ROS INTRODUCTION FOR SCHUNK COMPONENTS

Christopher Parlitz
Schunk Modular Robotics







Introduction to ROS



Research in robotics

- Reinvention of the Wheel
- Little Commonality
- Short Lifespan
- Inability to Compare Results
 - → ROS addresses these



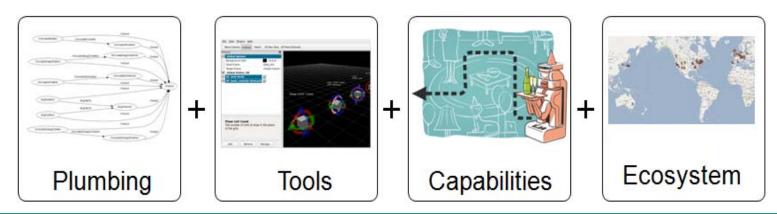




ROS – Robot Operating System



- ROS = Robot Operating System
- "ROS is an open-source, meta-operating system for your robot." [ROS-wiki]
- ROS is a "robot framework" [ROS-wiki]







ROS – Video



5 years of ROS



http://youtu.be/PGaXiLZD2KQ

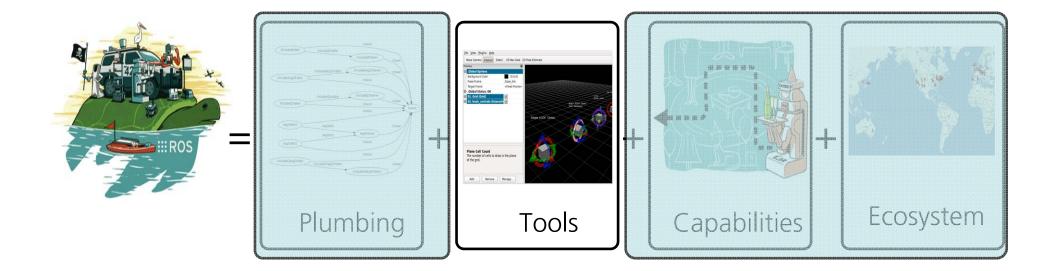
ROS – Robot Operating System



- What is ROS?
- Provides
 - Hardware abstraction
 - Low-level device control
 - Communication layer with message-passing between processes
 - Recursive package management and build system
 - Runs primarily on Linux but is intended to be cross-platform compatible to MAC OS X and Windows
- Content
 - ROS core build and runtime system
 - ROS packages, a collection of robotic algorithms

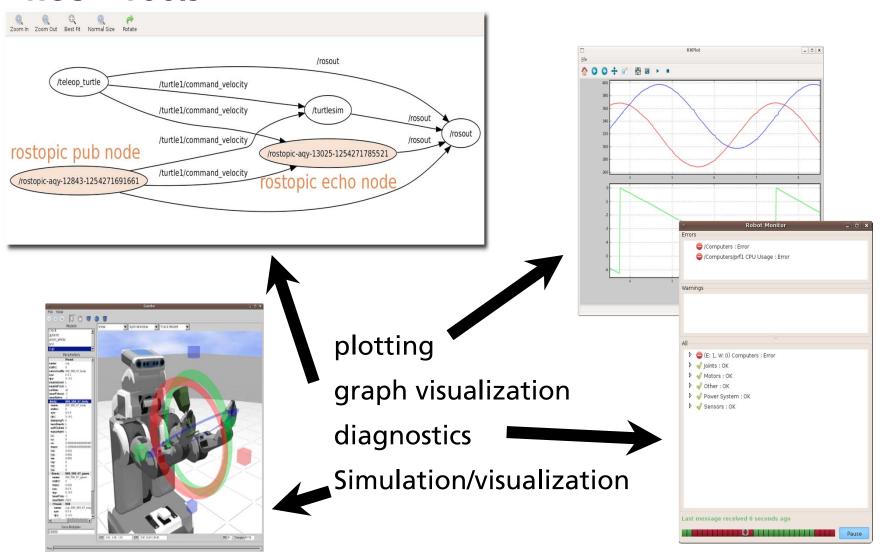
ROS – Tools





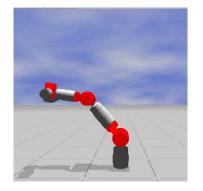
ROS – Tools

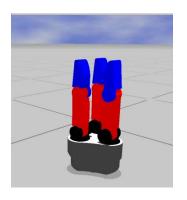


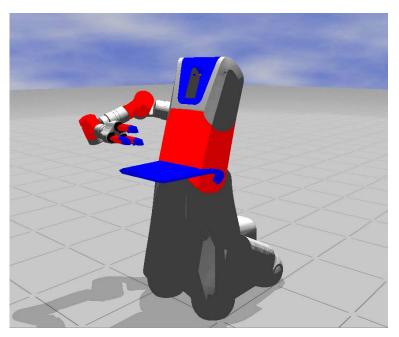




ROS – Tools Simulation







- Single simulated components
- Simulated sensors and actors, e.g.
 - Arm joints, hand
 - Cameras, laser scanners
- Model of the whole robot
- Kinematic and dynamic models of hardware components
- Environment model



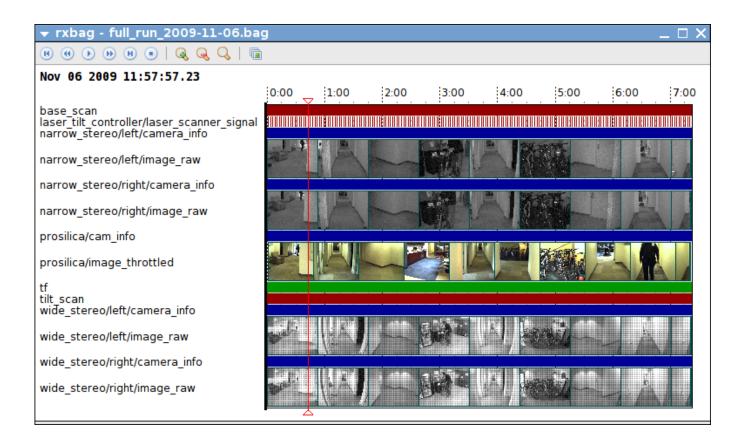








- Format for saving and playing back ROS message data
- Record and replay sensor data for developing and testing algorithms



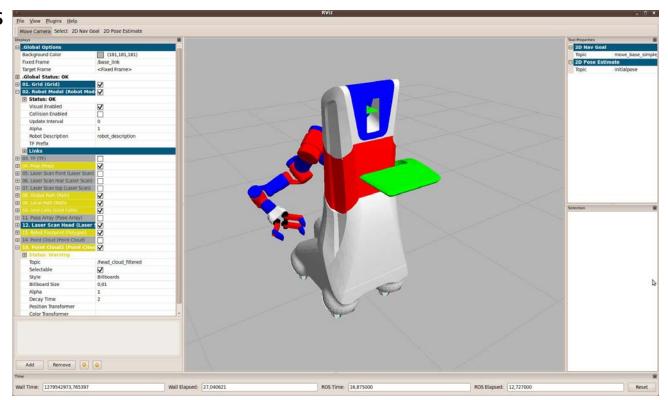




ROS – Tools Visualization



- Robot Model
- Sensor data (laser scanner, point cloud)
- 2D and 3D maps
- Markers



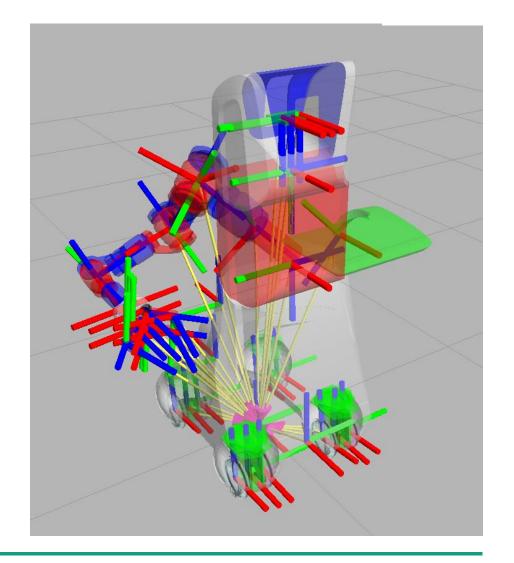








- Tree of coordinate systems
- Defined by urdf (Robot Description Language)
- Transformations between all coordinate systems available
- Generated automatically out of /joint_states topic

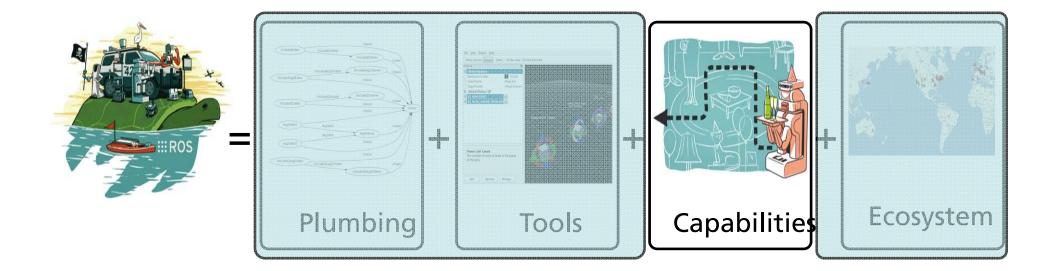






ROS – Capabilities





ROS – Capabilities



State of the art algorithms

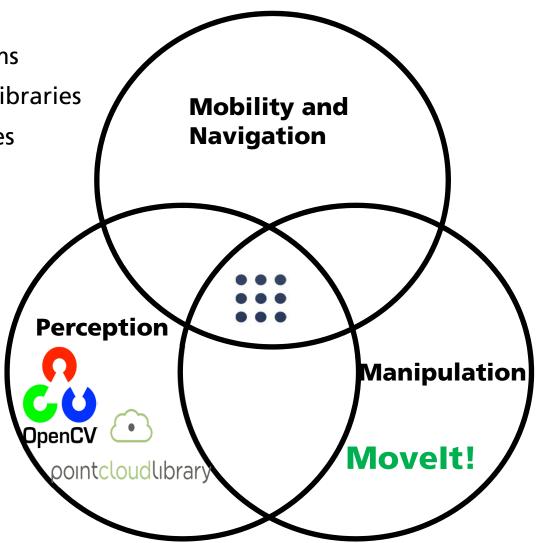
Integration of available libraries

Wide range of capabilities

Navigation

Perception

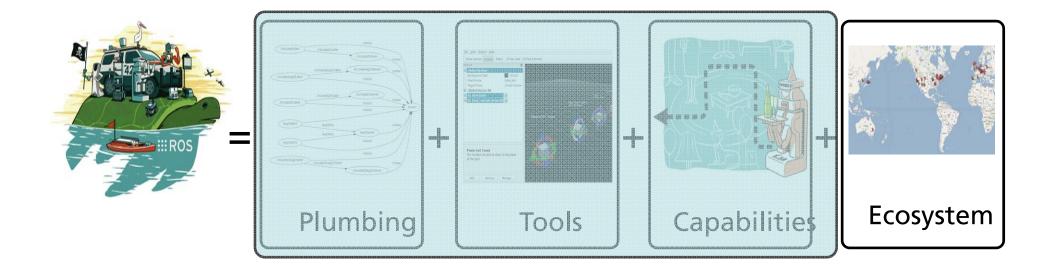
manipulation





ROS – Community/Ecosystem

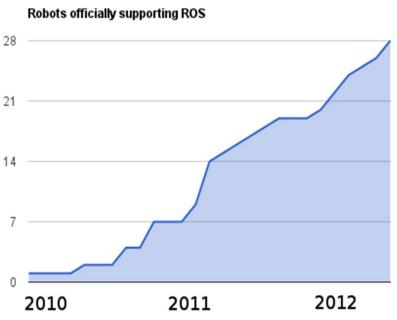




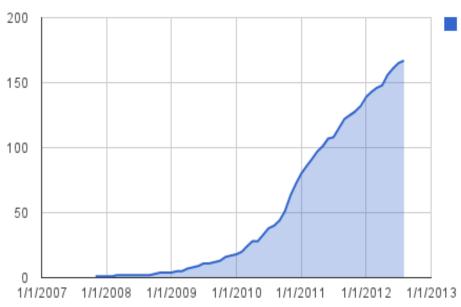
ROS – Community/Ecosystem



- Fast growing community
- De facto standard for service robotics



Publicly released and indexed repositories



ROS – Technical details



Three levels of ROS concepts

Robot Operating System

Filesystem Level

Packages

(Stacks)

Manifests

Messages

Services

Computational Graph Level

Nodes

Master

Parameter Server

Topic communication

Service communication

Bags

Community Level

Distributions

Repositories

ROS-Wiki

Mailing Lists

Blog





ROS – Filesystem Level

Packages

- Main unit for organizing software
- Typically one functionality, e.g. localisation or path planning
- Contains: runtime processes (nodes), libraries, datasets, configuration files, ...

Stacks

- Collection of packages
- Aggregate functionality, e.g. navigation stack
- Releases and versioning
- Stack- and Package- Manifests (*.xml)
 - Provide Metadata about a package/stack, e.g. license information and dependencies to other packages/stacks

ROS – Filesystem Level

- Messages types (*.msg)
 - Message descriptions, define data structures used for message communication
 - Language independent

TargetPoses.msg

Header header
Std_msgs/String name
Geometry_msgs/Pose2D[] poses

Pose2D.msg

Float64 x
Float64 y
Float64 theta

- Services types (*.srv)
 - Service descriptions, define request and response data structures used for service communication
 - Language independent

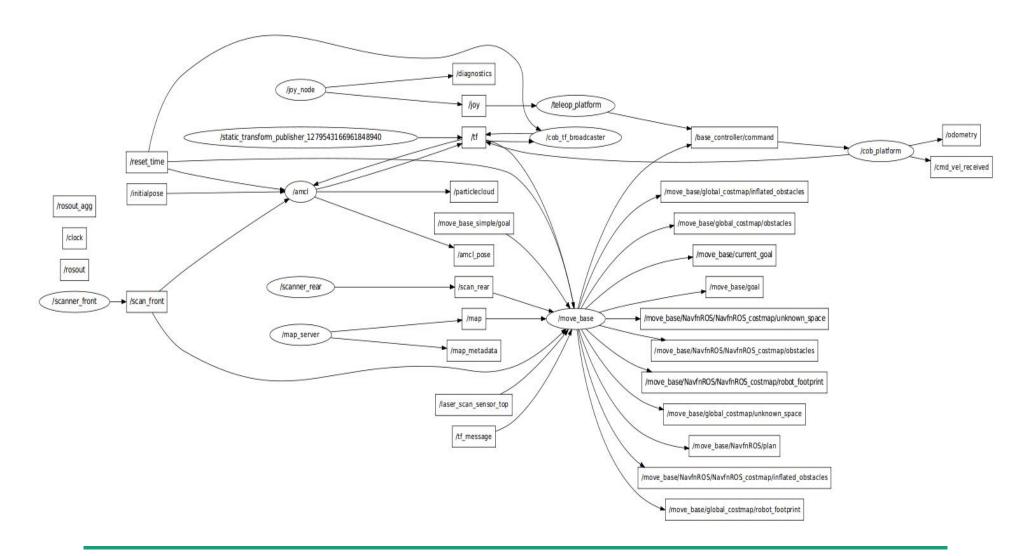
GetPose.srv

std_msgs/String name
-Geometry_msgs/Pose2D pose





ROS – Computational Graph Level





ROS – Computational Graph Level



- Processes to perform computation
- Usually many nodes at runtime
- Written by a ROS client library, e.g. roscpp, rospy,
- Master
 - Coordinating processes and communication
 - Name registration and lockup
- Parameter Server
 - Central location for storing data
- Messages
 - Nodes communicate by passing messages

Client Node a

register

register

Master

roscore

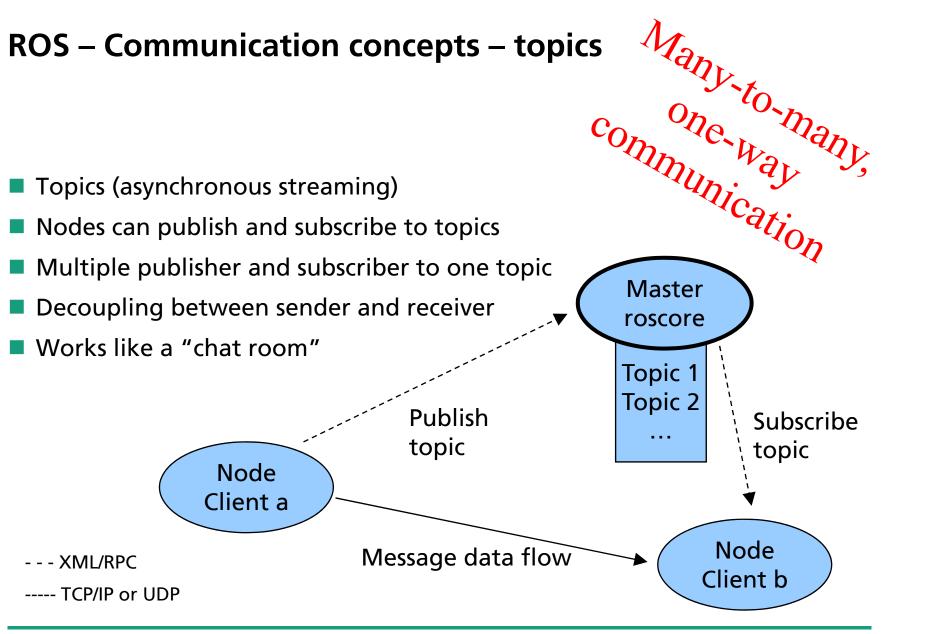
Node a

Node b

Client Node b

--- XML/RPC





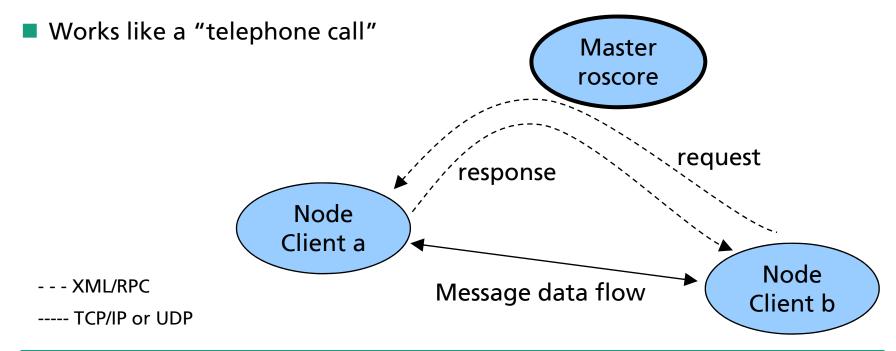




ROS – Communication concepts – services

ces One to one, communication

- Services (synchronous communication)
- Request and reply interaction
- Dedicated connection between two nodes



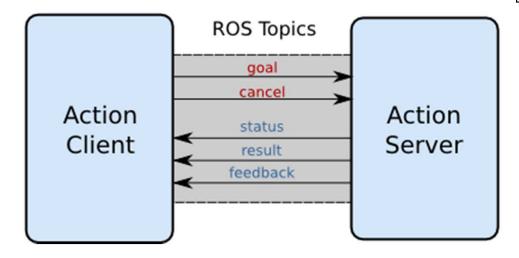




ROS – Communication concepts – actionlib

- Goal description similar to message and service definitions
- State-machine running on server and client

Action Interface



DetectBottle.action

```
#goal
std_msgs/String drink_name
---
#result
Geometry_msgs/Pose3D pose
---
#feedback
Int16 status
```





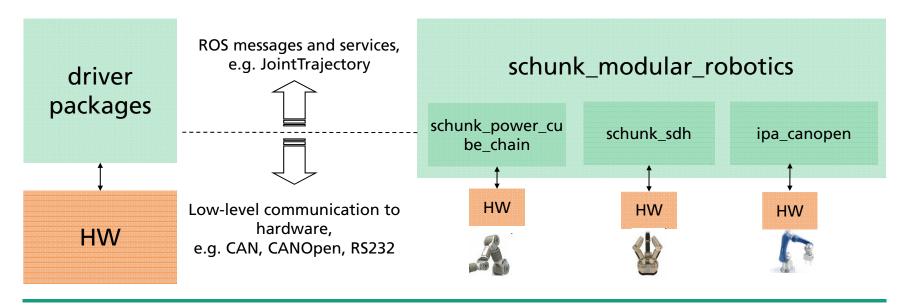


ROS support of Schunk components



ROS for Schunk components

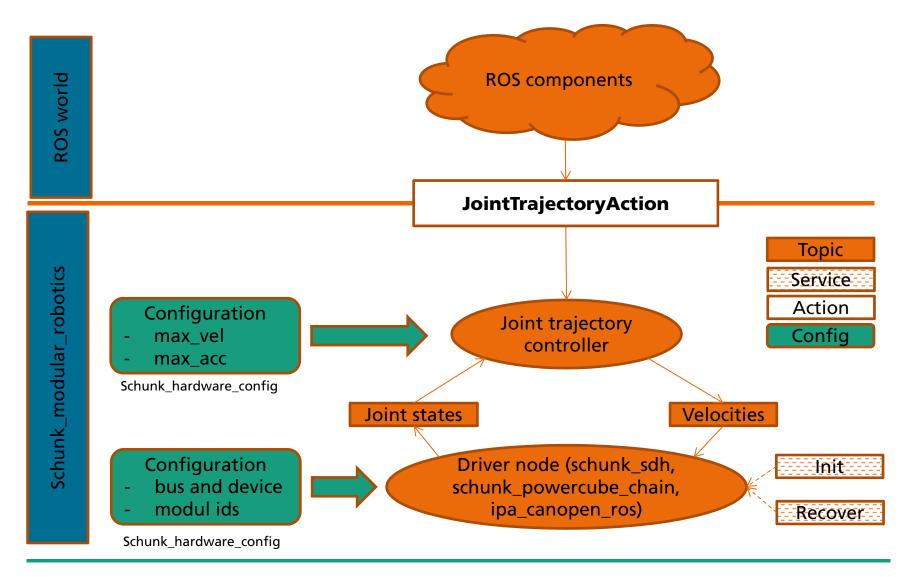
- Devices supported in ROS
 - SDH (Schunk Dextrous Hand, SDH library)
 - LWA (Lightweight Arm, M5API)
 - Powerball (Powerball arm, CanOpen)







SW Architecture



Installation and Operation Instructions



Installing ROS for Schunk components for Fuerte

- Install Ubuntu 12.04
- Install ROS Fuerte http://www.ros.org/wiki/fuerte/Installation/Ubuntu
- Download Schunk repositories to your ROS_PACKAGE_PATH
 - mkdir <<YOUR ROS_PACKAGE_PATH>>, e.g. ~/git/schunk_robots
 - cd <<YOUR ROS_PACKAGE_PATH>>

- UHR1
- rosinstall . https://raw.github.com/ipa320/schunk_robots/electric_dev/fuerte.rosinstall
- echo "export ROS_PACKAGE_PATH=<<YOUR ROS_PACKAGE_PATH>>:\$ROS_PACKAGE_PATH" >>~I.bashrc
- Source ~/.bashrc

Install additional dependencies

sudo apt-get install ros-fuerte-pr2-controllers ros-fuerte-arm_navigation ros-fuerte-arm-navigationexperimental ros-fuerte-audio-common ros-fuerte-pr2-power-drivers ros-fuerte-pr2-gui

Build Schunk packages

- rosdep install schunk_robots
- rosmake schunk robots

Configure your hardware (Example for powerball)

- roscd schunk_hardware_config/powerball/config
- Modify powerball_modules.yaml, see slide 40



ROS_PACKAGE_PATH: Missverständlich /typically: /opt UHR1

rosinstall: no ros stack in candidates

Reiser, Ulrich; 05.06.2013

Installing ROS for Schunk components for Fuerte

- Run driver (Example for Iwa with M5API)
 - roslaunch schunk_bringup powerball_solo.launch
- Move with dashboard (Example for Iwa with M5API)
 - roslaunch schunk_bringup dashboard_powerball.launch

UHR2

ROS_PACKAGE_PATH: Missverständlich /typically: /opt UHR2

rosinstall: no ros stack in candidates

Reiser, Ulrich; 05.06.2013

IPA CANopen Installation - Prerequisites

- 32Bit *nix operating system
- CMake (to manage the build process), in Ubuntu use the command

sudo apt-get install cmake

Git (to download the sources from github) in Ubuntu use the command

sudo apt-get install git

A C++ compiler with good support for the C++11 Standard

IPA CANopen Installation – CANopen library

- To install the C++ library independently from ROS
 - git clone git://github.com/ipa320/ipa_canopen.git
 - cd ipa_canopen/ipa_canopen_core
 - mkdir build
 - cd build
 - cmake ..
 - make
- To use the two command line tools, change the directory

cd ipa_canopen/ipa_canopen_core/build/tools

IPA CANopen Installation – ROS package

- To install the IPA CANopen ROS package
 - git clone git://gitub.com/ipa320/ipa_canopen.git
 - rosmake ipa_canopen_ros
- Test if the installation was successful
 - In one terminal

roscore

In another terminal

Rosrun ipa_canopen_ros canopen_ros

You get the following message

Missing parameter on server; shutting down

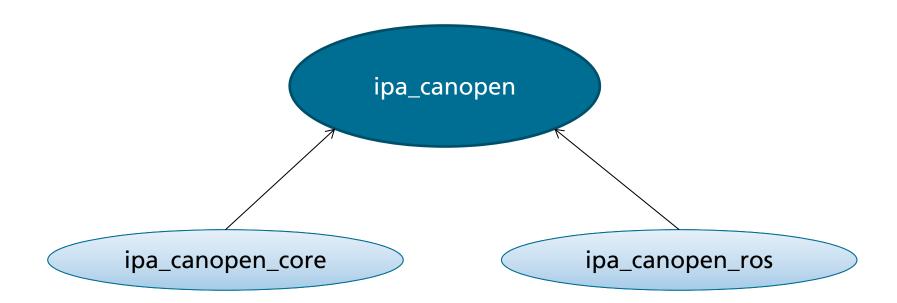


Technical details CANOpen



CanOpen Architecture

- C++ framwork
- Enables communication with CANopen motor devices
- Uses the pcan device driver
- Consits of two parts





ipa_canopen_core

- ROS independent library
- Provides two command line tools (in ipa_canopen/ipa_canopen_core/buidl/tools)
 - Homing (takes two arguments)
 - Name of the devicefile
 - CANdevice ID of the module
 - E.g. ./homing /dev/pcan32 12
 - Move device (takes five argumtents)
 - Name of the devicefile
 - CAN device ID of the module
 - Synchronization time
 - Target velocity in rad/msec
 - Target acceleration in rad/msec²
 - E.g. ./move_device /dev/pcan32 12 10 0.05 0.01





ipa_canopen_ros

- Wrapper to control CANopen motor devices in ROS
- Two ways to launch the node
 - Direct

```
rosrun ipa_canopen_ros ipa_canopen
```

■ Via a launchfile, e.g.

roslaunch schunk_bringup powerball_solo.launch

- In order to use the node you need
 - A trajectory controller
 - A rudimentary robot model (urdf)
- For a list of services, subcribed and published topics and necessary parameters on the parameter server follow the link below



Moving the Schunk LWA 4.6 (Powerball) arm

Driving the Schunk LWA 4.6 (Powerball) arm

- Make sure you have the following two repositories on your pc
 - Ipa320/schunk_robots.git
 - Ipa320/schunk_modular_robotics.git
- If not
 - git clone git://github.com/ipa320/schunk_robots.git
 - git clone git://github.com/ipa320/schunk_modular_robotics.git
- To launch the CANopen driver together with a trajectory controller roslaunch schunk_bringup powerball_solo.launch
- To launch the Powerball-arm in Gazebo
 - roslaunch schunk_bringup_sim powerball.launch
- To move the arm with a graphical command GUI
 - roslaunch schunk_bringup dashboard_powerball.launch
- To configure the necessary yaml-files: check the next slides



Configuring the Schunk LWA 4.6 (Powerball) arm I

Change directory

```
roscd schunk_hardware_config/powerball/config
```

Modify powerball_modules.yaml e.g.

Devices:

- name: /dev/pcan32 baudrate: 500K sync_interval: 10 Chains: ["arm_controller"]

Modify powerball_trajectory_controller.yaml e.g.

```
joint_names: ["arm_1_joint", "arm_2_joint", "arm_3_joint", "arm_4_joint", "arm_5_joint", "arm_6_joint"] module_ids: [3, 4, 5, 6, 7, 8] Devices: ["/dev/pcan32", "/dev/pcan32", "/dev/pcan32", "/dev/pcan32"]
```

Configuring the Schunk LWA 4.6 (Powerball) arm II

Change directory

roscd schunk_hardware_config/powerball/config

Modify powerball_CANopen.yaml e.g.

```
# canopen parameters can_module: PCAN can_baudrate: 1000
```

max_accelerations: [0.8, 0.8, 0.8, 0.8, 0.8]

OperationMode: position

```
# trajectory controll parameters
```

ptp_vel: 0.4 # rad/sec ptp_acc: 0.1 # rad/sec² max_error: 0.2 # rad

frequency: 100



Configuring the Schunk LWA 4.6 (Powerball) arm III

Change directory

roscd schunk_default_config/powerball/config

Modify powerball_joint_configurations.yaml e.g.

```
joint_names:
["arm_1_joint", "arm_2_joint", "arm_3_joint", "arm_4_joint", "arm_5_joint", "arm_6_joint"]

# back side positions
home: [[0,0,0,0,0,0]]
folded: [[0.32108866388214263, 0.6484189832579226, 2.06286710514828, -
1.2376313006847157, 5.658013215042093, -7.150174321779446e-05]]
wave_left: [[0.321033880484058, 0.49950722659008573, -0.4061025056033145, -
0.2370251233291425, 5.300248440143207e-06, 9.462633828505318e-06]]
wave_right: [[0.4741062629069983, -0.7912476227793528, 0.0041526706870680385, -
2.4662076334003302e-05, 2.4489075676648042e-05, 8.393716051102729e-06]]

# trajectories
wave_left-wave_right-home: [wave_left,wave_right,home]
```

Schunk Demo

Save new positions for the ROS parameter server:

rosrun schunk_demo save_position.py -p <position>

- home
- folded
- waveright
- waveleft

Moving through all the predefined positions:

rosrun schunk_demo demo_arm.py

Moving with the joystick

The joystick node is automatically loaded from the schunk_bringup.



ROS for Schunk components – command gui

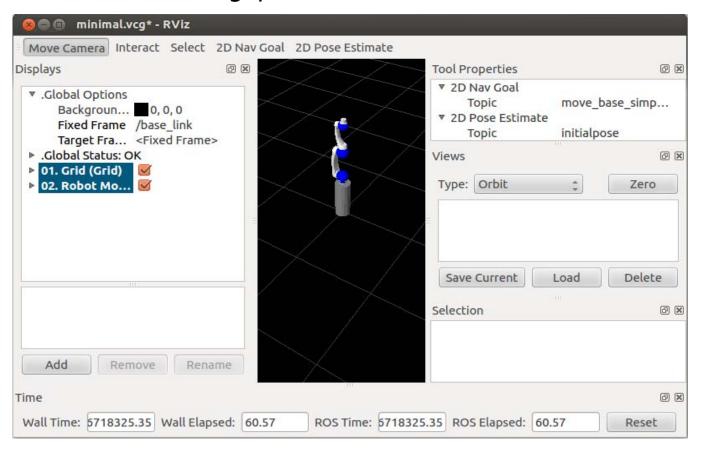
- Tool to easily move arm or hand
- Buttons for initializing and recover
- Buttons to move components in joint space
- "roslaunch schunk_bringup dashboard_lwa.launch"





ROS for Schunk components – rviz

- Tool to visualize robot model
- "roslaunch schunk_bringup rviz.launch"





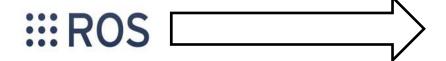
Thank you for your attention













www.schunk-modular-robotics.com www.ros.org/wiki/schunk_modular_robotics Contact:

Christopher Parlitz Christopher.Parlitz@de.schunk.com

