Hands on!

ROS: First Encounter

8 December 2015

Preliminary steps

Normally, ROS should be installed on your system. Test it quickly by openning two terminals. In the first one, run `roscore` and in the second one, run `rosrun rviz rviz`.

This should open RViz, the 3D visualization tool that comes with ROS.

> If this does not work, come and see me!

For today's tutorial, we will all use the same `roscore` (*i.e.* we will all be part of the same ROS network). So:

- Kill the `roscore` process you've just launched,
- Type: `export ROS_MASTER_URI=http://<ip on the whiteboard>:11311`

> **Note**
>
> You must set `ROS_MASTER_URI` in every terminal used to start a ROS application! Do not forget to type it when you open a new terminal. Alternatively, you can add it to your `.bashrc` file to automatically set it for every terminal.

Part I

On the way to the treasure

Note

As you will discover, this document does not provide much details on how to complete the below 'Goals': try, experiment, find code examples online (in particular here: `http://wiki.ros.org/ROS/Tutorials`), ask your neighbour, and blame my laziness!

Goal 1 – Find your robot

Now that everything is setup on your system, find out the colour of your robot!

Explore and test the core ROS tools (`rosnode`, `rostopic`, `rosservice`, etc.), and find a way to 'give a hug' to your robot.

No need to write any code for that yet!

Goal 2 – Move your robot out of the crowd

Can you get your robot to move?

Still no need to write code!

While you are at it, print out the position of your robot

Goal 3 – A keyboard controller

Some code, at last!

Fire up your favorite editor, and write a ROS Python node that uses the keys WASD to move the robot accordingly.

Part II

Where is my gold?

Goal 1 – Localize the treasure

Launch `rviz` and find a way to display the position (***i.e.*** the TF frame) of the treasure.

From the command line, try to obtain the exact location of the treasure, using `tf_echo` (`rosrun tf tf_echo`).

Goal 2 – Orient your robot

Your robot only output its 6D pose as a `geometry_msgs/PoseStamped` message (use `rostopic show` and `rostopic echo` if necessary).

Write a small Python node `robot_frame.py` that reads this pose every time the robot publishes it, and broadcasts it back as a TF frame (the TF tutorials `http://wiki.ros.org/tf/Tutorials` have useful examples!)

Goal 3 – Write a first autonomous controller

Alright, time to write a first ***real*** ROS package.

To make things more interesting, we are going to write a new node in C++.

First, create a package skeleton with `catkin_create_pkg`, build it, and install it:

```
$ catkin_create_pkg my_robot_controller tf roscpp geometry_msgs
$ cd my_robot_controller
$ mkdir build && cd build
$ ccmake ..
```

➠

If `ccmake` is not installed on your system, it is the right time to install it: `sudo apt-get install cmake-curses-gui`

Make sure you set `CMAKE_BUILD_TYPE` to `Release` and `CMAKE_INSTALL_PREFIX` to your ROS install prefix (for instance `/opt/ros/jade`) or, yet better, a dedicated development prefix like `/home/<username>/dev`. Press `c` to configure, then `g` to generate the makefiles. Quit `ccmake` and then:

```
$ make
$ sudo make install
```

If everything went fine, CMake should have installed an handful of files, but nothing has been compiled yet ...well, that's fair since we did not write any code yet.

Create a new file `src/my_robot_controller_node.cpp` and copy-paste the following code:

```
#include <ros/ros.h>
#include <tf/transform_listener.h>
#include <geometry_msgs/Twist.h>

int main(int argc, char** argv){
  ros::init(argc, argv, "my_robot_controller");

  ros::NodeHandle node;

  ros::Publisher cmd_vel = node.advertise<geometry_msgs::Twist>("/<your robot>/cmd_vel", 10);

  tf::TransformListener listener;

  ros::Rate rate(10.0);
  ROS_INFO("My Preeeeecious!");

  while (node.ok()){
    tf::StampedTransform transform;
    try{
      listener.lookupTransform("/<your robot frame>", "/map",
                               ros::Time(0), transform);
    }
    catch (tf::TransformException ex){
      ROS_ERROR("%s",ex.what());
      ros::Duration(1.0).sleep();
    }

    geometry_msgs::Twist twist;
    twist.angular.z = 1.;
    twist.linear.x = 0.5;
    cmd_vel.publish(twist);

    rate.sleep();
  }
  return 0;
};
```

Modify `CMakeLists.txt` to add your newly created node to the compilation (uncomment the lines 131, 135, 138-140, 157-161), build the project, install it and run it:

```
$ cd build
$ make
$ sudo make install
$ rosrun my_robot_controller my_robot_controller_node
```

> **Note**
>
> You also need to have your TF broadcaster running, otherwise TF will complaint that it could not find your robot's frame.

Modify the code to:

- Print (using `ROS_INFO` or `ROS_INFO_STREAM`) the distance and angle to the treasure,
- Adapt the angular velocity to actually move toward the treasure

The first to reach the chest gets... well, the treasure! Ask me!

Goal 4 – Simplify the launch procedure

Instead of having to manually launch the TF broadcaster and then, the robot controller, we can create a ***launch file*** that does that for us.

First, move your TF broadcaster node to the package `my_robot_controller` (by convention, in the `nodes/` directory) and modify `CMakeLists.txt` to install it along the node `my_robot_controller_node`.

Check it runs fine by calling:

```
$ rosrun my_robot_controller robot_frame.py
```

Now, create a launch file `control.launch` in `launch/`:

```
<launch>
    <node pkg="my_robot_controller" type="robot_frame.py" name="broadcaster" output="screen" />
    <node pkg="my_robot_controller" type="my_robot_controller_node" name="controller" output="screen" />
</launch>
```

Update `CMakeLists.txt` to install the launch file, and test it:

```
$ roslaunch my_robot_controller control.launch
```

Part III

What next?

Depending on your interests, here a selection of interesting ROS packages/examples (sorted by increasing complexity) that I encourage you to explore to gain more experience with ROS:

- 6D face tracking with a webcam
  `https://github.com/severin-lemaignan/attention-tracker`
- Accessing the Kinect point-cloud
  `http://wiki.ros.org/freenect_camera`
- ROS with Nao
  `http://wiki.ros.org/nao/`
- 2D map creation, SLAM and path planning (requires MORSE)
  `http://www.openrobots.org/morse/doc/latest/user/advanced_tutorials/ros_nav_tutorial.html`
- 3D object recognition (much more difficult!)
  `http://wg-perception.github.io/ork_tutorials/index.html`