Users manual: KUKA Sunrise Toolbox

Authors: Mohammad SAFEEA[1], Pedro NETO[2]

(1) Coimbra University & Ensam University, email: ms@uc.pt.
(2) Coimbra University, email: pedro.neto@dem.uc.pt.

28th-May-2018

Contents

Part I. Introduction

The KUKA Sunrise Toolbox (KST) is a MATLAB toolbox which can be used to control KUKA iiwa R 800 and R 820 manipulators from an external computer using MATLAB. The toolbox is available in the following Github repository:

`https://github.com/Modi1987/KST-Kuka-Sunrise-Toolbox`

In addition video tutorials about the toolbox are available in the link:

`www.youtube.com/watch?v=_yTK0Gi0p3g&list=PLz558OYgHuZdVzTaB79iM8Y8u6EjFe0d8`

The user may consult the videos in tandem with this document.

1 About

Using the KST, the user can control the KUKA iiwa robot from his/her computer without requiring a knowledge about programming the industrial manipulator, as such any person with a basic knowledge of MATLAB can control the robot remotely from an external PC . The toolbox provides numerous methods that can be divided roughly into the following categories:

- Networking
- Soft real-time control
- Point-to-point motion functions
- Setters
- Getters
- General purpose
- Physical interaction

This documentation provides instructions on the utilization of the KST, in addition it lists the various methods provided by Toolbox with a briefing about their functionality and way of use. Each method is elaborated in a subsection with the following entries:

- Syntax: lists the MALAB syntax for calling the method in subject in a MATLAB script.
- Description: describes the functionality of the method in subject.
- Arguments: describes the arguments taken by the method in subject.
- Return values: describes the variables returned after the execution of the method in subject.

Part II. Getting started, the requirements

Before starting to use the Toolbox for controlling the robot, the following software/hardware are required:

2 Hardware requirements

The user is required to have:

1. One of KUKA iiwa manipulators R800 or R820.
2. An up-to-date external PC/laptop with good computational power.
3. Good Ethernet cable, a category five or better.

A network between the robot and the PC shall be established. To establish the network the user shall do the following:

- Using an Ethernet cable, connect the X66 port of the robot to the Ethernet port of your PC.
- From the teach pendant of the manipulator, verify the IP of the robot, (explained in the video tutorials "Tutorial 1").
- Then on the external PC, the user shall change the IP of the PC into a static IP in the range of robot's IP.

3 Software requirements

The following software packages are required:

- MATLAB: the user shall have MATLAB installed at his PC, using MATLAB and KST, the user will be able to control the manipulator from the external PC, in such a case the user may use the operating system of his preference, Windows, Linux or Mac.
- Sunrsie.Workbench: the user will need the Sunrsie.Workbench only once to synchronize the (MatlabToolboxServer) application of the KST to the controller of the robot.

Part III. Functions list

In the following a list of the supported methods by KST is presented along with a brief description about its utilization in a MATLAB script.

4 Networking

Networking functions are used to administer the network communication between the external PC and the controller. This section lists those functions along with a brief description:

4.1 net_establishConnection

- Syntax:

  ```
  >> [t_Kuka] = net_establishConnection(ip)
  ```

- Description:

  This function is used to establish a connection with the MatlabToolboxServer application on the controller.

- Arguments:

  - ip: a string variable which contains the IP of the robot, e.g. "172.31.1.147". In order to figure out your robot IP you can click on "Station" button on the teach pendant, then on the "Information" button the IP of the robot will be listed under the title "Station Server IP".

- Return values:

  - t_Kuka: is the communication object.

4.2 net_turnOffServer

- Syntax:

  ```
  >> net_turnOffServer(t_Kuka)
  ```

- Description:

  This function is used to turn off the server on the robot.

- Arguments:

  - t_Kuka: is the communication object.

- Return values:

- None.

4.3 net_updateDelay

- Syntax:

  ```
  >> [time_stamps,time_delay] = net_updateDelay(t_Kuka)
  ```

- Description:

  This function is used to establish a plot of the communication delay between the computer and the controller, i. e., it tests the timing connection characteristics between KUKA iiwa and the computer.

- Arguments:

  - t_Kuka: is the communication object.

- Return values:

  - time_stamps: is the time stamp of each socket.
  - time_delay: is the delay for each socket.

4.4 net_pingIIWA

- Syntax:

  ```
  >> net_pingIIWA(ip)
  ```

- Description:

  This function is used to ping the robot controller, afterwards it prints on MATLAB's command window the timing characteristics of the ping operation.

- Arguments:

  - ip: is the IP of the KUKA controller.

- Return values:

  - None.

5 Soft real-time control

In this group of functions the user can move the robot on-the-fly by streaming the positions to the controller, this is important for dynamic paths where the path is changing according to changes in the surrounding environment, for example according to the position of dynamic obstacles moving in the environment. Below, it is listed the soft real-time control functions along with utilization instructions.

5.1 realTime_moveOnPathInJointSpace

- Syntax:

  ```
  >> realTime_moveOnPathInJointSpace(t_Kuka, trajectory, delayTime)
  ```

- Description:

  This function is used to move the robot continuously in joint space.

- Arguments:

  - t_Kuka: is the communication object.
  - trajectory: is a 7xn array, this array is a concatenation of the joints angles describing the configurations taken by the robot during the motion.
  - delayTime: is the time delay between two

- Return values:

  - None.

5.2 realTime_startDirectServoJoints

- Syntax:

  ```
  >> realTime_startDirectServoJoints(t_Kuka)
  ```

- Description:

  This method is used to turn on the DirectServo function on the robot for initiating the soft real-time control in a joint space. After starting DirectServo function, the user have to stream the joints' target positions to the robot using the function (sendJointsPositions).

- Arguments:

  - t_Kuka: is the communication object.

- Return values:
  - None.
- Related functions:
  - sendJointsPositions
  - realTime_stopDirectServoJoints

5.3 realTime_stopDirectServoJoints

- Syntax:

  ```
  >> realTime_stopDirectServoJoints(t_Kuka)
  ```

- Description:

  This function is used to turn off the DirectServo function on the robot ending the soft real-time control.

- Arguments:
  - t_Kuka: is the communication object.
- Return values:
  - None.
- Related functions:
  - sendJointsPositions
  - realTime_startDirectServoJoints

5.4 realTime_startImpedanceJoints

- Syntax:

  ```
  >> realTime_startImpedanceJoints(t_Kuka, tw, cOMx, cOMy, cOMz,
  cS, rS, nS)
  ```

- Description:

  This function is used for starting the soft real-time control at the joints level in the impedance mode.

- Arguments:
  - t_Kuka: is the communication object.
  - tw: is the weight of the tool attached to the flange (kg).
  - cOMx: is the X position of the tool's center of the mass in the flange referenced frame.
  - cOMy: is the Y position of the tool's center of the mass in the flange referenced frame.
  - cOMz: is the Z position of the tool's center of the mass in the flange referenced frame.
  - cS: is the Cartesian linear stiffness in the range of [0 to 4000].
  - rS: is the cartesian angular stifness in the range of [0 to 300].
  - nS: is the null space stiffness.
- Return values:
  - None.
- Related functions:
  - sendJointsPositions
  - realTime_stopImpedanceJoints

5.5 realTime_stopImpedanceJoints

- Syntax:

```
>> realTime_stopImpedanceJoints(t_Kuka)
```

- Description:

  This function is used to stop the soft real-time control at the joints level in the impedance mode.
- Arguments:
  - t_Kuka: is the communication object.
- Return values:
  - None.
- Related functions:
  - sendJointsPositions
  - realTime_startImpedanceJoints

5.6 sendJointsPositions

- Syntax:

  ```
  >> [ret]=sendJointsPositions(t_Kuka ,jPos)
  ```

- Description:

  This function is used to send target joints' position to the robot for the soft real-time control at the joints level, it can be used in both modes, the DirectServo mode and the impedance mode. This function is blocking, it awaits for an acknowledgment from the server before returning back to execution.

- Arguments:

  - `t_Kuka`: is the communication object.
  - `jPos`: is a 7x1 cell array representing the target joints' positions, units in radians.

- Return values:

  - `ret`: the return value is true if the joints' position message has been received and processed successfully by the server, or false otherwise.

5.7 sendJointsPositionsf

- Syntax:

  ```
  >> sendJointsPositionsf(t_Kuka, jPos)
  ```

- Description:

  This function is used to send target joints' position to the robot for the soft real-time control at the joints level, it can be used in both modes, the DirectServo mode and the impedance mode. This function is non-blocking, it is one way and does not awaits for an acknowledgment from the server. This function is computationally light-weight, and it is designed for implementation in computationally expensive algorithms. When utilized in low-computational-cost algorithms, this function can execute very fast, as a result sending command packets to the robot at high rates (4K hz), this might cause execution issues, to solve this problem the user may need to perform some timing as to guarantee a transmission rate of around 275 packets/second, for an example about the best practice for using this function please refer to script (`Tutorial_directServoFast.m`) in github repo.

- Arguments:

  - t_Kuka: is the communication object.
  - jPos: is a 7x1 cell array representing the target joints' positions, units in radians.

- Return values:
  - None.

5.8 sendJointsPositionsExTorque

- Syntax:

```
>> [torques] = sendJointsPositionsExTorque(t_Kuka, jPos)
```

- Description:

  This function is used to send target joints' position to the robot for the soft real-time control at the joints level while simultaneously it returns a feed back about the external torques in the joints due to external forces acting on the structure of the robot, this function can be used in both modes, the DirectServo mode and the impedance mode. This function is blocking, it awaits for the torques message from the server before returning back to execution.

- Arguments:
  - t_Kuka: is the communication object.
  - jPos: is a 7x1 cell array representing the target joints' positions, units in radians.

- Return values:
  - torques: a 7x1 cell array with joints' torques due to the external forces acting on the robot structure.

5.9 sendJointsPositionsMTorque

- Syntax:

```
>> [torques] = sendJointsPositionsMTorque(t_Kuka, jPos)
```

- Description:

  This function is used to send target joints' position to the robot for the soft real-time control at the joints level while simultaneously it returns a feed back about the torques in the joints as measured by the integrated sensors, this function can be used in both modes, the DirectServo mode and the impedance mode. This function is blocking, it awaits for the torques message from the server before returning back to execution.

- Arguments:
  - `t_Kuka`: is the communication object.
  - `jPos`: is a 7x1 cell array representing the target joints' positions, units in radians.

- Return values:
  - `torques`: a 7x1 cell array with joints' torques as measured by the integrated torque sensors.

5.10 sendJointsPositionsGetActualJpos

- Syntax:

```
>> [actual_JPOS] = sendJointsPositionsGetActualJpos(t_Kuka, target_jPos)
```

- Description:

  This function is used to send target joints' position to the robot for the soft real-time control at the joints level while simultaneously it returns a feed back about the actual joints positions as measured by the encoders integrated in the robot. This function can be used in both modes, the DirectServo mode and the impedance mode. This function is blocking, it awaits for the position message from the server before returning back to execution.

- Arguments:
  - `t_Kuka`: is the communication object.
  - `target_jPosos`: is a 7x1 cell array representing the target joints' positions, units in radians.

- Return values:
  - `actual_JPOS`: a 7x1 cell array with joints' actual positions as measured by the integrated sensors in the robot.

5.11 sendJointsPositionsGetActualEEFpos

- Syntax:

  ```
  >> [EEF_POS] = sendJointsPositionsGetActualEEFpos(t_Kuka, jPos)
  ```

- Description:

  This function is used to send target joints' position to the robot for the soft real-time control at the joints level while simultaneously it returns a feed back about the actual EEF positions as measured by the sensors integrated in the robot. This function can be used in both modes, the DirectServo mode and the impedance mode. This function is blocking, it awaits for the position message from the server before returning back to execution.

- Arguments:

  - `t_Kuka`: is the communication object.
  - `jPos`: is a 7x1 cell array representing the target joints' positions, units in radians.

- Return values:

  - `EEF_POS`: a 6x1 cell array with EEF actual positions as measured by the integrated sensors in the robot.

5.12 realTime_startDirectServoCartesian

- Syntax:

  ```
  >> realTime_startDirectServoCartesian(t_Kuka)
  ```

- Description:

  This function starts the DirectServo for controlling the robot in Cartesian space at the robot's EEF level.

- Arguments:

  - `t_Kuka`: is the communication object.

- Return values:

  - None.

- Related functions:

  - `sendEEfPosition`

- realTime_stopDirectServoCartesian

5.13 realTime_stopDirectServoCartesian

- Syntax:

  ```
  >> realTime_stopDirectServoCartesian(t_Kuka)
  ```

- Description:

  This function stop the DirectServo for controlling the robot in Cartesian space at the robot's EEF level.

- Arguments:

  - t_Kuka: is the communication object.

- Return values:

  - None.

- Related functions:

  - sendEEfPosition
  - realTime_startDirectServoCartesian

5.14 sendEEfPosition

- Syntax:

  ```
  >> [ret] = sendEEfPosition(t_Kuka, EEEFpos)
  ```

- Description:

  This function is used to set the target position of the End-Effector, this function is blocking, it awaits for an acknowledgment from the server before returning the execution.

- Arguments:

  - t_Kuka: is the communication object.
  - EEEFpos: is a 6x1 cell array where the first three elements represent (X, Y, Z) target positions of EEF (mm) and the last three elements represent the fixed rotation angles of EEF (radians).

- Return values:
  - ret: the return value is true if the position message has been received and processed successfully by the server, or false otherwise.
- Related functions:
  - realTime_startDirectServoCartesian
  - realTime_stopDirectServoCartesian

5.15 sendEEfPositionf

- Syntax:

  ```
  >> sendEEfPositionf(t_Kuka, EEEFpos)
  ```

- Description:

  This function is used to set the target position of the End-Effector, this function is not blocking, and does not wait for an acknowledgment from the server before returning the execution. As a result, this function is computationally light-weight, and it is designed for implementation in computationally expensive algorithms. When utilized in low-computational-cost algorithms, this function can execute very fast, as a result sending command packets to the robot at high rates (4K hz), this might cause execution issues, to solve this problem the user may need to perform some timing as to guarantee a transmission rate of around 275 packets/second.

- Arguments:
  - t_Kuka: is the communication object.
  - EEEFpos: is a 6x1 cell array where the first three elements represent (X, Y, Z) target positions of EEF (mm) and the last three elements represent the fixed rotation angles of EEF (radians).
- Return values:
  - None
- Related functions:
  - realTime_startDirectServoCartesian
  - realTime_stopDirectServoCartesian

5.16 sendEEfPositionExTorque

- Syntax:

  ```
  >> [ExTorque] = sendEEfPositionExTorque(t_Kuka, EEEFpos)
  ```

- Description:

  This function is used to send target position of EEF to the robot for the soft real-time control at the EEF level while simultaneously it returns a feed back about the external torques due to external forces acting on the robot. This function is blocking, it awaits for the torques message from the server before returning back to execution.

- Arguments:

  - `t_Kuka`: is the communication object.
  - `EEEFpos:` is a 6x1 cell array where the first three elements represent (X, Y, Z) target positions of EEF (mm) and the last three elements represent the fixed rotation angles of EEF (radians).

- Return values:

  - `ExTorque`: 7x1 cell array of the joint torques due to external forces acting on robot structure.

- Related functions:

  - `realTime_startDirectServoCartesian`
  - `realTime_stopDirectServoCartesian`

5.17 sendEEfPositionMTorque

- Syntax:

  ```
  >> [MT] = sendEEfPositionMTorque(t_Kuka, EEEFpos)
  ```

- Description:

  This function is used to send target position of EEF to the robot for the soft real-time control at the EEF level while simultaneously it returns a feed back about the measured torques due to external forces acting on the robot. This function is blocking, it awaits for the torques message from the server before returning back to execution.

- Arguments:

  - `t_Kuka`: is the communication object.

  - EEEFpos: is a 6x1 cell array where the first three elements represent (X, Y, Z) target positions of EEF (mm) and the last three elements represent the fixed rotation angles of EEF (radians).

- Return values:
  - MT: 7x1 cell array of the joint torques as measured by the sensors.

- Related functions:
  - realTime_startDirectServoCartesian
  - realTime_stopDirectServoCartesian

5.18 sendEEfPositionGetActualJpos

- Syntax:

```
>> [JPOS] = sendEEfPositionGetActualJpos(t_Kuka, EEEFpos)
```

- Description:

  This function is used to send target position of EEF to the robot for the soft real-time control at the EEF level while simultaneously it returns a feed back about the actual joints positions as measured by the integrated encoders. This function is blocking, it awaits for the position message from the server before returning back to execution.

- Arguments:
  - t_Kuka: is the communication object.
  - EEEFpos: is a 6x1 cell array where the first three elements represent (X, Y, Z) target positions of EEF (mm) and the last three elements represent the fixed rotation angles of EEF (radians).

- Return values:
  - JPOS: 7x1 cell array of the joint positions as measured by the encoders.

- Related functions:
  - realTime_startDirectServoCartesian
  - realTime_stopDirectServoCartesian

5.19 sendEEfPositionGetActualEEFpos

- Syntax:

  ```
  >> [actual_EEFpos] = sendEEfPositionGetActualEEFpos(t_Kuka, target_EEFpos)
  ```

- Description:

  This function is used to send target position of EEF to the robot for the soft real-time control at the EEF level while simultaneously it returns a feed back about the actual EEF position by measurements using the integrated encoders. This function is blocking, it awaits for the position message from the server before returning back to execution.

- Arguments:

  - `t_Kuka`: is the communication object.
  - `target_EEFpos:` is a 6x1 cell array where the first three elements represent (X, Y, Z) target positions of EEF (mm) and the last three elements represent the fixed rotation angles of EEF (radians).

- Return values:

  - `actual_EEFpos`: 6x1 cell array of EEF position.

- Related functions:

  - `realTime_startDirectServoCartesian`
  - `realTime_stopDirectServoCartesian`

5.20 realTime_startVelControlJoints

- Syntax:

  ```
  >> [ret] = realTime_startVelControlJoints(t_Kuka)
  ```

- Description:

  This function is used to start the soft real-time control at joints velocities level.

- Arguments:

  - `t_Kuka`: is the communication object.

- Return values:

  - `ret`: a boolean value, true if the function is executed successfully, false otherwise.

- Related functions:
  - sendJointsVel
  - realTime_stopVelControlJoints

5.21 realTime_stopVelControlJoints

- Syntax:

  ```
  >> realTime_stopVelControlJoints(t_Kuka)
  ```

- Description:

  This function is used to stop the soft real-time control at joints velocities level.

- Arguments:
  - t_Kuka: is the communication object.

- Return values:
  - None.

- Related functions:
  - sendJointsVel
  - realTime_startVelControlJoints

5.22 sendJointsVelocities

- Syntax:

  ```
  >> [ret] = sendJointsVelocities(t_Kuka, jvel)
  ```

- Description:

  This function is used to set reference joints' velocities for soft real-time control at joints level.

- Arguments:
  - t_Kuka: is the communication object.
  - jvel: is a 7x1 cell array representing the target angular velocity for each joint, in rad/sec.

- Return values:
  - ret: a boolean value, true if the function is executed successfully, false otherwise.
- Related functions:
  - realTime_startVelControlJoints
  - realTime_stopVelControlJoints

5.23 sendJointsVelocitiesExTorques

- Syntax:

```
>> [ExT] = sendJointsVelocitiesExTorques(t_Kuka, jvel)
```

- Description:

  This function is used to set reference joints' velocities for the soft real-time control at joints velocities level, simultaneously, this function returns a feed back about the external torques due to external forces acting on the robot. This function is blocking, it awaits for the torques message from the server before returning back to execution.

- Arguments:
  - t_Kuka: is the communication object.
  - jvel: is a 7x1 cell array representing the target angular velocity for each joint, in rad/sec.
- Return values:
  - ExT: 7x1 cell array of external torques.
- Related functions:
  - realTime_startVelControlJoints
  - realTime_stopVelControlJoints

5.24 sendJointsVelocitiesMTorques

- Syntax:

  ```
  >> [MT] = sendJointsVelocitiesMTorques(t_Kuka, jvel)
  ```

- Description:

  This function is used to set reference joints' velocities for the soft real-time control at joints velocities level, simultaneously, this function returns a feed back about the measured torques from integrated sensors in the robot joints. This function is blocking, it awaits for the torques message from the server before returning back to execution.

- Arguments:

  - t_Kuka: is the communication object.
  - jvel: is a 7x1 cell array representing the target angular velocity for each joint, in rad/sec.

- Return values:

  - MT: 7x1 cell array of measured torques.

- Related functions:

  - realTime_startVelControlJoints
  - realTime_stopVelControlJoints

5.25 sendJointsVelocitiesGetActualJpos

- Syntax:

  ```
  >> [JPoS] = sendJointsVelocitiesGetActualJpos(t_Kuka, jvel)
  ```

- Description:

  This function is used to set reference joints' velocities for the soft real-time control at joints velocities level, simultaneously, this function returns a feed back about the actual joints positions from integrated sensors in the robot. This function is blocking, it awaits for the position message from the server before returning back to execution.

- Arguments:

  - t_Kuka: is the communication object.
  - jvel: is a 7x1 cell array representing the target angular velocity for each joint, in rad/sec.

- Return values:
  - JPoS: 7x1 cell array of the actual positions of the joints.
- Related functions:
  - realTime_startVelControlJoints
  - realTime_stopVelControlJoints

5.26 sendJointsVelocitiesGetActualEEfPos

- Syntax:

```
>> [EEfPos] = sendJointsVelocitiesGetActualEEfPos (t_Kuka, jvel)
```

- Description:

  This function is used to set reference joints' velocities for the soft real-time control at joints velocities level, simultaneously, this function returns a feed back about the actual position of EEF of the robot. This function is blocking, it awaits for the position message from the server before returning back to execution.

- Arguments:
  - t_Kuka: is the communication object.
  - jvel: is a 7x1 cell array representing the target angular velocity for each joint, in rad/sec.
- Return values:
  - EEfPos: 6x1 cell array of the actual EEF position.
- Related functions:
  - realTime_startVelControlJoints
  - realTime_stopVelControlJoints

6 Point-to-point motion

This group of functions is used to move the robot towards a destination point, in such a case the path towards the destination point is planned by the software, so the user is not required to perform the path planning him-self.

6.1 movePTPJointSpace

- Syntax:

  ```
  >> [ret] = movePTPJointSpace(t_Kuka, jPos, relVel)
  ```

- Description:

  This function is used to move the robot from the current configuration to a new configuration in joint space.

- Arguments:

  - `t_Kuka`: is the communication object.
  - `jPos`: is a 7x1 cell array representing the target angular positions, in rad/sec.
  - `relVel`: is a double, from zero to one, specifying the override relative velocity.

- Return values:

  - `ret`: the return value is true if the command message has been received and processed successfully by the server, or false otherwise.

6.2 movePTPLineEEF

- Syntax:

  ```
  >> [ret] = movePTPLineEEF(t_Kuka, Pos, vel)
  ```

- Description:

  This function is used to move the end-effector in a straight line from the current position to a destination position. When called, the function causes the end-effector to move on a line. The robot can keep the orientation of the end-effector fixed or it can change the orientation while moving on a line.

- Arguments:

  - `t_Kuka`: is the communication object.

  - `Pos`: is a 6x1 cell array representing the destination position of EEF, the first three elements are the XYZ coordinates in (mm), the remaining three elements are the fixed rotation angles, in radians.
  - `vel`: linear velocity of the motion (mm/sec).

- Return values:
  - `ret`: the return value is true if the command message has been received and processed successfully by the server, or false otherwise.

6.3 movePTPHomeJointSpace

- Syntax:

```
>> [ret]= movePTPHomeJointSpace(t_Kuka, relVel)
```

- Description:

  This function is used to move the robot to home configuration.

- Arguments:
  - `t_Kuka`: is the communication object.
  - `relVel`: is a double, from zero to one, specifying the override relative velocity.

- Return values:
  - `ret`: the return value is true if the command message has been received and processed successfully by the server, or false otherwise.

6.4 movePTPTransportPositionJointSpace

- Syntax:

```
>> [ret] = movePTPTransportPositionJointSpace(t_Kuka, relVel)
```

- Description:

  This function moves the robot to the transport configuration.

- Arguments:
  - `t_Kuka`: is the communication object.

  - relVel: is a double, from zero to one, specifying the override relative velocity.

- Return values:
  - ret: the return value is true if the command message has been received and processed successfully by the server, or false otherwise.

6.5 movePTPLineEefRelBase

- Syntax:

```
>> [ret] = movePTPLineEefRelBase(t_Kuka, Pos, vel)
```

- Description:

  This function moves the end-effector in a straight-line path using relative motion, in such a case, the motion is defined using displacements along the axes of the robot base.

- Arguments:
  - t_Kuka: is the communication object.
  - Pos: is a 3x1 cell array representing the XYZ displacements along the base axes by which the EEF has to move, unit is (mm).
  - vel: linear velocity of the motion (mm/sec).

- Return values:
  - ret: the return value is true if the command message has been received and processed successfully by the server, or false otherwise.

6.6 movePTPLineEefRelEef

- Syntax:

```
>> [ret] = movePTPLineEefRelEef(t_Kuka, Pos, vel)
```

- Description:

  This function moves the end-effector in a straight-line path using relative motion, in such a case, the motion is defined using displacements along the axes of EEF.

- Arguments:

  - t_Kuka: is the communication object.
  - Pos: is a 3x1 cell array representing the XYZ displacements along the axes of EEF by which the EEF has to move, unit is (mm).
  - vel: linear velocity of the motion (mm/sec).

- Return values:
  - ret: the return value is true if the command message has been received and processed successfully by the server, or false otherwise.

6.7 movePTPCirc1OrintationInter

- Syntax:

```
>> [ret] = movePTPCirc1OrintationInter(t_Kuka, f1, f2, vel)
```

- Description:

  This function moves the end-effector in arc specified by two frames.

- Arguments:
  - t_Kuka: is the communication object.
  - f1: is 6x1 cell array, specifying the intermediate frame of the arc.
  - f2: is 6x1 cell array, specifying the final frame of the arc. In both frames, the first three elements are the X, Y and Z coordinates in (mm), the remaining elements are the fixed rotation angles (radians).
  - vel: linear velocity of the motion (mm/sec).

- Return values:
  - ret: the return value is true if the command message has been received and processed successfully by the server, or false otherwise.

6.8 movePTPArc_AC

- Syntax:

```
>> [ret] = movePTPArc_AC(t_Kuka, theta, c, k, vel)
```

- Description:

  This function moves the end-effector in arc specified by the arc's: center, starting point, angle and a normal to its plane.

- Arguments:
  - t_Kuka: is the communication object.
  - theta: is the arc'a angle in radians.
  - c: is a 3x1 vector representing the x, y and z coordinates of the arc's center.
  - k: is a 3x1 vector representing the normal vector on the plane of the arc.
  - vel: linear velocity of the motion (mm/sec).
- Return values:
  - ret: the return value is true if the command message has been received and processed successfully by the server, or false otherwise.

6.9 movePTPArcXY_AC

- Syntax:

```
>> [ret] = movePTPArcXY_AC(t_Kuka, theta, c, vel)
```

- Description:

  This function moves the end-effector in an arc parallel to the XY plane of the base of the robot, the arc is specified by its: center, starting point, and angle.
- Arguments:
  - t_Kuka: is the communication object.
  - theta: is the arc'a angle in radians.
  - c: is a 2x1 vector representing the x and y coordinates of the arc's center.
  - vel: linear velocity of the motion (mm/sec).
- Return values:
  - ret: the return value is true if the command message has been received and processed successfully by the server, or false otherwise.
- Similar functions:
  - movePTPArcXZ_AC
  - movePTPArcYZ_AC

7 Setters

Those functions are used to set the outputs of the pneumatic flange of the KUKA iiwa robot, in case the user is using a manipulator with another flange, then the functions can still be called but they will have no effect on the robot.

7.1 setBlueOff

- Syntax:

  ```
  >> [ret]= setBlueOff(t_Kuka)
  ```

- Description:

  This function is used to turn off the blue light of the pneumatic flange.

- Arguments:

  - t_Kuka: is the communication object.

- Return values:

  - ret: the return value is true if the command message has been received and processed successfully by the server, or false otherwise.

7.2 setBlueOn

- Syntax:

  ```
  >> [ret]= setBlueOn(t_Kuka)
  ```

- Description:

  This function is used to turn on the blue light of the pneumatic flange.

- Arguments:

  - t_Kuka: is the communication object.

- Return values:

  - ret: the return value is true if the command message has been received and processed successfully by the server, or false otherwise.

7.3 setPin1Off

- Syntax:

  ```
  >> [ret]= setPin1Off(t_Kuka)
  ```

- Description:

  This function is used to set off the output (Pin 1) of the output connector of the pneumatic media flange.

- Arguments:

  - t_Kuka: is the communication object.

- Return values:

  - ret: the return value is true if the command message has been received and processed successfully by the server, or false otherwise.

7.4 setPin1On

- Syntax:

  ```
  >> [ret]= setPin1On(t_Kuka)
  ```

- Description:

  This function is used to set on the output (Pin 1) of the pneumatic media flange.

- Arguments:

  - t_Kuka: is the communication object.

- Return values:

  - ret: the return value is true if the command message has been received and processed successfully by the server, or false otherwise.

8 Getters

Those functions are used to get a feedback about the various parameters and sensory-measurements from the robot.

8.1 getEEF_Force

- Syntax:

  ```
  >> [f] = getEEF_Force(t_Kuka)
  ```

- Description:

  This function is used to acquire the force at the flange reference frame. The components of the force are described in the base reference frame of the robot.

- Arguments:

  - `t_Kuka`: is the communication object.

- Return values:

  - `f`: is a 1x3 cell array representing the X, Y and Z components of the force (Newton).

8.2 getEEF_Moment

- Syntax:

  ```
  >> [m] = getEEF_Moment(t_Kuka)
  ```

- Description:

  This function returns the measured moment at the flange reference frame. The components of the moment are described in the base frame of the robot.

- Arguments:

  - `t_Kuka`: is the communication object.

- Return values:

  - `m`: is a 1x3 cell array representing the X, Y and Z components of the force (Newton).

8.3 getEEFCartesianOrientation

- Syntax:

  ```
  >> [ori] = getEEFCartesianOrientation(t_Kuka)
  ```

- Description:

  This function returns the orientation (Z, Y and X fixed rotations angles) in radians.

- Arguments:

  - `t_Kuka`: is the communication object.

- Return values:

  - `ori`: is a 1x3 cell array representing the ZYX fixed rotation angles of EEF.

8.4 getEEFCartesianPosition

- Syntax:

  ```
  >> [Pos] = getEEFCartesianPosition(t_Kuka)
  ```

- Description:

  This function returns the Cartesian position of the end-effector relative to robot base reference frame.

- Arguments:

  - `t_Kuka`: is the communication object.

- Return values:

  - `Pos`: is a 1x3 cell array representing the XYZ coordinates of EEF in base frame of robot (mm).

8.5 getEEFPos

- Syntax:

  ```
  >> [Pos] = getEEFPos(t_Kuka)
  ```

- Description:

  This function returns the position and orientation of the end-effector relative to robot base reference frame.

- Arguments:

  - t_Kuka: is the communication object.

- Return values:

  - Pos: is a 1x6 cell array, first three elements represent the XYZ coordinates of EEF (mm), remaining three elements represent the ZYX fixed rotation angles of EEF (radians).

8.6 getJointsExternalTorques

- Syntax:

  ```
  >> [torques] = getJointsExternalTorques(t_Kuka)
  ```

- Description:

  This function returns the robot joints' torques due to external forces.

- Arguments:

  - t_Kuka: is the communication object.

- Return values:

  - torques: is a 1x7 cell array of joints torques, in Newton.Meter.

8.7 getJointsMeasuredTorques

- Syntax:

  ```
  >> [torques] = getJointsMeasuredTorques(t_Kuka)
  ```

- Description:

  This function returns the robot joints' torques as measured by the integrated torque sensors.

- Arguments:

  - t_Kuka: is the communication object.

- Return values:

  - torques: is a 1x7 cell array of joints torques, in Newton.Meter.

8.8 getJointsPos

- Syntax:

  ```
  >> [jPos] = getJointsPos(t_Kuka)
  ```

- Description:

  This function returns the robot joints' angles.

- Arguments:

  - t_Kuka: is the communication object.

- Return values:

  - jPos: is a 1x7 cell array of actual joints positions as measured by the encoders, in radians.

8.9 getMeasuredTorqueAtJoint

- Syntax:

  ```
  >> [torque] = getMeasuredTorqueAtJoint(t_Kuka, k)
  ```

- Description:

  This function returns the measured torque in a specific joint.

- Arguments:
  - t_Kuka: is the communication object.
  - k: is the joint index, from 1 to 7.
- Return values:
  - torque: is the measured torque at joint k, torque unit is Newton.Meter.

8.10 getExternalTorqueAtJoint

- Syntax:

```
>> [torque] = getExternalTorqueAtJoint(t_Kuka, k)
```

- Description:

  This function returns the external torque in a specific joint.

- Arguments:
  - t_Kuka: is the communication object.
  - k: is the joint index, from 1 to 7.
- Return values:
  - torque: is the external torque at joint k, torque unit is Newton.Meter.

8.11 getEEFOrientationR

- Syntax:

```
>> [rMat] = getEEFOrientationR(t_Kuka)
```

- Description:

  This function returns the rotation matrix representing the orientation of the EEF relative to the base frame of the robot.

- Arguments:
  - t_Kuka: is the communication object.
- Return values:

– `rMat`: 3x3 rotation matrix representing the orientation of the EEF relative to the base frame of the robot.

8.12 getEEFOrientationQuat

- Syntax:

  ```
  >> [quaternion] = getEEFOrientationQuat(t_Kuka)
  ```

- Description:

  This function returns the quaternion representing the orientation of the EEF relative to the base frame of the robot.

- Arguments:

  – `t_Kuka`: is the communication object.

- Return values:

  – `quaternion`: 1x4 quaternion vector representing the orientation of the EEF relative to the base frame of the robot.

9 General purpose

Those functions are used for calculating the direct/inverse kinematics, the direct/inverse dynamics, and the various physical quantities of the iiwa 7 R 800 manipulator, it is worth to mention that in the following functions the base of the robot is considered to be mounted in the horizontal position.

9.1 gen_DirectKinematics

- Syntax:

  ```
  >> [eef_transform, J] = gen_DirectKinematics(q, TefTool)
  ```

- Description:

  This function calculates the forward kinematics/Jacobian of the manipulator.

- Arguments:

  - q: the angles of the robot joints.
  - TefTool: the transform matrix from the tool frame to the flange of the robot.

- Return values:

  - eef_transform: is the transformation matrix from end-effector to the base frame of the robot.
  - J: is the Jacobian at the tool center point of the EEF.

9.2 gen_partialJacobean

- Syntax:

  ```
  >> [Jp] = gen_partialJacobean(q, linkNum ,Pos)
  ```

- Description:

  This function calculates the partial Jacobian.

- Arguments:

  - q: is a 7x1 vector with joints angle.
  - linkNum: is the number of the link at which the partial Jacobian is associated, it shall be an integer in the range [1,7].
  - Pos: is a 3x1 vector that represents the position vector of the point where the partial Jacobian is going to be calculated

- Return values:
  - `Jp`: is the partial Jacobian.

9.3 gen_InverseKinematics

- Syntax:

```
>> [qs] = gen_InverseKinematics(qin, Tt, TefTool, n, lambda)
```

- Description:

  This function calculates the inverse kinematics.

- Arguments:
  - `qin`: is the initial (or current) joints' angles of the robot (radians).
  - `Tt`: is the target transformation matrix position/orientation of the robot.
  - `TefTool`: the transform matrix from the tool to the flange of the robot.
  - `n`: is the number of iterations
  - `lambda`: is the damping constant.

- Return values:
  - `qs`: is the solution joint angle of the robot.

9.4 gen_MassMatrix

- Syntax:

```
>> [M] = gen_MassMatrix(q, Pcii, Icii, mcii)
```

- Description:

  This function calculates the mass matrix.

- Arguments:
  - `q`: is a 1x7 vector with angles of the manipulator,
  - `Pcii`: is a 3x7 matrix where each column represents the position vector of the center of mass of the associated link represented in that link's local frame.

  - `Icii`: is a 3x3x7 matrix, where each 3x3 matrix represents the inertial tensor of the associated link described in its local inertial frame.
  - `mcii`: is a 1x7 vector representing the masses of the links (kg).

- Return values:
  - `M`, is a 7x7 matrix – the mass matrix of the manipulator.

9.5 gen_CoriolisMatrix

- Syntax:

```
>> [B] = gen_CoriolisMatrix(q , Pcii, Icii, mcii, dq)
```

- Description:

  This function calculates Coriolis matrix.

- Arguments:
  - `q`: is a 1x7 vector with angles of the manipulator,
  - `Pcii`: is a 3x7 matrix where each column represents the position vector of the center of mass of the associated link represented in that link's local frame.
  - `Icii`: is a 3x3x7 matrix, where each 3x3 matrix represents the inertial tensor of the associated link described in its local inertial frame.
  - `mcii`: is a 1x7 vector representing the masses of the links (kg).
  - `dq`: is a 1x7 vector with the joints' angular velocity.

- Return values:
  - `B`, is a 7x7 matrix – the Coriolis matrix of the robot.

9.6 gen_CentrifugalMatrix

- Syntax:

```
>> [C] = gen_CentrifugalMatrix(q, Pcii, Icii, mcii, dq)
```

- Description:

  This function calculates centrifugal matrix.

- Arguments:

  - q: is a 1x7 vector with angles of the manipulator,
  - Pcii: is a 3x7 matrix where each column represents the position vector of the center of mass of the associated link represented in that link's local frame.
  - Icii: is a 3x3x7 matrix, where each 3x3 matrix represents the inertial tensor of the associated link described in its local inertial frame.
  - mcii: is a 1x7 vector representing the masses of the links (kg).
  - dq: is a 1x7 cell array with the joints' angular velocity.

- Return values:
  - C: is a 7x7 matrix – the centrifugal matrix of the manipulator.

9.7 gen_GravityVector

- Syntax:

  ```
  >> [G] = gen_GravityVector(q, Pcii, mcii)
  ```

- Description:

  This function calculates joints torques due to gravity.

- Arguments:
  - q: is a 1x7 vector with angles of the manipulator,
  - Pcii: is a 3x7 matrix where each column represents the position vector of the center of mass of the associated link represented in that link's local frame.
  - mcii: is a 1x7 vector representing the masses of the links (kg).

- Return values:
  - G: is a 7x1 vector – the gravity vector of the manipulator.

9.8 gen_DirectDynamics

- Syntax:

  ```
  >> [d2q] = gen_DirectDynamics(q, Pcii, Icii, mcii, dq, taw)
  ```

- Description:

  This function calculates the direct dynamics of the manipulator.

- Arguments:

  - q: is a 1x7 vector with angles of the manipulator,
  - Pcii: is a 3x7 matrix where each column represents the position vector of the center of mass of the associated link represented in that link's local frame.
  - Icii: is a 3x3x7 matrix, where each 3x3 matrix represents the inertial tensor of the associated link described in its local inertial frame.
  - mcii: is a 1x7 vector representing the masses of the links (kg).
  - dq: is a 1x7 vector with the joints' angular velocity.
  - taw: is the torques at the joints.

- Return values:

  - d2q: is a 7x1 vector representing the angular accelerations of the joints.

9.9 gen_InverseDynamics

- Syntax:

  ```
  >> [taw] = gen_InverseDynamics(q, Pcii, Icii, mcii, dq, d2q)
  ```

- Description:

  This function calculates the inverse dynamics of the manipulator.

- Arguments:

  - q: is a 1x7 vector with angles of the manipulator,
  - Pcii: is a 3x7 matrix where each column represents the position vector of the center of mass of the associated link represented in that link's local frame.
  - Icii: is a 3x3x7 matrix, where each 3x3 matrix represents the inertial tensor of the associated link described in its local inertial frame.
  - mcii: is a 1x7 vector representing the masses of the links (kg).

  - `dq`: is a 1x7 vector with the joints' angular velocity.
  - `d2q`: is a 1x7 vector representing the angular accelerations of the joints.

- Return values:
  - `taw`: is the torques at the joints

9.10 gen_NullSpaceMatrix

- Syntax:

```
>> [N] = gen_NullSpaceMatrix(q)
```

- Description:

  This function calculates the null space matrix of the robot.

- Arguments:
  - `q`: is a 1x7 vector with angles of the manipulator.

- Return values:
  - `N`: is 7x7 matrix – a null space projection matrix for the KUKA iiwa 7 R 800.

10 Physical Interaction

This category contains functions that are used for physical human robot interaction.

10.1 startHandGuiding

- Syntax:

  ```
  >> startHandGuiding(t_Kuka)
  ```

- Description:

  This function is used to start KUKA's off-the-shelf the hand guiding for a manipulator with a pneumatic flange. Once called, the hand-guiding is initiated, afterwards to perform the hand-guiding the user has to press the flange's white button to deactivate the brakes and move the robot around. To terminate this functionality, the user has to press the green button continuously for more than 1,5 seconds, in such a case, after 1,5 seconds the blue LED light starts to flicker, when the green button is released the function is terminated.

- Arguments:

  - `t_Kuka`: is the TCP/IP communication object.

- Return values:

  - None.

10.2 startPreciseHandGuiding

- Syntax:

  ```
  >> startPreciseHandGuiding(t_Kuka, wot, com)
  ```

- Description:

  This function is used to initialize a precise hand-guiding functionality.

- Arguments:

  - `t_Kuka`: is the TCP/IP communication object.
  - `wot`: is the weight of the tool (Newton).
  - `com`: is 1x3 vector, representing the coordinates of the center of mass of the tool described in the reference frame of the flange (mm).

- Return values:

- None.

10.3 performEventFunctionAtDoubleHit

- Syntax:

  ```
  >> performEventFunctionAtDoubleHit(t_Kuka)
  ```

- Description:

  This function is used to detect the double touch in Z direction of the end-effector.

- Arguments:

  - t_Kuka: is the TCP/IP communication object.

- Return values:

  - None.

10.4 EventFunctionAtDoubleHit

- Syntax:

  ```
  >> EventFunctionAtDoubleHit()
  ```

- Description:

  This function is called when a double touch in the Z direction of the robot's end-effector is detected.

- Arguments:

  - None.

- Return values:

  - None.

10.5 moveWaitForDTWhenInterrupted

- Syntax:

  `>> moveWaitForDTWhenInterrupted (t_Kuka, Pos, VEL, joints_indexes, max_torque, min_torque, w)`

- Description:

  This function is used to perform an interruptible point-to-pint motion of the end-effector on a line. If the value of torque on any of the specified joints `joints_indexes` exceeds the predefined torque limits, `max_torque/ min_torque`, the motion is interrupted. Afterwards, the robot waits for a double tap on the end-effector along the Z axis, upon which the robot returns to motion execution again.

- Arguments:

  - `t_Kuka`: is the TCP/IP communication object. is the coordinates of the center of mass of the tool with respect to the reference frame of the flange.
  - `Pos`: is 1x6 cell array of EEF position.
  - `VEL`: is the linear velocity of the motion (mm/sec).
  - `joints_indexes`: indexes of the joints where torques limits are to be specified.
  - `max_torque`: the maximum torques limits of the joints specified by `joints_indexes`.
  - `min_torque`: the maximum torques limits of the joints specified by `joints_indexes`.
  - `w`: is the weight of the tool (Newton).

- Return values:

  - None.