Introduction to ROS

49274 Advanced Robotics

Overview

- Robot Operating System provides:
  - Process management
  - Hardware abstraction
  - Tools
- ROS runs on Ubuntu Linux
- The FEIT computer labs use Red Hat Linux, but an Ubuntu container is available
- ROS nodes can be written in C++ or Python

Using ROS on the FEIT computers

- First make sure you are in Linux: *Ctrl+Alt* then *Ctrl+Alt+F1*
- Enter your student number in the *Username* field
- Enter your password
- Open a terminal:

  Applications -> System Tools -> Terminal

- Start a shell in the Singularity container:

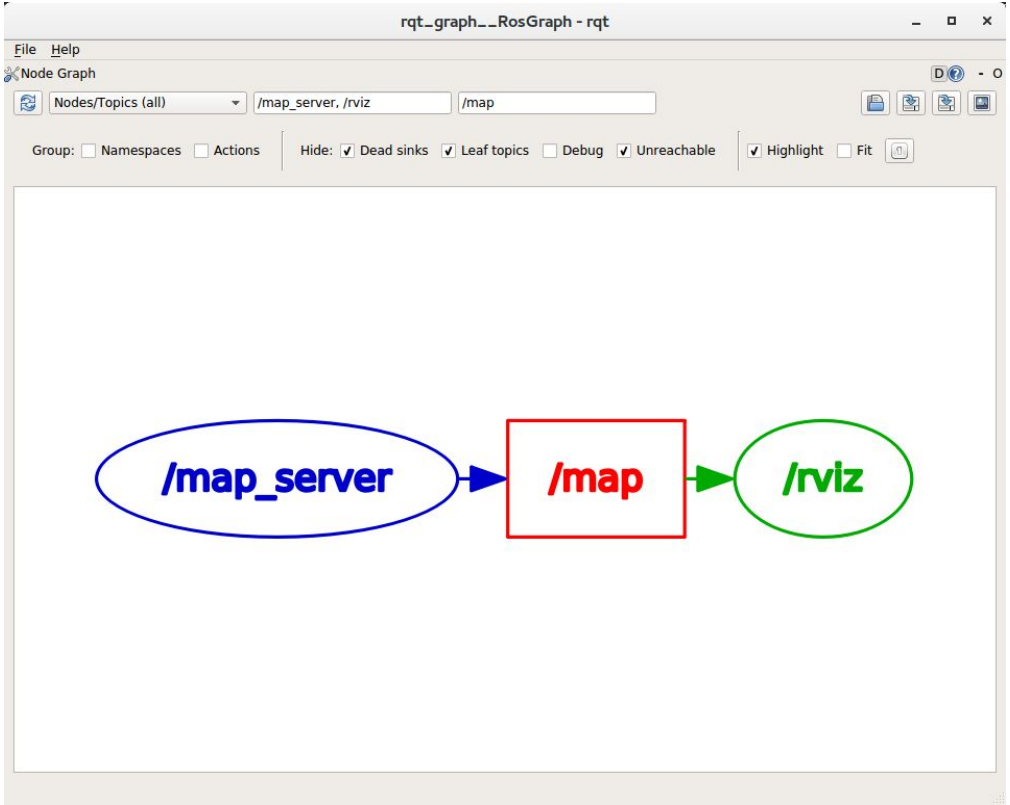
```
singularity shell /images/singularity_containers/ros-melodic-ar.sif
```

- The Singularity image is on all the Linux computers on level B1

Overview of ROS

- Node: an executable that uses ROS to communicate with other nodes
- Message: the data structure used for communication
- Topic: The name of the channel used to pass messages
- More information: ROS Tutorials: Understanding Nodes

Overview of ROS



Overview of ROS

- A node can publish messages on a topic
- Another node can subscribe to a topic to receive those messages
- A node can both publish and subscribe to different topics
- A topic has only one type of message
- More information: ROS Tutorials: Understanding Topics

Node and topic naming

- ROS nodes and topics can be nested, like files within a directory
- Like the Linux file system, “/” is the root of the hierarchy
- They can also be referred to by absolute or relative paths

Messages

- Messages can contain variables and/or other message data structures
- A “`[]`” suffix indicates that the data is a vector
- E.g. *nav_msgs/Path* contains a *std_msgs/Header* variable named header and a vector of *geometry_msgs/PoseStamped* variables named poses.
- How do you access an element in a *nav_msgs/Path* message named path?
  - `path.poses[0].pose.position.x`

Starting ROS

- The ROS master (that manages other nodes) can be started with:

```
roscore
```

- `roslaunch` will start nodes listed in a launch file, and it will start the ROS master automatically:

```
roslaunch <package_name> <launch_file>
```

Useful ROS commands

Command	**Description**
`rosnode list`	List running nodes
`rosnode info <node_name>`	Show information about a node
`rosrun <package_name> <node_name>`	Start a node
`rosnode kill <node_name>`	Stop a node (you can also *Ctrl+C* in the terminal the node was started from)

Useful ROS commands

Command	Description
`rostopic list`	List active topics
`rostopic info <topic_name>`	Show information about a topic
`rostopic hz <topic_name>`	Show the publish frequency of a topic
`rostopic echo <topic_name>`	Show the messages published on a topic
`rostopic type <topic_name>`	Show the message type of a topic
`rosmsg show <message_type>`	Show the structure of a message type

Catkin

- Catkin is the build system infrastructure for ROS
- First you need to make a workspace:

```
mkdir -p catkin_ws/src
cd catkin_ws/src
catkin_init_workspace
```

- Packages are built using `catkin_make` (you need to run this in the top level directory of the workspace):

```
cd ../
catkin_make
```

Catkin

Workspaces and packages have a specific layout:

```
catkin_ws/
↳    build/
     devel/
     src/
     ↳    CMakeLists.txt
          package_name/
          ↳    CMakeLists.txt
               package.xml
               src/
               ↳    node_name.cpp
               include/
               ↳    node_name.h
```

Additional resources

- Clearpath Robotics ROS Tutorials
- ROS Wiki: Tutorials
- ROS tutorials (YouTube playlist)
- Programming for Robotics (YouTube playlist)

Writing to the console

- rosconsole provides console output and logging:
- `ROS_INFO` can be used in the same way as `printf`:

```
ROS_INFO("Variable: %i\n", x);
```

- `ROS_INFO_STREAM` is another option:

```
ROS_INFO_STREAM("Variable: " << x);
```

Building assignment 1 part 2

- Create a Catkin workspace
- Copy the “particle_filter_localisation” folder into “catkin_ws/src”
- If you are using the UTS computer labs, first run:

```
singularity shell /images/singularity_containers/ros-melodic-ar.sif
```

- Compile all packages/nodes in the workspace by running (in “catkin_ws/”):

```
catkin_make
```

Running assignment 1 part 2

- If you are using the UTS computer labs, first run:

```
singularity shell /images/singularity_containers/ros-melodic-ar.sif
```

- Then in the shell:

```
source ~/catkin_ws/devel/setup.bash
```

- Run “roscore”, “particle_filter_localisation” and other nodes:

```
roslaunch particle_filter_localisation particle_filter_localisation.launch
```

- Use Ctrl+C to exit a running program in the terminal

Running assignment 1 part 2

- Control the robot with the keyboard by opening another terminal and running:

```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

- If you are using the UTS computer labs run (**the command is one line**):

```
singularity exec /images/singularity_containers/ros-melodic-ar.sif rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

- Use Ctrl+C to exit a running program in the terminal

Running assignment 1 part 2

- The launch file starts a “roscore”, which is terminated when you kill quit the launch file. You will need to restart “teleop_twist_keyboard” whenever you restart “particle_fitler_localisaiton.launch”
- Avoid this by starting a “roscore” before “particle_fitler_localisaiton.launch”:

```
roscore
```

- On the FEIT computers (**the command is one line**):

```
singularity exec /images/singularity_containers/ros-melodic-ar.sif rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```