Kapitel 9

Lernen von Steuerungs-Parametern

In diesem Kapitel zeige ich, wie die Parameter des Roboters so gelernt werden können, dass der Roboter in eine bestimmte Richtung kontrollierter und schneller beschleunigen, fahren und bremsen kann [Gloye, 2004]. Im zweiten Teil zeige ich, wie der Roboter lernt, an einem vorgegebenen Punkt zu halten. Die Optimierungen können schnell vorgenommen werden, wenn ein neuer Roboter gebaut wird und dieser auch so gut wie möglich fahren soll.

Auf den Robotern befindet sich ein Regelkreis, der versucht eine vorgegebene Geschwindigkeit möglichst schnell zu erreichen und auch zu halten, sei es nun eine Richtungsgeschwindigkeit oder eine Drehgeschwindigkeit. Dazu werden drei PID-Controller verwendet. Je einer für jeden Freiheitsgrad des Roboters.

Ein PID-Controller verfügt über mehrere Parameter, die alle optimal justiert werden müssen. Je komplizierter der zu regelnde Prozess ist, desto schwerer ist es auch die optimalen Parameter zu finden. Es gibt zwar Vorgehensweisen in der Regelungstechnik, wie die Parameter gefunden werden können, aber diese Werte sind dann auch nur für einen Zielwert optimal. In unserem Fall nur für eine Geschwindigkeit.

Eine Einführung in das Thema der PID-Regelung ist in Abschnitt 4.10.2 zu finden.

Das Verhalten „Fahren zum Ort“ ist ein elementares Verhalten in der Verhaltenssteuerung der FU-Fighters. Das Verhalten berechnet anhand von gewünschter Maximalgeschwindigkeit, Zielposition, Zielorientierung, Zielgeschwindigkeit, momentaner Position, momentaner Geschwindigkeit und momentaner Orientierung den gewünschten Geschwindigkeitsvektor für den Roboter.

Ich gehe zuerst auf den PID-Controller des Roboters ein, bevor ich im zweiten

Teil auf das höhere Verhalten eingehe.

9.1 Vorarbeiten

PID-Controller werden schon seit vielen Jahrzehnten eingesetzt [Callender, 1939]. Es wurde auch viel über die optimalen Einstellungen von PID-Parametern geschrieben, ob im Internet[1] oder in Zeitschriften [Aeström, 1992, Aeström, 1995].

Ein bekanntes Verfahren ist die Ziegler-Nichols-Methode [Ziegler, 1942]. Bei diesem Algorithmus wird zuerst der P-Term des Reglers variiert bis der Regler anfängt zu oszillieren, die I- und D-Parameter werden auf 0 gesetzt. Anhand der Periodendauer der Oszillation bei überschreiten eines Schwellwertes des P-Parameters werden alle Parameter auf bestimmte Werte gesetzt, die heuristisch ermittelt wurden.

Analytische Methoden benötigen auch ein Modell, welches analysiert werden kann. Gerade bei komplexen Systemen fehlt aber gerade ein hinreichendes Modell, deshalb wird auch versucht, genetische Algorithmen zum Finden optimaler Parameter anzuwenden [Koza, 2003].

In [Riedmiller, 1998] wurde ein PID-Regler für ein Saugrohrdrucksystem, in dem die Stellung einer Drosselklappe gesteuert wurde, durch ein neuronales Netz optimiert, ohne dass ein spezieller Regler verwendet wurde. Die besten Ergebnisse wurden erzielt, indem dem vorhandenen Regler ein neuronaler Regler vorgeschaltet wurde. Für viele industrielle Anwendungen werden auch PID-Controller benötigt, die im Laufe der Zeit selbst ihre Parameter anpassen [Tan, 1999]. Der in diesem Kapitel vorgestellte RL-Ansatz ist dafür gut geeignet.

Das Team der University of Texas hat für die vierbeinigen AIBOs die „policy gradient reinforcement“ Methode benutzt, die auch in dieser Arbeit Verwendung gefunden hat [Kohl, 2003]. Die Aufgabe war es die Geschwindigkeit des vorwärtsgerichteten Laufens des Roboters zu optimieren. Das Laufen besteht aus einer festprogrammierten Bewegungsabfolge, die durch zehn Konstanten parametrisiert ist. Diese Parameter wurden optimiert. Unser Ziel ist es dagegen, nicht nur die Geschwindigkeit der Roboter zu optimieren, sondern auch die Genauigkeit des Fahrens. Und das nicht nur nach vorne, sondern in alle Richtungen.

[1] Zum Beispiel www.chem.mtu.edu/tbco/cm416/cm416.html

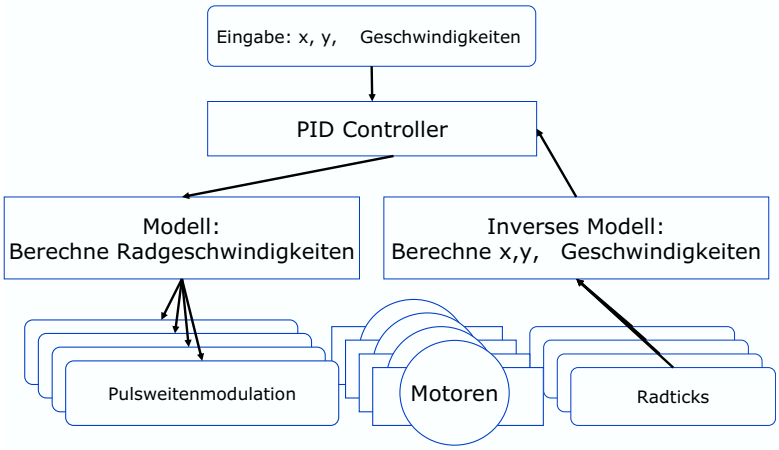

Abbildung 9.1: PID-Regelkreis auf dem Roboter

9.2 Optimales Regeln

Optimal im Sinne der Regelungstechnik bedeutet, dass ein Zielwert schnell erreicht und über die Zeit gehalten wird. Dabei soll der Regler nicht stark übersteuern und auch bei externen Störfaktoren den Zielwert erreichen.

Es stellt sich die Frage, was gesteuert werden soll. Der allgemeine Anwendungsfall von PID-Controllern ist eine direkte Regelung einer Komponente, zum Beispiel einer Drosselklappe, eines Ventils oder eines Elektromotors. Bei der Motorsteuerung wird der Motor in der Regel über ein PWM-Signal gesteuert.

Die FU-Fighters regeln ihre Roboter seit der zweiten Generation der dreirädrigen Roboter anders. Sie benutzen drei PID-Controller. Einen für die Vorwärtsrichtung, einen für die Seitwärtsrichtung und einen für die Drehung des Roboters. Der Roboter bekommt diese Sollwerte über Funk vom Zentralrecher übermittelt. Die drei Werte werden anhand eines einfachen physikalischen Fahrmodells auf dem Roboter in Motorgeschwindigkeiten umgerechnet. Die Motoren bekommen diese Werte als PWM-Signale. Die Umdrehungen der Motoren werden gemessen und mit Hilfe des inversen Modells zurück in Bewegungen des Roboters berechnet. Diese Bewegung bekommen dann die PID-Regler als Sensorwerte des Systems, sodass der Sollwert eingeregelt werden kann. Einen Überblick über den Regelkreis gibt Abbildung 9.1.

9.3 Parameter lernen

Zum Lernen der Parameter wird die „policy gradient reinforcement learning“ Methode benutzt [Kohl, 2003]. Dabei werden die Parameter angepasst, die die Qualität des Gesamtsystems unabhängig von dem Einfluss anderer Parameter verbessern. Es wird also eine lokale Unabhängigkeit der Parameter untereinander angenommen. Zu diesem Zweck werden alle Parameter zufällig variiert und die partiellen Ableitungen der einzelnen Parameter nach einigen Iterationen geschätzt und die Parameter korrigiert.

Ein Parametersatz P besteht aus 18 Komponenten $(p_1, \ldots, p_{18})$. Die Anzahl der Parameter ist unabhängig von der Anzahl der Räder, denn nur die Freiheitsgrade des Roboters werden jeweils durch einen PID-Regler geregelt. Jeweils sechs der 18 Werte bestimmen den PID-Regeler für einen Freiheitsgrad des Roboters. Die Komponenten sind die Gewichte für den proportionalen, integralen und differenziellen Anteil sowie Werte für Beschleunigungsbegrenzung, einen Skalierungsfaktor und einen Offset. Näheres dazu in Abschnitt 4.10.2.

Zur Initialisierung können die Parameter auf 0 gesetzt werden oder die handoptimierten Parameterwerte benutzt werden. In jeder Iterationsstufe werden n variierte Parametersätze

$$
\begin{aligned}
P^1 &= (p_1 + \pi_1^1, \ldots, p_{18} + \pi_{18}^1) && (9.1)\\
P^2 &= (p_1 + \pi_1^2, \ldots, p_{18} + \pi_{18}^2) && (9.2)\\
&\vdots && (9.3)\\
P^n &= (p_1 + \pi_1^n, \ldots, p_{18} + \pi_{18}^n) && (9.4)
\end{aligned}
$$

zufällig generiert, wobei die Variationen π_i^j zufällig gleichverteilt aus der Menge $\{-\epsilon_i, 0, +\epsilon_i\}$ gewählt werden. Dabei ist ϵ_i ein kleiner Wert, der für jeden Parameter p_i festgelegt wird.

Der Test um einen Parametersatz zu evaluieren ist recht einfach. Der Roboter startet in der Mitte des Spielfelds. Er hat die Aufgabe möglichst schnell und genau in einer Richtung 45^o, 135^o, 225^o oder 315^o relativ zu seiner eigenen Ausgangsorientierung zu fahren und dabei seine Orientierung beizubehalten (siehe Abbildung 9.2). Nach dieser Zeit soll er möglichst schnell zum Stehen kommen. Dabei findet keine Interaktion mit dem externen Computer statt. Das heißt, der Roboter muss Korrekturen der Richtung und Orientierung durch Messung der Impulsgeber der Motoren korrigieren. Ein Durchdrehen der Räder würde die Berechnung verfälschen. Ein zu langsames Beschleunigen würde die Distanz, die der Roboter zurücklegt, verkürzen. Die Aufgabe ist es also einen guten Kompromiss für eine optimale Regelung zu finden. Eine spezielle Aufgabe zum Lernen des Reglers für die Rotation des Roboters gibt es nicht, denn es hat sich gezeigt, dass die notwendige Korrektur der Orientierung beim Beschleunigen des Roboters ausreicht, um auch die Parameter dieses Reglers zu lernen.

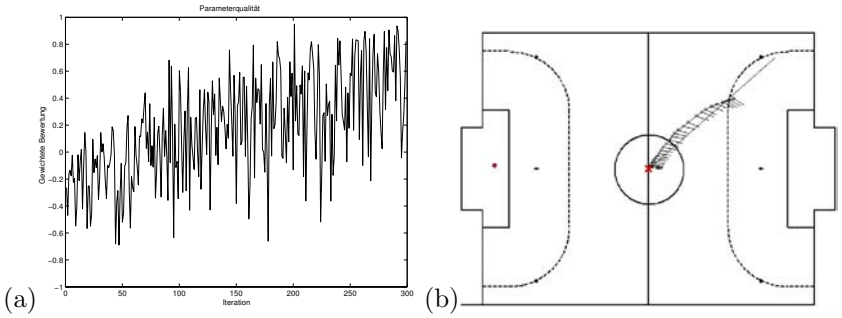

Abbildung 9.2: (a) Die berechneten Qualitäten der erzeugten Parametersätze bei der Anwendung des Lernverfahrens. Wie deutlich zu sehen ist, ist die Qualität mit sehr viel Rauschen behaftet. Trotzdem verbessert sich die Qualität über die Zeit. (b) Beispiel eines Testlaufs. Die durchgezogene Linie zeigt die vorgegebene Richtung, die gepunktete Linie mit den kleinen Strichen den Pfad des Roboters und dessen Orientierung, wie ihn das externe System sieht und welcher zur Bewertung der durchgeführten Aufgabe benutzt wird.

Die Qualität $Q(P^j)$ für jeden Parametersatz P^j ist eine gewichtete Summe folgender Merkmale:

1. Die Abweichung der Roboterposition von dem vorgegebenen Pfad.
2. Die Summe der Abweichungen der Orientierungen von der vorgegebenen Orientierung.
3. Die Distanz, die der Roboter zurückgelegt hat.
4. Die Distanz, die der Roboter zum Stoppen benötigt hat.

Bis auf Punkt 3 gehen alle Kriterien negativ in die Bewertungsfunktion ein.

Um die Auswirkung der Variation eines Parameters i auf den gesamten Parametersatz zu erhalten, werden die n Parametersätze P^j in drei Klassen unterteilt:

$$C_i^+ = \{P^j | \pi_i^j = +\epsilon_i, j = 1, \ldots, n\}, \tag{9.5}$$
$$C_i^- = \{P^j | \pi_i^j = -\epsilon_i, j = 1, \ldots, n\}, \tag{9.6}$$
$$C_i^0 = \{P^j | \pi_i^j = 0, j = 1, \ldots, n\}. \tag{9.7}$$
$$\tag{9.8}$$

Die Parametersätze werden also in die Klassen eingeteilt, in denen der betreffende Parameter positiv, negativ oder überhaupt nicht verändert wurde. Als nächstes wird festgestellt, ob dieser Parameter einen messbaren Einfluss auf die

Qualität der Parametersätze hatte. Es wird die Qualität dieses variierenden Parameters berechnet:

$$A_i^+ = \frac{\sum_{P \in Q_i^+} Q(P)}{||C_i^+||}, \tag{9.9}$$

$$A_i^- = \frac{\sum_{P \in Q_i^-} Q(P)}{||C_i^-||}, \tag{9.10}$$

$$A_i^0 = \frac{\sum_{P \in Q_i^0} Q(P)}{||C_i^0||} \tag{9.11}$$

Diese Qualitäten liefern den Gradienten des i-ten Parameters. Ist der Gradient eindeutig, dann ändern wir den Parameterwert p_i nach folgender Vorschrift:

$$p_i' = \begin{cases} p_i + \eta_i & \text{if } A_i^+ > A_i^0 > A_i^- \text{ or } A_i^+ > A_i^- > A_i^0 \\ p_i - \eta_i & \text{if } A_i^- > A_i^0 > A_i^+ \text{ or } A_i^- > A_i^+ > A_i^0 \\ p_i & \text{sonst} \end{cases} \tag{9.12}$$

wobei η_i die Lernkonstante für den Parameter i ist.

Die Testläufe werden so lange wiederholt, bis keine merkliche Verbesserung der Qualität mehr erreicht wird.

Der Roboter kann nun exakt fahren und schnell beschleunigen bzw. bremsen. Er soll aber auch exakt anhalten können, damit er, zum Beispiel, einen Ball abfangen kann. Eine Möglichkeit diese Aufgabe zu lösen ist es, den Abstand zu messen, den der Roboter zum Stoppen benötigt und bei entsprechendem Abstand den Haltebefehl zum Roboter zu senden. Durch Ungenauigkeiten beim Bremsen bleibt der Roboter jedoch nicht genau an der vorgegebenen Position stehen. Es muss also eine weitere Korrektur durchgeführt werden, am besten schon während des Bremsens. Daher gibt es ein Verhalten, welches das Fahren zu einem bestimmten Ort kontrolliert. Wie dieses Verhalten das exakte und gleichzeitig schnelle Anhalten an einem Punkt lernt wird im nächsten Abschnitt vorgestellt.

9.4 Bremskurve

Das Hauptziel des Verhaltens „Fahren zum Ort“ ist es, in möglichst kurzer Zeit mit einer akzeptablen Positionstoleranz und mit einer bestimmten Ausrichtung an der Zielposition anzukommen. Die Hardware des Roboters ist jedoch Beschränkungen unterworfen. So kann er nur mit einer gewissen Trägheit seine Richtung ändern und er darf nur mit einer begrenzten Kraft bremsen, um nicht umzukippen.

Das Verhalten „Fahren zum Ort“ hat deshalb eine Funktion, die die maximale Geschwindigkeit bezüglich des Abstands zur Zielposition angibt, die sogenannte

Bremsfunktion. Eine Beschleunigungsfunktion ist nicht notwendig, denn erstens bewahrt der Regelkreis auf dem Roboter den Roboter vor dem Umfallen und zweitens gibt es, im Gegensatz zum Bremsen, beim Beschleunigen keinen Punkt, an dem die maximale Geschwindigkeit erreicht sein soll.

In Abbildung 9.3 sind unterschiedliche Bremskurven zu sehen. Die schwarze Kurve zeigt die manuell eingestellte Bremskurve, die früher für alle Roboter gleichermaßen galt. Die ungewöhnliche Form ist das Ergebnis dieser manuellen Einstellung, die aus zwei Beobachtungen hervorging: Wenn die Bremskurve ab 70cm linear von 1 auf 0 abfallen würde, dann wären die Roboter in der Nähe der Zielposition sehr langsam. Sehr kurze Positionsänderungen würden ungeeignet langsam durchgeführt werden. Wenn andererseits die Bremskurve ab 40cm linear von 1 auf 0 abfallen würde, dann würde der Roboter sehr oft über das Ziel hinausschießen. Deshalb entstand eine Kombination der beiden linearen Funktionen.

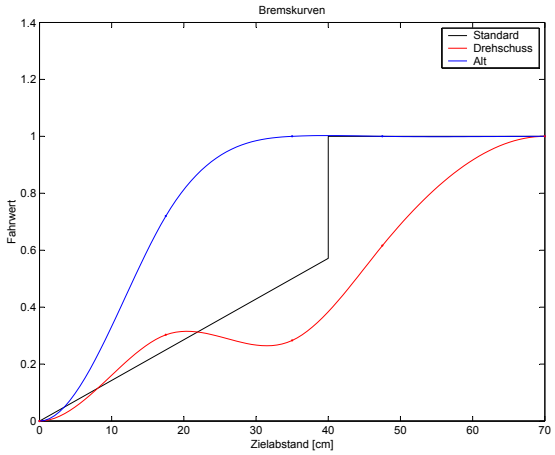

Abbildung 9.3: Unterschiedliche Bremskurven. Die schwarze Kurve zeigt die Standardbremskurve für alle Roboter vor der Optimierung. Die rote Kurve wurde für einen dreirädrigen Roboter von 2002 gelernt. Die blaue Kurve ist für den gleichen Roboter, jedoch mit einem anderen Regelkreis.

9.5 Lernmethode

Die Form der Bremskurve soll nun optimiert, also gelernt werden. Dazu werden in dem Abstandsbereich zwischen 70cm und 0cm kubische Splines [Press, 1992] mit 5 gleichabständigen Stützstellen gelegt. Der Wert bei 70cm wird auf 1 gesetzt, der Wert bei 0cm wird auf 0 festgelegt. Die Ableitungen an den Endpunkten werden auf 0 gesetzt. Es können also die drei Stützstellen bei 17,5cm,

35cm und 52,5cm variiert werden. Später können die Splines stückweise linear approximiert werden, damit der Rechenaufwand geringer wird. Kubische Splines wurden verwendet, da sie schnell zu berechnen sind und die Geschwindigkeitsänderungen stetig sind. Die Bremskurven sind daher glatt, was sich für den Roboter als materialschonend erweist.

Die Aufgabe ist es, von einem Punkt auf einer Seite des Spielfelds zu einem Punkt auf der anderen Spielfeldhälfte zu fahren. In Abbildung 9.4 ist die Aufgabe veranschaulicht. Der Roboter startet bei dem roten Kreuz und fährt durch das Verhalten gesteuert zum blauen Kreuz. Die Zeit vom Losfahren bis zum Stehen auf der Zielposition wird gemessen und geht in die Bewertung der aktuell eingestellten Bremskurve ein.

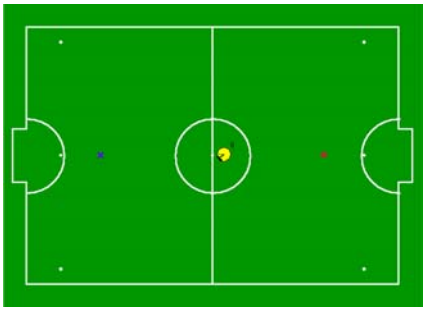

Abbildung 9.4: Testaufgabe zum Steuernlernen. Der Roboter startet auf dem roten Kreuz. Es soll so schnell wie möglich auf den blauen Punkt fahren und sich dort zur Tormitte ausrichten.

Die Bremskurve wurde mit einer Monte-Carlo-Methode optimiert. Dazu wurden zufällig die Stützstellen variiert und der Roboter musste mit der so modifizierten Bremskurve die Aufgabe bewältigen. Falls der Roboter die Aufgabe nicht innerhalb von 400 Frames (7,5 Sekunden) ausführte, wurde der Lauf abgebrochen. Da bei gleicher Bremskurve die Laufzeiten unterschiedlich sind, wurden pro Bremskurve zehn Läufe durchgeführt, um eine durchschnittliche Laufzeit zu schätzen. Liegt die durchschnittliche Laufzeit am Ende der zehn Läufe unter der aktuell am besten bewerteten Bremskurve, dann wird die aktuelle Bremskurve zur am besten bewerteten. Diese Variationen der Bremskurven wurden jeweils zehn mal durchgeführt, um eine gute Bremskurve zu erhalten (siehe Abbildung 9.5).

Die Bremskurve wurde auch mit einem genetischen Algorithmus optimiert. Dabei wurden keine Splines benutzt, sondern stückweise lineare Funktionen mit zehn gleichabständigen Stützstellen. Es wurde auch nicht jede Bremskurve zehn mal evaluiert, sondern nur einmal, da davon ausgegangen wurde, dass die unterschiedlichen Laufzeiten durch den genetischen Pool weniger ins Gewicht fallen. Im Ergebnis und der Lernzeit von insgesamt 100 Testläufen unterschieden sich die beiden Methoden jedoch nicht wesentlich.

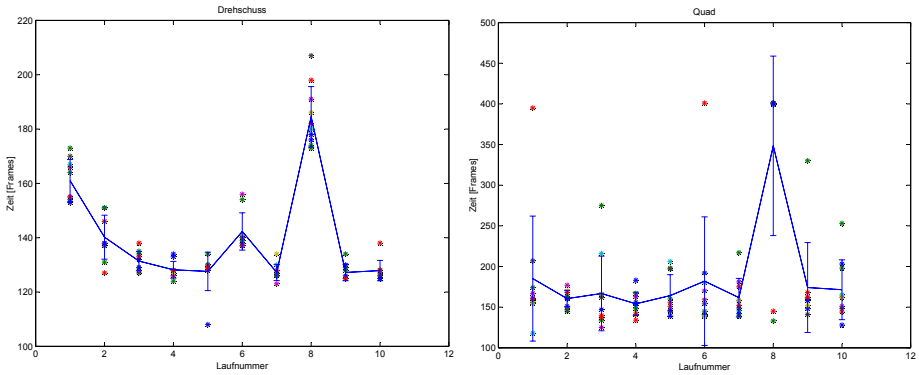

Abbildung 9.5: Testlauf zur Bremskurvenbestimmung. Rechts für einen dreirädrigen Roboter, links für einen vierrädrigen Roboter. Es wurden zu jeder Bremskurve zehn Testfahrten durchgeführt. Die Sterne geben die Zeit an, die der Roboter für die Aufgabe benötigte. Die blaue Linie verbindet die Mittelwerte der Testreihen, die blauen Balken geben die Standardabweichungen für die jeweilige Testreihe an.

9.6 Ergebnisse

Für die Optimierung des PID-Controllers wurden zwei Experimente durchgeführt. Im ersten Experiment wurden die handoptimierten Werte für den Offset, die maximale Beschleunigung und den Skalierungsfaktor beibehalten. Die P, I und D Werte wurden auf 0 gesetzt. Das Ergebnis der Qualitätsverbesserung eines Testlaufs ist in Abbildung 9.2(a) zu sehen, die in Abbildung 9.6 noch einmal nach den einzelnen Gütekriterien aufgeteilt ist. Nach 10 Iterationen der Parameteranpassung fährt der Roboter mit einer Durchschnittsgeschwindigkeit von 1m/s (mit Beschleunigen und Bremsen). Die durchschnittliche Abweichung von dem vorgegebenen Weg liegt nach dem Stoppen des Roboters bei 10% des zurückgelegten Wegs.

In einem zweiten Experiment blieben anfänglich alle 18 Parameter auf den handoptimierten Werten. Wie in Abbildung 9.7 zu sehen ist, sind die Parameter nicht unabhängig: Einige Parameter oszillieren am Anfang. Später stabilisiert sich jedoch das System. In Abbildung 9.7 ist außerdem gut zu erkennen, dass auch die PID Parameter für die Rotation gelernt werden. Dies liegt an der Bewertungsfunktion, denn auch die Orientierung der Roboters geht in die Bewertung ein. Um die Drehung beim Beschleunigen des Roboters zu korrigieren, die das inverse Modell innerhalb des Regelkreises erkennt, wird intern den empfangenen Fahrbefehlen ein Rotationsanteil aufaddiert.

Unterschiedliche Roboter haben auch verschiedene Bremskurven. In Abbildung 9.3 werden die Abweichungen deutlich. Daher ist es wichtig für jeden

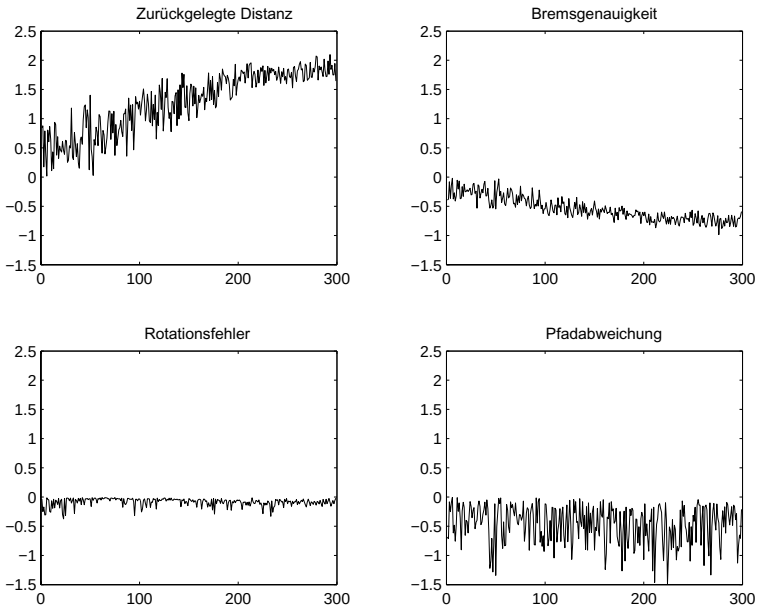

Abbildung 9.6: Die vier Bewertungskomponenten für die gewichtete Qualitätsfunktion. Während eines Testlaufs, bei dem die P-, I- und D-Werte mit 0 initialisiert wurden erhöht sich im Laufe des Experiments die zurückgelegte Distanz (oben links). Da der Roboter schneller fährt, benötigt er auch mehr Zeit zum Bremsen (oben rechts). Eine Fehlorientierung kann gut ausgeglichen werden (unten links), die Abweichung vom vorgegebenen Pfad auch (unten rechts). Bei den unteren beiden Kurven ist keine offensichtliche Verbesserung zu sehen. Es ist aber zu beachten, dass die Bewertungen nicht schlechter werden, obwohl die gefahrene Strecke immer länger wird.

Roboter, mindestens jedoch für jeden Robotertyp, eine eigene Bremskurve zu lernen. Dass die Roboter schneller werden, das Lernen also erfolgreich ist, geht aus Abbildung 9.5 hervor. Besonders bei dem dreirädrigen Roboter (linker Graph) wird dies deutlich. Die Fahrzeit sinkt von durchschnittlich 160 Frames (3 Sekunden) auf 130 Frames (2,5 Sekunden). Das ist eine Verbesserung um knapp 20%.

Die gelernten Parameter sind jetzt fester Bestandteil des FU-Fighters Systems. Bei geänderten Robotern, zum Beispiel durch andere Getriebe, ist es jetzt schnell möglich die Parameter des Regelkreises und der Bremskurve automatisch zu optimieren.

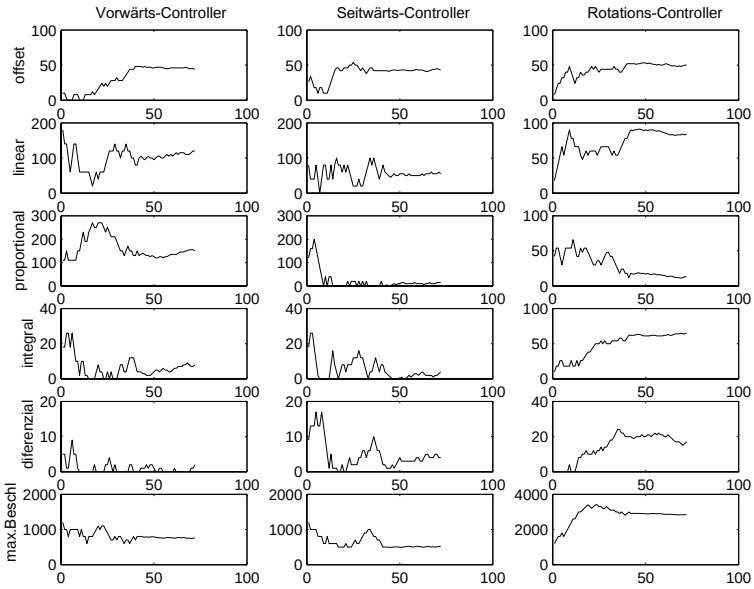

Abbildung 9.7: Die Änderung der 18 Parameter während eines Trainingslaufs. Die Parameter sind nicht unabhängig voneinander.

9.7 Ausblick und Zusammenfassung

Die Optimierung des Regelkreises ist eine Zwickmühle. Einerseits ist es notwendig schnell zu fahren und zu beschleunigen, andererseits ist es auch notwendig präzise zu fahren, um Kollisionen zu vermeiden, den Ball zu führen oder den Ball abzufangen. Beides schließt sich mit einem einfachen Regler gegenseitig aus. Es ist wünschenswert eine Steuerung zu haben, die das eine besonders gut kann, wenn gerade dies gefordert wird.

Der Regelkreis könnte noch auf zwei andere Arten verbessert werden: Erstens könnte das Modell des Roboters optimiert werden. Bisher wird davon ausgegangen, dass eine lineare Beziehung zwischen Motorgeschwindigkeiten und dem PWM-Wert besteht, der den Motor steuert. Dies ist jedoch eine zu einfache Annahme [Faulhaber, 2001]. Die Beziehung ist erstens nicht linear und zweitens auch von der Kraft abhängig, die der Motor zum drehen benötigt.

Die zweite Verbesserung des Regelkreises folgt aus den unterschiedlichen Zielwerten der Motorgeschwindigkeiten. Das Modell rechnet aus den Steuerwerten des PID-Reglers die Motorgeschwindigkeiten um und konvertiert diese in PWM-Werte. Da jedoch manche Motoren eine höhere Sollgeschwindigkeit haben als andere, erreichen sie diesen Zielwert auch langsamer als die mit einem niedrigeren Sollwert. Dies führt unmittelbar zu einer falschen Bewegung des Robo-

ters, die im nächsten Schritt durch das inverse Modell erkannt wird und vom Regelkreis ausgeglichen werden soll. Es wäre besser, wenn bei allen Motoren erreichbare, oder fast erreichbare Zielwerte gefordert würden. Dann würde das System immer noch maximal beschleunigen, aber in die „richtige Richtung“. Dazu ist eine lineare Skalierung der Zielwerte mit Berücksichtigung der größten Istwert-Zielwert-Differenz notwendig, die sich natürlich auch von Roboter zu Roboter ändern kann.

Genauso wie eine Bremskurve für die Position gibt es auch eine Bremskurve für die Orientierung. Diese Kurve wurde bisher noch nicht betrachtet. Gerade bei der Einführung neuer Roboter ist es hier schon zu Problemen gekommen. Das Vorhersagesystem kompensiert bis zu einem gewissen Grad unterschiedlich reagierende Roboter. Wenn sich die Hardware aber zu stark ändert, wie bei den FU-Fighters bei dem Einsatz des neuen vierrädrigen Torwarts, kann die Vorhersage die Unterschiede nicht mehr ausgleichen. In dem konkreten Beispiel neigte der Torwart regelmäßig zu starken Oszillationen bei der Orientierung, die auch nicht durch das erneute Training der Vorhersage verhindert werden konnten. Der Roboter ist viel leichter, hat einen 90^o statt 110^o Radöffnungswinkel und ein anderes Getriebe. Ein automatisches Training der Ausrichtungssteuerung hätte das Problem sicher schneller behoben.

Die drei Ebenen PID-Regelung, Generierung der Roboterbefehle und Fahren zu einer Zielposition hängen eng zusammen. Zur Vereinfachung des Verhaltens könnte die Struktur vereinfacht werden. Das würde auch einige Vorteile für die Optimierung der Fahrtrichtung haben, denn dann wäre dort der Zielabstand bekannt und die Drehung des Roboters könnte ebenfalls optimiert werden.

In diesem Kapitel wurde gezeigt, wie der Roboter optimal regeln bzw. gesteuert werden kann, um kontrolliert und in kurzer Zeit seine Maximalgeschwindigkeit zu erreichen und an einem Punkt zu stoppen. Auf dem Roboter wurde ein PID-Controller zum optimalen Beschleunigen optimiert, im Verhaltenssystem eine Bremskurve zum optimalen Bremsen gelernt. Die Optimierungen wurden auf unterschiedlichen Ebenes des Systems durchgeführt, da auf einer Ebene nicht alle notwendigen Informationen zur Verfügung stehen.