ELEG4701

Intelligent Interactive Robot Practice

Lab 5: Roslaunch and Service/Client

Jiewen Lai

Research Assistant Professor

EE, CUHK

jiewen.lai@cuhk.edu.hk

Today's Agenda

Lecture

1. Review on ROS topic & ROS Service/Client
2. Introduction to the 'roslaunch'
3. Write your first 'roslaunch' file
4. Create a simple '.srv' file
5. Write a ROS service/client node

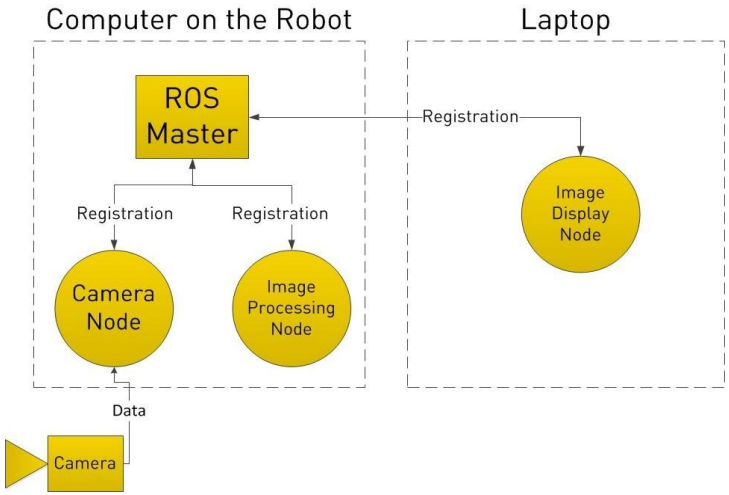
Tutorial

1. Lab Sheet 5

Review on ROS Topic & Service/Client

Node, ROS Master (Recap)

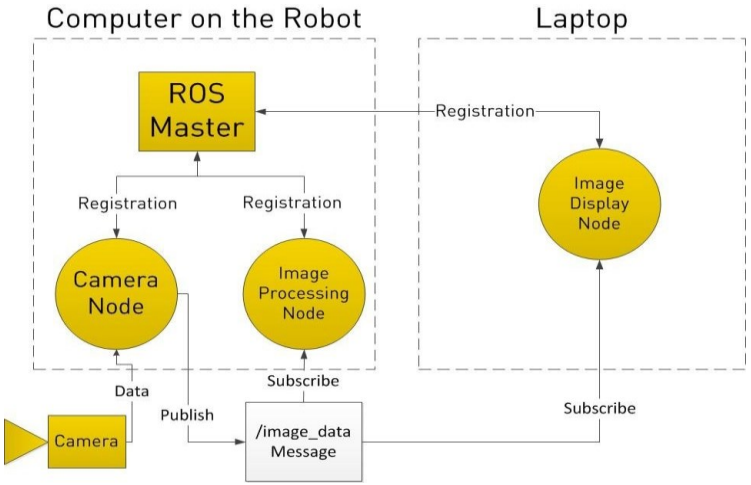
- **Node** – Execution Unit
  - Processes that perform specific tasks, independently run executables
  - Different nodes can use **different programming languages** and can be distributed to run on different hosts
  - The name of the node must be unique in the system



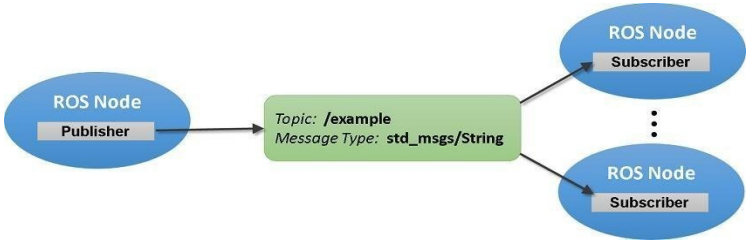
- **ROS Master** – Control center
  - Provide naming and registration services for nodes
  - Track and record topic/service communications to assist nodes in finding each other and establishing connections
  - Provides a parameter server that nodes use to store and retrieve runtime parameters

Topic, Message (Recap)

- **Topic** – *Asynchronous* communication
  - Important bus used to transfer data between nodes
  - Using the **publish/subscribe model**, data is transferred from publisher to subscriber, and publishers or subscribers of the same topic may not be unique

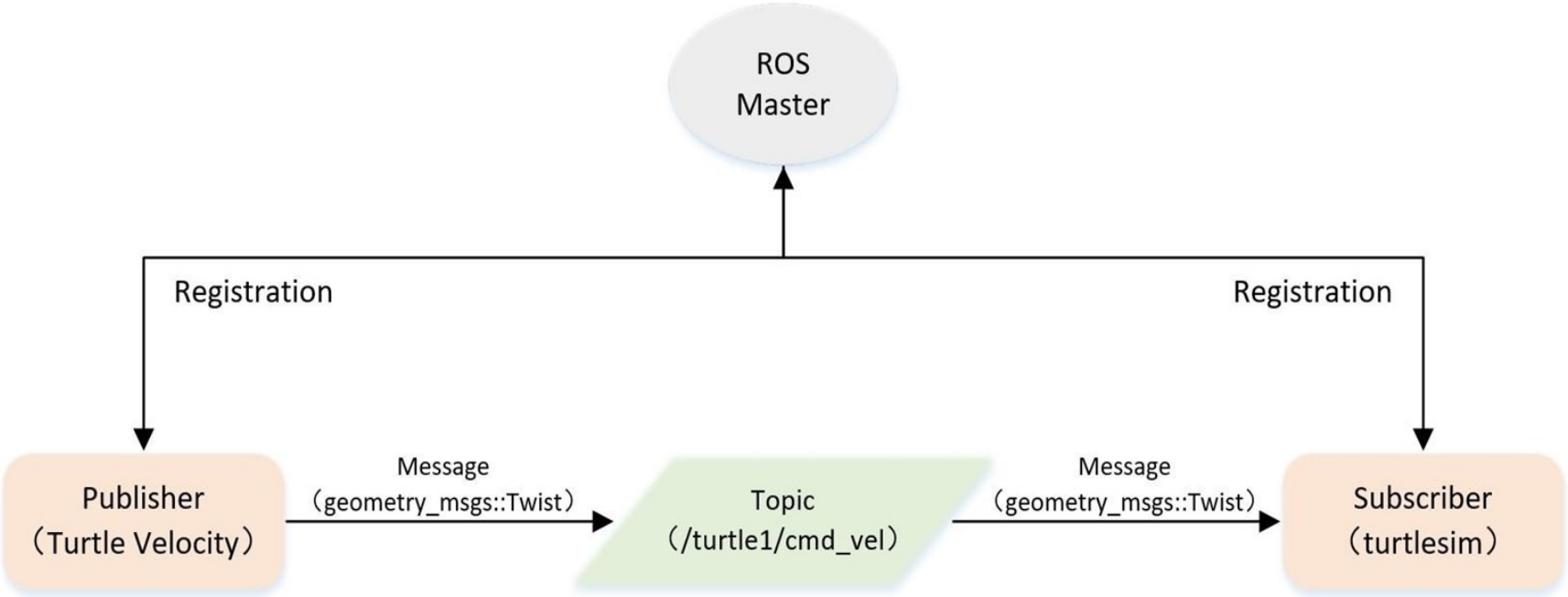


- **Message** – Topic data
  - Has certain types and data structures, including the standard types provided by ROS and user-defined types
  - Use **programming language-independent** `.msg` file define message, the programming process generates the corresponding code files



Topic Model (publish/subscribe)

Topic Model (Recap)



Topic Model (publish/subscribe)

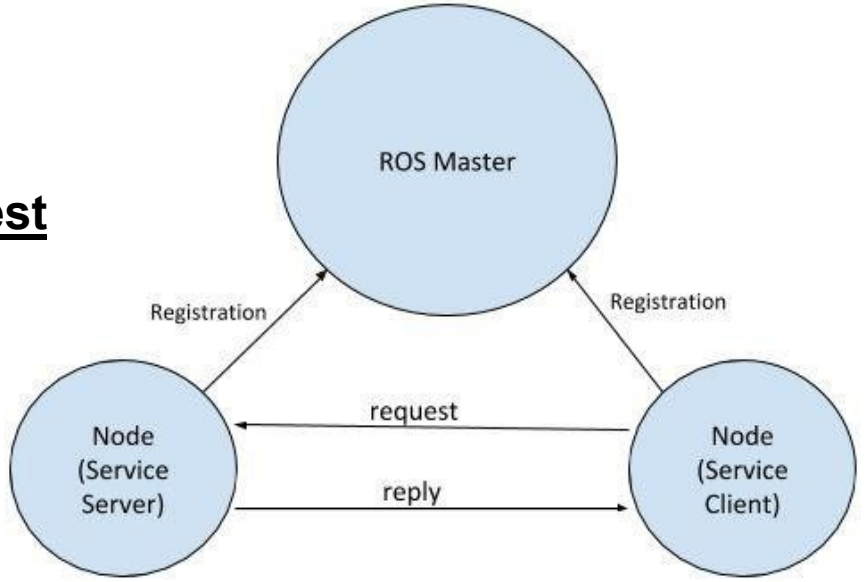
ROS Service/Client

Service

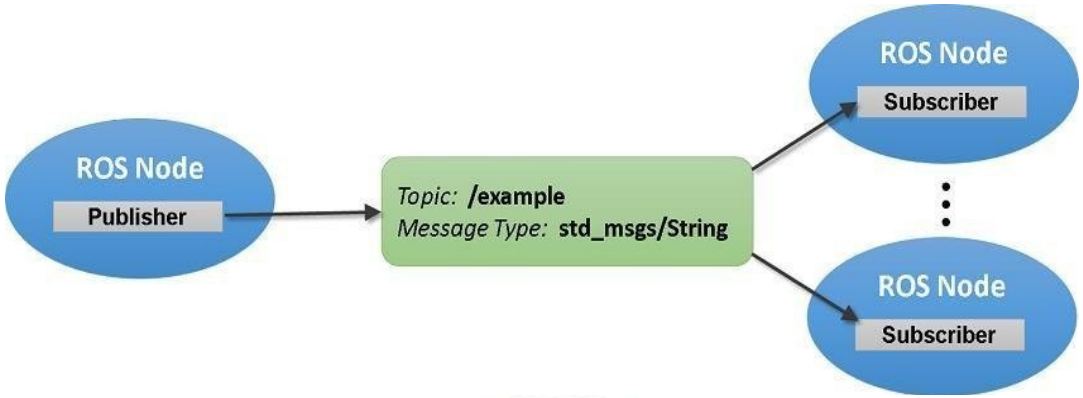
- Services are another way that nodes can communicate with each other
- Services allow nodes to **send a request** and **receive a response**

srv – Service data type

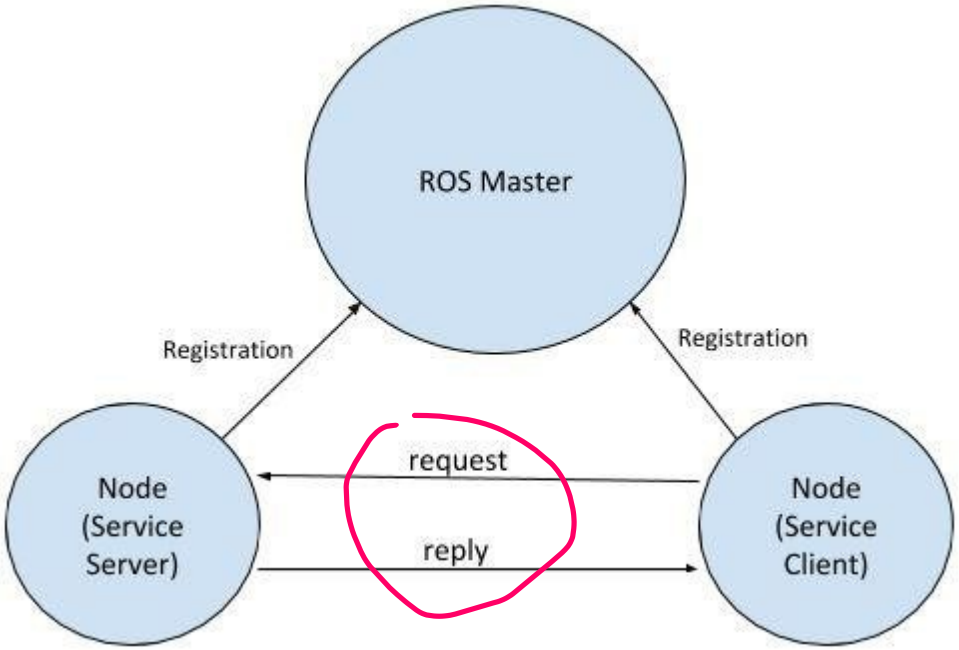
- '.srv' file describe a service
- It is composed of two parts: **a request** and **a response**



Comparing: ROS Topic & ROS service/client



Topic



Service

Introduction to the ‘roslaunch’

- **roslaunch** is a tool for easily launching **multiple ROS nodes**
- **roslaunch** takes in one or more XML configuration files (with the ‘.launch’ extension) that specify the parameters to set the nodes to launch, as well as the machines that they should be run on

To use the roslaunch command:

```
$ roslaunch <package_name> <example.launch>
```

or

```
$ roscd <package_name>/launch

$ roslaunch <example.launch>
```

Introduction to the ‘roslaunch’

- **roslaunch** was designed to fit the ROS architecture of complexity via composition.
- Understanding roslaunch’s architecture will give you better insight into how to construct your .launch files.
- ‘**Tag**’ in roslaunch file:

<launch>

<node>

<machine>

<include>

<remap>

<env>

<param>

<rosparam>

<group>

<test>

<arg>

Introduction to the ‘roslaunch’ <Tag>

<launch>

The <launch> tag is the root element of any roslaunch file. Its sole purpose is to act as a container for the other elements

<node>

The <node> tag specifies a ROS node that you wish to have launched. This is the most common roslaunch tag as it supports the most important features: bringing up and taking down nodes.

Example:

```
<node name="bar1" pkg="foo_pkg" type="bar"/>
```

Launch bar1 node from the foo_pkg package in bar type

<include>

The <include> tag enables you to import another roslaunch XML file into the current file. It will be imported within the current scope of your document, including <group> and <remap> tags.

Introduction to the 'roslaunch' <Tag>

<remap>

Remapping allows you to "trick" a ROS node so that when it thinks it is subscribing to or publishing to /some_topic, it is actually subscribing to or publishing to /some_other_topic, for instance.

Example:

```
<remap from="/different_topic" to="/needed_topic" />
```

Remapping makes the new node ends up subscribing to /needed_topic when it thinks it is subscribing to /different_topic

<param> & <rosparam>

The <param> tag defines a parameter to be set on the Parameter Server.

The <rosparam> tag enables the use of rosparam YAML files for loading and dumping parameters from the ROS Parameter Server.

Introduction to the ‘roslaunch’ <Tag>

<arg>

The <arg> tag allows you to create more re-usable and configurable launch files by specifying values that are passed via the command line, passing in via an <include>, or declared for higher-level files.

For the detailed API and explanation, please refer to:

https://wiki.ros.org/roslaunch/XML

Introduction to the ‘roslaunch’ <Tag>

A roslaunch file example:

```
<launch>
  <node name="talker" pkg="rospy_tutorials" type="talker" />
</launch>
```

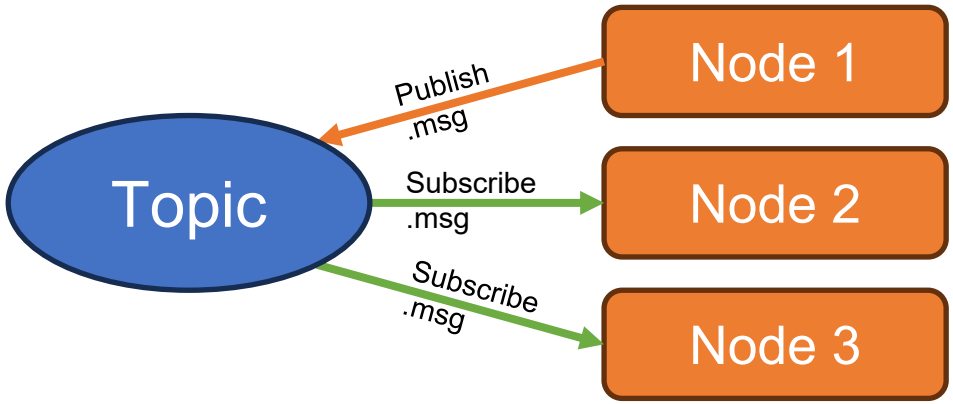
Introduction to the ‘roslaunch’ <Tag>

A more complicated roslaunch file example:

```
<launch>
  <!-- a basic listener node -->
  <node name="listener-1" pkg="rospy_tutorials" type="listener" />
  <!-- start listener node in the 'wg1' namespace -->
  <node ns="wg1" name="listener-wg1" pkg="rospy_tutorials" type="listener" respawn="true" />
  <!-- start a group of nodes in the 'wg2' namespace -->
  <group ns="wg2">
  <!-- remap applies to all future statements in this scope. -->
   <remap from="chatter" to="hello"/>
   <node pkg="rospy_tutorials" type="listener" name="listener" args="--test" respawn="true" />
   <node pkg="rospy_tutorials" type="talker" name="talker">
  <!-- set a private parameter for the node -->
    <param name="talker_1_param" value="a value" />
  <!-- nodes can have their own remap args -->
    <remap from="chatter" to="hello-1"/>
  <!-- you can set environment variables for a node -->
    <env name="ENV_EXAMPLE" value="some value" />
   </node>
  </group>
</launch>
```

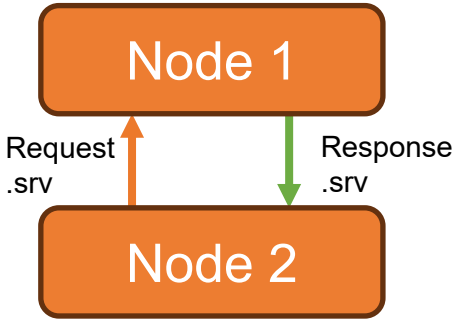
Both XML and HTML—along with other languages like LaTeX, SVG, Markdown, and SGML—belong to a family of programming languages called ***markup languages***.

ROS Node and Topic



.msg is where we define the data type for ROS topic communication

ROS Node and Service



Request/Response interactions

.srv is where we define the data type for ROS service communication

rosservice

- Services are another way that nodes can communicate with each other
- Services allow nodes to **send a request** and **receive a response**

Commands for debugging

```
rosservice list
rosservice call
rosservice type
rosservice find
rosservice uri
```

```
#print information about active services
#call the service with the provided args
#print service type
#find services by service type
#print service ROSRPC uri
```

We used it to **spawn the second turtle** before

Next

- Finish Lab sheet 5
- Tutorial for lab sheet 5