Mobile und verhaltensbasierte Robotik

Vorlesungen: Prof. Ralf Der - Zusammenfassung: Martin Christian

16. Dezember 2007

Mobile Robotik: Hauptziel der mobilen Robotik ist, dass Roboter lernen sich in ihrer Umgebung zu orientieren. Dies beinhaltet die Erfassung der Umgebung durch Sensoren, die Extraktion von Umgebungsmerkmalen, die Lokalisation anhand der Sensordaten und Merkmalen, evtl. die Aktualisierung der Karte und schlussendlich die Ansteuerung des Motors.

Verhaltensbasierte Robotik: Hauptziel der verhaltensbasierten Robotik ist das Verhalten des Roboters möglichst eng an die Sensordaten zu koppeln, ohne ein Weltmodell aufzubauen, so wie es bei niederen Tierarten vermutet wird. Dies beinhaltet ebenfalls die Erfassung der Sensordaten und die Extraktion von Umgebungsmerkmalen. Darüber hinaus geht es auch um die Architektur des Roboters und die kognitiven Prozesse wie Adaption und Lernen.

1 Sensoren

In der klassischen KI dienen Sensoren dem Weltverständnis. Daher sollen möglichst alle Informationen extrahiert werden. In der verhaltensbasierten Robotik dienen Sensoren der Motorsteuerung. Es werden daher nur relevante Informationen benötigt. Das veringert die Komplexität von NP auf P.

Die Sensormotorische Schleife kann offen oder geschlossen sein:

- Open Loop: Die Aktionen werden nicht durch Sensordaten überprüft. Dies wird z. B. bei der Berechnung einer Trajektorie von der Ausgangs- zur Zielposition benutzt.
- Closed Loop wird eine Aktion a_t, die im Zustand s_t ausgeführt wurde, bewertet. Dazu wird der Weltzustand s_{t+1} durch Sensoren ermittelt und mit dem geplanten Folgezustand $\widehat{s}_{t+1}$ verglichen. Das Ergebnis wird dem Controler zur Verfügung gestellt.

Logische Sensoren liefern bestimmte Perzepte. Sie bestehen aus einem Signalprozessor für die Rohdaten des physikalischen Sensors und einem Algorithmus zum Extrahieren der wichtigen Informationen. Logische Sensoren können einfach durch Perzeptive Schemata realisiert werden.

Sensoren können ausfallen oder in bestimmten Umgebungen nicht richtig funktionieren. Außerdem lassen sich manche Perzepte (z. B. Bewegung) nur durch Kombination verschiedener Sensoren erkennen. Daher müssen Sensoren kombiniert werden.

1.1 Klassifikation von Sensoren

Unterscheidung nach der Ausrichtungen:

- Propriozeptive S. erfassen den Zustand des Roboters, wie z. B. Gelenkpositionen, Kräfte, Momente
- Exterozeptiv S. erfassen den Zustand der Welt, wie z. B. Berührung, Abstand, Wärmeverteilung

Unterscheidung nach dem Energieverbrauch:

- Passive S. nehmen Energie aus der Umgebung auf (Lichtsensoren)
- Aktive S. senden Energie in die Umgebung (Ultraschall)

Sensoren können fixiert sein oder aktiv sein. Die Aktivität bezieht sich hier nicht auf die Energie, sondern auf die Fähigkeit des Roboters den Sensor auf die Umgebung einstellen zu können. Physikalisch können Sensoren nach Modalität, Energieverbrauch, Sichtfeld, Reichweite, Zuverlässigkeit, Genauigkeit, Antwortverhalten in einer bestimmten Umgebungen, Komplexität der Berechnung, Größe und Kosten unterschieden werden.

1.2 Sensorarten

- Rad/Motorsensoren messen die Radgeschwindigkeiten über Lichtwellen, z. B. Optical Encoders
- Richtungssensoren: messen die Bewegungsrichtung durch propriozeptive oder exterozeptive Sensoren. Zusammen mit der Geschwindigkeit ermöglicht es die Integration der Postion: $\vec{p} = \int \vec{v}$. Dies wird Dead Reckoning genannt. Beispiele:
  - Kompass:
    1. Die Hall Effekt Methode misst die Spannungsschwankungen in einem Halbleiter, wenn das magnetische Feld der Erde durchkreuzt wird.
    2. Die Flux Gate Methode basiert auf zwei Spulen, die die Stärke des Magnetfelds messen.
  - Kreiselinstrument: Durch die Rotation eines Rades bleibt seine Stellung stabil. Wenn eine Kraft auf das Rad wirkt, entsteht eine Gegenkraft. Diese Gegenkraft kann gemessen und zur Bestimmung der Richtung genutzt werden.

- Funkfeuer: funktionieren wie Leuchttürme, nur statt Licht wird Radar oder Funk verwendet. Beispiel: GPS
- Entfernungssensoren: messen die Entfernung zur Umgebung, z. B. Laserrange oder Ultraschall Sensoren. Methoden:
  - Time-of-flight: $d = c \cdot t$, wobei c die Ausbreitungsgeschwindigkeit ist
  - Phasenverschiebung: Das Licht wird von den meisten Gegenständen mit der gleichen Frequenz reflektiert wie es ausgesendet wurde. Daher kann die Verschiebung der Frequenz (Sinuswelle) gemessen werden und auf die Distanz zurückgerechnet werden.
  - Triangulation: Die Reflektion wird an einer anderen Stelle gemessen als dem Sendeort.
- Bewegungssensoren: Durch den Dopplereffekt von Radar oder Schallsensoren kann die Bewegung gemessen werden.
- Optische Sensoren: Messen Infrarot oder normales Licht mit entsprechenden Kameras.

1.3 Unsicherheit

Sei X die Zufallsvariable über einem Ereignisraum Ω:

Diskreter Ereignisraum: Die Elemente in Ω sind abzählbar.

1. Wahrscheinlichkeitsfunktion: $(P(X = x_i) = p_i$
2. Erwartungswert: $E(X) = \bar{x} = \sum_{i=1}^{n} x_i \cdot p_i = \frac{1}{n} \sum_{i=1}^{n} n_i \cdot x_i$
3. Varianz: $Var(X) = E\left((x_i - \bar{x})^2\right) = \sum_{i=1}^{n} (x_i - \bar{x})^2 \cdot p_i = \sum_{i=1}^{n} x_i^2 - n \cdot \bar{x}^2$
4. Kovarianz: $Cov(X, Y) = E\left[(x_i - \bar{x}) \cdot (y_i - \bar{y})\right] = \frac{1}{n} \sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y}) = E(X \cdot Y) - E(X) \cdot E(Y)$

Stetiger Ereignisraum: Die Elemente in Ω sind nicht abzählbar.

1. Dichtefunktion: $P(a \leq X \leq b) = \int_a^b f(x)\, dx$, mit $\int_{-\infty}^{\infty} f(x)\, dx = 1$ und $P(X = x) = \int_x^x f(x)\, dx = 0$
2. Erwartungswert: $E(X) = \mu = \int_{-\infty}^{\infty} x \cdot f(x)\, dx$
3. Varianz: $Var(X) = \sigma^2 = \int_{-\infty}^{\infty} (x - \mu)^2 \cdot f(x)\, dx = E\left((x - \mu)^2\right)$
4. Kovarianz: $Cov(X, Y) = E\left[(X - E(X)) \cdot (X - E(X))\right] = E\left[(\vec{x} - \vec{\mu}) \cdot (\vec{x} - \vec{\mu})^T\right]$

Für zwei unabhängige Zufallsvariablen X_1 und X_2 gilt:

- $E(X_1 \cdot X_2) = E(X_1) \cdot E(X_2)$
- $Var(X_1 + X_2) = Var(X_1) + Var(X_2)$
- $Cov(X_1, X_2) = E\left[(\vec{x} - \vec{\mu}) \cdot (\vec{x} - \vec{\mu})^T\right] = 0$

Die wichtigste Verteilungsfunktion ist die Normalverteilung nach Gauss: $f(x) = \frac{1}{\sigma\sqrt{2\pi}} \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$

Fehlerfortpflanung

Seien $\vec{x}$ die n Sensorwerte und $\vec{y} = \vec{f}(\vec{x})$ die m Motorwerte als Funktionen der Sensorwerte. Die Sensorwerte seien um ihren eigentlichen Wert Normalverteilt: $x_i = \mu_i + \xi_i$, mit $\mu_i = E(x_i)$, d. h. $E(\xi) = 0$.

Die Frage ist nun, wie sich das Rauschen der Senorwerte auf die Motorwerte auswirkt: $y_j = f_j(\vec{\mu} + \vec{\xi})$

1. Taylor Entwicklung von y_j: $f_j(\vec{\mu} + \vec{\xi}) = f_j(\vec{\mu}) + \sum_{i=1}^{n} (F_x)_{ij} \cdot \xi_i$, wobei $(F_x)_{ij} = \frac{\partial f_j(\vec{x})}{\partial x_i}$, d. h. $F_x = \vec{f} \cdot \left(\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_n}\right)$ ist die Jacobi Matrix.
2. Die Motorwerte können also dargestellt werden als: $\vec{y} = \vec{f}(\vec{\mu}) + F_x \cdot \vec{\xi}$
3. Die Kovarianz der Motorwerte ist gegeben durch: $Cov(\vec{Y}) = E\left[\left(\vec{y} - \vec{f}(\mu)\right) \cdot \left(\vec{y} - \vec{f}(\mu)\right)^T\right] = E\left[\left(F_x \cdot \vec{\xi}\right) \cdot \left(F_x \cdot \vec{\xi}\right)^T\right] = F_x \cdot E\left[\vec{\xi} \cdot \vec{\xi}^T\right] \cdot F_x^T$
4. Nebenüberlegung: Aus $x_i = \mu_i + \xi_i$ folgt $\xi_i = x_i - \mu_i$
5. Aus $Var(X) = E\left[(x_i - \mu)^2\right]$ und der Nebenüberlegung folgt: $Var(X) = E\left[\xi_i^2\right] = E\left[\xi \cdot \xi^T\right] = Cov(\vec{Y})$
6. Damit ergibt sich die Fehlerfortpflanzung mit: $Cov(\vec{y}) = F_x \cdot Cov(\vec{X}) \cdot F_x^T$

1.4 Feature Extraction

Bei Sensordaten gibt es zwei orthogonale Dimensionen: Details und Unterscheidbarkeit. Rohe Sensordaten besitzen sehr viele Details, die Bereiche unterscheiden sich aber nicht. Wenn man Kanten herausfiltert, löscht man Details aber erhöht die Unterscheidbarkeit. Notfallsysteme arbeiten besser mit schnell verfügbaren und hoch detailierten Daten, wohingegen Kartenalgorithmen besser mit bearbeiteten Sensordaten arbeiten. Die Pipline der Sensorverarbeitung sieht wie folgt aus: Sensing -> Signal treatment -> Feature extraction -> Scene interpretation. Bei der Feature Extraction kann man zwischen zwei Arten Features unterscheiden: Low Level (Linien, Kreise) und High Level (Kanten, Türen).

Linien Extraktion aus Entfernungsdaten

Seien die Messpunkte x_i gegeben in Polarkoordinaten $x_i = (\rho_i, \theta_i)$. Die gesuchte Linie ist ebenfalls in Polarkoordinaten gegeben durch die Parameter (r, α), wobei α den Winkel und r die Länge der Geraden angibt, die senkrecht auf der Linie steht und durch den Ursprung geht. Liegen alle Punkte auf der Geraden, dann gilt: $\forall x_i : \rho_i \cos(\theta_i - \alpha) - r = 0$.

Der Fehler durch Abweichungen vom Idealfall ist gegeben durch: $E = \sum_i d_i^2$, mit $d_i = \rho_i \cos(\theta_i - \alpha) - r$. Die optimale Lösung liegt am Tiepunkt des Gradienten E und kann durch Lösen der Gleichungen $\frac{\partial E}{\partial \alpha} = 0$ und $\frac{\partial E}{\partial r} = 0$ gefunden werden.

Probleme:

- Dieses Verfahren berücksichtigt nicht, dass manche Punkte eine größere Ungenauigkeit besitzen als andere (z. B. Punkte am Rande des Sichtfeldes). Dies kann durch das Einführen von Gewichten, z. B. $w_i = \frac{1}{\sigma_i^2}$ erreicht werden, so dass der Fehler durch $E = \sum_i w_i \cdot d_i^2$ gegen ist.
- Die Sensordaten gehören meistens nicht alle zu einem Objekt (Linie, Kreis), daher müssen die Punkte isoliert werden, die zum gleichen Objekt gehören. Dies kann durch Segmentierung oder Entfernungshistogramme erreicht werden.

2 Computer Vision

Computer Vision beschäftigt sich mit der Extraktion von Information aus Bildern. Für Algorithmen sind Bilder zweidimensionale Matrizen. Für den Einsatz in Robotern werden schnelle Verfahren verwendet.

Neben der Farbkodierung in Rot, Grün und Blau (RGB), kann Farbe auch mit Farbton, Sättigung und Helligkeit (HSV) kodiert werden. HSV entspricht der Wahrnehmung von Farbe im Auge, da die Wellenlänge des Lichts direkt im Farbton repräsentiert ist. HSV kann in RGB und umgekehrt transformiert werden.

2.1 Segmentierung

Die wichtigste Anwendung von Computer Vision bei Robotern ist die Segmentierung von Bildregionen:

1. Blende alle Pixel aus, deren Farbwerte nicht in einem gegebenen Intervall liegen.
2. Gruppiere die verbleibenden Pixel
3. Verwerfe alle Pixel die nicht zur größten Gruppe gehören

Mehrfarbige Regionen lassen sich durch Vergleich des Farbhistogramms erkennen:

1. Berechne für ein Ausgangsbild b_0 die Verteilung der Farbwerte in jeder der RGB-Dimensionen. Dabei werden die Farbwerte in gleich große Gruppen eingeteilt.
2. Berechne für jedes neue Bild b_i ebenfalls das Farbhistogramm.
3. b_0 und b_i lassen sich durch Subtraktion der Histogramme einfach vergleichen. Je größer die Differenz ist, desto unterschiedlicher ist die Farbverteilung.

Mit diesem Verfahren lassen sich nur Farbwerte vergleichen, keine Formen.

Räumliches Sehen entsteht durch Stereo Vision. Das Problem ist dabei, in zwei Bildern die gleichen Punkte zu identifizieren:

1. Finde interessanten Punkt p_1 im ersten Bild.
2. Berechne die Übereinstimmung der Bilder.
3. Der Abstand zwischen p_1 und p_2 heißt Disparität $d(p_1, p_2)$.
4. Für den Abstand zwischen Betrachter b und Punkt p gilt: $b \propto \frac{1}{p}$
5. Der Abstand von b zu p läßt sich mit Hilfe der Triangulation berechnen. Wichtig ist dabei der Abstand zwischen den beiden Kameras.

2.2 Objekterkennung

Sei $\chi(x,y) = \begin{cases} 1 & \text{falls } (x,y) \in Obj \\ 0 & \text{sonst} \end{cases}$ die Indikatorfunktion. Die Objekterkennung läuft in zwei Schritten ab:

1. Schwerpunkt berechnen: $x_S = \frac{\sum_{x,y} x \cdot \chi(x,y)}{\sum_{x,y} \chi(x,y)}$ und $y_S = \frac{\sum_{x,y} y \cdot \chi(x,y)}{\sum_{x,y} \chi(x,y)}$
2. Objektdrehung erkennen:
   (a) Lege einen Kreis K_0 um den Schwerpunkt.
   (b) Lege weitere n Kreise K_k gleichmäßig um K_0.
   (c) Berechne die Anzahl Objektpixel in jedem Kreis: $C(k) = \sum_{x,y \in K_k} \chi(x,y)$
   (d) Vergleiche die $C(k)$ mit einer Referenzverteilung $C_R(k)$. Das Objekt wurde um i Kreise gedreht, wenn die Übereinstimmung $\sum_k \left(C(k-i) - C_R(k)\right)^2$ maximal ist.

2.3 Kantendedektion

Betrachten wird die Helligkeit eines Bildes als Funktion f. Eine Kante ist durch eine starke Änderung der Helligkeitswerte gekennzeichnet. Das entspricht einem Hoch- oder Tiefpunkt in f'. Die erste Ableitung soll über die Faltung von f mit einer Maske P berechnet werden. Ein Faltung ist eine Operation, so dass $\widetilde{f} = P \otimes f$:

- Kontinuierlich: $\widetilde{f}(x) = \int_{x' \in D} P(x') \cdot f(x + x') \cdot dx'$
- Diskret: $\widetilde{f}(x) = \sum_{x' \in \mathcal{U}} P(x') \cdot f(x + x')$

Beispiel: Sei $\mathcal{U} = \begin{Bmatrix} (-1,-1), & (0,-1), & (1,-1), \\ (-1,0), & (0,0), & (1,0), \\ (-1,1), & (0,1), & (1,1) \end{Bmatrix}$ und $P = \frac{1}{9} \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$

Laplace Filter

Es wird eine Faltungsmaske im 1-Dimensionalen gesucht, die wie die erste Ableitung wirkt:

1. Entwicklung von $f(x)$ nach Taylor: $f(x+a) = f(x) + a \cdot f'(x) + \frac{a^2}{2} \cdot f'(x)$
2. Sei $a = 1$: $f(x+1) = f(x) + f'(x) + \frac{1}{2} \cdot f''(x)$
3. Sei $a = -1$: $f(x-1) = f(x) - f'(x) + \frac{1}{2} \cdot f''(x)$
4. Addiere 2 und 3 so, dass $f''(x)$ eliminiert wird: $2 \cdot f'(x) = -f(x-1) + f(x+1) \Rightarrow \begin{pmatrix} -1 & 0 & 1 \end{pmatrix}$

Damit erhält man aber nur den Helligkeitsgradienten. Die Hoch- und Tiefpunkte erhält man an den Nullstellen der 2. Ableitung, d. h. wenn $f''(x)$ das Vorzeichen wechselt. Die 2. Ableitung erhält man durch Addition von 2 und 3 so, dass $f'(x)$ eliminiert wird: $f(x-1) - 2 \cdot f(x) + f(x+1) = f''(x) \Rightarrow \begin{pmatrix} 1 & -2 & 1 \end{pmatrix}$

Für 2-dimensionale Anwendungen wird die partielle Ableitung zweiter Ordnung gesucht. Formell definiert der Laplace Operator die 2. Ableitung: $\nabla^2 = \left(\frac{\partial^2}{x^2} + \frac{\partial^2}{y^2}\right)$. Der Faltungskern ist definiert als: $\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$

Der Laplace Filter reagiert auch auf Rauschen. Daher ist es wichtig, dass das Bild vor Anwendung von Laplace geglättet wird.

Glättung

Beim Glätten durch Faltung werden die Nachbarfelder gewichtet zum aktuellen Feld addiert. Eine wichtige Randbedingung für Filter ist, dass sie in einer konstanten Umgebung nichts ändern. D. h. für $f(x) = c \Rightarrow (P \otimes f) = f$ oder $\sum_{x' \in \mathcal{U}} P(x') \cdot f(x+x') = c$. Der Gaußfilter basiert auf der Gaußkurve und erfüllt diese Anforderungen: $P = \frac{1}{16} \begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$

Der Medianfilter beruht nicht auf der Faltung. Statt dessen sortiert er die Helligkeitswerte in einer Umgebung und wählt den mittleren Wert (Median) aus.

Canny

Glättung und Kantendedektion sollen in einem Schritt durchgeführt werden. Sei G der Glättungskern nach Gauß und $I(x, y)$ die Funktion der Helligkeitswerte. Der Canny Algorithmus nutzt nur die erste Ableitung als Kantendedektor: $(G \otimes I)' = G' \otimes I$. Der Algorithmus funktioniert folgender Maßen:

1. Faltungskern für vertikale $f_v = G'(x) \cdot G(y)$ und horizontale Kanten $f_h = G(x) \cdot G'(y)$

2. Bild nach vertikalen $\widetilde{I}_v = f_v \otimes I$ und horizontalen $\widetilde{I}_h = f_h \otimes I$ Kanten filtern
3. Filter zusammenführen: $\widetilde{I}(x,y) = \widetilde{I}_v^2(x,y) + \widetilde{I}_h^2(x,y)$
4. Markiere alle Punkte $\widetilde{I}(x,y) > T$ oberhalb eines Schwellwertes
5. Lösche alle nicht maximalen Kanten (Nonmaxima suppression)

Dynamische Schwellwerte

Da sich die Beleuchtung in realen Umgebungen ändern kann, ist es nicht sinnvoll, einen festen Schwellwert für die Auswahl der Gradienten zu verwenden. Stattdessen könnte man ein Histogramm der Gradienten erstellen und den Gradienten des n größten Wertes als Schwellwert auswählen. Dies beruht auf der Annahme, dass nur die ersten n Werte wirklich relevant sind.

Hough Transformation

Die Hough Transformation findet gerade Linie in dem es jeden Punkt (x,y) für Werte von m und b wählen lässt, so dass $y = m \cdot x + b$. Wähle dazu folgendes Vorgehen:

1. Erstelle eine 2D Matrix $A = M \times B$ mit allen möglichen Wertepaaren für $m \in M$ und $b \in B$ und initialisiere $A[m,b] = 0$.
2. Betrachte für jedes Pixel $(x,y) \in I$ jede Kombination von m und b.
3. Falls die Gleichung $y = m \cdot x + b$ erfüllt ist: $A[m,b] = A[m,b] + 1$
4. Finde alle Zellen in A mit den größten Werten. Sie entsprechen den Geraden in I.

2.4 Optischer Fluss

Die Helligkeit eines Pixels an der Stelle (x,y) zum Zeitpunkt t sei gegeben durch $I(x,y,t)$. Die Veränderung eines Punktes in Raum und Zeit wird angegeben durch dx, dy und dt. Gesucht ist der Geschwindigkeitsvektor $\begin{pmatrix} u \\ v \end{pmatrix}$, mit $u = \frac{dx}{dt}$ und $v = \frac{dy}{dt}$.

Das veränderte Bild ist gegeben durch $I(x+dx, y+dy, t+dt)$. Die Taylorentwicklung dieser Funktion bis zum 1. Glied liefert: $I(x+dx, y+dy, t+dt) = I(x,y,t) + \frac{\partial I}{\partial x} \cdot dx + \frac{\partial I}{\partial y} \cdot dy + \frac{\partial I}{\partial t} \cdot dt$.

Randbedingungen:

1. Die Helligkeit eines Objektpunktes ist konsistent, d. h. sie ändert sich über die Zeit nicht: $I(x+dx, y+dy, t+dt) = I(x,y,t)$

(a) Daraus folgt, dass $\frac{\partial I}{\partial x} \cdot dx + \frac{\partial I}{\partial y} \cdot dy + \frac{\partial I}{\partial t} \cdot dt = 0$

(b) Aus der Definition des Geschwindigkeitsvektors folgt: $dx = u \cdot dt$ und $dy = v \cdot dt$

(c) Setzt man (b) in (a) ein und teilt durch dt, dann erhält man: $\frac{\partial I}{\partial x} \cdot u + \frac{\partial I}{\partial y} \cdot v + \frac{\partial I}{\partial t} = 0$

(d) Abkürzend setze man $I_x = \frac{\partial I}{\partial x}$, $I_y = \frac{\partial I}{\partial y}$, $I_t = \frac{\partial I}{\partial t}$ und erhält damit: $I_x \cdot u + I_y \cdot v + I_t = 0$

2. Die Bewegungen des Bildes sind glatt, d. h. alle Punkte bewegen sich in die gleiche Richtung.

(a) Die Glattheit ist gegeben durch: $S = \int \int_I \left(u_x^2 + u_y^2 + v_x^2 + v_y^2\right) dxdy$

(b) Die Abweichung der Konsistenz ist gegeben durch: $C = \int \int_I \left(I_x \cdot u + I_y \cdot v + I_t\right)^2 dxdy$

(c) u und v sind glatt, wenn die 1. Ableitung von $S + \lambda \cdot C$ klein ist, wobei λ die Abweichung von der Glattheit gewichtet.

Den Optical Flow erhält man nun durch lösen folgender Gleichungen:

$\nabla^2 u = \lambda \left(I_x \cdot u + I_y \cdot v + I_t\right) \cdot I_x$
$\nabla^2 v = \lambda \left(I_x \cdot u + I_y \cdot v + I_t\right) \cdot I_y$

3 Verhalltensbasierte Robotik

In der klassischen KI erfolgt die Verarbeitung der Sensordaten sequentiell: Update des Weltmodells -> Planung -> Aktion. Die Vorhersagen eines solchen Systems sind genau, die Berechnung dauert allerdings lang und benötigt ein exaktes und vollständiges Weltmodell. Dem gegenüber steht heute ein verhaltensbasiertes Modell. Bei diesem findet keine explizite Modellierung der Welt statt, sondern die Sensordaten werden durch parallel laufende Verhaltenseinheiten direkt auf Motordaten abgebildet. Dadurch ist die Reaktionszeit viel geringer, man braucht kein explizites Weltmodell. Dafür lassen sich aber keine Vorhersagen treffen. Eigenschaften verhaltensbasierter Roboter sind:

- Nicht oder nur ungenau modellierbare, reale Umgebungen
- Physische Präsenz des Roboters in der Welt
- Intelligenz entsteht aus der Interaktion von Roboter und Umgebung
- Vermeiden von Symbolverarbeitung
- Ungewiss ob verhaltensbasierte Intelligenz bis zur menschlichen Intelligenz wachsen kann

3.1 Verhaltensmodellierung

- Entwurf basierend auf der Verhaltensforschung: Biologische Modelle tierischen Verhaltens dienen als Triebkraft
- Situationsbasierter Entwurf: Aktionen werden aufgrund der erkannten Situationen ausgeführt, kein biologischer Hintergrund
- Experimenteller Entwurf: ausgehend von einem minimalen System wird in einem Iterativer Prozess das Verhalten verfeinert

Mathematische Modellierung von Verhalten: Aktionen = f(Sensordaten, Releaser)

Zur Darstellung des Roboterverahltens gibt es:

- Stimulus-Antwort-Diagramm
- Funktionale Darstellung
- Erweiterter Endlicher Automat

3.2 Verhaltenskodierung

Seien $B = (\beta_{km})$ die Matrix der Motorwerte aller k Verhaltensbausteine $\beta_k = (y_1, \ldots, y_m)$ und $\vec{g} = (g_1, \ldots, g_k)$ die Gewichte der Verhaltensbausteine. B kann als Funktion der Sensorwerte $\vec{s}$ betrachtet werden: $B(\vec{s})$. Die Koordinationsfunktion C liefert die endgültigen Motorwerte: $\vec{r} = C\,(\vec{g} \cdot B(\vec{s}))$.

Stimulus $s \in S$: $s = (\rho, \lambda)$, mit ρ = Klasse und λ = Stärke. Nicht jeder Stimulus muss Antwort erzeugen. Daher wird Schwellwert τ eingeführt: if $\lambda > \tau$ than generate r

Response $r \in R$: r = (x, y, z, θ, ϕ, ψ), wobei x, y, z die Stärke für die jeweilige Raumrichtung und θ, ϕ, ψ den Winkel für die Rotation um die Achse angibt.

Verhalten wird durch β definiert: $s \mapsto r$, wobei unterschiedliche Antwortkodierungen unterschieden werden:

Null: Stimulus wird verworfen

Diskret: Antwortmenge ist aufzählbar und endlich

Kontinuierlich: Mathematische Funktion liefert Antwort aus kontinuierlichem Wertebereich. Dies ermöglicht den Aufbau von Kraftfeldern um Hinternisse oder Attraktoren im Sichtfeld des Roboters. Das Kraftfeld um ein Hinternis ergibt sich z.B. aus: Kraft $\propto \frac{1}{\text{Entfernung}^2}$

Es wird nicht ein komplettes Kräftefeld berechnet, sondern nur die Kraft, die ein bestimmtest Objekt auf die aktuelle Position des Roboters hat. Die addierten Kräfte ergeben dann die Richtung in die sich der Roboter weiterbewegt.

Es gibt zwei Koordinationsansätze:

1. Konkurrierende Koordinierung: wählt aus den verschiedenen Antwortalternativen, die die Verhaltensbausteine liefern, eine zur Ausführung aus:

   - Feste Prioritäten: Verhalten mit der höchsten Priorität setzt sich durch. Prioritäten sind für die gesamte Laufzeit festgelegt.
   - Aktionsauswahl: Verhalten mit der höchsten Aktivierung setzt sich durch. Die Aktivierung ändert sich mit der Zielsetzung während der Laufzeit.
   - Abstimmung: Jedes Verhalten gibt eine *Stimme* für ein bestimmtes Motorverhalten ab. Das Motorverhalten mit den meisten Stimmen wird ausgeführt. Die Stimmabgabe verändert sich zur Laufzeit je nach Situation.

2. Kooperative Koordinierung: integriert verschiedene Verhaltsblöcke zu einem Verhalten. Dazu erzeugen die unterschiedlichen Verhalten Kraftfelder, die durch Vektoraddition zusammengebracht werden: $r_i = \sum_{i=1}^{k} (g_i \cdot \beta_i(\vec{s}))$

3.3 Architekturen

Bewertungskriterien für Roboterarchitekturen sind:

- Parallelität: Wie stark wird die Parallelität verhaltensbasierter Systeme unterstützt?
- Implementierung: Wie einfach ist die Umsetzung der Architektur in Hardware (Sensoren, PLAs, usw.)? Gibt es bereits Werkzeuge oder ist es ein rein philosophischer Ansatz?
- Spezialisierung: Wie einfach läßt sich die Architektur für Spezialanwendungen anpassen?
- Modularisierung: Ist die Architektur modular? Lassen sich die Module wiederverwenden?
- Robustheit: Reagiert das System stabil auf Ausfälle? Wie Ausfallsicher ist die Architektur?
- Flexibiltät: Wie flexibel ist das System zur Laufzeit? Kann es lernen? Kann es sich anpassen?
- Performanz: Wie gut erfüllt der Roboter die Anforderungen? Kann er die gestellten Aufgaben lösen?

Subsumption

Grundannahme der Subsumption Architketur ist, dass sich Roboter als Körper in einer bestimmten Situation befinden. D. h. sie stehe nicht außerhalb der Situation, sie befinden sich mitten darin und können die Situation über ihre Sensoren erfassen.

Bei der Subsumptions Architektur werden Verhaltensbausteine hierarchisch angeordnet. Die oberen Schichten sind abstrakter als die unteren Schichten und bauen auf diesen auf. Sei z. B. die unterste Schichte *Hinternissen ausweichen*, dann könnte man darauf die Schicht *Herumlaufen* aufbauen. Das Verhalten *Herumlaufen* würde die Aktionen von *Hinternissen ausweichen* auf jeden Fall berücksichtigen. Die unterste Schicht modelliert die Reaktionen des Systems, wohingegen obere Schichten die kognitiven Fähigkeiten modellieren. Die Schichten können parallel voneinander laufen.

Zur Koordination der Verhaltensbausteinen wird ein konkurierendes Verfahren verwendet: Komplexe Aktionen setzen sich aus einfachen Verhalten durch eine hierarchische Struktur zusammen. Prioritäten in der Hierarchie werden durch Hemmung (Inhibition) und Unterdrückung (Suppression) von Signalen realisiert. Inhibitoren hemmen Eingangssignale zu und Suppressionen unterdrücken Ausgangssignale von Verhaltensbausteinen.

Motor Schema

Beim Motor Schema beginnt der Entwurf mit der Erstellung von Motorreaktionen, die zur Lösung der Aufgaben ausgeführt werden. Abstrakte Motorreaktionen werden auf primitivere zurück geführt, bis die Motorreaktionen direkt an Sensoreingaben gebunden sind. Wenn möglich werden dazu Verhaltenswissenschaftliche Untersuchungen herangezogen.

Die untereste Ebene bilden perzeptive Schemata. Sie greifen rekursiv auf andere perzeptive Schemata oder direkt auf Sensordaten zurück. Die perzeptiven Schemata werden in Motorschemata eingebettet und verbinden somit Sensordaten direkt mit Motorreaktionen. Releaser können Verhalten - in diesem Fall Motorschemata - auslösen.

Die Ausgabe von Verhaltensbausteinen sind Vektoren von Potentionalfeldern: $\vec{v} = (m, d)$, mit m ist die Stärke (magnitude) und d ist die Richtung (direction). Die Koordniation der Verhalten erfogt durch Vektoraddition und somit kooperativ: $\rho = \sum \vec{(G)} \otimes \vec{v_i}$, wobei i der Index des Verhaltensbausteins ist. Die Gewichtung G der Verhaltensbausteine hängt von der aktuellen Intention des Roboters und seiner Umgebung ab. Es existiert eine vordefinierte Hierarchie. Während der Laufzeit können auch Motorschemata verworfen und neue erstellt werden. Bevor ρ an die Motoren weitergeleitet wird, muss er normalisiert werden, so dass sich die Werte auch in sinnvolle Motorbewegungen umsetzen lassen. Zuletzt lassen sich mit Hilfe von erweiterten endlichen Automaten die verschiedenen Verhalten sequentialisieren, z. B.: Wandere so lange umher bis Futter entdeckt wurde -> gehe zum Futter -> nimm Futter -> ...

Ein Problem bei der Vektoraddition sind lokale Minima, d. h. Punkte an denen sich die Vektoren zu 0 summieren. Gerät ein Roboter an eine Position mit einem lokalem Minimum, bleibt er immer stehen. Lösungen zu diesem Problem sind:

- Zufallsvektor (Noise) injizieren
- Ein Verhalten zum Vermeiden der Vergangenheit einführen
- Tangentialfelder um Hinternisse einfügen

Sei d der aktuelle Abstand des Roboters zu einem Hinternis, D die maximale Reichweite des Hinternisses und R dessen Radius. Dann ergibt sich die Liste der Kraftfelder für die Verhaltensarten aus:

Verhalten	Stärke	Richtung
Gehe zum Ziel	Konstante	zum Ziel hin
Ausweichen	$\frac{D-d}{D-R} \cdot G$	vom Hinternis weg
Pfad folgen	$\frac{d}{W/2} \cdot G$	90° zum Pfad hin
Vorwärts bewegen	Konstante	gegebene Richtung
Noise	Konstante	zufällig

4 Wissen

Informationen sind Daten mit einer Struktur.

Wissen ist die Fähigkeit Informationen zu nutzen. Dies setzt ein Verständnis, ein Bewußtsein und eine Vertrautheit mit der Welt vorraus.

Wissensrepräsentation ist eine physikalische Struktur mit Korrelation zur Umgebung, die dadurch Vorhersagekraft besitzt. Die Korrelationen mit der Umwelt sind einerseits durch ihre Dauerhaftigkeit und Metrik charakterisiert. Wenn nichts vorhergesagt werden muss, kann die Wissensrepräsentation auf die Sensordaten beschränkt bleiben.

4.1 Wissenstypen

Je schneller sich die Welt ändert, desto schneller veraltet die interne Repräsentation der Welt. Eine ständige Aktualisierung des Weltbilds ist kostenaufwendig. Reine reaktive Systeme haben dagegen kein Weltbild, können aber auch nicht vorausschauend handeln. Wie genau die Welt intern modeliert werden soll, hängt vom Einsatzgebiet und -zweck ab.

Einteilung von Wissen nach Anwendungsort:

Raum Verständnis der räumlichen Umgebung: Wo bin ich?

Objekte Verständnis der Dinge in der Umgebung: Was gibt es?

Wahrnehmung Verständnis der Sensoreigenschaften: Wie nutze ich die Sensoren?

Verhalten Verständnis der Reaktionsmöglichkeiten in verschiedenen Situationen: Was soll ich tun?

Ego Verständnis der eigenen Fähigkeiten und Grenzen: Was kann ich tun?

Intention Verständis der eigenen Ziele: Was will ich?

Einteilung von Wissen nach Dauerhaftigkeit:

Persistentes Wissen: A priori Karten oder Informationen über Objekte, gespeichert im Langzeitgedächtnis (LTM)

Transitorisches Wissen: Karten oder Informationen die Roboter selbst entdeckt hat, gespeichert im Kurzzeitgedächtnis (STM)

Kein Wissen: Reines reaktives System

4.2 Kurzzeitgedächtnis

Im Kurzzeitgedächtnis werden Informationen gespeichert, die wichtig für die Verhaltensbausteine sind. Die Sensordaten werden aufbereitet und als Perzeptionen im Kurzzeitgedächtnis gespeichert. Dort werden sie von den Verhaltensbausteinen abgerufen. Dadurch kann einerseits die Samplerate der Sensoren verringert werden, andererseits kann der Roboter mit Informationen außerhalb seiner Sensorerfassung versorgt werden.

Das Kurzzeitgedächtnis wird als Gitterstruktur realisiert. Diese unterscheiden sich in der Auflösung, Form und Einheitlichkeit: Rechtecke, Sektoren, Quadrate

4.3 Langzeitgedächtnis

Im Langzeitgedächtnis werden Karteninformationen gespeichert. Die Karten können aus Sensordaten des Roboters oder aus extern gesammelten Daten erstellt werden. Letzteres kann während der Laufzeit des Roboters oder a priori erfolgen. A priori erstellte Karten haben den Vorteil, dass man auf Quellen zurückgreifen kann und das System dadurch leichter implementierbar wird.

Es gibt zwei gängie Arten die Karten zu kodieren:

Metrisch Es werden die absoluten Distanzen von Perzeptionen gespeichert. Dazu werden unterschiedliche Sensortypen kombiniert.

Qualitativ Es werden *Landmarken* aus den Sensordaten extrahiert und in Beziehung zueinander gesetzt. Ein Beispiel wäre eine Subsumptions Architektur mit drei Bausteinen: Zielnavigation und Karten lernen, Landmarken Erkennung, Randerkennung.

4.4 Internalisierte Pläne

Internalisierte Pläne sind eine Anwendung von a priori erstellten Karten. Der Raum wird in Gitterzellen zerlegt. Jeder Zelle werden Kosten gemäß den Missionszielen gegeben. Dann wird für die Karte ein Gradientenfeld der Kosten berechnet. Das Gradientenfeld heißt internalisierter Plan, weil es für jeden Punkt im Raum die bevorzugte Richtung angibt.

5 Lokalisation

Lokalisation benötigt explizites Wissen über die Umwelt. Dies widerspricht den Vorgaben der verhaltensbasierten Robotik. Insbesondere müssen mobile Roboter eine Repräsentation ihrer Umwelt erstellen und benutzten: Sensoren -> Lokalisation / Kartographieren -> Planung -> Motorsteuerung. Die explizite Verwaltung von Karteninformationen erleichtert die Interaktion von Roboter und Mensch.

Bei der Lokalisierung wird die vom Roboter berechnete Position durch Belief modelliert. Der Belief ist die Position an der der Roboter zu sein glaubt. Es gibt zwei Arten von Beliefs:

- Einzelhypothesen Belief: Der Roboter glaubt an genau einer Position zu sein. Dies erleichtert die Planung, ermöglicht aber nicht, Unsicherheiten mit in die Berechnung einzubeziehen.
- Multihypothesen Belief: Der Roboter nimmt mehrerer mögliche Positionen an. Diese können entweder alle gleich Wahrscheinlich sein, dann wird die Position z. b. als Polygon modelliert oder die Positionen sind unterschiedlich Wahrscheinlich, d. h. sie folgen einer Wahrscheinlichkeitsverteilung.

5.1 Kartenmodelle

Die Wahl des Modells hängt von den Anforderungen an den Roboter, dem Detailgrad der Sensoren und dem Lokalisationsverfahren ab. Es gibt zwei Klassen von Karten:

Stetige Karten: Definieren die Umwelt (mathematisch) durch Linien, Kreise und Polygone. Dies kann in 2 oder 3 Dimensionen erfolgen, wobei letzteres sehr Platz- und Zeitaufwendig ist. Die Polygone können mit Texturen oder anderen Eigenschaften belegt werden. Für Umgebungen mit vielen Objekten können Karten sehr groß werden. Dagegen brauchen offene Flächen kein Platz.

Diskrete Karten: Die Karten werden nach bestimmten Kriterien in Bereiche zerlegt und diesen Bereichen Eigenschaften zugeordnet. Beispiele sind:

- Zerlegung in vertikale Bereiche: ein Bereich wird durch die Hindernisse und ihre Ecken begrenzt
- Feste Zerlegung in Zellen: jede Zelle die von einem Hindernis ganz oder teilweise belegt ist, wird schwarz markiert, alle freien Zellen weiß
- Adaptive Zerlegung in Zellen: das Gebiet wird in vier gleich große Zellen geteilt. Ist eine Zelle komplett frei oder komplett von einem Hindernis belegt, wird sie entsprechend eingefärbt und bleibt erhalten. Ist eine Zelle nur teilweise von einem Hindernis belegt wird sie weiter in vier gleich große Zellen geteilt. Auf diese Weise wird rekursiv fortgefahren, bis eine maximale Auflösung erreicht ist.

- Belegungsgitter: jedes Feld im Gitter hat einen Zähler der hochgesetzt wird, wenn eine Entfernungsmessung ein Hinderniss für diese Zelle meldet. Ein Wert von 0 bedeutet, dass für diese Zelle noch kein Hindernis gemeldet wurde. Wenn der Zähler über einem Schwellwert liegt, wird die Zelle als Hindernis betrachtet. Über die Zeit hinweg wird der Zähler wieder dekrementiert so, dass Rauschen und Bewegungen erfasst werden.
- Topologische Zerlegung: die Karte wird in Regionen geteilt, die als Knoten dargestellt werden. Benachbarte Regionen werden durch Kanten verbunden. Die Knoten bzw. Kanten werden mit Informationen annotiert.

5.2 Probabilistische Lokalisation

Die Lokalisation besteht aus zwei Schritten:

1. Aktionsupdate: Mit Hilfe der Odometriedaten o_t und der letzten Position s_{t-1} wird eine Position $s'_t = Act(o_t, s_{t-1})$ besimmt. Da die Odometrie sehr fehlerbehaftet ist, vergrößert dieser Schritt die Unsicherheit der Vorhersage.
2. Sensorupdate: Mit Hilfe der Sensordaten i_t soll die Unsicherheit in s'_t beseitigt und eine möglichst genaue Position bestimmt werden: $s_t = See(i_t, s'_t)$.

5.3 Markov Lokalisation

Bei der Markov Lokalisation wird das Modelle der Markov Prozesse mit Bayes'schen Wahrscheinlichkeiten kombiniert:

- Beim Markov Prozess wird eine Welt aus n Zuständen angenommen. Wenn sich die Welt in Zustand s_i befindet, ist $p(s_j \mid s_i)$ die Wahrscheinlichkeit, dass sich die Welt im nächsten Zeitschritt in Zustand s_j befindet.
- Die Bayes'schen Wahrscheinlichkeiten erlauben es bedingte Wahrscheinlichkeiten ineinander umzurechnen, ohne die gemeinsamen Wahrscheinlichkeiten berechnen zu müssen: $p(A \mid B) = \frac{p(B|A) \cdot p(A)}{p(B)}$

Die Wahrscheinlicheit, dass der Roboter an Position l ist, wird kurz $p(l)$ geschrieben. Das Aktions- und Sensorupdate ist wie folgt definiert:

1. Action Update: $l'_t = Act(o_t, l_{t-1}) = p(l_t \mid o_t) = \sum_{l_{t-1} \in L} p(l_t \mid l_{t-1}, o_t) \cdot p(l_{t-1})$, wobei L die Menge aller Positionen ist.
2. Sensor Update: $l_t = See(i_t, l'_t) = p(l \mid i) = \frac{p(i|l'_t) \cdot p(l'_t)}{p(i)}$, wobei $p(i)$ konstant ist und daher häufig weggelassen und durch einen Normierungsschritt ersetzt wird.

Ein Roboter mit Markov Lokalisation kann an einer unbekannten Position auf der Karte starten, da er in jedem Schritt alle Zellen aktualisiert. Das geht aber mit einem hohen Rechenaufwand einher - besonders wenn es sehr viele Zellen gibt. In diesem Fall kann man die Anzahl der betrachteten Zellen durch Gewichtung mit den Wahrscheinlichkeiten eingrenzen, was aber zu Lasten der Orientierung in ambigen Situationen geht. Da der Markov Prozess diskrete Weltzustände d. h. Karten Positionen voraussetzt, kann das Verfahren nur mit diskreten Karten (z. B. topologischen Karten) arbeiten.

5.4 Kalman Filter

Statischer Klaman Filter

Gegeben n Positionsschätzungen x_i mit den Varianzen σ_i^2, wobei $1 \leq i \leq n$. Gesucht ist die Schätzung der Position $\hat{x}$ mit minimalem Fehler.

Mit Hilfe der „Weighted Least Square" Methode soll der Fehler minimiert werden. Dazu wird der Fehler definiert durch: $S = \sum_{i=1}^{n} w_i(\hat{x} - x_i)^2$. Die Nullstelle der Ableitung liefert das Minimum: $\frac{\partial S}{\partial \hat{x}} = 0 \Leftrightarrow 2 \cdot \sum_{i=1}^{n} w_i \cdot (\hat{x} - x_i) = 0$. Durch Umstellen der Summe und auflösen nach $\hat{x}$ erhält man den optimalen Schätzwert der Position: $\hat{x} = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}$. Die Sensorwerte sollen über ihre Genauigkeit gewichtet werden. Die Genauigkeit ergibt sich als inverses der Varianz: $w_i = \frac{1}{\sigma_i^2}$. Nun werden die Messwerte iterativ zur Positionsschätzung herangezogen, dabei ist $\hat{x}_k$ die Positionsschätzung der vergangenen k Schritte:

$$\hat{x}_{k+1} = \hat{x}_k + K_{k+1} \cdot (x_{i+1} - \hat{x}_k)$$

Wobei sich $K_{k+1} = \frac{\sigma_k^2}{\sigma_k^2 + \sigma_{i+1}^2}$ aus obiger Gewichtung ergibt: $\frac{1}{\sigma^2} = \frac{1}{\sigma_k^2} + \frac{1}{\sigma_{i+1}^2} \Leftrightarrow \sigma^2 = \frac{\sigma_k^2 \sigma_{i+1}^2}{\sigma_k^2 + \sigma_{i+1}^2}$. Für die Iteration ergibt sich damit eine Varianz von

$$\sigma_{k+1}^2 = \sigma_k^2 - K_{k+1} \cdot \sigma_k^2$$

Dynamischer Kalman Filter

Angenommen der Roboter bewegt sich zwischen zwei Messungen mit der Geschwindigkeit $\frac{dx}{dt} = v + w$, wobei w der Rauschterm ist und zwischen den Messungen Δt vergeht. Die Lokalisation kann wieder in zwei Schritte geteilt werden:

1. Action Update: $l'_t = Act(o_t, l_{t-1}) = l_{t-1} + v \cdot \Delta t$ und $\sigma_{l'}^2 = \sigma_{t-1}^2 + \sigma_w^2 \cdot \Delta t$
2. Sensor Update: $l_t = See(i_t, l'_t) = l'_t + K_t \cdot (z_t - l'_t)$, mit $K_t = \frac{\sigma_{l'}^2}{\sigma_{l'}^2 + \sigma_z^2}$

Kalman Filter Lokalisation

1. Action Update: $\hat{x}_{t'} = f(\hat{x}_t, v_t)$ mit einem Fehler $\Sigma(\hat{x}_{t'}) = F_x \cdot \Sigma(\hat{x}_t) \cdot F_x^T + F_v \cdot \Sigma(v_t) \cdot F_v^T$

2. Beobachtung: Aus den rohen Sensordaten werden Objekte Z_{t+1} im Koordinatensystem $\{S\}$ des Sensors extrahiert.

3. Schätzung der Messwerte: Aufgrund von $\hat{x}_{t'}$ werden aus der Karten die Objekte M_{t+1} extrahiert, die der Roboter sehen sollte. In M_{t+1} werde die Objekte im Koordinatensystem $\{R\}$ des Roboters angegeben, daher müssen sie erst noch ins Sensorkoordinatensystem überführt werden: $\hat{Z}_{t+1} = h(M_{t+1}, \hat{x}_{t+1})$

4. Matching: Alle Objekte in Z_{t+1} werden Objekten in $\hat{Z}_{t+1}$ zugeordnet. Die Übereinstimmung von z_i mit $\hat{z}_j$ wird durch die Innovation v_{ij}angegeben. Ist die Übereinstimmung zu klein, wird die Zuordnung verworfen.

5. Sensor Update: $\hat{x}_{t+1} = \hat{x}_{t'} + K_{t+1} \cdot v_{t+1}$, wobei in K_{t+1}die Veränderung der Unsicherheit steckt und in v_{t+1} die Innovationswerte aus dem Matching Schritt.

Bei der Lokalisation mit dem Kalman Filter muss der Roboter an einer bekannten Position starten. Da der Filter nur mit den Parametern μ (Position) und σ (Abweichung) arbeitet, ist der Rechenaufwand gering und kann auch mit stetigen Karten genutzt werden.

5.5 Simultaneous Localisation And Mapping (SLAM)

Bei SLAM geht es darum, dass der Roboter parallel zur Lokalisation eine Karte der Umwelt erstellt, bzw. erweitert. Die Kalman Filter Lokalisation kann leicht derart erweitert werden, dass Objekte, die im Matching Schritt nicht gefunden wurden, gelöscht bzw. Objekte die unerwartet gefunden wurden, hinzugefügt werden. Problematisch sind dabei vor allem die Erkennung von Zyklen und die Reaktion auf dynamische Umgebungen.

6 Lernen

Umgangsprachlich meint man mit Lernen zwei Dinge:

- Adaption bezeichnet ein Prozess, bei dem die Parameter eine Systems an die Umwelt angepasst werden. Die führt entweder dazu, dass die Reaktion auf manche Perzeptionen gedämpft wird (Gewöhnung) oder, dass die Reaktion verstärkt wird (Sensibilisierung).
- Lernen beichnet dagegen einen stärkeren Prozess, bei dem auch Parameter verworfen und neue erzeugt werden können.

In der Robotik kennt man folgende Lernverfahren:

- Das Reinforcement Lernen passt die numerischen Parameter durch Belohnung oder Bestrafung der Umwelt an.

- Neuronale Netze sind eine Form des Reinforcement Lernens. Die Parameter sind die Synapsengewichte.
- Evolutionäres Lernen verbessert die Fähigkeiten nachfolgender Generationen durch Kreuzung und Vererbung. Ziel ist es dabei, die Strukturierung des Verhaltens nicht mehr selbst vorzunehmen, sondern von der Evolution erzeugen zu lassen.
- Induktives Lernen nutzt Fallbeispiele, um generalisierte Konzepte zu erstellen.
- Lernen durch Erklärung nutzt explizites Wissen über eine Domäne, um den Lernprozess zu leiten.

Ein Roboter, dessen Verhalten durch $\vec{r} = C\left(\vec{g} \cdot B(\vec{s})\right)$ charakterisiert wird, kann auf folgenden Gebieten lernen:

- B: Verhaltensbausteine verwerfen, neu erzeugen
- $\vec{g}$: Gewichte anpassen
- C: bessere Koordinationsstrategien entwickeln

Lernalgorithmen können numerisch oder symbolisch, induktiv oder deduktiv, online oder offline, beaufsichtigt oder unbeaufsichtigt sein.

6.1 Reinforcement

Unter Reinforcement Lernen versteht man ein Modell mit:

- einer Menge von Weltzuständen S und einer Menge von Aktionen A
- einer Policy $\pi : S \rightarrow A$, die für jeden Zustand die beste Aktion auswählt
- einer Belohnung $r \in \mathbb{R}$ und einer Wertefunktion $V : S \rightarrow \mathbb{R}$, die jedem Zustand die zu erwartetende Belohnung zuweist

Dem Reinforcement Lernen liegt ein Markov-Entscheidungsprozess zugrunde. D. h. die Wahrscheinlichkeit, dass in Zustand s_t die Aktion a zum Folgezustand s_{t+1} führt ist $P(s_{t+1} \mid s_t, a)$ und damit unabhängig von allen vorhergehenden Zuständen.

Zur Zeit t befindet sich die Welt in Zustand $s_t \in S$. Die Menge der verfügbaren Aktionen ist $\mathcal{A}(s_t) : S \rightarrow A$. Nach ausführen einer beliebigen Aktion $a \in \mathcal{A}(s_t)$ befindet sich die Welt im Zustand s_{t+1} und der Roboter erhält eine Belohnung r_{t+1}. Ziel des Roboters ist es nun, eine Policy $\pi : S \rightarrow A$ zu entwickeln, die die Belohnung $R = \sum_{t=0}^{\infty} \lambda \cdot r_t$ maximiert. Um die richtigen Entscheidungen treffen zu können, muss der Wert jeden Zustands bekannt sein. Dies ist a priori nicht der Fall, weswegen V ebenfalls gelernt werden muss.

Wertefunktion:

- $V(s) : s \mapsto \sum_{k=0}^{\infty} \lambda^k \cdot r(s^{k+1})$, wobei $r(s^k$ die Belohnung im k. Folgezustand und λ ein Faktor für die Abwertung nachfolgender Belohnungen ist. Sei $\lambda^k = e^{-k \cdot \tau}$, dann ist $\tau = -\ln \lambda$ der Zeithorizont, bis zu dem Belohnungen etwas zum Wert des Zustandes s beitragen.
- $V(s) : s \mapsto \sum_{s'} P(s) \cdot (r(s') + \lambda \cdot V(s')) = E[R \mid s, \pi]$

Es gibt zwei Arten von Policies:

- Deterministische Policy: $\pi(s_t) = \arg\max_a V(s_{t+1})$, mit einem Fehler $E = (r(s_{t+1}) + \gamma \cdot V(s_{t+1}) - V(s_t))^2$. Da der Folgezustand durch die Policy deterministisch vorgegeben ist, ist der Wert eines Zustandes eindeutig bestimmt: $V(s) = r_{s'} + \lambda \cdot V(s')$. Gibt es einen Zielzustand t, dann nimmt der Wert eines Zustands s exponentiell mit der Anzahl Schritte K_s ab, die benötigt werden um t von s aus zu erreichen: $V(s) = \lambda^{K_s - 1}$
- Stochastische Policy: $\pi(s_t) = \begin{cases} \arg\max_a V(s_{t+1}) & \text{, mit } p \\ a \in \mathcal{A}(s_t) & \text{, mit } 1-p \end{cases}$. Statt einer festen Wahrscheinlichkeit p, kann man auch die Bolzmannverteilung ansetzen: $p_s(a) = \frac{e^{V(s'_a)/T}}{\sum_{s'} e^{V(s')/T}}$, d. h. die Wahrscheinlichkeit, dass in Zustand s Aktion a ausgewählt wird.

Betrachtet man das Gebirge der Werte V über den Zuständen $S \times S$, dann hat bei optimaler Policy der kürzeste Weg von allen Zuständen zum Ziel auch den steilsten Aufstieg.

Die Wertefunktion V läßt sich allgemein mit folgendem Algorithmus lernen:

1. Wertefunktion initialisieren: $\forall s : V(s) = 0$ oder $\forall s : V(s) = \text{random}$
2. Auswahl der besten Aktion: $a_t = \pi(s_t)$
3. Ausführen von a_t
4. Update: $\Delta V(s_t) = \epsilon \cdot [r(s_{t+1}) + \gamma \cdot V(s_{t+1}) - V(s_t)]$, wobei s_{t+1} der neue Zustand ist
5. ϵ abkühlen; $t = t + 1$; GOTO 2

6.2 Q-Lernen

Der Reinforcement Lernalgorithmus hat den Nachteil, dass ein Weltmodell existieren muss, das den Folgezustand einer Aktion kennt, d. h. $s_{t+1} = \Phi(s_t, a)$. Beim Q-Lernen werden daher die Aktionen zusammen mit ihren Ausgangszuständen durch die Q-Funktion gelernt: $Q(s_t, a_t) = r(s_t) + \sum_{k=1}^{\infty} \gamma^k \cdot r(s_{t+k})$.

Beim Q-Lernen kommt eine stochastische Policy zum Einsatz: $\pi(s_t) = \begin{cases} \arg\max_a Q(s_t, a) & \text{, mit Wahrschein} \\ a \text{ beliebig} & \text{, mit Wahrschein} \end{cases}$

oder alternativ nach der Bolzmann Verteilung: $p_s(a) = \frac{e^{Q(s,a)/T}}{\sum_{a'} e^{Q(s,a')/T}}$

Der mittlere Fehler beim Abschätzen von $Q(s,a)$ ist $E = \frac{1}{2}\left(\hat{Q}(s,a) - Q(s,a)\right) = \frac{1}{2}\left([r(s'_a) + \gamma \cdot E(Q(s,a))] - Q(s,a)\right)$ Wenn man anstatt des Erwartungswertes $\langle Q(s',a')\rangle$, den größten Wert $max_{a'}Q(s',a')$ nimmt, beschleunigt man das Lernen. Trotz stochastischer Aktionsauswahl liefert $a = \arg\max_{a'} Q(s,a')$ eine deterministische Policy.

Algorithmus:

1. Init: $\forall s,a : Q(s,a) = 0$ oder $\forall s,a : Q(s,a) = \text{random}$
2. Sensoren erfassen Weltzustand s_t
3. Auswahl von $a_t = \pi(s) = \begin{cases} \arg\max_a Q(s,a), \text{ mit Wahrscheinlichkeit } p \\ a \text{ beliebig, mit Wahrscheinlichkeit } 1-p \end{cases}$
4. Update: $\Delta Q(s_t,a_t) = \epsilon \cdot [r(s_{t+1}) + \gamma \cdot V(s_{t+1}) - Q(s_t,a_t)]$, wobei $V(s_{t+1}) = \max_{a'} Q(s_{t+1},a_{t+1})$ der Wert des Folgezustandes ist
5. GOTO 2

Optimierungen

- Alle Zustände, die ähnlich sind wie s werden beim Update mit berücksichtigt: $\forall t \in \text{SIMILAR}(s) :$ Update $Q(t,a)$
- Direkte Wertiteration: Statt Zustands-Aktionspaaren werden die Wertefunktion $V(s)$ und die Wahrscheinlichkeiten $p_a(s)$ direkt geschätzt.
  - Update $V(s)$: $\Delta V(s) = \frac{1}{\tau}\left(r(s') + \gamma \cdot V(s') - V(s)\right)$
  - Update $p_a(s)$: $\Delta p_a(s) = \frac{1}{K \cdot \tau}\left(r(s') + \gamma \cdot V(s') - V(s)\right)$

  Wobei $\frac{1}{\tau}$ die Lernrate und K eine Konstante ist. Dieser Ansatz hat erstens den Vorteil, dass die Explorationsrate selbst eingestellt wird und damit eine Abkühlungsstrategie überflüssig wurde. Und zweitens kann Vorwissen durch die Initialisierung der Zustände und der Wahrscheinlichkeiten einfließen.
- Shaped Reinforcement
  - Zustandsraum = Verhalten x Bedingungen
  - Reinforcement = Heterogenes Reinforcement + Fortschrittsschätzer, wobei
    Heterogenes R.: Rückmeldung verschiedener interner und externer Sensoren koppeln und
    Fortschrittsschätzer: Metrische Schätzfunktion

6.3 Neuronale Netze

Eines der ersten Neuronen Modelle war das McCulloch Pitts Neuron. Es besteht aus n binären Eingangssignalen x_i, die erregend oder hemmend sein können und einem Schwellwert s. Die erregenden Eingangssignale haben Indizes in E, die hemmenden in H. Das binäre Ausgangssignal $y = \begin{cases} 1 & \text{if } \sum_{i \in E} x_i > s \ \nexists x_i \in H : x_i = 1 \\ 0 & \text{else} \end{cases}$.
Diese binäre Neuronen werden z. B. beim Hopfield Netz eingesetzt. Hopfield Netze nutzen eine Signalkodierung (0,1) oder (-1,1). Als Input bekommen die Neuronen den Output jedes anderen Neurons im Netz und einen externen Input. Ein Hopfield Netz wird duch seinen Energiezustand charakterisiert: $E = -\frac{1}{2} \sum_{i \neq j} w_{ij} \cdot s_i \cdot s_j + \xi$. Die Dynamik des Netzes konvergiert zu einem Energieminimum. Das Erlernen von Mustern entspricht dem Einstellen eines gewünschten Energieminimums bei Eingabe eines leicht verfälschten Musters.

Ein anderes Neuronen Modell ist das Perzeptron. Es besteht aus n reele Eingangssignale $\vec{x}$ mit Gewichten $\vec{w}$ und einem Schwellwert s. Die Eingangssignale werden gewichtet aufsummiert: $z = \vec{x} \cdot \vec{w}$. Das Ausgangssignal $y = g(z)$ wird von der Transferfunktion $g(z)$ und dem Schwellwert s bestimmt. Die Gewichte des Neurons ändern sich gemäß der Hepp'sche Lernregel: $\Delta w_i = -\epsilon \cdot (y - y^{soll}) \cdot g'(z) \cdot x_i$.

Sei der Fehler des Neurons gegebn durch $E = \frac{1}{2} \cdot \left(y - y^{soll}\right)^2$. Dieser Fehler soll minimiert werden. Dies entspricht einem Gradientenabstieg in der Fehlerlandschaft: $\Delta w_i = -\epsilon \cdot \frac{\partial E}{\partial w_i} = -\epsilon \cdot (y - y^{soll}) \cdot \frac{\partial y}{\partial w_i} = -\epsilon \cdot (y - y^{soll}) \cdot g'(z) \cdot x_i$.

Bei einlagigen Neuronalen Netzen erfolgt das Lernen gemäß dieser Lernregel. Mehrlagige Neuronale Netze verwenden zum Lernen den Back-Propagation Algorithmus:

Das Netz habe L Schichten, wobei $y^{(l)}$ die Ausgabe der l. Schicht sei. Insbesondere sei $x_i = y_i^{(0)}$ und $y_k^{Netz} = y_k^{(L)}$. Die Transferfunktion ist definiert als $y_i^{(l)} = g(z_i^{(l)})$, mit $z_i^{(l)} = \vec{y}^{(l-1)} \cdot \vec{w}_i^{(l)}$. Als Lernregel folgt: $\Delta w_{rs}^{(l)} = -\epsilon \cdot \delta_r^{(l)} \cdot y_s^{(l-1)}$, mit $\delta_j^{(l)} = \sum_i \delta_i^{(l+1)} \cdot w_{ij} \cdot g'\left(z_j^{(l)}\right)$.

6.4 Vektorquantifizierer

Ziel ist es, einen Eingaberaum mit Dimension n auf einen Ausgaberaum der Dimension $m < n$ abzubilden. Dazu werden m Codebuchvektoren $\vec{w}_i$ im Eingaberaum verteilt. Im Eingaberaum muss eine Abstandsfunktion $D(\vec{x}, \vec{y})$ definiert sein, z. B. der euklidische Abstand $D(\vec{x}, \vec{y}) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$. Ein Codebuchvektor $\vec{w}_i$ repräsentiert alle Eingaben $\vec{x}$, denen er am nächsten ist, d. h. $\forall \vec{w_j} : D(\vec{x}, \vec{w_i}) \leq D(\vec{x}, \vec{w_j})$, wobei $1 \leq i, j \leq m$ die Indizes der Codebuchvektoren sind.

- Unter einer Voronoi Zelle versteht man den Einzugsbereich eines Codebuchvektors: $V_i = \{\vec{x} \mid \forall j : D(\vec{w}_i, \vec{x}) \leq D(\vec{w}_j, \vec{x})\}$
- Eine Voronoi Parkettierung liegt vor, wenn der gesamte Raum in Vornoi Zellen eingeteilt wurde.

- Adaptiver Vektorquantifizierer: Die Zuordnung von Input zu Codebuchvektor wird als Klassifikationsproblem interpretiert.

  1. Init: Die Codebuchvektoren w_i werden zufällig im Raum verteilt
  2. Wettbewerb: Eine Eingabe $\vec{x}$ wird präsentiert und der Codebuchvektor i ermittelt für den gilt: $\vec{x} \in V_i$
  3. Adaption: Der Codebuchvektor i bewegt sich auf den Input zu: $\Delta\vec{w}_i = \begin{cases} \epsilon \cdot (\vec{x} - \vec{w}_i) & \text{richtig} \\ 0 & \text{falsch} \end{cases}$

  Um den Lernprozess zu beschleunigen und alle Neuronen zu nutzen, sollte auch bei falscher Klassifikation gelernt werden: $\Delta\vec{w}_i = \begin{cases} \epsilon \cdot (\vec{x} - \vec{w}_i) & \text{richtig} \\ -\epsilon \cdot (\vec{x} - \vec{w}_i) & \text{falsch} \end{cases}$

- Self Organizing Maps (SOM/Kohonen): Jedes Neuron wird in eine topologische Struktur eingebunden, die von vornherein festgelegt ist. Die Postion von Neuron i im Gitternetz des Ausgaberaumes wird durch den Vektor $\vec{r}_i$ angegeben.

  1. Init: Verteile Neuronengewichte $\vec{w}_i$ zufällig im Eingaberaum
  2. Wettbewerb: $i = \arg\min_j D(\vec{w}_j, \vec{x})$
  3. Adaption: Alle Neuronen in der Nachbarschaft des Siegerneuron lernen mit. Die Distanz zweier Neuronen in einem 2-dimensionalen Ausgaberaum ist definiert als: $d\left(\vec{a}, \vec{b}\right) = \sqrt{\sum_{k=1}^{2}(a_k - b_k)^2}$. Der Lernschritt ergibt sich nun als: $\forall j : \Delta\vec{w}_j = \epsilon \cdot h_{ij} \cdot (\vec{x} - \vec{w}_j)$, wobei $h_{ij} = \exp\left(-\frac{d(\vec{r}_i, \vec{r}_j)^2}{2 \cdot \sigma^2}\right)$ die Nachbarschaftsfunktion mit Reichweite σ ist.

- Neural Gas (Th. Martinetz): Beim Neural Gas wird die Zieltopologie mitgelernt und ist nicht wie bei SOM vorgegeben.

  1. Init: Verteile Neuronengewichte zufällig im Eingaberaum
  2. Wettbewerb: Es wird eine Rangfolge $R = \{i_1, \ldots, i_m\}$ erstellt so, dass $D(\vec{w}_{i_1}, \vec{x}) \leq \ldots \leq D(\vec{w}_{i_m})$
  3. Lernschritt: $\forall i : \Delta\vec{w}_i = \epsilon \cdot h_i \cdot (\vec{x} - \vec{w}_i)$, wobei $h_i = \exp\left(\frac{r_i}{\lambda}\right)$ die Nachbarschaftsfunktion und λ die Mitlernstärke ist.
  4. Topologie aufbauen:
     (a) Verbinde i_1 mit i_2 und setze das Alter der Verbindung auf 0
     (b) Erhöhe das Alter aller anderen Verbindungen um 1
     (c) Falls eine Verbindung zu alt ist, lösche sie

6.5 Pfad Planung

1. Straßenkarten: In die Freiräume zwischen den Hindernissen werden Straßen gelegt. Ausgangs- und Zielpunkt stellen selbst Knoten dar und werden mit den nächsten Knoten verbunden. Auf dem so enstandenen Graphen kann mit gängigen Mittel der kürzeste Pfad ermittelt werden.

- Visibility Graph: Die Straßen verbinden die Kanten der Hindernisse in Sichtweite. Dadurch führen die Pfade sehr nahe an die Hindernisse heran, was einerseits die Pfadlänge minimiert, andererseits aber gefahren für den Roboter birgt.
- Voronoi Diagram: Die Straßen verlaufen an den Kanten des maximalen Abstands zwischen den Hindernissen. So wie bei der Voronoi Zelle werden auch hier alle Punkte im Raum dem nächsten Hindernis zugeordnet. Die Kanten, an denen die Punkte im Raum von allen Hindernissen gleich weit entfernt liegen, sind die Straßen. Dieses Verfahren kann sehr einfach auf Sensorebene implementiert werden, denn es maximiert die Anzahl minimaler Abstände zu den Hindernissen. D. h. der Roboter versucht von allen Sensoren möglichst minimale Entfernungsdaten zu bekommen. Der Nachteil des Verfahrens ist, dass es die Pfadlänge nicht minimiert.

2. Zellular Zerlegung: Der Raum wird in freie und belegte Bereiche zerlegt. Die Pfadplanung erfolgt über benachbarte freie Bereiche.

   - Bei der exakten Zerlegung bleibt die Struktur des Raumes erhalten. Z. B. wenn der Raum durch vertikale Trennung an den Kanten der Hindernisse zerlegt wird. Die Freiräume werden als Knoten interpretiert und benachbarte Freiräume durch Kanten verbunden. Die Pfadplanung erfolgt dann wieder durch Minimierung der Pfadkosten im Graphen. Bei sehr vielen Hindernissen wird der Raum starkt fragmentiert und die Pfadplanung wird Zeit- und Speicherintensiv.
   - Bei der approximativen Zerlegung wird der Raum in ein vorgegebenes Gitternetz zerlegt. Zellen die von einem Hindernis ganz oder teilweise belegt sind, werden (schwarz) markiert. Die Pfadplanung kann sehr effizient mit dem „Grassfire" Algorithmus erfolgen. Er nutzt eine wellenartige Ausbreitung von der Zielzelle, wobei in jeder Zelle der Abstand zum Ziel gespeichert wird. Wenn die Welle den Ausgangsknoten erreicht, ergibt sich der Pfad als Gradient, der die Abstände zum Ziel mit jedem Schritt minimiert.

3. Potentialfeld: Der Roboter ist ein Punkt auf den die Kräfte eines Potentialfeldes wirken. Dies setzt sich aus attraktiven und abstoßenden Kräften zusammen. Der Pfad entspricht einem Gradienten, der am Zielpunkt sein Minimum hat. Darüber hinaus dient das Potentialfeld nicht nur der Pfadplanung, sondern kann auch zur Steuerung des Roboters genutzt werden, da das Potentialfeld zu jedem Zeitpunkt die Richtung der Bewegung angibt. Sei die Kraft des Potentialfeldes an Punkt $q = \begin{pmatrix} x \\ y \end{pmatrix}$ gegeben als $F(q) = -\nabla U(q)$, wobei $\nabla U = \begin{pmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{pmatrix}$. Das Potiential $U(q) = U_{att}(q) + U_{rep}(q)$ setzt sich aus attraktiven und abstoßenden Potentialen zusammen. Die resultierende Kraft $F(q) = -\nabla \left(U_{att}(q) + U_{rep}(q) \right)$. Sei im Folgenden $\rho_{obj}(q) = \parallel q - q_{obj} \parallel$ die euklidische Distanz von q zu dem Objekt. Die wirkenden Kräfte ergeben sich aus den jeweiligen Potentialen:

   (a) Attraktives Potential: $U_{att}(q) = \frac{1}{2} k_{att} \cdot \rho^2_{goal}(q)$, woraus eine attraktive Kraft $F_{att}(q) = -\nabla U_{att}(q) = -k_{att} \cdot (q - q_{goal})$ resultiert.

(b) Abstoßendes Potential: $U_{rep}(q) = \begin{cases} \frac{1}{2} \cdot k_{rep} \cdot \left(\frac{1}{\rho(q)} - \frac{1}{\rho_o}\right)^2 & \text{if } \rho(q) \leq \rho_0 \\ 0 & \text{if } \rho(q) > \rho_0 \end{cases}$,
wobei ρ_0 die Reichweite des Hindernisses ist. Die daraus resultierende Kraft ist: $F_{rep}(q) = \begin{cases} k_{rep} \cdot \left(\frac{1}{\rho(q)} - \frac{1}{\rho_o}\right) \cdot \frac{q - q_{ob}}{\rho^3(q)} & \text{if } \rho(q) \leq \rho_0 \\ 0 & \text{if } \rho(q) > \rho_0 \end{cases}$

Diese Fälle betrachten Hindernisse als Punkte. Für reale Hindernisse muss der Rand mit vielen abstoßenden Punkten belegt werden. Außerdem gibt es Probleme:

- bei konkaven Objekten, da der Roboter sich im Zwischenraum verfangen kann und dann nur noch hin und her fährt
- beim Folgen einer Wand, da der Roboter von allen Seiten abstoßenden Kräften ausgesetzt ist.

Eine Lösung ist das Einführen von rotierenden Potentialfeldern, die den Winkel des Roboters zum Hindernis berücksichtigen.

Ein weiteres Problem der Potentialfeld Methode in ihrer reinen Form ist, dass sie nur aktuelle Sensordaten auswertet. Sind diese fehlerhaft, kann die Robotersteuerung versagen. Durch Vektorfeld Histogramme werden Informationen über Hindernisse in einem Kurzzeitgedächtnis gespeichert. Dieses Kurzzeitgedächtnis hat die Form eines Gitters, in dem dem Hindernisse der näheren Umgebung eingezeichnet werden und durch neue Sensorinformationen aktuell gehalten werden. Damit lassen sich kurzzeitige Störungen der Sensoren überwinden.

7 Motor Steuerung

Für die Motorsteuerung werden zwei Koordinatensysteme genutzt:

1. Im globalen Koordinatensystem I wird die Position absolut durch drei Parameter bestimmt $(x, y, \theta)_I$, wobei x und y die Koordinaten der Ebene und θ die Drehung des Roboters beschreibt.
2. Das lokale Koordinatensystem R hat seinen Ursprung im Achsenmittelpunkt des Roboters und die x-Achse zeigt in seine Blickrichtung. θ entspricht in diesem Zusammenhang dem Winkel zwischen globaler und lokaler x-Achse.

7.1 Kinematik

Die Kinematik ist die Lehre von der Bewegung. Es geht also darum, ein Bewegungsmodell für Roboter zu erstellen. Die Bewegung des Roboter im lokalen Koordinatensystem ist gegeben durch $\vec{v}_R$. Sei v_i die Geschwindigkeit des

Rades i und r der Abstand der Räder zum Achsenmittelpunkt. Da sich normale Räder nur in x-Richtung bewegen können, ist die Geschwindigkeit in y-Richtung 0. Aus diesen Vorraussetzungen ergibt sich eine Geschwindigkeit von $\vec{v}_R = \begin{pmatrix} \frac{v_r+v_l}{2} \\ 0 \\ \frac{v_r-v_l}{2\cdot r} \end{pmatrix}_R$.

Den Richtungsvektor im globalen Koordinatensystem erhält man durch Rotation im Uhrzeigersinn um die z Achse. Dazu wird die inversen Rotationsmatrix verwendet: $R^{-1}(\theta) = \begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$. Ins globale Koordinatensystem transformiert ergibt das: $\vec{v}_I = R^{-1}(\theta) \cdot \vec{v}_R = \begin{pmatrix} \frac{v_r+v_l}{2} \cdot \cos\left(\frac{v_r-v_l}{2\cdot r}\right) \\ \frac{v_r+v_l}{2} \cdot \sin\left(\frac{v_r-v_l}{2\cdot r}\right) \\ \frac{v_r-v_l}{2\cdot r} \end{pmatrix}$.

Standardräder können sich nur entlang der x-Achse auf der Radebene bewegen. Das entspricht einem Freiheitsgrad (DDOF). Mathematisch wird diese Eigenschaft in zwei Bedingungen formuliert:

1. Rollbedingung: $\begin{pmatrix} \sin(\alpha+\beta) & -\cos(\alpha+\beta) & -r\cdot\cos(\beta) \end{pmatrix} \cdot \vec{v}_R = v_{rad}$
2. Rutschbedingung: $\begin{pmatrix} \cos(\alpha+\beta) & \sin(\alpha+\beta) & r\cdot\sin(\beta) \end{pmatrix} \cdot \vec{v}_R = 0$

Ein holonomer Roboter hat nur holonomen Randbedingungen. Holonome Randbedingungen sind Funktionen, die nur von der Position des Roboters abhängen. Nicht holonome Randbedingungen hängen auch von Ableitungen der Position, z. B. der Geschwindigkeit ab ($v = \dot{s} = \frac{ds}{dt}$). Außerdem lassen sich diese Differentialterme nicht durch Integration entfernen. Ein Roboter mit zwei Rädern und Differentalantrieb ist nicht holonome, da die Bedingung den Geschwindigkeitsvektor benötigt. Ein Fahrrad ohne Lenkung ist dagegen holonom. Die Rutschbedingung kann einfach definiert werden als $\{y = 0, \theta = 0\}$ und gleiches gilt für die Rollbedingung.

Holonomie kann auch über die Freiheitsgrade des Roboters definiert werden. Seien $DDOF$ der Freiheitsgrad der Räder und DOF der Freiheitsgrad auf der Arbeitsfläche. Wenn $DDOF = DOF$, dann braucht der Roboter nur holonome Randbedingungen. Ist darüber hinaus $DDOF = 3$, dann spricht man von einem omnidirektionalen Roboter.

7.2 Motor Steuerung

Open Loop

Ausgehende von der Startposition $\vec{p}_s$ soll eine Trajektorie (Orts- oder Geschwindigkeitsverlauf über die Zeit) berechnet werden, der den Roboter in die Zielposition $\vec{p}_z$ bringt. Zum Berechnen der Trajektorie wird der Weg in kontinuierliche Linien- und Kreissegmente zerlegt und für jedes Segment eine Trajektorie berechnet. Dabei müssen die kinematischen Randbedingungen des Roboter beachtet werden. Bei der Open Loop Steuerung werden Veränderungen der Umgebung über die Zeit nicht berücksichtigt (kein Sensorfeedback).

Closed Loop

Statt den kompletten Weg vorzuberechnen werden Zwischenziele definiert. Beim Erreichen eines Zwischenziels, wird die Umgebung auf Veränderungen geprüft und demgemäß eine Trajektorie zum nächsten Zwischenziel berechnet. Formal ist die Berechnung so definiert:

Der Ziepunkt $\vec{e}_R$ sei im lokalen Koordinatensystem gegeben und wird als Fehler interpretiert. Ziel des Roboters ist es, durch Regelung seiner Vorwärtsgeschwindigkeit $v(t)$ und seiner Drehgeschwindigkeit $\omega(t)$ den Fehler $\lim_{t\to\infty} \vec{e}_R(t) = 0$ gegen 0 zu bringen.

Ohne Einschränkung der Allgemeinheit sei $\vec{e}_I = \begin{pmatrix} 0 \\ 0 \\ \theta \end{pmatrix}$ im Ursprung des globalen Koordinatensystems. Die Geschwindigkeit des Roboters sei gegeben durch $\begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix}_I = \begin{pmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix}_R = \begin{pmatrix} v \cdot \cos(\theta) \\ v \cdot \sin(\theta) \\ \omega \end{pmatrix}$. Weiterhin sei der Achsenmittelpunkt P_I mit $\vec{e}_I$ durch den Vektor $\vec{c}_I$ verbunden. Der Winkel zwischen $\vec{c}_I$ und X_R sei α und zwischen $\vec{c}_I$ und X_I sei er β. Die Länge von $\vec{c}_I$ sei $\rho = \sqrt{x^2 + y^2}$. Daraus ergibt sich ein Geschwindigkeitsvektor in Polarkoordinaten: $\begin{pmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{pmatrix} = \begin{pmatrix} -\cos(\alpha) & 0 \\ \frac{\sin(\alpha)}{\rho} & -1 \\ -\frac{\sin(\alpha)}{\rho} & 0 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix}$, falls $\alpha \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right]$. Wenn der Winkel größer ist, setzte $v = -v$. Die Steuerung des Roboters erfolgt über die Regel $\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} k_\rho \cdot \rho \\ k_\alpha \cdot \alpha + k_\beta \cdot \beta \end{pmatrix}$, wobei $k_\rho, k_\alpha, k_\beta$ Konstanten sind.

Der Geschwindigkeitsvektor hat eine Sigularität bei $\rho = 0$, d. h. wenn der Roboter sein Ziel erreicht. Durch Multiplikation mit der Steuerregel verschwindet diese Singularität aber: $\begin{pmatrix} \dot{\rho} \\ \dot{\alpha} \\ \dot{\beta} \end{pmatrix} = \begin{pmatrix} -k_\rho \cdot \rho \cdot \cos(\alpha) \\ k_\rho \cdot \sin(\alpha) - k_\alpha \cdot \alpha - k_\beta \cdot \beta \\ -k_\rho \cdot \sin(\alpha) \end{pmatrix}$. Außerdem muss gelten: $\alpha, \beta \in [-\pi, \pi]$ und wenn der Roboter einmal rückwärts fährt, fährt er immer rückwärts und ändert die Richtung nicht.

7.3 Odometrie

Die Odometrie schätzt die Position eines Roboters aufgrund seiner Motorwerte. Dieses Verfahren ist sehr fehleranfällig (Rutschen, Abnutzung der Reifen) und die Fehler häufen sich an. Eine teuere alternative zur Odometrie wären einfache oder differenzielle Global Positioning Systeme (GPS).

Seien die Geschwindigkeiten des linken v_l und rechten v_r Rades bekannt, dann ergibt sich eine lokaler Richtungsvektor mit $\vec{d}_R = \begin{pmatrix} \frac{v_r + v_l}{2} \\ 0 \\ \frac{v_r - v_l}{2 \cdot l} \end{pmatrix}$, wobei l der Abstand der Räder zum Achsenmittelpunkt ist. Außerdem kann sich der Roboter mit Standardrädern nie seitwärts bewegen.

Die Bewegung im globalen Koordinatensystem erhält man durch: $\vec{d}_I = R(\theta) \cdot \vec{d}_R$.

Fehlermodell

Die Bewegung im letzten Zeitintervall sei gegeben durch: $(\Delta x, \Delta y, \Delta\theta)$, wobei

- die zurückgelegte Strecke: $\Delta s = \frac{\Delta s_r + \Delta s_l}{2}$
- der Drehwinkel: $\Delta\theta = \frac{\Delta s_r - \Delta s_l}{2 \cdot l}$
- die Bewegung in x-Richtung: $\Delta x = \Delta s \cdot \cos\left(\theta + \frac{\Delta\theta}{2}\right)$
- die Bewegung in y-Richtung: $\Delta y = \Delta s \cdot \sin\left(\theta + \frac{\Delta\theta}{2}\right)$

Im globalen Koordinatensystem ergibt sich somit eine Positionsänderung von: $\vec{p}_{t+1} = f(x, y, \theta, \Delta s_r, \Delta s_l = \vec{p}_t + \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta\theta \end{pmatrix}$.

Der Fehler durch die Bewegung sei durch die Covarianzmatrix $\Sigma_s = Cov(\Delta s_r, \Delta s_l) = \begin{pmatrix} k_r \cdot \mid \Delta s_r \mid & 0 \\ 0 & k_l \cdot \mid \Delta s_l \mid \end{pmatrix}$ gegeben, wobei k_r und k_l feste Fehlerkonstanten sind. Bei diesem Ansatz wird angenommen, dass die Fehler für das linke und das rechte Rad unabhängig voneinander sind und der Fehler proportional zur zurückgelegten Gesamtstrecke ist.

Jetzt wird das Fehlerfortpflanzungsgesetz von oben angewendet: $\Sigma_{p_{t+1}} = F_p \cdot \Sigma_p \cdot F_p^T + F_s \cdot \Sigma_s \cdot F_s^T$, mit $F_p = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial \theta} \end{pmatrix}$ und $F_s = \begin{pmatrix} \frac{\partial f}{\partial \Delta s_r} \\ \frac{\partial f}{\partial \Delta s_l} \end{pmatrix}$.