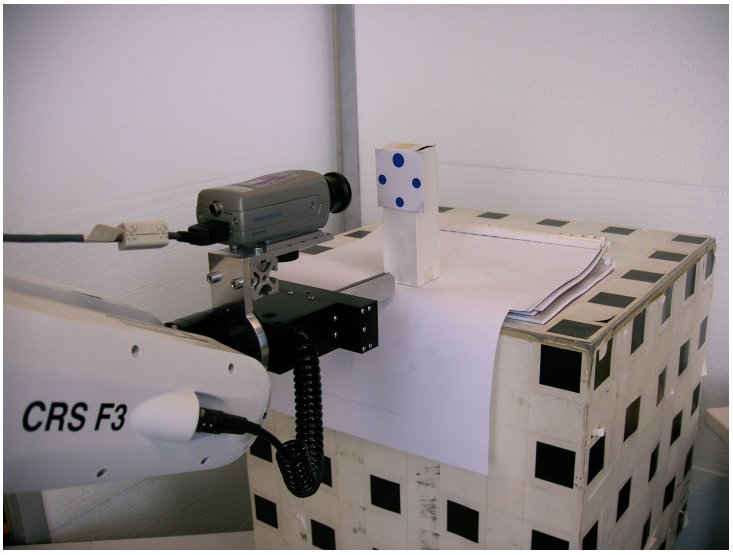
Christian-Albrechts-Universität
Institut für Informatik
Lehrstuhl Kognitive Systeme
Prof. Dr. Sommer
Olshausenstr. 40
24098 Kiel

Diplomarbeit

Zusammenspiel von Modellen und Reglern bei Roboterregelungen

Dennis Peters
(Matrikelnummer: 603371)

24. September 2009



Betreut durch Dr. Nils T. Siebel

Kiel, 24. September 2009

Hiermit versichere ich, dass ich die Arbeit selbsständig verfasst und ausschließlich die angegebenen Hilfsmittel und Quellen verwendet habe.

Dennis Peters

Zusammenfassung

Eine Roboterreglung, wie z. B. das Visual Servoing, unterliegt einem sogenannten Stellgesetz, das aus gesammelten Sensordaten eine Roboterbewegung liefert, die zur Lösung einer Roboteraufgabe führen soll. Dieses Stellgesetz wird mit Hilfe eines Modells beschrieben. Mit diesem Modell lassen sich Umweltveränderungen durch Roboterbefehle vorhersagen. Durch die Bildung eines inversen Modells, können die benötigte Bewegung des Roboters berechnet werden, um eine gewünschte Veränderung zu erreichen.

Ein Modell kann das reale System meist nur annäherungsweise abbilden. Messfehler, ungenaue Roboterbewegungen, plötzliche Veränderungen der Umwelt und andere Störeinflüsse können die Lösung einer Roboteraufgabe zusätzlich negativ beeinflussen. Ein Roboterregler muss diese Faktoren auf geeignete Art und Weise ausgleichen können, um trotzdem eine erfolgreiche Reglung zu garantieren. Dazu wird häufig in einem iterativen Prozess die Abweichung zwischen bestimmten Soll- und Ist-Zuständen benutzt, um einen Roboterbefehl zu berechnen, der zur Lösung der gestellten Aufgabe führt.

Diese Diplomarbeit beschäftigt sich mit dem Zusammenspiel von Modellen und Reglern, um eine Roboteraugabe möglichst genau und schnell zu lösen. Dazu werden verschieden Modelle und Regler miteinander kombiniert und verglichen. Insbesondere werden auch adaptive Regler betrachtet, die sich an Änderungen der Umwelt zur Laufzeit anpassen können. Als Roboterregler werden hier Visual-Servoing-Regler benutzt. Anhand von Experimenten mit sechs Freiheitsgraden an einem realen Robotersystem, sowie an zwei Simulationen werden die verschiedenen Regler und Modelle sowie deren Kombinationen validiert und miteinander verglichen.

Inhaltsverzeichnis

Zusammenfassung **v**

1 Einleitung **1**
1.1 Visual Servoing 1
1.2 Projektbeschreibung 2
1.2.1 Aufbau der Diplomarbeit 2
1.2.2 Aufgabenstellung 2
1.2.3 Versuchsaufbau 3
1.3 Testverfahren 4
1.3.1 Mulitlagen-Simulation 4
1.3.2 OpenGL-Simulation 5
1.3.3 Experimente mit dem Thermo CRS F3 6

2 Modelle bei Roboterreglungen **9**
2.1 Grundlegende Begriffe 9
2.1.1 Koordinatensysteme in der Robotik 9
2.1.2 Das Kameramodell 12
2.2 Lineares Modell mit Kartesischen Koordinatensystem 13
2.2.1 Ungenauigkeiten der Jacobimatrix 14
2.3 Lineares Modell im Zylinderkoordinatensystem 15
2.4 Ein Modell zweiter Ordnung 16

3 Regler mit konstantem Dämpfungsfaktor **19**
3.1 Der Traditionelle Regler 19
3.1.1 Bestimmung der Stellgröße 19
3.1.2 Dynamische und konstante Jacobimatrix 20
3.1.3 Das Retreat-Advance-Problem 21
3.2 Regler zweiter Ordnung: Der MJP- und PMJ-Regler 21
3.2.1 Pseudoinverse des Durchschnitts der Jacobimatrizen . . 22
3.2.2 Durchschnitt der Pseudoinversen der Jacobimatrizen . 22
3.2.3 Geometrische Erklärung des Retreat-Advance-Problems 23

3.3 Ein Regler in Zylinderkoordinaten 24
3.3.1 Herleitung des Reglers 24
3.4 Validierung der Regler 25
3.4.1 Die Multilagen-Simulation 26
3.4.2 Testläufe mit dem OpenGL-Simulator 30
3.4.3 Testläufe am realen System 54
3.5 Bewertung der Regler 65

4 Adaptive Regler **67**
4.1 Minimierung nichtlinearer Ausgleichsprobleme 67
4.1.1 Nichtlineares Ausgleichsproblem 67
4.1.2 Gradientenverfahren 68
4.1.3 Gauss-Newton-Methode 69
4.2 Trust-Region-Regler 70
4.2.1 Beschreibung des Reglers 70
4.3 Der Dog-Leg-Regler 74
4.3.1 Powells Dog-Leg-Methode 74
4.3.2 Herleitung des Reglers 76
4.4 Validierung der Regler 77
4.4.1 Multilagen-Simulation 78
4.4.2 Testläufe mit dem OpenGL-Simulator 90
4.4.3 Testläufe am realen System 129
4.5 Bewertung der Regler 146
4.5.1 Der Trust-Region-Regler 146
4.5.2 Der Dog-Leg-Regler 147

5 Zusammenfassung **151**
5.1 Ergebnisse 151
5.2 Ausblick 152

A Bezeichnungen und Variablen **155**
A.1 Variablen 155
A.2 Abkürzungen 156

B Visual-Servoing-Anwendungen **157**

Abbildungsverzeichnis **158**

Tabellenverzeichnis **163**

Literatur **166**

Kapitel 1

Einleitung

1.1 Visual Servoing

Visual Servoing ist die Bezeichnung für eine Roboterreglung mit visuellem Feedback zum Erreichen einer Ziellage. Ein Beispiel dafür wäre die Annäherung eines Roboters zu einem Objekt. Man unterscheidet zwischen Image-based und Postion-based, sowie zwischen Static Look-and-Move und Dynamic Visual Servoing.

Image-based heißt, dass die Roboterreglung nur mit den Bildinformationen der Kamera arbeitet. Im Gegensatz dazu wird bei den Position-based-Ansätzen die Position des Objektes anhand der aufgenommenen Bilder berechnet. Aus der ermittelten Position wird dann der nächste Regelschritt bestimmt.

Bei den Static-Look-and-Move-Methoden führt die Roboterreglung eine Bewegung aus, hält an und nimmt erst jetzt ein neues Bild auf, um danach einen weiteren Regelschritt auszuführen. Bei den dynamischen Methoden werden die Bilder und die nächsten Regelschritte schon während der aktuellen Bewegung aufgenommen und berechnet.

Die hier benutzten Regler sind Image-based Static Look-and-Move-Regler. Das in dieser Diplomarbeit benutzte System besteht aus einem Roboterarm und einer auf dem Endeffektor montierten CCD-Kamera. Da es ein bildbasiertes System ist, wird vorher eine sogenannte Teachposition aufgenommen. Dadurch wird die gewünschte Zielposition festgelegt, die der Roboter am Ende wieder einnehmen soll.

1.2 Projektbeschreibung

1.2.1 Aufbau der Diplomarbeit

Im weiteren Verlauf des ersten Kapitels wird die Aufgabenstellung und der Versuchsaufbau erläutert. Dabei wird auch auf drei verschiedene Testverfahren eingegangen, die hier zur Validierung der Regler benutzt werden.

Im zweiten Kapitel werden Modelle zur Regelstrecke eines Robotersystems betrachtet. Nachdem einige grundlegende Definitionen und Bezeichnungen eingeführt worden sind, werden insgesamt drei Modelle vorgestellt. Eines basiert auf ein Zylinderkoordinatensystem, die anderen beiden auf kartesischen Koordinatensystemen.

Kapitel 3 beschäftigt sich mit Visual-Servoing-Regler mit konstantem Dämpfungsfaktor. Dort werden neben dem Traditionellen Regler auch zwei weitere Verfahren betrachtet, die bestimmte Schwachstellen des traditionellen Ansatzes ausgleichen. Die dort vorgestellten Regler werden mit den drei Testverfahren validiert und bewertet.

Im vierten Kapitel werden adaptive Regler betrachtet. Ausgehend von einem Trust-Region-Regler, wird dieser mit den vorher entwickelten Modellen kombiniert und validiert. Zusätzlich wird noch ein Regler basierend auf der Dog-Leg-Methode von Michael J. D. Powell [Pow70] entwickelt.

Das fünfte Kapitel liefert eine Zusammenfassung der gesammelten Ergebnisse und betrachtet weiterführende Ansätze.

1.2.2 Aufgabenstellung

In dieser Diplomarbeit wird das Zusammenspiel von Modellen und Reglern betrachtet, die eine Roboteraufgabe möglichst genau und in möglichst kurzer Zeit lösen sollen. Um das zu erreichen, werden verschiedene Systemmodelle und Regler validiert und miteinander kombiniert. Dabei spielt auch die Adaption des Modells bzw. des Reglers zur Laufzeit eine wichtige Rolle.

Es werden Visual-Servoing-Regler an einem realen System, sowie an zwei Simulationen in sechs Freiheitsgraden getestet und miteinander verglichen.

1.2.3 Versuchsaufbau

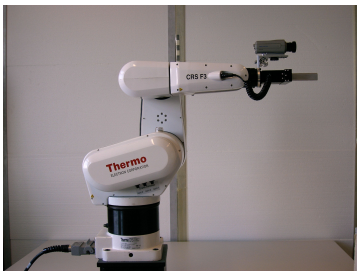


Abbildung 1.1: CRS F3

Als Roboterarm dient ein Thermo CRS F3 (siehe Abbildung 1.1). Die auf dem Endeffektor montierte Kamera ist eine Sony DFW-X710 mit einem 6,5 mm Objektiv. Die DFW-X710 ist mit der Firewireschnittstelle des Rechners verbunden, auf welchem die Visual-Servoing-Applikation läuft.

Die Anwendung benutzt dabei die Software-Bibliothek PACLib (Perception-Action-Components Library) des Lehrstuhls Kognitive Systeme. Diese Bibliothek enthält unter anderem Ansteuersoftware für die Kameras und den Robotern. Außerdem stellt sie Bildverarbeitungsroutinen und andere Tools zur Verfügung.

Aufgabe des Reglers ist es, den Roboter in eine bestimmte, vorher unbekannte Lage (Position und Orientierung) relativ zu einem Objekt zu bewegen. Dem Regler stehen nur die Bildinformationen der Kamera zur Verfügung. Dabei ist das zu erreichende Objekt mit $M \in \mathbb{N}$ Markierungen versehen (siehe Abbildung 1.2), den sogenannten *Blobs*.

Die gewünschte Ziellage wird in einem vorherigen Teachvorgang bestimmt (siehe Abbildung 1.3). Mit Bildverarbeitungsroutinen lassen sich dann die Soll-Bildmerkmale $y^\star \in \mathbb{R}^{2M}$ berechnen. Die zur Roboteransteuerung benötigte Stellgröße wird aus der Differenz von Soll-Merkmalen und Ist-Merkmalen, den sogenannten Bildfehler, berechnet. Dazu werden die $m = 2M$ Bildmerkmale zum n-ten Regelschritt paarweise in einem Messvektor $y_n \in \mathbb{R}^m$ geschrieben. Der Bildfehler ergibt sich dann durch

$$\Delta y_n := y^\star - y_n$$

Der Regler liefert zum Bildfehler Δy_n eine Stellgröße $u_n \in \mathbb{R}^6$, welche die relative Lage der Kamera so verändern soll, dass sich der Bildfehler verringert. Diese Werte werden dann durch interne Routinen an die Roboteransteuerung geschickt, welche den Greifarm mit Hilfe der inversen Kinematik in die gewünschte Lage bringt.

Zusätzlich zum echten Roboter wurden noch zwei Simulationen benutzt, welche für diese Arbeit entwickelt worden sind. Zum einen wurde eine OpenGL-Anwendung geschrieben, die eine Simulation eines Stäubli RX 90 des Lehrstuhls benutzt (siehe Abbildung 1.4), zum anderen eine sogenannte Multilagen-Simulation, welche eine großen Anzahl von Start- und Teachposen testet. Die beiden Simulatoren werden im nächsten Abschnitt genauer beschrieben.

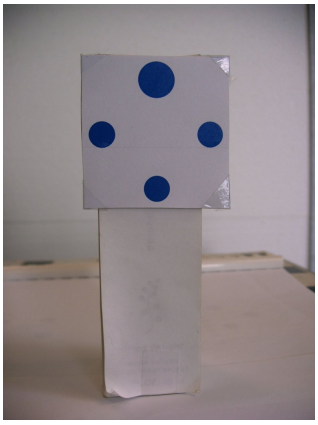


Abbildung 1.2: Objektmarkierungen

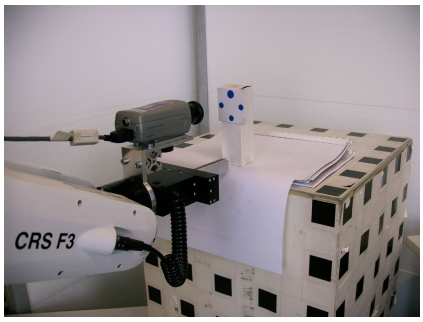


Abbildung 1.3: Aufnahme der Teachposition

1.3 Testverfahren

1.3.1 Mulitlagen-Simulation

Mit der sogenannten Mulitlagen-Simulation wird eine große Anzahl von unterschiedlichen Teach- und Startposen untersucht. Die Positionen der Bildmerkmale auf dem Kamerasensor werden direkt berechnet und nicht durch eine Bildverarbeitungsroutine bestimmt. Dadurch ist es möglich eine große Anzahl von Reglungen in relativ kurzer Zeit durchzuführen.

Bei den in dieser Arbeit durchgeführten Experimenten werden 69 463 Startposen und 29 Teachposen benutzt. Die Anwendung liefert zu dem jeweiligen Regler die Anzahl der erfolgreichen Reglungen, sowie die durchschnittliche Anzahl von Iterationsschritten, um die Teachposen zu erreichen. Ein Reglung gilt als erfolgreich, falls der Regler weniger als 100 Iterationsschritte benötigt und die Objektmarkierungen innerhalb des simulierten Kamerabildes bleiben.

Die Anwendung ist in der Lage verschiedene Kameratypen zu simulieren. Für die hier durchgeführten Experimente wurden die Werte der Sony DFW-X710 benutzt. Andere Kameras lassen sich mit Hilfe einer XML-Datei leicht definieren und einbinden.

Mit dem Multilagen-Simulator ist es auch möglich ein Gaußsches Rauschen[1] auf die Bildmerkmale zu legen. Mit der Simulation lassen sich somit

[1]Das Gaußsche Rauschen wurde mit der Polarmethode von George Marsaglia [Knu97] generiert. Ein Testlauf mit 1600 Iterationen liefert einen Erwartungswert von -0.0175266, eine Standardabweichung von 1.00365, einen Maximalwert von 3.07981 Pixel und einen minimalen Wert von -3.12589 Pixel.

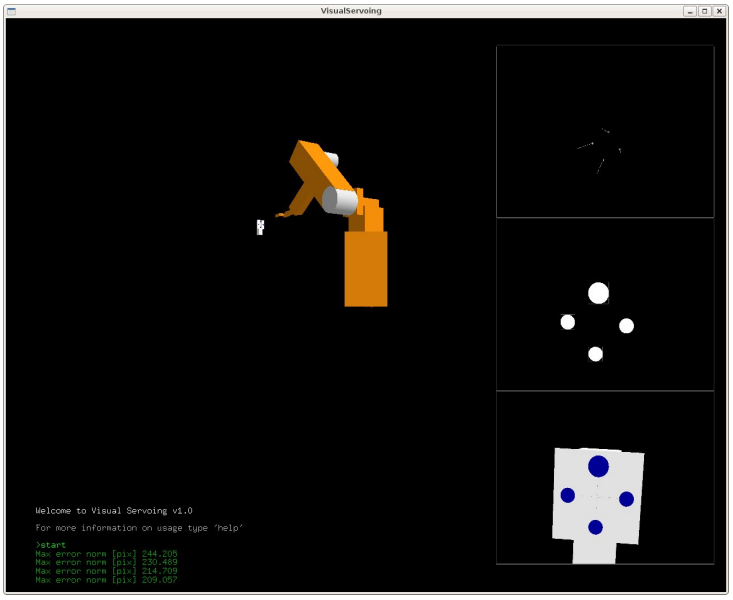


Abbildung 1.4: VisualServoing Simulator

Aussagen über die Robustheit des Reglers treffen. Durch die große Anzahl von Testlagen lassen sich mit dieser Anwendung bestimmte Parameter, wie z. B. der optimale Dämpfungsfaktor k eines Reglers experimentell bestimmen.

1.3.2 OpenGL-Simulation

Mit dem OpenGL-Simulator ist es möglich die verschiedenen Regler ohne äußere Störeinflüsse zu testen. Der Simulator benutzt einen Robotersimulation, die in der PACLib enthalten ist. Sie wurde von Andreas Jordt und Falko Kellner entwickelt. Mit Hilfe der OpenGL-API wird das Kamerabild simuliert. Bildverarbeitungsroutinen liefern die Position der Objektmarkierungen im simulierten Bild.

Es werden fünf feste Startposen betrachtet (siehe Abbildung 1.5). Dabei wird die Teachpose in Abbildung 1.5(a) benutzt.

Bei der ersten Lage in Abbildung 1.5(b) wird nur eine reine Translation von 300 mm in Richtung der z-Achse durchgeführt. Rotationen werden in dieser Lage nicht betrachtet. Die zweiten Lage (Abbildung 1.5(c)) kombiniert Rotation und Translation. Die einzelnen Objektmarkierungen liegen aber noch relativ zentral im Kamerabild. Mit der dritten Lage(Abbildung 1.5(d)) wird das Verhalten der Regler bzgl. einer reinen Rotationsänderung untersucht. In der vierten Lage (Abbildung 1.5(e)) wird wieder eine Kombination von Rotation und Translation betrachtet. Allerdings liegt hier das

Objekt nahe am Rand des Kamerabildes. Bei dieser Lage kann es somit leicht passieren, dass die Objektmarkierungen durch ungenaue Regelschritte den Bildbereich verlassen. Die fünfte und letzte Pose (Abbildung 1.5(f)) wird durch eine reine Rotation um die z-Achse um 45°erreicht. Sie wird benutzt um das in Abschnitt 3.1.3 beschriebene Retreat-Advance-Problem zu zeigen.

Aus diesen Experimenten lässt sich dann der Verlauf der Objektmarkierungen auf dem Sensor sowie die Entwicklung des Kamera- und des Bildfehlers bestimmen.

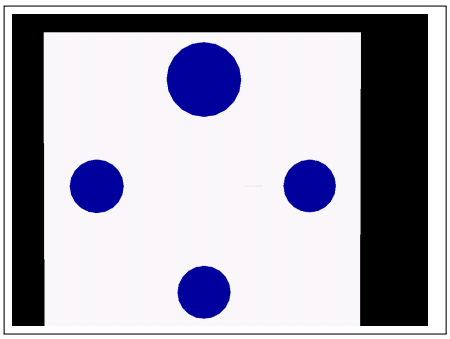
1.3.3 Experimente mit dem Thermo CRS F3

Abschließend werden die verschiedenen Regler auch auf einem realen System getestet. Dazu wird der im Abschnitt 1.2.3 beschriebene Versuchsaufbau und die fünf Lagen in Abbildung 1.5 benutzt.

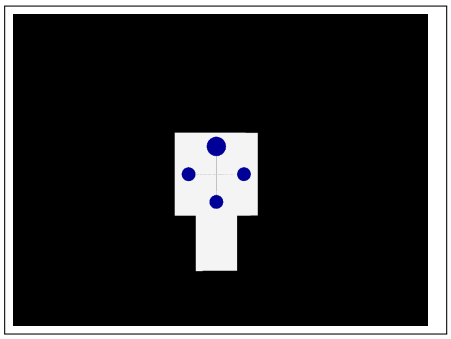
Da die CCD-Kamera keinen Autofokus besitzt, erscheinen Objekte, die eine kurze Distanz zur Kamera haben, unscharf. Das wirkt sich auch auf die Bilderkennung der Anwendung aus. Besonders bei Positionen nahe der Ziellage kann der Bildfehler unter Umständen kurzzeitig wieder steigen.

Bei den Aufnahmen der Kamerabilder kann es passieren, dass der Roboter seine neue Position noch nicht erreicht hat bzw. noch nicht vollständig zum Stehen gekommen ist. Dadurch werden falsche Daten an den Regler übergeben, die eine fehlerhafte Reglung verursachen. Deshalb werden mehrere Bilder hintereinander aufgenommen. Das aktuelle Bild wird erst dann akzeptiert, wenn der berechnete Bildfehler Δy_n mit den drei vorherigen Messungen übereinstimmt oder neun Bilder aufgenommen worden sind.

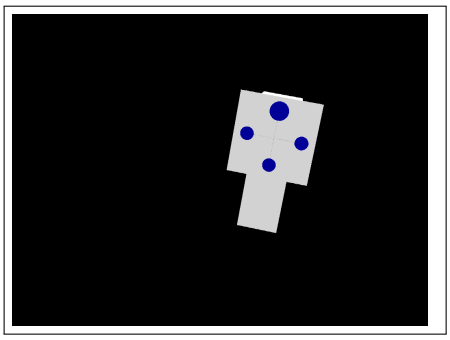
Aus den durchgeführten Experimenten werden dann die Bewegung der Objektmarkierungen auf dem Sensor und der Verlauf des Bildfehlers aufgezeichnet.



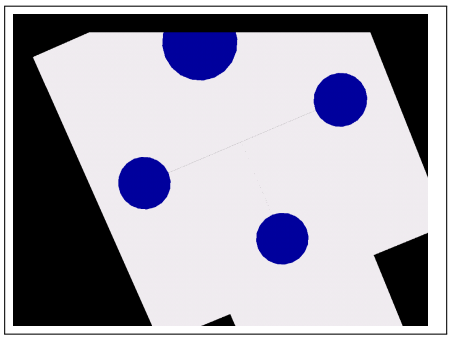
(a) Teachpose



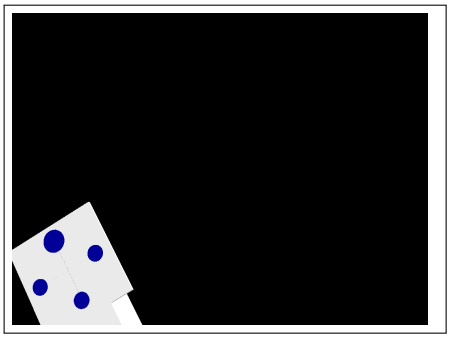
(b) Lage 1 (0, 0, -300, 0°, 0°, 0°)



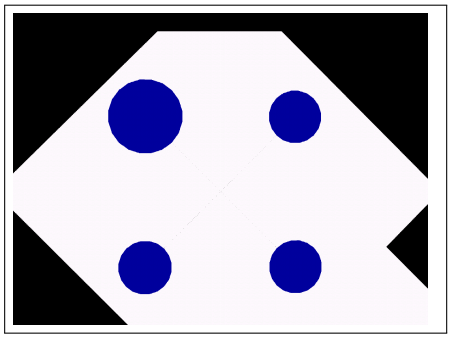
(c) Lage 2 (20, -50, -300, -10°, -10°, -10°)



(d) Lage 3 (0, 0, 0, -5°, -3°, 23°)



(e) Lage 4 (150, 90, -200, 10°, -15°, 30°)



(f) Lage 5 (0,0,0,0°,0°,45°)

Abbildung 1.5: Benutzte Testlagen bei den Experimenten

Kapitel 2

Modelle bei Roboterreglungen

2.1 Grundlegende Begriffe

Um die Pose eines Roboters im dreidimensionalen Raum beschreiben zu können, sind einige mathematische Definitionen notwendig. Hierbei werden größtenteils die Bezeichnungenen aus [Sie99] übernommen.

2.1.1 Koordinatensysteme in der Robotik

Zur Steuerung und Beschreibung eines Robotergreifarmes werden verschiedene Koordinatensysteme verwendet. In den meisten Ansätzen werden *kartesische Koordinatensysteme* benutzt. Sie bestehen aus einem Ursprung ${}^{K}O$ und den dazugehörigen orthogonalen Basisvektoren ${}^{K}x$, ${}^{K}y$, und ${}^{K}z$. Koordinatensysteme werden in dieser Arbeit mit in geschweiften Klammern geschriebenen Großbuchstaben, also z. B. mit $\{K\}$, gekennzeichnet. Der hochgestellte Buchstabe vor einer Koordinate ordnet diese einem bestimmten Koordinatensystem zu.

Beim Robotergreifarm wird zwischen dem Weltkoordinatensystem $\{W\}$, dem Toolkoordinatensystem $\{T\}$, dem Kamerakoordinatensystem $\{C\}$ sowie den zweidimensionalen Sensorkoordinatensystem $\{S\}$ und Bildkoordinatensystem $\{I\}$ unterschieden (siehe Abbildung 2.1).

Die *Lage* eines Koordinatensystems wird mit dessen Position und Orientierung beschrieben, die wie folgt definiert sind:

Definition 2.1 (Position) *Die Position eines Koordinatensystems $\{K\}$ bzgl. eines Referenzkoordinatensystems $\{K_0\}$ wird durch den Ortsvektor von ${}^{K}O$ in $\{K_0\}$ angegeben.*

Definition 2.2 (Orientierung) *Die Orientierung eines Koordinatensystems $\{K\}$ bzgl. eines Referenzkoordinatensystems $\{K_0\}$ wird mit den Win-*

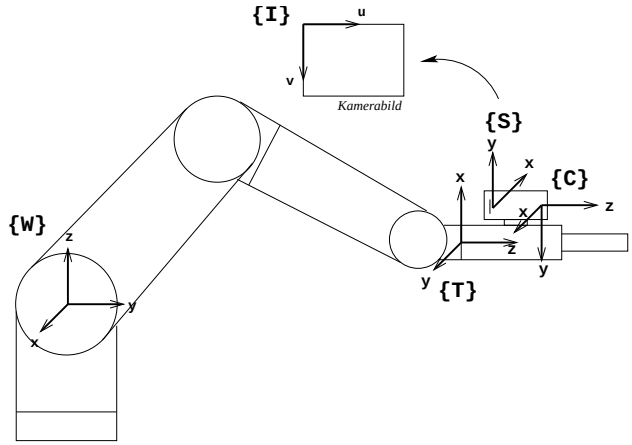


Abbildung 2.1: Koordinatensysteme (Grafik aus [Sie99])

keln α, β, $\gamma \in (-\pi, \pi]$ beschrieben. Durch rotieren von $\{K\}$ mit α um die x-Achse, β um die y-Achse und γ um die z-Achse, lässt sich $\{K\}$ mit $\{K_0\}$ gleichrichten.

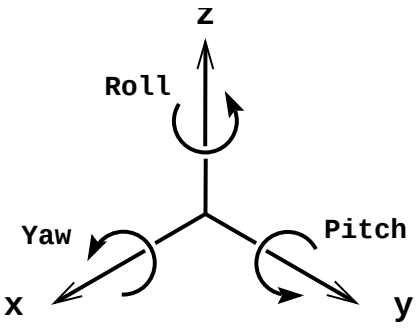


Abbildung 2.2: Yaw-, Pitch- und Roll-Winkel (Grafik aus [Sie99])

Für die drei Winkel werden die in der Literatur üblichen Bezeichnungen *Yaw*, *Pitch* und *Roll* benutzt (siehe Abbildung 2.2). Eine positive Drehung ist eine Drehung gegen den Uhrzeigersinn um die jeweiligen Achsen. Es wird dabei von einem *rechtshändigen Koordinatensystem*[1] gesprochen.

Die relative Lage eines Koordinatensystems wird also mit einem Vektor $x \in \mathbb{R}^3 \times (-\pi, \pi]^3$ beschrieben. Man spricht hier auch von sechs *Freiheitsgraden* (engl.: *six degrees of freedom* oder kurz *6 DOF*).

Als nächstes sollen auch Bewegungen von Koordinatensystemen bzgl. eines Referenzkoordinatensystems formalisiert werden. Dazu wird eine Positionsänderung als *Translation* und eine Änderung der Orientierung als *Rotation* bezeichnet. Alle Bewegungen innerhalb des Koordinatensystems lassen sich mit diesen beiden Operatoren beschreiben.

Ein Punkt $p \in \mathbb{R}^3$ wird um einen Wert von $u \in \mathbb{R}^3$ verschoben. Der neue

[1] Zeigt der Daumen der rechten Hand in Richtung einer Koordinatenachse, so zeigen die Finger in Richtung einer positiven Drehung.

Punkt p' wird durch

$$p' = p + u$$

berechnet.

Rotationen werden mit sogenannten Rotations- oder auch Drehmatrizen beschrieben. Eine Rotationsmatrix $R \in \mathbb{R}^{3\times3}$ ist eine orthonormale Matrix mit $det(R) = 1$. Es lässt sich für jede Achse eine Drehmatrix erstellen:

$$R_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix},$$

$$R_y(\beta) = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{pmatrix},$$

$$R_z(\gamma) = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Durch Hintereinanderausführung der einzelnen Matrizen wird wieder eine Drehmatrix gebildet, die alle Rotationen um die einzelnen Achsen in einer Matrix zusammenfasst:

$$R(\alpha, \beta, \gamma) = R_z(\gamma)R_y(\beta)R_x(\alpha) \tag{2.1}$$

In (2.1) wurden die einzelnen Drehmatrizen durch Linksmultiplikation kombiniert. Dadurch wird um ein statisches Koordinatensystem rotiert. Bei Rechtsmultiplikation wird um die jeweils schon gedrehten Koordinatenachsen rotiert.

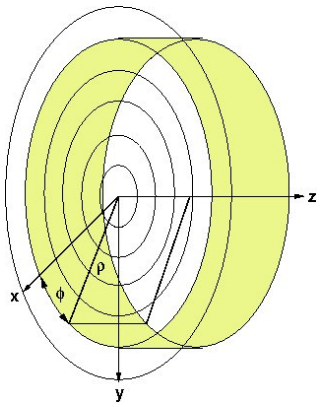


Abbildung 2.3: Zylinderkoordinatensystem

Neben dem kartesischen Koordinatensystemen werden auch Zylinderkoordinatensysteme betrachtet (siehe Abbildung 2.3). Hier bildet ein Polarkoordinatensystem die Grundfläche des Koordinatensystems. Die z-Achse zeigt wie beim kartesischen Koordinatensystem senkrecht vom Ursprung weg.

Die Umwandlung eines Punktes $(x, y, z)^T$ von kartesischen in Zylinderkoordinaten $(\rho, \phi, z)^T$ wird mit folgenden Formeln erhalten:

$$\begin{aligned} \rho &= \sqrt{x^2 + y^2} \\ \phi &= \arctan2(y, x) \\ z &= z \end{aligned} \tag{2.2}$$

2.1.2 Das Kameramodell

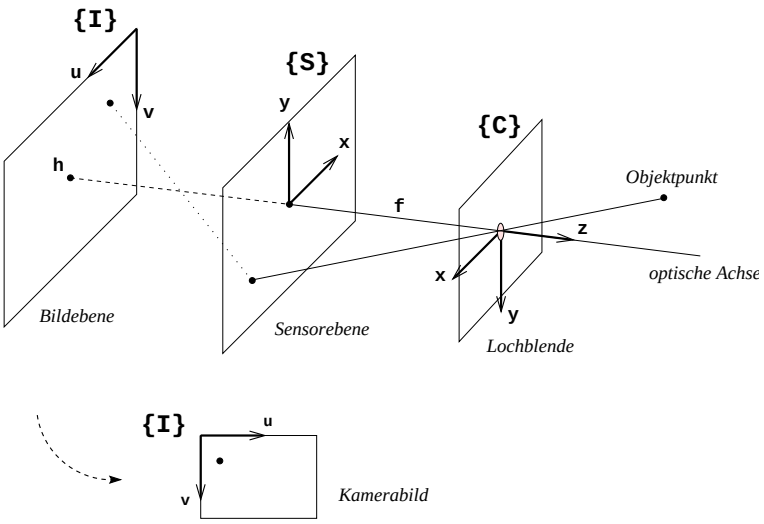


Abbildung 2.4: Lochkameramodell (Grafik aus [Sie99])

Zur Modellierung der Regelstrecke werden die Prinzipien der Lochkamera ausgenutzt (siehe Abbildung 2.4). Es lassen sich aus den Kamerakoordinaten $({}^C x, {}^C y, {}^C z)^T \in \mathbb{R}^3$ die Sensorkoordinaten $({}^S x, {}^S y)^T \in \mathbb{R}^2$ mit Hilfe der Strahlensätze folgendermaßen berechnen:

$${}^S x = \frac{{}^C x \cdot f}{{}^C z} \qquad \text{sowie} \qquad {}^S y = \frac{{}^C y \cdot f}{{}^C z} \tag{2.3}$$

Bei der Berechnung ist es notwendig, die Brennweite f der Kamera und den Abstand ${}^C z$ der M Objektmarkierungen zur Kamera zu kennen. ${}^C z$ lässt sich mit einem Verfahren über die relative Größe der Markierungen schätzen.

2.2 Lineares Modell mit Kartesischen Koordinatensystem

Mit Hilfe des Lochkameramodells wird ein lineares Modell der Regelstrecke gebildet. In diesem Abschnitt wird die analytische Herleitung aus [Sie99] betrachtet.

Es wird mit $x_n \in \mathbb{R}^6$ die momentane Pose des Roboters und mit $y_n \in \mathbb{R}^m$ die zugehörigen Positionen der Objektmarkierungen zum Zeitpunkt $n \in \mathbb{N}_0$ auf dem Sensor der CCD-Kamera bezeichnet. Die Funktion $\vartheta : \mathbb{R}^6 \to \mathbb{R}^m$ liefert die Bildmerkmale y_n zu einer Pose x_n.

Mit $u_n \in \mathbb{R}^6$ wird die sogenannte *Stellgröße* bezeichnet. Sie gibt die Lageänderung der momentanen Pose x_n an. Die neue Pose x_{n+1} lässt sich mit einer Funktion $\varphi : \mathbb{R}^6 \times \mathbb{R}^6 \to \mathbb{R}^6$ mit $\varphi(x_n, u_n) = x_{n+1}$ berechnen.

Insgesamt wird also eine Funktion $\Phi : \mathbb{R}^6 \times \mathbb{R}^6 \to \mathbb{R}^m$ mit $\Phi(x_n, u_n) = \vartheta(\varphi(x_n, u_n)) = y_{n+1}$ gesucht. Mit ihr wird für eine Stellgröße u_n und den momentanen Zustand x_n die Position der Bildmerkmale y_{n+1} auf dem Sensor nach Durchführung des Regelschrittes berechnet. Es wird im Folgenden x_n fest gewählt und $\Phi_n(u) = \Phi(x_n, u)$ gesetzt.

Sei weiter ${}^C p = (p_1, p_2, p_3)^T$ die Position einer Objektmarkierung im Kamerakoordinatensystems des Roboters und $u = (u_1, u_2, u_3, u_4, u_5, u_6)^T \in \mathbb{R}^6$ die auszuführende Stellgröße. Die neue Position p' der Objektmarkierung wird wie folgt berechnet:

$$\begin{aligned} p' &= R_x(-u_4) R_y(-u_5) R_z(-u_6) \begin{pmatrix} p_1 - u_1 \\ p_2 - u_2 \\ p_3 - u_3 \end{pmatrix} \\ &= \begin{pmatrix} c_5 c_6 & c_5 s_6 & -s_5 \\ s_4 s_5 c_6 - c_4 s_6 & s_4 s_5 s_6 + c_4 c_6 & s_4 c_5 \\ c_4 s_5 c_6 + s_4 s_6 & c_4 s_5 s_6 - s_4 c_6 & c_4 c_5 \end{pmatrix} \begin{pmatrix} p_1 - u_1 \\ p_2 - u_2 \\ p_3 - u_3 \end{pmatrix} \end{aligned} \tag{2.4}$$

mit

$$s_i := \sin(u_i), c_i := \cos(u_i) \text{ für } i = 4, 5, 6.$$

Da sich hier das Kamerakoordinatensystem und nicht die Markierungen bewegen, wurde die Rotationsmatrix invertiert.

Nun wird (2.3) in (2.4) eingesetzt. Dadurch ergibt sich für $\Phi_n(u)$

$$\begin{aligned} \begin{pmatrix} {}^S x' \\ {}^S y' \end{pmatrix} &= \Phi_n(u) \\ &= f \cdot \begin{pmatrix} \frac{c_5 c_6 (p_1 - u_1) + c_5 s_6 (p_2 - u_2) - s_5 (p_3 - u_3)}{(c_4 s_5 c_6 + s_4 s_6)(p_1 - u_1) + (c_4 s_5 s_6 - s_4 c_6)(p_2 - u_2) + c_4 c_5 (p_3 - u_3)} \\ \\ \frac{(s_4 s_5 c_6 - c_4 s_6)(p_1 - u_1) + (s_4 s_5 s_6 + c_4 c_6)(p_2 - u_2) + s_4 c_5 (p_3 - u_3)}{(c_4 s_5 c_6 + s_4 s_6)(p_1 - u_1) + (c_4 s_5 s_6 - s_4 c_6)(p_2 - u_2) + c_4 c_5 (p_3 - u_3)} \end{pmatrix} \end{aligned}$$

Die Funktion Φ_n wird im nächsten Schritt mit Hilfe einer Taylorentwicklung erster Ordnung an der Stelle $\Phi_n(0+u)$ für u mit kleinem $\|u\|_2$ approximiert:

$$\begin{aligned} y_{n+1} &= \Phi_n(0) + \Phi_n'(0)u + O(\|u\|^2) \\ &= y_n + J_{\Phi_n}(0)u + O(\|u\|^2) \end{aligned}$$

wobei $O(\|u\|^2)$ der Rest der Taylorreihe und $J_{\Phi_n}(0) = J_k$ mit

$$J_k = \begin{pmatrix} -\frac{f}{C_z} & 0 & \frac{{}^S x}{C_z} & \frac{{}^S x\,{}^S y}{f} & -f - \frac{{}^S x^2}{f} & {}^S y \\ 0 & -\frac{f}{C_{z_1}} & \frac{{}^S y_1}{C_{z_1}} & f + \frac{{}^S y^2}{f} & -\frac{{}^S x\,{}^S y}{f} & -{}^S x \end{pmatrix} \tag{2.5}$$

die Jacobimatrix von Φ_n für eine Objektmarkierung ist. Sie wird *Bildjacobimatrix* genannt. Für kleine $\|u\|_2$ wird folgendes lineare Modell der Regelstrecke erhalten :

$$\begin{aligned} y_{n+1} - y_n &\approx J_n u \\ &= \begin{pmatrix} -\frac{f}{C_{z_1}} & 0 & \frac{{}^S x_1}{C_{z_1}} & \frac{{}^S x_1\,{}^S y_1}{f} & -f - \frac{{}^S x_1^2}{f} & {}^S y_1 \\ 0 & -\frac{f}{C_{z_1}} & \frac{{}^S y_1}{C_{z_1}} & f + \frac{{}^S y_1^2}{f} & -\frac{{}^S x_1\,{}^S y_1}{f} & -{}^S x_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -\frac{f}{C_{z_M}} & 0 & \frac{{}^S x_M}{C_{z_M}} & \frac{{}^S x_M\,{}^S y_M}{f} & -f - \frac{{}^S x_M^2}{f} & {}^S y_M \\ 0 & -\frac{f}{C_{z_M}} & \frac{{}^S y_M}{C_{z_M}} & f + \frac{{}^S y_M^2}{f} & -\frac{{}^S x_M\,{}^S y_M}{f} & -{}^S x_M \end{pmatrix} \begin{pmatrix} u_1 \\ \vdots \\ u_6 \end{pmatrix}. \end{aligned} \tag{2.6}$$

Neben diesem analytische Ansatz aus [Sie99], wird häufig in der Literatur, wie z. B. in [HHC96], das Modell mit Hilfe einer Ableitung von (2.3) nach der Zeit gebildet.

2.2.1 Ungenauigkeiten der Jacobimatrix

Wie vorher schon erwähnt, handelt es sich bei der Jacobimatrix um ein lineares Modell der Regelstrecke. Somit ist J_n nur eine lineare Annährung an die tatsächliche Regelstrecke. Außerdem werden die Parameter der Matrix z. T. nur geschätzt, wie z. B. der Abstand des Objektes zur Kamera. Hinzu kommt, dass dieses Modell Stellgrößen u mit hinreichend kleinem $\|u\|_2$ betrachtet. Es wird also nicht die gesamte Wirklichkeit, sondern nur ein lokaler Bereich abgebildet.

Alle diese Faktoren führen dazu, dass es zu einigen Ungenauigkeiten kommen kann. Diese müssen dann auf geeignete Art und Weise durch die Regler unter Kontrolle gebracht werden.

2.3 Lineares Modell im Zylinderkoordinatensystem

In diesem Abschnitt wird ein lineares Modell bzgl. eines Zylinderkoordinatensystems entwickelt. Dieser Ansatz wird in [IO05] verfolgt.

Die Positionen der Objektmarkierungen auf dem CCD-Sensor werden in Polarkoordinaten angegeben, also durch $(\rho, \phi)^T \in \mathbb{R} \times (-\pi, \pi]$, wobei ρ und ϕ wie in (2.2) gewählt sind.

Im kartesischen Koordinatensystem sind die x- und y-Achse zueinander invariant. Dadurch sind Positionen auch nach Verschiebung des Mittelpunktes eindeutig zugeordnet. Anders verhält es sich beim Polarkoordinatensystem. Hier variieren die ρ- und ϕ-Koordinate bei der Verschiebung des Mittelpunktes. Im Folgenden geben $\xi, \eta \in \mathbb{R}$ die Verschiebung des Mittelpunktes des kartesischen Koordinatensystems an. Für einen Punkt $(x, y)^T \in \mathbb{R}^2$ ergibt sich dann im neuen Koordinatensystem

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x - \xi \\ y - \eta \end{pmatrix} \tag{2.7}$$

Sei $(\rho, \phi)^T$ eine Objektmarkierung im Polarkoordinatensystem des CCD-Sensors. Leitet man nun diesen Punkt nach der Zeit ab, so ergibt sich

$$\begin{pmatrix} \dot{\rho} \\ \dot{\phi} \end{pmatrix} = U_i(\phi) \begin{pmatrix} {}^S\dot{x}' \\ {}^S y' \end{pmatrix} = U_i(\phi) \begin{pmatrix} {}^S\dot{x} \\ {}^S y \end{pmatrix}$$

mit

$$\begin{aligned} U_i(\phi) &= \begin{pmatrix} \cos(\phi) & \sin(\phi) \\ -\sin(\phi) & \cos(\phi) \end{pmatrix} \\ \begin{pmatrix} \dot{\rho} \\ \dot{\phi} \end{pmatrix} &= \begin{pmatrix} \frac{d\rho}{dt} \\ \rho \frac{d\phi}{dt} \end{pmatrix}, \end{aligned}$$

wobei ρ ein Skalierungsfaktor der ϕ-Komponente im Polarkoordinatensystem ist und $i = 1, \ldots, M$. $({}^S x', {}^S y')^T$ und $({}^S x, {}^S y)^T$ sind wie in (2.7) gewählt. $U_i(\phi)$ ist eine Rotationsmatrix im zweidimensionalen Raum.

Wir erhalten also die Änderung eines Bildmerkmals auf dem Kamerasensor bzgl. eine Stellgröße u wie folgt:

$$\begin{aligned} \begin{pmatrix} \dot{\rho} \\ \dot{\phi} \end{pmatrix} &= U_i(\phi) J' u' \\ &= U_i(\phi) J u \\ &= \tilde{J} u \end{aligned}$$

wobei J' die Bildjacobimatrix mit den um η und ξ verschobenen Objektpunkt ist. u' unterscheidet sich von u, da sich durch die Verschiebung des Ursprungs die Rotationsachsen geändert haben. $\tilde{J}$ ist die mit $U_i(\phi)$ multiplizierte Jacobimatrix J. Es ist

$$\tilde{J} = \begin{pmatrix} \tilde{J}_{\text{tran}} & \tilde{J}_{\text{rot}} \end{pmatrix}$$

mit

$$\tilde{J}_{\text{tran}} = \begin{pmatrix} -\frac{fc}{{}^C z} & -\frac{fs}{{}^C z} & \frac{{}^C ys + {}^C xc}{{}^C z} \\ \frac{fs}{{}^C z} & -\frac{fc}{{}^C z} & \frac{{}^C yc + {}^C xs}{{}^C z} \end{pmatrix}$$

$$\tilde{J}_{\text{rot}} = \begin{pmatrix} (f + \frac{{}^C y^2}{f})s + \frac{{}^C x {}^C yc}{f} & (-f - \frac{{}^C x^2}{f})c - \frac{{}^C x {}^C ys}{f} & {}^C yc - {}^C xs \\ (f + \frac{{}^C y^2}{f})c - \frac{{}^C x {}^C ys}{f} & (f + \frac{{}^C x^2}{f})s - \frac{{}^C x {}^C yc}{f} & -{}^C ys - {}^C xc \end{pmatrix}$$

$s = \sin(\phi)$ und $c = cos(\phi)$.

Das Modell lässt sich wieder auf M Objektmarkierungen erweitern. Man wählt die Jacobimatrix wie in (2.6) und setzt

$$U = \begin{pmatrix} U_1(\phi) & & 0 \\ & \ddots & \\ 0 & & U_M(\phi) \end{pmatrix}.$$

Die Jacobimatrix für Zylinderkoordinaten ergibt sich dann durch

$$\tilde{J} = UJ \tag{2.8}$$

2.4 Ein Modell zweiter Ordnung

In [Mal04] wird ein ein Modell zweiter Ordnung gebildet. Es wird wieder die Funktion Φ betrachtet. Sei x_n die momentane Pose des Roboters und $x^\star$ der Zustand während der Teachpose. Für eine Stellgröße u wird $\Phi_n(u) = \Phi(x_n, u)$ und $\Phi^\star(u) = \Phi(x^\star, u)$ gesetzt, so dass $\Phi_n(0) = y_n$ und $\Phi^\star(0) = y_{n+1}$.

Die beiden Funktionen werden diesmal mit einer Taylorentwicklung zweiter Ordnung approximiert. Es ergibt sich

$$\Delta y_n = \Phi_n'(0)u + \frac{1}{2}u^T\Phi_n''(0)u + O_{\Phi_n}(\|u\|^3) \tag{2.9}$$

$$\Delta y_n = \Phi^{\star'}(0)u - \frac{1}{2}u^T\Phi^{\star''}(0)u + O_{\Phi^\star}(\|u\|^3) \tag{2.10}$$

Einer Taylorentwicklung erster Ordnung von Φ_n' an der Stelle $u - u = 0$ liefert

$$\begin{aligned} &\Leftrightarrow \Phi_n'(0) = \Phi^{\star'}(0) - u^T\Phi^{\star''}(0) + O_{\Phi_n'}(\|u\|^2) \\ &\Leftrightarrow \Phi^{\star''}(0) = \Phi^{\star'}(u) - \Phi_n'(0) - O_{\Phi_n'}(\|u\|^2) \end{aligned} \tag{2.11}$$

Durch einsetzen von (2.11) in (2.10) wird folgende Gleichung erhalten:

$$\begin{aligned}\Delta y_n &= \frac{1}{2}(\Phi'_n(0) + \Phi^{\star'}(0))u + O_{\mathrm{PMJ}}(\|u\|^3) \\ &= \frac{1}{2}(J_n + J^\star)u + O_{\mathrm{PMJ}}(\|u\|^3)\end{aligned}$$

mit

$$O_{\mathrm{PMJ}}(\|u\|^3) = O_{\Phi^\star}(\|u\|^3) + O_{\Phi'_n}(\|u\|^2)u.$$

J_n ist die Jacobimatrix zum Zeitpunkt n aus (2.5). Sie wird auch dynamische Jacobimatrix genannt. Mit $J^\star$ wird die sogenannte konstante Jacobimatrix bezeichnet. Sie ist genauso aufgebaut wie J_n. Allerdings werden statt den aktuellen Werten der Bildmerkmale, die Werte der Teachpose bentutzt. Insgesamt ergibt sich also

$$\Delta y_n \approx \frac{1}{2}(J_n + J^\star)u \tag{2.12}$$

Es wurde somit ein quadratisches Modell der Regelstrecke mit Hilfe von linearen Modellen hergeleitet.

Kapitel 3

Regler mit konstantem Dämpfungsfaktor

In diesem Kapitel werden die berechneten Stellgrößen durch einen konstanten Dämpfungsfaktor reguliert. Die Stellgrößen werden dazu mit diesem Faktor multipliziert. Dieses Vorgehen ähnelt dem gedämpften Gauß-Newton-Verfahren oder auch dem Gradientenabstieg. Auch hier werden konstante Werte zur Dämpfung benutzt. Dadurch soll eine Konvergenz sichergestellt werden. Zur Herleitung der Regler werden die Modelle aus Kapitel 2 benutzt.

3.1 Der Traditionelle Regler

Der in diesem Abschnitt beschriebene Traditionelle Regler gehört zu den klassischen Ansätzen beim Visual Servoing. Er wird z. B. in [Sie99] oder auch in den Standardwerken von Chaumette in [HHC96] und [CH06] beschrieben.

3.1.1 Bestimmung der Stellgröße

Mit dem in Kapitel 2 entwickelten linearen Modell der Regelstrecke im kartesischen Koordinatensystem soll im Folgenden der sogenannte Traditionelle Regler entworfen werden.

Nach dem Festlegen von $y^\star$ im Teachvorgang soll der Regler, ausgehend von einer Startpose $x_0 \in \mathbb{R}^6$, eine Lage $x_N \in \mathbb{R}^6, N \in \mathbb{N}$ erreichen, die der Teachpose ähnelt. Es soll also für ein $\varepsilon > 0$ folgendes gelten:

$$\|\Delta y_n\|_\infty = \|y^\star - y_N\|_\infty < \varepsilon \tag{3.1}$$

Bei einem momentanen Bildfehler von Δy_n, wäre es wünschenswert, wenn sich die Bildmerkmale y_n durch u_n um $-\Delta y_n$ ändern würden. Es ergibt sich

also folgendes Minimierungsproblem:

$$u_n \in \underset{u \in \mathcal{U}(x_n)}{\operatorname{argmin}} \|\Delta y_n + J_n u\|_2^2$$

Dabei bezeichnet $\mathcal{U}(x_n)$ die Menge der zulässigen Stellgrößen zu einem Zustand $x_n \in \mathbb{R}^6$. Eine Stellgröße heißt zulässig, falls durch sie das Objekt nicht aus dem Sichtbereich der Kamera gerät und die Bewegung durch den Roboter ausführbar ist. Ohne diese Beschränkung auf zulässige Stellgrößen ergibt sich

$$u_n = J_n^+(-\Delta y_n).$$

Mit J_n^+ wird die Pseudoinverse der Bildjacobimatrix bezeichnet. In der Implementierung des Systems gibt es vier Markierungen, also $m = 2 \cdot 4 = 8$ Bildmerkmale. Damit ist $J_n \in \mathbb{R}^{8\times 6}$. Man kann zeigen, dass J_n Maximalrang besitzt, also rang $J_n = 6$. Daraus ergibt sich für die Pseudoinverse $J_n^+ \in \mathbb{R}^{6\times 8}$

$$J_n^+ = (J_n^T J_n)^{-1} J_n^T.$$

Die Durchführung des Regelschrittes ohne Beschränkung auf zulässige Stellgrößen wird auch *voller Gauß-Newton-Schritt zur Minimierung des Bildfehlers* genannt und wird definiert durch

$$\Delta u_n = J_n^+(-\Delta y_n).$$

Wie beim *gedämpften Gauß-Newton-Verfahren* (siehe Abschnitt 4.1.3) wird hier ein sogenannter Dämpfungsfaktor $k \in \mathbb{R}$ mit $0 < k \leq 1$ benutzt, um die Konvergenz des Reglers sicherzustellen. Man erhält

$$u_n = k \cdot \Delta u_n = k \cdot J_n^+(-\Delta y_n).$$

Der Faktor k wird in den späteren Experimenten mit Hilfe der Multilagen-Simulation bestimmt. Siebel hat in seinen Experimenten in [Sie99] einen Wert von $k = 0.1$ ermittelt. Dieser deckt sich mit den Ergebnissen, die in der vorliegenden Diplomarbeit gemacht worden sind.

3.1.2 Dynamische und konstante Jacobimatrix

Wie schon in Abschnitt 2.4 angedeutet, existieren bei der Wahl der Parameter zur Berechnung der Jacobimatrix zwei Ansätze. Beim ersten Ansatz wählt man die Werte der momentanen Bildmerkmale des Objekts. Dadurch muss nach jedem Regelschritt die Matrix neu berechnet werden. Die so bestimmte Jacobimatrix wird im Folgenden *dynamische Bildjacobimatrix* genannt und mit J_n bezeichnet.

Der zweite Ansatz benutzt die Werte der Teachpose als Eingabe. In diesem Fall muss die Matrix nur einmal am Anfang der Reglung berechnet werden. Dementsprechend wird diese Jacobimatrix $J^{\star}$ als *konstante Bildjacobimatrix* bezeichnet.

In den späteren Experimenten wird sich zeigen, dass die dynamische Bildjacobimatrix die besseren Ergebnisse liefert. Durch die Neuberechnung nach jedem Regelschritt, kann sich das lineare Modell besser dem System anpassen.

3.1.3 Das Retreat-Advance-Problem

In [CH01] und [Cha98] wird das sogenannte *Retreat-Advance-Problem* oder auch *Chaumette Conundrum* beschrieben. Dazu wird die Startpose in Abbildung 3.1 betrachtet. Diese entspricht der fünften Lage der in Abschnitt 1.3 beschriebenen Testverfahren.

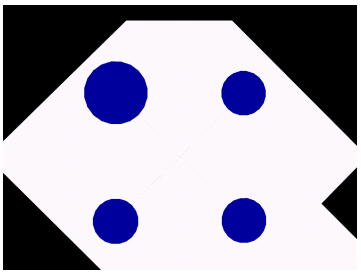


Abbildung 3.1: Reine Rotation um z-Achse

Ausgehend von der Teachpose in Abbildung 1.5(a) wurde eine Drehung um die z-Achse ausgeführt. Startet man von dieser Lage aus den Traditionellen Regler, so läßt sich eine Bewegung entlang der z-Achse beobachten. Bei der konstanten Jacobimatrix bewegt sich der Roboter auf das Objekt zu, bei der dynamischen vom Objekt weg. Führt man eine Roll-Bewegung um $\pm\pi$ aus, so findet eine reine Bewegung entlang der z-Achse statt. Dadurch ist es möglich, dass der Roboter mit dem Objekt kollidiert oder einen nicht zulässigen Zustand erreicht.

Dieses Phänomen beruht auf der Invarianz der dritten und sechsten Spalte der Jacobimatrix [Cha98]. Die beiden folgenden Regler lösen genau dieses Problem des Traditionellen Reglers.

3.2 Regler zweiter Ordnung: Der MJP- und PMJ-Regler

Die in diesem Abschnitt vorgestellten Regler werden in [Mal04] entwickelt, um das vorher beschriebene Retreat-Advance-Problem zu lösen.

3.2.1 Pseudoinverse des Durchschnitts der Jacobimatrizen

Der sogenannte PMJ-Regler (engl.: *Pseudo-inverse of the Mean of the Jacobians*) benutzt das Modell zweiter Ordnung in (2.12). Mit Hilfe der Pseudoinversen und einem Dämpfungsfaktor $0 < k \leq 1$ lässt sich die Stellgröße wie folgt bestimmen:

$$u = k(\frac{1}{2}(J_n + J^\star))^+(-\Delta y_n) \tag{3.2}$$

3.2.2 Durchschnitt der Pseudoinversen der Jacobimatrizen

Zur Herleitung des MJP-Reglers (engl.: *Mean of the Jacobian Pseudo-inverse*) werden die Durchschnitte der Pseudoinversen der beiden Jacobimatrizen betrachtet. Zur Herleitung werden (2.9) und (2.10) benutzt. (2.9) wird mit J_n^+ und (2.10) mit $J^{\star+}$ multipliziert. Es ergibt sich:

$$u = J_n^+\Delta y + \frac{1}{2}J_n^+u^T\Phi_n''(0)u + O_{\Phi_n}(\|u\|^3) \tag{3.3}$$

$$u = J^{\star+}\Delta y - \frac{1}{2}J^{\star+}u^T\Phi^{\star''}(0)u + O_{\Phi^\star}(\|u\|^3) \tag{3.4}$$

An $J_n^+u^T\Phi_n''(0)$ wird nun eine Taylorentwicklung erster Ordnung durchgeführt.

$$J_n^+u^T\Phi_n''(0) = J^{\star+}u^T\Phi^{\star''}(0) + O_{J_n^+}(\|u\|^2) \tag{3.5}$$

Durch bilden des Durchschnitts von (3.3) und (3.4) und durch einsetzen von (3.5) ergibt sich dann:

$$u = \frac{1}{2}(J_n^+ + J^{\star+})\Delta y + O_{\mathrm{MJP}}(\|u\|^3)$$

mit

$$O_{\mathrm{MJP}}(\|u\|^3) = O_{\Phi_n}(\|u\|^3) + O_{\Phi^\star}(\|u\|^3) + O_{J_n^+}(\|u\|^2)u$$

Mit einem konstanten Dämpfungsfaktor $0 < k \leq 1$ wird durch folgende Annährung die Stellgröße berechnet:

$$u = \frac{1}{2}k(J_n^+ + J^{\star+})(-\Delta y)$$

Auch für diesen Regler zweiter Ordnung wurden nur die beiden linearen Modelle J_n und $J^\star$ benutzt. Allerdings wurde hier der Durchschnitt von den Pseudoinversen der Jacobimatrizen betrachtet.

3.2.3 Geometrische Erklärung des Retreat-Advance-Problems

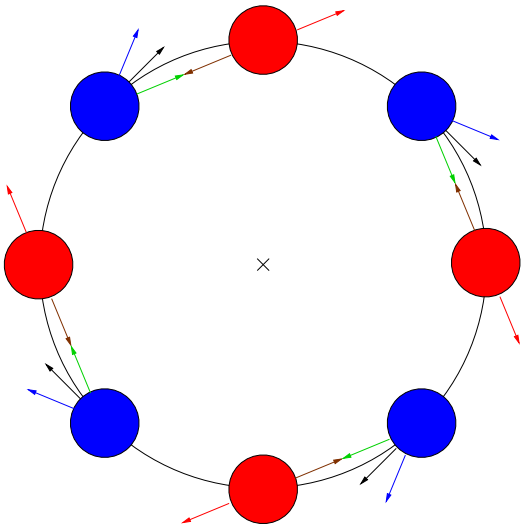


Abbildung 3.2: Gemometrische Interpretation des Retreat-Advance-Problems (Grafik nach [CH06])

Hier wird das Retreat-Advance-Problem sowie die Verbesserung durch den PMJ- und den MJP-Regler mit Hilfe einer geometrische Interpretation aus [CH06] erklärt.

In Abbildung 3.2 wird mit den roten Kreisen die Teachposition und mit den blauen die Startpostion der Objektmarkierungen gekennzeichnet. Hier wurde eine reine Rotation um die z-Achse durchgeführt. Ausgehend von dieser Lage soll jetzt wieder die Teachposition angesteuert werden.

Der Traditionelle Regler mit konstanter bzw. mit dynamischer Jacobimatrix ist bestrebt eine geradlinige Bewegung in Richtung ihrer Ziellage durchzuführen. Der Regler mit der dynamischen also von der Startposition zur Teachposition (grüne Pfeile) und die konstante von der Teachposition zur Startposition (braune Pfeile).

Um eine direkte Bewegung in Richtung der Teachpose durchführen zu können (siehe dazu Abbildung 3.28(a) in den späteren Experimenten), muss der Traditonelle Regler mit dynamischer Bildjacobimatrix sich vom Objekt wegbewegen.

Der Regler mit konstanter Jacobimatrix versucht virtuell die Teachpose zur Startpose hinzubewegen (braune Pfeile). Als Eingabe bekommt er aber den Bildfehler der Startpose zur Teachpose, was genau dem negativen Bildfehler von der Teachpose zur Startpose entspricht. Dadurch zeigt die berech-

nete Stellgröße genau in die entgegengesetzte Richtung (rote Pfeile). Wird diese Stellgröße zur Ansteuerung des Roboters benutzt, so beschreiben die Objektmarkierungen eine Bewegung in Richtung der blauen Pfeile.Um diese Bewegung durchführen zu können (siehe Abbildung 3.27(a) in den späteren Experimenten), muss sich der Regler auf das Objekt zubewegen.

Betrachtet man jetzt z. B. den MJP-Regler, so wird hier der Durchschnitt der beiden Bewegungen gebildet. Der entspricht dann den schwarzen Pfeilen. Die Bewegung durch den Regler ist eine tangentiale Bewegung, mit der eine Kreisbewegung angenähert wird. Es wird somit keine Translation entlang der z-Achse durchgeführt. Das Verhalten lässt sich im Abschnitt 3.4.2 in Abbildung 3.29(a) beobachten.

3.3 Ein Regler in Zylinderkoordinaten

Dieser Regler wird in [IO05] beschrieben. Es wird das lineare Modell der Regelstrecke in Zylinderkoordinaten aus Abschnitt 2.3 benutzt.

3.3.1 Herleitung des Reglers

Wie beim Traditionellen Regler soll auch hier der Bildfehler minimiert werden. Dieser wird hier allerdings anders definiert:

$$e_i = \begin{pmatrix} \rho - \rho^\star \\ \rho(\phi - \phi^\star) \end{pmatrix}$$

für die i-te Objektmarkierung. Dabei ist $(\rho, \phi)^T$ die momentane Position und $(\rho^\star, \phi^\star)$ die Teachposition der Objektmarkierung. Die Stellgröße u ergibt sich dann durch

$$u = k\tilde{J}^+(-e),$$

wobei $\tilde{J}^+$ die Pseudoinverse der in (2.8) definierten Jacobimatrix ist. e enthält die paarweise in einen Vektor geschriebenen Bildfehler der einzelnen Objektmarkierungen. Wie in Abschnitt 2.3 beschrieben, ist es wichtig, wo sich der Mittelpunkt des Polarkoordinatensystems befindet. Die Wahl des Ursprungs wird im nächsten Abschnitt beschrieben.

Berechnung des Ursprungs des Koordinatensystems

Es wird eine Rotationsache gesucht, mit der man die Startpose durch eine reine Rotation in die Teachpose überführen kann. Der Schnittpunkt dieser Achse mit der Sensorebene beschreibt dann den neue Ursprung des Koordinatensystems. Diese Methode wird in [Kan96] beschrieben.

Zunächst wird der sogenannten *N-Vektor* durch

$$m = N(\begin{pmatrix} x \\ y \end{pmatrix}) = \frac{1}{\sqrt{x^2+y^2+1}} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$$

für einen Punkt $(x, y)^T \in \mathbb{R}^2$ in homogenen Koordinaten definiert.

Sei nun m_{start} der N-Vektor der Startpose und m_{teach} der der Teachpose. Sei weiter $R = VU^T$ eine Rotationsmatrix. Um den neuen Ursprung zu finden, wird folgendes nichtlineares Ausgleichsproblem minimiert:

$$E(R) = \sum_{i=1}^{n} \|m_{teach} - m_{start}\|^2$$

Die orthogonalen Matrizen U und V ergeben sich durch folgende Singulärwertzerlegung:

$$M = \sum_{i=1}^{n} m_{teach} m_{start}^T = V \Lambda U^T$$

Aus der so berechneten Rotationsmatrix läßt sich dann wie folgt die gesuchte Rotationsachse berechnen:

$$l = \begin{pmatrix} l_x \\ l_y \\ l_z \end{pmatrix} = N\left(\begin{pmatrix} R_{32} - R_{23} \\ R_{13} - R_{31} \\ R_{21} - R_{21} \end{pmatrix}\right)$$

Nun wird noch der Schnittpunkt der Rotationsachse l mit dem CCD-Sensor bestimmt:

$$p_o = \begin{pmatrix} x_o \\ y_o \end{pmatrix} = \begin{pmatrix} \frac{l_x}{l_z} \\ \frac{l_y}{l_z} \end{pmatrix}$$

Die beiden Parameter zu Verschiebung des Ursprungs werden auf $(\eta, \xi) = (x_o, y_o)$ gesetzt.

Ist $|l_z| < \delta$ für eine kleine Zahl $\delta > 0$, so liegt die Rotationsachse nahezu parallel zum Sensor. Die beiden Werte η und ξ streben in diesem Fall gegen ∞. Dabei nähert sich das Polarkoordinatensystem dem kartesischen Koordinatensystem an, so dass dann die Jacobimatrix in (2.6) benutzt wird.

3.4 Validierung der Regler

In diesem Abschnitt sollen die Regler mit konstantem Dämpfungsfaktor validiert werden. Dazu werden die Testverfahren aus Abschnitt 1.3 benutzt.

Ein Regler hat seine Teachpose erreicht, falls die Bedingung in (3.1) für $\varepsilon = 2\,\text{Pixel}$ erfüllt ist. Mit diesem Wert schafft es das reale Robotersystem auch mit den in Abschnitt 1.3.3 beschriebenen Ungenauigkeiten die Ziellagen einzunehmen. Für den Regler in Zylinderkoordinaten wird für δ der Wert 0.05 verwendet.

3.4.1 Die Multilagen-Simulation

Mit der Multilagen-Simuation sollen für die einzelnen Regler die optimalen Dämpfungsfaktoren bestimmt werden. Dadurch sollen die Regler möglichst schnell und genau die Zielposition erreichen. Außerdem soll ihre Robustheit gegen Störungen festgestellt werden

Traditioneller Regler mit konstanter Jacobimatrix

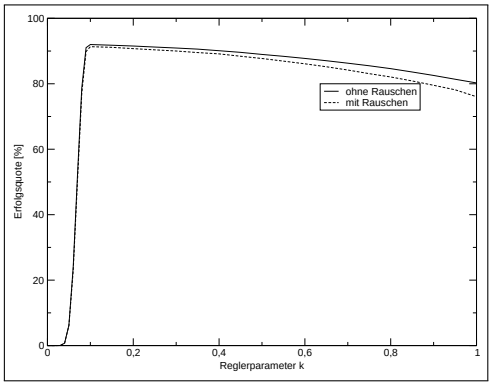
Die Ergebnisse des Traditionellen Reglers mit konstanter Jacobimatrix sind in Abbildung 3.3 abzulesen. Hier liegt der beste Wert bei einer Dämpfung von 0.1 mit einer Erfolgsquote von 91.99 %. Bei einer Vergrößerung der Dämpfung auf 0.2 erhält man eine Reduktion der Iterationsschritte von 66 auf 32. Dabei verschlechtert sich die Erfolgsquote nur minimal. Das Rauschen nimmt hier keinen großen Einfluss auf den Regler.

Für k-Werte, die kleiner als 0.1 sind, nimmt die Erfolgsquote stark ab. Das liegt daran, dass bei diesen Reglungen z. T. mehr als 100 Schritte benötigt worden sind. Dadurch gelten sie nicht mehr als erfolgreiche Reglung.

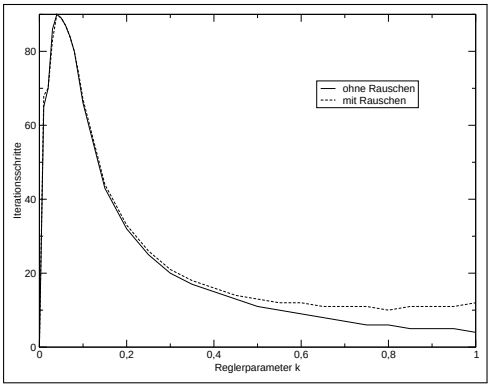
Mit der Wahl von $k = 0.2$ wird eine Erfolgsquote von 91.53 % erreicht. Es lässt sich beobachten, dass sich auch bei größeren Dämpfungsfaktoren die Erfolgsquote nur langsam verschlechtert.

Traditioneller Regler mit dynamischer Jacobimatrix

In Abbildung 3.4, die die Ergebnisse des Traditionellen Relgers darstellt, liegt der optimale Wert von k bei 0.07 mit einer Erfolgsquote von 99.11 %. Das Gaußsche Rauschen hat bei dieser Wahl keinen großen Einfluss auf die Ergebnisse des Reglers. Bei einen k-Wert von 0.07 liegt die durchschnittliche Iterationsanzahl bei 76. Durch die Wahl von 0.1 lassen sich die Reglerschritte auf 52 reduzieren. Die Erfolgsquote verschlechtert sich dabei nur minimal auf 98.59 %. Dieses Ergebnis deckt sich auch mit den Experimenten von Siebel in [Sie99].

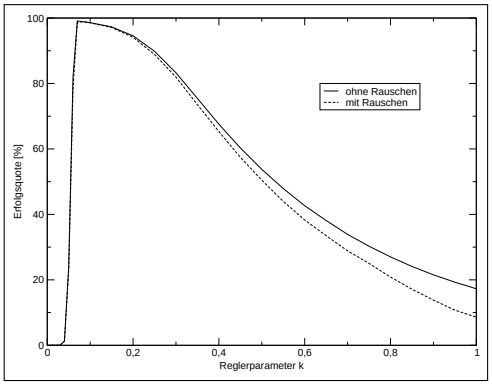


(a) Erfolgsquote bei verschiedenen Parametern k

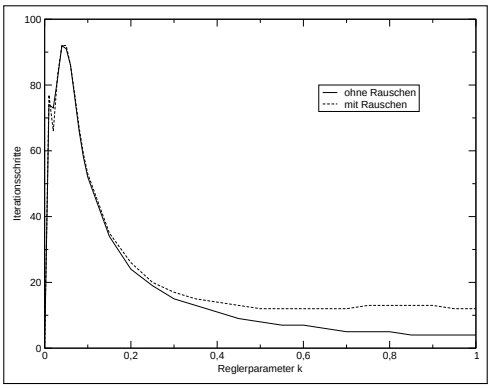


(b) Iterationsschritte bei verschiedenen Parametern k

Abbildung 3.3: Multilagen-Simulation mit konstanter Jacobimatrix



(a) Erfolgsquote bei verschiedenen Parametern k

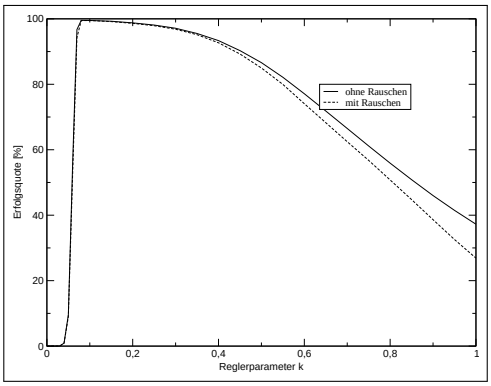


(b) Iterationsschritte bei verschiedenen Parametern k

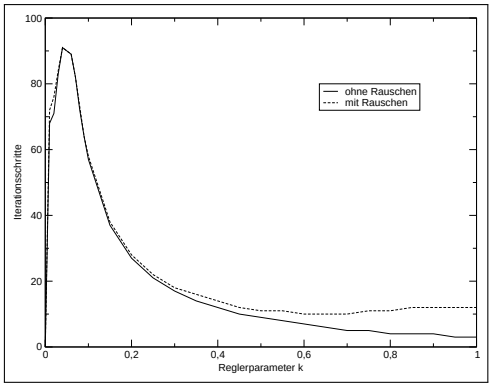
Abbildung 3.4: Multilagen-Simulation mit dynamischer Jacobimatrix

MJP-Regler

Mit dem MJP-Regler lässt sich bei $k = 0.08$ eine Erfolgsquote von 99.53 % erreichen, wobei durchschnittlich 72 Regelschritte benötigt werden. Auch bei einem Wert von $k = 0.15$ liefert der Regler noch gute Ergebnisse. Dabei verringert sich die durchschnittliche Schrittzahl auf 37 und die Erfolgsquote auf 99.27 %. In beiden Fällen hat das Gaußsche Rauschen keinen großen Einfluss auf den Regler.



(a) Erfolgsquote bei verschiedenen Parametern k



(b) Iterationsschritte bei verschiedenen Parametern k

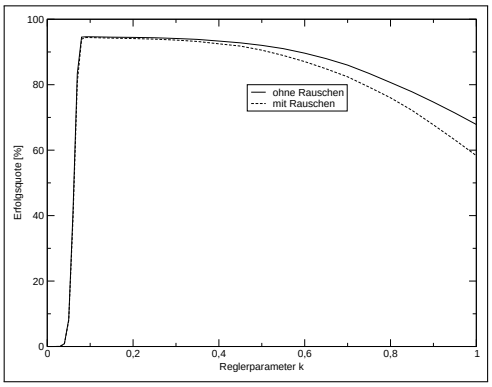
Abbildung 3.5: Multilagen-Simulation mit dem MJP-Regler

PMJ-Regler

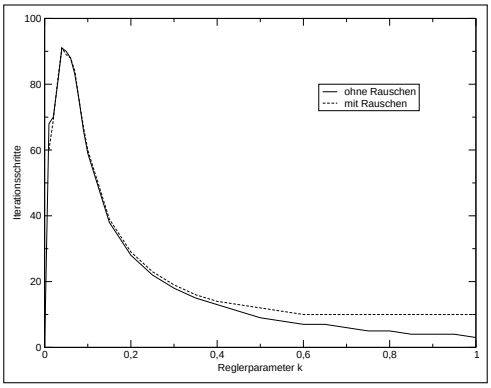
Die beiden Kurven in Abbildung 3.6 ähneln denen des Traditionellen Reglers mit konstanter Jacobimatrix. Auch hier verringert sich die Erfolgsquote bei wachsenden k nur langsam, fällt aber ab 0.06 stärker ab. Bei einem Wert von $k = 0.1$ erreicht der Regler bei einer Erfolgsquote von 94.65 % sein Optimum. Wird für k der Wert 0.15 gewählt, lässt sich die Geschwindigkeit von 59 auf 38 Iterationsschritten steigern. Die Erfolgsquote beträgt hier 94.5223 %.

Regler in Zylinderkoordinaten

Der Regler im Zylinderkoordinatensystem erreicht sein Optimum bei einem Dämpfungsfaktor von $k = 0.07$ mit einer Erfolgsquote von 93.94 %. Auch hier lässt sich bei minimaler Verschlechterung der Erfolgsquote auf 91.18 %, die Schrittzahl durch setzen von $k = 0.1$ von 76 auf 52 reduzieren.

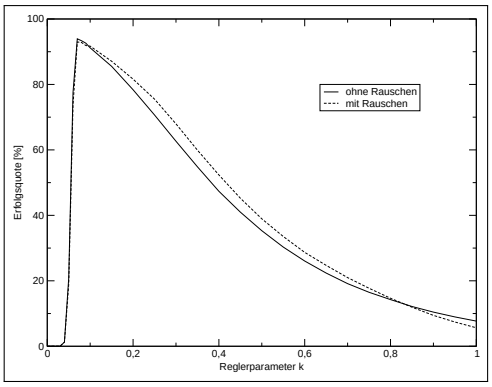


(a) Erfolgsquote bei verschiedenen Parametern k

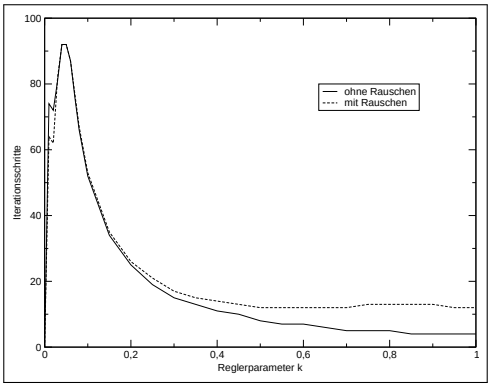


(b) Iterationsschritte bei verschiedenen Parametern k

Abbildung 3.6: Multilagen-Simulation mit dem PMJ-Regler



(a) Erfolgsquote bei verschiedenen Parametern k



(b) Iterationsschritte bei verschiedenen Parametern k

Abbildung 3.7: Multilagen-Simulation mit dem Regler im Zylinderkoordinatensystem

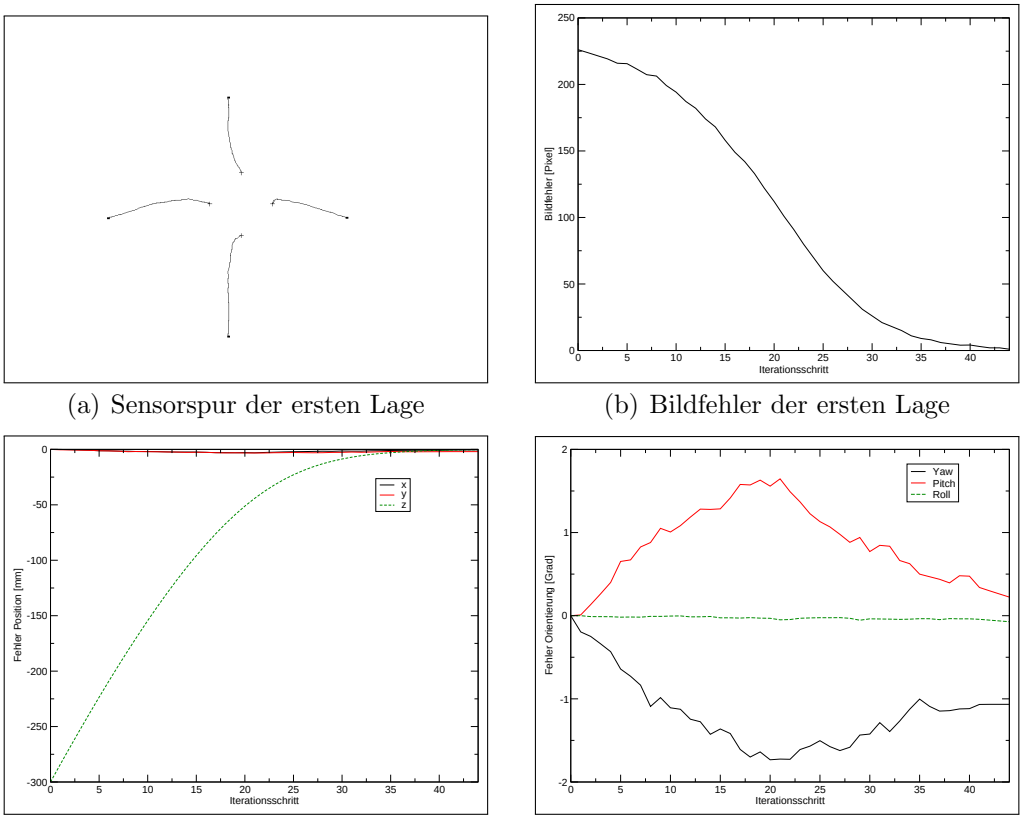
3.4.2 Testläufe mit dem OpenGL-Simulator

Ausgangslage 1: [0, 0, -300, 0°, 0°, 0°]

Traditioneller Regler mit konstanter Jacobimatrix

Die Ergebnisse für den Traditionellen Regler sind in Abbildung 3.8 zusammengefasst. Der Regler braucht für die erste Lage 44 Regelschritte. In Abbildung 3.8(a) zeigt sich, dass sich die Objektmarkierungen fast direkt auf die Zielposition zubewegen. Dabei wird mit den Kreuzen die Startpose und mit den Rechtecken die Zielpose der einzelnen Objektmarkierungen gekennzeichnet. Wie in Abbildung 3.8(b) zu sehen, sind die Bewegungen am Anfang und besonders am Schluss kleiner als die in der Mitte der Reglung.

In Abbildung 3.8(d) und 3.8(c) ändert sich der Yaw- und Pitch-Winkel bzw. der x- und y-Wert nur nur minimal.



(a) Sensorspur der ersten Lage

(b) Bildfehler der ersten Lage

(c) Fehler der Kameraposition

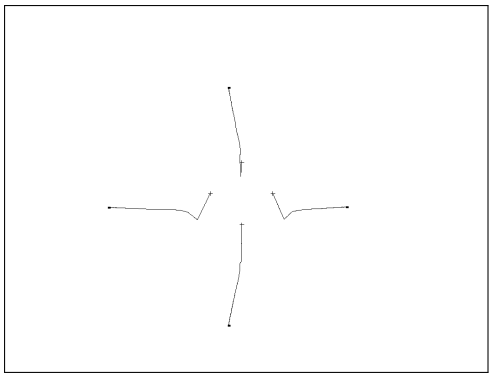
(d) Fehler der Kameraorientierung

Abbildung 3.8: Traditioneller Regler mit konstanter Jacobimatrix, Lage 1

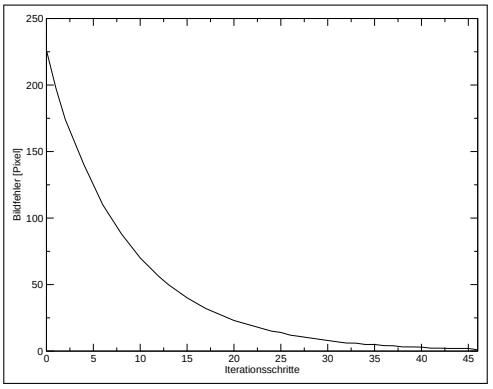
Traditioneller Regler mit dynamischer Jacobimatrix

In Abbildung 3.9(a) fällt auf, dass im ersten Iterationsschritt ein großer Schritt in Richtung unterem Bildrand durchgeführt wird. Drei der vier Objektmarkierungen bewegen sich dabei in die falsche Richtung. Wie sich in Abbildung 3.9(b) aber zeigt, verringert sich trotzdem der Bildfehler. Abbildungen 3.9(c) und 3.9(d) zeigen, dass im ersten Iterationsschritt eine relativ große Bewegung entlang der y-Achse und eine Drehung um die x-Achse durchgeführt wird.

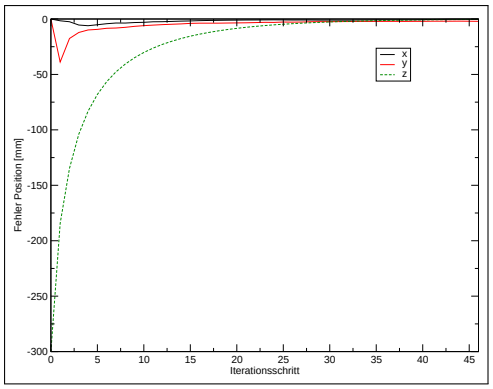
Der Regler erreicht seine Zielposition nach 46 Iterationen. Bis auf den ersten Schritt bewegen sich die Objektmarkierungen wieder fast direkt in Richtung ihrer Zielpositionen. Auch hier werden die Regelschritte zum Schluss wieder kleiner.



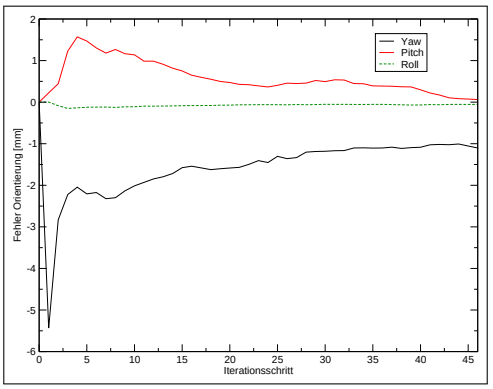
(a) Sensorspur der ersten Lage



(b) Bildfehler der ersten Lage



(c) Fehler der Kameraposition

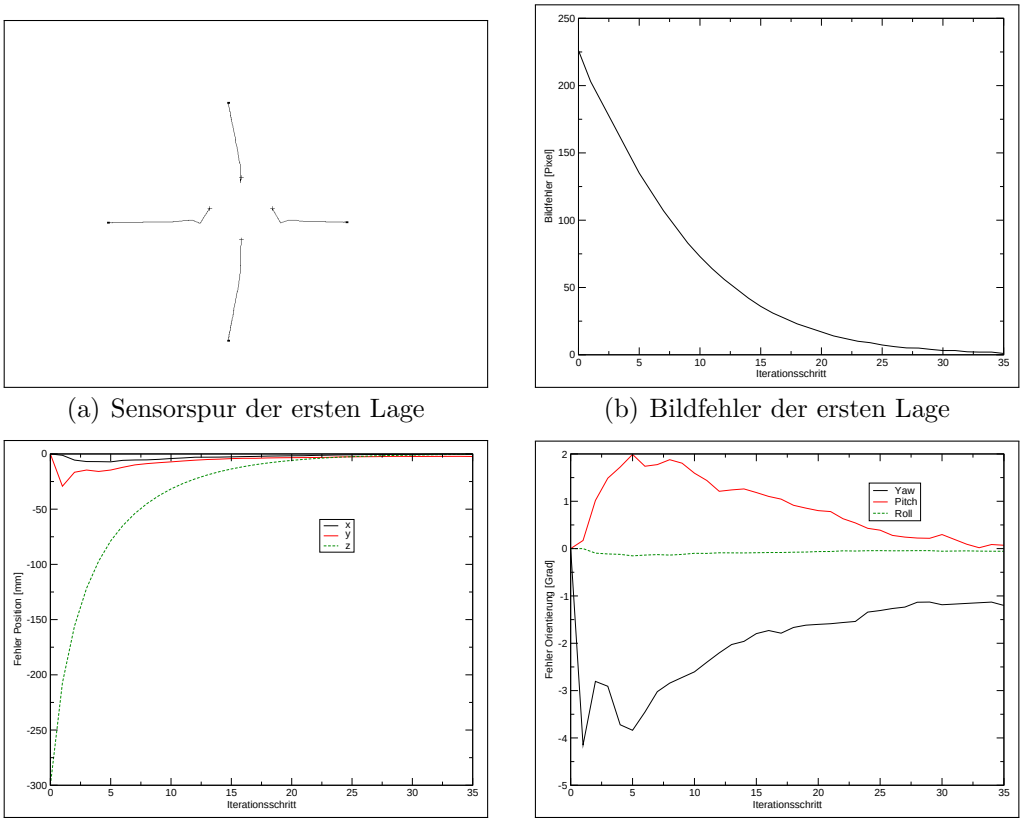


(d) Fehler der Kameraorientierung

Abbildung 3.9: Traditioneller Regler mit dynamischer Jacobimatrix, Lage 1

MJP-Regler

Der MJP-Regler beendet nach 35 Iterationen erfolgreich die Reglung. Ähnlich wie beim Traditionellen Regler mit dynamischer Bildjacobimatrix, wird auch hier beim ersten Iterationsschritt eine relativ große Bewegung der Bildmerkmale in Richtung des unteren Bildrandes beobachtet (siehe Abbildung 3.10(a)). Diese fällt aber kleiner als beim Traditionellen Regler aus.



(a) Sensorspur der ersten Lage

(b) Bildfehler der ersten Lage

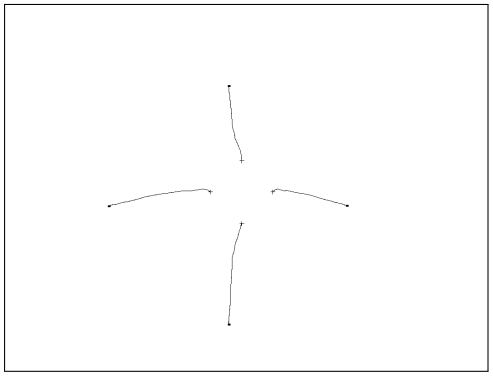
(c) Fehler der Kameraposition

(d) Fehler der Kameraorientierung

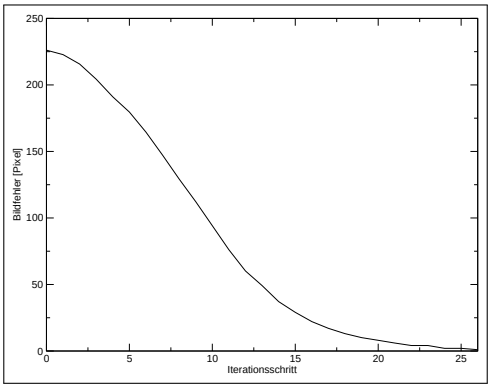
Abbildung 3.10: MJP-Regler, Lage 1

PMJ-Regler

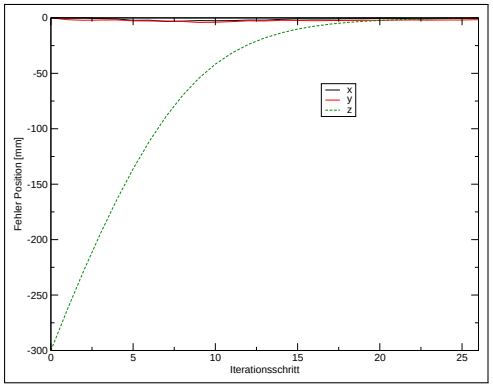
Nach 26 Schritten erreicht der PMJ-Regler die Zielposition. Die Ergebnisse in Abbildung 3.11 lassen sich mit denen des Traditionellen Reglers mit konstanter Jacobimatrix vergleichen.



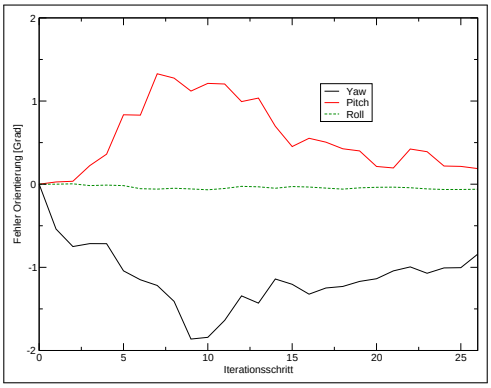
(a) Sensorspur der ersten Lage



(b) Bildfehler der ersten Lage



(c) Fehler der Kameraposition

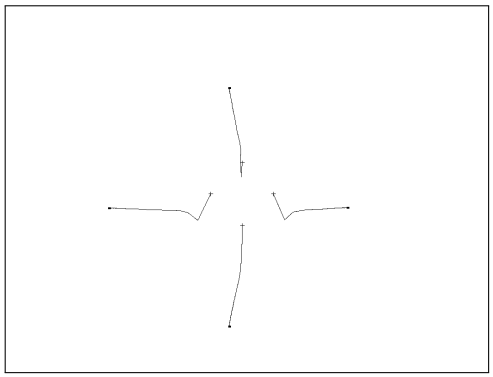


(d) Fehler der Kameraorientierung

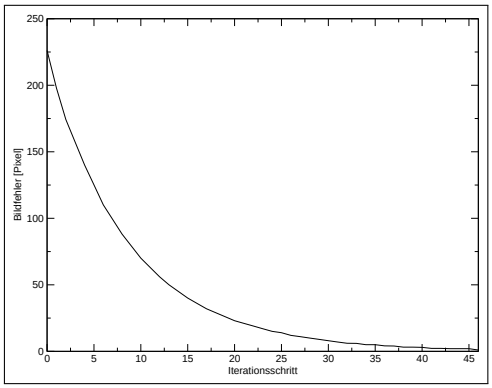
Abbildung 3.11: PMJ-Regler, Lage 1

Regler in Zylinderkoordinaten

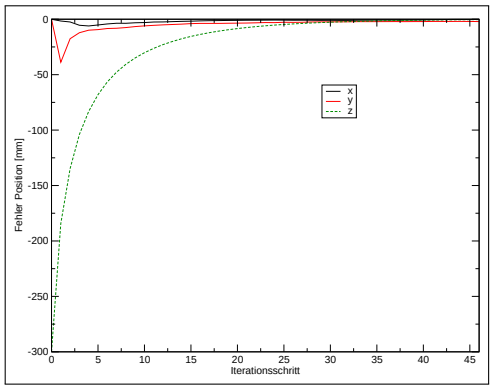
Die Ergebnisse in Abbildung 3.12 entsprechen denen des Traditionellen Reglers. Hier streben die beiden Parameter η und ξ gegen unendlich, da die gesuchte Rotationsachse parallel zur Bildebene liegt. Aus diesem Grund wird die normale Jacobimatrix benutzt.



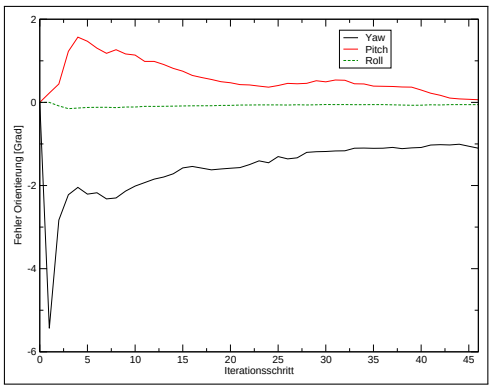
(a) Sensorspur der ersten Lage



(b) Bildfehler der ersten Lage



(c) Fehler der Kameraposition



(d) Fehler der Kameraorientierung

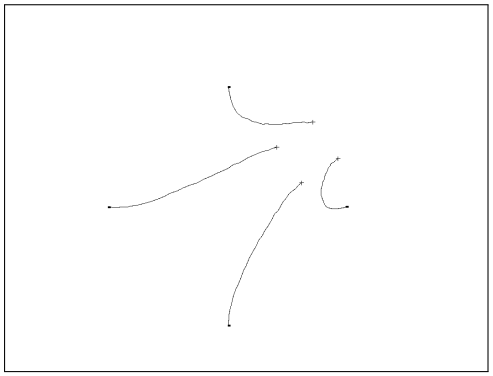
Abbildung 3.12: Regler in Zylinderkoordinaten, Lage 1

Ausgangslage 2: [20, -50, -300, -10°, -10°, -10°]

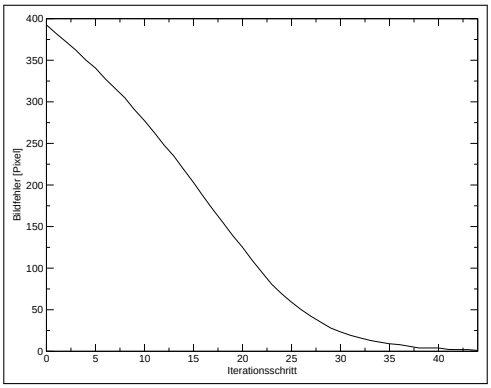
Traditioneller Regler mit konstanter Jacobimatrix

Für die zweite Lage benötigt der Traditionelle Regler mit konstanter Bildjacobimatrix 44 Iterationsschritte. Die Abbildungen 3.15(b) bis 3.15(d) zeigen, dass sich anfangs der Bildfehler sowie der Lagefehler der Kamera fast linear verringert. Zum Schluss verkleinert sich die Schrittgröße wieder.

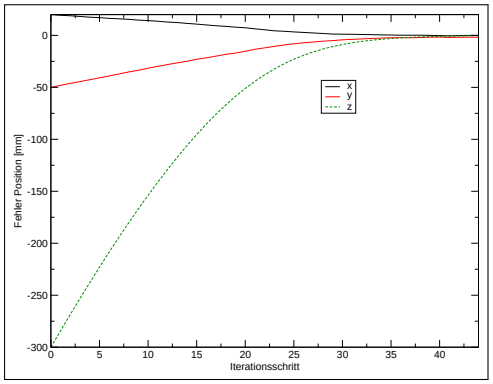
Die obere und die rechte Objektmarkierung in Abbildung 3.15(a) bewegen sich bei den ersten Schritten nicht direkt auf die Zielposition zu. Sie beschreiben einen leichten Bogen.



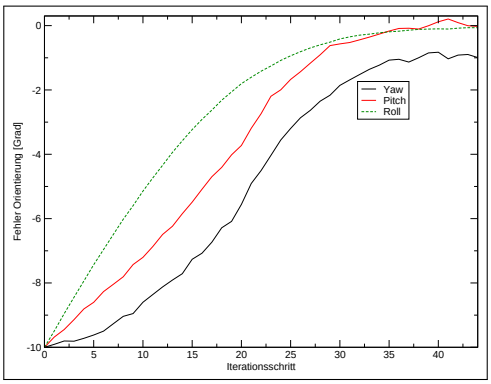
(a) Sensorspur der zweiten Lage



(b) Bildfehler der zweiten Lage



(c) Fehler der Kameraposition

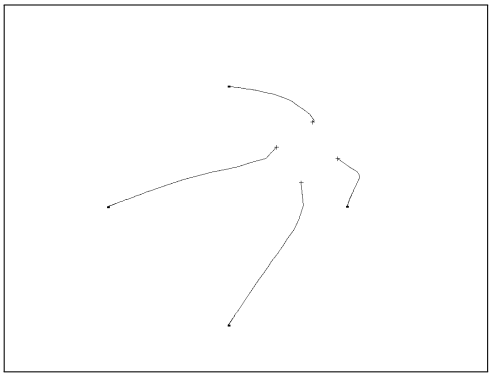


(d) Fehler der Kameraorientierung

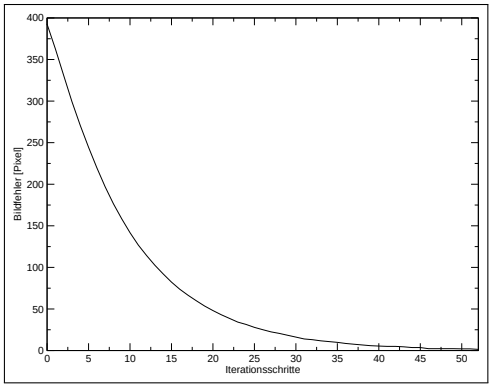
Abbildung 3.13: Traditioneller Regler mit konstanter Jacobimatrix, Lage 2

Traditioneller Regler mit dynamischer Jacobimatrix

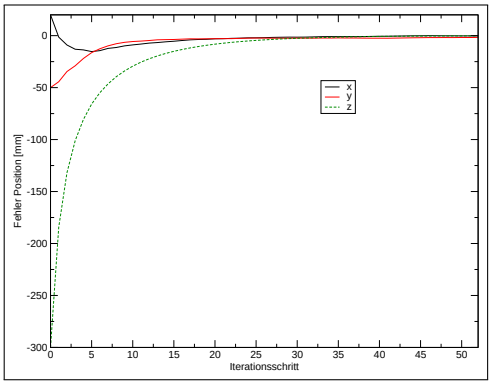
Der Traditionelle Regler mit dynamischer Jacobimatrix verringert den Bild- und den Lagefehler der Kamera im Gegensatz zum vorherigen Regler anfangs schneller (siehe Abbildungen 3.14(b) bis 3.14(d)). Insgesamt braucht er allerdings mit 52 Regelschritten ein wenig länger.



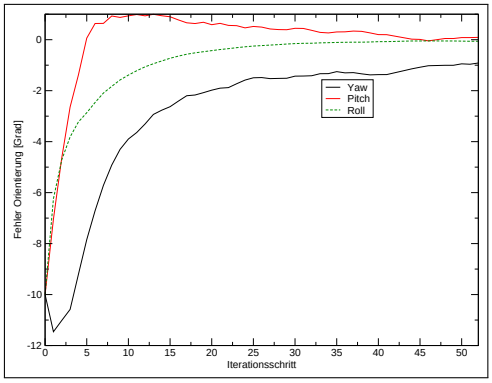
(a) Sensorspur der zweiten Lage



(b) Bildfehler der zweiten Lage



(c) Fehler der Kameraposition

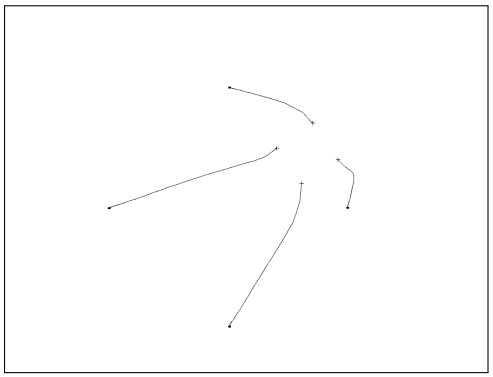


(d) Fehler der Kameraorientierung

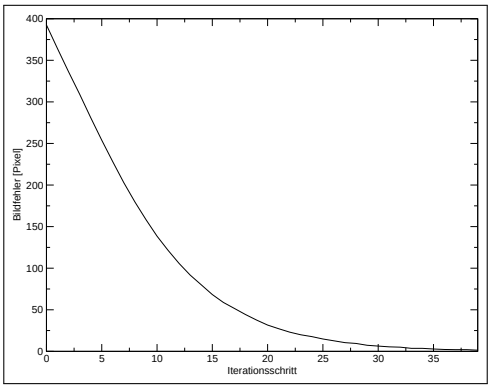
Abbildung 3.14: Traditioneller Regler mit dynamischer Jacobimatrix, Lage 2

MJP-Regler

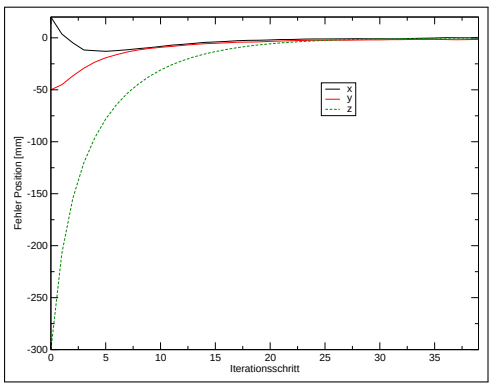
Auch in dieser Lage ähneln die Ergebnisse des MJP-Reglers in Abbildung 3.15 denen des Traditonellen Reglers mit dynamischer Jacobimatrix. Wie zuvor sind auch hier die Bewegungen ein wenig glatter. Er erreicht die Teachpose nach 39 Iterationen.



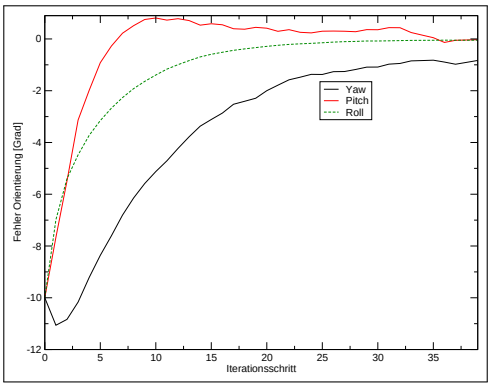
(a) Sensorspur der zweiten Lage



(b) Bildfehler der zweiten Lage



(c) Fehler der Kameraposition

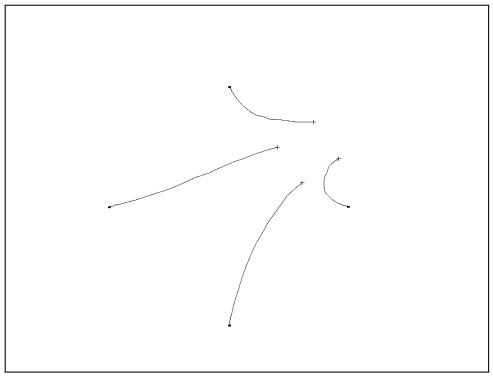


(d) Fehler der Kameraorientierung

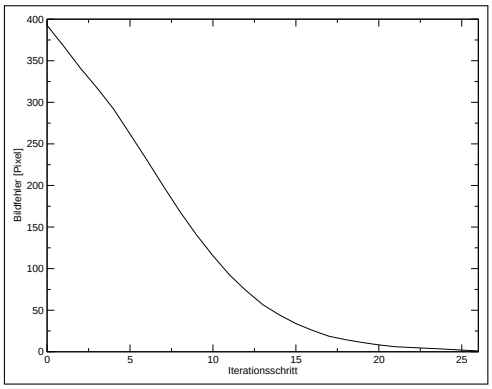
Abbildung 3.15: MJP-Regler, Lage 2

PMJ-Regler

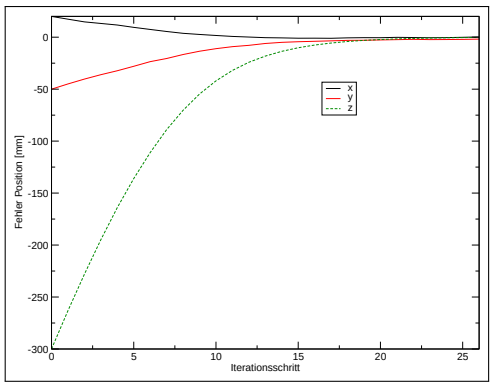
Der PMJ-Regler schafft es nach 26 Schritten die Objektmarkierungen in die Ziellage zu verschieben. Die Bewegung der Bildmerkmale auf dem Kamerasensor (siehe Abbildung 3.16(a)) ähneln denen in Abbildung 3.13(a).



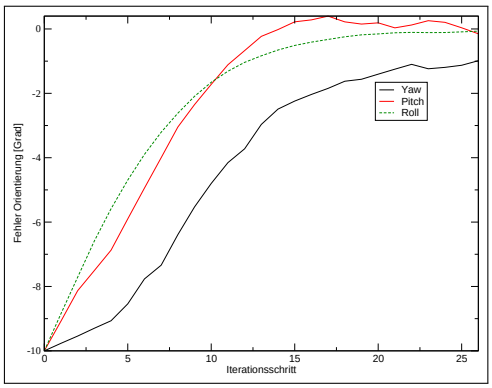
(a) Sensorspur der zweiten Lage



(b) Bildfehler der zweiten Lage



(c) Fehler der Kameraposition



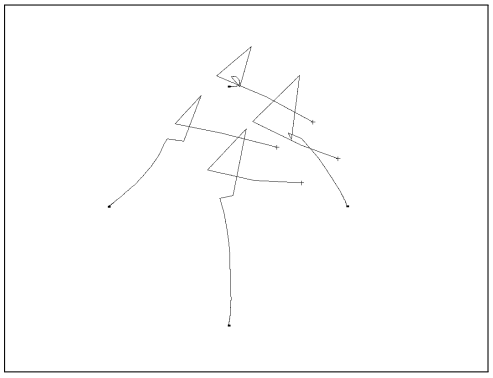
(d) Fehler der Kameraorientierung

Abbildung 3.16: PMJ-Regler, Lage 2

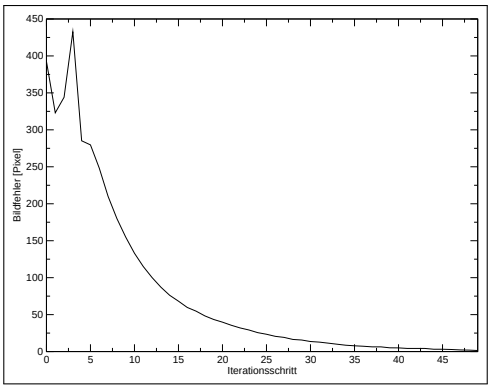
Regler in Zylinderkoordinaten

Der Regler in Zylinderkoordinaten liefert bei dieser Lage (siehe Abbildung 3.17) ein eher schlechtes Ergebnis. Er erreicht zwar nach 49 Schritten seine Ziellage, jedoch steigt der Bildfehler in Abbildung 4.18(b) zwischendurch auf ca. 433 Pixel. Auch der Yaw- und Pitch-Winkel des Orientierungsfehlers der Kamera verschlechtern sich auf einen Wert von über 30° bzw. unter -30°(siehe Abbildung 3.17(d)).

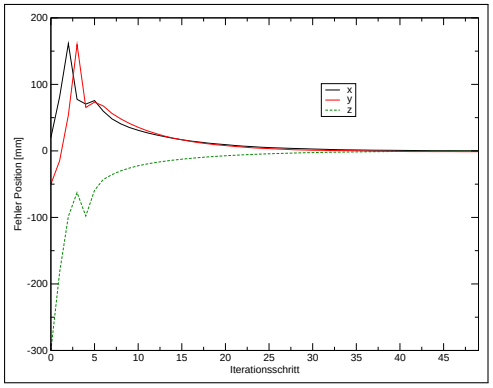
Dieser Regler hat auch in ähnlichen Startposen Schwierigkeiten. Es sind solche Posen, in denen die Objektmarkierungen relativ zentral im Bildbereich liegen, bzgl. der z-Achse verdreht sind und einen gewissen Abstand von der Kamera haben.



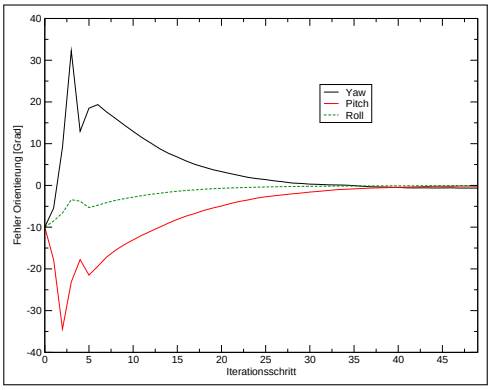
(a) Sensorspur der zweiten Lage



(b) Bildfehler der zweiten Lage



(c) Fehler der Kameraposition



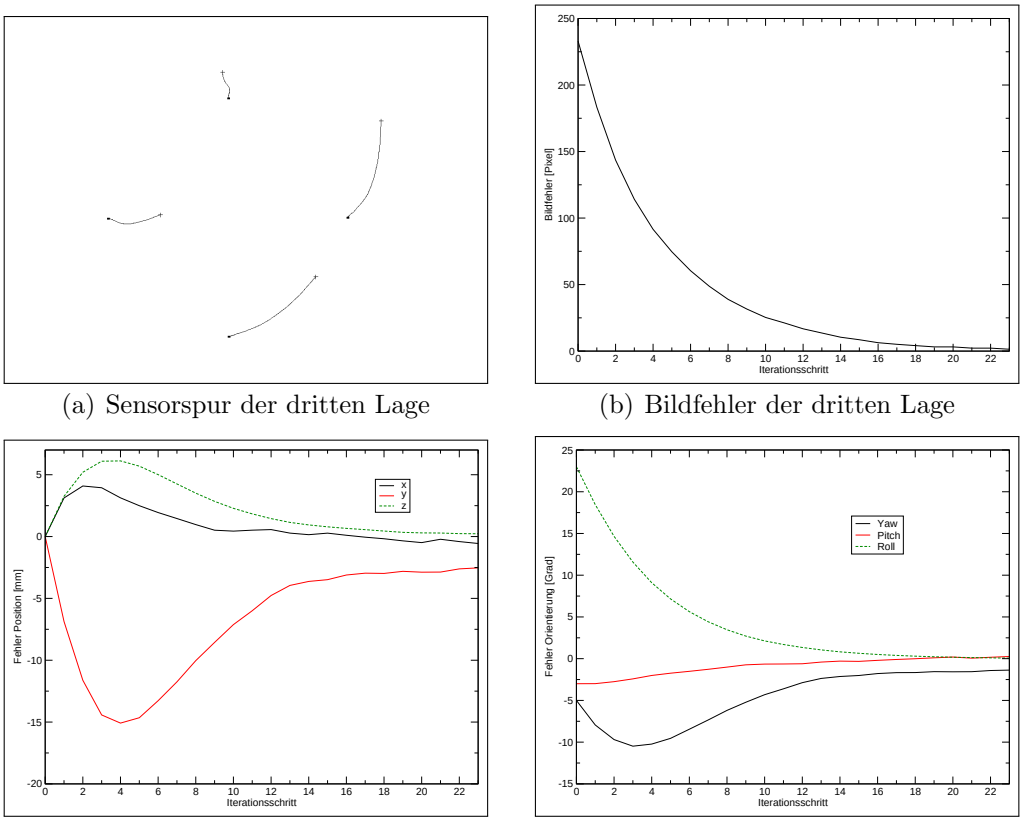
(d) Fehler der Kameraorientierung

Abbildung 3.17: Regler in Zylinderkoordinaten, Lage2

Ausgangslage 3: [0, 0, 0, -5°, -3°, 23°]

Traditioneller Regler mit konstanter Jacobimatrix

Der Ansatz mit der konstanten Jacobimatrix erreicht nach 23 Iterationen die Teachpose. Auffällig bei den Ergebnissen in Abbildung 3.18 ist, dass sich der Positionsfehler der Kamera bzgl. der y-Achse verschlechtert (siehe Abbildung 3.13(d)). Gleichzeitig vergrößert sich der Orientierungsfehler der Kamera bzgl. des Yaw-Winkels. Beide Bewegungen gleichen sich aber wieder aus. Dieses Verhalten lässt sich bei allen hier getesteten Reglern beobachten.



(a) Sensorspur der dritten Lage

(b) Bildfehler der dritten Lage

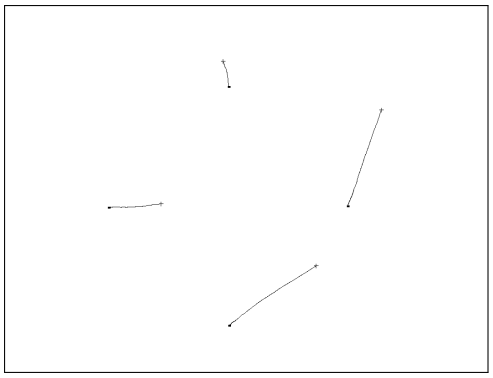
(c) Fehler der Kameraposition

(d) Fehler der Kameraorientierung

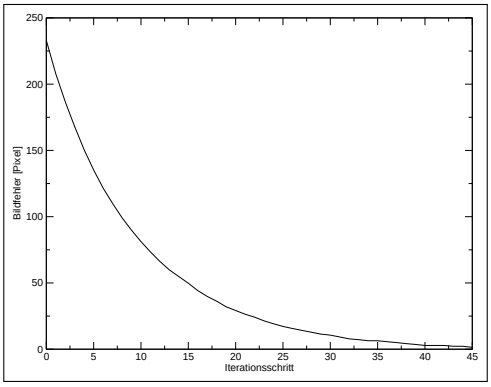
Abbildung 3.18: Traditioneller Regler mit konstanter Jacobimatrix, Lage 3

Traditioneller Regler mit dynamischer Jacobimatrix

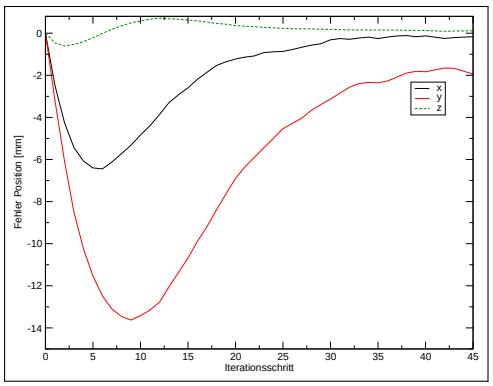
Der Traditionelle Regler mit dynamischer Bildjacobimatrix hat nach 45 Schritten die Ziellage erreicht. In Abbildung 3.19(a) sieht man, dass sich die Objektmarkierungen auf einer geraden Linie auf die Zielpositionen zubewegen. Im Gegensatz zu den anderen Ansätzen lässt sich in Abbildung 3.19(c) eine Verschlechterung des Lagefehlers der Kamera in negativer Richtung der x-Achse beobachten. Gleichzeitig wir eine ausgleichende Rotation um die y-Achse durchgeführt, so dass sich der Orientierungsfehler bzgl. des Pitch-Winkels verschlechtert.



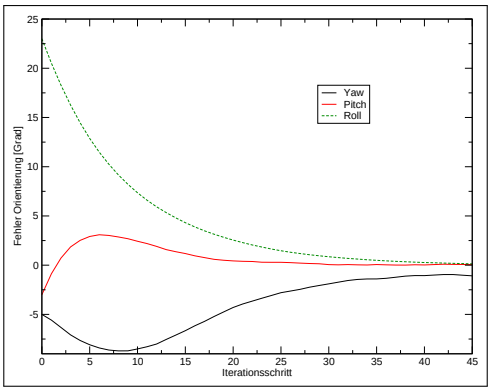
(a) Sensorspur der dritten Lage



(b) Bildfehler der dritten Lage



(c) Fehler der Kameraposition

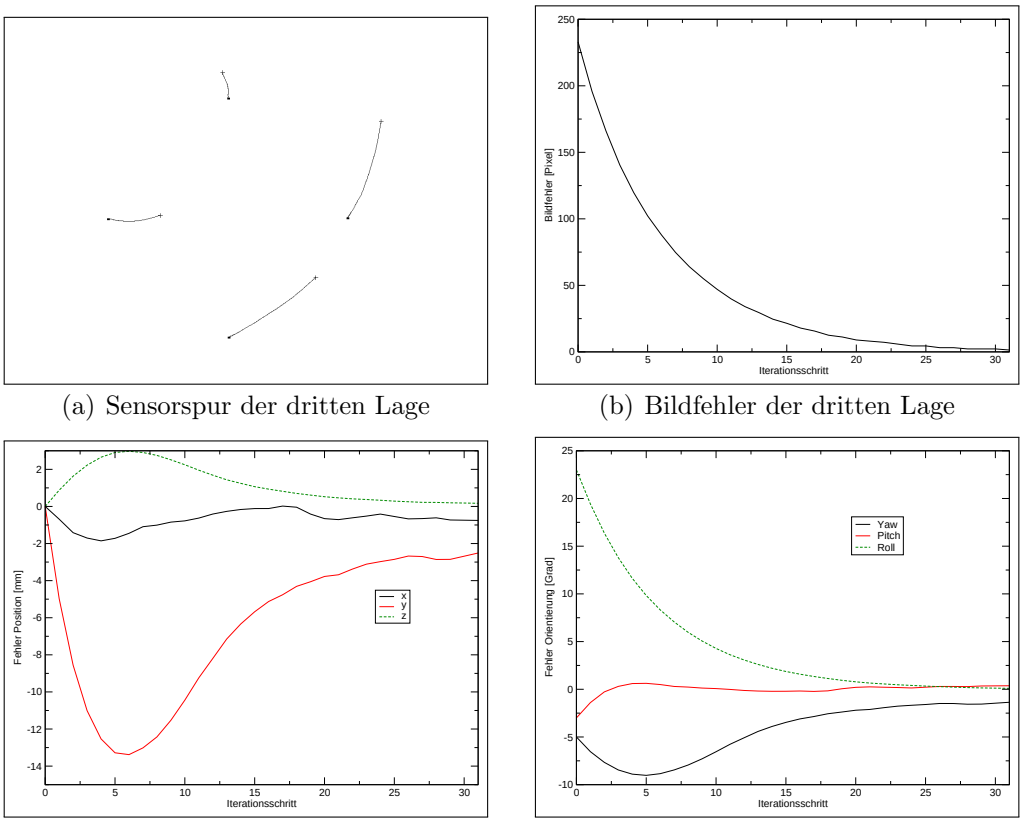


(d) Fehler der Kameraorientierung

Abbildung 3.19: Traditioneller Regler mit dynamischer Jacobimatrix, Lage 3

MJP-Regler

Der Regler hat nach 31 Iterationen sein Ziel erreicht. Die Bildmerkmale in Abbildung 3.20(a) beschreiben hier keine gerade Linie. Der Orientierungsfehler (siehe Abbildung 3.20(d)) aber auch der Positionsfehler (Abbildung 3.20(c)) der Kamera steigt insgesamt nicht so stark wie beim traditionellen Ansatz.



(a) Sensorspur der dritten Lage

(b) Bildfehler der dritten Lage

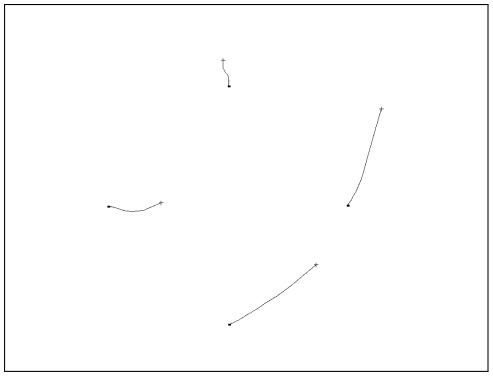
(c) Fehler der Kameraposition

(d) Fehler der Kameraorientierung

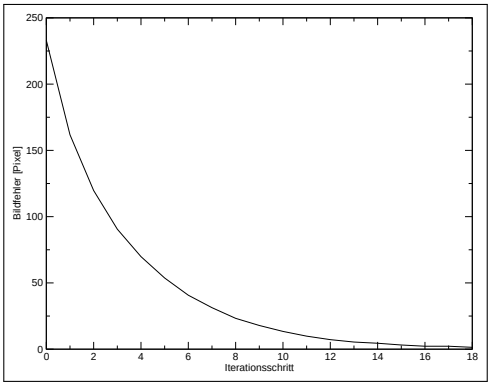
Abbildung 3.20: MJP-Regler, Lage 3

PMJ-Regler

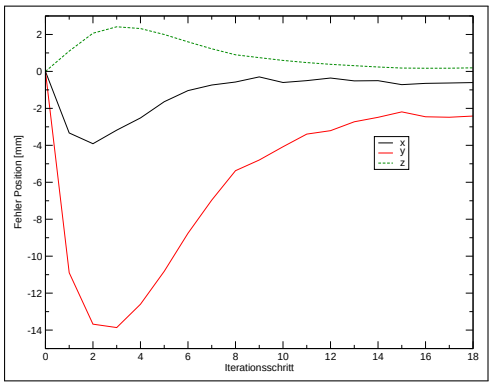
Der PMJ-Regler erreicht nach 18 Regelschritten die Teachpose und liefert ähnliche Ergebnisse wie der MJP-Regler.



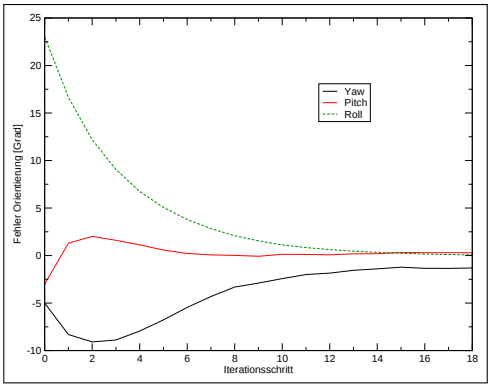
(a) Sensorspur der dritten Lage



(b) Bildfehler der dritten Lage



(c) Fehler der Kameraposition

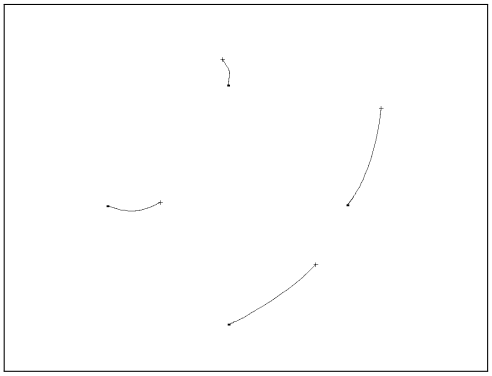


(d) Fehler der Kameraorientierung

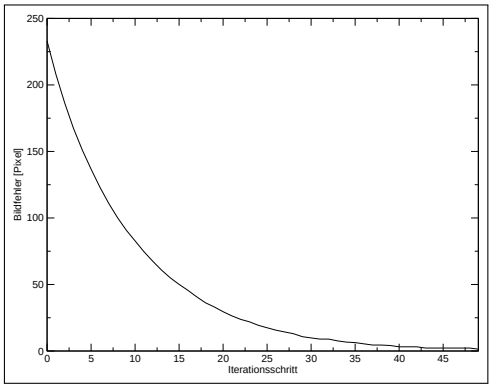
Abbildung 3.21: PMJ-Regler, Lage 3

Regler in Zylinderkoordinaten

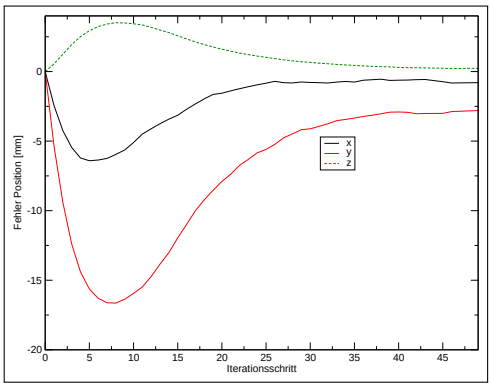
Der Regler in Zylinderkoordinaten stoppt nach 49 Iteration erfolgreich die Reglung. Er braucht damit länger als die beiden Regler zweiter Ordnung, liefert aber ähnliche Ergebnisse.



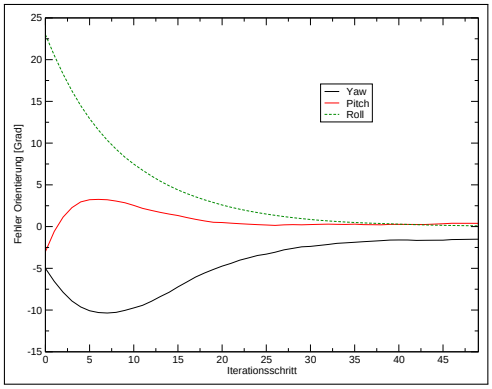
(a) Sensorspur der dritten Lage



(b) Bildfehler der dritten Lage



(c) Fehler der Kameraposition



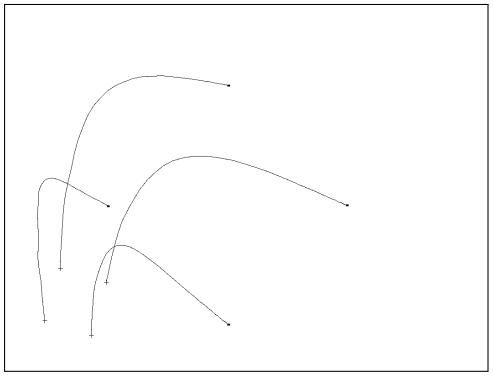
(d) Fehler der Kameraorientierung

Abbildung 3.22: Regler in Zylinderkoordinaten, Lage3

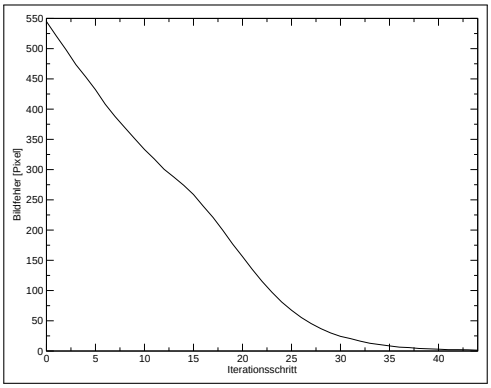
Ausgangslage 4: [150, 90, -200, 10°, -15°, 30°]

Traditioneller Regler mit konstanter Jacobimatrix

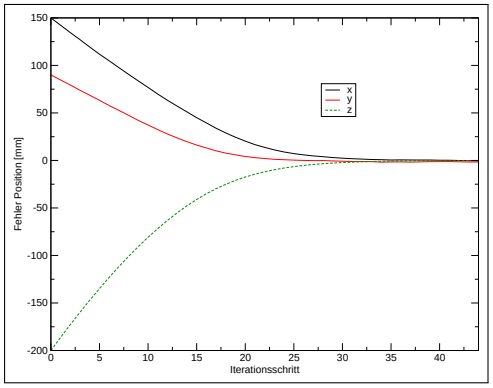
In Abbildung 3.23(a) lässt sich beobachten, dass sich die Bildmerkmale zunächst in Richtung des oberen Bildbereichs bewegen, bevor sie dann wieder zur Mitte streben. Trotzdem erreichen die Objektmarkierungen nach 44 Schritten ihr Ziel. Bild- und Kamerafehler werden fortlaufend reduziert, ohne dass sie an einer Stelle wieder wachsen.



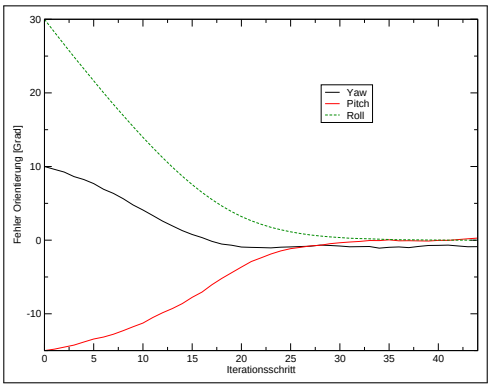
(a) Sensorspur der vierten Lage



(b) Bildfehler der vierten Lage



(c) Fehler der Kameraposition

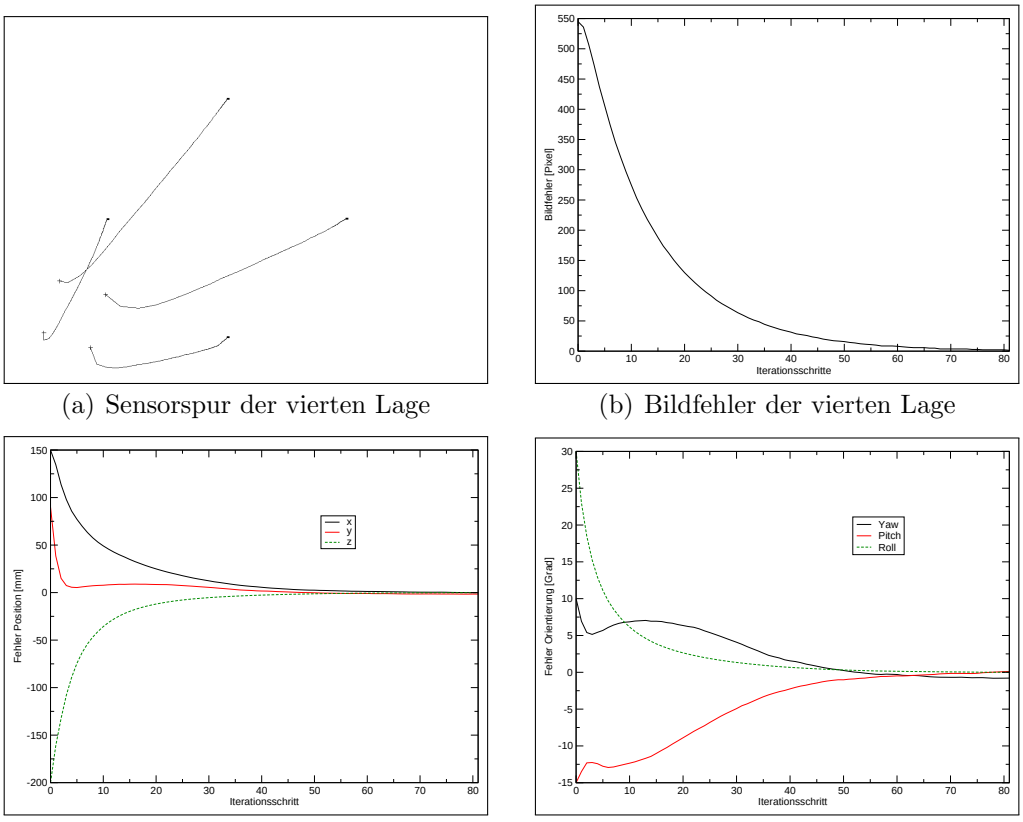


(d) Fehler der Kameraorientierung

Abbildung 3.23: Traditioneller Regler mit konstanter Jacobimatrix, Lage 4

Traditioneller Regler mit dynamischer Jacobimatrix

Der Traditionelle Regler mit dynamischer Jacobimatrix erreicht mit einem Dämpfungsfaktor von 0.1 die Teachpose nicht. Die Bildmerkmale verlassen bei der Reglung am unteren Bereich des Bildes den Sensor. Wird der Dämpfungsfaktor auf 0.07 gesetzt, so ist die Reglung erfolgreich (siehe Abbildung 3.24). Das Ergebnis deckt sich mit dem in [Sie99]. Allerdings gelangen die Objektmarkierungen auch hier recht nahe am Rand des Bildbereiches. Die Teachpose wird nach 81 Iterationen erreicht.



(a) Sensorspur der vierten Lage

(b) Bildfehler der vierten Lage

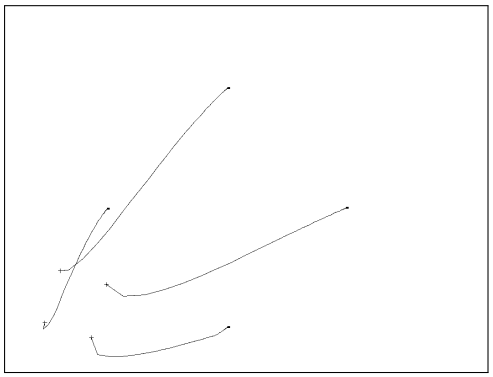
(c) Fehler der Kameraposition

(d) Fehler der Kameraorientierung

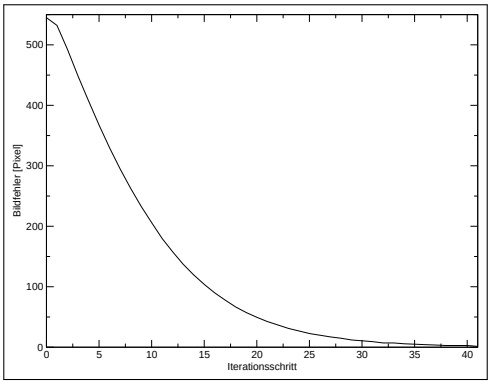
Abbildung 3.24: Traditioneller Regler mit dynamischer Jacobimatrix, Lage 4

MJP-Regler

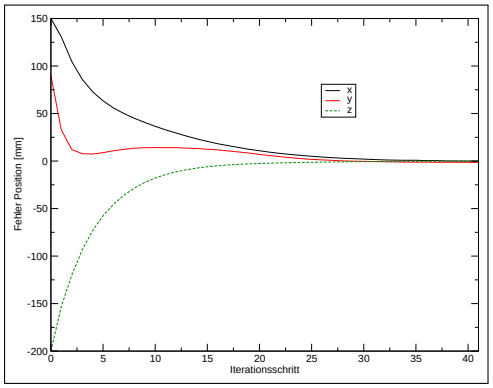
Der MJP-Regler erreicht seine Ziellage nach 41 Schritten. Die Spur der Objektmarkierungen in Abbildung 3.25(a) sind auch hier relativ nahe am Bildrand.



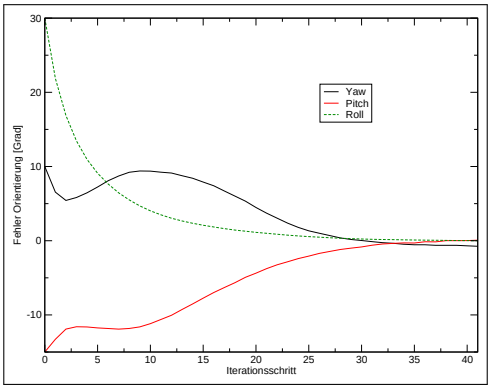
(a) Sensorspur der vierten Lage



(b) Bildfehler der vierten Lage



(c) Fehler der Kameraposition



(d) Fehler der Kameraorientierung

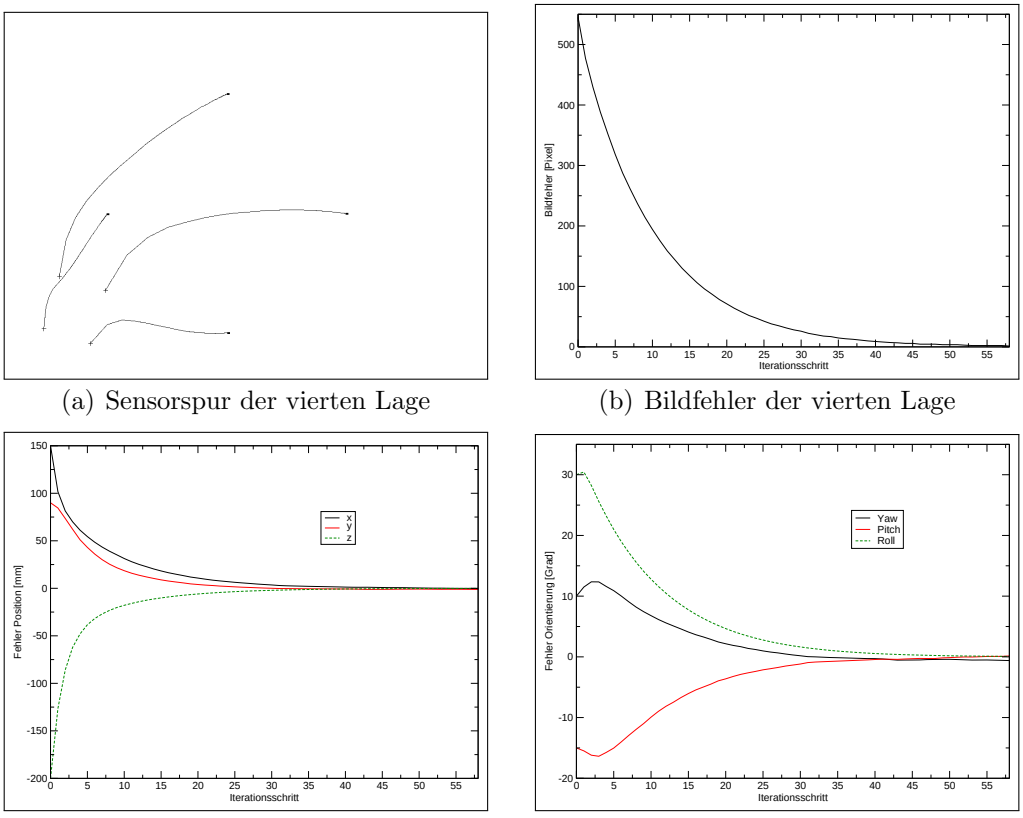
Abbildung 3.25: MJP-Regler, Lage 4

PMJ-Regler

Der PMJ-Regler erreicht in dieser Lage nicht die Teachpose. Die Spur der Objektmarkierungen verläuft am Anfang ähnlich der des Traditionellen Reglers mit konstanter Jacobimatrix. Allerdings verlassen die Bildmerkmale dann den Bildbereich auf der linken Seite des Sensors. Auch die Anpassung des Dämpfungsfaktors führen nicht zur Konvergenz des Reglers.

Regler in Zylinderkoordinaten

Die Objektmarkierungen in Abbildung 3.26(a) bewegen sich in keinem Iterationsschritt auf den Rand des Bildbereichs der Kamera zu. Die Teachpose wird nach 58 Schritten eingenommen.



(a) Sensorspur der vierten Lage

(b) Bildfehler der vierten Lage

(c) Fehler der Kameraposition

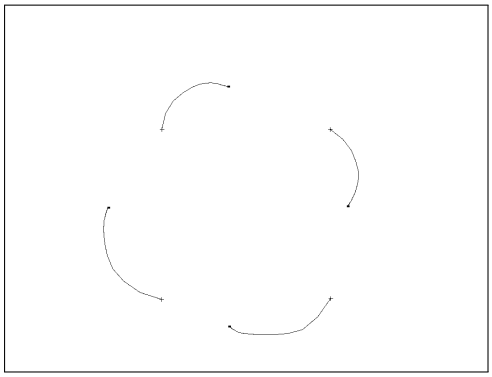
(d) Fehler der Kameraorientierung

Abbildung 3.26: Regler in Zylinderkoordinaten, Lage 4

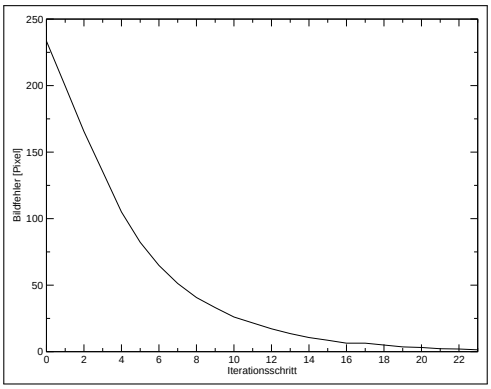
Ausgangslage 5: [0, 0, 0, 0°, 0°, 45°]

Traditioneller Regler mit konstanter Jacobimatrix

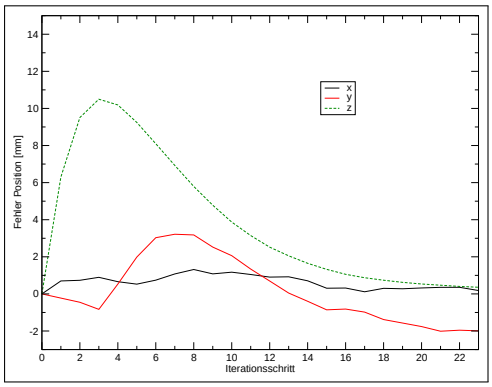
Der Traditionelle Regler mit konstanter Jacobimatrix erreicht nach 23 Schritten die Ziellage. In Abbildung 3.27(c) lässt sich das in Abschnitt 3.2.3 beschriebene Retreat-Advance-Problem beobachten. Der Roboterarm bewegt sich hier anfangs auf das Objekt zu.



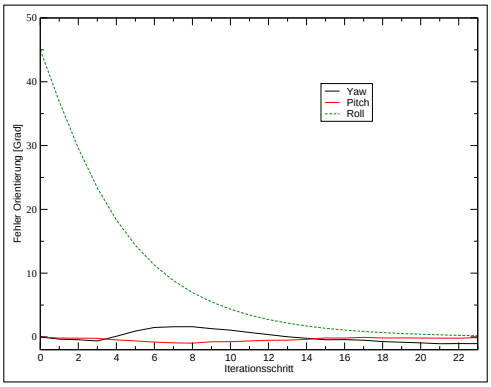
(a) Sensorspur der fünften Lage



(b) Bildfehler der fünften Lage



(c) Fehler der Kameraposition

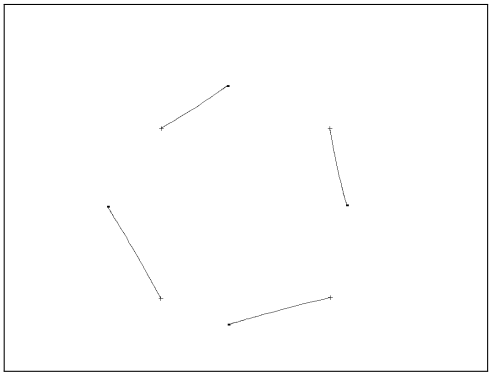


(d) Fehler der Kameraorientierung

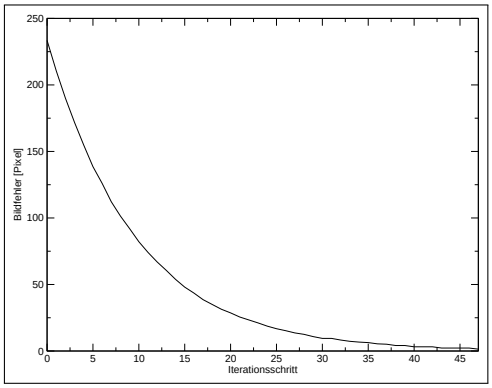
Abbildung 3.27: Traditioneller Regler mit konstanter Jacobimatrix, Lage 5

Traditioneller Regler mit dynamischer Jacobimatrix

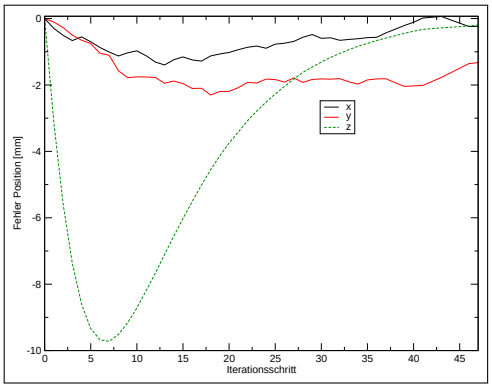
Auch hier lässt sich das Retreat-Advance-Problem beobachten. In Abbildung 3.28(c) entfernt sich der Roboterarm in den ersten Iterationsschritten von dem Objekt. Nach 47 Schritten erreicht der Regler seine Ziellage.



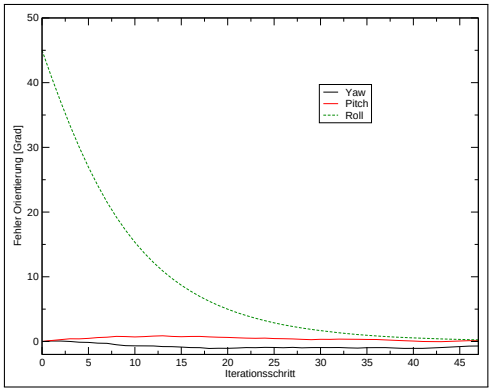
(a) Sensorspur der fünften Lage



(b) Bildfehler der fünften Lage



(c) Fehler der Kameraposition

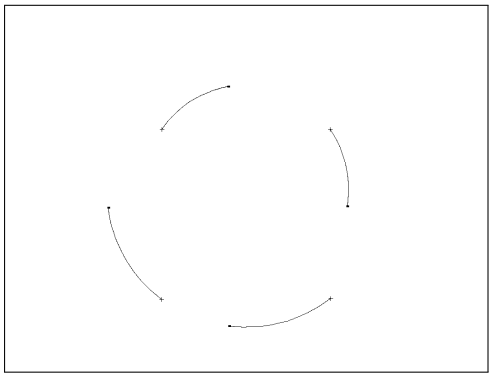


(d) Fehler der Kameraorientierung

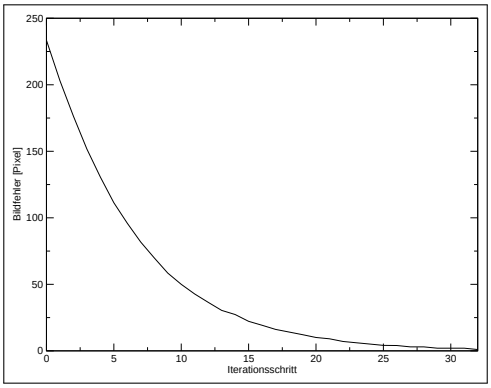
Abbildung 3.28: Traditioneller Regler mit dynamischer Jacobimatrix, Lage 5

MJP-Regler

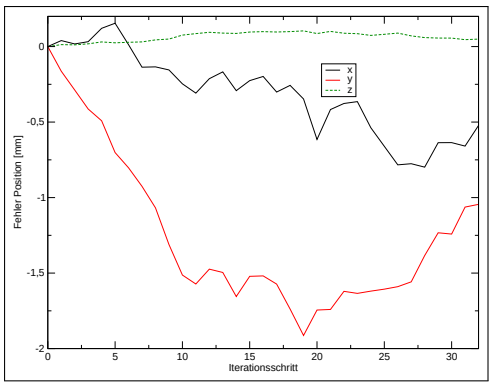
Der Regler hat nach 31 Iterationen sein Ziel erreicht. Die Bewegung der Bildmerkmale beschreibt keine Linie (siehe Abbildung 3.20(a)) wie beim Traditionellen Regler mit dynamischer Jacobimatrix. Hier wird fast eine reine Rotation um die z-Achse durchgeführt. Die Position der Kamera wird nur geringfügig verändert (siehe Abbildung 3.29(a)).



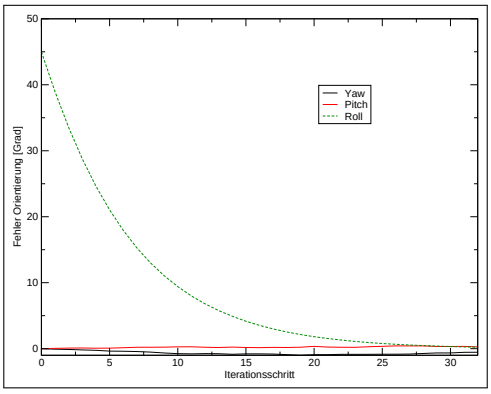
(a) Sensorspur der fünften Lage



(b) Bildfehler der fünften Lage



(c) Fehler der Kameraposition

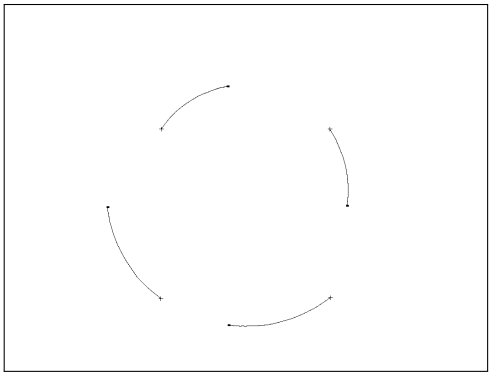


(d) Fehler der Kameraorientierung

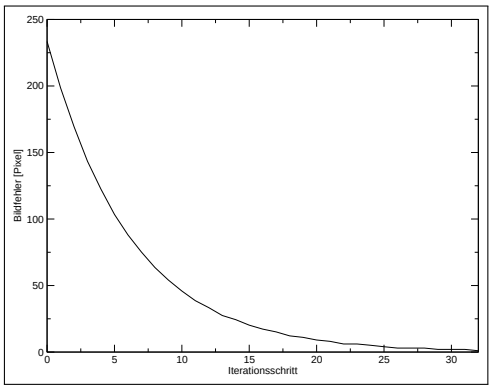
Abbildung 3.29: MJP-Regler, Lage 5

PMJ-Regler

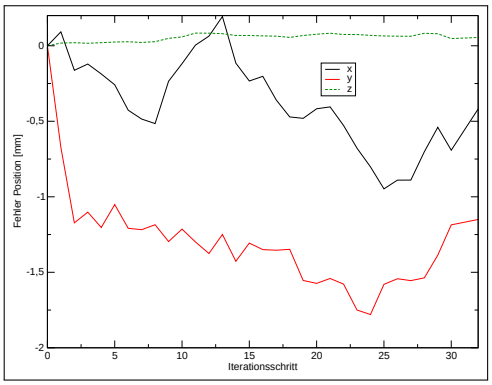
Der PMJ-Regler erreicht nach 18 Regelschritten die Teachpose und liefert ähnliche Ergebnisse wie der MJP-Regler. Auch hier wird fast nur der Roll-Winkel der Kamera verändert.



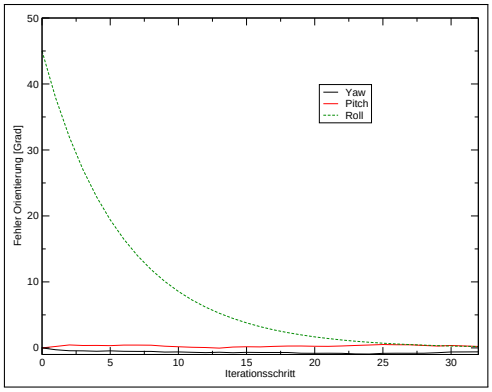
(a) Sensorspur der fünften Lage



(b) Bildfehler der fünften Lage



(c) Fehler der Kameraposition

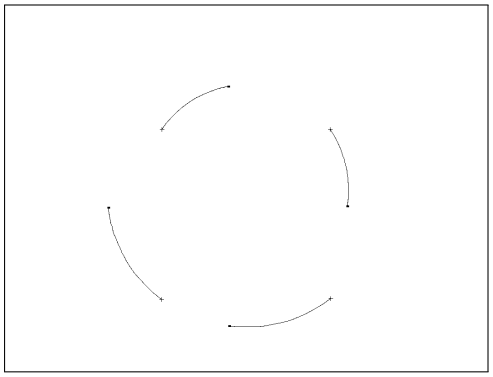


(d) Fehler der Kameraorientierung

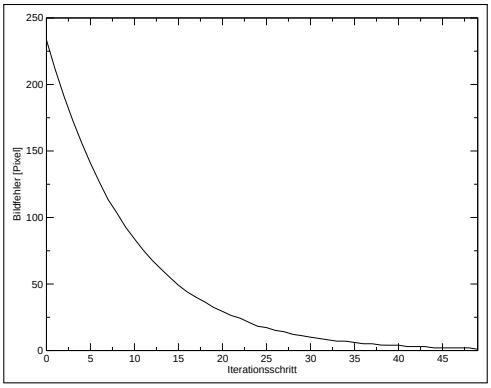
Abbildung 3.30: PMJ-Regler, Lage 5

Regler in Zylinderkoordinaten

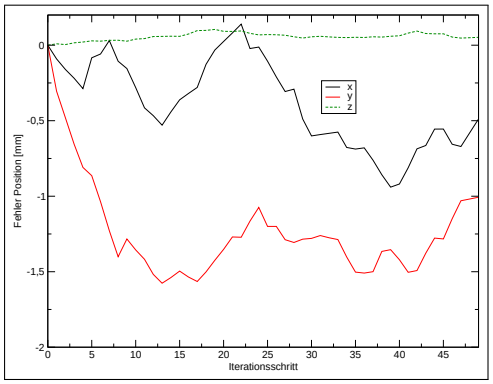
Der Regler in Zylinderkoordinaten erreicht die Teachpose nach 49 Iterationen. Auch dieser Regler wird durch das Retreat-Advance-Problem nicht beeinflusst. Es wird fast ausschließlich eine Rotation um die z-Achse durchgeführt.



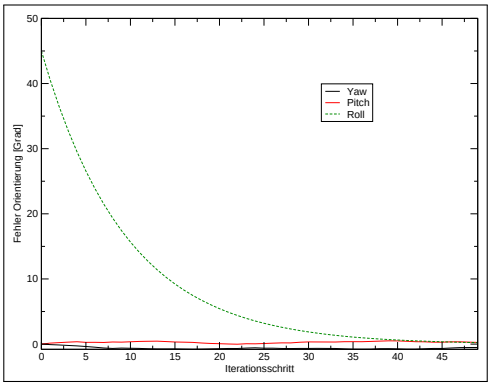
(a) Sensorspur der fünften Lage



(b) Bildfehler der fünften Lage



(c) Fehler der Kameraposition



(d) Fehler der Kameraorientierung

Abbildung 3.31: Regler in Zylinderkoordinaten, Lage5

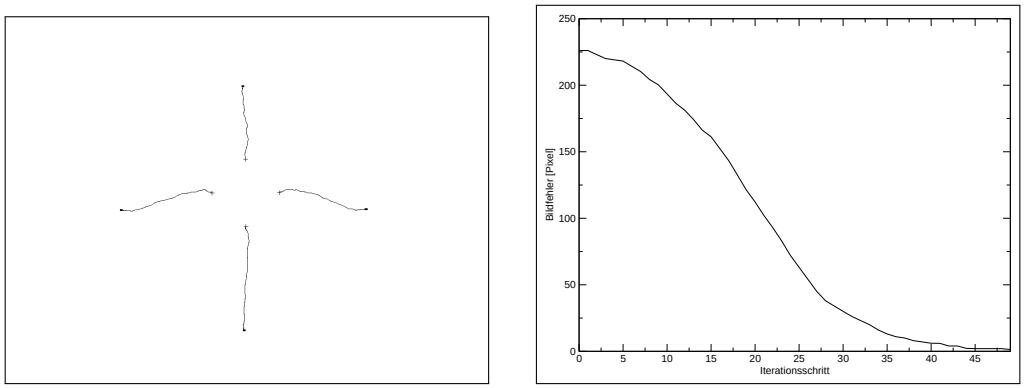
3.4.3 Testläufe am realen System

Im Folgenden werden die fünf Lagen am realen Roboter getestet. Die Ergebnisse stimmen größtenteils mit denen des OpenGL-Simulators überein. Die durch den fehlenden Autofokus auftretenden unscharfen Objektmarkierungen bei kurzen Distanzen verlangsamen die jeweiligen Regler.

Außerdem schafft es der Regler in Zylinderkoordinaten nicht die Zielpose in der dritten Lage zu erreichen. Der Roboter bekommt einen Steuerbefehl, der außerhalb seiner Reichweite liegt. Die Objektmarkierungen verlassen aber nicht den Bildbereich.

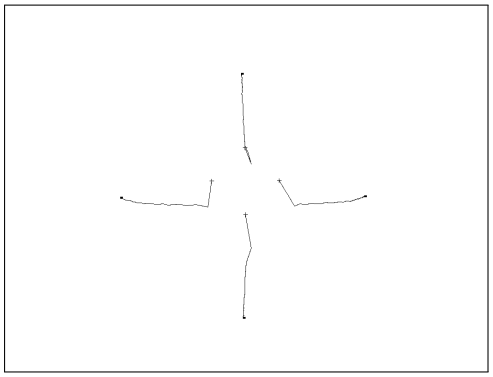
Die Ergebnisse der Testläufe am realen Robotersystem werden in Tabelle 3.2 zusammengefasst.

Ausgangslage 1: [0, 0, -300, 0°, 0°, 0°]

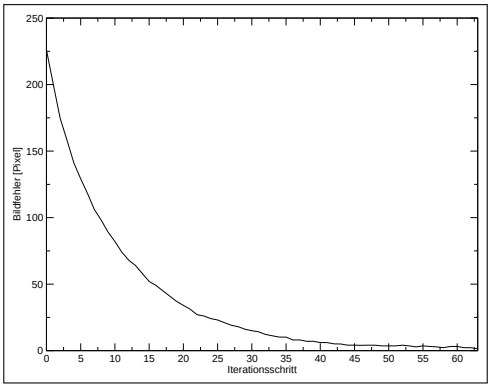


(a) Sensorspur der ersten Lage (b) Bildfehler der ersten Lage

Abbildung 3.32: Traditioneller Regler mit konstanter Jacobimatrix, Lage 1

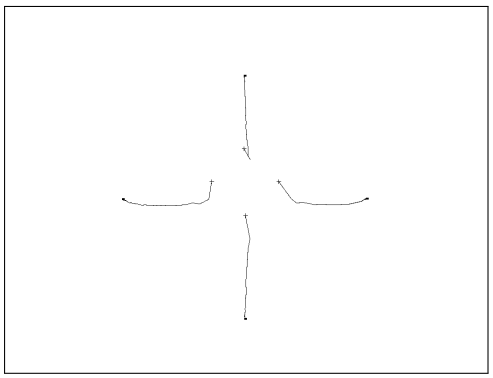


(a) Sensorspur der ersten Lage

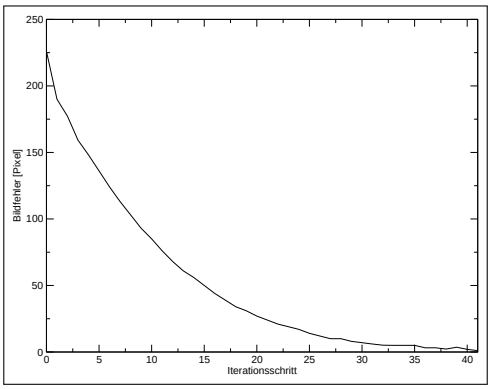


(b) Bildfehler der ersten Lage

Abbildung 3.33: Traditioneller Regler mit dynamischer Jacobimatrix, Lage 1

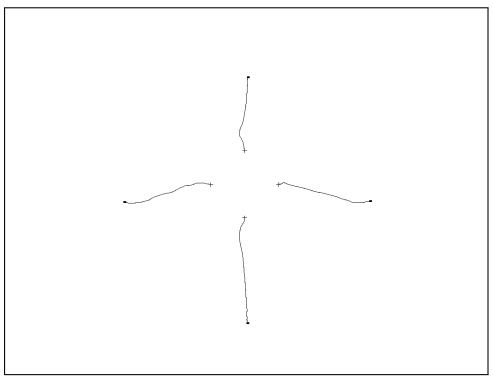


(a) Sensorspur der ersten Lage

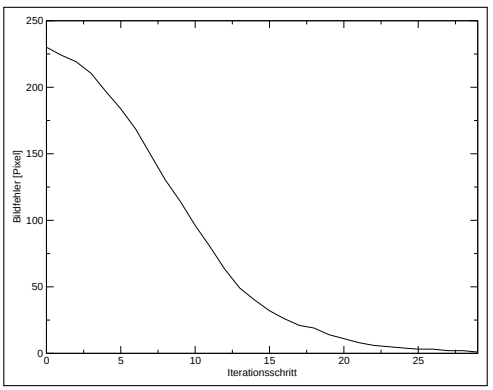


(b) Bildfehler der ersten Lage

Abbildung 3.34: MJP-Regler, Lage 1

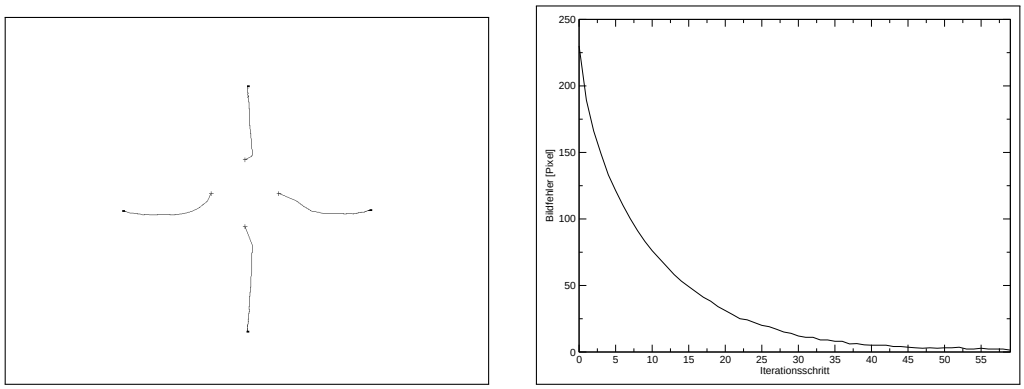


(a) Sensorspur der ersten Lage



(b) Bildfehler der ersten Lage

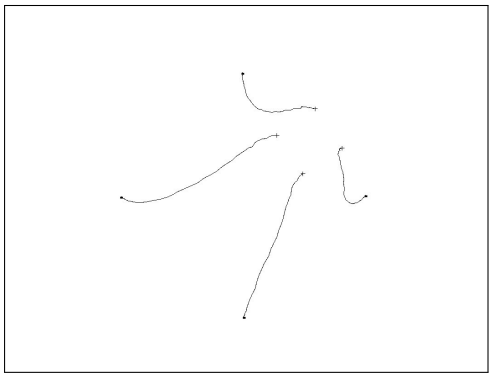
Abbildung 3.35: PMJ-Regler, Lage 1



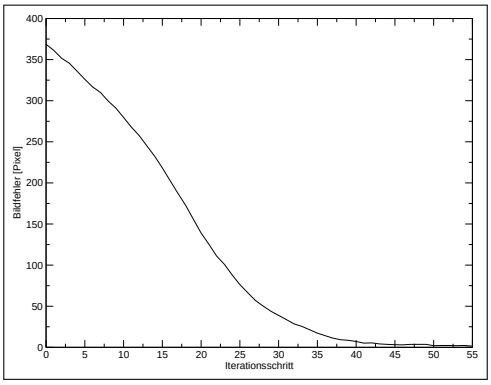
(a) Sensorspur der ersten Lage (b) Bildfehler der ersten Lage

Abbildung 3.36: Regler in Zylinderkoordinaten, Lage 1

Ausgangslage 2: [20, -50, -300, -10°, -10°, -10°]

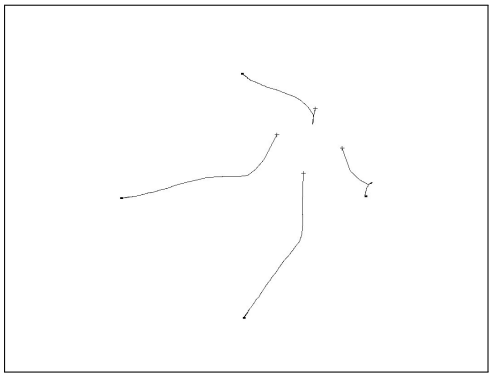


(a) Sensorspur der zweiten Lage

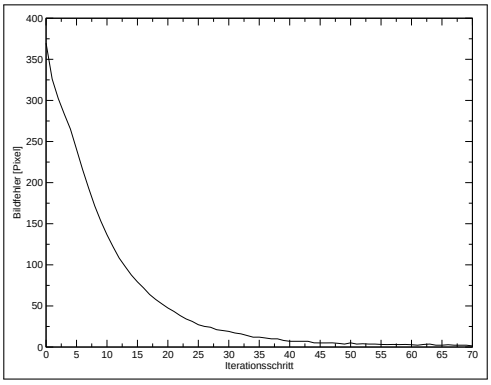


(b) Bildfehler der zweiten Lage

Abbildung 3.37: Traditioneller Regler mit konstanter Jacobimatrix, Lage 2

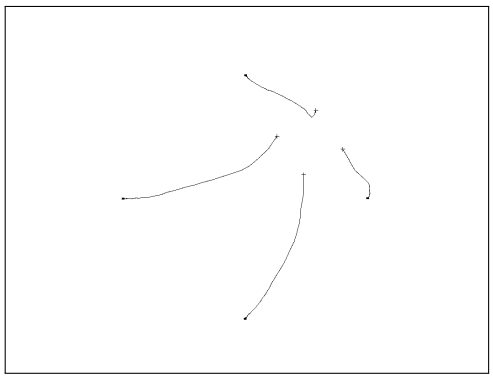


(a) Sensorspur der zweiten Lage

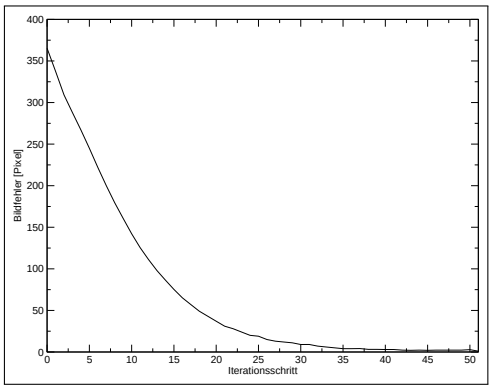


(b) Bildfehler der zweiten Lage

Abbildung 3.38: Traditioneller Regler mit dynamischer Jacobimatrix, Lage 2

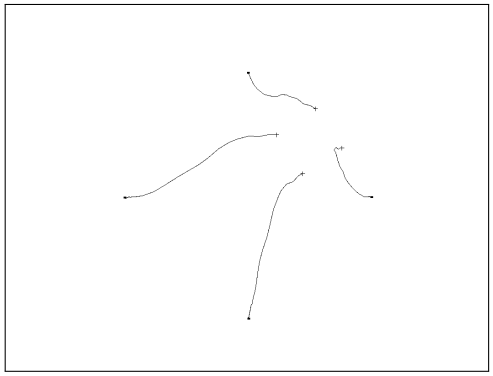


(a) Sensorspur der zweiten Lage

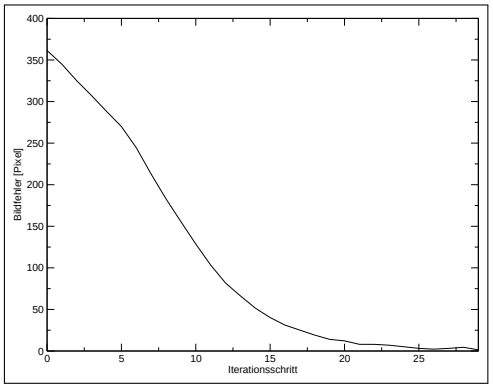


(b) Bildfehler der zweiten Lage

Abbildung 3.39: MJP-Regler, Lage 2



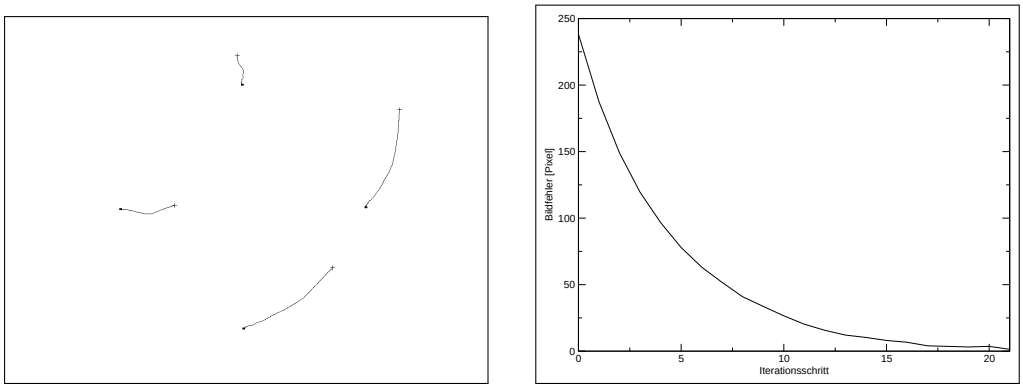
(a) Sensorspur der zweiten Lage



(b) Bildfehler der zweiten Lage

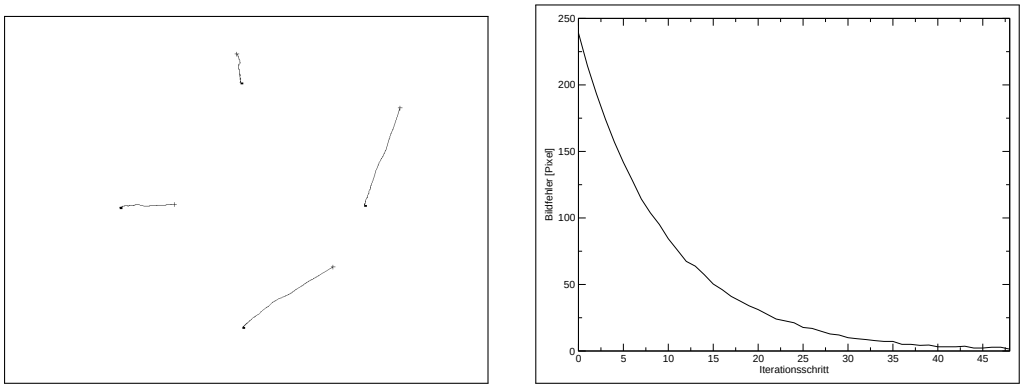
Abbildung 3.40: PMJ-Regler, Lage 2

Ausgangslage 3: [0, 0, 0, -5°, -3°, 23°]



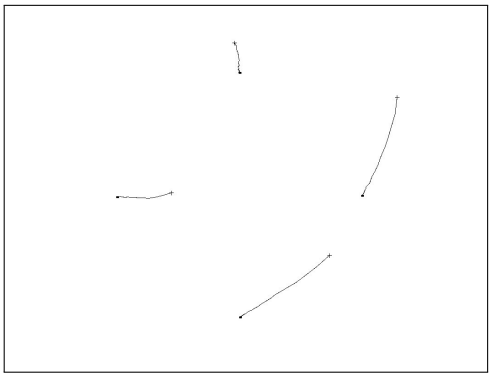
(a) Sensorspur der dritten Lage (b) Bildfehler der dritten Lage

Abbildung 3.41: Traditioneller Regler mit konstanter Jacobimatrix, Lage 3

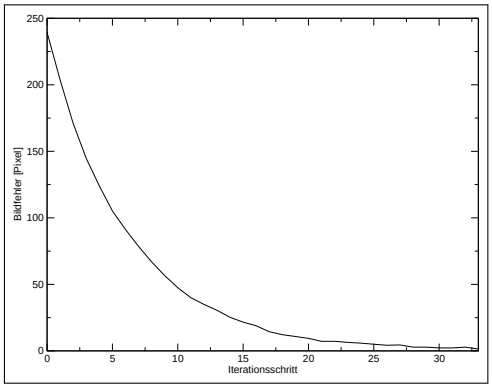


(a) Sensorspur der dritten Lage (b) Bildfehler der dritten Lage

Abbildung 3.42: Traditioneller Regler mit dynamischer Jacobimatrix, Lage 3

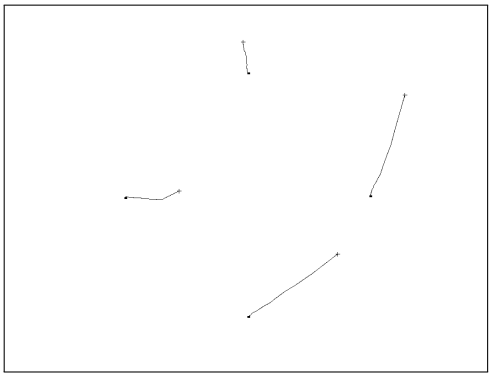


(a) Sensorspur der dritten Lage

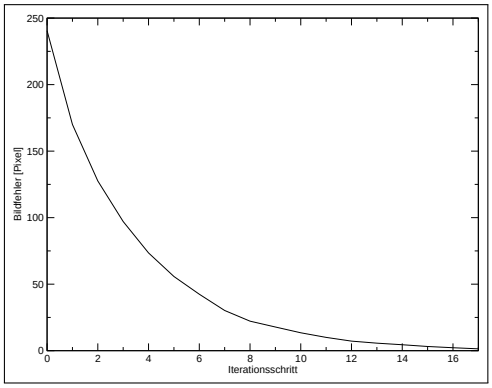


(b) Bildfehler der dritten Lage

Abbildung 3.43: MJP-Regler, Lage 3

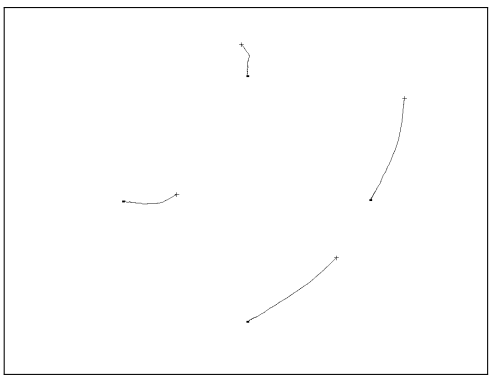


(a) Sensorspur der dritten Lage

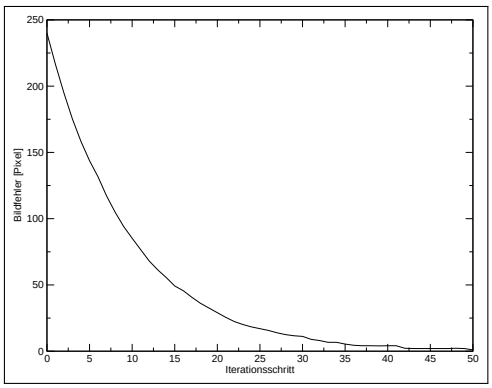


(b) Bildfehler der dritten Lage

Abbildung 3.44: PMJ-Regler, Lage 3



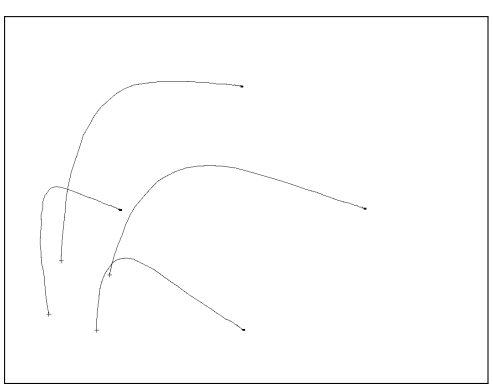
(a) Sensorspur der dritten Lage



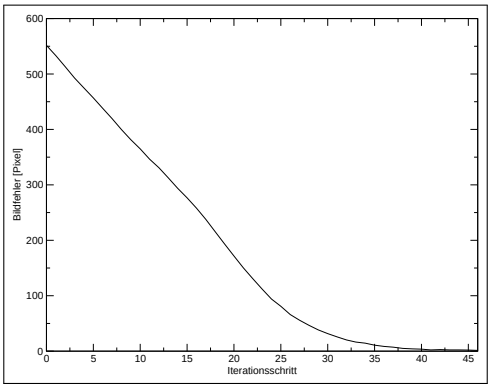
(b) Bildfehler der dritten Lage

Abbildung 3.45: Regler in Zylinderkoordinaten, Lage 3

Ausgangslage 4: [150, 90, -200, 10°, -15°, 30°]

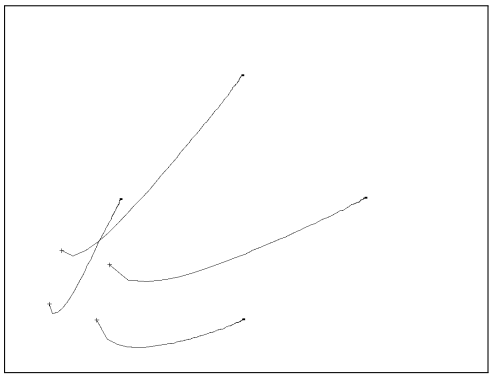


(a) Sensorspur der vierten Lage

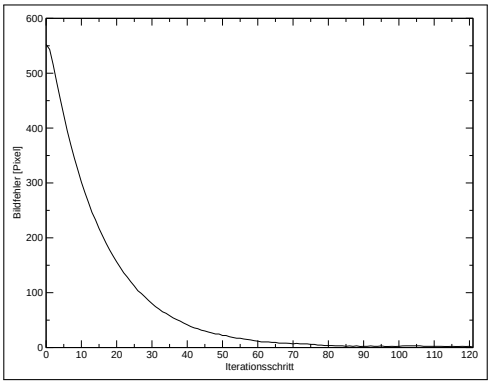


(b) Bildfehler der dritten Lage

Abbildung 3.46: Traditioneller Regler mit konstanter Jacobimatrix, Lage 4

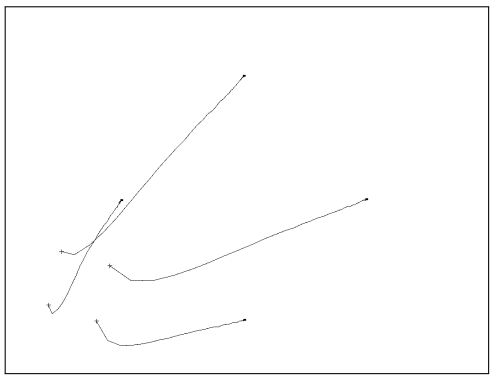


(a) Sensorspur der vierten Lage

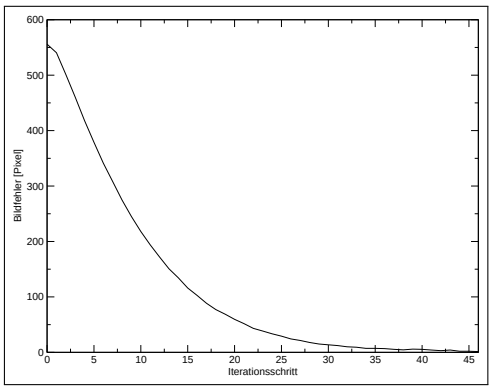


(b) Bildfehler der dritten Lage

Abbildung 3.47: Traditioneller Regler mit dynamischer Jacobimatrix, Lage 4

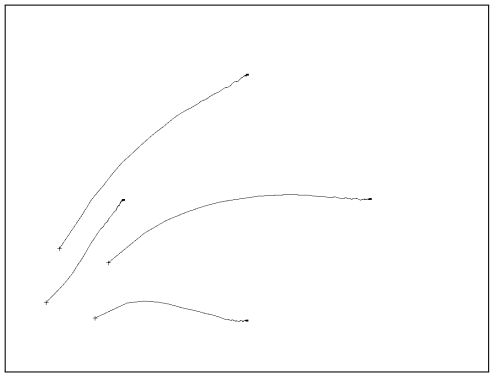


(a) Sensorspur der vierten Lage

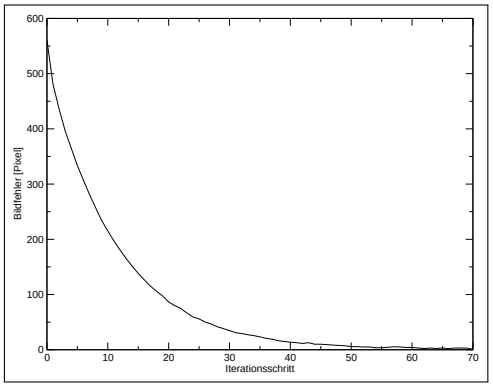


(b) Bildfehler der dritten Lage

Abbildung 3.48: MJP-Regler, Lage 4



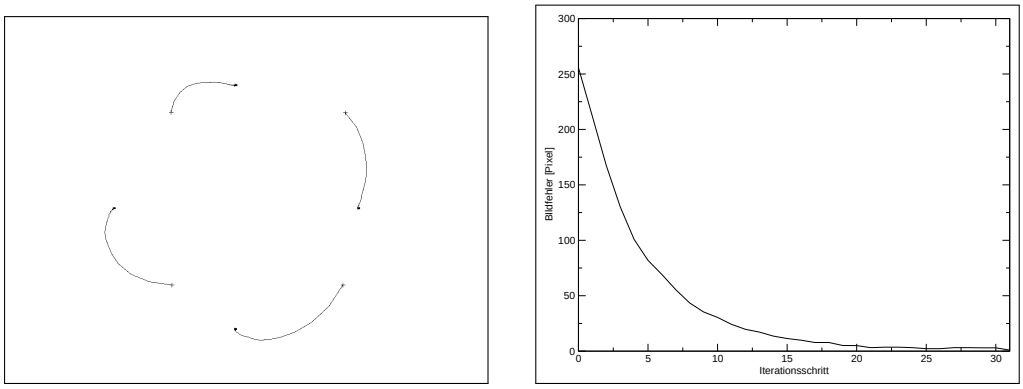
(a) Sensorspur der vierten Lage



(b) Bildfehler der dritten Lage

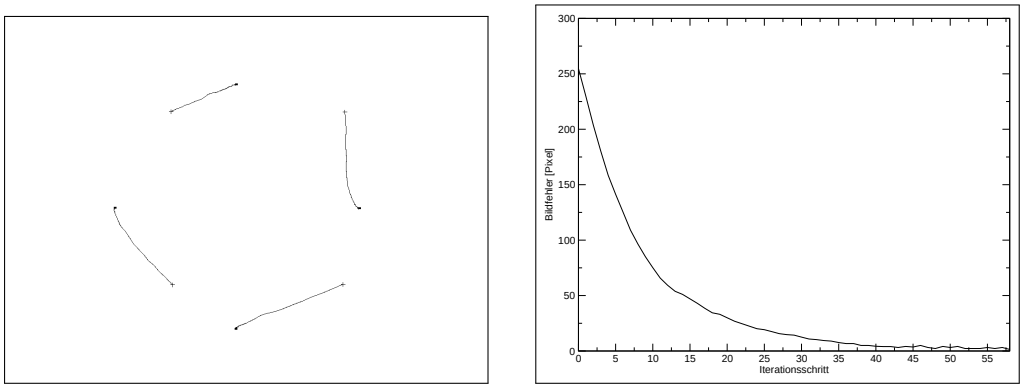
Abbildung 3.49: Regler in Zylinderkoordinaten, Lage 4

Ausgangslage 5: [0, 0, 0, 0°, 0°, 45°]



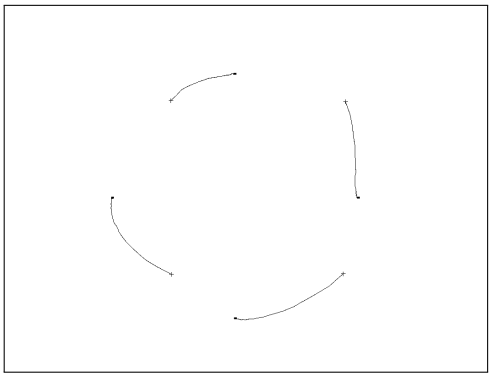
(a) Sensorspur der fünften Lage (b) Bildfehler der fünften Lage

Abbildung 3.50: Traditioneller Regler mit konstanter Jacobimatrix, Lage 5

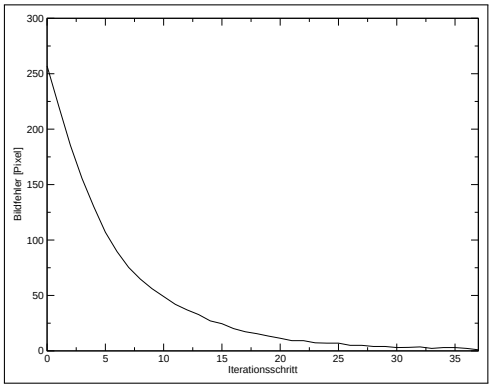


(a) Sensorspur der fünften Lage (b) Bildfehler der fünften Lage

Abbildung 3.51: Traditioneller Regler mit dynamischer Jacobimatrix, Lage 5

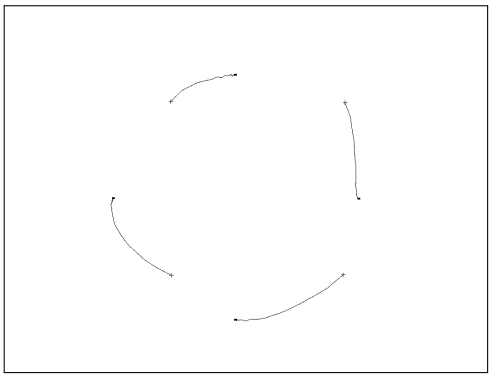


(a) Sensorspur der fünften Lage

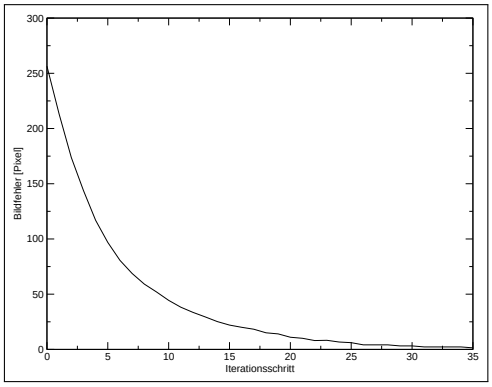


(b) Bildfehler der fünften Lage

Abbildung 3.52: MJP-Regler, Lage 5

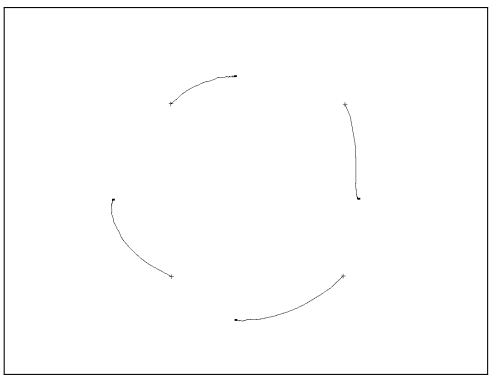


(a) Sensorspur der fünften Lage

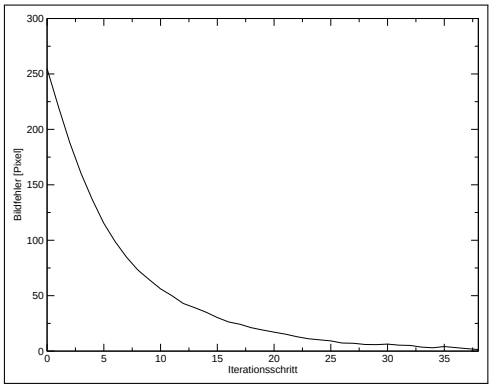


(b) Bildfehler der fünften Lage

Abbildung 3.53: PMJ-Regler, Lage 5



(a) Sensorspur der fünften Lage



(b) Bildfehler der fünften Lage

Abbildung 3.54: Regler in Zylinderkoordinaten, Lage 5

3.5 Bewertung der Regler

Die Ergebnisse der drei Testverfahren werden in Tabelle 3.1 und in Tabelle 3.2 zusammengefasst.

Der Traditionelle Regler mit konstanter Jacobimatrix konvergiert in allen fünf Lagen. Allerdings bewegt er sich in der vierten Lage relativ nahe am Bildrand. In der fünften Ausgangslage lässt sich das in Abschnitt 3.2.3 beschriebene Retreat-Advance-Problem beobachten. Hier bewegt sich der Greifarm am Anfang der Reglung auf das Objekt zu, anstatt eine reine Drehung um die z-Achse durchzuführen. Bei der Multilagen-Simulation erreicht er eine Erfolgsquote von 91.53 %. Auffällig ist, dass sich die Erfolgsquote bei Vergrößerung des Dämpfungsfaktors nur langsam verringert.

Auch der Traditionelle Regler mit dynamischer Jacobimatrix konvergiert in allen Fällen. Dazu musste aber in der vierten Lage der Dämpfungsfaktor reduziert werden. Auch hier wird in der fünften Lage das Retreat-Advance-Problem beobachtet. Der Roboter bewegt sich in den ersten Regelschritten vom Objekt weg. Es wird mit der Multilagen-Simulation eine Erfolgsquote von 98.59 % erreicht.

Der MJP-Regler ist in allen fünf Lagen erfolgreich, ohne das nachträgliche Korrekturen durchgeführt werden mussten. In der fünften Lage wird hier fast eine reine Rotation um die z-Achse durchgeführt. Mit einer Erfolgsquote von 99.27 % liefert er das beste Ergebnis.

Die Experimente zeigen, dass der PMJ-Regler Schwierigkeiten mit Lagen hat, die nahe am Rand liegen und verdreht sind. Er konvergiert nicht in der vierten Lage. Selbst eine Korrektur des Dämpfungsfaktors führt zu keiner Verbesserung. Allerdings liefert er in allen anderen Lagen gute Ergebnisse. Das Retreat-Advance-Problem tritt hier nicht mehr auf. Die Erfolgsquote bei der Multilagen-Simulation beträgt 94.52 %.

Der Regler in Zylinderkoordinaten hat Probleme mit zentralliegenden, verdrehten Posen, die einen größeren Abstand von der Kamera haben. Das lässt sich in Ausgangslage 2 beobachten. Während er in der OpenGL-Simulation noch konvergiert, wird beim realen Robotersystem eine nicht erreichbare Lage angesteuert. Die Multilagen-Simulation ermittelt eine Erfolgsquote von 91.18 %.

Bei allen Reglern werden am Anfang der Reglung recht große Schritte ausgeführt. Das führt z. B. beim Traditionellen Regler mit dynamischer Jacobimatrix dazu, dass in der vierten Ausgangslage die Objektmarkierungen den Bildbereich verlassen. Zum Schluss wird die Schrittweite immer kleiner, so dass es relativ lange dauern kann, bis die Ziellage gefunden wurde. Darum werden im nächsten Abschnitt adaptive Regler betrachtet, die den Dämpfungsfaktor bzgl. der momentanen Umgebung anpassen.

Regler	k	OpenGL-Simulation Ausgangslage					Multilagen	
		1	2	3	4	5	Iter.	Erfolg. [%]
Traditionelle Regler mit konstanter Jacobimatrix	0.2	44	44	23	44	23	32	91.53
Traditionelle Regler mit dynamischer Jacobimatrix	0.1 (0.07)	46	52	45	∞ (81)	47	52	98.59 (99.11)
MJP-Regler	0.15	35	39	31	41	32	37	99.27
PMJ-Regler	0.25	26	26	18	∞	32	38	94.52
Regler in Zylinderkoordinaten	0.1	46	49	49	58	49	52	91.18

Tabelle 3.1: Ergebnisse der Simulatoren mit konstantem Dämpfungsfaktor

Regler	k	Ausgangslage				
		1	2	3	4	5
Traditionelle Regler mit konstanter Jacobimatrix	0.2	49	55	21	46	31
Traditionelle Regler mit dynamischer Jacobimatrix	0.1 (0.07)	63	70	48	∞ (121)	58
MJP-Regler	0.15	41	51	33	46	37
PMJ-Regler	0.25	29	29	17	∞	35
Regler in Zylinderkoordinaten	0.1	59	∞	50	70	38

Tabelle 3.2: Ergebnisse des realen Systems mit konstantem Dämpfungsfaktor

Kapitel 4

Adaptive Regler

4.1 Minimierung nichtlinearer Ausgleichsprobleme

4.1.1 Nichtlineares Ausgleichsproblem

In diesem Kapitel werden Methoden zur Lösung von nichtlinearen Ausgleichsproblemen (engl: *nonlinear least squares problem*) betrachtet. Hierzu wird folgende Definition aus [MNT99] benutzt:

Definition 4.1 (Nichtlineare Ausgleichsprobleme) *Nichtlineare Ausgleichsprobleme werden durch eine Funktion*

$$F(x) = \frac{1}{2}\sum_{i=1}^{m}(f_i(x))^2$$

mit $f_i : \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, m, m \geq n$ *beschrieben, für die ein* $x^\star \in \mathbb{R}^n$ *bestimmt werden soll, so dass* F *minimal ist.*

Es wird dabei zwischen globalen und lokalen Minima unterschieden:

Definition 4.2 (Globales und lokales Minimum) *Sei* $F : \mathbb{R}^n \to \mathbb{R}$ *eine Funktion. Falls* $x^\star \in \mathbb{R}^n$ *so gewählt ist, dass*

$$x^\star = \operatorname*{argmin}_{x \in \mathbb{R}^n} F(x),$$

so spricht man von einem globalen Minimum. Falls $x^\star \in \mathbb{R}^n$ *für ein* $\delta > 0$ *so gewählt ist, dass*

$$F(x^\star) \leq F(x) \text{ für } \|x - x^\star\| < \delta, x \in \mathbb{R}^n$$

so spricht man von einem lokalem Minimum.

Zum Lösen nichtlineare Ausgleichsprobleme werden unter anderem Methoden benutzt, die das Minimum iterativ bestimmen. Das heißt, dass die Suche nach dem Minimum $x^\star$ von einem Startpunkt x_0 beginnt und in einer Folge von Vektoren $x_1, x_2, \ldots$ gegen $x^\star$ konvergiert. Nach jedem Iterationsschritt soll gelten, dass

$$F(x_{k+1}) < F(x_k).$$

Diese Methoden lassen sich anhand ihrer Konvergenzgeschwindigkeit charakterisieren [DH91]:

Definition 4.3 (Konvergenzgeschwindigkeit) *Eine Folge $\{x_k\}_{k\in\mathbb{N}}$ im $\mathbb{R}^n$ konvergiert mit der Ordnung $p \geq 1$ gegen $x^\star$, falls eine Konstante $C \leq 0$ existiert, so dass*

$$\|x_{k+1} - x^\star\| \leq C\|x_k - x^\star\|^p.$$

Für $p = 1$ wird noch $C < 1$ vorausgesetzt. Es wird dann von linearer Konvergenz, bei $p = 2$ von quadratischer Konvergenz gesprochen. Die Folge $\{x_k\}_{k\in\mathbb{N}}$ konvergiert superlinear, falls eine Folge $\{C_k\}_{k\in\mathbb{N}} \geq 0$ mit $\lim_{k\to\infty} C_K = 0$ existiert, mit

$$\|x_{k+1} - x^\star\| \leq C_k\|x_k - x^\star\|^p.$$

4.1.2 Gradientenverfahren

Eine iterative Methode zum Lösen nichtlinearer Ausgleichsprobleme ist das sogenannte *Gradientenverfahren* (engl.: *steepest descent method*). Dabei wird das Minimum in Richtung des negativen Gradienten angenähert.

Seien f und F wie in Definition 4.1.1 gewählt. Zuerst wird eine Taylorentwicklung erster Ordnung für f betrachtet:

$$f(x + u) = f(x) + J(x)u + O(\|h\|^2),$$

wobei $u \in \mathbb{R}^n$ und $J(x) \in \mathbb{R}^{m\times n}$ mit

$$(J(x))_{ij} = \frac{\partial f_i}{\partial x_j}(x),$$

die Jacobimatrix von f ist. Als nächstes wird die erste Ableitung von F gebildet. Es ergibt sich

$$F'(x) = J^T(x)f(x),$$

wobei F' der Gradient ist.

Durch iteratives Ausführen der folgende Anweisung wird das Minimum von F angenähert:

$$x_{n+1} = x_n - kJ^T(x_n)f(x_n)$$

Dabei ist $0 < k \leq 1$ ein Dämpfungsfaktor, der auf geeignete Weise gewählt werden muss. Dazu gibt es diverse Verfahren [Fle87]. Wird k zu groß gewählt, so kann es passieren, dass der neu berechnete Wert über das gesuchte Minimum hinaus geht. Der Fehler kann wieder größer werden. Es ist auch möglich, dass die berechneten Werte zwischen zwei Werten pendeln, so dass das Minimum nicht erreicht wird.

Ist k zu klein gewählt, so können die Schritte bis zum Minimum immer kleiner werde, so dass es unter Umständen sehr lange dauern kann, bis der Zielwert erreicht worden ist. Die Konvergenzgeschwindigkeit des Gradientenabstiegs ist linear [NW99].

4.1.3 Gauss-Newton-Methode

Ein weiteres Verfahren, um nichtlineare Ausgleichsprobleme zu lösen ist die sogenannte *Gauß-Newton-Methode*. Seien F und f wieder wie in Definition 4.1.1. Es wird abermals ein lineares Modell der Funktion f mit Hilfe der Taylorentwicklung erster Ordnung gebildet:

$$f(x+u) \approx \ell(u) = f(x) + J(x)u, \tag{4.1}$$

mit einem kleinen $\|u\|_2$, $u \in \mathbb{R}^n$. Durch einsetzen von (4.1) in F ergibt sich

$$\begin{aligned} F(x+u) \approx L(u) &= \frac{1}{2}\ell^T(u)\ell(u) \\ &= \frac{1}{2}f^T f + u^T J^T f + \frac{1}{2}u^T J^T J u \\ &= F(x) + u^T J^T f + \frac{1}{2}u^T J^T J u \end{aligned}$$

Der Gauß-Newton-Schritt u_{gn} ist dann

$$u_{gn} = \underset{u \in \mathbb{R}^n}{\operatorname{argmin}}(L(u))$$

Dazu wird die erste Ableitung von $L(u)$ gebildet:

$$L'(u) = J^T f + J^T J u$$

Der Gauß-Newton-Schritt ergibt sich dann durch lösen der Gleichung

$$J^T J u_{gn} = -J^T f \tag{4.2}$$

4.2 Trust-Region-Regler

Der Dämpfungsfaktor k beim Traditionellen Regler muss sehr klein gewählt werden, damit bei einer Regelung nicht das Objekt aus dem Sichtbereich der Kamera gerät. Dadurch werden die Regelschritte zum Schluss immer kleiner, weshalb es dann relativ lange dauert, bis die Zielposition erreicht wird.

Um den Regler zu verbessern, wird im Folgenden der Dämpfungsfaktor dynamisch gewählt. Dadurch soll gewährleistet werden, dass am Anfang die Merkmale nicht aus dem Sichtbereich geraten und zum Schluss die Regelschritte nicht zu klein werden. Dies wird mit dem Trust-Region-Regler aus [Sie99] realisiert.

4.2.1 Beschreibung des Reglers

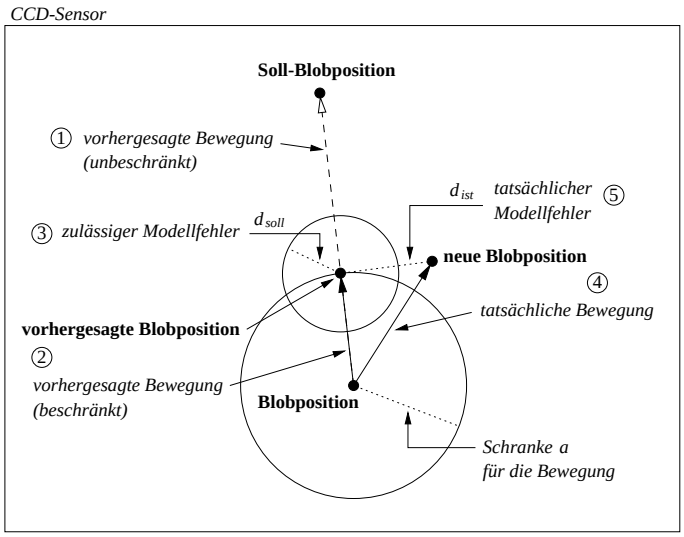


Abbildung 4.1: Beschreibung des Trust-Region-Reglers (Grafik aus [Sie99])

Das Prinzip des Trust-Region-Reglers wir im Folgenden anhand der Abbildung 7 beschrieben:

① Mit der unbeschränkten Stellgröße $\Delta u_{n-1} = J^+_{n-1}(-\Delta y_{n-1})$ läßt sich die Länge $\ell_{n-1} \in \mathbb{R}_{>0}$ der maximalen Bewegung der Objektmarkierungen auf dem Sensor berechnen:

$$\ell_{n-1} := \max_{i=1,\ldots,M} \left\| \begin{bmatrix} (J_{n-1}\Delta u_{n-1})_{2i-1} \\ (J_{n-1}\Delta u_{n-1})_{2i} \end{bmatrix} \right\|_2$$

② Die Stellgröße soll nun so beschränkt werden, dass sich die vorhergesagten Blobpositionen $\hat{y}_n$ auf bzw. innerhalb der sogenannten *Trust Region* befinden. Dies wird mit der Schranke $\alpha \in \mathbb{R}$ oder auch α_{n-1} (α zum Zeitpunkt $n-1$) erreicht. Zusammen mit der Länge ℓ_{n-1} wird dann ein Dämpfungsfaktor $\lambda_{n-1} \in (0, 1]$ bestimmt, der die Stellgröße wie folgt beschränkt:

$$\begin{aligned} u_{n-1} &:= \lambda_{n-1} \cdot \Delta u_{n-1} \\ &= \min\{1, \frac{\alpha_{n-1}}{\ell_{n-1}}\} \cdot J^{+}_{n-1}(-\Delta y_{n-1}) \end{aligned}$$

③ Bei der tatsächlichen Bewegung der Blobs wird ein Modellfehler von $d_{\text{soll}} \in \mathbb{R}_{>0}$ als zulässig betrachtet.

④ Der Roboterarm wird nun mit der berechneten Stellgröße angesteuert. Die tatsächlichen Blobpositionen y_n werden dann ausgelesen.

⑤ Es soll nun der tatsächliche Modellfehler $d_{\text{ist}} \in \mathbb{R}_{\geq 0}$ oder auch d_n (tatsächlicher Bildfehler zum Zeitpunkt n) bestimmt werden. Dazu werden zuerst die Positionen berechnet, an denen sich die Objektmarkierungen nach Ausführung des letzten Regelschrittes theoretisch befinden müssten:

$$\hat{y}_n := y_{n-1} + J_{n-1} u_{n-1}$$

Der momentane Bildfehler d_n der M Markierungen ist dann

$$d_n := \max_{i=1,\ldots,M} \left\| \begin{bmatrix} (\hat{y}_n)_{2i-1} \\ (\hat{y}_n)_{2i} \end{bmatrix} - \begin{bmatrix} (y_n)_{2i-1} \\ (y_n)_{2i} \end{bmatrix} \right\|_2 . \tag{4.3}$$

Der relative Modellfehler $r_n \in \mathbb{R}_{>0}$ berechnet sich durch

$$r_n := \frac{d_{\text{soll}}}{d_n}, \text{ für } d_n > 0.$$

Für den nächsten Regelschritt wird α_n so angepasst, dass $d_{n+1} \approx d_{\text{soll}}$ gilt. Dabei wird α_n abhängig von α_{n-1} gesetzt und durch den Werten $\alpha_{\min}, \alpha_{\max} \in \mathbb{R}_{>0}$ beschränkt. Für die α-Adaption werden zwei Varianten benutzt.

In der ersten Variante wird ein Akzeptanzintervall mit den Grenzen $r_{\min}, r_{\max} \in \mathbb{R}_{>0}$ festgelegt. Falls r_n sich innerhalb dieses Intervalls befindet, so wird α_{n-1} beibehalten. Liegt r_n außerhalb der Grenzen, so

wird α_{n-1} verdoppelt oder halbiert. Die erste Variante setzt sich also wie folgt zusammen:

$$\alpha_n = \begin{cases} \frac{\alpha_{n-1}}{2} & \text{für } r_n < r_{\min} \\ \alpha_{n-1} & \text{für } r_{\min} \leq r_n \leq r_{\max} \\ 2 \cdot \alpha_{n-1} & \text{für } r_{\max} < r_n \end{cases}$$

Bei der zweiten Variante der α-Adaption wird α_n durch Multiplikation von α_{n-1} mit dem relativen Modellfehler angepasst. Für $d_n \neq 0$ setze

$$\alpha_n = \alpha_{n-1} \cdot \frac{d_{\text{soll}}}{d_n} = \alpha_{n-1} \cdot r_n. \tag{4.4}$$

Falls $d_n = 0$, so wählt man für α_n die maximale Verstärkung, z. B. $\alpha_{\max}$. Um zu verhindern, dass bei dieser Variante α_n zu stark erhöht wird, ist sein Anstieg zusätzlich durch einen Faktor beschränkt. Dieser beträgt hier 2.0. Für alle $n > 1$ wird gefordert, dass $\alpha_n \leq 2.0 \cdot \alpha_{n-1}$.

Als Modell wird für diesen Regler die dynamische Jacobimatrix benutzt. Zur Bestimmung der Stellgröße wird der Traditionelle Regler mir dynamische Jacobimatrix verwendet. Allerdings wird der Dämpfungsfaktor hier dynamisch bestimmt.

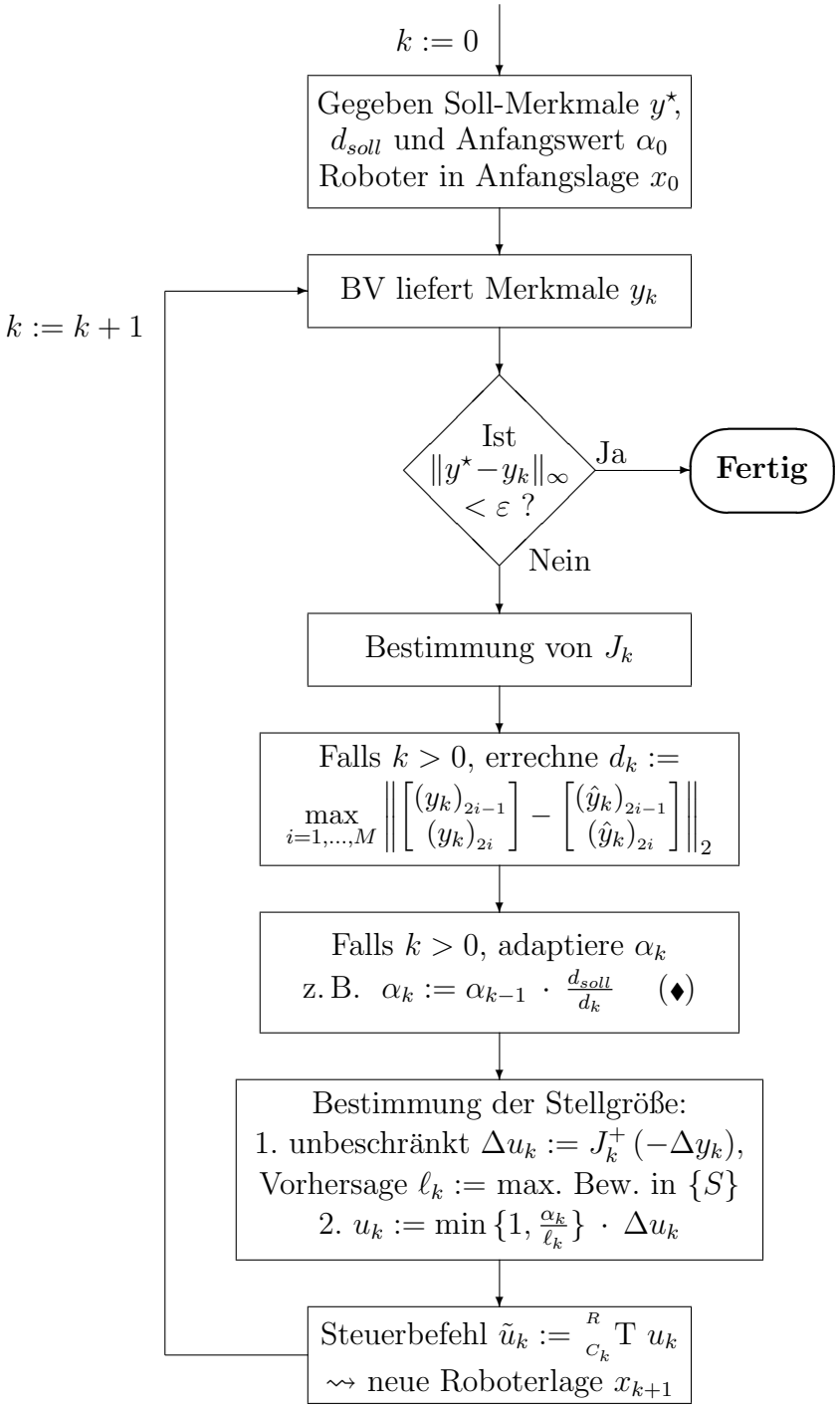


Abbildung 4.2: Ablaufplan einer Regelung mit dem Trust-Region-Regler (Grafik aus [Sie99])
(♦) abhängig vom gewählten Schema der α-Adaption und Reglerparametern

4.3 Der Dog-Leg-Regler

Im folgenden Abschnitt wird das sogenannte Dog-Leg-Verfahren beschrieben. Ausgehend von diesem Verfahren wird dann im zweiten Teil ein adaptiver Regler für das Visual-Servoing entwickelt. Dabei ist die Vorgehensweise ähnlich der von Siebel zur Herleitung des Trust-Region-Reglers.

4.3.1 Powells Dog-Leg-Methode

Die sogenannte Dog-Leg-Methode[1] von Michael J. D. Powell wird z. B. in [Pow70], [MNT99] und [Fle87] beschrieben. Folgende Beschreibung stammt aus [MNT99]. Genau wie beim Trust-Region-Verfahren wird auch hier eine Trust Region dynamisch während den durchgeführten Schritten bzgl. eines linearen Modells angepasst. Dabei wird neben dem Gauß-Newton-Verfahren auch das Gradientenverfahren benutzt.

Sei $f : \mathbb{R}^n \to \mathbb{R}^m$ und $u_{\text{sd}} = -J^T(x)f(x)$ die Richtung des Gradienten. Setze $a = \kappa u_{\text{sd}}$ mit

$$\kappa = \frac{\|u_{\text{sd}}\|_2^2}{\|J(x)u_{\text{sd}}\|_2^2} \tag{4.5}$$

Mit $b = u_{\text{gn}}$ wird der Gauß-Newton-Schritt bezeichnet, wobei u_{gn} wie in (4.2) gewählt wird. α sei wieder der Radius der Trust Region. Mit u_{dl} wird der Dog-Leg-Schritt bezeichnet.

Die Dog-Leg-Schritt wird wie folgt berechnet:

$$\begin{aligned}
&\texttt{Falls } \|u_{\text{gn}}\|_2 \leq \alpha \\
&\quad u_{\text{dl}} = u_{\text{gn}} \\
&\texttt{sonst falls } \|\kappa u_{\text{sd}}\|_2 \geq \alpha \\
&\quad u_{\text{dl}} = (\alpha/\|u_{\text{sd}}\|_2)u_{\text{sd}} \\
&\texttt{sonst} \\
&\quad u_{\text{dl}} = \kappa u_{\text{sd}} + \beta(u_{\text{gn}} - \kappa u_{\text{sd}}) \\
&\qquad \texttt{mit } \beta\texttt{, so dass } \|u_{\text{dl}}\|_2 = \alpha
\end{aligned} \tag{4.6}$$

Der letzte Fall wird in Abbildung 4.3 illustriert. Um β zu bestimmen, setze $c = a^T(b - a)$. Sei

$$\psi(\beta) = \|a + \beta(b - a)\|_2^2 - \alpha^2$$

[1] Dogleg ist ein Begriff aus dem Golf. Er beschreibt einen Fairway, der nach links oder rechts gebogen ist. Das entspricht in Abbildung 4.3 einer Linie vom Mittelpunkt des Kreises entlang des Vektors a und dann zum Endpunkt von u_{dl}.

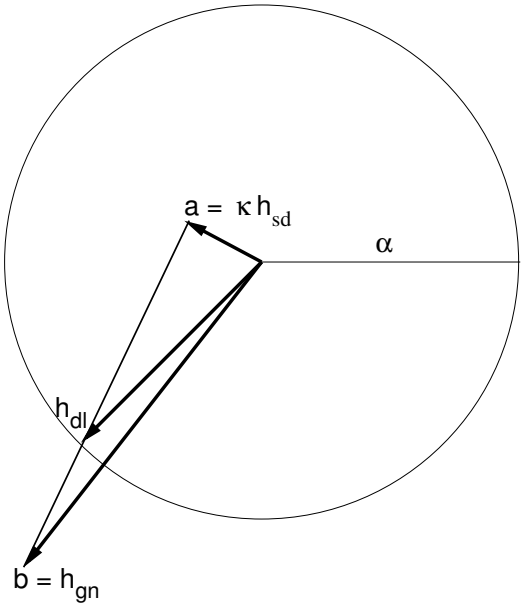


Abbildung 4.3: Bestimmung des Dog-Leg-Schrittes (Grafik nach [MNT99])

β muss also so gewählt werden, dass $\psi(\beta) = 0$ gilt. Das wird durch folgende Fallunterscheidung erreicht:

$$\beta = \begin{cases} \frac{-c+\sqrt{c^2+\|b-a\|_2^2(\alpha^2-\|a\|_2^2)}}{\|b-a\|_2^2}, & \text{für } c \leq 0 \\ \\ \frac{\alpha^2-\|a\|_2^2}{c+\sqrt{c^2+\|b-a\|_2^2(\alpha^2-\|a\|_2^2)}}, & \text{sonst.} \end{cases} \tag{4.7}$$

Nach jedem Iterationsschritt wird der relative Modellfehler r_n bestimmt. Er berechnet sich aus folgendem Quotienten:

$$r_n = \frac{F(x) - F(x + u_{\text{dl}})}{L(0) - L(h_{\text{dl}})}$$

mit dem linearen Modell

$$L(u) = \frac{1}{2}\|f(x) - J(x)u\|^2$$

wobei F wie in Definition 4.1.1 gewählt ist.

Der so berechnete relative Modellfehler wird benutzt, um die Trust Region anzupassen:

$$\alpha_{n+1} = \begin{cases} \max\{\alpha_n, 3 \cdot \|u_{\text{dl}}\|\}, & \text{für } r_n > \frac{3}{4} \\ \frac{\alpha_n}{2}, & \text{für } r_n < \frac{1}{4} \\ \alpha_n, & \text{sonst.} \end{cases} \tag{4.8}$$

Die Trust Region wird solange verändert, bis der Dog-Leg-Schritt der letzten Bedingung in (4.8) genügt. Erst dann wird der nächste Iterationsschritt durchgeführt.

Falls das lineare Modell gut mit den tatsächlichen Werten übereinstimmt, so wird die Trust Region vergrößert. Dabei nähert sich der Dog-Leg-Schritt immer mehr dem Gauß-Newton-Schritt an. Bei einem ungenauen Modell wird die Trust Region verkleinert. Dabei entspricht der Dog-Leg-Schritt immer mehr dem Gradientenabstieg. Im Gegensatz zum Levenberg-Marquardt-Algorithmus (siehe z. B. [MNT99]) wird der Modellfehler benutzt, um die Trust Region anzupassen und nicht den Dämpfungsfaktor. Einsatz findet die Dog-Leg-Methode beim Lösen von nichtlinearen Gleichungssystemen.

4.3.2 Herleitung des Reglers

Da erst nach Durchführung eines Regelschrittes Aussagen über die Genauigkeit eines Modells gemacht werden können, muss der Regler entsprechend modifiziert werden. Außerdem muss für die Berechnung immer das Bildmerkmal benutzt werden, dass die größte vorrausgesagte Bewegung besitzt. Dadurch soll sichergestellt werden, dass alle Bildmerkmale innerhalb der Trust Region bleiben.

Die Berechnung des Gauß-Newton-Schritts u_{gn} entspricht der ungedämpften Variante des Traditionellen Reglers in Kapitel 3, also

$$u_{\mathrm{gn}_n} = -J_n^+ \Delta y_n$$

Der Schritt des Gradientenabstiegs wird durch

$$u_{\mathrm{sd}_n} = -J_n^T \Delta y_n$$

berechnet. J_n ist jeweils wieder die dynamische Bildjacobimatrix aus (2.5). Die Jacobimatrix bildet auch das von der Dog-Leg-Methode benötigte lineare Modell.

Der Faktor κ_n zum Zeitpunkt n zur Dämpfung des Gradientenabstiegs wird wie folgt bestimmt:

$$\kappa_n = \frac{\|u_{\mathrm{sd}_n}\|_2^2}{\ell_{\mathrm{sd}_n}}$$

mit

$$\ell_{\mathrm{sd}_n} = \max_{i=0,\ldots,M} \left\| \begin{pmatrix} (\Delta\hat{y}_{\mathrm{sd}_n})_{2i} \\ (\Delta\hat{y}_{\mathrm{sd}_n})_{2i+1} \end{pmatrix} \right\|_2^2,$$

die maximal vorhergesagte Bewegung einer Objektmarkierung durch den Gradientenabstieg. $\Delta\hat{y}_n$ sind die durch das Modell berechneten Änderungen der Bildmerkmale.

Sei weiter

$$\ell_{\mathrm{gn}_n} = \max_{i=0,\ldots,M} \left\| \begin{pmatrix} (\Delta\hat{y}_{\mathrm{gn}_n})_{2i} \\ (\Delta\hat{y}_{\mathrm{gn}_n})_{2i+1} \end{pmatrix} \right\|_2^2,$$

die maximale vorhergesagte Änderung der Bildmerkmale durch den Gauß-Newton-Schritt.

Anhand der Fallunterscheidung von (4.6) wird dann der Dog-Leg-Schritt berechnet. Falls also $\ell_{\mathrm{gn}_n} \leq \alpha_n$, so wird $u_{\mathrm{dl}_n} = u_{\mathrm{gn}_n}$ gesetzt. Sonst ist $u_{\mathrm{dl}_n} = (\alpha_n / \|u_{\mathrm{sd}}\|_2) u_{\mathrm{sd}}$, falls $\ell_{\mathrm{sd}_n} \geq \alpha_n$ gillt. Treffen beide Fälle nicht zu, so ist $u_{\mathrm{dl}_n} = \kappa u_{\mathrm{sd}_n} + \beta_n (u_{\mathrm{gn}_n} - \kappa u_{\mathrm{sd}_n})$. Dabei wird bei der Berechnung des Faktors β_n in (4.7) die Objektmarkierung betrachtet, die die größte vorhergesagte Bewegung besitzt.

Zur Adaption der Trust Region wird statt (4.8) die zweite Variante des Trust-Region-Reglers in (4.4) benutzt. d_n ist wieder der momentane Fehler des linearen Modells. Mit dem Wert d_{soll} wird dann der relative Bildfehler r_n mit (4.3) berechnet.

Der Ablauf der Reglung wird mit folgendem Algorithmus zusammengefasst:

`Setze` $n := 0$; $\alpha_n = \alpha_{\mathrm{start}}$; $y^\star$
`Bestimme momentane Bildmerkmale` y_n
`Solange` $\|\Delta y_n\|_\infty \geq \varepsilon$
  `Berechne` J_n
  `Falls` $n > 0$
    `Bestimme den relativen Modellfehler` r_n `mit` (4.3)
    `Adaptiere` α_n `mit` (4.4)
  `Berechne` $u_{\mathrm{sd}_n} = -J^T \Delta y_n$, $\kappa_n = \|u_{\mathrm{sd}_n}\| / \ell_{\mathrm{sd}_n}$ `und` $u_{\mathrm{sd}_n} = -J^+ \Delta y_n$
  `Berechne` u_{dl_n} `mit` (4.6)
  `Sende den Steuerbefehl` u_{dl_n} `an den Roboter`
  `Bestimme` y_{n+1} `und setze` $n := n + 1$

4.4 Validierung der Regler

Die beiden adaptiven Regler sollen in diesem Abschnitt validiert werden. Zusätzlich werden sie mit den Reglern in Kapitel 3 kombiniert. Dazu wird beim Trust-Region-Regler die Berechnung der ungedämpften Stellgröße durch die jeweilige unbeschränkte Variante des Reglers ersetzt. Zur Alphaadaption wird dann das dazugehörige Modell benutzt. Da das MJP-Modell ein inverses Modell ist, wird hier die dynamische Jacobimatrix zur Anpas-

sung der Trust Region benutzt. Es wird in den folgenden Experimenten die zweite Variante der Alphaadaption benutzt.

Ähnlich wird beim Dog-Leg-Regler verfahren. Hier wird der Gauß-Newton-Schritt auch durch den jeweiligen Regler ersetzt. Der Gradientenabstieg wird bei allen Kombinationen beibehalten. Allerdings musste der Dämpfungsfaktor zusätzlich mit dem Faktor 0.5 multipliziert werden, um eine Konvergenz sicherzustellen. Andernfalls ist die maximale Schrittweite der Bildmerkmale zu groß, so dass die zweite Bedingung in (4.6) während den Experimenten immer erfüllt wurde. Bedingt durch die lineare Konvergenzgeschwindigkeit des Gradientenabstiegs, erreicht der Regler die getesteten Ziellagen dann nicht.

Zur Alphaadaption werden wieder die zugehörigen Modelle benutzt. Auch hier wird beim MJP-Modell die dynamische Jacobimatrix benutzt.

Zunächst werden mit der Multilagen-Simulation die optimale Startgröße der Trust Region bestimmt. Dazu wird in den Experimenten das α fest gewählt. Mit dem so erhaltenen Wert wird dann der zulässige Modellfehler d_{soll} ermittelt. Die Parameter der Alphaadaption werden dabei wie folgt gewählt: $\alpha_{\min} = \alpha_{\text{start}}$, $\alpha_{\max} = 0.5$.

Die beiden Regler und deren Variationen werden dann mit den so gewählten Werten in der OpenGL-Simulation und am realen System validiert.

4.4.1 Multilagen-Simulation

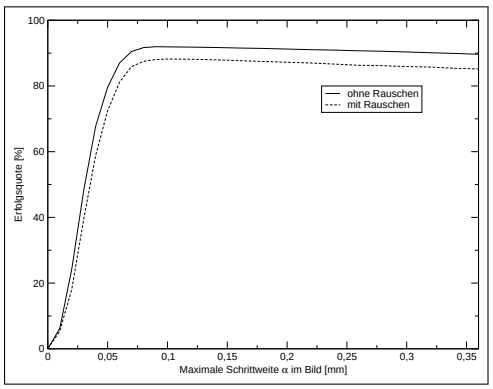
Im ersten Schritt sollen mit der Multilagen-Simulation die Parameter α_{start} und d_{soll} bestimmt werden. Dazu wird zuerst der optimale Wert für einen konstanten Radius α der Trust Region bestimmt. Es wird also keine Alphaadaption durchgeführt. Für α_{start} wird dann der passende Wert für d_{soll} bestimmt.

Der Trust-Region-Regler mit konstanter Jacobimatrix

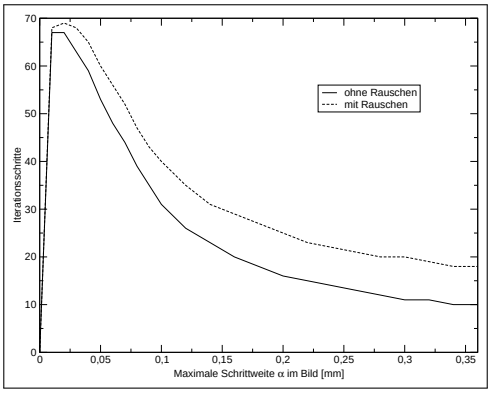
In Abbildung 4.4(a) ergibt sich bei konstanter Größe der Trust Region für $\alpha_{\text{start}} = 0.09$ mit einer Erfolgsquote von 91.95 % der beste Wert. Die durchschnittliche Iterationszahl liegt hier bei 35 Schritten (siehe Abbildung 4.4(b)).

Mit dem so bestimmten Startwert der Trust Region wird nun ein Simulationsdurchlauf mit Alphaadaption durchgeführt. Bei einem gewünschten Modellfehler von 0.18 (siehe Abbildung 4.4(c)) braucht der Regler durchschnittlich 18 Regelschritte, um eine Zielpose zu erreichen. Die Erfolgsquote liegt hier bei 91.43 %. Die Abbildungen zeigen, dass dieser Regler stärker von dem Rauschen beeinflusst wird als der traditionelle Ansatz. Dafür konvergiert dieser Ansatz fast doppelt so schnell und hat eine fast identische

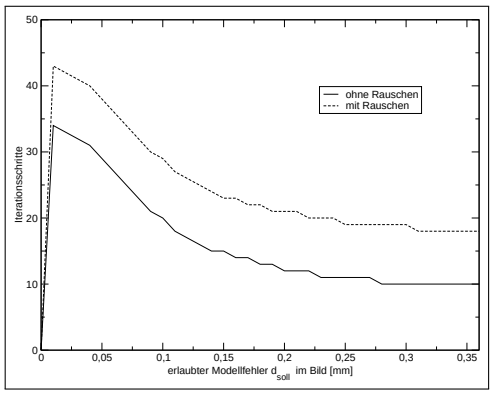
Erfolgsquote.



(a) Erfolgsquote bei konstantem α



(b) Iterationsschritte bei konstantem α



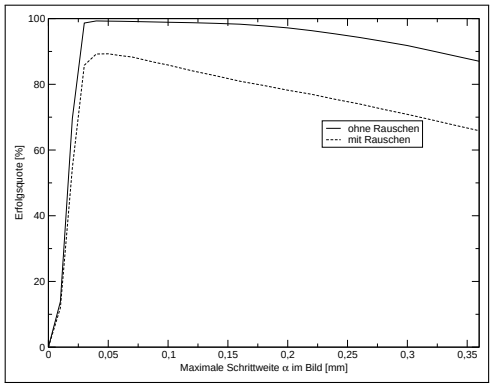
(c) Iterationsschritte bei für unterschiedliche d_{soll} mit $\alpha_{\text{start}} = 0.09$

Abbildung 4.4: Multilagen-Simulation mit konstanter Jacobimatrix

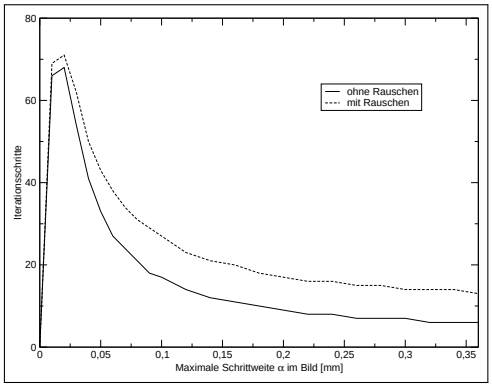
Der Trust-Region-Regler mit dynamischer Jacobimatrix

Für den Ansatz mit dynamischer Jacobimatrix wird in Abbildung 4.5(a) ein Startwert von $\alpha_{\text{start}} = 0.07$ bestimmt. Die Erfolgsquote liegt hier bei 99.18 % mit einer durchschnittlichen Anzahl von 24 Regelschritten. Bereits hier liefert der Regler schon bessere Ergebnisse als der traditionelle Ansatz mit dynamischer Jacobimatrix.

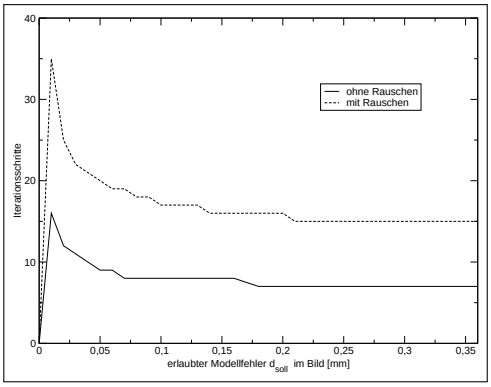
In Abbildung 4.5(c) erhält man für $d_{\text{soll}} = 0.04$ mit aktiver Alphaadaption bei gleicher Erfolgsquote eine mittlere Iterationsanzahl von 10 Schritten. Hier ergibt sich eine deutliche Leistungssteigerung durch den Trust-Region-Ansatz. Aber auch hier hat das Gaußsche Rauschen einen größeren Einfluss auf den Regler.



(a) Erfolgsquote bei konstantem α



(b) Iterationsschritte bei konstantem α



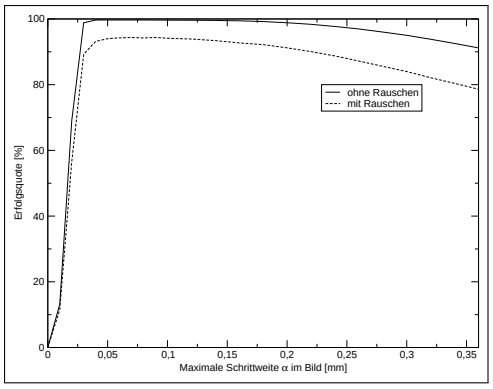
(c) Iterationsschritte bei für unterschiedlichem d_{soll} mit $\alpha_{\text{start}} = 0.07$

Abbildung 4.5: Multilagen-Simulation mit dynamischer Jacobimatrix

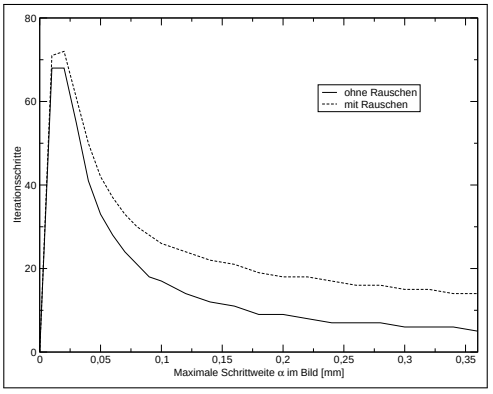
Der Trust-Region-Regler mit MJP-Modell

Als nächstes wird das MJP-Modell betrachtet. Bei konstanter Trust-Region-Größe wird für $\alpha_{\text{start}} = 0.05$ mit durchschnittlich 33 Regelschritten eine Erfolgsquote von 99.69 % erreicht (siehe Abbildung 4.6(a)).

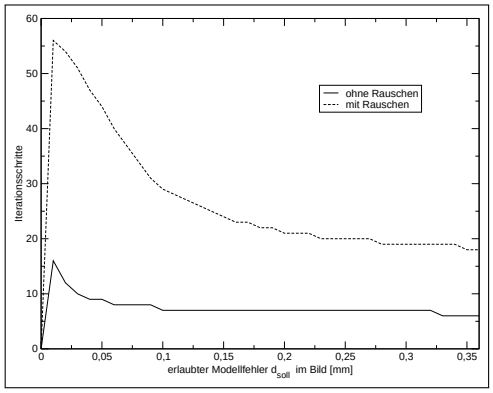
Der Durchlauf mit Alphaadaption liefert für einen gewünschten Modellfehler $d_{\text{soll}} = 0.1$ eine Erfolgsquote von 99.58 %. Die durchschnittliche Anzahl von Iterationsschritten beträgt hier 7 Schritte (siehe Abbildung 4.6(c)). Auch hier lassen sich starke Abweichungen bei den Experimenten mit aktivem Rauschen beobachten.



(a) Erfolgsquote bei konstantem α



(b) Iterationsschritte bei konstantem α



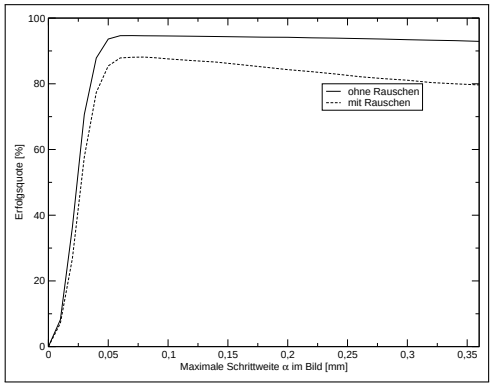
(c) Iterationsschritte bei für unterschiedlichem d_{soll} mit $\alpha_{\text{start}} = 0.05$

Abbildung 4.6: Multilagen-Simulation mit MJP-Modell

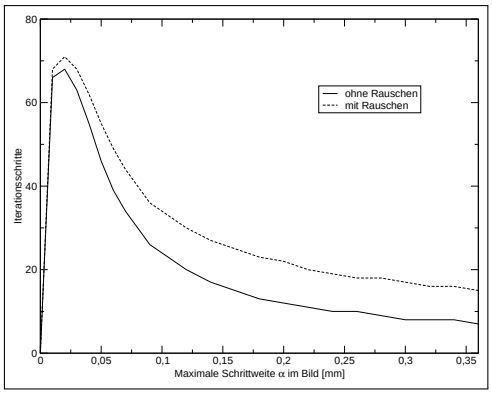
Der Trust-Region-Regler mit PMJ-Modell

Der Trust-Region-Regler mit dem PMJ-Modell liefert bei konstanter Trust Region für $\alpha_{\text{start}} = 0.07$ mit einer Erfolgsquote von 94.67 %(siehe Abbildung 4.6(a)) den besten Wert. Durchschnittlich braucht der Regler mit dieser Wahl 34 Iterationen, um seine Ziellage zu erreichen.

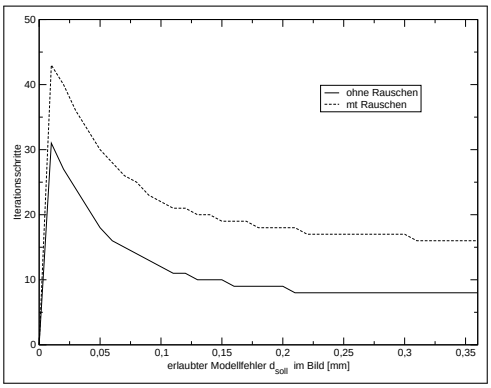
Mit $d_{\text{soll}} = 0.09$ ergibt sich in Abbildung 4.7(c) eine Erfolgsquote von 94.57 % mit einer mittleren Anzahl von 13 Reglerschritten.



(a) Erfolgsquote bei konstantem α



(b) Iterationsschritte bei konstantem α



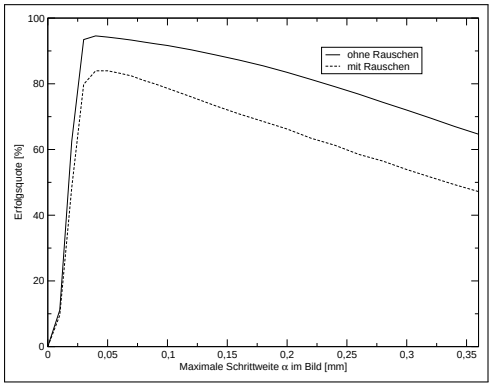
(c) Iterationsschritte bei für unterschiedlichem d_{soll} mit $\alpha_{\text{start}} = 0.07$

Abbildung 4.7: Multilagen-Simulation mit PMJ-Modell

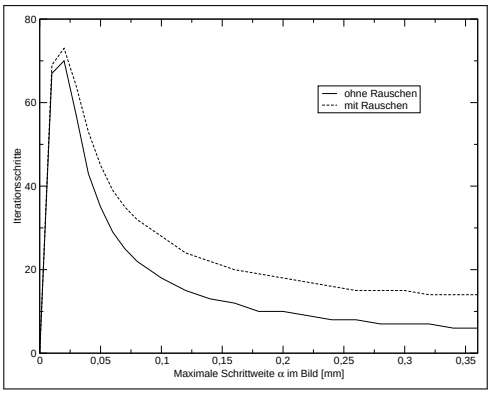
Der Trust-Region-Regler in Zylinderkoordinaten

Mit einer Wahl von $\alpha_{\text{start}} = 0.04$ erreicht der Regler eine Erfolgsquote von 94.57 % (siehe Abbildung 4.8(a)). Die Ziellagen werde durchschnittlich nach 43 Schritten erreicht.

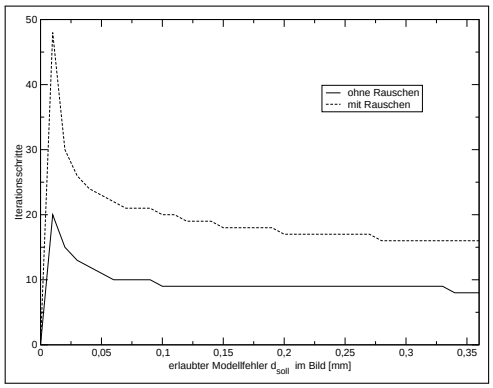
Bei den Durchläufen mit aktiver Alphaadaption wird für den gewünschten Modellfehler d_{soll} der Wert 0.1 bestimmt. Mit durchschnittlich 9 Iterationsschritten konvergiert der Regler in 93.5 % aller getesteten Lagen.



(a) Erfolgsquote bei konstantem α



(b) Iterationsschritte bei konstantem α



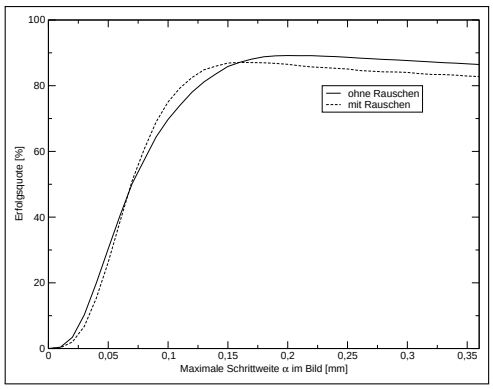
(c) Iterationsschritte bei für unterschiedlichem d_{soll} mit $\alpha_{\text{start}} = 0.04$

Abbildung 4.8: Multilagen-Simulation in Zylinderkoordinaten

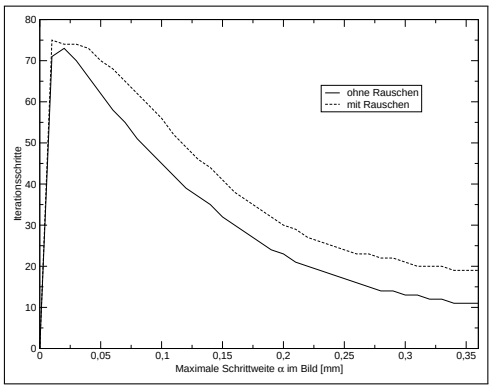
Der DogLeg-Regler mit konstanter Jacobimatrix

Bei konstanter Trust-Region-Größe ergibt sich bei $\alpha_{\text{ist}} = 0.22$ mit einer Erfolgsquote von 89.14 % eine durchschnittliche Anzahl von 20 Iterationsschritten (siehe Abbildungen 4.9(a) und 4.9(b)).

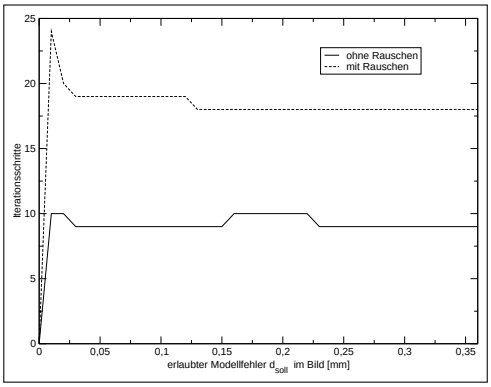
Bei der Wahl von $d_{\text{soll}} = 0.22$ sind in Abbildung 4.9(c) 85.05 % der Regelungen erfolgreich und benötigen im Mittel 10 Iterationen um die Teachpose zu erreichen. Auffällig ist, dass der für α_{start} bestimmte Wert im Gegensatz zum Trust-Region-Regler relativ groß ist.



(a) Erfolgsquote bei konstantem α



(b) Iterationsschritte bei konstantem α



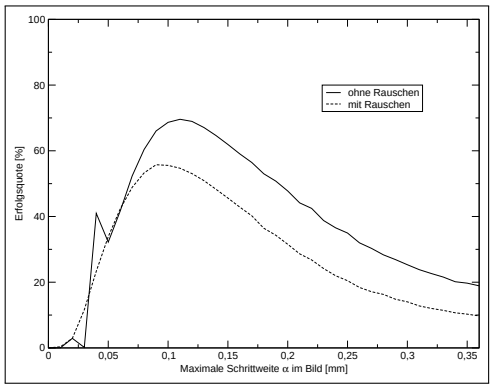
(c) Iterationsschritte bei für unterschiedliche d_{soll} mit $\alpha_{\text{start}} = 0.22$

Abbildung 4.9: Multilagen-Simulation mit konstanter Jacobimatrix

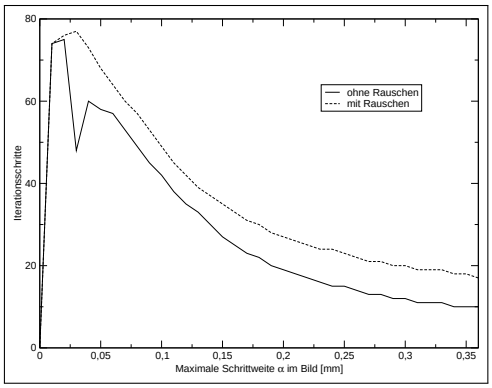
Der DogLeg-Regler mit dynamischer Jacobimatrix

Beim Dog-Leg-Regler mit dynamischer Jacobimatrix und inaktiver Alphaadaption wird ein optimaler Wert von $\alpha_{\text{start}} = 0.11$ bestimmt. Dabei liegt bei einer durschnittlichen Anzahl von 38 Iterationsschritten die Erfolgsquote bei 69.59 % (siehe Abbildung 4.10(a)).

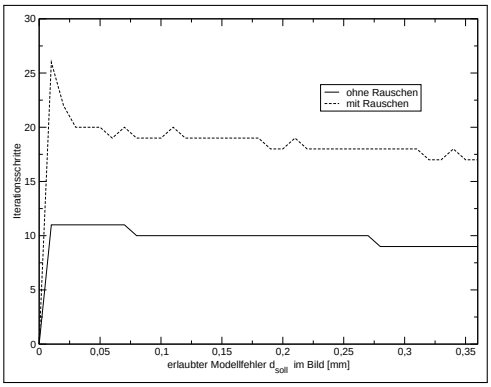
Für den so bestimmten Wert ergibt sich in Abbildung 4.10(c) ein zulässiger Modellfehler von $d_{\text{soll}} = 0.28$. Hier braucht der Dog-Leg-Regler durschnittlich 9 Iterationsschritte. Allerdings besitzt er dort eine Erfolgsquote von nur noch 8.4 %. Durch den großen zulässigen Modellfehler werden dem Regler auch bei relativ großer Ungenauigkeit des Modells große Schritte erlaubt. Wie schon bei den Experimenten mit dem Traditonellen Regler in Abbildung 3.4 im Abschnitt 3.4.1 beschrieben, nimmt die Erfolgsquote mit steigendem Dämpfungsfaktor stark ab. Im Gegesatz dazu reagiert der Regler mit konstanter Jacobimatrix unempfindlicher auf größere Werte (siehe Abbildung 3.3).



(a) Erfolgsquote bei konstantem α



(b) Iterationsschritte bei konstantem α



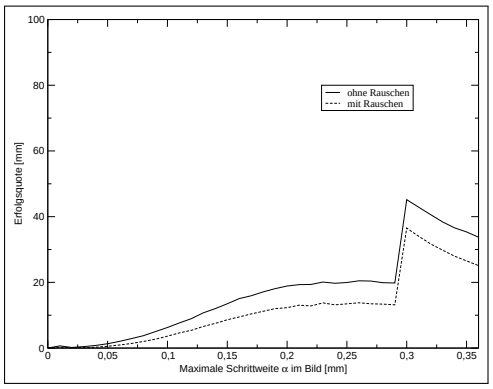
(c) Iterationsschritte bei für unterschiedliche d_{soll} mit $\alpha_{\mathrm{start}} = 0.11$

Abbildung 4.10: Multilagen-Simulation mit dynamischer Jacobimatrix

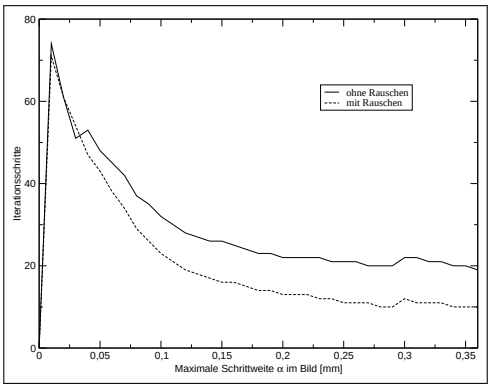
Der DogLeg-Regler mit MJP-Modell

Für den Dog-Leg-Regler mit MJP-Modell ergibt sich bei inaktiver Alphaadaption für α_{start} in Abbildung 4.11(a) ein Wert von 0.3 mit einer durschnittlichen Erfolgsquote von 36.51 %. Sie steigt an dieser Stelle sprunghaft auf diesen Wert. Die durschnittliche Anzahl von Iterationen beträgt hier 22 Schritte.

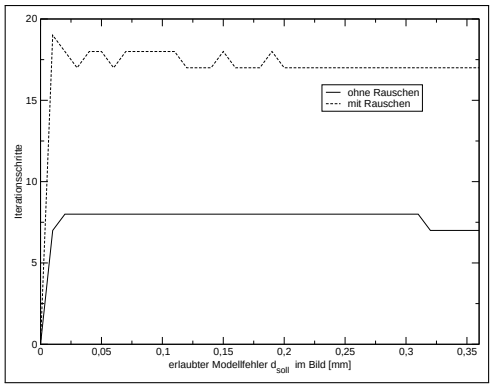
Für d_{soll} wird in Abbildung 4.11(c) ein Wert von 0.02 gewählt. Mit diesem Parameter sind 25.65 % aller Durchläufe erfolgreich. Im Mittel werden 8 Regelschritte benötigt, um die Ziellagen zu erreichen.



(a) Erfolgsquote bei konstantem α



(b) Iterationsschritte bei konstantem α



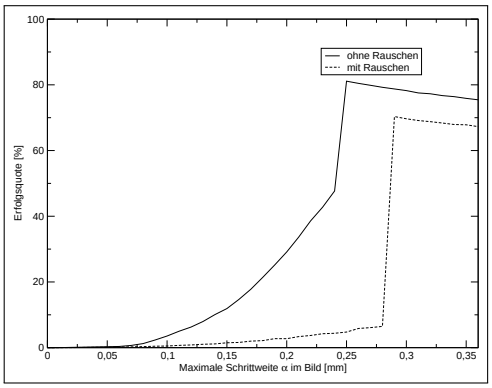
(c) Iterationsschritte bei für unterschiedliche d_{soll} mit $\alpha_{\text{start}} = 0.3$

Abbildung 4.11: Multilagen-Simulation mit MJP-Modell

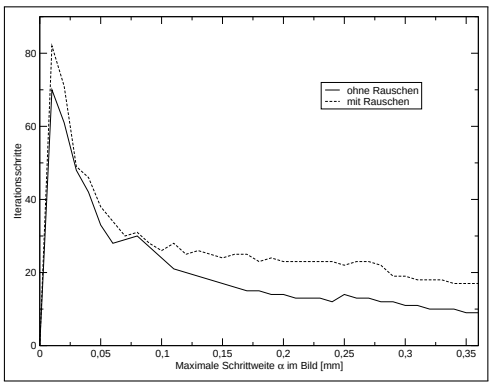
Der DogLeg-Regler mit PMJ-Modell

Wird für den Dog-Leg-Regler das PMJ-Modell benutzt, so liegt bei konstantem Alpha der beste Wert für α_{start} bei 0.29. Hier sind durschnittlich 12 Iterationsschritte notwendig, um die Teachpose zu erreichen (siehe Abbildung 4.12(b)). Diese wird von 78.75 % aller Startposen erreicht.

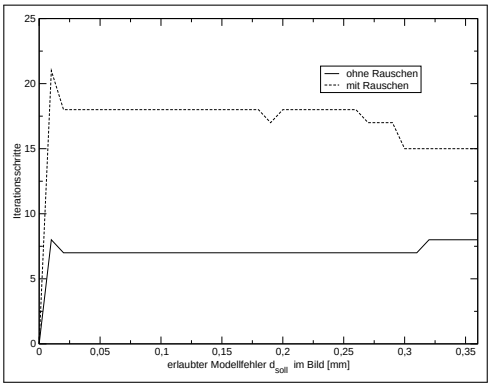
Für den so gewählten Parameter α_{start} ergibt sich in Abbildung 4.12(c) ein Wert von 0.03 für den zulässigen Modellfehler d_{soll}. Die Erfolgsquote liegt hier bei 31.47 % mit durchschnittlich 8 Iterationsschritten.



(a) Erfolgsquote bei konstantem α



(b) Iterationsschritte bei konstantem α

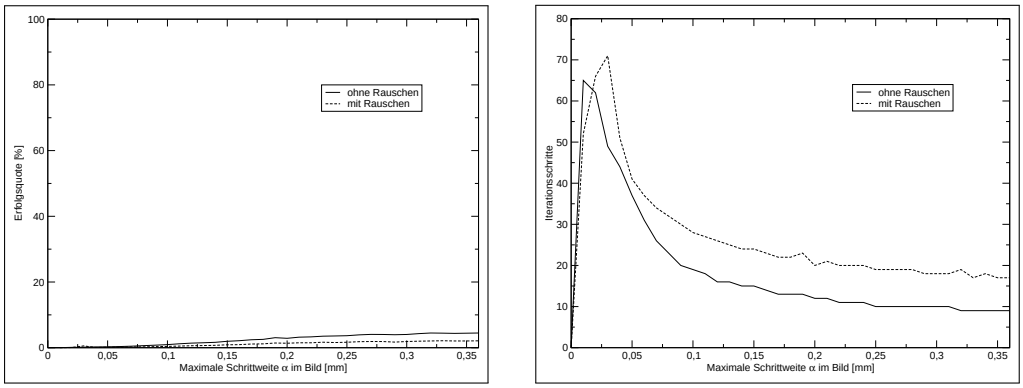


(c) Iterationsschritte bei für unterschiedliche d_{soll} mit $\alpha_{\text{start}} = 0.29$

Abbildung 4.12: Multilagen-Simulation mit PMJ-Modell

Der DogLeg-Regler in Zylinderkoordinaten

Hier wird bei deaktivierter Alphaadaption für $\alpha_{\text{start}} = 0.36$ nur eine Erfolgsquote von 4.48 % erreicht. Der Ansatz mit der Jacobimatrix in Zylinderkoordinaten wird deshalb für den Dog-Leg-Regler hier nicht weiter verfolgt.



(a) Erfolgsquote bei konstantem α (b) Iterationsschritte bei konstantem α

Abbildung 4.13: Multilagen-Simulation in Zylinderkoordinaten

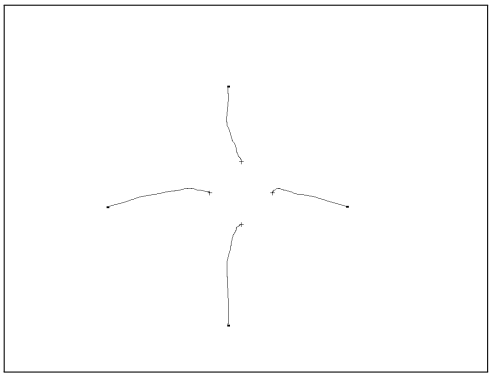
4.4.2 Testläufe mit dem OpenGL-Simulator

Die durch die Multilagen-Simulation bestimmten Werte werden nun für die fünf Lagen benutzt. Es folgen die Testläufe für die einzelnen Regler.

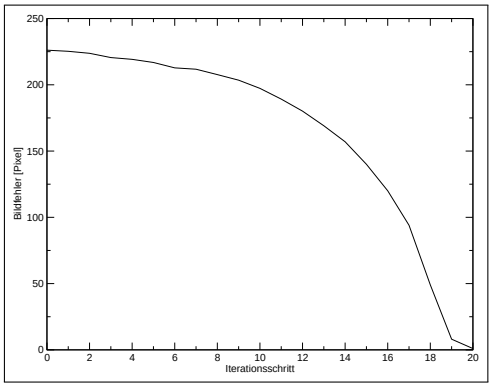
Ausgangslage 1: [0, 0, -300, 0°, 0°, 0°]

Trust-Region-Regler mit konstanter Jacobimatrix

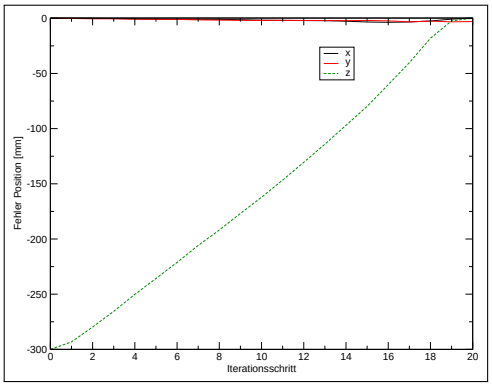
Der Regler erreicht seine Zielpose nach 20 Iterationsschritten. In Abbildung 4.14 läßt sich beobachten, dass der Bildfehler zuerst sehr langsam sinkt. Ab dem zwölften Regelschritt verringert sich der Bildfehler pro Iterationsschritt wieder schneller. Das ist genau das Verhalten, was durch den Regler erreicht werden soll. Durch die kleineren Schritte am Anfang soll verhindert werden, dass die Bildmerkmale das Kamerabild verlassen. Die größeren Schritte zum Schluss ermöglichen eine schnellere Konvergenz.



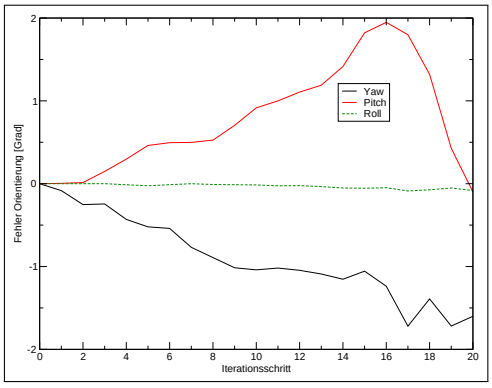
(a) Sensorspur der ersten Lage



(b) Bildfehler der ersten Lage



(c) Fehler der Kameraposition

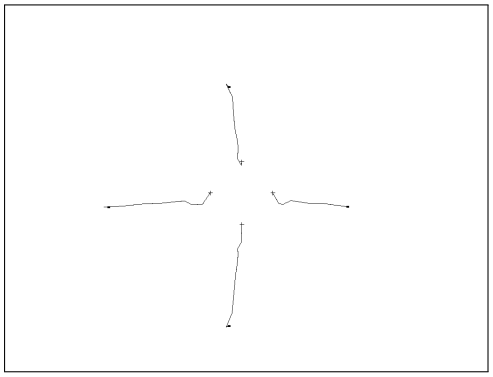


(d) Fehler der Kameraorientierung

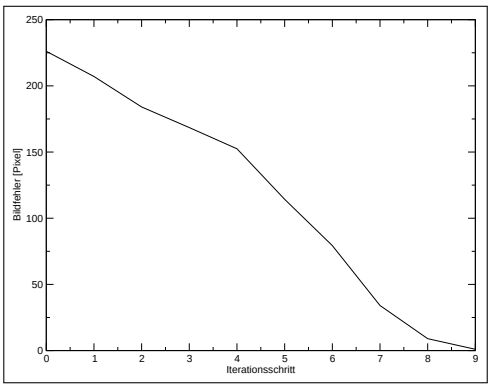
Abbildung 4.14: Trust-Region-Regler mit konstanter Jacobimatrix, Lage 1

Trust-Region-Regler mit dynamischer Jacobimatrix

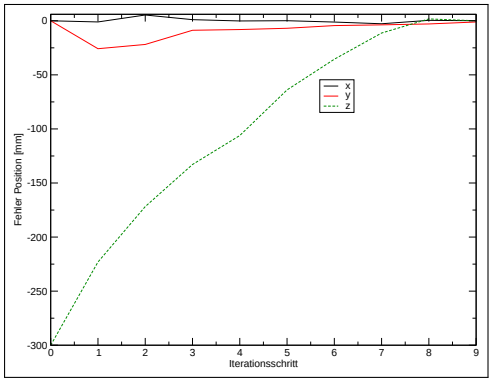
Der Trust-Region-Regler mir dynamischer Jacobimatrix erreicht die Ausgangslage nach 9 Iterationen. Im Gegensatz zum traditionellen Ansatz ist hier in Abbildung 4.15(a) die Bewegung in Richtung unterem Bildrand nicht ganz so groß (siehe dazu Abbildung 3.9 im vorherigen Kapitel).



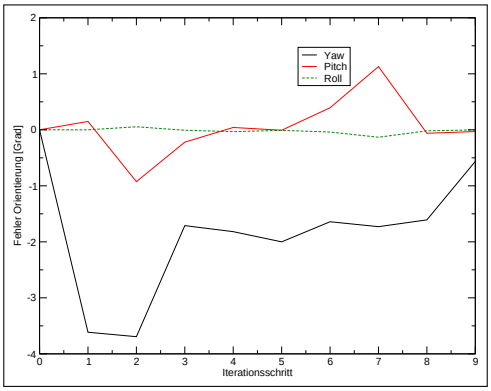
(a) Sensorspur der ersten Lage



(b) Bildfehler der ersten Lage



(c) Fehler der Kameraposition

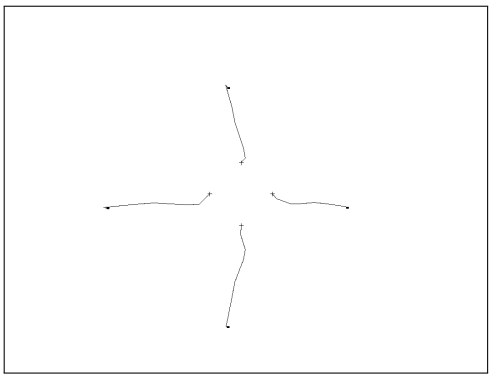


(d) Fehler der Kameraorientierung

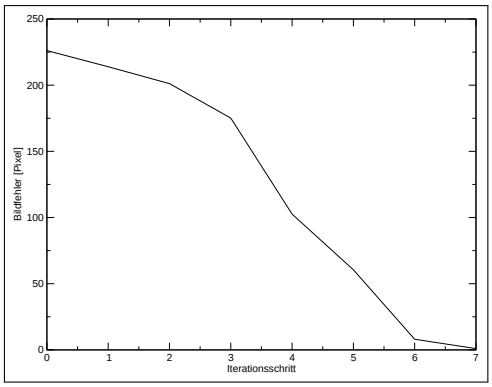
Abbildung 4.15: Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 1

Trust-Region-Regler mit MJP-Modell

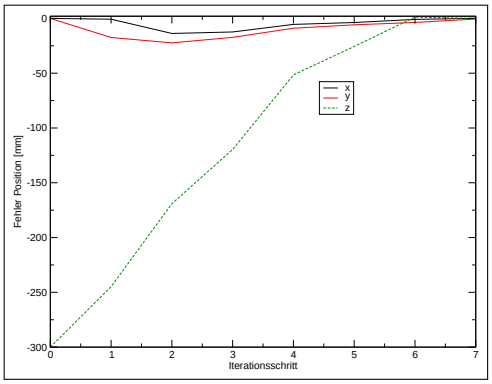
Der Regler mit dem MJP-Modell konvergiert bereits nach 7 Regelrschritten. Hier ist die Bewegung zum unteren Bildrand (siehe Abbildung 4.17(a)) kaum noch vorhanden. Im Gegensatz zum Trust-Region-Regler mit dynamischer Jacobimatrix sinkt hier der Bildfehler am Anfang langsamer. Auch die Bewegung des Roboters entlang der y-Achse (siehe Abbildung 4.16(c)) fällt geringer aus.



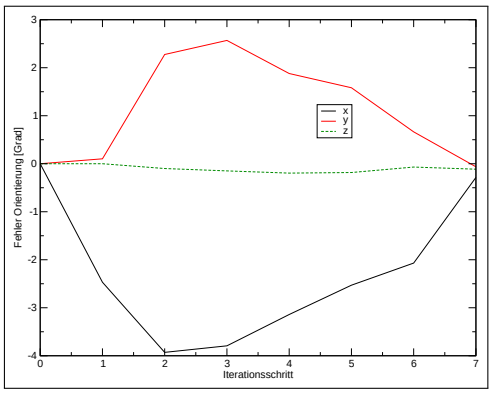
(a) Sensorspur der ersten Lage



(b) Bildfehler der ersten Lage



(c) Fehler der Kameraposition

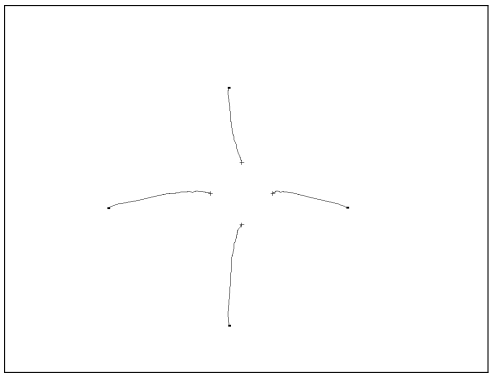


(d) Fehler der Kameraorientierung

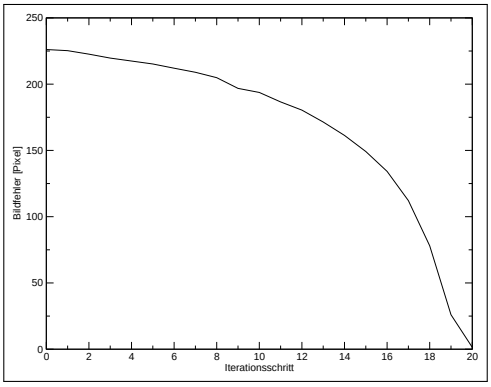
Abbildung 4.16: Trust-Region-Regler mit MJP-Modell, Lage 1

Trust-Region-Regler mit PMJ-Modell

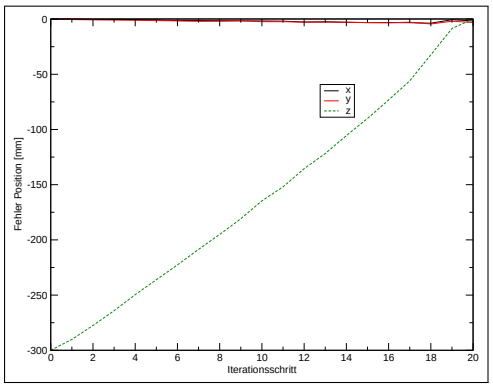
Mit 20 Iterationsschritten braucht der Trust-Region-Regler länger als seine Vorgänger. Der Regler bewegt sich direkt auf die Zielposition zu (siehe Abbildung 4.16(a)). In den Abbildungen 4.17(c) und 4.17(d) fallen die Schwankungen geringer aus als bei den Ansätzen mit konstanter und dynamischer Jacobimatrix. Dies ist unter anderem auf die kleinere Schrittgröße des Reglers zurückzuführen. Insgesamt verbessert sich der Regler mit dem Trust-Region-Ansatz nicht in dem Maße, wie es bei den vorherigen Modellen der Fall war.



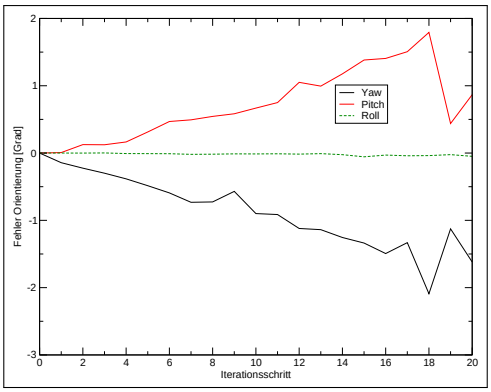
(a) Sensorspur der ersten Lage



(b) Bildfehler der ersten Lage



(c) Fehler der Kameraposition

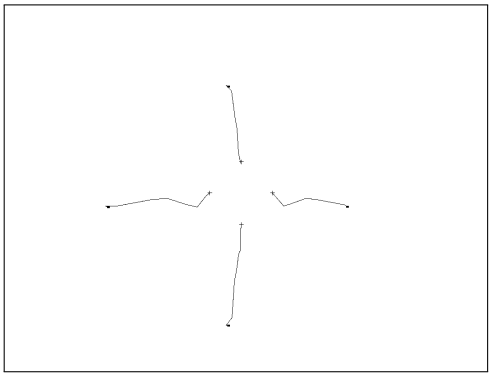


(d) Fehler der Kameraorientierung

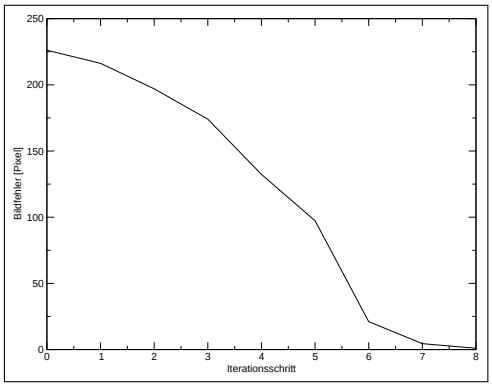
Abbildung 4.17: Trust-Region-Regler mit PMJ-Modell, Lage 1

Trust-Region-Regler in Zylinderkoordinaten

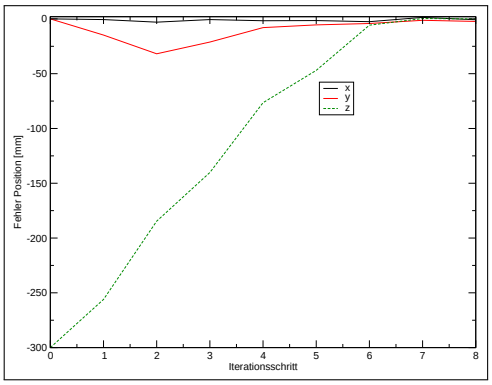
Wie bei den Versuchen mit dem traditionellen Ansatz im vorherigen Kapitel, liegt hier die zu bestimmende Rotationsachse wieder parallel zur Sensorebene, so dass die dynamische Jacobimatrix benutzt wird. Der Regler ereicht die Teachpose nach 8 Iterationsschritten. Da der zulässige Modellfehler hier größer gewählt ist als beim Trust-Region-Ansatz mit dynamischer Bildjacobimatrix, fällt die Schrittweite hier größer aus. Dementsprechend lässt sich in Abbildung 4.18(a) eine größere Bewegung zum unteren Bildrand beobachten. Auch die Schwankungen in den Abbildungen 4.18(c) und 4.18(d) sind stärker.



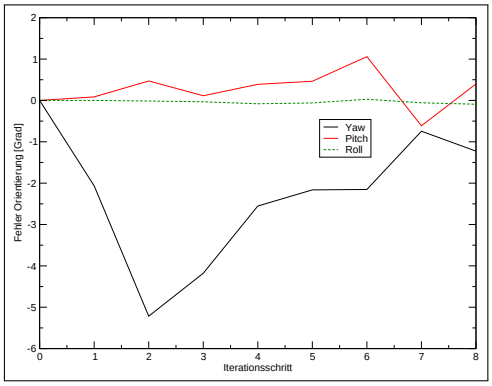
(a) Sensorspur der ersten Lage



(b) Bildfehler der ersten Lage



(c) Fehler der Kameraposition



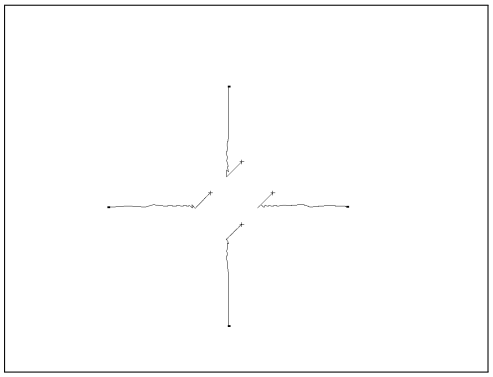
(d) Fehler der Kameraorientierung

Abbildung 4.18: Trust-Region-Regler in Zylinderkoordinaten, Lage 1

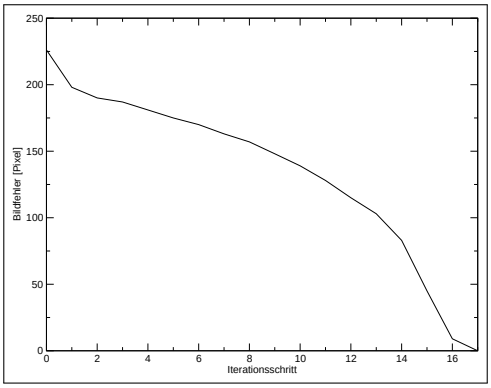
Dog-Leg-Regler mit konstanter Jacobimatrix

Der Dog-Leg-Regler mit konstanter Jacobimatrix erreicht die Ziellage nach 17 Schritten. In den ersten beiden Schritten wird der Gradientenabstieg benutzt. Bis zu den letzten beiden Iterationen wir dann der Dog-Leg-Schritt ausgeführt. Zum Schluss folgen noch zwei Gauß-Newton-Schritte.

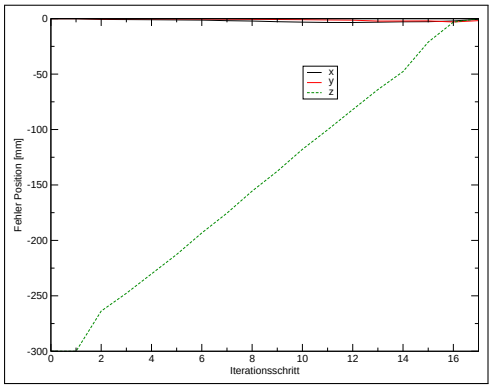
Hier lässt ich die Vorgehensweise des Dog-Leg-Reglers erkennen. Zuerst wird mit dem Gradientenabstieg das Objekt in die Bildmitte gebracht. Dann wird sich den Objektmarkierungen angenähert. Der Bildfehler in Abbildung 4.19(b) nimmt stetig ab. Der Verlauf der Bildmerkmale auf dem CCD-Sensor in Abbildung 4.19(a) ist allerdings nicht so gleichmäßig wie bei den anderen Reglern.



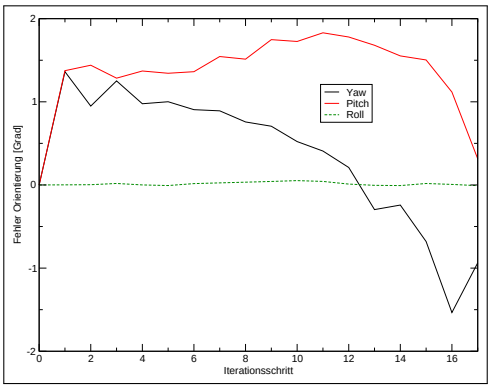
(a) Sensorspur der ersten Lage



(b) Bildfehler der ersten Lage



(c) Fehler der Kameraposition



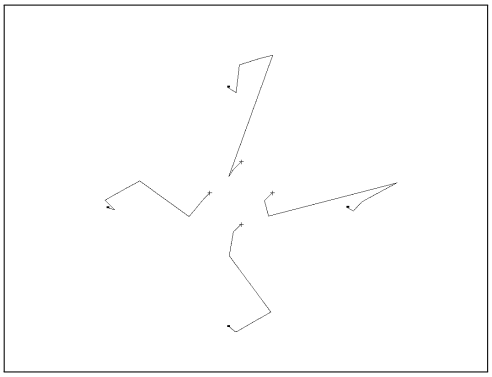
(d) Fehler der Kameraorientierung

Abbildung 4.19: DogLeg-Regler mit konstanter Jacobimatrix, Lage 1

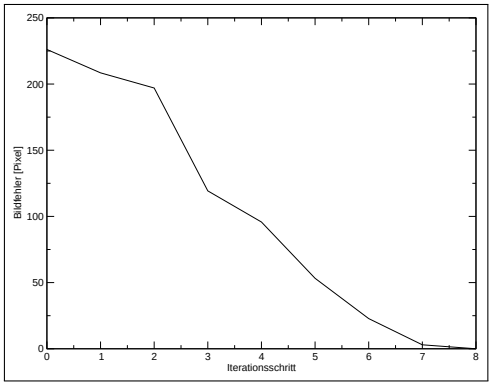
Dog-Leg-Regler mit dynamischer Jacobimatrix

Diese Version des Dog-Leg-Reglers konvergiert nach 8 Schritten. Auch hier lässt sich in Abbildung 4.20(b) beobachten, dass sich an keiner Stelle der Bildfehler erhöht.

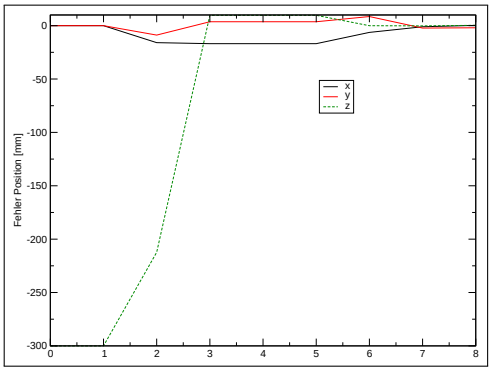
Der erste Schritt ist ein Gradientenabstieg. Die Abbildungen 4.20(c) und 4.20(d) zeigen, dass beim ersten Iterationsschritt die Position der Kamera nicht verändert wird. Nur deren Orientierung ändert sich. Ab dem zweiten Schritt wird dann bis auf die letzten beiden der Dog-Leg-Schritt ausgeführt.



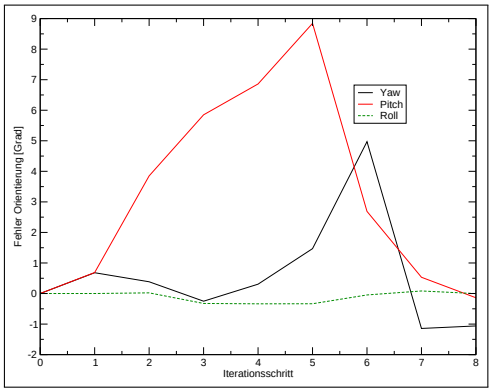
(a) Sensorspur der ersten Lage



(b) Bildfehler der ersten Lage



(c) Fehler der Kameraposition



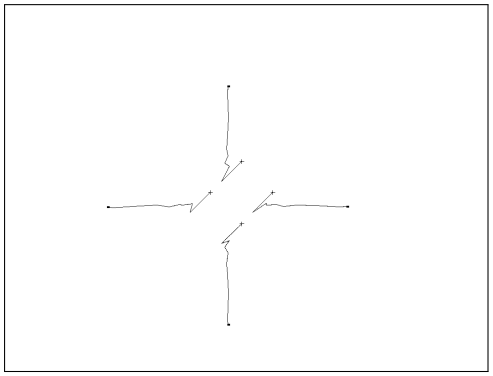
(d) Fehler der Kameraorientierung

Abbildung 4.20: DogLeg-Regler mit dynamischer Jacobimatrix, Lage 1

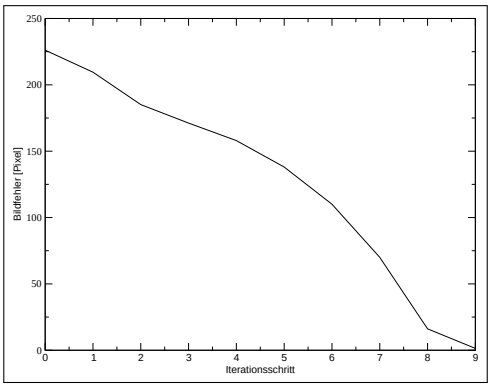
Dog-Leg-Regler mit PMJ-Modell

Der Dog-Leg-Regler mit dem PMJ-Modell erreicht die Teachpose nach 9 Iterationsschritten. Der Bildfehler in Abbildung 4.21(a) verschlechtert sich an keiner Stelle.

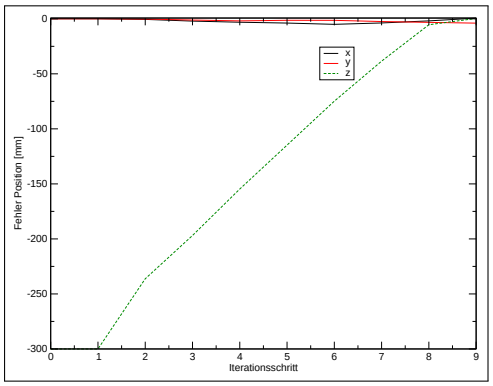
Beim ersten Regelschritt findet nur eine Änderung der Orientierung statt. Die Position der Kamera bleibt unverändert. Erst ab dem zweiten Iterationsschrit wird der Positionsfehler korrigiert. Das liegt wieder daran, dass im ersten Schritt der Gradientenabstieg benutzt wurde. Bis auf den letzten Schritt wird der Dog-Leg-Schritt benutzt.



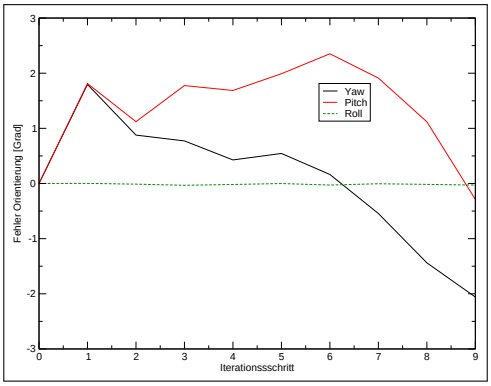
(a) Sensorspur der ersten Lage



(b) Bildfehler der ersten Lage



(c) Fehler der Kameraposition



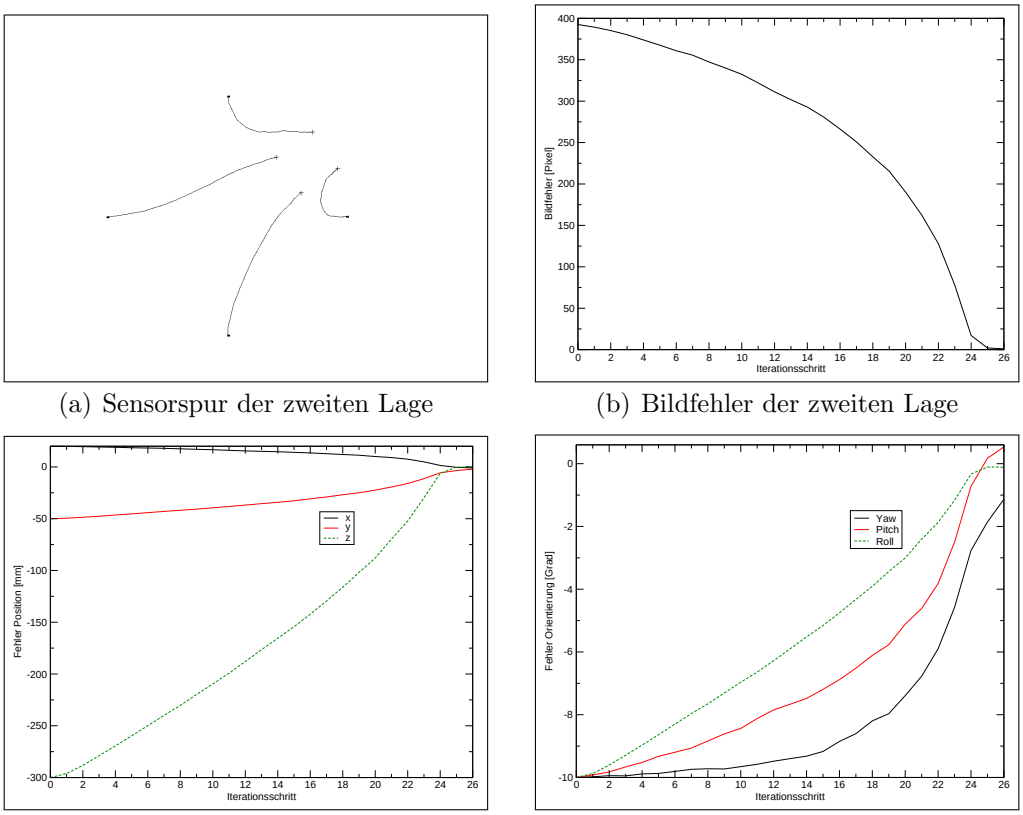
(d) Fehler der Kameraorientierung

Abbildung 4.21: DogLeg-Regler mit PMJ-Modell, Lage 1

Ausgangslage 2: [20, -50, -300, -10°, -10°, -10°]

Trust-Region-Regler mit konstanter Jacobimatrix

Der Trust-Region-Regler mit konstanter Jacobimatrix erreicht seine Ziellage nach 26 Regleschritt. Damit ist diese Version des Regler fast doppelt so schnell wie die Variante mit dem traditionellen Ansatz.



(a) Sensorspur der zweiten Lage

(b) Bildfehler der zweiten Lage

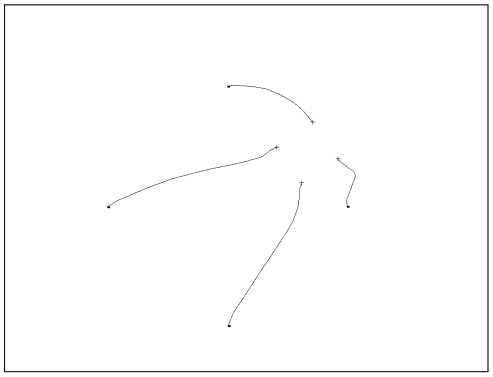
(c) Fehler der Kameraposition

(d) Fehler der Kameraorientierung

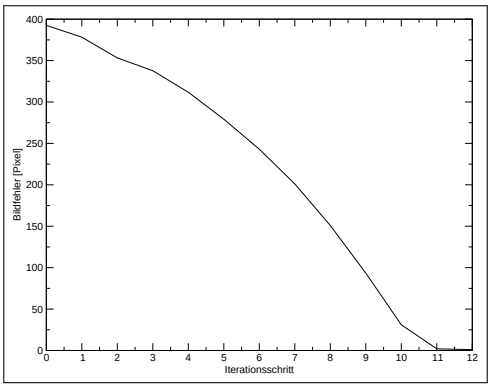
Abbildung 4.22: Trust-Region-Regler mit konstanter Jacobimatrix, Lage 2

Trust-Region-Regler mit dynamischer Jacobimatrix

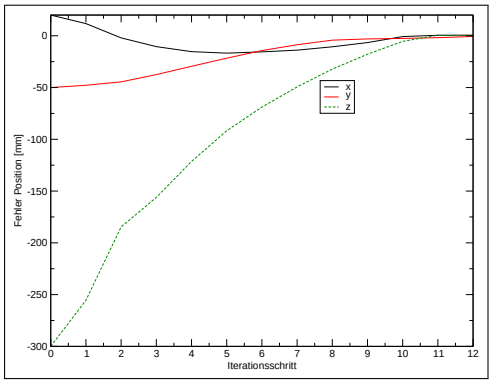
Nach 12 Iterationen konvergiert der Regler mit dynamischer Jacobimatrix. Die Spur der Bildmerkmale in Abbildung 4.23(a) ist fast mit denen in Abbildung 3.14(a) im vorherigen Kapitel identisch.



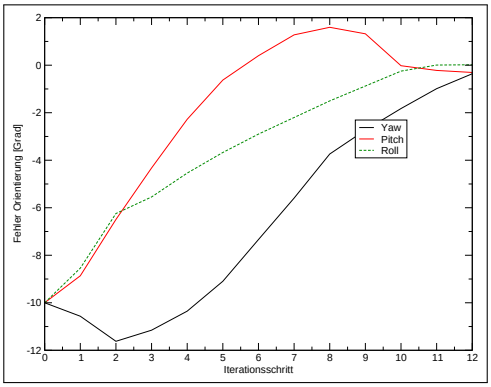
(a) Sensorspur der zweiten Lage



(b) Bildfehler der zweiten Lage



(c) Fehler der Kameraposition

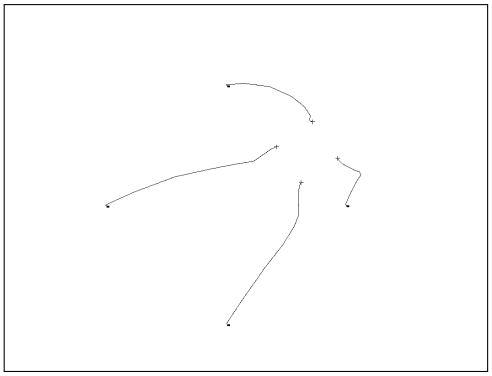


(d) Fehler der Kameraorientierung

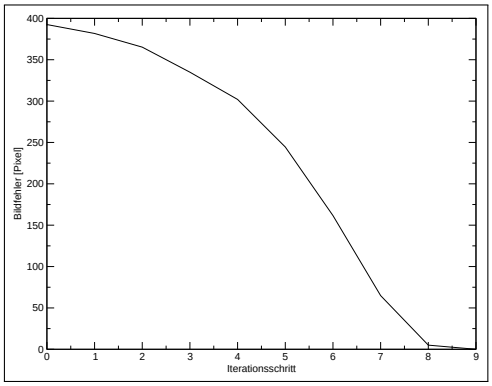
Abbildung 4.23: Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 2

Trust-Region-Regler mit MJP-Modell

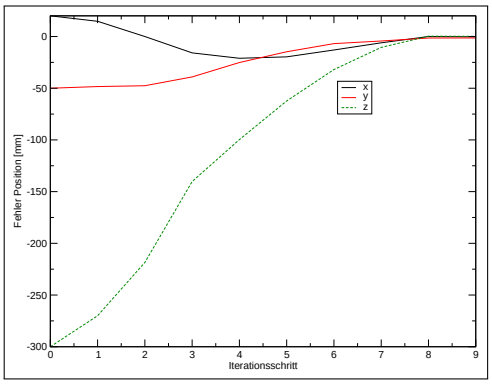
Der Trust-Region-Regler mit MJP-Modell erreicht nach 9 Regelschritten die Teachpose. Die Ergebnisse in Abbildung 4.24 ähneln denen des vorherigen Experiments.



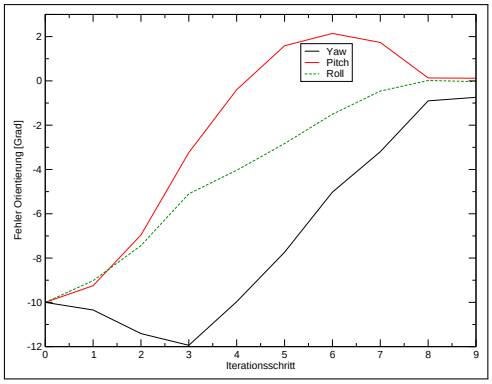
(a) Sensorspur der zweiten Lage



(b) Bildfehler der zweiten Lage



(c) Fehler der Kameraposition

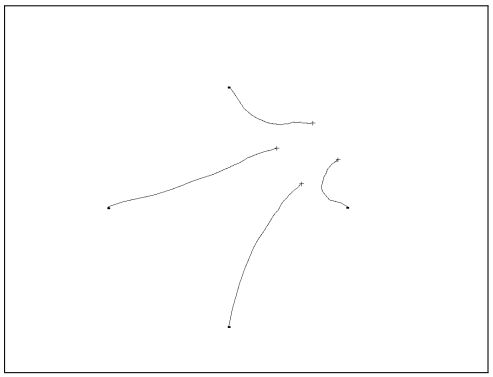


(d) Fehler der Kameraorientierung

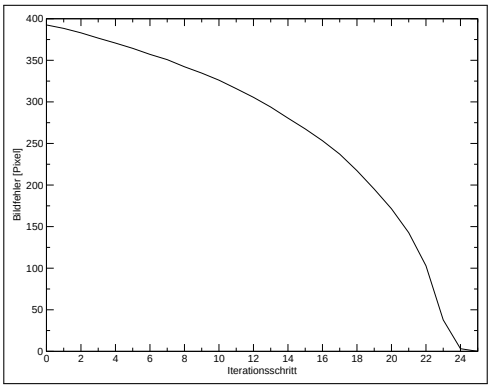
Abbildung 4.24: Trust-Region-Regler mit MJP-Modell, Lage 2

Trust-Region-Regler mit PMJ-Modell

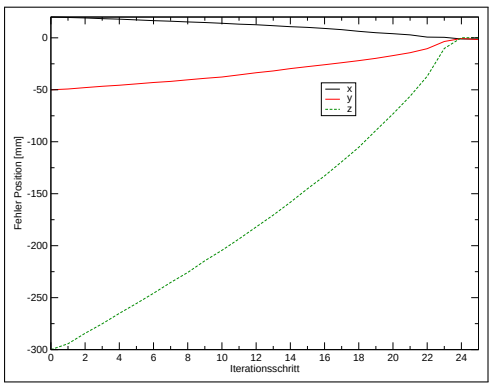
Mit dem PMJ-Modell nimmt der Trust-Region-Regler die Zielpose nach 25 Iterationen ein. Hier lassen sich wieder in den Ergebnisse in Abbildung 4.25 Ähnlichkeiten zum Trust-Region-Regler mit konstanter Jacobimatrix erkennen.



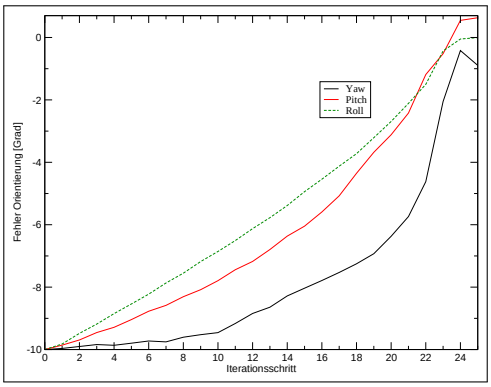
(a) Sensorspur der zweiten Lage



(b) Bildfehler der zweiten Lage



(c) Fehler der Kameraposition

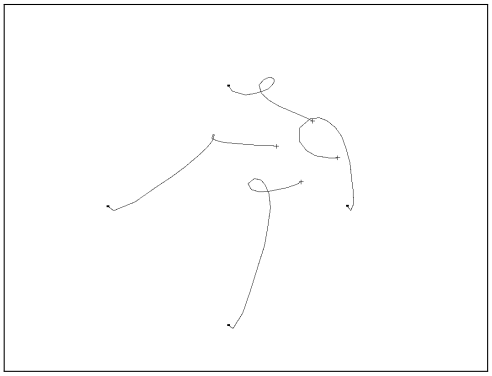


(d) Fehler der Kameraorientierung

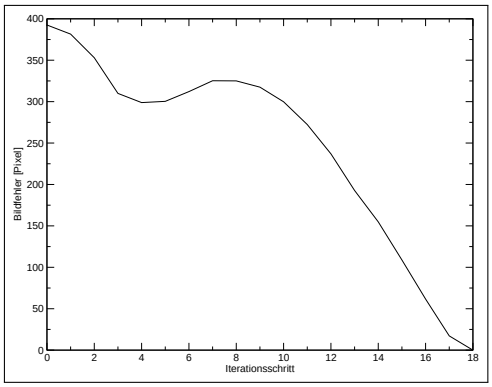
Abbildung 4.25: Trust-Region-Regler mit PMJ-Modell, Lage 2

Trust-Region-Regler in Zylinderkoordinaten

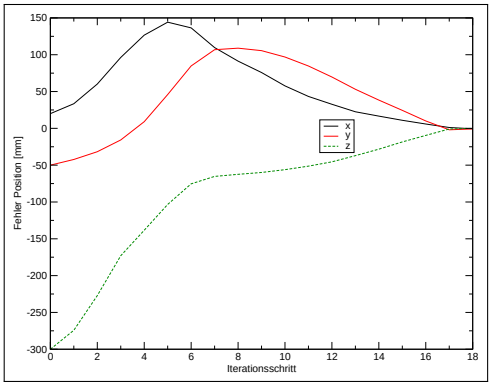
Wie auch bei dem traditionellen Ansatz hat die Jacobimatrix in Zylinderkoordinaten auch hier Probleme mit der zweiten Ausgangslage. Dabei wirken die Bewegungen auf dem Sensor (siehe Abbildung 4.26(a)) glatter als die im vorherigen Kapitel (siehe Abbildung 3.17(a)). Insgesamt sind die Schwankungen beim Bild- und beim Kamerafehler hier geringer. Der Roboter erreicht sein Ziel nach 18 Schritten.



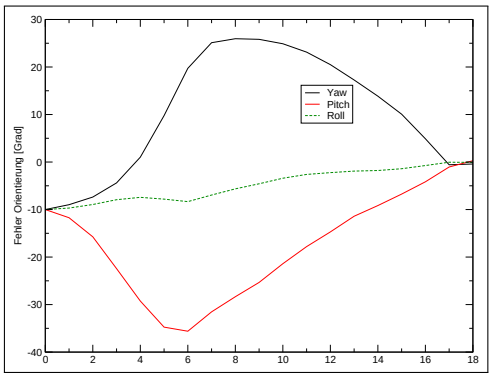
(a) Sensorspur der zweiten Lage



(b) Bildfehler der zweiten Lage



(c) Fehler der Kameraposition



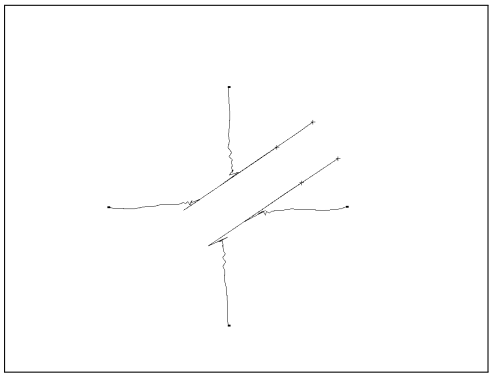
(d) Fehler der Kameraorientierung

Abbildung 4.26: Trust-Region-Regler in Zylinderkoordinaten, Lage 2

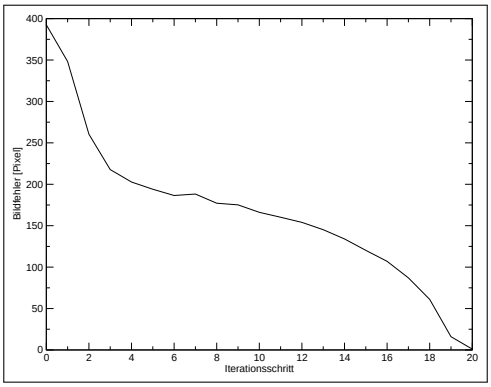
Dog-Leg-Regler mit konstanter Jacobimatrix

Der Dog-Leg-Regler benutzt in den ersten drei Schritten den Gradientenabstieg. Dabei findet wieder nur eine Änderung der Kameraorientierung statt. Bis auf die letzten beiden Schritte wird dann der Dog-Leg-Schritt ausgeführt. Nach 25 Iterationen hat der Roboter die Teachpose erreicht.

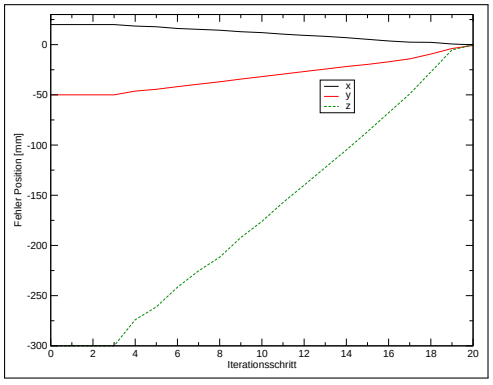
In Abbildung 4.27(a) steigt der Bildfehler beim sechsten Schritt kurzzeitig an. Die Bewegungen auf dem Sensor (siehe Abbildung 4.27(a)) sind Anfangs sehr ungleichmäßig. Die dort zu beobachtenden Schwingungen klingen aber ab, je mehr sich der Roboter dem Ziel nähert. Dadurch wird der Einfluss des Gradientenabstieges beim Dog-Leg-Schritt immer Schwächer.



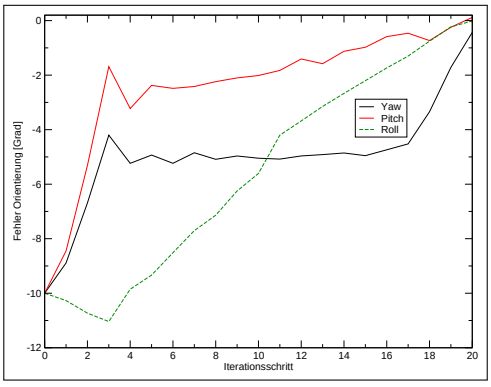
(a) Sensorspur der zweiten Lage



(b) Bildfehler der zweiten Lage



(c) Fehler der Kameraposition



(d) Fehler der Kameraorientierung

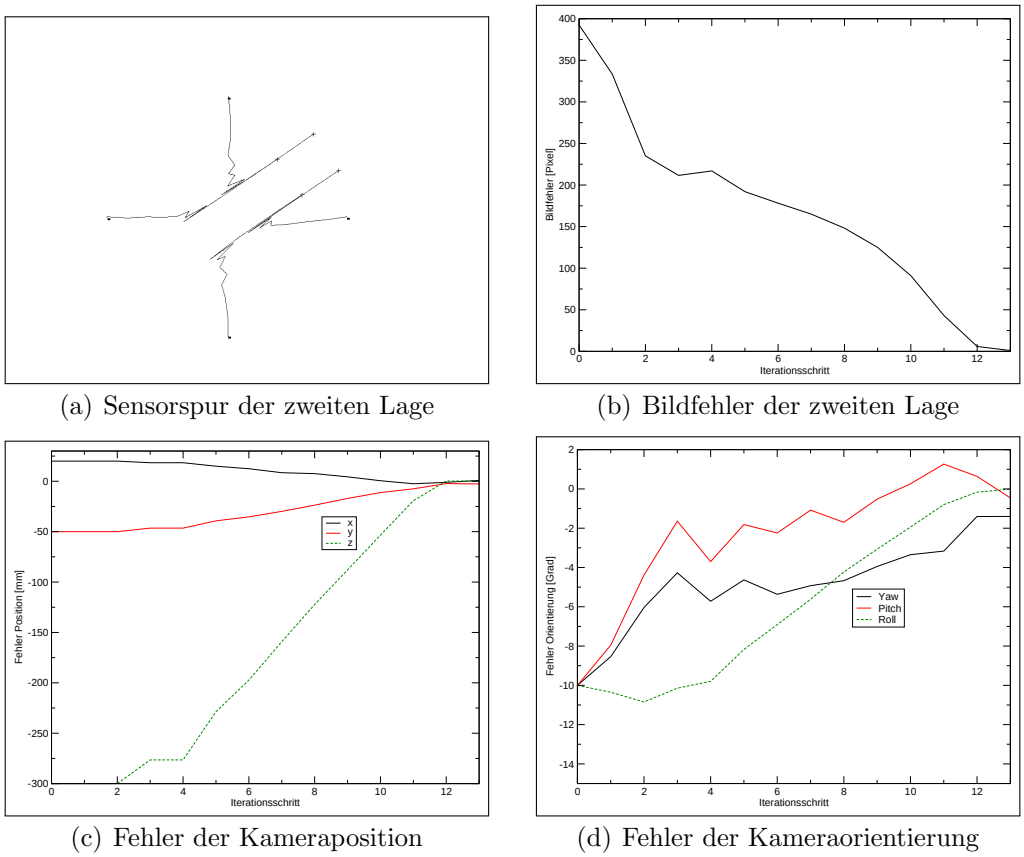
Abbildung 4.27: DogLeg-Regler mit konstanter Jacobimatrix, Lage 2

Dog-Leg-Regler mit dynamischer Jacobimatrix und MJP-Modell

Beide Varianten des Dog-Leg-Reglers schaffen es nicht die Teachpose zu erreichen. Nachdem zuerst einige Schritte mit dem Gradientenabstieg durchgeführt worden sind, verlassen die Bildmerkmale beim Dog-Leg-Schritt den Bildbereich der Kamera. Das liegt an der zu großen Schrittweite, die dort erreicht wird. Wie schon bei der Multilagen-Simulation im vorherigen Kapitel gesehen, nimmt die Erolgsquote der beiden Regler stark ab, je größer der Dämpfungsfaktor gewählt wurde.

Dog-Leg-Regler mit PMJ-Modell

Die Ergebnisse in Abbildung 4.28 lassen sich wieder mit denen des Ansatzes mit konstanter Jacobimatrix vergleichen. Allerdings wird hier das Ziel schon nach 13 Iterationen erreicht.



(a) Sensorspur der zweiten Lage

(b) Bildfehler der zweiten Lage

(c) Fehler der Kameraposition

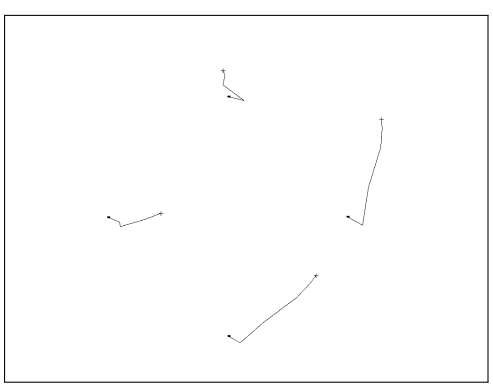
(d) Fehler der Kameraorientierung

Abbildung 4.28: DogLeg-Regler mit PMJ-Modell, Lage 2

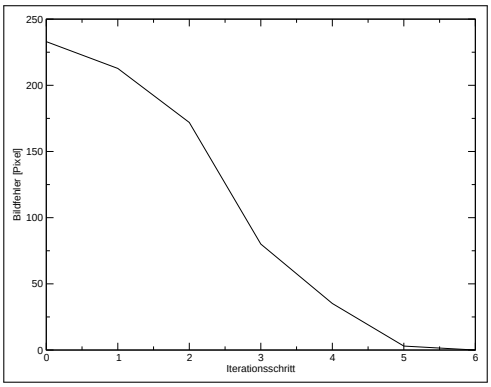
Ausgangslage 3: [0, 0, 0, -5°, -3°, 23°]

Trust-Region-Regler mit konstanter Jacobimatrix

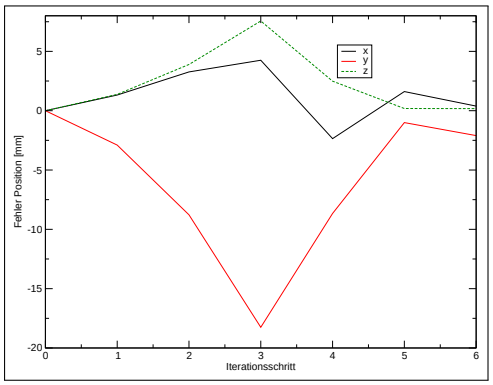
In der dritten Ausgangslage lässt sich wie schon beim Traditionellen Regler eine gleichzeitige Verschlechterung des Kamerafehlers bzgl. der y-Achse und des Yaw-Winkels beobachten, die sich beide wieder ausgleichen (siehe Abbildung 4.29(c) und 4.29(d)). Das Verhalten tritt auch bei den anderen Reglern auf. Der hier benutze Trust-Region-Regler mit konstanter Jacobimatrix konvergiert nach 6 Reglerschritten.



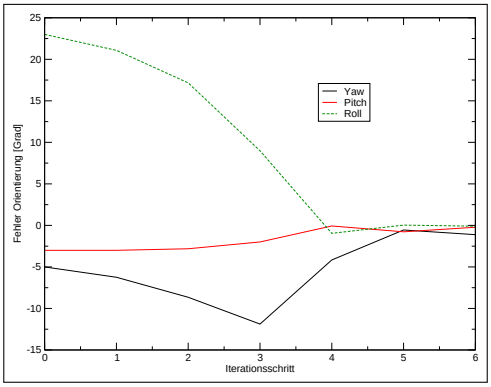
(a) Sensorspur der dritten Lage



(b) Bildfehler der dritten Lage



(c) Fehler der Kameraposition

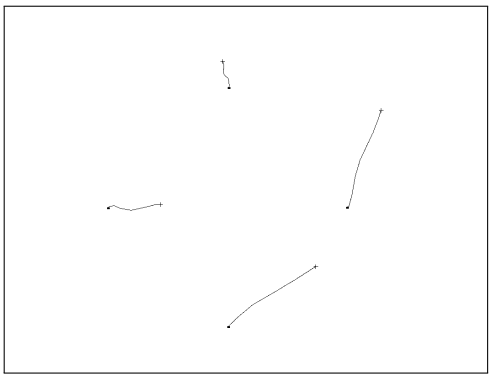


(d) Fehler der Kameraorientierung

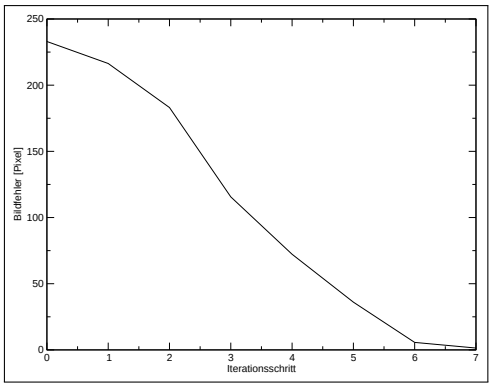
Abbildung 4.29: Trust-Region-Regler mit konstanter Jacobimatrix, Lage 3

Trust-Region-Regler mit dynamischer Jacobimatrix

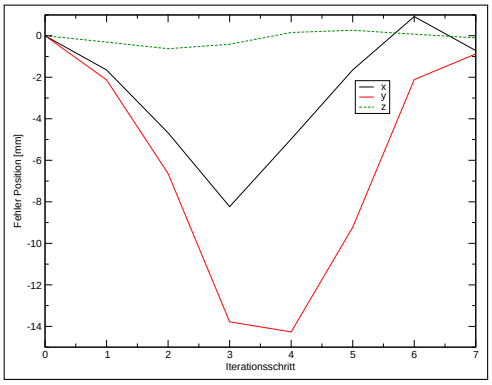
Der Trust-Region-Regler mit dynamischer Jacobimatrix erreicht die Zielpose nach 7 Schritten. Wie auch beim traditionellen Ansatz findet auch hier eine sich gegenseitig ausgleichende Vergrößerung des Kamerafehlers bzgl. der x-Achse und des Pitch-Winkels statt (siehe Abbildung 4.30(c) und 4.30(d)). Die Bewegung der Objektmarkierungen in Abbildung 4.30(a) beschreiben eine gerade Linie.



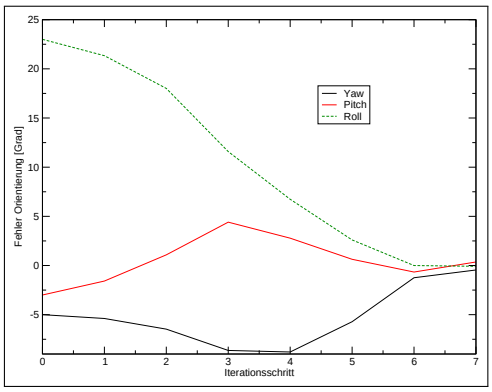
(a) Sensorspur der dritten Lage



(b) Bildfehler der dritten Lage



(c) Fehler der Kameraposition

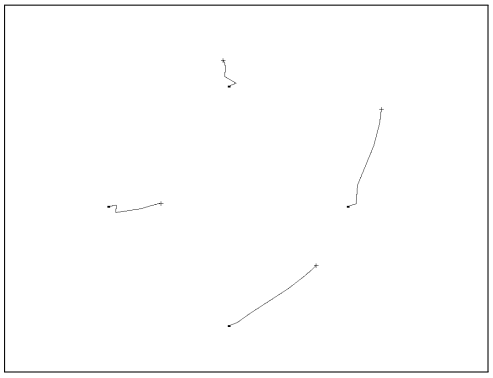


(d) Fehler der Kameraorientierung

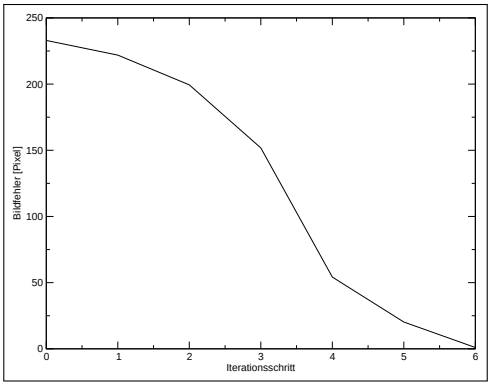
Abbildung 4.30: Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 3

Trust-Region-Regler mit MJP-Modell

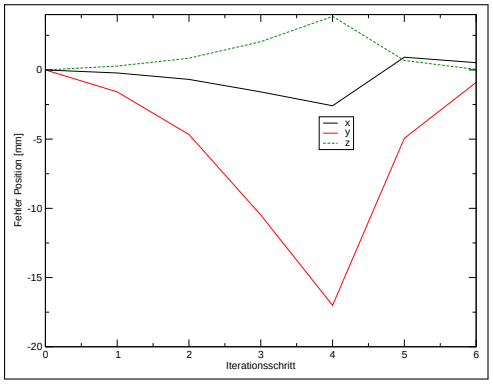
Die Ergebnisse der Trust-Region-Reglers mit MJP-Modell in Abbildung 4.31 ähneln wieder dem Ansatz mit dynamischer Jacobimatrix. Hier steigt der Bildfehler der Kamera bzgl. der x-Achse und des Pitch-Winkels nicht ganz so stark. Der Regler konvergiert nach 6 Iterationsschritten.



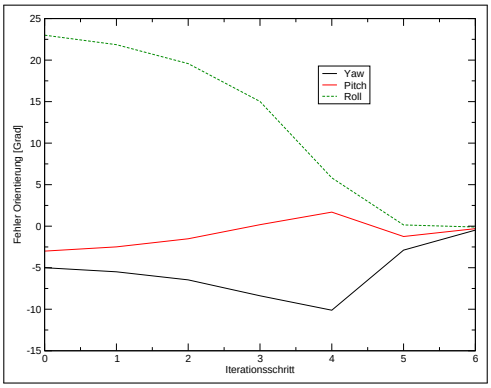
(a) Sensorspur der dritten Lage



(b) Bildfehler der dritten Lage



(c) Fehler der Kameraposition

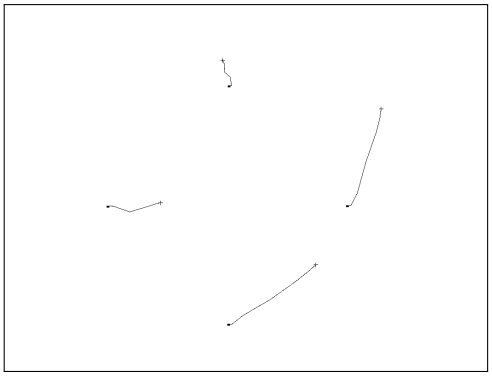


(d) Fehler der Kameraorientierung

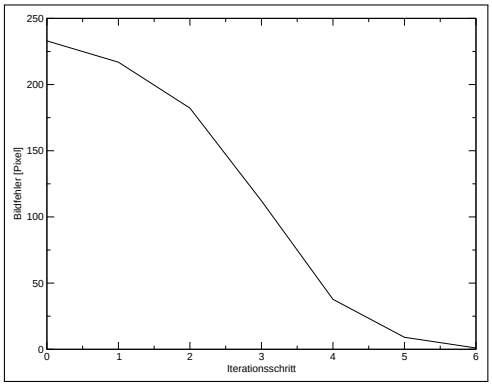
Abbildung 4.31: Trust-Region-Regler mit MJP-Modell, Lage 3

Trust-Region-Regler mit PMJ-Modell

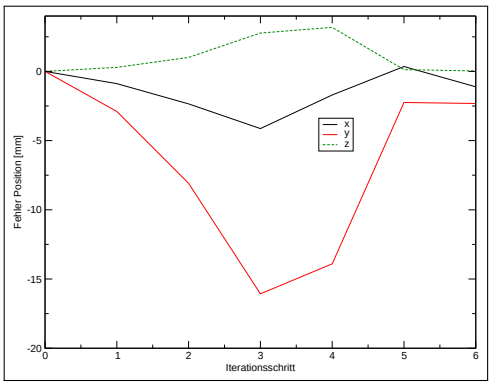
Nach 6 Regelschritten erreicht der Trust-Region-Regler mit PMJ-Modell die Teachpose. Die Ergebnisse in Abbildung 4.32 sind vergleichbar mit denen des Trust-Region-Reglers mit konstanter Jacobimatrix.



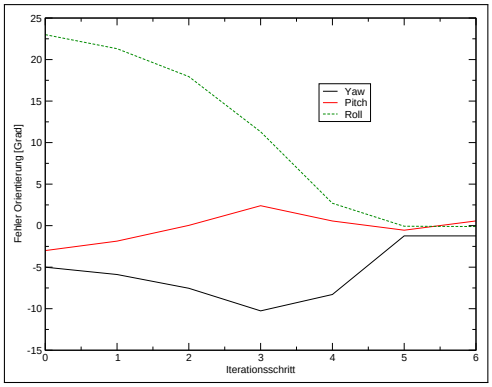
(a) Sensorspur der dritten Lage



(b) Bildfehler der dritten Lage



(c) Fehler der Kameraposition

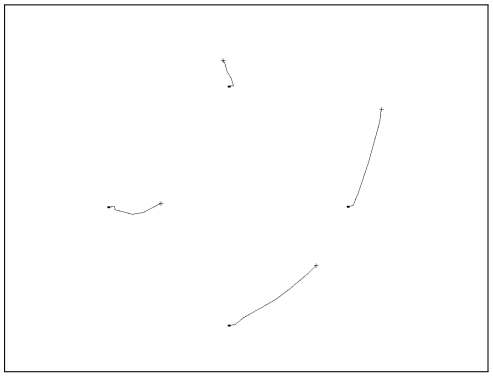


(d) Fehler der Kameraorientierung

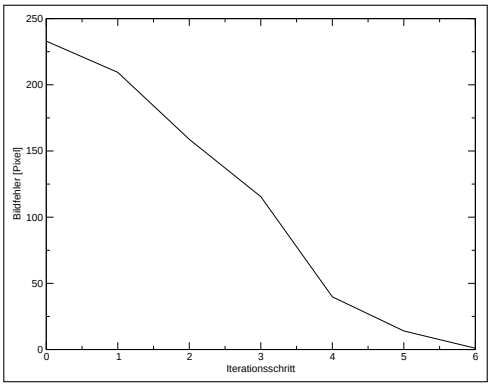
Abbildung 4.32: Trust-Region-Regler mit PMJ-Modell, Lage 3

Trust-Region-Regler in Zylinderkoordinaten

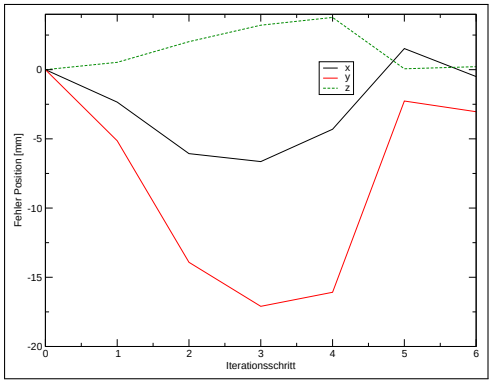
Auch der Trust-Region-Regler in Zylinderkoordinaten konvergiert nach 6 Iterationen.



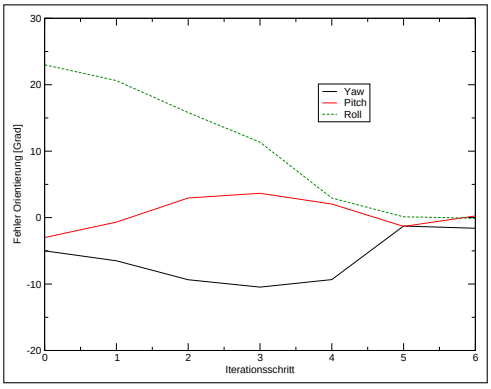
(a) Sensorspur der dritten Lage



(b) Bildfehler der dritten Lage



(c) Fehler der Kameraposition

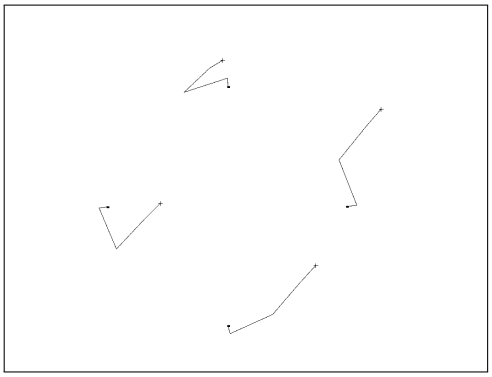


(d) Fehler der Kameraorientierung

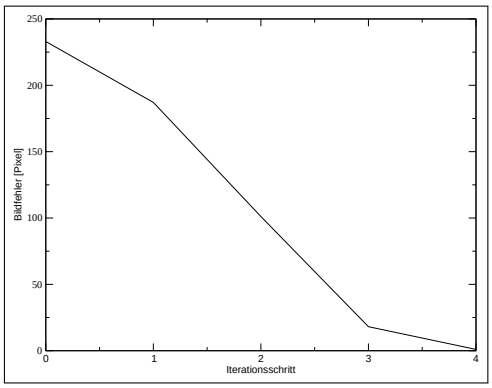
Abbildung 4.33: Trust-Region-Regler in Zylinderkoordinaten, Lage 3

Dog-Leg-Regler mit konstanter Jacobimatrix

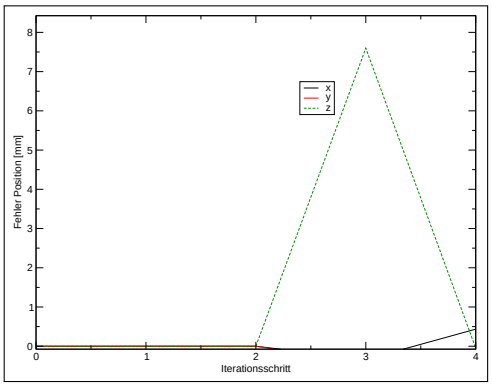
Der Dog-Leg-Regler mit konstanter Jacobimatrix konvergiert bereits nach 4 Iterationen. In den ersten beiden Schritten wird das Gradientenverfahren benutzt. Dadurch werden wieder zuerst Korrekturen bzgl. des Yaw- und Pitch-Winkels durchgeführt (siehe Abbildun 4.34(d)). Beim dritten Iterationsschritt bewegt sich der Roboter um 8 mm auf das Objekt zu, wodurch der Positionsfehler der Kamera steigt (siehe Abbildung 4.34(c)). Im nächsten Schritt wird die Ziellage erreicht.



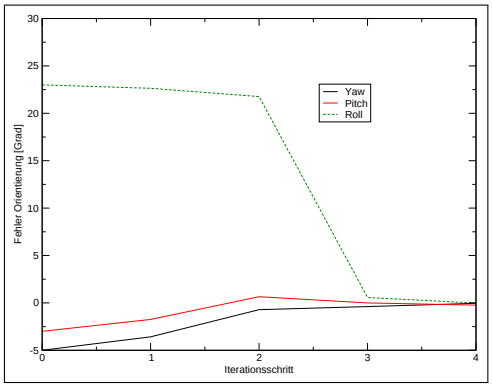
(a) Sensorspur der dritten Lage



(b) Bildfehler der dritten Lage



(c) Fehler der Kameraposition

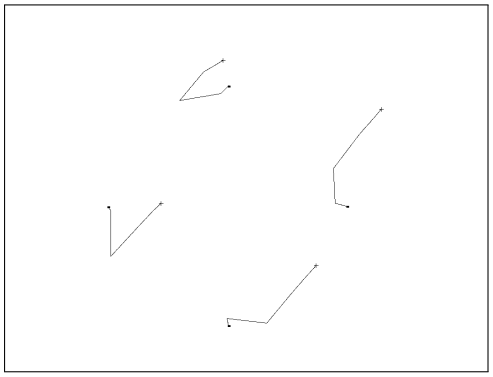


(d) Fehler der Kameraorientierung

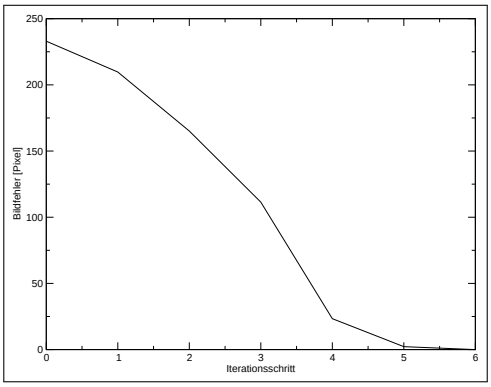
Abbildung 4.34: DogLeg-Regler mit konstanter Jacobimatrix, Lage 3

Dog-Leg-Regler mit dynamischer Jacobimatrix

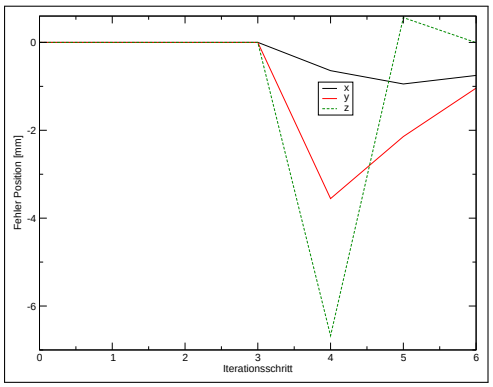
Die Variante des Dog-Leg-Reglers mit dynamischer Jacobimatrix konvergiert nach 6 Regelschritten. Das verhalten ähnelt dem des vorherigen Ansatzes. Allerdings bewegt sich hier der Roboterarm im vierten Iterationsschritt von dem Objekt weg (siehe Abbildung 4.35(c)).



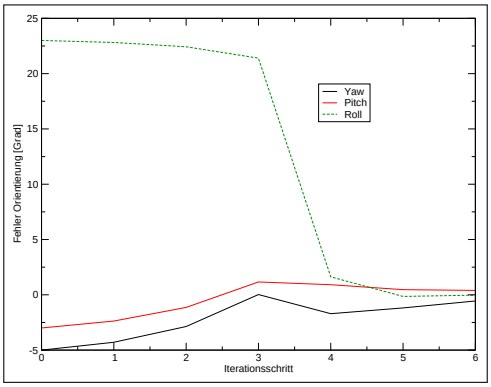
(a) Sensorspur der dritten Lage



(b) Bildfehler der dritten Lage



(c) Fehler der Kameraposition

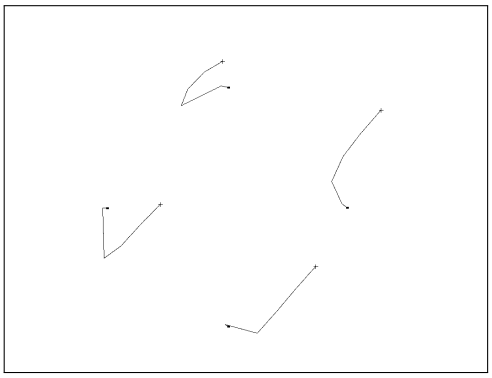


(d) Fehler der Kameraorientierung

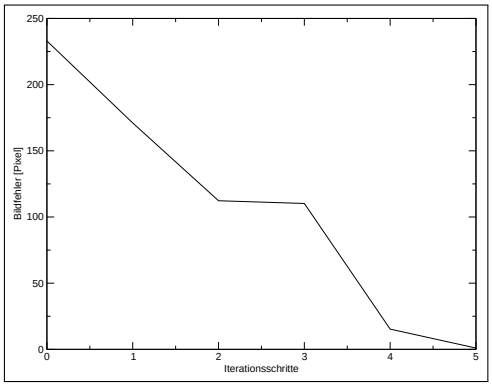
Abbildung 4.35: DogLeg-Regler mit dynamischer Jacobimatrix, Lage 3

Dog-Leg-Regler mit MJP-Modell

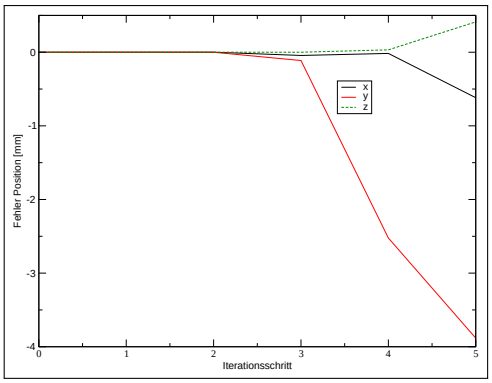
Der Dog-Leg-Regler mit MJP-Modell erreicht die dritte Ausgangslage nach 6 Iterationsschritten. Bei dieser Variante wird keine Bewegung entlang der z-Achse durchgeführt. Allerdings kommt es im vierten Regelschritt zu einer Vergrößerung des Kamerafehlers bzgl. der y-Achse. Dies wird aber durch eine gegenläufige Bewegung des Yaw-Winkels ausgeglichen.



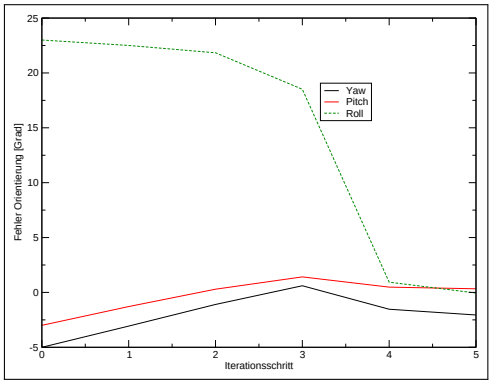
(a) Sensorspur der dritten Lage



(b) Bildfehler der dritten Lage



(c) Fehler der Kameraposition

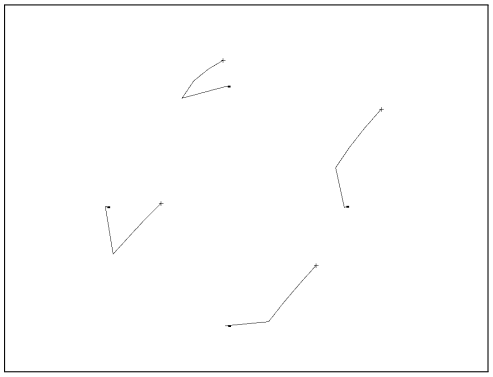


(d) Fehler der Kameraorientierung

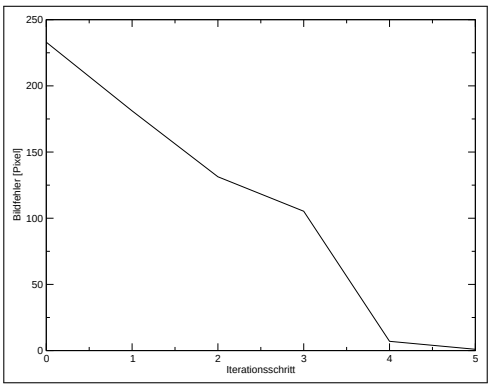
Abbildung 4.36: DogLeg-Regler mit MJP-Modell, Lage 3

Dog-Leg-Regler mit PMJ-Modell

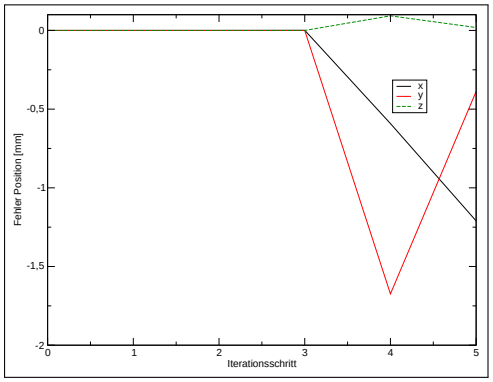
Die Variante des Dog-Leg-Reglers mit dem PMJ-Modell erreicht die Teachpose nach 5 Regelschritten. Die bei den drei vorherigen betrachteten Dog-Leg-Reglern auftretende Verschlechterung des Kamerafehlers ist hier nur noch minimal (siehe Abbildung 4.37 und 4.37(d)).



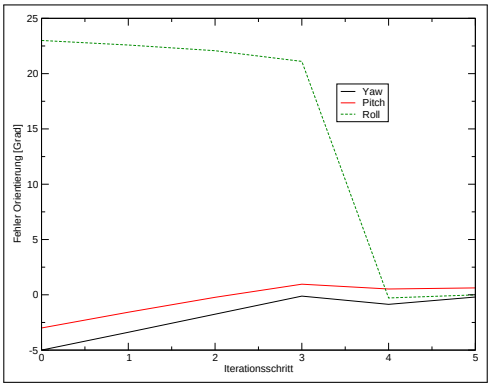
(a) Sensorspur der dritten Lage



(b) Bildfehler der dritten Lage



(c) Fehler der Kameraposition



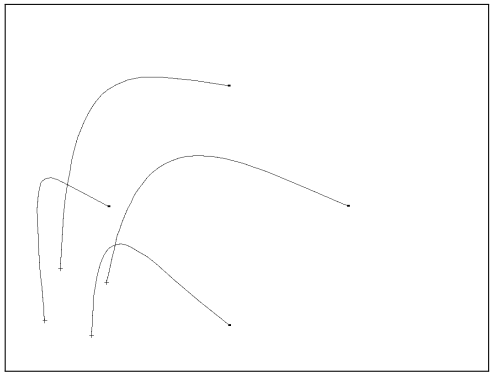
(d) Fehler der Kameraorientierung

Abbildung 4.37: DogLeg-Regler mit PMJ-Modell, Lage 3

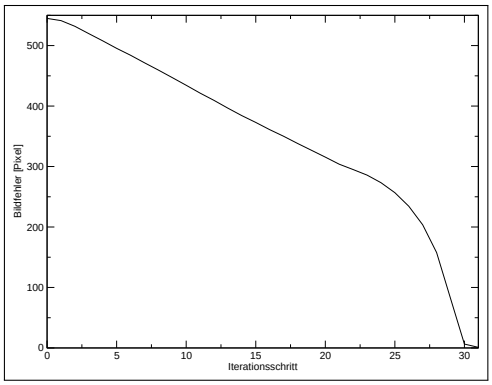
Ausgangslage 4: [150, 90, -200, 10°, -15°, 30°]

Trust-Region-Regler mit konstanter Jacobimatrix

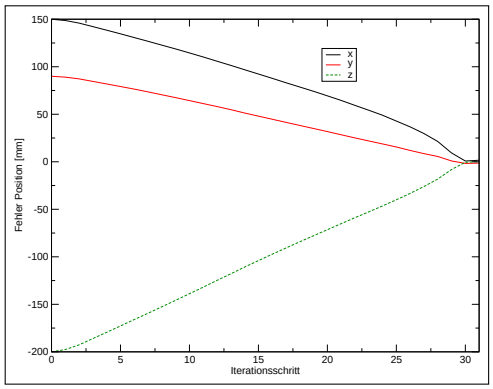
Die Ergebnisse des Trust-Region-Regler mit konstanter Jacobimatrix in Abbildung 4.38 sind mit denen des traditionellen Ansatzes vergleichbar (siehe Abbildung 3.23). Mit dem hier benutzten Regler erreicht der Roboter nach 31 Iterationen die Teachpose. Im Gegensatz zum traditionllen Ansatz sinkt hier der Bildfehler anfangs langsamer und ab Iterationsschritt 25 schneller (siehe Abbildung 3.23(b)).



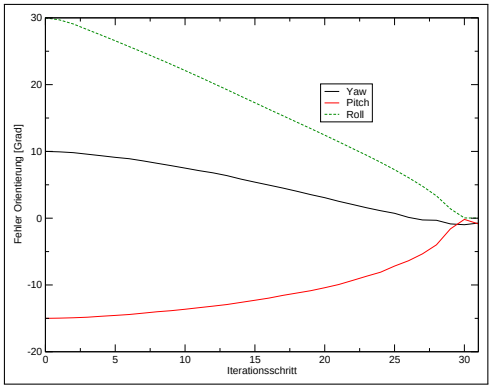
(a) Sensorspur der vierten Lage



(b) Bildfehler der vierten Lage



(c) Fehler der Kameraposition

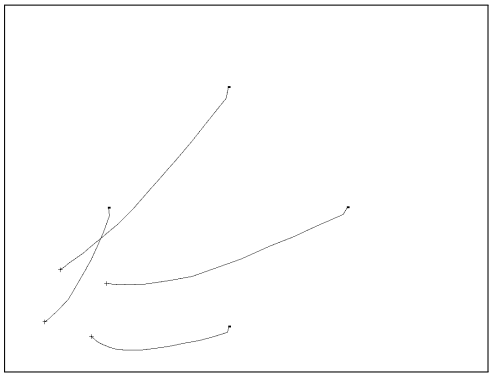


(d) Fehler der Kameraorientierung

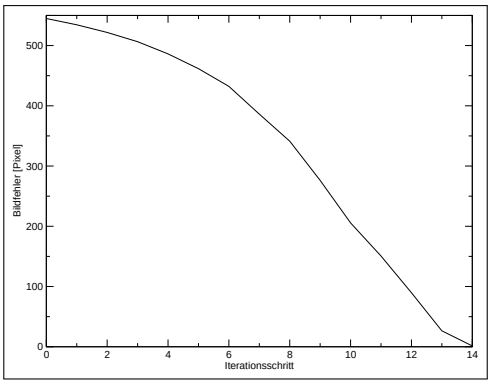
Abbildung 4.38: Trust-Region-Regler mit konstanter Jacobimatrix, Lage 4

Trust-Region-Regler mit dynamischer Jacobimatrix

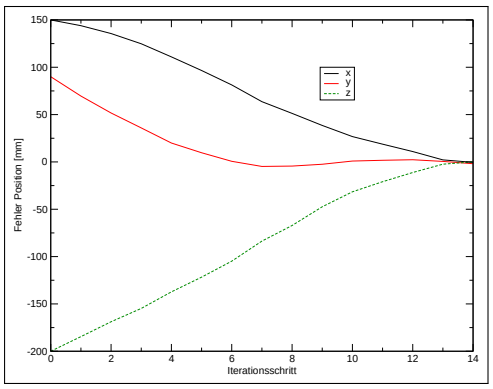
Bei dem Trust-Region-Regler mit dynamischer Jacobimatrix wird die Verbesserung bzgl. des Traditionellen Reglers besonders deutlich. Der Regler konvergiert, ohne dass Parameter manuell angepasst werden müssen. Die Bewegung der Objektmarkierungen zum unteren Rand ist hier nicht mehr so stark wie beim traditionellen Ansatz (siehe Abbildung 4.39(a)). Der Regler erreicht seine Zielpose nach 14 Regelschritten.



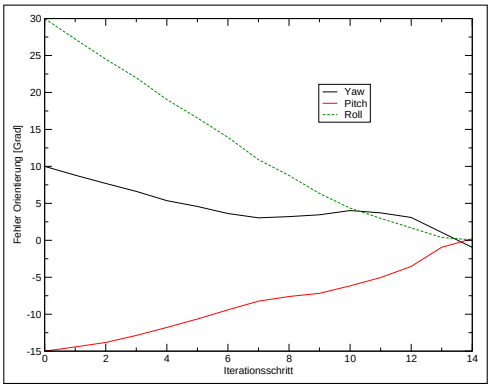
(a) Sensorspur der vierten Lage



(b) Bildfehler der vierten Lage



(c) Fehler der Kameraposition

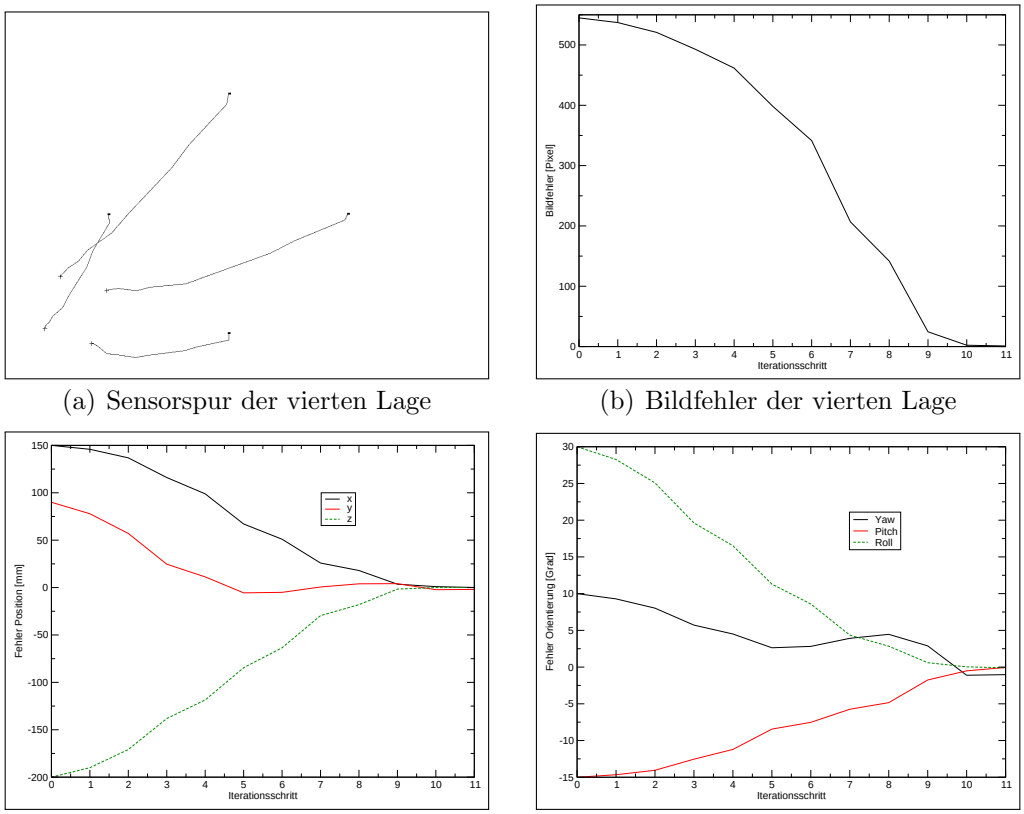


(d) Fehler der Kameraorientierung

Abbildung 4.39: Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 4

Trust-Region-Regler mit MJP-Modell

Bei der Trust-Region-Variante mit dem MJP-Regler und der dynamischen Jacobimatrix als Modell wird ein ähnliches Ergebnis wie beim vorherigen Regler erreicht. Der Regler konvergiert in dieser Lage nach 11 Iterationen.



(a) Sensorspur der vierten Lage

(b) Bildfehler der vierten Lage

(c) Fehler der Kameraposition

(d) Fehler der Kameraorientierung

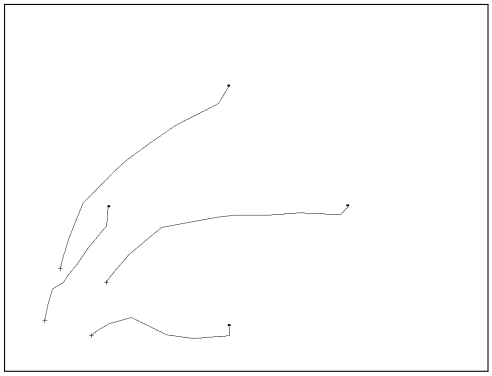
Abbildung 4.40: Trust-Region-Regler mit MJP-Modell, Lage 4

Trust-Region-Regler mit PMJ-Modell

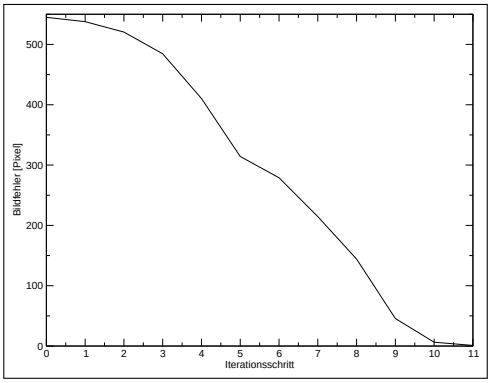
Wie schon bei dem Ansatz mit konstantem Dämpfungsfaktor, verlassen auch hier die Objektmarkierungen den Bildbereich. Die Schwierigkeiten des Reglers lassen sich also nicht durch einen Trust-Region-Ansatz beseitigen.

Trust-Region-Regler in Zylinderkoordinaten

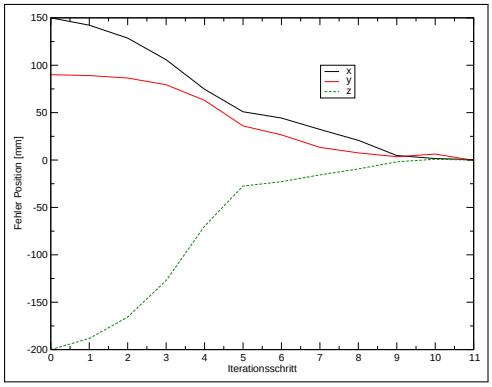
Der Trust-Region-Regler in Zylinderkorrdinaten erreicht die Teachpose nach 11 Regelschritten. Genau wie beim Ansatz mit konstantem Dämpfungsfaktor bewegen sich die Spuren der Objektmarkierungen auf dem Kamerasensor (siehe Abbildung 4.41(a)) vom Bildrand weg. Die Geschwindigkeit wurde durch den Trust-Region-Ansatz um den Faktor 5 gesteigert.



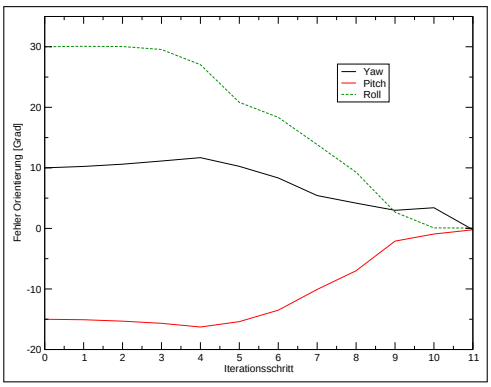
(a) Sensorspur der vierten Lage



(b) Bildfehler der vierten Lage



(c) Fehler der Kameraposition

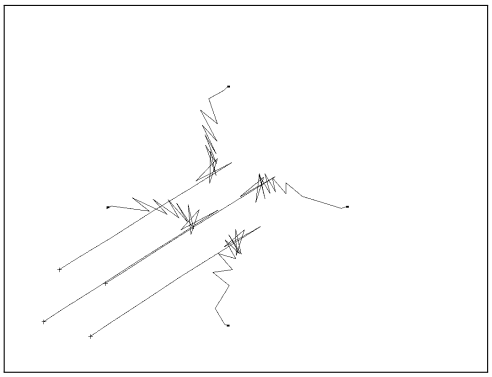


(d) Fehler der Kameraorientierung

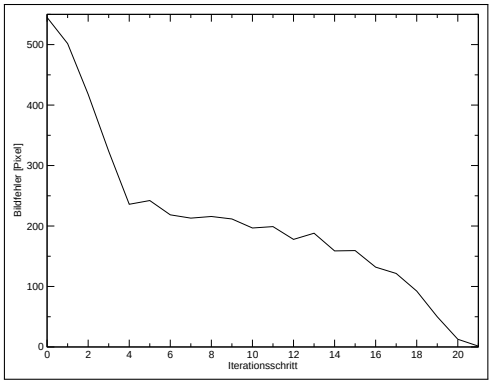
Abbildung 4.41: Trust-Region-Regler in Zylinderkoordinaten, Lage 4

Dog-Leg-Regler mit konstanter Jacobimatrix

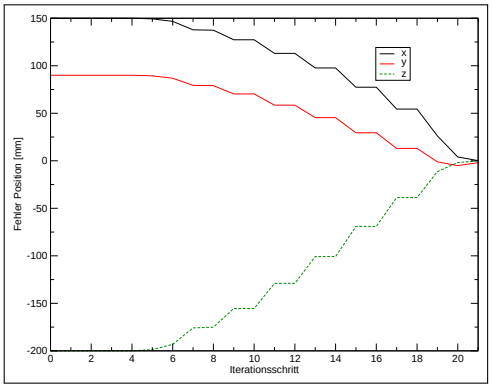
In Abbildung 4.42(a) des Dog-Leg-Reglers mit konstanter Jacobimatrix lassen sich starke Zick-Zack-Bewegungen erkennen. Diese werden durch ein häufiges Wechseln zwischen Gradientenverfahren und Dog-Leg-Schritt hervorgerufen. Der Regler konvergiert nach 21 Iterationen.



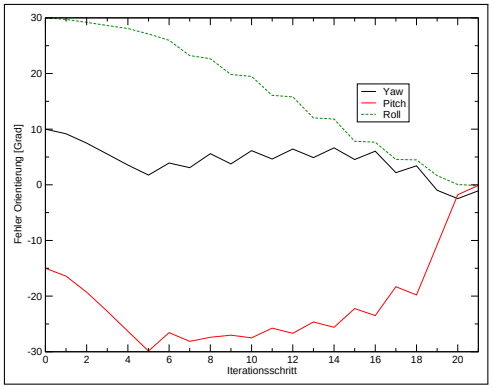
(a) Sensorspur der vierten Lage



(b) Bildfehler der vierten Lage



(c) Fehler der Kameraposition



(d) Fehler der Kameraorientierung

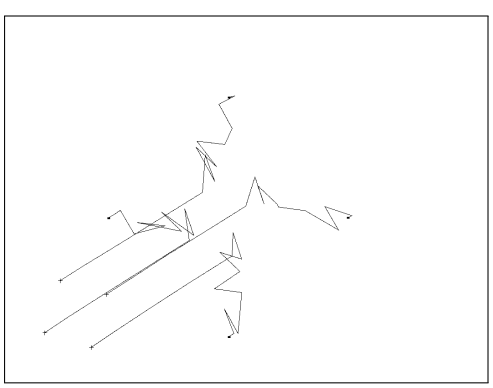
Abbildung 4.42: DogLeg-Regler mit konstanter Jacobimatrix, Lage 4

Dog-Leg-Regler mit dynamischer Jacobimatrix und dem MJP-Modell

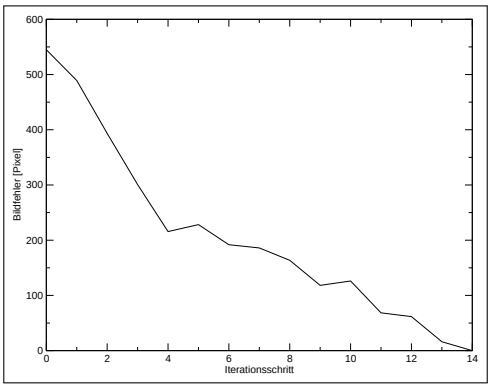
Beide Regler schaffen es nicht in dieser Ausgangslage die Teachpose zu erreichen. Nach dem ersten Dog-Leg-Schritt verlassen die Objektmarkierungen das Kamerabild.

Dog-Leg-Regler mit PMJ-Modell

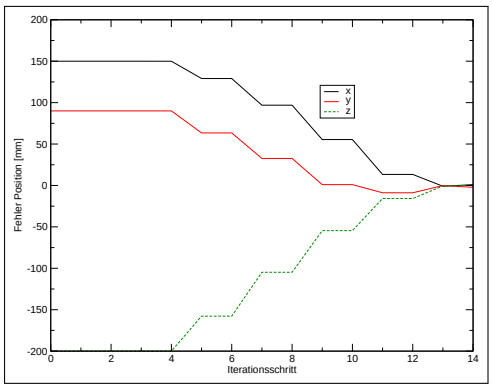
Beim Dog-Leg-Regler mit PMJ-Modell fallen die Schwankungen in Abbildung 4.43(a) nicht so stark aus wie beim Ansatz mit konstanter Jacobimatrix. Der Regler erreicht seine Ziellage nach 14 Regelschritten.



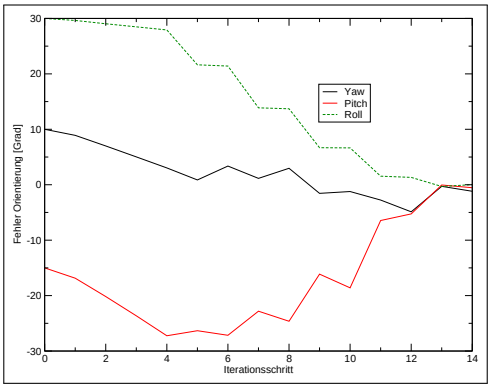
(a) Sensorspur der vierten Lage



(b) Bildfehler der dritten Lage



(c) Fehler der Kameraposition



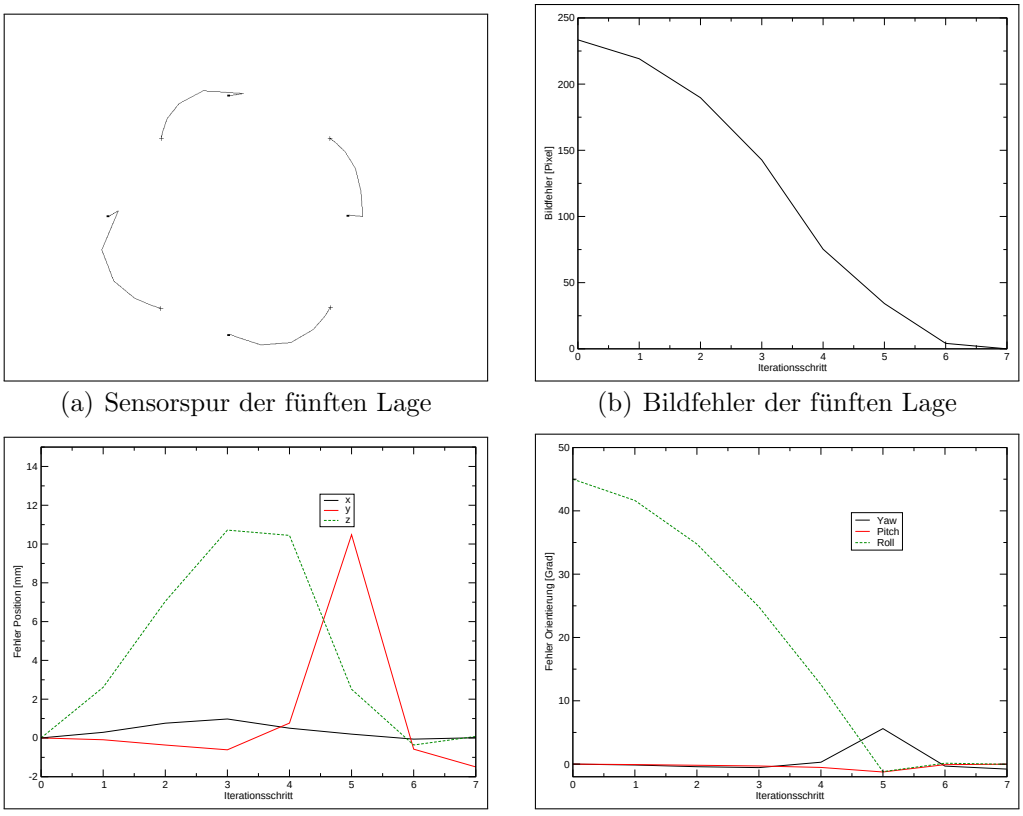
(d) Fehler der Kameraorientierung

Abbildung 4.43: DogLeg-Regler mit PMJ-Modell, Lage 4

Ausgangslage 5: [0, 0, 0, 0°, 0°, 45°]

Trust-Region-Regler mit konstanter Jacobimatrix

Bei der fünften Lage lässt sich auch mit der konstanten Jacobimatrix unter Verwendung des Trust-Region-Reglers das Retreat-Advance-Problem beobachten (siehe Abbildung 4.44(c)). Zusätzlich vergrößert sich der Bildfehler auch bzgl. der y-Achse. Das wird aber durch eine Drehung um die x-Achse (siehe Abbildung 4.44(d)) ausgeglichen. Der Regler erreicht seine Ziellage nach 7 Iterationen.



(a) Sensorspur der fünften Lage

(b) Bildfehler der fünften Lage

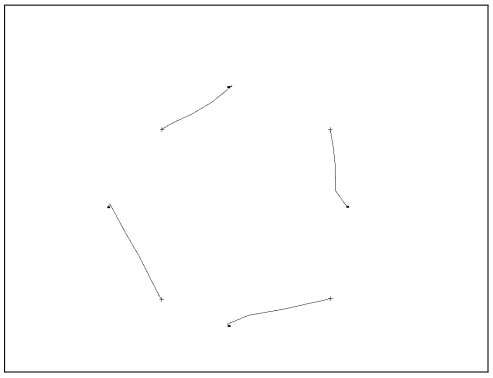
(c) Fehler der Kameraposition

(d) Fehler der Kameraorientierung

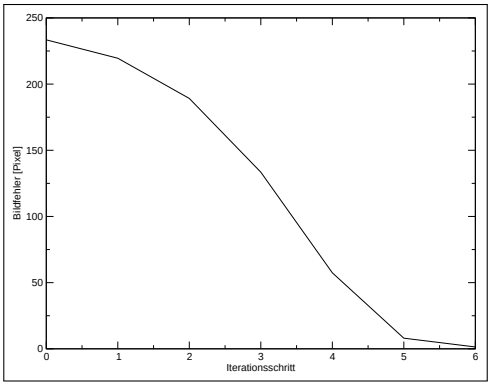
Abbildung 4.44: Trust-Region-Regler mit konstanter Jacobimatrix, Lage 5

Trust-Region-Regler mit dynamischer Jacobimatrix

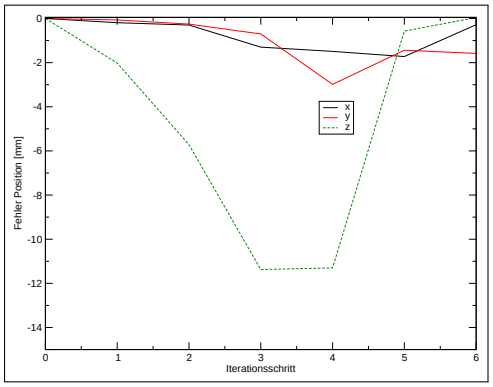
Auch beim Trust-Region-Regler mit dynamischer Jacobimatrix tritt das Retreat-Advance-Problem auf (siehe Abbildung 4.45(a)). Die vorherige Vergrößerung des Kamerafehlers bzgl. der y-Achse fällt hier geringer aus. Nach 6 Iterationen konvergiert der Regler.



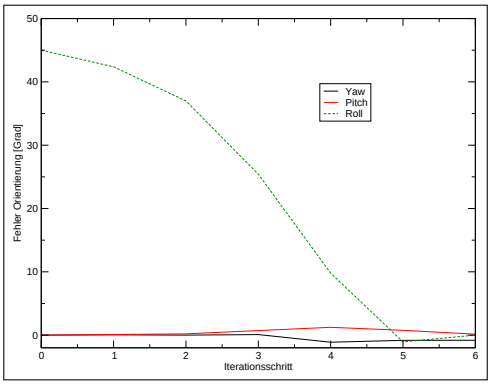
(a) Sensorspur der fünften Lage



(b) Bildfehler der fünften Lage



(c) Fehler der Kameraposition

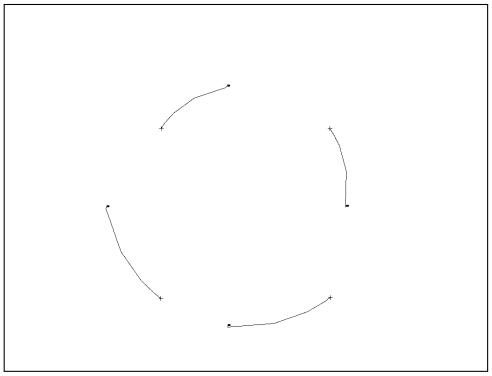


(d) Fehler der Kameraorientierung

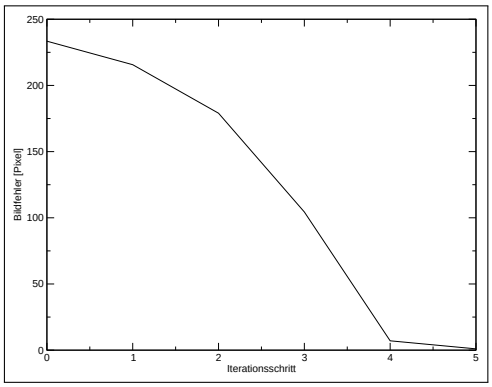
Abbildung 4.45: Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 5

Trust-Region-Regler mit MJP-Modell

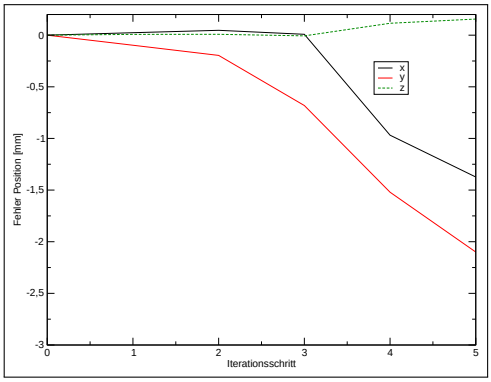
Der Trust-Region-Regler mit dem MJP-Regler und der dynamischen Jacobimatrix als Modell zeigt nur noch geringe Vergrößerungen des Kamerafehlers bzgl. der Position (siehe Abbildung 4.46(c)). Es wird fast eine reine Rotation um die z-Achse durchgeführt. Die Teachpose wird nach 5 Iterationen erreicht.



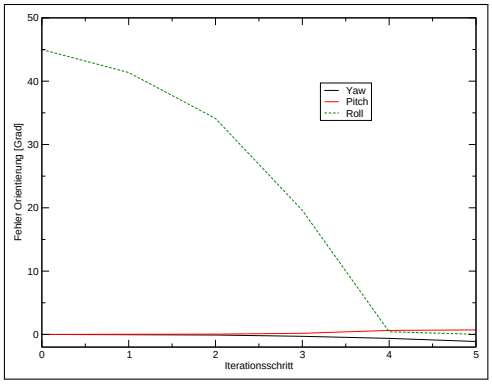
(a) Sensorspur der fünften Lage



(b) Bildfehler der fünften Lage



(c) Fehler der Kameraposition

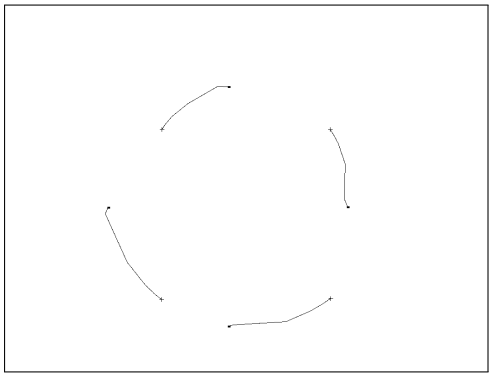


(d) Fehler der Kameraorientierung

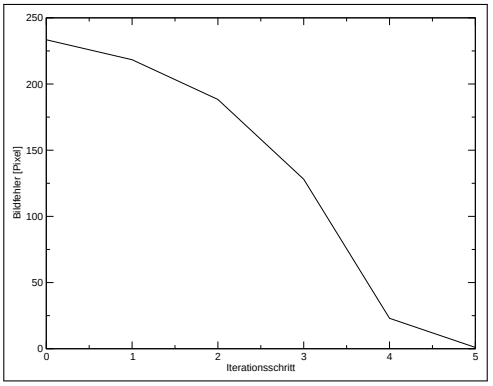
Abbildung 4.46: Trust-Region-Regler mit MJP-Modell, Lage 5

Trust-Region-Regler mit PMJ-Modell

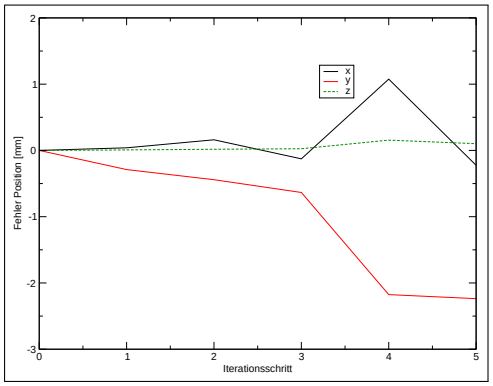
Auch beim Trust-Region-Regler mit PMJ-Modell wird fast nur eine Rotation um die z-Achse durchgeführt. Der Regler konvergiert nach 5 Regelschritten.



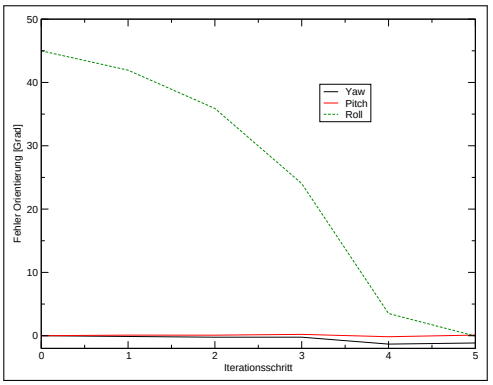
(a) Sensorspur der fünften Lage



(b) Bildfehler der fünften Lage



(c) Fehler der Kameraposition

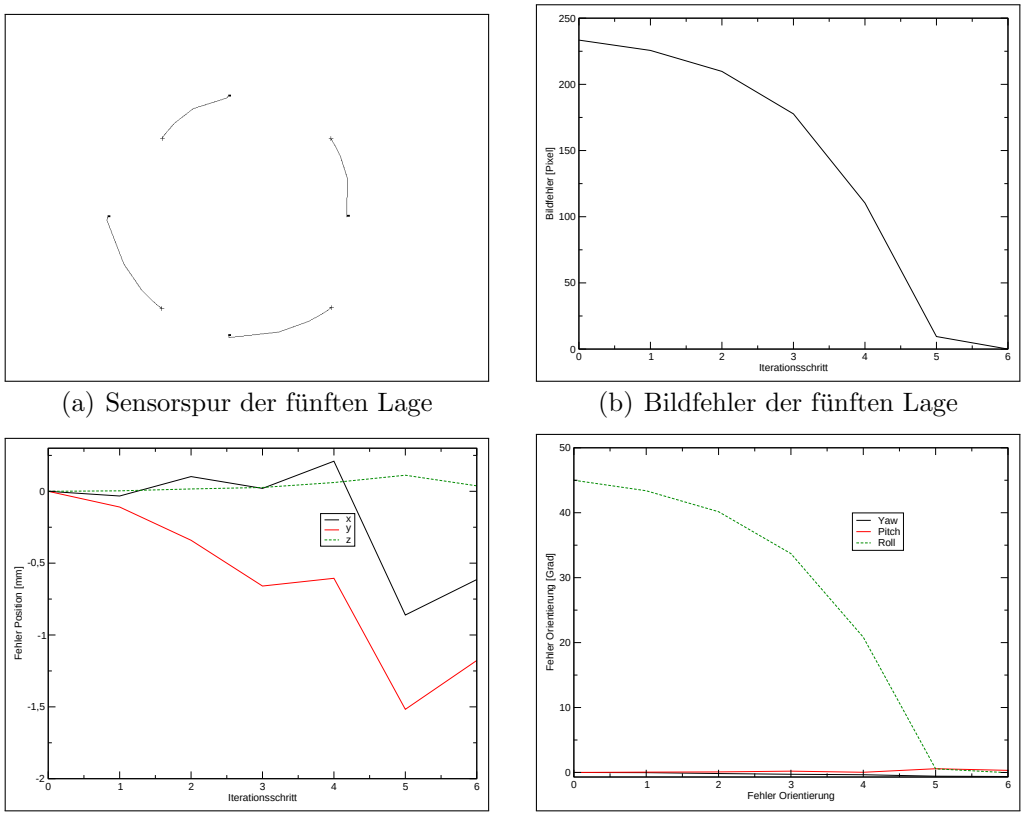


(d) Fehler der Kameraorientierung

Abbildung 4.47: Trust-Region-Regler mit PMJ-Modell, Lage 5

Trust-Region-Regler in Zylinderkoordinaten

Der Trust-Region-Regler in Zylinderkoordinaten erreicht die Ziellage nach 6 Regelschritten. Auch hier tritt das Retreat-Advance-Problem nicht mehr auf.



(a) Sensorspur der fünften Lage

(b) Bildfehler der fünften Lage

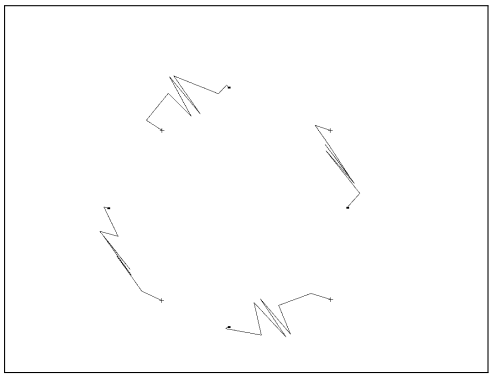
(c) Fehler der Kameraposition

(d) Fehler der Kameraorientierung

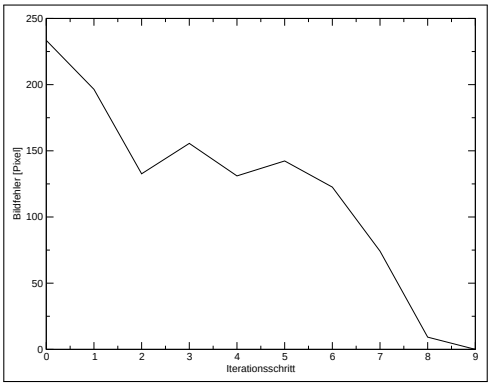
Abbildung 4.48: Trust-Region-Regler in Zylinderkoordinaten, Lage 5

Dog-Leg-Regler mit konstanter Jacobimatrix

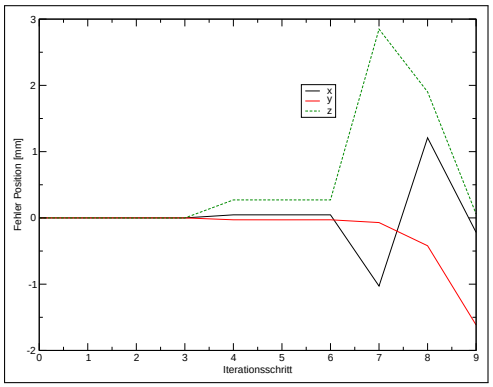
In der fünften Ausgangslage konvergiert der Dog-Leg-Regler mit konstanter Jacobimatrix nach 9 Iterationen. Allerdings erhöht sich beim dritten und beim fünften Regelschritt der Bildfehler (siehe Abbildung 4.49(a)). Insgesamt ist die Bewegung der Bildmerkmale auf dem Kamerasensor sehr unregelmäßig.



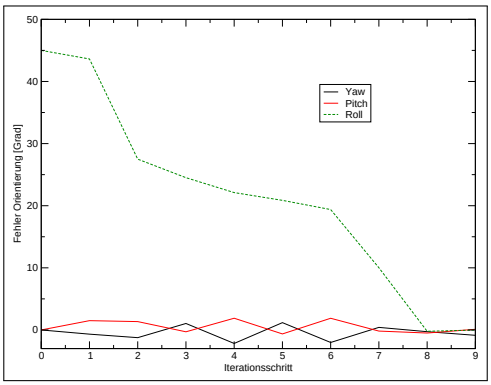
(a) Sensorspur der fünften Lage



(b) Bildfehler der fünften Lage



(c) Fehler der Kameraposition

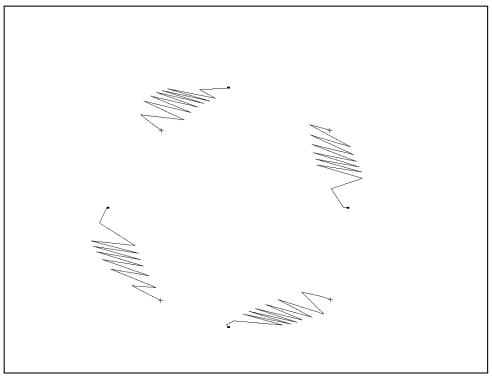


(d) Fehler der Kameraorientierung

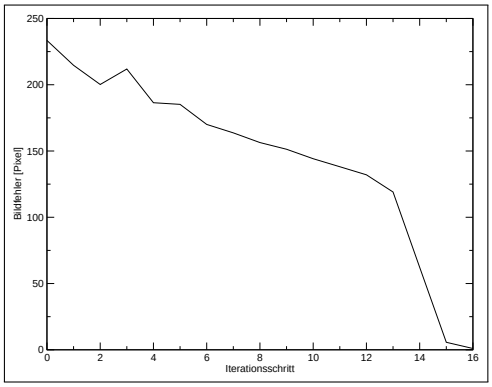
Abbildung 4.49: DogLeg-Regler mit konstanter Jacobimatrix, Lage 5

Dog-Leg-Regler mit dynamischer Jacobimatrix

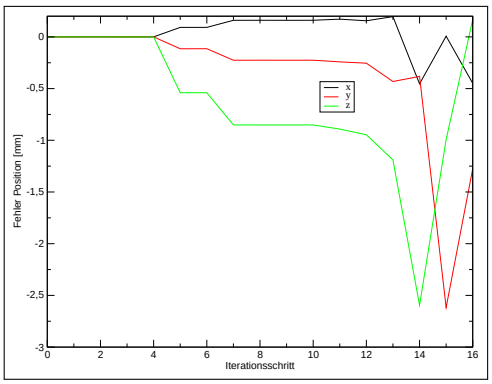
Der Dog-Leg-Regler mit dynamischer Jacobimatrix erreicht seine Ziellage nach 16 Regelschritten. Die Bildmerkmale beschreiben auf dem Sensor noch viel stärkere Zick-Zack-Bewegungen als bei der vorherigen Variante des Reglers (siehe Abbildung 4.50(a)). Hierfür ist ein häufiger Wechsel zwischen dem Gradientenverfahren und dem Dog-Leg-Schritt verantwortlich.



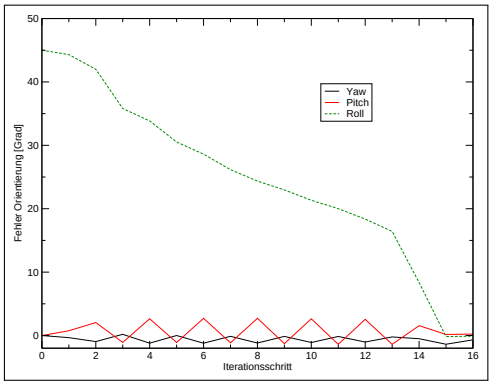
(a) Sensorspur der fünften Lage



(b) Bildfehler der fünften Lage



(c) Fehler der Kameraposition

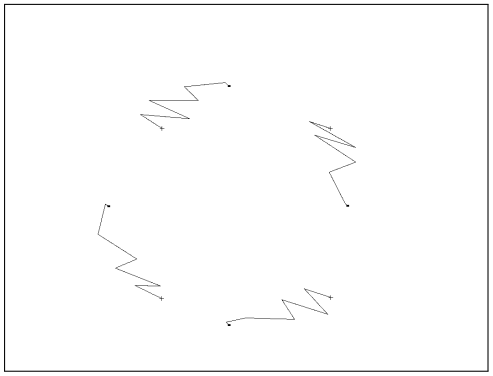


(d) Fehler der Kameraorientierung

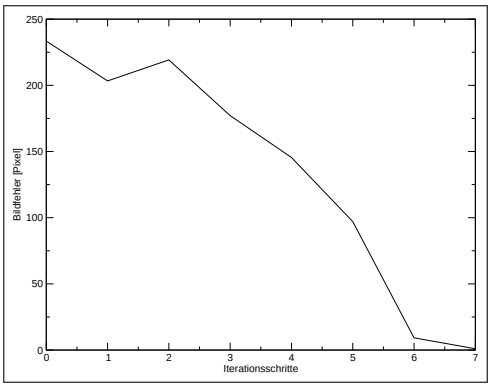
Abbildung 4.50: DogLeg-Regler mit dynamischer Jacobimatrix, Lage 5

Dog-Leg-Regler mit MJP-Regler

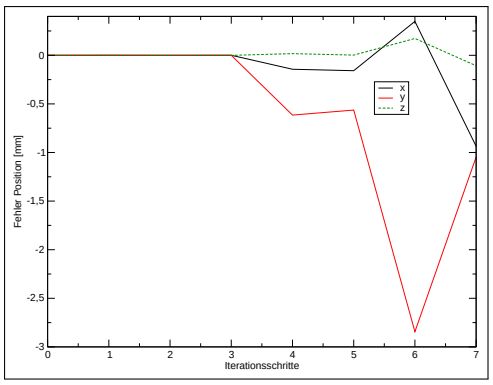
Nach 7 Schritten konvergiert der Dog-Leg-Regler mit dem MJP-Regler und der dynamischen Jacobimatrix als Modell. Auch hier sind die Bewegungen der Objektmarkierungen nicht gleichmäßig (siehe Abbildung 4.51(a)).



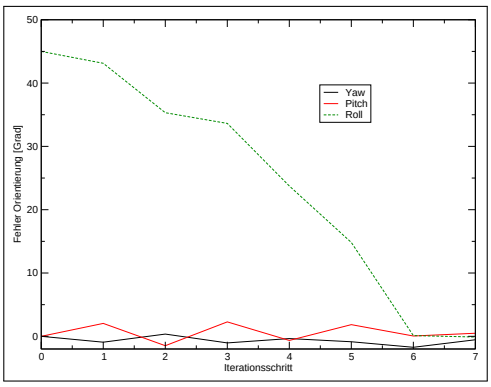
(a) Sensorspur der fünften Lage



(b) Bildfehler der fünften Lage



(c) Fehler der Kameraposition

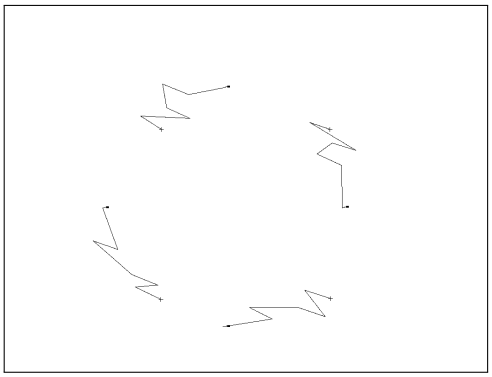


(d) Fehler der Kameraorientierung

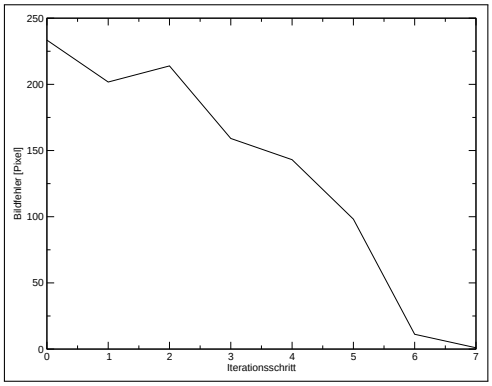
Abbildung 4.51: DogLeg-Regler mit MJP-Modell, Lage 5

Dog-Leg-Regler mit PMJ-Modell

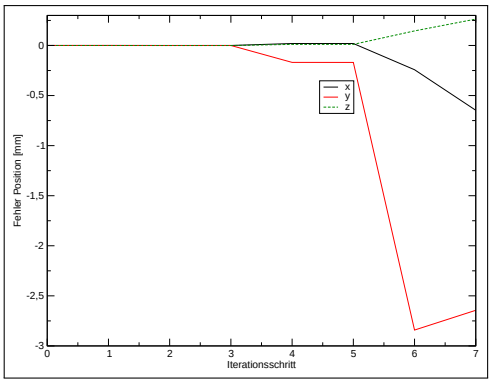
Die Ergebnisse des Dog-Leg-Reglers mit PMJ-Modell sind mit denen in Abbilldung 4.51 identisch. Auch hier konvergiert der Regler nach 7 Iterationen.



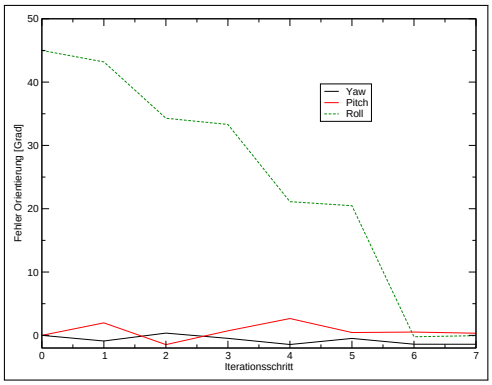
(a) Sensorspur der fünften Lage



(b) Bildfehler der fünften Lage



(c) Fehler der Kameraposition



(d) Fehler der Kameraorientierung

Abbildung 4.52: DogLeg-Regler mit PMJ-Modell, Lage 5

4.4.3 Testläufe am realen System

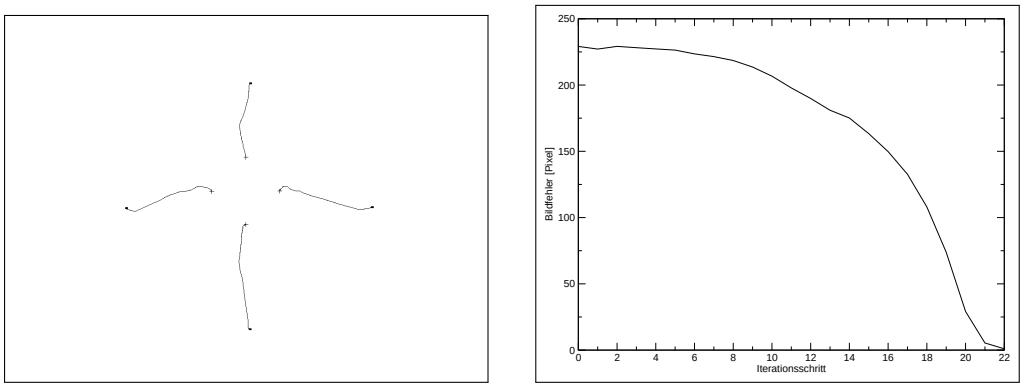
Nun werden die einzelnen Regler wieder auf einem realen System getestet. Genau wie bei den durchgeführten Tests im vorherigen Kapitel terminiert der Trust-Region-Regler mit dem PMJ-Modell nicht in der vierten Lage. Die Objektmarkierungen verlassen auch hier das Bild.

Auch beim Trust-Region-Regler mit der Jacobimatrix im Zylinderkoordinatensystem kommt es zu den gleichen Problemen wie beim Ansatz mit konstanten Dämpfungsfaktor. In der zweiten Lage erhält der Roboter einen Steuerbefehl, der außerhalb seiner Reichweite liegt.

Leider konnten für den Dog-Leg-Regler in der vierten Ausgangslage keine Experimente durchgeführt werden. Es bestand die Gefahr, dass die Kamera während der Reglung beschädigt wird.

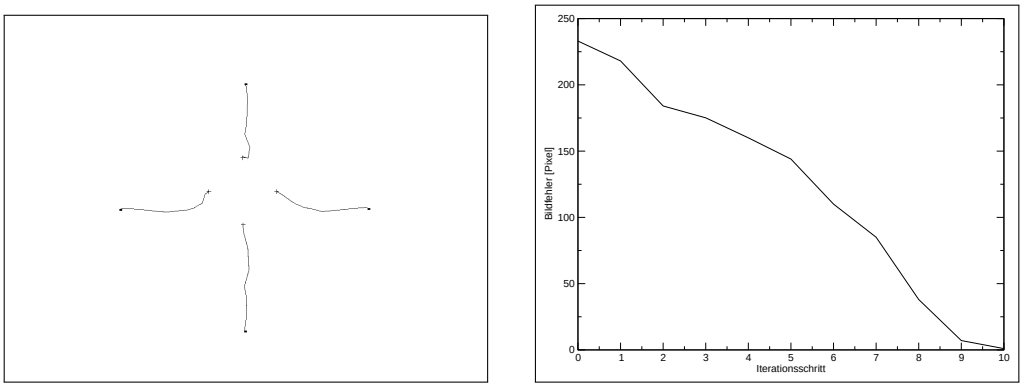
Die Ergebnisse sind in der Tabelle 4.2 zusammengefasst.

Ausgangslage 1: [0, 0, -300, 0°, 0°, 0°]



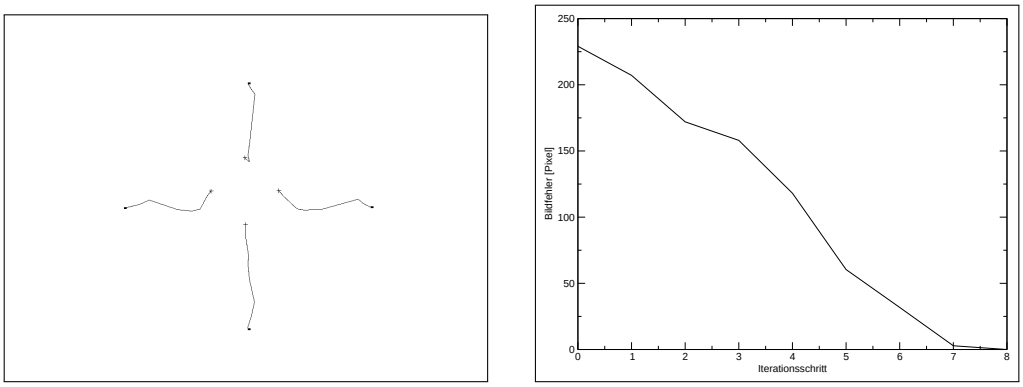
(a) Sensorspur der ersten Lage (b) Bildfehler der ersten Lage

Abbildung 4.53: Trust-Region-Regler mit konstanter Jacobimatrix, Lage 1



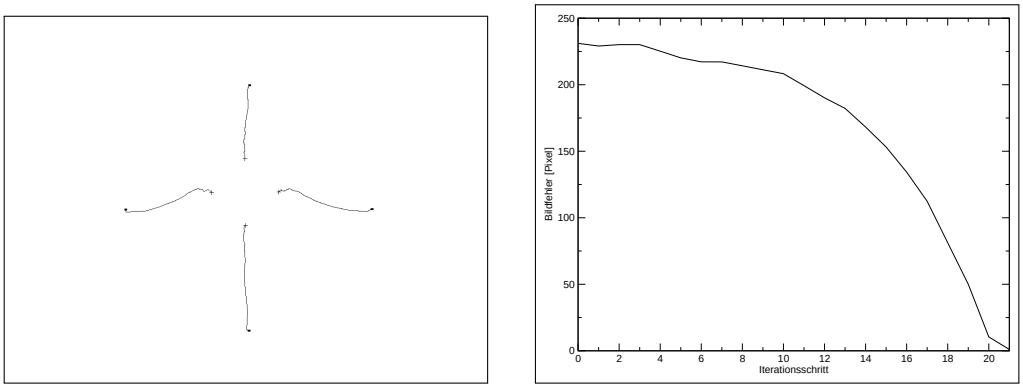
(a) Sensorspur der ersten Lage (b) Bildfehler der ersten Lage

Abbildung 4.54: Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 1



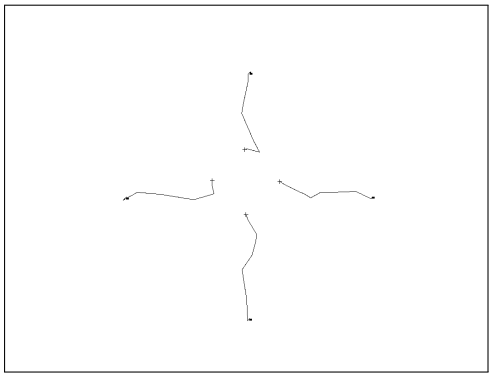
(a) Sensorspur der ersten Lage (b) Bildfehler der ersten Lage

Abbildung 4.55: Trust-Region-Regler mit MJP-Modell, Lage 1

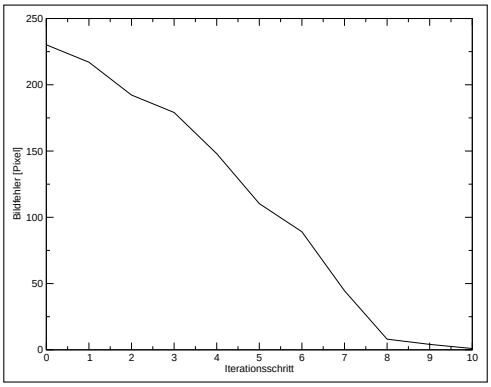


(a) Sensorspur der ersten Lage (b) Bildfehler der ersten Lage

Abbildung 4.56: Trust-Region-Regler mit PMJ-Modell, Lage 1

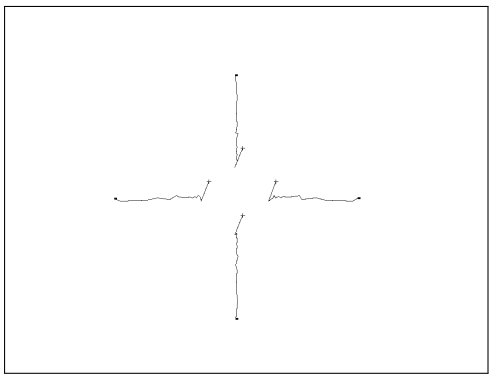


(a) Sensorspur der ersten Lage

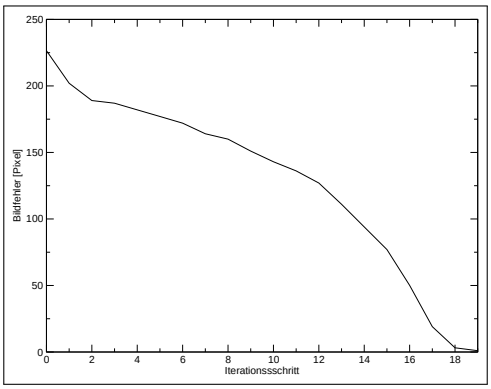


(b) Bildfehler der ersten Lage

Abbildung 4.57: Trust-Region-Regler in Zylinderkoordinaten, Lage 1

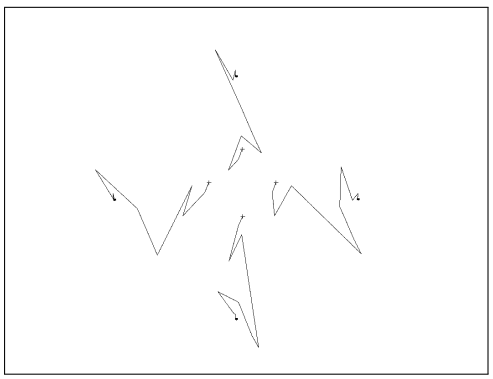


(a) Sensorspur der ersten Lage

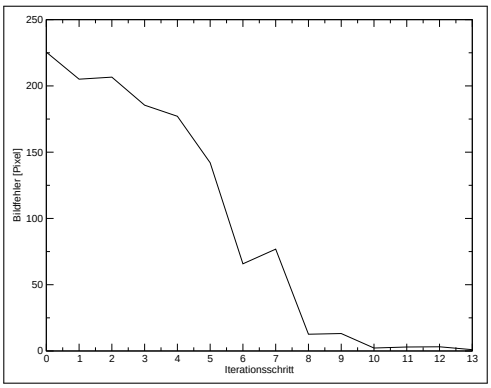


(b) Bildfehler der ersten Lage

Abbildung 4.58: Dog-Leg-Regler mit konstanter Jacobimatrix, Lage 1

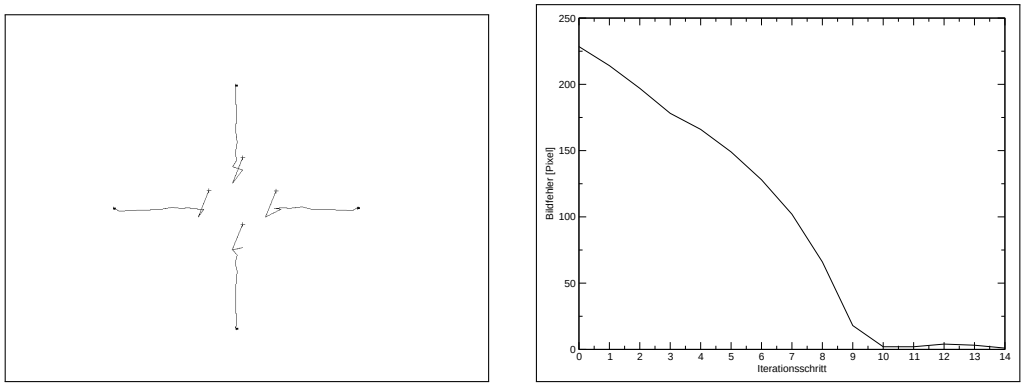


(a) Sensorspur der ersten Lage



(b) Bildfehler der ersten Lage

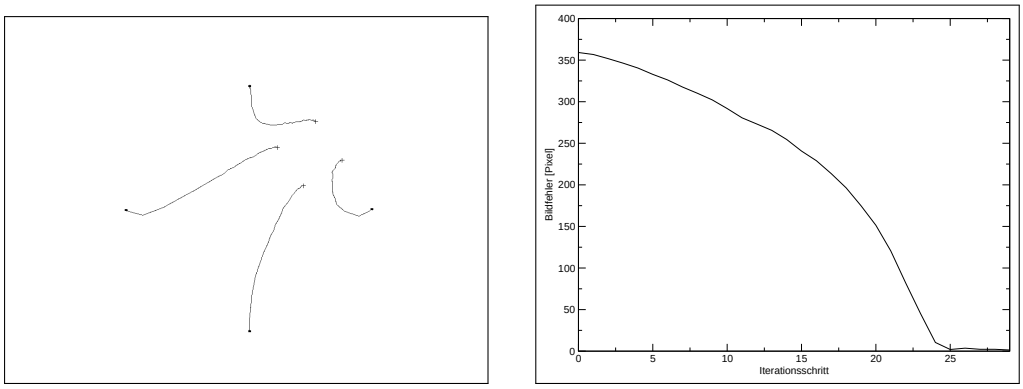
Abbildung 4.59: Dog-Leg-Regler mit dynamischer Jacobimatrix, Lage 1



(a) Sensorspur der ersten Lage (b) Bildfehler der ersten Lage

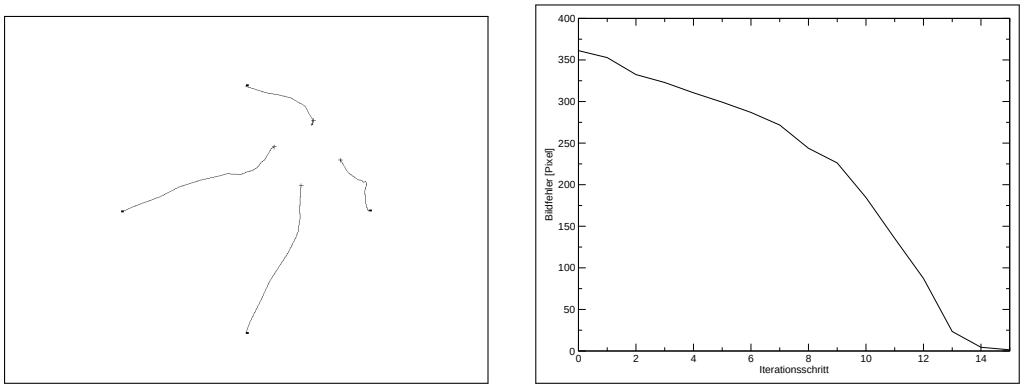
Abbildung 4.60: Dog-Leg-Regler mit PMJ-Modell, Lage 1

Ausgangslage 2: [20, -50, -300, -10°, -10°, -10°]



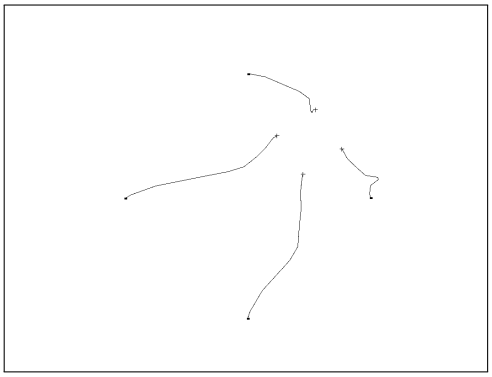
(a) Sensorspur der zweiten Lage (b) Bildfehler der zweiten Lage

Abbildung 4.61: Trust-Region-Regler mit konstanter Jacobimatrix, Lage 2

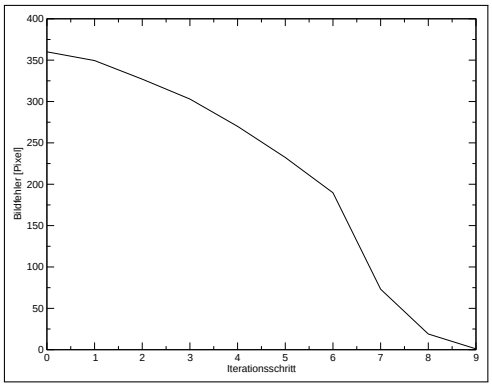


(a) Sensorspur der zweiten Lage (b) Bildfehler der zweiten Lage

Abbildung 4.62: Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 2

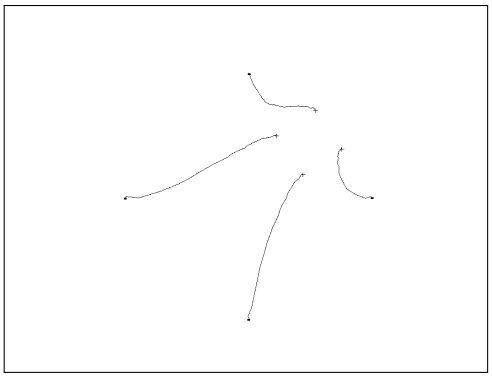


(a) Sensorspur der zweiten Lage

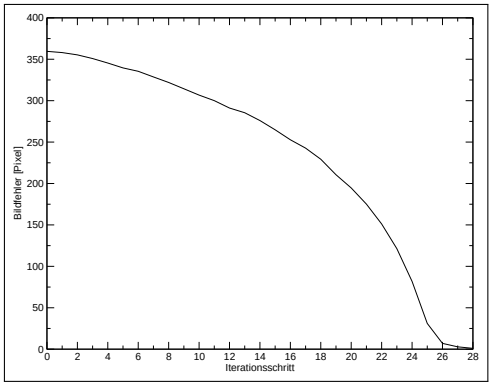


(b) Bildfehler der zweiten Lage

Abbildung 4.63: Trust-Region-Regler mit MJP-Modell, Lage 2

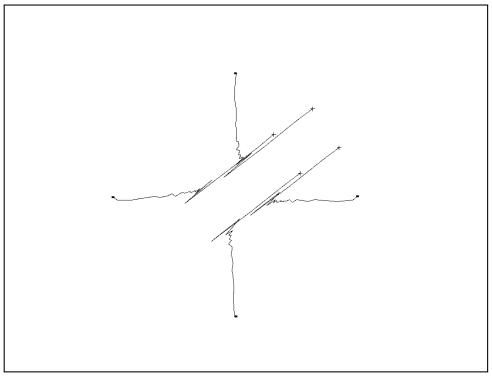


(a) Sensorspur der zweiten Lage

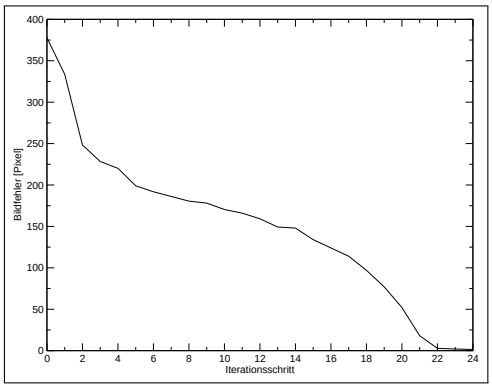


(b) Bildfehler der zweiten Lage

Abbildung 4.64: Trust-Region-Regler mit PMJ-Modell, Lage 2

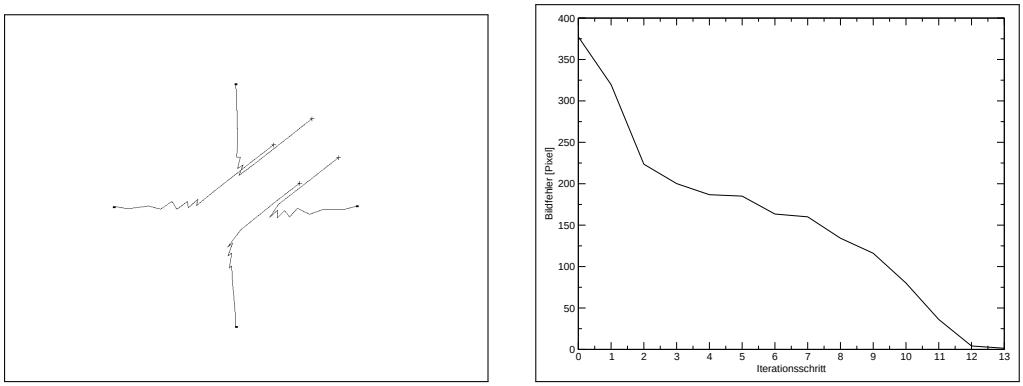


(a) Sensorspur der zweiten Lage



(b) Bildfehler der zweiten Lage

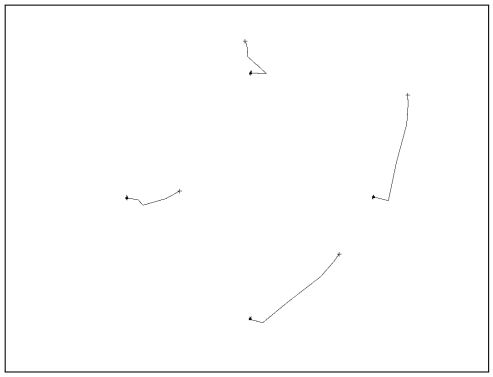
Abbildung 4.65: Dog-Leg-Regler mit konstanter Jacobimatrix, Lage 2



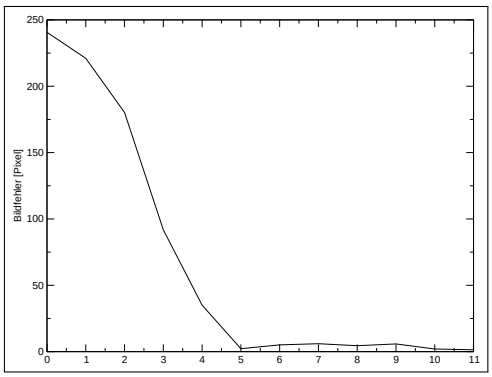
(a) Sensorspur der zweiten Lage (b) Bildfehler der zweiten Lage

Abbildung 4.66: Dog-Leg-Regler mit PMJ-Modell, Lage 2

Ausgangslage 3: [0, 0, 0, -5°, -3°, 23°]

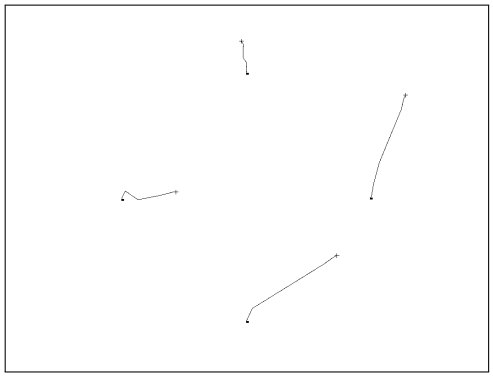


(a) Sensorspur der dritten Lage

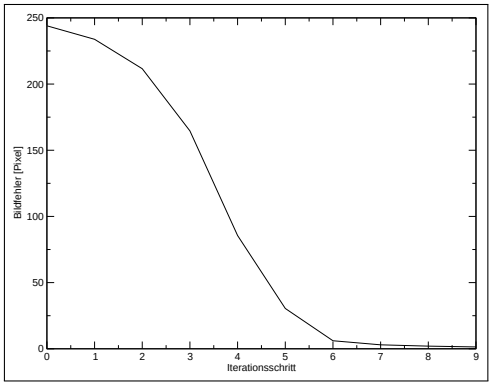


(b) Bildfehler der dritten Lage

Abbildung 4.67: Trust-Region-Regler mit konstanter Jacobimatrix, Lage 3

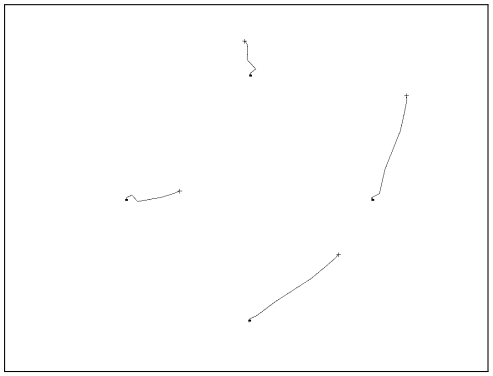


(a) Sensorspur der dritten Lage

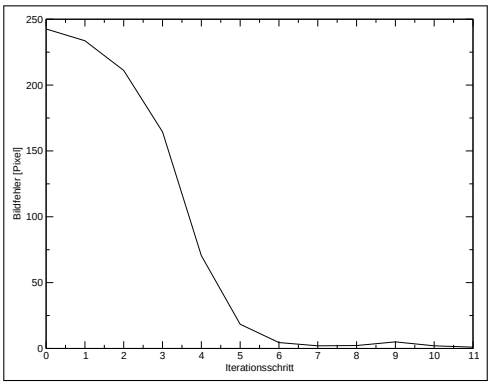


(b) Bildfehler der dritten Lage

Abbildung 4.68: Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 3

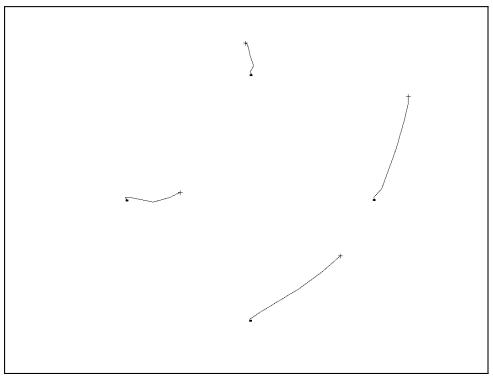


(a) Sensorspur der dritten Lage

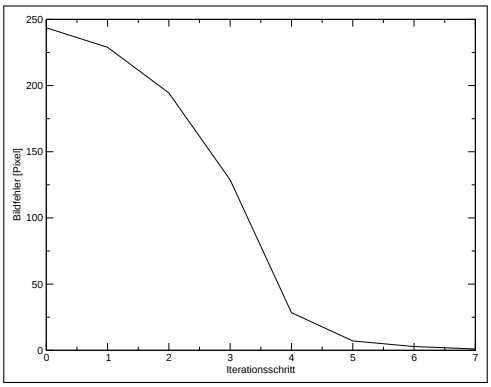


(b) Bildfehler der dritten Lage

Abbildung 4.69: Trust-Region-Regler mit MJP-Modell, Lage 3

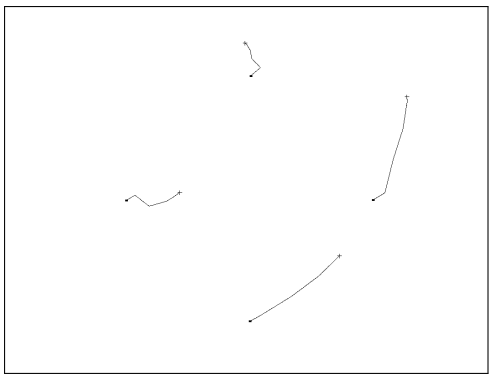


(a) Sensorspur der dritten Lage

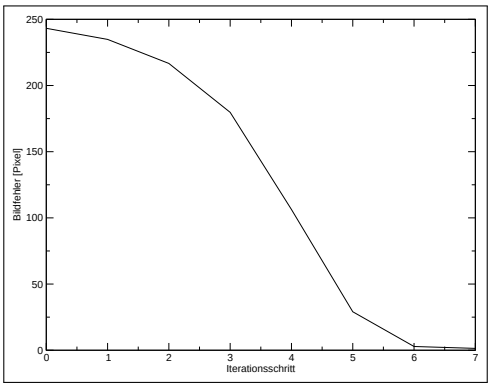


(b) Bildfehler der dritten Lage

Abbildung 4.70: Trust-Region-Regler mit PMJ-Modell, Lage 3

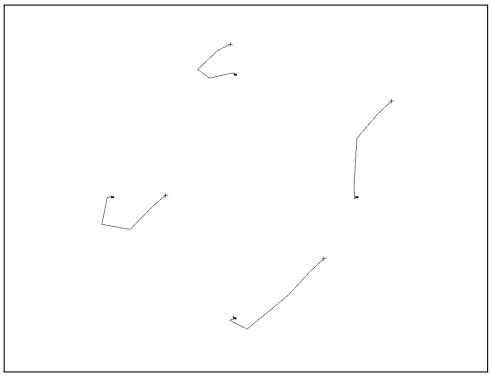


(a) Sensorspur der dritten Lage

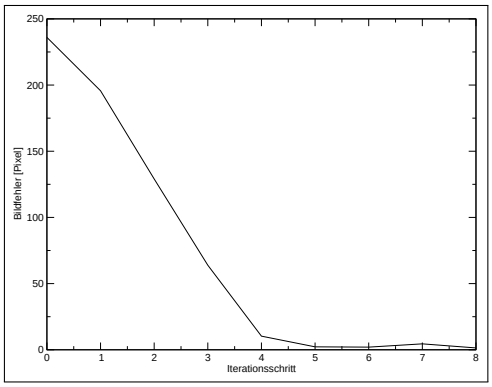


(b) Bildfehler der dritten Lage

Abbildung 4.71: Trust-Region-Regler in Zylinderkoordinaten, Lage 3

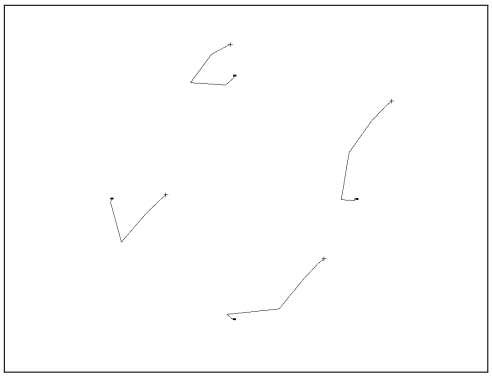


(a) Sensorspur der dritten Lage

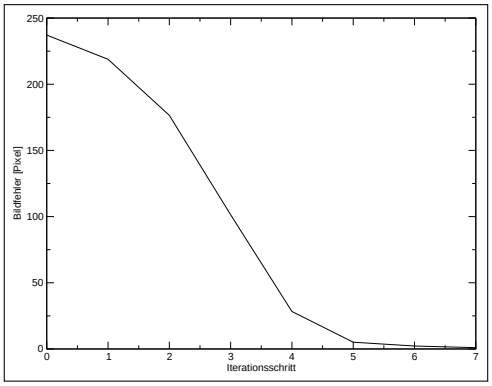


(b) Bildfehler der dritten Lage

Abbildung 4.72: Dog-Leg-Regler mit konstanter Jacobimatrix, Lage 3

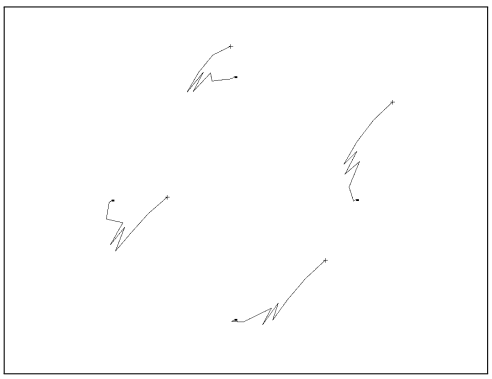


(a) Sensorspur der dritten Lage

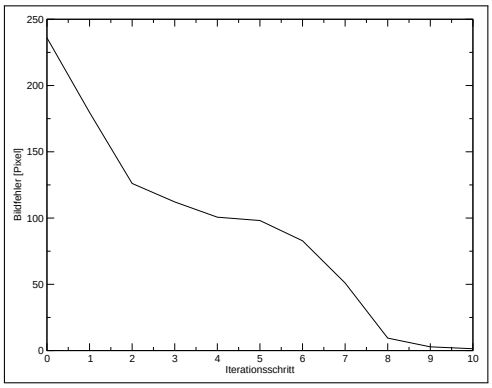


(b) Bildfehler der dritten Lage

Abbildung 4.73: Dog-Leg-Regler mit dynamischer Jacobimatrix, Lage 3

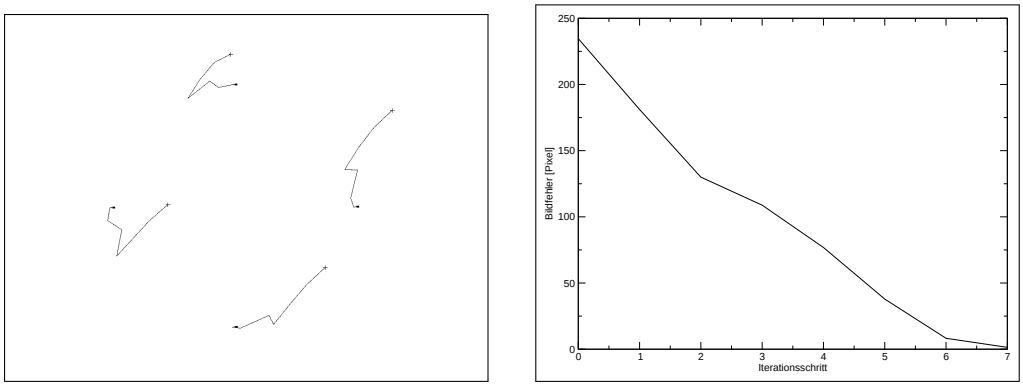


(a) Sensorspur der dritten Lage



(b) Bildfehler der dritten Lage

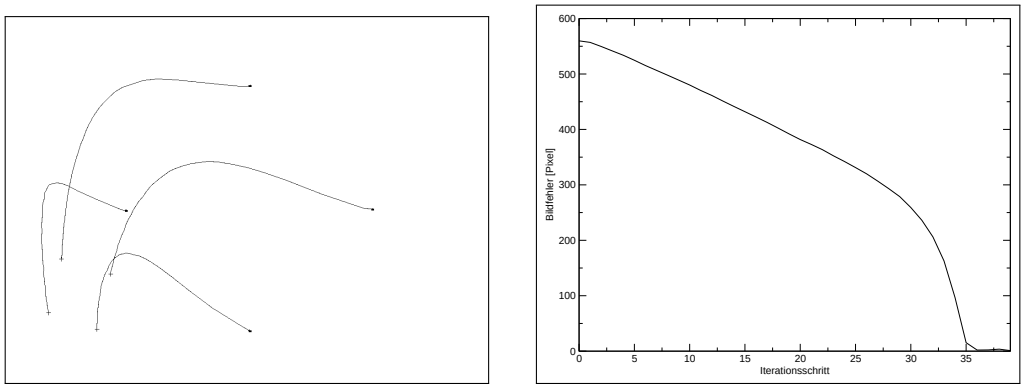
Abbildung 4.74: Dog-Leg-Regler mit MJP-Modell, Lage 3



(a) Sensorspur der dritten Lage (b) Bildfehler der dritten Lage

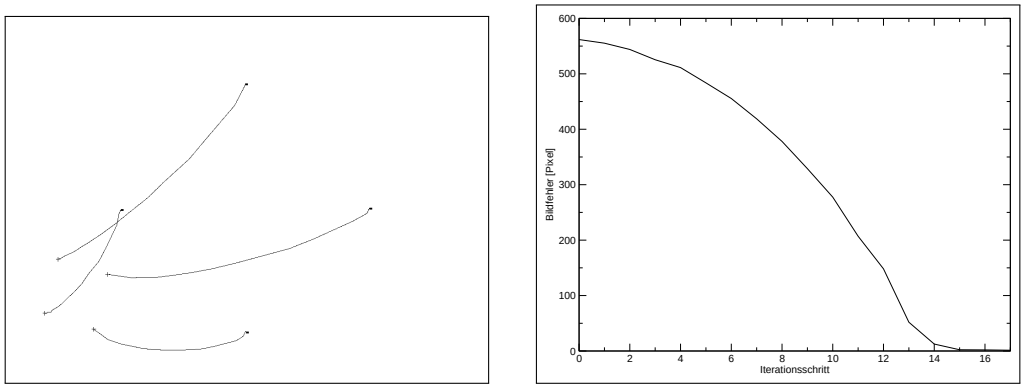
Abbildung 4.75: Dog-Leg-Regler mit PMJ-Modell, Lage 3

Ausgangslage 4: [150, 90, -200, 10°, -15°, 30°]



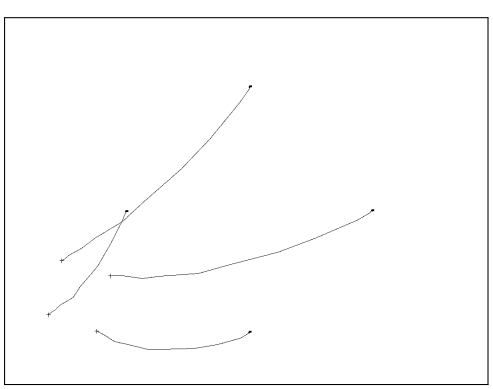
(a) Sensorspur der vierten Lage (b) Bildfehler der vierten Lage

Abbildung 4.76: Trust-Region-Regler mit konstanter Jacobimatrix, Lage 4

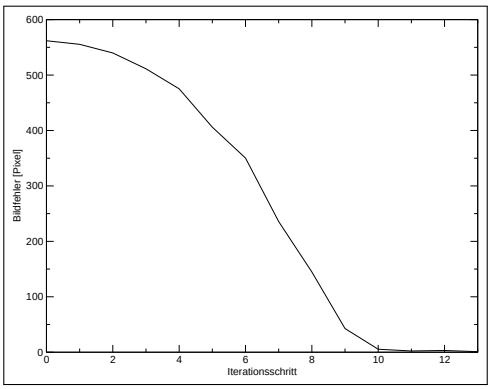


(a) Sensorspur der vierten Lage (b) Bildfehler der vierten Lage

Abbildung 4.77: Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 4

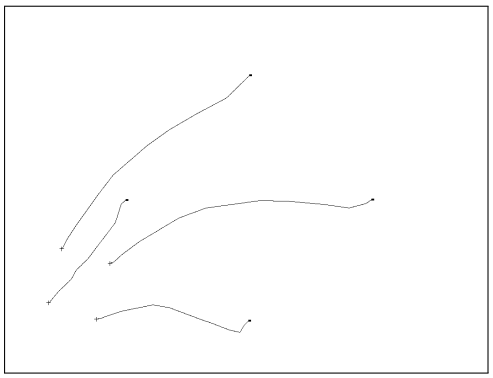


(a) Sensorspur der vierten Lage

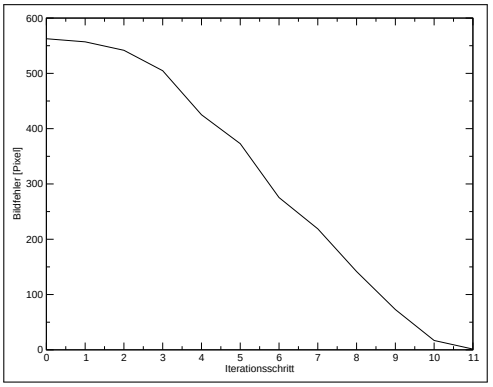


(b) Bildfehler der vierten Lage

Abbildung 4.78: Trust-Region-Regler mit MJP-Modell, Lage 4



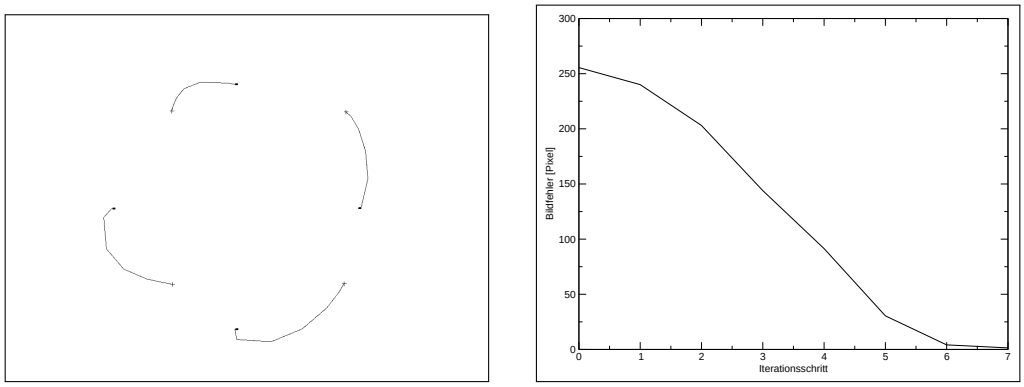
(a) Sensorspur der vierten Lage



(b) Bildfehler der vierten Lage

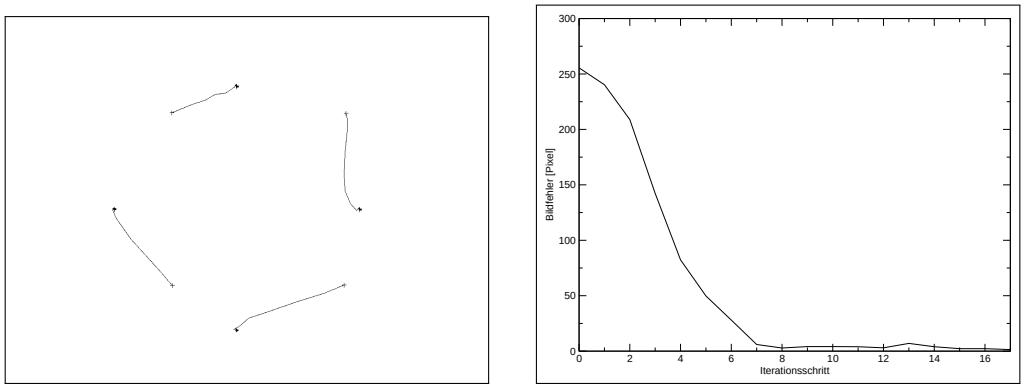
Abbildung 4.79: Trust-Region-Regler in Zylinderkoordinaten, Lage 4

Ausgangslage 5: [0, 0, 0, 0°, 0°, 45°]



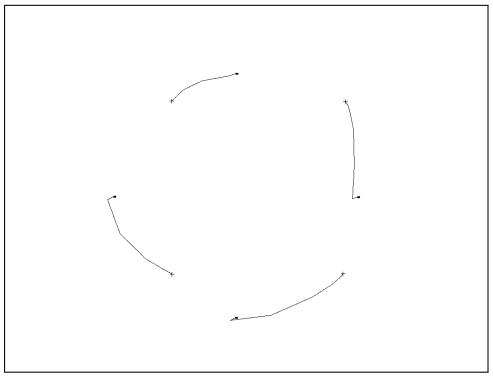
(a) Sensorspur der fünften Lage (b) Bildfehler der fünften Lage

Abbildung 4.80: Trust-Region-Regler mit konstanter Jacobimatrix, Lage 5

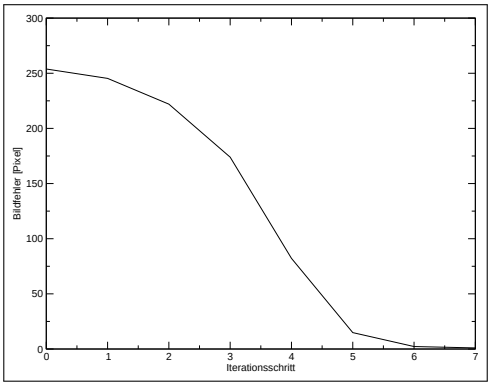


(a) Sensorspur der fünften Lage (b) Bildfehler der fünften Lage

Abbildung 4.81: Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 5

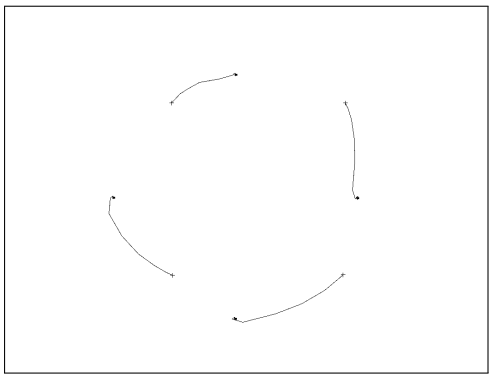


(a) Sensorspur der fünften Lage

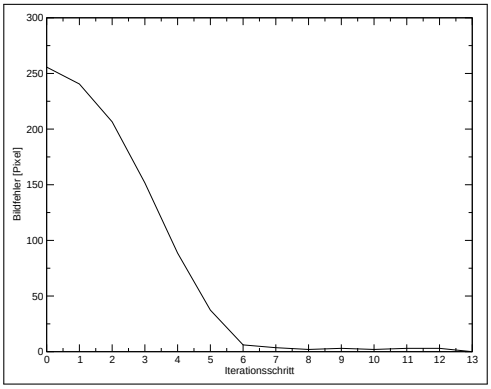


(b) Bildfehler der fünften Lage

Abbildung 4.82: Trust-Region-Regler mit MJP-Modell, Lage 5

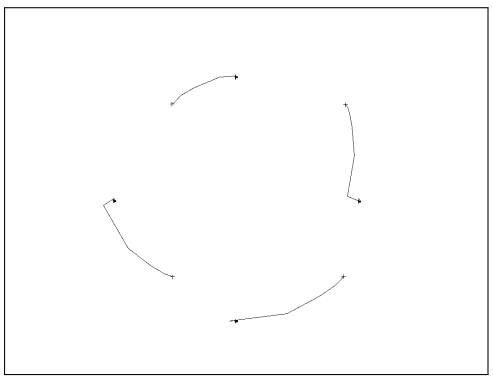


(a) Sensorspur der fünften Lage

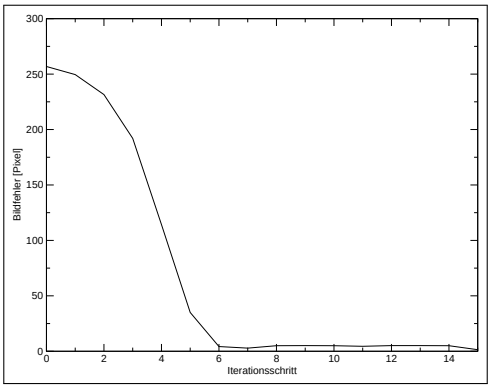


(b) Bildfehler der fünften Lage

Abbildung 4.83: Trust-Region-Regler mit PMJ-Modell, Lage 5

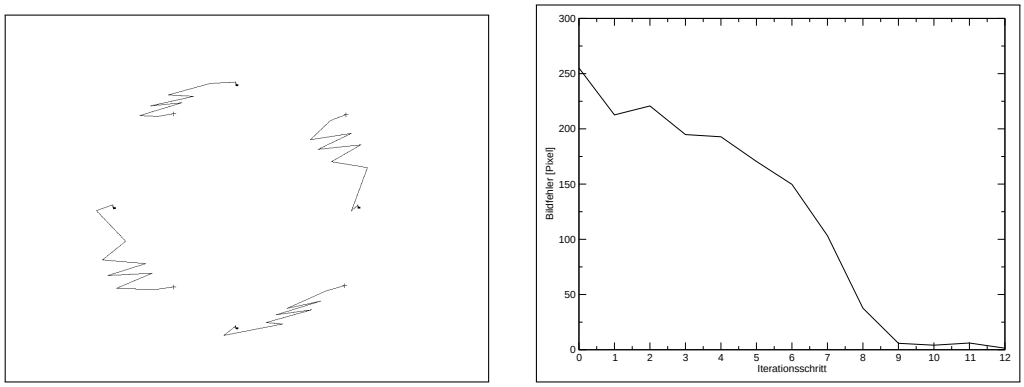


(a) Sensorspur der fünften Lage



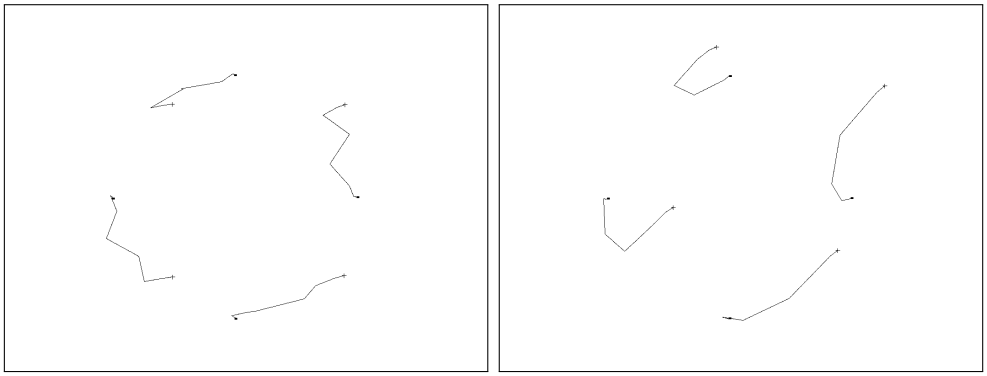
(b) Bildfehler der fünften Lage

Abbildung 4.84: Trust-Region-Regler im Zylinderkoordinatensystem, Lage 5



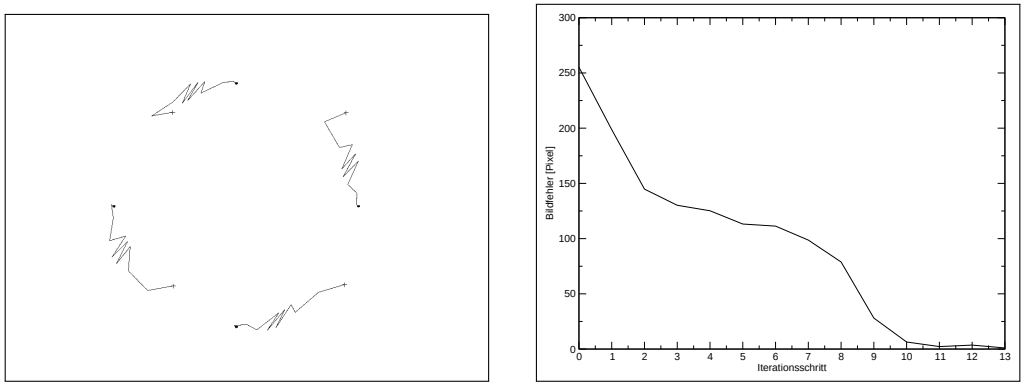
(a) Sensorspur der fünften Lage (b) Bildfehler der fünften Lage

Abbildung 4.85: Dog-Leg-Regler mit konstanter Jacobimatrix, Lage 5



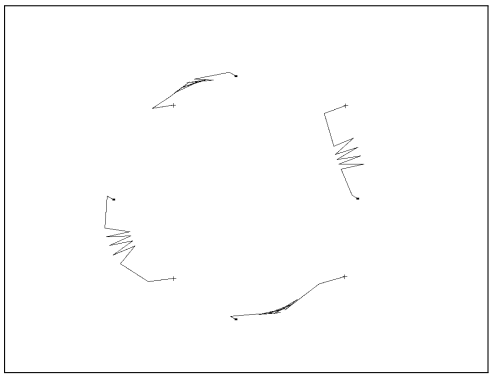
(a) Sensorspur der fünften Lage (b) Bildfehler der fünften Lage

Abbildung 4.86: Dog-Leg-Regler mit dynamischer Jacobimatrix, Lage 5

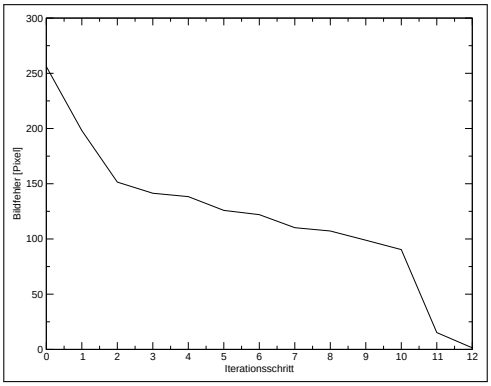


(a) Sensorspur der fünften Lage (b) Bildfehler der fünften Lage

Abbildung 4.87: Dog-Leg-Regler mit MJP-Modell, Lage 5



(a) Sensorspur der fünften Lage



(b) Bildfehler der fünften Lage

Abbildung 4.88: Dog-Leg-Regler mit PMJ-Modell, Lage 5

4.5 Bewertung der Regler

Die Ergebnisse der Experimente mit dem Trust-Region-Regler und den Dog-Leg-Regler sind in den Tabellen 4.1 und 4.2 zusammen gefasst.

4.5.1 Der Trust-Region-Regler

Mit dem Trust-Region-Regler lässt sich die Geschwindigkeit – verglichen mit den Ansätzen mit konstantem Dämpfungsfaktor – drastisch erhöhen. Zum Teil wird eine Steigerung um den Faktor 5 erreicht.

Am Anfang der Reglung werden erst kleine Bewegungen durchgeführt, so dass die Gefahr geringer ist, dass die Objektmarkierungen den Bildbereich verlassen. Je genauer die Ergebnisse des benutzten Modells während der Reglung sind, um so größer wird auch die Trust Region. Überschreitet der Modellfehler aber einen gewissen Toleranzbereich, so wird sie wieder verkleinert. Zum Schluss werden ungedämpfte Bewegungen des jeweiligen inversen Modells erlaubt.

Der Trust-Region-Regler mit konstanter Jacobimatrix konvergiert in allen fünf Lagen. Auch hier bewegen sich die Objektmarkierungen in der vierten Ausgangslage (siehe Abbildung 4.38) relativ nahe am Bildrand. In der fünften Lage (siehe Abbildung 4.44) ist der Einfluss des Retreat-Advance-Problems weiterhin zu beobachten. Bei der Multilagen-Simulaton ergibt sich eine durchschnittliche Erfolgsquote von 91.43 %, was ungefähr der des traditionellen Ansatzes entspricht. Allerdings konvergiert der Regler fast doppelt so schnell.

Mit der dynamischen Jacobimatrix lässt sich durch den Trust-Region-Regler eine noch viel größere Leistungssteigerung beobachten. Die mittlere Anzahl der benötigten Regelschritte verringert sich um den Faktor 5 auf 10 Iterationen. Auch die Erfolgsquote erhöht sich auf 99.17 %. In der vierten Ausgangslage schafft es der Regler ohne zusätzliche Anpassungen die Teachpose zu erreichen (siehe Abbildung 4.39). Die Objektmarkierungen bewegen sich nicht ganz soweit zum Bildrand wie es noch beim traditionellen Ansatz der Fall war (siehe Abbildung 3.24). Statt 81 Iterationsschritte werden nun nur noch 14 benötigt. Wie auch beim Trust-Region-Regler mit konstanter Jacobimatrix, ist auch hier das Retreat-Advance-Problem in der fünften Lage zu beobachten.

Wie schon zuvor erwähnt, besitzt der MJP-Ansatz nur ein inverses Modell, so dass hier die dynamische Jacobimatrix zu Bestimmung des Modellfehlers benutzt wird. Der so konstruierte Trust-Region-Regler liefert die besten Ergebnisse aller getesteten Regler. Seine Erfolgsquote liegt bei der Multilagen-Simulation bei 99.58 % mit durchschnittlich 7 Iterationsschritten.

Auch hier nimmt die Geschwindigkeit um den Faktor 5 zu. Der Regler liefert bei allen fünf Lagen gute Ergebnisse.

Genau wie der PMJ-Regler, schafft es auch der Trust-Region-Regler mit PMJ-Modell nicht die Ziellage der vierten Ausgangslage zu erreichen. Bei allen anderen Lagen liefert er gute Ergebnisse. Bei der Multilagen-Simulation ist der Regler bei 94.57 % aller getesteten Lagen mit durchschnittlich 13 Regelschritten erfolgreich.

Der Trust-Region-Regler in Zylinderkoordinaten konvergiert in allen fünf Lagen. Allerdings vergrößert sich auch hier wie beim Ansatz mit konstantem Dämpfungsfaktor in der zweiten Lage kurzeitig der Bildfehler (siehe Abbildung 4.26). Bei den Tesläufen auf dem realen Robotersystem wird in dieser Lage wieder eine nicht erreichbare Lage angesteuert, so dass die Reglung abgebrochen wird. Die Multilagen-Simulation liefert bei durchschnittlich 9 Iterationen eine Erfolgsquote von 93.5 %.

4.5.2 Der Dog-Leg-Regler

Der Dog-Leg-Regler versucht in den durchgeführten Experimenten zuerst mit dem Gradientenabstieg die Objektmerkmale in das Zentrum des Kamerabildes zu bringen. Dazu werden nur die Yaw- und Pitch-Winkel der Kameraorientierung verändert. Die Position sowie der Roll-Winkel der Kamera bleiben in dieser Phase unverändert.

Wird dann der eigentliche Dog-Leg-Schritt benutzt, also die Kombination aus Gradientenverfahren und dem jeweiligen benutzen inversen Modell, so sind zum Teil Zick-Zack-Bewegungen auf den Kamerasensor zu beobacheten (z. B. in Abbildung 4.28(a)). Diese Schwankungen werden stärker, falls während der Reglung häufig zwischen Gradientenverfahren und Dog-Leg-Schritt gewechselt wird, wie z. B. in Abbildung 4.42(a). Das ist dann der Fall, falls die vorhergesagte Bewegung stark von der tatsächlichen abweicht. Dadurch wird die Trust Region wieder verkleinert, eventuell wieder das Gradientenverfahren benutzt und die Objektmarkierungen erneut in Richtung des Kamerabildzentrums verschoben. Unter Umständen wird dieser Vorgang mehrmals wiederholt.

Insgesamt liefert der Regler ein eher unbefriedigendes Ergebnis. Nur die Varianten mit dem PMJ-Modell und der konstanten Jacobimatrix konvergieren in allen getesteten Lagen. Das liegt an der Tatsache, dass die beiden Modelle unempfindlicher auf große Dämpfungsfaktoren reagieren.

Regler	α_{start} / d_{soll}	OpenGL-Simulation Ausgangslage 1	2	3	4	5	Multilagen Iter.	Erfolg. [%]
Trust-Region-Regler mit konstanter Jacobimatrix	0.09 / 0.18	20	26	6	31	7	18	91.43
Trust-Region-Regler mit dynamischer Jacobimatrix	0.07 / 0.04	9	12	7	14	6	10	99.17
Trust-Region-Regler mit MJP-Modell	0.05 / 0.1	7	9	6	11	5	7	99.58
Trust-Region-Regler mit PMJ-Regler	0.07 / 0.09	20	25	6	∞	5	13	94.57
Trust-Region-Regler in Zylinderkoordinaten	0.04 / 0.1	8	18	6	11	6	9	93.5
Dog-Leg-Regler mit konstanter Jacobimatrix	0.22 / 0.16	17	25	4	21	9	10	85.05
Dog-Leg-Regler mit dynamischer Jacobimatrix	0.11 / 0.28	8	∞	6	∞	16	9	8.4
Dog-Leg-Regler mit MJP-Modell	0.3 / 0.02	∞	∞	5	∞	7	8	26.65
Dog-Leg-Regler mit PMJ-Regler	0.29 / 0.03	9	13	5	14	7	8	31.47

Tabelle 4.1: Ergebnisse der Simulatoren mit adaptiven Reglern

Regler	α_{start} / d_{soll}	Ausgangslage 1	2	3	4	5
Trust-Region-Regler mit konstanter Jacobimatrix	0.09 0.18	22	29	11	39	7
Trust-Region-Regler mit dynamischer Jacobimatrix	0.07 0.04	10	15	9	17	17
Trust-Region-Regler mit MJP-Modell	0.05 0.1	8	9	11	13	7
Trust-Region-Regler mit PMJ-Regler	0.07 0.09	21	28	7	∞	13
Trust-Region-Regler in Zylinderkoordinaten	0.04 0.1	10	∞	7	11	15
Dog-Leg-Regler mit konstanter Jacobimatrix	0.22 0.16	19	24	8	∞	12
Dog-Leg-Regler mit dynamischer Jacobimatrix	0.11 0.28	13	∞	∞	∞	13
Dog-Leg-Regler mit MJP-Modell	0.3 0.02	∞	∞	10	∞	13
Dog-Leg-Regler mit PMJ-Regler	0.29 0.03	14	13	5	∞	12

Tabelle 4.2: Ergebnisse des realen Systems mit adaptiven Reglern

Kapitel 5

Zusammenfassung

5.1 Ergebnisse

In dieser Diplomarbeit wurden verschiedene Modelle und Regler miteinander verglichen und kombiniert. Die Ergebnisse sollen im Folgenden zusammengefasst werden.

Es wurden zwei Klassen von Reglern betrachtet. Zum einen Regler mit konstantem, zum anderen Regler mit adaptivem Dämpfungsfaktor. Bis auf den Regler in Zylinderkoordinaten, lassen sich die Regler mit konstantem Dämpfungsparameter einfach anwenden. Bei dem Ansatz in Zylinderkoordinaten muss noch zusätzlich der Ursprungs des Polarkoordinatensystems mit Hilfe einer Singulärwertzerlegung bestimmt werden. Da sich die Dämpfungsfaktoren während der Reglung nicht verändern, ist die Schrittgröße am Anfang der Reglung noch recht groß. Dadurch kann es passieren, dass die Bildmerkmale das Kamerabild verlassen, wie es z. B. beim Traditionellen Regler mit dynamischer Jacobimatrix in der vierten Ausgangslage der Fall war (siehe Abbildung 3.24). Zum Schluss werden die durchgeführten Schritte immer kleiner. Dadurch kann es unter Umständen relativ lange dauern, bis die Ziellage erreicht wird. Das beste Ergebnis lieferte in dieser Klasse der MJP-Regler. Bei der Multilagen-Simulation wurden durschnittlich 37 Iterationen benötigt um die Teachpose zu erreichen. Die Erfolgsquote liegt hier bei 99.27 %.

Eine Verbesserung bieten adaptive Regler. Sie passen bestimmte Parameter der Regler zu Laufzeit an und können somit viel besser auf Änderungen der Umwelt reagieren. Der Trust-Region-Regler passt dazu mit Hilfe des Modellfehlers eine Trust Region an. Je kleiner der Modellfehler ist, desto größer wird die Trust Region gewählt. Verlässt der Modellfehler einen gewissen Toleranzbereich, so wird sie wieder verkleinert. Durch dieses Ver-

fahren werden am Anfang der Reglung kleinere Schritte ausgeführt, während zum Schluss die Schritte vergrößert werden. Dadurch wird zum einen eine schnellere Konvergenz sichergestellt und zum anderen vermieden, dass Objektmarkierungen gerade am Anfang der Reglung den Bildbereich verlassen. Durch Kombination des Trust-Region-Reglers mit den anderen Reglern mit konstantem Dämpfungsfaktor und den dazugehörigen Modellen, wurde die Leistung jedes einzelnen ernorm gesteigert. Auch hier liefert der MJP-Regler mit der dynamische Jacobimatrix als Modell das beste Ergebnis. Der so konstruierte Regler benötigt durschnittlich 7 Iterationen um seine Ziellage zu erreichen. Die Erolgsquote liegt bei 99.58 %.

Der in dieser Diplomarbeit entwickelte Dog-Leg-Regler ist ein weiterer adaptiver Regler. Auch hier wird anhand des Modellfehlers eine Trust Region angepasst. Zusätzlich wird durch eine Fallunterscheidung entweder ein ungedämpfter Gauß-Newton-Schritt, ein auf die Trust Region begrenzter Schritt des Gradientenverfahrens oder eine Kombination aus beiden durchgeführt. In den Experimenten wurde neben den Gauß-Newton-Verfahren, was dem Traditonellem Regler mit dynamischer Jacobimatrix entspricht, auch die anderen Regler mit zugehörigen Modellen eingesetzt. Allerdings lieferte dieser Ansatz keine befriedigenden Ergebnisse. Der Regler bringt am Anfang mit Hilfe des Gradientenverfahrens die Objektmarkierungen in die Mitte des Kamerabildes. Dazu werden nur die Yaw- und Pitch-Winkel der Kameraorientierung angepasst. Die Position bleibt zu diesem Zeitpunkt noch unverändert. Die Bewegungen, die mit der Kombination von Gradientenabstieg und den jeweiligen Regler erzeugt wurden, sind anfangs sehr ungleichmäßig und beschreiben oft Zick-Zack-Linien. Das ist durch den Eingfluss des Gradientenverfahrens zu erklären. Nur zwei Dog-Leg-Regler schaffen es in allen fünf Lagen die Teachpose zu erreichen. Das beste Ergebnis liefert hier der Ansatz mit der konstanten Jacobimatrix. Er erreicht 85.05 % aller Ziellagen mit einer durchschnittlichen Anzahl von 10 Iterationen.

Die besten Ergebnisse liefert in den durchgeführten Experimenten also der Trust-Region-Regler kombiniert mit dem MJP-Regler und der dynamischen Jacobimatrix als Modell. Für α_{start} wurde der Wert 0.05 und für d_{soll} 0.1 gewählt.

5.2 Ausblick

Ein weitere Möglichkeit Roboterreglungen zu entwerfen, ist die Verwendung von künstlichen neuronalen Netzen. Ein Ansatz wird in [SK06] verfolgt. Er basiert auf das *EANT-Verfahren* (*Evolutionary Acquisition of Neural Topologies*) aus [Kas06] und wurde mit der *CMA-ES* (*Covariance Matrix Adap-*

tion Evolution Strategy) [HO01] kombiniert. Durch evolutionäres Reinforcement Learning werden künstliche neuronale Netze generiert, die das Visual-Servoing-Problem in drei Freiheitsgraden lösen können. Dazu werden nicht nur die Parameter der Netze, sondern auch gleichzeitig deren Struktur optimiert.

In Zukunft sind Experimente mit sechs Freiheitsgraden geplant. Leider lagen während der Bearbeitungszeit dieser Diplomarbeit noch keine Ergebnisse vor, so dass kein Vergleich mit den hier betrachteten Reglern gemacht werden konnte.

Anhang A

Bezeichnungen und Variablen

A.1 Variablen

Variable	Beschreibung
$\mathbb{N}$	Menge der natürlichen Zahlen
$\mathbb{N}_0$	$\mathbb{N} \cup \{0\}$
$\mathbb{R}$	Menge der reellen Zahlen
$\{K\}$	Koordinatensystem K
${}^{K}O$	Ursprung des Koordinatensystems K
$m \in \mathbb{N}$	Anzahl der Bildmerkmale
$M \in \mathbb{N}$	Anzahl der Objektmarkierungen mit $m = 2 \cdot M$
$x_n \in \mathbb{R}^6$	Pose des Roboterarmes zum Zeitpunkt $n \in \mathbb{N}_0$
$x^\star \in \mathbb{R}^6$	Ziellage des Roboterarmes
$y_n \in \mathbb{R}^m$	Position der M Objektmarkierungen auf dem Kamerasensor
$y^\star \in \mathbb{R}^m$	Teachpose der Objektmarkierungen
$\Delta y_n \in \mathbb{R}^m$	Bildfehler zum Zeitpunkt $n \in \mathbb{N}_0$
$k \in \mathbb{R}$	Konstanter Dämpfungsfaktor mit $0 < k \leq 1$
$\kappa_n \in \mathbb{R}$	Dämpfungsfaktor des Gradientenverfahrens beim Dog-Leg-Regler zum Zeitpunkt $n \in \mathbb{N}_0$ mit $0 < \kappa_n \leq 1$
$\alpha_n \in \mathbb{R}$	Trust Region zum Zeitpunkt $n \in \mathbb{N}_0$
$\alpha_{\min}, \alpha_{\max} \in \mathbb{R}_{>0}$	minimaler und maximaler Trust-Region-Wert
$\alpha_{\text{start}} \in \mathbb{R}$	Startwert für die Trust Region
$d_{\text{soll}} \in \mathbb{R}_{>0}$	zulässiger Modellfehler
$d_n \in \mathbb{R}_{\geq 0}$	Tatsächlicher Modellfehler zum Zeitpunkt $n \in \mathbb{N}_0$
$r_n \in \mathbb{R}_{>0}$	Relativer Modellfehler zum Zeitpunkt $n \in \mathbb{N}_0$
$J_n \in \mathbb{R}^{m \times 6}$	Dynamische Bildjacobimatrix zum Zeitpunkt $n \in \mathbb{N}_0$
$J^\star \in \mathbb{R}^{m \times 6}$	Konstante Bildjacobimatrix
$u_n \in \mathbb{R}^6$	Stellgröße zum Zeitpunkt $n \in \mathbb{N}_0$

Variable	Beschreibung
$u_{\mathrm{sd}_n} \in \mathbb{R}^6$	Stellgröße, berechnet durch das Gradientenverfahren zum Zeitpunkt $n \in \mathbb{N}_0$
$u_{\mathrm{gn}_n} \in \mathbb{R}^6$	Stellgröße, berechnet durch das Gauß-Newton-Verfahren zum Zeitpunkt $n \in \mathbb{N}_0$
$u_{\mathrm{dl}_n} \in \mathbb{R}^6$	Stellgröße, berechnet durch das Dog-Leg-Verfahren zum Zeitpunkt $n \in \mathbb{N}_0$

A.2 Abkürzungen

Abkürzung	Beschreibung
EANT	Evolutionary Acquisition of Neural Topologies
CMA-ES	Covariance Matrix Adaption Evolution Strategy
PMJ	Pseudo-inverse of the Mean of the Jacobians
MJP	Mean of the Jacobian Pseudo-inverse

Anhang B

Visual-Servoing-Anwendungen

Auf der beigelegten CD-ROM befindet sich der Quellcode zur Visual-Servoing-Anwendung sowie zu den beiden Simulationen.

Abbildungsverzeichnis

1.1 CRS F3 3
1.2 Objektmarkierungen 4
1.3 Aufnahme der Teachposition 4
1.4 VisualServoing Simulator 5
1.5 Benutzte Testlagen bei den Experimenten 7

2.1 Koordinatensysteme (Grafik aus [Sie99]) 10
2.2 Yaw-, Pitch- und Roll-Winkel (Grafik aus [Sie99]) 10
2.3 Zylinderkoordinatensystem 11
2.4 Lochkameramodell (Grafik aus [Sie99]) 12

3.1 Reine Rotation um z-Achse 21
3.2 Gemometrische Interpretation des Retreat-Advance-Problems (Grafik nach [CH06]) 23
3.3 Multilagen-Simulation mit konstanter Jacobimatrix 27
3.4 Multilagen-Simulation mit dynamischer Jacobimatrix 27
3.5 Multilagen-Simulation mit dem MJP-Regler 28
3.6 Multilagen-Simulation mit dem PMJ-Regler 29
3.7 Multilagen-Simulation mit dem Regler im Zylinderkoordinatensystem 29
3.8 Traditioneller Regler mit konstanter Jacobimatrix, Lage 1 . . . 30
3.9 Traditioneller Regler mit dynamischer Jacobimatrix, Lage 1 . 31
3.10 MJP-Regler, Lage 1 32
3.11 PMJ-Regler, Lage 1 33
3.12 Regler in Zylinderkoordinaten, Lage 1 34
3.13 Traditioneller Regler mit konstanter Jacobimatrix, Lage 2 . . . 35
3.14 Traditioneller Regler mit dynamischer Jacobimatrix, Lage 2 . 36
3.15 MJP-Regler, Lage 2 37
3.16 PMJ-Regler, Lage 2 38
3.17 Regler in Zylinderkoordinaten, Lage2 39
3.18 Traditioneller Regler mit konstanter Jacobimatrix, Lage 3 . . . 40

3.19 Traditioneller Regler mit dynamischer Jacobimatrix, Lage 3 . 41
3.20 MJP-Regler, Lage 3 42
3.21 PMJ-Regler, Lage 3 43
3.22 Regler in Zylinderkoordinaten, Lage3 44
3.23 Traditioneller Regler mit konstanter Jacobimatrix, Lage 4 . . . 45
3.24 Traditioneller Regler mit dynamischer Jacobimatrix, Lage 4 . 46
3.25 MJP-Regler, Lage 4 47
3.26 Regler in Zylinderkoordinaten, Lage 4 48
3.27 Traditioneller Regler mit konstanter Jacobimatrix, Lage 5 . . . 49
3.28 Traditioneller Regler mit dynamischer Jacobimatrix, Lage 5 . 50
3.29 MJP-Regler, Lage 5 51
3.30 PMJ-Regler, Lage 5 52
3.31 Regler in Zylinderkoordinaten, Lage5 53
3.32 Traditioneller Regler mit konstanter Jacobimatrix, Lage 1 . . . 54
3.33 Traditioneller Regler mit dynamischer Jacobimatrix, Lage 1 . 55
3.34 MJP-Regler, Lage 1 55
3.35 PMJ-Regler, Lage 1 55
3.36 Regler in Zylinderkoordinaten, Lage 1 56
3.37 Traditioneller Regler mit konstanter Jacobimatrix, Lage 2 . . . 57
3.38 Traditioneller Regler mit dynamischer Jacobimatrix, Lage 2 . 57
3.39 MJP-Regler, Lage 2 58
3.40 PMJ-Regler, Lage 2 58
3.41 Traditioneller Regler mit konstanter Jacobimatrix, Lage 3 . . . 59
3.42 Traditioneller Regler mit dynamischer Jacobimatrix, Lage 3 . 59
3.43 MJP-Regler, Lage 3 60
3.44 PMJ-Regler, Lage 3 60
3.45 Regler in Zylinderkoordinaten, Lage 3 60
3.46 Traditioneller Regler mit konstanter Jacobimatrix, Lage 4 . . . 61
3.47 Traditioneller Regler mit dynamischer Jacobimatrix, Lage 4 . 61
3.48 MJP-Regler, Lage 4 62
3.49 Regler in Zylinderkoordinaten, Lage 4 62
3.50 Traditioneller Regler mit konstanter Jacobimatrix, Lage 5 . . . 63
3.51 Traditioneller Regler mit dynamischer Jacobimatrix, Lage 5 . 63
3.52 MJP-Regler, Lage 5 64
3.53 PMJ-Regler, Lage 5 64
3.54 Regler in Zylinderkoordinaten, Lage 5 64

4.1 Beschreibung des Trust-Region-Reglers (Grafik aus [Sie99]) . . 70
4.2 Ablaufplan einer Regelung mit dem Trust-Region-Regler . . . 73
4.3 Bestimmung des Dog-Leg-Schrittes (Grafik nach [MNT99]) . . 75
4.4 Multilagen-Simulation mit konstanter Jacobimatrix 79

4.5 Multilagen-Simulation mit dynamischer Jacobimatrix 80
4.6 Multilagen-Simulation mit MJP-Modell 81
4.7 Multilagen-Simulation mit PMJ-Modell 82
4.8 Multilagen-Simulation in Zylinderkoordinaten 83
4.9 Multilagen-Simulation mit konstanter Jacobimatrix 84
4.10 Multilagen-Simulation mit dynamischer Jacobimatrix 86
4.11 Multilagen-Simulation mit MJP-Modell 87
4.12 Multilagen-Simulation mit PMJ-Modell 88
4.13 Multilagen-Simulation in Zylinderkoordinaten 89
4.14 Trust-Region-Regler mit konstanter Jacobimatrix, Lage 1 . . . 90
4.15 Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 1 . . 91
4.16 Trust-Region-Regler mit MJP-Modell, Lage 1 92
4.17 Trust-Region-Regler mit PMJ-Modell, Lage 1 93
4.18 Trust-Region-Regler in Zylinderkoordinaten, Lage 1 94
4.19 DogLeg-Regler mit konstanter Jacobimatrix, Lage 1 95
4.20 DogLeg-Regler mit dynamischer Jacobimatrix, Lage 1 96
4.21 DogLeg-Regler mit PMJ-Modell, Lage 1 97
4.22 Trust-Region-Regler mit konstanter Jacobimatrix, Lage 2 . . . 98
4.23 Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 2 . . 99
4.24 Trust-Region-Regler mit MJP-Modell, Lage 2 100
4.25 Trust-Region-Regler mit PMJ-Modell, Lage 2 101
4.26 Trust-Region-Regler in Zylinderkoordinaten, Lage 2 102
4.27 DogLeg-Regler mit konstanter Jacobimatrix, Lage 2 103
4.28 DogLeg-Regler mit PMJ-Modell, Lage 2 104
4.29 Trust-Region-Regler mit konstanter Jacobimatrix, Lage 3 . . . 105
4.30 Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 3 . . 106
4.31 Trust-Region-Regler mit MJP-Modell, Lage 3 107
4.32 Trust-Region-Regler mit PMJ-Modell, Lage 3 108
4.33 Trust-Region-Regler in Zylinderkoordinaten, Lage 3 109
4.34 DogLeg-Regler mit konstanter Jacobimatrix, Lage 3 110
4.35 DogLeg-Regler mit dynamischer Jacobimatrix, Lage 3 111
4.36 DogLeg-Regler mit MJP-Modell, Lage 3 112
4.37 DogLeg-Regler mit PMJ-Modell, Lage 3 113
4.38 Trust-Region-Regler mit konstanter Jacobimatrix, Lage 4 . . . 114
4.39 Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 4 . . 115
4.40 Trust-Region-Regler mit MJP-Modell, Lage 4 116
4.41 Trust-Region-Regler in Zylinderkoordinaten, Lage 4 117
4.42 DogLeg-Regler mit konstanter Jacobimatrix, Lage 4 118
4.43 DogLeg-Regler mit PMJ-Modell, Lage 4 119
4.44 Trust-Region-Regler mit konstanter Jacobimatrix, Lage 5 . . . 120
4.45 Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 5 . . 121

4.46 Trust-Region-Regler mit MJP-Modell, Lage 5 122
4.47 Trust-Region-Regler mit PMJ-Modell, Lage 5 123
4.48 Trust-Region-Regler in Zylinderkoordinaten, Lage 5 124
4.49 DogLeg-Regler mit konstanter Jacobimatrix, Lage 5 125
4.50 DogLeg-Regler mit dynamischer Jacobimatrix, Lage 5 126
4.51 DogLeg-Regler mit MJP-Modell, Lage 5 127
4.52 DogLeg-Regler mit PMJ-Modell, Lage 5 128
4.53 Trust-Region-Regler mit konstanter Jacobimatrix, Lage 1 . . . 129
4.54 Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 1 . . 130
4.55 Trust-Region-Regler mit MJP-Modell, Lage 1 130
4.56 Trust-Region-Regler mit PMJ-Modell, Lage 1 130
4.57 Trust-Region-Regler in Zylinderkoordinaten, Lage 1 131
4.58 Dog-Leg-Regler mit konstanter Jacobimatrix, Lage 1 131
4.59 Dog-Leg-Regler mit dynamischer Jacobimatrix, Lage 1 131
4.60 Dog-Leg-Regler mit PMJ-Modell, Lage 1 132
4.61 Trust-Region-Regler mit konstanter Jacobimatrix, Lage 2 . . . 133
4.62 Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 2 . . 133
4.63 Trust-Region-Regler mit MJP-Modell, Lage 2 134
4.64 Trust-Region-Regler mit PMJ-Modell, Lage 2 134
4.65 Dog-Leg-Regler mit konstanter Jacobimatrix, Lage 2 134
4.66 Dog-Leg-Regler mit PMJ-Modell, Lage 2 135
4.67 Trust-Region-Regler mit konstanter Jacobimatrix, Lage 3 . . . 136
4.68 Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 3 . . 136
4.69 Trust-Region-Regler mit MJP-Modell, Lage 3 137
4.70 Trust-Region-Regler mit PMJ-Modell, Lage 3 137
4.71 Trust-Region-Regler in Zylinderkoordinaten, Lage 3 137
4.72 Dog-Leg-Regler mit konstanter Jacobimatrix, Lage 3 138
4.73 Dog-Leg-Regler mit dynamischer Jacobimatrix, Lage 3 138
4.74 Dog-Leg-Regler mit MJP-Modell, Lage 3 138
4.75 Dog-Leg-Regler mit PMJ-Modell, Lage 3 139
4.76 Trust-Region-Regler mit konstanter Jacobimatrix, Lage 4 . . . 140
4.77 Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 4 . . 140
4.78 Trust-Region-Regler mit MJP-Modell, Lage 4 141
4.79 Trust-Region-Regler in Zylinderkoordinaten, Lage 4 141
4.80 Trust-Region-Regler mit konstanter Jacobimatrix, Lage 5 . . . 142
4.81 Trust-Region-Regler mit dynamischer Jacobimatrix, Lage 5 . . 142
4.82 Trust-Region-Regler mit MJP-Modell, Lage 5 143
4.83 Trust-Region-Regler mit PMJ-Modell, Lage 5 143
4.84 Trust-Region-Regler im Zylinderkoordinatensystem, Lage 5 . . 143
4.85 Dog-Leg-Regler mit konstanter Jacobimatrix, Lage 5 144
4.86 Dog-Leg-Regler mit dynamischer Jacobimatrix, Lage 5 144

4.87 Dog-Leg-Regler mit MJP-Modell, Lage 5 144
4.88 Dog-Leg-Regler mit PMJ-Modell, Lage 5 145

Tabellenverzeichnis

3.1 Ergebnisse der Simulatoren mit konstantem Dämpfungsfaktor 66
3.2 Ergebnisse des realen Systems mit konstantem Dämpfungsfaktor 66

4.1 Ergebnisse der Simulatoren mit adaptiven Reglern 148
4.2 Ergebnisse des realen Systems mit adaptiven Reglern 149

Literaturverzeichnis

[CH01] Peter I. Corke and Seth A. Hutchinson. A new partioned approach to image-based visual servo control. *IEEE Transactions on Robotics and Automation*, 237(4):507–515, August 2001.

[CH06] F. Chaumette and S. Hutchinson. Visual servo control, part i: Basic approaches. *IEEE Robotics and Automation Magazine*, 13(4):82–90, December 2006.

[Cha98] Potential problems of stability and convergence in image-based and position-based visual servoing. In D. Kriegmann, G. Hager, and S. Morse, editors, *The Confluence of Vision and Control*, volume 237, pages 66–78, New York, 1998. Springer-Verlag.

[DH91] Peter Deuflhard and Andreas Hohmann. *Numerische Mathematik: Eine algorithmisch orientierte Einführung.* de Gruyter, Berlin, 1st edition, 1991.

[Fle87] Roger Fletcher. *Practical Methods of Optimization.* John Wiley & Sons, New York, Chichester, 2nd edition, 1987.

[HHC96] S. A. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12(5):651–670, October 1996.

[HO01] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9:159–195, 2001.

[IO05] Masami Iwatsuki and Norimitsu Okiyama. A new formulation of visual servoing based on cylindrical coordinate system. *Robotics, IEEE Transactions on Robotics*, 21(2):266–273, April 2005.

[Kan96] K. Kanatani. *Statistical Optimization for Geometric Computation: Theory and Practice.* Elsevier Science, Amsterdam, The Netherlands, 1996.

[Kas06] Yohannes Kassahun. *Towards a Unified Approach to Learning and Adaptation*. PhD thesis, Cognitive Systems Group, Institute of Computer Science, Christian-Albrechts-University of Kiel, Germany, February 2006.

[Knu97] Donald E. Knuth. *The Art of Computer Programming*, volume 2. Addison-Wesley, 3rd edition, 1997.

[Mal04] Ezio Malis. Improving vision-based control using efficient second-order minimization techniques. *IEEE Transactions on Robotics and Automation*, pages 1843–1848, April 2004.

[MNT99] K. Madsen, H. B. Nielsen, and O. Tingleff. Methods for non-linear least squares problems, 1999.

[NW99] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer Verlag, New York, 1st edition, 1999.

[Pow70] Michael J. D. Powell. A hybrid method for non-linear equations. In Philip Rabinowitz, editor, *Numerical Methods for Non-Linear Algebraic Equations*, pages 87–114, London, 1970. Gordon and Breach.

[Sie99] Nils T Siebel. Bildbasierte Roboterregelung in sechs Freiheitsgraden unter Verwendung einer Trust-Region-Methode. Diplomarbeit, Zentrum für Technomathematik und Institut für Automatisierungstechnik, Universität Bremen, Bremen, August 1999.

[SK06] Nils T Siebel and Yohannes Kassahun. Learning neural networks for visual servoing using evolutionary methods. In *Proceedings of the 6th International Conference on Hybrid Intelligent Systems (HIS'06), Auckland, New Zealand*, page 6 (4 pages), Los Alamitos, USA, December 2006. IEEE Computer Society.