Robot Operating System

Lab 4: Image topics

1 Goals

In this lab you will practice image subscription and publishing in ROS. You will program a ROS node that subscribes to one of the camera images of the robot and after overlaying your names onto the image it will be re-published in order to be visualized on Baxter's screen.

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 Tasks

Start by creating a ROS package (`catkin create pkg`) with dependencies on `baxter_core_msgs`, `sensor_msgs` and `ecn_common`. Then modify `package.xml` and `CMakeLists.txt` to add the dependency on `cv_bridge` by hand.

3.1 First task: Baxter's camera images

- Display the images in RViz. What are the name of the image topics?
- Use the `image_view` package to display images coming from the left and right arm cameras.
- Search for the name of the subscribed topic from Baxter that allows displaying images on its screen.

3.2 Second task: Programming an image subscriber

In this task you will create a ROS node that subscribes to the `image_in` topic and display it on your screen (with OpenCV's `imshow`).
Run it through a launch file with the correct topic remapping in order to display either the left or the right image.

3.3 Third task: Programming an image republisher

Now that you know how to subscribe to an image topic, add a text parameter to your node. You then have to grab the current image, write the text parameter on it (see OpenCV function) and publish the resultant image on the image_out topic.
Run this node through a launch file with the correct topic remapping and parameter in order to see your text on Baxter's screen.

4 Informations

Tutorials can be found online about *image_transport* and *cv_bridge*.

4.1 Avoiding conflicts between controllers

During this lab, all the groups will try to display images on Baxter's screen 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");

    ecn::TokenHandle token();
    // 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')

token = TokenHandle()
# 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.