Kinova Mico RR Bridge Documentation

Contents

This documentation is for mico_rr_bridge catkin package that hosts ROS topics/services/actions of Kinova Mico Arm as Robot Raconteur Services which makes communication and control through various languages, such as MATLAB, python, java, C++, etc., possible. It has been tested using Ubuntu 12.04 and ROS Hydro on the server side and MATLAB R2014a on the client side. Robot Raconteur version 0.5-testing has been used.

The documentation assumes jaco-ros for the Kinova Mico is already setup. If not, user can follow installation instructions on *https://github.com/Kinovarobotics/jaco-ros* before proceeding further.

1 Mico RR Bridge installation

To include mico_rr_bridge in your workspace, follow these steps:

```
# cd catkin_ws/src
# git clone https://github.com/ckaur/mico_rr_bridge.git
# cd catkin_ws
# catkin_make
```

The script needs to be executable. To do so, run the following:

```
# cd catkin_ws/src/mico_rr_bridge/scripts
# chmod +x MicoRRService.py
```

2 Startup Kinova Mico

To start communication with Kinova Mico Arm, run the following in terminal:

```
# cd catkin_ws
# roslaunch jaco_driver mico_arm.launch
```

3 MicoRRService.py

This service provides functions and properties to control and obtain measurements for the Kinova Arm. To run, open a new terminal and type the following command:

```
# rosrun mico_rr_bridge MicoRRService.py [--port p]
```

Once the service is up and running, the properties and functions can be accessed by connecting to the same. For MATLAB:

```
# kinova = RobotRaconteur.Connect('tcp://localhost:[p]/MicoJointServer/Mico')
```

3.1 Basic Services

To "home" the arm:
function void home_arm()

```
# kinova.home_arm()
```

To activate e-stop function:
function void stop()
```
# kinova.stop()
```

To restore arm function:
function void start()
```
# kinova.start()
```

3.2 Cartesian Control

To set the arm position using Cartesian co-ordinates:
function void setArmPosition(double[] position, double[] orientation)

where *position* is a 3 x 1 vector containing end-effector position (x,y,z) in meters and *orientation* is 4 x 1 vector containing end-effector quarternion orientation (x,y,z,w).

To obtain current arm position, use following function to update properties:
function void tool_position()

and then, access co-ordinates using:
property double[] position
property double[] orientation

3.3 Angular Control

To set the joint angles using DH transformed angles in radians:
function void setJointAngle(double[] angle_set)

where *angles_set* is 6 x 1 vector of joint angles in order - base, shoulder, elbow, wrist, wrist and hand.

To obtain current joint angles in **degrees**, use following function to update properties:
function void joint_angles()

and then, access joint angles using:
property double[] joints_deg

To obtain current joint angles in **radians**, use following function to update properties:
function void joint_state()

and then, access joint angles using:
property double[] joints_rad

3.4 Finger Control

To set the finger positions:
function void setFingerPosition(double[] position)

where *position* is 2 x 1 vector containing positions for finger 1 and finger 2.

To obtain current finger position, use following function to update properties:
function void finger_position()

and then, access position using:
property double[] fingers

3.5 Velocity Control

3.5.1 Joint Velocity Control

To control arm using joint velocity commands:
function void jointvel(double[] vjoint, double r, double nc)

where *vjoint* is the vector of size 6 x 1 containing joint velocities for base, shoulder, elbow, wrist, wrist and hand respectively, *r* is the rate to publish the command and *nc* is the number of commands.

For example,
```
# kinova.jointvel([0;0;0;0;0;10], 10, 5)
```
will spin the sixth joint (hand) and the command will be executed 5 times at 10 Hz.

3.5.2 Cartesian Velocity Control

To control arm using cartesian velocity commands:
function void carvel(double[] vlinear, double[] vangular, double r, double nc)

where *vlinear* and *angular* are vectors of size 3 x 1 containing linear (x,y,z) and angular velocities (x,y,z) for end-effector respectively, *r* is the rate to publish the command and *nc* is the number of commands.

For example,
```
# kinova.carvel([0;0;0.1], [0;0;0], 10, 5)
```
will move the arm in +ve z-direction and the command will be executed 5 times at 10 Hz.