Echtzeit-Lernverfahren zur Fehlerbegrenzung und Driftkorrektur für Odometrie-basierte Lokalisationsverfahren

Ralf Der, Thomas Pantzer

21. Juni 2000

Zusammenfassung

Positionsbestimmung und Navigation sind essentielle Grundaufgaben im Forschungsgebiet "autonome Roboter". Die Koppelnavigation (Odometrie) ist ein bekanntes und mit geringem Hardwareaufwand zu realisierendes Verfahren zur lokalen Positionsbestimmung. Eine indoor-Umgebung, eines der häufigsten Einsatzgebiete von Robotern, ist meist durch orthogonale Strukturen geprägt. Der im folgenden Dokument beschriebene Algorithmus macht sich diese Eigenschaften zunutze, um einen Korrekturbeitrag für die fehlerbehaftete Koppelnavigation zu geben. Es werden mit einem Echtzeit-Lernverfahren Vorzugsrichtungen (Kanten) extrahiert, die u.a. auch zur Generierung topologischer Karten benutzt werden können.

1 Einleitung

1.1 Odometrie

Bei der Odometrie, in der Literatur auch als Koppelnavigation oder engl. "dead reckoning" bezeichnet, benutzt man Signalgeber an den Rädern, um den zurückgelegten Weg des Fahrzeugs (Roboter) messen zu können. Die Odometrie ist eine relative Positionsbestimmung. Die neue Position wird dabei aus einer vorher bekannten Position plus zurückgelegter Wegstrecke errechnet. Die Odometrie kann auf kurzen Distanzen sehr genaue Ergebnisse liefern. Allerdings entstehen auch Fehler, die mit zunehmender Entfernung wachsen. Die Räder können rutschen (Schlupf), unrund sein oder unterschiedliche Durchmesser haben (Reifenluftdruck), oder es können durch Bodenunebenheiten falsche Entfernungen gemessen werden. In [2] und [3] sind weitere Ansätze zur Selbstlokalisation ausführlich beschrieben.

1.1.1 Formeln

Die Position des Roboters ist das Tupel $P_{robot} = \{x, y, \alpha\}$, die Werte ΔL und ΔR werden direkt von den Radwegsensoren (Umdrehungszähler) abgelesen.

Drehwinkel ist Wegdifferenz des rechten und linken Rades geteilt durch Radabstand

$$\Delta\alpha = \frac{\Delta R - \Delta L}{D} \tag{1}$$

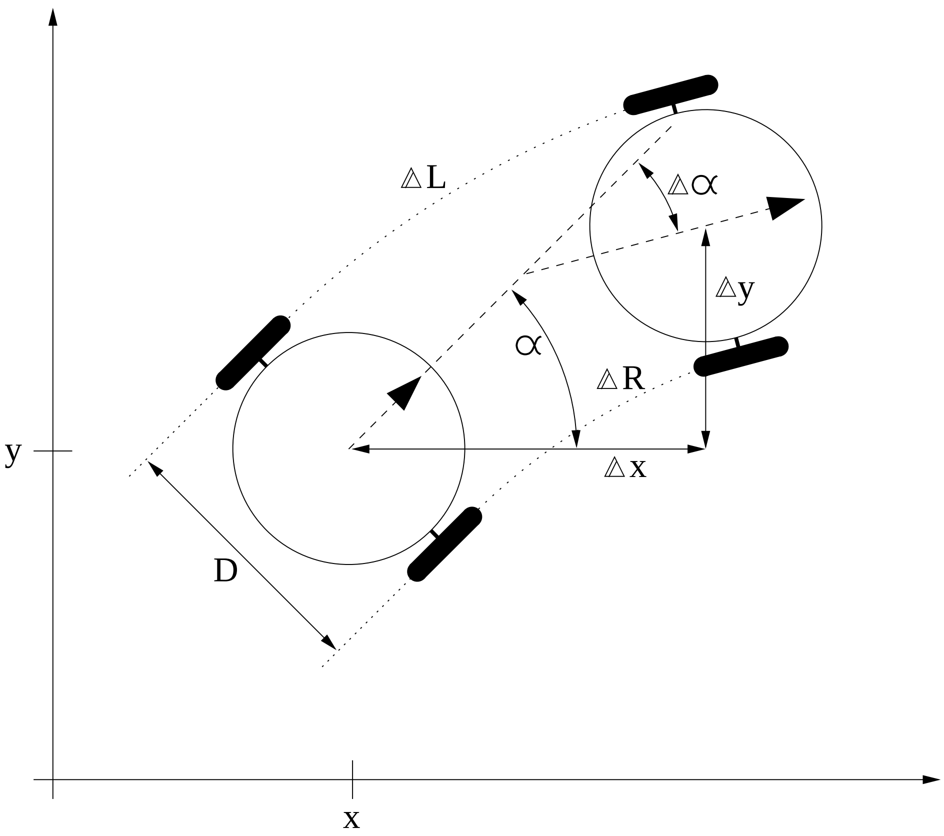

Abbildung 1: Odometrie

Wegstrecke ist mittlerer Weg

$$\Delta s = \frac{\Delta L + \Delta R}{2} \tag{2}$$

Positionsveränderung bei Geradeausfahrt ($\Delta L = \Delta R$)

$$\Delta x = \Delta R * \cos(\alpha) \tag{3}$$

$$\Delta y = \Delta L * \sin(\alpha) \tag{4}$$

Positionsveränderung bei Kurve ($-\Delta L \neq \Delta R$ und $\Delta L \neq 0$)

$$\Delta x = \frac{\Delta s}{\Delta \alpha} * \left(\cos\left(\frac{\pi}{2} + \alpha - \Delta\alpha\right) + \cos\left(\alpha - \frac{\pi}{2}\right)\right) \tag{5}$$

$$\Delta y = \frac{\Delta s}{\Delta \alpha} * \left(\sin\left(\frac{\pi}{2} + \alpha - \Delta\alpha\right) + \sin\left(\alpha - \frac{\pi}{2}\right)\right) \tag{6}$$

1.2 Roboter und Welt

Unsere Robotik-Experimente führten wir mit einem Modell des Khepera-Roboters durch. Dieser Roboter mißt ca. 50 mm im Durchmesser und besitzt 8 auf den Umfang verteilte Umgebungssensoren die im günstigsten Fall ungefähr 25 mm Reichweite haben. Angetrieben wird der Roboter durch zwei unabhängig steuerbare Schrittmotoren. Die Schrittweite beträgt ca. 0,08 mm.

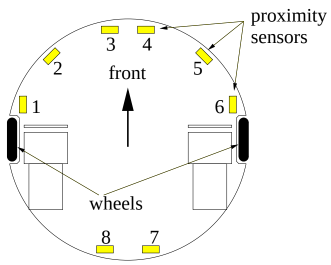

Abbildung 2: Prinzipskizze des Roboters

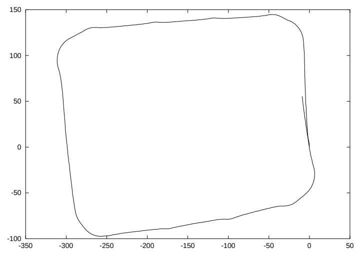

Abbildung 3: zurückgelegter Weg des Roboters in einfacher Umgebung

Das Versuchsgelände bestand aus vier rechtwinklig angeordneten Wänden. Der Roboter wurde mit einem einfachen Controller zur Wandverfolgung programmiert.

2 Algorithmus

2.1 Differentialgleichungen beschreiben Kurvenformen

Betrachtet man den zurückgelegten Weg eines Roboters, so stellt man fest, daß sich die Bahnkurve aus Elementen einfacher Kurvenformen zusammensetzen läßt. Die einfachste von ihnen ist die Gerade. Mit dem Anstieg der Geraden ist zugleich eine ganz charakteristische Eigenschaft dieser Spezialform einer “Kurve” in nur einem einzigen Wert repräsentiert. Dazu benötigt man nur noch die Koordinaten eines einzigen Punktes (Startpunkt) und die Länge des Wegstückes, um diesen Teil der Bahnkurve eindeutig festzulegen.

Verallgemeinert man diesen Ansatz auf das Prinzip “Differentialgleichungen beschreiben Kurvenformen”, hat man ein Mittel, um komplizierte Kurvenformen darzustellen und in den meist beschränkten Speichermedien der Roboter-CPU effektiv abzuspeichern. So kann zum Beispiel während der Verfolgung einer geraden Wand die Lage bzw. die Ausrichtung dieser Wand im Koordinatensystem oder bei der Umrundung eines Hindernisses sein Durchmesser festgestellt werden. Die Odometriedaten des Roboters werden dabei gleichzeitig gemittelt, um eventuelle Schlängelbewegungen des Wandfolgealgorithmus auszugleichen. Der aus den Odometriedaten resultierende Fehler kann klein angenommen werden, da nur die relative Positionsveränderung betrachtet wird.

2.2 Wettbewerb zwischen Differentialgleichungen

Die Basis des Algorithmus bildet ein Differentialgleichungsobjekt $\Theta = \{\overrightarrow{v}, p, \Phi, E\}$. Es enthält einen Vektor $\overrightarrow{v}$ mit allen Ableitungen, zum Beispiel $v_1 = \frac{\partial x}{\partial s}$, $v_2 = \frac{\partial y}{\partial s}$, die Wahrscheinlichkeit p, daß genau dieses Objekt beim nächsten Update berücksichtigt wird, die aktuelle Fitness Φ bzw. den momentanen Fehler E. Mehrere Differentialgleichungsobjekte Θ_i, eine Abstandsfunktion ϑ und die durchschnittliche Fitness φ bilden nun einen Pool $\Omega = \{\Theta_i, \vartheta, \varphi\}\ i \epsilon N$, aus dem in jedem Schritt immer ein Objekt herausgegriffen und einem Update unterzogen wird. Das Objekt mit der höchsten Wahrscheinlichkeit gewinnt den Update seiner Parameter. Die Wahrscheinlichkeiten der einzelnen Objekte werden nach ihren Fitnesswerten bzw. Fehlern eingestellt. Dazu wird mit der geeignet gewählten Abstandsfunktion ϑ festgestellt, wie gut eine Differentialgleichung auf das gerade zurückgelegte Wegstück paßt, und der zugehörige Fehler ermittelt. Wenn mehrere Differentialgleichungsobjekte annähernd gleich gut repräsentiert sind, soll kein Update vorgenommen werden. Nach ausreichend vielen Updateschritten sind die Koeffizienten konvergiert und die Differentialgleichungen widerspiegeln die geometrischen Eigenschaften der abgelaufenen Wegstücke.

3 Formalisierung am Beispiel

3.1 Differentialgleichung für Geraden

Die Differentialgleichung in der Form $\frac{dy}{dx} = m$ ist für unsere Zwecke ungünstig, da $m = \pm\infty$ werden kann, wenn die Gerade parallel zur y-Achse des Koordinatensystems verläuft. Deshalb gehen wir von der gebräuchlichen Form $y = m(x - x_0)$ zur Parameterdarstellung der Gerade über. Die Variable s ist in diesem Fall die Länge des zurückgelegten Weges auf der Geraden.

$$y = sa_y + y_0 \tag{7}$$

$$x = sa_x + x_0 \tag{8}$$

Nach Differentation erhalten wir

$$\frac{\Delta y}{\Delta s} = a_y \tag{9}$$

$$\frac{\Delta x}{\Delta s} = a_x \tag{10}$$

Diese Gleichungen sind nun genau dann erfüllt, wenn der Roboter sich auf einer Geraden bewegt, die durch die Parameter a_x und a_y beschrieben wird. Die Werte Δx, Δy und Δs können direkt aus den Odometriedaten des Roboters entnommen werden. Da es sich um relative Koordinaten handelt, ist der zu erwartende Fehler auch entsprechend klein. Um die Parameter a_x und a_y in kleinen Schritten auf einen gewünschten Wert hinzuführen, benötigt man eine Fehlerfunktion als Maß der Abweichung des aktuell zurückgelegten differentiellen Wegstücks $(\Delta x, \Delta y)$ und den intern angenommenen Werten (a_x, a_y).

3.1.1 Fehlerfunktion und Fitness

Mit den Beziehungen aus 9 und 10 formulieren wir für die Gleichung mit dem Index i folgende Fehlerfunktion.

$$E_i = \left(\frac{\Delta x}{\Delta s} - a_{x,i}\right)^2 + \left(\frac{\Delta y}{\Delta s} - a_{y,i}\right)^2 \tag{11}$$

Die Fitness der Gleichung i ergibt sich einfach aus

$$\Phi_i = -\frac{E_i}{4} \tag{12}$$

und die mittlere Fitness über alle betrachteten Differentialgleichungsobjekte beträgt

$$\overline{\Phi} = \sum_{i=1}^{n} p_i \Phi_i \tag{13}$$

3.1.2 Fisher-Eigen-Dynamik

Die Fisher-Eigen-Dynamik [1] beschreibt die Selektionsdynamik in einem Pool von Individuen unterschiedlicher Fitness ϕ_i und Wahrscheinlichkeit p_i. Dabei werden die Wahrscheinlichkeiten p_i so verändert, daß nach hinreichend langer Zeit das Individuum i^* mit der höchsten Fitness selektiert wird, d.h.

$$p_i = \left\{ \begin{array}{rl} 1, & falls\, i = i^* \\ 0, & sonst \end{array} \right\}$$

In jedem Zeitschritt wird ein Update für alle Wahrscheinlichkeiten der Differentialgleichungsobjekte im Pool durchgeführt (Fitness Φ_i nach Gleichung 12, n Anzahl der Differentialgleichungen).

$$\Delta p_i = \eta * \left(\Phi_i - \overline{\Phi}\right) * p_i \tag{14}$$

Die Wahrscheinlichkeiten werden normiert. Es gilt $0 \leq p_i \leq 1$ und $\sum_{i=1}^{n} p_i = 1.0$. Die Geschwindigkeit, mit der die Wahrscheinlichkeiten umschlagen können, kann mit dem Parameter η eingestellt werden ($\eta \approx [0.5\,..\,5.0]$).

3.1.3 Information und Update

Gleichzeitig mit dem Update der Wahrscheinlichkeiten p_i erfolgt ein Update der Koeffizienten der Differentialgleichungsobjekte zur Minimierung des Fehlers E_i. Die Größe des Updateschrittes für das Objekt i wird proportional zur Wahrscheinlichkeit p_i gewählt.

$$\Delta a_{j,i} = \varepsilon * p_i * \left(\frac{\Delta x}{\Delta s} - a_j\right) \qquad j = \{x, y\} \tag{15}$$

Bessere Ergebnisse erzielt man, wenn die Wahrscheinlichkeit p_i durch eine monoton wachsende Funktion der Gestalt

$$\sigma(p_i) = 1 - \frac{1}{1 + e^{(p-p_0)*\chi}} \tag{16}$$

ersetzt wird, wobei $p_0 \epsilon [0.4\,..\,0.7]$ und $1 \leq \chi \leq 25$ gewählt wird.

Läßt man den Algorithmus einige Zeit "mitlaufen" während der Roboter an einer längeren geraden Wand entlang läuft, werden die Parameter $a_{x,i}$ und $a_{y,i}$ sukzessive nachgeführt und spiegeln schließlich den Verlauf der Geraden wider. Der Vektor $\begin{pmatrix} a_{x,i} \\ a_{y,i} \end{pmatrix} = \overrightarrow{a_i}$ aus den gelernten Koeffizienten ist dann ein Einheitsvektor entlang dieser Geraden.

Die Information nach Shannon [4] dient dazu, Updates zu unterdrücken, falls mehrere Differentialgleichungen ein bestimmtes Wegstück gleich gut repräsentieren.

$$I = -\frac{1}{n}\sum_{i=1}^{n} p_i log(p_i) \tag{17}$$

Ein Update der Koeffizienten wird grundsätzlich nur dann durchgeführt, wenn I kleiner als ein vorgegebener Schwellwert ist. Günstige Werte für diese Schwelle liegen bei $\approx \frac{2}{3}\frac{1}{n}\log(n)$. In Abbildung 4 ist der Verlauf der Information über der Zeit bei einer Rundfahrt des Roboters (Trajektorie Abbildung 5) gezeigt. In die Roboterumgebung wurden vier unterschiedlich lange Kanten, ein Innen- und ein Außenradius eingebaut. Jede Nadelspitze beim Wert der Information markiert eine Richtungsänderung des Roboters. Der Wert der Information erreicht seinen tiefsten Wert beim Entlangfahren an der längsten Kante.

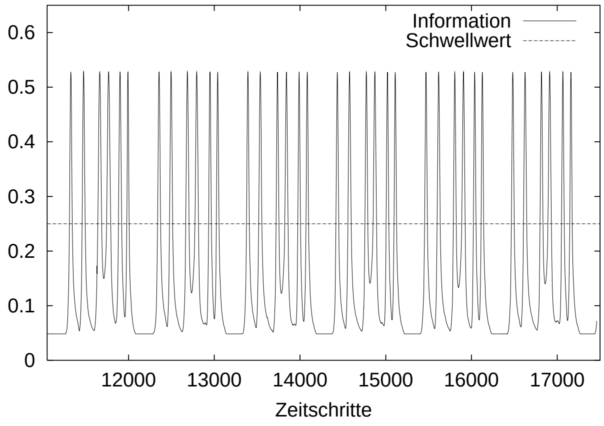

Abbildung 4: Verlauf der Information

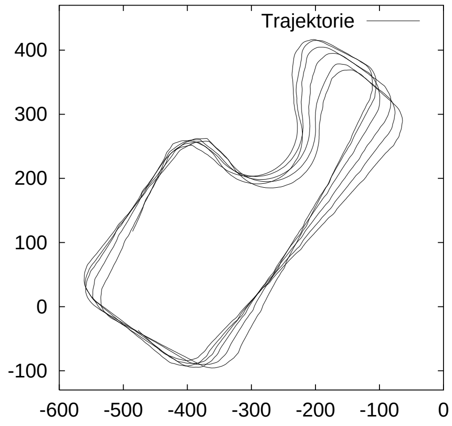

Abbildung 5: Weg des Roboters

4 Anwendung in der Praxis

4.1 Das Umschaltproblem

4.1.1 Verzögerter Anfang des Updates

Wird der Update der Differentialquotienten nicht in einem Schritt vorgenommen, und das ist der Fall, wenn der Faktor ε aus Gleichung 15 kleiner als 1.0 ist, dann erfolgt eine Mittelung über mehrere Zeitschritte. Ebenso erfahren die Wahrscheinlichkeiten p_i aus Gleichung 14 durch den Faktor η eine Mittelung. Das führt dazu, daß die Wahrscheinlichkeiten erst nach einigen Schritten ihr Maximum erreichen können. Der Update und damit die Verringerung des Fehlers und Erhöhung der Fitness erfolgen aber nur optimal, wenn die Wahrscheinlichkeit p über mehrere Schritte einen Wert in der Umgebung von 1.0 aufweist. Nach Gleichung 14 ist die Summe aller Wahrscheinlichkeiten gleich 1.0, d.h. wenn eine Gleichung eine Wahrscheinlichkeit von 95% hat, müssen sich alle anderen die restlichen 5% teilen. Die Wahrscheinlichkeit für eine unpassende Gleichung ist also sehr klein. Schwenkt der Roboter auf seiner Fahrt jetzt auf eine Bahn, deren zugehörige Gleichung noch eine niedrige Wahrscheinlichkeit hat, kann es sehr lange dauern, bis die neue, eigentlich passende Differentialgleichung ein Update erfährt. Bei sehr kurzen Wegstücken kann es sogar dazu kommen, daß überhaupt kein Update durchgeführt wird, weil dieses Wegstück bereits wieder verlassen wurde, bevor die Fisher-Eigen-Dynamik umschlagen konnte.

4.1.2 Schranke für p_i

Um diese Verzögerung klein zu halten, definiert man eine untere Schranke für die Wahrscheinlichkeiten, die von keinem p_i unterschritten werden soll. Ist das doch der Fall, setzt man das entsprechende p_i gleich dem Wert dieser Schranke. Die obere Schranke, die die p_i nicht überschreiten können, ergibt sich damit durch folgende Gleichung.

$$p_o = 1.0 - (N-1) * p_u \qquad p_u \approx 0.02 \tag{18}$$

Diese Schranken müssen beim Update der Wahrscheinlichkeiten p_i nach Gleichung 14 berücksichtigt werden.

4.1.3 Verzögerter Update

Die Änderungen der Wahrscheinlichkeiten p_i folgen im Falle eines Umschwenkens immer den Fitnesswerten nach. In ungünstigen Situationen kann es passieren, daß die falschen Koeffizienten aufgrund ihrer noch hohen Wahrscheinlichkeitswerte einen Update mit völlig unpassenden Werten erfahren. Um das zu verhindern, kann man

1. die Lernparameter etwas vergrößern, damit schneller gelernt werden kann, oder
2. den Update mit verzögerten Werten von Δx und Δy vornehmen, während man die Fitness aus aktuellen Werten errechnet. Das Lernen der Koeffizienten findet damit ebenfalls verzögert statt.

Der erste Ansatz löst das Problem nicht prinzipiell, weil die Lernrate η indirekt die Länge des Wegstückes bestimmt, das mit einer bestimmten Kurve (Differentialgleichung) verglichen werden soll. Je höher die Lernrate gewählt wird, umso kürzer ist das “passende” Wegstück.

Der zweite Ansatz funktioniert nur unter der Annahme, daß der Roboter sich größtenteils auf geradlinigen Bahnen bewegt und ein “Einschwenken” auf andere Richtung relativ schnell erfolgt. Weiterhin muß nun doch ein Teilstück des zurückgelegten Weges mit Hilfe von Punktkoordinaten abgespeichert werden. Die gewählte Verzögerung muß der Weglänge entsprechen, die während der Richtungsänderung zurückgelegt wird. Der Betrag der Verzögerung kann durch Auswerten der Information nach Shannon über den Fitnesswerten Φ_i und den Wahrscheinlichkeiten p_i relativ gut geschätzt werden.

4.2 Spezielle Fehlerfunktion für Geraden

Für die Fehlerfunktion benötigen wir ein Maß für den Abstand zweier Differentialgleichungen[1]. Die aus dem gerade zurückgelegtem Wegstück angenommene Gleichung soll mit jeder anderen Differentialgleichung verglichen werden. Der größte erreichbare Abstand soll per Definition den Wert 1.0 erhalten. Da wir in unserem Spezialfall den Anstieg der Geraden in zwei Variablen ausgedrückt haben, können wir aus deren Vorzeichen auch noch eine Richtung bestimmen (Vektor). Der Abstand soll den Wert 1.0 haben, wenn die zwei Geraden den größten Abstand voneinander aufweisen, d.h. die Vektoren genau in die entgegengesetzte Richtung zeigen bzw. einen Winkel von 180° einschließen. Bei Identität soll der Abstand 0.0 betragen.

Die Argumente für die Abstandsfunktion sind die Wegstrecken-Informationen $\Delta x_t, \Delta y_t$ (oder Winkel α_t und Weglänge Δs_t) aus einem Zeit-Schritt t und die zu lernenden Werte $a_{x,i}, a_{y,i}$, die den Winkel β_i (Ausrichtung) der Gerade i (Kante) im internen Koordinaten-System repräsentieren. Im ideal gelernten Zustand im Zeitschritt t sind die folgenden Ausdrücke äquivalent.

$$\tan(\alpha_t) = \frac{\Delta y_t}{\Delta x_t}, \quad \sqrt{\Delta x_t^2 + \Delta y_t^2} = \Delta s_t \tag{19}$$

und

$$\tan(\beta_i) = \frac{a_{y,i}}{a_{x,i}}, \quad \sqrt{a_{x,i}^2 + a_{y,i}^2} = 1 \tag{20}$$

Mehrere Abstandsfunktionen wurden getestet.

1. Die erste betrachtete Abstandsfunktion ist die Fehlerfunktion nach Gleichung 11
2. In der zweiten Version wurde das Quadrat aus Gleichung 11 durch den Betrag ersetzt.

$$E = j(\alpha_t, \beta_i) = \frac{\left|\frac{\Delta x_t}{\Delta s_t} - a_{x,i}\right| + \left|\frac{\Delta y_t}{\Delta s_t} - a_{y,i}\right|}{2\sqrt{2}} \tag{21}$$

3. Die dritte Version der Abstandsfunktion widerspiegelt den Betrag des Winkelabstandes.

$$E = g(\alpha_t, \beta_i) = \arccos\left(\frac{\Delta x_t * a_{x,i} + \Delta y_t * a_{y,i}}{\Delta s_t}\right) * \pi^{-1} \tag{22}$$

In Abbildung 6 auf Seite 9 ist der Verlauf der einzelnen Funktionen an der Stelle $\beta = \frac{\pi}{5}$ gezeichnet, der Wert für α läuft zwischen $-\pi$ und π. Alle Abstandsfunktionen haben den Maximalwert 1.0, wenn die zu vergleichenden Vektoren in die entgegengesetzte Richtung zeigen. Das sehr langsame Ansteigen des Abstandes nach Gleichung 11 bei kleinen Winkel-Differenzen hat zur Folge, daß die Wahrscheinlichkeitswerte der einzelnen Differentialgleichungen nicht

[1] In diesem Spezialfall kann es der Winkel zwischen zwei Geraden sein.

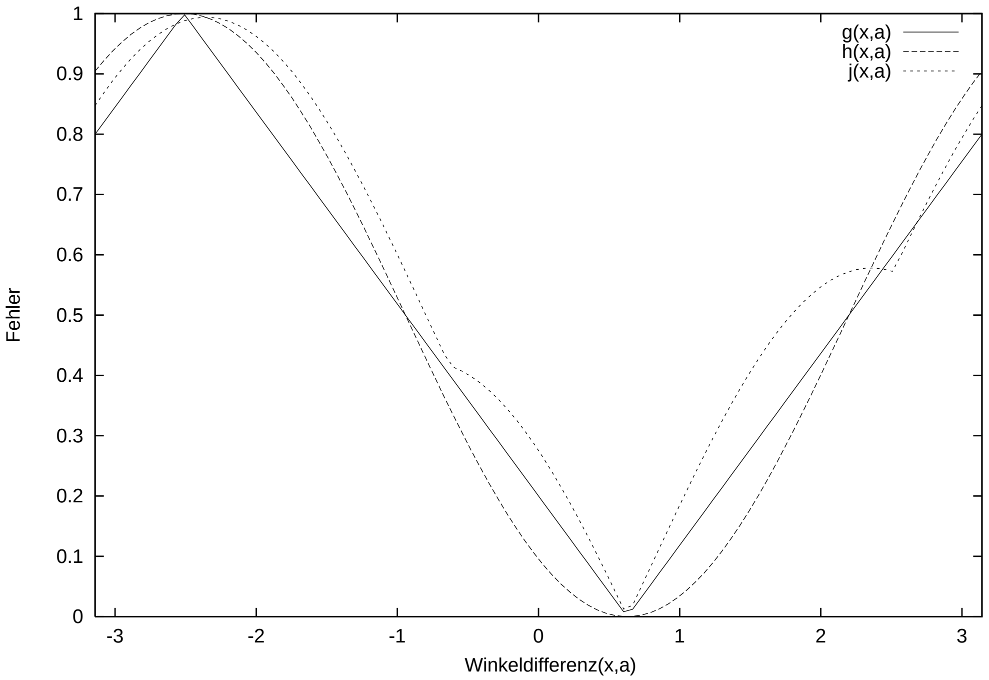

Abbildung 6: Abstandsfunktionen für Geradengleichungen

entsprechend eingestellt werden und die Information geringer ist (d.h. der Wert der Information nach Gleichung 17 steigt an). Für den Fall, daß mehrere Geraden nahe beieinander liegen, können Effekte wie das Heranziehen eigentlich unpassender Geraden auftreten. Bei der Funktion nach Gleichung 21 treten diese Effekte gar nicht mehr auf. Sie weist die größte Trennschärfe im Nahbereich auf, d.h. der Unterschied zwischen zwei ähnlich gut passenden Geradengleichungen ist größer, wodurch sich die Wahrscheinlichkeitswerte schneller neu einstellen können. Der gegenüber den Gleichungen 22 und 11 sehr ungewöhnliche Funktionsverlauf kann in Kauf genommen werden, da die Unregelmäßigkeiten nur in dem Bereich bemerkbar werden, in dem der Abstand zweier Geraden größer als 90° ist. Wenn wir annehmen, daß mindestens vier verschiedene Geradengleichungen der Menge der betrachteten Gleichungen angehören und der Raum, in dem sich unser Roboter bewegt, vier rechtwinklig zueinander stehende Wände hat, so ist dieser Abstand gleichzeitig der größte erreichbare Abstand überhaupt.

Die Abstandsfunktion nach Gleichung 22 gibt mit der direkten Differenz zwischen zwei Winkeln zwar ein begrifflich sehr gut faßbares Abstandsmaß, benötigt aber den meisten Rechenaufwand. Es lassen sich außerdem Funktionen finden, die steilere Anstiege aufweisen. Im allgemeinen eignen sich Funktionen mit steilen Anstiegen in der näheren Umgebung des Referenzpunktes besser als Abstandsfunktion.

Die Fehlerfunktion nach Gleichung 11 ist im Prinzip eine zwischen 0.0 und 1.0 normierte Kosinusfunktion die als Argument den Differenzwinkel der beiden Geraden (Vektoren) hat. Bei der Suche nach Funktionen mit steileren Anstiegen im Nahbereich fiel die Funktion $d = \frac{\cos(x)^3+1}{2}$ auf. Ein weiterer Vorteil dieser Funktion ist das Plateau bei $x = \pm\frac{\pi}{2}$, das bewirkt, daß alle in der näheren Umgebung liegenden Vektoren annähernd gleich stark berücksichtigt werden und die Entscheidung über den Updatesieger stärker durch die Wahrscheinlichkeiten p_i erfolgt.

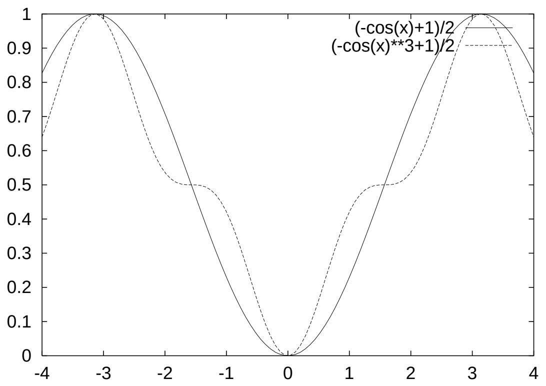

Abbildung 7: Abstandsfunktion mit steilerem Anstieg

$$E = \frac{1}{2} * \left(\cos\left(\left(\arctan\left(a_{y,i}, a_{x,i}\right) - \arctan\left(\Delta y_t, \Delta x_t\right)\right) - \pi\right)^3 + 1\right) \tag{23}$$

Diese Abstandsfunktion lieferte bei unseren Experimenten die besten Ergebnisse. Bei Richtungsänderungen des Roboters schlugen die Wahrscheinlichkeiten schnell um und die Anzahl der Fälle, in denen zwei Differentialgleichungsobjekte über einen längeren Zeitraum gleiche Wahrscheinlichkeiten aufwiesen, wurde minimiert.

5 Erweiterungen

5.1 Kreisbahnsegmente

Nachdem gezeigt wurde, daß das Prinzip der lernenden Differentialgleichungen für einfache Kurvenformen anwendbar ist, kann man nun zu komplizierteren Kurvenformen übergehen. Die einfache Gleichung für den Kreisumfang $U = 2\pi R$ kann nach R umgestellt werden. Der Radius ergibt sich aus dem zurückgelegtem Weg auf der Kreisbahn geteilt durch die Winkeldifferenz in der Ausrichtung.

$$R = \frac{U}{2\pi} \approx \frac{\Delta s}{\Delta \alpha} = a_i \tag{24}$$

Der Radius einer Kreisbahn ist also der Parameter a_i, der mit dieser Differentialgleichung gelernt werden kann. Durch negative Winkel kann der Parameter a_i auch ein negatives Vorzeichen bekommen. Kreisradien sind aber immer positiv, das Vorzeichen bezeichnet hier die Richtung, auf der sich der Roboter auf der Kreisbahn bewegt (positiv = entgegen dem Uhrzeigersinn). Die Werte von $\Delta\alpha$ und Δs können sehr stark fluktuieren. Das hat zur Folge, daß der Parameter a, der ja der Quotient aus $\Delta\alpha$ und Δs ist, noch stärkeren Schwankungen unterliegen würde. Um Fehler bei der Berechnung im Computer entgegenzuwirken, werden $\Delta\alpha$ und Δs als getrennte Variablen im Speicher gehalten. Der Update der Kreisgleichungsparameter erfolgt (analog zu Gleichung 15) so:

$$\Delta a_i = \varepsilon * p_i * (a_i * \Delta\alpha - \Delta s) \tag{25}$$

5.2 Abstandsfunktion für Kreisradien

Für die Fehlerfunktion benötigt man eine Funktion die den Abstand von zwei Kreisradien auf den Wertebereich $[0\,..\,1.0]$ abbildet.

$$E_i = 1 - \frac{\left(\frac{k}{2}\right)^2}{(a_i * \Delta\alpha - \Delta s)^2 + \left(\frac{k}{2}\right)^2} \qquad k\epsilon[1\,..\,1000] \tag{26}$$

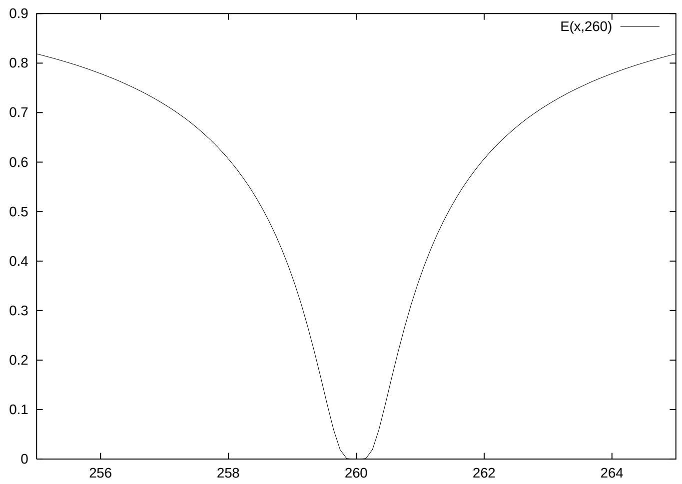

Abbildung 8: Abstandsfunktion für Kreisradien

Diese Funktion ist der Woods-Saxon-Funktion aus [???] sehr ähnlich. Mit dem Parameter k läßt sich die Steilheit bzw. mittlere Breite der Funktion einstellen und damit die Trennschärfe je nach Anwendungsgebiet beinflussen oder geeignet auswählen. Im Unendlichen steigt der Wert der Funktion jeweils auf 1.0.

5.3 Kreisgleichungen gegen Geradengleichungen

Differentialgleichungsobjekte für Kreisbahnen können sich nur dann gegen die für Geraden durchsetzen, wenn wenige andere Differentialgleichungsobjekte vorhanden sind. Anderenfalls ist der Konkurrenzdruck zu hoch und die Wahrscheinlichkeiten können sich nicht auf einem Wert stabilisieren, bei dem Updates möglich werden. Durch die Wahl eines größeren Wertes für k in Gleichung 26 kann man zwar den Einzugsbereich für bestimmte Instanzen von Differentialgleichungsobjekten vergrößern, beschränkt aber damit gleichzeitig die Anzahl der sinnvoll nutzbaren Objekte. Ein zu großer Wert für k führt allerdings dazu, daß diese eine Gleichung immer wieder einen sehr hohen Fitnesswert aufweist und damit kein Umschlagen der Wahrscheinlichkeiten erfolgen kann.

Allgemein kann man sagen, daß kompliziertere Kurvenformen seltener einen Update gewinnen, weil die einfacheren Kurvenformen (z.B. Geraden) in ihnen schon enthalten sind. Damit weist neben der betrachteten komplizierten Kurvenform auch immer eine einfache Kurvenform eine relativ hohe Fitness auf und die Information nach Gleichung 17 bleibt unter dem Schwellwert für zulässige Updates. Da sich die Konstruktion von wirklich gleichwertigen Abstandsfunktionen für verschiedene Klassen von Differentialgleichungsobjekten schwierig gestaltet, faßt man immer nur gleichartige Objekte in einer Menge Ω zusammen.

6 Virtuelle Rotation und Driftkorrektur

6.1 Typische Fehlerbilder

In Abbildung 9a ist der zurückgelegte Weg des Roboters bei einer Versuchsfahrt dargestellt.

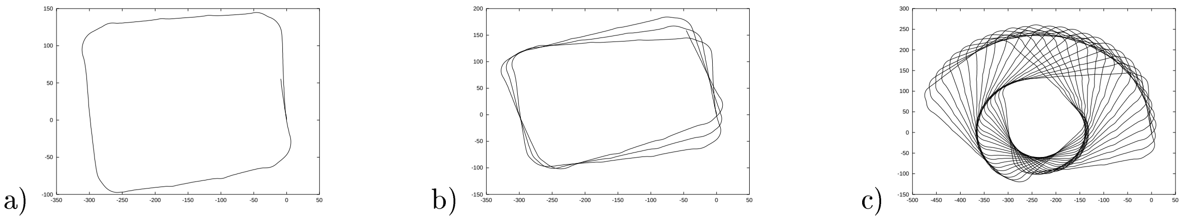

Abbildung 9: Trajektorie des Roboters nach 1, 3 und 21 Umrundungen, Rechteck-Welt

Ein Skalenteil entspricht dabei einem Millimeter in der wirklichen Welt. Der Roboter wurde auf Wandverfolgung programmiert und hat die aufgebaute Welt (ein Rechteck mit abgerundeten Ecken) ca. 21 mal in einer Richtung umrundet. Dabei wurde nur aus den Odometriedaten der Räder die aktuelle Position des Roboters ca. aller 1.2 mm zurückgelegten Weges neu berechnet. Deutlich ist ein systematischer Fehler bei der Berechnung der aktuellen Position zu erkennen, der von verschiedenen Reibwerten der Räder oder Differenzen der Raddurchmesser herrühren kann. Durch diesen Fehler wird die momentane Ausrichtung des Roboters falsch berechnet und erzeugt eine virtuelle Drehung des Weltkoordinatensystems. Die Winkelgeschwindigkeit[2] dieser Drehung wird nur durch Eigenschaften des Roboters (Raddurchmesser, Radstand, Spurweite, Bereifung) und Eigenschaften des Untergrundes bestimmt. Die Abhängigkeit dieses Fehlers von der Länge oder Art des zurückgelegten Weges oder der Gesamtdrehung des Roboters bezüglich seiner ursprünglichen Ausrichtung ist nicht durch eine von vornherein bekannte Funktion darstellbar. Die in Abbildung 10 dargestellte Trajektorie macht das deutlich. Sie wurde unter den gleichen Bedingungen wie die erste Trajektorie auf-

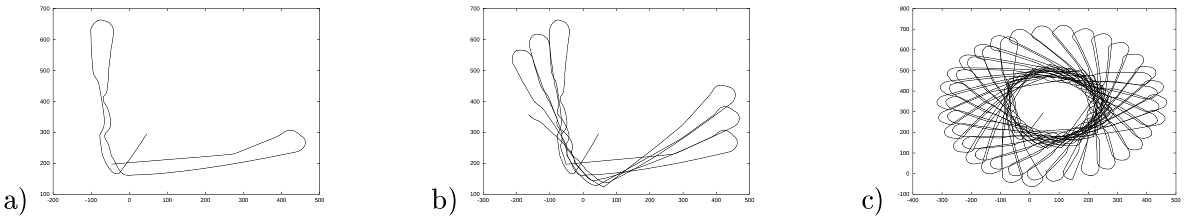

Abbildung 10: Trajektorie des Roboters nach 1, 3 und 24 Umrundungen, L-Welt

genommen. Im Gegensatz zu der in Bild 9 gezeigten Trajektorie beträgt die Gesamtdrehung

[2]Winkelabweichung pro Umlauf bzw. pro zurückgelegter Wegstrecke

des Roboters in diesem Fall nach jeder durchlaufenen Runde wieder 0°. Die Bahn ist im Prinzip eine geknickte, liegende Acht. Der Roboter mußte auf seinem Weg im selben Maß Links- und Rechtsdrehungen ausführen. Die Winkelabweichung war in diesem Fall kleiner, bleibt aber trotzdem ein systematischer Störfaktor. Auf das Phänomen der virtuellen Rotation der Weltkoordinaten durch fehlerbehaftete Positionsbestimmung wird auch in [2] hingewiesen.

6.2 Driftkorrektur

Durch eine Erweiterung des oben beschriebenen Algorithmus kann zusätzlich zu den gelernten Bahnparametern ein Driftparameter γ gelernt werden, der die Winkelgeschwindigkeit der virtuellen Rotation des Weltkoordinatensystems widerspiegelt. Durch Integration von γ kann zu jedem Zeitpunkt die aktuelle Verdrehung φ des Koordinatensystems bestimmt und damit der systematische Fehler korrigiert werden. Die Parameter γ und φ werden in jedem Zeitschritt neu berechnet.

$$\varphi_{neu} = \varphi_{alt} + \gamma \tag{27}$$

und

$$\gamma_{neu} = \gamma_{alt} + \Delta\gamma \tag{28}$$

Um die Änderung der Winkelgeschwindigkeit $\Delta\gamma$ erfassen zu können, benötigt man Geradengleichungen, die bereits gut angepaßt worden sind. Damit diese Anpassung durch die permanente virtuelle Drehung der Weltkoordinaten nicht verlorengeht, wird der Update der Geradengleichungen nach Gleichung 15 schon mit den rückgedrehten Werten δx_s und δy_s anstelle von Δx und Δy durchgeführt.

$$\vec{r} = \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}, \quad T = \begin{pmatrix} \cos(-\varphi) & \sin(-\varphi) \\ -\sin(-\varphi) & \cos(-\varphi) \end{pmatrix}$$

$$\begin{pmatrix} \Delta x_t \\ \Delta y_t \end{pmatrix} = \vec{r_t} = T\vec{r} \tag{29}$$

$$\Delta\gamma = \sum_{i=1}^{n} \frac{\varepsilon * \sigma(p_i) * \Delta s}{K} * (((\delta x_s - a_{x,i}) * \delta y_s) - ((\delta y_s - a_{y,i}) * \delta x_s)) \tag{30}$$

$$\delta x_s = \frac{\Delta x_t}{\Delta s}, \quad \delta y_s = \frac{\Delta y_t}{\Delta s}, \quad K \epsilon [5000 .. 50000]$$

Mit dem Faktor K wird das Verhältnis der Lerngeschwindigkeit der einzelnen Koeffizienten a_i zur Winkelgeschwindigkeit $\Delta\gamma$ eingestellt. Für ε wird der gleiche Wert wie in Gleichung 15 angesetzt. In den Abbildungen 11 und 12 sind die in Echtzeit korrigierten Trajektorien der oben gezeigten Testläufe zu sehen. Die restliche Parallelverschiebung resultiert aus der Differenz von angenommenen und eigentlichen Mittelpunkt des Roboterkoordinatensystems während der Rückdrehung der Punkte einer Trajektorie. Da die Parameter für den Verdrehungsfehler online gelernt werden, können auch Änderungen des Untergrundes, die eine Änderung des systematischen Fehlers hervorrufen, automatisch korrigiert werden.

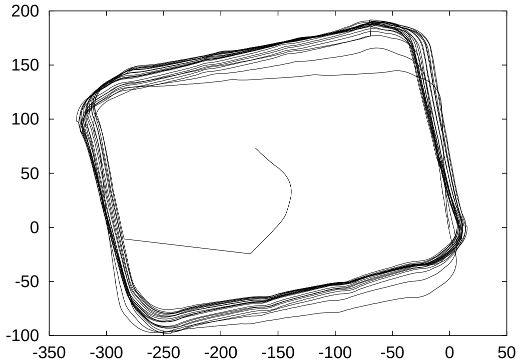

Abbildung 11: online korrigierte Version der Trajektorie aus Abbildung 9

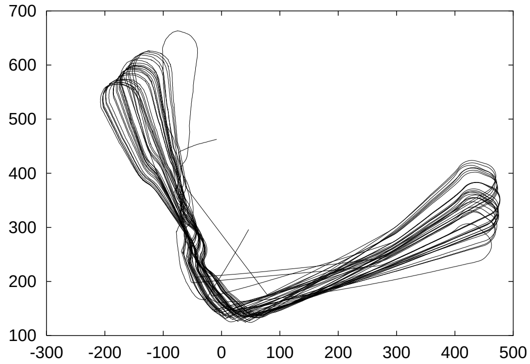

Abbildung 12: online korrigierte Version der Trajektorie aus Abbildung 10

7 Ausblick

Der vorgestellte Algorithmus hat gezeigt, daß er zur Landmarkenerkennung, speziell zur Erkennung von bestimmten Pfadprofilen, genutzt werden kann. Er ist dabei nicht an das Verfahren der Odometrie gebunden. Die Strukturen der Umgebung (Wände) müssen nicht durch Wandverfolgung "erlaufen" werden, sondern können auch das Ergebnis eines Laserscans sein. Sortiert man die Punkte eines Laserscans nach dem Winkel, unter dem sie aufgenommen wurden (bereits während des Scanvorgangs möglich), erhält man annähernd dieselbe Trajektorie wie bei der Wandverfolgung[3]. Mit mehreren aufeinanderfolgenden Scans (auch aus der Bewegung heraus) kann die Verdrehung direkt laufend bestimmt werden und man erhält eine Art Kompaß, der sich an den Gegebenheiten des Raumes orientiert.

In [2], Abschnitt 5.3 beschreibt Gutmann einen Algorithmus zur Berechnung der Verdrehung eines Laserscans unter Benutzung eines Winkelhistogramms. Der oben beschriebene Algorithmus erzielt in ähnlicher Weise die gleichen Ergebnisse, nur mit dem Unterschied, daß die Winkel direkt aus den Koeffizienten der Differentialgleichungsobjekte abgelesen werden können, ohne daß eine Kreuzkorrelation benötigt wird. Weiterhin sind erste (wenn auch nur partiell korrekte) Ergebnisse schon nach einigen wenigen Updateschritten verfügbar und eine Erhöhung der Anzahl der Scanpunkte verbessert auch die Genauigkeit, ohne das signifikant mehr Rechenleistung benötigt wird. Da in jedem Schritt immer dieselben einfachen Operationen durchgeführt werden, kann dieser Algorithmus auch direkt in Hardware realisiert werden.

[3] verdeckte Wände sind nicht berücksichtigt

Literatur

[1] Werner Ebeling, Andreas Engel, and Rainer Feistel. *Pysik der Evolutionsprozesse.* Akademie Verlag, Berlin, 1990.

[2] Jens Steffen Gutmann. Vergleich von Algorithmen zur Selbstlokalisierung eines mobilen Roboters. Master's thesis, Universität Ulm, 1996.

[3] Jens Steffen Gutmann, Wolfgang Hatzack, Immanuel Herrmann, Bernhard Nebel, Frank Rittinger, Augustinus Topor, Thilo Weigel, and Bruno Welsch. Reliable self-localization, multirobot sensor integration and basic soccer skills. 1998.

[4] Claude E. Shannon and Warren Weaver. *Mathematische Grundlagen der Informationstheorie.* R. Oldenbourg, München, 1976.