Robot Operating System

Lab 3: The “Puppet hand” node - services and TF

1 Goals

The goal of this lab is to program a node that can move the left arm of Baxter in such a way that the left hand remains at the same 3D pose from the right hand, with their z-axes opposed. To do so, we will use services and 3D transformations listeners and broadcasters.

2 Deliverables

After validation, the whole package should be zipped and sent by mail (G. Garcia) or through the lab upload form (O. Kermorgant).

3 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 reference frame of the left hand.

In order to set the goal pose of the left arm the “ 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 robot and calls a service (Baxter inverse kinematics service) to calculate the joint positions which allows reaching the goal when sending a `joint_command` topic for the left hand. 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. You must check before starting to use the service that it is available. Moreover, you have to take into accound that maybe the service call is unsuccessful because there is no solution for that pose.

3.1 Informations

Tutorials can be found online about *tf listeners* and *tf broadcasters*.

You will also need to use the Baxter Inverse Kinematic Service. The corresponding service is defined as `baxter_core_msgs/SolvePositionIK`. You can use `rossrv show` to get the request / response structure of this service. Tutorials also exist on how to define a service client in C++ and Python.

3.2 Tasks

- Define the nodes and topics that your application will use.
- Create a ROS package (`catkin create pkg`) with dependencies on `baxter_core_msgs`, `sensor_msgs` and `ecn_common`
- Draw the expected graph of the application.
- Program the node in C++ and/or Python
- Write a launch file that runs your node.

3.3 Avoiding conflicts between controllers

During this lab, all the groups will try to control the left arm and thus it is not advised that you all run your nodes at the same time. In order to avoid this, a tool is provided in the `ecn_common` package that will have your node wait for the availability of Baxter. The code in explained below:

C++: The token manager relies on the class defined in `ecn_common/token_handle.h`. It can be used this way:

```
#include <ecn_common/token_handle.h>
// other includes and function definitions

int main(int argc, char** argv)
{
    // initialize the node
    ros::init(argc, argv, "my_node");

    // define a group name
    ecn::TokenHandle token(group_name);
    // this class instance will return only when Baxter is available

    // initialize other variables

    // begin main loop
    while(ros::ok())
    {
        // do stuff


        // tell Baxter that you are still working and spin
        token.update();
        loop.sleep();
        ros::spinOnce();
    }
}
```

Python: The token manager relies on the class defined in `ecn_common.token_handle`. It can be used this way:

```
#! /usr/bin/env python
from ecn_common.token_handle import TokenHandle
# other imports and function definitions

# initialize the node
rospy.init_node('my_node')
```

```
# define a group name
token = TokenHandle(group_name)
# this class instance will return only when Baxter is available

#initialize other variables

# begin main loop
while not rospy.is_shutdown():
    # do stuff


    # tell Baxter that you are still working and spin
    token.update();
    rospy.sleep(0.1)
```

With this code, two groups can never control Baxter at the same time. When the controlling group ends their node (either from Ctrl-C or because of a crash) the token passes to the group that has been asking it for the longuest time.

Remind the supervisor that they should have a `token_manager` running on some computer of the room.