(1) urdf, blf

VF: - bridge ROS1 ↔ ROS2
- Speicher-Verteilung auf spez. Grid
- Syslogic

ROS

- package ≙ SW-org. einh. v. ROS-Code → libraries, exes, scripts...
- tab completion
- rospack rosls roscd
- uefi ≙ unified extensible firmware interface
- multisens packages (pack) ≙ multisens_...
- catkin (Palmkätzchen), da damals in Garten „vom Erfinder"
  - ↳ nachfolgend zu rosbuild
    - ↳ build system → kombiniert CMake macros und Python Skripts
    - ↳ Workflow ähnlich CMake
- ROS: source code in Packages organisiert → deren Code im Buildproz. zu ziehen (Binaries) targets
- CMakeLists.txt ≙ „CMake Makefile"
- packages möglichst in catkin-Workspace (in src)
- xdg-open → files öffnen
- bsd-Lizenz: berkeley sw distr → freizügige (sw) open-source Lizenzen
- distribution (ros) ≙ versioned set of ROS packages

! workspace auf environment überlegen → ~/ws/devel source setup.bash

- Überblick: - node → Executable, das ROS verwendet, um mit anderen Knoten zu kommunizieren
  - Message → ROS data type, um zu einem Topic zu subscriben o. publishen
  - Topic → nodes publishen msgs zu einem Topic oder empfangen/subscriben messages von einem Topic
  - Master → Namensdienst für ROS (um andere Nodes zu finden)
  - rosout → Äquivalent zu stdout
- node: - kann Service providen o. nutzen
  - verwendet ROS client lib (erlaubt Knoten, in allen versch. Progr.spr. geschr., zu kommuniz.)
    - ↳ rospy o. roscpp
- top (cmd) → table of processes (statisch: ps aux)

- bashrc ≙ Konfigdatei, die bei Start/Öffnen des Terminals geladen wird
  ↳ zwei gibt's: - global, liegt @ /etc/bash.bashrc → wirkt auf alle Benutzer
  - lokal, liegt versteckt (.) @ Homeverz.. (als .bashrc)
  ↳ grundsätzl. sollte man lokal eintragen
- rosout (läuft immer)
- rosnode cleanup (wenn node mit ⊗ anstatt ctrl + c gestoppt)
- roscore ≙ master (nameservice) + rosout + parameter service
  rosnode ≙ Tool, um Info über Knoten zu bekommen
  rosrun ≙ führt Knoten eines jew. Pkg aus (rosrun <pkg> <node>)
- rostopic (list)
- Kommunikation @ ROS via: ROS messages between nodes
  ↳ publisher u. subscriber müssen selben Nachrichttyp senden u. empfangen
  ↳ topic-typ def. durch msg-typ, die auf topic gesendet wird

z.B. /turtle1/pose
$\vee$
- rostopic type [topic]
- rostopic pub [topic] [msg-type] [args]
- rosnode info [node]
  $\wedge$
  turtlesim
- rqt_graph
- rqt_plot
- echo $HOME / echo $PATH
- rqt ≙ graph. ui framework (ROS + QT)

3

- rosservice list → listet services d. akt. lfd. Node
- services → erlauben Nodes, untereinander zu kommuniz.
  ↳ via requests und responses
- Bsp. Spawnen:
  rosservice type /spawn | rossrv show
  ↳ rosservice call /spawn 2 2 0.2 "" ← optional
- rosparam → erlaubt Daten zu manipulieren (auf ROS Param. Server)
  (d. akt. lfd. Node)
- Param setzen: rosparam set <param>
  rosservice call /clear
- Params. speichern (in File) u. lesen: rosparam dump/load <filename> <namesp> ← opt.
- logging levels: fatal, error, warn, info, debug (→)
- rosrun rqt_console rqt_c...
  rosrun rqt_logger rqt_lo...
- echo $ROS_PACKAGE_PATH
- roslauch <package> <filename.launch>
  ↑ launchfile (ähnl. XML) → liegt in: ws_dir/src/pkgname

9.

- edit files „ohne Auslagern": rosed [package_name] <tab><tab> /launch/ ↑
  echo $EDITOR
  export EDITOR = $\underbrace{\text{'nano -w'}}$
  ↳ gew. Editor
- msg: simple txt files that describe fields of a ROS msg → stored in msg dir
  (f. source code for msgs in diff. languages)
- srv: files that describe a service, composed of a req. and resp. part
  ↳ req. and resp. separated via '---'
  ↳ stored in srv dir
- vim [file_to_open_name]

- rosmsg show $\underbrace{\text{[pkg-name]/[msg-name]}}_{\text{o. only [msg-name]}}$

- roscp → copy (from/to ros-packages)

! - sudo service network-manager stop (Netzwerkmngr. aussch)

- Start skript: sh <skript-name>.sh

! - TODO: ≙ eth via shell
  - ↳ status, config,
  - = reread multisense ros

- sudo lshw -class network (Netzwerkressourcen anzeigen)
  - list, hardware
  - o. -C

- sudo ethtool eth#

  sudo ip addr add 10.162.66.200/24 dev eth#

- Firewall ≙ Traffic filter between network (clients)

- ifconfig → netw. interface config

- netstat -r (routing)

- host www.....de → gibt IP einer Adr.

- tracepath → zeigt Weg (IP-Adr. o.ä.) eines Pakets

- IPv4: $\underbrace{\_\_.\_\_.\_\_}_{\text{Netzwerkteil}}.\underbrace{\_\_}_{\text{Gerätteil}}$ | wenn Netzmaske 255.255.255.0
  - je 0...255

- $\underbrace{\text{192.168.2}}_{\text{erste 24 bits gesetzt}}$.107/24
  - erste 24 bits gesetzt ← ≙ Netzmaske 255.255.255.0

- ip addr (show) → zeigt akt. IP-Adr. (des Geräts/Host)

- nmap-scan: nmap $\underbrace{\text{192.168.2.107/24}}_{\text{IP d. akt. Geräts}}$

- daemon ≙ disk and execution monitor

bedingter Operator

- cd / → root
  cd ~ → home ($\hat{=}$ <username_of-account>)

- rmem_max → Empfangspuffergröße von Netzwerkpaketen
  wmem_max → Sendpuffergröße von Netzwerk "

- ifco... mtu 7200 (Ethernet norm. w. 1500)
  ↳ max transmission unit (Byte)

- sudo ifup eth0
  o.ä.

- netstat -i (list all inet-interfaces)

- Unterschied Kommunikation via Msgs o. Services
  Msgs: many-to-many via Topics
  Services: Request/Response (between individuals)

- Recording: mkdir ~/bagfiles
  cd ~/bagfiles
  rosbag record -a
  ↳ all published topics to be recorded

- rosbag info <your bagfile>
- " play " (im dir. wo's liegt)
- rosbag record -O subset /turtle1/cmd_vel ...
- ros::ok() → false, wenn Ctrl+C o. alle ros::NodeHandles zerstört
- Callback-Fcn: Fkt. die erst aufgerufen wird, wenn best. Ereignis eintritt
  „Rückruf-Fkt"
  ↳ kurz: Fkt, die in Argument einer anderen Fkt gesteckt u. ausgef. w. kann
- ros::spin(Once) nötig, um callbacks zu empfangen/aufzurufen
- executable: .bin
! netstat -tupn (zeige Eth.-Verbindungen)
- arp -a (zeigt alle aktiven IPs, die mit (z. NW verb. sind)

⑥

fudt123

- phython3-rospkg-modules
- phython3-catkin-pkg-modules
- phython3-pyparsing

- code ~/.bashrc
- apt list --install (listet alle inst. Pack. auf Ubuntu)
- uname -a (Ubuntu/Linux Version)
- su(sudo)
- unter 1500 @ CAD-N.
- rosrun rqt_reconfigure rqt_rec...
- rosrun rviz rviz
- du <foldername> (Speicherverbr. eines Verz.)
  + Unterverzeichnisse

look @:
- sensor_msg/PointCloud
→ " " 2
- read msg: contents via shell

- $PYTHONPATH: Umgebungsvar'en, die auf Verz. mit priv. Libs zeigen
  (wenn nichts angeg., dann wird per default python.exe aufgeru.)

unset PYTHONPATH

- ping <dst_ip> -I <src_dev>
- Dateityp (der Files im akt. Verz.) anzeigen: files *

Zusätlz. zu inst. Python-Packages (via pip3), für Super-Glue auf Melodic/18.04:
- tdqum
- torch (>1.5)
- numpy (>1.8)
- opencv (>3.2)
- ~~tools~~ 3.10.21 MD

BTW: Python3 add on (zu Melodic) installieren
↳ Melodic nicht from source bauen

- uninstall ros package: sudo apt-get remove <pkg-name>
- rosbag record -a / -O <name> /topic/...
- sudo -s
- rospack find <package_name>
- rostopic echo <topic> (| rosmsg show)
- rosmsg show
- history -c : löscht shell-historie
- ctl+c :