Test of ROS Package by FZI (2019/11/21, Yamaguchi)

Universal_Robots_ROS_Driver was developed in collaboration between Universal Robots and the FZI Research Center for Information Technology.
This package is expected to have the best compatibility with UR than the other third party packages.

Some of features:
- Works for all CB3 (with **software version >= 3.7**) and e-Series (**software >= 5.1**) robots and uses the RTDE interface for communication, whenever possible.
- **Factory calibration** of the robot inside ROS to reach Cartesian targets precisely.
- **Realtime-enabled** communication structure to robustly cope with the 2ms cycle time of the e-Series. To use this, compile and run it on a kernel with the PREEMPT_RT patch enabled. (See the Real-time setup guide on how to achieve this)
- Transparent integration of the **teach-pendant**. Using the URCaps system, a program is running on the robot that handles control commands sent from ROS side. With this, the robot can be paused, stopped and resumed without restarting the ROS driver. This will in the future also enable the usage of ROS-components as part of a more complex UR-program on the teach pendant. ROS-control of the robot can be quit using a service call to continue program execution on the TP.

Requirements
- ROS: Ubuntu 18.04 with ROS melodic, however using Ubuntu 16.04 with ROS kinetic should also work
- Real-time system (update Ubuntu kernel): real-time setup guide
- UR software version: CB3 (with **software version >= 3.7**), e-Series (**software >= 5.1**)

Installation (PC)
Dependencies:
```
$ sudo apt-get -f install ros-kinetic-joint-trajectory-controller ros-kinetic-joint-state-controller ros-kinetic-force-torque-sensor-controller
```
It's better to update the ROS packages.

The main packages:
```
$ cd ~/catkin_ws/src/
# clone the driver
$ git clone https://github.com/UniversalRobots/Universal_Robots_ROS_Driver.git Universal_Robots_ROS_Driver
# clone fork of the description to use the calibration feature
$ git clone -b calibration_devel https://github.com/fmauch/universal_robot.git fmauch_universal_robot
$ cd ..
$ catkin_make
```
It's better to remove catkin_ws/{build,devel} before catkin_make.

Installation (UR pendant)

For using the ur_robot_driver with a real robot you need to install the **externalcontrol-1.0.1.urcap** (Universal_Robots_ROS_Driver/ur_robot_driver/resources/externalcontrol-1.0.1.urcap).

Procedure (UR3e, version 5.2)

1. Copy the data into a USB memory.
2. Insert the USB memory on pendant.
3. On pendant, Hamburger button -> Settings -> URCaps -> "+" -> usbdisk -> externalcontrol-1.0.1.urcap -> Open
   -> Restart
4. On pendant, Installation -> URCaps -> External Control
   -> Change the remote host's IP to IP of a PC (e.g. 192.168.1.12).

Extract calibration information

To make use of factory calibration of the robot in ROS (Not Tested Yet).

```
$ roslaunch ur_calibration calibration_correction.launch \
  robot_ip:=<robot_ip> target_filename:="${HOME}/my_robot_calibration.yaml"
```

It is recommended keeping calibrations for all robots in your organization in a common package.
cf. Package's documentation.

Run

Test run:
(On pendant)

1. Make sure the remote host's IP: Installation -> URCaps -> "External Control"
   -> Change the remote host's IP to IP of a PC (e.g. 192.168.1.12).
2. Program -> URCaps -> Add "External Control"
3. Execute (press the play button).

(On PC)

```
$ roslaunch ur_robot_driver <robot_type>_bringup.launch robot_ip:=<robot_ip>
$ roslaunch ur_robot_driver ur3e_bringup.launch robot_ip:=ur3ea
$ rviz
```

Worked.

Passing the calibration data (Not Tested Yet):

```
$ roslaunch ur_robot_driver <robot_type>_bringup.launch robot_ip:=<robot_ip> \
  kinematics_config:=$(rospack find ur_calibration)/etc/ur10_example_calibration.yaml
```

Further test

The next goal is executing the test codes.
Since some of topic (and service?) names are changed, we create ay_test/ros/py_ros/ur2.

```
$ ./kdl_test1.py
```

$ ./kdl_test2.py
Worked without changes from ay_test/ros/py_ros/ur.

Since the joint name order is changed
from ['shoulder_pan_joint', 'shoulder_lift_joint', 'elbow_joint', 'wrist_1_joint', 'wrist_2_joint', 'wrist_3_joint']
to an alphabetical order ['elbow_joint', 'shoulder_lift_joint', 'shoulder_pan_joint', 'wrist_1_joint', 'wrist_2_joint', 'wrist_3_joint']
we need a remapping of joint angles.
It was due to ros_control.
e.g. **/joint_states topic uses the alphabetical order**.
$ ./get_q1.py
$ ./get_q2.py
Worked.

In the following trajectory code, we needed to modify the action server name
from /follow_joint_trajectory
to **/scaled_pos_traj_controller/follow_joint_trajectory**.
$ ./follow_q_traj2.py
Worked. It also uses reordering version of GetState defined in get_q1.py.

Velocity-based control is planned, but not implemented yet.
cf.
https://github.com/UniversalRobots/Universal_Robots_ROS_Driver/blob/261fac97ade85b0030d5fcfcc8eb9cc1f01654d8/ur_robot_driver/doc/features.md
We cannot control the robot with velocity commands.
(Note: It may be possible by using the /ur_hardware_interface/**script_command** topic, but there is the same issue explained below.)

So, we consider using the **non-ROS version of velocity control**.
Since it sends the URScript through Ethernet, we **need to activate the Remote Control mode when we use E-series**. However once we operate the robot with the Remote Control mode, the **External Control is terminated**. It seems that we can use a **dashboard service** to activate the External Control again. But programming may become a bit complicated; we need to write a **control mode switching code**.
cf.
https://github.com/UniversalRobots/Universal_Robots_ROS_Driver/tree/master/ur_robot_driver

> **Custom script snippets** can be sent to the robot on a topic basis. By default, they will interrupt other programs (such as the one controlling the robot). For a certain subset of functions, it is however possible to send them as secondary programs. See UR documentation on details.
> **Note to e-Series users**: The robot won't accept script code from a remote source unless the robot is put into*remote_control-mode*. However, if put into *remote_control-mode*, the program containing the **External Control** program node can't be started from the panel. For this purpose, please use the **dashboard services** to load, start and stop the main program running on the robot. See the ROS-API documentation for details on the dashboard services.

$ ./velctrl_noros2.py
Anyway, this code worked.