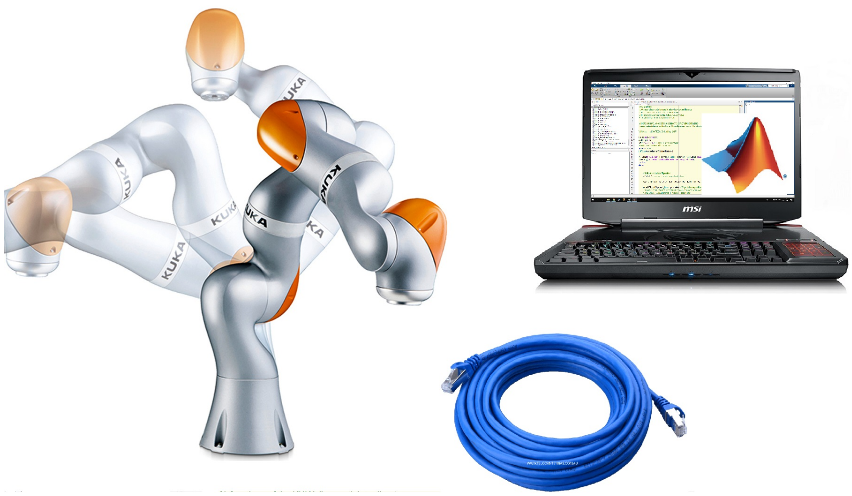
User's manual: KUKA Sunrise Toolbox



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.

Document version 1.84

26th-August-2018

Contents

List of Figures

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, under MIT license, from 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`

and the link (recommended but still in progress):

`https://www.youtube.com/playlist?list=PLz558OYgHuZd-Gc2-OryITKEXefAmrvae`

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

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 class, in addition it lists the various methods provided by the Toolbox along with a briefing about their functionality and their way of use. Each method is elaborated in a subsection with the following main entries:

- Syntax: lists the MALAB syntax for calling the method in subject, written in a MATLAB script style.
- Description: a brief description about the 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.

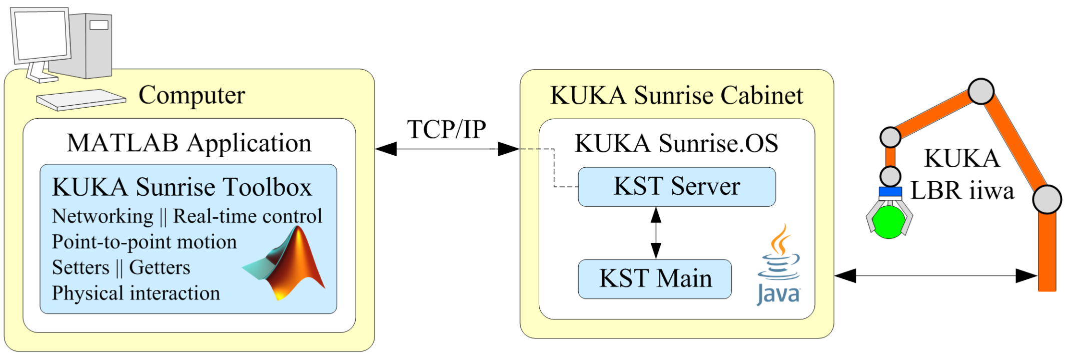


Fig. 0.1: Architecture of the toolbox

Some methods may contain one or both of the following extra entries:

- About: used to give a prerequisite info or an in-depth explanation about the method in subject or about the robot's operation mode, as in the "precise hand-guiding method".
- Tutorial script: this entry refers to the file of the repository where the method in subject is used, the tutorial script is a MATLAB script (m file), the user can open it using MATLAB, it is documented with comments. For a hands on experience the user can run this script to control the robot.

Architecture of KST

The KST has a client server architecture, Figure 0.1, the server is a multi-threaded Java application that runs inside the robot controller. Before starting to utilize the toolbox the user shall synchronize the server application into the robot controller, after being synchronized the user shall run the server, once is running, the toolbox can be used to connect to the robot from Matlab.

On the other hand, the client part is written using Matlab scripting language, the various functions offered by the toolbox are wrapped inside the KST.m class, the user can use this class to control the robot from Matlab.

Part II. Getting started, the requirements

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

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.

Software requirements

The following software packages are required:

- MATLAB: the user shall have MATLAB installed on 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.

Synchronizing KST server to the controller

The server application that runs in the robot controller is named (MatlabToolboxServer), it is a Java application written using Kuka's Sunrsie.Workbench program.

The source code of the server application is given inside the folder ***KUKA Sunrise server source code,*** which is found inside the toolbox repository, several versions of the server are provided, for more info the user is advised to read the ***About.txt*** file which is found along with the server source code. This server application shall be synchronized to the robot controller and shall be running before controlling the robot from an external computer using Matlab.

The steps of synchronizing the MatlabToolboxServer application into the robot controller is described in the video tutorial:

`https://youtu.be/fhzCyQRUNiA?list=PLz558OYgHuZd-Gc2-OryITKEXefAmrvae`

The same steps are also described in the file, ***Import KST to Sunrise-Workbench.pdf*** , which is found in the root folder of the toolbox repository.

Part III. Tutorial scripts

The toolbox is provided with Tutorial examples (Demos), written in MATLAB scripting language. The user may consult those examples for a quick start up. The examples vary in difficulty, some are simple with straightforward implementation of KST motion functions, others are more complex and show the user how to utilize the full power of the toolbox by utilizing KST's motion functions along with direct/inverse kinematics, implementing graphical user interfaces and integrating other MALAB toolboxes and/or external hardware to control the manipulator.

How to run the provided examples

To run any of the provided examples on your own iiwa, the user shall follow the steps:

- Using MATLAB, open the example that you want to run.
- Go to the line where the IP of the robot is defined, shown in the code snippet Fig 0.2, change its value into the IP of your own robot.
- Go to the lines where the type of the robot and the type of the flange are defined, Fig 0.2, change them according to the configuration of your own robot/flange combination.
- Go to the line where the transformation matrix of the Tool Center Point (TCP) is defined, change it according to the tool attached to the flange.
- Start the MatlabToolboxServer application from the teach pendant of the robot, shown in Fig 0.3.

```
%% Create the robot object
ip='172.31.1.147'; % The IP of the controller
arg1=KST.LBR7R800; % choose the robot iiwa7R800 or iiwa14R820
arg2=KST.Medien_Flansch_elektrisch; % choose the type of flange
Tef_flange=eye(4); % transofrm matrix of EEF with respect to flange
iiwa=KST(ip,arg1,arg2,Tef_flange); % create the object
```

Fig. 0.2: Code snippet from examples, defining IP , type of robot/flange.

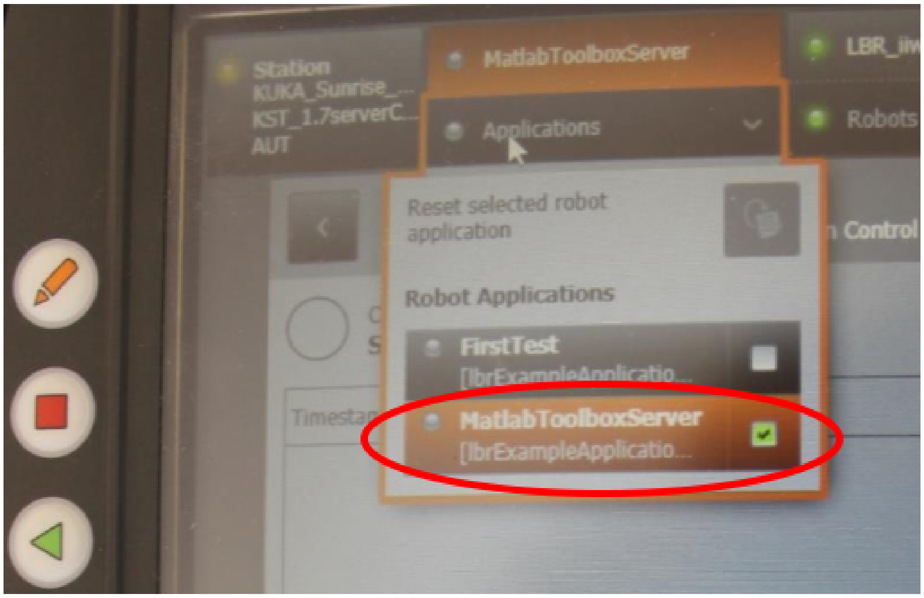


Fig. 0.3: Starting the MatlabToolboxServer application from teach pendant.

- Run The MATLAB script example that you chose.
- Note that by default, the MatlabToolboxServer application turns off automatically if a connection is not established during sixty seconds interval from its initiation. If this happened, you shall turn on the application manually before starting a connection again.

Total list of examples

Tutorial Script	Description
KSTclass_Tutorial_getters.m	Demonstrates how to utilize the getters functions supported by KST, which are used to read the various parameters of the robot.
KSTclass_Tutorial_generalPorpuse .m	Demonstrates how to utilize the general purpose functions of the KST for calculating: Jacobian matrix, mass matrix, centrifugal matrix, Coriolis matrix. In addition it demonstrates how to use KST for calculating direct/inverse kinematics, direct/inverse dynamics, etc.
KSTclass_Tutorial_move_ptp.m	Demonstrates how to utilize the point to point motion functions supported by KST.
KSTclass_Tutorial_circles.m	Demonstrates how to utilize the various circle motion functions supported by KST.
KSTclass_Tutorial_ptpConditional Torques.m	Demonstrates how to utilize the conditional point to point motion functions supported by the KST. In those functions the motion can be interrupted during physical interaction with the robot, such is the case if one of the joint's torques exceeds a predefined limit.
KSTclass_Tutorial_nonBlocking PTPLinesJoints.m	Demonstrates how to utilize the non blocking functions provided by KST toolbox.
KSTclass_Tutorial_ptpPickPlace PhysicalInteraction.m	A pick and place application. When executed, the robot manipulates an object from one position to another using a pneumatic gripper attached to the media flange. During the motion, the conditional point to point motion functions of KST are utilized, as such if the torque on any of the joints exceeds a predefined value, for example: due to physical interaction with the coworker, the robot stops in place and waits for the coworker to double tap its EEF so it returns back to normal operation.

Tab. 1: Tutorial scripts provided with the KST repository (1)

Tutorial Script	Description
KSTclass_Tutorial_directServo Cartesian.m	Demonstrates how to utilize KST's soft-real-time motion functionalities based on KUKA's DirectServo interface. In this example the robot is in joints position control mode, the motion of the first joint of the robot is controlled by streaming joints destination angles to the robot. In this example the motion of the first joint is specified by a sinusoidal function, while the other joints angles are kept fixed.
KSTclass_Tutorial_softRealTime JointsVelControl.m	Demonstrates how to utilize KST's soft-real-time motion functionalities based on KUKA's DirectServo interface. In this example the robot is in joints velocity control mode, the motion of the first joint of the robot is controlled by streaming the joints angular velocities to the robot. The motion of the first joint is specified by a sinusoidal function.
KSTclass_Tutorial_realTime ImpedencePlotTorqueFeedBack.m	Demonstrates how to utilize KST's soft-real-time impedance motion functionalities based on KUKA's SmartServo interface. In this example the robot is in impedance control mode, the motion of the various joints of the robot is controlled by streaming angular positions to the robot. The motion of the joints are specified by a sinusoidal function. At the same time Matlab plots a graph of joints torques as acquired from the robot.

Tab. 2: Tutorial scipts provided with the KST repository (2)

Tutorial Script	Description
KSTclass_Tutorial_teleOperation.m	In this example the user can control the EEF of the robot using a 3Dconnexion SpaceMouse. To run this example the user shall first install the appropriate 3Dconnexion drivers, then he/she shall connect the SpaceMouse to the PC. In general, this example demonstrates how to use external hardware to control the robot, in addition it demonstrates how to use the direct/inverse kinematics functions provided by KST.
KSTclass_Tutorial_iiwa_vrep.m	Example of controlling KUKA iiwa from 3D simulation by using KST as a middle-ware. The simulator used is V-rep. This software is a powerful robotic simulation environment that offers users the option to control their 3D simulations from MATLAB. In such a case, by using KST the user can control KUKA iiwa manipulator directly from his/her 3D simulation developed using V-rep & MATLAB. This example is provided with V-rep simulation scene: iiwaFromVrep.ttt
KSTclass_Tutorial_gampade_EEFPosControl.m	In this example the user is able to use a game pad for controlling the position and orientation of the EEF of KUKA iiwa robot. This example demonstrates how to integrate external hardware to control the robot, at the same time it demonstrates how to utilize direct/inverse kinematics functions of KST toolbox.
KSTclass_Tutorial_gampade_jointsPosControl.m	In this example the user is able to use a game pad for controlling joints positions of KUKA iiwa robot. This example demonstrates how to integrate external hardware to control the robot, at the same time it demonstrates how to integrate a graphical user interface into the application.

Tab. 3: Tutorial scripts provided with the KST repository (3)

Part IV. KST class

Starting from version KST_1.7, object oriented programming is implemented, as such the various functionalities of the toolbox is encapsulated inside the KST.m class, this class contains various properties and methods, using which the user can control the robot. To get an access to those methods, the user shall instantiate an object from the KST class first, as in the following:

```
>> iiwa = KST(ip,robot_Type,flange_Type,Tef_flange);
```

Where:

- `iiwa`: is an instance of the KST class.
- `ip`: a string variable which contains the IP of the robot, e.g. “172.31.1.147”. In order to figure out the IP of your robot, first go to the teach pendant of the robot and click on the button “Station” that is found at the top side of the teach pendant's screen, then click on the button “Information”, finally you can find the IP of the robot listed under the title “Station Server IP”.
- `robot_Type`: is a constant referring to robot type 7R800 or 14R820, please refer to section V **Properties of KST class** for more info.
- `flange_Type`: is a constant referring to flange's type, please refer to section V **Properties of KST class** for more info.
- `Tef_flange`: is 4x4 matrix, representing the transformation matrix of EEF with respect to the frame of the flange. `Tef_flange` is an optional argument if omitted the KST uses the identity matrix by default.

Example:

In this example an instance of the KST class associated with LBR7R800 robot is created.

```
>> ip='172.31.1.147';
>> robot_Type=KST.LBR7R800;
>> flange_Type=KST.Medien_Flansch_elektrisch;
>> Tef_flange=eye(4);
>> iiwa = KST(ip,robot_Type,flange_Type,Tef_flange)
```

Fields of **I_data**	Description
m	1x7 matrix of links masses
pcii	3x7 matrix, each column specifies the position vector of Center Of Mass (COM) of each link described in its local frame
I	3x3x7 matrix, each 3x3 matrix specifies the inertial tensor of a link
Fs	1x7 matrix of joints Coulomb friction
Fv	1x7 matrix of joints viscous friction coefficients

Tab. 4: Different data fields of the property, **I_data**, from an instance of KST class.

Part V. Properties of KST class

KST class implements several properties, some are read only while others can be modified. The different properties are:

1. ip: a string variable specifying the IP of the robot, this variable is assigned when the constructor of the KST class is called.

2. I_data: a structure with several fields, Table 4, used to specify the inertial/dynamic constants of the robot, this structure is loaded automatically when the constructor of the KST class is called. The appropriate data of the LBR7R800 or the LBR14R820 is loaded according to the type of robot specified when the constructor of the KST class is called. The inertial data identified in [1] are used for the LBR14R820, while the inertial data identified [2] in are used for LBR7R800

3. dh_data: a structure with DH parameters of the robot, the appropriate DH parameters, of the LBR7R800 or the LBR14R820, are loaded automatically according the the type of robot specified when the constructor of the KST class is called.

4. Teftool: is the transformation matrix from the frame of EEF to the frame of the flange.

Also the toolbox provides the following two groups of enumerations, one is used to specify the robot's type, Table 5, while the other is used to specify the flange's type, Table 6.

Property	Description
LBR7R800	For specifying 7 R 800 robot
LBR14R820	For specifying 14 R 820 robot

Tab. 5: Robot's type enumeration

Property	Description
Medien_Flansch_elektrisch	For an electric median flange
Medien_Flansch_pneumatisch	For a pneumatic flange
Medien_Flansch_IO_pneumatisch	For a pneumatic flange with IOs
Medien_Flansch_Touch_pneumatisch	For a touch pneumatic flange

Tab. 6: Flange type enumeration

Part VI. Methods of KST class

In this part of the documentation the various methods supported by KST are listed, for each method a brief description about its function is presented along with a MATLAB snippet code showing its way of use. In the following the variable `iiwa` is used as an instance of the KST class.

1 Networking

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

1.1 `net_establishConnection`

- Syntax:

  ```
  >> iiwa.net_establishConnection()
  ```

- Description:

  This method is used to establish a connection with the **MatlabToolboxServer** application on the controller of the robot, before calling this method make sure that you: (1) have synchronized the Java application **MatlabToolboxServer** into the controller of the robot, (2) make sure that you have a network established between the PC and the controller, to check this you can use the method **net_pingIIWA** described in subsection 1.4, (3) make sure that the Java application **MatlabToolboxServer** is running in the smart-pad of the robot before calling this method, notice that by default the application **MatlabToolboxServer** has a time out of 60 seconds, if the user did not call the method, **net_establishConnection**, during this time interval, then the application **MatlabToolboxServer** will turn off automatically, in such a case you have to restart it before being able to connect to the robot again.

- Arguments:

  - `None`.

- Return values:

  - `None`.

- Tutorial script:

  - `KSTclass_Tutorial_plotNetHealth.m`

1.2 `net_turnOffServer`

- `Syntax:`

  `>> iiwa.net_turnOffServer()`

- Description:

  This method is used to turn off the server application **MatlabToolboxServer** on the robot controller, when you are done using the KST toolbox for controlling iiwa you shall turn off the application **MatlabToolboxServer** by calling this method.

- Arguments:
  - None.

- Return values:
  - None.

- Tutorial script:
  - `KSTclass_Tutorial_plotNetHealth.m`

1.3 `net_updateDelay`

- `Syntax:`

  `>> [time_stamps,time_delay] = iiwa.net_updateDelay()`

- Description:

  This method is used to establish a plot of the communication delays between the computer and the controller, i. e., it tests the timing characteristics of the connection between KUKA iiwa and the computer. Low timing delay is important when utilizing the soft real-time control methods in-order to achieve a jitter-less motion, due to the importance of this point, the repository provides instructions on how to optimize the network connection for achieving low time delays, those instructions are presented in the **Tips and tricks** folder of the KST repository.

- Arguments:
  - None.

- Return values:
  - `time_stamps`: is the time stamp of each socket.

- `time_delay`: is the delay for each socket.

- Tutorial script:
  - `KSTclass_Tutorial_plotNetHealth.m`

1.4 net_pingIIWA

- **Syntax:**

  ```
  >> iiwa.net_pingIIWA()
  ```

- Description:

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

- Arguments:
  - None.

- Return values:
  - None.

- Tutorial script:
  - `KSTclass_Tutorial_networking.m`

2 Soft real-time control

In this group of methods 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 methods along with utilization instructions.

2.1 `realTime_moveOnPathInJointSpace`

- `Syntax:`

  `>> iiwa.realTime_moveOnPathInJointSpace(trajectory, delayTime)`

- Description:

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

- Arguments:

  - `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 consecutive configurations.

- Return values:

  - None.

- Tutorial script:

  - `KSTclass_Tutorial_do_some_stuff.m`

2.2 `realTime_startDirectServoJoints`

- `Syntax:`

  `>> iiwa.realTime_startDirectServoJoints()`

- Description:

  This method is used to turn on the DirectServo functionality in the robot for initiating the soft real-time control in joint space. After starting DirectServo functionality, the user have to stream the joints' target positions to the robot using the method `(sendJointsPositions)` for example, several other methods can also be used for streaming the target positions, for a list of those methods please refer to the **Related methods** bullet point below.

- Arguments:
  - None.
- Return values:
  - None.
- Related methods:
  - sendJointsPositions
  - sendJointsPositionsf
  - sendJointsPositionsExTorque
  - sendJointsPositionsMTorque
  - sendJointsPositionsGetActualJpos
  - sendJointsPositionsGetActualEEFpos
  - realTime_stopDirectServoJoints
- Tutorial script:
  - KSTclass_Tutorial_directServo.m

2.3 realTime_stopDirectServoJoints

- Syntax:

  `>> iiwa.realTime_stopDirectServoJoints()`
- Description:

  This method is used to turn off the DirectServo function on the robot ending the soft real-time control in joints space.
- Arguments:
  - None.
- Return values:
  - None.
- Related methods:
  - sendJointsPositions
  - sendJointsPositionsf

- sendJointsPositionsExTorque
- sendJointsPositionsMTorque
- sendJointsPositionsGetActualJpos
- sendJointsPositionsGetActualEEFpos
- realTime_startDirectServoJoints

- Tutorial script:
  - KSTclass_Tutorial_directServo.m

2.4 realTime_startImpedanceJoints

- Syntax:

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

- Description:

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

- Arguments:
  - 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 (mm).
  - cOMy: is the Y position of the tool's center of the mass in the flange referenced frame (mm).
  - cOMz: is the Z position of the tool's center of the mass in the flange referenced frame (mm).
  - cS: is the Cartesian linear stiffness in the range of [0 to 4000].
  - rS: is the Cartesian angular stiffness in the range of [0 to 300].
  - nS: is the null space stiffness.

- Return values:
  - None.

- Related methods:
  - sendJointsPositions
  - sendJointsPositionsf

  - sendJointsPositionsExTorque
  - sendJointsPositionsMTorque
  - sendJointsPositionsGetActualJpos
  - sendJointsPositionsGetActualEEFpos
  - realTime_stopImpedanceJoints

- Tutorial script:
  - KSTclass_Tutorial_realTimeImpedencePlotTorqueFeedBack.m

2.5 realTime_stopImpedanceJoints

- Syntax:

  `>> iiwa.realTime_stopImpedanceJoints()`

- Description:

  This method is used to stop the soft real-time control at the joints level in the impedance mode.

- Arguments:
  - None.

- Return values:
  - None.

- Related methods:
  - sendJointsPositions
  - sendJointsPositionsf
  - sendJointsPositionsExTorque
  - sendJointsPositionsMTorque
  - sendJointsPositionsGetActualJpos
  - sendJointsPositionsGetActualEEFpos
  - realTime_startImpedanceJoints

- Tutorial script:
  - KSTclass_Tutorial_realTimeImpedencePlotTorqueFeedBack.m

2.6 sendJointsPositions

- Syntax:
  `>> [ret]=iiwa.sendJointsPositions(jPos)`
- Description:
  This method 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 control mode. This method is blocking, it awaits for an acknowledgment from the server before returning back to execution.
- Arguments:
  - jPos: is a 1x7 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.
- Tutorial script:
  - KSTclass_Tutorial_directServo.m

2.7 sendJointsPositionsf

- Syntax:
  `>> iiwa.sendJointsPositionsf(jPos)`
- Description:
  This method 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 method is non-blocking, it is one way and does not awaits for an acknowledgment from the server. This method is computationally light-weight, and it is designed for implementation in computationally expensive algorithms. When utilized in low-computational-cost algorithms, this method 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 method please refer to script (KSTclass_Tutorial_directServoFast.m) in github repository.

- Arguments:
  - jPos: is a 1x7 cell array representing the target joints' positions, units in radians.
- Return values:
  - None.
- Tutorial script:
  - KSTclass_Tutorial_directServoFast.m

2.8 sendJointsPositionsExTorque

- Syntax:

```
>> [torques] = iiwa.sendJointsPositionsExTorque(jPos)
```

- Description:

  This method 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 method can be used in both modes, the DirectServo mode and the impedance mode. This method is blocking, it awaits for the torques message from the server before returning back to execution.
- Arguments:
  - jPos: is a 1x7 cell array representing the target joints' positions, units in radians.
- Return values:
  - torques: a 1x7 cell array with joints' torques due to the external forces acting on the robot structure.

2.9 sendJointsPositionsMTorque

- Syntax:

  `>> [torques] = iiwa.sendJointsPositionsMTorque(jPos)`

- Description:

  This method 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 method can be used in both modes, the DirectServo mode and the impedance mode. This method is blocking, it awaits for the torques message from the server before returning back to execution.

- Arguments:

  – `jPos`: is a 1x7 cell array representing the target joints' positions, units in radians.

- Return values:

  – `torques`: a 1x7 cell array with joints' torques as measured by the integrated torque sensors.

- Tutorial script:

  – `KSTclass_Tutorial_realTimeImpedencePlotTorqueFeedBack.m`

2.10 sendJointsPositionsGetActualJpos

- Syntax:

  `>> [actual_JPOS] = iiwa.sendJointsPositionsGetActualJpos(target_jPos)`

- Description:

  This method 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 method can be used in both modes, the DirectServo mode and the impedance mode. This method is blocking, it awaits for the position message from the server before returning back to execution.

- Arguments:

  – `target_jPos`: is a 1x7 cell array representing the target joints' positions, units in radians.

- Return values:
  - actual_JPOS: a 1x7 cell array with joints' actual positions as measured by the integrated sensors in the robot.

2.11 sendJointsPositionsGetActualEEFpos

- Syntax:

  `>> [EEF_POS] = iiwa.sendJointsPositionsGetActualEEFpos(jPos)`

- Description:

  This method 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 method can be used in both modes, the DirectServo mode and the impedance mode. This method is blocking, it awaits for the position message from the server before returning back to execution.

- Arguments:
  - jPos: is a 1x7 cell array representing the target joints' positions, units in radians.

- Return values:
  - EEF_POS: a 1x6 cell array with EEF actual positions as measured by the integrated sensors in the robot.

2.12 realTime_startDirectServoCartesian

- Syntax:

  `>> iiwa.realTime_startDirectServoCartesian()`

- Description:

  This method starts the DirectServo for controlling the robot in the Cartesian space at robot's EEF level. In such a case, after calling this method the user can stream to the controller the target positions of the EEF causing it to move on the fly, several methods are available for streaming the target positions they are listed in the bullet point **Related methods** below.

- Arguments:
  - None.
- Return values:
  - None.
- Related methods:
  - sendEEfPosition
  - sendEEfPositionf
  - sendEEfPositionExTorque
  - sendEEfPositionMTorque
  - sendEEfPositionGetActualEEFpos
  - sendEEfPositionGetActualJpos
  - realTime_stopDirectServoCartesian
- Tutorial script:
  - KSTclass_Tutorial_directServoCartesian.m

2.13 realTime_stopDirectServoCartesian

- Syntax:

  `>> iiwa.realTime_stopDirectServoCartesian()`

- Description:

  This method stops the DirectServo for controlling the robot in Cartesian space at the robot's EEF level.

- Arguments:
  - None.
- Return values:
  - None.
- Related methods:
  - sendEEfPosition
  - sendEEfPositionf

  - sendEEfPositionExTorque
  - sendEEfPositionMTorque
  - sendEEfPositionGetActualEEFpos
  - sendEEfPositionGetActualJpos
  - realTime_startDirectServoCartesian

- Tutorial script:
  - KSTclass_Tutorial_directServoCartesian.m

2.14 sendEEfPosition

- Syntax:

  `>> [ret] = iiwa.sendEEfPosition(EEEFpos)`

- Description:

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

- Arguments:
  - EEEFpos: is a 1x6 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), the target position shall be specified with respect to the base frame of the robot.

- 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 methods:
  - realTime_startDirectServoCartesian
  - realTime_stopDirectServoCartesian

- Tutorial script
  - KSTclass_Tutorial_directServoCartesian1.m

2.15 sendEEfPositionf

- **Syntax:**

  `>> iiwa.sendEEfPositionf(EEEFpos)`

- Description:

  This method is used to set the target position of the End-Effector, this method is not blocking, and does not wait for an acknowledgment from the server before returning the execution. As a result, this method is computationally light-weight, and it is designed for implementation in computationally expensive algorithms. When utilized in low-computational-cost algorithms, this method 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 timing procedure as to guarantee a transmission rate of around 275 packets/second.

- Arguments:

  - `EEEFpos:` is a 1x6 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 methods:

  - `realTime_startDirectServoCartesian`
  - `realTime_stopDirectServoCartesian`

- Tutorial script

  - `KSTclass_Tutorial_directServoCartesian.m`

2.16 sendEEfPositionExTorque

- **Syntax:**

  `>> [ExTorque] = iiwa.sendEEfPositionExTorque(EEEFpos)`

- Description:

  This method 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 method is blocking, it awaits for the torques message from the server before returning back to execution.

- Arguments:
  - EEEFpos: is a 1x6 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: 1x7 cell array of the joint torques due to external forces acting on robot structure.
- Related methods:
  - realTime_startDirectServoCartesian
  - realTime_stopDirectServoCartesian

2.17 sendEEfPositionMTorque

- Syntax:
  `>> [MT] = iiwa.sendEEfPositionMTorque(EEEFpos)`
- Description:
  This method 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 method is blocking, it awaits for the torques message from the server before returning back to execution.
- Arguments:
  - EEEFpos: is a 1x6 cell array where the first three elements represent (X, Y, Z) target positions of EEF (mm) and the last three elements represent the XYZ fixed rotation angles of EEF (radians), those coordinates are taken with respect to the base frame of the robot.
- Return values:
  - MT: 1x7 cell array of the joint torques as measured by the sensors.
- Related methods:
  - realTime_startDirectServoCartesian
  - realTime_stopDirectServoCartesian

2.18 sendEEfPositionGetActualJpos

- Syntax:
  `>> [JPOS] = iiwa.sendEEfPositionGetActualJpos(EEEFpos)`
- Description:
  This method 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 method is blocking, it awaits for the position message from the server before returning back to execution.
- Arguments:
  - `EEEFpos:` is a 1x6 cell array where the first three elements represent (X, Y, Z) target positions of EEF (mm) and the last three elements represent the XYZ fixed rotation angles of EEF (radians).
- Return values:
  - `JPOS`: 1x7 cell array of the joint positions as measured by the encoders.
- Related methods:
  - `realTime_startDirectServoCartesian`
  - `realTime_stopDirectServoCartesian`

2.19 sendEEfPositionGetActualEEFpos

- Syntax:
  `>> [actual_EEFpos] = iiwa.sendEEfPositionGetActualEEFpos(target_EEFpos)`
- Description:
  This method 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 as measured by the integrated encoders. This method is blocking, it awaits for the position message from the server before returning back to execution.
- Arguments:
  - `target_EEFpos:` is a 1x6 cell array where the first three elements represent (X, Y, Z) target positions of EEF (mm) and the last three elements represent the XYZ fixed rotation angles of EEF (radians), those coordinates are taken relative to the base frame of the robot.

- Return values:
  - actual_EEFpos: 1x6 cell array where the first three elements represent (X, Y, Z) target positions of EEF (mm) and the last three elements represent the XYZ fixed rotation angles of EEF (radians), those coordinates are taken relative to the base frame of the robot.
- Related methods:
  - realTime_startDirectServoCartesian
  - realTime_stopDirectServoCartesian

2.20 realTime_startVelControlJoints

- Syntax:

  `>> [ret] = iiwa.realTime_startVelControlJoints()`
- Description:

  This method is used to start the soft real-time control at joints velocities level, after calling this method the user can stream the target joints velocities to the controller through the methods presented in the **Related methods** bullet point. This soft real-time control at joints velocities level is built upon the DirectSerovo interface, where a midpoint Riemann integrator is implemented in the KST server, as such the velocities commands are used to calculate the target positions, which then are updated into the DirectServo.
- Arguments:
  - None.
- Return values:
  - ret: a boolean value, true if the method is executed successfully, false otherwise.
- Related methods:
  - sendJointsVelocities
  - sendJointsVelocitiesExTorques
  - sendJointsVelocitiesMTorques
  - sendJointsVelocitiesGetActualEEfPos
  - sendJointsVelocitiesGetActualJpos

- realTime_stopVelControlJoints

- Tutorial script:
  - KSTclass_Tutorial_softRealTimeJointsVelControl.m

2.21 realTime_stopVelControlJoints

- Syntax:

  `>> iiwa.realTime_stopVelControlJoints()`

- Description:

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

- Arguments:
  - None.

- Return values:
  - None.

- Related methods:
  - sendJointsVelocities
  - sendJointsVelocitiesExTorques
  - sendJointsVelocitiesMTorques
  - sendJointsVelocitiesGetActualEEfPos
  - sendJointsVelocitiesGetActualJpos
  - realTime_startVelControlJoints

- Tutorial script:
  - KSTclass_Tutorial_softRealTimeJointsVelControl.m

2.22 sendJointsVelocities

- **Syntax:**

  `>> [ret] = iiwa.sendJointsVelocities(jvel)`

- Description:

  This method is used to set reference joints' velocities for soft real-time control at joints level. Unlike the position control modes, in the velocity control mode, the user is not restricted to stream the velocities commands at a specified frequency.

- Arguments:

  - `jvel`: is a 1x7 cell array representing the target angular velocity for each joint, in rad/sec.

- Return values:

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

- Related methods:

  - `realTime_startVelControlJoints`
  - `realTime_stopVelControlJoints`

- Tutorial script:

  - `KSTclass_Tutorial_softRealTimeJointsVelControl.m`

2.23 sendJointsVelocitiesExTorques

- **Syntax:**

  `>> [ExT] = iiwa.sendJointsVelocitiesExTorques(jvel)`

- Description:

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

- Arguments:

  - `jvel`: is a 1x7 cell array representing the target angular velocity for each joint, in rad/sec.

- Return values:
  - `ExT`: 1x7 cell array of external torques.

- Related methods:
  - `realTime_startVelControlJoints`
  - `realTime_stopVelControlJoints`

2.24 `sendJointsVelocitiesMTorques`

- Syntax:

  `>> [MT] = iiwa.sendJointsVelocitiesMTorques(jvel)`

- Description:

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

- Arguments:
  - `jvel`: is a 1x7 cell array representing the target angular velocity for each joint, in rad/sec.

- Return values:
  - `MT`: 1x7 cell array of measured torques.

- Related methods:
  - `realTime_startVelControlJoints`
  - `realTime_stopVelControlJoints`

2.25 sendJointsVelocitiesGetActualJpos

- Syntax:

  `>> [JPoS] = iiwa.sendJointsVelocitiesGetActualJpos(jvel)`

- Description:

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

- Arguments:
  - `jvel`: is a 1x7 cell array representing the target angular velocity for each joint, in rad/sec.

- Return values:
  - `JPoS`: 1x7 cell array of the actual positions of the joints (radian).

- Related methods:
  - `realTime_startVelControlJoints`
  - `realTime_stopVelControlJoints`

2.26 sendJointsVelocitiesGetActualEEfPos

- Syntax:

  `>> [EEfPos] = iiwa.sendJointsVelocitiesGetActualEEfPos (jvel)`

- Description:

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

- Arguments:
  - `jvel`: is a 1x7 cell array representing the target angular velocity for each joint, in rad/sec.

- Return values:

  - EEfPos: 1x6 cell array of the actual EEF position, first three elements are the X, Y and Z coordinates of EEF (mm), remaining three elements are the XYZ fixed rotation angles of EEF (rads), the coordinates are with respect to the base frame of the robot.

- Related methods:
  - realTime_startVelControlJoints
  - realTime_stopVelControlJoints

3 Point-to-point motion

This group of methods is used to move the robot towards a destination point, according to the method used the destination point can be specified in Cartesian space or in joints space. KST implements several point-to-point motion methods that allow the user to move the robot from one configuration to another in joint space, or to move the EEF of the robot on a linear, circular or elliptical trajectory in Cartesian space. By using the point-to-point motion methods the path towards the destination point is planned by the software, so the user is not required to perform the path planning him-self. In this section the point-to-point motion methods are described.

3.1 `movePTPJointSpace`

- `Syntax:`

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

- Description:

  This method is used to move the robot from the current configuration to a new configuration in joint space. Demonstrated in Figure 3.1, after calling the method in subject the robot moves from its current configuration, opaque, to the new configuration, transparent.

- Arguments:

  - `jPos`: is a 1x7 cell array representing the target angular positions, in rad/sec.
  - `relVel`: is a double, from zero to one, specifying the override velocity, a value of one means that the robot will move its joints with the maximum possible velocity, 0.5 value means that the robot will move its joints with 50% of the maximum possible 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.

- Tutorial script:

  - `KSTclass_Tutorial_move_ptp.m`

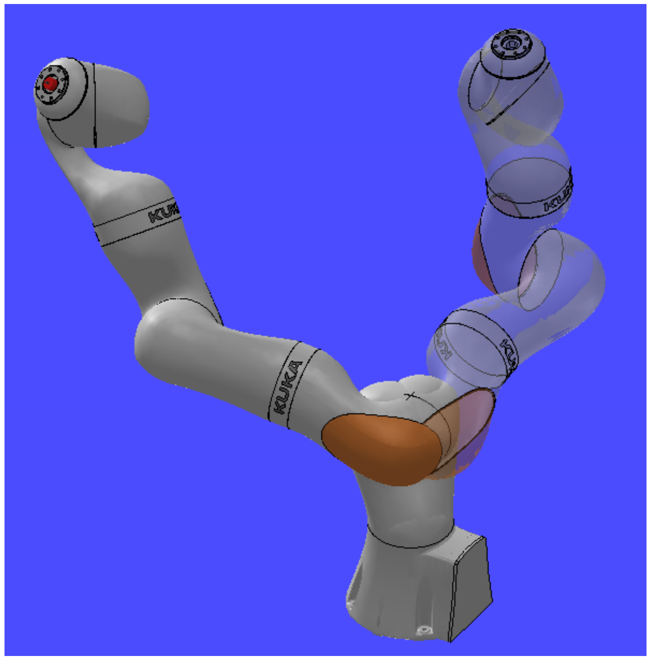


Fig. 3.1: Moving robot to some configuration

3.2 movePTPLineEEF

- Syntax:

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

- Description:

  This method is used to move the end-effector in a straight line from the current pose (position/orientation) to a destination pose. When called, the method 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 the line, this depends of the values of the XYZ fixed joint angles passed by the argument.

  In Figure 3.2, after calling the method in subject the robot moves in a linear motion from its current configuration, opaque, to the new configuration, transparent, the target point of EEF position is defined by a vector with respect to base frame of the robot, the dashed black arrow in Figure 3.2. The target orientation of the EEF is defined by the XYZ fixed rotation angles with respect to the base frame of the robot.

- Arguments:

  - `Pos`: is a 1x6 cell array representing the destination position of EEF,

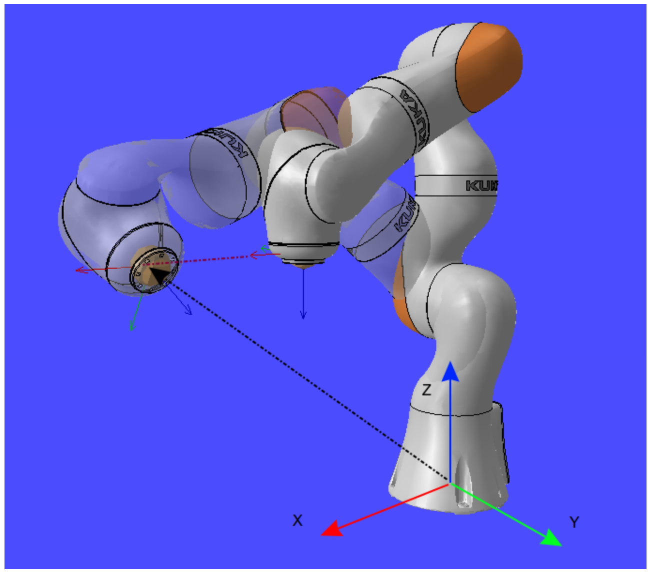


Fig. 3.2: Moving EEF in a straight line towards destination pose

the first three elements are the XYZ coordinates in (mm), the remaining three elements are the XYZ fixed rotation angles, in radians, those coordinates are with respect to robot's base frame.

  - `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.

- Tutorial script:
  - `KSTclass_Tutorial_move_lin.m`

3.3 `movePTPHomeJointSpace`

- **Syntax:**

  `>> [ret]= iiwa.movePTPHomeJointSpace(relVel)`

- Description:

  This method is used to move the robot to home configuration. Demonstrated in Figure 3.3, after calling the method in subject the robot moves

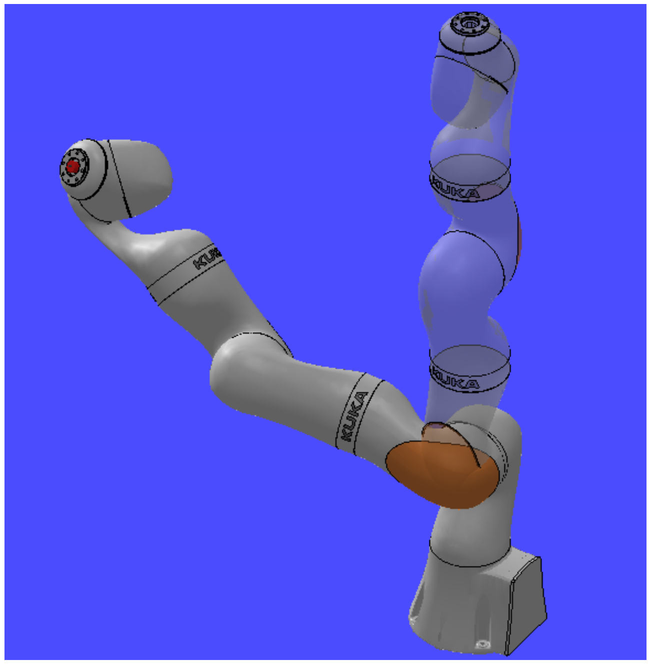


Fig. 3.3: Moving robot to home configuration

from its current configuration, opaque, to the home configuration, transparent.

- Arguments:
  - `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.
- Tutorial script:
  - `KSTclass_Tutorial_move_ptp.m`

3.4 `movePTPTransportPositionJointSpace`

- Syntax:

```
>> [ret] = iiwa.movePTPTransportPositionJointSpace(relVel)
```

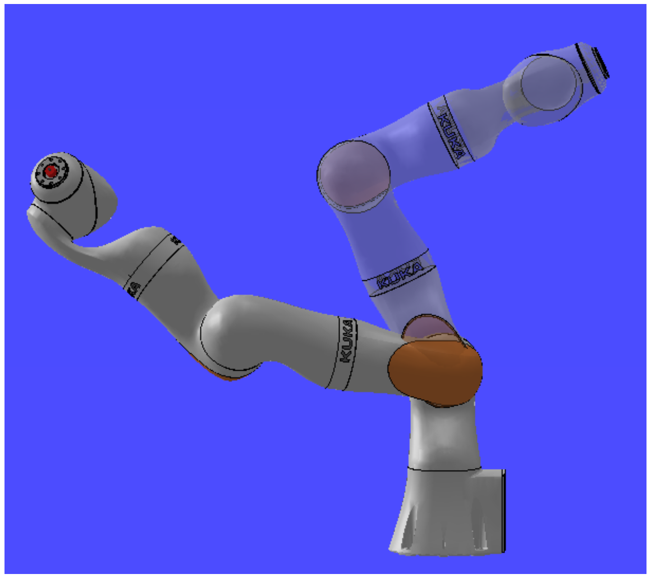


Fig. 3.4: Moving robot to transport configuration

- Description:

  This method moves the robot to the transport configuration. Demonstrated in Figure 3.4, after calling the method in subject the robot moves from its current configuration, opaque, to the transport configuration, transparent.

- Arguments:

  - `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.

- Tutorial script:

  - `KSTclass_Tutorial_move_ptp.m`

3.5 movePTPLineEefRelBase

- Syntax:

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

- Description:

  This method 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, during the motion the orientation of the EEF remains fixed.

  Demonstrated in Figure 3.5, after calling the method in subject the EEF moves from its current position, opaque, to a new position, transparent, defined by the displacement vector, dashed black arrow, this vector is defined by displacements $(\Delta X, \Delta Y, \Delta Z)$ along the XYZ axes of the base.

- Arguments:

  – `Pos`: is a 1x3 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.

- Tutorial script:

  – `KSTclass_Tutorial_move_lin_relative.m`

3.6 movePTPLineEefRelEef

- Syntax:

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

- Description:

  This method 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. during the motion the orientation of the EEF remains fixed.

  Demonstrated in Figure 3.6, after calling the method in subject the EEF moves from its current position, opaque, to a new position, transparent, defined by the displacement vector, dashed black arrow, this vector is defined by displacements $(\Delta X, \Delta Y, \Delta Z)$ along the XYZ axes of the EEF frame.

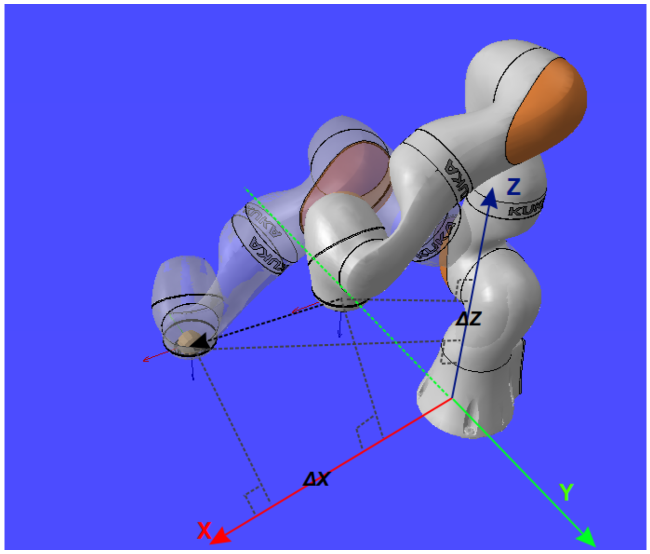


Fig. 3.5: Moving EEF in a straight line using displacements along base axes

- Arguments:
  - `Pos`: is a 1x3 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.
- Tutorial script:
  - `KSTclass_Tutorial_move_lin_relative.m`

3.7 `movePTPCirc1OrintationInter`

- `Syntax:`

  `>> [ret] = iiwa.movePTPCirc1OrintationInter(f1, f2, vel)`
- Description:

  This method moves the end-effector in an arc specified by two frames. During the arc motion, the EEF passes by the first frame and stops when

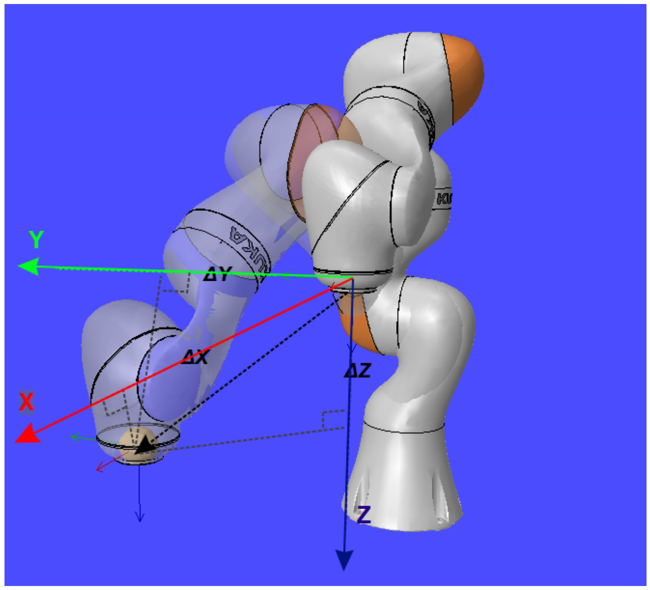


Fig. 3.6: Moving EEF in a straight line using displacements along EEF axes

it reaches the second frame. In this method the EEF can change orientation while moving on the arc, this is specified through the XYZ fixed rotation angles of the frames passed by the arguments. Giving the difficulty associated with specifying an arc using two frames, KST supports other methods for performing the arc motion, those methods are listed in the bullet point **Related methods**.

- Arguments:
  - `f1`: is 1x6 cell array, specifying the intermediate frame of the arc.
  - `f2`: is 1x6 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 XYZ 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.
- Related methods:
  - `movePTPArc_AC`

- movePTPArcXY_AC
- movePTPArcXZ_AC
- movePTPArcYZ_AC

- Tutorial script:
  - KSTclass_Tutorial_circles.m

3.8 movePTPArc_AC

- **Syntax:**

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

- Description:

  This method moves the end-effector in arc in 3D space as shown in Figure 3.7, the arc is specified by: center, angle, normal to the plane of the arc, and a starting point at the current position of EEF.

- Arguments:
  - theta: is the arc's angle in radians, this angle is shown in Figure 3.7 by the symbol θ.
  - c: is a 1x3 vector representing the x, y and z coordinates of the arc's center, this pint is shown in Figure 3.7 by the symbol c.
  - k: is a 1x3 vector representing the normal vector on the plane of the arc, this angle is shown in Figure 3.7 by the symbol $\boldsymbol{k}$.
  - 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.

- Tutorial script:
  - KSTclass_Tutorial_circles.m

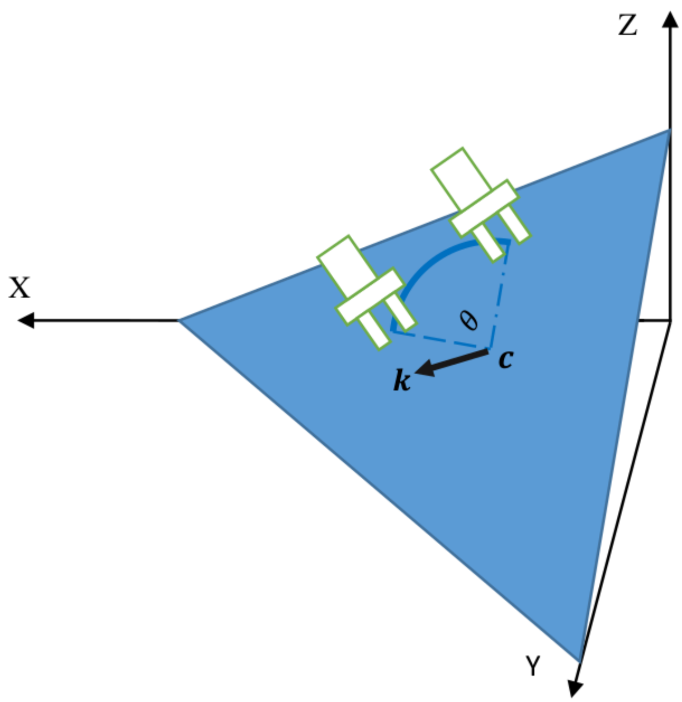


Fig. 3.7: Moving EEF on an arc using movePTPArc_AC

3.9 movePTPArcYZ_AC

- **Syntax:**

  `>> [ret] = iiwa.movePTPArcYZ_AC(theta, c, vel)`

- Description:

  This method moves the end-effector in an arc parallel to the YZ plane of the base of the robot, Figure 3.8, the arc is specified by its: center, angle, and a starting point at the current position of EEF.

- Arguments:

  - `theta`: is the arc's angle in radians.
  - `c`: is a 2x1 vector representing the Y and Z coordinates of the arc's center, Figure 3.8.
  - `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 methods:

  - `movePTPArcXY_AC`
  - `movePTPArcXZ_AC`

- Tutorial script:

  - `KSTclass_Tutorial_circles.m`

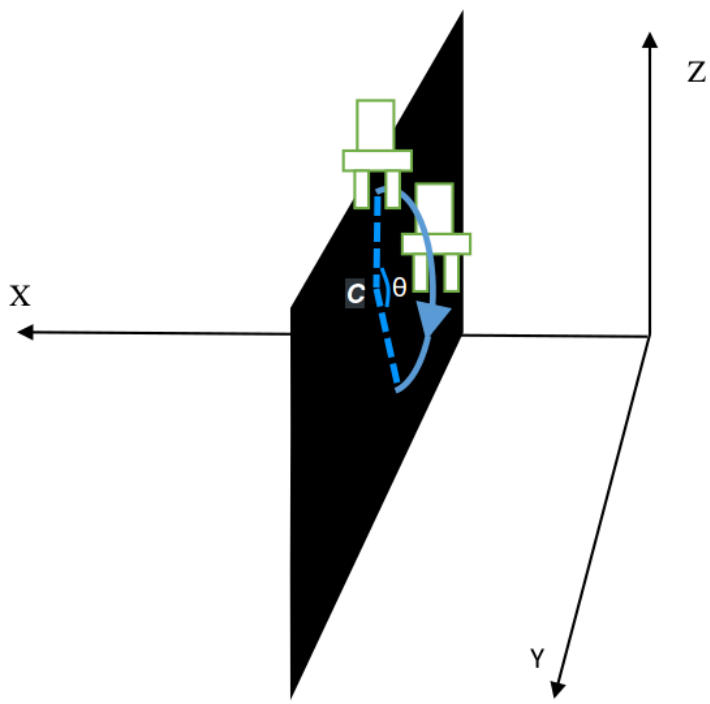


Fig. 3.8: Moving EEF on arc in a plane parallel to YZ

4 Setters

Those methods 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 methods can still be called but they will have no effect on the robot.

4.1 `setBlueOff`

- `Syntax:`

  `>> [ret]= iiwa.setBlueOff()`

- Description:

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

- Arguments:

  – None.

- Return values:

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

4.2 `setBlueOn`

- `Syntax:`

  `>> [ret]= iiwa.setBlueOn()`

- Description:

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

- Arguments:

  – None.

- Return values:

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

4.3 setPin1Off

- Syntax:

  `>> [ret]= iiwa.setPin1Off()`

- Description:

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

- Arguments:

  – None.

- 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:

  `setPin11Off,setPin12Off,setPin2Off` used to set off the pins 11, 12 and 2.

4.4 setPin1On

- Syntax:

  `>> [ret]= iiwa.setPin1On()`

- Description:

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

- Arguments:

  – None.

- 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:

  `setPin11On,setPin12On,setPin2On` used to set on the pins 11, 12 and 2.

5 Getters

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

5.1 getEEF_Force

- Syntax:

  `>> [f] = iiwa.getEEF_Force()`

- Description:

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

- Arguments:
  - None.

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

- Tutorial script:
  - `KSTclass_Tutorial_getters.m`

5.2 getEEF_Moment

- Syntax:

  `>> [m] = iiwa.getEEF_Moment()`

- Description:

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

- Arguments:
  - None.

- Return values:

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

- Tutorial script:
  - `KSTclass_Tutorial_getters.m`

5.3 getEEFCartesianOrientation

- **Syntax:**

  `>> [ori] = iiwa.getEEFCartesianOrientation()`

- Description:

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

- Arguments:
  - None.

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

- Tutorial script:
  - `KSTclass_Tutorial_getters.m`

5.4 getEEFCartesianPosition

- **Syntax:**

  `>> [Pos] = iiwa.getEEFCartesianPosition()`

- Description:

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

- Arguments:
  - None.

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

5.5 getEEFPos

- **Syntax:**

  `>> [Pos] = iiwa.getEEFPos()`

- Description:

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

- Arguments:
  - None.
- 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).
- Tutorial script:
  - `KSTclass_Tutorial_getters.m`

5.6 getJointsExternalTorques

- **Syntax:**

  `>> [torques] = iiwa.getJointsExternalTorques()`

- Description:

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

- Arguments:

   - None.

- Return values:

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

- Tutorial script:

   - `KSTclass_Tutorial_getters.m`

5.7 getJointsMeasuredTorques

- Syntax:

```
>> [torques] = iiwa.getJointsMeasuredTorques()
```

- Description:

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

- Arguments:

   - None.

- Return values:

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

- Tutorial script:

   - `KSTclass_Tutorial_getters.m`

5.8 getJointsPos

- Syntax:

```
>> [jPos] = iiwa.getJointsPos()
```

- Description:

  This method returns the robot joints' angles.

- Arguments:

   - None.

- Return values:
  - jPos: is a 1x7 cell array of actual joints positions as measured by the encoders, in radians.
- Tutorial script:
  - KSTclass_Tutorial_getters.m

5.9 getMeasuredTorqueAtJoint

- **Syntax:**

  `>> [torque] = iiwa.getMeasuredTorqueAtJoint(k)`
- Description:

  This method returns the measured torque in a specific joint.
- Arguments:
  - None.
  - k: is the joint index, from 1 to 7.
- Return values:
  - torque: is the measured torque at joint k, torque unit is Newton.Meter.
- Tutorial script:
  - KSTclass_Tutorial_getters.m

5.10 getExternalTorqueAtJoint

- **Syntax:**

  `>> [torque] = iiwa.getExternalTorqueAtJoint(k)`
- Description:

  This method returns the external torque in a specific joint.
- Arguments:
  - k: is the joint index, from 1 to 7.

- Return values:
  - torque: is the external torque at joint k, torque unit is Newton.Meter.
- Tutorial script:
  - KSTclass_Tutorial_getters.m

5.11 getEEFOrientationR

- Syntax:

  `>> [rMat] = iiwa.getEEFOrientationR()`
- Description:

  This method returns the rotation matrix representing the orientation of the EEF relative to the base frame of the robot.
- Arguments:
  - None.
- Return values:
  - rMat: 3x3 rotation matrix representing the orientation of the EEF relative to the base frame of the robot.
- Tutorial script:
  - KSTclass_Tutorial_getters.m

5.12 getEEFOrientationQuat

- Syntax:

  `>> [quaternion] = iiwa.getEEFOrientationQuat()`
- Description:

  This method returns the quaternion representing the orientation of the EEF relative to the base frame of the robot.
- Arguments:
  - None.

- Return values:
  - quaternion: 1x4 quaternion vector representing the orientation of the EEF relative to the base frame of the robot.
- Tutorial script:
  - KSTclass_Tutorial_getters.m

6 General purpose

Those methods are used for calculating the forward/inverse kinematics, the forward/inverse dynamics, and the various physical quantities of the LBR7R800 and the LBR14R820 manipulator. It is worthy to mention that in the calculations of the general purpose methods, the base of the robot is considered to be mounted horizontally where the Z axes is facing upwards. In those functions the dynamic parameters used for LBR14R820 are as reported in article [1], and the dynamic parameters used for LBR7R800 are as reported in the article [2]. While the Denavit-Hartenberg data used for of LBR7R800 and LBR14R820 are as reported in the drawings and the dimensions of manual [3].

As described in part IV **KST class**, when the iiwa object is instantiated by calling the constructor of the KST class, The user has to pass arrangements specifying the type of the robot and the mounting flange. In such a case, the constructor automatically loads the appropriate Kinematics/Inertial data of the robot.

Kinematics Methods

The KST toolbox offers several methods that allows the user to calculate the various kinematic quantities required to implement the control algorithms, including:

- Forward kinematics, this method is used to calculate the direction/orientation of the EEF at certain joints angles of the robot, KST utilizes the mathematics of the transformation matrices for calculating the forward kinematics, the Denavit-Hartenberg (DH) parameters are loaded automatically when the constructor of KST class is called. According to the arguments passed to the constructor specifying the type of the robot and the type of the flange, the right DH parameters are calculated and loaded implicitly by the toolbox into a read only property, as such the user does not have to load the DH parameters by himself. If a tool is attached to the flange of the robot, then the user shall specify the transformation matrix of the Tool Center Point (TCP) with respect to the flange of the robot and pass it as an argument to the constructor .

- The Jacobian of the manipulator, which is a 6xn matrix, where n is the number of joints of the manipulator, this matrix gives a relationship between the velocity of the EEF linear/angular and the angular velocities of the joints of the robot.

- Inverse kinematics, this function is used to calculate the joint angles of the robot for a given position/orientation (pose) of the EEF, for redundant manipulators there are infinite solutions that gives the same pose of the EEF, KST utilizes the damped least squares for calculating a solution of the joint angles that correspond to a specific pose of the EEF.

6.1 gen_DirectKinematics

- **Syntax:**
  `>> [eef_transform, J] = iiwa.gen_DirectKinematics(q)`
- Description:
  This method calculates the forward kinematics/Jacobian of the manipulator, the Jacobian returned by this function is the geometric Jacobian.
- Arguments:
  - `q`: the angles of the robot joints.
- 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.
- Tutorial script:
  - `KSTclass_Tutorial_generalPorpuse.m`

6.2 gen_partialJacobean

- **Syntax:**
  `>> [Jp] = iiwa.gen_partialJacobean(q, linkNum ,Pos)`
- Description:
  This method 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.
- Tutorial script:
  - `KSTclass_Tutorial_generalPorpuse.m`

6.3 gen_InverseKinematics

- Syntax:

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

- Description:

  This method calculates the inverse kinematics of the robot at a given position/orientation of its EEF. The joints' angles are calculated numerically, where the damped least squares (DLS) method is utilized for performing the calculations, given this fact the user shall keep in mind that the achieved solution is approximate, and that the precision of the answer depends on the values of parameters passed to the function as described below.

- Arguments:

  - `qin`: is the initial guess used by the solver, in general case this vector can be chosen to be zero. On the other hand for soft real-time control applications, where the requirement is to control the joints angles as to perform certain motion of the EEF, the user shall consider to use the current joints' angles of the robot as an initial guess (radians), an example code is shown in the file KST-class_Tutorial_moveRealtimeEllipse.m.
  - `Tt`: is the target transformation matrix position/orientation of the robot.
  - `n`: is the number of iterations used by the solver, given that the DLS method is a numerical algorithm, the solution is achieved by performing several iterations, in general higher number of iterations gives a more precise answer, this is at the expense of an increasing computational cost.
  - `lambda`: is the damping constant, this factor gives the algorithm more numerical stability near singular configurations, yet a high value damping value results in less precise answer.

- Return values:

  - `qs`: is the solution joints' angles of the robot.

- Tutorial script:

  - `KSTclass_Tutorial_generalPorpuse.m`

6.4 gen_NullSpaceMatrix

- **Syntax:**
  `>> [N] = iiwa.gen_NullSpaceMatrix(q)`
- Description:
  This method 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.
- Tutorial script:
  - `KSTclass_Tutorial_generalPorpuse.m`

Dynamics Methods

KST integrates several methods for calculating the different dynamics quantities of KUKA iiwa robot, including:

- Mass matrix of the robot: it is a positive definite matrix, of dimension of 7x7, this matrix gives a relationship between the angular accelerations of the joints and the resulting torques due to acceleration only.
- Coriolis matrix: this matrix gives is 7x7 matrix, it gives a relationship between the angular velocities of the joints and the torques due to Coriolis and centrifugal forces.
- Centrifugal matrix: this matrix gives is 7x7 matrix, it gives a relationship between the square of the angular velocities of the joints and the torques due to centrifugal forces only.
- Inverse dynamics: this method calculates the joints' torques required to generate certain motion of the robot.
- Forward dynamics: this method calculates the joints' accelerations in some state due to joint torques.
- Gravity vector: this method calculates the torques acting on the joints of the robot due to gravitational pull, in this function the robot is considered to be mounted in the upright position, with its base is in the horizontal position.

6.5 gen_MassMatrix

- Syntax:
  `>> [M] = iiwa.gen_MassMatrix(q)`
- Description:
  This method is used to calculate the mass matrix of the robot, in the configuration specified by the angular positions `q`.
- Arguments:
  - `q`: is a 1x7 vector with angles of the manipulator,
- Return values:
  - `M`, is a 7x7 matrix – the mass matrix of the manipulator.
- Tutorial script:
  - `KSTclass_Tutorial_generalPorpuse.m`

6.6 gen_CoriolisMatrix

- Syntax:
  `>> [B] = iiwa.gen_CoriolisMatrix(q, dq)`
- Description:
  This method calculates Coriolis matrix of the manipulator at configuration `q` with angular velocities `dq`.
- Arguments:
  - `q`: is a 1x7 vector with angles of the manipulator,
  - `dq`: is a 1x7 vector with the joints' angular velocity.
- Return values:
  - `B`, is a 7x7 matrix – the Coriolis matrix of the robot.
- Tutorial script:
  - `KSTclass_Tutorial_generalPorpuse.m`

6.7 gen_CentrifugalMatrix

- **Syntax:**
  `>> [C] = iiwa.gen_CentrifugalMatrix(q, dq)`
- Description:
  This method calculates Coriolis matrix of the manipulator at configuration `q` with angular velocities `dq`.
- Arguments:
  - `q`: is a 1x7 vector with angles of the manipulator,
  - `dq`: is a 1x7 cell array with the joints' angular velocity.
- Return values:
  - `C`: is a 7x7 matrix – the centrifugal matrix of the manipulator.
- Tutorial script:
  - `KSTclass_Tutorial_generalPorpuse.m`

6.8 gen_GravityVector

- **Syntax:**
  `>> [G] = iiwa.gen_GravityVector(q)`
- Description:
  This method calculates joints torques due to gravity at configuration `q`.
- Arguments:
  - `q`: is a 1x7 vector with angles of the manipulator,S
- Return values:
  - `G`: is a 7x1 vector – the gravity vector of the manipulator.
- Tutorial script:
  - `KSTclass_Tutorial_generalPorpuse.m`

6.9 gen_DirectDynamics

- **Syntax:**
  `>> [d2q] = iiwa.gen_DirectDynamics(q, dq, taw)`
- Description:
  This method calculates the forward dynamics of the manipulator.
- Arguments:
  - `q`: is a 1x7 vector with angles of the manipulator,
  - `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.
- Tutorial script:
  - `KSTclass_Tutorial_generalPorpuse.m`

6.10 gen_InverseDynamics

- **Syntax:**
  `>> [taw] = iiwa.gen_InverseDynamics(q, dq, d2q)`
- Description:
  This method calculates the inverse dynamics of the manipulator.
- Arguments:
  - `q`: is a 1x7 vector with angles of the manipulator.
  - `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
- Tutorial script:
  - `KSTclass_Tutorial_generalPorpuse.m`

7 Physical Interaction

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

7.1 startHandGuiding

- Syntax:

  `>> iiwa.startHandGuiding()`

- Description:

  This method is used to start KUKA's off-the-shelf the hand guiding for a manipulator with a pneumatic-touch 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:

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

- Return values:

  - None.

- Tutorial script:

  - `KSTclass_Tutorial_HandTeaching.m`

7.2 startPreciseHandGuiding

- About:

Why the precision hand-guiding? during our work in the Collaborative Robotics Lab at the university of Coimbra, we had a requirement to use KUKA manipulator as a third hand, where KUKA will assist the worker in performing assembly tasks of satellite components using the hand-guiding functionality, in such a case KUKA will carry most of the weight of the equipment, offer precision, and will hold the component in place, some times in hard-to-access tight locations, while the worker fixes the component in place.

The first idea that came to mind is to utilize KUKA hand-guiding, we started to test this mode using a dummy satellite, unfortunately KUKA hand-guiding did not match the precision criteria required. The experiments carried out are demonstrated in the following video:

`https://youtu.be/p459YeagBXM`

As a conclusion, several draw backs have been experienced while testing KUKA hand-guiding:

1. From the video you can see the author is struggling to adjust the position/orientation of the dummy, attached to the robot, into its place in the structure. Even after several trials the author was not able to insert the dummy into its place. The test was repeated with other people performing the same operation, the result was always the same: the dummy instrument was always overshooting right and left, up and down, while trying to adjust its position in place inside the structure.

2. The test was performed on a dummy that weighs around 1 Kg, though the requirement is to use the hand-guiding for installing sensitive instruments that have a weight range of several kilograms, overshooting could cause the instrument to hit the structure of the satellite, which might damage sensitive and expensive equipment, unacceptable situation.

3. Using KUKA hand-guiding the user can not adjust the orientation of the dummy while keeping its position fixed.

4. Using KUKA hand-guiding the user can not adjust the position of the dummy while keeping its orientation fixed.

5. KUK hand-guiding works at the joints level, as such it can not offer precision in Cartesian space. At best case scenario, it can be as precise as the operator can be.

When we faced this problem, a top-of-the-head solution that first came into mind was to utilize the teach-pendant, while it offers precision it did not match the requirements due to the following:

1. Unlike the hand-guiding, when using the teach-pendant the user does not have a feel of the force applied between the instrument and its surrounding in case of contact. Accidents could happen and the user might over press the sensitive instrument against the surrounding without noticing.

2. When using the teach-pendant to position the EEF in Cartesian space, the user has to keep a track of the orientation of the robot base, this could become confusing even for the experienced worker, not to mention that in our study case, KUKA iiwa manipulator is mounted on a mobile platform, as such the robot is always moving around the structure of the satellite.

3. The teach pendant convention in describing the orientation is the XYZ, fixed rotation angles. This was for describing orientation is not intuitive for humans. Even the experienced roboticest agrees that in the general case it is hard to imagine an orientation of an object based on three numbers.

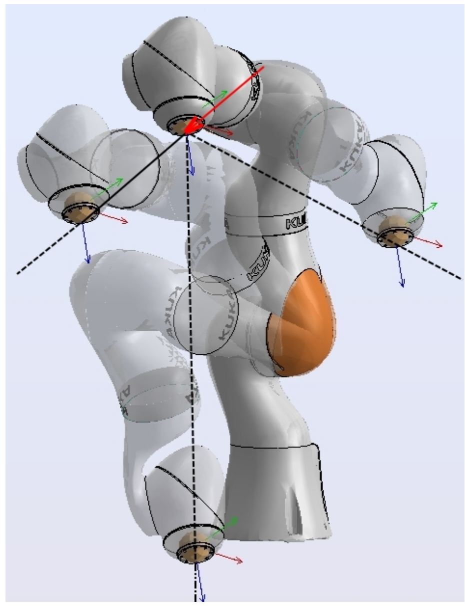


Fig. 7.1: Precise hand-guiding first motion group

4. The override control to change the velocity of the robot while performing the positioning operation is not that convenient.

As such the author proposed the precision hand-guiding that addresses the precision requirements in precise positioning operations. The precision hand-guiding convention for defining EEF orientation is build upon a well known action that we always do when we try to insert a key inside a keyhole. From users feedback, the proposed orientation convention is more intuitive for use than the teach pendant convention which utilizes fixed rotation angles convention.

To show case the advantages of the precision hand-guiding, a comparison between the precision hand-guiding and KUKA hand-guiding is presented in the following videos. A qualitative assessment proves that the precision hand-guiding offers superior precision, convenience and smoother operation when it comes to performing precise positioning operations.

In the following video KUKA hand-guiding is utilized, you can notice precision criteria was not satisfied, and the author is struggling to maintain the orientation of the tool in the required direction:

`https://youtu.be/xKlyHxptIEE`

And in the following video precise hand-guiding offered a reasonable solution:

`https://youtu.be/SM_2TSsq8kQ`

To address the precision requirements, each motion allowed by the precision hand-guiding is constrained in specific way, so the allowed motions can be divided into three main motion groups:

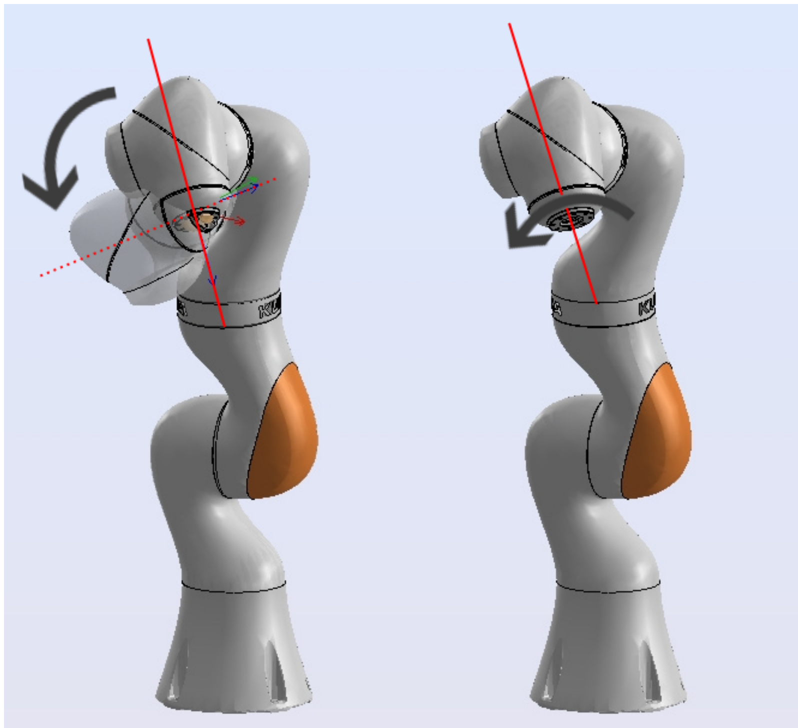


Fig. 7.2: Precise hand-guiding, (left) 2^{nd} motion group, (right) 3^{rd} motion group

1. First motion group, Fig. 7.1: in this case the operator can move the EEF on a line along the axes of the robot base frame, once at a time. In such a case, the orientation of the EEF is kept unchanged while performing the motion.
2. Second motion group, left of Fig. 7.2, in this case the operator can orient the axis of EEF by applying a moment. In such a case the position of the EEF is kept fixed.
3. Third motion group, right of Fig. 7.2, in this case the user can rotate the EEF of the robot round its axis. In such a case the position of the EEF is fixed, and the orientation of the EEF's axis is also fixed.

The first motion group is demonstrated in Fig. 7.1. To explain this motion group, consider the case where the operator is applying a force on the EEF of the robot, the force applied is approximately aligned with the positive X-axis of the robot-base and exceeds a predefined threshold value, as such the robot starts moving from its initial position, opaque, on a line parallel to X axes. When the applied force drops below the predefined threshold or seizes to exist the robot stops in its final position, transparent robot. During the motion the magnitude of EEF velocity is proportional to the magnitude of the applied force, and the orientation of the EEF stays the same.

This allows the user to move the robot precisely on a line in the directions of base axes one at a time, while keeping the orientation of the EEF of the robot fixed. Similar to the way the teach-pendant works but more intuitively and with an ability to control the velocity of the motion through the magnitude of the applied force.

The second motion group is demonstrated in left of Fig. 7.2, in this mode the user can rotate the axis of the EEF by applying a moment, when the EEF is rotated its position stays fixed. The third motion group is demonstrated in right of Fig. 7.2, in this motion group the user can rotate the EEF around its axis by applying a torque around EEF's axis.

- `Syntax:`

  `>> iiwa.startPreciseHandGuiding(wot, com)`

- Description:

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

- Arguments:

  - `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.

- Tutorial script:

  - `KSTclass_Tutorial_preciseHandGuidingExample.m`

7.3 `performEventFunctionAtDoubleHit`

- `Syntax:`

  `>> iiwa.performEventFunctionAtDoubleHit()`

- Description:

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

- Arguments:

  - None.

- Return values:
  - None.

7.4 `EventFunctionAtDoubleHit`

- `Syntax:`

  `>> iiwa.EventFunctionAtDoubleHit()`

- Description:

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

- Arguments:
  - None.

- Return values:
  - None.

7.5 `moveWaitForDTWhenInterrupted`

- `Syntax:`

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

- Description:

  This method 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:
  - `Pos`: is 1x6 cell array of EEF position, the first three elements are X,Y and Z coordinates of EEF, second three elements are the XYZ fixed rotation angles of EEF, the coordinates/rotation angles are with respect to base frame of the robot.

  - 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.

- Tutorial script:
  - KSTclass_Tutorial_ptpPickPlacePhysicalInteraction.m

Part VII. Studies/examples where KST is used

The toolbox has been used extensively in the Collaborative robotics lab at the university of Coimbra for education and research purposes. As such the toolbox has been utilized to control KUKA iiwa robot in several research applications and many use cases. In this part of the documentation, some of which are listed along with multimedia links (you-tube videos).

Gesture recognition from skeletal data

KUKA sunrise toolbox was utilized in [4], by my colleagues André Brás et al, during their work on gesture recognition from skeletal data and its application for achieving an intuitive human machine interaction. A Kinect camera was used to retrieve the skeletal data of the human, a neural network was utilized for the recognition of dynamic gestures. To show case the applicability of the system in industrial robotics, the gesture recognition system was utilized to control KUKA iiwa robot from an external computer through the KST toolbox, Fig 7.3. When the human performs the gesture, Matlab collects the skeletal data from the Kinect camera, the gesture recognition algorithm written in Matlab performs the gesture recognition, after wards using the KST toolbox the robot is controlled to perform the associated operation. In this study case, the use of KST was advantageous due to several factors:

- The system of dynamic gestures recognition was developed in Matlab.
- Matlab offers support for Kinect, and offers various functions for acquiring/treating the data from the Kinect camera.
- The simplicity of integrating several toolboxes/algorithms in the same program using Matlab made it an optimal choice.

Associated video is in the link:
`https://youtu.be/2S-z3WCAAAU`

Objects recognition

In this study unsupervised feature extraction from RGB-D data was utilized to segment objects previously placed above a table and to extract the visual features from each object. The Yale-Carnegie Mellon University (CMU)-Berkeley (YCB) object model set was utilized, a first generation Kinect sensor is used to collect RGB-D data using Matlab, and a feed forward artificial neural network was utilized for objects classification, Kuka iiwa robot was controlled through KST for grasping the object selected by the human from a tablet, as demonstrated in Fig 7.4.
Associated video is in the link:

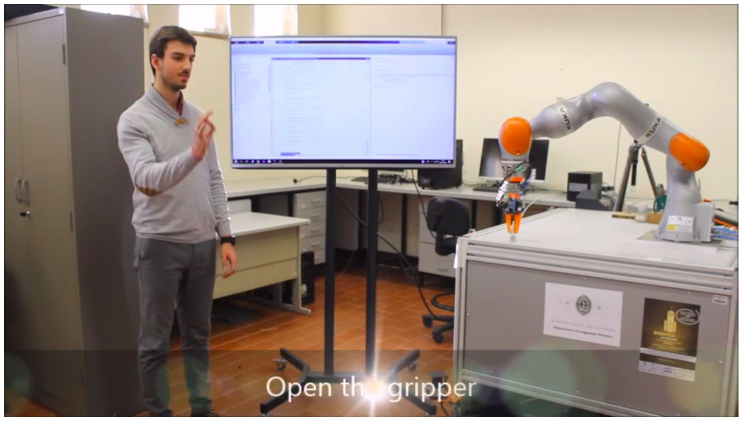


Fig. 7.3: Dynamic gesture recognition from skeletal data

Fig. 7.4: Objects recognition from RGB-D data

https://youtu.be/bwBRwdwVE2w
Associated code is in the link:
https://github.com/AndreBrasUC/Object_Recognition_From_RGBD_Data

IIWA plotter

KUKA Sunrise Toolbox was utilized to control iiwa for plotting on a sheet of paper using a pen attached to the end-effector. The program can plot any vector based file with the extension (PLT). In this example various functionalities of KST was utilized, the inverse kinematics function was used to generate the joints trajectories from the curves of the plot described in Cartesian space, and the soft real-time control functionality was used to move the robot in joint space. This example can be also used to convert IIWA into a laser cutter, by attaching

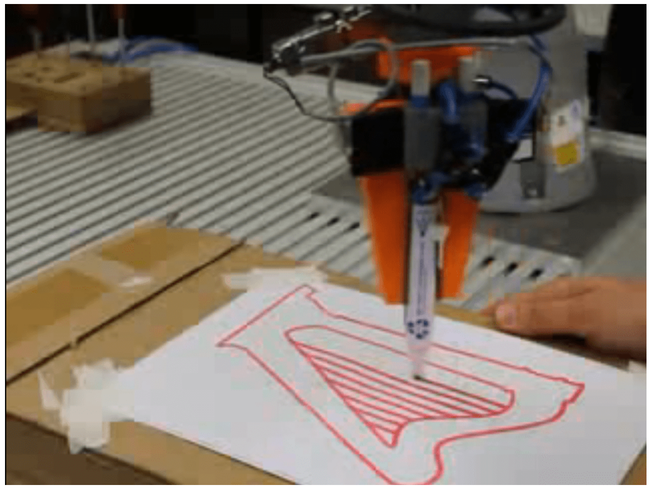


Fig. 7.5: IIWA drawing a harp

a laser source at the end-effector, and by using the outputs from the media flange to control the state of the laser source (turn on or off). Figure 7.5 shows the robot drawing the shape of a harp from a PLT file.
Associated video is in the link:
https://youtu.be/R9s02lBE6OQ

Testing 3D printed gripper

In this case KST was utilized to test a 3D printed griper for grasping objects, the gripper was built to be controlled from Matlab, once the object is recognized the robot moves to grasp it virtue to Matlab and KST. The Matlab controlled gripper allows the user to control the grasping force, the grasping stroke,also the velocity of the grasping motion .etc. Figure 7.6 show the robot grasping an object using the 3D printed gripper.
Associated video is in the link:
https://youtu.be/5ZVVzBlWo4o

Fig. 7.6: IIWA with 3D printed gripper

Human robot collision avoidance

The toolbox has been used to control KUKA iiwa robot in an industrial robotic cell with on the fly collision avoidance capacity, in those algorithms the robot is provided with a capacity to avoid collision with human coworker (dynamic obstacle) in addition to avoiding collision with static obstacles. The control algorithm used is based on potential fields method, the configuration of the human coworker in space is acquired using sensors, several sensors has been utilized ranging from laser scanners, IMUs and magnetic trackers. The algorithms were implemented in Matlab on an external computer, data acquisition was also implemented in Matlab, and KUKA iiwa was controlled from Matlab using KST, this allowed to speed up the implementation of the collision avoidance algorithm due to several factors:

- Matlab provides powerful and bug-free libraries for implementing complex mathematical calculations, which reduces the amount of programming required by the developer, allowing him to focus more on the scientific problem rather than getting distracted by programming those calculation algorithms.

Fig. 7.7: Pin in slot, car door study case

Fig. 7.8: Collision avoidance, car door study case

- Matlab comes with various toolboxes including control, instrumentation and signal processing toolboxes which can be used to speed up the development of the control and the data acquisition algorithms.
- Matlab provide plenty of drivers to interface with wide range of hardware, again reducing the amount of coding required to interface with the hardware.

In our lab, several use cases that implements collision avoidance algorithm has been realized, as listed in following subsections:

Car doors industry

In this study case KUKA iiwa is used for the assembly the of the internal plastic parts of car door, the task required from the robot is to insert pins inside slots in the car door, shown in Figure 7.7. At the same time a human coworker could interrupt the robot at any time, where he has to put a sticker on the car door, in such a case the robot shall respond on the fly avoiding collision with the coworker and giving the coworker a space to work, as shown in the sequence Figure 7.8.

In this use case a laser range finder and IMUs were used to capture the configuration of the human in space, the human and the robot are approximated by cylinders. Due to their geometry, cylinders offer a perfect compromise offering low computational cost of minimum distance calculation while reasonably representing the human and the robot, algorithms are developed using Matlab, and the robot is controller using KST toolbox. Associated video is in the link:

`https://youtu.be/mHhYHzWPqCc`

Part VIII. References

[1] Stürz, Y. R., Affolter, L. M., & Smith, R. S. (2017). Parameter Identification of the KUKA LBR iiwa Robot Including Constraints on Physical Feasibility. IFAC-PapersOnLine, 50(1), 6863-6868.

[2] Hayat, A. A., Abhishek, V., & Saha, S. K. Dynamic identification of manipulator: Comparison between cad and actual parameters. iNaCoMM, 15, 1-6.

[3] KUKA Roboter GmbH. Medien-Flansch Für Produktfamilie LBR iiwa Montage- und Betriebsanleitung. Stand: 20.01.2015. Version: Option Media Flange V3

[4] Brás, André & Simão, Miguel & Neto, Pedro. (2018). Gesture Recognition from Skeleton Data for Intuitive Human-Machine Interaction.