SOFAR Lab 3
The « puppet hand» node
Using services and transformations

A few pieces of advice:

- Read the whole text before starting.
- Do not try to write the whole program in one shot: go through the steps that are proposed in the text, and make sure each step is validated before proceeding to the next.
- Do not copy and paste whole chunks of code from the tutorial: it's the surest way not to understand what you are doing. Try to copy/paste just the declaration of the objects you need (a tf2 broadcaster, a tf2 listener, etc) and figure out the rest by yourself. A correctly configured Qtcreator IDE goes a long way to make this easy.

Goal and purpose

The goal of this lab is to program a node that can move the left arm of the Baxter robot in such a way that the left hand remains at a given distance from the right hand, with their Z axes opposed.

The purpose is to teach you how to use ROS services and transformation listeners/broadcasters.

How to reach the goal

One way to obtain the desired behavior of the « puppet hand » is to attach an additional frame to the right hand of the Baxter robot at the position and orientation where we wish to move the left hand frame.

In order to inform the node which controls the left arm of the current position of the target, a « **tf broadcaster** » broadcasts the transformation between the new frame and the base frame of the Baxter robot.

The controller node is a « **tf listener** »: it monitors the position and orientation of the target with respect to the base frame of the Baxter and calls a **service** to calculate a joint position which allows reaching the goal, if any.

To use a ROS service, the client node sends a **service request message** and waits for the corresponding **service response message**. Both messages are strictly typed data structures.

Information

We will use packages **tf2** and **tf2_ros**, not the (still usable but deprecated) **tf** package as is the case in the slides.

You will need to read about tf2 listeners and broadcasters :

- http://wiki.ros.org/tf2/Tutorials/Writing%20a%20tf2%20listener%20%28C%2B%2B%29
- http://wiki.ros.org/tf2/Tutorials/Writing%20a%20tf2%20listener%20%28C%2B%2B%29

Quick warning: if you do your own research, you will see that tf2 also has « static broadcasters », which are meant to be used when broadcasting a **constant** transform. As it is the case here, you may wonder why not use that. The reason is that I want you to learn the general tool for time-varying transforms, as it also works for constant transforms. The interest of « static broadcasters » is beyond the scope of this lab, but feel free to have a look at that notion by yourself if you are interested.

The service you will need to use is Baxter's Inverse Kinematics Service. Beware: ROS defines a more general IK service. Most information you will find online using a search engine is likely to be

about the general ROS IK service, not Baxter's IK service and it can be very confusing. If you limit yourself to the resources given in the present text, you will avoid that confusion.

Baxter's IK service is defined in the baxter_core_msgs package. To list the service messages defined in the package, type: **`rosservice find baxter_core_msgs/`** and type the TAB key twice. The inverse kinematics service message is listed as **`baxter_core_msgs/SolvePositionIK`**.

Then if you type rosservice find **baxter_core_msgs/SolvePositionIK**, you will see the names of the two inverse kinematics services, one for each arm of the Baxter.

To get the description of the message, use either of these methods

- Go to https://github.com/RethinkRobotics/baxter_common/blob/master/baxter_core_msgs/srv/SolvePositionIK.srv
- Type **`rossrv show baxter_core_msgs/SolvePositionIK`**.

You will need to write a service client, which is described at :

- http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28c%2B%2B%29

Programming tips

Especially in the "puppet_hand_node" program, there are a few tests that the program has to pass before the IK service can be successfully called. The following is by no means compulsory, but I find it useful to write the programs in this way:

```
if( test1GoesWrong ){
   ROS_INFO("Test 1 went wrong.");  // Or ROS_ERROR
   rate.sleep() ;  // Assuming "rate" is the name of my ros::Rate object.
   Continue ;
}
if( test2GoesWrong ){
   ROS_INFO("Test 2 went wrong.");  // Or ROS_ERROR
   rate.sleep() ;
   Continue ;
}
// Now that all is OK, the code ...
```

As opposed to:

```
if( test1GoesWrong ){
   ROS_INFO("Test 1 went wrong.");
}else{
   if( test2GoesWrong ){
      ROS_INFO("Test 1 went wrong.");
   }else{
                …
                                                    if( testnGoesWrong ){
                                                       ROS_ERROR("Can't
                                                             even write
                                                             it!");
                                                    }else{ //All OK at last
                                                       but not much room
                                                       to code…
                                                    }
   }
}
```

Tasks

- First program the tf2 broadcaster node, starting from the node skeleton. The name of the file must be tf2_broadcaster.cpp, as it is the name given in the CMakeLists.txt file. You will name the frame that the node broadcasts "frame_xx", with xx the number of your PC. Remember, you are all on the same network and all your frames will be added to the same tf tree, so you must use different frame names. The new frame will be attached to "right_gripper", at relative coordinates (0,0,0) but with a rotation of pi around either x or y. The method **`setRPY`** of the tf2::Quaternion class will be helpful.
- Use rviz to check that it works. Visualize the robot and the tfs. Unfold the "frames" in "tf", unclick "All enabled", then select your frame (if it's not in the tree, maybe your node is not running, or not working properly). Move the right arm of the robot, your frame should move with it.
- Once the broadcaster works, proceed to writing the listener. The listener will be use to have tf calculate the transformation between your "frame_xx" and "base". It is important to know that, when you use the method "**`lookupTransform("A", "B",…)`** you get the transformation ${}^{A}T_{B}$. The order of arguments matters.
- To check that it works, first calculate and print the (easily predictable) transformation between frame_xx and right_gripper. Then calculate and print the one we're interested in, the transformation between "frame_xx" and "base".
- Once the listener works, proceed to define the service client

How to manage the tests

- A single node (from one group) at a time will be controlling the robot.
- We will use the **token** to manage access to the robot. The team who have the token are allowed to run their node. Others wait until they get the token.
- **Any** failure of the test causes the token to pass to the next group.
- Tests have no reason to be long for this lab.
- Tests of the broadcaster can be run from QT. To test the listener, you need your broadcaster to be running. Just keep it running in a terminal (**`rosrun puppet_hand tf2_broadcaster`**), and run the tests of the listener from QT. But also write and test a launch file to start both your nodes. It's a very easy one.

Deliverables

After validation, upload the zip file of your package to Hippocampus.

At the end of the lab

Backup your work to a personal medium (*e.g.* a pendrive, an online folder, attached to an email…). Make sure both members of the group have the files for next session.