

## Starting the KRL Script

Start the Orocos scripts on the robot controller. Do not do this alone if you use the robot for the first time. Contact the responsible person listed [here](#).

## Running the server side software

This is now the time to start the IISOrocos server side software. The software is run by starting roscore and running an instance of IISOrocos for each arm:

```
$IIS_INCLUDE_PATH/iis_scripts/both_arms.sh
```

## Arm Control Topics

The topic names have the following structure (according to the lab internal convention orocos topic names have been changed compared to older versions):

```
$execution_type$/hardware$/control_type$/control_topic$
```

```
execution_type = {simulation, real}
hardware = {right_arm, left_arm, right_sdh, left_sdh, kinect1, kinect2}
control_type = {joint_control, cartesian_control, settings, sensing}
```

## Message Types

todo

## Readable Topics

### joint\_control/ptp\_reached

Returns the state of the current ptp movement (see joint\_control/ptp). It returns an array with the first element being 1 when the final position is reached and 0 otherwise.

Message type:

```
std_msgs/Int32MultiArray
```

Sample call:

```
rostopic echo /real/right_arm/joint_control/ptp_reached
```

Sample output:

## TODO

### joint\_control/get\_state

Returns the current joint positions of the arm

Message type:

```
sensor_msgs/JointState
```

Sample call:

```
rostopic echo /real/right_arm/joint_control/get_state
```

Sample output:

```
header:  
  seq: 26774  
  stamp:  
    secs: 0  
    nsecs: 0  
  frame_id: dummy_frame_id  
name: ['arm_0_joint', 'arm_1_joint', 'arm_2_joint', 'arm_3_joint',  
       'arm_4_joint', 'arm_5_joint', 'arm_6_joint']  
position: [-0.025538215413689613, 0.8757621049880981, 1.995240569114685,  
           0.5836411118507385, -0.18975336849689484, 0.7183101773262024,  
           1.3192434310913086]  
velocity: [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0] <-- not working currently  
effort: [0.24101980030536652, 0.2610560357570648, 0.6392512917518616,  
         -0.3438339829444885, -0.23192289471626282, 1.0096474885940552,  
         -0.44344362616539]
```

### joint\_control/get\_impedance

Returns joint impedance values

Message type:

```
iis_orocos/FriJointImpedance
```

Sample call:

```
rostopic echo /real/right_arm/cartesian_control/get_impedance
```

Sample output:

```
stiffness: [250.0, 250.0, 250.0, 250.0, 250.0, 250.0, 250.0]  
damping: [0.699999988079071, 0.699999988079071, 0.699999988079071,
```

```
0.699999988079071, 0.699999988079071, 0.699999988079071, 0.699999988079071]
```

### **cartesian\_control/get\_impedance**

Returns Cartesian impedance values

Message type:

```
iis_orocos/CartesianImpedance
```

Sample call:

```
rostopic echo /real/right_arm/cartesian_control/get_impedance
```

Sample output:

```
stiffness:  
  linear:  
    x: 1000.0  
    y: 1000.0  
    z: 1000.0  
  angular:  
    x: 100.0  
    y: 100.0  
    z: 100.0  
damping:  
  linear:  
    x: 0.3  
    y: 0.3  
    z: 0.3  
  angular:  
    x: 0.3  
    y: 0.3  
    z: 0.3  
cpmaxdelta: 0.0  
maxforce: 0.0  
axismaxdeltatrq: 0.0
```

### **cartesian\_control/get\_pose**

Returns cartesian pose of the arm

Message type:

```
geometry_msgs/Pose
```

Sample call:

```
rostopic echo /real/right_arm/cartesian_control/get_pose
```

Sample output:

```
position:  
  x: -0.671013116837  
  y: 0.200215816498  
  z: 0.752071797848  
orientation:  
  x: 0.452136724939  
  y: 0.0641745917401  
  z: -0.889025856601  
  w: 0.0331853931426
```

### **cartesian\_control/get\_velocity\_limit**

Returns Cartesian velocity limit (see cartesian\_control/set\_velocity\_limit)

Message type:

```
std_msgs/Float32
```

Sample call:

```
rostopic echo /real/right_arm/cartesian_control/get_velocity_limit
```

Sample output:

```
data: 0.0
```

### **joint\_control/get\_velocity\_limit**

Returns joint velocity limit (see joint\_control/set\_velocity\_limit)

Message type:

```
std_msgs/Float32
```

Sample call:

```
rostopic echo /real/right_arm/joint_control/get_velocity_limit
```

Sample output:

```
data: 0.10000000149
```

## Writable Topics

### joint\_control/set\_impedance

Sets joint impedance values (stiffness, damping)

Message type:

```
iis_orocos/FriJointImpedance
```

Sample call:

```
rostopic pub /real/right_arm/joint_control/set_impedance
iis_orocos/FriJointImpedance "stiffness: [250.0, 250.0, 250.0, 250.0, 250.0,
250.0, 250.0]
damping: [0.699999988079071, 0.699999988079071, 0.699999988079071,
0.699999988079071, 0.699999988079071, 0.699999988079071, 0.699999988079071]"
```

### joint\_control/ptp

Movement in joint space for point to point movement. The robot has to be in an appropriate control mode before using this topic (otherwise it will not move) (see settings/switch\_mode).

Message type:

```
std_msgs/Float64MultiArray
```

Sample call:

```
rostopic pub /real/right_arm/joint_control/ptp std_msgs/Float64MultiArray
"layout:
  dim:
    - label: 'RAD'
      size: 7
      stride: 0
      data_offset: 0
  data: [-0.15538215413689613, 0.8757621049880981, 1.995240569114685,
0.5836411118507385, -0.18975336849689484, 0.7183101773262024,
1.3192434310913086]"
```

### joint\_control/move

Trajectory movement in joint space. The trajectory has to be provided in small pieces. This must *not* be used for ptp movement as it can yield high accelerations. It accepts the angles in which the arm joints should be after the next cycle (one cycle has a duration between 1 and 20 ms)

Message type:

## std\_msgs/Float64MultiArray

Sample call:

The following is not recommend to use outside of a program providing proper trajectories, as it can yield high accelerations when used wrongly  
TODO

## cartesian\_control/move

Trajectory movement in Cartesian space. The trajectory has to be provided in small pieces. This must *\*not\** be used for ptp movement as it can yield high accelerations. It accepts the pose in which the arm should be after the next cycle (one cycle has a duration between 1 and 20 ms)

Message type:

geometry\_msgs/Pose

Sample call:

The following is not recommend to use outside of a program providing proper trajectories, as it can yield high accelerations when used wrongly  
TODO

## cartesian\_control/ptp

Movement in Cartesian space for point to point movement. The robot has to be in an appropriate control mode before using this topic (otherwise it will not move) (see settings/switch\_mode).

Message type:

std\_msgs/Float64MultiArray

Sample call:

TODO

## cartesian\_control/set\_impedance

Sets cartesian impedance values (stiffness, damping, cpmaxdelta, maxforce, axismaxdeltatrq)

Message type:

iis\_orocos/CartesianImpedance

Sample call:

```
rostopic pub /real/right_arm/cartesian_control/set_impedance
iis_orocos/CartesianImpedance "stiffness:
  linear: {x: 1000.0, y: 1000.0, z: 1000.0}
  angular: {x: 100.0, y: 100.0, z: 100.0}
damping:
  linear: {x: 0.3, y: 0.3, z: 0.3}
  angular: {x: 0.3, y: 0.3, z: 0.3}
cpmaxdelta: 0.0
maxforce: 0.0
axismaxdeltatrq: 0.0"
```

### **cartesian\_control/set\_velocity\_limit**

Sets maximum velocity limit for trajectory movements in cartesian space (cartesian\_control/move). It expects the maximum allowed absolute Cartesian moving distance within one cycle.

Message type:

```
std_msgs/Float32
```

Sample call:

```
rostopic pub /real/right_arm/cartesian_control/set_velocity_limit
std_msgs/Float32 0.0
```

### **settings/switch\_mode (write)**

```
modes:
  10    Position Controller
  20    Cartesian Stiffness Controller (be careful in mode 20 -
calibration seems to be not complete for right arm - drifting observed)
  30    Axis-specific Stiffness Controller
  101   Gravity Compensation
```

usage example:

```
rostopic pub /real/right_arm/settings/switch_mode std_msgs/Int32 10
```

### **sensing/cartesian\_wrench (read)**

returns forces and torques

### **sensing/state (read)**

returns int array containing robot state  
[0] power

```
[1] control  
[2] error  
[3] warning
```

### sensing/temperature (read)

```
returns temperature of robot joints in float array
```

### joint\_control/get\_impedance (read)

```
returns joint impedance values
```

### settings/get\_command\_state (read)

```
returns current command state in float32multiarray ([0] --> 0 = monitor mode, 1 = command mode; [1] --> number of control mode)
```

### joint\_control/move

```
movement in joint space
```

### joint\_control/get\_state

```
returns joint pose of the arm
```

### joint\_control/set\_impedance

```
sets joint impedance values
```

### joint\_control/set\_velocity\_limit

```
sets maximum velocity limit for movements in joint space
```

From:

<https://iis.uibk.ac.at/> - IIS

Permanent link:

<https://iis.uibk.ac.at/intranet/projects/robot/armtopics?rev=1401896109>

Last update: **2018/09/03 14:57**



