

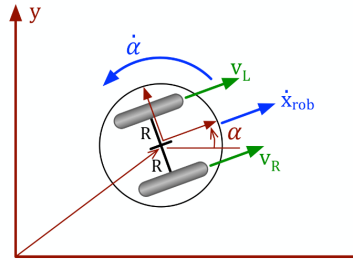
Robotik und Mechatronik 2

Fabian Starc

Version: June 17, 2026

Kinematik mobiler Roboter

Roboter mit Differentialantrieb



Bei Robotern mit Differentialantrieb gilt:

$$v_L = \dot{x}_{rob} - R\dot{\alpha}$$

$$v_R = \dot{x}_{rob} + R\dot{\alpha}$$

Auflösen nach $\dot{x}_{rob}$ und $\dot{\alpha}$ ergibt:

$$\dot{x}_{rob} = \frac{1}{2}(v_L + v_R)$$

$$\dot{\alpha} = \frac{1}{2R}(v_R - v_L)$$

Position und Orientierung ermitteln:

$$x = \int \cos(\alpha) \cdot \dot{x}_{rob} \cdot dt + x_0$$

$$y = \int \sin(\alpha) \cdot \dot{x}_{rob} \cdot dt + y_0$$

$$\alpha = \int \dot{\alpha} \cdot dt + \alpha_0$$

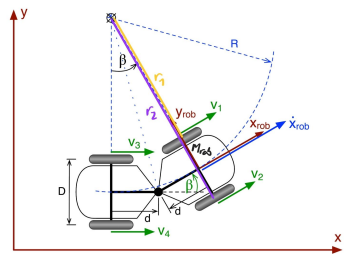
Darstellung in zeitdiskreter Form:

$$x_{k+1} = x_k + \cos(\alpha_k + \dot{\alpha}_k \cdot T) \cdot \dot{x}_{rob,k} \cdot T$$

$$y_{k+1} = y_k + \sin(\alpha_k + \dot{\alpha}_k \cdot T) \cdot \dot{x}_{rob,k} \cdot T$$

$$\alpha_{k+1} = \alpha_k + \dot{\alpha}_k \cdot T$$

Baumaschine



M_{rob} bewegt sich im Kreis mit Radius R mit Vorwärtsgeschwindigkeit $\dot{x}_{rob}$

es gilt für die Kreisbewegung:

$$v = \omega \cdot r \quad \omega = \frac{v}{r} \quad \omega = \frac{\dot{x}_{rob}}{R}$$

Distanz vom Momentanpol zum Rad 1:

$$r_1 = R - \frac{D}{2}$$

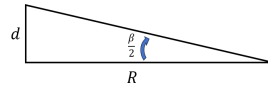
$$v_1 = \omega \cdot r$$

$$v_1 = \dot{x}_{rob} \left(1 - \frac{D}{2R}\right)$$

wiederholen für r_2 um auf v_2 zu kommen.

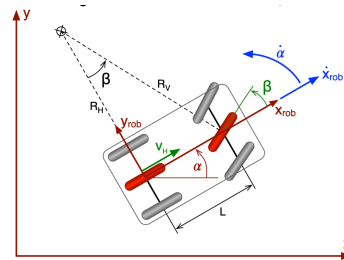
$$v_1 = v_3 \quad v_2 = v_4$$

β durch Trigonometrie:



$$\tan\left(\frac{\beta}{2}\right) = \frac{d}{R} \quad \beta = 2 \cdot \arctan\left(\frac{d}{R}\right)$$

Roboter mit gelenkten Rädern



$$\dot{\alpha} = \frac{\dot{x}_{rob}}{R_H} \quad \tan(\beta) = \frac{L}{R_H}$$

$$\dot{\alpha} = \frac{\dot{x}_{rob}}{L} \cdot \tan(\beta)$$

nach β auflösen für Rückwärtskinematik:

$$\beta = \arctan\left(\frac{\dot{\alpha} L}{\dot{x}_{rob}}\right)$$

(ob der Roboter jeweils zwei Vorder- und Hinterräder hat oder nur jeweils eins spielt keine Rolle)

Berechnung der Geschwindigkeiten und Positionen:

$$\dot{x} = \cos(\alpha) \cdot \dot{x}_{rob}$$

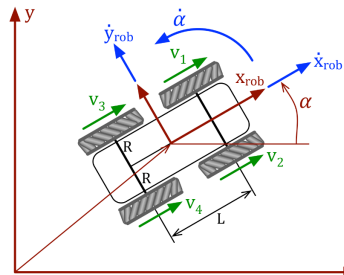
$$\dot{y} = \sin(\alpha) \cdot \dot{x}_{rob}$$

$$x = \int \cos(\alpha) \cdot \dot{x}_{rob} \cdot dt + x_0$$

$$y = \int \sin(\alpha) \cdot \dot{x}_{rob} \cdot dt + y_0$$

$$\alpha = \int \dot{\alpha} \cdot dt + \alpha_0$$

Roboter mit Mecanum-Wheels



$$v_1 = \dot{x}_{rob} - \dot{y}_{rob} \cdot \left(\frac{L}{2} + R\right) \dot{\alpha}$$

$$v_2 = \dot{x}_{rob} + \dot{y}_{rob} \cdot \left(\frac{L}{2} + R\right) \dot{\alpha}$$

$$v_3 = \dot{x}_{rob} + \dot{y}_{rob} \cdot \left(\frac{L}{2} + R\right) \dot{\alpha}$$

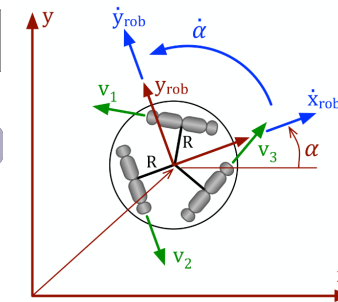
$$v_4 = \dot{x}_{rob} - \dot{y}_{rob} \cdot \left(\frac{L}{2} + R\right) \dot{\alpha}$$

$$\dot{x}_{rob} = \frac{v_1 + v_2 + v_3 + v_4}{4}$$

$$\dot{y}_{rob} = \frac{-v_1 + v_2 + v_3 - v_4}{4}$$

$$\dot{\alpha} = \frac{-v_1 + v_2 - v_3 + v_4}{4 \left(\frac{L}{2} + R\right)}$$

3-Rad Roboter



$$v_1 = -\frac{\sqrt{3}}{2} \cdot \dot{x}_{rob} + \frac{1}{2} \cdot \dot{y}_{rob} + R \cdot \dot{\alpha}$$

$$v_2 = -\dot{y}_{rob} + R \cdot \dot{\alpha}$$

$$v_3 = \frac{\sqrt{3}}{2} \cdot \dot{x}_{rob} + \frac{1}{2} \cdot \dot{y}_{rob} + R \cdot \dot{\alpha}$$

$$\dot{x}_{rob} = \frac{1}{\sqrt{3}} \cdot (v_3 - v_1)$$

$$\dot{y}_{rob} = \frac{1}{3} \cdot (v_1 - 2 \cdot v_2 + v_3)$$

$$\dot{\alpha} = \frac{1}{3R} \cdot (v_1 + v_2 + v_3)$$

Geschwindigkeiten:

$$\dot{x} = \cos(\alpha) \cdot \dot{x}_{rob} - \sin(\alpha) \cdot \dot{y}_{rob}$$

$$\dot{y} = \sin(\alpha) \cdot \dot{x}_{rob} + \cos(\alpha) \cdot \dot{y}_{rob}$$

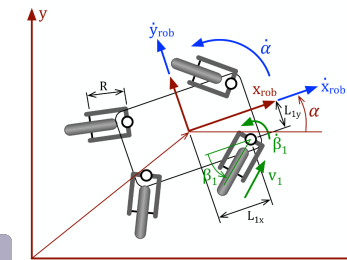
Position und Orientierung:

$$x = \int \dot{x} \cdot dt + x_0$$

$$y = \int \dot{y} \cdot dt + y_0$$

$$\alpha = \int \dot{\alpha} \cdot dt + \alpha_0$$

Roboter mit Castorrädern



1. Berechnung der Position des Radmittelpunktes im lokalen KOS:

$$x_{rob,1} = L_{1x} - R \cdot \cos(\beta_1)$$

$$y_{rob,1} = -L_{1y} - R \cdot \sin(\beta_1)$$

2. Geschwindigkeit des Radmittelpunktes im lokalen KOS:

$$\dot{x}_{rob,1} = \dot{x}_{rob} - y_{rob,1} \cdot \dot{\alpha}$$

$$\dot{y}_{rob,1} = \dot{y}_{rob} + x_{rob,1} \cdot \dot{\alpha}$$

3. Die Geschwindigkeiten des Radmittelpunktes auf die Radgeschwindigkeit und auf die Drehrate der vertikalen Drehachse umrechnen:

$$v_1 = \dot{x}_{rob,1} \cdot \cos(\beta_1) + \dot{y}_{rob,1} \cdot \sin(\beta_1)$$

$$\dot{\beta}_1 = -\frac{1}{R} \cdot \left(-\dot{x}_{rob,1} \cdot \sin(\beta_1) + \dot{y}_{rob,1} \cdot \cos(\beta_1)\right)$$

Positionsregler

Aufgaben des Positionsreglers:

Der Positionsregler hat die Aufgaben:

- die selbstständige Führung des mobilen Roboters von seiner aktuellen Lage zu einer vorgegebenen Ziellage bestehend aus Position und Orientierung,
- die Berechnung geeigneter Translations- und Rotationsgeschwindigkeiten aus der aktuellen Roboterlage und der gewünschten Ziellage,
- die Umwandlung des Positionsfehlers zwischen aktueller Lage und Ziellage in Stellgrößen für die unterlagerten Geschwindigkeits- und Motorregler,
- die Erzeugung einer kontinuierlichen und möglichst stetigen Bewegung des Roboters zur Ziellage,
- die Berücksichtigung des Abstands zur Ziellage sowie der Orientierung des Roboters relativ zum Ziel,
- die fortlaufende Neuberechnung der Stellgrößen in jedem Regelungstakt auf Basis der aktuellen Roboterlage,
- die Führung des Roboters in eine vorgegebene Ziellage, ohne dabei die Aufgaben einer vollständigen Navigation oder Hindernisvermeidung zu übernehmen.

Positionsregelung verschiedener Kinematiken

Holonome Roboter

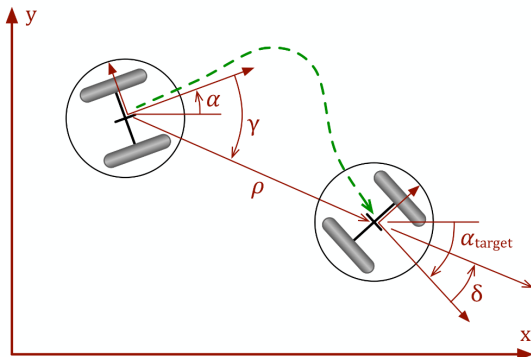
1. Die aktuelle Lage des Roboters wird bestimmt, also seine Position und Orientierung.
2. Die gewünschte Ziellage wird vorgegeben.
3. Aus aktueller Lage und Ziellage wird ein Geschwindigkeitsvektor in globalen kartesischen Koordinaten berechnet.
4. Dieser globale Geschwindigkeitsvektor wird mit einer Koordinatentransformation in das lokale Koordinatensystem des Roboters umgerechnet.
5. Die lokalen Geschwindigkeiten werden mit der Rückwärtskinematik in die benötigten Drehzahlen der Antriebsräder umgerechnet.
6. Die Raddrehzahlen werden geregelt, sodass der Roboter die gewünschte Ziellage erreicht.

Roboter mit Differentialantrieb

Einfaches Manöver:

1. Eine Rotation des Roboters, bis die lokale X-Achse in die Richtung des Ziels zeigt
2. Eine Translationsbewegung bis zur Ziellage
3. Und schliesslich erneut eine Rotation des Roboters, bis er die gewünschte Orientierung erreicht hat

Algorithmus:



Drei Hilfsgrößen werden benötigt:

$$\rho = \sqrt{(x_{target} - x)^2 + (y_{target} - y)^2}$$

$$\gamma = \arctan 2(y_{target} - y, x_{target} - x) - \alpha$$

$$= \arctan \left(\frac{y_{target} - y}{x_{target} - x} \right) - \alpha$$

$$\delta = \gamma + \alpha - \alpha_{target}$$

Positionsregelungs-Algorithmus zur Berechnung der Translations- und Rotationsgeschwindigkeit:

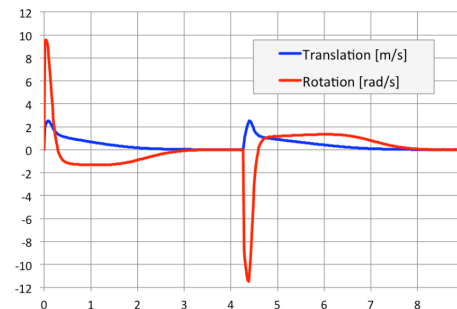
$$\dot{x}_{rob} = k_1 \cdot \rho \cdot \cos(\gamma)$$

$$\dot{\alpha} = k_2 \cdot \gamma + k_1 \cdot \sin(\gamma) \cdot \cos(\gamma) \cdot \frac{\gamma + k_3 \cdot \delta}{\gamma}$$

Durch Anpassung der Parameter k_1, k_2, k_3 kann das Verhalten des Reglers angepasst werden.

- k_1 : Bestimmt wie schnell sich der Roboter in Vorwärtsrichtung bewegt in Abhängigkeit von der Distanz zum Ziel. Hierbei wird auch die Richtung berücksichtigt durch $\cos(\gamma)$. Wenn bspw. das Ziel sich seitlich des Roboters befindet dann ist $\cos(\gamma) = 0$ und die Translationsgeschwindigkeit ist ebenfalls null.
- k_2 : Je grösser dieser Wert, desto schneller dreht sich der Roboter in Richtung des Ziels.
- k_3 : Je grösser dieser Parameter umso stärker dreht der Roboter sich vom Ziel weg, so dass er mit der gewünschten Orientierung ins Ziel fährt.

Dieser Algorithmus führt teilweise zu sehr hohen Schnelligkeiten vorallem wenn Abstand zum Ziel noch sehr gross ist.



Startposition:
 $x = 0.0 \text{ m}$
 $y = 0.0 \text{ m}$
 $\alpha = 0.0 \text{ rad}$

Ziellage:
 $x = 1.0 \text{ m}$
 $y = 1.0 \text{ m}$
 $\alpha = -1.0 \text{ rad}$

Regelparameter:
 $k_1 = 2$
 $k_2 = 5$
 $k_3 = 2$

Implementierungsaspekte:

- Der Regelungsalgorithmus muss nicht mehr berechnet werden, wenn der Abstand ρ zur Ziellage Null ist, da sich der Roboter bereits im Ziel befindet.
- Ist der Winkel $\gamma = 0$, darf die Rotationsgeschwindigkeit nicht mit der gegebenen Formel berechnet werden, da sonst eine Division durch Null auftritt. In diesem Fall wird die Rotationsgeschwindigkeit auf Null gesetzt.
- Die Winkel γ und δ müssen stets im Bereich $[-\pi, +\pi]$ liegen.
- Ist ein Winkel kleiner als $-\pi$, muss 2π addiert werden.
- Ist ein Winkel grösser als $+\pi$, muss 2π subtrahiert werden.
- Diese Randfälle müssen bei der Implementierung berücksichtigt werden, um fehlerhafte Geschwindigkeiten oder Programmabstürze zu vermeiden.

Roboter mit gelenkten Rädern

Gleiche drei Hilfsgrößen werden benötigt ρ, γ, δ

Algorithmus zur Positionsregelung:

$$\dot{x}_{rob} = k_1 \cdot \rho \quad k_1 > 0$$

$$k_3 < 0$$

$$\beta = k_2 \cdot \gamma + k_3 \cdot \delta \quad k_2 > k_1$$

Wiederum kann durch Anpassung der Parameter k_i die Positionsregelung nach Wunsch angepasst werden.

Lokalisierung

Drei zentrale Fragen der mobilen Robotik sollen beantwortet werden:

- Where am I?
- Where am I going?
- How should I get there?

Die Lokalisierung sollte die erste Frage beantworten.

Relative Lagemessung

Sensoren zur relativen Lagemessung:

- Encoder und Drehzahlsensoren
- Inertial Measurement Unit (IMU) [Beschleunigungssensoren/Gyrosensoren]
- Doppler Sensoren
- Kameras und Visionssysteme

Ein Überbegriff für eine Positionsbestimmung basierend auf Messungen der Bewegungsrichtung, der Geschwindigkeit und der Zeit

Vorteile:

- In der Regel sehr kostengünstig. Drehzahlsensoren sind bei mob. Robotern oft schon verbaut.
- Sensoren zur relativen Lagemessung sind sehr schnell. Encoder Counter oder IMUs bspw. haben abtastraten von 1kHz.

Nachteile:

- Die Startposition wird nicht gemessen und muss initialisiert werden.
- Gemessene Lage driftet mit der Zeit von der tatsächlichen Lage weg: Systematische Fehler wie abweichende Radpositionen oder Radumfänge sowie beschränkte Auflösung von Winkelencodern. Schlupf oder Schleudern beim Fahren.

Absolute Lagemessung

Sensoren zur absoluten Lagemessung:

- GPS
- Ultraschall, IR Distanzsensoren
- LiDAR / Laserscanner
- 3D und time of flight Kameras
- Radio Beacons (Bluetooth oder Ultra Wide Band)
- 2D Kameras
- Kompass

Vorteile:

- Eine absolute Lagemessung liefert direkt die Information, die anschliessend für die Navigation des Roboters benötigt wird

Nachteile:

- Eine absolute Lagemessung ist oft langsam.
- Absolute Lagemessungen stehen oft nicht zur Verfügung: GPS funktioniert nicht in Gebäuden, Garagen oder Tunnels oder Referenzmarken können temporär durch Hindernisse verdeckt sein.

Lagemessung mit LiDAR und Referenzmarken

Der LiDAR misst für jede Referenzmarke den Radius R sowie den Scanwinkel γ :

$${}^{rob}P_{1,x} = R_1 \cdot \cos(\gamma_1)$$

$${}^{rob}P_{1,y} = R_1 \cdot \sin(\gamma_1)$$

Homogene Transformation ins globale KOS:

$$\begin{bmatrix} {}^0P_{1,x} \\ {}^0P_{1,y} \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} {}^{rob}P_{1,x} \\ {}^{rob}P_{1,y} \\ 1 \end{bmatrix}$$

Transformation kann mathematisch umgeformt werden, so dass die Koordinaten von mehreren Referenzmarken in einer einzigen Matrixmultiplikation berücksichtigt werden:

$$\underbrace{\begin{bmatrix} {}^0P_{1,x} & {}^0P_{1,y} \\ {}^0P_{2,x} & {}^0P_{2,y} \\ \vdots & \vdots \\ {}^0P_{n,x} & {}^0P_{n,y} \end{bmatrix}}_{\text{Koord. Referenzmarken Global } P_0} = \underbrace{\begin{bmatrix} {}^{rob}P_{1,x} & {}^{rob}P_{1,y} & 1 \\ {}^{rob}P_{2,x} & {}^{rob}P_{2,y} & 1 \\ \vdots & \vdots & \vdots \\ {}^{rob}P_{n,x} & {}^{rob}P_{n,y} & 1 \end{bmatrix}}_{\text{Koord. Referenzmarken Lokal } P_{rob}} \cdot \underbrace{\begin{bmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \\ 0 & 0 \end{bmatrix}}_{\text{Transformationsmatrix } T}$$

$$P_0 = P_{rob} \cdot T$$

Aufgelöst nach T :

$$T = P_{rob}^{-1} \cdot P_0$$

für $n > 3$:

$$T = P_{rob}^* \cdot P_0 \quad P_{rob}^* = P_{rob}^T \cdot (P_{rob} \cdot P_{rob}^T)^{-1}$$

Wenn man mehr als 3 Referenzmarken detektieren kann, dann kann die Spaltenmatrix rechts nicht invertiert werden. Stattdessen ist eine Pseudoinverse zu berechnen.

Mit dieser Pseudoinversen Matrix wird eine least-square Approximation der Lösung berechnet. Die Lösung ist damit umso genauer, je mehr Referenzmarken detektiert werden können.

gesuchte Terme ableiten:

$${}^0x = T_{3,1}$$

$${}^0y = T_{3,2}$$

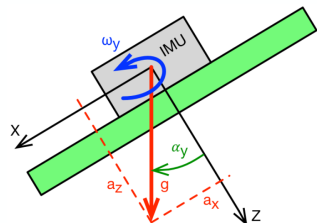
$$\alpha = \text{atan2}(T_{1,2}, T_{1,1})$$

Sensor Fusion

Sobald verschiedene Sensoren in Kombination für die Bestimmung des Zustandes eines Systems verwendet werden, spricht man von Sensor Fusion.

Komplementärfilter

Neigungswinkel um die y-Achse des Roboters bestimmen:



Durch Beschleunigungen in x- und z-Richtung:

$$\alpha_y = \arctan\left(\frac{a_x}{a_z}\right)$$

Durch integration der gemessenen Drehrate:

$$\alpha_y = \int \omega_y dt + \alpha_{y,0}$$

- Die Berechnung des Neigungswinkels aus den Beschleunigungssensoren liefert einen absoluten Winkel ohne Drift, ist jedoch verrauscht und empfindlich gegenüber Bewegungsstörungen.

- Die Berechnung des Neigungswinkels durch Integration der Drehrate liefert ein rauscharmes Signal mit hoher Dynamik, führt jedoch aufgrund von Offsetfehlern zu Drift und liefert keinen absoluten Winkel.

- Durch Sensor Fusion können die Vorteile beider Methoden kombiniert werden: die Langzeitstabilität der Beschleunigungssensoren und die Dynamik der Drehratensensoren.

Neigungswinkel durch Messung von zwei Beschleunigungssensoren berechnet und Tiefpassfilter eingesetzt:

$$\alpha_{y,k+1} = s_f \cdot \arctan\left(\frac{a_{x,k+1}}{a_{z,k+1}}\right) + (1 - s_f) \cdot \alpha_{y,k}$$

Neigungswinkel mittels Integration berechnet und einsatz eines Hochpassfilters:

$$\hat{\alpha}_{y,k+1} = \hat{\alpha}_{y,k} + \omega_{y,k+1} \cdot T$$

$$\alpha_{y,k+1} = \frac{1}{1 + \omega_c T} \alpha_{y,k} + \frac{1}{1 + \omega_c T} (\hat{\alpha}_{y,k+1} - \hat{\alpha}_{y,k})$$

Kalman-Filter

Zustandsraum-Darstellung des Systems benötigt:

$$\begin{bmatrix} \alpha_{y,k+1} \\ \omega_{y,k+1} \end{bmatrix} = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \alpha_{y,k} \\ \omega_{y,k} \end{bmatrix} + w_k \quad w_k = \text{Prozessrauschen}$$

$$z_{k+1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \alpha_{y,k+1} \\ \omega_{y,k+1} \end{bmatrix} + v_{k+1} \quad v_k = \text{Messrauschen}$$

Prozessrauschen und Messrauschen wird mit Kovarianzmatrizen modelliert:

$$Q = \begin{bmatrix} \sigma_{q,\alpha}^2 & 0 \\ 0 & \sigma_{q,\omega}^2 \end{bmatrix} \quad R = \begin{bmatrix} \sigma_{r,\alpha}^2 & 0 \\ 0 & \sigma_{r,\omega}^2 \end{bmatrix}$$

Prädiktion und Korrektur mittels Kalman-Filter:

- Schätzung des neuen Zustandes:

$$\hat{\alpha}_{y,k+1} = \hat{\alpha}_{y,k} + T \cdot \omega_{y,k}$$

$$\hat{\omega}_{y,k+1} = \hat{\omega}_{y,k}$$

- Kovarianzmatrix P :

$$P_{k+1} = A \cdot P_k \cdot A^T + Q = \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix} \cdot P_k \cdot \begin{bmatrix} 1 & T \\ 0 & 1 \end{bmatrix}^T + Q$$

- Berechnung der Korrektur:

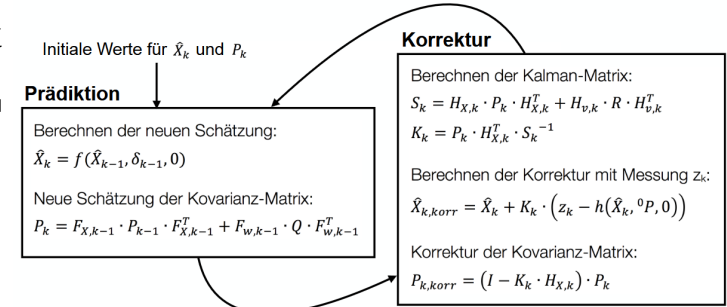
Dazu wird zuerst die Kalman-Matrix K und damit sowohl der Zustand wie auch die Kovarianz-Matrix P korrigiert:

$$K_{k+1} = P_{k+1} \cdot H^T \cdot (H \cdot P_{k+1} \cdot H^T + R)^{-1} = P_{k+1} \cdot (P_{k+1} + R)^{-1}$$

$$\begin{bmatrix} \hat{\alpha}_{y,k+1,\text{korrt}} \\ \hat{\omega}_{y,k+1,\text{korrt}} \end{bmatrix} = \begin{bmatrix} \hat{\alpha}_{y,k+1} \\ \hat{\omega}_{y,k+1} \end{bmatrix} + K_{k+1} \cdot \begin{bmatrix} z_{k+1,\alpha} - \hat{\alpha}_{y,k+1} \\ z_{k+1,\omega} - \hat{\omega}_{y,k+1} \end{bmatrix}$$

$$P_{k+1,\text{korrt}} = (I - K_{k+1} \cdot H) \cdot P_{k+1} = (I - K_{k+1}) \cdot P_{k+1}$$

Lokalisierung mobiler Roboter mit Kalmanfilter



- Kalman-Filter benötigen das zeitdiskrete Modell der Dynamik des Systems (siehe Roboter mit Differentialantrieb)
- Prozessrauschen muss berücksichtigt werden also wird Gleichung mit Termen für das Rauschen erweitert:

$$X_{k+1} = \begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \alpha_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + (\delta_{d,k} + w_{d,k}) \cos(\alpha_k + \delta_{\alpha,k} + w_{\alpha,k}) \\ y_k + (\delta_{d,k} + w_{d,k}) \sin(\alpha_k + \delta_{\alpha,k} + w_{\alpha,k}) \\ \alpha_k + \delta_{\alpha,k} + w_{\alpha,k} \end{bmatrix}$$

- Prozessrauschen wird wiederum mit Kovarianzmatrix Q beschrieben:

$$Q = \begin{bmatrix} \sigma_d^2 & 0 \\ 0 & \sigma_\alpha^2 \end{bmatrix} \quad \text{mit} \quad \sigma_d, \sigma_\alpha \text{ Standardabweichung der Translation und Rotation}$$

in Kurz:

$$X_{k+1} = f(X_k, \delta_k, w_k)$$

- Prozessrauschen wird Null gesetzt da aktuelles Rauschen nicht bekannt ist:

$$\hat{X}_{k+1} = f(\hat{X}_k, \delta_k, 0)$$

- Kalman-Filter setzt voraus dass Systemgleichungen linear sind (sind sie aber nicht wegen sin- und cos-Termen). Dafür wird lineare Approximation berechnet:

$$X_{k+1} = f(\hat{X}_k, \delta_k, 0) + F_x \cdot (X_k - \hat{X}_k) + F_w \cdot w_k$$

Matrizen F_X und F_K sind Jacobi-Matrizen welche durch partielle Ableitung nach den Zustandswerten x, y, α bzw. w_d und w_α entstehen:

$$F_x = \frac{\partial f}{\partial X} \Big|_{w=0} = \begin{bmatrix} 1 & 0 & -\delta_{d,k} \sin(\alpha_k + \delta_{\alpha,k}) \\ 0 & 1 & \delta_{d,k} \cos(\alpha_k + \delta_{\alpha,k}) \\ 0 & 0 & 1 \end{bmatrix}$$

$$F_w = \frac{\partial f}{\partial w} \Big|_{w=0} = \begin{bmatrix} \cos(\alpha_k + \delta_{\alpha,k}) & -\delta_{d,k} \sin(\alpha_k + \delta_{\alpha,k}) \\ \sin(\alpha_k + \delta_{\alpha,k}) & \delta_{d,k} \cos(\alpha_k + \delta_{\alpha,k}) \\ 0 & 1 \end{bmatrix}$$

- nun kann die Prädiktion des Kalman-Filters berechnet werden: als erstes berechnung der Schätzung des Zustandes X zum Zeitpunkt k :

$$\hat{X}_{k+1} = f(\hat{X}_k, \delta_k, 0) = \begin{bmatrix} \hat{x}_k + \delta_{d,k} \cos(\hat{\alpha}_k + \delta_{\alpha,k}) \\ \hat{y}_k + \delta_{d,k} \sin(\hat{\alpha}_k + \delta_{\alpha,k}) \\ \hat{\alpha}_k + \delta_{\alpha,k} \end{bmatrix}$$

dann die Kovarianz-Matrix P_k :

$$P_{k+1} = F_{x,k} \cdot P_k \cdot F_{x,k}^T + F_{w,k} \cdot Q \cdot F_{w,k}^T$$

7.

$$\begin{bmatrix} \vdots \\ \vdots \end{bmatrix}$$

8.

Navigation ohne Karte

Zentrale Frage "How should I get there?" soll beantwortet werden

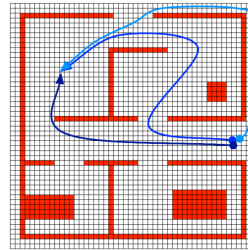
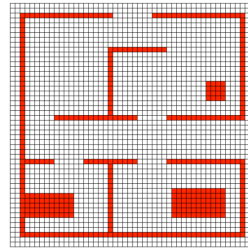
- Navigation ohne Karte wird auch als reaktive Navigation bezeichnet.
- Der Roboter verwendet dabei keine gespeicherte Karte der Umgebung.
- Der Roboter muss seine genaue Position nicht kennen und benötigt daher keine vollständige Lokalisierung.
- Die Bewegung wird direkt aus aktuellen Sensordaten oder einfachen Regeln abgeleitet.
- Ein Beispiel ist ein Staubsaugerroboter, der geradeaus fährt, bei einem Hindernis abdreht und danach wieder weiterfährt.
- Eine Zielposition kann auch durch ein Signal markiert werden, zum Beispiel durch Licht oder Funk. Der Roboter folgt dann dem Signalgradienten zum Ziel.
- Diese Methode funktioniert einfach und robust, ist aber meist nicht optimal und kann bei Hindernissen zwischen Roboter und Ziel problematisch sein.

Bug-Algorithmus



- Der Bug-Algorithmus ist ein Navigationsverfahren, mit dem ein Roboter auch ohne vollständige Kartenplanung ein Ziel erreichen kann.
- Zuerst wird eine direkte Linie von der Startposition zur Zielposition berechnet.
- Der Roboter bewegt sich entlang dieser Linie in Richtung Ziel.
- Trifft der Roboter auf ein Hindernis, verlässt er die direkte Linie.
- Danach folgt er dem Rand des Hindernisses, zum Beispiel im Uhrzeigersinn.
- Sobald der Roboter die ursprüngliche Verbindungslinie zum Ziel wieder erreicht, prüft er, ob diese neue Position näher am Ziel liegt als der Punkt, an dem er die Linie verlassen hat.
- Ist die neue Position näher am Ziel, folgt der Roboter wieder der direkten Linie zum Ziel.
- Der Algorithmus kann den Roboter grundsätzlich zum Ziel führen, ist aber oft ineffizient.
- Der zurückgelegte Weg kann deutlich länger sein als der kürzeste mögliche Weg.
- Der Algorithmus setzt voraus, dass der Roboter seine aktuelle Position kennt und Hindernisse in unmittelbarer Nähe erkennen kann.

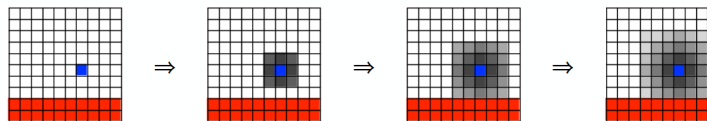
Navigation mit Gitterkarte



- Bei der Navigation mit Gitterkarten wird die Umgebung in diskrete Zellen unterteilt.
- Hindernisse werden als besetzte Zellen markiert, während freie Zellen vom Roboter befahren werden können.
- Gitterkarten stellen eine einfache und anschauliche Möglichkeit dar, die Umgebung eines Roboters digital abzubilden.
- Auf Basis dieser Karte können Navigationsalgorithmen einen Weg von einer Startposition zu einer Zielposition berechnen.
- Idealerweise wird dabei nicht nur ein möglicher Weg, sondern der kürzeste oder effizienteste Weg zum Ziel bestimmt.
- Ein Nachteil von Gitterkarten ist der hohe Speicher- und Rechenaufwand, der mit zunehmender Kartengröße und Auflösung stark ansteigt.

Distanztransformation

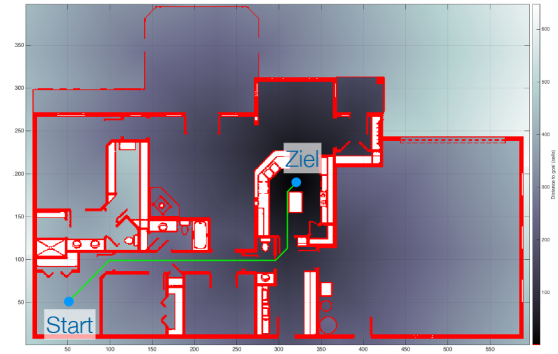
- Die Distanztransformation ist ein Verfahren zur Vorbereitung der Pfadplanung auf einer Gitterkarte.
- Ausgehend von der Zielzelle wird für jede freie Zelle der Abstand zum Ziel berechnet.
- Die berechneten Distanzen werden in einer Distanzkarte gespeichert.
- Die Ausbreitung der Distanzen erfolgt schrittweise von der Zielzelle zu benachbarten freien Zellen.
- Hindernisse werden bei der Berechnung nicht berücksichtigt und erhalten keinen gültigen Distanzwert.
- Nach Abschluss der Transformation enthält jede freie Zelle die Distanz bis zum Ziel.
- Ein Pfad kann anschließend bestimmt werden, indem der Roboter von seiner Startposition aus jeweils in die benachbarte Zelle mit dem kleinsten Distanzwert fährt.
- Die Distanztransformation garantiert, dass auf einer bekannten Gitterkarte ein kürzester Weg zum Ziel gefunden werden kann.
- Das Verfahren eignet sich besonders für statische Umgebungen, da die Distanzkarte bei Änderungen der Umgebung neu berechnet werden muss.



Einschränkungen der Distanztransformation:

- Die Distanztransformation benötigt eine vollständige und bekannte Karte der Umgebung.
- Bei Änderungen der Umgebung, beispielsweise durch neue Hindernisse, muss die Distanzkarte neu berechnet werden.
- Der Speicherbedarf steigt mit der Größe und Auflösung der Gitterkarte.

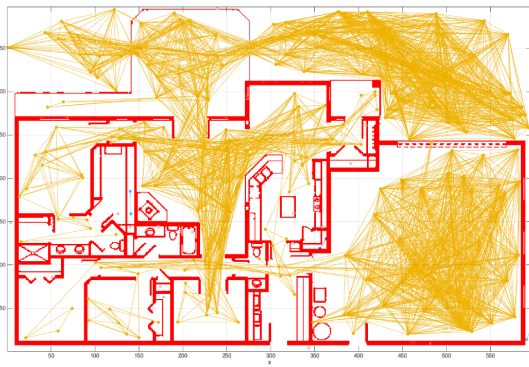
- Die Rechenzeit nimmt bei großen Karten oder feiner Rasterung deutlich zu.
- Der berechnete Pfad führt häufig sehr nahe an Hindernissen vorbei, da ausschließlich die Distanz zum Ziel berücksichtigt wird.
- Sicherheitsabstände zu Hindernissen werden nicht automatisch berücksichtigt und müssen zusätzlich eingeplant werden.
- Das Verfahren eignet sich hauptsächlich für statische Umgebungen und ist für dynamische Hindernisse nur eingeschränkt geeignet.
- Die Qualität des Ergebnisses hängt stark von der gewählten Auflösung der Gitterkarte ab. Eine grobe Rasterung kann zu ungenauen Pfaden führen.



- Nach der Distanztransformation enthält jede freie Zelle einen Distanzwert zum Ziel.
- Der Roboter folgt dem Gradienten dieser Distanzwerte, indem er jeweils die benachbarte Zelle mit dem kleinsten Distanzwert auswählt.
- Durch das fortlaufende Folgen des Gradienten bewegt sich der Roboter schrittweise zum Ziel.
- Das Verfahren endet, sobald die Zielzelle mit dem kleinsten Distanzwert erreicht wird.

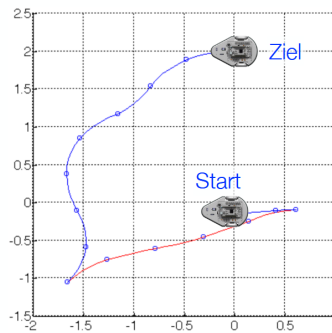
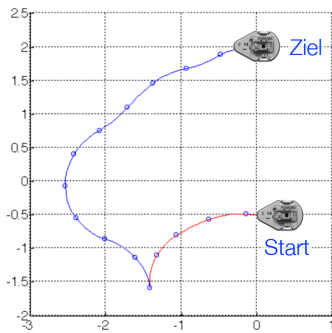
Probabilistic Roadmap Method (PRM)

- Die Probabilistic Roadmap Method (PRM) kombiniert eine Gitterkarte mit einer topologischen Karte zur Navigation.
- Dazu werden zufällig Knoten in freien Bereichen der Karte erzeugt.
- Benachbarte Knoten werden miteinander verbunden, sofern zwischen ihnen kein Hindernis liegt.
- Dadurch entsteht ein Netzwerk möglicher Fahrwege durch die Umgebung.
- Für die Navigation wird die Startposition mit dem nächstgelegenen Knoten verbunden.
- Anschließend wird der kürzeste Weg durch das Netzwerk zum Knoten in der Nähe des Ziels gesucht.
- Vom letzten Knoten wird ein direkter Weg zur Zielposition berechnet.
- Die topologische Karte muss für eine statische Umgebung nur einmal erzeugt werden und kann danach mehrfach verwendet werden.
- Dadurch ist der Rechenaufwand meist deutlich geringer als bei der Distanztransformation.
- Die berechneten Wege sind jedoch häufig weniger optimal als die durch eine Distanztransformation gefundenen Pfade.

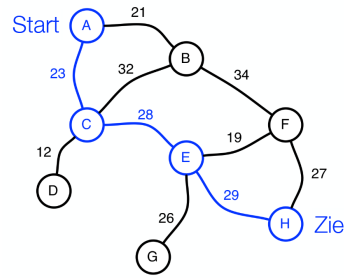


Rapidly-exploring Random Tree (RRT)

- Der Rapidly-exploring Random Tree (RRT) ist ein stochastischer Suchalgorithmus zur Pfadplanung in komplexen Umgebungen.
- Ausgehend von der Startposition wird schrittweise ein Baum aufgebaut.
- Dazu werden zufällige Punkte in der Umgebung erzeugt.
- Für jeden zufälligen Punkt wird der nächstgelegene Knoten des bestehenden Baumes bestimmt.
- Vom nächstgelegenen Knoten wird ein neuer Knoten in Richtung des zufälligen Punktes erzeugt.
- Der neue Knoten wird nur hinzugefügt, wenn die Verbindung kollisionsfrei ist.
- Durch die wiederholte Erweiterung wächst der Baum in freie Bereiche der Umgebung hinein.
- Die Suche endet, sobald ein Knoten das Ziel erreicht oder sich ausreichend nahe am Ziel befindet.
- Der gefundene Pfad ergibt sich durch Rückverfolgung der Knoten vom Ziel zur Startposition.
- RRT eignet sich besonders für hochdimensionale Suchräume und komplexe Hindernisumgebungen.
- Die berechneten Pfade sind meist nicht optimal und werden deshalb häufig nachträglich geglättet oder optimiert.
- RRT berücksichtigt als einzige Einschränkungen der Kinematik des Roboters.

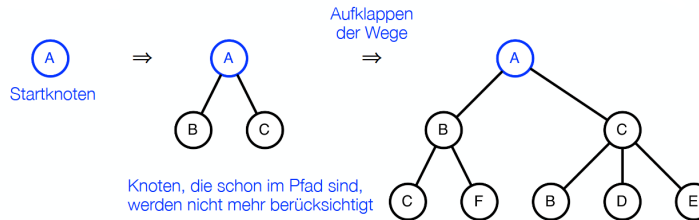


Navigation mit topologischen Karten

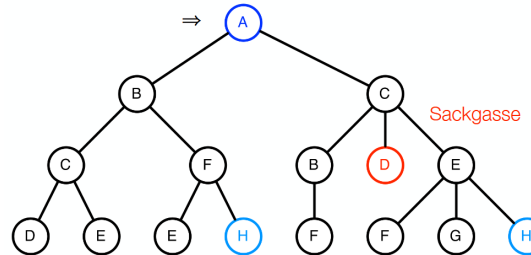


Aufgabe besteht darin, den besten (kürzesten oder schnellsten Weg) von Startknoten zum Zielknoten
Dafür stehen verschiedene Algorithmen zur Verfügung.

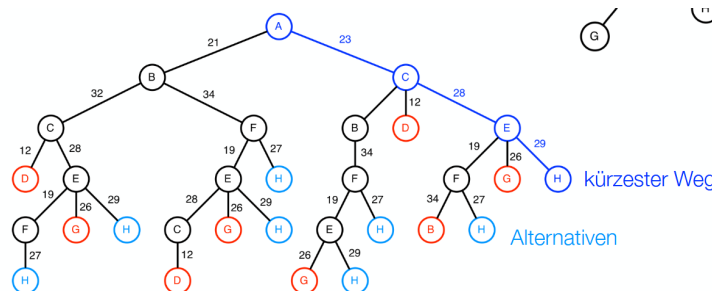
1. Beginn beim Startknoten. Man verbindet mit Ästen alle Knoten welche in der Karte mit dem Startknoten verbunden sind.



2. Bei Sackgassen wird kein weiterer Ast hinzugefügt:



3. Algorithmus wird dann gestoppt, wenn alle Äste in einer Sackgasse oder aber beim gewünschten Zielknoten enden. Danach wird jeder Ast gewichtet bspw. Proportional zur Distanz zwischen den jeweiligen Knoten



4. So erhält man die Längen von allen möglichen Lösungen die es gibt und daraus die beste wählen.

Verbesserungen und Optimierungen:

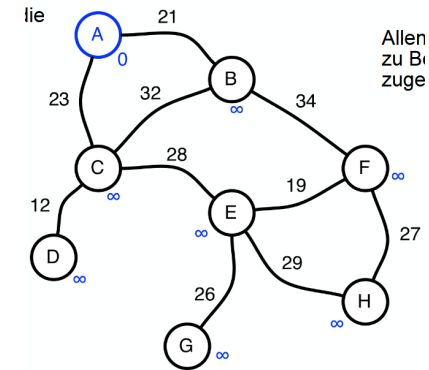
- Wege welche zu Sackgassen führen löschen.
- Zuerst Äste bis zum Zielknoten bilden bevor andere Äste untersucht werden. (erst in Tiefe statt in die Breite beim erstellen des Graphen)
- Suche nach Lösung kann bidirektional durchgeführt werden (Erstellen von Graphen von Start zu Ziel und umgekehrt)

Algorithmus von Dijkstra

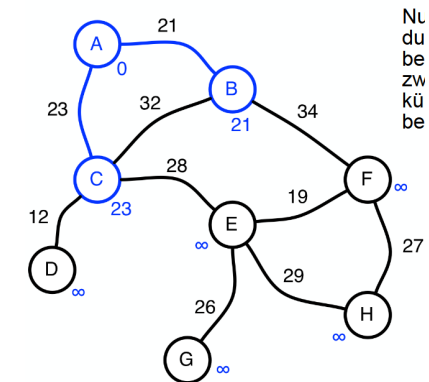
Grundsätzliche Idee:

Allen Knoten welche die kürzeste Distanz bis zum Startknoten und Werte von Startknoten zum Zielknoten hin korrigiert.

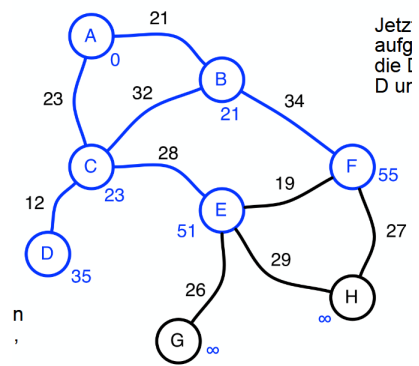
1. Startknoten wird Distanz 0 zugewiesen. Allen anderen Knoten die Distanz ∞



2. Distanzen zum Knoten A werden bei Knoten B und C eingetragen:



3. Nun werden die Verbindungen vom Knoten zu den benachbarten Knoten aufgeklappt (in der Reihenfolge des kürzesten Wegs)
4. Wenn berechnete Distanzen kleiner sind als die die schon eingetragen sind, werden diese korrigiert.
5. Knoten C wird aufgeklappt und es werden Distanzen zu den Knoten D und E berechnet.



6. So können die Distanzen zu allen Knoten gefunden werden.

