# General startup information

### Starting the KRL Script

Start the Orocos scripts on the robot contoller. 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
```

If you are only running one arm, you can also run this one separately by using

```
  $IIS_INCLUDE_PATH/iis_scripts/left_arm.sh
```

or

```
  $IIS_INCLUDE_PATH/iis_scripts/right_arm.sh
```

respectively (depending on the script you are running on the KUKA controller side). As soon as you have started both scripts, you should be able to see the arm control topics listed below.

### Using the robot in compliant mode to do gesture demonstration

In order to be able to manipulate the robot by hand to record gesture, follow these steps :

- Start the FriOrocos script normally in AUT mode (Spiral) - Start roscore / arm / gripper scripts as indicated previously - Through ROS, switch mode to mode 0 (Monitor mode, that should be acknowledged on the KCP) then switch to mode 30 (mode 101 not working at this date - June 23rd, 2017). You can just adjust the impedance (stiffness/damping) to make the robot more compliant :

```
   rostopic pub /real/right_arm/joint_control/set_impedance
iis_robot_dep/FriJointImpedance "stiffness: [0.2, 0.2, 0.2, 0.2, 0.2, 0.2,
0.2]
   damping: [0.2, 0.2, 0.2, 0.2, 0.2, 0.2, 0.2]"
```

**BEWARE !!! If you set the impedance values back to a less compliant system, the arm will try to reach VERY FAST back to its initial position before manipulation. Thus, anything on the path will be damaged !!! Be sure to bring manually the arm back to its starting position to reduce this effect !**

# 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, sensoring}
```

## Message Types

todo

## Readable Topics

### settings/get_speed

Returns the current speed for ptp movements (see joint_control/ptp) as a number between 0 (no movement) and 1 (maximum speed)

Message type:

```
std_msgs/Float64
```

Sample call:

```
rostopic echo /real/right_arm/settings/get_speed
```

Sample output:

```
data: 0.5
```

### settings/get_command_state

Returns the current command state. The output is a 2-dim array, where the first number indicates if the arm is in command mode and the second number contains the exact command mode number.

Message type:

```
std_msgs/Float32MultiArray
```

Sample call:

```
rostopic echo /real/right_arm/settings/get_command_state
```

Sample output:

```
layout:
  dim: []
  data_offset: 0
data: [1.0, 10.0]
```

#### sensoring/temperature

Returns the temperature in each robot joint as array.

Message type:

```
std_msgs/Float32MultiArray
```

Sample call:

```
rostopic echo /real/right_arm/sensoring/temperature
```

Sample output:

```
layout:
  dim: []
  data_offset: 0
data: [42.40, 46.20, 45.90, 46.70, 45.90, 62.40, 61.20]
```

#### sensoring/state

Returns the current state of the robot as array. It contains 4 numbers:

- m_msr_data.robot.power
- m_msr_data.robot.control
- m_msr_data.robot.error
- m_msr_data.robot.warning

Message type:

```
std_msgs/Int32MultiArray
```

Sample call:

```
rostopic echo /real/right_arm/sensoring/state
```

Sample output:

```
layout:
```

```
  dim: []
  data_offset: 0
data: [0, 1, 0, 0]
```

#### sensoring/jacobian

Returns the Jacobian in the current robot position.

Message type:

```
std_msgs/Float64MultiArray
```

Sample call:

```
rostopic echo /real/right_arm/sensoring/jacobian
```

Sample output:

```
layout:
  dim: []
  data_offset: 0
data: [0.11856745183467865, 0.8065090179443359, 0.18409588932991028,
0.1728156954050064, -0.049717362970113754, -0.019414713606238365, 0.0,
0.6405692100524902, -0.00604418246075511, 0.08719471842050552,
-0.3324144184589386, -0.012777105905115604, 0.07554514706134796, 0.0,
0.25680267810821533, -0.04223119094967842, 0.026677558198571205,
-0.25205785036087036, 0.0, 0.0, 0.0, -0.808161735534668,
0.016004111617803574, -0.06539558619260788, 0.9157959818840027,
0.16380904614925385, -0.9685274958610535, -0.0, -0.0841047391295433,
0.9862498044967651, -0.16312111914157867, 0.38198092579841614,
-0.6374020576477051, -0.24890658259391785, 0.0, 0.5829245448112488,
0.16448475420475006, 0.9844363331794739, 0.12413008511066437,
0.7529177665710449, -0.0, 1.0]
```

#### sensoring/msr_jnt_trq

Returns the current measured total torques on each joint (7-dim float array).

Message type:

```
std_msgs/Float64MultiArray
```

Sample call:

```
rostopic echo /real/right_arm/sensoring/msr_jnt_trq
```

Sample output:

```
layout:
  dim: []
  data_offset: 0
data: [24.940000534057617, 33.150001525878906, 7.269999980926514,
-4.840000152587891, -0.18000000715255737, 1.159999966621399,
-0.46000000834465027]
```

#### sensoring/est_ext_jnt_trq

Returns the current torques applied on each joint (7-dim float array) excluding the torques generated by the rest of the arm itself. These values are estimated by the KUKA controller.

Message type:

```
std_msgs/Float64MultiArray
```

Sample call:

```
rostopic echo /real/right_arm/sensoring/est_ext_jnt_trq
```

Sample output:

```
layout:
  dim: []
  data_offset: 0
data: [0.4312626123428345, 0.05431468039751053, 0.6109476685523987,
-0.3116573691368103, -0.22619108855724335, 0.9795454144477844,
-0.44677433371543884]
```

#### sensoring/error

Returns an error if one has been observed during the loop cycle. The message consists of an error code and an error message.

Message type:

```
iis_orocos/OrocosError
```

Sample call:

```
rostopic echo /real/right_arm/sensoring/error
```

Sample output:

```
TODO
```

**sensoring/cartesian_wrench**

Returns the current cartesian forces and torques at the end effector

Message type:

```
geometry_msgs/Wrench
```

Sample call:

```
rostopic echo /real/right_arm/sensoring/cartesian_wrench
```

Sample output:

```
force:
  x: 2.77582025528
  y: -0.254617959261
  z: -3.10084724426
torque:
  x: 0.430095881224
  y: 0.816247463226
  z: 0.547478497028
```

**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_dir

Returns cartesian pose of the arm In World space. The first 3 elements are the Cartesian position, the next 3 are the orientation of end-effector and the last one is the rotation around the end-effector axis

Message type:

```
std_msgs/Float64MultiArray
```

Sample call:

```
rostopic echo /real/right_arm/cartesian_control/get_pose_dir
```

Sample output:

```
layout:
dim: []
data_offset: 0
data: [-0.1646200986390433, -0.30508448189857773, 0.23461338424203065,
0.3470495466622819, -0.8735133502317335, -0.34131054534893845,
0.9007184461592337]
```

### cartesian_control/get_pose_quat

Returns cartesian pose of the arm in World space

Message type:

```
geometry_msgs/Pose
```

Sample call:

```
rostopic echo /real/right_arm/cartesian_control/get_pose_quat
```

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

#### settings/switch_mode

Allows to set the current command mode of the arm according to the following 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

Sample call:

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

#### settings/set_speed

Sets the current speed for ptp movements (see joint_control/ptp) as a number between 0 (no movement) and 1 (maximum speed).

Message type:

```
std_msgs/Float64
```

Sample call:

```
rostopic pub /real/right_arm/settings/set_speed std_msgs/Float64 0.4
```

#### 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 (see cartesian_control/get_pose_dir for the message structure). 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 /simulation/right_arm/cartesian_control/ptp
std_msgs/Float64MultiArray "layout:
dim:
- label: ''
  size: 7
  stride: 0
data_offset: 0
data: [-0.16415267011979597, -0.30530589495884486, 0.23607497182886716,
0.3474850741947719, -0.8739457439933868, -0.33975740849966135,
0.9000890133095816]"
```

#### /simulation/right_arm/cartesian_control/ptpQuaternion

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:

```
geometry_msgs/Pose
```

Sample call:

```
rostopic pub /simulation/left_arm/cartesian_control/ptpQuaternion
geometry_msgs/Pose "position:
  x: -0.38878687339
  y: 1.80645028202
  z: 0.240840127477
orientation:
  x: 0.0784292923653
  y: 0.788223576718
  z: 0.5547947503
  w: 0.254470194702"
```

#### 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
```