International Journal of Software Engineering and Knowledge Engineering
© World Scientific Publishing Company

Research Notes: Representation of Robots in Matlab

Antonio Sanchez[*], Luis Gracia[†], Ricardo Morales[*] and Carlos Perez-Vidal[*]

*Universidad Miguel Hernández, Avda. de la Universidad s/n,
03202 Elche, Spain

[†]*Instituto IDF, Universitat Politècnica de València, Camino de Vera s/n,*
46022 Valencia, Spain
luigraca@isa.upv.es

Received 29 May 2017
Revised 24 February 2018
Accepted 19 March 2018

This paper presents a new software tool, namely RoboClass, to include and manage realistic robots and elements of the environment in Matlab simulations. These elements are load from CAD models using an STL-file and can be as detailed as desired. All the steps involved in the process are explained in detail. Furthermore, two illustrative examples are considered to show the effectiveness and versatility of the proposed approach: the ABB-IRB120 industrial robot and the CSA research robot. The developed tool is especially useful both for robotics research and teaching.

Keywords: Robot simulation; Matlab; STL Importation.

1. Introduction

Nowadays, software tools are used in many engineering areas [1][2][3]. Matlab® is a technical software environment that has become a standard in many fields of engineering and education, including control systems and robotics [4]. This software allows the user to program using a high-level language specially designed for numerical computation, statistical analysis and data visualization. To this aim, Matlab includes a high amount of mathematical libraries and toolboxes that work fast and keep the code clear. In this sense, some toolboxes and open source code can be found for robotics although none of them are especially devoted to improve the visualization quality of the elements involved.

This article presents a new open-source software tool, namely *RoboClass*, developed to improve the appearance of serial robots to get a much more realistic representation environment in computer simulation, which is useful both for research and educational purposes. Each one of the different pieces is stored in a different Stereo-Lithographic file (STL) that the class will load to visualize as a whole. This allows the class to load 3D models from ASCII STL-files, that are easily readable by the user, or binary STL with short load times. With different configuration files, it is possible to generate different robots as objects with the capability of operate independently in the same simulation. Thus, RoboClass is beyond educational purposes and can be adapted to act as a general driver able to control and model the behavior of any serial robot using Matlab. Furthermore, it is possible to create multiple objects within the same workspace for research purposes like master-slave control algorithms or kinematic and dynamic simulation of parallel robots, amongst others.

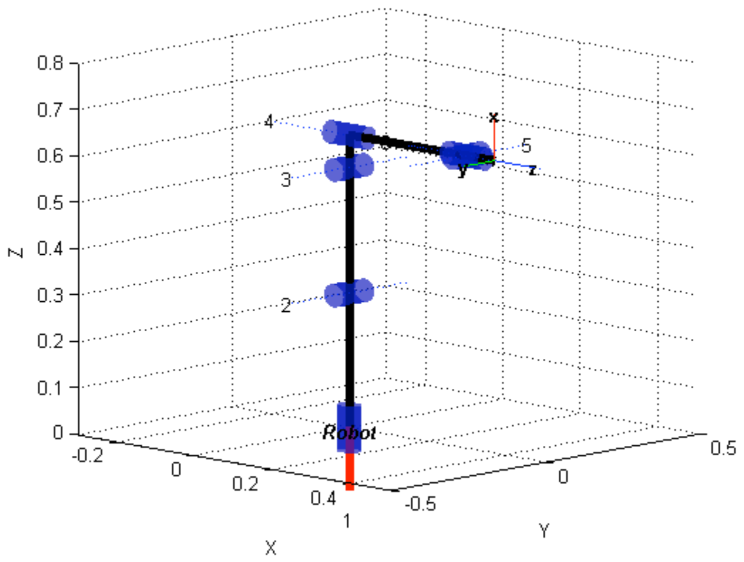


Fig. 1. Representation of a robot using the Robotic Toolbox by Corke.

A realistic simulation is especially useful for some applications of robotic research, like surgical interventions [5] or virtual reality [6], where the environment and the user perception are relevant.

The structure of the paper is as follows. Section 2 describes the libraries related with this work, while Section 3 describes the proposed method and the procedure to use it. Section 4 illustrates the use of the class for an industrial robot and a research robot. Finally some conclusions are presented in Section 5.

2. Libraries related with RoboClass

The Toolbox by P. Corke for Matlab [7] includes a set of functions to deal with the robot system, whereas its visualization is based on a wired drawing, see Fig. 1. This toolbox was the first one of its type and is still a reference for robotics research and teaching. In fact, it is probably the most used library in robotics nowadays all over the world and, typically, new contributions in this area are compared with it. The Kuka Control Toolbox [8] integrates a good amount of functions for robot simulation but only Kuka robots can be involved. The IGES Toolbox [9] is designed to load Initial Graphics Exchange Specification (IGES) format files to Matlab and operate with models. To do that, it adds commands and utilities to perform plots, transformations and projections. The Puma project [10] simulates a six degrees-of-freedom (DOFs) 3D Puma 560 robot with pre-loaded easy models to help the user on the visual comprehension of the robot position. In addition, it includes a GUI with slides to move the joints. This approach has inspired other practice program projects [11], where students are encouraged to choose a robot and build a simulator to better understand the kinematics related to it. Dynamics can be simulated using SimMechanics [12], which is a library in SolidWorks® software to model mechanical systems and that uses Simulink toolbox to perform the control of the systems. Simbad [13] is a Java 3D robot simulator dedicated to help researchers and programmers at study of situated artificial intelligence, machine learning, and artificial intelligence algorithms but, like others robot simulators as [14], [15] is focused on teaching purposes and not to use it in an actual environment.

3. Description of the Files and Configuration

As mentioned above, this work presents an open-source class to simulate serial robots in realistic Matlab environments. The class loads the STL-files of the robot (commercial robot or custom made robot) establishing the mechanical restriction for each joint. Joints can be defined as rotational or prismatic with a maximum of seven joints, although the class can be easily modified to consider robots with more than 7-DOFs. RoboClass has a wide group of functions to set and move the robot, see Fig. 2 and Fig. 3. These functions allow adjusting of the simulation view, computing the forward and inverse kinematics of the robot and operating with the robot dynamics. Moreover, the class also includes many functions of the Robotics Toolbox [7] to improve its versatility.

RoboClass creates robot objects from an external configuration files, therefore, it is possible to build different robots in the same workspace, perform cooperative simulations or even include realistic elements of the environment loading STL-files as well.

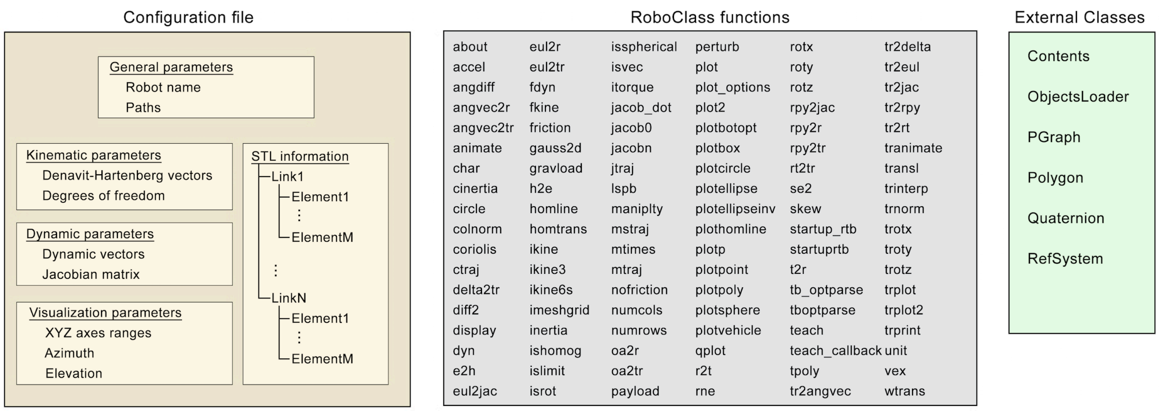
RoboClass has been developed for serial arms, but it is feasible to use it with other kinematic architectures such as parallel robots. In this case, the class should be modified to use it with this kind of mechanism but it will generate a different class for that particular mechanical structure.

As public information, a file called *RoboClass.zip* can be downloaded from the website: http://personales.upv.es/luigraca/RoboClass/code.htm

This file contains all the information regarding the class itself, Robotics Toolbox Matlab functions, several samples of use and many STL-files for different robots. The general procedure to work with a serial robot is described in 4 basic steps: (1) Generation or downloading the STL-files; (2) Editing of the configuration file; (3) Creation and use of functions; (4) Visualization. Next, the procedure to deal with these basic steps is detailed.

3.1. *Generation of STL-files*

The STL-file format can be generated by many mechanical software packages from a CAD model. It is widely used for rapid prototyping, 3D printing and computer-aided manufacturing. STL files describe only the surface geometry of a 3D object without any representation of surface texture, color or other model attributes. An STL file describes a raw unstructured triangulated surface by vertices. STL files contain no scale information, and the units are arbitrary.



Configuration file

General parameters
- Robot name
- Paths

Kinematic parameters
- Denavit-Hartenberg vectors
- Degrees of freedom

Dynamic parameters
- Dynamic vectors
- Jacobian matrix

Visualization parameters
- XYZ axes ranges
- Azimuth
- Elevation

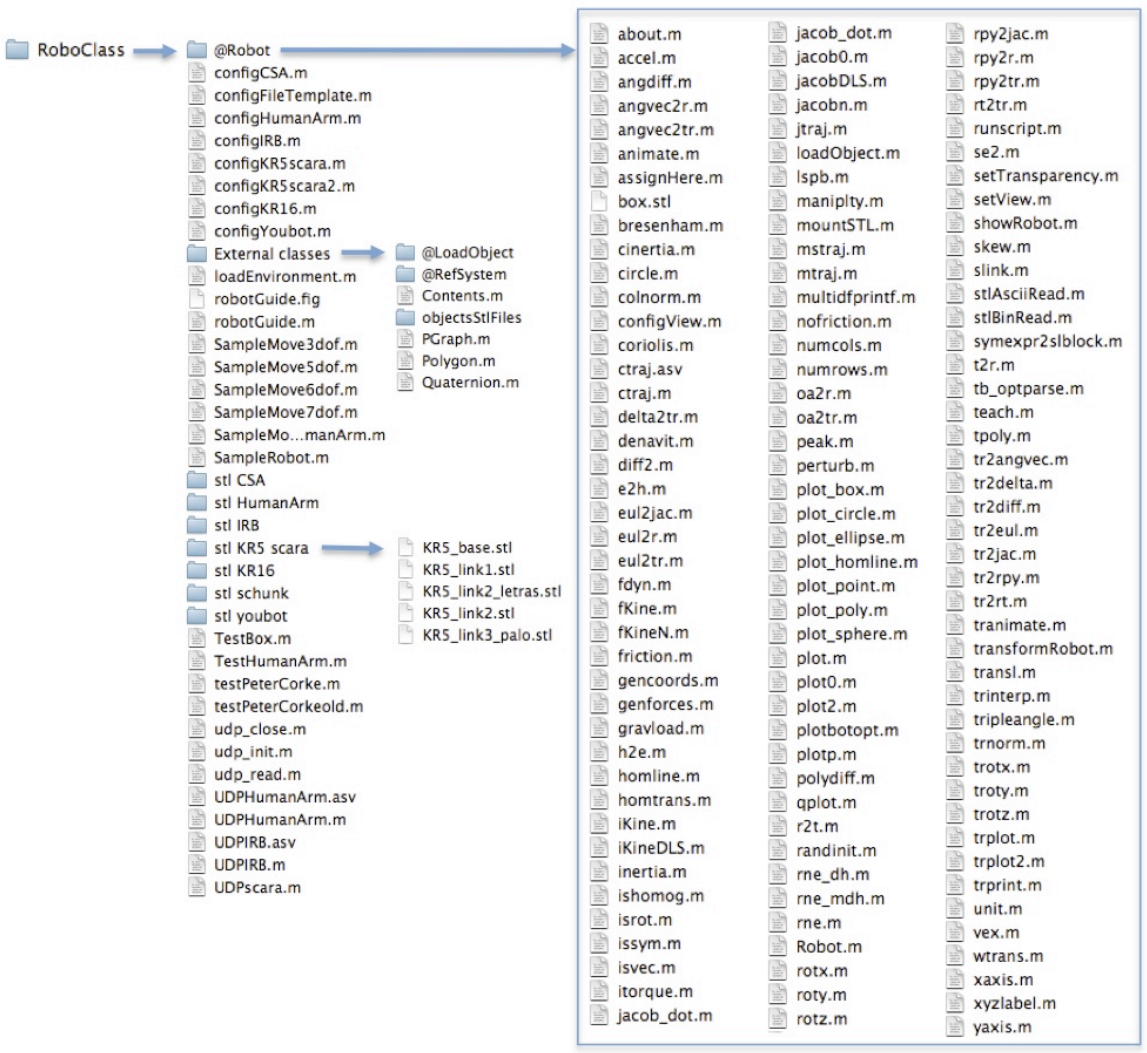
STL information
- Link1
  - Element1
  - ⋮
  - ElementM
- ⋮
- LinkN
  - Element1
  - ⋮
  - ElementM

RoboClass functions

about	eul2r	isspherical	perturb	rotx	tr2delta
accel	eul2tr	isvec	plot	roty	tr2eul
angdiff	fdyn	itorque	plot_options	rotz	tr2jac
angvec2r	fkine	jacob_dot	plot2	rpy2jac	tr2rpy
angvec2tr	friction	jacob0	plotbotopt	rpy2r	tr2rt
animate	gauss2d	jacobn	plotbox	rpy2tr	tranimate
char	gravload	jtraj	plotcircle	rt2tr	transl
cinertia	h2e	lspb	plotellipse	se2	trinterp
circle	homline	maniplty	plotellipseinv	skew	trnorm
colnorm	homtrans	mstraj	plothomline	startup_rtb	trotx
coriolis	ikine	mtimes	plotp	startuprtb	troty
ctraj	ikine3	mtraj	plotpoint	t2r	trotz
delta2tr	ikine6s	nofriction	plotpoly	tb_optparse	trplot
diff2	imeshgrid	numcols	plotsphere	tboptparse	trplot2
display	inertia	numrows	plotvehicle	teach	trprint
dyn	ishomog	oa2r	qplot	teach_callback	unit
e2h	islimit	oa2tr	r2t	tpoly	vex
eul2jac	isrot	payload	rne	tr2angvec	wtrans

External Classes
- Contents
- ObjectsLoader
- PGraph
- Polygon
- Quaternion
- RefSystem

Fig. 2. Class information schematic.



RoboClass → @Robot →

@Robot
configCSA.m
configFileTemplate.m
configHumanArm.m
configIRB.m
configKR5scara.m
configKR5scara2.m
configKR16.m
configYoubot.m
External classes →
loadEnvironment.m
robotGuide.fig
robotGuide.m
SampleMove3dof.m
SampleMove5dof.m
SampleMove6dof.m
SampleMove7dof.m
SampleMo...manArm.m
SampleRobot.m
stl CSA
stl HumanArm
stl IRB
stl KR5 scara →
stl KR16
stl schunk
stl youbot
TestBox.m
TestHumanArm.m
testPeterCorke.m
testPeterCorkeold.m
udp_close.m
udp_init.m
udp_read.m
UDPHumanArm.asv
UDPHumanArm.m
UDPIRB.asv
UDPIRB.m
UDPscara.m

@LoadObject
@RefSystem
Contents.m
objectsStlFiles
PGraph.m
Polygon.m
Quaternion.m

KR5_base.stl
KR5_link1.stl
KR5_link2_letras.stl
KR5_link2.stl
KR5_link3_palo.stl

about.m	jacob_dot.m	rpy2jac.m
accel.m	jacob0.m	rpy2r.m
angdiff.m	jacobDLS.m	rpy2tr.m
angvec2r.m	jacobn.m	rt2tr.m
angvec2tr.m	jtraj.m	runscript.m
animate.m	loadObject.m	se2.m
assignHere.m	lspb.m	setTransparency.m
box.stl	maniplty.m	setView.m
bresenham.m	mountSTL.m	showRobot.m
cinertia.m	mstraj.m	skew.m
circle.m	mtraj.m	slink.m
colnorm.m	multidfprintf.m	stlAsciiRead.m
configView.m	nofriction.m	stlBinRead.m
coriolis.m	numcols.m	symexpr2slblock.m
ctraj.asv	numrows.m	t2r.m
ctraj.m	oa2r.m	tb_optparse.m
delta2tr.m	oa2tr.m	teach.m
denavit.m	peak.m	tpoly.m
diff2.m	perturb.m	tr2angvec.m
e2h.m	plot_box.m	tr2delta.m
eul2jac.m	plot_circle.m	tr2diff.m
eul2r.m	plot_ellipse.m	tr2eul.m
eul2tr.m	plot_homline.m	tr2jac.m
fdyn.m	plot_point.m	tr2rpy.m
fKine.m	plot_poly.m	tr2rt.m
fKineN.m	plot_sphere.m	tranimate.m
friction.m	plot.m	transformRobot.m
gencoords.m	plot0.m	transl.m
genforces.m	plot2.m	trinterp.m
gravload.m	plotbotopt.m	tripleangle.m
h2e.m	plotp.m	trnorm.m
homline.m	polydiff.m	trotx.m
homtrans.m	qplot.m	troty.m
iKine.m	r2t.m	trotz.m
iKineDLS.m	randinit.m	trplot.m
inertia.m	rne_dh.m	trplot2.m
ishomog.m	rne_mdh.m	trprint.m
isrot.m	rne.m	unit.m
issym.m	Robot.m	vex.m
isvec.m	rotx.m	wtrans.m
itorque.m	roty.m	xaxis.m
jacob_dot.m	rotz.m	xyzlabel.m
		yaxis.m

Fig. 3. Description of files included in RoboClass.

As far as this file is only used for display and not to perform the calculations, a low quality model results in an image with fewer triangles, which will reduce the load time and improve the frame-rate of the image in the simulation. The class loader can load both ASCII and binary files, although the latter are loaded considerably faster. However, the ASCII files' information is readable for the user, which is useful to hand-edit it. All the STL-files must be placed in the same folder. This path will be indicated in a field at the configuration file. With all the files ready, open Matlab and select the folder that contains RoboClass as *Current folder*.

3.2. *Configuration file*

The configuration file creates a structure with all the parameters related to the robot behavior and the information about the STL-files to load. This data structure is sent as an argument to the constructor function to create a robot object. Thus, multiple different robots can be simulated in a single workspace. The parameters contained in the data structure of the configuration file are divided into different fields.

- General parameters: Robot name and paths.
- Kinematics: Degrees-of-freedom (DOFs), Denavit–Hartenberg (DH) parameters and joint vector.
- Dynamics: link dynamic properties.
- STL info: Link number, file name of the elements of the link and their colors.
- View info: Axes range limits and camera position.

The data structure has another field called *Real*. The purpose of this field is containing the drivers to manipulate an actual robot. Thus, this field will have the Matlab conversion of the manufacturer commands, which are different for each robot brand. The developing related to this field is not included in this work and remains as further work.

Next, the five fields indicated above for the configuration file are explained.

General Parameters. The user must introduce the name of the robot model. If the main path has the pwd command it will detect automatically the class path. In *stlpath* specify the folder where the STL-files are.

```
nameRobot = <'robotName'>;
path = pwd;
stlpath = <STL files path>;
```

Kinematics. Robot kinematics describes the motion of the rigid bodies related to the robot joint angles. The kinematic parameters of the configuration file are: the robot DOFs, the DH parameters and a vector with the rotational or prismatic character for each joint as shown below:

```
dof = <degrees of freedom>;
originTr = <placement matrix>;
joinType = <joints vector>;
theta = <DH theta vector>; % theta is the kinematic joint angle
d = <DH d vector>;         % d is the kinematic link offset
a = <DH a vector>;         % a is the kinematic link length
alpha = <DH alpha vector>; % alpha is the kinematic link twist [2]
```

Dynamics. Robot dynamics describes how the robot moves in response to the actuator torques or forces. Dynamic vectors and the Jacobian matrix of each joint define this information. Dynamic parameters must be given by the manufacturer in order to perform dynamic calculations, as can be seen in Fig. 4.

$$\begin{aligned}
m &= \begin{pmatrix} m_1 & m_2 & \dots & m_n \end{pmatrix}\\
cg\{1\} &= \begin{pmatrix} cg1_1 & cg1_2 & cg1_3 \end{pmatrix}\\
&\vdots\\
cg\{n\} &= \begin{pmatrix} cgn_1 & cgn_2 & cgn_3 \end{pmatrix}\\
J\{1\} &= \begin{pmatrix} j1_{(1,1)} & j1_{(1,2)} & j1_{(1,3)}\\ j1_{(2,1)} & j1_{(2,2)} & j1_{(2,3)}\\ j1_{(3,1)} & j1_{(3,2)} & j1_{(3,3)} \end{pmatrix}\\
&\vdots\\
J\{n\} &= \begin{pmatrix} jn_{(1,1)} & jn_{(1,2)} & jn_{(1,3)}\\ jn_{(2,1)} & jn_{(2,2)} & jn_{(2,3)}\\ jn_{(3,1)} & jn_{(3,2)} & jn_{(3,3)} \end{pmatrix}
\end{aligned}$$

Fig. 4. Dynamics of the robot.

STL Info. The STL structure contains a field for each of the N-links of the robot. Each link has a field with the number of the previous joint and other M-fields, one for each element composing the link. Each element field has the name and color of one STL-file.

```
LinkN.Link = <linkNumber>; LinkN.robot = [];
LinkN.element1.name = <'nameStlFile'>;
LinkN.element1.color = <color in [r g b] format>;
                        ...
LinkN.elementM.name = <'nameStlFile'>;
LinkN.elementM.color =<colorin[rgb]format>;
```

View Info. The view field contains the plot values: maximum and minimum limit of each axis and the angles of the camera (azimuth, i.e. horizontal rotation, and the vertical elevation).

```
minX = <minimum value axis X>;
maxX = <maximum value axis X>;
minY = <minimum value axis Y>;
maxY = <maximum value axis Y>;
minZ = <minimum value axis Z>;
maxZ = <maximum value axis Z>;
cameraAZ = <azimuth>;
cameraEL =<elevation>;
```

3.3. *Handling functions*

To create robot objects it is necessary to call the constructor with the configuration file as parameter. To do that, the line below must be written in Matlab command window.

```
≫ MyRobot = Robot(config());
```

Once the robot is created, it is possible to use all their functions. As shown in Fig. 2 and Fig. 3, most of the functions of the Robotics Toolbox [2] are included to help the robot handling. Fig. 3 shows the complete set of functions contained in @Robot folder together with the functions to configure and test this Class. The files contained into the External Classes folder define the data structure meanwhile the STL-files are in separate folders. In the application examples shown in Section 4 and Section 5 there are some examples of use. The function showRobot places the robot in a plot. If the function is called without arguments, the robot will be showed in the export position or configuration.

```
≫ MyRobot.showRobot();
```

If the function is called has a vector argument with the αi angles of the joints, the robot will be showed in the configuration defined by the vector.

```
≫ MyRobot.showRobot([α1 α2 α3 α4 α5 α6]);
```

In order to visualize a robot in movement insert this function within a *for loop*. The function *configView* can adjust the viewing parameters. If the p does not receive parameters, it adjusts the plot view to a standard.

```
≫ MyRobot.configView();
```

If this function receives a vector v, it can adjust the view of the plot. This vector can have three different sizes: azimuth (Az) and vertical elevation (El), the XYZ axes limits, or all of them.

```
v = [Az El]
v = [XMIN XMAX YMIN YMAX ZMIN ZMAX]
v = [XMIN XMAX YMIN YMAX ZMIN ZMAX Az El]
≫ MyRobot.configView(v);
```

The user can simulate the interaction between the robot and other objects. To insert objects in the workspace use the external class *LoadObject (<fileName>, <color>, Tr)*:

```
≫ MyObject = LoadObject('box', [1 0 0], eye(4));
```

To create and show the axes of the reference system use the external class *RefSystem*:

```
≫ SR=RefSystem();
```

Note that it is required to add the *External classes* folder to the Matlab path in order to create objects of these classes. To do this, in Matlab interface, click with the right button of the mouse on the folder you want to add and select *Add to Path→Selected Folders*.

3.4. *How to make a GUI*

It may be interesting to create a visual interface to modify the position of the robot. This section explains the general procedural to create a GUI and how to adapt it to a robot. The applied steps are shown for the ABB application example in this work.
The steps required to get a GUI are described below:

- Type the command *guide* in the Matlab console and the quick start window will appear. In that window, select *Blank GUI* and press *OK button*. In the editing window add a slider object for each of the robot joints. Click on the image of each slide with the second mouse button to open the properties. To be able to identify what slide is linked to each joint, rename the field *slider* of each slide with the name of the joint that the slide will change. Edit *Max* and *Min* fields with maximum and minimum rotation for each joint too. Save the file as *guideRobot.m* and open that file with the text editor.

- Execute the function *robotGuide_OpeningFcn* after the guide is opening. That function must create the robot and a vector to handle the angles from the slides.

```
≫ handles.MyRobot = Robot(configFile());
≫ handles.q = zeros(1,handles.MyRobot.kinematics.dof);
```

- Link each slide to the robot joint angles. To do that, it is necessary to add some code under the *callback function* of each slide. The code must get the angle from the slide, set the robot position and save the values with *guidata* function.

```
≫ handles.q(<jointN>) = get(hObject,'Value');
≫ handles.MyRobot.plot(handles.q);
≫ guidata(hObject, handles);
```

- Follow the step above to add as much elements as needed. The general guide is described below:
  - shown with the slides,
  - add an element in the editing window,
  - open the guide file,
  - add the code linking the robot parameters with the object values.

3.5. *Denavit–Hartenberg parameters*

The DH parameters have to be properly defined to use the proposed method for graphically representing the robot system. These parameters are used for attaching reference frames to the links of a spatial kinematic chain or robot. The procedure to obtain the DH parameters is depicted in Fig. 5 and it is composed of the following steps:

1. Identify links and joints: Links are numbered from 0 (base) to n (end-effector). Joints are numbered from 1 to n. In this version of the procedure, joint i connects links i-1 and i.
2. Define the reference frames for the internal links: Locate z_i axis along the axis of joint i+1. The origin of the frame O_i is positioned along joint i+1 axis. If the z axes

are parallel O_i is arbitrarily chosen. Otherwise, it is located in the intersection between z_i and the common normal to z_{i-1} and z_i. y_i axis is chosen to compose a right-hand frame.

3. Define the reference frames for the extremities links: z_0 is located along the axis of joint 1. x_0 and y_0 are arbitrary. x_n axis is normal to the joint n axis, while y_n and z_n are arbitrarily defined.
4. Identify the DH parameters for each link: a_i is the distance between z_{i-1} and z_i. d_i is the distance between x_{i-1} and x_i. a_i is the angle between z_{i-1} and z_i measured along x_i, while y_i is the angle between x_{i-1} and x_i, measured along z_i.
5. Determine the homogeneous transformation matrices for each joint.
6. Determine the overall homogeneous transformation matrix by premultiplication of the individual joint transformation matrices.

The homogeneous transformation matrix ${}^{i-1}A_i$ mentioned in Step 5 can be seen as a composition of rotations and translations by the DH parameter values in order to move a frame coincident to frame i-1 until it coincides with frame i [1]. Its complete form is in Eq. (1), where *c* and *s* correspond to the trigonometric functions cos(·) and sin(·), respectively. The overall transformation matrix is defined in Eq. (2), where ${}^{0}R_n$ and ${}^{0}p_n$ represent the orientation (in rotation matrix form) and the position, respectively, of the robot end-effector.

$$ {}^{i-1}\mathbf{A}_i = \begin{bmatrix} c_{\theta_i} & -c_{\alpha_i}s_{\theta_i} & c_{\alpha_i}s_{\theta_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\alpha_i}c_{\theta_i} & -s_{\alpha_i}c_{\theta_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1} $$

$$ {}^{0}\mathbf{A}_n = {}^{0}\mathbf{A}_1 {}^{1}\mathbf{A}_2 \cdots {}^{n-1}\mathbf{A}_n = \begin{bmatrix} {}^{0}\mathbf{R}_n & {}^{0}\mathbf{p}_n \\ \mathbf{0} & 1 \end{bmatrix} \tag{2} $$

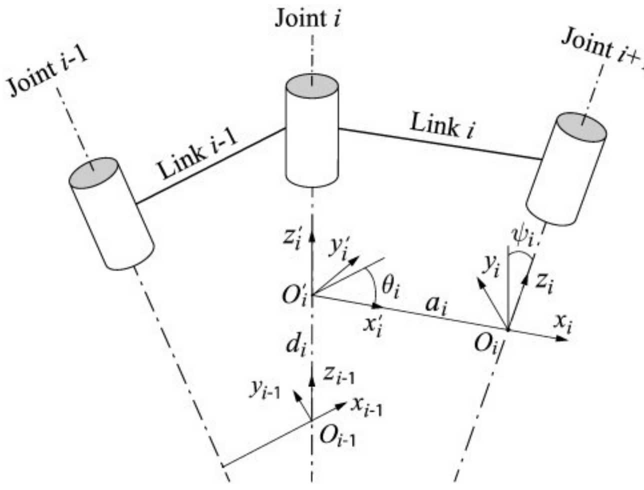


Fig. 5. Graphical representation of the Denavit–Hartenberg parameters.

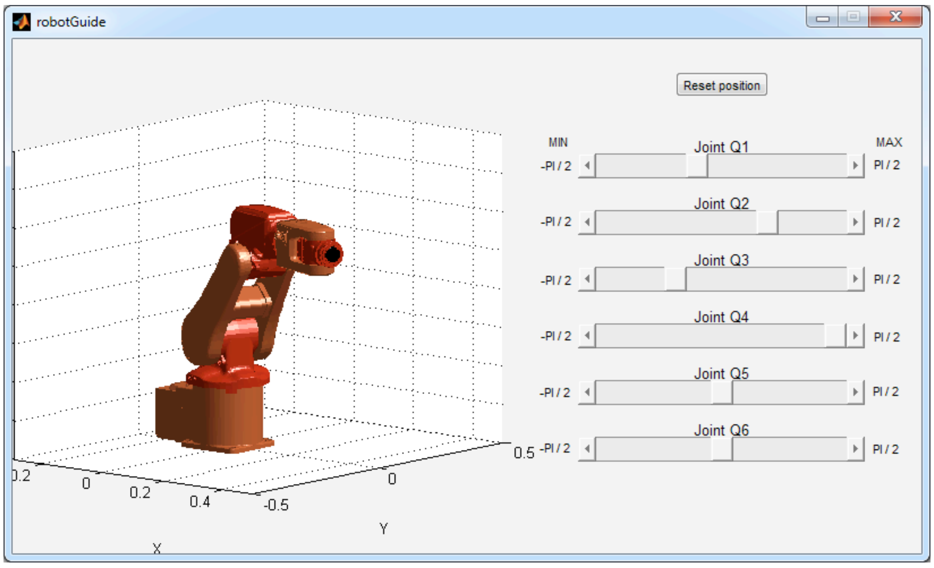


Fig. 6. Representation of the ABB-IRB120 robot with a guide (visual interface).

4. Application Examples

Two examples are shown below to illustrate the applicability of the proposed class. The first one is referred to the *ABB-IRB120*, see Fig. 6, which is a 6-DOFs commercial industrial robot. It will be shown how to export the links of the *IRB120 robot from Autodesk Inventor*, to set the configuration file to consider its behavior, to use the basic functions and to make a GUI to move each joint.

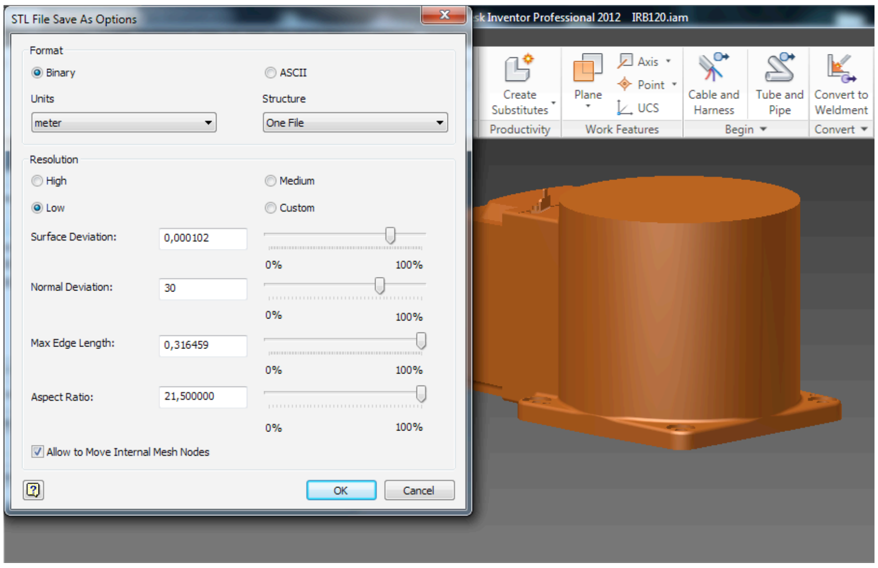
4.1. *Export STL-files*

One of the available options to generate the STL-files is through Autodesk Inventor® although the process is analogue for any other CAD modeling tool. Once the robot links and any other possible objects are loaded on the workspace, the robot base frame has to be located at the axes origin and the auxiliary frames for each robot link must be located according to the DH convention. Then, each piece is exported individually to a file.

STL-files can be exported in ASCII mode or binary mode. To export each file in Autodesk Inventor, uncheck the visibility field of all the pieces of the robot except the one to be exported. To do this, select all the pieces to hide and click with the right mouse button. In the emergent menu, uncheck the visibility parameter. With only one piece in the workspace, select *File→Export→Cad model*. The export window specifies the name and type of the file to be created, in this case STL. If you click on the options button, you can select the file type and the resolution of the new model.

First, the CAD files (STEP type) have been downloaded from the ABB website [16]. The representation of this robot is made of 7 files. The first file has the base model of the robot. The base must be placed on the origin of the coordinates system. The other six links will be placed according to the DH convention. To export the base model in a STL-file Autodesk Inventor software, amongst others, can be used following the next steps. Uncheck the visibility field of the other pieces but the robot base. To do this, select all the models of the robot and then unselect the base model. Click with the right mouse button on the selected group of pieces and uncheck the visibility parameter. With only the base model in the workspace, select *File→Export→Cad format*.

In the export window shown in Fig. 7, introduce the name of the file to be created, and select STL-file in the type field. Click on *options* button, select *binary format* and set the resolution of the model. It is possible to check the export resolution clicking on the preview button. Note that less resolution results in a model with fewer faces. This impacts directly on the appearance of the link/s, see Fig. 8 and the performance of the simulation, see Fig. 9.



STL File Save As Options

Format

[✓ Binary] ASCII

Units: meter

Structure: One File

Resolution

High Medium

[✓ Low] Custom

Surface Deviation: 0,000102

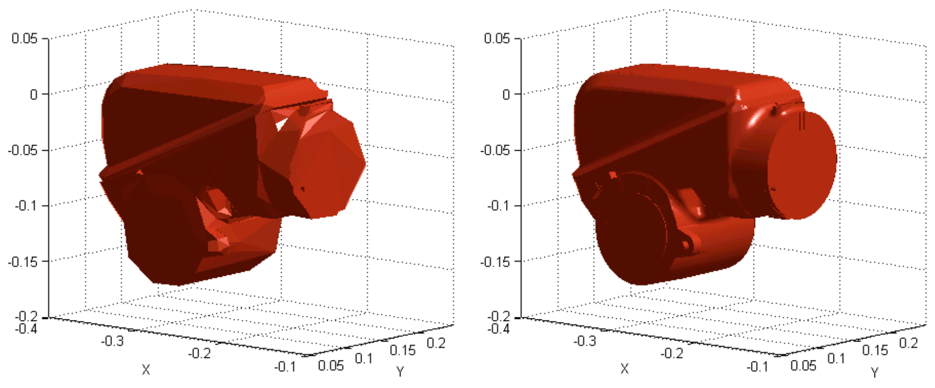
Normal Deviation: 30

Max Edge Length: 0,316459

Aspect Ratio: 21,500000

[✓] Allow to Move Internal Mesh Nodes

Fig. 7. STL options file window.



a) 862 faces IRB120 Link 3 model

b) 43.146 faces IRB120 Link 3 model

Fig. 8. Differences between model qualities.

Different simulations have been done to compare the load time of STL ASCII and binary files depending on the number of triangles. Table 1 shows load times for a complete robot. Loading binary files is always faster than loading ASCII files. However, as the complexity of the model increases, the difference between load times raises dramatically. Fig. 9 shows how fast raises up ASCII load time when the number of triangles increases in comparison to binary load times. In this example there are 7 files:

base, five links and final effector. All the files are stored in a folder called stl IRB beside to RoboClass folder.

Table 1. Load time of STL files.

Face number	Load ascii time (s)	Load binary time (s)	Ratio
15120	4.195026	0.076050	55
30249	10.873158	0.118491	92
75622	48.718367	0.214152	227
151255	200.434706	0.401590	499
302513	788.170335	0.777622	1014
756290	5037.88186	2.308548	2182

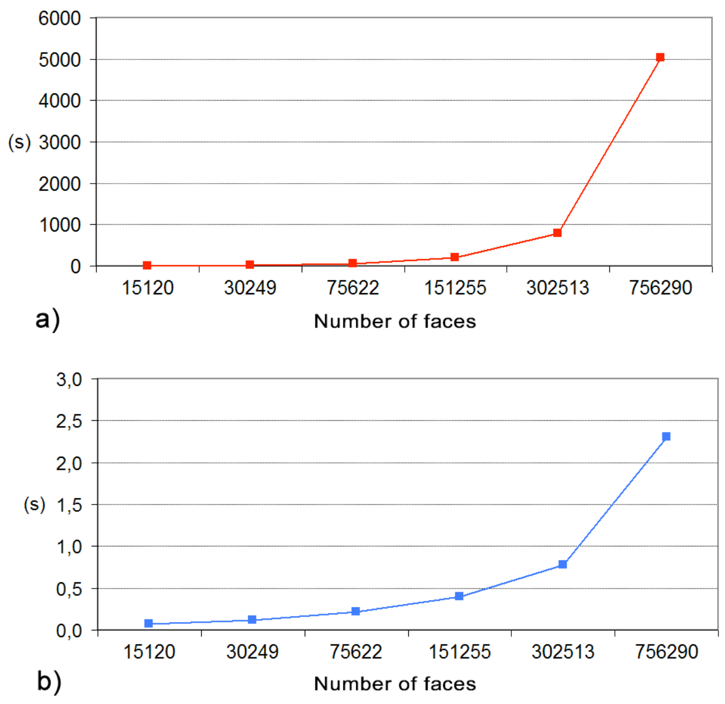


Fig. 9. Load time vs. Number of faces. a) ASCII files. b) Binary files.

4.2. *Configuration file*

Open the configuration file template, save it as configIRB.m and begin to adapt the parameters to the ABB-IRB120 robot.

General Parameters. The robot name and paths are defined in the general parameters. The robot name is 'IRB120'. The general path has the pwd command to detect automatically the class path. The STL-files are all in a folder called stl IRB.

```
nameRobot = 'IRB120';
path = pwd;
stlpath = [pwd '\ stl IRB \'];
```

Kinematics. The robot has 6-DOFs and all the joints have a rotational behavior. Moreover, the DH parameters (theta, d, a, alpha) are obtained applying the DH algorithm to the IRB120 robot.

```
dof = 6; originTr = eye(4);
jointType = [’r’ ’r’ ’r’ ’r’ ’r’ ’r’];
theta = [0 -π/2 0 0 0 0];
d = [0.29 0 0 0.30 0 0.072];
a=[0 0.27 0.07 0 0 0];
alpha = [-π/2 0 -π/2 -π/2 π/2 0];
```

Dynamics. The dynamic parameters are required to perform dynamic calculations and must be given by the manufacturer and filled as indicated in previous section.

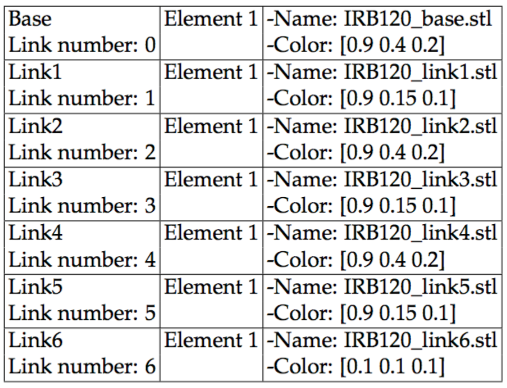
STL Info. As shown in Table 2, the base and each link are made by one element. Each element has a field with the name of the STL-file and another field with the color of the model in Red-Green-Blue (RGB) format. The code must be modified as shown in previous section.

```
base.Link=0;
base.robot=[];
base.element1.name='IRB120\_base';
base.element1.color=[0.89 0.39 0.21];
```

View Info. The selected maximum and minimum axes limit and camera angles are shown below:

minX = -0.4; minY = -0.8; minZ = -0.4; maxX=1.0; maxY=1.8; maxZ=1.0;
AZ = 40; EL=10;

Table 2. STL structure for the ABB-IRB120 robot.



Base Link number: 0	Element 1	-Name: IRB120_base.stl -Color: [0.9 0.4 0.2]
Link1 Link number: 1	Element 1	-Name: IRB120_link1.stl -Color: [0.9 0.15 0.1]
Link2 Link number: 2	Element 1	-Name: IRB120_link2.stl -Color: [0.9 0.4 0.2]
Link3 Link number: 3	Element 1	-Name: IRB120_link3.stl -Color: [0.9 0.15 0.1]
Link4 Link number: 4	Element 1	-Name: IRB120_link4.stl -Color: [0.9 0.4 0.2]
Link5 Link number: 5	Element 1	-Name: IRB120_link5.stl -Color: [0.9 0.15 0.1]
Link6 Link number: 6	Element 1	-Name: IRB120_link6.stl -Color: [0.1 0.1 0.1]

4.3. *Handling functions*

An IRB120 robot object is created when the constructor is called with the IRB120 configuration file as argument.

```
≫ MyIRB = Robot(configIRB());
```

To show a plot of the robot in the export position use the *showRobot* function.

```
≫ MyIRB.showRobot()
```

To set the transparency level to 50% use the function *setTransparency* as follows:

```
≫ MyIRB.setTransparency(0.5)
```

To set the robot in a specific configuration, introduce a vector with the joint angles in radians.

```
≫ MyIRB.showRobot([pi/4 pi/4 0 pi/2 0 0])
```

To simulate a movement use a for loop and the *drawnow* command as follows:

```
≫ for i=0:0.05:pi/2,
≫ MyIRB.showRobot([i/4 i/4 0 i 0 0]);
≫ drawnow;
≫ end;
```

If needed, use the *configView* function to adjust the viewing parameters. Calling the function without an argument, i.e. MyIRB.configView(), will set the viewing parameters to x[-1,1], y[-1,1], z[0,1], azimuth to 40 degrees and vertical elevation to 10 degrees. To set a specific view, e.g., x[-0.5,0.5], y[-0.4,0.4], z[0,0.8], azimuth=40 and vertical elevation=10, type the command:

```
≫ MyIRB.configView([-0.5 0.5 -0.4 0.4 0 0.8 40 10]);
```

4.4. *Makin a GUI*

Create a *Blank GUI* as explained in subsection 2.4. In the editing window guide add six slides, one for each of the robot joints. Open the properties of the first slide and rename the *slider* field with the name *slide_Q1* and the *Max* and *Min* fields with rotation the rotation limits. Repeat for each joint the steps described for *slide_Q1*. Save the file as *guideIRB.m* next to configuration files, and open that file with the text editor. At the function *robotGuide_OpeningFcn* create the robot and the vector to handle the angles from the slides:

```
≫ handles.MyIRB = Robot(configIRB());
≫ handles.q = zeros(1,handles.MyIRB.kinematics.dof);
```

To link the slides to the robot joints and show it in the guide, follow the instructions below. Under *slider_Q1_Callback* get the angle from the slide and show the robot with the updated position. Then, save the values with *guidata* function.

```
≫ handles.q(1) = get(hObject,'Value');
≫ handles.MyIRB.showRobot(handles.q);
≫ guidata(hObject, handles);
```

To add more elements to the guide as in Fig. 6 the same steps are repeated.

4.5. *Second application example*

The second example is referred to a 7-DOFs robot arm developed at the Miguel Hernandez University: the Custom Schunk Arm (CSA). The procedure is basically the same to that used for the IRB120 example in Sections 3.1 to 3.4, except that the CSA comprises more elements: 36. Details omitted for brevity. The result obtained for this second example is shown in Fig. 10, while Fig. 11-a) to Fig. 11-i) represent a sequence of nine frames showing the CSA robot during the tracking of a linear trajectory.

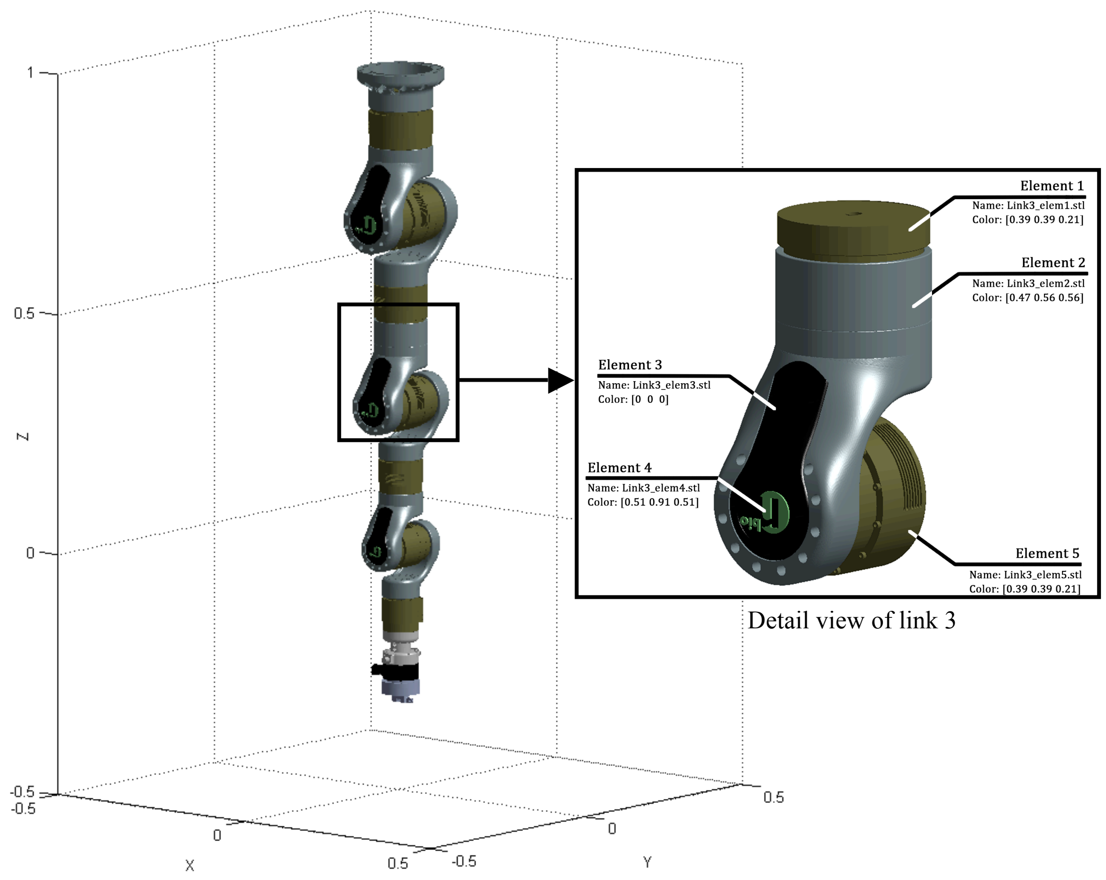


Fig. 10. Custom Schunk Arm (CSA), a 7-DOFs robot, with a detail view of link 3

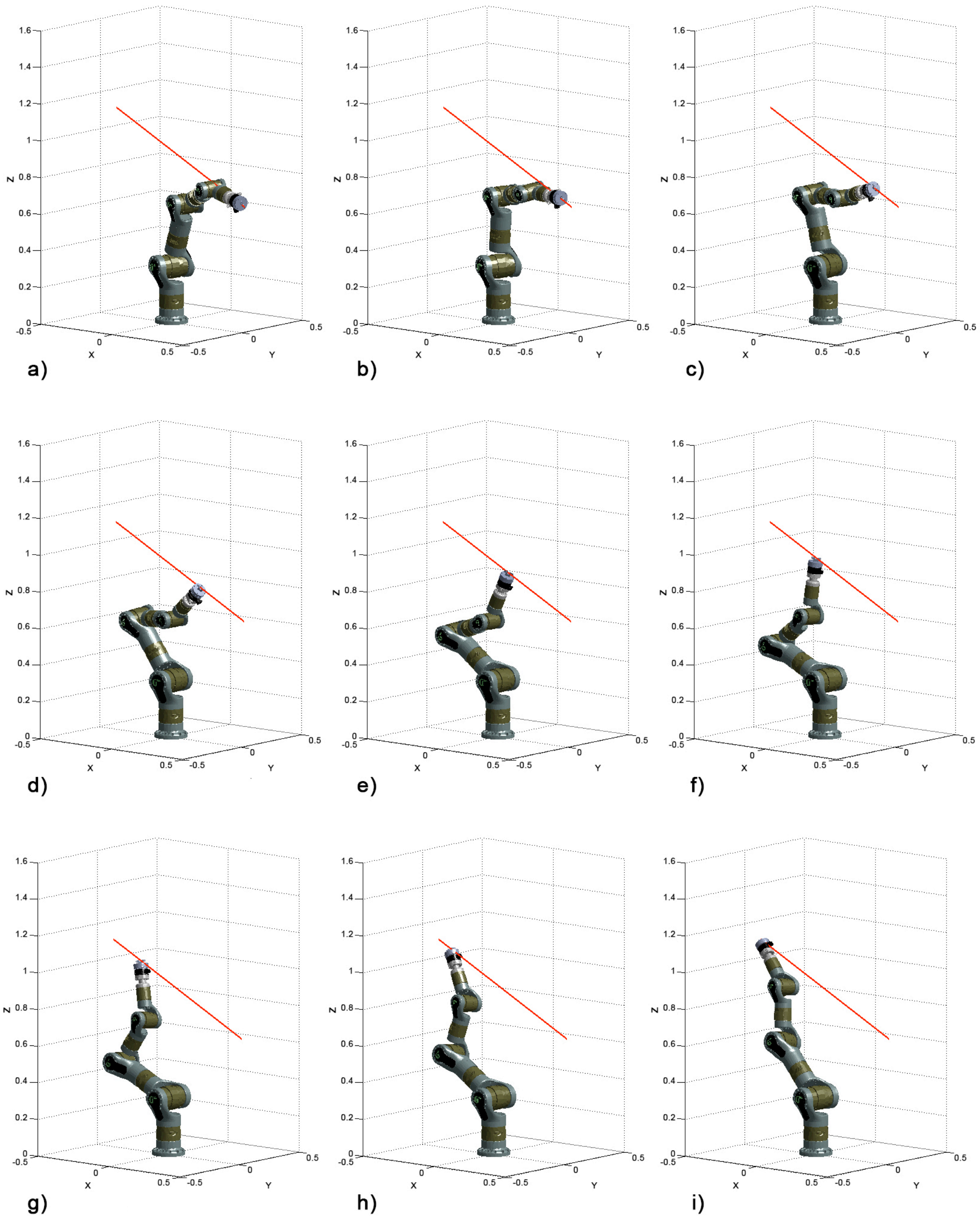


Fig. 11. CSA movement through a linear trajectory.

5. Conclusion

In this article a novel open-source software tool to simulate realistic serial robots in Matlab has been presented. The library allows loading the CAD models of the robot links from ASCII or binary STL format in order to avoid a representation of the robot based on wires. As such, it allows performing very realistic simulations (e.g., see Fig. 6 in comparison to Fig. 1) in an easy and fast way, which is useful both for research and teaching purposes due to the increase in simulation accuracy. The effectiveness and usability of RoboClass use has been demonstrated throughout two application examples. The class presented in this paper is ready to adapt the CAD models of each robot manufacturer to Matlab environment in order to control serial robots in a realistic scene. This adaptation must be done only once for a given robot before using it.

Since the class creates the robot parts from external configuration files, it is possible to include different robots in the same workspace, see Fig. 12, or even to include other elements of the manufacturing process (e.g., elements of the production line, security elements, manipulated pieces, etc.) in order to get a more complete simulation scene.

Although the class has been developed for serial arms, it is possible to use it with other kinematic architectures, like parallel kinematic machines. The class is basically a wrapper for the mechanical information (STL and dynamic parameters) and the robot analysis libraries (Robotics Toolbox). When using another toolbox for parallel robots, like the one for a 3UPS1S robot in [17] or the toolbox for planar 3RRR device [18], the class could be extended to use it with this kind of mechanism but it will be a different class for each parallel robot, not a generalized class like in the case of serial robots.

The developed tool is especially useful both for robotics research and teaching in order to show a realistic representation of the robot and its environment in the widespread software Matlab. Moreover, the user can customize the open source code to its requirements and use the advanced functions to get an accurate simulation.

The main advantages of the simulation tool developed in this work are listed below: It can be used both for *research and education* purposes; It allows a *realistic representation* of the robot and its environment; It is compatible with *kinematic and dynamic* analysis; It can be easily extended to be used with *parallel robots*.

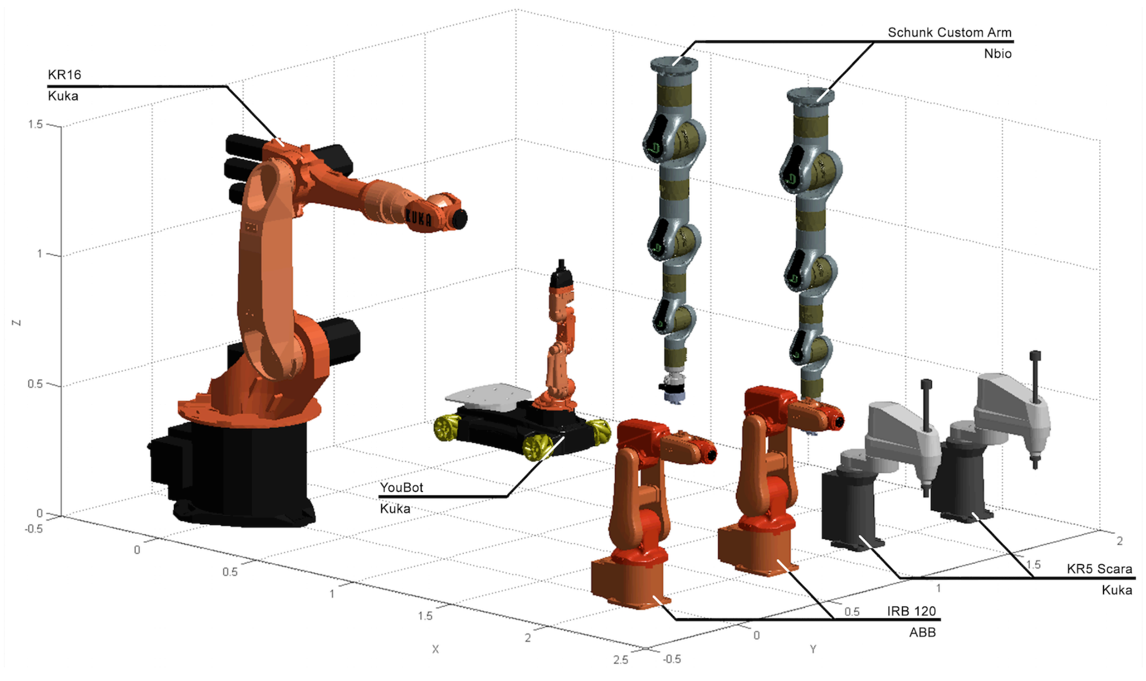


Fig. 12. Multiple robot simulation.

References

[1] M. Bashari, E. Bagheri and W. Du, "Dynamic Software Product Line Engineering: A Reference Framework", International Journal of Software Engineering and Knowledge Engineering, 2017, Vol. 27, No. 2, pp. 191-234.

[2] K. Honda, H. Washizaki and Y. Fukazawa, "Generalized Software Reliability Model Considering Uncertainty and Dynamics: Model and Applications", International Journal of Software Engineering and Knowledge Engineering, 2017, Vol. 27, No. 6, pp. 967-993.

[3] Z.K. Aghdamand B. Arasteh, "An Efficient Method to Generate Test Data for Software Structural Testing Using Artificial Bee Colony Optimization Algorithm", Int. Journal of Software Engineering and Knowledge Engineering, 2017, Vol. 27, No. 6, pp. 951-966.

[4] M. Flanders and R.C. Kavanagh, "Build-A-Robot: Using virtual reality to visualize the Denavit–Hartenberg parameters", *Computer Applications in Engineering Education*, 2015, Vol. 23, No. 6, p. 846-853.

[5] N.A. Bates, A.L. McPherson, R.J. Nesbitt, J.T. Shearn and G.D. Myer, T.E. Hewett. Robotic simulation of identical athletic-task kinematics on cadaveric limbs exhibits a lack of differences in knee mechanics between contralateral pairs, Journal of Biomechanics, Vol. 53, No. 28, February 2017, pp. 36-44.

[6] A. Doshi, R.T. Smith, B.H. Thomas and C. Bouras, "Use of projector based augmented reality to improve manual spot-welding precision and accuracy for automotive manufacturing", The Int. Journal of Advanced Manufacturing Technology, 2017, Vol. 89, No. 5-8, pp. 1279-1293.

[7] P. Corke, "A Robotics Toolbox for MATLAB", *IEEE Robotics and Automation Magazine*, March 1996, Vol.3-1, p.24-32.

[8] F. Chinello, S. Scheggi, F. Morbidi and D. Prattichizzo, "Kuka Control Toolbox", *IEEE Robotics and Automation Magazine*, December 2011, Vol.18-1, p.69-79.

[9] PerBergström,"IGESToolbox".Mathworks (accessed May 2017), http://www.mathworks.com/matlabcentral/fileexchange/13253-iges-toolbox.

[10] D. Riley, "3D Puma Robot Demo". Mathworks (accessed May 2017) http://www.mathworks.com/matlabcentral/fileexchange/14932-3d-puma-robot-demo.

[11] Y. Cao, "Learning robotics through developing a virtual robot simulator in Matlab", *American Society for Engineering Education*, August 2011, AC 2011-609.

[12] SimMechanics, The Mathworks, *User 's Guide*, 2005.

[13] L. Huges, N. Bredeche and T. Futurs, "Simbad: an Autonomous Robot Simulation Package for Education and Research", *Proceedings of The International Conference on the Simulation of Adaptive Behavior* (SAB'06), Roma, Italy - Springer's Lecture Notes in Computer Sciences / Artificial Intelligence series (LNCS/LNAI), March 2006, p.831-842.

[14] I. Bonev, RoKiSim - Robot Kinematic Simulation, Multi-platform educational software tool for 3D simulation of serial six-axis robots, (accessed October 2017) https://www.parallemic.org/RoKiSim.html

[15] H. Arshad, J. Jamal and S. Sahran, "Teaching Robot Kinematic in a Virtual Environment", *Proc. of the World Congress on Eng. and Comp. Science*, October 2010, Vol.1, p.307-310.

[16] ABB webpage (accessed May 2017), http://www.abb.es/product/en/9AAC100735.aspx

[17] J.M. Sabater-Navarro, N. García, C. Pérez, E. Fernández, R. Saltarén and M. Almonacid, "Kinematics Of A Robotic 3ups1s Spherical Wrist Designed For Laparoscopic Applications", Int J. of Medical Robotics and Computer Assisted Surgery, Vol. 6, No. 3, 2010, pp. 291-300.

[18] C. Jara, José M. Sabater-Navarro, J.M. Azorín, N. García C. Pérez, R. Saltarén and E. Yime, "Analisis Del Espacio De Trabajo De Un Robot Paralelo 3RRR", Sep 2010, *XXIX Jornadas de Automática*, Tarragona (Spain).