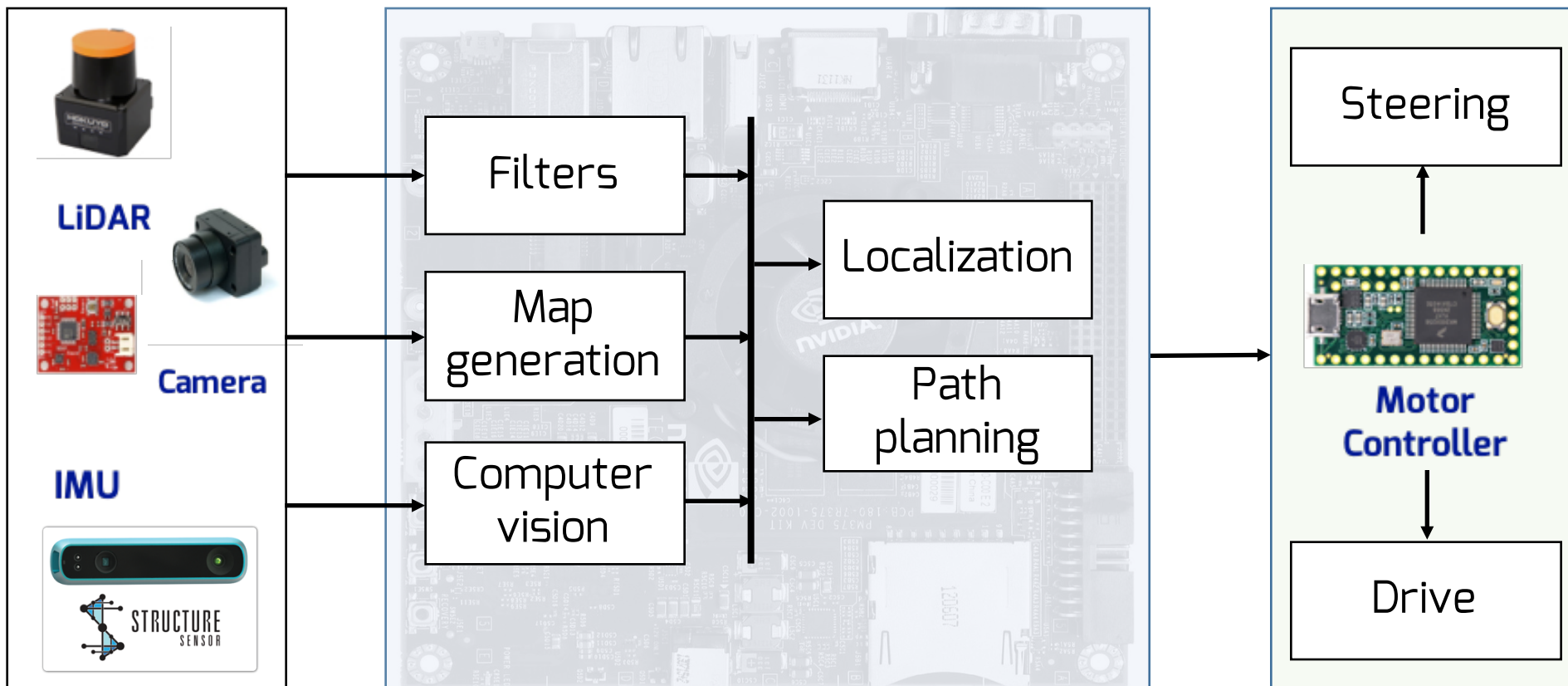


System Architecture



Perception

Planning

Control

ROS: Robot Operating System

ROS

[About](#)

[Why ROS?](#)

[Getting Started](#)

[Get Involved](#)

[Blog](#)

What is ROS?

The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source.

[Read More](#)



ROS.org



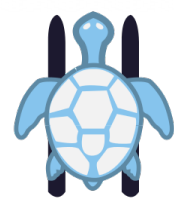
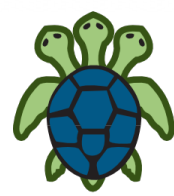
Open Source Robotics Foundation

ROS distributions

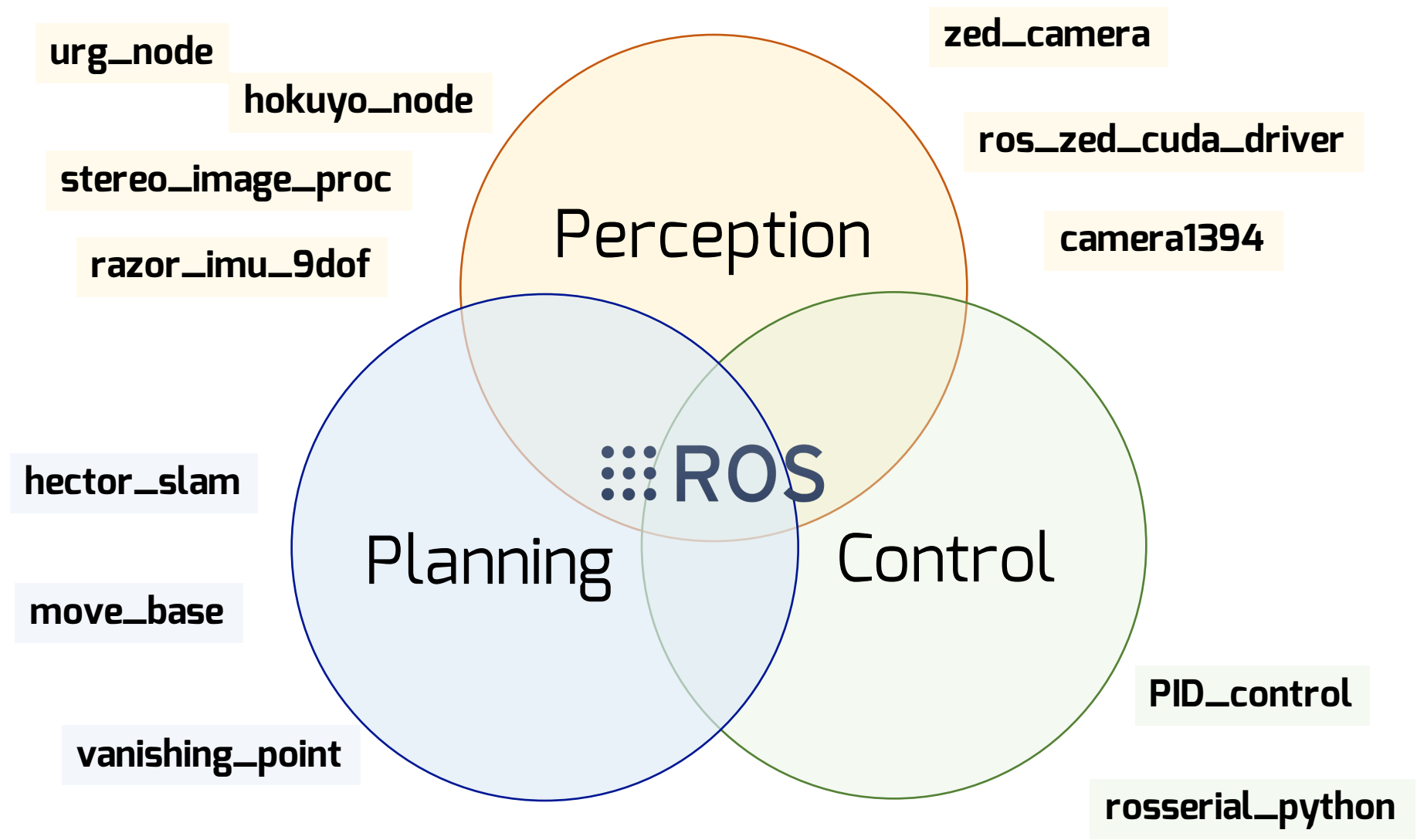
This course



2012 2013 2014 2015



ROS Capabilities



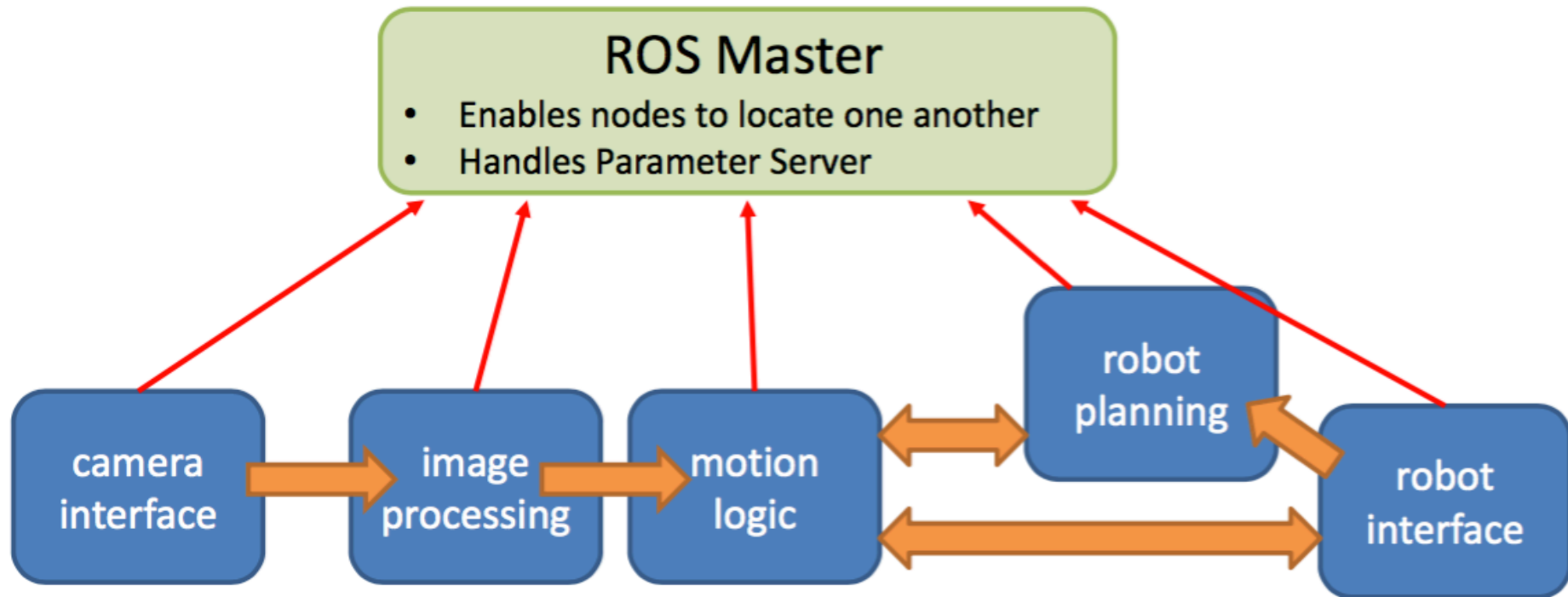
ROS Tools

Visualization, debugging and diagnostics, logging, and simulation

The image displays several ROS tool interfaces:

- ROS Graph:** Shows a network of nodes. `/teleop_turtle` and `/turtle1/command_velocity` are circled in blue. `/turtlesim` is circled in black. `/rostopic_14245_1355179857944` is circled in red.
- RViz:** A 3D visualization of a robot's environment with a point cloud and a robot model.
- Gazebo:** A 3D simulation environment showing a yellow robot and a black cube.
- ROS Console:** Displays system messages and logs.
- ROS Topic Monitor:** Shows a plot of `/cmd_vel3/data` with a red sine wave and a blue sine wave.

ROS: Nodes



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

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

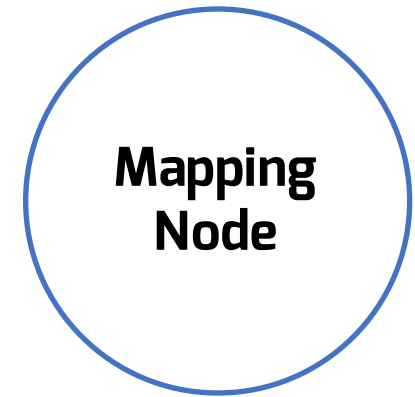
ROS: Topics

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

They are for **streaming data**



Publishes on topic: Scan



**Mapping
Node**

Subscribes to topic: Scan

hokuyo_node

Publisher Node

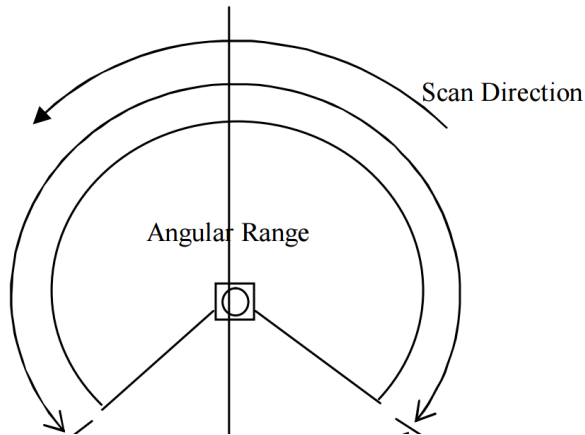
Subscriber Node

ROS: Messages

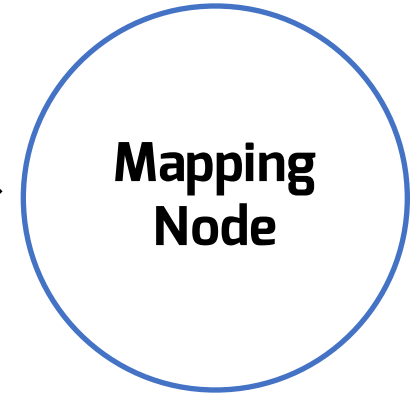
Messages are the strongly-typed **data structure** for a *topic*.



hokuyo_node



LaserScan (Message)



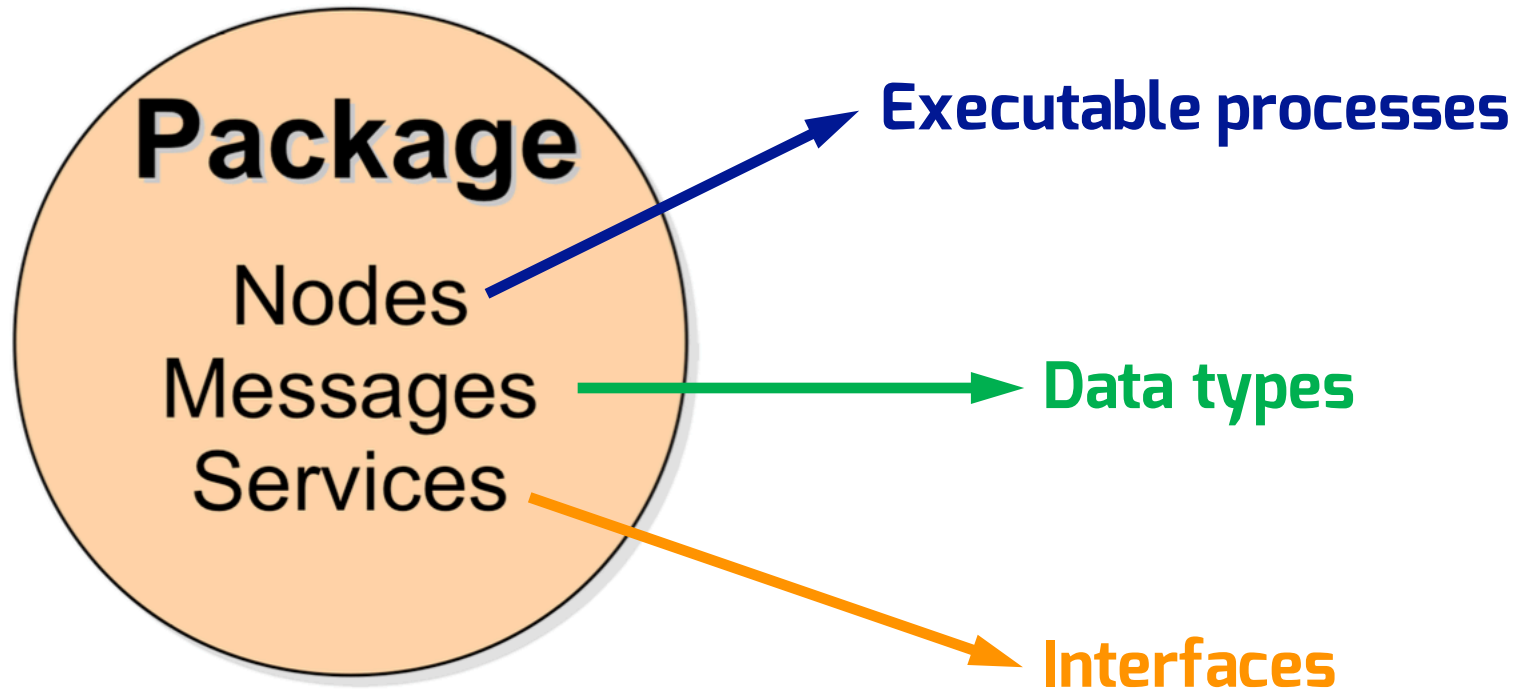
```
std_msgs/Header header
float32 angle_min
float32 angle_max
float32 angle_increment
float32 time_increment
float32 scan_time
float32 range_min
float32 range_max
float32[] ranges
float32[] intensities
```



ROS: Packages

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

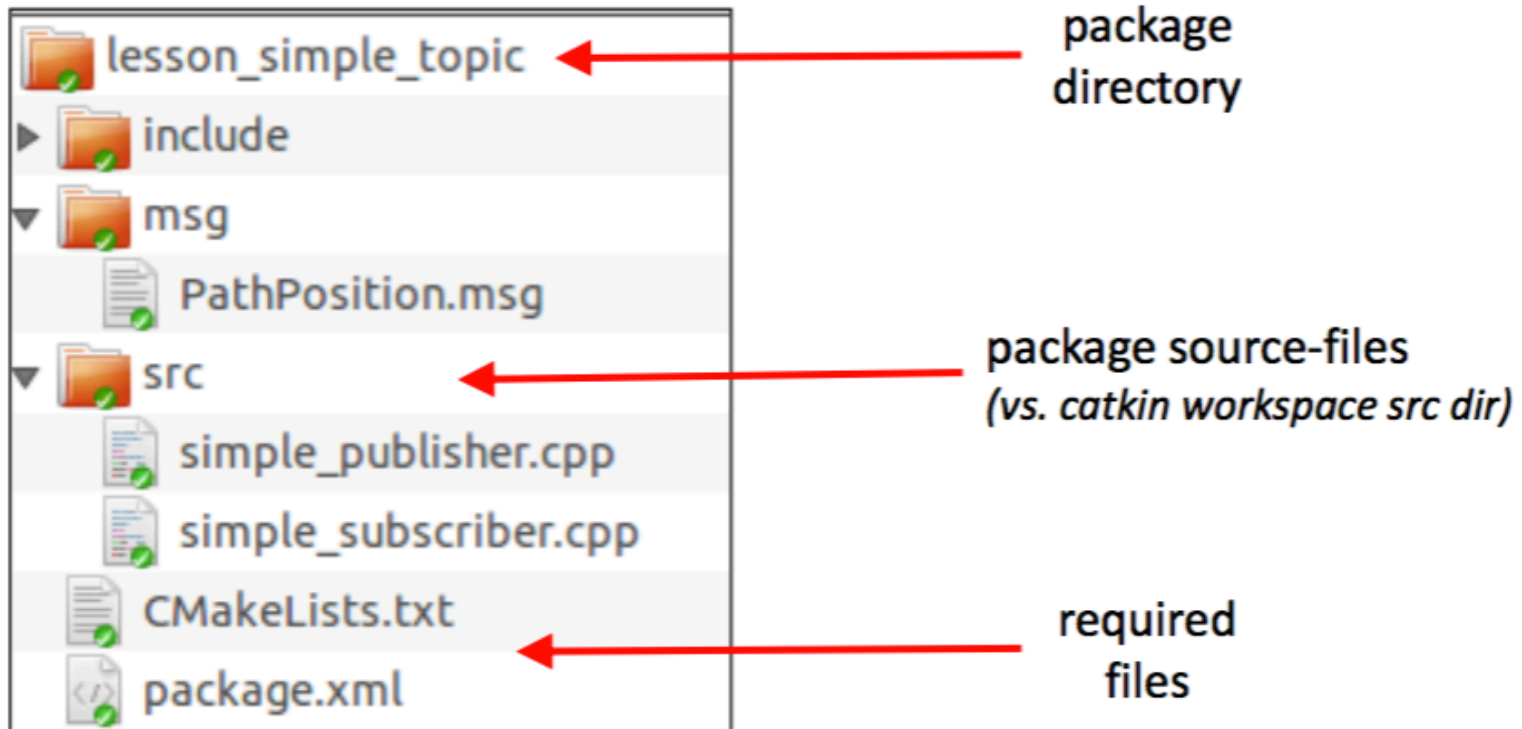
A **package** contains one or more **nodes**.



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: `roslaunch`

`roslaunch` executes a ROS node.

```
$ roslaunch <package_name> <node_name>
```

Example

```
$ roslaunch hokuyo_node hokuyo_node
```



Basic ROS commands: **roscall**

Command	Description
<code>roscall list</code>	List all active nodes
<code>roscall info node_name</code>	Display information about a node
<code>roscall kill node_name</code>	Kill running node
<code>roscall ping node_name</code>	Test connectivity to an active node

Basic ROS commands: `rostopic`

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

3D visualization tool: *rviz*

```
$ rosrun rviz rviz
```

The screenshot displays the RViz 3D visualization tool interface. The main window shows a 3D view of a robot on a grid with a map overlay and laser scan data. The left sidebar contains a 'Displays' panel with various visualization options like Grid, RobotModel, LaserScan, Map, Global_Plan, Local_Plan, Marker, Image, Inflated_Obs..., Obstacles, Particle Cloud, PoseArray, Axes, and TF. The right sidebar shows a 'Views' panel with 'Orbit (rviz)' selected, displaying camera parameters like Near Clip, Target Fra..., Distance, Yaw, Pitch, and Focal Point. The bottom status bar shows ROS Time, ROS Elapsed, Wall Time, and Wall Elapsed.

Displays Panel:

- Global Options
 - Fixed Frame: map
 - Background Color: 48; 48; 48
 - Frame Rate: 30
- Global Status:
- Grid
 - Status: Ok
 - Reference Frame: <Fixed Frame>
 - Plane Cell Count: 10
 - Normal Cell Count: 0
 - Cell Size: 1
 - Line Style: Lines
 - Color: 160; 160; 164
 - Alpha: 0.5
 - Plane: XY
 - Offset: 0; 0; 0
- RobotModel:
- LaserScan:
- Map:
- Global_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:

Views Panel:

Type: Orbit (rviz) Zero

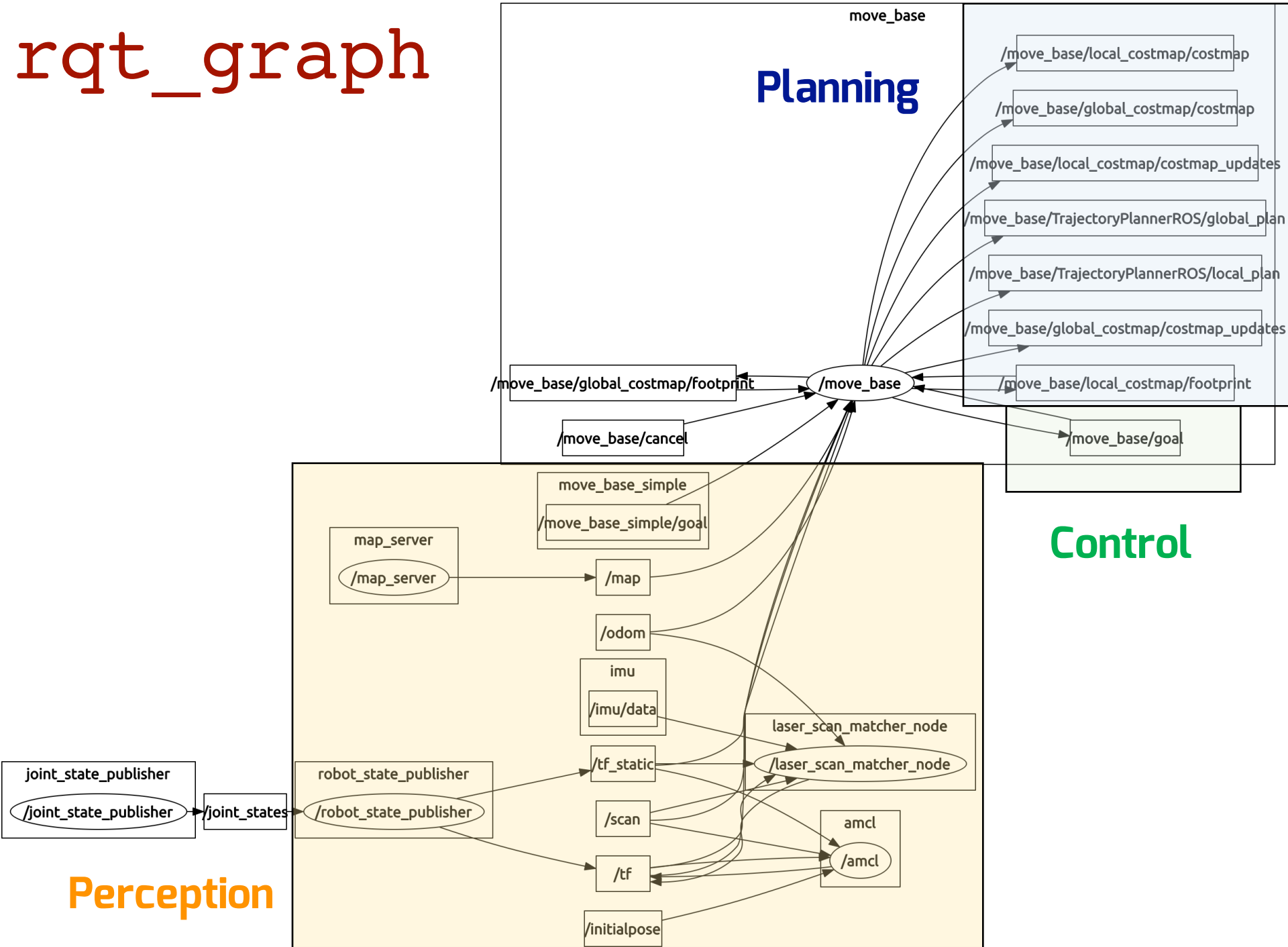
Current View	Orbit (rviz)
Near Clip ...	0.01
Target Fra...	map
Distance	9.52754
Yaw	3.95541
Pitch	0.364797
Focal Point	1.2137; 1.9063; 0...

Status Bar:

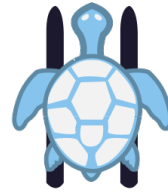
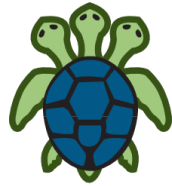
ROS Time: 1407551277.42 ROS Elapsed: 251.65 Wall Time: 1407551277.46 Wall Elapsed: 251.65

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

rqt_graph



But what about the turtles ?



Next time

TurtleSim

Keyboard Control

Practice Session 1