Einführung in das Robot Operating System (ROS)

Michael Korn

Raum: BC414, Tel.: 0203 - 379 - 3583,
E-Mail: michael.korn@uni-due.de

Nachteile von ERSP und Windows XP:

- Compilieren nur mit alten VS-Compiler
- Software an Hardware gebunden
- API oft nicht ausreichend und Quelltext nicht einsehbar
- Keine Weiterentwicklung (weder Entwickler, noch Community)
- Der Support für Windows XP endet

- Überblick über das Robot Operating System und darin aufgehende grundlegende Konzepte
- Erfahrungsaustausch (was funktioniert und was fehlt?)
- Einsatzmöglichkeiten am Lehrstuhl
- Aktueller Stand der Umsetzung am Lehrstuhl

- bietet insbesondere Hardwareabstraktion, Treiber, diverse Bibliotheken, Visualisierung, Nachrichtensystem und Paketverwaltung
- Großteil der Kernfunktionalitäten wurde von Willow Garage und Stanford entwickelt
- ros.org wird derzeitig von Willow Garage gepflegt
- BSD-Lizens(frei zu kopieren, zu verändern und zu verbreiten)
- Derzeitig ca. 300 Stacks(jeweils mit mehreren Packages) verfügbar
- Manche Stacks unter GPL und LGPL

Supported:

Ubuntu

Experimental:

OS X

Arch

Fedora

Gentoo

OpenSUSE

Slackware

Debian

Partial functionality:

Windows

FreeBSD

AscTec Pelican / Hummingbird

Care-O-bot

Erratic

Lego NXT

PR2

Shadow Robot

TurtleBot

Nao robot

Bilibot

Clearpath Robotics Kingfisher

Stanford Racing's Junior

Player Project Treiber leicht zu portieren

- genutzte Sprachen: C/C++, Phyton, XML und YAML
- ROS nutzt Boost
- OpenCV durchgehend für die Bildverarbeitung
- Für Ubuntu sind Paketquellen vorhanden

- Komplexe Funktionen werden durch Netzwerke von Knoten(Nodes) bewerkstelligt
- Ein Master stellt Kommunikation her
- Asynchrone Nachrichten durch Topics (Publisher und Subscriber)
- Synchrone Nachrichten durch Services (Service und Client; Request und Response)
- Konfiguration von Knoten durch Parameter

- Alle Ressourcen (z.B. Knoten, Parameter und Nachrichten) existieren in einer hierarchischen Namensstruktur
- Kapselung durch Namenspfad
- Verbindungen von Knoten durch remapping

Das nodelet-Paket:

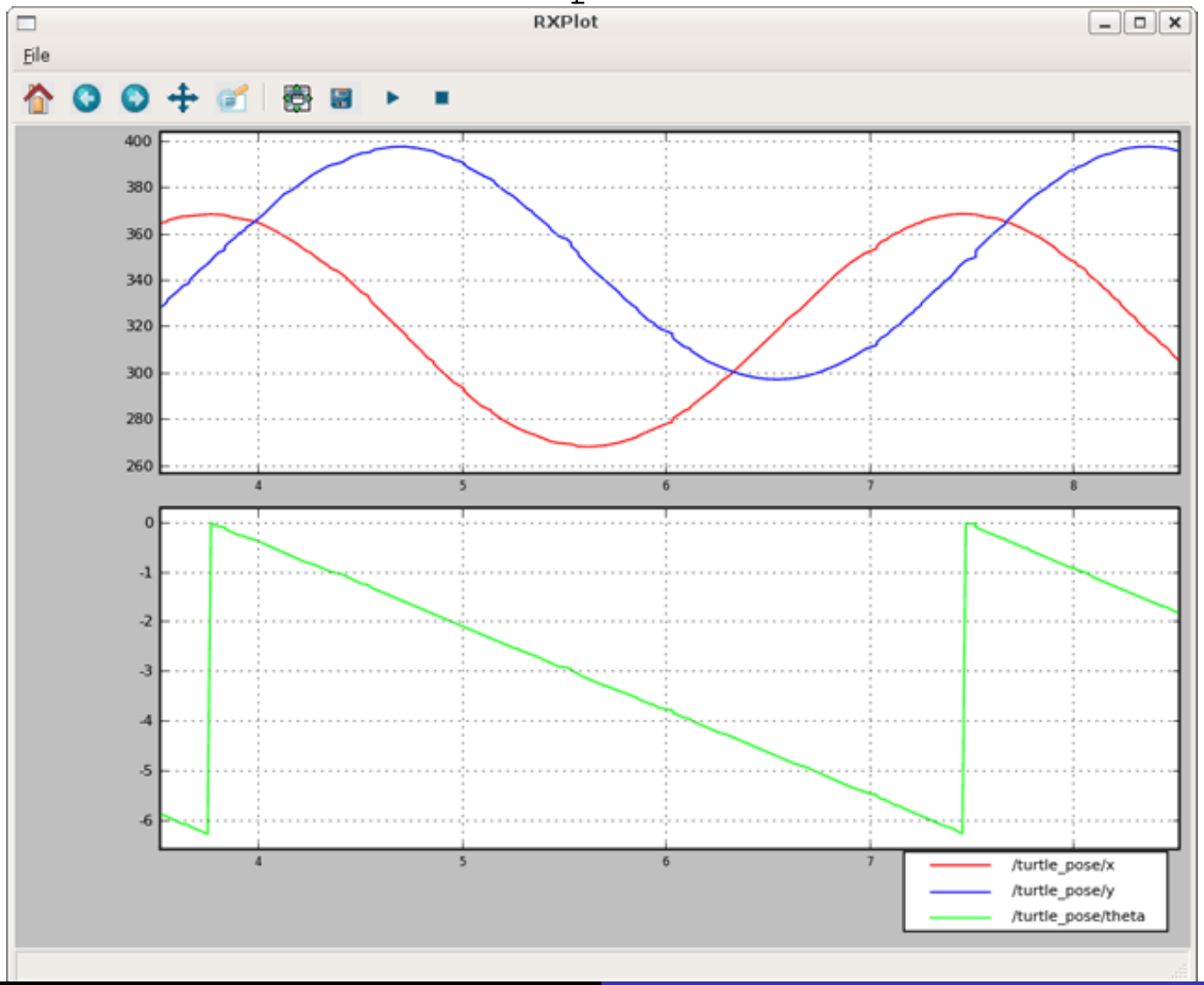
- das Kopieren von Daten zwischen zwei nodelet-Konten (auf einer Maschine) kostet nichts
- dynamisches Laden von Klassen
- der Code für einen `node` und einen `nodelet` ist nahezu identisch
- der `nodelet manager` verfügt über einen Threadpool

- `roscore`: startet das Kernsystem (Master)
- `rosrun`: startet ein einzelnes Programm (einen `node`)
- `roslaunch`: startet und konfiguriert ein Knotennetz(Berücksichtigung von Umgebungsvariablen; remote processes)

Parameter können gesetzt werden durch ...

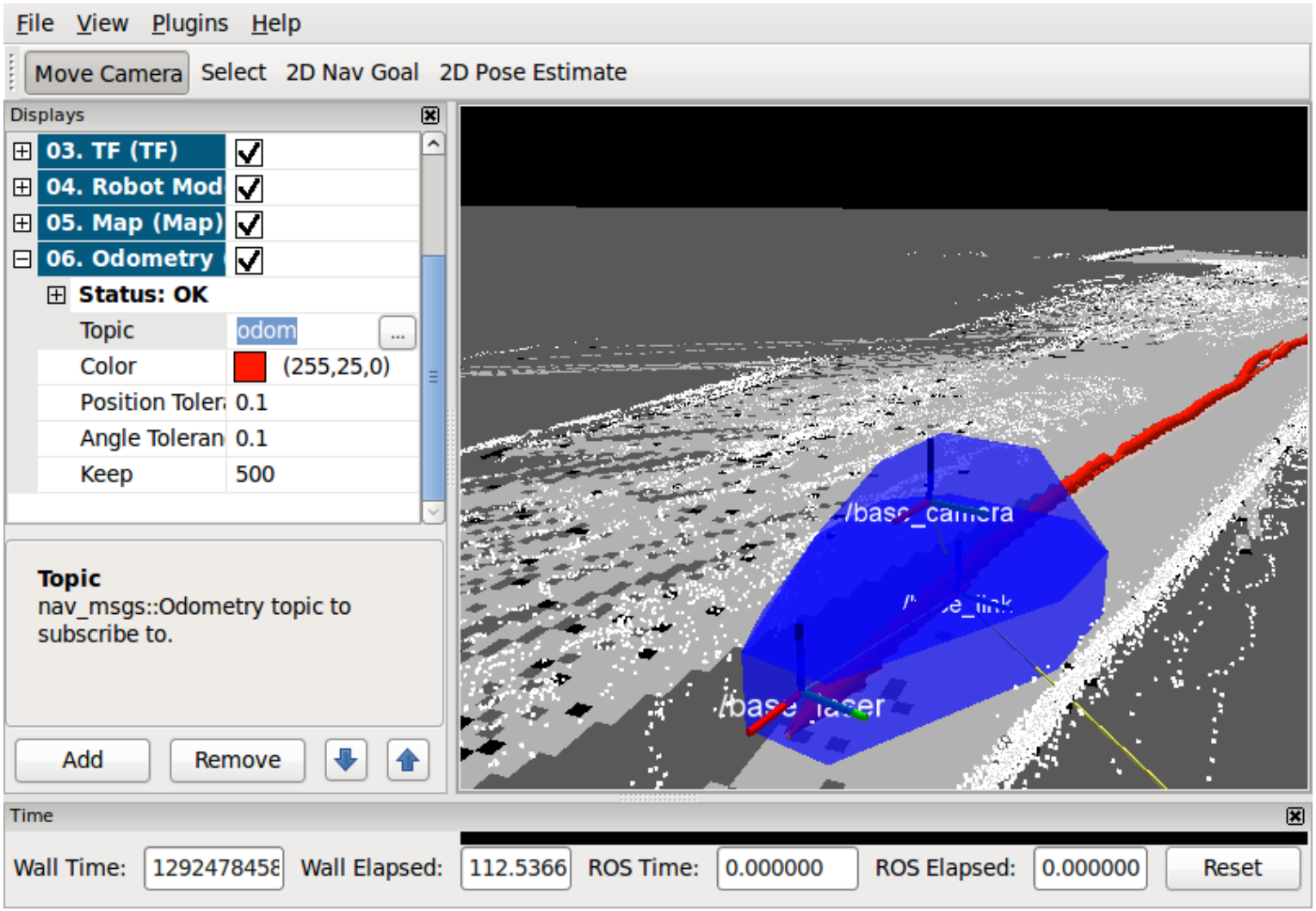
- Argumentangabe bei `rosrun`
- Angabe in der XML-Datei bei `roslaunch`
- Angabe in YAML-Dateien (Verweis in XML-Datei) bei `roslaunch`
- Manipulation mit `rosparam` auf dem Parameter Server
- Manipulation mit `dynamic_reconfigure`
- Konfigurationsdatei

rxplot

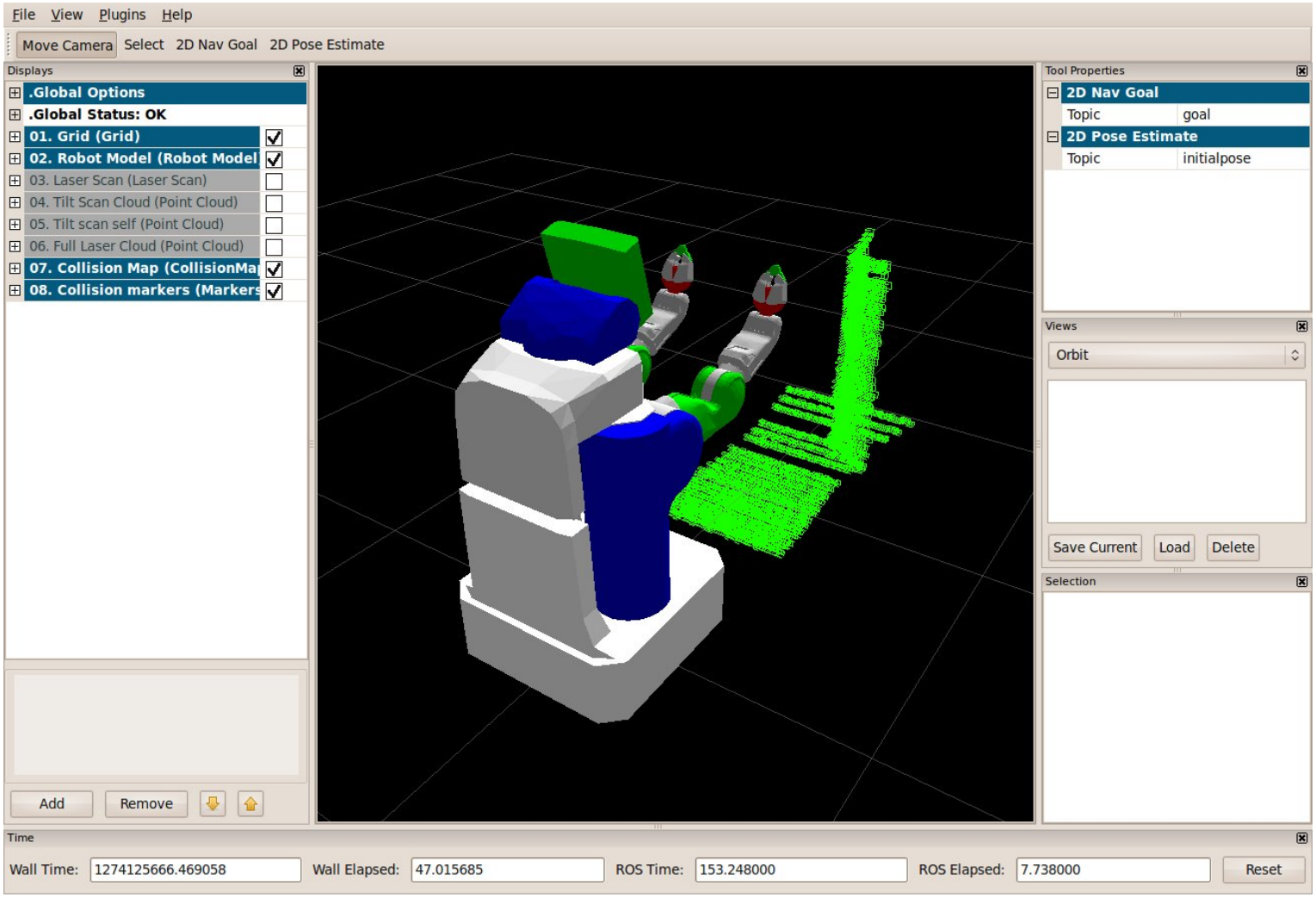

RXPlot
File
400
380
360
340
320
300
280
260
4
5
6
7
8
0
-1
-2
-3
-4
-5
-6
4
5
6
7
/turtle_pose/x
/turtle_pose/y
/turtle_pose/theta

- `rosbag`: zum Aufzeichnen und Abspielen von Nachrichten
- `rxbag`: zur Visualisierung von aufgezeichneten Daten(z.B. von der Kinect)

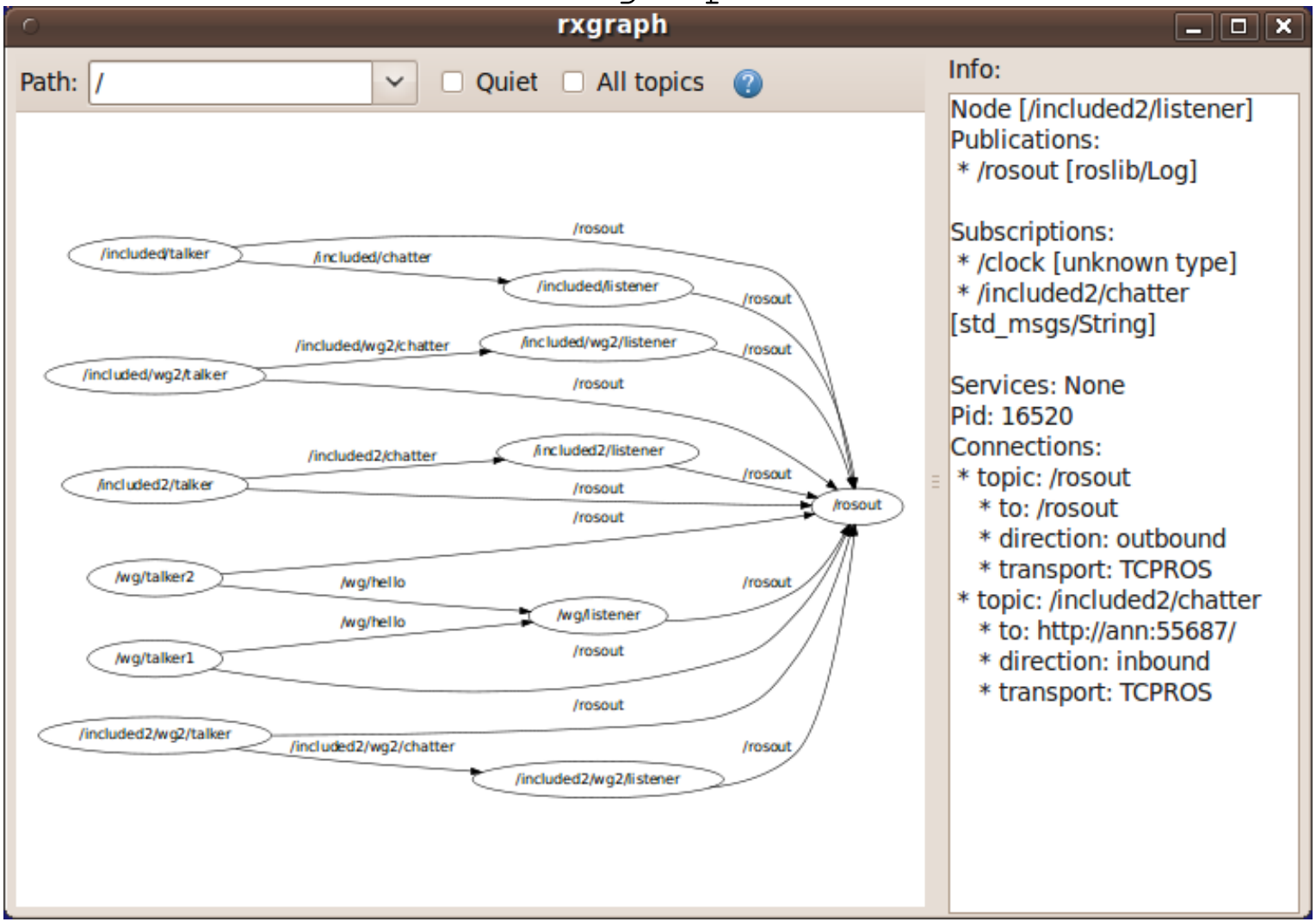
RViz(3D-Visualisierung)



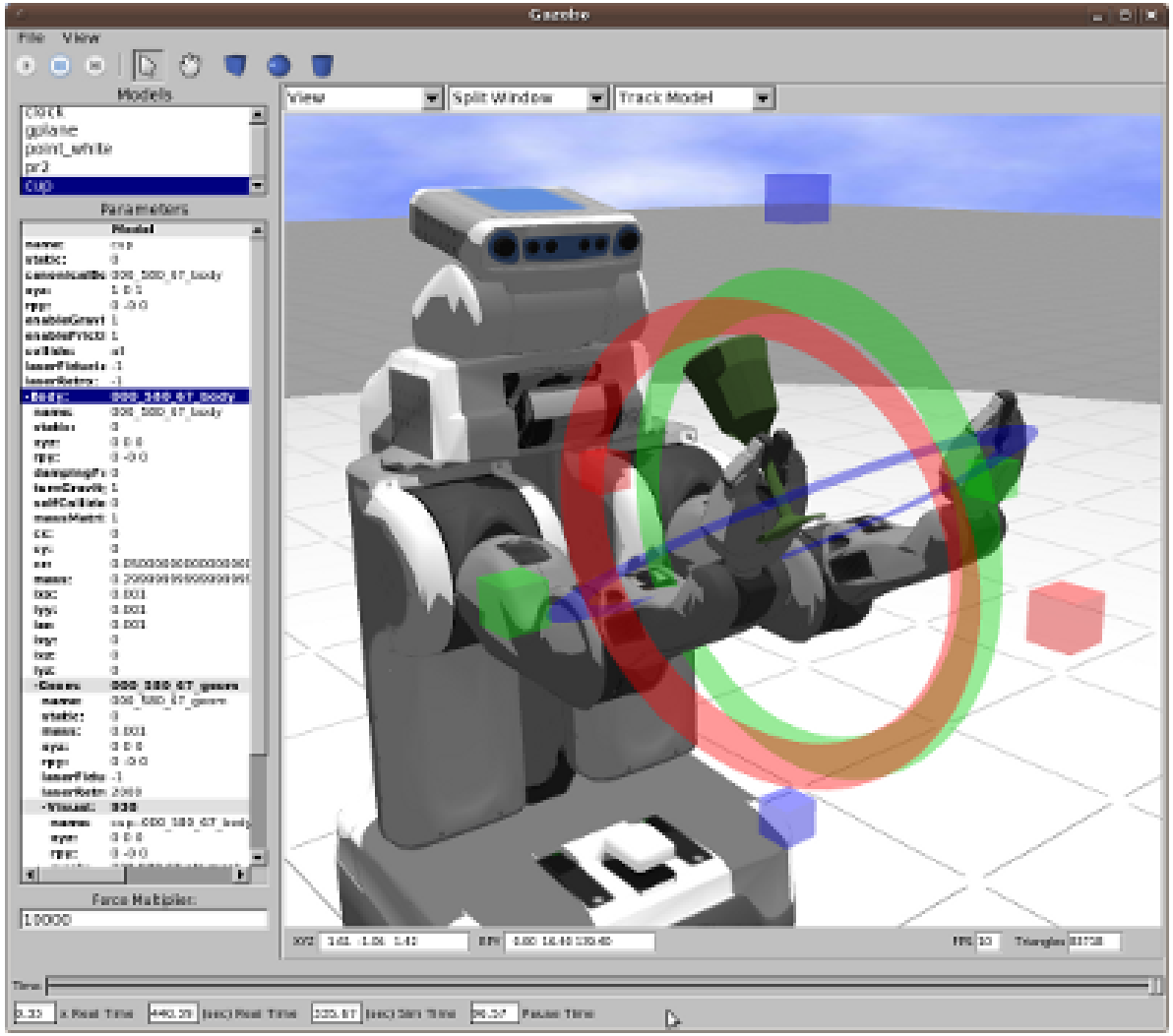
RViz(3D-Visualisierung)



rxgraph


rxgraph
Path: /
Quiet
All topics
Info:
Node [/included2/listener]
Publications:
* /rosout [roslib/Log]
Subscriptions:
* /clock [unknown type]
* /included2/chatter
[std_msgs/String]
Services: None
Pid: 16520
Connections:
* topic: /rosout
* to: /rosout
* direction: outbound
* transport: TCPROS
* topic: /included2/chatter
* to: http://ann:55687/
* direction: inbound
* transport: TCPROS
/included/talker
/included/chatter
/included/listener
/included/wg2/chatter
/included/wg2/listener
/included/wg2/talker
/included2/chatter
/included2/listener
/included2/talker
/rosout
/wg/talker2
/wg/hello
/wg/listener
/wg/talker1
/included2/wg2/talker
/included2/wg2/chatter
/included2/wg2/listener

Gazebo simulator(3D-Simulation)



- Qualität der Dokumentation auf dem Niveau der Dokumentation von OpenCV.
- Dies bedeutet: oft ausreichend und ansonsten hat man das “Glück” eines offenen Quelltextes
- jedoch gibt es zahlreiche Beispiele und Tutorial
- `rosdoc` nutzt (standardmäßig) doxygen zur automatisierten API-Generierung

- Im Bereich des Maschinellen Sehens ist das ROS sehr homogen
- In Bezug auf konkrete Treiber fehlt Beständigkeit
- Frage der korrekten Lösung bei vielen Optionen (z.B. wie und wo Parameter angeben)
- Das ROS ist extrem umfangreich
- Stetige Weiterentwicklung
- Relativ stabile API
- Extrem gute Tab-Vervollständigung
- Leider fehlt bisher eine zentralle GUI(incl. grafischer Bearbeitung der Netzwerke)

- Weiterbetrieb der ER1 Roboter mit aktueller Software
- Möglicherweise Beibehaltung der Software bei neuen Robotern
- Treibersammlung(z.B. joystick_drivers, camera_drivers, laser_drivers, sound_drivers, imu_drivers)
- Hardwareabstraktion für zukünftigen Sensoren
- Mächtige Visualisierungswerkzeuge auch ohne Roboter einsätzbar (reine Bildverarbeitung)
- Als Paketquelle für OpenCV

- Unterstützung der Hardware durch Ubuntu und ROS getestet $\Rightarrow$ es kann davon ausgegangen werden, dass die gesamte vorhandene Hardware am Lehrstuhl unterstützt wird
- Treiber für das Robot Control Module (RCM) von Evolution entwickelt (kann noch erweitert werden)
- eine MS Kinect auf Roboter montiert und erprobt
- W-Lan-Verbindung optimiert
- SVN: https://fsstud/svn/ros/
- (Trac-)Wiki: https://fsstud/wiki/ros/

- Wenn möglich (fast immer) nichts lokal auf den Rechner speichern!
- Quellcode, die in der Entwicklung sind und wahrscheinlich fehlerhaft sind gehörten meistens noch nicht ins SVN, sondern auf das eigene Netzwerklaufwerk (damit man selbst von jedem Rechner aus daran weiter arbeiten kann).
- Funktionsfähige Programme gehören in das SVN (damit auch die Gruppenmitglieder darauf zugreifen können).
- Alles was im SVN landet bleibt dort für immer (jede Version ist wiederherzustellen)!
- große Datenmenge(z.B. aufgenommene Sensordaten) sollten auf ein spezielles Netzwerklaufwerk

- erlaubt direkten Bezug auf den Quellcode
- die eigene Arbeit soll hier dokumentiert werden
- es muss möglich sein, dass in folgenden Projekten die geleistete Arbeit schnell nachvollzogen werden kann
- uns (somit auch für die Note) sehr wichtig!
- Informationen zum Umgang mit der Hardware und zur Arbeitsumgebung

- (Abbildungs-)Quelle: ros.org
- Tutorials: http://www.ros.org/wiki/ROS/Tutorials