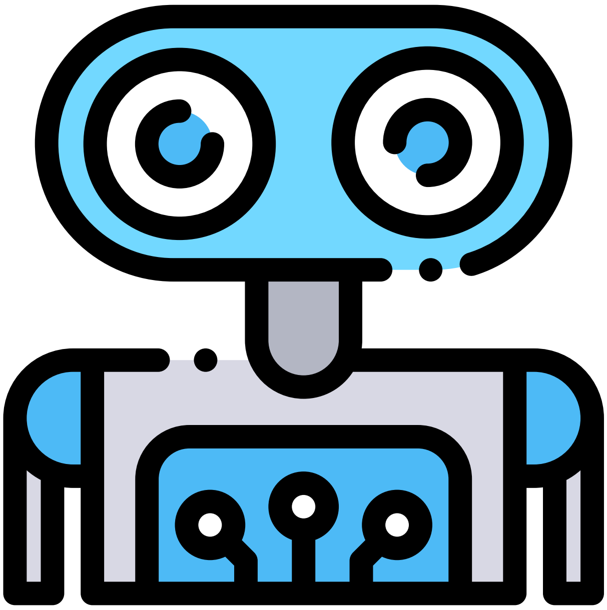
Zawiki

Release v0.1

tschinz

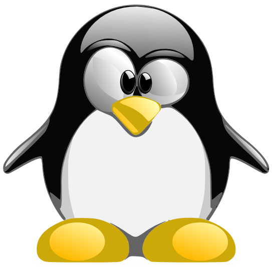
Feb 01, 2020

Content

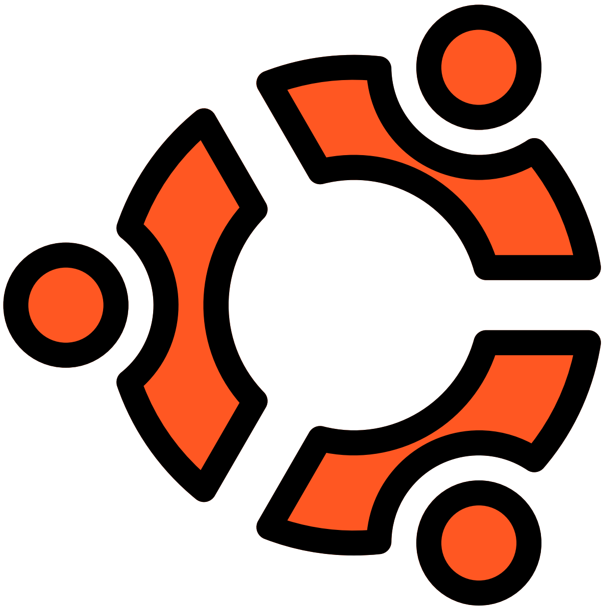


This Repo is a collection of markdown and ReStructuredText pages. Here you can find various informations about topics I've always forget. This pages let me help to remember less but know more.

1 Linux



1.1 Ubuntu



1.1.1 Installation and Config

Installation

This installation is based on Ubuntu 18.4 LTS and ROS Melodic Morenia.

Default Tools

```
sudo apt-get install git curl vim openssh-server krename rar unrar kget diffutils␣
↪kate x11vnc
echo "Configure Firewall and Port for ssh"
sudo ufw allow ssh
sudo ufw enable
sudo ufw status
sudo service ssh restart
```

ZSH

```
sudo apt-get install zsh
sudo chsh -s /bin/zsh $USER
```

Oh My ZSH

```
cd ~/Downloads
sh -c "$(curl -fsSL https://raw.github.com/robbyrussell/oh-my-zsh/master/tools/
↪install.sh)"
```

SublimeText 3

```
wget -qO - https://download.sublimetext.com/sublimehq-pub.gpg | sudo apt-key add -
sudo apt-get install apt-transport-https
echo "deb https://download.sublimetext.com/ apt/stable/" | sudo tee /etc/apt/
↪sources.list.d/sublime-text.list
sudo apt-get update
sudo apt-get install sublime-text
```

SublimeMerge

```
wget -qO - https://download.sublimetext.com/sublimehq-pub.gpg | sudo apt-key add -
sudo apt-get install apt-transport-https
echo "deb https://download.sublimetext.com/ apt/stable/" | sudo tee /etc/apt/
↪sources.list.d/sublime-text.list
sudo apt-get update
sudo apt-get install sublime-merge
```

Krusader

```
sudo apt-get install krusader
```

Yakuake

```
sudo apt-get install yakuake
```

FSearch

```
sudo add-apt-repository ppa:christian-boxdoerfer/fsearch-daily
sudo apt update
sudo apt-get install fsearch-trunk
```

Anaconda

```
cd ~/Downloads
wget https://repo.anaconda.com/archive/Anaconda3-2019.10-Linux-x86_64.sh
bash Anaconda3-2019.10-Linux-x86_64.sh
```

QT-Creator

```
cd ~/Downloads
wget http://download.qt.io/official_releases/qt/5.13/5.13.1/qt-opensource-linux-
↪x64-5.13.1.run
chmod +x qt-opensource-linux-x64-5.13.1.run
./qt-opensource-linux-x64-5.13.1.run
sudo apt-get install build-essential
sudo apt-get install libfontconfig1
sudo apt-get install mesa-common-dev
sudo apt-get install libglu1-mesa-dev -y
```

Visual Studio Code

```
curl https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > packages.
↪microsoft.gpg
sudo install -o root -g root -m 644 packages.microsoft.gpg /usr/share/keyrings/
sudo sh -c 'echo "deb [arch=amd64 signed-by=/usr/share/keyrings/packages.microsoft.
↪gpg] https://packages.microsoft.com/repos/vscode stable main" > /etc/apt/sources.
↪list.d/vscode.list'

sudo apt-get install apt-transport-https
sudo apt-get update
sudo apt-get install code # or code-insiders
```

Configuration

Oh My ZSH Config

Listing 1: ~/.zshrc additions

```
echo"#-----------------------------------------------------------------------------
↪-" >> ~/.zshrc
echo "# Program in Path" >> ~/.zshrc
echo "#" >> ~/.zshrc
echo "#----------------------------------------------------------------------------
↪--" >> ~/.zshrc
echo "# Special zsh config" >> ~/.zshrc
echo "# Show hidden files and folders" >> ~/.zshrc
echo "setopt globdots" >> ~/.zshrc
echo "#----------------------------------------------------------------------------
↪--" >> ~/.zshrc
echo "# Goto Alias" >> ~/.zshrc
echo "# Common home locations" >> ~/.zshrc
echo "alias home='cd ~'" >> ~/.zshrc
echo "alias root='cd /'" >> ~/.zshrc
echo "alias dtop='cd ~/Desktop'" >> ~/.zshrc
echo "alias dwld='cd ~/Downloads'" >> ~/.zshrc
```

(continues on next page)

(continued from previous page)

```
echo "alias docs='cd ~/Documents'" >> ~/.zshrc
echo "alias www='cd /var/www/html'" >> ~/.zshrc
echo "alias workspace='cd ~/Workspace'" >> ~/.zshrc
echo "alias aptlock-rm='sudo rm /var/lib/dpkg/lock && sudo rm /var/lib/dpkg/lock-
↪frontend'" >> ~/.zshrc
echo "# Common commands" >> ~/.zshrc
echo "alias o=open" >> ~/.zshrc
echo "alias ..='cd ..'" >> ~/.zshrc
echo "alias ...='cd ..; cd ..'" >> ~/.zshrc
echo "alias ....='cd ..; cd ..; cd ..'" >> ~/.zshrc
echo "# Common command shortcuts" >> ~/.zshrc
echo "alias cls=clear" >> ~/.zshrc
echo "alias ll='ls -la'" >> ~/.zshrc
```

SublimeText 3 Config

Listing 2: ~/.zshrc additions

```
echo "# Sublime Text" >> ~/.zshrc
echo "export PATH=$PATH:/opt/sublime_text" >> ~/.zshrc

echo "# Sublime Text" >> ~/.bashrc
echo "export PATH=$PATH:/opt/sublime_text" >> ~/.bashrc

cp ./../config/sublimetext/Package Control.sublime-settings ~/.config/sublime-text-
↪3/Packages/User/
```

SublimeMerge Config

Listing 3: ~/.zshrc additions

```
echo "#Sublime Merge" >> ~/.zshrc
echo "export PATH=$PATH:/opt/sublime_merge" >> ~/.zshrc

echo "#Sublime Merge" >> ~/.bashrc
echo "export PATH=$PATH:/opt/sublime_merge" >> ~/.bashrc
```

How To Use Ubuntu Tools

SSH

SSH connection without password

```
# On your local machine generate a RSA Key pair
ssh-keygen -t rsa

# Copy your local public key to the remote machine safely
ssh-copy-id -i ~/.ssh/id_rsa.pub "<user>@<remoteip> -p <portnumber>"
# OR
scp id_rsa.pub <user>@<remoteip>:~/.ssh/machine.pub

# Append key to file authorized_keys
cat ~/.ssh/*.pub | ssh <user>@<remoteip> -p <portnumber> 'umask 077; cat >>.ssh/
↪authorized_keys'
```

(continues on next page)

(continued from previous page)

Open SSH Connection

```
# Just ssh
ssh <user>@<remoteip>

# ssh with portforwarding
ssh -L <local-port>:localhost:<remote-port> <user>@<remoteip>
# ssh with vnc port forwarding
ssh -L 5900:localhost:5900 spl@<remoteip>
```

VNC

On remote PC x11vnc needs to be installed and launched. Prefereable add to startup commands

Create password

Only needed if not only localhost used.

```
x11vnc -storepasswd
```

Launch x11vnc

```
# Command with all options
x11vnc -usepw -forever -display :0 -safer -bg -o ~/Documents/log/vnc/x11vnc.log -
↪localhost

# Minimal command but still restricted to localhost
x11vnc -forever -display :0 -safer -bg -localhost
```

1.1.2 Introduction

Additional Informations

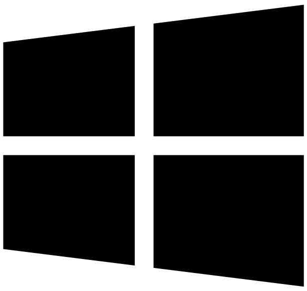
- https://ubuntu.com/ - Ubuntu Webpage
  - https://ubuntu.com/#download - Ubuntu Download
- https://www.osboxes.org/ubuntu/ - Virtual Box images

- Additional Tools
  - ZSH
    - Oh My ZSH

2

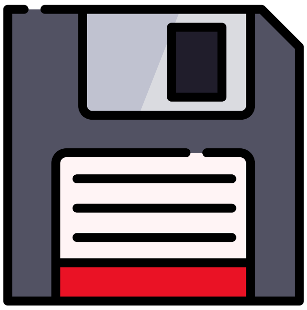
Mac

3 Windows

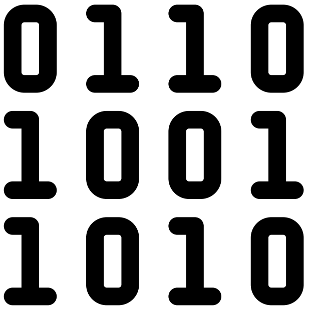


4

Tools



5 Coding



5.1 ROS - Robot Operating System

5.1.1 Introduction

- *Philosophy*

ROS aka Robotic Operating System is not a OS itself but a framework and middleware.

- Software Framework for programming robots
- Prototype from Standfort AI Research Institute and created by Willow Garage in 2007
- Since 2013 maintained by the Open Source Robotics Foundation (OSRF)
- Consists of infrastrucutre, tools, capabilities and a ecosystem

Table 1: Source : ROS Tutorial #1 - https://www.youtube.com/watch?v=9U6GDonGFHw&t=1s

Advantages	Disadvantages
Provides lots of infrastructure, tools and capabilities	Approaching maturity, but still changing
Easy to try other people’s work and shar your own	Security and scalability are not first-class concerns
Large community	OSes other than Ubuntu Linux are not well supported
Free, open source, BSD license	
Great for open-source and researchers	**Not great for mission-critical tasks**

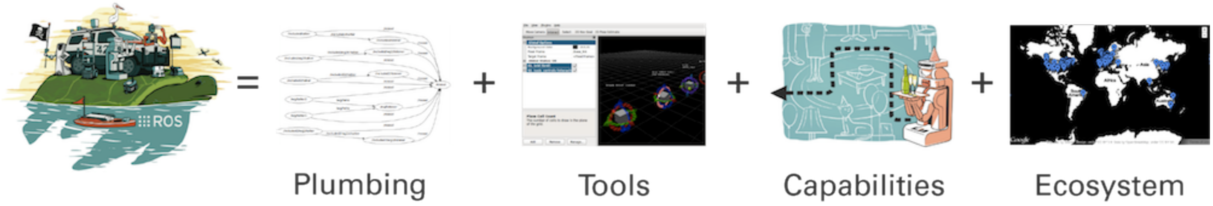


Fig. 1: ROS Equation

Plumbing	Tools	Capabilities	Ecosystem
Process management	Simulation	Control	Package organization
Inter-process communication	Visualization	Planning	Software distribution
Device drivers	Graphical user interface	Perception	Documentation
	Data logging	Mapping	Tutorials
		Manipulation	

Philosophy

- **Peer to peer** - Individual programs communicate over defined API (ROS messages, services, etc.).
- **Distributed** - Programs can be run on multiple computers and communicate over the network.
- **Multi-lingual** - ROS modules can be written in any language for which a client library exists (C++, Python, MATLAB, Java, etc.).
- **Thin** - The ROS conventions encourage contributors to create standalone libraries and then wrap those libraries so they can send and receive messages to and from other ROS modules.
- **Free and open source** - The core of ROS is released under the permissive BSD license, which allows commercial and noncommercial use.

5.1.2 Basics

Coding Rules

The following rules apply when writing code with ROS.

Table 2: ROS Robot Programming by TurtleBot3 Developers, section 7.1.3

Type	Naming Rule	Example
Package	under_scored	`first_ros_package`
Topic, Service	under_scored	`raw_image`
File	under_scored	`turtlebot3_fake.cpp`
Namespace	under_scored	`ros_awesome_package`
Variable	under_scored	`string table_name;`
Type	camelCased	`typedef int32_t PropertiesNumber;`
Class	camelCased	`class UrlTable`
Structure	camelCased	`struct UrlTableProperties`
Enumeration Type	camelCased	`enum ChoiceNumber`
Function	camelCased	`addTableEntry()`
Method	camelCased	`void setNumEntries(int32_t num_entries)`
Constant	ALL_CAPITALS	`const uint8_t DAYS_IN_A_WEEK = T;`
Marco	ALL_CAPITALS	`#define PI_ROUNDED 3.0`

Standard Unit in ROS

Table 3: Source : ROS Robot Programming by TurtleBot3 Developers, section 7.1.1

Quantity	Unit
Length	Meter
Mass	Kilogram
Time	Second
Current	Ampere
Angle	Radian
Frequency	Hertz
Force	Newton
Power	Watt
Voltage	Volt
Temperature	Celsius

Master

ROS `master` is a Server tracking all network addresses of all nodes. In addition to network addresses it also tracks other information like parameters. All `nodes` must know the network address of the master on startup `ROS_MASTER_URI`.

A `master` can be started with the `roscore` command or a `roslaunch` will also start a `master` if it doesn't exists already.

```
roscore
```

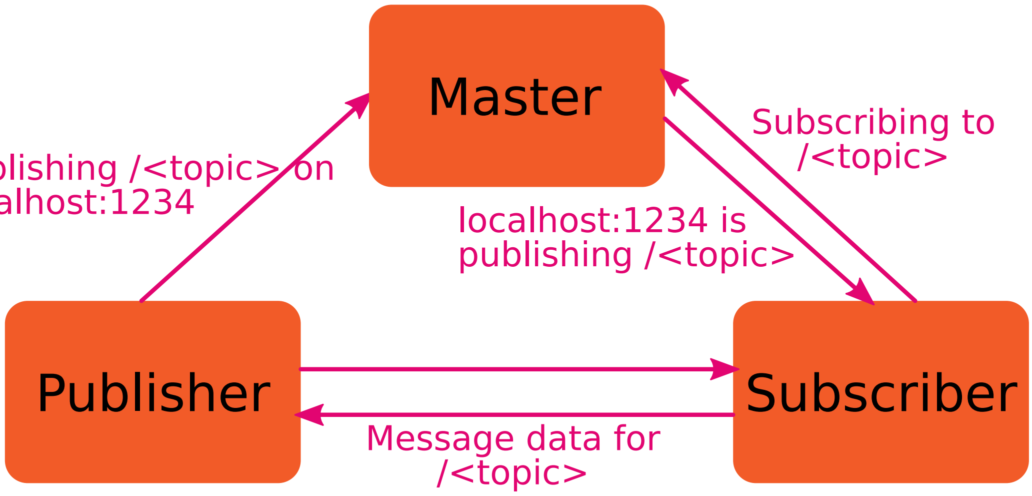

Master
Publishing /<topic> on localhost:1234
Subscribing to /<topic>
localhost:1234 is publishing /<topic>
Publisher
Subscriber
Message data for /<topic>

Fig. 2: ROS Master Publisher Slave

Publisher and Subscribers

With help of the `master`, `publisher` and `subscriber` establish a peer-to-peer connection. All `nodes` must know the network address of the master on startup `ROS_MASTER_URI`.

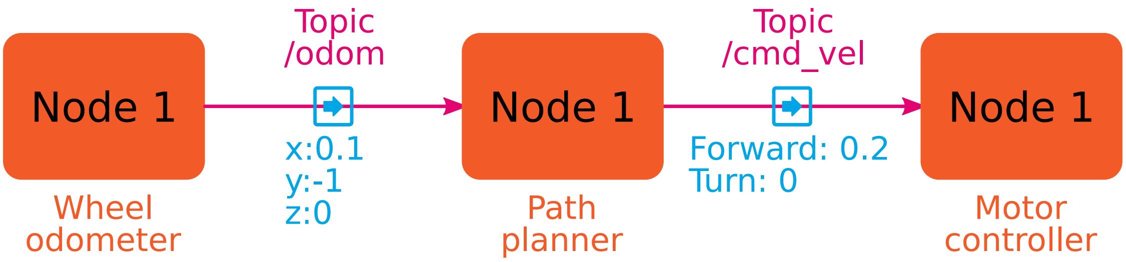


Fig. 3: ROS Publisher Slave

- Any node can publish a message to any topic
- Any node can subscribe to any topic
- Multiple nodes can publish to the same topic
- Multiple nodes can subscribe to the same topic
- A node can publish to multiple topics
- A node can subscribe to multiple topics

Catkin Overview

`src/` Folder

Location for create and clone new packages

The command `catkin_make` searches only in the `src/` folder for packages and builds them

It is a good practice to clone the ros packages into a different folder e.g. `~/git/<package_name>` and create a symlink into you catkin workspace

```
ls -s ~/git/<package_name>/ ~/catkin_ws/src/
```

`build/` Folder

`catkin_make` create buixld files and intermediate cache CMake files inside the `build/` folder.

devel/ Folder

catkin_make builds each package, if successful, the target executable file is created. Executables are stored inside the devel/ folder. Current workspace packages can be access by the command line if the following command is used:

```
# for bash
source ~/<workspace_name>/devel/setup.bash

# for zsh
source ~/<workspace_name>/devel/setup.zsh
```

It is beneficial to add this the the ~/.bashrc or ~/.zshrc file.

In addtion there is the catkin_tools program which simplifies the use.

See dedicated page: *Catkin Tools*

install/ Folder

After building the executables in the devel/ folder, this executables can be install by:

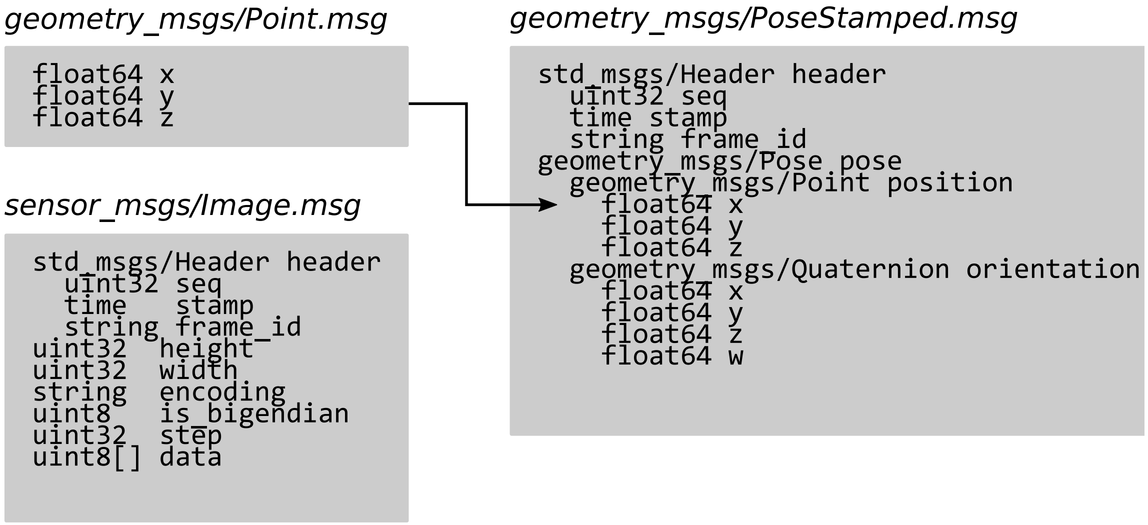
```
catkin_make install
```

See also:

- http://wiki.ros.org/catkin/workspaces#Catkin_Workspaces

Messages

- Serialization format for structured data
- Defined in a .msg file
- Compiled to C++/Python classes before using them
- more info https://wiki.ros.org/Messages



geometry_msgs/Point.msg

```
float64 x
float64 y
float64 z
```

sensor_msgs/Image.msg

```
std_msgs/Header header
  uint32 seq
  time   stamp
  string frame_id
uint32  height
uint32  width
string  encoding
uint8   is_bigendian
uint32  step
uint8[] data
```

geometry_msgs/PoseStamped.msg

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
    float64 x
    float64 y
    float64 z
  geometry_msgs/Quaternion orientation
    float64 x
    float64 y
    float64 z
    float64 w
```

5.1.3 Books summary

Topics

Basic topics such as workspace description, packages and nodes creation can be found in most of the book mentioned in this summary. They are not part of this summary since it focuses on more advanced topics. Tutorials to understand those topics are available in books or on the ROS wiki.

This summary lists all the books we have related to ROS, and some more specific PDF documents. Storage of the referenced documents :

- books : ros/books/
  - Effective_Robotics_Programming_with_ROS_3E.pdf
  - Learning_ROS_for_Robotics_Programming_2E.pdf
  - Mastering_ROS_for_Robotics_Programming.pdf
  - Programming_Robots_with_ROS.pdf
  - Programming_Robots_with_ROS-A_Practical_Introduction_to_the_Robot_Operating_System.pd
  - Robot_Operating_System_for_Absolute_Beginners.pdf
  - ROS_Robot_Programming.pdf
  - ROS_Robotics_By_Example.pdf
  - ROS_Robotics_By_Example_2E.pdf
  - Teach_ROS_with_No_Hassle_2E.pdf
- other documents : ros/slides/
  - octomap.pdf
  - ros-ethz-1.pdf
  - ros-ethz-2.pdf
  - ros-ethz-3.pdf
  - ros-ethz-4.pdf
  - ros-ethz-5a.pdf
  - ros-ethz-5b.pdf
  - ros-ethz-5c.pdf

  - ros-misc.pdf
  - ros-tf.pdf
  - ros-tf-2.pdf

SLAM (Simultaneous localization and modeling)

- Mastering_ROS_for_Robotics_Programming.pdf page 146

TF (Transform Frames)

- Effective_Robotics_Programming_with_ROS_3E.pdf page 171
- Learning_ROS_for_Robotics_Programming_2E.pdf page 305

QR code reader

- TODO

3D

- Effective_Robotics_Programming_with_ROS_3E.pdf page 120
- Learning_ROS_for_Robotics_Programming_2E.pdf page 143
- Mastering_ROS_for_Robotics_Programming.pdf page 265

BAG recording

- Effective_Robotics_Programming_with_ROS_3E.pdf page 128
- Learning_ROS_for_Robotics_Programming_2E.pdf page 120

Odometry and navigation

- Effective_Robotics_Programming_with_ROS_3E.pdf page 179
- Learning_ROS_for_Robotics_Programming_2E.pdf page 303
- Mastering_ROS_for_Robotics_Programming.pdf page 140

Point Clouds

- Effective_Robotics_Programming_with_ROS_3E.pdf page 394
- Learning_ROS_for_Robotics_Programming_2E.pdf page 231
- Mastering_ROS_for_Robotics_Programming.pdf page 251

OpenCV

- Effective_Robotics_Programming_with_ROS_3E.pdf page 359
- Mastering_ROS_for_Robotics_Programming.pdf page 250

5.1.4 Catkin Tools

Catkin build system

This Python package provides command line tools for working with the catkin meta-buildsystem and catkin workspaces. These tools are separate from the Catkin CMake macros used in Catkin source packages. It has to be installed seperately.

- https://catkin-tools.readthedocs.io/

Installation Catkin Tools

```
# Add ROS Repositories
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu `lsb_release -sc` main" >␣
↪/etc/apt/sources.list.d/ros-latest.list'
wget http://packages.ros.org/ros.key -O - | sudo apt-key add -

# Install via apt-get
sudo apt-get update
sudo apt-get install python-catkin-tools

# Install via pip
sudo pip install -U catkin_tools
```

Cheat Sheet

This is a non-exhaustive list of some common and useful invocations of the `catkin` command. All of the commands which do not explicitly specify a workspace path (with `--workspace`) are assumed to be run from within a directory contained by the target workspace. For thorough documentation, please see the chapters on each verb.

Initialize Workspaces

Initialize a workspace with a default layout (`src/build/devel`) in the *current* directory:

```
catkin init
catkin init --workspace .
catkin config --init
mkdir src && catkin build
```

… with a default layout in a *different* directory:

```
catkin init --workspace /tmp/path/to/my_catkin_ws
```

… which explicitly extends another workspace:

```
catkin config --init --extend /opt/ros/indigo
```

Initialize a workspace with a **source space** called `other_src`:

```
catkin config --init --source-space other_src
```

… or a workspace with **build**, **devel**, and **install space** ending with the suffix `_alternate`:

```
catkin config --init --space-suffix _alternate
```

Configuring Workspaces

View the current configuration:

```
catkin config
```

Setting and unsetting CMake options:

```
catkin config --cmake-args -DENABLE_CORBA=ON -DCORBA_IMPLEMENTATION=OMNIORB
```

```
catkin config --no-cmake-args
```

Toggle installing to the specified **install space**:

```
catkin config --install
```

Building Packages

Build all the packages:

```
catkin build
```

… one at a time, with additional debug output:

```
catkin build -p 1
```

… and force CMake to re-configure for each one:

```
catkin build --force-cmake
```

Build a specific package and its dependencies:

```
`catkin build <package_name>
```

… or ignore its dependencies:

```
catkin build <package_name> --no-deps
```

Build the package containing the current working directory:

```
catkin build --this
```

… but don’t rebuild its dependencies:

```
catkin build --this --no-deps
```

Build packages with additional CMake args:

```
catkin build --cmake-args -DCMAKE_BUILD_TYPE=Debug
```

… and save them to be used for the next build:

```
catkin build --save-config --cmake-args -DCMAKE_BUILD_TYPE=Debug
```

Build all packages in a given directory:

```
catkin build $(catkin list -u /path/to/folder)
```

… or in the current folder:

```
catkin build $(catkin list -u .)
```

Cleaning Build Products

Blow away the build, devel, and install spaces (if they exist):

```
catkin clean
```

… or just the **build space**:

```
catkin clean --build
```

… or just clean a single package:

```
catkin clean PKGNAME
```

… or just delete the build directories for packages which have been disabled or removed:

```
catkin clean --orphans
```

5.1.5 Commandline Commands

Commandline Variables

```
echo $<variable_name>         # To display value

ROS_DISTRO                    # Distro name e.g. melodic
ROS_ETC_DIR                   #
ROS_LISP_PACKAGE_DIRECTORIES # common-lisp folder e.g. ~/catkin_ws/devel/share/
↪common-lisp
ROS_HOSTNAME                  # ros hostname e.g. localhost
ROS_MASTER_URI                # ros master url e.g. http://localhost:11311
ROS_PACKAGE_PATH              # package path's e.g. ~/catkin_ws/src:/opt/ros/$(ROS_
↪DISTRO)/share
ROS_PYTHON_VERSION            # python version 2 or 3 e.g. 2
ROS_ROOT                      # ros installation e.g. /opt/ros/$(ROS_DISTRO)/share/
↪ros
ROS_VERSION                   # ros version 1 or 2 e.g. 1
```

Useful commands

ROS tools

roscore

Launch ROS master core

```
roscore
```

rosversion

```
rosversion -d                           # Print ROS distro name
rosversion <package_name>               # Print package vrosnode
```

rosparam

Nodes use the parameter server to store and retrieve parameters at runtime.

http://wiki.ros.org/rosparam

```
rosparam list                           # list parameter names
rosparam set /<parameter_name> <value>  # set parameter
rosparam get /<parameter_name>          # get parameter
rosparam delete /<parameter_name>       # delete parameter
rosparam dump <file>                    # dump parameter to file
rosparam load                           # load parameter from file
```

rosnode

Work with nodes

```
rosnode list                            # list all nodes
rosnode ping /<node_name>               # check node connectivity
rosnode info /<node_node>               # print information about node
rosnode machine                         # list nodes running on a particular␣
↪machine
rosnode kill /<node_name>               # kill a running node
```

rostopic

Work with topics

```
rostopic list                           # list all topics
rostopic info /<topic_name>             # print information about active topic
rostopic echo /<topic_name>             # print messages to screen
rostopic pub /<topic_name> msg/MessageType "data:value" # pubish data to topic

rostopic type /<topic_name>             # print topic or field type
rostopic find <type>                    # find topics by type
rostopic bw /<topic_name>               # display bandwidth used by topic
rostopic hz /<topic_name>               # display publishing rate of topic
```

roslaunch

To start a launch file which can contain multiple nodes.

```
roslaunch <ros_pkg_name> <launch_file_name> # Launch ros launch file
```

rosrun

To run a node

```
rosrun <ros_pkg_name> <node_name>          # Start a ros node
rosrun <PACKAGE_NAME><NODE_NAME> __name:=<INSTANCE_NAME> # Start another instance␣
↪of a node, the parameter *INSTANCE_NAME* can be whatever you want, but it must␣
↪be unique.
```

rosservice

Work with services

```
rosservice list                          # list active services
rosservice info <service_name>           # print information about service
rosservice uri <service_name>            # print service ROSRPC
```

rosbag

ROS offers the possibility to record the data published on topics into bag files :

1. create a directory to store the bag files:

   ```
   ~/ mkdir ros_bag_files && cd ros_bag_files
   ```

2. run the *record* command :

   ```
   rosbag record -O <bag_name>.bag <topic_name> <topic_name>
   ```

3. play the bag file :

   ```
   rosbag play <bag_name>.bag
   ```

Many options are available for the *rosbag* command, see this page for more details.

Note : to play a bag with point clouds, it is required to have the following topics :

- /cloud
- /tf_static

The TF transformation is required, otherwise RViz can't display the point clouds.

```
rosbag record -O cloud.bag /cloud /tf_static
...
rosbag play cloud.bag
```

rosmsg

Display information about ros messages.

```
rosmsg list                     # List all messages
rosmsg info <message_name>      # Show message description
rosmsg package <package_name>   # List messages in a package
rosmsg packages <package_name>  # List packages that contain messages
```

Other Commands

```
roscd <PKG_NAME>                 # move to the folder of the package

rosinstall <PKG_NAME>            # install a ROS package

rosdep <PKG_NAME                 # install all the dependencies of a package

rqt                              # tool with many plug-ins available such as topic␣
↪publisher, service caller, …

rqt_graph                        # display the connections between nodes

rviz                             # launch the graphical tool to visualize robots,␣
↪point clouds, …

view_frames                      # create a PDFcalled ``frames.pdf`` with the TF␣
↪frames that are active
evince frames.pdf                # show with evince the generated frames.pdf
```

Catkin

More info:

- http://wiki.ros.org/catkin/Tutorials

Create Package

1. new terminal
2. navigate to the source folder of the catkin workspace : `.../catkin_ws/src`
3. run : `catkin_create_pkg <PACKAGE_NAME> <DEPENDENCIES>`
4. update both CMakeLists.txt and package.xml note : *run_depend* has to be replaced by the *exec_depend*
5. write source code in the source folder of the package :
6. build the catkin workspace with the alias command : `cm`
7. launch the master as explained [here](ros-commands.md#roscore).
8. now launch the node as explained [here](#roslaunch) and [here](rosrun).

```
catkin_create_pkg <PKG_NAME> <PKG_DEPENDENCIES> # create a package, must be called␣
↪inside a catkin workspace
```

Build

```
cm
catkin_make                                     # build the whole workspace
catkin_make <PKG_NAME>                          # build a single package
```

Install

```
catkin_make install                             # installs all executables
catkin_make install <PKG_NAME>                  # installs single executables
```

Python modules

Tips :

- put the script in a folder called *scripts*
- make sure to run `chmod +x <script_name>.py` so that the script is recognized as an executable by ROS

Update services with RQT

1. launch *RQT* from a new terminal : run `rqt`
2. Search for the plugin *Service Caller*
3. Choose the service that you want to update
4. Fill the *expression* field with an expected parameter of this service
5. Call the service and the response is displayed

5.1.6 Installation

How to install ROS

This installation is based on Ubuntu 18.4 LTS and ROS Melodic Morenia.

Prerequisites

Some tools are not mandatory.

NTP

Only needed in a multi-pc system.

```
echo "Install Chrony and ntpdate"
sudo apt-get install -y chrony ntpdate
sudo ntpdate -q ntp.ubuntu.com
```

Sources

ROS Ubunbtu apt-get packages sources.

```
echo "Add ROS Package Sources"
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
↪ /etc/apt/sources.list.d/ros-latest.list'
```

Ubuntu 18.04 LTS (Bionic Beaver)

```
echo "Add ROS Package Sources for Ubuntu 18.04 LTS Bionic Beaver"
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" >
↪ /etc/apt/sources.list.d/ros-latest.list'
```

Keys

- ROS Kinetic
- ROS Melodic

```
echo "Add ROS Package Key"
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key␣
↪C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

ROS Base

```
echo "Install ROS Base Desktop Full"
sudo apt-get install ros-melodic-desktop-full
```

ROS Additional Packages

RQT

```
echo "Install ROS R-QT"
sudo apt-get install ros-melodic-rqt*
```

Individual ROS packages

Search & install individual ROS packages

```
echo "Install ROS R-QT"
apt-cache search ros-melodic
sudo apt-get install ros-melodic-[NAME_OF_PACKAGE]
```

Setup ROS Environment

Initialise rosdep

```
echo "[Initialize rosdep]"
sudo sh -c "rosdep init"
rosdep update
```

Environment setup

Differs depending if it's zsh or bash

```
echo "[Environment setup and getting rosinstall]"
if [ -n "$ZSH_VERSION" ]; then
   # assume Zsh
  source /opt/ros/$name_ros_version/setup.zsh
elif [ -n "$BASH_VERSION" ]; then
  # assume Bash
  source /opt/ros/$name_ros_version/setup.sh
fi
```

ROS Install

```
sudo apt install -y python-rosinstall python-rosinstall-generator python-wstool
```

Create catkin workspace

```
echo "[Make the catkin workspace and test the catkin_make]"
mkdir -p $HOME/$name_catkin_workspace/src
cd $HOME/$name_catkin_workspace/src
catkin_init_workspace
cd $HOME/$name_catkin_workspace
catkin_make
```

Shell Scripts

All the above can be done with help of the `ros-melodic-install.bash`

Additional Install

Hitachi SDK

```
cd ~/Downloads
echo "$INDENT Manually download http://hlds.co.jp/download/tofsdk/v2.3.0/
↪HldsTofSdk.2.3.0ubuntu16_x64.zip into your Downloads/ folder"
echo ""
echo "PRESS [ENTER] WHEN YOU'RE FINISHED AND TO CONTINUE THE INSTALLATION"
read
mkdir HldsTofSdk.2.3.0ubuntu16_x64
unzip HldsTofSdk.2.3.0ubuntu16_x64.zip -d ./HldsTofSdk.2.3.0ubuntu16_x64
sudo apt install HldsTofSdk.2.3.0ubuntu16_x64/libtof-dev_2.3.0-4ubuntu16_amd64.deb
```

Configuration

ROS Configuration

.bashrc

```
echo "[Set the ROS evironment in ~/.bashrc]"
echo "alias eb='vim ~/.bashrc'" >> ~/.bashrc
echo "alias sb='source ~/.bashrc'" >> ~/.bashrc
echo "alias gs='git status'" >> ~/.bashrc
echo "alias gp='git pull'" >> ~/.bashrc
echo "alias cw='cd ~/$name_catkin_workspace'" >> ~/.bashrc
echo "alias cs='cd ~/$name_catkin_workspace/src'" >> ~/.bashrc
echo "alias cm='cd ~/$name_catkin_workspace && catkin_make'" >> ~/.bashrc

echo "source /opt/ros/$name_ros_version/setup.bash" >> ~/.bashrc
echo "source ~/$name_catkin_workspace/devel/setup.bash" >> ~/.bashrc

echo "export ROS_MASTER_URI=http://localhost:11311" >> ~/.bashrc
echo "export ROS_HOSTNAME=localhost" >> ~/.bashrc
```

.zshrc

```
echo "[Set the ROS evironment in ~/.zshrc]"
echo "alias eb='vim ~/.zshrc'" >> ~/.zshrc
echo "alias sb='source ~/.zshrc'" >> ~/.zshrc
echo "alias gs='git status'" >> ~/.zshrc
echo "alias gp='git pull'" >> ~/.zshrc
echo "alias cw='cd ~/$name_catkin_workspace'" >> ~/.zshrc
echo "alias cs='cd ~/$name_catkin_workspace/src'" >> ~/.zshrc
echo "alias cm='cd ~/$name_catkin_workspace && catkin_make'" >> ~/.zshrc

echo "source /opt/ros/$name_ros_version/setup.zsh" >> ~/.zshrc
echo "source ~/$name_catkin_workspace/devel/setup.zsh" >> ~/.zshrc

echo "export ROS_MASTER_URI=http://localhost:11311" >> ~/.zshrc
echo "export ROS_HOSTNAME=localhost" >> ~/.zshrc
```

ROS Test

```
roscore
```

5.1.7 Launch

Launcher

- launch os a tool for launchine multiple nodes (as well as setting parameters)
- Are written in XM as `*.launch` files
- If not yet running, launch atuomatically stars a roscore

Browse to the folder and start a launch file with

```
roslaunch <file_name>.launch
```

Start a launch file from a package with

```
roslaunch <package_name> <file_name>.launch
```

Launch file

Listing 1: talker_listerner.launch

```
<launch>
<node name="listener" pkg="roscpp_tutorials" type="listener" output="screen"/>
<node name="talker" pkg="roscpp_tutorials" type="talker" output="screen"/>
</launch>
```

launch: Root element of the launch file

- **node**: Each `<node>` tag specifies a node to be launched
- **name**: Name of the node (free to choose)
- **pkg**: Package containing the node
- **type**: Type of the node, there must be a corresponding executable with the same name
- **output**: Specifies where to output log messages (screen: console, log: log file)

More Info

- http://wiki.ros.org/roslaunch/XML
- http://wiki.ros.org/roslaunch/Tutorials/Roslaunch%20tips%20for%20larger%20projects

Arguments

- Create re-usable launch files with `<arg>` tag, which works like a parameter (default optional)

  ```
  <arg name="arg_name" default="default_value"/>
  ```

- Use arguments in launch file with

  ```
  $(arg arg_name)
  ```

- When launching, arguments can be set with

  ```
  roslaunch launchf_file.launch arg_name:value
  ```

Example:

Listing 2: range_world.launch

```
<?xml version="1.0"?>
  <launch>
    <arg name="use_sim_time" default="true"/>
    <arg name="world" default="gazebo_ros_range"/>
    <arg name="debug" default="false"/>
    <arg name="physics" default="ode"/>

    <group if="$(arg use_sim_time)">
      <param name="/use_sim_time" value="true" />
    </group>
```

(continues on next page)

(continued from previous page)

```
    <include file="$(find gazebo_ros) /launch/empty_world.launch">
    <arg name="world_name" value="$(find gazebo_plugins)/ test/test_worlds/$(arg␣
↪world).world"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="physics" value="$(arg physics)"/>
  </include>
</launch>
```

More info http://wiki.ros.org/roslaunch/XML/arg

Including other launch files

- Include other launch files with `<include>` tag to organize large projects

```
<include file="package_name" />
```

- Find the system path to other packages with

```
$(find package_name)
```

- Pass arguments to the included file

```
<arg name="arg_name" value="value"/>
```

Listing 3: range_world.launch

```
<?xml version="1.0"?>
<launch>
  <arg name="use_sim_time" default="true"/>
  <arg name="world" default="gazebo_ros_range"/>
  <arg name="debug" default="false"/>
  <arg name="physics" default="ode"/>

    <group if="$(arg use_sim_time)">
    <param name="/use_sim_time" value="true" />
  </group>

    <include file="$(find gazebo_ros) /launch/empty_world.launch">
    <arg name="world_name" value="$(find gazebo_plugins)/test/test_worlds/
↪$(arg world).world"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="physics" value="$(arg physics)"/>
  </include>
</launch>
```

More info: http://wiki.ros.org/roslaunch/XML/include

Create a launcher in a new package

1. move to the folder of the package
2. run : `mkdir launch && cd launch`
3. run : `gedit <LAUNCHER_NAME>.launch`
4. fill the launcher file, for example:

```
<launch>
  <node pkg="<PACKAGE1_NAME>" type="<NODE1_NAME>" name="<INSTANCE0>"/>
  <node pkg="<PACKAGE2_NAME>" type="<NODE2_NAME>" name="<INSTANCE1>"/>
  <node pkg="<PACKAGE2_NAME>" type="<NODE2_NAME>" name="<INSTANCE2>"/>
  <node pkg="<PACKAGE2_NAME>" type="<NODE2_NAME>" name="<INSTANCE3>"/>
</launch>
```

Include another launcher inside this launcher

Add the include directive :

```
<launch>
  <include file="$(find <PKG_NAME>)/launch/<LAUNCHER_NAME>.launch" />
  </launch>
```

This is very useful to combine launcher together, or complete a first launcher :

- the first launcher is responsible to launch a driver
- the second launcher that includes the first one launches also a graphical tool on top of that

The advantage being that it is not necessary to copy paste all the code of the first launcher into the second one to use them together.

Parameters in launcher

Parameters can be set in the launcher and get by the node at run time. This is a convenient way to avoid rebuilding the code each time it is necessary to change the value of a variable, for example a path to a file.

The syntax is the following one :

```
<param name="<PARAM_NAME>" type="<TYPE>" value="<VALUE>" />
```

Get the value of a parameter at run time

It can be used in the node at run time with this C++ code :

```
ros::NodeHandle nh;
std::string iniPath;
nh.getParam("ini_path", iniPath);
```

The node handler gets the parameter called *ini_path* in the launcher and will store it in the variable *iniPath*. If the parameter is public, therefore accessible by all the nodes, this is sufficient to get its value. If the parameter is private to a node, then the node handler needs to know the name of the node :

```
ros::NodeHandle nh;
std::string iniName;
nh.getParam("tof_driver_1/ini_name", iniName);
```

To get the name of the node at run time, it is possible to use this line :

```
std::string nodeName = ros::this_node::getName();
```

Public vs Private parameters

Depending of where the parameter is declared in the launcher, the parameter will be either private to a node, or accessible by all the nodes. If the parameter is declared outside of a <node></node> tag, it is public and accessible to all the nodes. At the opposite, if the parameter is declared inside a <node></node> tag, it will only be accessible by the node, with the specific method described above.

In this example :

- The parameter *ini_path* is public and accessible by all the nodes only with its name.
- The parameter *ini_name* is private to each node and is accessible with the name of the node and its name, concatenated together. This allows to declare two time the same parameter with different value, as long as they are declared inside different nodes.

```
<launch>
      <!-- Public parameters for both nodes -->
      <param name="ini_path" type="str"
             value="$(find ros_driver_for_multiple_tof_sensors)/launch/" />

      <!-- Call the driver node for sensor 1 (IP = 192.168.0.105)-->
      <node pkg="ros_driver_for_multiple_tof_sensors"
            type="ros_driver_multiple_sensors_node" name="tof_driver_1"
            args="" required="true" output="screen" >

            <!-- Private parameter for node 1 -->
            <param name="ini_name" type="str" value="tof_sensor1.ini" />
      </node>

      <!-- Call the driver node for sensor 2 (IP = 192.168.1.105)-->
      <node pkg="ros_driver_for_multiple_tof_sensors"
            type="ros_driver_multiple_sensors_node" name="tof_driver_2"
            args="" required="true" output="screen" >

            <!-- Private parameter for node 2 -->
            <param name="ini_name" type="str" value="tof_sensor2.ini" />
      </node>
</launch>
```

Rviz configuration

After setting up the display configuration in Rviz, you can save it with the tab `File -> Save config as -> ...`

Then you can call it directly in the launch file by adding :

```
<node pkg="rviz" type="rviz" name="rviz"
    args="-d <PATH_TO_FILE>/<CONFIG_NAME>.rviz"/>
```

This will open Rviz with the saved configuration when the *launch* file is launched.

5.1.8 Lidar Driver

Install the SDK

run in a new terminal:

```
sudo dpkg -i libtof-dev_<version_number>ubuntu16_amd64.deb
```

5.1.9 Packages

Package Structure

ROS software is organized into packages, which can contain source code, launch files, configuration files, message definitions, data, and documentation. A package can depend on other packages called *dependencies*.

```
catkin_create_pkg <package_name> {dependencies}
```

A package need two things, its source code and the message definition. It is encouraging to place message definition into a separate folder.

- `package_name`
  - |folder| `config` - parameter files (YAML)
  - `include/package_name` - C++ include headers
  - `launch` - `*.launch` files
  - `src` - Source files
  - `test` - Unit and or ROS Tests
  - `CMakeList.txt` - CMake build file
  - `package.xml` - Package information
- `package_name_msgs`
  - `action` - Action definitions
  - `msg` - Message definitions
  - `src` - Service definitions
  - `CMakeList.txt` - CMake build file
  - `package.xml` - Package information

More info

- http://wiki.ros.org/Packages

Package Files

`package.xml`

- The `package.xml` file defines the properties of the package
  - Package name
  - Version number
  - Authors
  - Dependencies on other packages

- …

Listing 4: package.xml

```
<?xml version="1.0"?>
<package format="2">
    <name>ros_package_template</name>
    <version>0.1.0</version>
    <description>A template for ROS packages.</description>
    <maintainer email="user@email.ch">Firstname Lastname</maintainer>
    <license>BSD</license>
    <url type="website">https://github.com/link/ros_</url>
    <author email="user@email.ch">Firstname Lastname</author>

    <buildtool_depend>catkin</buildtool_depend>

    <depend>roscpp</depend>
    <depend>sensor_msgs</depend>
</package>
```

More info

- http://wiki.ros.org/catkin/package.xml

CMakeLists.txt

The `CMakeLists.txt` is the input to the CMake build system

1. Required CMake Version (`cmake_minimum_required`)
2. Package Name (`project()`)
3. Find other CMake/Catkin packages needed for build (`find_package()`)
4. Message/Service/Action Generators (`add_message_files()`, `add_service_files()`, `add_action_files()`)
5. Invoke message/service/action generation (`generate_messages()`)
6. Specify package build info export (`catkin_package()`)
7. Libraries/Executables to build (`add_library()/add_executable()/target_link_libraries()`)
8. Tests to build (`catkin_add_gtest()`)
9. Install rules (`install()`)

Listing 5: CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(husky_highlevel_controller)
add_definitions(--std=c++11)

find_package(catkin REQUIRED COMPONENTS roscpp sensor_msgs )

catkin_package(
  INCLUDE_DIRS include
  # LIBRARIES
  CATKIN_DEPENDS roscpp  sensor_msgs
  # DEPENDS
)
```

(continues on next page)

(continued from previous page)

```
include_directories(include ${catkin_INCLUDE_DIRS})

add_executable(${PROJECT_NAME} src/${PROJECT_NAME}_node.cpp src/
↪HuskyHighlevelController.cpp)

target_link_libraries(${PROJECT_NAME} ${catkin_LIBRARIES})
```

More info

- http://wiki.ros.org/catkin/CMakeLists.txt

Eclipse integration

- Build the Eclipse project files with additional build flags

```
catkin build package_name --cmake-args -G"Eclipse CDT4 - Unix Makefiles” -D__
↪cplusplus=201103L D__GXX_EXPERIMENTAL_CXX0X__=1
```

- To use flags by default in your catkin environment, use the *catkin config* command.
- The Eclipse project files will be generated in *~/catkin_ws/build*

C++ Client Library

- http://wiki.ros.org/roscpp
- http://wiki.ros.org/roscpp/Overview

Example

```
#include <ros/ros.h>

int main(int argc, char** argv)                   // ROS main head file
{
    ros::init(argc, argv, "hello_world");         // has to be called before ROS␣
↪func's
    ros::NodeHandle nodeHandle;                   // access poiunt for␣
↪communication
    ros::Rate loopRate(10);                       // ros:Rate runs loops at␣
↪desired freq e.g. 10 = 10 Hz

    unsigned int count = 0;
    while (ros::ok()) {                           // checks if a node should␣
↪continue running
        ROS_INFO_STREAM("Hello World " << count); // ROS_info() logs messages from␣
↪fs
        ros::spinOnce();                          // processes incommind msg via␣
↪callbacks
        loopRate.sleep();
        count++;
    }
    return 0;
}
```

Node Handle

http://wiki.ros.org/roscpp/Overview/NodeHandles

```
// Default (public) node handle:      // Recommended
nh_ = ros::NodeHandle();              // /namespace/topic

// Private node handle:               // Recommended
nh_private_ = ros::NodeHandle("~");   // /namespace/node/topic

// Namespaced node handle:
nh_eth_ = ros::NodeHandle("hevs");    // /namespace/hevs/topic

// Global node handle:                // NOT Recommended
nh_global_ = ros::NodeHandle("/");    // /topic
```

Logging ROS_INFO

- http://wiki.ros.org/rosconsole
- http://wiki.ros.org/roscpp/Overview/Logging

Send text to log files and console. Instead of `std::cout`, use e.g. `ROS_INFO`.

Severity Levels

	Debug	Info	Warn	Error	Fatal
stdout	✓	✓			
stderr			✓	✓	✓
Log file	✓	✓	✓	✓	✓
/rosout	✓	✓	✓	✓	✓

Formatting Style

```
ROS_INFO("Result: %d", result);           // printf style
ROS_INFO_STREAM("Result: " << result);    // stream style
```

Launchfile

To see the output in the console set configuration to *screen* in the launch file.

```
<launch>
    <node name="listener" more="stuff" output="screen"/>
</launch>
```

Subscriber

http://wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers

Start listening to a topic by calling the method subscribe() of the node handle

```
ros::Subscriber subscriber = nodeHandle.subscribe(topic, queue_size, callback_
↪function);
```

Example

Listing 6: listener.cpp

```
#include "ros/ros.h"
#include "std_msgs/String.h"

// callback function when a message is received
void chatterCallback(const std_msgs::String& msg) {
     ROS_INFO("I heard: [%s]", msg.data.c_str());
}

int main(int argc, char **argv) {
     ros::init(argc, argv, "listener");
    ros::NodeHandle nodeHandle;
    // Subscript to topic with a queue size of 10 (1-10 is recommended)
    ros::Subscriber subscriber = nodeHandle.subscribe("chatter",10,
↪chatterCallback);
    ros::spin(); // stay's here forever
    return 0;
}
```

Publisher

http://wiki.ros.org/roscpp/Overview/Publishers%20and%20Subscribers

Create a publisher with help of the node handle

```
ros::Publisher publisher = nodeHandle.advertise<message_type>(topic, queue_size);
```

Example

```
:caption: talker.cpp
#include <ros/ros.h>
#include <std_msgs/String.h>

int main(int argc, char **argv) {
    ros::init(argc, argv, "talker");
    ros::NodeHandle nh;
    // Node handle queue size of 1
    ros::Publisher chatterPublisher = nh.advertise<std_msgs::String>("chatter", 1);
    ros::Rate loopRate(10);

    unsigned int count = 0;
    while (ros::ok()) {
        std_msgs::String message;
        // Cretae message content
        message.data = "hello world " + std::to_string(count);
        ROS_INFO_STREAM(message.data);
        chatterPublisher.publish(message);
        ros::spinOnce();
```

(continues on next page)

(continued from previous page)

```
        loopRate.sleep();
        count++;
    }
    return 0;
}
```

OOP

http://wiki.ros.org/roscpp_tutorials/Tutorials/UsingClassMethodsAsCallbacks

Example

```
:caption: my_package_node.cpp
#include <ros/ros.h>
#include "my_package/MyPackage.hpp"
int main(int argc, char** argv) {
    ros::init(argc, argv, "my_package");
    ros::NodeHandle nodeHandle("~");
    // Call
    my_package::MyPackage myPackage(nodeHandle);

    ros::spin();
    return 0;
}
```

class MyPackage	class Algorithm
Main node class providing ROS interface (subscribers, parameters, timers etc.)	Class implementing the algorithmic part of the node **Note: The algorithmic part of the code could be separated in a (ROS-independent) library**

Parameter Server

http://wiki.ros.org/roscpp/Overview/Parameter%20Server

Example Parameter File

```
:caption: config.yaml

camera:
     left:
             name: left_camera
             exposure: 1
     right:
             name: right_camera
             exposure: 1.1
```

Example Launch file

```
<launch>
    <node name="name" pkg="package" type="node_type">
        <rosparam command="load" file="$(find package)/config/config.yaml" />
    </node>
</launch>
```

C++ API

```
ros::NodeHandle nodeHandle("~");
std::string topic;
if (!nodeHandle.getParam("topic", topic)) {
    ROS_ERROR("Could not find topic parameter!");
}
```

Get a parameter in C++ with

```
nodeHandle.getParam(parameter_name, variable)
```

- Method returns `true` if parameter was found, `false` otherwise
- Global and relative parameter access:
  - Global parameter name with preceding /

    ```
    nodeHandle.getParam("/package/camera/left/exposure", variable)
    ```

    Relative parameter name (relative to the node handle)

    ```
    nodeHandle.getParam("camera/left/exposure", variable)
    ```

- For parameters, typically use the private node handle

  ```
  ros::NodeHandle("~")
  ```

5.1.10 External Packages and Nodes

Terminology

- + : interesting topics and hardware abstraction
- ~ : interesting, but quite a lot of work to do for hardware compatibility or mapping
- - : bad solution

Overview

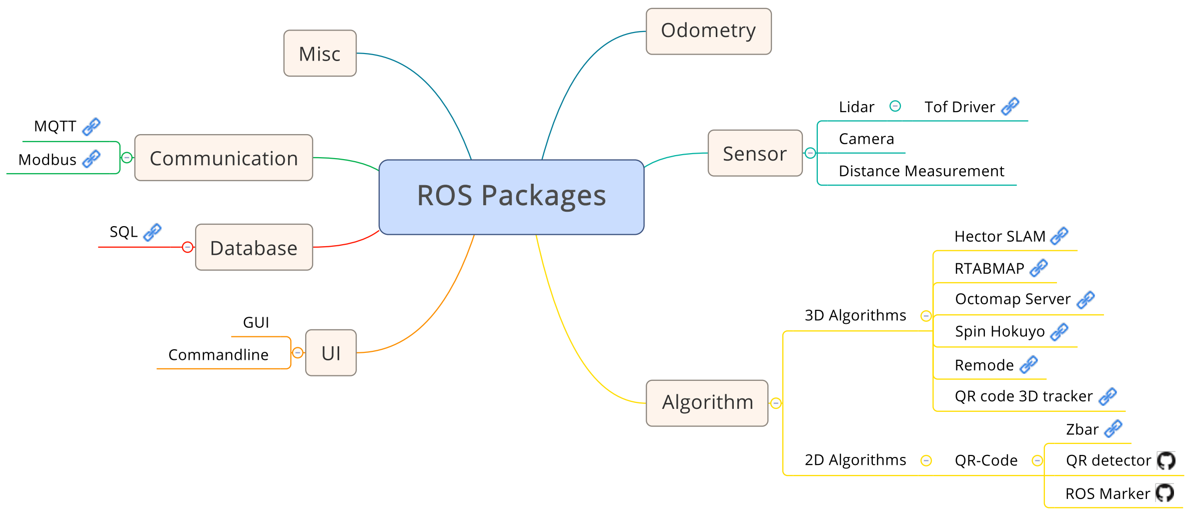


Fig. 4: ROS Packages Overview

3D Mapping

SLAM

Octomap_server : +

3D occupancy grid mapping, independent from sensor, looks like it does not need odometry

- https://youtu.be/yp0f8-AKvDU
- https://wiki.ros.org/octomap_mapping
- https://wiki.ros.org/octomap_server
- http://octomap.github.io/

Plus :

- maintained
- compatible with melodic
- documentation available as well as many
- no odometry
- independent from hardware (only require the right input topics)

Minus :

- …

Inputs required :

- sensor_msgs/PointCloud2

Hector slam : +

- https://github.com/tu-darmstadt-ros-pkg/hector_slam
- http://wiki.ros.org/hector_slam

Not sure whether we're interested in hector slam itself, or on the

Plus :

- maintained
- not directly compatible with melodic, but easy to build it from source for melodic
- odometry not needed

Minus :

- mostly created for 2D mapping and robot navigation
- not much documentation

Inputs required :

- …

REMODE : ~

- https://www.ros.org/news/2016/02/open-source-release-remode-probabilistic-monocular-dense-rec
  html

modeling of many 3D objects, like rooms, persons, …

Plus :

- noise reduction
- nice rendering

Minus :

- not much documentation and precisions about `hardware/drivers/topics`
- maybe "too much" for our needs ?
- looks like it is not maintained anymore : latest commit was 4 years ago

Inputs :

- …

LOAM

RTABMAP : +

- http://wiki.ros.org/rtabmap_ros

Plus :

- maintained
- compatible with melodic

- real time mapping
- publishes :
  - 3D point clouds
  - 2D occupancy maps
- tutorials and documentation available

Minus :

- oriented towards robot navigation, although “top-down” modeling seems to be possible

Inputs required :

- odometry (not mandatory in all cases)
- scan 2D or 3D

Spin Hokuyo : +

- https://github.com/RobustFieldAutonomyLab/spin_hokuyo
- http://wiki.ros.org/spin_hokuyo

It creates a point cloud with a 2D LiDaR and a servomotor. The interesting node compiles small point clouds to make one big point cloud. Could be very useful to make our digital model.

Plus :

- has a node that compiles point clouds and publish them on a topic
- great rendering

Minus :

- designed for another sensor, but the node that compiles point clouds does not care about that
- need some odometry work

Inputs required :

- laser scan
- odometry

Lego-LOAM : ~

- https://github.com/RobustFieldAutonomyLab/LeGO-LOAM

Plus :

- good rendering

Minus :

- designed for robot navigation, not for “top-down mapping”
- designed for another sensor (velodyne)

Inputs :

- …

Velodyne loam : ~

- http://wiki.ros.org/loam_velodyne

Plus :

- good rendering
- builds 3D maps

Minus :

- for velodyne sensor
- robot navigation

Inputs :

- …

Bad solution -

- https://github.com/koide3/hdl_graph_slam : not what we need. creates maps with corridors and doors, but not "top-down" mapping
- http://wiki.ros.org/robot_pose_ekf : not what we need
- http://wiki.ros.org/ethzasl_icp_mapper : doc not up to date, slowly not maintained anymore, …
- https://github.com/ethz-asl/libpointmatcher/blob/master/doc/index.md

Modbus

- http://wiki.ros.org/modbus

Object Tracking

Multiple objects lidar tracking : ~

- https://github.com/praveen-palanisamy/multiple-object-tracking-lidar

Plus :

- tracks objects in real time
- hardware independent

Minus :

- 2D maps, most likely used for robot navigation

Inputs :

- …

Object Detection

- https://www.acin.tuwien.ac.at/vision-for-robotics/software-tools/v4r-library/
- https://rgit.acin.tuwien.ac.at/v4r/v4r_ros_wrappers
- http://wiki.ros.org/object_recognition
- https://www.osrfoundation.org/ros2-object-detection-demo/
- http://wiki.ros.org/find_object_2d

QR code readers

- http://wiki.ros.org/zbar_ros
- https://github.com/mdrwiega/qr_detector
- http://wiki.ros.org/visp_auto_tracker

5.1.11 RViz

Overview

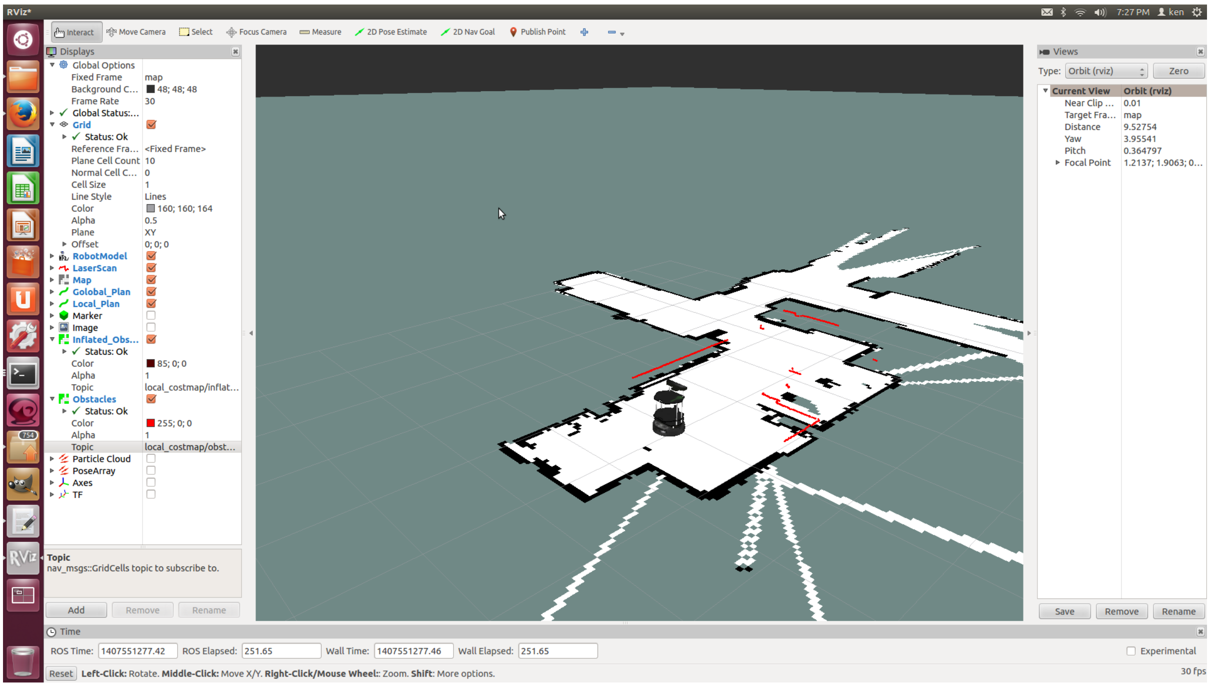
http://wiki.ros.org/rviz

- 3D visualization tool for ROS
- Subscribes to topics and visualizes the message contents
- Different camera views (orthographic, top-down, etc.)
- Interactive tools to publish user information
- Save and load setup as RViz configuration
- Extensible with plugins

Run

```
rosrun rviz rviz
```

Save configuration with `ctrl+s`



Displays	
Global Options	
Fixed Frame	map
Background C…	48; 48; 48
Frame Rate	30
✓ Global Status:…	
Grid	☑
✓ Status: Ok	
Reference Fra…	<Fixed Frame>
Plane Cell Count	10
Normal Cell C…	0
Cell Size	1
Line Style	Lines
Color	160; 160; 164
Alpha	0.5
Plane	XY
Offset	0; 0; 0
RobotModel	☑
LaserScan	☑
Map	☑
Golobal_Plan	☑
Local_Plan	☑
Marker	☐
Image	☐
Inflated_Obs…	☑
✓ Status: Ok	
Color	85; 0; 0
Alpha	1
Topic	local_costmap/inflat…
Obstacles	☑
✓ Status: Ok	
Color	255; 0; 0
Alpha	1
Topic	local_costmap/obst…
Particle Cloud	☐
PoseArray	☐
Axes	☐
TF	☐

Topic
nav_msgs::GridCells topic to subscribe to.

Type: Orbit (rviz)

Current View	Orbit (rviz)
Near Clip …	0.01
Target Fra…	map
Distance	9.52754
Yaw	3.95541
Pitch	0.364797
Focal Point	1.2137; 1.9063; 0…

Time

ROS Time: 1407551277.42 ROS Elapsed: 251.65 Wall Time: 1407551277.46 Wall Elapsed: 251.65 ☐ Experimental

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift**: More options.

30 fps

Built-In Display Types

Name	Description	Messages Used
Axes	Displays a set of Axes	
Effort	Shows the effort being put into each revolute joint of a robot.	sensor_msgs/JointStates
Camera	Creates a new rendering window from the perspective of a camera, and overlays the image on top of it.	sensor_msgs/Image, sensor_msgs/CameraInfo
Grid	Displays a 2D or 3D grid along a plane	
Grid Cells	Draws cells from a grid, usually obstacles from a costmap from the navigation stack.	nav_msgs/GridCells
Image	Creates a new rendering window with an Image. Unlike the Camera display, this display does not use a CameraInfo. *Version: Diamondback+*	sensor_msgs/Image
Interactive-Marker	Displays 3D objects from one or multiple Interactive Marker servers and allows mouse interaction with them. *Version: Electric+*	visualization_msgs/InteractiveMarker
Laser Scan	Shows data from a laser scan, with different options for rendering modes, accumulation, etc.	sensor_msgs/LaserScan
Map	Displays a map on the ground plane.	nav_msgs/OccupancyGrid
Markers	Allows programmers to display arbitrary primitive shapes through a topic	visualization_msgs/Marker, visualization_msgs/MarkerArray
Path	Shows a path from the navigation stack.	nav_msgs/Path
Point	Draws a point as a small sphere.	geometry_msgs/PointStamped
Pose	Draws a pose as either an arrow or axes.	geometry_msgs/PoseStamped
Pose Array	Draws a “cloud” of arrows, one for each pose in a pose array	geometry_msgs/PoseArray
Point Cloud(2)	Shows data from a point cloud, with different options for rendering modes, accumulation, etc.	sensor_msgs/PointCloud, sensor_msgs/PointCloud2
Polygon	Draws the outline of a polygon as lines.	geometry_msgs/Polygon
Odometry	Accumulates odometry poses from over time.	nav_msgs/Odometry
Range	Displays cones representing range measurements from sonar or IR range sensors. *Version: Electric+*	sensor_msgs/Range
Robot-Model	Shows a visual representation of a robot in the correct pose (as defined by the current TF transforms).	
TF	Displays the ros wiki tf transform hierarchy.	
Wrench	Draws a wrench as arrow (force) and arrow + circle (torque)	geometry_msgs/WrenchStamped
Oculus	Renders the RViz scene to an Oculus headset	

5.1.12 Transform Frames

A frame in the ROS language is a specific coordinate system in the space. ROS abstracts elements of a robot as coordinates frames. Each physical part of a robot that has a particular meaning will most likely have its own frame :

- a sensor : *laser_frame*
- an arm : *left_arm_frame*

It is up to the programmer to create frames where it is necessary, but some frames are already defined by ROS (see below).

Each frame has it own origin and coordinate system :

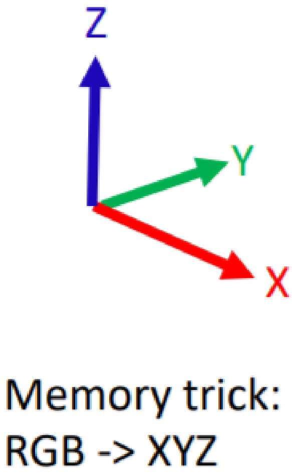


Fig. 5: coordinate frame axis

To keep trace of the frames in the whole coordinate system, they must all refer to a main frame. Knowing the position of the main frame and the relative positions of all the other frames, ROS is able to know the exact position of each frame all continuously.

The TF2 package tracks the coordinate frames. There are several predefined frames :

- *world* : kind of the parent of all the frames, does not move, there is only one single *world*
- *map* : child of *world*, can be freely fixed in the world frame, does not move compared to the *world*, but it can be several *map* frames in a *world* (usually one *map* per robot)
- *odom* : child of *map*, fixed at the start point of the robot in the *map* frame, does not move compared to *world* and *map*
- *base_link* : kind of the reference frame of a robot, it is moving in *odom*, therefore moving in *map* and *world*
- …

The TF tree shows the relations between the frames :

One can create coordinate frames for each part of the robot that needs to be tracked, for example :

- *scanner_frame* : position of the scanner on a robot, somehow linked to the *base_link*
- *wheels_frame* : position of the wheels on a robot, somehow linked to the *base_link*

The links between the *base_link* and the other frames can be direct, or they can be relative to it via other frames.

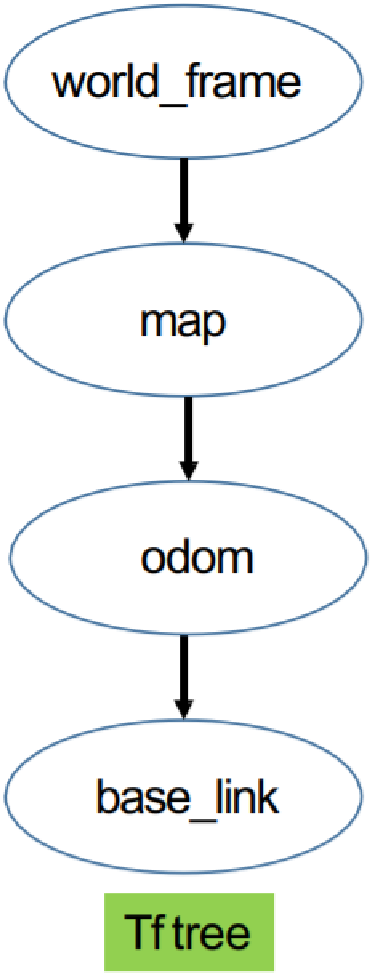


Fig. 6: tf tree

They are linked together by TF (transform frames). TF can be either static, which means that the relation between two frames will never change (for example two sensors being fixed 1 meter away), or dynamic when the relation evolves in the time (for example the arm of a robot compared to its head).

Let us use the example our two LIDAR sensor : they are oriented in the same way, they are on the same table, the only difference being there is 2.15 meter between them. For this example, they will never move nor rotate. We can use the node *static_transform_publisher* to inform other nodes that will use their data of their relative position. We will also fix them in the *world*, *map* and *base_link* frame.

Since the *base_link* frame will not move neither, it will also be fixed to the *map* by a static transform. The static transformations are called as a node from a launcher :

```
<launch>
      <!-- From world to map, same origin -->
      <node pkg="tf2_ros" type="static_transform_publisher" name="world_to_map"
            args="0 0 0 0 0 0 /world /map" />

      <!-- From map to base_link, fixed in this case -->
      <node pkg="tf2_ros" type="static_transform_publisher" name="map_to_base_link"
            args="0 0 0 0 0 0 /map /base_link" />

      <!-- From base_link to laser_frame1, position of the first lidar -->
      <node pkg="tf2_ros" type="static_transform_publisher" name="base_link_to_
↪lidar1"
            args="0 0 0 0 0 0 /base_link /lidar1_frame" />

      <!-- From base_link to laser_frame2, position of the second lidar -->
      <node pkg="tf2_ros" type="static_transform_publisher" name="base_link_to_
↪lidar2"
            args="-2.15 -0.01 0 0 0 0 /base_link /lidar2_frame" />
</launch>
```

Which will produce the following TF tree :

The arguments are :

- translations in X, Y, Z
- rotations around X, Y, Z
- parent *frame_id*
- child *frame_id*

Each topic has a reference frame. This means that each message being published on a topic kind of contains the position "from where it comes". This is the *frame_id* parameter. The node that will published the data of the LIDAR shall publish them with the right *frame_id*, otherwise the TF tree will not be able to link all the TF together.

Documentation about frames and transformations can be found there :

- tf2

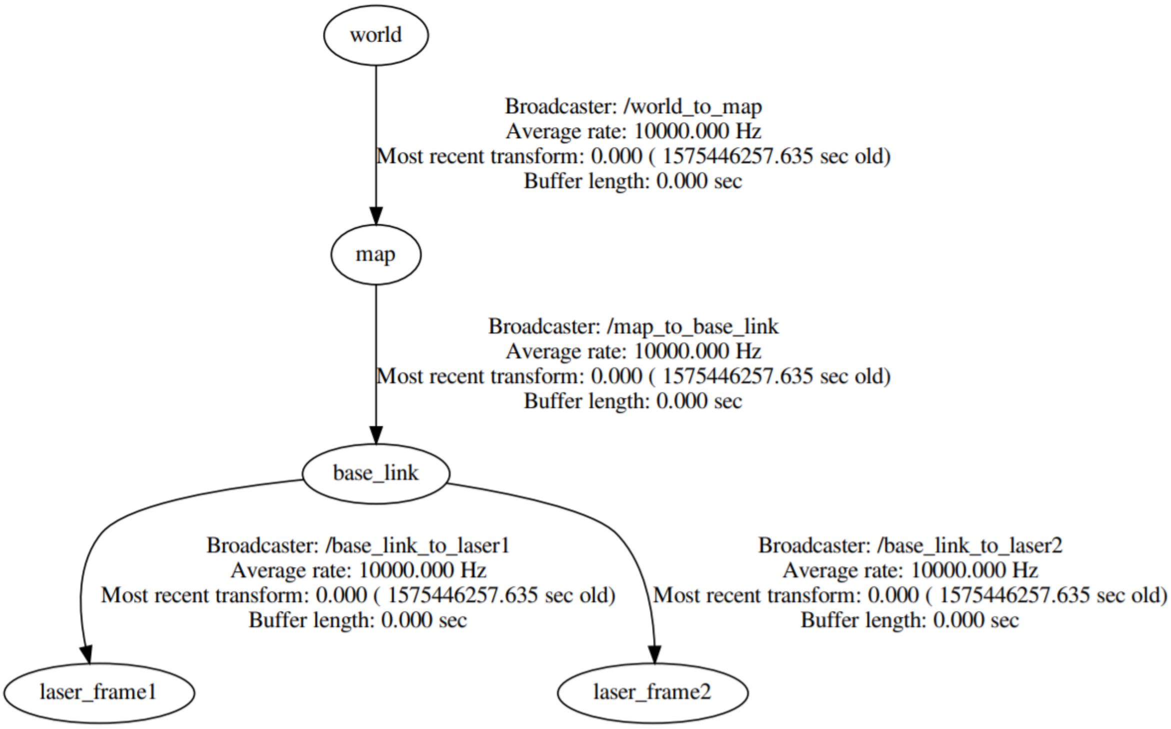

world
Broadcaster: /world_to_map
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1575446257.635 sec old)
Buffer length: 0.000 sec
map
Broadcaster: /map_to_base_link
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1575446257.635 sec old)
Buffer length: 0.000 sec
base_link
Broadcaster: /base_link_to_laser1
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1575446257.635 sec old)
Buffer length: 0.000 sec
Broadcaster: /base_link_to_laser2
Average rate: 10000.000 Hz
Most recent transform: 0.000 (1575446257.635 sec old)
Buffer length: 0.000 sec
laser_frame1
laser_frame2

Fig. 7: lidar tf tree

6 Writing

6.1 LaTeX

6.1.1 Introduction

- *Some LaTeX helppages*
- *Generate PDF files*

Some LaTeX helppages

- HEI SPL Latex Templates
- Cheatsheet A Guide to Latex
- Tex Stackexchange Forum

Generate PDF files

Latex is best suited to insert images as pdf. In order to convert images or svg into pdf use inkscape Convert `*.svg` images with inkscape to `*.pdf` and `*.pdf_tex`

```
inkscape -D -z --file=image.svg --export-pdf=image.pdf --export-latex
```

6.1.2 Installation LaTeX

Base Install

Linux

Package	Archives	Disk Space
texlive-latex-base	59 MB	216 MB
texlive-latex-recommended	74 MB	248 MB
texlive-pictures	83 MB	277 MB
texlive-fonts-recommended	83 MB	281 MB
texlive	98 MB	314 MB
texlive-plain-generic	82 MB	261 MB
texlive-latex-extra	144 MB	452 MB
texlive-full	2804 MB	5358 MB

see also Tex Stack Exchange

```
                      texlive-latex-extra
                         |   |   :   :
       +-----------------+   |   :   :..................
       |                     |   :                     :
       v                     |   :                     v
texlive-pictures             |   :           texlive-plain-generic
       |                     |   :
       |    +----------------+   :...............
       |    |                                   :
```

(continues on next page)

(continued from previous page)

```
       |    |              texlive              :
       |    |               | | |               :
       |    |    +----------+ | +----------+    :
       |    |    |            |            |    :
       v    v    v            |            v    v
texlive-latex-recommended     |  texlive-fonts-recommended
                 |            |            :
                 |            |            v
                 |            |           tipa
                 |            |            |
                 +----------+ | +----------+
                            | | |
                            v v v
                     texlive-latex-base
```

```
sudo apt-get install texlive-latex-extra
```

Windows

- Install MikTeX - https://miktex.org/download
- MikTeX Packages
  - minted

    ```
    pip install pygments
    ```

    add Python Scripts to PATH Environment Variable. `%USERPROFILE%\AppData\Local\Continuum\anaconda3\Scripts\`
- Install TeXstudio
  - https://texstudio.org
  - Options => Configure TeXstudio => Commands => add Interpreter Flag -shell-escape
  - enable line numbers
  - enable white spaces
- Install Inkscape
  - https://inkscape.org/release/

Manual Package install

For manual installing `*.sty` Packages and `*.cls` Class files.

> **Warning:** For every package create a separate folder

Manual Package Linux

- Find TEXMFHOME directory

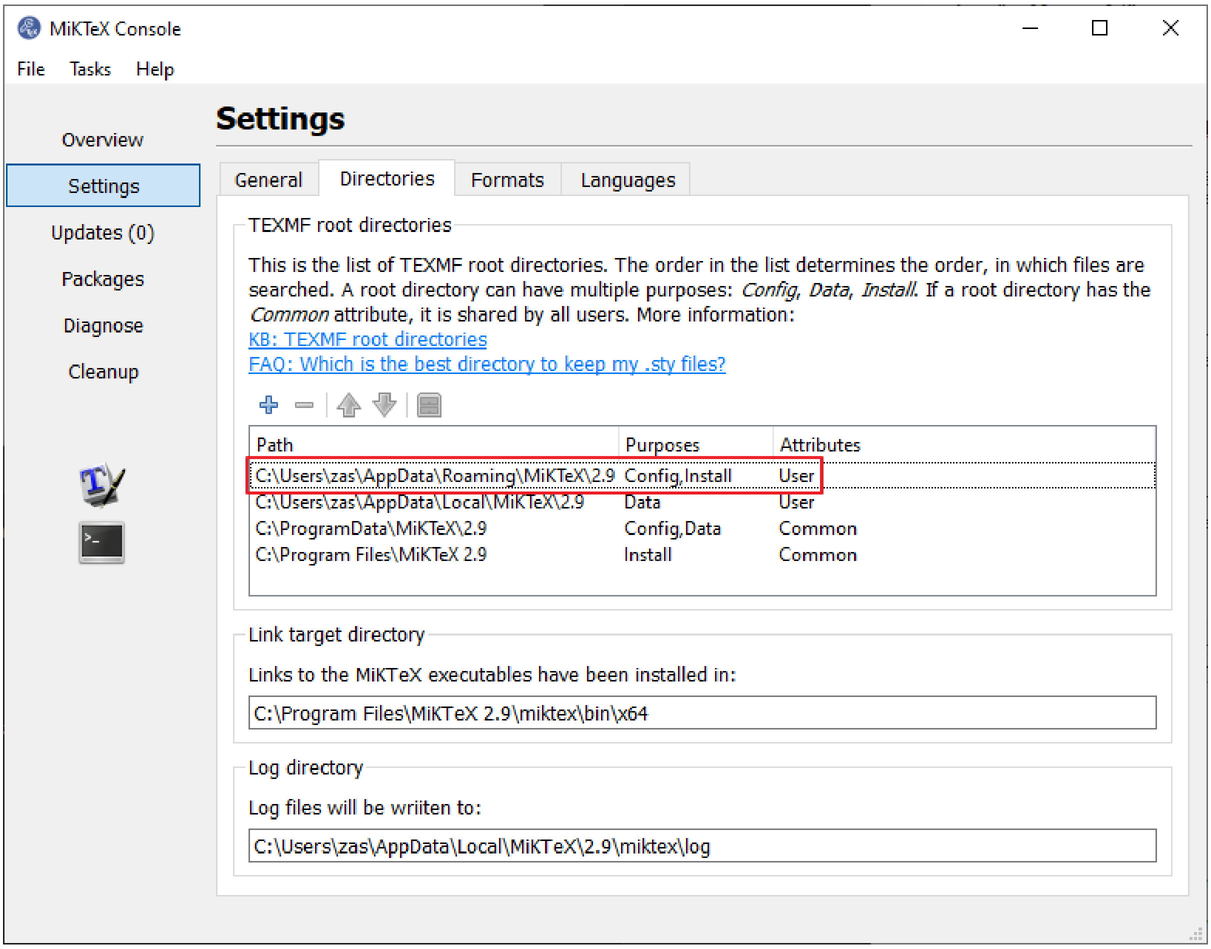
```
kpsewhich -var-value TEXMFHOME
```

- Navigate to $(TEXMFHOME)/tex/latex
- Copy paste your *.sty and *.cls
- Update Package index

```
texhash
```

Manual Package Windows

- Open MikTeX Console and go to Settings -> Directories
- The Config,Install and User folder is the location of your Packages: %USERPROFILES%/AppData/Roaming/MikTeX/2.9/
- Inside you have to navigate to tex/latex/ folder
- %USERPROFILES%/AppData/Roaming/MikTeX/2.9/tex/latex/



Settings

TEXMF root directories

This is the list of TEXMF root directories. The order in the list determines the order, in which files are searched. A root directory can have multiple purposes: *Config*, *Data*, *Install*. If a root directory has the *Common* attribute, it is shared by all users. More information:

KB: TEXMF root directories

FAQ: Which is the best directory to keep my .sty files?

Path	Purposes	Attributes
C:\Users\zas\AppData\Roaming\MiKTeX\2.9	Config,Install	User
C:\Users\zas\AppData\Local\MiKTeX\2.9	Data	User
C:\ProgramData\MiKTeX\2.9	Config,Data	Common
C:\Program Files\MiKTeX 2.9	Install	Common

Link target directory

Links to the MiKTeX executables have been installed in:

C:\Program Files\MiKTeX 2.9\miktex\bin\x64

Log directory

Log files will be wriiten to:

C:\Users\zas\AppData\Local\MiKTeX\2.9\miktex\log

- Copy paste your *.sty and *.cls
- Update Package index

```
texhash
```

6.2 ReStructuredText

6.2.1 Introduction

Some RST Syntax helppages

- `rst-cheatsheet.pdf`
- Thomas Cokelaer RST Sphinx Syntax
- Docutil Quickref
- Raslina RST Cheatsheet

6.2.2 RST and Sphinx Cheatsheet

In this page you will get a quick overview about the most used syntax.

Table of content

To include a table of content of all title in a page use

```
.. contents:: :local:
```

Titles

The lines have to be as long or longer than the text.

```
=============
Section Title
=============

Titles
======

Paragraph
---------

Sub-Pragraph
^^^^^^^^^^^^
```

Markup

emphasis	*emphasis*
strong emphasis	**strong emphasis**
\`interpreted text\`	The rendering and meaning of interpreted text is domain- or application-dependent.
\`\`inline literal\`\`	`inline literal`
:markup:	**markup**
> quote markup	> quote markup

Links

External Links

Internet

```
`python <http://www.python.org/>`_
`<http://www.python.org/>`_
http://www.python.org/
```

python

http://www.python.org/

http://www.python.org/

Other Sphinx Pages

- absolute link from root *About*
- relative link from document location *About*

```
* absolute link from root
  :doc:`/about/index`

* relative link from document location
  :doc:`../../about/index`
```

In order to link to another subheader in another document you need to use *Internal References*.

In the page to be jumped to add `.. _ref_name:`, and then you can:

```
:ref:`ref_name`
:ref:`link title<ref_name>`
```

Like so:

- *How to use Sphinx Documentation*
- *Sphinx Doc Link*

Internal Links

Link to Titles

Link to titles directly is done with the extension `sphinx.ext.autosectionlabel`.

Important: You need to add the `folder_name` and `subfolder(s)`_name` name as well as `file_name` without `.rst` extension in order to reference a section title. This avoids the duplicated label warning.

```
:ref:`Displayname␣
↪<folder_name/subfolder_name/file_name/section_title>`
```

```
:ref:`Back␣
↪to top <writing/rst/cheatsheet:RST and Sphinx Cheatsheet>`

:ref:`writing/rst/cheatsheet:Images`
```

Back to top

Images

Internal References

In any place of the document a reference point can be inserted and later refered to.

```
.. _ref-point:

see :ref:`ref-point`
```

see *Internal References*

File Links

To link to a file within the Sphinx file structure use the Role `:download:`

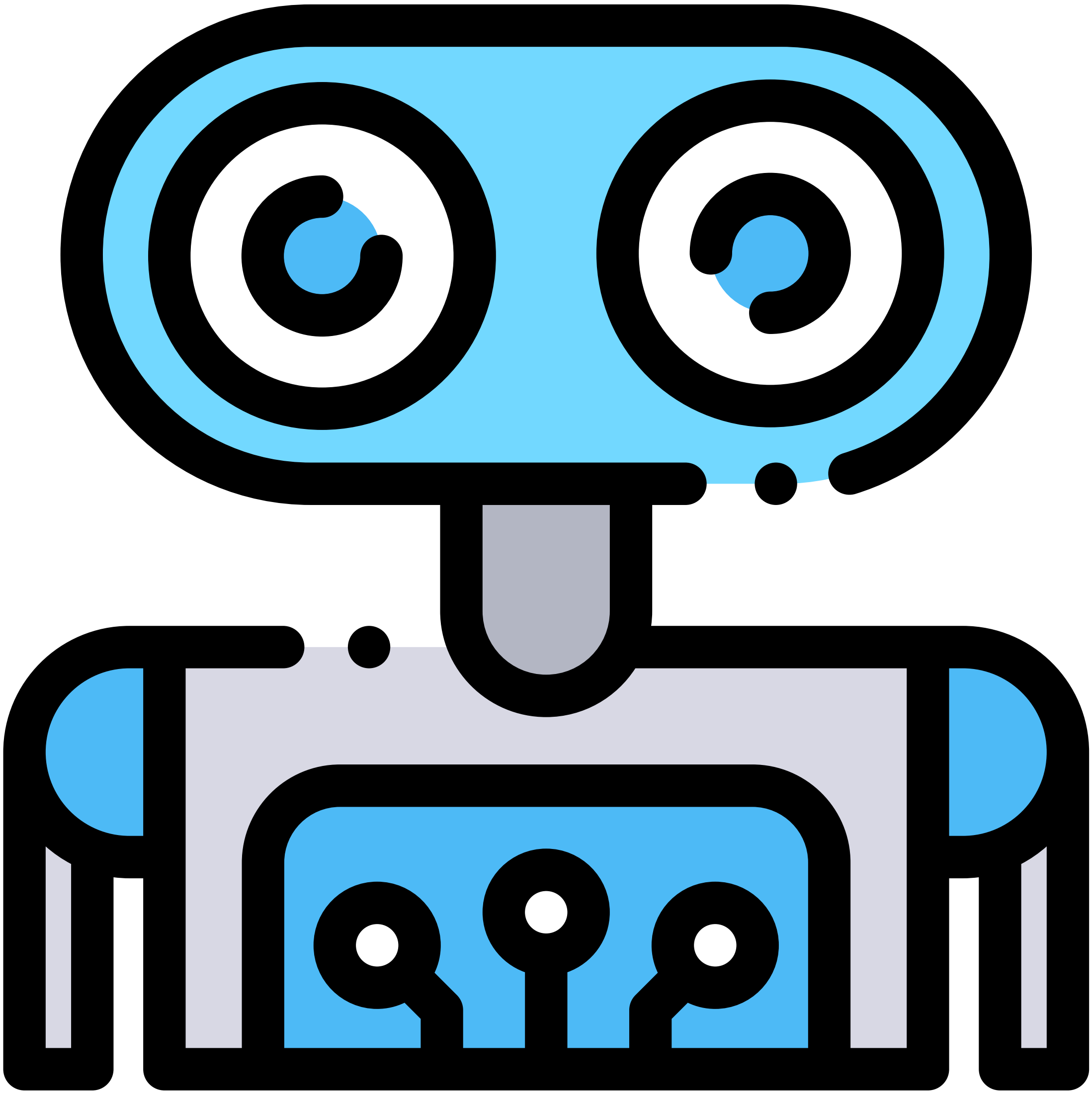
```
:download:`../../coding/
↪ros/books/Mastering_ROS_for_Robotics_Programming.pdf`

:download:`Mastering_
↪ROS_for_Robotics_Programming <../../coding/
↪ros/books/Mastering_ROS_for_Robotics_Programming.pdf>`
```

`../../coding/ros/books/Mastering_ROS_for_Robotics_Programming.pdf`

`Mastering_ROS_for_Robotics_Programming`

Images

```
.. figure:: /img/logo.*
```

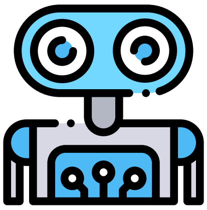


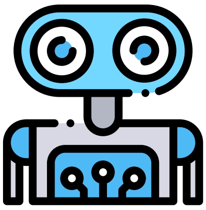
Important: Images should be either in png or svg format

Important: For `*.svg` files the file ending needs to be changed from `svg` to `*`. That way for html svg is used and pdf or pn for the latex or pdf output.

Image Placement

```
.. figure:: /img/logo.*
   :align: left
   :width: 100px

.. figure:: /img/logo.*
   :align: center
   :width: 100px

.. figure:: /img/logo.*
   :align: right
   :width: 100px

.. figure:: /img/logo.*
   :align: center
   :width: 100px
   :height: 100px
   :scale: 50 %
   :alt: this is the knowhow logo

   Caption of figure
```





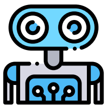
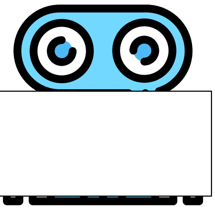


Fig. 1: Caption of figure

Inline Images



For inline images to work, a substitution needs to be made

```
.. |folder| image:: /img/icons/folder.*

After that the image |folder| can be integrated inline.
```

After that the image 📁 can be integrated inline.

Lists

- item 1
  - item 1.1
  - item 1.2
- item 2
  - item 2.1
    - item 2.1.1

1. auto enumerated list item 1
2. auto enumerated list item 1
3. auto enumerated list item 1
4. auto enumerated list item 1

3. enumerated list start with item 3
4. auto enumerated list item 4
5. auto enumerated list item 5
6. auto enumerated list item 6

Tables

```
+------------+------------+-----------+
| Header 1   | Header 2   | Header 3  |
+============+============+===========+
| body row 1 | column 2   | column 3  |
+------------+------------+-----------+
| body row 2 | Cells may span columns.|
+------------+------------+-----------+
| body row 3 | Cells may  | - Cells   |
+------------+ span rows. | - contain |
| body row 4 |            | - blocks. |
+------------+------------+-----------+
```

Header 1	Header 2	Header 3
body row 1	column 2	column 3
body row 2	Cells may span columns.	
body row 3	Cells may span rows.	• Cells • contain • blocks.
body row 4		

```
=====  =====  ======
Inputs     Output
------------  ------
  A      B    A or B
=====  =====  ======
False  False  False
True   False  True
False  True   True
True   True   True
=====  =====  ======
```

Inputs		Output
A	B	A or B
False	False	False
True	False	True
False	True	True
True	True	True

```
.. table:: Table caption

   =====  =====  ======
      Inputs     Output
   ------------  ------
     A      B    A or B
   =====  =====  ======
   False  False  False
   =====  =====  ======
```

Table 1: Table caption

Inputs		Output
A	B	A or B
False	False	False

Code

```
.. code-block:: python

   import antigravity

   def main():
       antigravity.fly()
   if __name__=='__main__':
       main()
```

```python
import antigravity

def main():
    antigravity.fly()
if __name__=='__main__':
    main()
```

```
.. code-block:: python
   :linenos:
   :caption: Code Blocks can have captions.

   import antigravity

   def main():
       antigravity.fly()
   if __name__=='__main__':
       main()
```

Listing 1: Code Blocks can have captions.

```python
import antigravity

```

(continues on next page)

(continued from previous page)

```
def main():
    antigravity.fly()
if __name__=='__main__':
    main()
```

```
.. code-block:: python
   :linenos:
   :lineno-start: 10

   import antigravity

   def main():
       antigravity.fly()
   if __name__=='__main__':
       main()
```

```
import antigravity

def main():
    antigravity.fly()
if __name__=='__main__':
    main()
```

Infoboxes

```
.. note::
   This is a Note Box
```

Note: This is a Note Box

```
.. warning::
   This is a Warning Box
```

Warning: This is a Warning Box

```
.. important::
   This is a Important Box
```

Important: This is a Important Box

```
.. seealso::
   This is a See Also Box
```

See also:

This is a See Also Box

Special Formatting

```
.. versionadded:: 2.5
   The *spam* parameter.

.. versionchanged:: 2.5
   Feature description

.. deprecated:: 3.1
   Use :func:`spam` instead.
```

New in version 2.5: The *spam* parameter.

Changed in version 2.5: Feature description

Deprecated since version 3.1: Use `spam()` instead.

Math

```
Inline math :math:`a^2 + b^2 = c^2`.
```

Inline math $a^2 + b^2 = c^2$.

```
.. math::

   f(x) &= x^2\\
   g(x) &= \frac{1}{x}\\
   F(x) &= \int^a_b \frac{1}{3}x^3
```

$$\begin{aligned} f(x) &= x^2\\ g(x) &= \frac{1}{x}\\ F(x) &= \int^a_b \frac{1}{3}x^3 \end{aligned}$$

Exclude

In order to exclude some parts for a certain output use the `.. only:: output` directive.

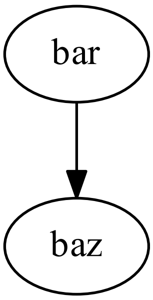
```
.. only:: html
.. only:: draft
.. only:: latex
.. only:: html or draft or latex
.. only:: html and draft
```

Important: This is needed for the all the *Wavedrom* code

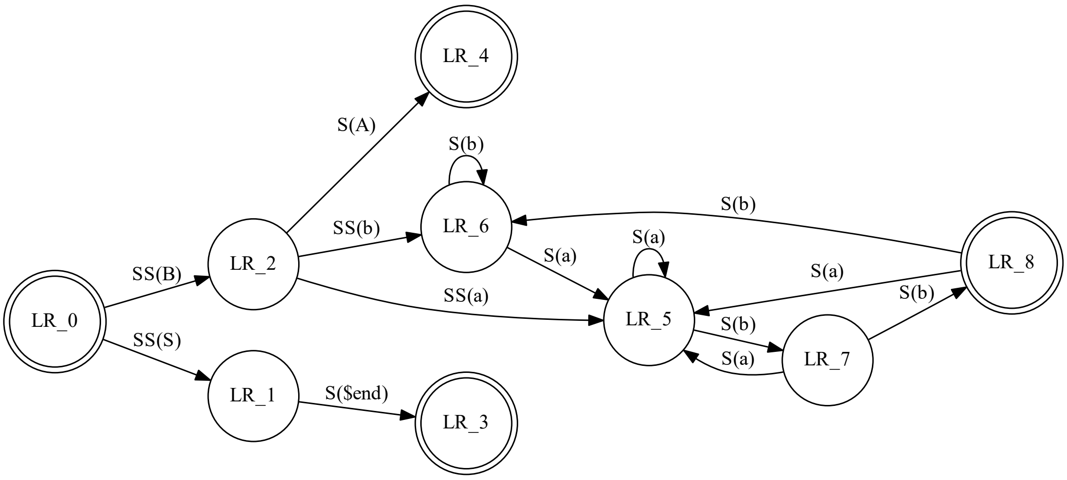
GraphViz

Get more samples herer: https://graphviz.gitlab.io/gallery/

```
.. graphviz::

   digraph foo {
      "bar" -> "baz";
   }
```



```
.. graphviz::

   digraph finite_state_machine {
     rankdir=LR;
     size="8,5"
     node [shape = doublecircle]; LR_0 LR_3 LR_4 LR_8;
     node [shape = circle];
     LR_0 -> LR_2 [ label = "SS(B)" ];
     LR_0 -> LR_1 [ label = "SS(S)" ];
     LR_1 -> LR_3 [ label = "S($end)" ];
     LR_2 -> LR_6 [ label = "SS(b)" ];
     LR_2 -> LR_5 [ label = "SS(a)" ];
     LR_2 -> LR_4 [ label = "S(A)" ];
     LR_5 -> LR_7 [ label = "S(b)" ];
     LR_5 -> LR_5 [ label = "S(a)" ];
     LR_6 -> LR_6 [ label = "S(b)" ];
     LR_6 -> LR_5 [ label = "S(a)" ];
     LR_7 -> LR_8 [ label = "S(b)" ];
     LR_7 -> LR_5 [ label = "S(a)" ];
     LR_8 -> LR_6 [ label = "S(b)" ];
     LR_8 -> LR_5 [ label = "S(a)" ];
   }
```



Wavedrom

For more information see:

- Wavedrom JSON Wiki
- Wavedrom Tutorial

Timing Diagrams

This documentation makes use of the `sphinxcontrib-wavedrom` plugin, So you can specify a timing diagram, or a register description with the `WaveJSON` syntax like so:

```
.. wavedrom::

   { "signal": [
     { "name": "pclk", "wave": 'p.......' },
     { "name": "Pclk", "wave": 'P.......' },
     { "name": "nclk", "wave": 'n.......' },
     { "name": "Nclk", "wave": 'N.......' },
     {},
     { "name": 'clk0', "wave": 'phnlPHNL' },
     { "name": 'clk1', "wave": 'xhlhLHl.' },
     { "name": 'clk2', "wave": 'hpHplnLn' },
     { "name": 'clk3', "wave": 'nhNhplPl' },
     { "name": 'clk4', "wave": 'xlh.L.Hx' },
   ]}
```

and you get:

Note: if you want the Wavedrom diagram to be present in the pdf export, you need to use the “non relaxed” JSON dialect. long story short, no javascript code and use `"` arround key value (Eg. `"name"`).

Register

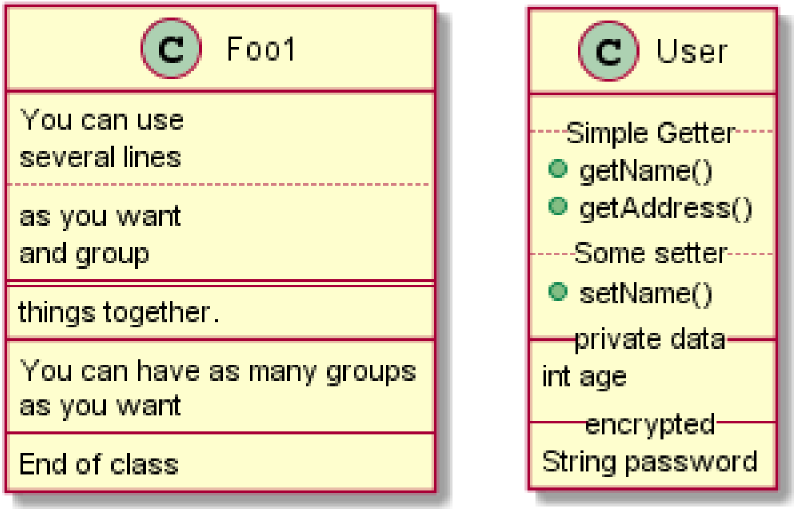
you can describe register mapping with the same syntax:

```
{"reg":[
  {"bits": 8, "name": "things"},
  {"bits": 2, "name": "stuff" },
  {"bits": 6},
 ],
 "config": { "bits":16,"lanes":1 }
 }
```

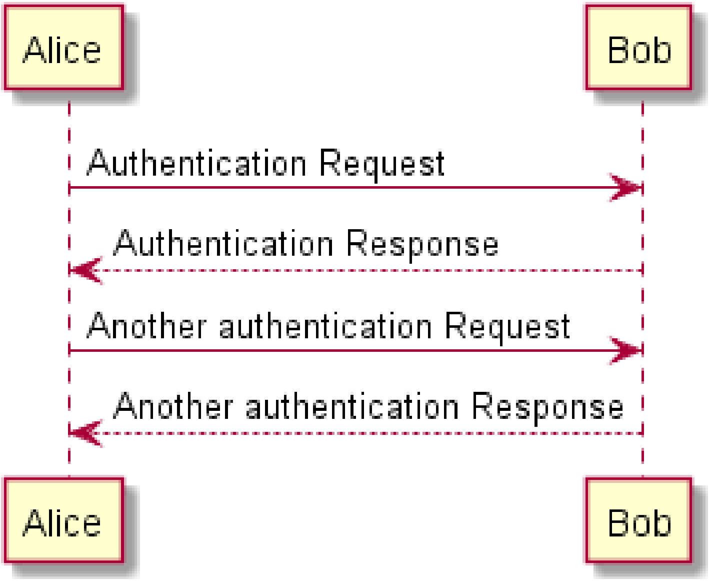
PlantUML

This documentation makes use of the `sphinxcontrib.plantuml` plugin, for more information see the sphinxcontrib.plantuml plugin and the PlantUML Webpage. For a small Cheatsheet for PlantUML see https://ogom.github.io/draw_uml/plantuml/

```
.. uml::

   class Foo1 {
     You can use
     several lines
     ..
     as you want
     and group
     ==
     things together.
     __
     You can have as many groups
     as you want
     --
     End of class
   }

   class User {
     .. Simple Getter ..
     + getName()
     + getAddress()
     .. Some setter ..
     + setName()
     __ private data __
     int age
     -- encrypted --
     String password
   }
```



```
.. uml::

   Alice -> Bob: Authentication Request
   Bob --> Alice: Authentication Response

   Alice -> Bob: Another authentication Request
   Alice <-- Bob: Another authentication Response
```

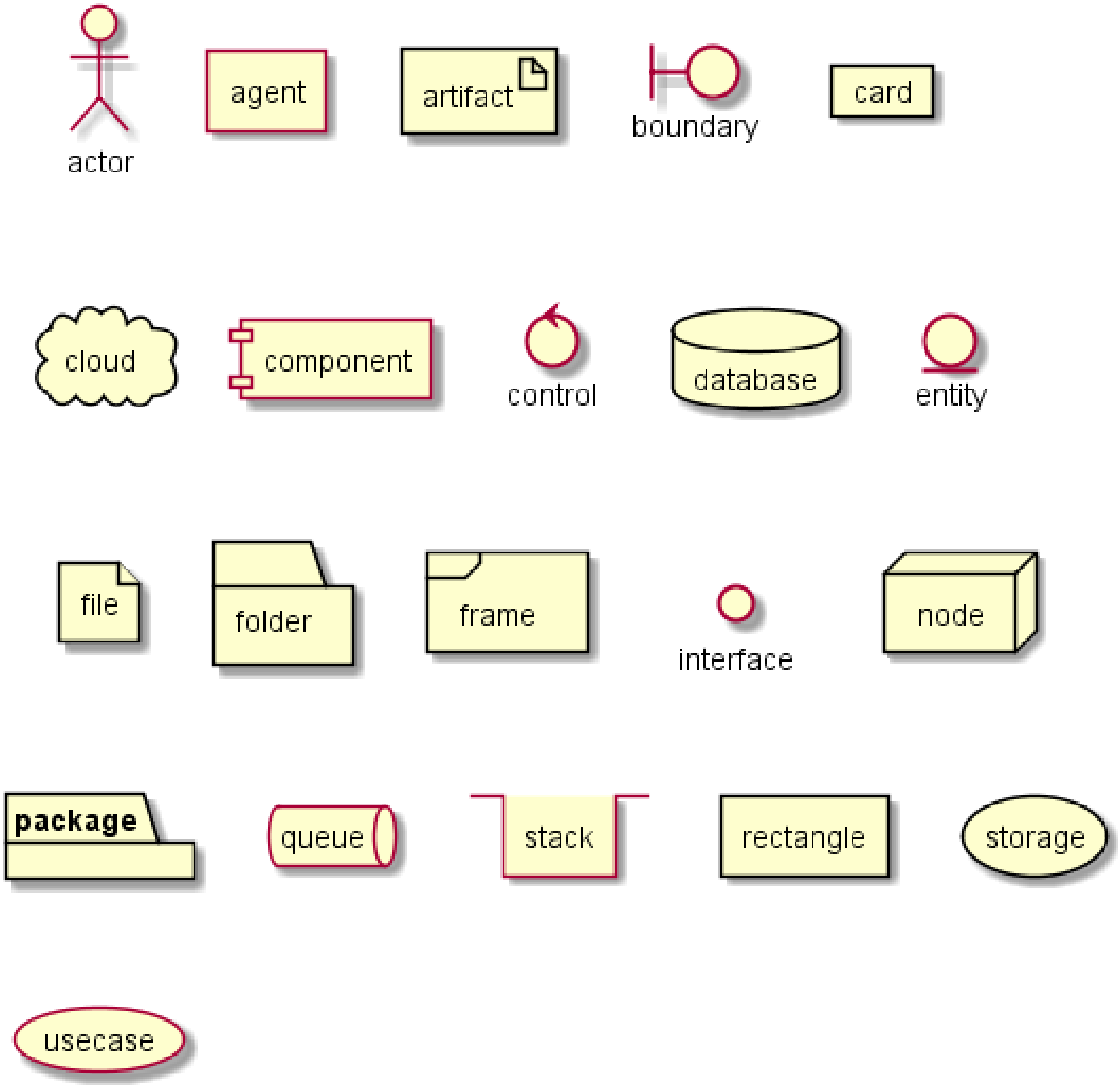

Alice
Bob
Authentication Request
Authentication Response
Another authentication Request
Another authentication Response
Alice
Bob

```
.. uml::

   actor actor
   agent agent
   artifact artifact
   boundary boundary
   card card
   cloud cloud
   component component
   control control
   database database
   entity entity
   file file
   folder folder
```

(continues on next page)

(continued from previous page)

```
    frame frame
    interface  interface
    node node
    package package
    queue queue
    stack stack
    rectangle rectangle
    storage storage
    usecase usecase
```



7

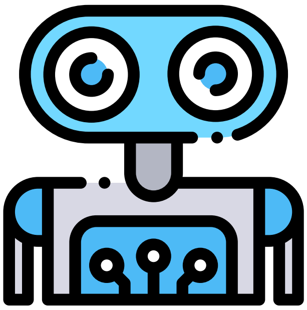
Multimedia

🙈🙉🙊

8

Security

9
About



9.1 About

9.1.1 Authors

- tschinz - Github Profile

9.1.2 Find me at

- Github
- Flickr
- Twitter @tschinz

9.2 Credits

On this website information, images and documents are used. Hereafter these credits are all listed.

Icons made by Freepik from Flaticon

9.3 How to use Sphinx Documentation

9.3.1 Sphinx Requirements

- make
  - Windows - GnuWin32
  - Linux

    ```
    sudo apt-get install build-essential
    ```

- Python 3
  - Python
  - Anaconda
- Python Modules (can be installed with pipenv)

  ```
  pip install sphinx
  pip install sphinx-rtd-theme
  pip install sphinxcontrib-wavedrom
  pip install sphinxcontrib-plantuml
  pip install recommonmark
  ```

- Latex Tools (only for latex build)
  - Windows
    - MikTex
    - TexStudio
  - Linux

    ```
    sudo apt install texlive-fonts-recommended texlive-latex-recommended␣
    ↪texlive-latex-extra
    ```

- Inkscape (for `.svg` to `.pdf` and to `.png` conversion)
  - Windows - Inkscape
  - Linux

    ```
    sudo apt-get install inkscape
    ```

9.3.2 How to create a new Sphinxdoc

```
sphinx-quickstart
```

9.3.3 How to Build Sphinxdoc locally

Without pipenv

- Install requirements see: *Sphinx Requirements*
- cd to the git folder
- Generate the desired output

```
make          # list all the available output format
make help     # list all the available output format

make html     # for html
make latex    # for latex
make latexpdf # for latex (will require latexpdf installed)

make clean    # cleans all generated file, TODO before commiting
make clean-images # cleans all autogerated png and pdf files
```

With pipenv

- Install requirements *Sphinx Requirements*
- Create a virtual environment with pipenv (will use the Pipfile for installing the necessary packages)

```
pipenv install
```

- then you can build the documentation

```
pipenv run make html
```

- if you want run `make` multiple times, prepone `pipenv run` on each command can be annoying, you can spawn a subshell with

```
pipenv shell
```

- and then you can use `make` the usual way

```
make          # list all the available output format
make help     # list all the available output format

make html     # for html
make latex    # for latex
make latexpdf # for latex (will require latexpdf installed)

make clean    # cleans all generated file, TODO before commiting
make clean-images # cleans all autogerated png and pdf files
```

all the outputs will be in `_build` folder

- html: `_build/html`
- pdf & tex: `_build/latex`

Continuous Build

During developement or creation of a page, the script build-loop.bash will rebuild the webpage every X seconds. In this way a constant preview of the page can be shown.

9.3.4 Commit to Repository

Before performing a commit the following steps are required:

- Verify the html documentation local *How to Build Sphinxdoc locally*

  ```
  make html
  ```

- Solve all build Warnings and Errors display during build in the commandline
- Generate pdf

  ```
  make latexpdf
  ```

- Clean the repo from generated files

  ```
  make clean
  ```

- Commit and push the changes *SPL Knowhow CI*

9.3.5 Continuous Integration(CI)

The .travis.yml will run on each master commit and create a _build/ folder which will be pushed onto the branch gh-pages and consequently be used by github to displayed static html pages.

9.4 HACK this documentation

If you want to add your page to this documentation you need to add your source file in the appropriate section. Every main section has its own folder structure and its own img/ folder containing all images for this section.

This documentation uses a recursive index tree: every folder have a special index.rst file that tell sphinx witch file, and in what order put it in the documentation tree.

If you don’t have enough knowledge about ReStructuredText then you can also use the pandoc translator or use the internal *Cheatsheet*

9.4.1 New Documentation Section

If you want to add a new section, you need to specify in the main index.rst, the section/ index.rst file of the new section.

```
.. toctree::
   :hidden:
   :glob:
   :maxdepth: 2
   :titlesonly:
   :caption: Content

```

(continues on next page)

(continued from previous page)

```
   linux/index
   mac/index
   windows/index
   tools/index
   coding/index
   writing/index
   multimedia/index
   security/index
   about/index
```

The section name should be the same as the folder name, but for Sphinx this is not required. Sphinx will take the name of the section from the title of the `section/index.rst` file.

9.4.2 Example Section

I want to document the new topic in SPL Knowhow repo, and want to create a section for it; let's call it `Section`

So I need to create a folder named `section/` (name is not important), and in it create a `section/index.rst` file like:

```
=============
Section Title
=============

.. figure:: img/logo.*
   :align: right
   :width: 150px

.. contents:: :local:

.. toctree::
   :glob:
   :maxdepth: 2
   :titlesonly:
   :caption: Content

   *
```

Note: The `.. toctree::` directive accept some parameters, in this case `:glob:` makes so you can use the * to include all the remaining files.

Note: The file path is relative to the index file, if you want to specify the absolute path, you need to prepend `/`

Now I can add additional `ReST` files like `section/intro.rst` and other files like `section/section_part_1.rst`, `ssection/ection_part_2.rst`, etc.

Section Images

Add an image folder in the section folder `section/img`, in case of additional documents ass a `section/docs` folder too.

Write the contents

That’s it, now you can add all you want in the new section `section` and all pages will show up in the documentation automatically.

9.5 License

10

Site purpose and structure

10.1 Getting started

Want to try it for yourself? Then jump to the *getting started* page and have fun, but first you need to learn *ReStructured Text* !!!

You can view the content as a:

- Webpage
- PDF
- Repo

10.2 Known Issues / TODOs

- Github CI not working for PDF creation
- Missing pages from original Zawiki
- missing pages from config repo