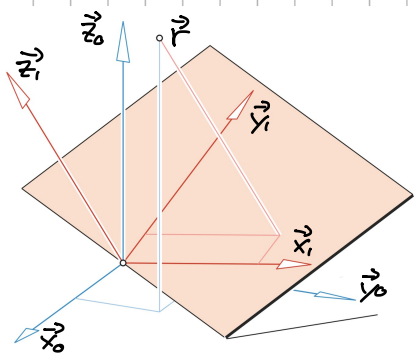
Roboterkinematik

Rotation eines kartesischen Koordinatensystems



$${}^0\vec{r} = {}^0_1R\,{}^1\vec{r} \quad \text{resp.} \quad \begin{bmatrix} {}^0r_x \\ {}^0r_y \\ {}^0r_z \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} {}^1r_x \\ {}^1r_y \\ {}^1r_z \end{bmatrix}$$

${}^0_1R$ kann wie folgt berechnet werden:

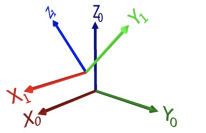
$${}^0_1R = \begin{bmatrix} \vec{x}_1\cdot\vec{x}_0 & \vec{y}_1\cdot\vec{x}_0 & \vec{y}_1\cdot\vec{x}_0 \\ \vec{x}_1\cdot\vec{y}_0 & \vec{y}_1\cdot\vec{y}_0 & \vec{y}_1\cdot\vec{y}_0 \\ \vec{x}_1\cdot\vec{z}_0 & \vec{y}_1\cdot\vec{z}_0 & \vec{y}_1\cdot\vec{z}_0 \end{bmatrix}$$

Verkettung von Rotationen:

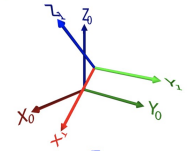
$${}^0_nR = {}^0_1R \cdot {}^1_2R \cdots {}^{n-1}_nR$$

elementare Rotationen:

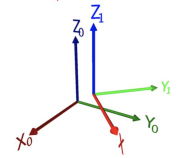
$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$



$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$



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



Eulerwinkel

jede Rotation im $\mathbb{R}^3$ kann als Sequenz von höchstens 3 Rotationen um Koordinatenachsen betrachtet werden, wobei aufeinanderfolgende Rotationen um verschiedene Achsen erfolgen müssen

damit ergeben sich 12 mögliche Sequenzen

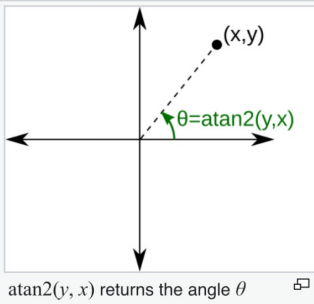
X-Y-Z	Z-Y-X	X-Y-X	X-Z-X
Y-Z-X	Y-X-Z	Y-X-Y	Y-Z-Y
Z-X-Y	X-Z-Y	Z-X-Z	Z-Y-Z

wenn Rotationen um alle 3 Achsen erfolgen spricht man auch von Kardanwinkel

Z-Y-Z Eulerwinkel:

$${}^0_1R = R_z(\alpha)\cdot R_y(\beta)\cdot R_z(\gamma) = \begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} c\beta & 0 & s\beta \\ 0 & 1 & 0 \\ -s\beta & 0 & c\beta \end{bmatrix}\begin{bmatrix} c\gamma & -s\gamma & 0 \\ s\gamma & c\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c\alpha\cdot c\beta\cdot c\gamma - s\alpha\cdot s\gamma & -c\alpha\cdot c\beta\cdot s\gamma - s\alpha\cdot c\gamma & c\alpha\cdot s\beta \\ s\alpha\cdot c\beta\cdot c\gamma + c\alpha\cdot s\gamma & -s\alpha\cdot c\beta\cdot s\gamma + c\alpha\cdot c\gamma & s\alpha\cdot s\beta \\ -s\beta\cdot c\gamma & s\beta\cdot s\gamma & c\beta \end{bmatrix}$$



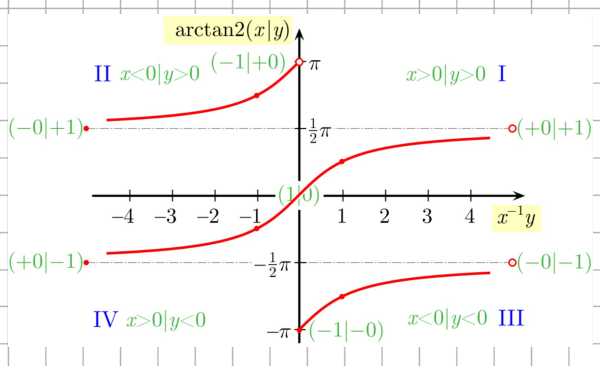
atan2(y, x) returns the angle θ between the positive x-axis and the ray from the origin to the point (x, y), confined to $(-\pi, \pi]$.

aus gegebener Rotationsmatrix Z-Y-Z Eulerwinkel bestimmen:

$$\beta = \operatorname{acos}(r_{33})$$

$$\alpha = \operatorname{atan2}\left(\frac{r_{23}}{\sin(\beta)}, \frac{r_{13}}{\sin(\beta)}\right)$$

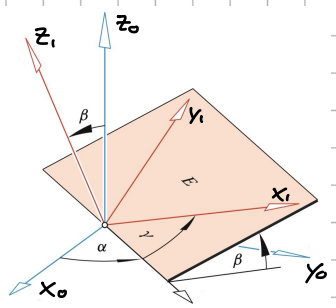
$$\gamma = \operatorname{atan2}\left(\frac{r_{32}}{\sin(\beta)}, \frac{-r_{31}}{\sin(\beta)}\right)$$



Z-X-Z Eulerwinkel:

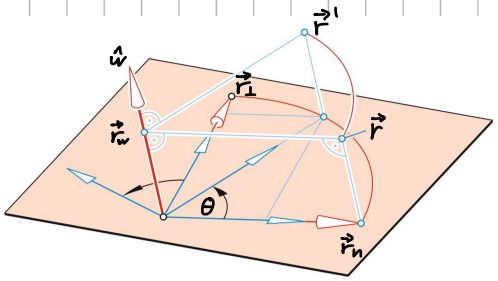
$$\underline{R} = \underline{R}_z(\alpha)\cdot\underline{R}_x(\beta)\cdot\underline{R}_z(\gamma) = \begin{bmatrix} c\alpha & -s\alpha & 0 \\ s\alpha & c\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & c\beta & -s\beta \\ 0 & s\beta & c\beta \end{bmatrix}\begin{bmatrix} c\gamma & -s\gamma & 0 \\ s\gamma & c\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} c\alpha & -s\alpha c\beta & s\alpha s\beta \\ s\alpha & c\alpha c\beta & -c\alpha s\beta \\ 0 & s\beta & c\beta \end{bmatrix}\begin{bmatrix} c\gamma & -s\gamma & 0 \\ s\gamma & c\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} c\alpha c\gamma - s\alpha c\beta s\gamma & -c\alpha s\gamma - s\alpha s\beta c\gamma & s\alpha s\beta \\ s\alpha c\gamma + c\alpha c\beta s\gamma & -s\alpha s\gamma + c\alpha c\beta c\gamma & -c\alpha s\beta \\ s\beta s\gamma & s\beta c\gamma & c\beta \end{bmatrix}$$



falls umgekehrt eine Position des Rechtsachsenkreuzes (x_1, x_2, x_3) gegeben ist so sind bei linear unabhängigen $\{z_0, z_1\}$ die zugehörigen Eulerschen Drehwinkel eindeutig bestimmt sofern man deren Grenzen mit $0° \le \alpha < 360°$, $0° \le \beta \le 180°$, $0° \le \gamma < 360°$ festsetzt

Drehwinkel und Drehvektor



jede Rotation im $\mathbb{R}^3$ kann mit Drehwinkel θ um Drehvektor $\hat{w}$ (Einheitsvektor) beschrieben werden

$$\vec{r}' = \vec{r}_w + \cos\theta\,\vec{r}_n + \sin\theta\,\vec{r}_\perp$$

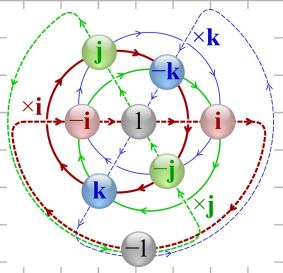
$$= (\vec{w}\cdot\vec{r})\,\vec{w} + \cos\theta\,(\vec{r} - \vec{r}_w) + \sin\theta\,(\hat{w}\times\vec{r}_n)$$

$$= \left((\hat{w}\hat{w}^T) + \cos\theta\,(\underline{E}_3 - \hat{w}\hat{w}^T) + \sin\theta\,\underline{S}_{\hat{w}}\right)\vec{r}$$

$$= \underline{R}_{\hat{w},\theta}\,\vec{r} \quad \text{wobei} \quad \underline{S}_{\hat{w}} = [[0, -w_z, w_y], [w_z, 0, -w_x], [-w_y, w_x, 0]]$$

$$\theta = \arccos\left(\frac{r_{11}+r_{22}+r_{33}-1}{2}\right), \quad \begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix} = \frac{1}{2\sin\theta}\cdot\begin{bmatrix} r_{32}-r_{23} \\ r_{13}-r_{31} \\ r_{21}-r_{12} \end{bmatrix}$$

Quaternionen (auch: hamiltonsche Quaternionen / Hamilton-Zahlen)



$\mathbb{H} = \{a + bi + cj + dk \mid a,b,c,d \in \mathbb{R}\}$ a = Skalarteil, $bi+cj+dk$ = Vektorteil

Hamilton-Regeln: $\boxed{ij = -ji = k,\ jk = -kj = i,\ ki = -ik = j,\ i^2 = j^2 = k^2 = -1}$

vierdimensionaler Vektorraum über $\mathbb{R}$ mit Basis $1, i, j, k$

assoziativ, distributiv aber nicht kommutativ bez. Multiplikation («Schiefkörper»)

Rotationen im $\mathbb{R}^3$ können mit Quaternionen dargestellt werden:

$$\boxed{\epsilon = \epsilon_0 + \epsilon_1\cdot i + \epsilon_2\cdot j + \epsilon_3\cdot k}$$

können aus Drehwinkel & -vektor oder aus Rotationsmatrix berechnet werden:

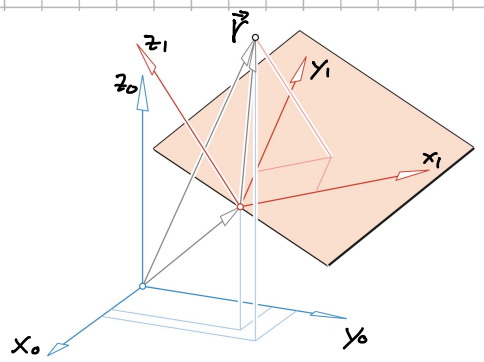
$$\epsilon_0 = \cos(\theta/2),\ \epsilon_1 = w_x\sin(\theta/2),\ \epsilon_2 = w_y\sin(\theta/2),\ \epsilon_3 = w_z\sin(\theta/2)$$

$$\epsilon_0 = \frac{1}{2}\sqrt{1 + r_{11} + r_{22} + r_{33}},\ \epsilon_1 = \frac{r_{32}-r_{23}}{4\epsilon_0},\ \epsilon_2 = \frac{r_{13}-r_{31}}{4\epsilon_0},\ \epsilon_3 = \frac{r_{21}-r_{12}}{4\epsilon_0}$$

Rotationsmatrix aus Quaternionen berechnen:

$$\underline{R} = \begin{bmatrix} 1-2(\epsilon_2^2+\epsilon_3^2) & 2(\epsilon_1\epsilon_2 - \epsilon_0\epsilon_3) & 2(\epsilon_1\epsilon_3 + \epsilon_0\epsilon_2) \\ 2(\epsilon_1\epsilon_2 + \epsilon_0\epsilon_3) & 1-2(\epsilon_1^2+\epsilon_3^2) & 2(\epsilon_2\epsilon_3 - \epsilon_0\epsilon_1) \\ 2(\epsilon_1\epsilon_3 - \epsilon_0\epsilon_2) & 2(\epsilon_2\epsilon_3 + \epsilon_0\epsilon_1) & 1-2(\epsilon_1^2+\epsilon_2^2) \end{bmatrix}$$

Rotation und Translation



Koordinaten eines Punktes von verschobenem, gedrehtem Koordinatensystem $(\vec{x}_1, \vec{y}_1, \vec{z}_1)$ ins Basissystem $(\vec{x}_0, \vec{y}_0, \vec{z}_0)$ umrechnen:

$$ {}^0\vec{r} = {}^0_1\vec{p} + {}^0_1\underline{R} \cdot {}^1\vec{r} \quad \text{mit} \quad {}^0_1\vec{p} = \begin{bmatrix} {}^0p_x \\ {}^0p_y \\ {}^0p_z \end{bmatrix}, \quad {}^0_1\underline{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} $$

homogene Transformation

Rotation und Translation können in einer einzigen Matrix-Multiplikation zusammengefasst werden:

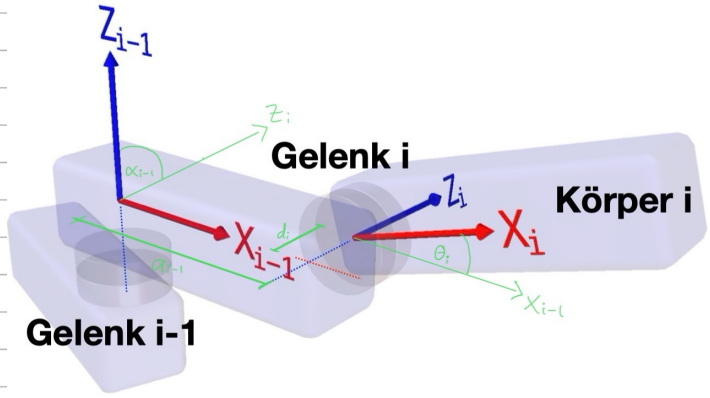
$$ {}^0\vec{r} = {}^0_1\vec{p} + {}^0_1\underline{R} \cdot {}^1\vec{r} \iff \begin{cases} {}^0_1\underline{R} \cdot {}^1\vec{r} + {}^0_1\vec{p} \cdot 1 = {}^0\vec{r} \\ \vec{0}_3^T \cdot {}^1\vec{r} + 1 \cdot 1 = 1 \end{cases} \iff \begin{bmatrix} {}^0\vec{r} \\ 1 \end{bmatrix} = \begin{bmatrix} {}^0_1\underline{R} & {}^0_1\vec{p} \\ \vec{0}_3^T & 1 \end{bmatrix} \cdot \begin{bmatrix} {}^1\vec{r} \\ 1 \end{bmatrix} $$

$$ \begin{bmatrix} {}^0r_x \\ {}^0r_y \\ {}^0r_z \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & {}^0p_x \\ r_{21} & r_{22} & r_{23} & {}^0p_y \\ r_{31} & r_{32} & r_{33} & {}^0p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} {}^1r_x \\ {}^1r_y \\ {}^1r_z \\ 1 \end{bmatrix} \quad \text{resp} \quad \begin{bmatrix} {}^0\vec{r} \\ 1 \end{bmatrix} = {}^0_1\underline{T} \cdot \begin{bmatrix} {}^1\vec{r} \\ 1 \end{bmatrix} $$

Verkettung von Transformationen:

$$ \begin{bmatrix} {}^0\vec{r} \\ 1 \end{bmatrix} = {}^0_1\underline{T} \cdot {}^1_2\underline{T} \cdots {}^{n-1}_n\underline{T} \cdot \begin{bmatrix} {}^n\vec{r} \\ 1 \end{bmatrix} = {}^0_n\underline{T} \cdot \begin{bmatrix} {}^n\vec{r} \\ 1 \end{bmatrix} $$

Denavit-Hartenberg Parameter



4 Parameter von einem Gelenk zum nächsten:

- α_{i-1} = Winkel(Z_{i-1}, Z_i) um X_{i-1} Achse → Rotation
- a_{i-1} = Distanz(Z_{i-1}, Z_i) entlang X_{i-1} Achse → Translation
- θ_i = Winkel(X_{i-1}, X_i) um Z_i Achse → Rotation
- d_i = Distanz(X_{i-1}, X_i) entlang Z_i Achse → Translation

damit kann wiederum die entsprechende Transformationsmatrix berechnet werden:

$$ {}^{i-1}_i\underline{T} = \underline{T}_{rot}(x, \alpha_{i-1}) \cdot \underline{T}_{trans}(x, a_{i-1}) \cdot \underline{T}_{rot}(z, \theta_i) \cdot \underline{T}_{trans}(x, d_i) $$

$$ = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) & 0 & a_{i-1} \\ \cos(\alpha_{i-1}) \cdot \sin(\theta_i) & \cos(\alpha_{i-1}) \cdot \cos(\theta_i) & -\sin(\alpha_{i-1}) & -d_i \cdot \sin(\alpha_{i-1}) \\ \sin(\alpha_{i-1}) \cdot \sin(\theta_i) & \sin(\alpha_{i-1}) \cdot \cos(\theta_i) & \cos(\alpha_{i-1}) & d_i \cdot \cos(\alpha_{i-1}) \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

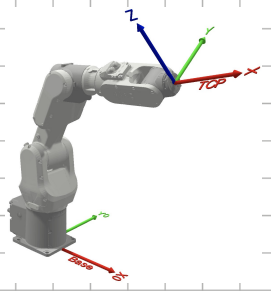
Vorteile:

- Transformation zw. 2 Körpern d. Roboters kann mit nur 4 Parametern beschrieben werden
- werden von vielen Software-Tools für Modellierung von Robotern verwendet

Nachteile gegenüber direkten Definitionen von 4×4 Transformationsmatrizen:

- Platzieren d. Koordinatensysteme n. trivial
- Verfahren ist n. eindeutig: gibt verschiedene Definitionen und es sind auch mehrere Lösungen möglich
- Basis- und Werkzeug-Koordinatensysteme n. vorgesehen und müssen separat berücksichtigt werden

Vorwärtskinematik



mit Gleichungen der Vorwärtskinematik können Position und Orientierung des Werkzeuges aus bekannten (gemessenen) Gelenkwinkel berechnet werden:

$$\vec{X} = f(\vec{\theta})$$

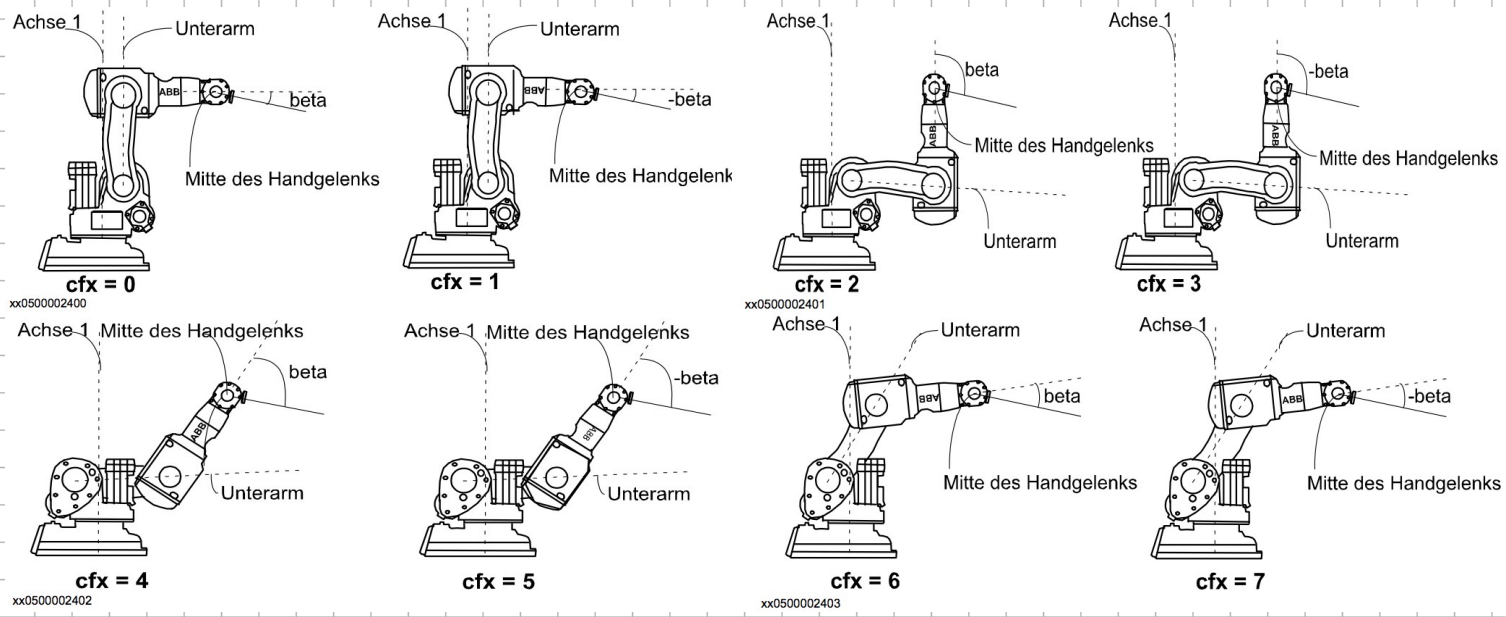
Rückwärtskinematik

mit Gleichungen der Rückwärtskinematik können Gelenkwinkel aus gewünschter Position und Orientierung des Werkzeuges berechnet werden:

$$\vec{\theta} = f^{-1}(\vec{X})$$

Industrieroboter oft so konstruiert dass Gleichungen symbolisch hergeleitet werden können

gibt meistens mehrere Lösungen, beim ABB IRB1100 gibt es 8 Konfigurationen:



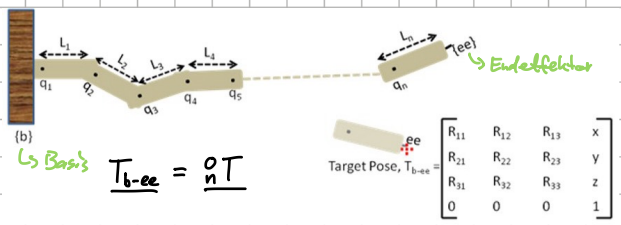
bei kartesischen Bewegungen muss man beachten, dass Anfangs- und Endlage dieselbe Konfiguration haben, da beim Wechsel von Konfigurationen singuläre Lagen auftreten!

Parallelroboter

Vorgehensweise & Schwierigkeiten bei Herleitung der Koordinatentransformation ist bei Parallelrobotern umgekehrt wie bei Robotern mit seriellen kinematischen Ketten:

- Gleichungen der Rückwärtskinematik können normalerweise relativ einfach hergeleitet werden
- Gleichungen der Vorwärtskinematik können i.A. nicht in symbolischer Form hergeleitet werden

Geschwindigkeiten



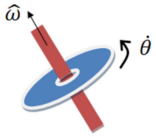
Werkzeuggeschwindigkeit aus Gelenkgeschwindigkeiten:

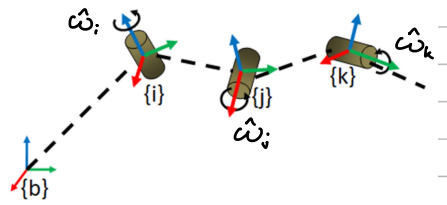
$$\vec{x} = f(\vec{\theta}) \quad \Rightarrow \quad \boxed{\dot{\vec{x}} = \underline{Jf}(\vec{\theta}) \cdot \dot{\vec{\theta}}}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\gamma} \end{bmatrix} = \begin{bmatrix} \underline{J_v} \\ \underline{J_\omega} \end{bmatrix} \cdot \begin{bmatrix} \dot{\theta}_1 \\ \vdots \\ \dot{\theta}_n \end{bmatrix}$$

$$\underline{J_v} = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \cdots & \frac{\partial x}{\partial \theta_n} \\ \frac{\partial y}{\partial \theta_1} & \cdots & \frac{\partial y}{\partial \theta_n} \\ \frac{\partial z}{\partial \theta_1} & \cdots & \frac{\partial z}{\partial \theta_n} \end{bmatrix} \in \mathbb{R}^{3 \times n}$$

$$\underline{J_\omega} = \begin{bmatrix} \hat{\omega}_1^b & \cdots & \hat{\omega}_n^b \end{bmatrix} \in \mathbb{R}^{3 \times n}$$





$\hat{\omega}_i^b$ ist die Rotationsachse des i-ten Gelenks in Bezug auf das Basis-Koordinatensystem

$\hat{\omega}_i^b$ entspricht den ersten 3 Einträgen derjenigen Spalte von $\underline{{}^0_iT}$, dessen korrespondierende Achse der Rotationsachse entspricht, denn es gilt $\hat{\omega}_i^b = \underline{{}^b_iR}\,\hat{\omega}_i$ und $\hat{\omega}_i$ ist ein Standardbasis-Vektor → pickt die richtige Spalte heraus

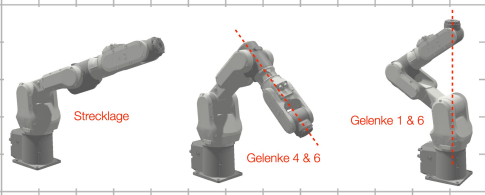
Gelenkgeschwindigkeiten aus Werkzeuggeschwindigkeit:

$$\boxed{\dot{\vec{\theta}} = \underline{Jf^{-1}}(\vec{\theta}) \cdot \dot{\vec{x}}}$$

dazu muss Inverse der Jakobi berechnet werden, was nur möglich ist falls:
- Jakobi quadratisch ist
- Jakobi vollen Rang hat (Zeilen/Spalten v. lin. abh. voneinander) ⇒ Determinante darf nicht 0 sein

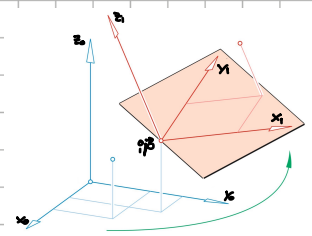
falls Jakobi in gewisser Lage n. invertierbar ist, nennt man dies eine singuläre Lage

die Beweglichkeit in kartesischen Koordinaten ist dann eingeschränkt



typische Singularitäten:
- Strecklagen
- 2 Drehachsen fluchten (schwieriger vorzubergen)

Roboterdynamik



Bewegungsgleichung eines Systems mit linearem Antrieb ist gegeben durch

$$F = ma + f_v v + f_c \operatorname{sign}(v) + g$$ «inverse Dynamik»

wobei $f_{v/c}$ den viskosen / Coulombschen Reibungskoeffizient und g das Gewicht bezeichnet

inverse Dynamik berechnet Antriebskraft für gegebene Bewegung

Anwendungen:
- Auslegung von Antrieben
- Vorsteuerung zur Verbesserung des Bahnfolgeverhaltens (Regelung mit Kompensation der dynamischen Kräfte)

Bewegungsgleichung kann man nach a auflösen und erhält

$$a = \frac{1}{m}\left(F - f_v v - f_c \operatorname{sign}(v) - g\right)$$ «direkte Dynamik»

welche die Bewegung für gegebene Antriebskraft berechnet

Anwendung: z.B. Simulation von Systemen

Mehrkörpersystem

allgemeine Form der Bewegungsgleichung für Mehrkörpersysteme ist gegeben durch

$$\vec{\tau} = \underline{M}(\vec{\theta})\ddot{\vec{\theta}} + \underline{V}(\vec{\theta}, \dot{\vec{\theta}})\dot{\vec{\theta}} + \vec{F}(\dot{\vec{\theta}}) + \vec{G}(\vec{\theta}) + \underline{JX}(\vec{\theta})\vec{f}$$

wobei $\underline{M}$ die Massenmatrix, $\vec{\theta}$ die Gelenkwinkel, $\vec{V}$ geschwindigkeitsabh. Terme, $\vec{F}$ Reibungsmomente, $\vec{G}$ Gewichtsmomente, $\underline{JX}$ die Jakobi-Matrix der Vorwärtskinematik und $\vec{f}$ den Kraftvektor am TCP (Tool Centre Point) darstellen

auch diese Gleichung kann in die (für die Berechnung der Bewegung bei gegebenen Gelenkmomente benötigte) direkte Dynamik umgeformt werden:

$$\ddot{\vec{\theta}} = \underline{M}(\vec{\theta})^{-1}\left(\vec{\tau} - \underline{V}(\vec{\theta}, \dot{\vec{\theta}})\dot{\vec{\theta}} - \vec{F}(\dot{\vec{\theta}}) - \vec{G}(\vec{\theta}) - \underline{JX}(\vec{\theta})\vec{f}\right)$$

Bewegungsgleichungen von Mehrkörpersystemen in kartesischen Koordinaten:

$$\vec{f} = \underline{M}(\vec{x})\ddot{\vec{x}} + \underline{V}(\vec{x}, \dot{\vec{x}})\dot{\vec{x}} + \vec{F}(\dot{\vec{x}}) + \vec{G}(\vec{x})$$

$$\vec{\tau} = \underline{JX}(\vec{x})^T\left(\underline{M}(\vec{x})\ddot{\vec{x}} + \underline{V}(\vec{x}, \dot{\vec{x}})\dot{\vec{x}} + \vec{F}(\dot{\vec{x}}) + \vec{G}(\vec{x})\right)$$

Kalman-Filter

Problem klassische Filter: bei stark verrauschten Signalen braucht Tiefpass => phasenverschoben
↳ (eher) Ansatz: modellbasierte Filter

normalerweise setzt lineares System (in ZDR) voraus, gibt aber auch modifizierte Kalman-Filter (Extended Kalman Filter) für nichtlineare Systeme (z.B. kinematik mobiler Roboter)

Lineares System:

$$\vec{x}_k = \underline{A}\,\vec{x}_{k-1} + \underline{B}\,\vec{u}_{k-1} + \vec{w}_k$$
$$\vec{z}_k = \underline{H}\,\vec{x}_{k-1} + \vec{v}_k$$

$\vec{x}_k$ Zustandsvektor
$\vec{u}_k$ Steuergrösse
$\vec{z}_k$ Messungen
$\vec{w}_k$ Prozessrauschen
$\vec{v}_k$ Messrauschen

Prozessrauschen $\vec{w}_{k-1}$ und Messrauschen $\vec{v}_k$ sind aber nicht bekannt -> Gleichungen können in dieser Form also nicht berechnet werden

↳ Rauschen muss mit Kovarianzmatrizen beschrieben werden ($\vec{w}_k \to \underline{Q}$, $\vec{v}_k \to \underline{R}$ (σ Sensoren)):

$$\underline{Q}_k = \underline{B}\,E\{\vec{w}_k\,\vec{w}_k^T\}\,\underline{B}^T$$
$$\underline{R}_k = E\{\vec{v}_k\,\vec{v}_k^T\}$$

Prädiktion

$$\hat{\vec{x}}_k = \underline{A}\,\hat{\vec{x}}_{k-1} + \underline{B}\,\vec{u}_{k-1}$$ Schätzung Zustand
$$\underline{P}_k = \underline{A}\,\underline{P}_{k-1}\,\underline{A}^T + \underline{Q}$$ Kovarianz-Matrix Zustand (Sicherheit Schätzung)

Korrektur

$$\underline{K}_k = \underline{P}_k\,\underline{H}^T(\underline{H}\,\underline{P}_k\,\underline{H}^T + \underline{R})^{-1}$$ Kalman-Matrix
$$\hat{\vec{x}}_{k,korr} = \hat{\vec{x}}_k + \underline{K}_k(\vec{z}_k - \underline{H}\,\hat{\vec{x}}_k)$$ Korrektur Schätzung Zustand
$$\underline{P}_{k,korr} = (\underline{I} - \underline{K}_k\,\underline{H})\,\underline{P}_k$$ Korrektur Kovarianzmatrix-Matrix Zustand

Tuning

mit Kovarianz-Matrizen $\underline{Q}$ (↳ Prozessrauschen) und $\underline{R}$ (↳ Messrauschen)

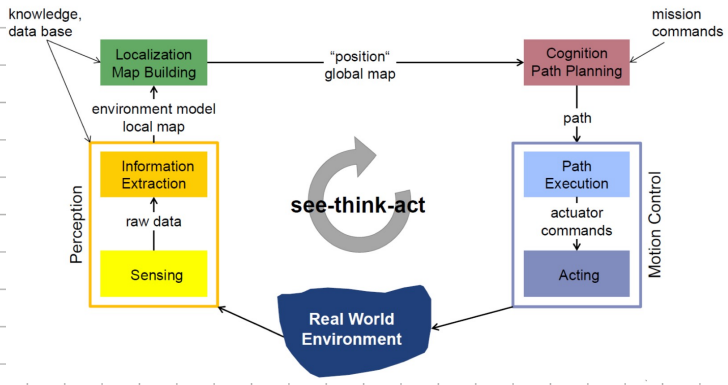
$\underline{R}$ → Standardabweichungen Sensoren

$\underline{Q}$ → schätzen oder (mit etwas Aufwand) experimentell bestimmen

Mobile Roboter

3 zentrale Fragen:

- Where am I? → Lokalisierung
- Where am I going?
- How do I get there? → Navigation



Navigation := automatische Bestimmung Weg aktuelle Position zu Ziel

ohne Karte (reaktive Navigation)

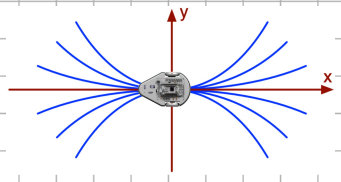
Bug-Algorithmus:
- gerade Linie von Start- zu Zielposition berechnen
- Bewegung entlang Linie
- bei Hindernis folgt im Uhrzeigersinn dem Hindernis bis wieder auf Linie
- wenn Position auf Linie näher am Ziel als beim Verlassen der Linie → folgt wieder Linie

mit Gitterkarte

Distanztransformation:
- von Ziel aus den befahrbaren Zellen einen Distanzwert zuweisen (+= 1 für direkte, += √2 für diagonale Nachbarn)
- Bewegung entlang Gradient der Distanzwerte
- rechenaufwändig
- findet optimalen/kürzesten Weg vom Start zum Ziel

Probabilistic Roadmap Method (PRM):
- Möglichkeit den grossen Rechenaufwand d. Distanztransformation zu reduzieren
- zufällig Knoten einer Netzkarte innerhalb Gitterkarte verteilen (nicht auf Hindernisse)
- geradlinige Verbindungen mit Distanzangaben zwischen allen Knoten (sofern keine Hindernisse dazwischen)
- Berechnung kürzester Weg entlang Verbindungen

Rapidly-Exploring Random Tree (RRT):
- Verbindungen müssen auf Kreissegmenten liegen
- für beliebige Kinematik geeignet



mit topologischer Karte (Netzkarte)

topologische Karte = gerichteter (mit Distanz oder irgendeinem anderen quantifizierbaren Mass (Fahrzeit, Energieverbrauch, Kosten) Graph

Algorithmus von Dijkstra:
- allen Knoten kürzeste Distanz zum Startknoten zuweisen
  - ↳ allenfalls korrigieren
- für jeden Knoten muss man sich merken können, von welchem benachbarten Knoten aus dieser Knoten erreicht werden muss, damit die Distanzangabe dieses Knotens gültig ist

```
1  function Dijkstra(Graph, source):
2
3      for each vertex v in Graph.Vertices:
4          dist[v] ← INFINITY
5          prev[v] ← UNDEFINED
6          add v to Q
7      dist[source] ← 0
8
9      while Q is not empty:
10         u ← vertex in Q with minimum dist[u]
11         remove u from Q
12
13         for each neighbor v of u still in Q:
14             alt ← dist[u] + Graph.Edges(u, v)
15             if alt < dist[v]:
16                 dist[v] ← alt
17                 prev[v] ← u
18
19     return dist[], prev[]
```