenden sowie abgeschlossenen Projekten auf die Kameras von FLIR gesetzt, infolgedessen sich bereits Wissen und Erfahrung im Umgang mit deren Produkten angesammelt hat und überdies eine betriebsinterne ROS-Node (**SpinnakerWrapper**) für die zugehörige Spinnaker SDK existiert. In dieser Arbeit wird die BFS-PGE-16S2C-CS der Familie Blackfly® S verwendet (siehe Abbildung 4.2), die über einen 1/2.9" CMOS Sensor mit globalen Shutter verfügt, wodurch ein simultanes Belichten aller Transistoren des Sensors ermöglicht wird und es somit nicht zu einem Rolling-Shutter-Effekt kommt, der zu Lagefehler in dynamischen Aufnahmen führt. Die Auflösung der Kamera mit 1440×1080 Pixel ist für die Andockexperimente, in denen Bilder aus Entfernungen von über drei Metern aufgenommen werden,

unzureichend, da die Markerdetektierung bei einer geringen Pixelbreite an Genauigkeit verliert oder fehlschlägt. Dies wird jedoch mit einem passenden Objektiv ausgeglichen.

4.2.3 Objektiv

Das Objektiv A4Z2812CS-MPIR von Computar (siehe Abbildung 4.3) ist eines der wenigen Objektive, das auf das CS-Gewinde der Blackfly passt und einen verhältnismäßig großen Zoombereich besitzt. Das Objektiv hat eine Brennweite von 2.8 - 10.0 mm und manuelle Blenden-, Zoom-, und Fokussteuerung. Nach der Kalibrierung ist der Zoom und der Fokus gut mit einer Feststellschraube zu arretieren, wobei der Blendenring mit beispielsweise Klebeband fixiert werden muss. Um die geringe Auflösung der Kamera auszugleichen, wird die Brennweite so hoch wie möglich - bei nicht degradierender Schärfe - gewählt und beträgt nach Auswertung der Kalibrierung 9.78 mm.

4.2.4 Pattern

Mit Pattern wird in dieser Arbeit eine bestimmte Anordnung von AprilTags auf einer ebenen Fläche bezeichnet. Auf der Kiste links in Abbildung 4.6 ist das hier verwendete Pattern, das aus 24 Tags in einer 6×4 Anordnung besteht, zu sehen. Die einzelnen Tags haben eine Seitenlänge von 0.085 m und somit einen Abstand zwischen den schwarzen Eckpunkten von 0.068 m, wodurch sich eine laterale Distanz zwischen der linken und rechten oberen schwarzen Ecke des Patterns von 0.493 m ergibt. Das Pattern ist auf eine Glasplatte mittels spezieller Tinte in einem UV-Druck-Verfahren aufgebracht und unterliegt daher nur sehr minimalen Deformationen, was von äußerster Wichtigkeit ist, denn die an den Perspective-n-Point-Algorithmus übergebenen 3D Punkte werden aus den zuvor angegebenen Maßen der AprilTags händisch bestimmt. Abweichungen durch eine Deformation des Aufdrucks oder der Platte selbst führt zu systematischen Ungenauigkeiten bei der Reprojektion.

4.2.5 iGPS

Im Abschnitt 4.4 dieses Kapitels wird die Genauigkeit des **Controller**-Moduls, der Odometrie und die Wiederholgenauigkeit des Andockmanövers getestet. Bei diesen Experimenten fährt der HelMo Roboter Testpfade ab oder steuert eine vorgegebene Pose relativ zu einer Station so präzise wie möglich mehrfach an. Für die Evaluierung dieser Tests wird ein externes System benötigt, das Präzisionsvermessungen der Roboterpose im Submillimeterbereich durchführen kann. Das Zentrum für Telematik verfügt für solche Anwendungen über das modulare iGPS (infrarot GPS) von Nikon Metrology (ehemals Metris) mit dem sich Messungen in dieser Qualität in relativ einfacher Weise realisieren lassen. Unter optimalen Bedingungen lassen sich für kleinere Messbereiche mittlere Punktfehler von etwa 25 µm erreichen und das bei einer Messrate von ca. 40 Hz, wodurch die Durchführung von Experimenten in Echtzeit möglich ist. Das iGPS besteht aus ortsfesten Transmittern und mobilen Sensoren, deren Position nach dem Prinzip eines 3D-Vorwärtsschnitts bestimmt wird. Die Anzahl von gleichzeitig betriebenen Transmittern ist

Abbildung 4.6: Aufbau der Andockstation mit dem Pattern auf Höhe der auf der Roboterplattform montierten Kamera.

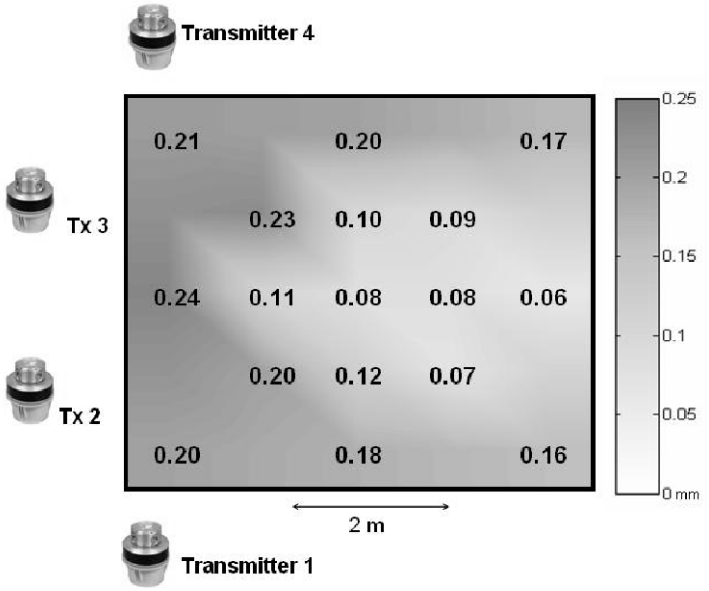


Abbildung 4.7: Darstellung der anzunehmenden räumlichen Durchschnittsfehler bei Positionierung der Transmitter in einem C-Shape. [35]

unbegrenzt. In der Regel wird mit vier Transmittern gearbeitet, um bei der Punktbestimmung neben einer guten Kontrollierbarkeit und besseren Genauigkeit auch weniger Einschränkungen aufgrund der Abschattung einzelner Transmitter berücksichtigen zu müssen. Für eine höchstmögliche Präzision und Verminderung von Abschattungen ist die Positionierung der Transmitter von Bedeutung. Daher wird die von Depenthal [35] empfohlene C-Shape-Konfiguration verwendet und versucht, die Andockstation mit dem Pattern nahe der 0.06 rechts in Abbildung 4.7 zu positionieren. Zur leichteren Auswertung und Validierung der Ausgaben während der Versuchsdurchführung, wird ein Koordinatensystem definiert. In Abbildung 4.9 ist der dafür vorgesehene Sensor zu sehen, dessen Spitze sehr präzise im Messraum erfasst wird. Abbildung 4.8 zeigt einen der vier ortsfesten Transmitter im Hintergrund und den Aufbau am HelMo mit einem weiteren Sensor in Form eines Rahmens, der auf dem Zentrum der Kamera angebracht ist. Dieser Sensorrahmen eignet sich besonders gut um nicht nur seine Position, sondern auch seine Orientierung zu bestimmen und besitzt - so positioniert - eine zur Kamera nahezu identische Pose, wodurch eine Analyse der Messwerte während der Experimente ermöglicht wird. Beim Platzieren der Sensoren ist stets darauf zu achten, dass die schwarzen Bereiche der Stäbe für möglichst alle Transmitter sichtbar sind. Daher war ein Montieren der Kamera innerhalb des Rahmens nicht möglich.

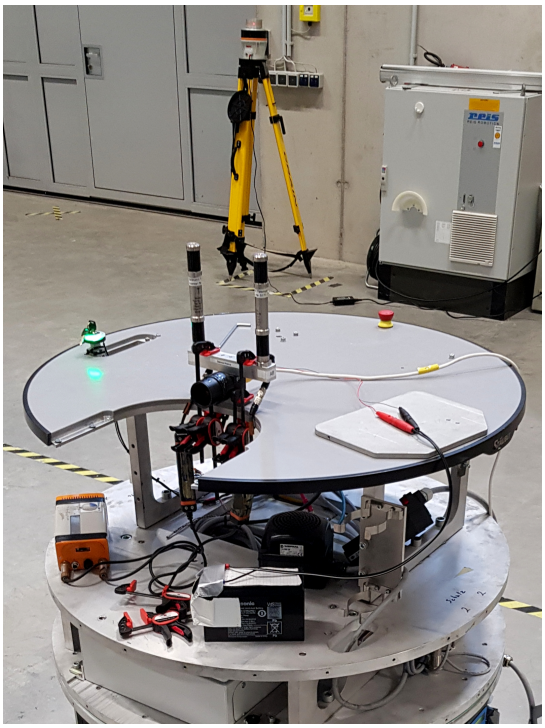


Abbildung 4.8: Kamera, Objektiv und der Sensorrahmen des iGPS montiert auf der Roboterplattform HelMo.

Abbildung 4.9: Sensor zur Punktabtastung des iGPS ausgerichtet auf das Patternzentrum der Andockstation.

4.3 Detektierungsgenauigkeit von AprilTags

Dieser Abschnitt beginnt mit einer ausführlichen Simulation, gefolgt von einem Test mit Aufnahmen einer realen Kamera. Die Simulation soll zunächst feststellen, ob es überhaupt möglich ist, das in Unterabschnitt 2.2.3 vorgestellte Edge Refinement der Bibliothek AprilTag3, zu verbessern und ob alle implementierten Methoden, frei von eventuellen Einflüssen der Kamera und dessen Kalibrierung, optimal funktionieren.

4.3.1 Simulation

Für die Simulation wird dem Modul **ImageProcessing** eine neue Klasse `TagGenerator` hinzugefügt, die AprilTags der Familie 36h11 generiert und bei Bedarf zu Pattern zusammenfasst. Zudem verfügt die Klasse über Methoden, die es ihr erlaubt, die generierten Tags beliebig zu transformieren und fehlerfreie Pixelkoordinaten der Markerecken als Ground Truth zurückzuliefern. Da die Bilder der Tags nicht von einem Objektiv erfasst und nicht von einem Sensor digitalisiert werden, treten keine Verzerrungen auf und es muss lediglich eine sinnvolle Kameramatrix (siehe Unterabschnitt 2.1.1) für den Perspective-n-Point Algorithmus bestimmt werden.

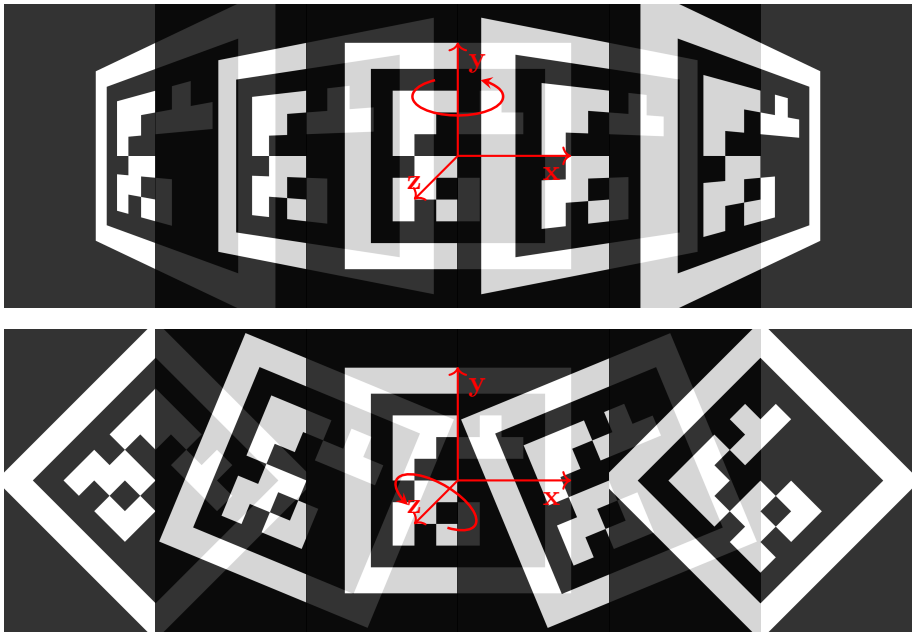


Abbildung 4.10: Veranschaulichung der zwei Simulationen mit generiertem AprilTag.

Der Skew-Koeffizient γ ist aufgrund der Verzerrungsfreiheit ebenfalls null und die restlichen vier Parameter errechnen sich folgendermaßen:

$$c_x = \frac{Pixelspalten}{2} \tag{4.4}$$

$$c_y = \frac{Pixelzeilen}{2} \tag{4.5}$$

$$f_{x/y} = \frac{\sqrt{Pixelspalten^2 + Pixelzeilen^2}}{2 * \tan\frac{FOV}{2}} \tag{4.6}$$

Versuchsdefinition und -aufbau

Für alle Simulationen ist der Field of View (FOV) konstant auf 42°. Das Bild des generierten AprilTags wird, wie in Abbildung 4.10 zu sehen, um 150° von $-75°$ auf 75° um die y-Achse und anschließend auf gleiche Weise um die z-Achse gedreht. Die Schrittgröße ist 1.5° und in jedem Schritt wird das transformierte Bild an den erweiterten Kantenerkennungsalgorithmus, beschrieben in Unterabschnitt 2.2.4, übergeben. Die geschätzten Pixelkoordinaten der Eckpunkte werden dann mit den bekannten exakten Pixelkoordinaten vom `TagGenerator` verglichen.

Dabei muss beachtet werden, dass AprilTag3, sowie Bibliotheken wie OpenCV, den Ursprung eines Pixels im Zentrum definieren, der `TagGenerator` und die erweiterte Kantenerkennungs-

methode dieser Arbeit den Ursprung jedoch in der oberen linken Ecke setzen. Das obere linke Pixel erstreckt sich also in Gleitkommazahlen von $(-0.5, -0.5)$ bis $(0.5, 0.5)$ und demnach das gesamte Bild von $(-0.5, -0.5)$ bis $(Pixelspalten - 0.5, Pixelzeilen - 0.5)$. Von den von AprilTag3 zurückgelieferten Eckpunkten muss also lediglich ein Wert von 0.5 abgezogen werden, um diese systematische Abweichung zu beheben. Die Einpflegung der Definition des Subpixelkoordinatensystems in die Dokumentation von OpenCV ist zum Zeitpunkt der Bearbeitung noch nicht erfolgt [36].

Auswertung und Diskussion

Die so gewonnenen Pixelkoordinaten der Eckpunkte werden jetzt noch mittels der Perspective-n-Point-Methode aus Unterabschnitt 2.1.2 reprojiziert und die Abweichungen verglichen. Das Ergebnis der ersten Simulation ist in Abbildung 4.11 dargestellt, mit der Messreihe AprilTag3 in rot, der erweiterten Eckpunkterfassung (Std. Erweiterung) in grün und der in Abschnitt 3.3 vorgestellten Filter-Methode in blau.

Wie in Unterabschnitt 2.2.4 beschrieben, versucht die erweiterte Eckpunkterfassung die Kanten des Markers an die der S-Funktion 2.14 zu fitten. Die Kanten eines generierten Markers ähneln jedoch eher der Kantenfunktion 2.10, dargestellt in Abbildung 2.3.

In den Plots 4.13 wird zum einen aufgeschlüsselt, welche Methode in welchem Schritt den geringsten Positionsfehler erzielt und zum anderen in welchem Schritt diese mit der Methode, die den geringsten Reprojektionsfehler erzielt, übereinstimmt. In den Schritten in denen dies nicht der Fall ist, wird der Hintergrund rot schraffiert gekennzeichnet. Der Grund dafür, dass geringster Positionsfehler nicht gleich geringster Reprojektionsfehler bedeutet, liegt vermutlich daran, dass die Positionsfehler der einzelnen Methoden sehr nahe beieinander liegen und der Perspective-n-Point Algorithmus auch nur eine Schätzung der Kamerapose liefert. Die Abbruchbedingungen für den iterativen PnP-Algorithmus von OpenCV sind fest im Code initialisiert. Unterschreiten die Reprojektionsfehler aufeinanderfolgender Iterationen ein Epsilon von 1.19×10^{-7} px oder wird die maximale Anzahl von 20 Iterationen überschritten, wird die letzte Posenschätzung zurückgeliefert. Zudem wird die vorige Posenschätzung einer jeden Methode des vorherigen Bildes als Anfangswert der extrinsischen Parameter verwendet, wodurch die Güte der vorigen Schätzung ebenfalls einen Einfluss hat.

Abbildung 4.13 **(a)** zeigt, dass in vielen Schritten ein vorheriges Bearbeiten der ROIs mit einem GaussianBlur-Filter zu besseren Ergebnissen geführt hat. Abbildung 4.13 **(b)** zeigt zwar, dass im Fall der Rotation um die z-Achse der Bilateral-Filter überwiegt, jedoch hat die Optimierung der lateralen Rotationen Priorität, da eine perfekte Draufsicht in der Realität einen seltenen Ausnahmefall darstellt und, aufgrund von Mehrdeutigkeiten beim Schätzen der Pose aus dieser Perspektive, ohnehin vermieden wird.

Aus diesem Grund wird eine weitere Simulation gestartet, bei der das generierte Bild bereits zu Beginn mit einem GaussianBlur-Filter der Größe 3×3 bearbeitet wird. Das Ergebnis dieser Simulation ist in Abbildung 4.12 zu sehen und zeigt eine deutliche Verbesserung. Auch gut zu erkennen ist, dass die Erweiterung und die Filter-Methode jetzt ähnlich gute Ergebnisse erzie-

Methode	3×3 -Filter	$\sigma_r[10^{-3}]$	$\mu_r[10^{-3}]$	$\sigma_p[10^{-3}]$	$\mu_p[10^{-3}]$
AprilTag3	-	34.7419	46.7619	76.0603	202.9303
	+	34.3947	41.3505	61.6882	175.3330
Std. Erweiterung	-	28.3324	25.4643	95.2865	130.9649
	+	12.8442	8.0157	50.2945	40.9599
Filter-Methode	-	9.0196	3.5188	73.4213	56.7802
	+	7.9674	3.3684	28.7507	25.1204

Rotation um die y-Achse der Simulation.

Methode	3×3 -Filter	$\sigma_r[10^{-3}]$	$\mu_r[10^{-3}]$	$\sigma_p[10^{-3}]$	$\mu_p[10^{-3}]$
AprilTag3	-	14.8670	19.8485	43.8077	84.7448
	+	8.7379	11.7599	37.1436	53.6931
Std. Erweiterung	-	0.5770	0.7334	7.8075	6.3597
	+	0.2610	0.3851	1.3879	3.9553
Filter-Methode	-	0.1265	0.1511	1.4051	2.5931
	+	0.1545	0.1760	1.0208	2.9557

Rotation um die z-Achse der Simulation.

Tabelle 4.1: Standardabweichung und Mittelwert der Reprojektions- und Positionsfehler von AprilTag3, der erweiterten Eckpunkterkennung ohne Vorverarbeitung (Std. Erweiterung) und mit Vorverarbeitung durch selektierte Unschärfefilter (Filter-Methode).

len und bei dem Positionsfehler in z-Achse nahezu identisch sind. Das Weichzeichnen mit dem 3×3 Kernel verbessert das Ergebnis merklich, jedoch erzielen andere Filter weiterhin bessere Ergebnisse, was Abbildung 4.14 zeigt. Die Filter-Methode setzt jede ROI einer Vielzahl unterschiedlicher Unschärfeeffekten aus, bewertet die erfassten Punkte mittels Reprojektion und übernimmt schließlich die mit dem geringsten Fehler. Alle Versuche den passenden Filter vorab zu bestimmen, z.B. durch Schätzung der Unschärfe oder Zuhilfenahme des Kontrastes, sind bisher gescheitert. Somit besteht die Möglichkeit für weitere Untersuchungen, wenn die Laufzeitperformanz weiter in den Vordergrund rückt. Je nach Positionierung im Bild kann nicht nur jedem Bild ein passender Filter zugewiesen werden, sondern jeder Kante beziehungsweise Maske, denn erhöhte Unschärfe am Rand des Bildes oder Schattenwurf auf den Marker machen jede Kante eines jeden Markers einzigartig.

In Tabelle 4.1 werden die Mittelwerte und Standardabweichungen der einzelnen Simulationen aufgeschlüsselt.

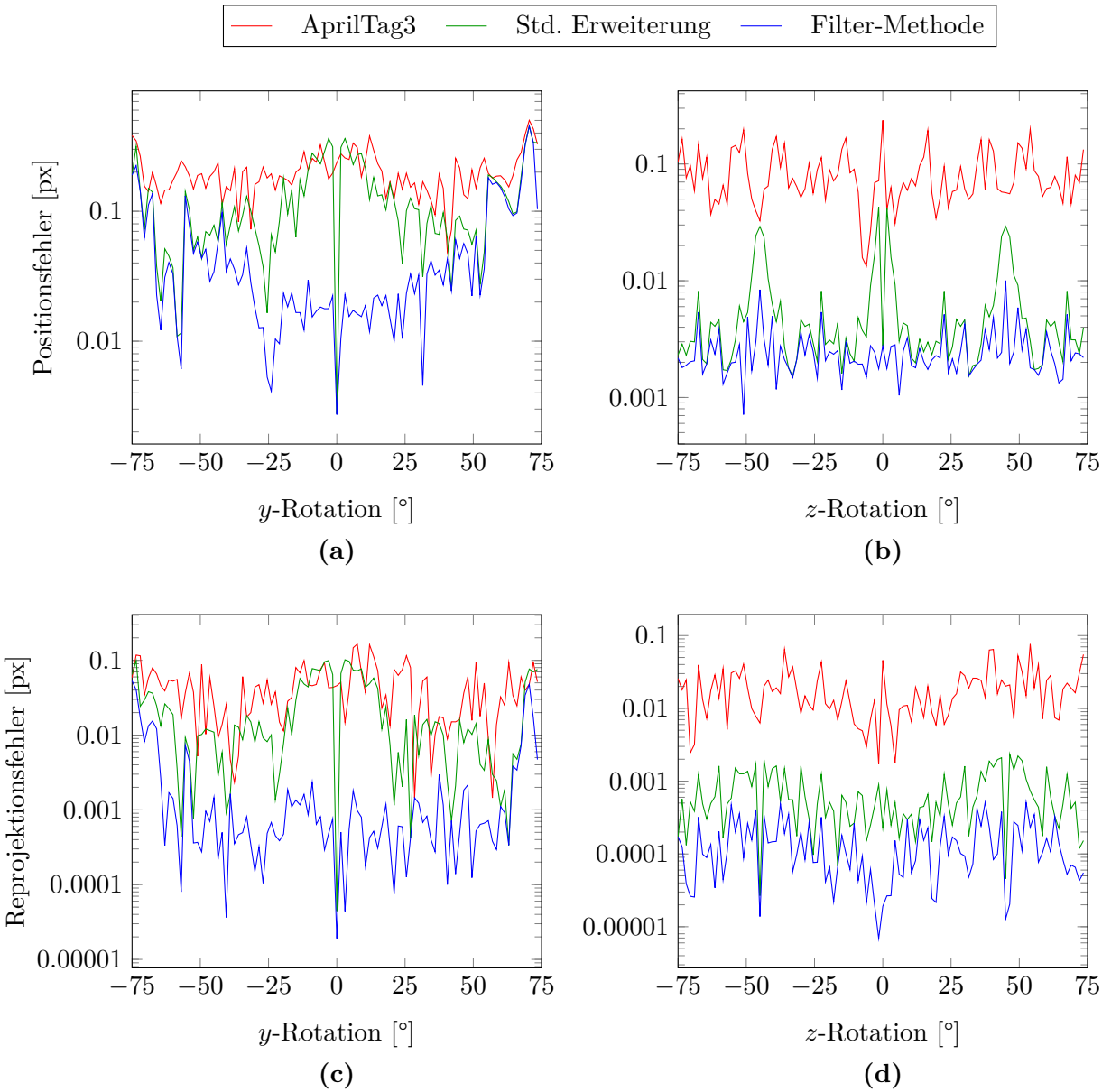


Abbildung 4.11: Simulation Reprojektions- und Positionsfehler von AprilTag3, der erweiterten Eckpunkterkennung ohne Vorverarbeitung (Std. Erweiterung) und mit Vorverarbeitung durch selektierte Unschärfefilter (Filter-Methode).

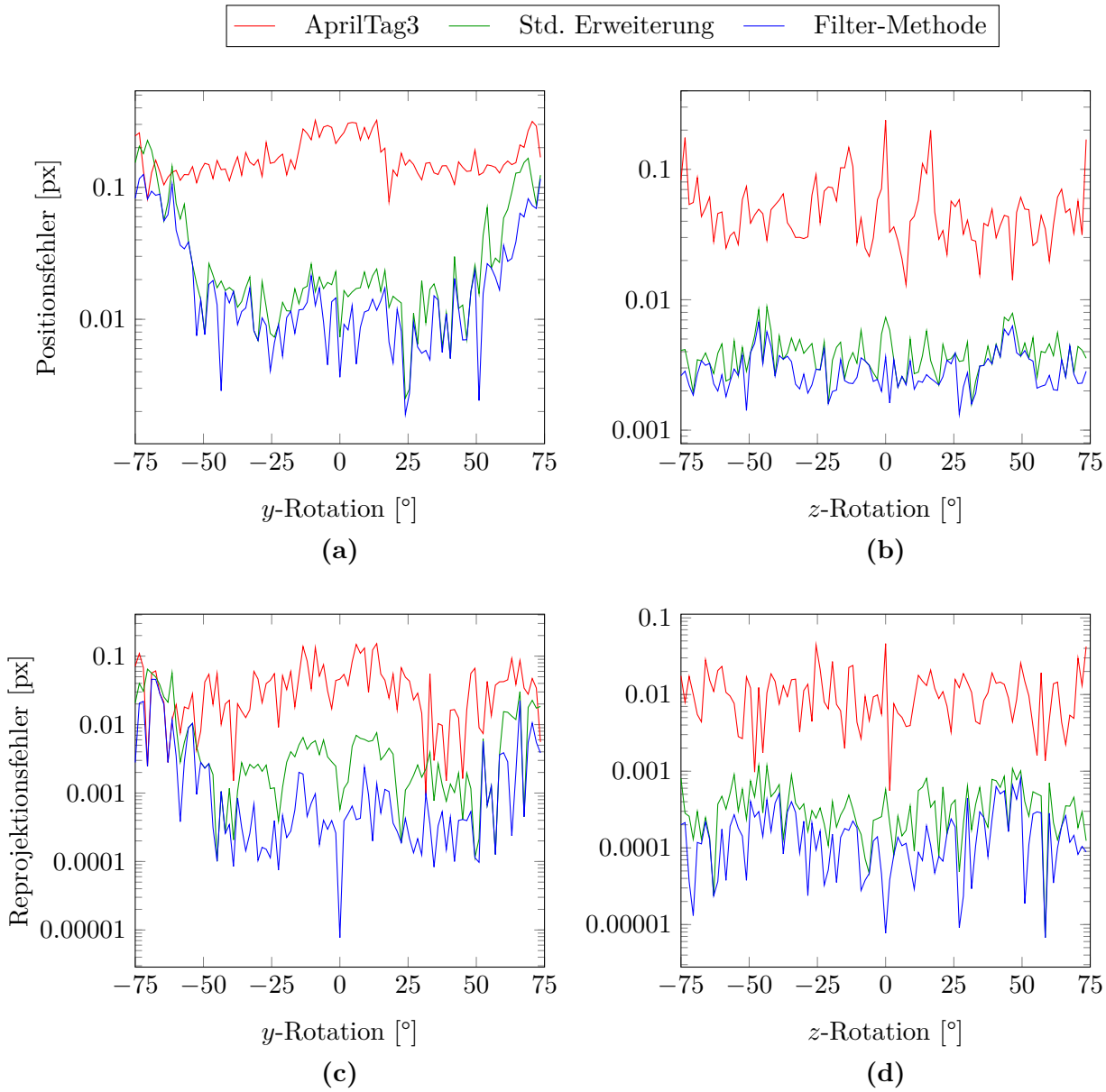


Abbildung 4.12: Simulation Reprojektions- und Positionsfehler von AprilTag3, der erweiterten Eckpunkterkennung ohne Vorverarbeitung (Std. Erweiterung) und mit Vorverarbeitung durch selektierte Unschärfefilter (Filter-Methode). Das Originalbild wurde zuvor mit OpenCV's 3×3 GaussianBlur bearbeitet.

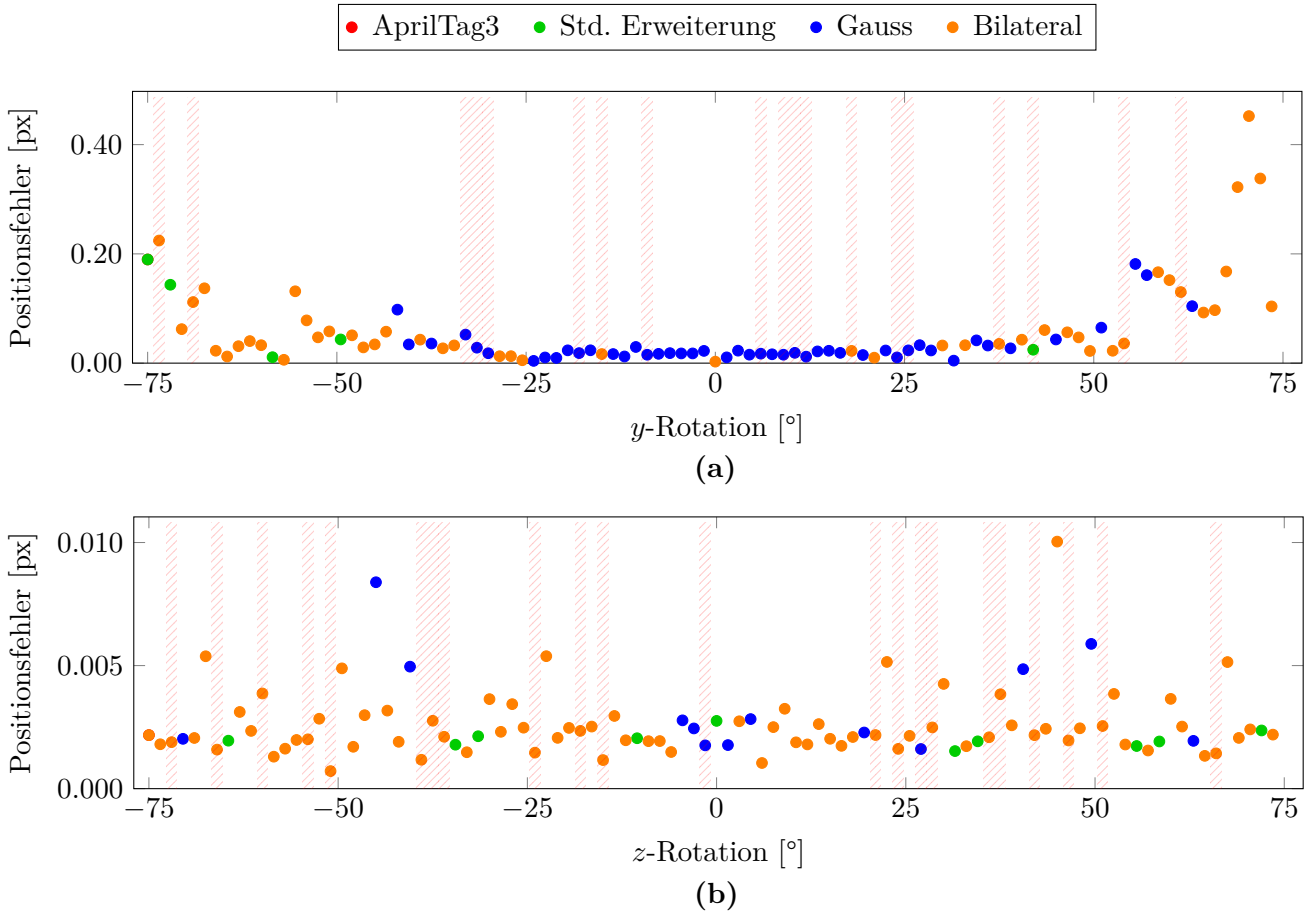


Abbildung 4.13: Erweiterte Illustration der Methoden, die das beste Ergebnis erzielt haben, bezüglich der Messungen 4.11 **(a)** und **(b)**. Zudem werden durch die schraffierten Flächen die Messpunkte hervorgehoben, bei denen die beste Positionierung und Reprojektion von unterschiedlichen Methoden ermittelt wird.

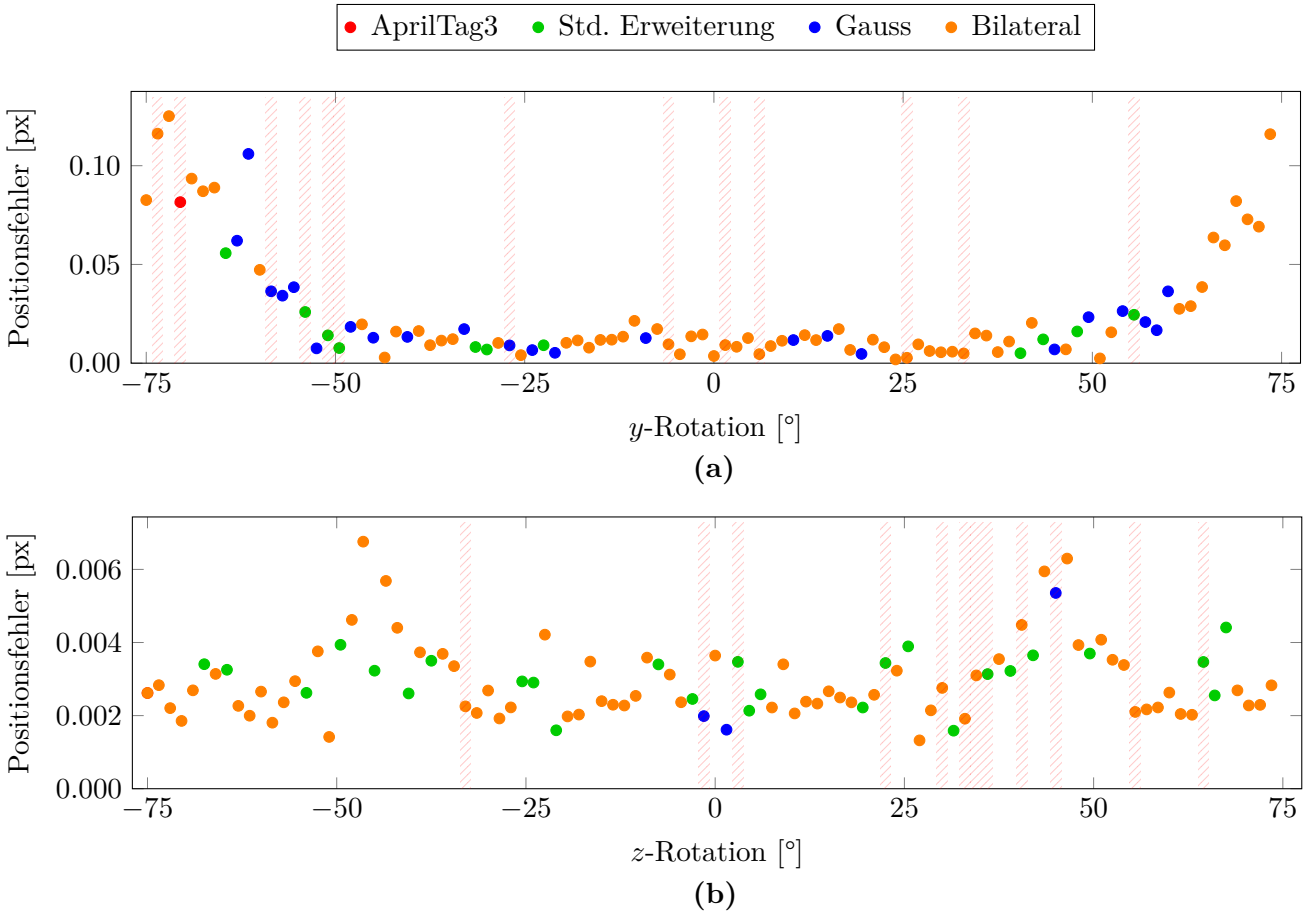


Abbildung 4.14: Erweiterte Illustration der Methoden, die das beste Ergebnis erzielt haben (bezüglich der Messungen 4.12 **(a)** und **(b)**), bei vorigem Weichzeichnen mittels 3×3 GaussianBlur-Filter. Zudem werden durch die schraffierten Flächen die Messpunkte hervorgehoben, bei denen die beste Positionierung und Reprojektion von unterschiedlichen Methoden ermittelt wird.

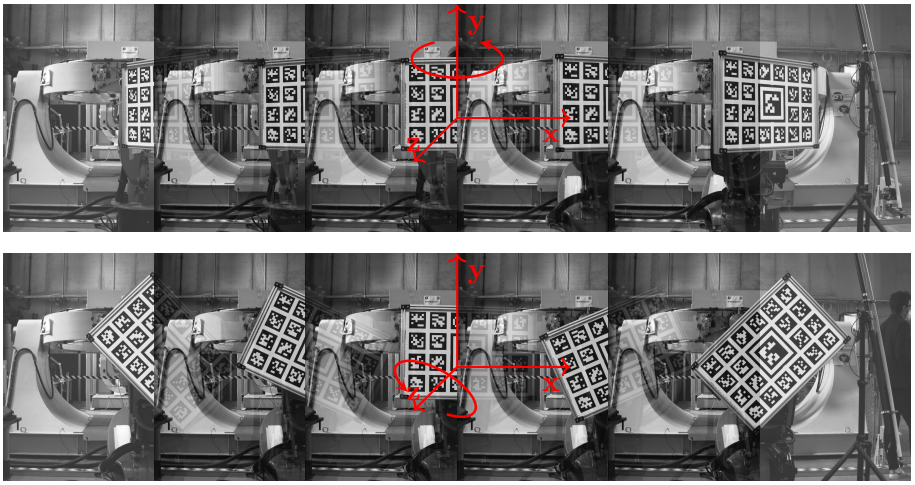


Abbildung 4.15: Aufbau des Kuka-Experiments und Illustration des Bewegungsablaufs der Rotation um die y- und z-Achse.

4.3.2 Kuka-Experiment

Nach den vielversprechenden Ergebnissen der Simulation wird jetzt versucht, den Versuchsablauf der Simulation so gut wie möglich mit realen Bildern nachzustellen.

Versuchsdefinition und -aufbau

Hierfür wird ein auf DIN A4 ausgedrucktes AprilTag mit Seitenlänge 0.1 m exakt auf die mittleren vier Tags des AprilTag-Patterns geklebt (siehe Abbildung 4.15). Dabei ist darauf zu achten, dass das neue Tag eine von den anderen Tags verschiedene ID besitzt, damit nur das eine Tag detektiert wird und dass das Blatt so glatt wie möglich aufgebracht wird, um Verzerrungen zu vermeiden. Eine Abfolge von Bildern durchzuführen und dabei sicherzustellen, dass das Pattern im Raum verharrt und sich nur um eine Achse dreht, ist von Hand nahezu unmöglich. Aus diesem Grund wird ein vom ZfT bereitgestellter Kuka-Roboterarm verwendet. Das Pattern wird so an den Endeffektor des Kuka-Roboters angebracht, dass die Rotation in y- und z-Achse, wie in den Bildern 4.15, durch den Mittelpunkt des zentralen Tags verlaufen. Die Aufnahmen werden mit der Kamera 4.2.2 und dem Objektiv 4.2.3 aus einer Entfernung von ungefähr 2.5 m gemacht. Der Roboterarm fährt hierfür zunächst in die Anfangsstellung von $-75°$ der y- bzw. z-Achse, wobei die jeweils restlichen zwei Achsenstellungen fixiert werden. Dann wird abwechselnd ein Bild der Szene gemacht und eine neue Stellung in positiver Drehrichtung der derzeitigen freien Achse eingenommen. Anders als bei der Simulation wird nur alle 3° eine Aufnahme gemacht.

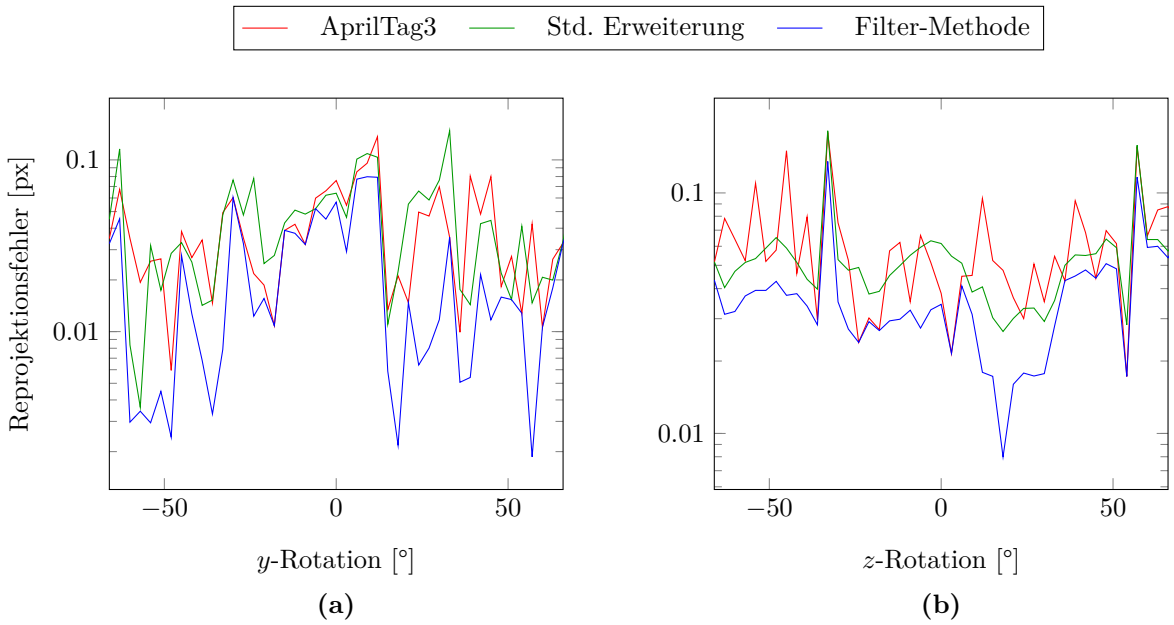


Abbildung 4.16: Kuka-Experiment Reprojektions- und Positionsfehler von AprilTag3, der erweiterten Eckpunkterkennung ohne Vorverarbeitung (Std. Erweiterung) und mit Vorverarbeitung durch selektierte Unschärfefilter (Filter-Methode).

Auswertung und Diskussion

Da keine exakten Pixelkoordinaten für die Eckpunkte existieren, ist es nicht möglich, die Positionsabweichungen der Eckpunkterfassung und der erweiterten Kantenerfassungsmethoden zu überprüfen, wodurch zumindest die systematische Abweichung durch die Definition des Subpixelkoordinatensystems von AprilTag3 und OpenCV keine Rolle mehr spielt. Ein Vergleich der Methoden durch Reprojektion mittels des Perspective-n-Point Algorithmus ist jedoch möglich und die Ergebnisse sind in Abbildung 4.16 dargestellt.

Nach dem vielversprechenden Ergebnis der Simulation fällt das Kuka-Experiment ernüchternd aus. Obwohl die erweiterte Eckpunkterkennung mit der Filter-Methode noch immer überwiegend bessere Ergebnisse erzielt als AprilTag3 (siehe Abbildung 4.17), hat sich der Genauigkeitszuwachs jedoch enorm reduziert. Die genauen Werte für den Mittelwert und die Standardabweichung der einzelnen Methoden kann der Tabelle 4.2 entnommen werden. So verbessert sich durch die Filter-Methode der Mittelwert um ca. 20 % bei der y-Rotation und ca. 45 % bei der z-Rotation, wodurch die Verteilung der anzunehmenden Reprojektionsfehler in y und z zumindest etwas angeglichener ist und es mit der über 30 % geringeren Standardabweichung zu weniger starken Ausreißern kommt.

Methode	Achse	$\sigma_r[10^{-2}]$	$\mu_r[10^{-2}]$
AprilTag3	y	2.5950	4.1752
	z	3.2529	6.2448
Std. Erweiterung	y	3.1498	4.6555
	z	2.6678	5.5096
Filter-Methode	y	2.1715	2.4423
	z	2.2467	3.9345

Tabelle 4.2: Standardabweichung und Mittelwert der Reprojektionsfehler bei Rotation um die y- und z-Achse des Kuka-Experiments.

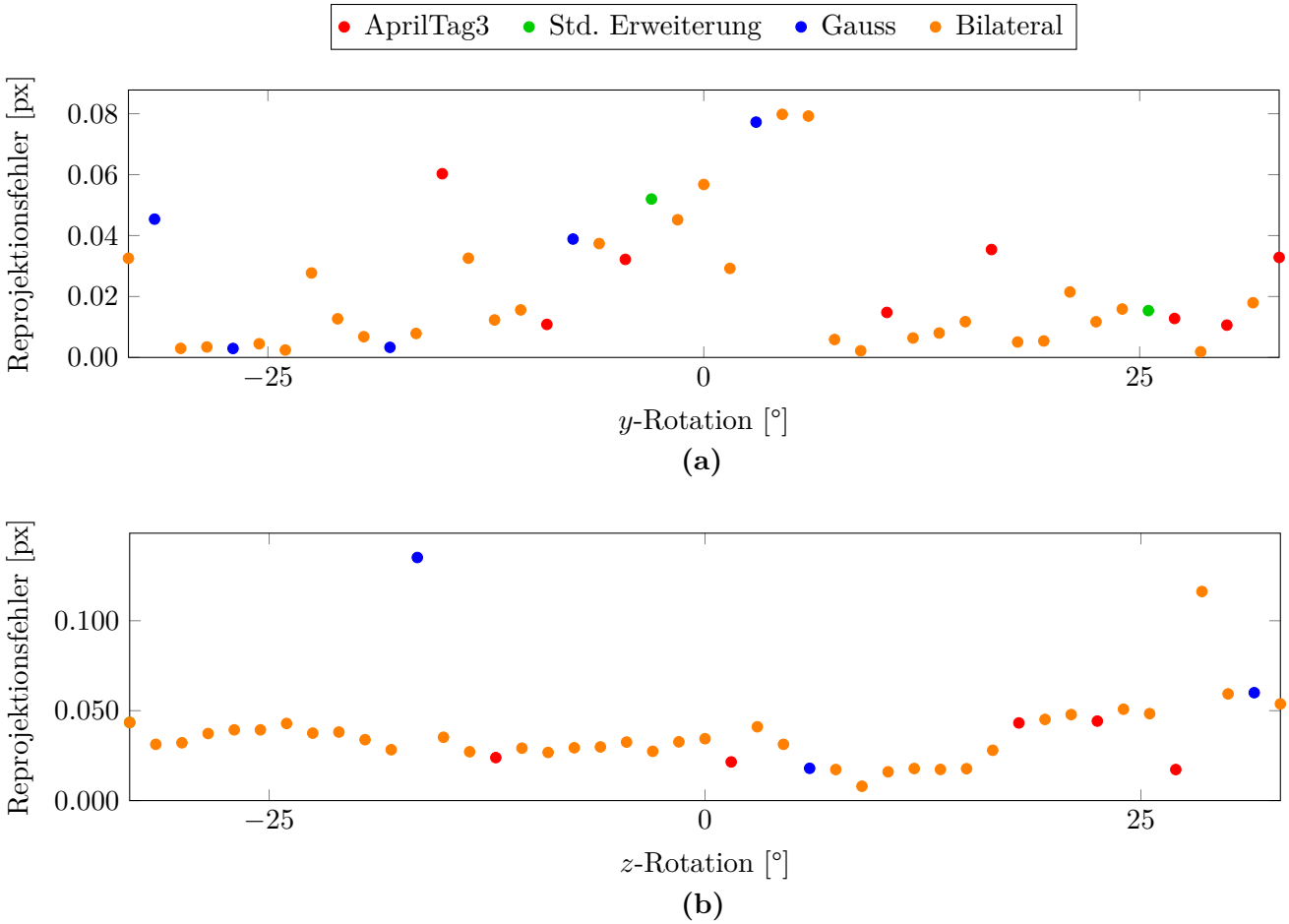


Abbildung 4.17: Erweiterte Illustration der Methoden, die das beste Ergebnis erzielt haben, bezüglich der Messungen 4.16 **(a)** und **(b)**.

4.3.3 Visuelle Positionsbestimmung

In diesem Unterabschnitt wird die Detektierungsgenauigkeit des vorherigen Tests nochmals in einer praktischen Anwendung getestet. Statt nur die Fähigkeit der Reprojektion der detektierten Markerecken zu testen, wird nun überprüft, ob sich mittels der korrespondierenden 2D-3D Punktpaare auch die korrekte Pose der Kamera schätzen lässt, also ob der PnP Algorithmus aus Unterabschnitt 2.1.2 den korrekten Positions- und Rotationsvektor liefert und zwar in Abhängigkeit von Orientierung und Distanz der realen Kamera zum Pattern.

Zu Beginn wird kurz erläutert, mit welchem Einfluss auf die Genauigkeit mit zunehmender Distanz zu rechnen ist. Zum Schätzen des Abstands zu einem Objekt werden die Abmessungen des Objekts beziehungsweise der detektierten Punkte benötigt. Zusammen mit den von der Detektierung gelieferten Abmessungen auf der Bildfläche in Pixelkoordinaten und der bekannten Brennweite des Objektivs, lässt sich dann durch Anwendung der Strahlensätze die folgende Beziehung zwischen dem 3D Welt- und dem 2D Bildkoordinatensystem formulieren:

$$Z = \frac{fT_x}{d_x} \tag{4.7}$$

mit der Brennweite f in Pixel, der sich auf die horizontalen Abmessungen beziehenden realen Breite T_x des Objekts in Meter und der Disparität, also der Pixelbreite, d_x. Die Gleichung 4.7 kann jedoch auch für vertikale Abmessungen formuliert werden.

Wie im Unterabschnitt 4.2.3 des Objektivs erwähnt, beträgt die Brennweite 9.78 mm. Geteilt durch die Pixelgröße 3.45 µm der Kamera aus Unterabschnitt 4.2.2 ergibt dies eine Brennweite von 2834.78 px. Für T_x und d_x wird sich auf die lange Seite des AprilTag-Patterns bezogen, dessen kleinstmögliche Pixelbreite 58 ist mit einer realen Breite von 0.493 m. Die Maße ergeben sich jeweils aus einer Tag-Anzahl von sechs minus des äußeren weißen Rahmens des ersten und letzten Tags. Die minimale Pixelbreite, um das AprilTag in Abbildung 2.2 darzustellen, ist zehn, somit 58 für ein sechs Tag breites Pattern. Die reale Breite eines Tags beträgt also 8.5 cm und die des Patterns 49.3 cm.

Bei dieser Pixelgröße und der Brennweite entspricht dies jedoch einer Entfernung von ca. 24 m. In Abbildung 4.18 wird durch die blaue Linie verdeutlicht, wie weit ein Objekt in Abhängigkeit der detektierten Pixelbreite, angefangen bei der doppelten der minimalen Breite, von der Kamera entfernt ist. Mit der roten Linie wird verdeutlicht, welcher Fehler einzig durch eine leichte Fehlschätzung der Pixelbreite in der Horizontalen um 0.4 px, also eines Reprojektionsfehlers von ca. 0.2 px, verursacht wird. Das Verhältnis wird mit abnehmender Breite größer, wodurch sich der Fehler erhöht. Der tatsächliche Distanzabhängige Reprojektionsfehler variiert von ca. 0.15 px bei 5 m bis hin zu 0.4 px bei 1 m, was folgende Experimente noch zeigen werden.

Versuchsdefinition und -aufbau

Wie zuvor erwähnt wird in diesem Versuch die Abhängigkeit von Orientierung und Distanz der realen Kamera aus Abbildung 4.2 zum Pattern untersucht. Hierfür wird der Roboter in ca. 1 m Entfernung auf das Pattern ausgerichtet und dann langsam mit einer linearen Geschwindigkeit

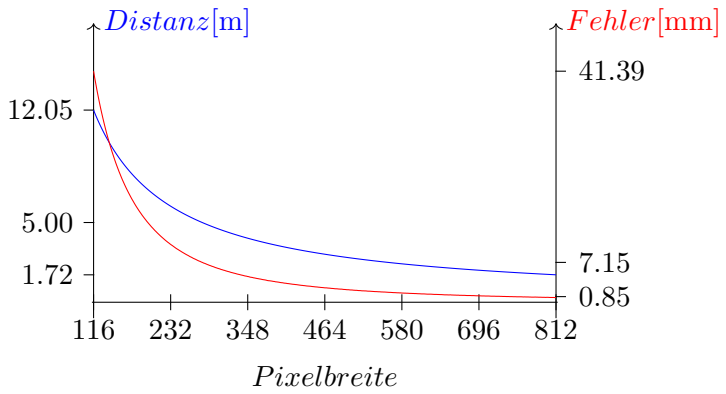


Abbildung 4.18: Darstellung der zu erwartenden Distanz in Abhängigkeit der detektierten Patternbreite auf der Bildebene in blau und der Distanzfehler bei einem konstanten Reprojektionsfehler von 0.2 px in rot. Die Distanzen links korrespondieren der Reihe nach mit den Fehlern rechts.

von ca. 0.1 m/s rückwärts gefahren, so dass die Orientierung bestmöglich konstant und das Pattern fokussiert bleibt. Insgesamt werden drei Fahrten auf diese Weise durchgeführt mit jeweils unterschiedlicher Ausrichtung des Roboters zum Pattern.

Zur Aufzeichnung der tatsächlich abgefahrenen Strecke kann auf keines der roboterinternen Systeme zurückgegriffen werden, da die Odometrie bei äußeren Einflüssen nicht in der Lage ist Abweichungen zu erfassen und AMCL aufgrund der Diskontinuität unzuverlässig ist. Zudem ist es schwer einen Bezug vom `map_frame` zum Pattern-Koordinatensystem zu ermitteln geschweige denn vom `odom_frame`. Daher wird hier das in Unterabschnitt 4.2.5 vorgestellte infrarot GPS verwendet, wodurch sich viele Vorteile ergeben. Zum einen wird dessen Sensorrahmen, wie zu sehen in Abbildung 4.8, sehr nahe an dem vermuteten Projektionspunkt der Kamera montiert, wodurch sich Betrag und Richtung von Differenzen zwischen den beiden Sensoren bereits gut abschätzen lassen. Zum anderen ist es möglich, das Koordinatensystem vom iGPS beliebig im Raum zu definieren, indem mittels des Sensors für Punktabtastungen, zu sehen in Abbildung 4.9, zuerst die x/y-Ebene durch mindestens drei Punkte und dann die y-Achse durch mindestens zwei Punkte eingemessen wird. Durch die Koordinatentransformation in Abschnitt 3.4 werden die extrinsischen Parameter der Kamera in das Weltkoordinatensystem überführt, das sich in der oberen rechten Ecke des Patterns befindet. Die Achsenorientierung mit der x-Achse nach unten, y-Achse nach links und der z-Achse in die Pattern-Ebene zeigend, wird mit dem Abtastsensor, der in Abbildung 4.9 bereits auf den Ursprung zeigt, nachgebildet, wodurch das Koordinatensystem des iGPS ebenfalls auf dem Pattern definiert ist. Dies ermöglicht eine Ad-hoc-Analyse der geschätzten Kameraposen mit den Messungen des Sensors an der Kamera. Einzig die Orientierung des Sensorrahmens bedarf einer Transformation, um sie gut mit den ermittelten Kameraposen zu vergleichen, da die Koordinatensystemdefinition des Rahmens, zu sehen in Abbildung 4.4, zu Gierbewegungen um die y-Achse führt und in zentrierter Position, wenn die y-Achse des Rahmens parallel zur y-Achse des Weltkoordinatensystems ist, 90° statt 0° liefert. Zudem sind die gemessenen Winkel links- sowie rechtsseitig der zentrierten Stellung

positiv. Das heißt, dass der Wertebereich des Winkels auf der Vorderseite des Patterns nicht [-90°, 90°] ist, sondern zwischen 0° und 90° pendelt. Von dem Gierwinkel des iGPS muss also 90° abgezogen und je nach Vorzeichen der Kameraorientierung das Vorzeichen des Sensorrahmens angepasst werden.

In der Arbeit von Depenthal [35] über das iGPS wird empfohlen die Transmitter (zu sehen im Hintergrund der Abbildung 4.8) entweder in einem Viereck um das zu vermessende Objekt zu stellen oder, wenn es sich wie in diesem Versuch um einen gerichteten Versuchsaufbau - frontseitig des Patterns - handelt, die Transmitter in einer Art C-Shape nach Abbildung 4.24 zu positionieren.

Auswertung und Diskussion

Die drei Fahrten werden in kurzer Folge hintereinander unter gleichen Bedingungen und gleicher Kamera- und iGPS-Kalibrierung durchgeführt. Abbildung 4.19 zeigt die Auswertung des ersten Tests, der im 45°-Winkel ca. einen Meter vor dem Pattern startet und dann so weit und konstant wie möglich rückwärts fährt, was an der linken Achse der grafischen Darstellung des Gierwinkels abzulesen ist. Um den Verlauf der Messungen der Kamera und des iGPS besser zu vergleichen, teilen sich beide einen Plot und beziehen sich jeweils auf die linke Achse. Die rechte Achse mit dem dazugehörigen grünen Verlauf bezieht sich auf die Differenz der Kamera- und iGPS-Daten, wobei die iGPS-Daten von den Kameradaten subtrahiert werden. So ergibt sich, dass die Gierwinkel-Differenz ziemlich konstant bei $-2°$ liegt und die z/y-Position der Kamera etwas vor dem Sensorrahmen liegt, was zu einem Distanzunterschied von ca. 4 cm führt. Der Sensorrahmen ist mit dessen Vorderkante an der vorderen Verbindungsfuge des Kameragehäuses ausgerichtet und der Sensor befindet sich direkt hinter dem Objektivadapter, wodurch sich das Zentrum des Sensorrahmens ca. 2 cm hinter dem Kamerasensor befindet. Durch die Kalibrierung ist bekannt, dass die Brennweite ca. 1 cm beträgt, wodurch mit einem Versatz von 3 cm zu rechnen ist. Zumindest zu Beginn der Strecke wird mit einer vergrößerten Wahrnehmung der AprilTags gerechnet, da sie sich nicht im Fokusbereich befinden. Wird das Tag größer detektiert als es wirklich ist, bewirkt das, dass die Kamerapose näher am Pattern geschätzt wird und somit weiter vom iGPS-Messwert. Die starke Schwankung des Gierwinkels kurz nach der Abfahrt wird auf das Umschwenken der Castor-Rollen in Abbildung 4.5 zurückgeführt. Es ist gut zu sehen, wie der über 700 kg schwere Roboter durch die massiven Lenkrollen weggedrückt wird, was in zukünftigen Experimenten durch ein Ausrichten der Rollen vor Versuchsbeginn verhindert wird. Die stärkeren Ausschläge der Gierwinkeldifferenz werden durch Reflektionen oder einer beeinträchtigten Sicht des Sensorrahmens zu den Transmittern verursacht. Die Transmitter und Sensoren so zu positionieren, dass in der gesamten Versuchsumgebung perfekte Sicht gegeben ist, gestaltet sich schwierig, da der Sensorrahmen sich teilweise selbst blockiert, wenn die Stäbe mit einem Transmitter auf einer Linie liegen.

Der zweite Versuch (siehe Abbildung 4.20) verläuft ähnlich, nur in dem anderen Quadranten vor dem Pattern, was an den positiven y-Werten zu erkennen ist. Zudem wird in einem noch spitzeren Winkel zum Pattern von ca. $-55°$ losgefahren. Ein leichtes Ausschwenken durch die Castor-Rollen findet gleich zu Beginn noch immer statt, was darauf schließen lässt, dass noch

nicht alle der drei Rollen vollständig ausgerichtet waren. Ansonsten ähneln die Ergebnisse denen des ersten Versuchs. In beiden nimmt die Differenz zwischen Kamera- und der iGPS-Position mit der Distanz ab, was auf den Fokusbereich des Objektivs zurückgeführt wird. Es ist jedoch schwer, einen eindeutigen Fehlerbereich anzugeben, da zusätzlich der Reprojektionsfehler zu geringen Fehlschätzungen der Distanz führt. Abbildung 4.22 zeigt den Verlauf des Reprojektionsfehlers des zweiten Versuchs relativ zur Distanz der Kamera zum Pattern und den maximal anzunehmenden Distanzfehler, der sich aus der Differenz der wahren iGPS-Distanz und der Gleichung 4.7 mit dem doppelten Reprojektionsfehler Δd_x als distanzabhängigen Fehlerterm der Pixelbreite $d_x = d_x + \Delta d_x$ errechnen lässt. Der erhöhte Reprojektionsfehler zu Beginn wird durch die große Pixelbreite kompensiert, wodurch ein geringerer Distanzfehler entsteht als in größeren Entfernungen mit geringerem Reprojektionsfehler, aber deutlich kleinerer Pixelbreite (vgl. Abbildung 4.18).

Der dritte Versuchspfad (siehe Abbildung 4.21) wird so gut wie möglich entlang der z-Achse des Weltkoordinatensystems geführt. Dies ist gut daran zu erkennen, dass sich die Distanz zwischen dem Zentrum des Sensorrahmens und der Kameraprojektion fast ausschließlich auf die z-Differenz beschränkt. Anders als bei den ersten beiden Versuchen wird ein deutliches divergieren der Gierwinkel registriert, was die Ergebnisse aus den Arbeiten [8] und [9] bestätigt (Hierzu im nächsten Unterabschnitt mehr). Trotz der Basis von fast einem halben Meter kommt es bei der Posenschätzung im Bereich zwischen $\pm 25°$ zu starken Unsicherheiten, besonders bei der relativen Orientierung. In der Versuchsumgebung entlang der z-Achse vor dem Pattern sind besonders starke Störungen des iGPS zu entnehmen, da zeitweise der Sensorrahmen mit zwei Transmittern auf einer Linie liegt. Zudem ist vor dem Pattern am wenigsten Platz, aufgrund dessen die Strecke bereits bei ca. 4 m stoppt. Dadurch wird jedoch der signifikante Unterschied der Kameraposenschätzung in Ruhelage in den letzten drei Sekunden der Darstellung 4.21 sichtbar. In größeren Distanzen ist dies auch der Fall, was die letzten Sekunden der unbeschnittenen grafischen Darstellungen der ersten zwei Versuche B.1 und B.2 zeigen. Ab einer Distanz von ca. 6 m treten immer häufiger Fehler in der Kameraposenschätzung auf, wodurch die Ablesegenauigkeit der Achsenskala beeinträchtigt wird. Aufgrund dessen werden diese Bereiche in den Grafiken 4.19 und 4.20 ignoriert. Interessant sind dennoch die letzten Sekunden der unbeschnittenen Darstellungen, da die Posenschätzung in Ruheposition ein ähnlich geringes Rauschen aufweist, wie in den Bereichen vor der 6 m Marke, sich der Roboter hier jedoch in gut 8 m Entfernung befindet. Eine Posenschätzung ist im statischen Zustand also auch aus größerer Entfernung möglich.

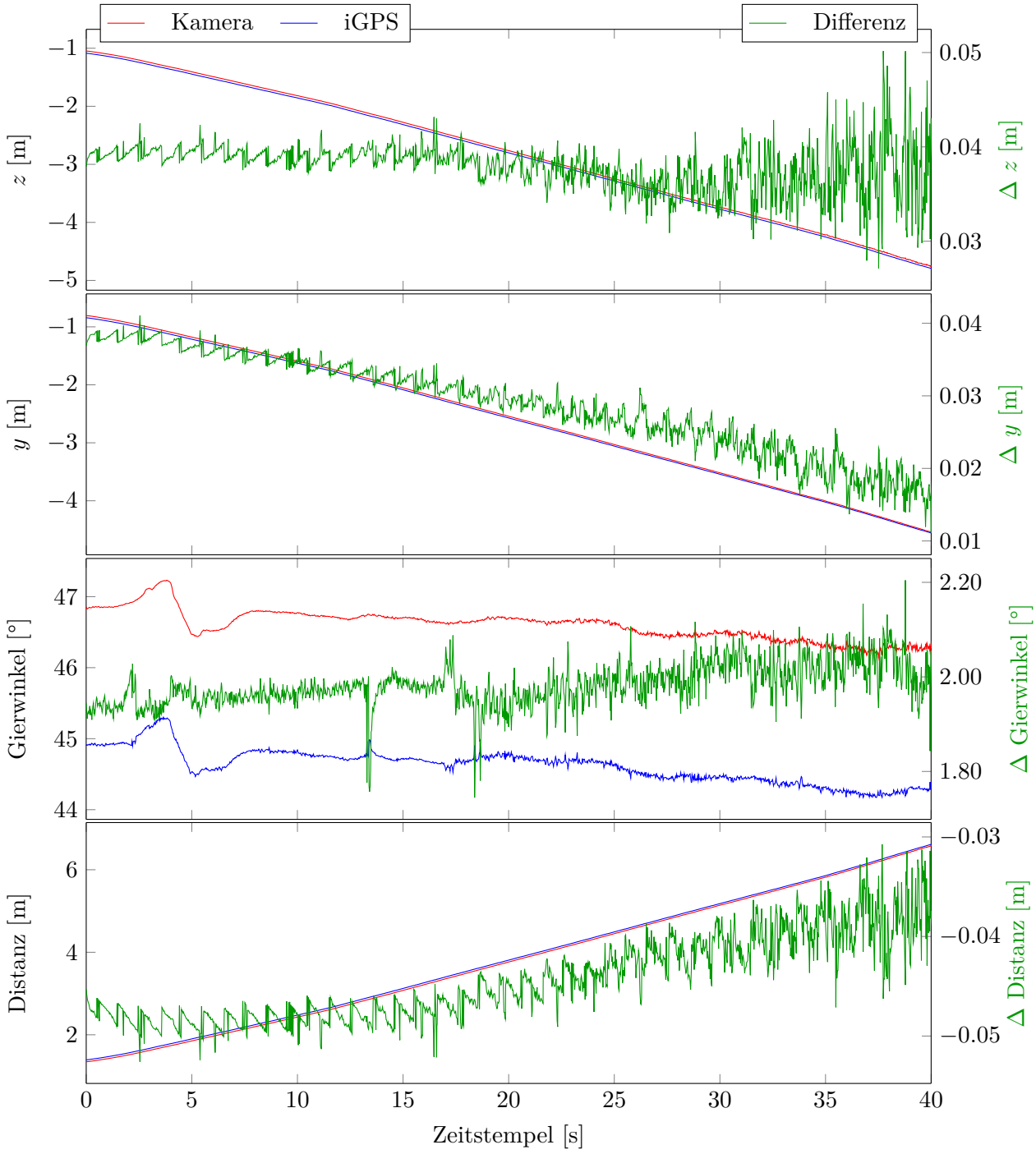


Abbildung 4.19: Erster visueller Positionsbestimmungstest mit Ausgangsposition 1.4 m vor dem Pattern im Winkel von 45°.

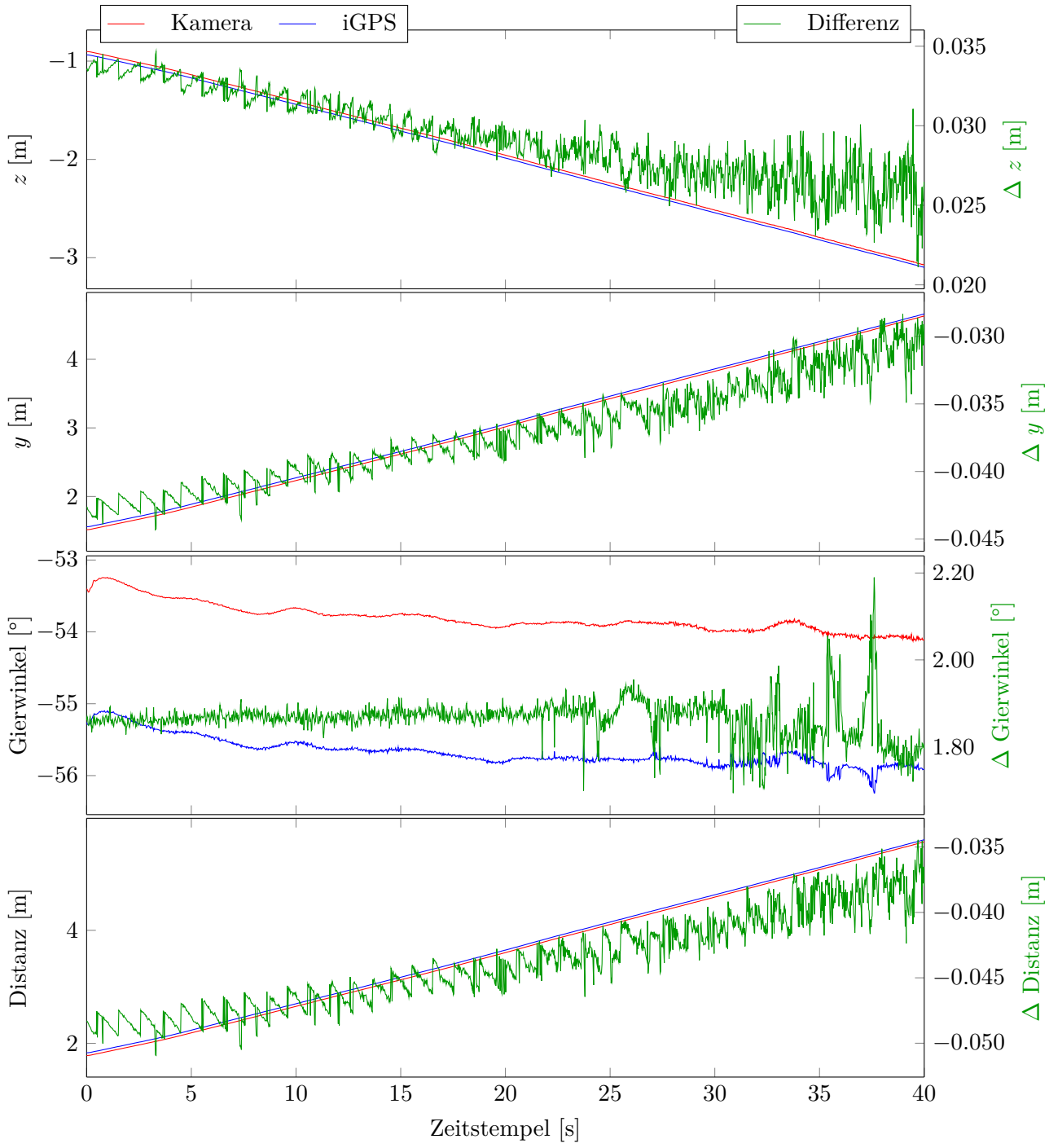


Abbildung 4.20: Zweiter visueller Positionsbestimmungstest mit Ausgangsposition 1.8 m vor dem Pattern im Winkel von −55°.

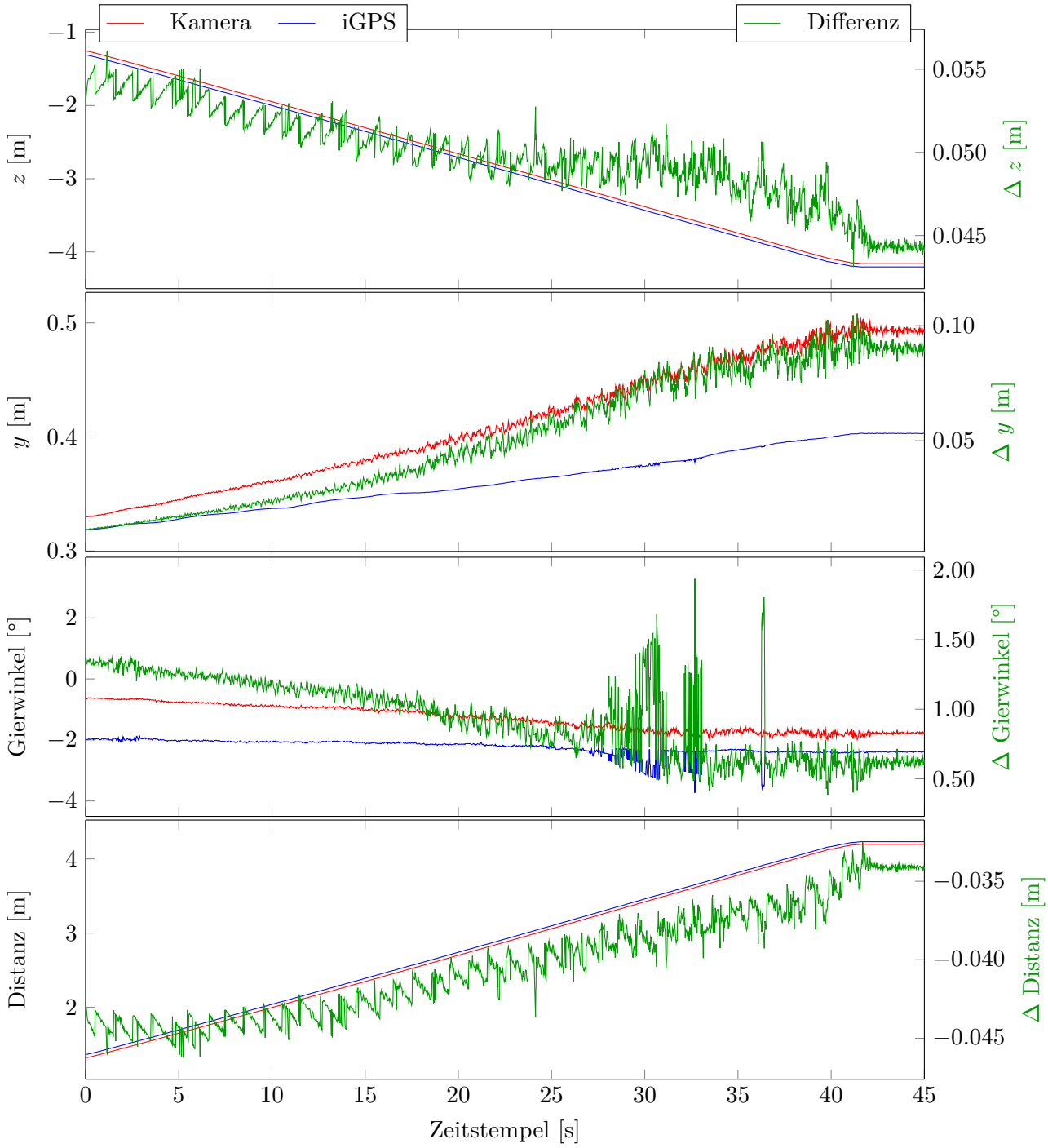


Abbildung 4.21: Dritter visueller Positionsbestimmungstest mit Ausgangsposition 1.4 m vor dem Pattern im Winkel von −2°.

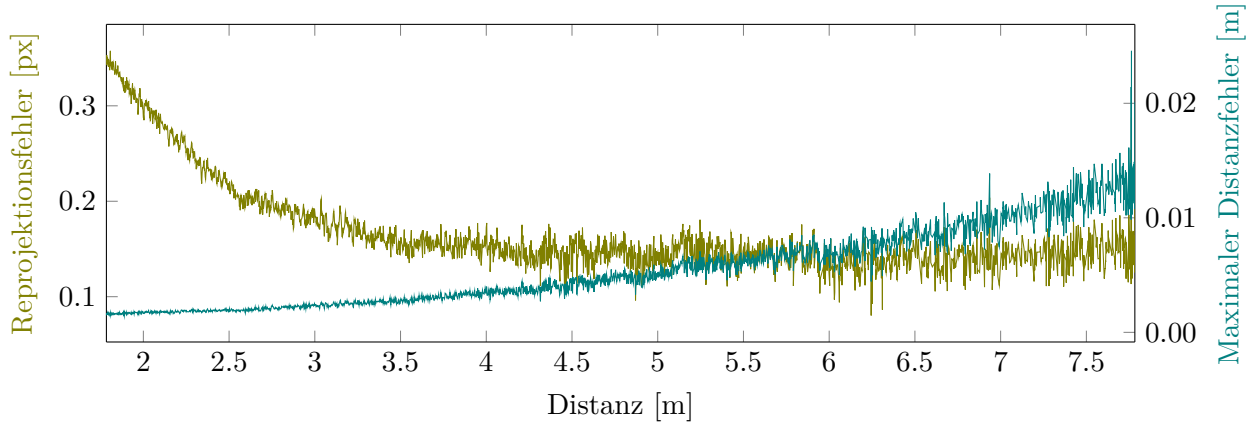


Abbildung 4.22: Reprojektionsfehler relativ zur Distanz der Kamera zum Pattern des zweiten visuellen Positionsbestimmungstests (Abbildung 4.20) und der dadurch entstehende maximal anzunehmende Distanzfehler nach Gleichung 4.7.

iGPS-Kamera-Transformation

Dass die genaue Pose der Kamera von der des Sensorrahmens in allen Versuchen in einem ähnlichen Winkel und Abstand abweicht, eröffnet die Möglichkeit die Transformation zwischen Kamera und Sensorrahmen zu berechnen, um dadurch noch bessere Aussagen über die visuellen Positionsschätzungsfähigkeiten der Implementierung zu machen. Das Verfahren zur Bestimmung dieser Transformation wird Hand-Eye-Kalibrierung genannt und in Abschnitt 2.7 beschrieben. Wie auch in dem Beispiel in Abbildung 2.13 wird folgend der Roboter durch die erfassten Posen des iGPS repräsentiert und eine Eye-in-Hand-Kalibrierung durchgeführt.

Von den nacheinander eingenommenen Posen werden letztendlich neun verwendet, die in Abbildung 4.23 als orientierte Pfeilköpfe dargestellt sind. Es ist gut zu erkennen, dass die roten Kameraposen leicht weiter nach vorne gerückt und um einen gewissen Grad in negativer Richtung verdreht sind. Dies trifft jedoch nicht auf die vier Posepaare im Zentrum zu. Hier sind alle Kameraposen in positiver Richtung relativ zur iGPS-Pose verdreht und besitzen einen deutlich größeren Abstand in positiver y-Richtung von bis zu 10 cm. Damit stimmen diese Ergebnisse gut mit denen des dritten Versuchs der grafischen Auswertung 4.21 überein.

Das Problem der Ungenauigkeiten und Mehrdeutigkeiten der Posenbestimmung bei direkter Draufsicht wird detailliert in Abawi et al. [8] untersucht und in Abbas et al. [9] zusätzlich noch der Einfluss der Position des Tags im Bild selbst, mit dem Ergebnis, dass die Posenschätzung die höchste Genauigkeit in dem Winkelbereich von 25° bis 75° und bei zentrierter Kameraausrichtung hat. Aus diesen Gründen werden die vier Posepaare im Zentrum von Abbildung 4.23 nicht mit in die Berechnung der Hand-Eye-Transformation einbezogen.

Das Ergebnis, beschränkt auf die relevanten Parameter, bezieht sich auf das Kamerakoordinatensystem aus Unterabschnitt 2.1.1 und ergibt sich zu

$$z = 0.041\,81\,\mathrm{m} \tag{4.8}$$
$$x = -0.000\,136\,2\,\mathrm{m} \tag{4.9}$$
$$\alpha = -1.8469\,^\circ \tag{4.10}$$

beziehungsweise als Transformation

$$X = \begin{bmatrix} \cos(\alpha) & 0 & \sin(\alpha) & x \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.11}$$

und ist direkt vor Überführung ins Weltkoordinatensystem (siehe Abschnitt 3.4) auf die geschätzte Kamerapose anzuwenden, um eine Trajektorienplanung bezüglich der iGPS-Pose durchzuführen.

Die 13 Posepaare sind zusätzlich in Tabelle C.1 und C.2 aufgeführt.

Die Hand-Eye-Kalibrierung ist jedoch lediglich eine Schätzung und somit ebenfalls fehlerbehaftet. Zudem ist für folgende Versuche eine relative Transformation zum Drehpunkt des Roboterfahrwerks von größerer Bedeutung. Eine solche Transformation wird auf ähnliche Weise bestimmt, wie die Hand-Eye-Kalibrierung, indem statt der Kamerapose die Odometriepose verwendet wird. Einziger Unterschied - und zugleich Fallstrick - ist, dass die einzelnen Posen der Odometrie aus einem zusammenhängenden Pfad entnommen werden, da der `base_frame` bei jedem Neustart des Roboters neu definiert wird. Die relativen Posen A_i^0 sind somit von dem Drift der Odometrie betroffen, wodurch das Finden einer präzisen statischen Transformation mittels der Hand-Eye-Kalibrierung bezüglich des iGPS unmöglich ist. Zur Veranschaulichung sind die Odometrieposen dennoch in Abbildung C.3 zusammen mit den korrespondierenden iGPS-Posen visualisiert. Alle Odometrieposen wurden händisch so transformiert, dass die Pose Nummer eins der Odometrie in Tabelle C.4 exakt mit der Pose eins des iGPS in Tabelle C.2 übereinstimmt. Dadurch ist der fortlaufende Drift der Odometrie gut zu erkennen. Um das zusätzliche Induzieren von Fehlerquellen zu verhindern und weil die Reproduzierbarkeitsgenauigkeit im Vordergrund steht, wird in den abschließenden Versuchen auf die Verwendung der hier bestimmten Transformation verzichtet. In einem betriebsfähigen System ist ein Einmessen der Kamera bezüglich des `base_frames` jedoch zu empfehlen.

Zur Überprüfung der Richtigkeit der Transformation wird sie auf die Kamerapose des ersten und zweiten Versuchs angewendet. Da die in der mittleren Region befindlichen Posen bei der Hand-Eye-Kalibrierung ausgeschlossen wurden, wird der dritte Versuch nicht weiter ausgewertet. Die grafischen Darstellungen C.5 und C.6 zeigen eine deutlich bessere Übereinstimmung mit den iGPS-Daten. Die folgenden Tabellen 4.3 und 4.4 zeigen abschließend noch die stochastische Analyse der Pose und Distanz der Versuche eins und zwei, jeweils bis zur Sekunde 40 nach Abbildung C.5 und C.6.

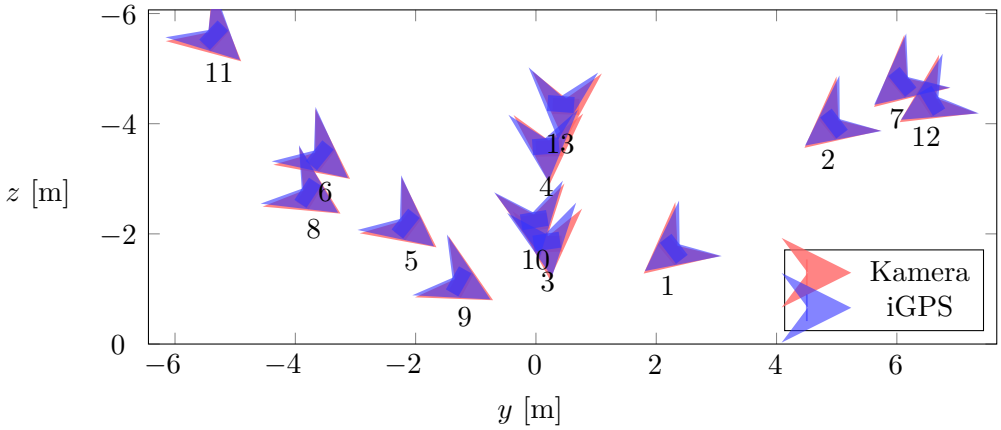


Abbildung 4.23: Die erfassten Posen für die Hand-Eye-Kalibrierung. Kamera aus Tabelle C.1 und iGPS aus Tabelle C.2.

Parameter	σ	$(\mu \pm \sigma_\mu)$
z	$2.4386 \times 10^{-3}\,\mathrm{m}$	$(8.3985 \pm 0.0602) \times 10^{-3}\,\mathrm{m}$
y	$6.4602 \times 10^{-3}\,\mathrm{m}$	$(-0.5441 \pm 0.1596) \times 10^{-3}\,\mathrm{m}$
Distanz	$4.2867 \times 10^{-3}\,\mathrm{m}$	$(-2.8831 \pm 0.1059) \times 10^{-3}\,\mathrm{m}$
ψ	$0.0442\,^\circ$	$(-0.1348 \pm 0.0011)\,^\circ$

Tabelle 4.3: Standardabweichung und Mittelwert mit Standardfehler der Differenzen zwischen den Hand-Eye-kalibrierten Kamera- und den iGPS-Daten des ersten visuellen Positionsbestimmungsversuchs nach C.5.

Parameter	σ	$(\mu \pm \sigma_\mu)$
z	$2.5399 \times 10^{-3}\,\mathrm{m}$	$(5.6094 \pm 0.0628) \times 10^{-3}\,\mathrm{m}$
y	$3.7646 \times 10^{-3}\,\mathrm{m}$	$(-2.7660 \pm 0.0931) \times 10^{-3}\,\mathrm{m}$
Distanz	$3.6152 \times 10^{-3}\,\mathrm{m}$	$(-2.2556 \pm 0.0894) \times 10^{-3}\,\mathrm{m}$
ψ	$0.0433\,^\circ$	$(-0.0183 \pm 0.0011)\,^\circ$

Tabelle 4.4: Standardabweichung und Mittelwert mit Standardfehler der Differenzen zwischen den Hand-Eye-kalibrierten Kamera- und den iGPS-Daten des zweiten visuellen Positionsbestimmungsversuchs nach C.6.

Die Werte in den Tabellen 4.3 und 4.4 beziehen sich bei dem ersten Versuch auf den Distanzbereich von 1.39 m bis 6.62 m und beim zweiten Versuch von 1.82 m bis 5.61 m jeweils bei stetiger Fahrt von ca. 0.1 m/s. Im statischen Zustand und bei unbeeinträchtigtem iGPS-Empfang sind noch deutlich bessere Werte zu erwarten.

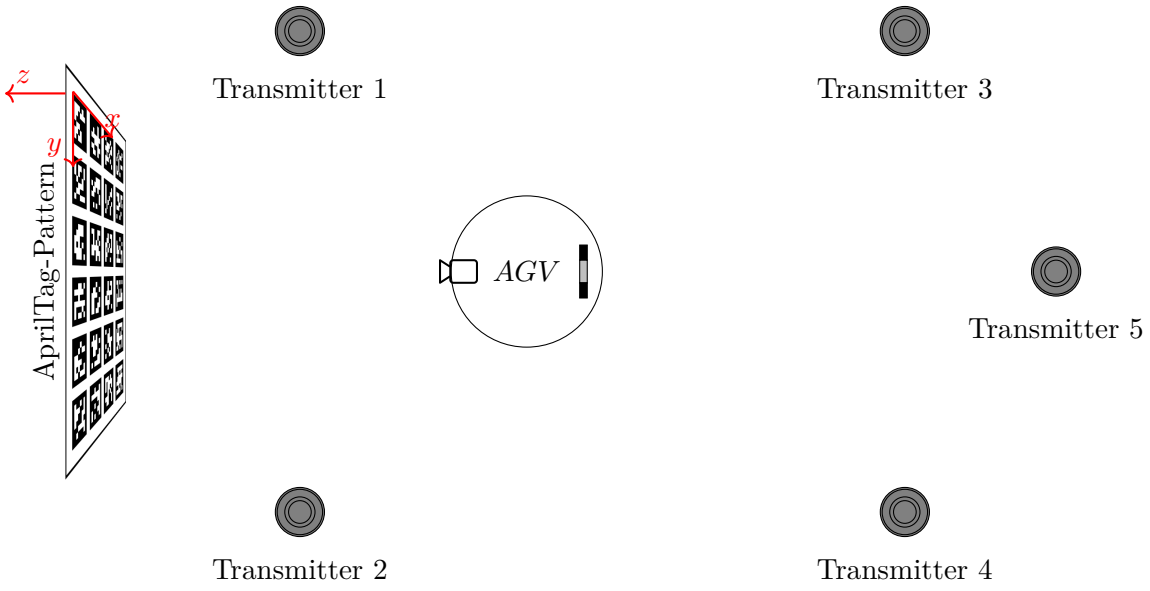


Abbildung 4.24: Illustration des Versuchsaufbaus der visuellen Positionsbestimmung und Andockmanöver

4.4 Trajektorienplanung und Andockmanöver

Zunächst wird das Modul **PathPlanning** der **ControlStation** in einer Simulation getestet, um etwaige Fehler, systematische Abweichungen und Planungseinschränkungen zu ermitteln. Gefolgt von umfangreichen Tests des **Controller**-Moduls, um die optimalen - in Unterabschnitt 3.5.2 behandelten - Parameter für die Dämpfung, Rate und Geschwindigkeit zu finden, die den in Unterabschnitt 3.5.1 behandelten Drift der Odometrie so weit wie möglich reduzieren. Im Anschluss erfolgen die abschließenden Tests des gesamten Systems anhand von Andockmanövern.

4.4.1 Simulation

Die implementierte Simulation besitzt zwei Modi zum Testen der allgemeinen Funktionsfähigkeit, Genauigkeit und der Einschränkungen, die bereits in Unterabschnitt 3.5.3 vorgestellt wurden. Das Ergebnis, welche Einschränkungen es gibt und welche Anfahrposen zu brauchbaren Trajektorien mit stetiger Krümmung führen, wird in Abbildung 3.10 dargestellt. Dabei wird der Planungsalgorithmus A.1 aus diversen Startpunkten eines zuvor definierten Areals gestartet und in einer Art Heatmap protokolliert, ob die Annäherung an eine Trajektorie mit stetiger Krümmung erfolgreich war. Die Wahl des Areals, der Schrittgröße und der Zielpose ist beliebig, wobei die Orientierung der Startposen immer in Richtung Zielpose gerichtet ist. Das Wissen über die notwendige Distanz zum Ziel in Abhängigkeit des Anfahrstellungswinkels gibt Aufschluss darauf, wie das verwendete Objektiv eingestellt werden muss, um eine scharfe und ausreichend große Aufnahme der Marker zu erhalten. Dies beeinflusst direkt den möglichen FOV und damit die Genauigkeit, die beim Einnehmen der markerzentrierenden Stellung mindestens nötig ist.

Der zweite Modus nimmt eine beliebige Anzahl von Posen entgegen und berechnet einen zusammenhängenden, durch alle Posen gehenden, Pfad. Dabei steht zur Wahl, ob die Orientierung eines jeden Knotens in Richtung des Folgeknotens ausgerichtet sein soll oder ob entweder die mit den Posen übergebenen Orientierungen oder eine bestimmte Anzahl an zufällig generierten Knoten verwendet werden sollen. Wird eine Liste ohne Orientierungen übergeben oder die Zufallsgenerierung gewählt, wird die Standardmethode der Selbstausrichtung verwendet. Das Ergebnis wird dann mittels eines Tkinter[1]-basierten Grafikmoduls "Turtle" visualisiert.

Der erste Test umfasst die Planung einer Trajektorie durch neun Knoten bei automatischer Ausrichtung. Das Ergebnis wird in Abbildung 4.25 **(a)** dargestellt. Der mit Kreissegmenten angenäherte stetig gekrümmte Pfad ist in rot eingezeichnet und alle Anderen repräsentieren die gültigen Reed-Shepp-Segmente, welche als Grundlage für die Klothoidenberechnung dienen. Die Subpfade zwischen den Posen sind dabei unabhängig voneinander und somit wählt der Algorithmus A.1 auch jedes Mal einen neuen Subpfad beziehungsweise "Farbe" zum Approximieren. Das ursprüngliche Kreissegment ist in langen Kurven, z.B. bei (-3, -5), gut zu erkennen. Abbildungen 4.25 **(b)** und **(c)** verdeutlichen dies nochmals genauer durch eine Trajektorienplanung durch die Posen (0, 0, $.5\pi$) und (0, 3, $-.15\pi$), einmal mit dem Klothoidenpfad und einmal ohne. Die Reed-Shepp-Pfade in **(b)** und **(c)** haben alle drei Segmente, jedoch hat nur der grüne Pfad eine gleichbleibende Fahrtrichtung, aufgrund dessen dieser in Zeile 18 des Algorithmus A.1 nicht aussortiert wird. Durch die Approximierung des Kreissegments an eine Klothoide durch viele kürzere Kreissegmente diverser Krümmungen, erhöht sich die Anzahl der einzelnen Segmente zwischen den Posen. So erhöht sich die Anzahl der Segmente in Abbildung 4.25 **(a)** von 24 auf 84 und in **(c)** die des grünen Pfades von drei auf elf.

Zu guter Letzt liefert die Simulation noch die Abweichung der Endpose des Klothoidenpfades mit den Sollwerten der übergebenen Knoten. Diese werden zum einen verwendet, um den Erfolg der Approximierung festzustellen und zum anderen, um die Werte zur Evaluierung heranzuziehen. Bei längeren Trajektorien, wie in Abbildung 4.25 **(a)**, werden die Endposen des letzten Klothoidenpfades als Anfangspose für den nächsten verwendet, um ein eventuelles Aufaddieren von Abweichungen zu überprüfen. Die Fehler der Zielposenparameter in **(a)** betragen demnach:

$$\begin{aligned} dx &= -8.8818 \times 10^{-16} \\ dy &= 4.4408 \times 10^{-16} \\ d\theta &= 2.2204 \times 10^{-16} \end{aligned}$$

und die der Zielposenparameter in **(c)**:

$$\begin{aligned} dx &= -1.1102 \times 10^{-16} \\ dy &= -4.4409 \times 10^{-16} \\ d\theta &= 0 \end{aligned}$$

Diese Fehler sind verschwindend klein und vermutlich zu ignorieren. Da jedoch alle weiteren Prozesse von der Genauigkeit dieser Trajektorienplanung abhängen, wird die Ursache dieser Abweichungen überprüft.

[1] Tkinter ist die Standard-GUI-Bibliothek für Python.

In dem komplexen Pfad **(a)** werden die Gleitkommazahlen für x, y, und θ nicht größer als acht, somit liegen die Werte maximal im Zweierpotenzbereich zwischen 2^2 und 2^3 beziehungsweise vier und acht. Nach Norm IEEE 754 [38], der Definition von Standarddarstellungen für binäre Gleitkommazahlen in Computern, beträgt die Bitanzahl der Mantisse 52 bei dem hier verwendeten Gleitkommazahltyp Double, also ist die Genauigkeit beziehungsweise die Schrittgröße in dem Bereich

$$\frac{8-4}{2^{52}} = 8.8818 \times 10^{-16}, \tag{4.12}$$

was genau der maximalen Abweichung des Zielposenparameters dx in **(a)** entspricht. Somit liegen die Fehler im Bereich der Gleitkommapräzision von Zahlen des Typs Double, was die Korrektheit der Trajektorienplanungsalgorithmen in den getesteten Szenarien beweist.

4.4.2 Ausführung von Trajektorien

Wie in Abschnitt 3.5 dargelegt, wird die Ausführung der von der **ControlStation** vorgegebenen Trajektorie im Odometrie-Transformationsbezugssystem stattfinden, da es im Gegensatz zu AMCL kontinuierlich ist. Es wurden jedoch auch die Nachteile der Odometrie erläutert und in Unterabschnitt 3.5.2 Parameter und Methoden vorgestellt, die zu einer Verbesserung des **Controller**-Moduls im Allgemeinen führen.

In diesem Abschnitt wird nun die Ausführungsgenauigkeit mit unterschiedlichen Werten für die Dämpfung a des Reglers und der dynamischen Regelschleifenrate r_{dyn} aus Unterabschnitt 3.5.2 getestet. Die lineare Geschwindigkeit der einzelnen Segmente ist für alle nachfolgenden Versuche fest auf 0.05 m/s begrenzt, was einem Sechstel der Standardgeschwindigkeit der freien Navigation entspricht.

Versuchsdefinition und -aufbau

Wie im zuvorigen Experiment der visuellen Positionsbestimmung wird auch hier das iGPS zur Aufzeichung der tatsächlich abgefahrenen Strecke verwendet. Der Versuch wird in einer Halle mit glattem Hochleistungs-Industriebetonboden durchgeführt, auf dem die Roboterplattform HelMo 4.1 mühelos navigieren kann. Die Bewegung des Roboters beschränkt sich also nur auf die Bodenebene, aufgrund dessen der Boden mit dem Punktabtastsensor aus Abbildung 4.9 als x/y-Ebene des iGPS-Bezugssystems mit z-Achse nach oben zeigend eingemessen wird. Alle Rotationen beschränken sich dadurch auf Gierbewegungen um die z-Achse. Auf diese Weise ist zudem ein direkter Vergleich der Odometrieaufzeichnungen mit denen des iGPS möglich, da die `odom_frame`-Ausrichtung analog definiert ist. Der zur Aufzeichnung von 6DOF Posen vorgesehene Sensorrahmen wird, wie in Abbildung 4.8 dargestellt, an dem Roboter montiert.

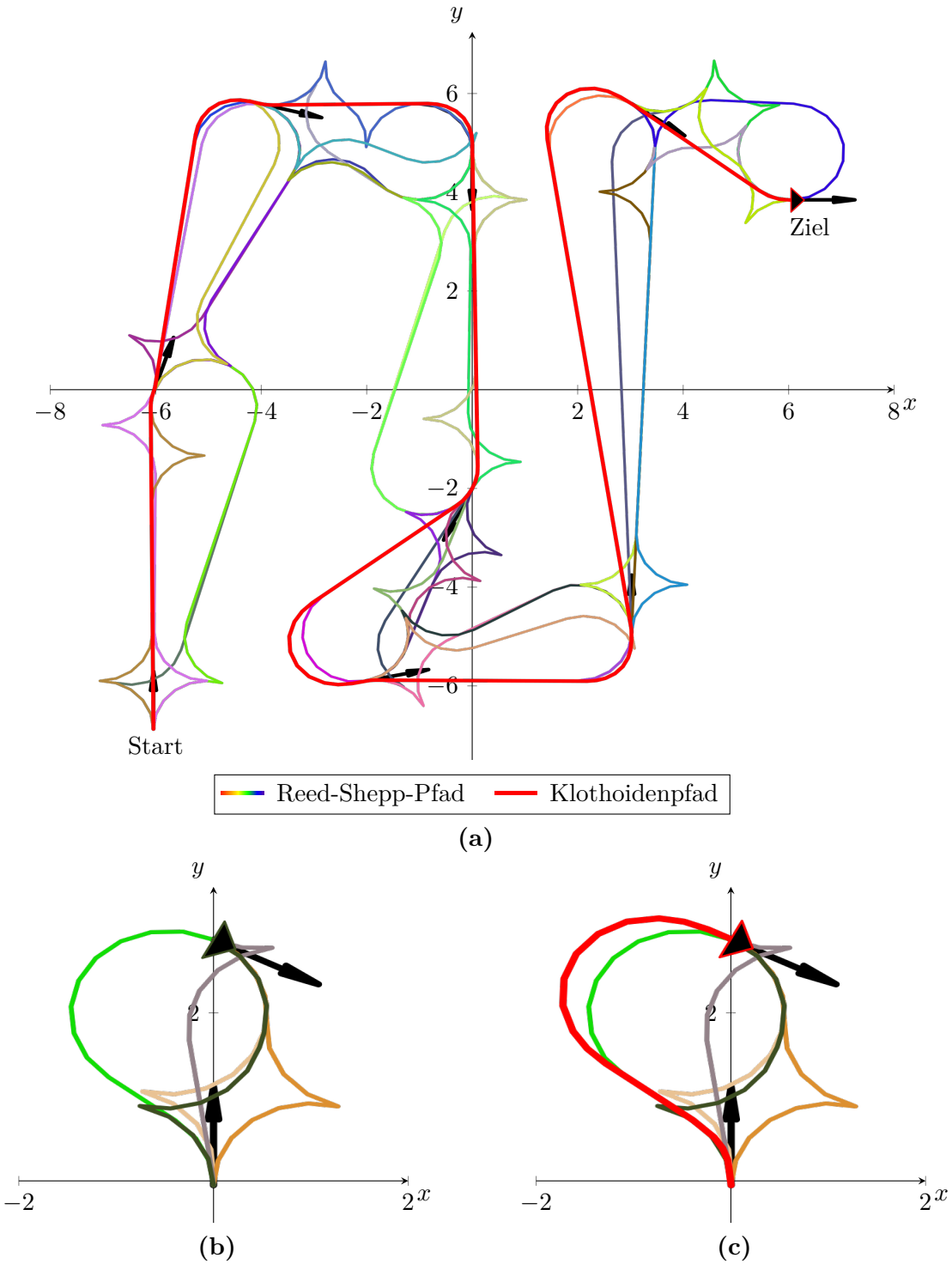


Abbildung 4.25: **(a)** visualisiert alle generierten Reed-Shepp-Pfade einer komplexen Trajektorie und in rot der angenäherte Pfad mit stetiger Krümmung. Der zur Annäherung gewählte Originalpfad besteht aus 24 Segmenten, wohingegen der rote Pfad aus 84 besteht. **(b)** und **(c)** visualisieren nochmal die Annäherung an eine Klothoide durch Kreissegmente. Der grüne Pfad in **(b)** hat drei Segmente und der neue Pfad in rot mit stetiger Krümmung in **(c)** hat elf.

Zur Erstellung der Testtrajektorie wird der zweite Modus der zuvor behandelten Simulation verwendet, um sieben Kreissegmente mit einem Meter Radius im Gegenuhrzeigersinn zu generieren, die von dem Roboter - dieses Mal nicht simuliert - ausschließlich mit Odometrielokalisierung abgefahren werden sollen. Durch die Klothoide als Übergangsbogen entsteht jedoch ein Pfad, der eher einem stark abgerundeten Quadrat ähnelt, mit relativ geraden Seitensegmenten gefolgt von Kreissegmenten mit stetig kleiner werdenden Krümmungsradien.

Warum nicht zwei volle Kreise sondern nur sieben Viertelkreise als Testtrajektorie verwendet werden, liegt an der Implementierung des **TaskHandler**-Moduls, bei der zu Beginn die Start- und Endpose verglichen wird und bei ausreichender Übereinstimmung das Ziel direkt als Erreicht zurückgegeben wird. Der Startpunkt liegt auf einer der Geraden, die an der x-Achse des iGPS-Bezugssystems ausgerichtet wird. Hierfür wird der Roboter langsam manuell mit einer Handfernsteuerung entlang der virtuellen Linie gefahren, während parallel die Ausgabe des iGPS beobachtet wird. Durch vorsichtige Eingaben wird der Roboter so lange ausgerichtet, bis sich der Gierwinkel stabil bei ca. 0° einpendelt. Diese Vorgehensweise dient nicht nur der Erleichterung der späteren Auswertung der Daten, sondern stellt auch sicher, dass die Versuche wiederholbar mit nahezu identischen Anfangsbedingungen durchgeführt werden. Wie bereits bei der Auswertung der visuellen Positionsbestimmung erwähnt, sorgt das robuste Fahrwerk, zu sehen in Abbildung 4.5, durch die massiven Castor-Rollen, trotz des enormen Gewichts des Roboters von ca. 710 kg, dafür, dass der Roboter, beim Eindrehen der Rollen in Fahrtrichtung, leicht zur Seite der Drehrichtung weggedrückt wird. Wenn die Castor-Rolle in eine andere Richtung geschwenkt wird, entsteht Reibung am Boden wofür zusätzliche Energie benötigt wird. Ist der Nachlauf, also der Abstand zwischen der vertikalen und horizontalen Drehachse, klein, so wie in diesem Fall, wird dabei wegen des Hebelgesetzes zusätzlich Kraft benötigt. Von Hand lassen sich die Castor-Rollen nicht in Fahrtrichtung drehen, da nicht genug Kraft aufgebracht werden kann. Der Einfluss der Rollen ist zu Beginn der grafischen Darstellung des Gierwinkels in den Auswertungen 4.19 und 4.20 gut durch starke Schwankungen der Kamera- und iGPS-Daten zu erkennen.

Auswertung und Diskussion

Nach Ausrichtung des Roboters auf einen Gierwinkel von 0° des iGPS und der Castor-Rollen in Fahrtrichtung wird nun mit der Ausführung des Testpfades begonnen. Zu Beginn wird die Auswirkung des Dämpfungskoeffizienten a, mit Standardwert 0.2, isoliert getestet. Abbildung 4.26 **(a)** zeigt den vom iGPS aufgezeichneten Pfad in rot und die von der **ControlStation** berechnete Ground Truth in blau. Obwohl der **Controller** in diesem Modus die Odometriewerte direkt mit den im `odom_frame` definierten Sollwerten der Pfadsegmente vergleicht, kommt es direkt nach Erreichen des ersten Kreisbogens zu einem starken Oszillieren und auch der Abschnitt mit nahezu geraden Segmenten bewirkt kein Konvergieren der Schwingung. Die Gleichungen für die Regeleingaben u_1 und u_2 (jeweils aus Gleichungen 3.4 und 3.5) und die Definition der Regelverstärkung 3.6 zeigen, dass sich der Dämpfungskoeffizient a direkt proportional auf u_1 und u_2 auswirkt und somit die Werte des Fehlervektors $\vec{e}$ stärker gewichtet. Warum sich der Regler bei gleichem Dämpfungskoeffizient a unter Verwendung von AMCL anders verhält als im reinen Odometriebetrieb, ist nicht eindeutig. Nach den ersten zwei Viertelkreisen registriert der

Controller eine zu große Abweichung von der gewünschten Position und beendet die Ausführung. Dieser Versuch wurde bereits mit der dynamischen Rate r_{dyn} 3.9 durchgeführt, um das Verhalten des zu hoch gewählten Dämpfungskoeffizienten zu illustrieren.

Ähnlich wird mit dem zweiten Versuch verfahren, bei dem ein besser geeigneter Dämpfungskoeffizient von $a = 0.075$ verwendet wird, der Regler sich jedoch mit einer statischen Rate von 50 Hz aktualisiert. Abbildung 4.26 **(b)** zeigt das Ergebnis, bei dem es dem Roboter dieses Mal gelungen ist, die gesamte Trajektorie von sieben Viertelkreisen abzufahren. Jedoch fängt er ebenfalls nach Erreichen des ersten kreisförmigen Segments an, stark von der geplanten Trajektorie zu divergieren. Durch den niedriger gewählten Wert des Dämpfungskoeffizienten sind zumindest kaum noch Schwingungen zu erkennen. Lediglich ein leichtes Nachregeln beim Durchschreiten der geraden Segmente ist zu beobachten. Der Verlauf der Trajektorie in Abbildung 4.26 **(b)** zeigt die zu erwartenden Auswirkungen durch leichte Überschreitungen der Kreissegmentzeiten, die im Unterabschnitt 3.5.2 als Grund für die Implementierung der dynamischen Rate aufgelistet wurden. Die durchschnittliche Abweichung der gefahrenen Strecke von dem geplanten Pfad beträgt 17.4 cm. Dieser Wert errechnet sich aus dem nächstgelegenen Punkt der iGPS Daten zu den Endpositionen der einzelnen Segmente, jedoch unter Beachtung der chronologischen Reihenfolge. Um die Güte der einzelnen Versuche dieses Experiments zu vergleichen, ist, in Verbindung mit der visuellen Bestätigung des Graphen, diese vereinfachte Berechnung ausreichend, auch wenn die tatsächliche mittlere Abweichung vermutlich etwas höher ist.

Nach den zwei Versuchen mit jeweils Rate und Dämpfungskoeffizient isoliert, werden jetzt weitere Versuche zur Ermittlung eines optimalen Dämpfungskoeffizienten durchgeführt. Die Abbildungen 4.26 **(c)** und **(e)** zeigen die obere und untere Grenze des genauer zu untersuchenden Intervalls. Mit $a = 0.05$ in **(e)** schafft es der Regler offenbar nicht mehr auftretende Abweichungen auszugleichen, bis er letztendlich so weit von den Segmentposen abweicht, dass der **Controller**, wie auch in **(a)**, die Ausführung aus Sicherheitsgründen abbricht. Mit $a = 0.1$ in **(c)** ist bereits eine deutliche Verbesserung zu **(a)** zu erkennen. Tatsächlich sind die Resultate bei einer Dämpfung zwischen 0.1 und 0.065 sehr ähnlich, mit dem besten Ergebnis zu sehen in Abbildung 4.26 **(d)**.

Bei wiederholten Versuchen eines bestimmten Wertes des Koeffizienten a sind die Resultate nicht konsistent und es kommt in unregelmäßigen Abständen zu unterschiedlich starken Einlenkungen, was zum Teil an den noch immer vorkommenden Segmentzeitintervallüberschreitungen des **Controller**-Moduls liegt. In derzeit nicht vorhersehbaren Iterationen der Kontrollschleife kommt es zu längeren Berechnungszeiten der Hindernisvermeidung und Positionsdifferenzerkennung, wodurch das betreffende Segment länger als geplant ausgeführt wird. So wird in seltenen Fällen eine Zeitintervallüberschreitung von annähernd 100 ms registriert, obwohl die durchschnittliche Überschreitung durch die dynamische Rate bereits von

$$\mu_{t_{st}} = 13.29\,\text{ms} \tag{4.13}$$

bei einer statischen Rate von 50 Hz auf

$$\mu_{t_{dyn}} = 2.18\,\text{ms} \tag{4.14}$$

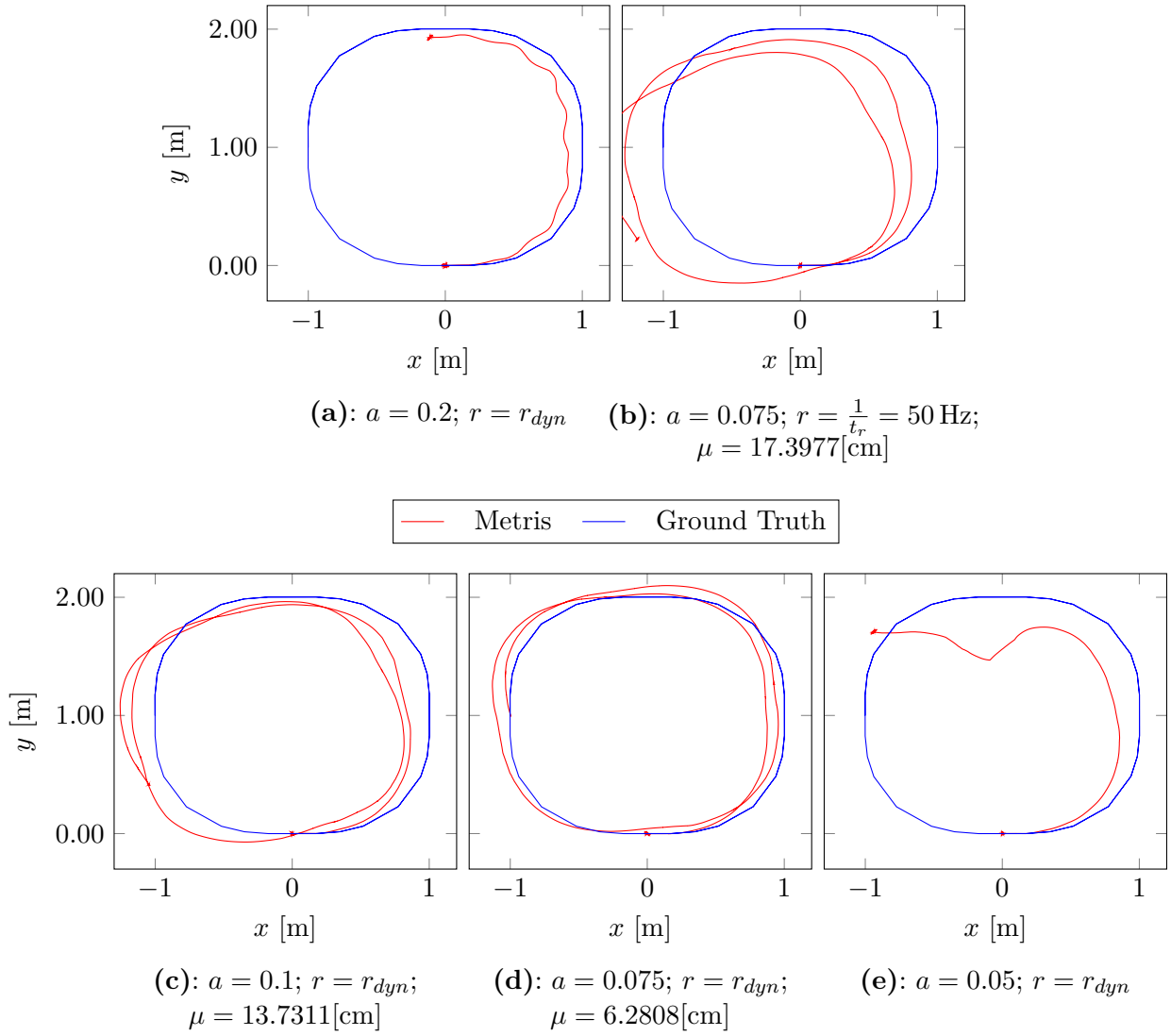
reduziert wird.

Abbildung 4.27 zeigt ausführlich die Messdaten der Zeitüberschreitungsfehler t_{Fehler} der einzelnen Segmente bei dynamischer **(a)** und statischer Rate **(b)**. Wie zuvor erwähnt besteht der Testpfad aus sieben Viertelkreisen. Jedes dieser Viertel besteht wiederum aus vier verschiedenen Segmenten, die, angefangen im Zentrum eines geraden Abschnitts der Plots in Abbildung 4.26, in der Reihenfolge $0, 1, 2, 3, 2, 1, 0$ durchgeführt werden. In der Legende der Abbildung 4.27 ist die Dauer t und Winkelgeschwindigkeit $w_d = \frac{v_d}{Krümmungsradius}$ aufgeführt. Überschreitungen der Segmente eins bis drei tragen daher, aufgrund der hohen Krümmung, besonders stark zu Pfadabweichungen bei.

Im direkten Vergleich der Daten aus **(a)** und **(b)** zeigt sich, dass zwar der Mittelwert der dynamischen Rate um eine Größenordnung geringer ist, sich dafür aber eine leicht höhere Standardabweichung ergibt, mit signifikant erhöhter Häufigkeit von starken Ausreißern von über 20 ms. Eventuell hat der **Controller** Probleme mit sich ständig verändernden Raten, jedoch traten Ausreißer dieser Art ähnlich häufig auch bei Versuchen mit höheren statischen Raten auf.

Um den Einfluss dieser Überschreitungen isoliert zu bewerten, wird eine weitere Simulation durchgeführt, bei der die Dauer t von jedem Segment um die durchschnittlichen Überschreitung erhöht wird. Bei einem Wert von $\mu_{t_{st}} = 13.29\,\text{ms}$ (4.13) der statischen Rate ergibt sich eine durchschnittliche Abweichung des Testpfades von 0.013 86 m und bei einem Wert von $\mu_{t_{dyn}} = 2.18\,\text{ms}$ (4.14) der dynamischen Rate ergibt sich eine durchschnittliche Abweichung von 0.002 671 m. Dies entspricht 1/23 der durchschnittlichen Abweichung in 4.26 **(d)** und kann somit vorerst vernachlässigt werden.

Zurück zur Auswertung der Abbildung 4.26 ergibt sich selbst bei dem besten Ergebnis in **(d)**, bei dem es zu keiner nennenswerten Zeitintervallüberschreitung kam, eine Tendenz zum Übersteuern. Zusätzlich zu den bereits erwähnten Einflussfaktoren der Hardware und Software ist der Einfluss des massiven Fahrwerks des Roboters nicht zu unterschätzen, dessen breite Hartgummirollen der differenziellen Antriebseinheit unweigerlich zu einem gewissen Rutschen der Bereiche der Rollen führt, die nicht Drehpunkt der Kurve sind, was in einem gut wahrnehmbaren Quietschen in engen Kurven resultiert. Vermutlich wandert dieser Drehpunkt je nach Verschmutzungsgrad des Bodens und dessen Beschaffenheit. Aufgrund des Alters und der regelmäßigen Benutzung des Roboters ist eine allgemeine Dekalibrierung der Antriebseinheit ebenfalls nicht auszuschließen. Die genaue Gewichtung der Einflussfaktoren auf den Fehler ist schwer oder unmöglich zu erforschen.



(a): $a = 0.2$; $r = r_{dyn}$

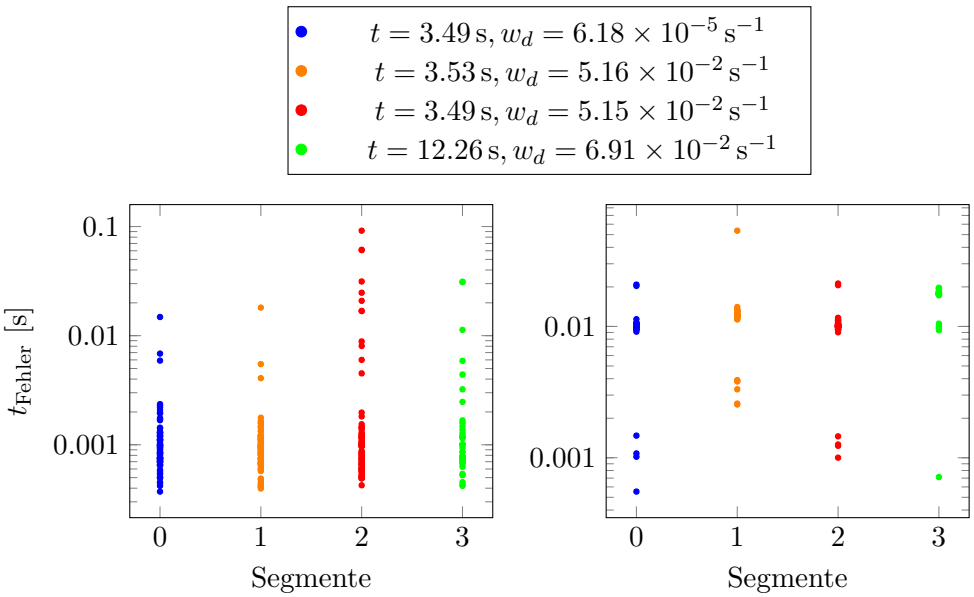
(b): $a = 0.075$; $r = \frac{1}{t_r} = 50\,\text{Hz}$; $\mu = 17.3977[\text{cm}]$

(c): $a = 0.1$; $r = r_{dyn}$; $\mu = 13.7311[\text{cm}]$

(d): $a = 0.075$; $r = r_{dyn}$; $\mu = 6.2808[\text{cm}]$

(e): $a = 0.05$; $r = r_{dyn}$

Abbildung 4.26: Metrisaufzeichnungen der Roboterposition bei verschiedenen **Controller**-Parametern für die Rate r und den Dämpfungsfaktor a. Die durchschnittliche Pfadabweichung wird durch μ angegeben (außer bei **(a)** und **(e)**, da hier der Pfad nicht vollständig abgefahren wurde).



(a) Dynamische Rate
$\sigma = 7.5186 \times 10^{-3}\,\mathrm{s}$
$\mu = (2.1789 \pm 0.3985) \times 10^{-3}\,\mathrm{s}$

(b) Statische Rate
$\sigma = 5.7573 \times 10^{-3}\,\mathrm{s}$
$\mu = (1.3286 \pm 0.0371) \times 10^{-2}\,\mathrm{s}$

Abbildung 4.27: Darstellung der Zeitintervallüberschreitungen der vier Segmente des Testpfades aus Unterabschnitt 4.4.2 mit **(a)** der dynamischen Rate und **(b)** der statischen Rate von 50 Hz. Jede Spalte plottet die Überschreitungen t_{Fehler} der einzelnen Segmente, dessen Eigenschaften - Dauer t und Winkelgeschwindigkeit w_d - in der Legende aufgelistet sind.

4.4.3 Dockingmanöver

Nach ausgiebigen Tests der einzelnen Komponenten dieser Arbeit und Aufschlüsselung der Imperfektionen wird in diesem Unterabschnitt das System als Gesamtheit getestet. Anders als in den Versuchen zuvor bedarf dies nicht nur ein wiederholbares Ausführen eines isolierten Moduls, sondern ein Zusammenarbeiten einer komplexen Softwarestruktur, die aus drei Programmiersprachen und dutzenden Bibliotheken besteht. Von der Auftragserstellung in der Benutzeroberfläche nach Abbildung 3.11 bis zur letzten Aktion des Programmablaufplans in Abbildung 3.1 werden Benutzereingaben entgegengenommen, Datenbanken aktualisiert, Auftragsscheduling betrieben, Datenaustausch mit Robotern aufrechterhalten, photogrammetrische Methoden angewendet, Trajektorien geplant und dynamisch auf den Status des Auftrags reagiert.

Versuchsdefinition und -aufbau

Wie in der visuellen Positionsbestimmung und der Trajektorienausführung wird ein externes System benötigt, um die Präzision der Andockmanöver zu ermitteln, wofür wieder auf das vom ZfT bereitgestellte iGPS zurückgegriffen wird. Des Weiteren wird die Positionierung des Sensorrahmens direkt über dem Zentrum der Kamera, wie in Abbildung 4.8 zu sehen, beibehalten. Der Aufbau der iGPS-Module und die Koordinatensystemdefinition ist in Abbildung 4.24 dargestellt. Die Anordnung der Transmitter des iGPS orientiert sich nach dem in [35] empfohlenen C-Shape, damit die provisorische Andockstation im Zentrum des geringsten anzunehmenden Fehlers liegt und nicht den Blickkontakt der Transmitter zu dem Sensorrahmen stört. Das iGPS-Bezugssystems wird mit dem Ursprung in die obere rechte Ecke des Patterns der provisorischen Andockstation (siehe Abbildung 4.6) gelegt und die Achsenorientierung mit der x-Achse nach unten, y-Achse nach links und der z-Achse in die Pattern-Ebene zeigend, festgelegt. Durch die Koordinatentransformation in 3.4 werden die extrinsischen Parameter der Kamera in das Weltkoordinatensystem überführt.

Im Folgenden wird einer von drei Andockexperimenten evaluiert, die jeweils aus mindestens 30 vollständigen Andockmanövern bestehen. Die Ausgangslage jedes Manövers ist gleich für die vorgegebenen Posen der Anfahrstellung, markerzentrierenden Stellung und Andockstellung. Die Anfahrstellung wird aufgrund des Einflusses der Castor-Rollen auf die Trajektorie ca. einen Meter hinter die markerzentrierende Stellung platziert, so dass beide Stellungen auf einer Linie mit dem Mittelpunkt des Patterns liegen und zum Mittelpunkt des Patterns orientiert sind.

Zur Fehlerfortpflanzungsvermeidung wird die in Unterabschnitt 4.3.3 bestimmte Hand-Eye-Transformation nicht auf die Kamerapose angewendet. Die relativen Abweichungen zwischen dem Sensorrahmen und der Kamera sind also identisch zu den visuellen Positionsbestimmungsexperimenten. Das Hauptaugenmerk liegt bei der Reproduzierbarkeit des Erreichens einer Pose nahe der Andockstellung ($-1\,\mathrm{m}$, $0\,\mathrm{m}$, $0°$) und der anschließenden Zuordnung der systematischen Abweichungen zu den zuvor getesteten einzelnen Komponenten.

Auswertung und Diskussion

Die Histogramme 4.29 **(a)**, **(b)** und **(c)** zeigen die Verteilung der Kamera- und iGPS-Posen an der markerzentrierenden Stellung. Aus den isolierten Experimenten der einzelnen Komponenten zuvor ist bekannt, dass ein relativer Gierwinkel kleiner als 25° zwischen Kamera und Pattern zu vermeiden ist, da dies zu Ungenauigkeiten und Mehrdeutigkeiten bei der Posebestimmung führt und dass die Trajektorie so geradlinig wie möglich zu wählen ist, um den Drift der Odometrie möglichst gering zu halten. Aus diesen Gründen ist die markerzentrierende Stellung an der äußersten Intervallgrenze bei 25° bis 27° in einer Entfernung von ca. 3.45 m positioniert.

Nach den Abbildungen 4.29 ergibt sich somit für die Kamera aus 37 gemessenen Posen eine mittlere markerzentrierende Stellung von

$$\begin{aligned} z &= (-2.9314 \pm 0.0039)\,\mathrm{m} \\ y &= (1.7021 \pm 0.0047)\,\mathrm{m} \\ \psi &= (-25.2381 \pm 0.1957)\,^\circ \end{aligned}$$

und für den Sensorrahmen aus 37 gemessenen Posen

$$\begin{aligned} z &= (-2.9773 \pm 0.0039)\,\mathrm{m} \\ y &= (1.7084 \pm 0.0046)\,\mathrm{m} \\ \psi &= (-26.7297 \pm 0.1927)\,^\circ \end{aligned}$$

Wie in den visuellen Positionsbestimmungsexperimenten ist auch hier an den Mittelwerten der gemessenen Orientierung die bekannte systematische Differenz der Kameraorientierung von ca. 1.5° abzulesen, sowie der Versatz entlang der Ursprungsgeraden von ca. 4.5 cm.

Die Histogramme 4.30 **(a)**, **(b)** und **(c)** zeigen die Verteilung der Kamera- und iGPS-Posen an der Andockstellung. Aus dem dritten visuellen Positionsbestimmungexperiment, sowie der Hand-Eye-Kalibrierung, ist bekannt, dass die Posenschätzung bei direkter Draufsicht der Kamera auf die AprilTags zu Ungenauigkeiten führt und dass sich, aufgrund der Brennweite des Objektivs, das Pattern in einem Meter Entfernung außerhalb des Fokusbereichs befindet und zusätzlich mit einem erhöhten Reprojektionsfehler gerechnet werden muss. All diese Effekte sind an der Abbildung 4.31 der Kamerawiedergabe der GUI nach Erreichen der Andockstellung gut zu erkennen. Zudem wird aus einem Meter Entfernung nicht das ganze Pattern erfasst. Dennoch werden die gemessenen Kameraposen mit aufgeführt, um die Auswirkungen dieser Effekte zu evaluieren und sie mit den gemessenen iGPS-Daten zu vergleichen.

Nach den Abbildungen 4.29 ergibt sich somit für die Kamera aus 31 gemessenen Posen eine mittlere Andockstellung von

$$z = (-1.0082 \pm 0.0018)\,\mathrm{m}$$
$$y = (0.0214 \pm 0.0023)\,\mathrm{m}$$
$$\psi = (-1.9851 \pm 0.2732)\,^\circ$$

und für den Sensorrahmen aus 31 gemessenen Posen

$$z = (-1.0447 \pm 0.0017)\,\mathrm{m}$$
$$y = (0.0257 \pm 0.0026)\,\mathrm{m}$$
$$\psi = (-1.8221 \pm 0.1998)\,^\circ$$

Verglichen mit den Mittelwerten der markerzentrierenden Stellung ist zu erkennen, dass die Standardverteilung der von der Kamera geschätzten Gierwinkel in der Andockstellung deutlich von der Standardverteilung des iGPS abweicht, wodurch es auch zu einer stärkeren Streuung der y-Werte kommt.

Wider Erwarten hat sich die Distanz zwischen der Kamera- und iGPS-Pose so nah am Pattern von 4.5 cm auf 3.7 cm verringert. Abbildung 4.31 zeigt, dass nur ein AprilTag in der Andockstellung von der Kamera erfasst wird, wodurch lediglich vier Punktpaare in Korrespondenz gebracht werden, was zu einem verringerten Reprojektionsfehler von ca. 0.07 px führt. Zugleich verringert sich jedoch die Pixelbreite d_x und die reale Breite T_x um ein Siebtel auf 193 px beziehungsweise 0.068 m, wodurch sich der Reprojektionsfehler wiederrum stärker auswirkt. Nach den Methoden aus Unterabschnitt 4.3.3 entsteht somit ein zu erwartender maximaler Distanzfehler von 1.45 mm. Jedoch ist der zu erwartende Distanzfehler, wenn das gesamte Pattern im Sichtfeld ist und der Reprojektionsfehler ca. 0.35 px beträgt, deutlich geringer mit 0.5 mm, weshalb davon auszugehen ist, dass der Unschärfeeffekt auf die Posenschätzung deutlich geringer ist, wenn lediglich vier Punktkorrespondenzen registriert werden.

Wie zuvor erwähnt ist die vorgegebene Andockpose (−1 m, 0 m, 0°), somit ergibt sich eine Abweichung der iGPS-Pose um ca. (−4.47 cm, −2.57 cm, −1.82°). Die Verteilung der gemessenen Posen des iGPS und der Kamera in der markerzentrierenden Stellung ist erstaunlich ähnlich, so dass mit präziser Transformation der Kamerapose bezüglich der Roboterbasis eine Verschiebung der iGPS-Zielpose in die vorgegebene Andockstellung problemlos möglich ist. Um zu überprüfen, ob der Sensorrahmen evtl. der Roboterorientierung mehr entspricht als die Kamera, wird die Pfadberechnung mit den Mittelwerten der Kamerapose durchgeführt und dann der Pfad von den Mittelwerten des iGPS an der markerzentrierenden Pose ausgeführt, wodurch sich eine erwartete Zielpose bei (−1.0831 m, −0.0468 m, −1.4335°) ergibt. Somit weicht die zu erwartende Pose um ca. (3.84 cm, −2.29 cm, 0.39°) von der iGPS-Zielpose ab. In Abbildung 4.28 wird der Pfadverlauf mit den zwei unterschiedlichen Anfangsposen dargestellt. Da ein Gierwinkel von 0° einer Orientierung parallel zur z-Achse bedeutet und die Zielpose bei $y = 0$ m und $\psi = 0°$ vorgegeben ist, führt dies systematisch zu einer geringeren Abweichung von der markerzentrierenden Orientierung zur Zielorientierung. Diese Abweichung wirkt sich direkt auf die Trajektorienplanung aus, die über den resultierenden geplanten Pfad ca. 1.5° weniger einlenkt, was schließlich

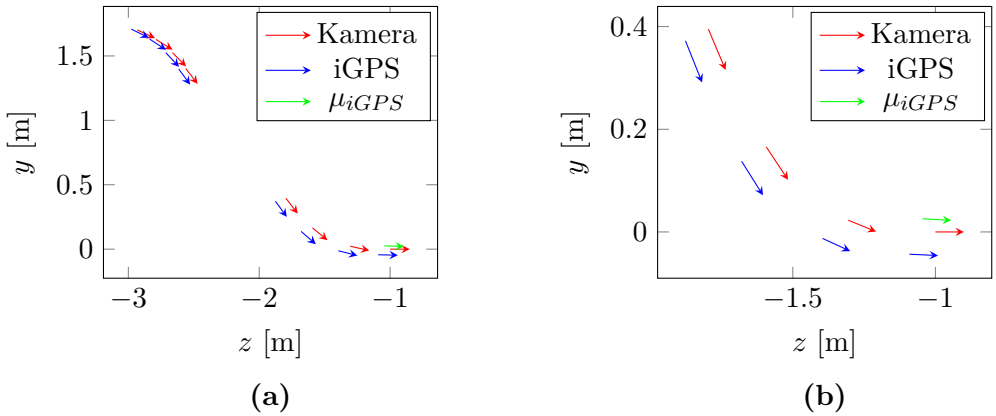


Abbildung 4.28: **(a)** Darstellung des durchschnittlich berechneten Pfades beginnend bei der mittleren Kamerapose in rot beziehungsweise bei der mittleren iGPS-Pose in blau und die mittlere iGPS Andockstellung in grün. **(b)** Zeigt die letzten vier Trajektorieposen zur besseren Veranschaulichung.

zu einer Verlagerung des Mittelwertes der Zielorientierung in Abbildung 4.30 **(c)** von 0° zu −1.8° führt. Dies führt also zu ähnlichen Werten für z und ψ, jedoch zu einer Verlagerung der y-Werte ins Negative. Der grüne Pfeil in 4.28 **(b)** zeigt die mittlere iGPS-Pose, der letzte rote Pfeil die vorgegebene Andockpose und der letzte blaue Pfeil die erwartete Pose, wenn der Sensorrahmen mit der Roboterbasis in der markerzentrierenden Stellung übereinstimmt.

Bisher wurde jedoch noch nicht der Einfluss des **Controller**-Moduls berücksichtigt. Es ist folglich anzunehmen, dass die Abweichungen zu der zu erwartenden Pose nach Abbildung 4.28 von ca. (3.84 cm, −2.29 cm, 0.39°) zu einem großen Teil auf die bisher noch nicht mit einbezogenen Einflüsse des **Controller**-Moduls zurückzuführen sind, der, wie die grafischen Darstellungen 4.26 zeigen, dazu tendiert etwas weiter einzudrehen und, beim Übergang in Segmente geringerer Krümmung, leicht auszuschwingen.

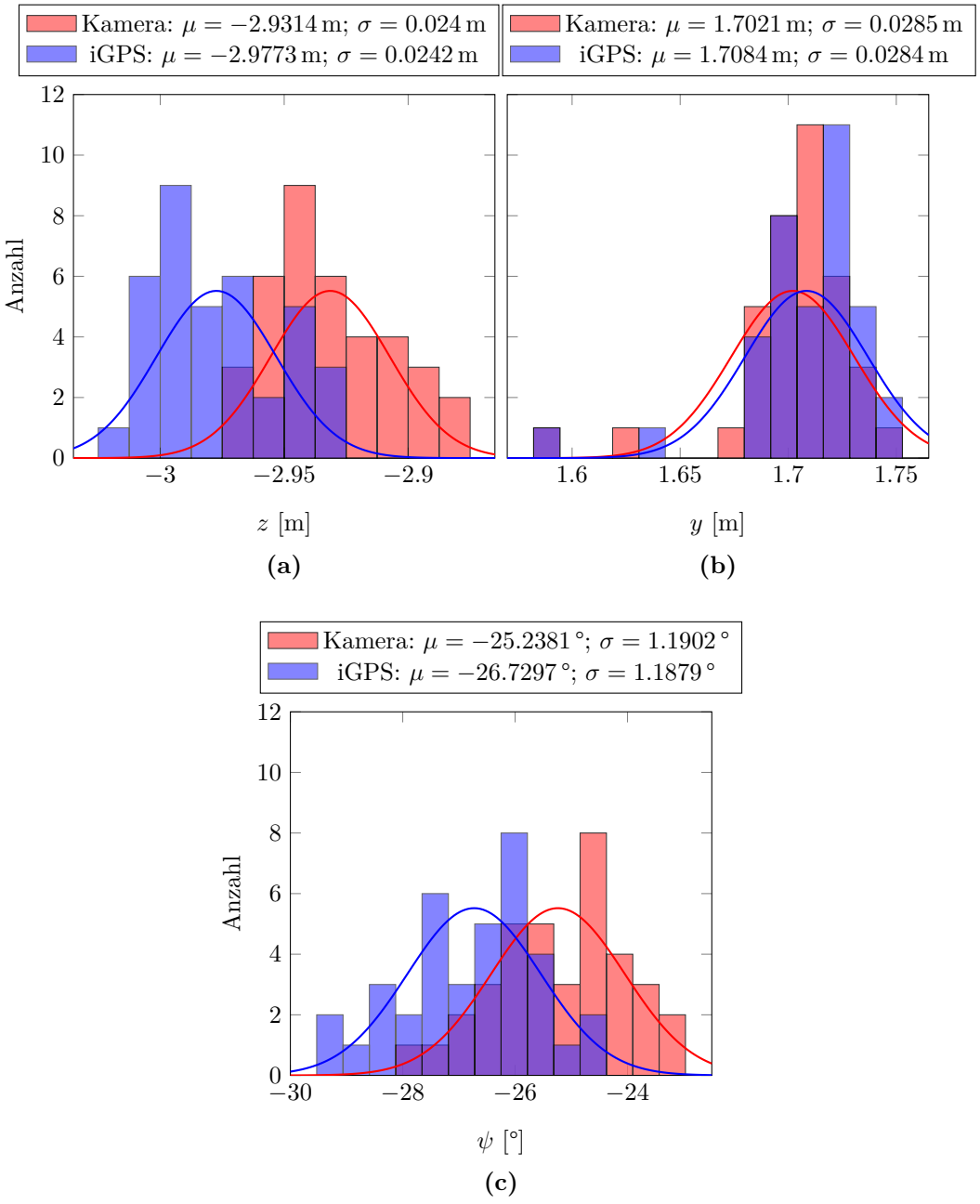


Abbildung 4.29: Auswertung der Kamera- und iGPS-Daten an der markerzentrierenden Stellung.

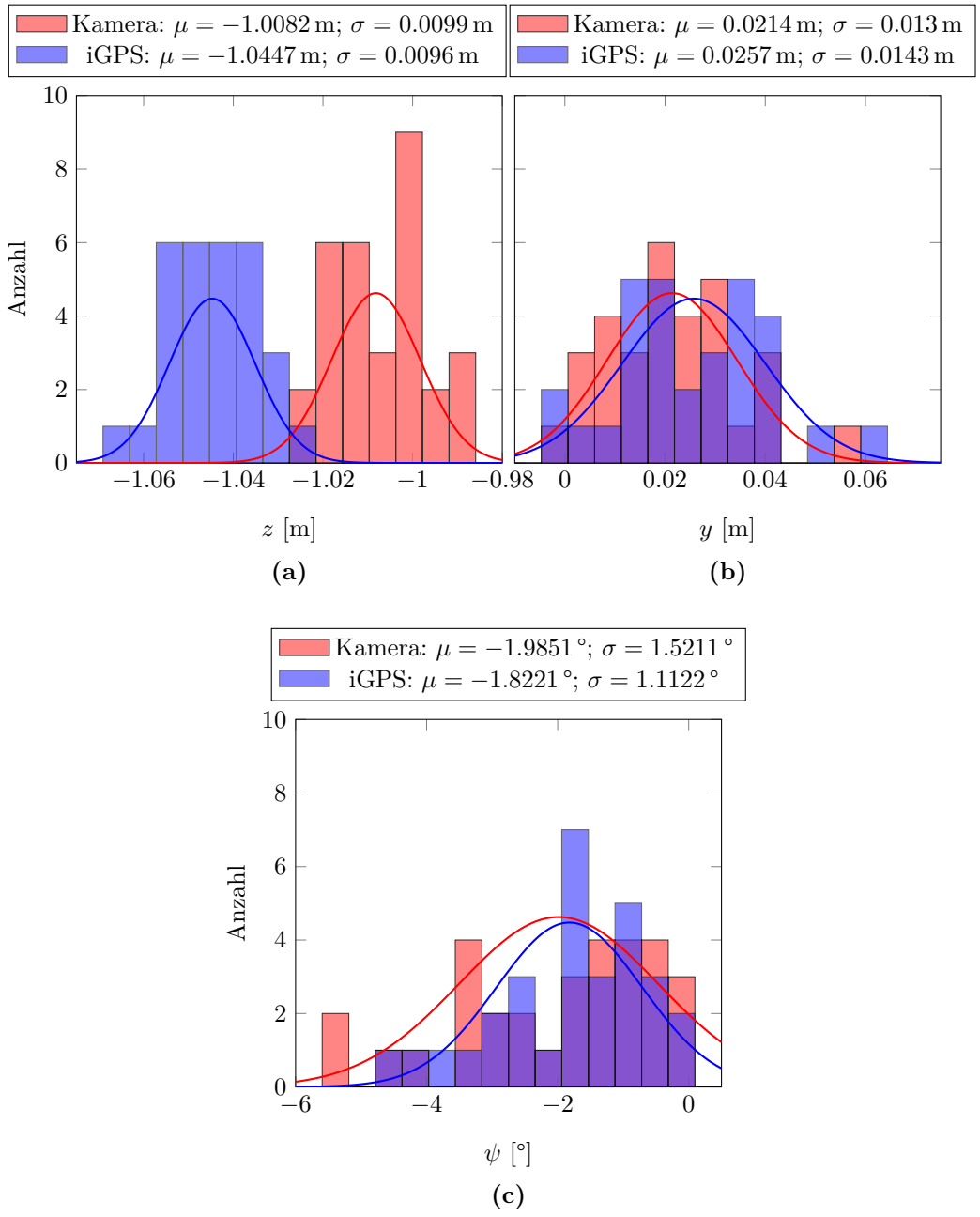


Abbildung 4.30: Auswertung der Kamera- und iGPS-Daten an der Andockstellung. Die Sollpose ist: $z = -1\,\mathrm{m}$, $y = 0\,\mathrm{m}$, $\psi = 0\,°$.

Abbildung 4.31: GUI-Kameraausgabe des Roboters nach Einnahme der Andockstellung bei (−1 m, 0 m, 0°) relativ zum Patternursprung. Die Kamerapose ergibt sich aus der letzten Spalte der Matrix in Meter und des Yaw-Winkels (Gierwinkel) in rad.

Resümee

In dieser Arbeit wurde eine voll funktionsfähige, auf visuelle Fiducial-Detektierung basierende Andockroutine in eine komplexe Softwarearchitektur, bestehend aus Benutzeroberfläche, Kontrollstation und Roboterbetriebssoftware, integriert.

Zu diesem Zweck wurde eine erweiterte Eckpunkterfassung entwickelt, die ein Stufenkantenmodell mittels Kostenminimierung an jedes der vier AprilTag-Seiten fittet, um so die Eckpunkte mit bis zu 40 % geringerer Abweichung zur Standardmethode von AprilTag3 zu schätzen und zugleich die Robustheit gegen Ausreißern zu erhöhen. Unter perfekten Simulationsbedingungen wurde sogar eine 10- bis 100-fache Reduzierung der Reprojektionsfehler erreicht. Dies führt zu sehr präzisen visuellen Positionsbestimmungen von durchschnittlich unter 3 mm und Gierwinkelbestimmungen von durchschnittlich unter 0.15° Abweichung zum hochpräzisen Vermessungssystem iGPS. Die sich darauf beziehenden Messungen wurden aus gleich verteilten Distanzen zum Pattern zwischen 1.4 m und 6.2 m und unter konstanter Fahrgeschwindigkeit von 0.1 m/s durchgeführt.

Zur Umsetzung der präzisen relativen Lokalisierung in einen zur Andockstellung führenden Pfad wurde ein Trajektorienplaner entwickelt, der einen groben Pfad aus Reed-Shepp-Segmenten generiert und durch Klothoidenannäherung eine kontinuierliche Krümmung der Segmentübergänge sicherstellt. Schließlich wurde, zur Ausführung der generierten Pfade, der Robotercontroller und dessen Nachlaufregler, durch dynamische Hyperparameter für die Koppelnavigation optimiert und die Roboterbetriebssoftware so angepasst, dass eine stetige Lokalisierung gewährleistet ist. Trotz alledem ergaben die Auswertungen, dass die Koppelnavigation noch große Fehler in das Gesamtsystem des Andockmanövers induziert, die durch ein zu starkes Eindrehen in Kurvensegmenten zu einer mittleren Abweichung von ca. 4.5 cm und 0.4° zur erwarteten Andockstellung führt.

Über eine Vielzahl von Andockmanövern konnte schließlich eine Wiederholungsgenauigkeit von

$$\begin{aligned}\sigma_z &= 0.96\,\mathrm{cm}\\ \sigma_y &= 2.57\,\mathrm{cm}\\ \sigma_\psi &= 1.11\,^\circ\end{aligned}$$

erreicht werden.

Ausblick

Für die Verbesserung der Andockroutine gibt es eine Vielzahl an komplexen sowie grundlegenden Weiterentwicklungsmöglichkeiten. Im Fokus steht dabei vor allem die präzisere Abfahrt der geplanten Trajektorie durch eine verbesserte roboterinterne Lokalisierungsmethode.
Im Folgenden werden drei komplexe Weiterentwicklungen vorgestellt:

- **Lokalisierung**
  Die Entwicklung einer besseren roboterinternen Lokalisierungsmethode, die den Drift der Koppelnavigation ausgleicht und dabei nicht unter Diskontinuität leidet. Eine mögliche Methode ist eine auf der visuellen Odometrie basierende Lokalisierung, die zudem auf die gleiche Kamera und Infrastruktur zugreift. Grundsätzlich driftet die visuelle Odometrie ebenfalls, jedoch für gewöhnlich nicht in dem Ausmaß der in dieser Arbeit verwendeten Rad-Odometrie und ist zweifellos als Referenz geeignet.

- **Fortführende visuelle Positionsbestimmung**
  Auf dem Weg zur Andockstellung ist eine Wiederholung der präzisen visuellen Positionsbestimmungen an bestimmten Stellen erwägenswert, wodurch bei Bedarf der Pfad neu berechnet oder Ausgleichseingaben an den Controller übergeben werden. Zusätzlich ist die Entwicklung eines Gier-Gimbals in Erwägung zu ziehen, das nach [9] potenziell nicht nur die Detektierungsgenauigkeit deutlich verbessert, sondern auch das Fokussieren des Patterns beim Abfahren der gesamten Trajektorie ermöglicht und somit als roboterinterne Online-Lokalisierung fungiert.

- **Umfassende Kalibrierung**
  Unabhängig von Positionsbestimmungsmethoden ist eine Implementation einer umfangreiche Kalibrierungsroutine vorstellbar, die das Andocken auf vordefinierte markerzentrierende Stellungen und Andockstellungen beschränkt. Die vordefinierten Anfahrtrouten werden mit externen Messsystemen, wie beispielsweise dem iGPS, über mehrere Anfahrten justiert. Dadurch ist das System zwar weniger flexibel, aber alle systematischen Abweichungen können bestimmt und ausgeglichen werden. Mit der in dieser Arbeit vorgestellten Andockroutine steht die Option aus beliebigen Stellungen hochpräzise anzudocken vermutlich im Gegensatz, da sich früher oder später jede Hardware dekalibrieren wird. Das gelegentliche Nachjustieren wird durch eine umfassende Kalibrierungsroutine erleichtert.

Eine Weiterentwicklung darüber hinaus betrifft das Pattern selbst, das in dieser Arbeit lediglich aus AprilTags in einer Ebene besteht. Wie in [9] beschrieben, ermöglichen abgewinkelte Pattern nicht nur eine robustere Orientierungsbestimmung, sondern verbessern auch die deutlich verschlechterte Posenschätzung in der zentralen Region des Patternvorbereichs zwischen ±25°.

Schließlich sind noch grundlegende Verbesserungen möglich, wie z.B. die Verwendung größerer Pattern bzw. AprilTags, mit breiterer Basis, die aus mehreren Metern Entfernung auch ohne Zoom detektiert werden. Die Breite der Basis trägt wesentlich zur Präzision der Posenschätzung bei und die geringere Brennweite vergrößert den Blickwinkel und erweitert die Einsatzmöglichkeiten der Kamera. Ganz allgemein lässt sich vermuten, dass die Präzision durch die Größe der Marker, Kameraauflösung, Brennweite des Objektivs, Lichtverhältnisse und Abstand beeinflusst wird, die alle für sich genommen es wert sind weiter untersucht zu werden.

Anhänge

Anhang A

Projektspezifische Algorithmen

Algorithmus A.1: Reeds-Shepp-Pfadgenerierung und Annäherung der Kreissegmente an Klothoide.

Input : Die Startpose p_s des Agenten und die Zielpose p_z.
Output: Eine Liste mit Segmenten des Pfades mit den wenigsten Reeds-Shepp-Pfadsegmenten, dessen Parameter der Agent interpretieren kann und dessen Kreissegmente kontinuierlich gekrümmt sind.

```
1  dx, dy, φ ← p_z − p_s
2  x ← dx ∗ Cos(p_s.theta) + dy ∗ Sin(p_s.theta)
3  y ← −dx ∗ Sin(p_s.theta) + dy ∗ Cos(p_s.theta)
4
5  P ← ∅
6
7  foreach path ρ in this.reedsSheppPaths do          // 12 in [29] beschriebene
     Pfade
8      P ← P ∪ [
9          ρ(x, y, φ),
10         InvertDrivingDirection (ρ(−x, y, −φ)),
11         InvertSteeringDirection (ρ(x, −y, −φ)),
12         InvertSteeringDirection (InvertDrivingDirection (ρ(−x, −y, φ))))
13         ]
14 P ← SortBySegmentCountAscending(P)
15 L' ← ∅
16
17 foreach path L in P do
18     foreach segment s in L do
19         if s.direction is not L[0].direction then
20             continue to next path        // Pfade mit Richtungswechsel
                überspringen
21     L' ← ContinuousCurvatureApproximation(L, p_s)        // Algorithmus A.2
22     if 1e − 3 > Abs(L'[−1] − p_z) > −1e − 3 then
23         break
24 L' ← ∪ MakeTerminalSegment (L'[−1])  // Segment mit Endpose des letzten
25                                      // Segments in L' als Start-/Endpose
26 return L'
```

Algorithmus A.2: Segmentierung einer Reeds-Shepp-Trajektorie und Annäherung der Kreissegmente an eine Klothoide.

Input : Die Startpose p_0 des Agenten und eine Liste L der Reeds-Shepp-Pfadsegmente.
Output: Eine Liste mit Segmenten, dessen Parameter der Agent interpretieren kann.

```
1  L' ← ∅
2  p_i ← p_0
3  p ← {x : 0, y : 0, theta : 0}
4  foreach segment s in L do
5      d ← s.direction is forward ? 1 : −1
6      if s.steering is not 0 then            // (-1, 0, 1) → (left, straight,
         right)
7          d ← −s.steering ∗ d
8
9          x ← p_i.x − d ∗ Sin(p_i.theta)
10         y ← p_i.y + d ∗ Cos(p_i.theta)
11
12         p.theta ← Mod2Pi(p_i.theta + d ∗ s.length)
13         p.x ← x + d ∗ Sin(p.theta)
14         p.y ← y − d ∗ Cos(p.theta)
15
16         A ← ∅
17         L' ← L' ∪ ApproximateClothoid (p_i, p, d, A)        // Algorithmus A.3
18     else
19         p.x += d + s.length ∗ Cos(p_i.theta)
20         p.y += d + s.length ∗ Sin(p_i.theta)
21
22         L' ← L' ∪ MakeSegment(
23             angularSpeed : 0,
24             linearSpeed : d,
25             duration : s.length,
26             startPose : p_i,
27             endPose : p
28         )
29     p_i ← p
30 return L'
```

Algorithmus A.3: Rekursive Approximierung von Klothoiden mit Kreissegmenten.

Input : Die Startpose p_a, Endpose p_b und die Drehrichtung d des Kreissegments, sowie eine Liste A zum aggregieren der Subsegmente.
Output: Eine Liste mit Kreissegmenten nahezu kontinuierlicher Krümmung, dessen Parameter der Agent interpretieren kann.

```
1  d ← Mod2Pi (d)
2  p_c ← GetCenterOfCurvature (p_a, p_b, d)          // Algorithmus A.4
3
4  if p_c is None then
5      return this is root ? A : None
6  ab ← GetSquaredDistance (p_a, p_b)
7  ac ← GetSquaredDistance (p_a, p_c)
8  bc ← GetSquaredDistance (p_b, p_c)
9
10 if this is root or Abs (ac, bc) > 0.001 then
11     C ← GetClothoidSegments (p_a, p_b)
12     foreach pose p_1, p_2 in C starting index 0, C staring index 1 do
13         A ← A ∪ ApproximateClothoid (p_1, p_2, p_1.theta − p_2.theta, A)
14     A ← A ∪ ApproximateClothoid (C[−1], p_b, C[−1].theta − p_b.theta, A)
15
16     return this is root ? A : None
17 r^2 ← (ac + bc)/2
18 r ← Sqrt (r)
19 φ ← Acos ((2 ∗ r^2 − ab)/2 ∗ r^2)          // Mittelpunktswinkel
20
21 L' ← L' ∪ MakeSegment(
22     angularSpeed : d/r,
23     linearSpeed : d,
24     duration : φ ∗ r,
25     startPose : p_a,
26     endPose : p_b
27 )
28 return L'
```

Algorithmus A.4: Bestimmung des Krümmungsmittelpunktes zweier Posen auf einem Kreissegment.

Input : Die Startpose p_a, Endpose p_b und die Drehrichtung d des Kreissegments.
Output: Den Krümmungsmittelpunkt, also den Schnittpunkt der Linien, die senkrecht zur Tangente der Posen p_a und p_b verlaufen.

```
1  dx ← p_b.x − p_a.x
2  dy ← p_b.y − p_a.y
3
4  if d > 0 then
5  |   θ_a ← Mod2Pi (p_a.theta − π/2)
6  |   θ_b ← Mod2Pi (p_b.theta − π/2)
7  else if d < 0 then
8  |   θ_a ← Mod2Pi (p_a.theta + π/2)
9  |   θ_b ← Mod2Pi (p_b.theta + π/2)
10 else
11 |   θ_a ← p_a.theta
12 |   θ_b ← p_b.theta
13 c_a, s_a ← Cos (θ_a), Sin (θ_a)
14 c_b, s_b ← Cos (θ_b), Sin (θ_b)
15
16 det = c_b * s_a − s_b * c_a
17
18 if det is 0 then
19 |   return None
20 u ← (dy * cb − dx * sb)/det
21 v ← (dy * ca − dx * sa)/det
22
23 if u < 0 or v < 0 then
24 |   return None
25 return (p_a.x + ca * u, p_a.y + sa * u)
```

Anhang B

Visuelle Positionsbestimmung

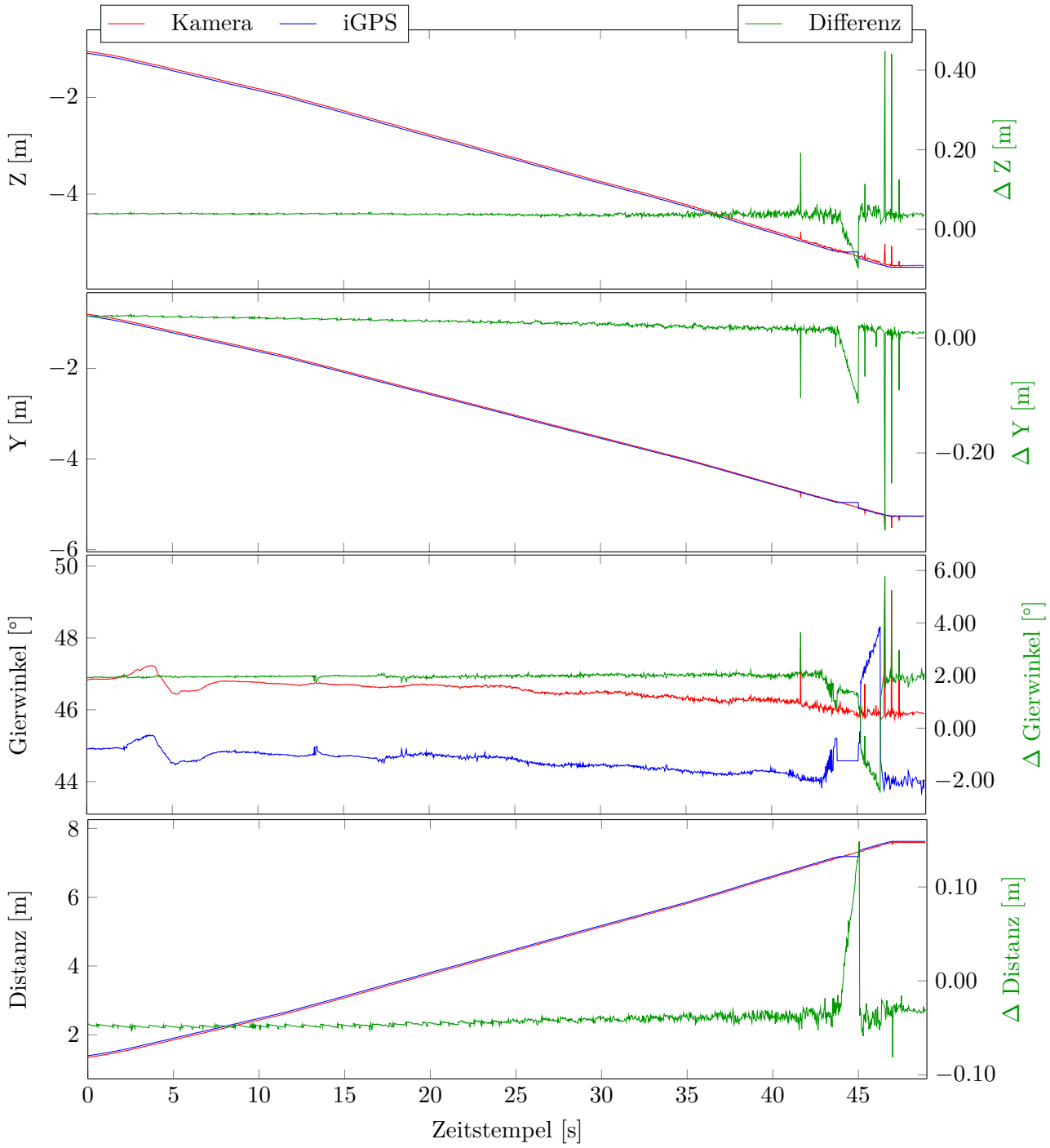


Abbildung B.1: Erster visueller Positionsbestimmungstest in vollständiger Länge.

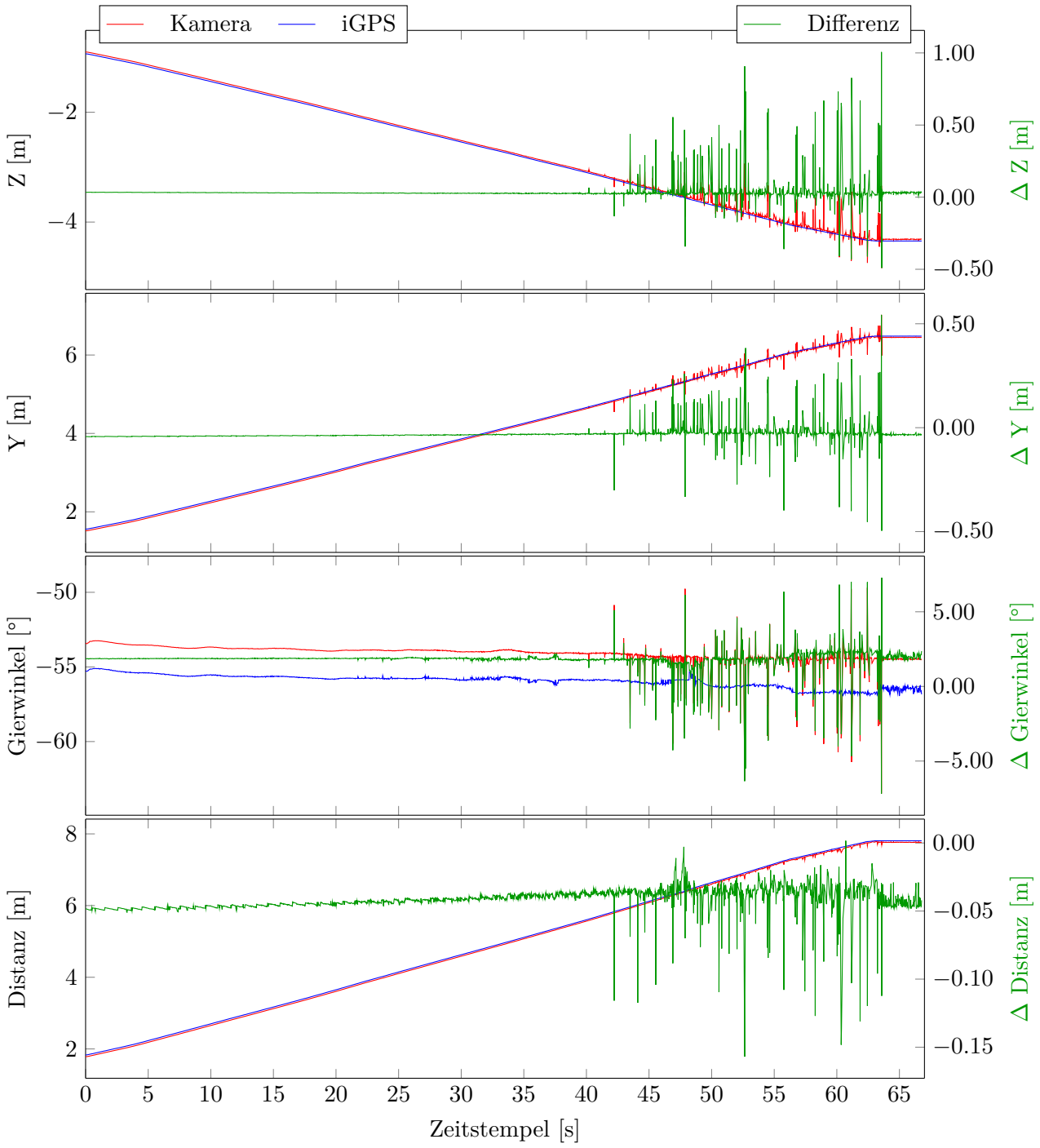


Abbildung B.2: Zweiter visueller Positionsbestimmungstest in vollständiger Länge.

Anhang C

Hand-Eye-kalibrierte visuelle Positionsbestimmung

Nr.	z[m]	y[m]	ψ[°]
1	−1.6138	2.1620	−47.9539
2	−3.8944	4.8366	−48.2428
3	−1.6698	0.2143	1.8981
4	−3.4051	0.2206	1.0634
5	−2.0728	−2.0402	50.6172
6	−3.3153	−3.4757	50.5592
7	−4.6457	5.9847	−46.8404
8	−2.6474	−3.6634	56.0037
9	−1.0517	−1.1415	59.1409
10	−2.0633	0.0047	7.8212
11	−5.4798	−5.2578	45.9063
12	−4.3178	6.4535	−54.4718
13	−4.1682	0.4938	−1.7486

Tabelle C.1: Kameraposen für Hand-Eye-Kalibrierung.

Nr.	z[m]	y[m]	ψ[°]
1	−1.6470	2.1941	−49.9489
2	−3.9224	4.8580	−50.1918
3	−1.7152	0.1988	7.7822
4	−3.4454	0.1731	3.9200
5	−2.1021	−2.0702	48.8643
6	−3.3438	−3.4993	48.6264
7	−4.6763	5.9959	−48.2654
8	−2.6741	−3.6931	54.3597
9	−1.0730	−1.1810	58.1574
10	−2.1070	−0.0052	8.8817
11	−5.5147	−5.2664	43.8837
12	−4.3475	6.4855	−56.4954
13	−4.2121	0.4037	−3.9333

Tabelle C.2: iGPS-Posen für Hand-Eye-Kalibrierung.

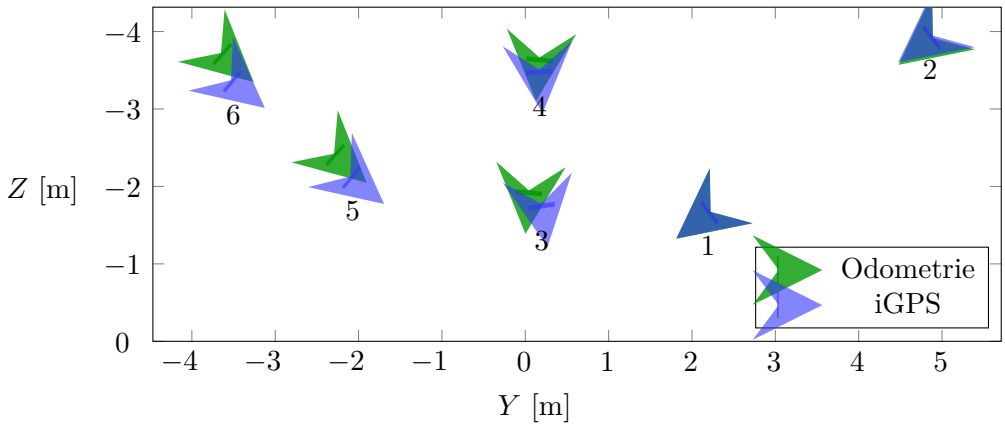


Abbildung C.3: Die erfassten Posen für die Eye-Base-Kalibrierung. Odometrie aus Tabelle C.4 und iGPS die ersten sechs aus Tabelle C.2.

Nr.	z[m]	y[m]	ψ[°]
1	−1.6470	2.1941	−49.9489
2	−3.8920	4.8621	−49.9405
3	−1.8864	0.0391	−4.2536
4	−3.6027	0.1660	−4.2434
5	−2.3943	−2.2643	46.1999
6	−3.6927	−3.6195	46.2303

Tabelle C.4: Odometrieposen für Hand-Eye-Kalibrierung.

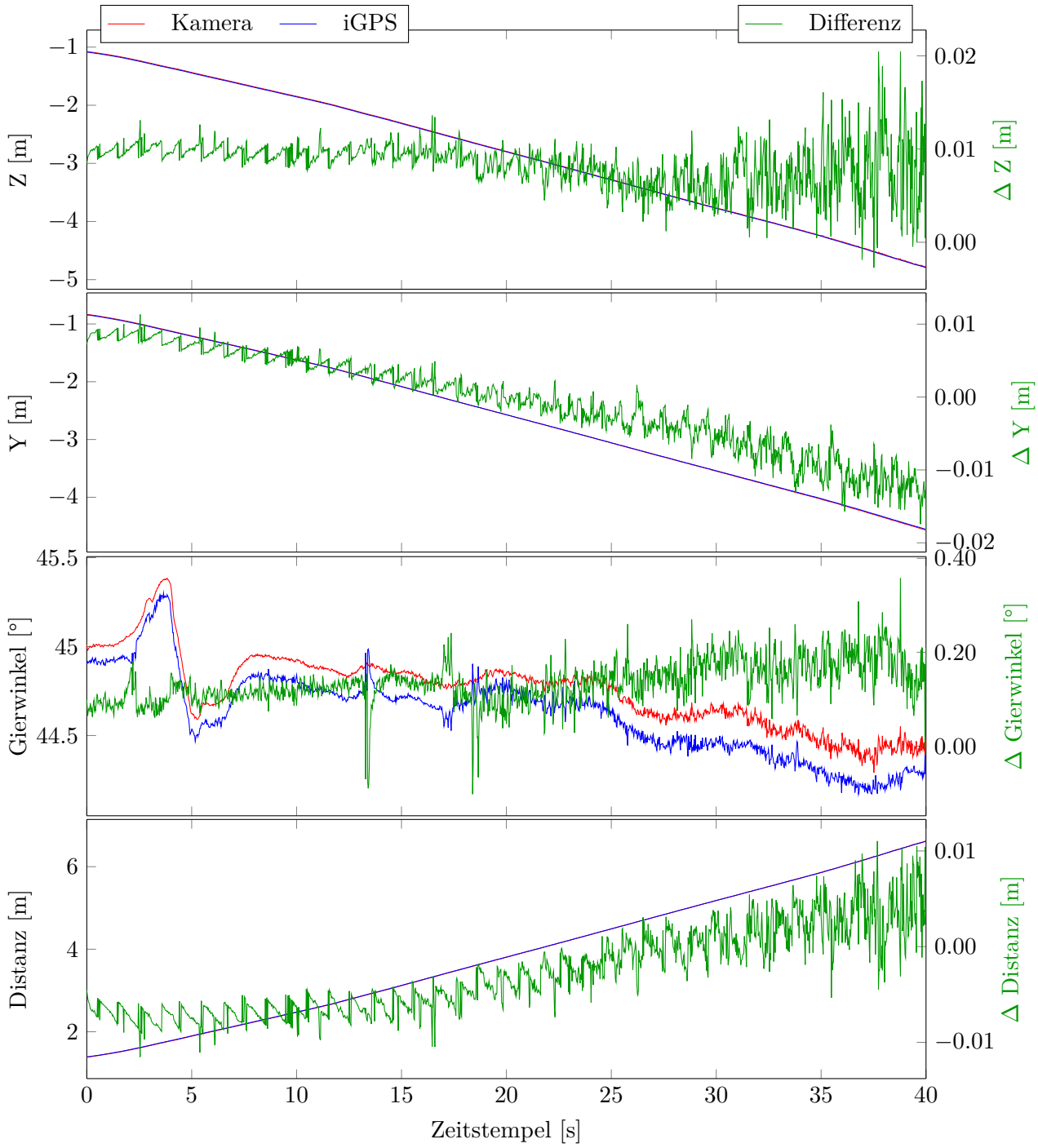


Abbildung C.5: Erster visueller Positionsbestimmungstest in vollständiger Länge und Hand-Eye-kalibrierter Kamerapose.

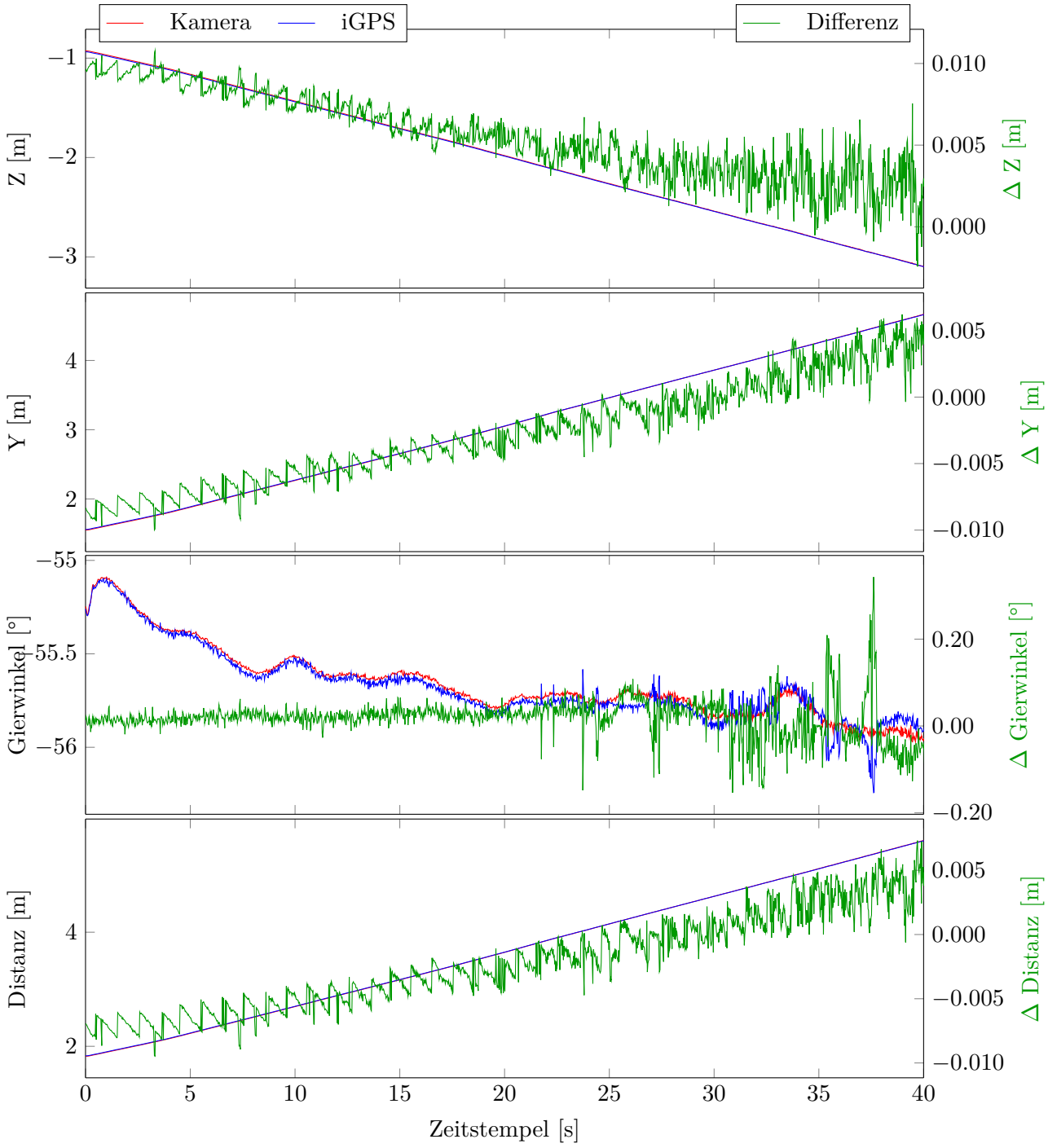


Abbildung C.6: Zweiter visueller Positionsbestimmungstest in vollständiger Länge und Hand-Eye-kalibrierter Kamerapose.

Anhang D

Inhalt des beigefügten Datenträgers

Der im Umschlag dieser Arbeit beigefügte Datenträger enthält die digitale Version der Arbeit im PDF-Format und den Quellcode der implementierten und modifizierten Projekte aus Abbildung 3.2 zzgl. aller Hilfsmodule zur Evaluierung.

Abbildungsverzeichnis

Tabellenverzeichnis

Algorithmenverzeichnis

Literaturverzeichnis

[1] F. Zeitler, "Innospace masters - spin-in und spin-off ideen." `https://innospace-masters.de/wp-content/uploads/2019/05/RC-30-32_INNOspaceMasters.pdf`, 2017.

[2] F. De Ponte Müller, "Survey on ranging sensors and cooperative techniques for relative positioning of vehicles," *Sensors*, vol. 17, no. 2, 2017.

[3] Y. Ju, "Automatic docking for kobuki." `http://wiki.ros.org/kobuki_auto_docking`, 2019.

[4] M. Doumbia, X. Cheng, and V. Havyarimana, "An auto-recharging system design and implementation based on infrared signal for autonomous robots," in *2019 5th International Conference on Control, Automation and Robotics (ICCAR)*, pp. 894–900, 2019.

[5] R. Quilez, A. S. Zeeman, N. Mitton, and J. Vandaele, "Docking autonomous robots in passive docks with infrared sensors and qr codes," *EAI Endorsed Trans. Self Adapt. Syst.*, vol. 1, p. e2, 2015.

[6] F. Alijani, *Autonomous Vision-based Docking of a Mobile Robot with four Omnidirectional Wheels*. PhD thesis, 01 2017.

[7] L. A. Mateos, "Apriltags 3d: Dynamic fiducial markers for robust pose estimation in highly reflective environments and indirect communication in swarm robotics," *CoRR*, vol. abs/2001.08622, 2020.

[8] D. Abawi, J. Bienwald, and R. Dorner, "Accuracy in optical tracking with fiducial markers: an accuracy function for artoolkit," in *Third IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 260–261, 2004.

[9] S. M. Abbas, S. Aslam, K. Berns, and A. Muhammad, "Analysis and improvements in apriltag based state estimation," *Sensors*, vol. 19, no. 24, 2019.

[10] J. Wang and E. Olson, "AprilTag 2: Efficient and robust fiducial detection," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2016.

[11] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second ed., 2004.

[12] opencv dev team, "Camera calibration and 3d reconstruction." https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html, 2019.

[13] Y. Abdel-Aziz, H. Karara, and M. Hauck, "Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry," *Photogrammetric Engineering and Remote Sensing*, vol. 81, no. 2, pp. 103–107, 2015.

[14] A. R. L. M. R. Wall, Michael E., "Singular value decomposition and principal component analysis," *A Practical Approach to Microarray Data Analysis*, 2003.

[15] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, June 1981.

[16] OpenCV, "PnPRANSAC — OpenCV, camera calibration and 3d reconstruction." https://docs.opencv.org/master/d9/d0c/group__calib3d.html#ga50620f0e26e02caa2e9adc07b5fbf24e. [Online; accessed 17-July-2020].

[17] B. Pfrommer, "Tagslam: Flexible slam with tags.."

[18] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3400–3407, IEEE, May 2011.

[19] E. Olson, "Apriltag 3." https://github.com/AprilRobotics/apriltag, 2021.

[20] M. Hagara and O. Ondrácek, "Comparison of methods for edge detection with sub-pixel accuracy in 1-d images," in *2014 3rd Mediterranean Conference on Embedded Computing (MECO)*, pp. 124–127, 2014.

[21] M. Hagara and P. Kulla, "Edge detection with sub-pixel accuracy based on approximation of edge with erf function," 2011.

[22] S. W. Lee, S. Y. Lee, and H. J. Pahk, "Precise edge detection method using sigmoid function in blurry and noisy image for tft-lcd 2d critical dimension measurement," *Curr. Opt. Photon.*, vol. 2, pp. 69–78, Feb 2018.

[23] W. Zhang and F. Bergholm, "Multi-scale blur estimation and edge type classification for scene analysis," *International Journal of Computer Vision (IJCV)*, vol. 24, no. 3, pp. 219–250, 1997.

[24] E. W. Weisstein, "Erf. from mathworld–a wolfram web resource.." https://mathworld.wolfram.com/Erf.html.

[25] S. Astanin, "Benchmark various sigmoid functions.." https://gist.github.com/astanin/5270668.

[26] D.-Z. Du, P. M. Pardalos, and W. Wu, *History of Optimization*, pp. 1538–1542. Boston, MA: Springer US, 2009.

[27] S. Agarwal, K. Mierle, and Others, "Ceres solver." http://ceres-solver.org.

[28] A. Bjorck, *Numerical Methods for Least Squares Problems*. Other Titles in Applied Mathematics, Society for Industrial and Applied Mathematics, 1996.

[29] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and bachwards," *Pacific Journal of Mathematics*, vol. 145, pp. 367–393, 1990.

[30] R. Tsai and R. Lenz, "Real time versatile robotics hand/eye calibration using 3d machine vision," in *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pp. 554–561 vol.1, 1988.

[31] B. P. Gerkey, "Adaptive monte carlo localization." http://wiki.ros.org/amcl, 2020.

[32] W. Meeussen, "Coordinate frames for mobile platforms." https://www.ros.org/reps/rep-0105.html#references, 2010.

[33] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control*. Advanced Textbooks in Control and Signal Processing, Springer London, 2010.

[34] Stäubli, "Linking machines with mobile robot systems." https://www.staubli.com/en/robotics/emo-2019/mobile-robot-helmo/, 2019.

[35] C. Depenthal, "igps - a new system for static and kinematic measurements," 2009.

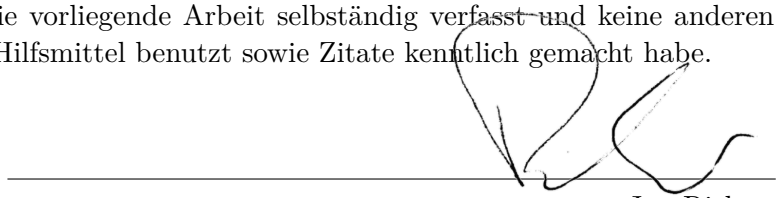
[36] K. Kouzoubov, "Absent documentation for sub-pixel coordinate system." https://github.com/opencv/opencv/issues/10130, 2017.

[37] R. Tsai and R. Lenz, "Real time versatile robotics hand/eye calibration using 3d machine vision," in *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pp. 554–561 vol.1, 1988.

[38] "Ieee standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, 2019.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.



Jan Richter
Würzburg, November 2021