

Robotik

Priv.-Doz. Dr. Thomas Wiemann
Institut für Informatik
Autonome Robotik

SoSe 2021



Inhalt



Gliederung

1. Einleitung
2. Robot Operating System
3. Sensorik
4. Sensordatenverarbeitung
5. Fortbewegung
6. Lokalisierung
7. Mapping
8. Navigation
9. Ausblick

Inhalt



Gliederung

1. Einleitung
2. Robot Operating System
3. Sensorik
4. Sensordatenverarbeitung
5. Fortbewegung
6. Lokalisierung
7. Mapping
8. Navigation
9. Ausblick

Inhalt

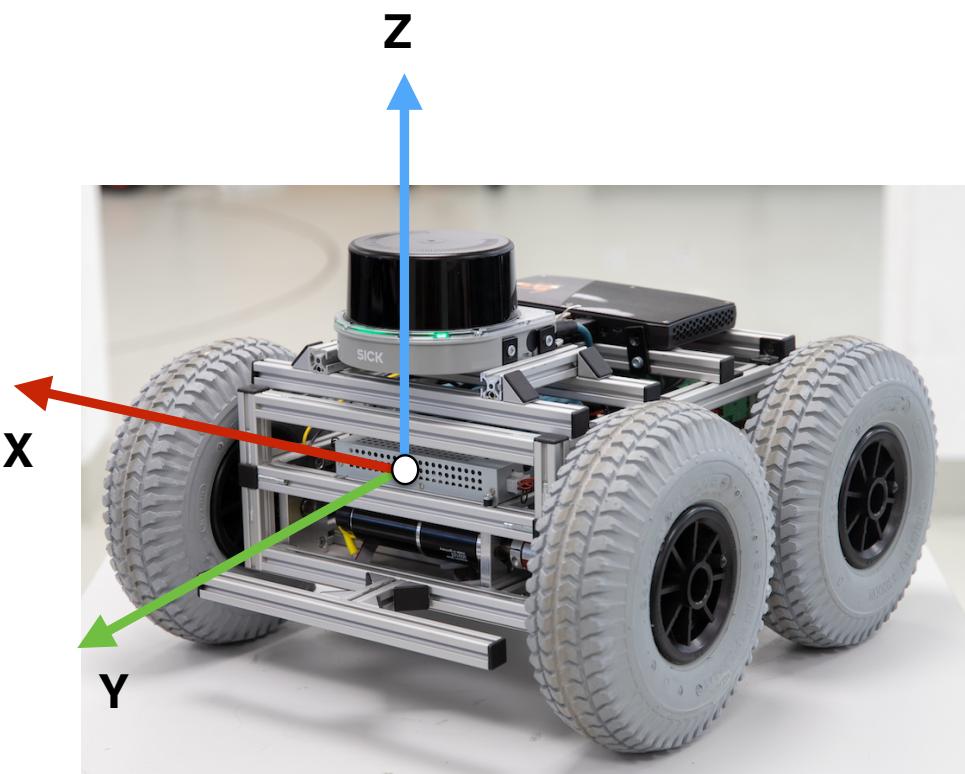
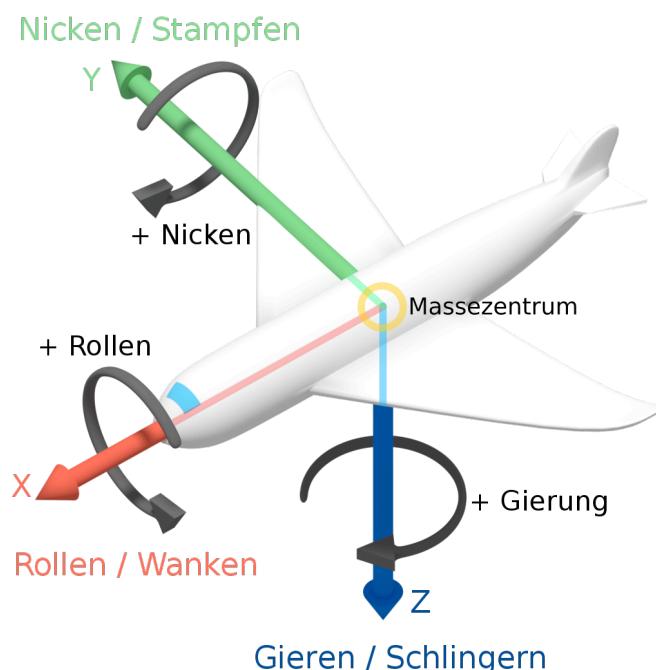


Gliederung

- 2. Robot Operating System
 - 1. Konventionen / Mathe
 - 2. Grundlagen
 - 3. Nachrichten
 - 4. Transformationen
 - 5. Robotermodellierung
 - 6. Mapping
 - 7. Navigation
 - 8. Ausblick

Allgemeines

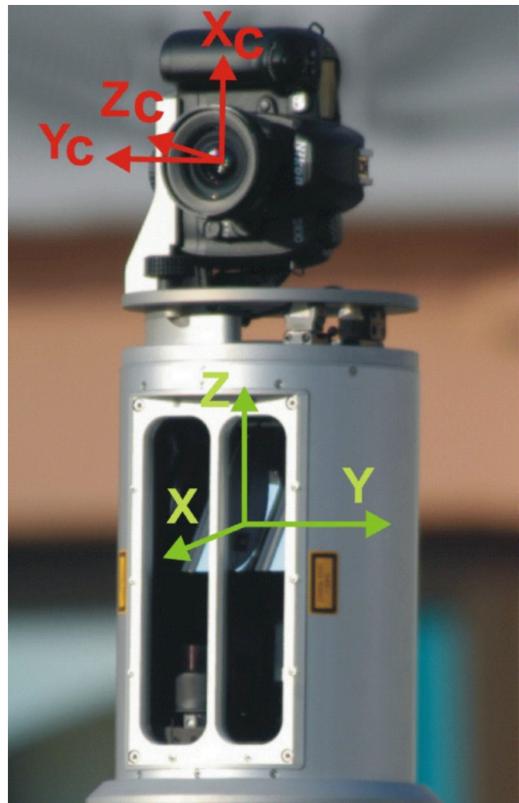
► Koordinatensysteme:



ROS' Koordinatensystem ist rechtshändig

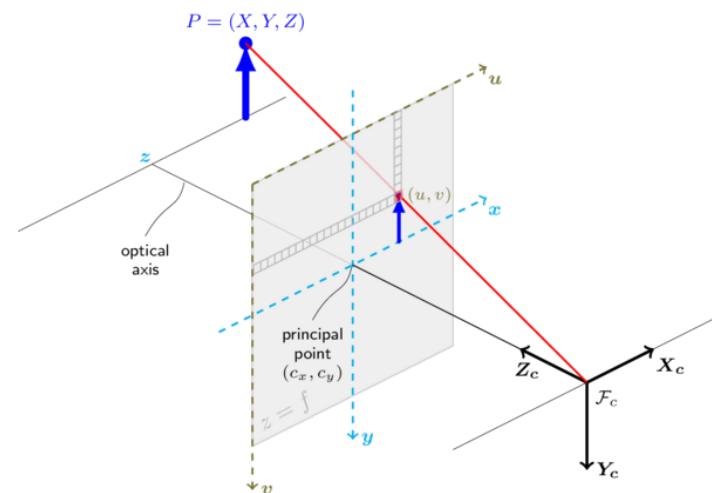
Es gibt auch andere Konventionen. Im Buch von Hertzberg / Nüchter / Lingemann wird durchgängig ein linkshändiges Koordinatensystem verwendet. Entsprechend sind alle Formeln in linkshändischen Koordinaten. Umrechnung erforderlich!

Koordinatensysteme - Beispiele



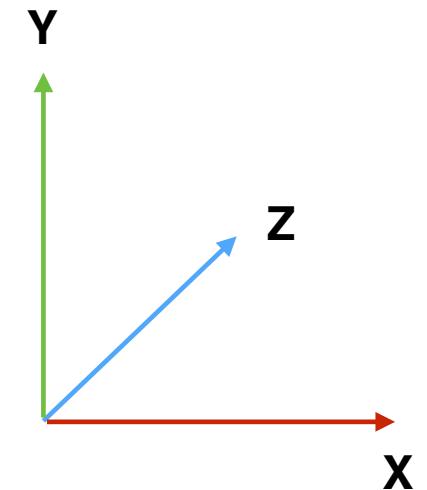
Riegl LMS

(right-handed)



OpenCV

(right-handed)

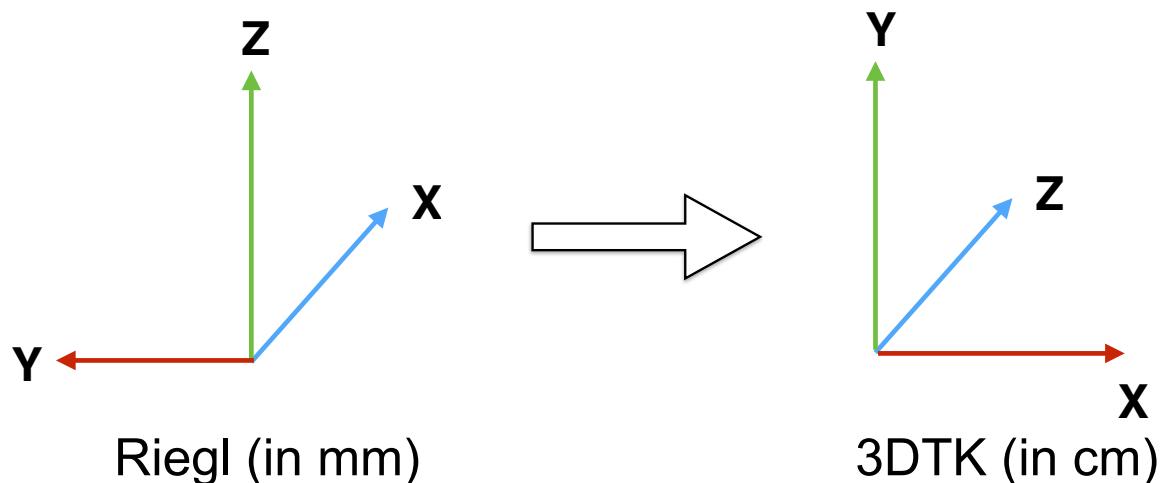


3DTK / Buch

(left-handed)

Koordinatensysteme - Umrechnung

► Beispiel Riegl \Rightarrow 3DTK



$$\begin{pmatrix} x_{3DTK} \\ y_{3DTK} \\ z_{3DTK} \end{pmatrix} = \begin{pmatrix} -10 \cdot y_{Riegl} \\ 10 \cdot z_{Riegl} \\ 10 \cdot x_{Riegl} \end{pmatrix}$$

Rotationen (1)

- Rotationen lassen sich mit Hilfe von Matrix / Vektormultiplikation realisieren
- Für 2D-Rotationen gilt:

$$R_\alpha = \begin{pmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{pmatrix}$$

- Rotationsmatrizen in 3D:

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

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

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

Homogene Koordinaten

- ▶ Rotation, Skalierung und Scherung eines 3D-Punktes lassen sich durch 3x3-Matrizen ausdrücken
- ▶ Problem: Translation erfordert eine Vektor-Addition
- ▶ Lösung: Erweiterung des Raumes auf 4-Dimensionen
- ▶ Nun lassen sich auch Translationen durch Matrixoperationen beschreiben:

$$(x, y, z)^T \mapsto (x, y, z, w)^T$$

- ▶ Rücktransformation von homogenen Koordinaten

$$(x', y', z', w')^T \mapsto \left(\frac{x'}{w'}, \frac{y'}{w'}, \frac{z'}{w'} \right)^T$$

- ▶ Beispiel Translation

$$Tx = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ z + t_z \\ 1 \end{pmatrix} = \begin{pmatrix} x + t_x \\ y + t_y \\ z + t_z \\ 1 \end{pmatrix}$$

Transformationematrizen

- Wichtige Transformationen:

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Translation

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation um x-Achse

$$\begin{pmatrix} \cos\beta & 0 & \sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation um y-Achse

$$\begin{pmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Rotation um z-Achse

- Achtung: Darstellungen hängen vom Koordinatensystem ab!!

- Weitere Transformationen:

$$\begin{pmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Skalierung

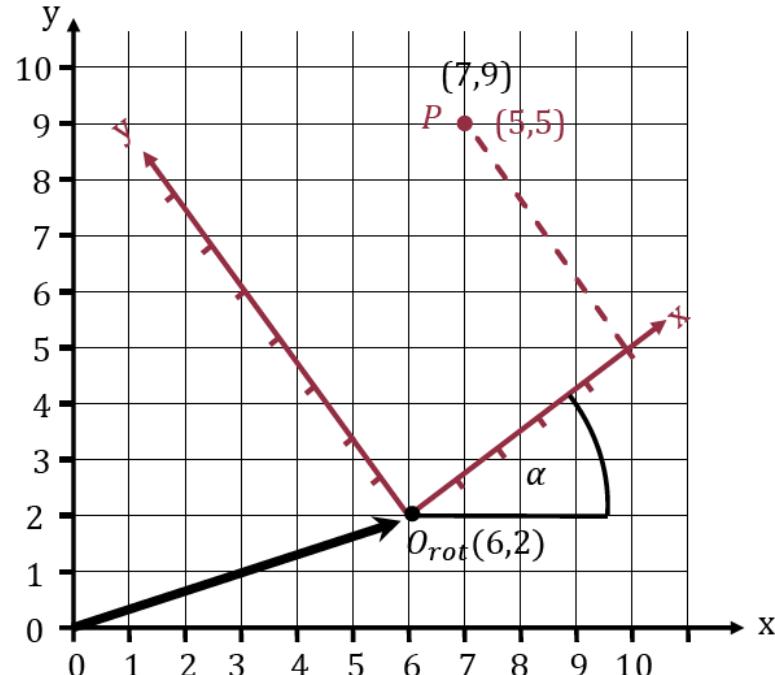
$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \end{pmatrix}$$

Perspektive

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Orthogonale Projektion

Basiswechsel (1)



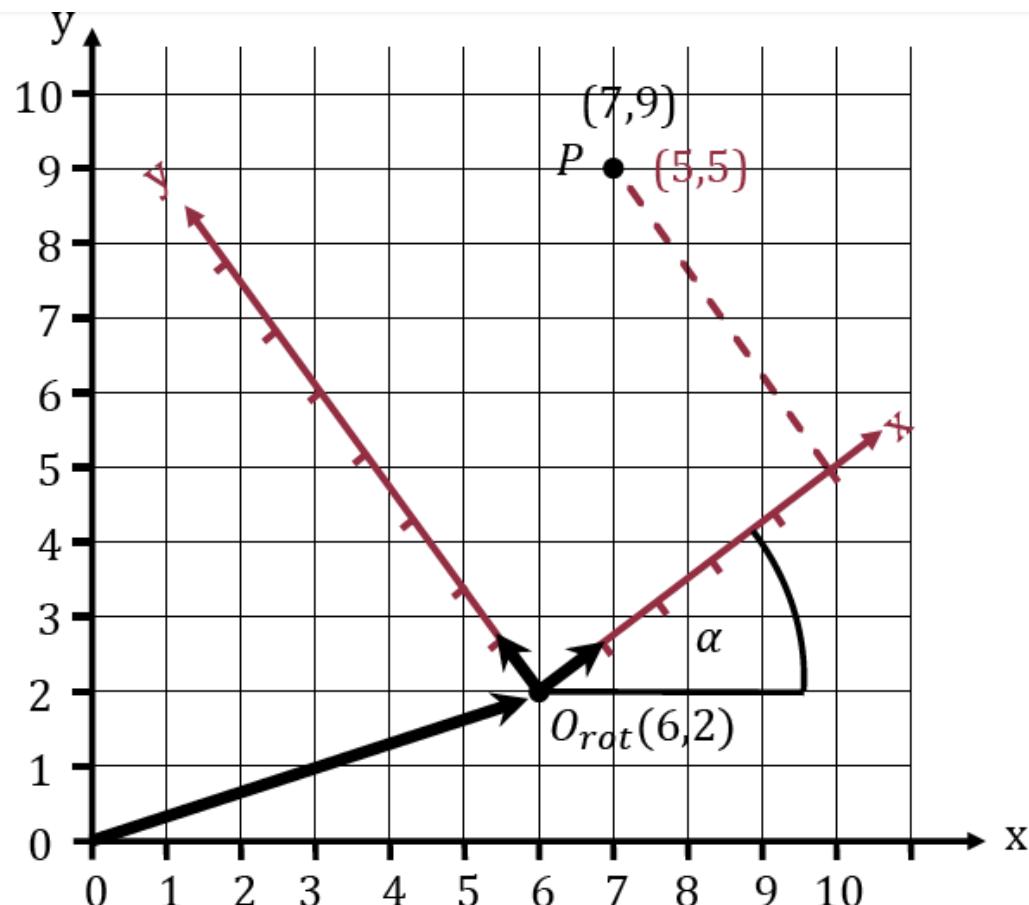
- ▶ Beschreibe P im schwarzen Koordinatensystem
- ▶ Suche Matrix, die den P_{rot} nach P_{schw} transformiert
- ▶ Drehe schwarzes System um α :
 - $\cos(\alpha) = 4 / 5 = 0.8$
 - $\sin(\alpha) = 3 / 5 = 0.6$
- ▶ Rotations- und Translationsmatrix:

$$R(\alpha) = \begin{pmatrix} 0.8 & -0.6 & 0 \\ 0.6 & 0.8 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad T_{rot} = \begin{pmatrix} 1 & 0 & 6 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

- ▶ Komplette Transformation:

$$P_{schw} = T_{rot} \cdot R(\alpha) \cdot T_{rot} = \begin{pmatrix} 0.8 & -0.6 & 6 \\ 0.6 & 0.8 & 2 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 5 \\ 5 \\ 1 \end{pmatrix} = \begin{pmatrix} 7 \\ 9 \\ 1 \end{pmatrix}$$

Basiswechsel (2)

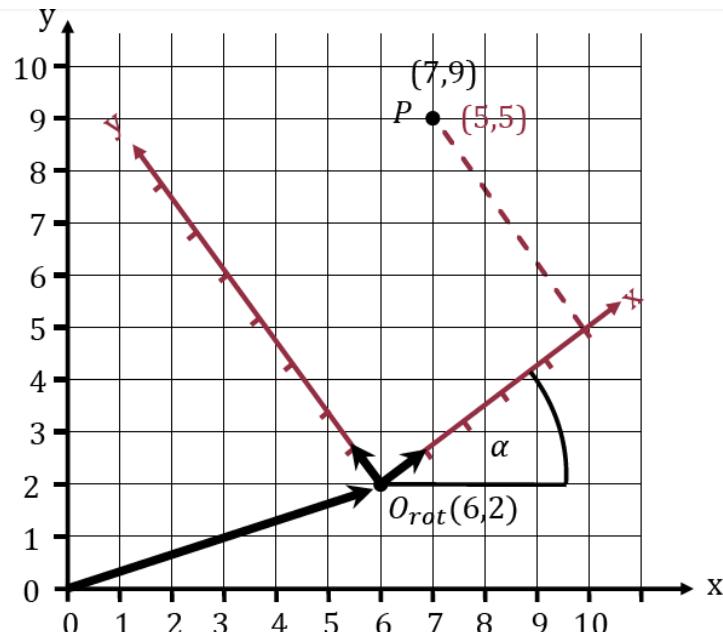


$$M_{rot \rightarrow schw} = \begin{pmatrix} 0.8 & -0.6 & 6 \\ 0.6 & 0.8 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

Annotations pointing to the matrix:

- e_y von rot, beschrieben in schwarz
- O von rot, beschrieben in schwarz
- e_x von rot, beschrieben in schwarz

Basiswechsel (3)



$$\begin{aligned}
 M_{schw \rightarrow rot} &= M_{rot \rightarrow schw}^{-1} \\
 M_{schw \rightarrow rot} &= R(\alpha)^{-1} \cdot T(O_{rot})^{-1} \\
 M_{schw \rightarrow rot} &= R(-\alpha) \cdot T(-O_{rot}) \\
 M_{schw \rightarrow rot} &= \begin{pmatrix} 0.8 & 0.6 & -6 \\ -0.6 & 0.8 & -2 \\ 0 & 0 & 1 \end{pmatrix}
 \end{aligned}$$

► Test:

$$P_{rot} = M_{schw \rightarrow rot} \cdot P_{schw} = \begin{pmatrix} 0.8 & 0.6 & -6 \\ -0.6 & 0.8 & -2 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 7 \\ 9 \\ 1 \end{pmatrix} = \begin{pmatrix} 5 \\ 5 \\ 1 \end{pmatrix}$$

Basiswechsel (4)

$$\mathbf{M}_{B \rightarrow A} \cdot \mathbf{p}_B = \mathbf{p}_A = \underbrace{\begin{pmatrix} e_{(x,x)}^A & e_{(y,x)}^A & e_{(z,x)}^A & O_x^A \\ e_{(x,y)}^A & e_{(y,y)}^A & e_{(z,y)}^A & O_y^A \\ e_{(x,z)}^A & e_{(y,z)}^A & e_{(z,z)}^A & O_z^A \\ 0 & 0 & 0 & 1 \end{pmatrix}}_{\text{Beschrieben in A}} \cdot \begin{pmatrix} p_x^B \\ p_y^B \\ p_z^B \\ 1 \end{pmatrix}$$

Punkt beschrieben in A Ursprung von B Punkt beschrieben in B

$$(\mathbf{M}_{B \rightarrow A})^{-1} \cdot \mathbf{p}_A = \mathbf{M}_{A \rightarrow B} \cdot \mathbf{p}_A = \mathbf{p}_B$$

Inhalt



Gliederung

- 2. Robot Operating System
 - 1. Konventionen / Mathe
 - 2. Grundlagen**
 - 3. Nachrichten
 - 4. Transformationen
 - 5. Robotermodellierung
 - 6. Mapping
- 7. Navigation
- 8. Ausblick

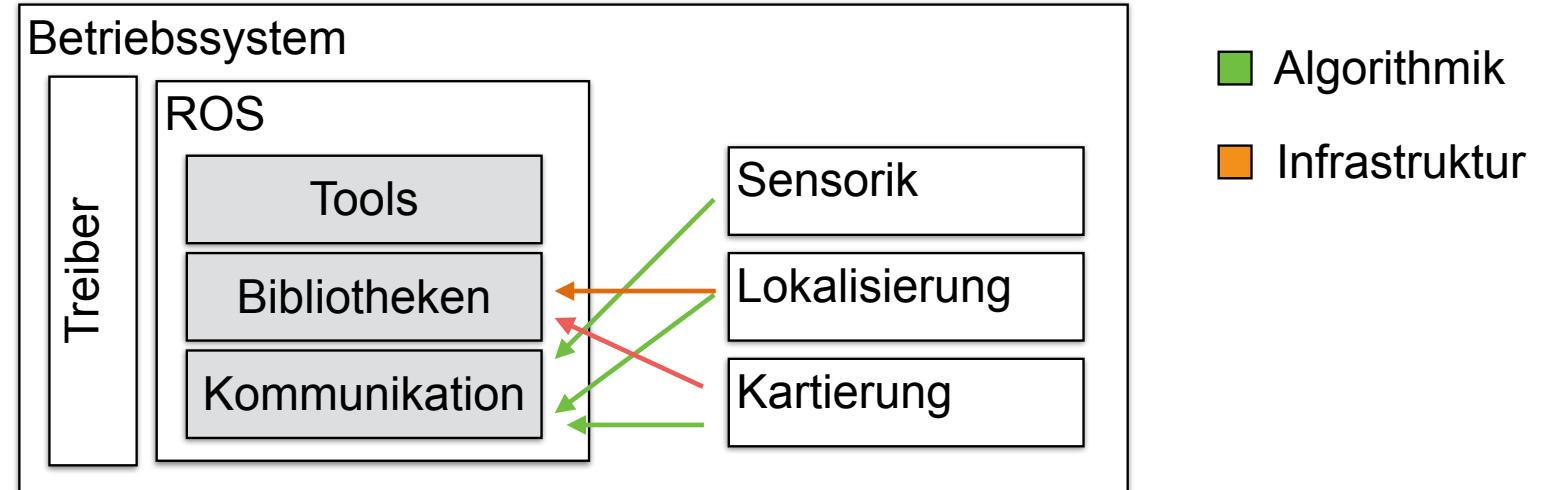
Prä-ROS Robotik

- ▶ Keine Standards / Konventionen für Koordinatensysteme, Repräsentation von Sensordaten und Austauschformaten
- ▶ Wenig wiederverwertbarer Code
- ▶ Viele Grundlegende Funktionalitäten mussten selbst entwickelt werden:
 - Treiber für Komponenten
 - Interfaces für den Datenaustausch zwischen Komponenten
 - Betriebssystemspezifische Anpassungen
 - Interprozesskommunikation
 - Resourcenhandling
 - ...
- ▶ Wenn diese implementiert wurden, mussten die Standardalgorithmen für das jeweilige System neu programmiert werden
- ▶ Wenn ein neuer Roboter ins Labor kam ➔ Goto #1

Darstellung der Einführung basierend auf der ROS Lecture der CMU (G. di Caro)

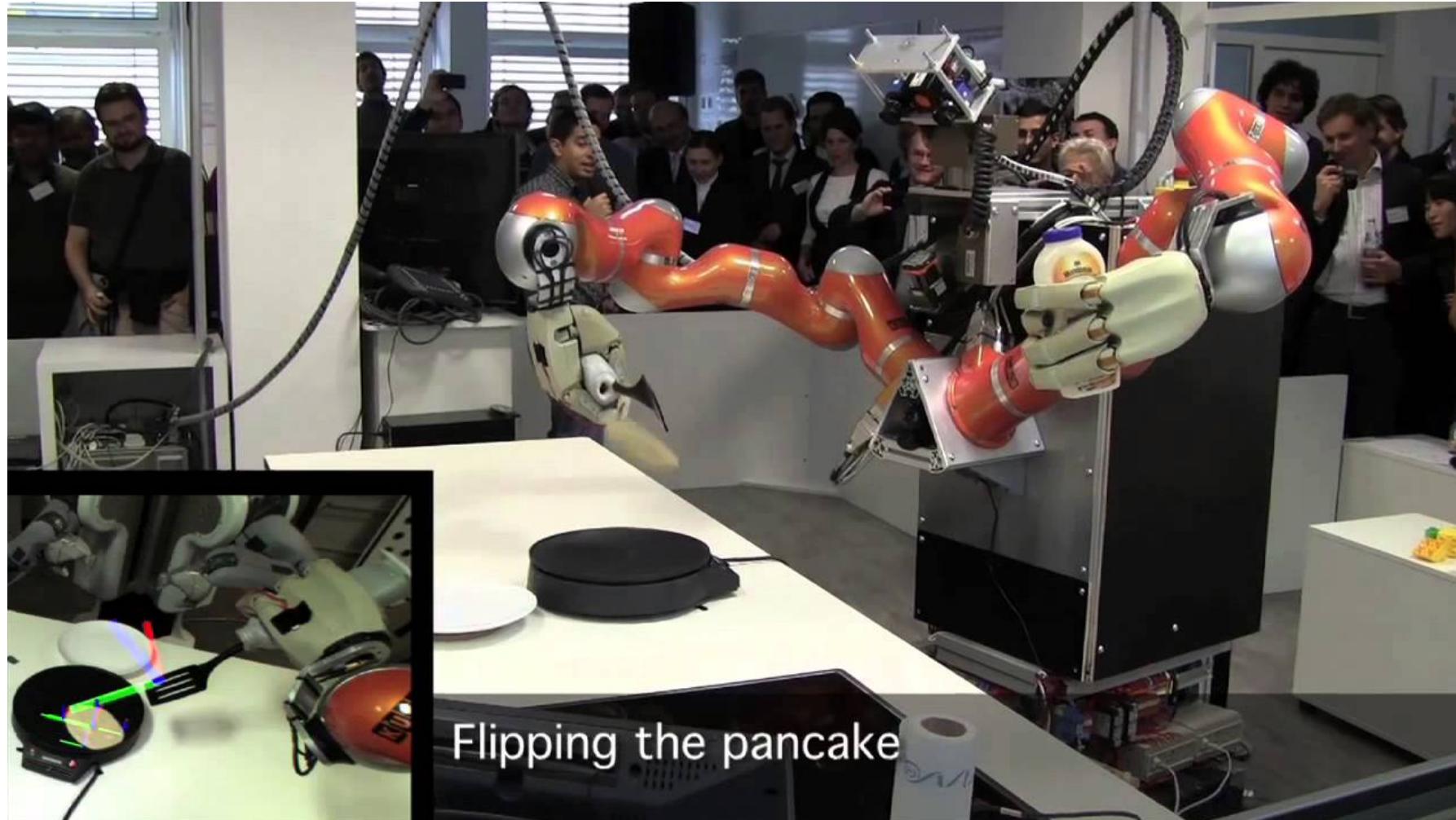
Was ist ROS

- ▶ Open-Source Robot Operating System
- ▶ Bibliotheken und Hilfsprogramme, die es erleichtern Software für verschiedene Plattformen zu entwickeln
- ▶ Ursprünglich 2007 beim Stanford AI-Lab entwickelt
- ▶ Ab 2009 Hauptentwicklung bei Willow Garage
- ▶ Seit 2013 in Händen der Open Source Robotics Foundation (OSRF)



■ Algorithmik
■ Infrastruktur

ROS - Roboter



Flipping the pancake

<https://youtu.be/PGaXiLZD2KQ>

Die zwei Gesichter von ROS

- ▶ Das “Betriebssystem”...
 - Stellt Komponenten zur bereit, die bei direkter Implementierung eigentlich vom Betriebssystem übernommen würden
 - Beispiele:
 - Hardware-Abstraktion
 - Low-level Treiber
 - Nachrichtenversand zwischen Prozessen
 - Paketmanagement
 - Gemeinsam benötigte Funktionalität
- ▶ Die Software-Seite...
 - stellt Bibliotheken zur Verfügung
 - stellt Pakete mit Implementierungen von Nutzern zur Verfügung
 - z.B. Vision, SLAM, Planung usw.
- ▶ Im Zusammenspiel ermöglicht dies dank der gemeinsamen Schnittstellen Software platformübergreifend wiederzuverwenden

ROS - Grundsätze

▶ Peer-to-peer

- Ein ROS-System besteht aus vielen kleinen Programmen “Nodes”
- Die Knoten verbinden sich miteinander und tauschen kontinuierlich Nachrichten aus
- Kommunikation via TCP/IP
- Die Knoten müssen nicht zwangsläufig auf demselben Rechner laufen

▶ Werkzeugbasiert

- ROS stellt viele nützliche Tools zur Verfügung, die helfen, den Zustand des Systems zu inspizieren
- Visualisierung, Logging, Anzeigen von Datenströmen

▶ Bindings für verschiedene Sprachen

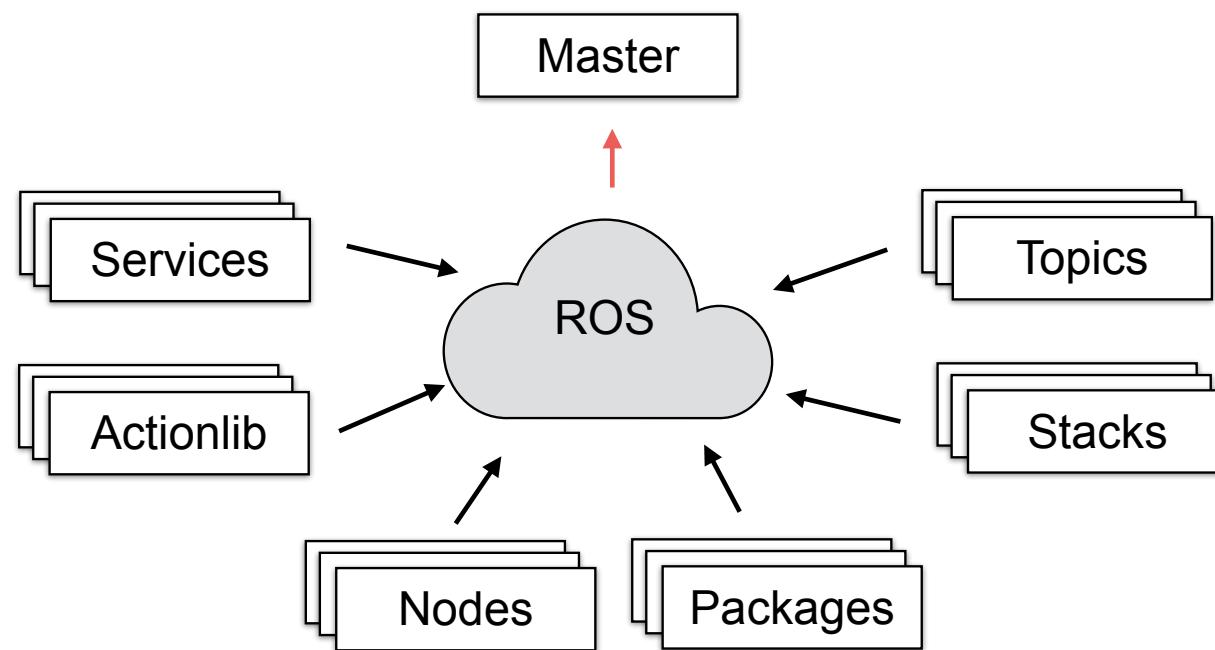
- ROS-Komponenten können in jeder Sprache entwickelt werden, für die es entsprechende Client-Libraries gibt
- Hauptsächlich C++ und Python aber auch Java, MATLAB, LISP und andere

▶ Kapselung

- Softwarepakete sollen eigene Bibliotheken sein, die dann in ROS gekapselt werden

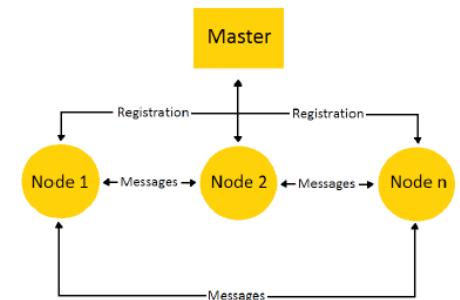
▶ Frei und Open-Source, Community-Repos

ROS - Kernkonzepte



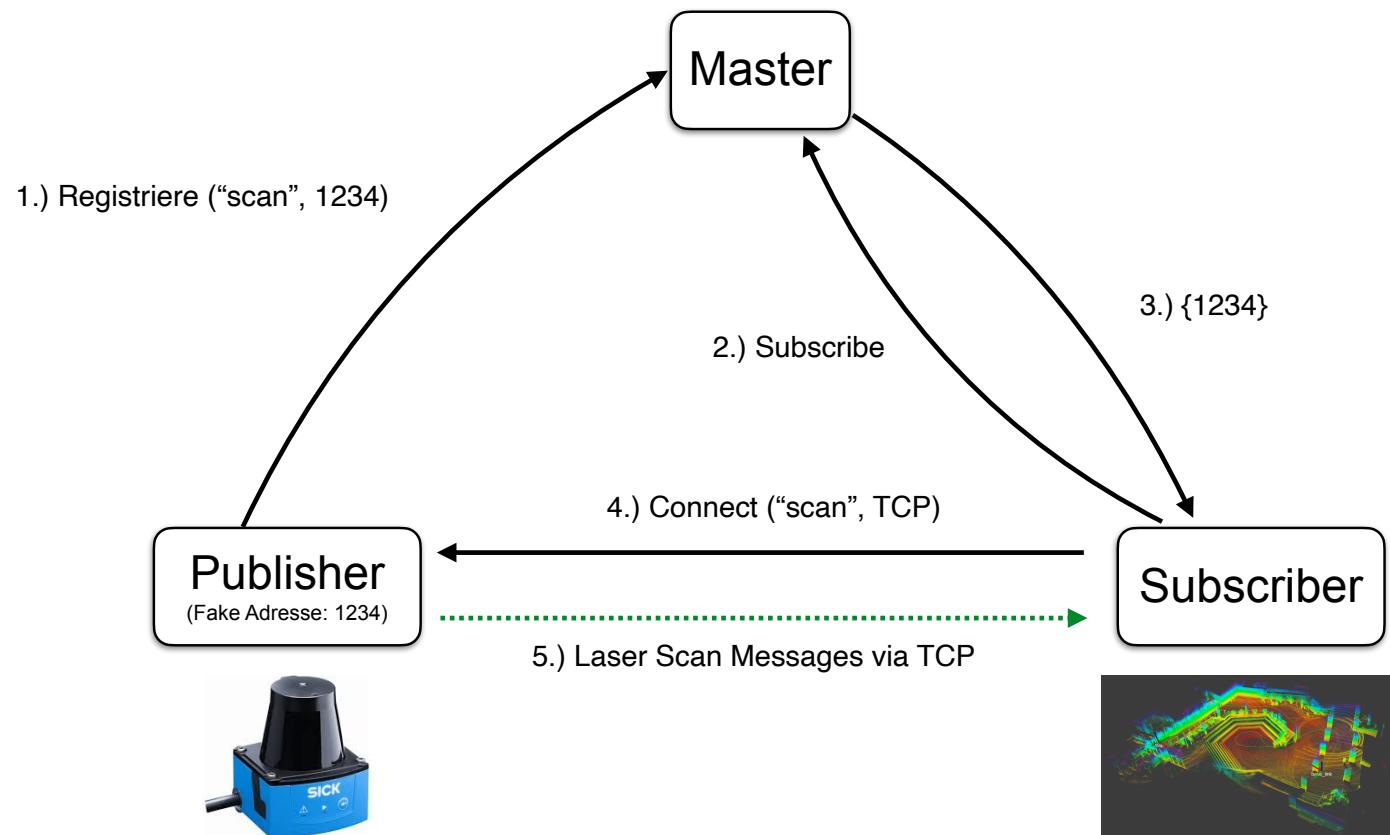
ROS Master (1)

- ▶ Stellt die Informationen zur Verfügung, die die Knoten benötigen um miteinander kommunizieren zu können
- ▶ Wenn ein Knoten startet, verbindet er sich zu einem Master und teilt mit:
 - ▶ ... welche Daten (Messages / Topics) er zur Verfügung stellt
 - ▶ ... welche Services bereitgestellt werden
 - ▶ ... welche Actions bereitgestellt werden
 - ▶ ... zu welchen Topics er sich subscribed
- ▶ Jeder Knoten bekommt vom Master die Informationen, die er benötigt um eine peer-to-peer Verbindung zu allen anderen Knoten herzustellen, die Informationen anfordern oder zur Verfügung stellen



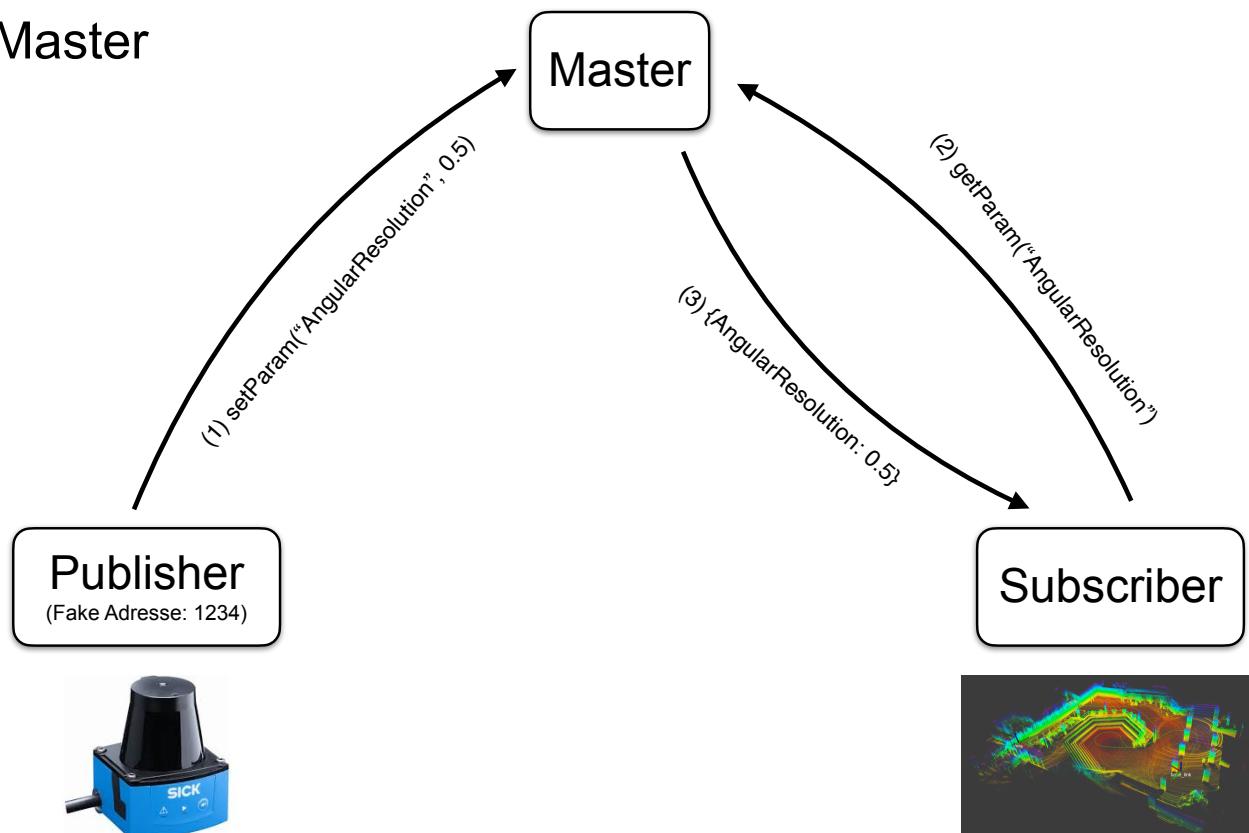
ROS Master (2)

► Ablauf (stark vereinfacht)



Parameter Server

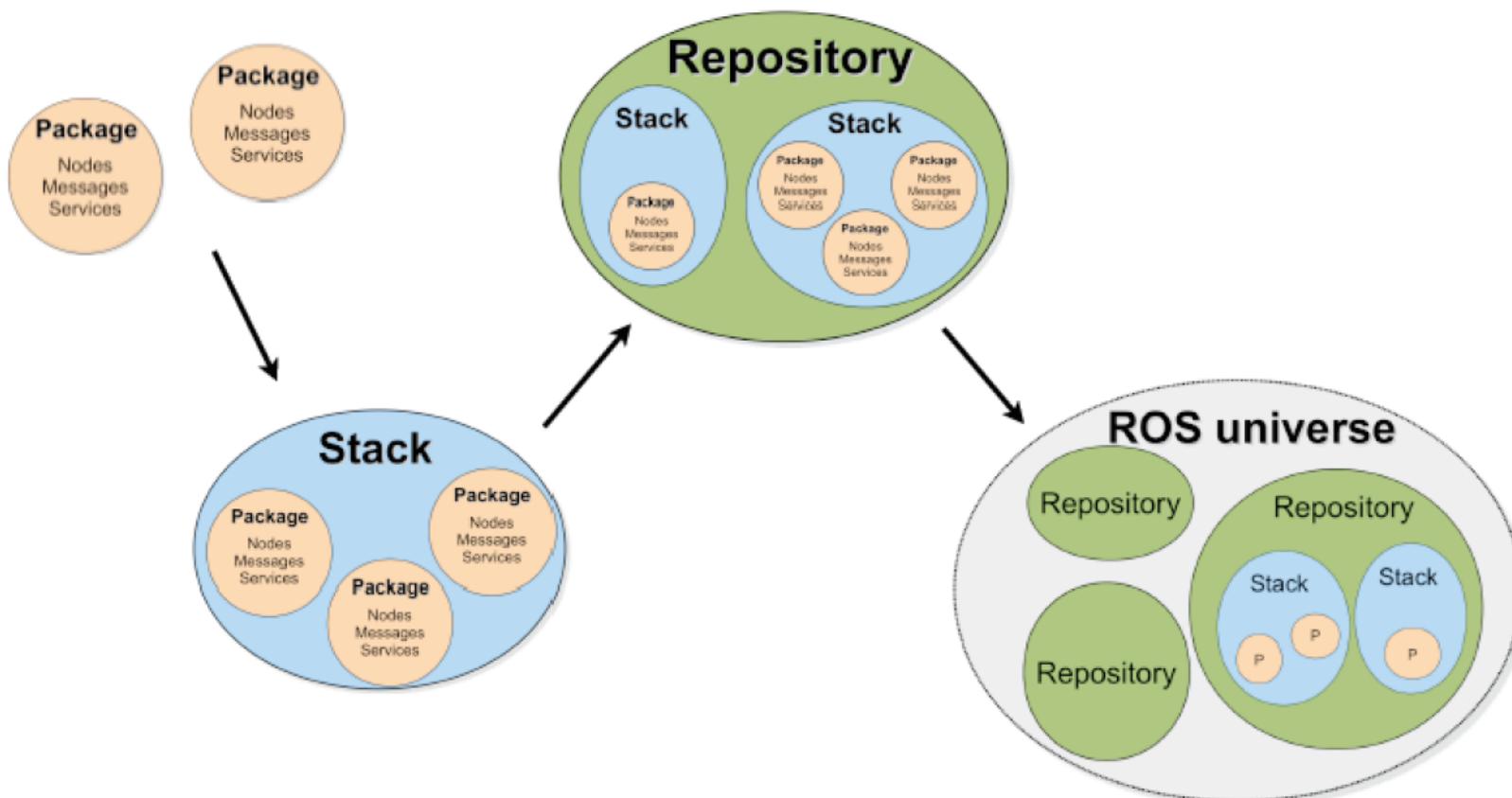
- ▶ Dictionary, das über die ROS API abgefragt werden kann
- ▶ Am besten geeignet für statische, nicht-binäre Daten wie Konfigurationsparameter
- ▶ Läuft im ROS Master



ROS Bags

- ▶ ROS Bags erlauben es einfach Daten zu loggen
- ▶ Die serialisierten Nachrichten eines oder mehrer Topics werden in einer Datei gespeichert
- ▶ Bagfiles lassen sich nachträglich abspielen
- ▶ Erlauben
 - ▶ Nachträgliche Analyse des Systemverhaltens
 - ▶ Referenzdatensätze zu erstellen
 - ▶ Daten ohne Zugang zu echten Robotern in Übungen zu einem Online-Robotikkurs zur Verfügung zu stellen ;-)

Das ROS - Ökosystem



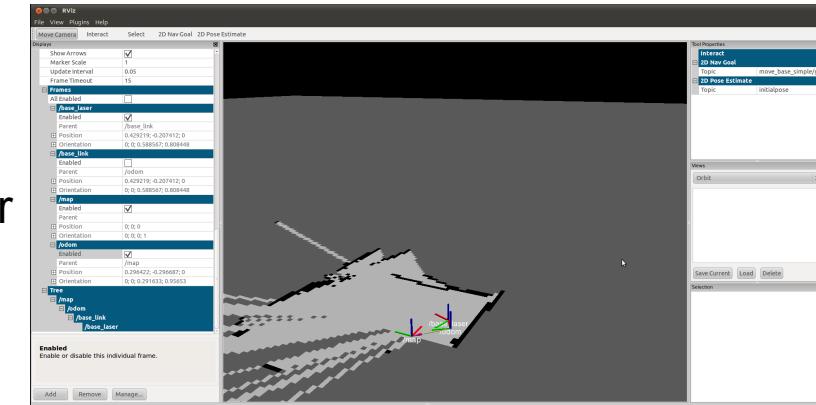
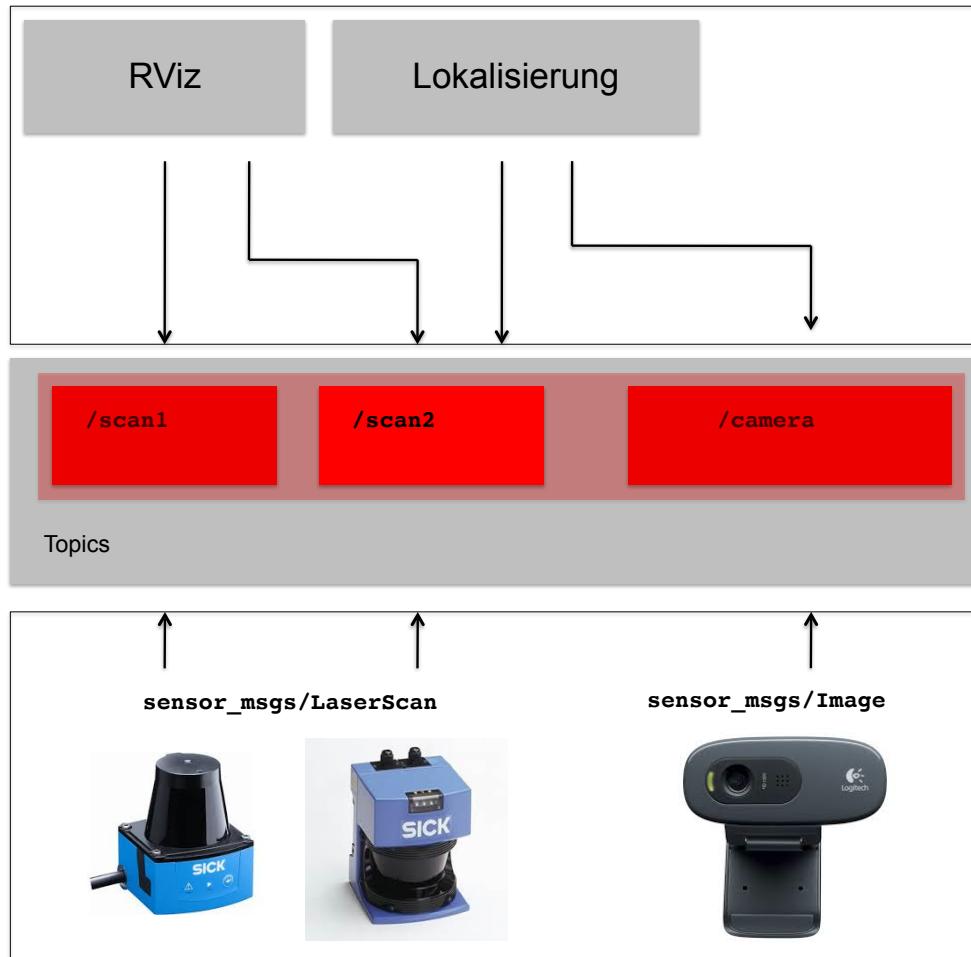
Inhalt



Gliederung

- 2. Robot Operating System
 - 1. Konventionen / Mathe
 - 2. Grundlagen
 - 3. Nachrichten**
 - 4. Transformationen
 - 5. Robotermodellierung
 - 6. Mapping
 - 7. Navigation
 - 8. Ausblick

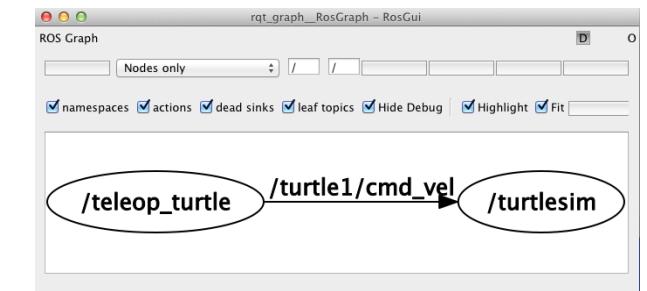
ROS Messaging (1)



Subscriber

Roscore

Publisher



ROS Messaging (2)

```

# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

Header header          # timestamp in the header is the acquisition time of
                        # the first ray in the scan.
#
# in frame frame_id, angles are measured around
# the positive z axis (counterclockwise, if z is up)
# with zero angle being forward along the x axis

float32 angle_min      # start angle of the scan [rad]
float32 angle_max      # end angle of the scan [rad]
float32 angle_increment # angular distance between measurements [rad]

float32 time_increment  # time between measurements [seconds] - if your scanner
                        # is moving, this will be used in interpolating position
                        # of 3d points
float32 scan_time       # time between scans [seconds]

float32 range_min       # minimum range value [m]
float32 range_max       # maximum range value [m]

float32[] ranges        # range data [m] (Note: values < range_min or > range_max should be discarded)
float32[] intensities   # intensity data [device-specific units]. If your
                        # device does not provide intensities, please leave
                        # the array empty.

```

ROS Messaging (3)

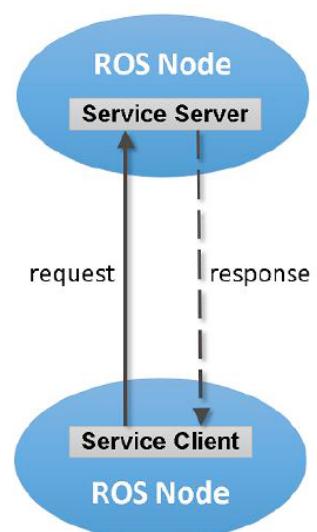
- ▶ Publisher geben kein Feedback, nachrichten können “verloren gehen”
- ▶ Andere ROS-Konstrukte zur Kommunikation zwischen Nodes:

Type	Strengths	Weaknesses
Message	<ul style="list-style-type: none"> •Good for most sensors (streaming data) 	<ul style="list-style-type: none"> •Messages can be <u>dropped</u> without knowledge •Easy to overload system with too many messages
Service	<ul style="list-style-type: none"> •Knowledge of missed call •Well-defined feedback 	<ul style="list-style-type: none"> •Blocks until completion •Connection typically re-established for each service call (slows activity)
Action	<ul style="list-style-type: none"> •Monitor long-running processes •Handshaking (knowledge of missed connection) 	<ul style="list-style-type: none"> •Complicated •Mixed feedback from multiple action servers (not for SimpleAction)

http://events.ccc.de/congress/2014/Fahrplan/system/attachments/2497/original/How_I_Learned_to_Stop_Reinventing_and_Love_the_Wheels-bihlmaier-expanded.pdf

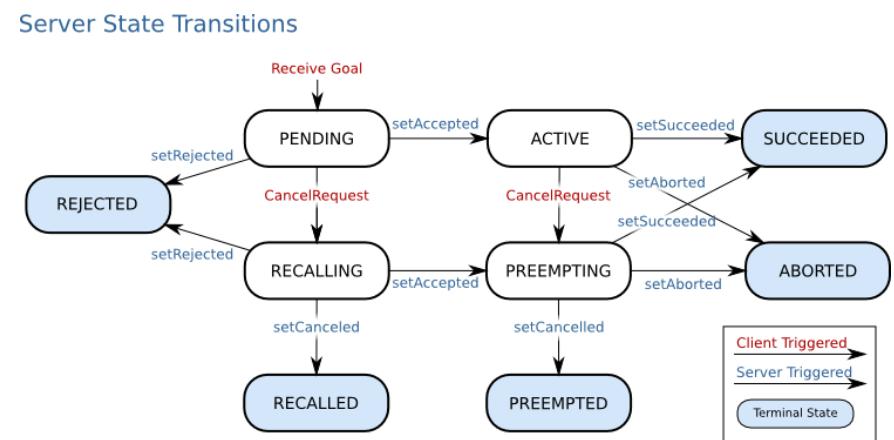
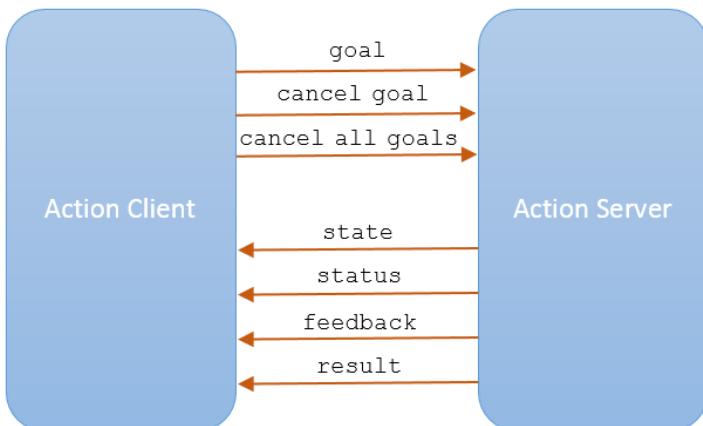
ROS - Services

- ▶ Synchronisierter Datenaustausch zwischen zwei Knoten
- ▶ *Blockierender* RPC: Frage nach Daten und Warte auf die Antwort
- ▶ 1-to-1 Request-response
- ▶ Beispiele:
 - ▶ Stoße eine externe Berechnung an und warte auf das Ergebnis
 - ▶ Triggere eine Funktionalität / ein Verhalten
 - ▶ Erfrage größere Daten, die nicht periodisch gesendet werden (Karte)
- ▶ Sollte nicht verwendet werden, um auf aufwendige Berechnungen zu warten
- ▶ Dafür gibt es die Actionlib



ROS - Actions

- ▶ Actions sind für langwierige Berechnung ausgelegt
- ▶ Der Actionserver arbeitet die Berechnung nebenläufig ab und gibt Feedback zu seinem aktuellen Status
- ▶ Kommunikation über das ROS Action Protocol



- Intermediate States
 - Pending - The goal has yet to be processed by the action server
 - Active - The goal is currently being processed by the action server
 - Recalling - The goal has not been processed and a cancel request has been received from the action client, but the action server has not confirmed the goal is canceled
 - Preempting - The goal is being processed, and a cancel request has been received from the action client, but the action server has not confirmed the goal is canceled
- Terminal States
 - Rejected - The goal was rejected by the action server without being processed and without a request from the action client to cancel
 - Succeeded - The goal was achieved successfully by the action server
 - Aborted - The goal was terminated by the action server without an external request from the action client to cancel
 - Recalled - The goal was canceled by either another goal, or a cancel request, before the action server began processing the goal
 - Preempted - Processing of the goal was canceled by either another goal, or a cancel request sent to the action server

Inhalt



Gliederung

- 2. Robot Operating System
 - 1. Konventionen / Mathe
 - 2. Grundlagen
 - 3. Nachrichten
 - 4. Transformationen**
 - 5. Robotermodellierung
 - 6. Mapping
 - 7. Navigation
 - 8. Ausblick