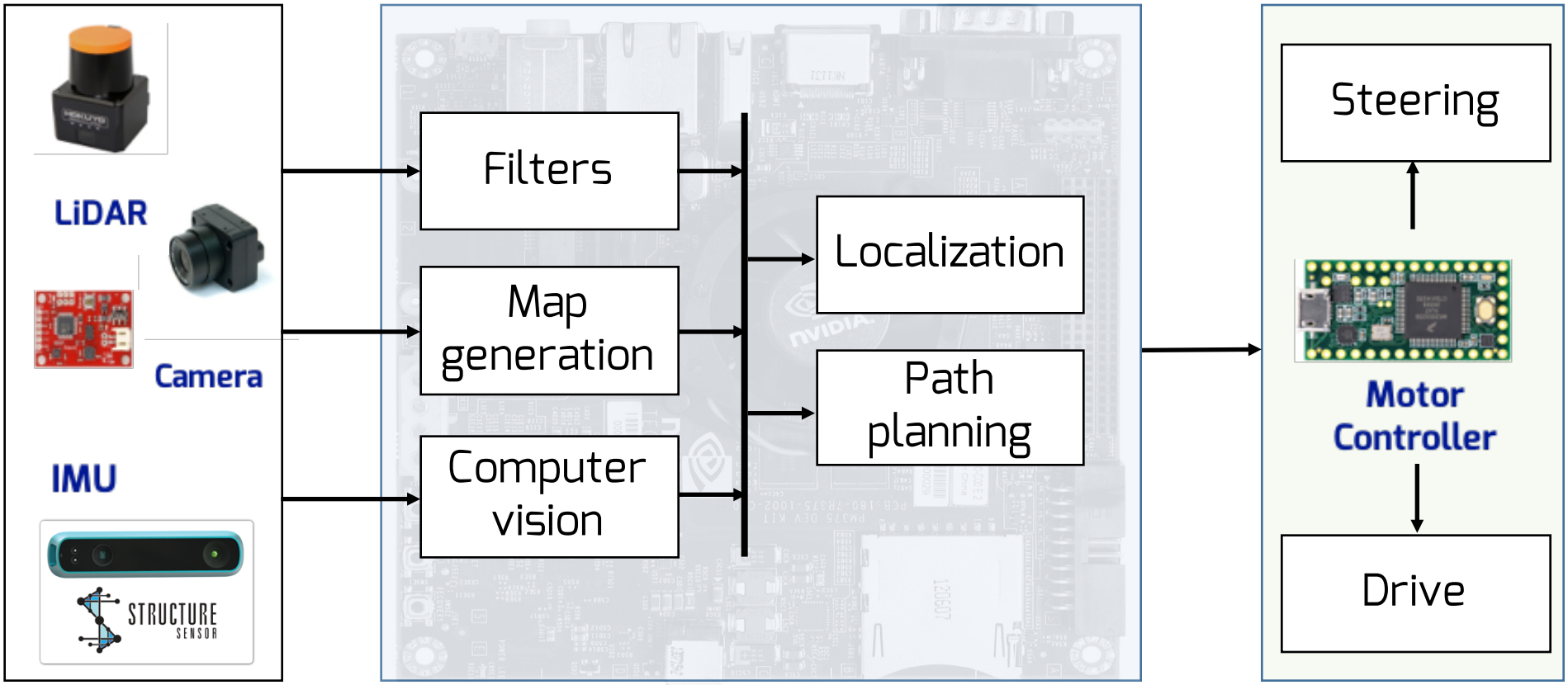
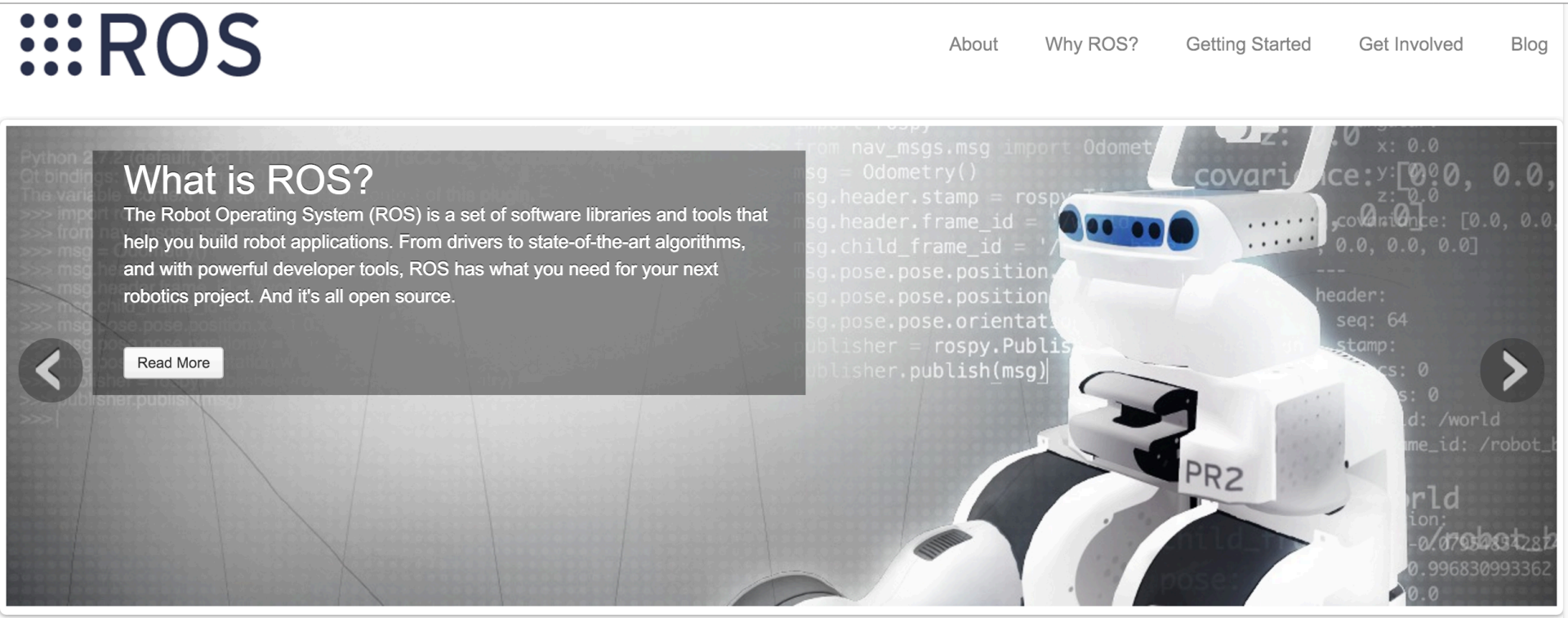
System Architecture



ROS: Robot Operating System



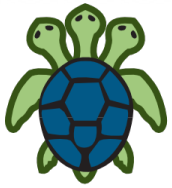
ROS.org

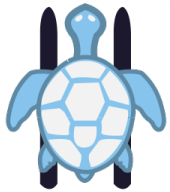
Open Source Robotics Foundation

ROS distributions

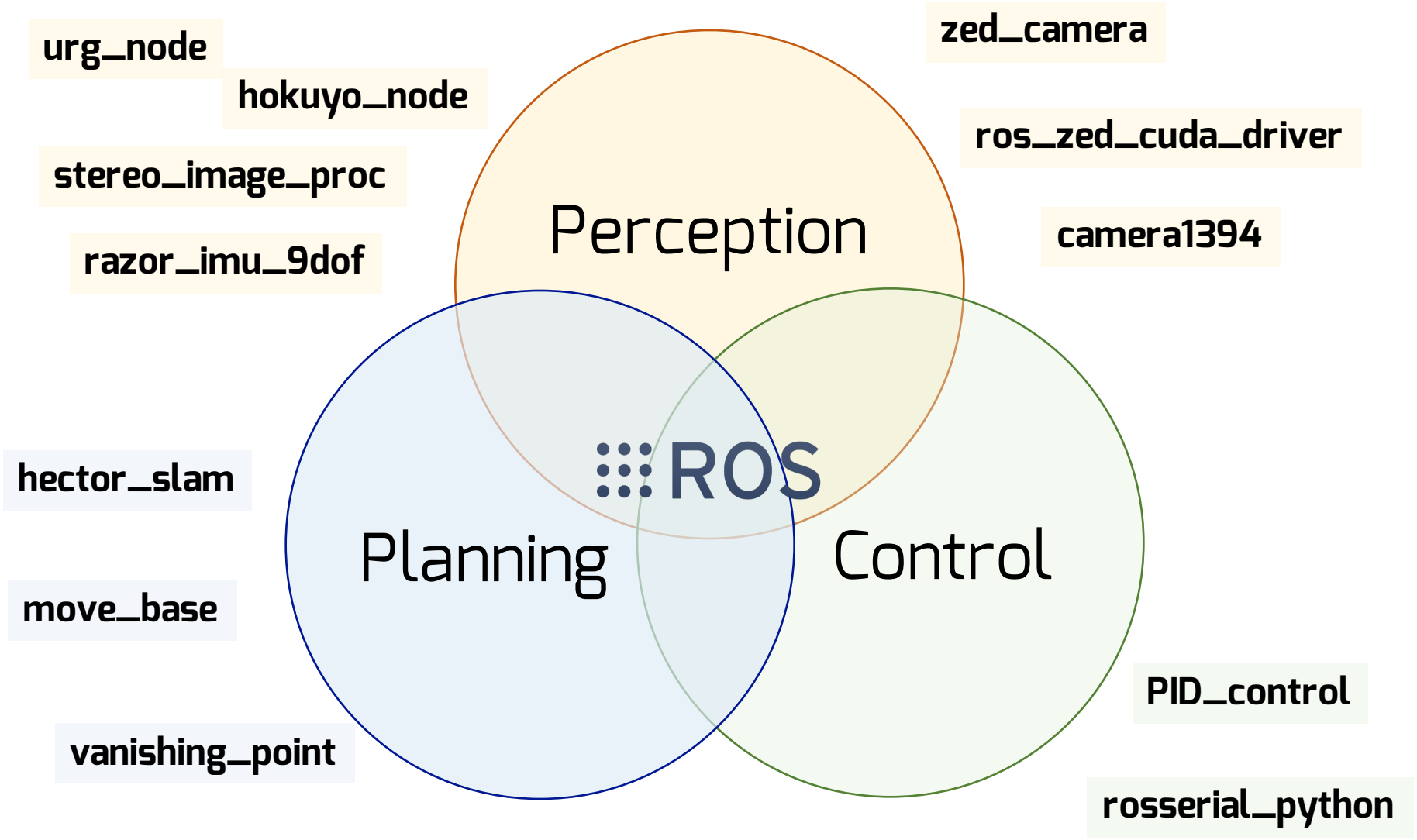
This course

2012 2013 2014 2015



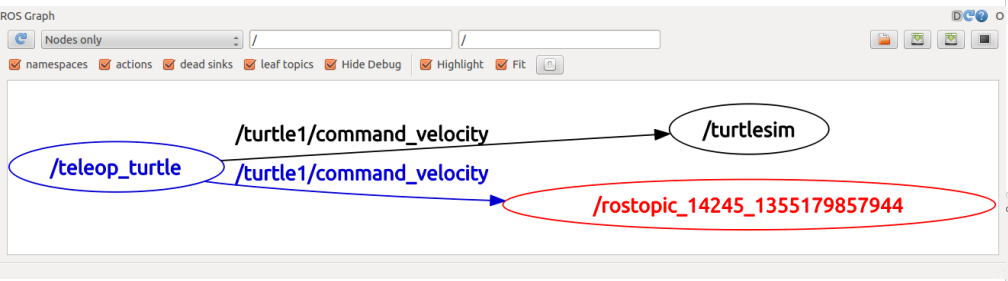


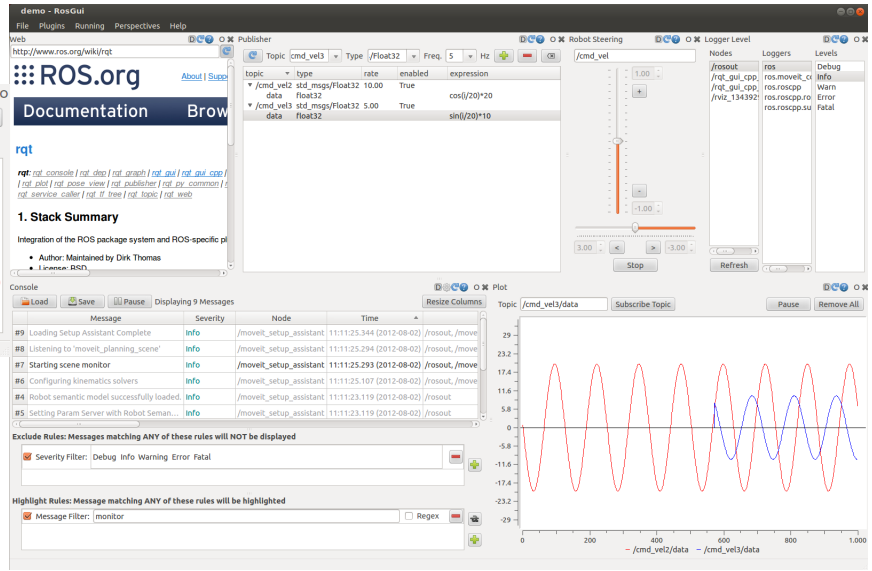
ROS Capabilities

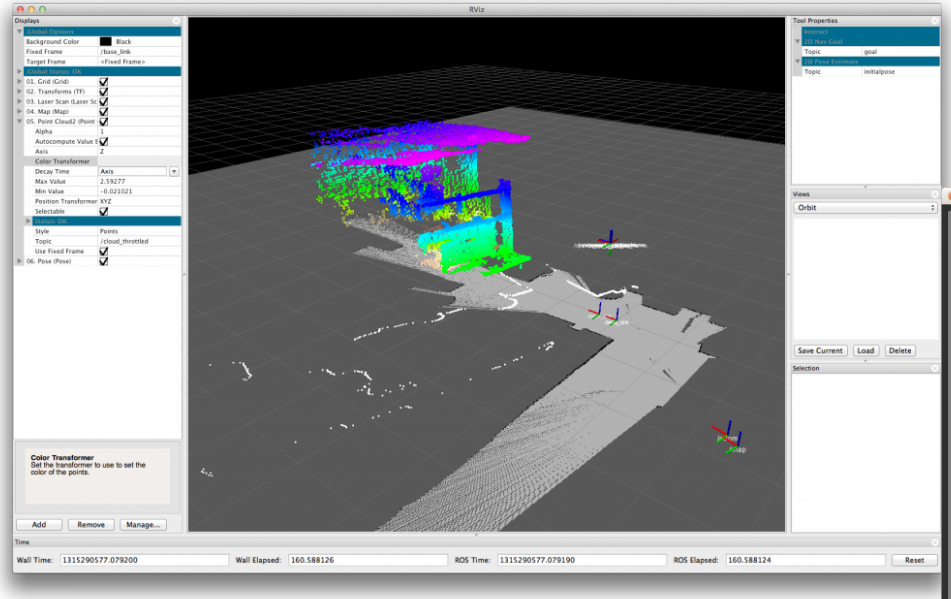


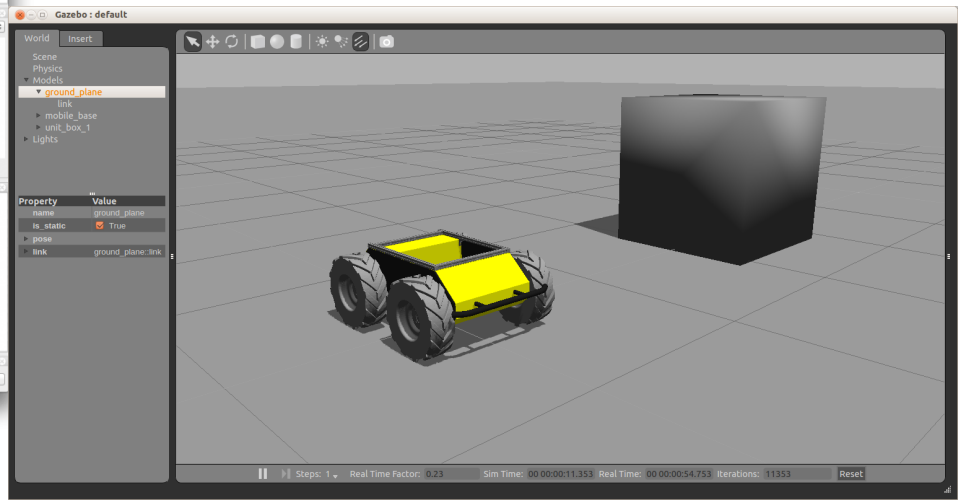
ROS Tools

Visualization, debugging and diagnostics, logging, and simulation

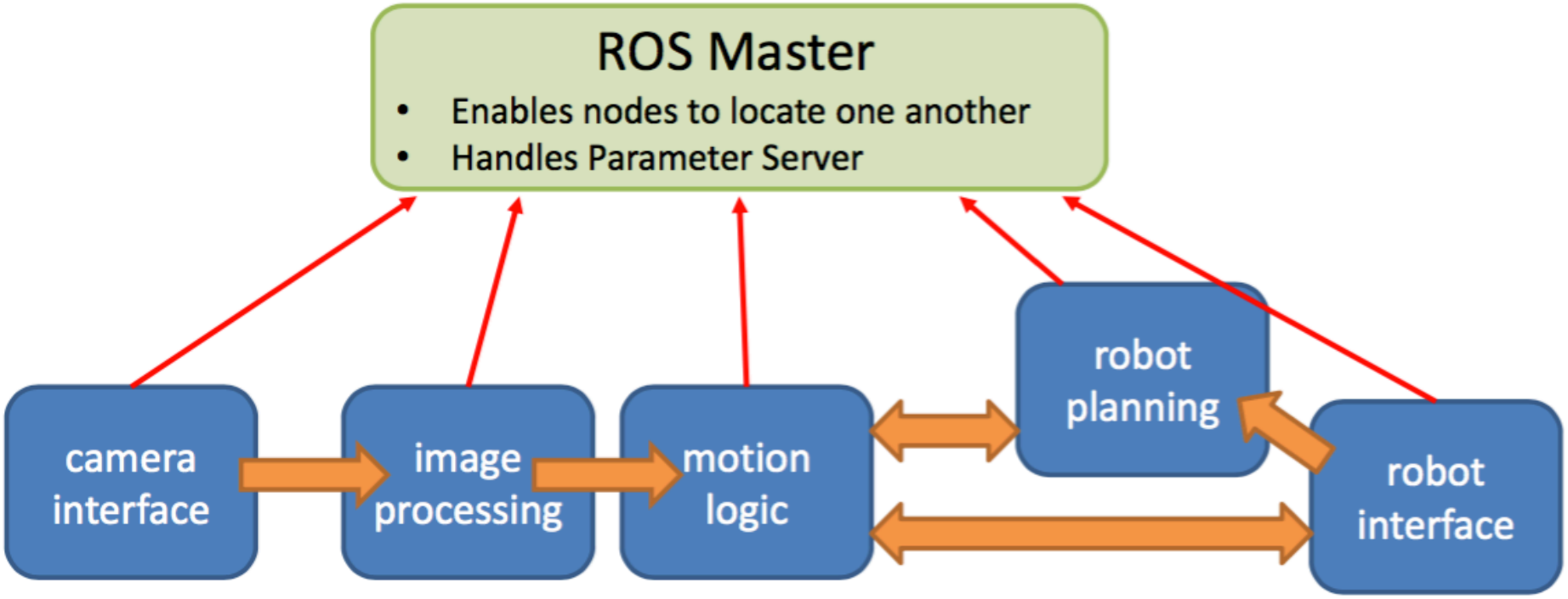








ROS: Nodes



Node: Program with a specific functionality, that runs as a single process.

Nodes communicate with other nodes using **topics** and **messages**

ROS: Topics

Topics are channels over which *nodes* exchange *messages*.

They are for **streaming data**

Publishes on topic: Scan

Scan [Topic]

Mapping Node

hokuyo_node

Subscribes to topic: Scan

Publisher Node

Subscriber Node

ROS: Messages

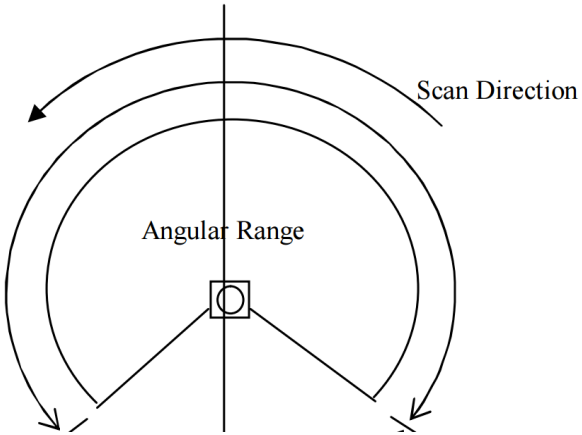
Messages are the strongly-typed **data structure** for a *topic*.

hokuyo_node

LaserScan [Message]

Scan [Topic]

Mapping Node



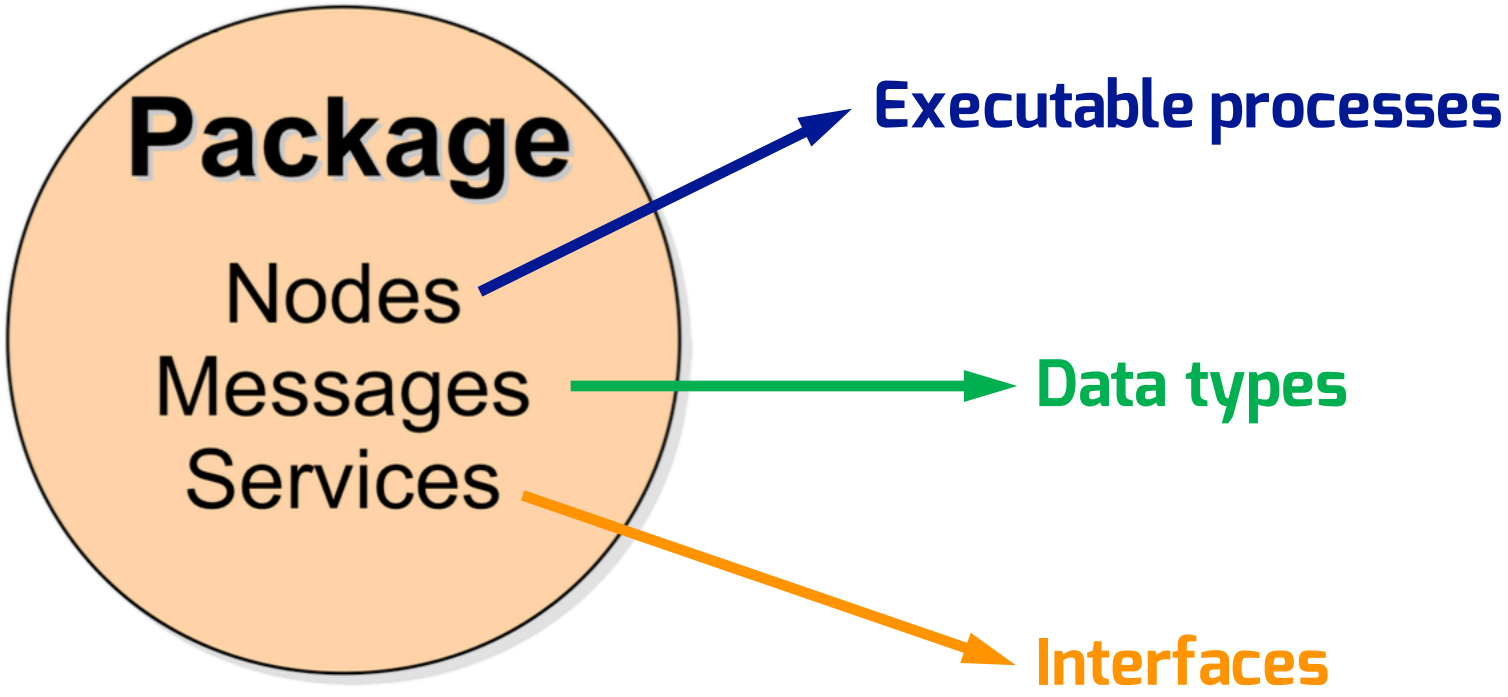

```
std_msgs/Header header
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```

Subscriber Node

ROS: Packages

Software in ROS is organized into **packages.**

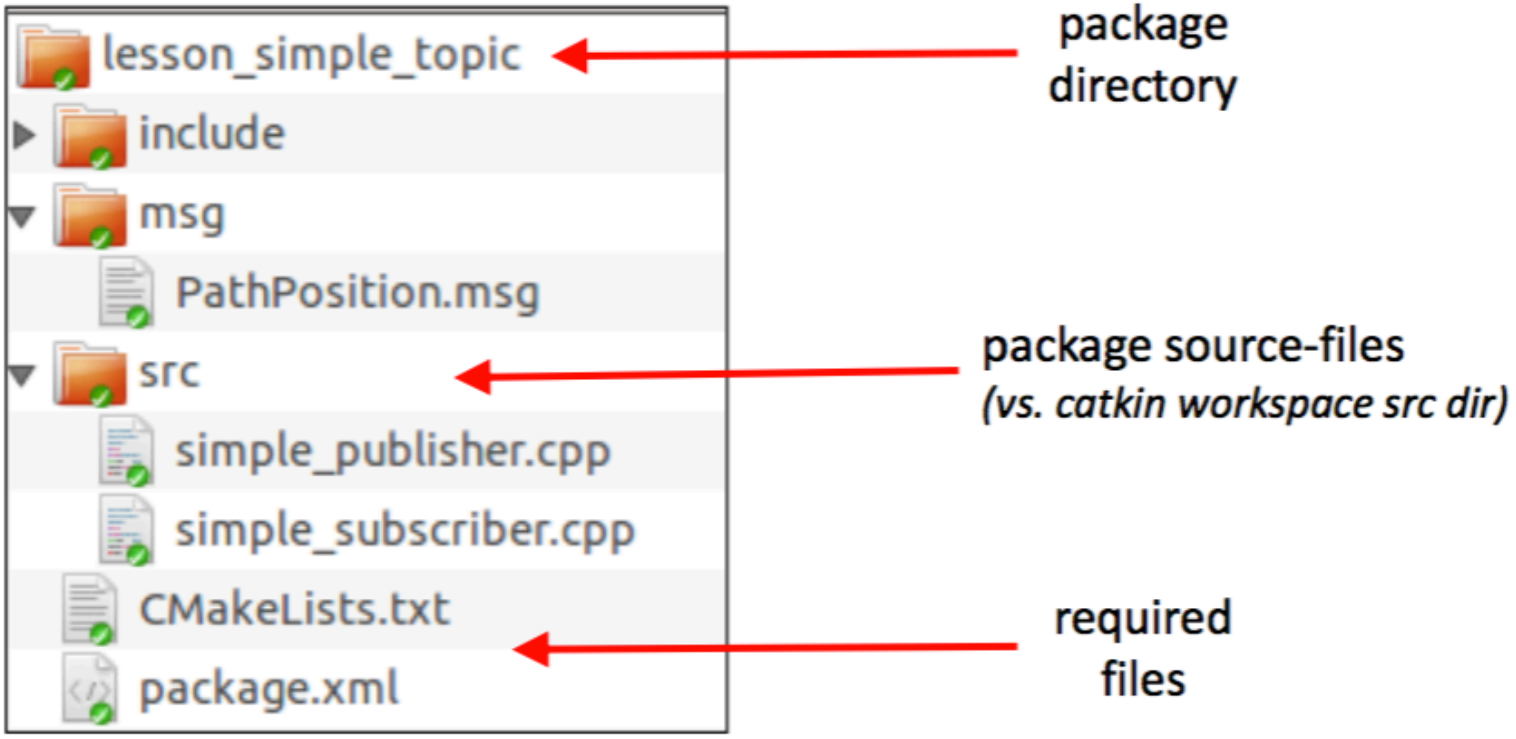
A **package** contains one or more **nodes**.



ROS: Packages

Packages contain several required files:

- `package.xml`
- `CMakeLists.txt`



package.xml

Name, description, author, license…

```
<package>
  <name>lesson_simple_parameters</name>
  <version>0.0.0</version>
  <description>The lesson_simple_parameters package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example:  -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="jane.doe@example.com">Jane Doe</maintainer>


  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>BSD</license>
```

package.xml

Dependencies:

- `<build_depend>` : Needed to compile the package.
- `<run_depend>` : Needed to run the package

```
<build_depend>roscpp</build_depend>
<build_depend>industrial_robot_client</build_depend>
<build_depend>simple_message</build_depend>

<run_depend>roscpp</run_depend>
<run_depend>industrial_robot_client</run_depend>
```

CMakeLists.txt

Rules for compiling and building the software.

```
include_directories(include ${catkin_INCLUDE_DIRS})
```

Adds directories to CMAKE include rules

```
add_executable(myNode src/myNode.cpp src/widget.cpp)
```

Builds program `myNode`, from `myNode.cpp` and `widget.cpp`

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

Links node `myNode` to dependency libraries

Basic ROS commands: `roscore`

`roscore` is the first thing that you should run when starting ROS.

```
$ roscore
```

Collection of nodes and programs that are pre-requisites of a ROS-based system.

It starts up:

- The ROS Master
- A rosout logging node.

Basic ROS commands: `rosrun`

`rosrun` executes a ROS node.

```
$ rosrun <package_name> <node_name>
```

Example

```
$ rosrun hokuyo_node hokuyo_node
```

Basic ROS commands: `rosnode`

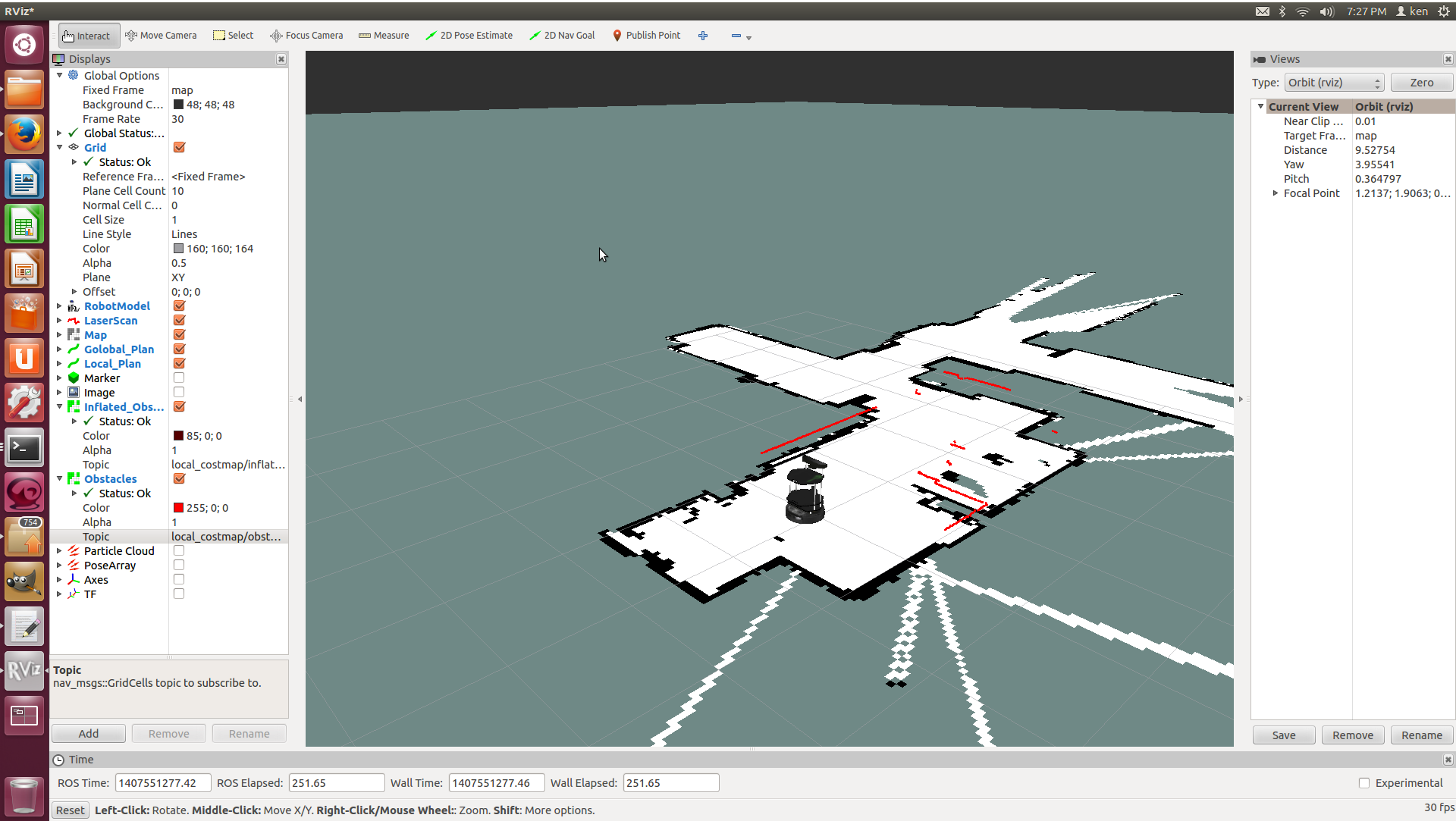
Command	Description
`rosnode list`	List all active nodes
`rosnode info node_name`	Display information about a node
`rosnode kill node_name`	Kill running node
`Rosnode ping node_name`	Test connectivity to an active node

Basic ROS commands: `rostopic`

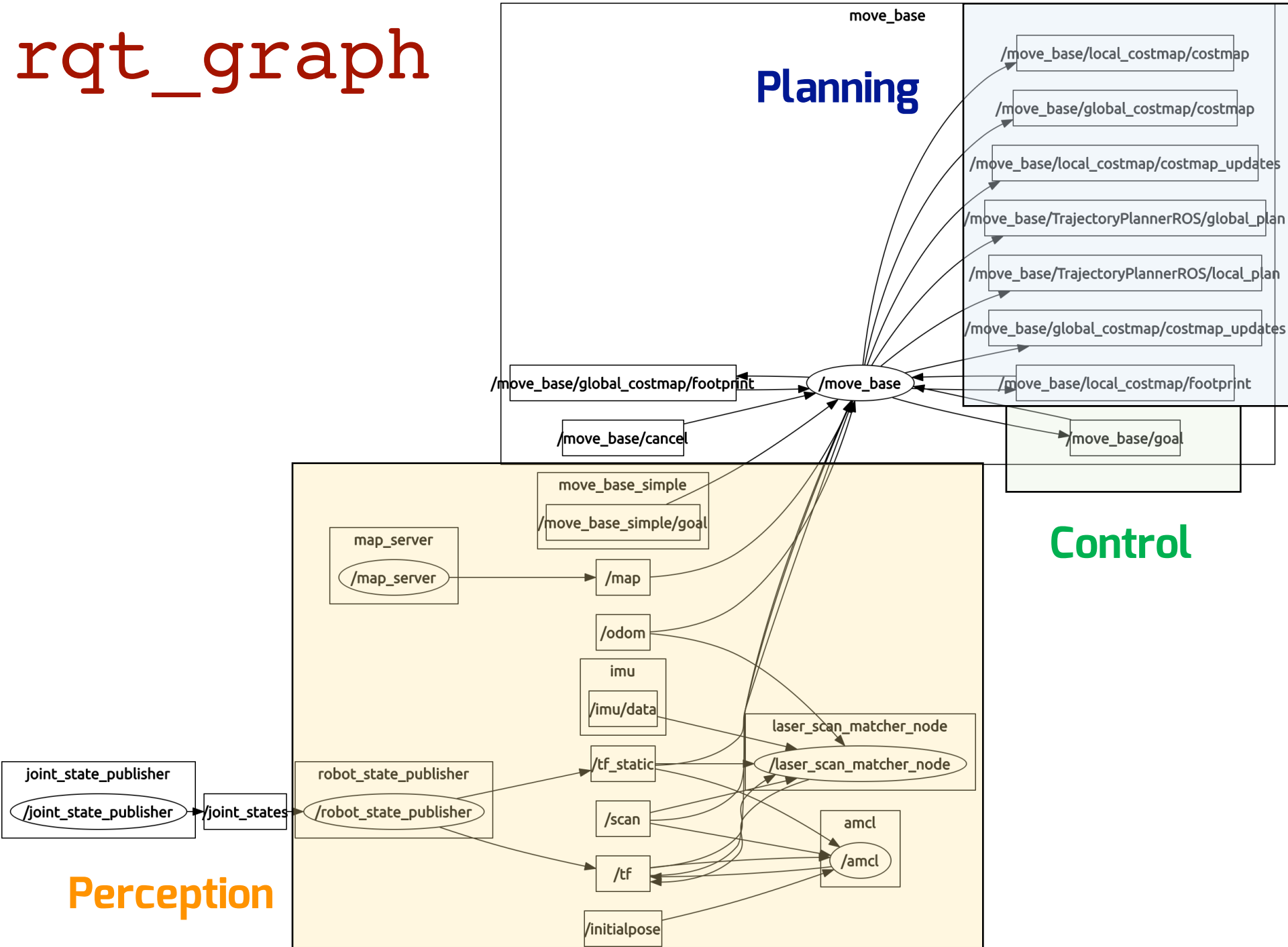
Command	Description
`rostopic list`	List all topics currently subscribed to and/or publishing
`rostopic info <topic>`	Show topic message type, subscribers, publishers etc.
`rostopic echo <topic>`	Echo messages published to the topic on the terminal window
`rostopic find <message_type>`	Find topics of the given message type

3D visualization tool: rviz

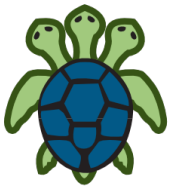
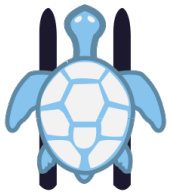
```
$ rosrun rviz rviz
```



rqt_graph


move_base
Planning
/move_base/local_costmap/costmap
/move_base/global_costmap/costmap
/move_base/local_costmap/costmap_updates
/move_base/TrajectoryPlannerROS/global_plan
/move_base/TrajectoryPlannerROS/local_plan
/move_base/global_costmap/costmap_updates
/move_base/local_costmap/footprint
/move_base/global_costmap/footprint
/move_base
/move_base/cancel
/move_base/goal
Control
move_base_simple
/move_base_simple/goal
map_server
/map_server
/map
/odom
imu
/imu/data
laser_scan_matcher_node
/laser_scan_matcher_node
/tf_static
joint_state_publisher
/joint_state_publisher
/joint_states
robot_state_publisher
/robot_state_publisher
/scan
amcl
/amcl
/tf
/initialpose
Perception

But what about the turtles ?


Next time

TurtleSim

Keyboard Control

Practice Session 1