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
```

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

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

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/Float64MuiArray
"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
```

sensoring/cartesian_wrench (read)

```
returns forces and torques
```

sensoring/state (read)

```
returns int array containing robot state
[0] power
[1] control
[2] error
[3] warning
```

sensoring/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
```