FreiCAR: Introduction to ROS/Dev-Environment

Johan Vertens
Overview

- ROS
- FreiCAR Docker
- Build Process
ROS

In the following we leverage content from the slides from RSL (Robotic System Lab, ETH Zurich) [link]
What is ROS?

ROS = Robot Operating System

Plumbing
- Process management
- Inter-process communication
- Device drivers

Tools
- Visualization
- Graphical user interface
- Data logging

Capabilities
- Control
- Planning
- Perception
- Mapping
- Manipulation

Ecosystem
- Package organization
- Software distribution
- Documentation
- Tutorials
History of ROS

- Originally developed in 2007 at the Stanford Artificial Intelligence Laboratory
- Since 2013 managed by OSRF
- Today used by many robots, universities and companies
- De facto standard for robot programming
ROS 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.).

- **Light-weight**
  Stand-alone libraries are wrapped around with a thin ROS layer.

- **Free and open-source**
  Most ROS software is open-source and free to use.
ROS Master

- Manages the communication between nodes (processes)
- Every node registers at startup with the master

Start a master with

> roscore

More info
http://wiki.ros.org/Master
ROS Nodes

- Single-purpose, executable program
- Individually compiled, executed, and managed
- Organized in packages

Run a node with

> `rosrunc package_name node_name`

See active nodes with

> `rosnodellist`

Retrieve information about a node with

> `rosnodinfo node_name`

More info

http://wiki.ros.org/rosnodellist
**ROS Topics**

- Nodes communicate over *topics*
  - Nodes can *publish* or *subscribe* to a topic
  - Typically, 1 publisher and *n* subscribers
- Topic is a name for a stream of *messages*

**List active topics with**
- `rostopic list`

**Subscribe and print the contents of a topic with**
- `rostopic echo /topic`

**Show information about a topic with**
- `rostopic info /topic`

---

**More info**

[http://wiki.ros.org/rostopic](http://wiki.ros.org/rostopic)
**ROS Messages**

- Data structure defining the *type* of a topic
- Comprised of a nested structure of integers, floats, booleans, strings etc. and arrays of objects
- Defined in *.msg* files

See the type of a topic
> rostopic type /topic

Publish a message to a topic
> rostopic pub /topic type data

More info
http://wiki.ros.org/Messages
**ROS 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
Start a roscore with

> roscore
Example Part 2

Run a talker demo node with

```bash
> rosrn beginner_tutorials talker
```

```
[ INFO] [1486051798.424661519]: hello world 0
[ INFO] [1486051798.52527845]: hello world 1
[ INFO] [1486051798.624747012]: hello world 2
[ INFO] [1486051798.724826782]: hello world 3
[ INFO] [1486051798.82598577]: hello world 4
[ INFO] [1486051798.925379775]: hello world 5
[ INFO] [1486051799.024971332]: hello world 6
[ INFO] [1486051799.12545096]: hello world 7
[ INFO] [1486051799.22527747]: hello world 8
[ INFO] [1486051799.32538921]: hello world 9
```
Example Part 3

See the list of active nodes
> rosnodes list

Show information about the **talker** node
> rosnodes info /talker

See information about the **chatter** topic
> rostopic info /chatter
Check the type of the *chatter* topic

```bash
> rostopic type /chatter
```

Show the message contents of the topic

```bash
> rostopic echo /chatter
```

Analyze the frequency

```bash
> rostopic hz /chatter
```
Example Part 5

Run a listener demo node with

> rosrun beginner_tutorials listener

```
student@ubuntu:/catkin_ws$ rosrun roscpp_tutorials listener
[ INFO] [1486053802.204104598]: I heard: [hello world 19548]
[ INFO] [1486053802.804538827]: I heard: [hello world 19549]
[ INFO] [1486053802.493853395]: I heard: [hello world 19550]
[ INFO] [1486053802.504438133]: I heard: [hello world 19551]
[ INFO] [1486053802.604297608]: I heard: [hello world 19552]
```
Example Part 6

See the new *listener* node with

```
> rosnod list
```

Show the connection of the nodes over the chatter topic with

```
> rostopic info /chatter
```

```
student@ubuntu:~/catkin_ws$ rosnod list
/listener
/rosout
/talker
```

```
student@ubuntu:~/catkin_ws$ rostopic info /chatter
Type: std_msgs/String

Publishers:
  * /talker (http://ubuntu:39173/)

Subscribers:
  * /listener (http://ubuntu:34664/)
```
Launch Files

- `launch` is a tool for launching multiple nodes (as well as setting parameters)
- Are written in XML as `*.launch` files
- If not yet running, launch automatically starts 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
```

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

Example console output for
```
roslaunch roscpp_tutorials talker_listener.launch
```

```
student@ubuntu:/catkin_ws roslaunch roscpp_tutorials talker_listener.launch ... logging to /home/student/.ros/log/794321aa-e950-11e6-95db-800c297bd368/roslog Checking log disk usage for disk usage. This may take awhile. Press Ctrl-C to interrupt Done checking log file disk usage. Usage is <16GB.

started roslaunch server http://ubuntu:37592/
```

```
PARAMETERS
* /rosdistro: indigo
* /rosversion: 1.11.20

NODES

- listener (roscpp_tutorials/listener)
- talker (roscpp_tutorials/talker)

auto-starting new master
process[master]: started with pid [5772]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 794321aa-e950-11e6-95db-800c297bd368
process[rosout-1]: started with pid [5765]

started core service [rosout]
process[listener-2]: started with pid [5788]
process[talker-3]: started with pid [5795]
[ INFO] [1486044252.537801350]: hello world 0
[ INFO] [1486044252.639886504]: hello world 1
[ INFO] [1486044252.738279674]: hello world 2
[ INFO] [1486044252.838357245]: hello world 3
```
The catkin workspace contains the following spaces

**Work here**
- **src**
  - The *source space* contains the source code. This is where you can clone, create, and edit source code for the packages you want to build.

**Build space**
- **build**
  - The *build space* is where CMake is invoked to build the packages in the source space. Cache information and other intermediate files are kept here.

**Development (devel) space**
- **devel**
  - The *development (devel) space* is where built targets are placed (prior to being installed).

If necessary, clean the entire build and devel space with

```bash
> catkin clean --all
```

More info

http://wiki.ros.org/catkin/workspaces
FreiCAR: Environment

- In the FreiCAR-Docker everything you need is already installed: ROS, FreiCarSim, Libraries, Anaconda, Clion, Pycharm
- We use Clion as C++ IDE and Pycharm as Python IDE
- Try to run the ROS example in the docker!
- Persistent data will be stored in the folders `freicar_deps` and `freicar_ws`
- The folder `freicar_ws` holds the ROS workspace
FreiCAR: Environment

- Everything you run should be started inside a terminal that is attached to the docker
- Follow the tutorial on [docs](#) to install FreiCAR-Docker

then:

Start a new terminal and type

```
> fcc
```

This starts the freicar-docker environment. Always make sure that this is only running once at all times.
For multiple terminals that are attached to the same docker container run (in a new terminal):

```
> fct
```

Note that for this the fcc (freicar container) needs to run in some other terminal already.

There is still only one container running, check with:

```
> docker ps
```
FreiCAR: Changing the Image

If you change something within the image that is not stored in the persistent folders you can commit your changes when your docker is still running

> ./commit_changes.bash

If you do this, make sure you keep track on your changes!
FreiCAR: Coding - Python

- Most ROS functionalities will only work in Python 2.7
- For Pytorch: Set the interpreter to the freicar-env
- For more information see recording

Start the PyCharm IDE with:

```bash
> pycharm
```
FreiCAR: Coding - C++

- Most templates for the exercises are written in C++
- For performance critical nodes always prefer C++
- HD Map is in C++ available only

Start the Clion IDE with:

```
> clion
```

In Clion set the CMake options (for every project) to:

```
-DCATKIN_DEVEL_PREFIX:PATH=/home/freicar/freicar_ws/devel.
```

And the build path to:

```
/home/freicar/freicar_ws/build/NAME_OF_PROJECT
```
Hello FreiCarSim

Clone submodules (if you didn’t do it yet) (inside the freicar_docker repo)

> git submodule update --init --recursive

Build everything in the freicar workspace

> cd ~/freicar_ws && catkin build

Or start the base script that spawns a car as well!

> roslaunch freicar_launch sim_base.launch

Visualize all rostopics

> rviz
Hello FreiCarSim

Go to python example script

> roscd beginner_tutorials/scripts/

Let the car drive circles and get images

> python startCar.py