



Introduction to Robotics

Introduction to the Robot Operating System (ROS)

Rodrigo Ventura

Institute for Systems and Robotics

Instituto Superior Técnico, Universidade de Lisboa

rodrigo.ventura@isr.tecnico.ulisboa.pt



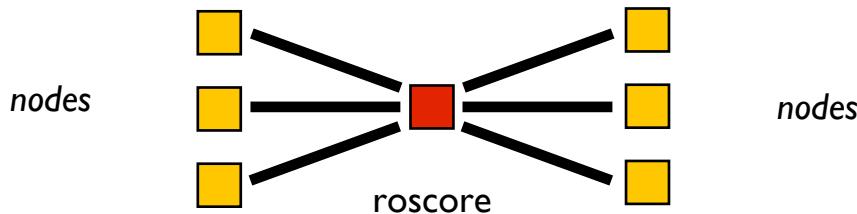
What is ROS?

- **ROS = Robot Operating System**
- Framework for robot software development providing operating system-like functionality
- Originated at Stanford Artificial Intelligence Lab, then further developed at Willow Garage
- Works quite well in Linux Ubuntu, but there are bindings to Java, C#, and can be tunneled via websockets
- Large user base; getting widespread use
- ROS users forum: <http://answers.ros.org>



Basic concept #1: Node

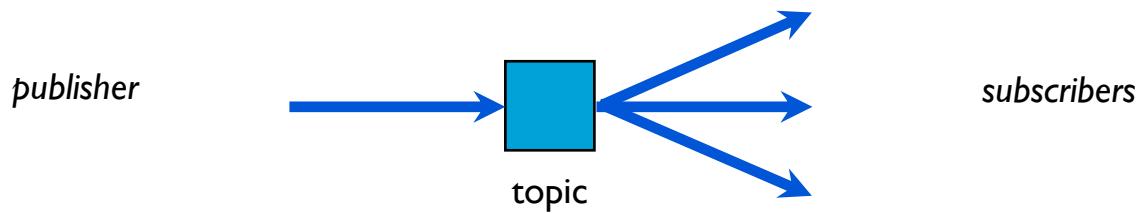
- Modularization in ROS is achieved by separated operating system processes
- **Node** = a process that uses ROS framework
- Nodes may reside in different machines transparently
- Nodes get to know one another via roscore



- roscore acts primarily as a name server
- Nodes use the roscore running in localhost by default overridden by the environment variable `ROS_MASTER_URI`

Basic concept #2: Topic

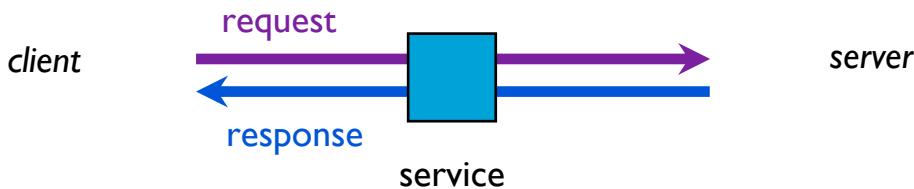
- **Topic** is a mechanism to send messages from a node to one or more nodes
- Follows a publisher-subscriber design pattern



- **Publish** = to send a message to a topic
- **Subscribe** = get called whenever a message is published
- Published messages are broadcast to all Subscribers
- Example: LIDAR publishing scan data

Basic concept #3: Service

- **Service** is a mechanism for a node to send a request to another node and receive a response in return
- Follows a request-response design pattern



- A service is called with a request structure, and in return, a response structure is returned
- Similar to a Remote Procedure Call (RPC)
- Example: set location to a localization node

Message types

All messages (including service requests/responses) are defined in text files

Contents of sensor_msgs/msg/LaserScan.msg:

```

Header header           # timestamp in the header is the acquisition time of
# the first ray in the scan.
#
# in frame frame_id, angles are measured around
# the positive Z axis (counterclockwise, if z is up)
# with zero angle being forward along the x axis

float32 angle_min      # start angle of the scan [rad]
float32 angle_max      # end angle of the scan [rad]
float32 angle_increment # angular distance between measurements [rad]

float32 time_increment  # time between measurements [seconds] - if your scanner
# is moving, this will be used in interpolating position
# of 3d points
float32 scan_time       # time between scans [seconds]

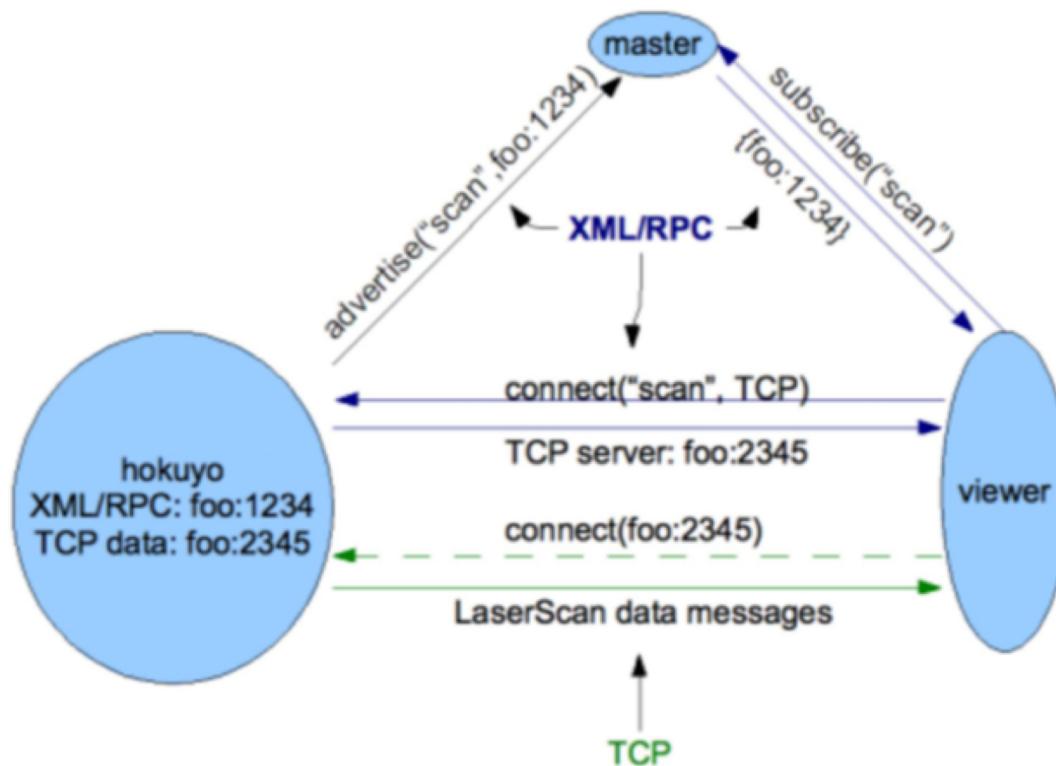
float32 range_min       # minimum range value [m]
float32 range_max       # maximum range value [m]

float32[] ranges        # range data [m] (Note: values < range_min or > range_max should be
discarded)
float32[] intensities   # intensity data [device-specific units]. If your
# device does not provide intensities, please leave
# the array empty.

```



Topics internals



Development

- Two major languages are supported:
 - C++
 - Python
- ROS provides a portable build system (catkin, replacing rosbuild)
- **Package** = encapsulation of sources, data files, and building files
- The code reuse units in ROS are packages
- A large variety of packages can be found on the web
- examples: sensor drivers, simulators, SLAM, image processing, etc.

Command line tools

rosnode is a command-line tool for printing information about ROS Nodes.

Commands:

rosnode ping	test connectivity to node
rosnode list	list active nodes
rosnode info	print information about node
rosnode machine	list nodes running on a particular machine or list machines
rosnode kill	kill a running node
rosnode cleanup	purge registration information of unreachable nodes

Command line tools

rostopic is a command-line tool for printing information about ROS Topics.

Commands:

```
rostopic bw      display bandwidth used by topic
rostopic echo    print messages to screen
rostopic find    find topics by type
rostopic hz      display publishing rate of topic
rostopic info    print information about active topic
rostopic list    list active topics
rostopic pub     publish data to topic
rostopic type    print topic type
```



Command line tools

rosservice is a command-line tool for printing information about ROS Services.

Commands:

```
rosservice args print service arguments
rosservice call call the service with the provided args
rosservice find find services by service type
rosservice info print information about service
rosservice list list active services
rosservice type print service type
rosservice uri print service ROSRPC uri
```

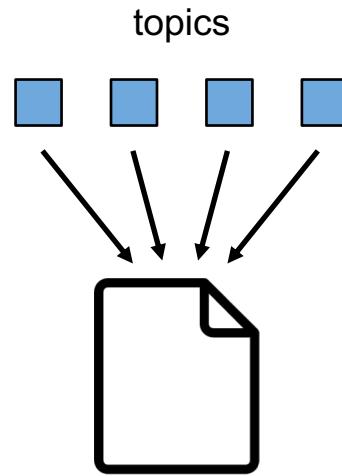
Command line tools

rosbag is a command-line tool for manipulating log files (a.k.a. bags)

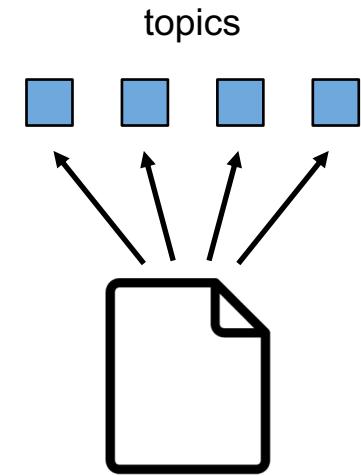
Available subcommands:

- check
- compress
- decompress
- filter
- fix
- help
- info
- play
- record
- reindex

rosbag record ...



rosbag play ...



Useful ROS facilities

- **Parameters:** repository of parameters
 - Loading from files (formatted in YAML)
 - Dynamic update
 - Command-line utility: rosparam

params.yaml

```
course_name: "SAut"

robot1:
    name: "Calvin"
    height: 0.5

robot2:
    name: "Hobbes"
    height: 1.0
```

```
$ rosparam load params.yaml
$ rosparam list
/course_name
/robot1/height
/robot1/name
/robot2/height
/robot2/name
[...]
$ rosparam get course_name
SAut
$ rosparam get /robot2/name
Hobbes
```

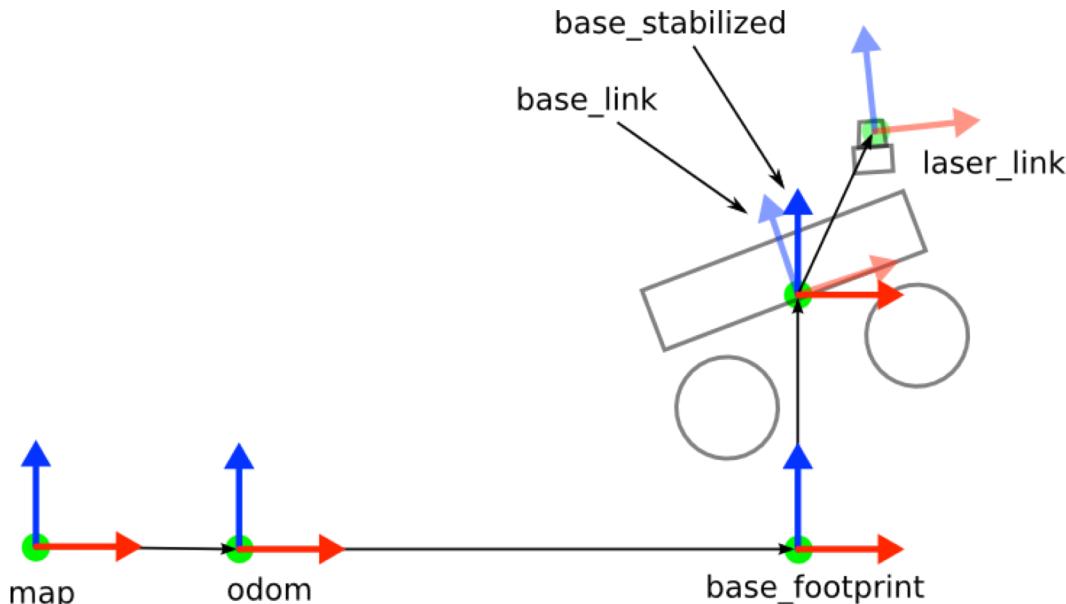
Useful ROS facilities

- **Launch files:** XML file specifying the launch of multiple nodes
 - Loading of parameters
 - Remapping topic names, parameters, etc.
 - Multiple machines support
 - Command-line utility: roslaunch

```
<?xml version="1.0"?>
<launch>
    <arg name="map" default="$(find scout_maps)/isr8-v05cr.yaml"/>
    <param name="map" type="string" value="$(arg map)"/>
    <rosparam file="$(find scout_config)/mbot.yaml"/>
    <include file="amcl.launch"/>
    <node name="navigation" pkg="scout_navigation" type="navigator">
        <param name="~guidance_method" type="string" value="fmm"/>
        <param name="~platform_mode" type="string" value="omni"/>
    </node>
</launch>
```

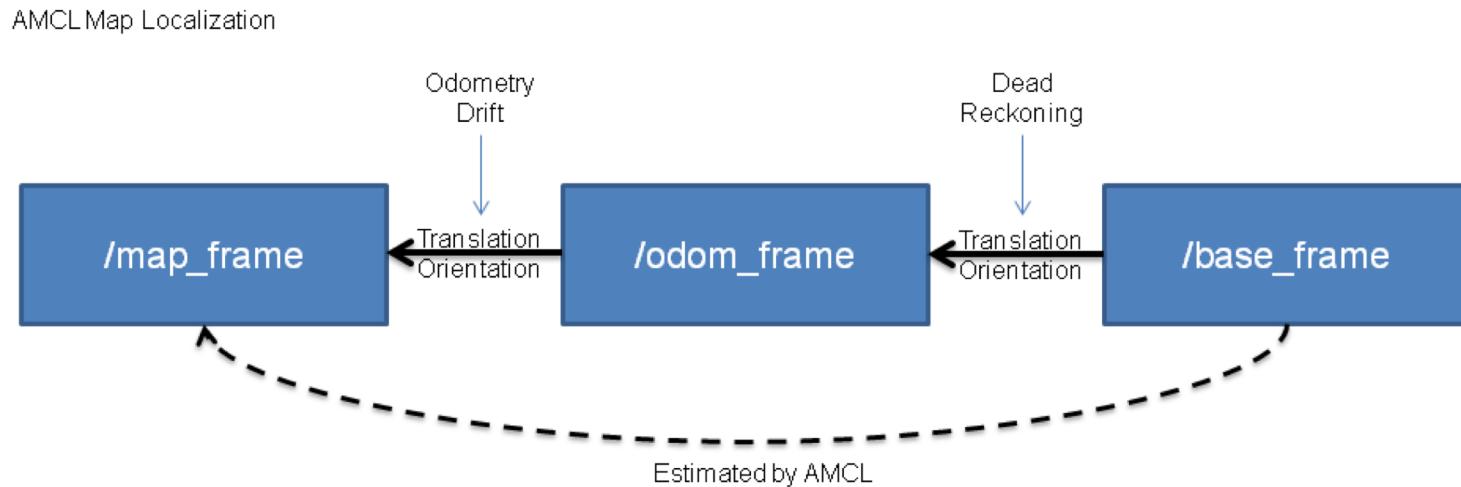
Useful ROS facilities

- **TF framework:** represents geometric transformations in 3D, position and orientation (6-DoF)



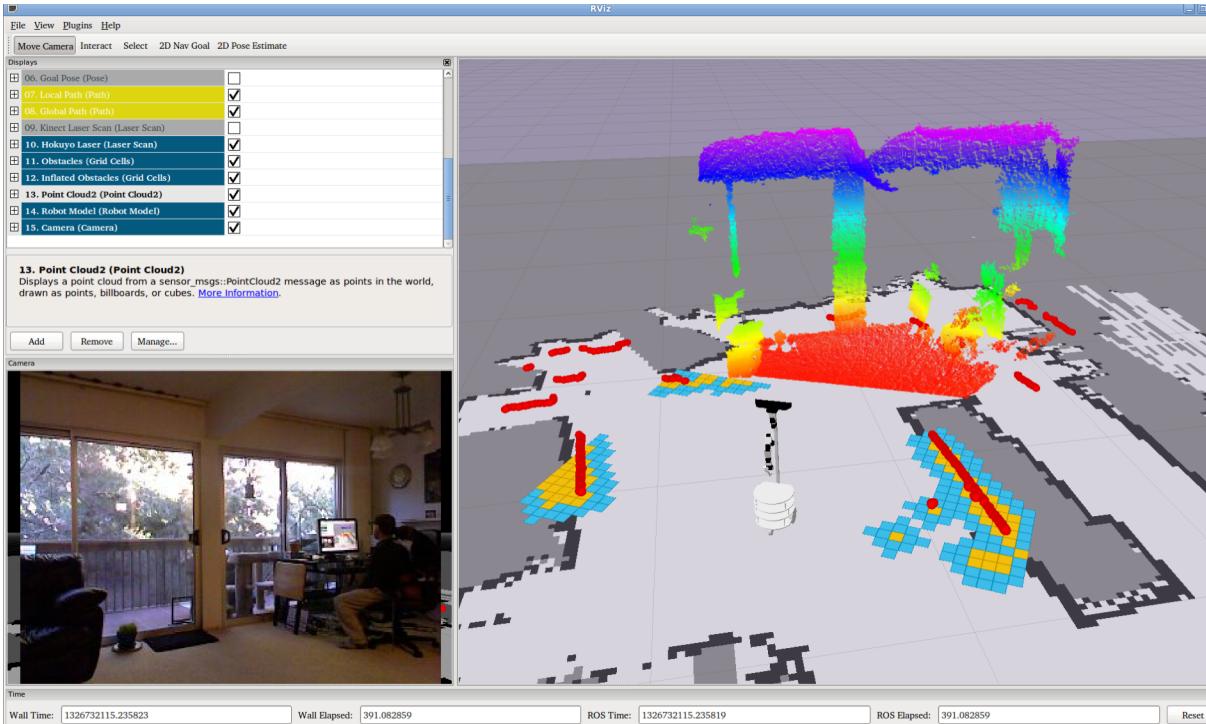
Useful ROS facilities

- **TF** framework: *de facto* standard frame assignment:



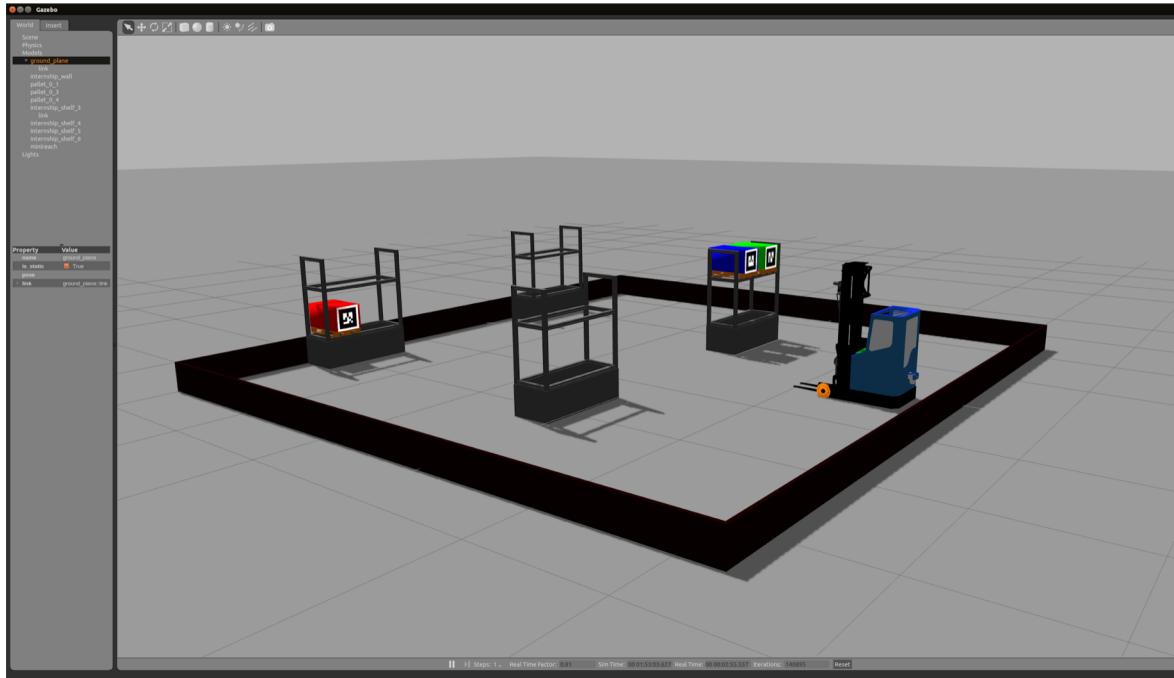
Useful ROS facilities

- **RVIZ:** visualisation framework



Useful ROS facilities

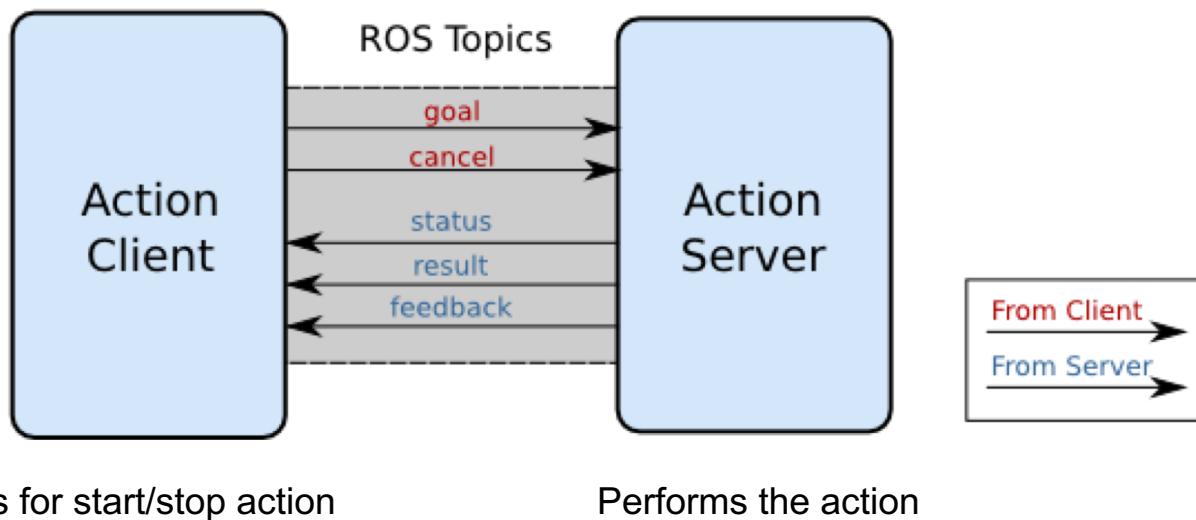
- **Gazebo:** physics simulation framework



Useful ROS facilities

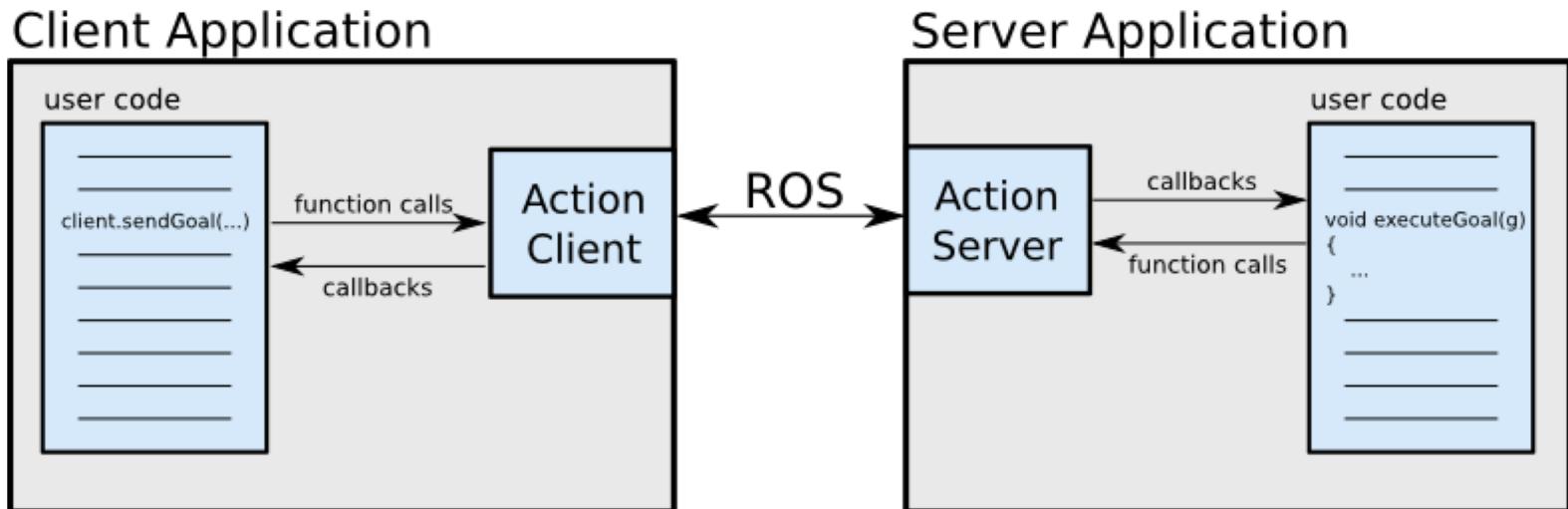
- **Actionlib** framework: state-full scheme to manage action execution

Action Interface



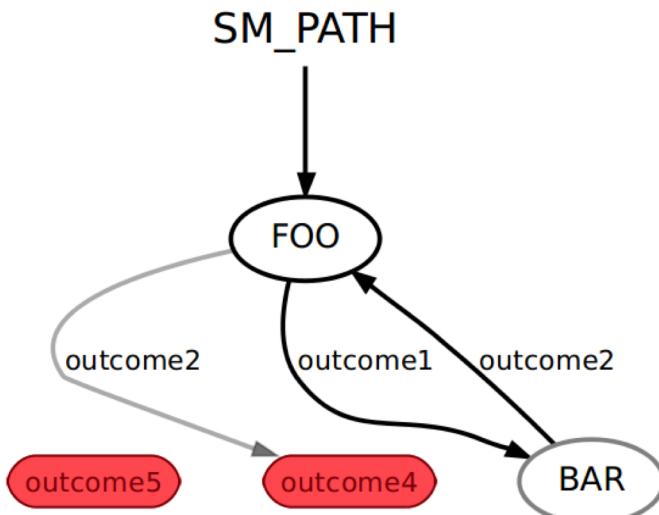
Useful ROS facilities

- **Actionlib** framework: state-full scheme to manage action execution



Useful ROS facilities

- **SMACH** framework: Finite State Machine (FSM) executor fully integrated into ROS
Ingredients: states, transitions, and outcomes



Useful ROS facilities

- **SMACH** framework:
 - Types of states:
 - MonitorState -- subscribes to topic, waits while condition True
 - ConditionState -- polls a callback function, waits until True
 - SimpleActionState -- calls actionlib action and can be a container
 - Types of containers:
 - StateMachine -- finite state machine
 - Concurrence -- all states run in parallel (split/join logic)
 - Sequence -- StateMachine with linear sequence of states

Useful ROS facilities

- Other off-the-shelf packages:
 - **Gmapping**: creates occgrid maps from laser data
 - **AMCL**: localizes on occgrid maps using laser data
 - **Move_base**: path planning and guidance with obstacle avoidance using laser data
 - **MovIt**: trajectory planner for robotic arms
 - **Octomap**: creates 3D occupancy maps using RGB-D
 - **ROSPlan**: integrates classical planner into ROS