Tutorial 2 Outline

• ROS Graph
• ROS Services and Parameters
  • Turtlesim – service application (spawn a new turtle)
• Installing ROS Packages
  • Using apt manager
  • Cloning from repository
• Gazebo
  • Introduction - Installation
  • Testing – Kinematic and Dynamic Simulation
  • Understanding Gazebo – ROS Filesystem
  • URDF – XACRO files
**rqt_graph**

- rqt_graph is a useful tool that provides a GUI plugin for visualizing the ROS computation graph. Namely, it creates a **dynamic** graph of what’s going on in the system (i.e. the ROS nodes that are currently running, as well as the ROS topics that connect them)
- Its components are made generic so that other packages where you want to achieve graph representation can depend upon it.
- Command:
  
  ```
  rosrun rqt_graph rqt_graph
  ```
rqt_graph – example

Turtlesim simulation

$ roscore
$ rosrun turtlesim turtlesim_node
$ rosrun rqt_graph rqt_graph
$ rosrun turtlesim turtle_teleop_key
**rqt_graph – example**

Turtlesim application

$ rosrun turtlesim turtlesim_node __name:=A
$ rosrun turtlesim turtlesim_node __name:=B
$ rosrun turtlesim turtle_teleop_key __name:=C
$ rosrun turtlesim turtle_teleop_key __name:=D
Services

- ROS supports the concept of "remote procedure call" (in contrast to local function call), in the form of ROS services. With ROS services, calling a function in another node is as easy as calling local functions.
- One time actions. Use services when your program can't continue until it receives the result from the service.

rosservice list  print information about active services
rosservice call  call the service with the provided args
rosservice type  print service type

$ rosservice list
$ rosservice type [service] | rossrv show
$ rosservice [service] [args]
Parameters

- rosparam allows you to store and manipulate data on the ROS Parameter Server. The Parameter Server can store integers, floats, boolean, dictionaries, and lists.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rosparam list</td>
<td>list parameter names</td>
</tr>
<tr>
<td>rosparam set</td>
<td>set parameter</td>
</tr>
<tr>
<td>rosparam get</td>
<td>get parameter</td>
</tr>
<tr>
<td>rosparam delete</td>
<td>delete parameter</td>
</tr>
</tbody>
</table>

```bash
$ rosparam list
$ rosparam set [param_name]
$ rosparam get [param_name]
```
Application on Turtlesim

Spawn a new turtle:

$ rosservice list
$ rosservice type /spawn | rossrv show
$ rosservice call /spawn 2 7 0.2 "turtle_tutorial"
Recall: What is a package?

- All the files that a specific ROS program contains; all its cpp files, python files, configuration files, compilation files, launch files, and parameters files.

- Generally all those files in the package are organized with the following structure:
  - **launch** folder: Contains launch files
  - **src** folder: Source files (cpp, python)
  - **CMakeLists.txt**: List of cmake rules for compilation
  - **package.xml**: Package information and dependencies

Not always
Installing ROS package

Using Ubuntu’s package manager

• The **apt-get** command is a powerful command-line tool, which works with Ubuntu's Advanced Packaging Tool (APT) performing such functions as downloading and installation of new software packages, updating, etc.

• This method is used for installing released ROS packages (packages verified by ROS developers) and install the package’s necessary dependencies.

• Command:

```
sudo apt-get install ros-<ros_distro>-<package-name>
```

• Example

```
sudo apt-get install ros-kinetic-urdf
```
Installing ROS package

Installing from source code

• If the package is not released, you will need to install it from source code. Find out where the code is hosted (mostly github), and install the package in the src folder from your workspace directory (~/catkin_ws/src)


• Command:

```bash
$ cd ~/catkin_ws/src
$ git clone <address> / git clone -b <branch> <address>
```

• Example:

```
$ cd ~/catkin_ws/src
$ git clone -b kinetic-devel https://github.com/ros/urdf.git
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3
$ cd ~/catkin_ws
$ catkin_make
```
Installing ROS package

Installing from source code

- Problem: You have to take care of the dependencies by yourself.
- Whenever you see "Could not find a configuration file for package <package_name>" it means this package is missing and is needed for compiling your code.
- Possible solution: Use `rosdep` command, which will automatically try to find all the dependencies of your package and install them.

IMPORTANT NOTE

- Sometimes the `<package-name>` or `<address>` arguments correspond to a directory which contains more than one package.
Gazebo simulator

- Goal: Best possible substitute for physical robot

- Architecture

- Advantages
  - Design and testing of robot’s components and control
  - Software testing and verification (controllers)
  - Save time and money

- Installation: Built-in along with the ROS desktop-full.
Testing Gazebo

- Gazebo runs two executables: Gazebo server (simulation process) and Gazebo client (Gazebo GUI)

- Add a square block and a sphere using the upper tool bar
- Right click on the sphere
- Select Apply Force/Torque
- Choose a value for the torque and force and select apply.
- Observe kinematics and dynamics simulation
Understanding ROS-Gazebo Filesystem

Controllers .yaml files

Robot modelling .urdf, .xacro, .gazebo files

Simulation Environment .world files

Notice that each one of this folders is a different package and usually they are located inside the robot’s name folder. Packages’ names are recommended but not mandatory.
World

```xml
<?xml version="1.0" ?>
<sdf version="1.4">
  <world name="default">
    <include>
      <uri>model://ground_plane</uri>
    </include>
    <!-- Global light source -->
    <include>
      <uri>model://sun</uri>
    </include>
  </world>
</sdf>

<?xml version="1.0" ?>
<sdf version="1.4">
  <world name="default">
    <include>
      <uri>model://ground_plane</uri>
    </include>
    <!-- Global light source -->
    <include>
      <uri>model://sun</uri>
    </include>
    <!-- Include model of gas station-->
    <include>
      <uri>model://gas_station</uri>
      <name>gas_station</name>
      <pose>-2.0 7.0 0 0 0 0</pose>
    </include>
  </world>
</sdf>
```
The Universal Robotic Description Format (URDF) is an XML file format used in ROS to describe all elements of a robot. To use a URDF file in Gazebo, some additional simulation-specific tags must be added to work properly with Gazebo.

**URDF**
- Specify the kinematic and dynamic properties
- Tags: link, joint, transmission
- Order in the file does not matter

**XACRO**
- XML Macro Language used for URDF simplification
- Reduce redundancy and increase modularity
- Use parametrization (Use parameters for lengths and links and math for origin and inertia calculation)
Link and joint representation

<!-- Front right End arm -->
<link name="link3">
  <visual>
    <origin xyz="0 0 -0.1" rpy="0 0 0"/>
    <geometry>
      <box size="0.025 0.025 0.025"/>
    </geometry>
    <material name="orange"/>
  </visual>
</link>

<joint name="joint3" type="continuous">
  <parent link="link2"/>
  <child link="link3"/>
  <origin xyz="0 0.05 -0.07" rpy="0 0 0"/>
  <axis xyz="0 1 0"/>
</joint>

<!-- Front right Upper arm -->
<link name="link2">
  <visual>
    <origin xyz="0 0 0.075" rpy="0 0 0"/>
    <geometry>
      <box size="0.05 0.05 0.15"/>
    </geometry>
    <material name="red"/>
  </visual>
</link>
Thanks for your attention