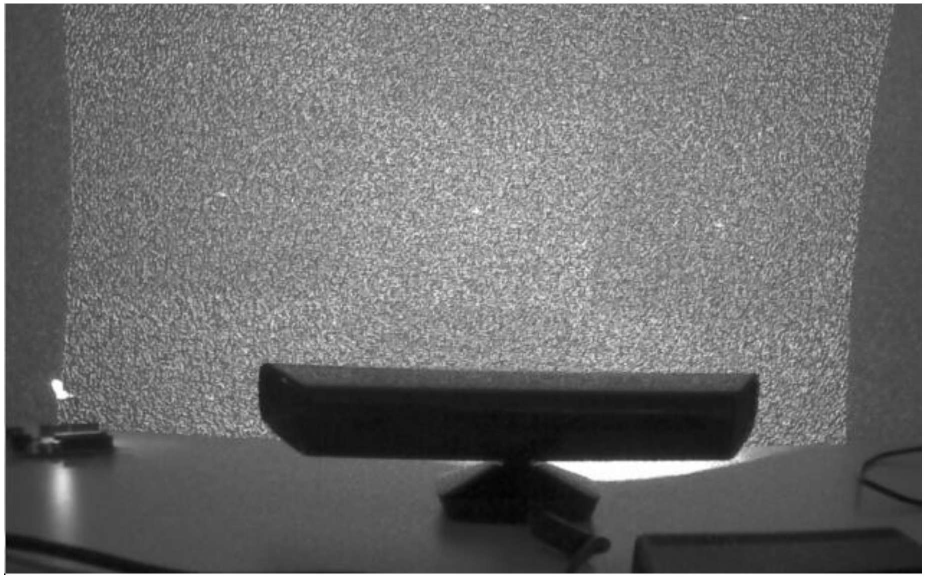
Advanced Robot Perception

Fortgeschrittene Konzepte der Wahrnehmung für Robotersysteme

Georg von Wichert, Siemens Corporate Technology

Abstand: RGB-D Kameras

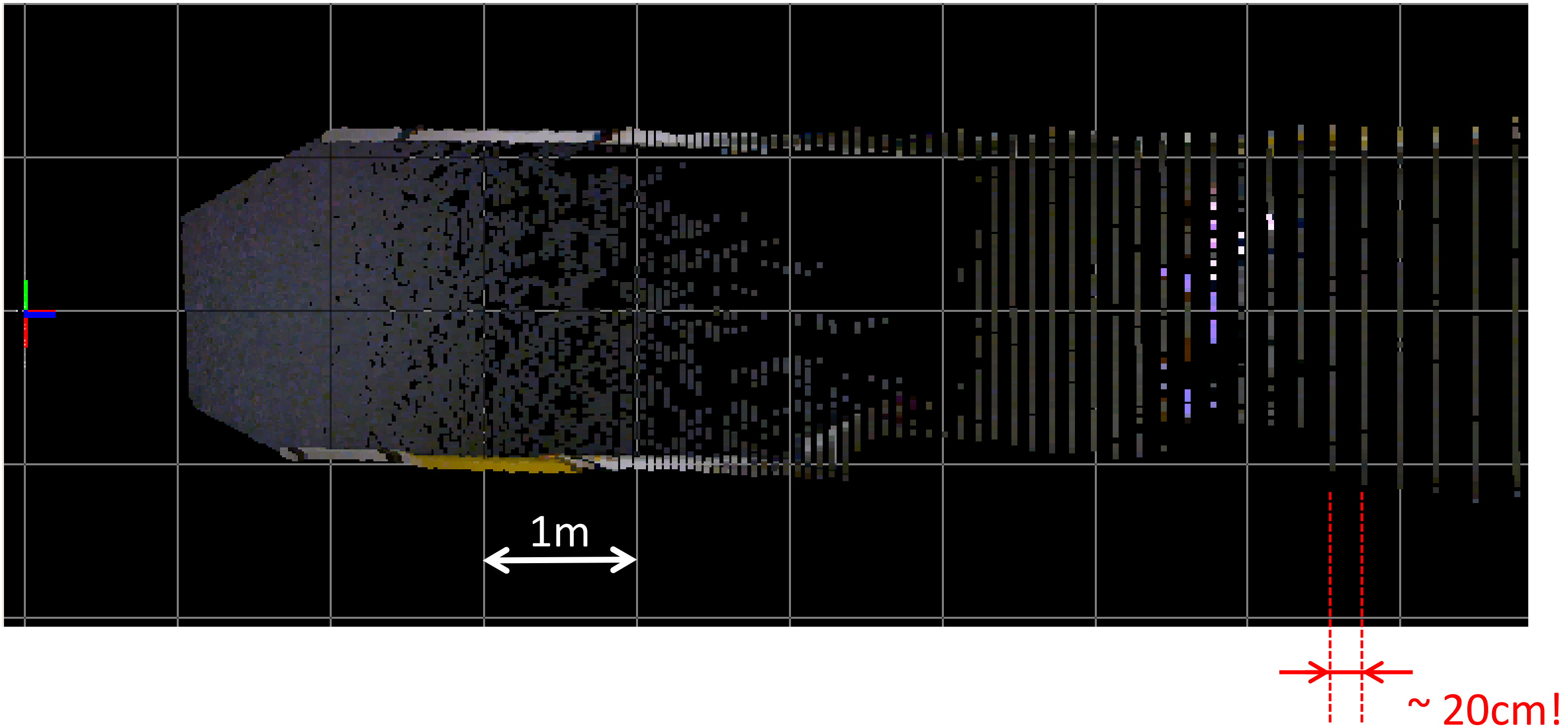
- Microsoft / Primesense
  - Infrarotprojektor
  - Infrarotkamera
  - Farbkamera
- Hohe Auflösung (640x480 -> 320x240)
- Bildfrequenz 30 Hz
- Reichweite: 0.8m bis etwa 4m
- gute Tiefendaten
- sehr winkeltreu
- Pixelgenaues Farbbild
- aber empfindlich gegen Fremdlicht (Sonne)
- Farbbild etwa wie bei billigen Webcams



Kodiertes Strukturmuster (Infrarot)

Abstand: RGB-D Kameras

- Mit dem Messabstand steigender Distanzfehler
  - Ähnliche Fehler machen alle Stereokameras
  - Kinect hat deutliches Diskretisierungsrauschen



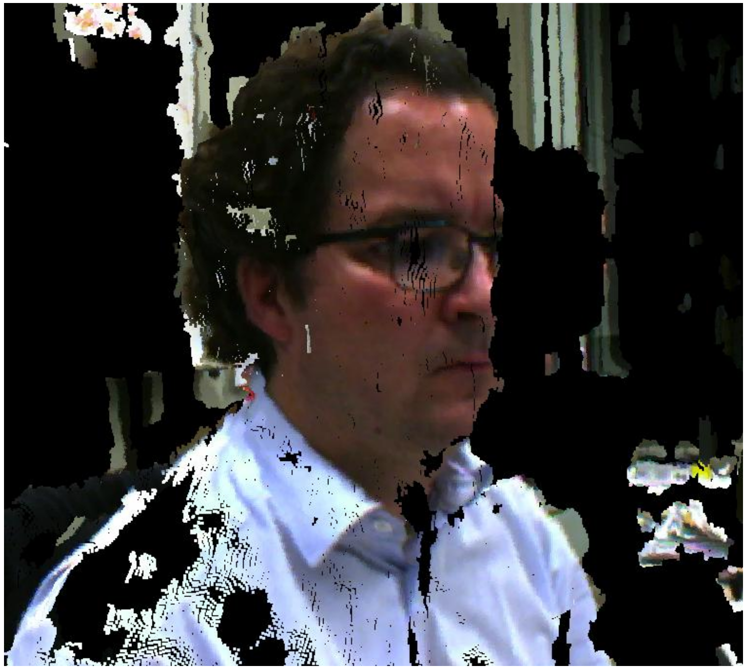
Abstand: Laser 3D (ToF-) Kameras

- Lichtlaufzeitmessung
  - Lichtpulse
- Auflösung typisch 200 x 200 Pixel
- Reichweite: wenige Meter
- Abstandsmessung stark vom Ziel anhängig
- Starke systematische Fehler insbesondere in Ecken (Mehrfachreflexion)
- Cross Talk bei mehreren Sensoren
- Empfindlich gegen Fremdlicht

Vergleich Kinect vs. ToF-Kamera

Abstand: Stereokameras

- Hohe Auflösung (Kameraabhängig)
- Korrespondenzsuche erfordert Struktur im Bild
- Keine 3D-Daten von homogenen Flächen



2D- UND 3D-GEOMETRIE

Geometrische Primitive in 2D

- 2D-Punkt

$$\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{R}^2$$

- Augmentierter Vektor

$$\bar{\mathbf{x}} = \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \in \mathbb{R}^3$$

- Homogene Koordinaten

$$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} \in \mathbb{P}^2$$

Geometrische Primitive in 2D

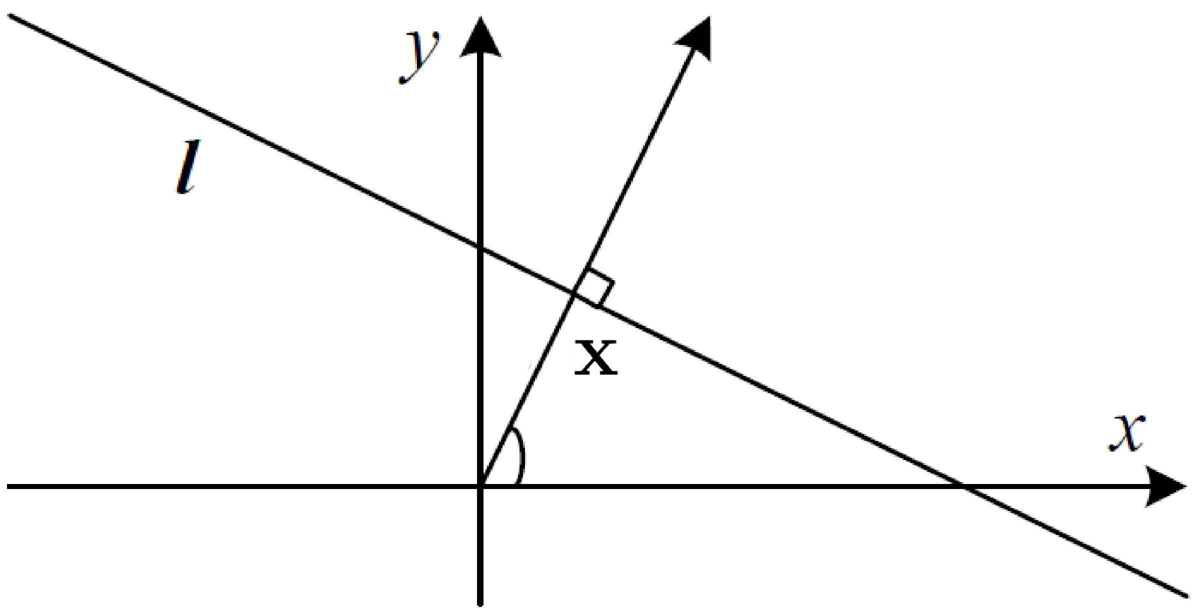
- Homogene Vektoren, die sich nur in der Skalierung unterscheiden, repräsentieren denselben 2D-Punkt
- Konvertierung in inhomogene Koordinaten über Division durch das letzte Element

$$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{w} \end{pmatrix} = \begin{pmatrix} \tilde{x}/\tilde{w} \\ \tilde{y}/\tilde{w} \\ 1 \end{pmatrix} = \tilde{w} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \tilde{w}\bar{\mathbf{x}}$$

- Punkte mit $\tilde{w} = 0$ werden als „ideale Punkte“ bezeichnet, sie liegen im Unendlichen und repräsentieren eine Richtung

Geometrische Primitive in 2D

- 2D Gerade $\quad \tilde{\mathbf{l}} = (a, b, c)^\top$
- 2D Geradengleichung $\quad \bar{\mathbf{x}} \cdot \tilde{\mathbf{l}} = ax + by + c = 0$

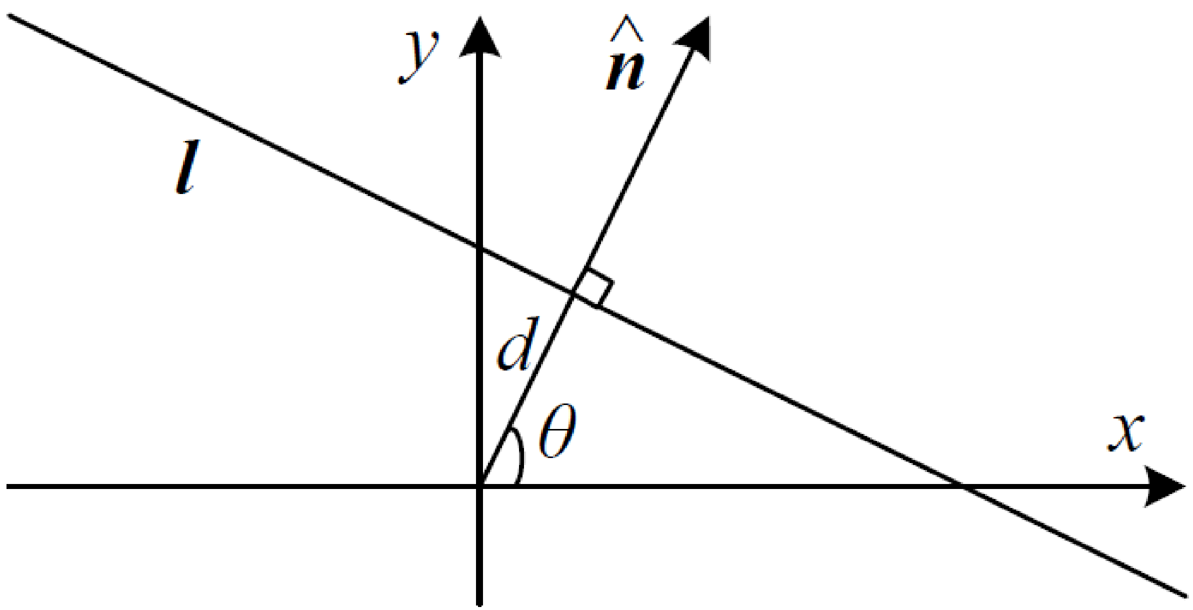


Geometrische Primitive in 2D

- Vektorielle Darstellung über die normierte Normale

$$\tilde{\mathbf{l}} = (\hat{n}_x, \hat{n}_y, d)^\top = (\hat{\mathbf{n}}, d)^\top \quad \text{with} \quad \|\hat{\mathbf{n}}\| = 1$$

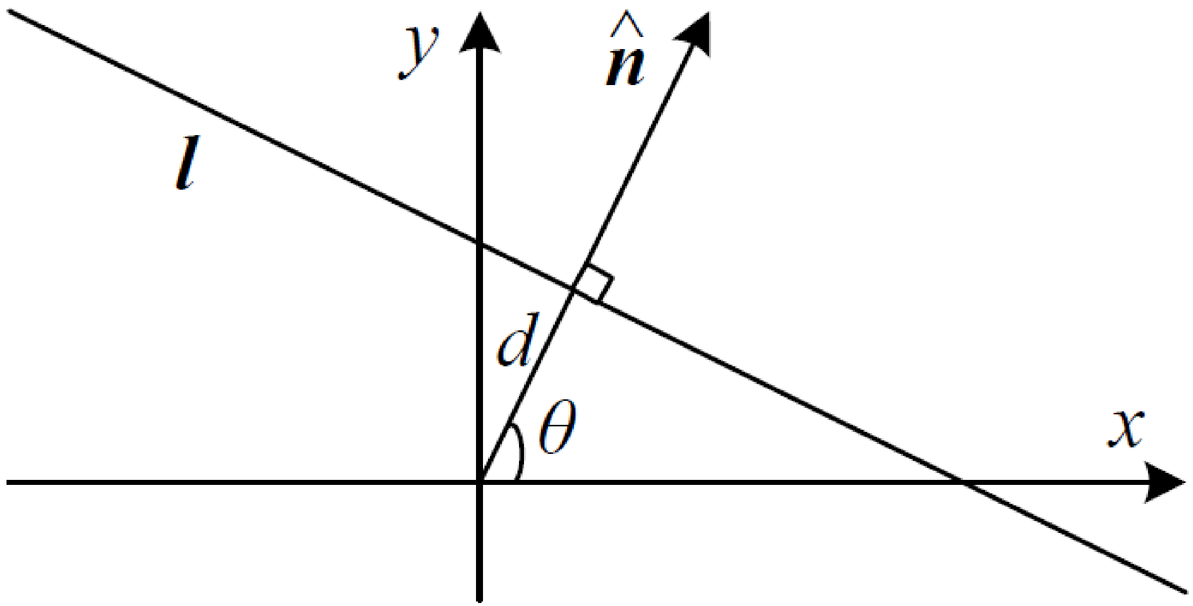
wobei d der Abstand der Gerade zum Ursprung ist



Geometrische Primitive in 2D

- Polarkoordinaten der Gerade: $(\theta, d)^\top$
  (wird z.B. bei der Houghtransformation verwendet)

$$\hat{\mathbf{n}} = (\cos\theta, \sin\theta)^\top$$



Geometrische Primitive in 2D

- Gerade durch zwei Punkte $\tilde{\mathbf{l}} = \tilde{\mathbf{x}}_1 \times \tilde{\mathbf{x}}_2$
- Schnittpunkt zweier Geraden $\tilde{\mathbf{x}} = \tilde{\mathbf{l}}_1 \times \tilde{\mathbf{l}}_2$

Geometrische Primitive in 3D

- 3D-Punkt

$$\mathbf{x} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \in \mathbb{R}^3$$

- Augmentierter Vektor

$$\bar{\mathbf{x}} = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \in \mathbb{R}^4$$

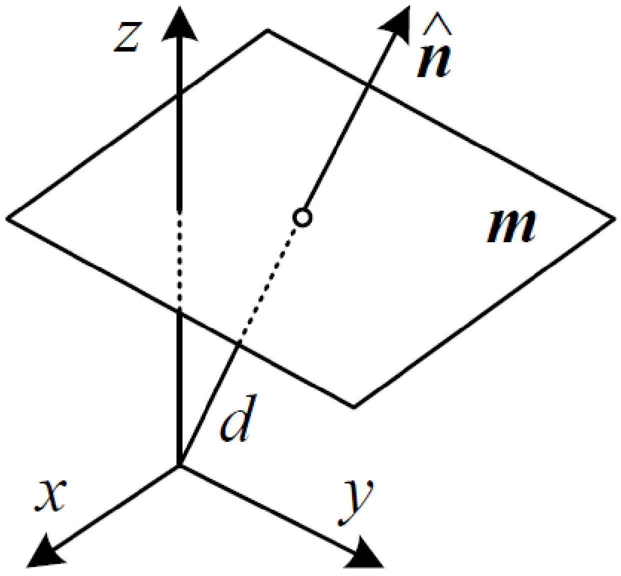
- Homogene Koordinaten

$$\tilde{\mathbf{x}} = \begin{pmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \\ \tilde{w} \end{pmatrix} \in \mathbb{P}^3$$

Geometrische Primitive in 3D

- 3D Ebene $\tilde{\mathbf{m}} = (a, b, c, d)^\top$
- 3D Ebenengleichung $\bar{\mathbf{x}} \cdot \tilde{\mathbf{m}} = ax + by + cz + d = 0$

- Normalisierte Ebene
  mit normalenvektor
  $\mathbf{m} = (\hat{n}_x, \hat{n}_y, \hat{n}_z, d)^\top = (\hat{\mathbf{n}}, d)$
  $(\|\hat{\mathbf{n}}\| = 1)$
  und Abstand d



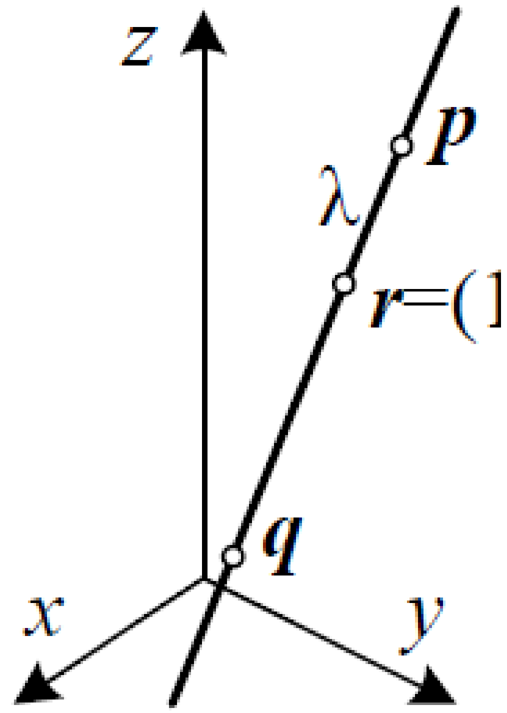
Geometrische Primitive in 3D

- 3D Gerade $\mathbf{r} = (1-\lambda)\mathbf{p} + \lambda\mathbf{q}$ durch die Punkte $\mathbf{p}, \mathbf{q}$

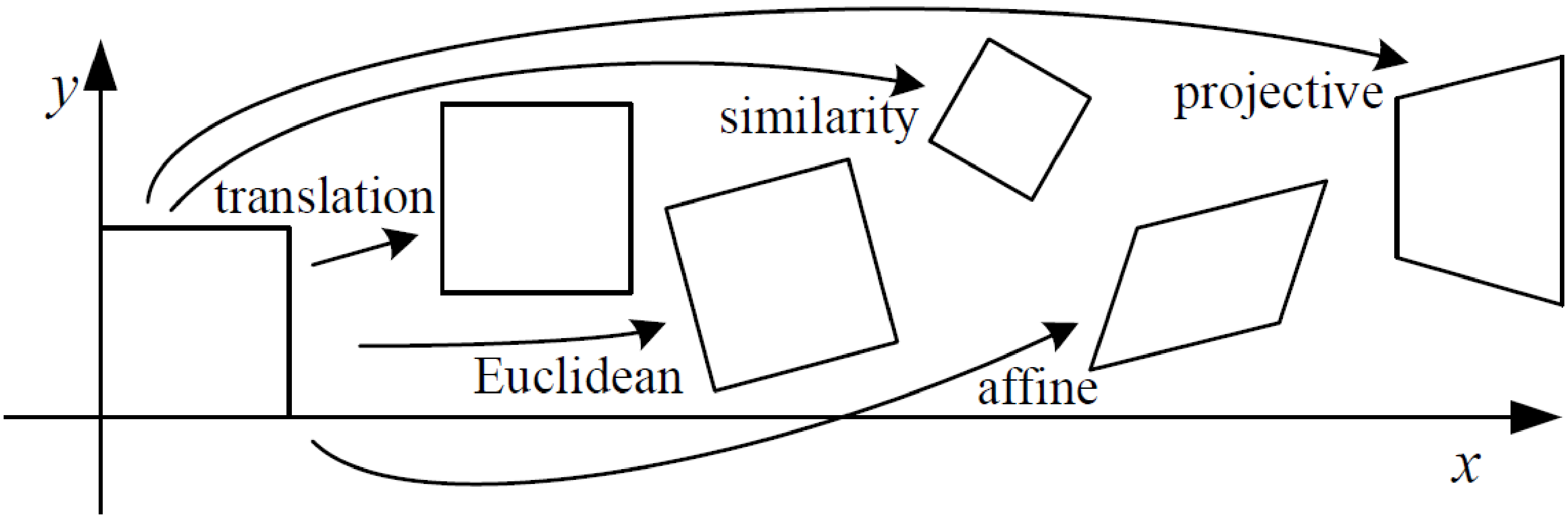
- Unendliche Gerade: $\lambda \in \mathbb{R}$

- Geradenstück zwischen $\mathbf{p}, \mathbf{q}$:

$$0 \leq \lambda \leq 1$$



Planare Transformationen in 2D



2D-Transformationen

- Translation

$$\mathbf{x}' = \mathbf{x} + \mathbf{t}$$

$$\mathbf{x}' = \underbrace{\begin{pmatrix} \mathbf{I} & \mathbf{t} \end{pmatrix}}_{2\times 3} \bar{\mathbf{x}}$$

$$\bar{\mathbf{x}}' = \underbrace{\begin{pmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix}}_{3\times 3} \bar{\mathbf{x}}$$

wobei $\mathbf{t} \in \mathbb{R}^2$ Translationsvektor, und
$\mathbf{I}$ die Einheitsmatrix ist. $\mathbf{0}$ ist der Nullvektor

2D-Transformationen

- Translation

$$\mathbf{x}' = \mathbf{x} + \mathbf{t}$$

$$\mathbf{x}' = \underbrace{\begin{pmatrix} \mathbf{I} & \mathbf{t} \end{pmatrix}}_{2\times 3} \bar{\mathbf{x}}$$

$$\bar{\mathbf{x}}' = \underbrace{\begin{pmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix}}_{3\times 3} \bar{\mathbf{x}}$$

Frage: Wieviele Freiheitsgrade hat diese Transformation?

wobei $\mathbf{t} \in \mathbb{R}^2$ Translationsvektor, und
$\mathbf{I}$ die Einheitsmatrix ist. $\mathbf{0}$ ist der Nullvektor

2D-Transformationen

- Strarre Bewegung or Euklidische Transformation (Rotation + Translation)

$$\mathbf{x}' = \mathbf{R}\mathbf{x} + \mathbf{t} \qquad \text{bzw.} \qquad \bar{\mathbf{x}}' = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \bar{\mathbf{x}}$$

wobei $\mathbf{R} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$

die orthonormale Rotationsmatrix ist, d.h. $\mathbf{R}\mathbf{R}^\top = \mathbf{I}$

- Abstände **und** Winkel bleiben erhalten

2D-Transformationen

- Ähnlichkeitstransformation (Skalierung, Rotation und Verschiebung)

$$\mathbf{x}' = s\mathbf{R}\mathbf{x} + t$$

or

$$\bar{\mathbf{x}}' = \begin{pmatrix} s\mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \bar{\mathbf{x}}$$

- Winkel zwischen Geraden bleiben erhalten

2D-Transformationen

- Affine Transformation

$$\bar{\mathbf{x}}' = A\bar{\mathbf{x}} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{pmatrix} \bar{\mathbf{x}}$$

- Parallele Geraden bleiben parallel

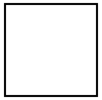
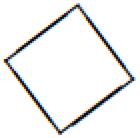
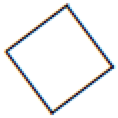
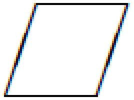
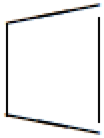
2D-Transformationen

- Projektive/perspektivische Transformation

$$\tilde{\mathbf{x}}' = \tilde{H} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \tilde{\mathbf{x}}$$

- Beachte $\tilde{H}$ ist homogen (only defined up to scale)
- Resultierende Koordinaten sind homogen
- Geraden bleiben Geraden, immerhin…. ;-)

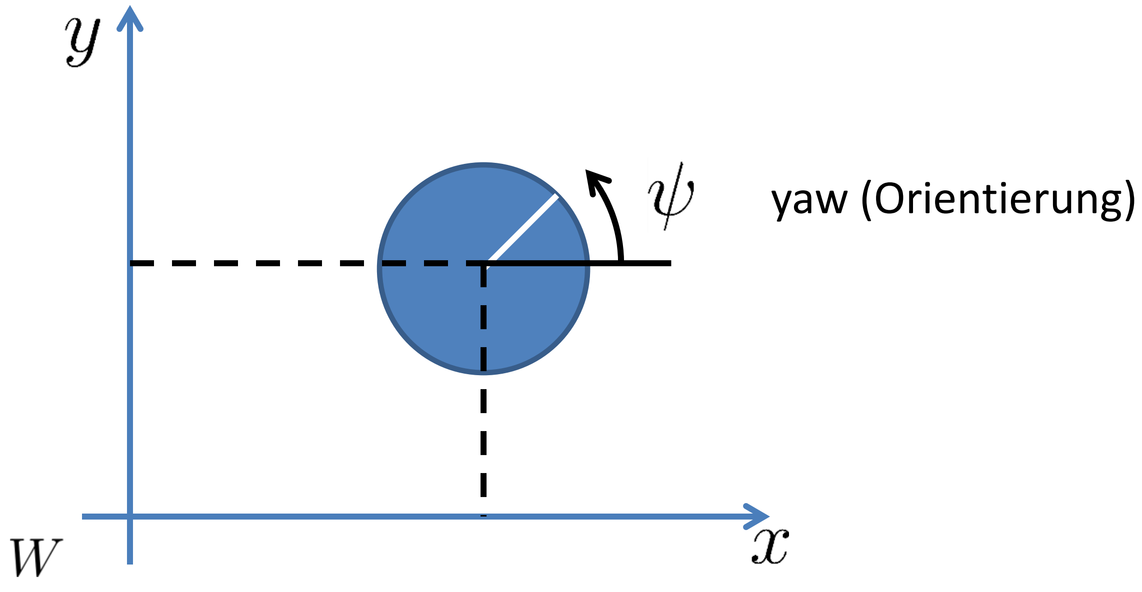
2D-Transformationen

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	2	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	3	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{2\times 3}$	4	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{2\times 3}$	6	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{3\times 3}$	8	straight lines	

BEISPIEL: VERWENDUNG EUKLIDISCHER TRANSFORMATIONEN IN DER ROBOTIK

Koordinatensysteme

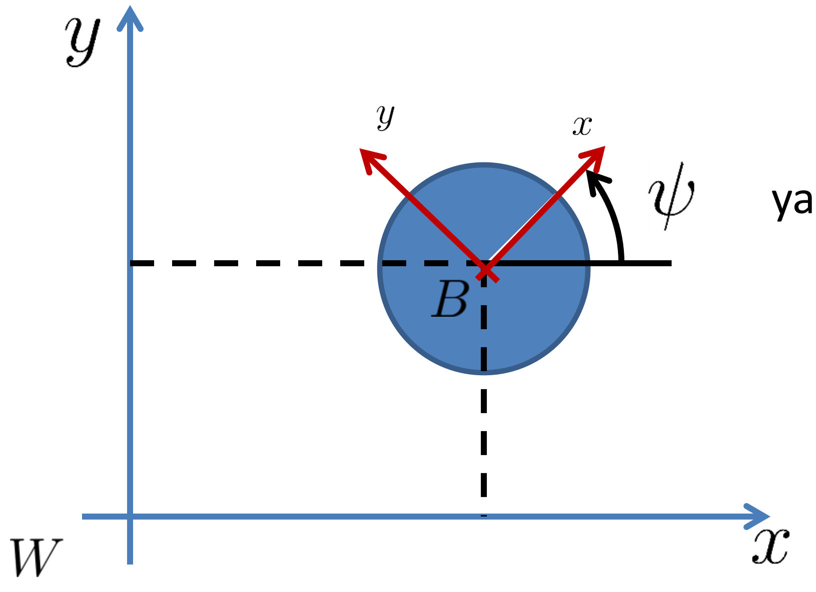
- Ein Roboter irgendwo in der Ebene



Koordinatensysteme

- Ein Roboter irgendwo in der Ebene

$$ {}^{W}T_{B} = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} = \begin{pmatrix} \cos\psi & -\sin\psi & x \\ \sin\psi & \cos\psi & y \\ 0 & 0 & 1 \end{pmatrix} \in \mathrm{SE}(2) \subset \mathbb{R}^{3x3} $$

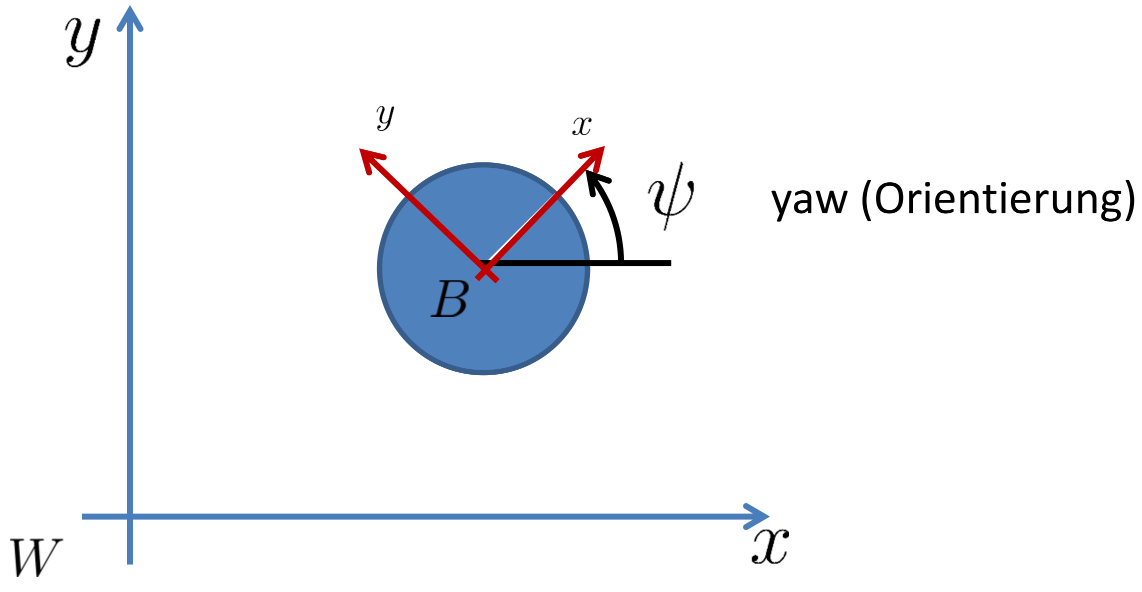


yaw (Orientierung)

Koordinatensysteme

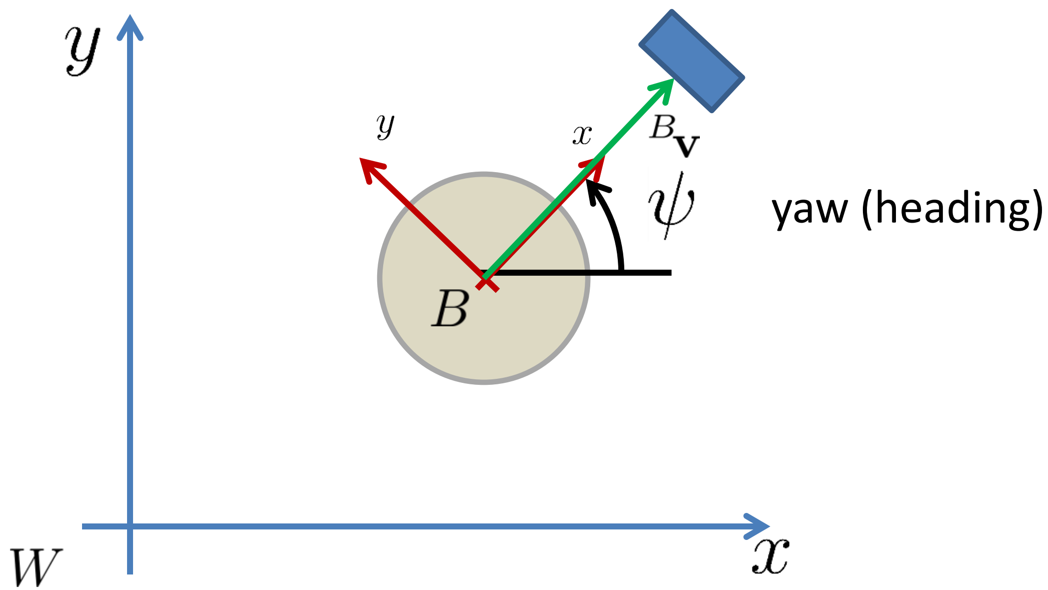
- Roboter an der Stelle x=0.7, y=0.5, yaw=45deg

$$
{}^{W}T_{B} = \begin{pmatrix} \cos 45 & -\sin 45 & 0.7 \\ \sin 45 & \cos 45 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.71 & -0.71 & 0.7 \\ 0.71 & 0.71 & 0.5 \\ 0 & 0 & 1 \end{pmatrix}
$$



Koordinatentransformation

- Roboter an der Stelle x=0.7, y=0.5, yaw=45deg
- Roboter hat einen Sensorausleger von 1m Länge
- Was ist die Position des Sensors in *W*-Koordinaten?



Koordinatentransformation

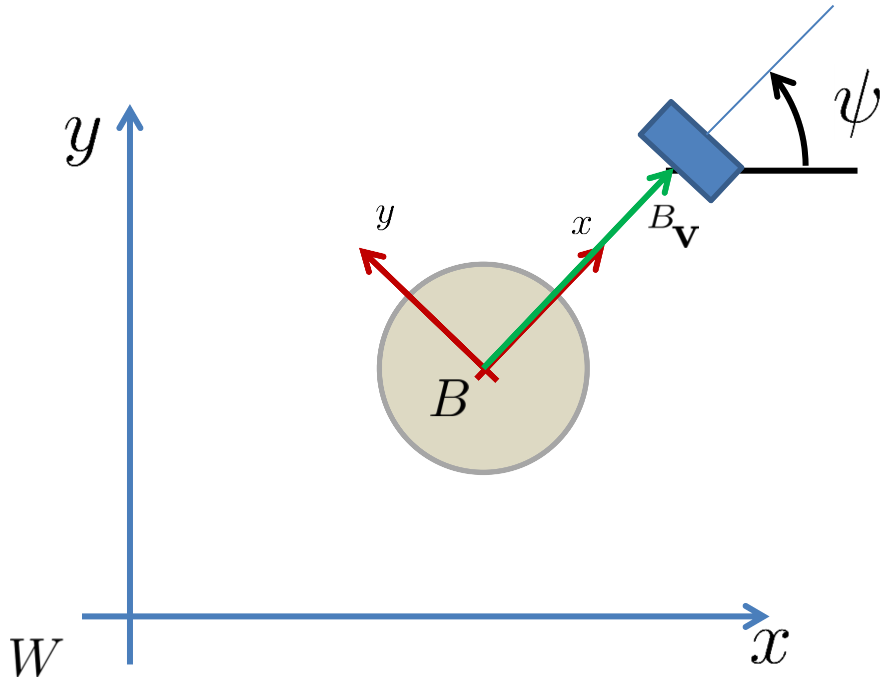
- Roboter an der Stelle x=0.7, y=0.5, yaw=45deg
- Roboter hat einen Sensorausleger von 1m Länge

Inhomogene Koordinaten

$$^{B}\mathbf{v} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Homogene Koordinaten

$$^{B}\tilde{\mathbf{v}} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$



Koordinatentransformation

- Roboter an der Stelle x=0.7, y=0.5, yaw=45deg
- Roboter detektiert Objekt 1m voraus

$$ {}^{W}\tilde{\mathbf{v}} = {}^{W}T_{B}\,{}^{B}\tilde{\mathbf{v}} $$

Koordinatentransformation

- Roboter an der Stelle x=0.7, y=0.5, yaw=45deg
- Roboter hat einen Sensorausleger von 1m Länge

$$
\begin{aligned}
{}^{W}\tilde{\mathbf{v}} &= {}^{W}T_{B}\,{}^{B}\tilde{\mathbf{v}} \\
&= \begin{pmatrix} 0.71 & -0.71 & 0.7 \\ 0.71 & 0.71 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}
\end{aligned}
$$

Koordinatentransformation

- Roboter an der Stelle x=0.7, y=0.5, yaw=45deg
- Roboter hat einen Sensorausleger von 1m Länge

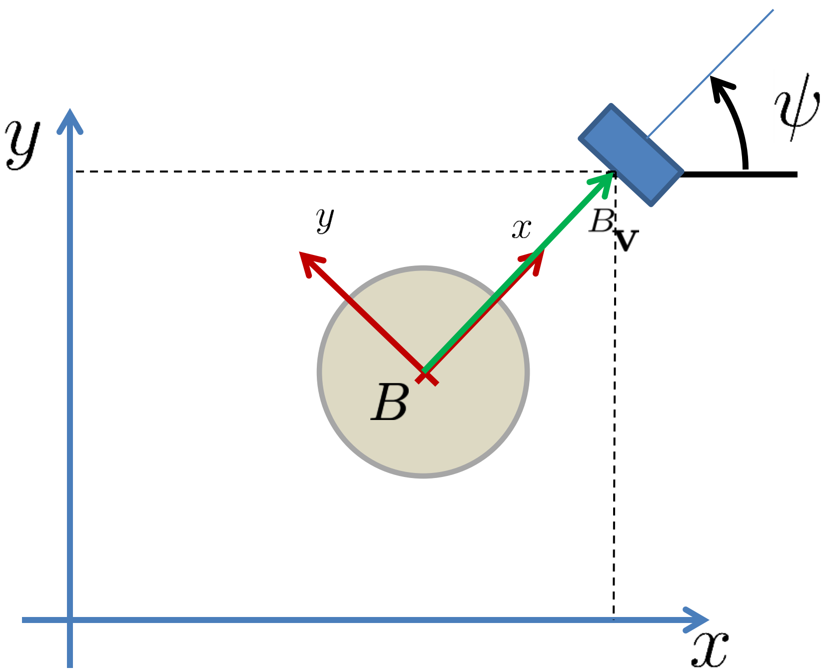
$$
\begin{aligned}
{}^{W}\tilde{\mathbf{v}} &= {}^{W}T_{B}\,{}^{B}\tilde{\mathbf{v}} \\
&= \begin{pmatrix} 0.71 & -0.71 & 0.7 \\ 0.71 & 0.71 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} 1.41 \\ 1.21 \\ 1 \end{pmatrix}
\end{aligned}
$$

Koordinatenransformation

- Roboter an der Stelle x=0.7, y=0.5, yaw=45deg
- Roboter hat einen Sensorausleger von 1m Länge

$$^{B}\tilde{\mathbf{v}} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix}$$

$$^{W}\tilde{\mathbf{v}} = \begin{pmatrix} 1.41 \\ 1.21 \\ 1 \end{pmatrix}$$



Inverse Transformation

- Wir haben Roboter- in Weltkoordinaten transformiert
- Manchmal muss man das Umgekehrte tun
- Wie transformiert man Welt- in Roboterkoordinaten?

$$ {}^{W}\tilde{\mathbf{v}} = {}^{W}T_{B}\,{}^{B}\tilde{\mathbf{v}} $$

Inverse Transformation

- Wir haben Roboter- in Weltkoordinaten transformiert
- Manchmal muss man das Umgekehrte tun
- Wie transformiert man Welt- in Roboterkoordinaten?

$$ {}^{W}\tilde{\mathbf{v}} = {}^{W}T_{B}\,{}^{B}\tilde{\mathbf{v}} $$

$$ {}^{B}\tilde{\mathbf{v}} = {}^{B}T_{W}\,{}^{W}\tilde{\mathbf{v}} $$

Inverse Transformation

- Wir haben Roboter- in Weltkoordinaten transformiert
- Manchmal muss man das Umgekehrte tun
- Wie transformiert man Welt- in Roboterkoordinaten?

$$ {}^{W}\tilde{\mathbf{v}} = {}^{W}T_{B}\,{}^{B}\tilde{\mathbf{v}} $$

$$ {}^{B}\tilde{\mathbf{v}} = {}^{B}T_{W}\,{}^{W}\tilde{\mathbf{v}} $$

$$ {}^{B}\tilde{\mathbf{v}} = \left({}^{W}T_{B}\right)^{-1}\,{}^{W}\tilde{\mathbf{v}} $$

Inverse Transformation

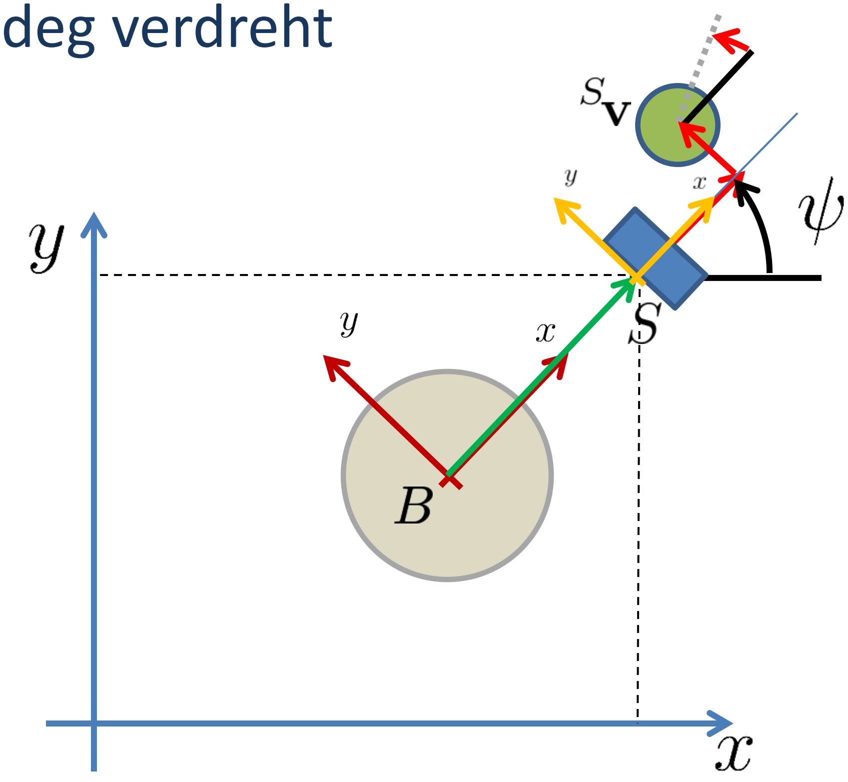
- Wir haben Roboter- in Weltkoordinaten transformiert
- Manchmal muss man das Umgekehrte tun
- Wie transformiert man Welt- in Roboterkoordinaten?

$$
{}^{W}\tilde{\mathbf{v}} = {}^{W}T_{B}\,{}^{B}\tilde{\mathbf{v}} = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} {}^{B}\tilde{\mathbf{v}}
$$

$$
{}^{B}\tilde{\mathbf{v}} = \left({}^{W}T_{B}\right)^{-1}\,{}^{W}\tilde{\mathbf{v}} = \begin{pmatrix} R^{\top} & -R^{\top}\mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} {}^{W}\tilde{\mathbf{v}}
$$

Verkettung von Transformationen

- Gegeben: Objekt wird von Sensor gesehen
  - Position relativ zum Sensor: 0.2m in x-Richtung, 0.1m in y-Richtung, 10deg verdreht



Verkettung von Transformationen

- Position relativ zum Sensor:
  0.2m in x-Richtung, 0.1m in y-Richtung, 10deg verdreht

$$^{B}T_{S} = \begin{pmatrix} \cos 10 & -\sin 10 & 0.2 \\ \sin 10 & \cos 10 & 0.1 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.98 & -0.17 & 0.2 \\ 0.17 & 0.98 & 0.1 \\ 0 & 0 & 1 \end{pmatrix}$$

Verkettung von Transformationen

- Die Position des Objekts in W-Koordinaten?

$${}^{W}T_S = {}^{W}T_B \, {}^{B}T_S =$$

$$= \begin{pmatrix} 0.71 & -0.71 & 0.7 \\ 0.71 & 0.71 & 0.5 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0.98 & -0.17 & 0.2 \\ 0.17 & 0.98 & 0.1 \\ 0 & 0 & 1 \end{pmatrix} = \cdots$$

Verkettung von Transformationen

Beachte: Auf die Reihenfolge kommt es an!!!

- 1m gehen, 90 Grad drehen
- 90 Grand drehen, 1m gehen

$$AB \neq BA$$

3D-Transformationen

- Translation

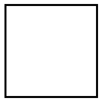
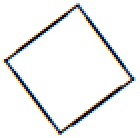
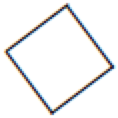
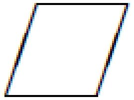
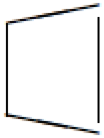
$$\bar{\mathbf{x}}' = \underbrace{\begin{pmatrix} \mathbf{I} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix}}_{4\times 4} \bar{\mathbf{x}}$$

- Euklidische Transformation (Translation + Rotation),

$$\bar{\mathbf{x}}' = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \bar{\mathbf{x}}$$

- Ähnlichkeitstransformation, Affine Transformation, …

3D-Transformationen

Transformation	Matrix	# DoF	Preserves	Icon
translation	$\begin{bmatrix} \boldsymbol{I} \mid \boldsymbol{t} \end{bmatrix}_{3\times 4}$	3	orientation	
rigid (Euclidean)	$\begin{bmatrix} \boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{3\times 4}$	6	lengths	
similarity	$\begin{bmatrix} s\boldsymbol{R} \mid \boldsymbol{t} \end{bmatrix}_{3\times 4}$	7	angles	
affine	$\begin{bmatrix} \boldsymbol{A} \end{bmatrix}_{3\times 4}$	12	parallelism	
projective	$\begin{bmatrix} \tilde{\boldsymbol{H}} \end{bmatrix}_{4\times 4}$	15	straight lines	

Euklidische Transformation in 3D

- Translation $\mathbf{t}$ hat 3 Freiheitsgrade
- Rotation R hat 3 Freiheitsgrade

$$X = \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3D-Rotationen

- Rotationsmatrix

- Euler-Winkel
- Rodriguez-Darstellung (Drehachse / Winkel)
- Einheitsquaternionen

Rotationsmatrix

- Orthonormale 3x3 Matrix

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

- Spaltenvektoren entsprechen den Koordinatenachsen

Rotationsmatrix

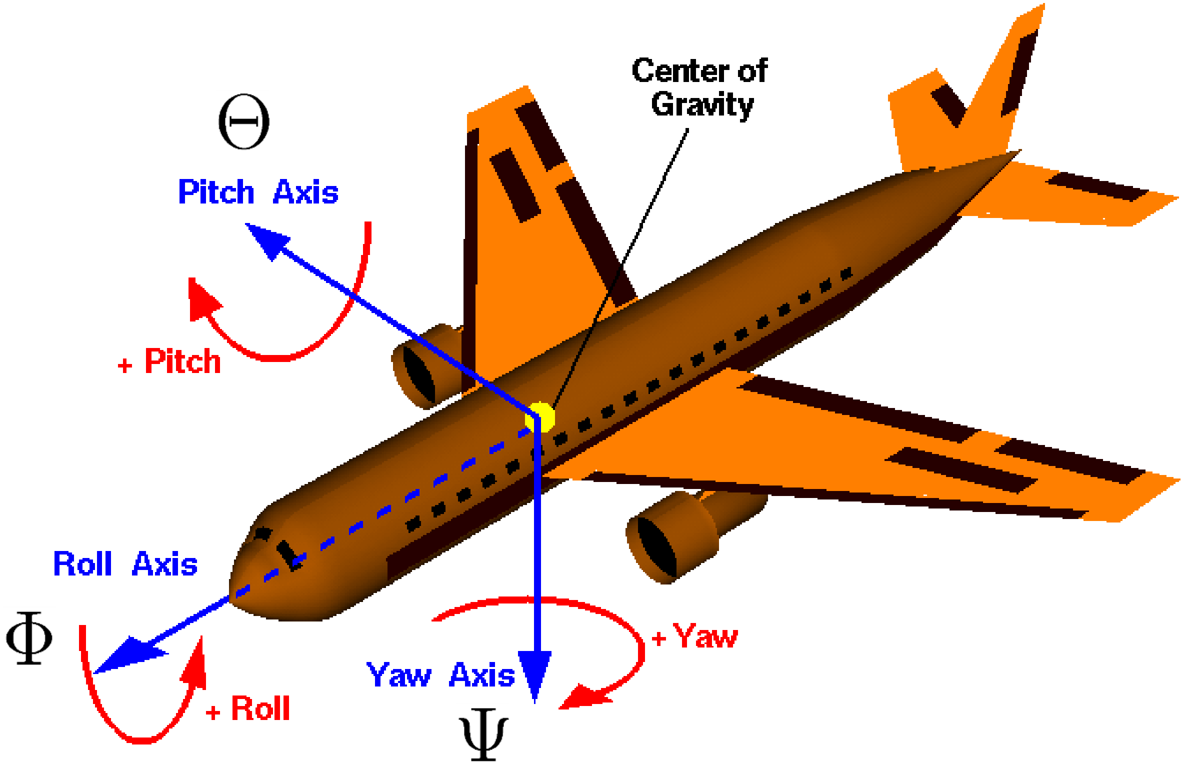
- Orthonormale 3x3 Matrix

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix}$$

- Vorteil: Einfache Verkettung
- Nachteil: Überparametrisiert (9 Parameter statt 3)

Euler-Winkel

- Rotation durch Verkettung von 3 Achsrotationen (z.B., um X-Y-Z Achsen)
- Aus der Luftfahrt: Roll-Pitch-Yaw Konvention



Roll-Pitch-Yaw Konvention

- Yaw Ψ , Pitch Θ , Roll Φ in Rotationsmatrix umrechnen

$$
\begin{aligned}
R &= R_Z(\Psi) R_Y(\Theta) R_X(\Phi) \\
&= \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\Phi & \sin\Phi \\ 0 & -\sin\Phi & \cos\Phi \end{pmatrix} \begin{pmatrix} \cos\Theta & 0 & -\sin\Theta \\ 0 & 1 & 0 \\ \sin\Theta & 0 & \cos\Theta \end{pmatrix} \begin{pmatrix} \cos\Psi & \sin\Psi & 0 \\ -\sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} \cos\Theta\cos\Psi & \cos\Theta\sin\Psi & -\sin\Theta \\ \sin\Phi\sin\Theta\cos\Psi - \cos\Phi\sin\Psi & \sin\Phi\sin\Theta\sin\Psi + \cos\Phi\cos\Psi & \sin\Phi\cos\Theta \\ \cos\Phi\sin\Theta\cos\Psi + \sin\Phi\sin\Psi & \cos\Phi\sin\Theta\sin\Psi - \sin\Phi\cos\Psi & \cos\Phi\cos\Theta \end{pmatrix}
\end{aligned}
$$

- Rotationsmatrix nach Yaw-Pitch-Roll

$$
\phi = \mathrm{Atan2}\left(-r_{31}, \sqrt{r_{11}^2 + r_{21}^2}\right)
$$

$$
\psi = -\mathrm{Atan2}\left(\frac{r_{21}}{\cos(\phi)}, \frac{r_{11}}{\cos(\phi)}\right)
$$

$$
\theta = \mathrm{Atan2}\left(\frac{r_{32}}{\cos(\phi)}, \frac{r_{33}}{\cos(\phi)}\right)
$$

Euler-Winkel

- Vorteil:
  - Minimale Repräsentation (3 Parameter)
  - "Einfach" interpretierbar
- Nachteile:
  - Es gibt viele "alternative" Euler-Winkel-Darstellungen (XYZ, ZXZ, ZYX, …)
  - Schwierig zu Verketten
  - Singularitäten (sog. "gimbal lock")

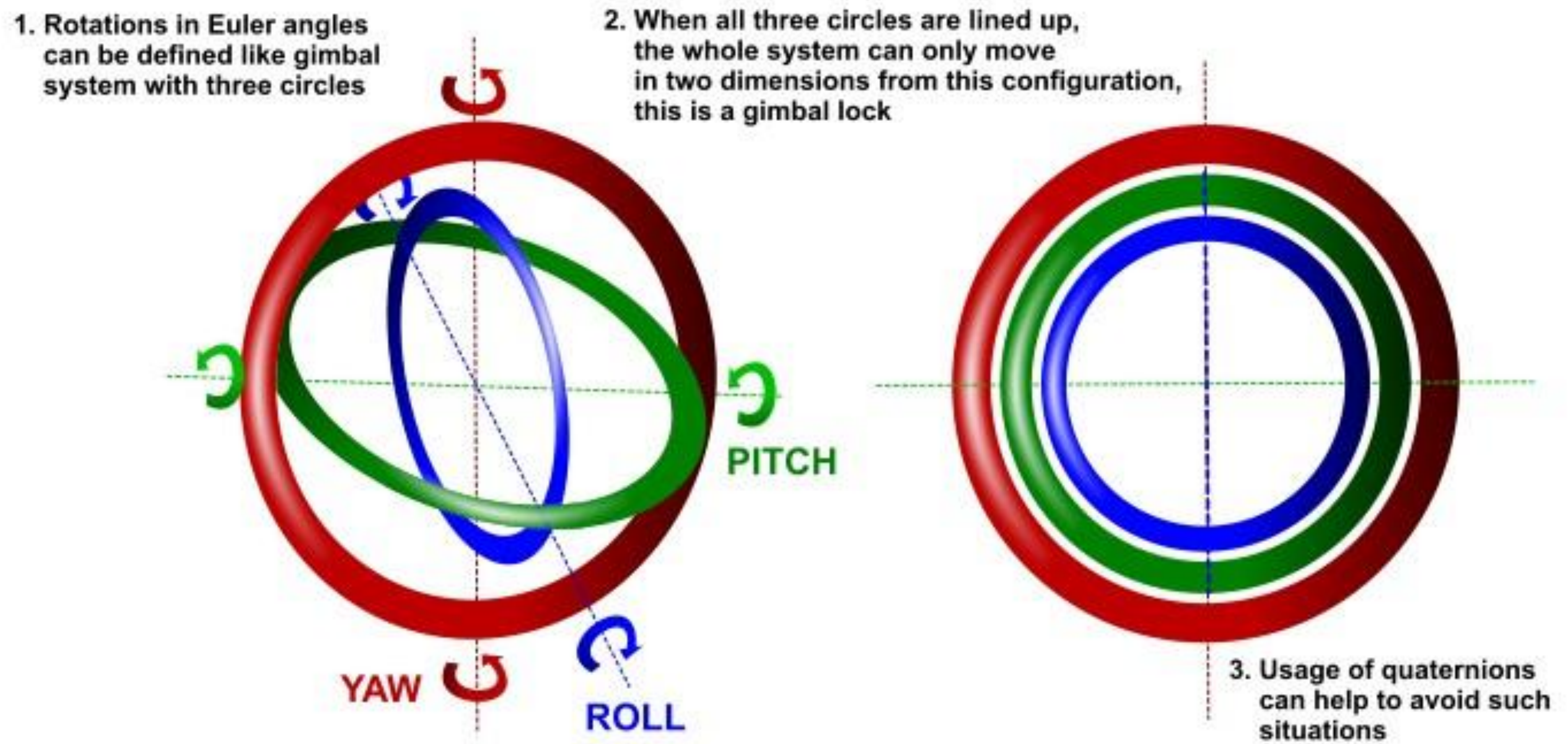
Euler-Winkel

- Verkettung: Konversion in Rotationsmatrix, Multiplikation, Re-Konversion
- Invertierung: Konversion in Rotationsmatrix, Matrixinversion, Re-Konversion

$$R_Z(\psi_1)R_Y(\theta_1)R_X(\phi_1) \cdot R_Z(\psi_2)R_Y(\theta_2)R_X(\phi_2)$$
$$\neq R_Z(\psi_1 + \psi_2)R_Y(\theta_1 + \theta_2)R_X(\phi_1 + \phi_2)$$

Singulatitäten

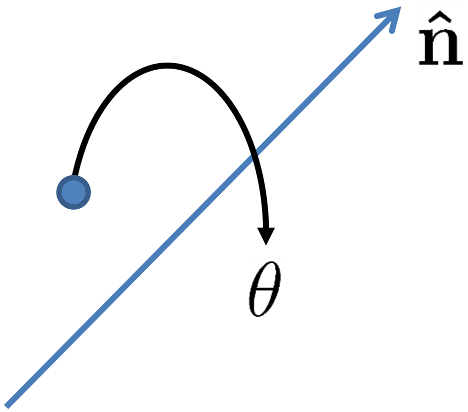
- Verlust eines Freiheitsgrades



Rodriguez-Darstellung

(Drehachse / Winkel)

- Representiert Rotation durch
  - Drehachse $\hat{\mathbf{n}}$ und
  - Drehwinkel θ
- 4 Parameter $(\hat{\mathbf{n}}, \theta)$
- 3 Parameter $\boldsymbol{\omega} = \theta\hat{\mathbf{n}}$
  - Länge kodiert Winkel
  - Minimal aber nicht eindeutig (Warum?)



Konversion

- Rodriguez-Formel

$$R(\hat{\mathbf{n}}, \theta) = I + \sin\theta[\hat{\mathbf{n}}]_\times + (1 - \cos\theta)[\hat{\mathbf{n}}]_\times^2$$

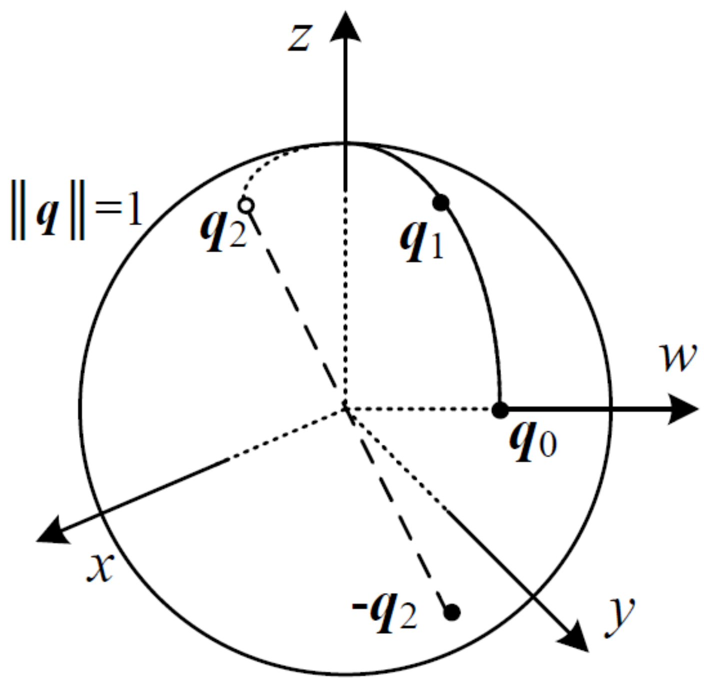
- Inverse

$$\theta = \cos^{-1}\left(\frac{\mathrm{trace}(R) - 1}{2}\right), \hat{\mathbf{n}} = \frac{1}{2\sin\theta}\begin{pmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{pmatrix}$$

- see: An Invitation to 3D Vision, Y. Ma, S. Soatto, J. Kosecka, S. Sastry, Chapter 2 (available online)

Einheitsquaternionen

- Quaternion $\mathbf{q} = (q_x, q_y, q_z, q_w)^\top \in \mathbb{R}^4$
- Einheitsquaternionen $\|\mathbf{q}\| = 1$
- Entgegengesetzte Quaternionen repräsentieren dieselbe Rotation $\mathbf{q} = -\mathbf{q}$
- Ansonsten eindeutig



Einheitsquaternionen

- Vorteil: Operationen für Multiplikation und Inversion sind effizient
- Quaternion-Quaternion Multiplikation

$$\begin{aligned}\mathbf{q}_0\mathbf{q}_1 &= (\mathbf{v}_0, w_0)(\mathbf{v}_1, w_1) \\ &= (\mathbf{v}_0 \times \mathbf{v}_1 + w_0\mathbf{v}_1 + w_1\mathbf{v}_0, w_0w_1 - \mathbf{v}_0\mathbf{v}_1)\end{aligned}$$

- Inverse (Vorzeichenwechsel bei v oder w)

$$\begin{aligned}\mathbf{q}^{-1} &= (\mathbf{v}, w)^{-1} \\ &= (\mathbf{v}, -w)\end{aligned}$$

Einheitsquaternionen

- Quaternion-Vektor Multiplikation (rotiert Punkt p mit Rotation q)

$$\mathbf{p}' = \mathbf{v}\bar{\mathbf{p}}\mathbf{q}^{-1}$$

mit $\quad \bar{\mathbf{p}} = (x, y, z, 0)^\top$

- Beziehung zu Rodriguez-Darstellung

$$\mathbf{q} = (\mathbf{v}, w) = (\sin\frac{\theta}{2}\hat{\mathbf{n}}, \cos\frac{\theta}{2})$$

3D Orientierung im Allgemeinen

- **Beachte:** “Lesen von Rotationen” im Allgemeinen schwierig, egal in welcher Darstellung
- **Beobachtung:** Rotationen sind einfach zu visualisieren und dann intuitiv verständlich
- **Rat:** Zum debuggen immer visualisieren!! Es gibt viele gute 3D Visualisierungstools

C++ Libraries für Lin. Alg./Geometry

- Es gibt viele C/C++ libraries für lineare Algebra und 3D- Geometry

- Beispiele:
  - C arrays, std::vector (no linear alg. functions)
  - gsl (gnu scientific library, umfangreich, plain C)
  - boost::array (ROS messages)
  - Bullet library (3D-Geometry und Dynamik, ROS tf)
  - Eigen (Sowohl lineare Algebra als auch Geometry, guter Tipp)