ARTE

A ROBOTICS TOOLBOX FOR EDUCATION

PRACTICAL SESSION 5: RAPID PROGRAMMING

Arturo Gil Aparicio

arturo.gil@umh.es

OBJECTIVES

After this practical session, the student should be able to:

- Program a robotic arm using the RAPID language.
- Simulate a program using the ARTE library. This program will then be translated to a RAPID program so that you can program the real robot.

1 Introduction

The ARTE library includes a subset of instructions of the ABB RAPID language. Programming in ARTE will be done in the following way:

i) The `teach` graphical user interface (GUI) allows to simulate the robot and program target points. The user will place the robot in different points in the workspace that will be needed, for example, to pick a piece or place it inside a box.

ii) These points will be defined in a .m file. In order to do this, the points created in the teach aplication can be exported to a .m file. Next, the user should write a program using the equivalent Matlab functions provided, such as `MoveJ()`, `MoveL()` or `MoveAbsJ()`.

iii) The program can be simulated under Matlab. By using its debugging tools you may execute the program step by step ore ven look into the Matlab's functions.

iv) Finally, the program can be translated to a RAPID file. This is done by means of the matlab2RAPID function. The resulting file can be used to program the real robot. There are different ways to transfer the RAPID file to the robot's controller, by using Robotstudio or by means of a basic FTP client.

The whole process will be summarized in Section 3.

The simulation speed of the GUI highly depends on your computer resources. Thus, if you find that the simulation is too slow, you may edit the configuration.delta_time variable. Values near 0.01 s will increase the simulation speed whereas 0.001 will make the simulation too slow in most PCs.

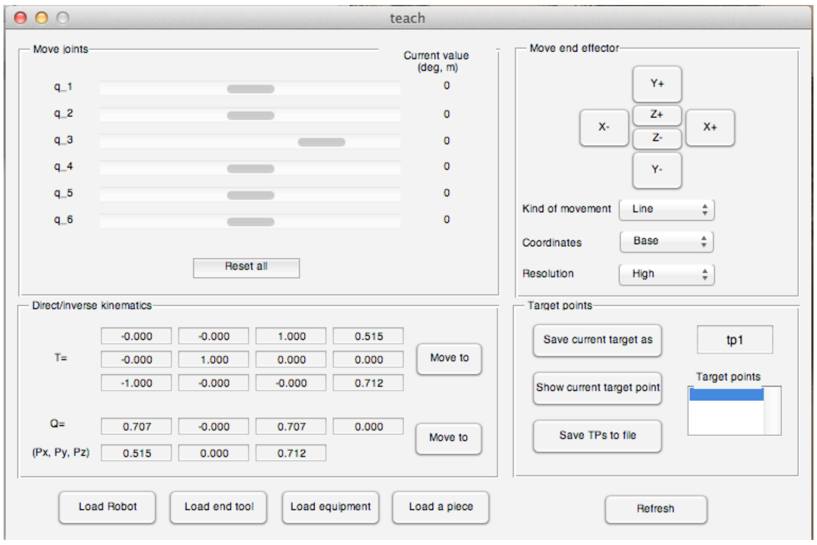
```
>> init_lib
>> configuration.delta_time=0.08
```

2 Using `teach` to program the robot in simulation

This section explains the usage of the teach graphical user interface (GUI). First, initialize the library and load a robot into the Matlab's workspace.

```
>> init_lib
>> robot=load_robot('abb', 'IRB140');
>> teach
```

You can try loading other robots. Robots from other manufacturers different from ABB can also be simulated and programmed using RAPID language. After calling `teach`, the graphical interface shown in Figure 1 appears. In addition, a figure showing a 3D representation of the robot is presented. The `robot` variable is special inside the library. It is declared as a global variable in the workspace so that the `teach` application i sable to access the current loaded robot easily.



	Current value (deg, m)
q_1	0
q_2	0
q_3	0
q_4	0
q_5	0
q_6	0

Kind of movement: Line
Coordinates: Base
Resolution: High

T=	-0.000	-0.000	1.000	0.515
	-0.000	1.000	0.000	0.000
	-1.000	-0.000	-0.000	0.712
Q=	0.707	-0.000	0.707	0.000
(Px, Py, Pz)	0.515	0.000	0.712	

Save current target as: tp1
Target points

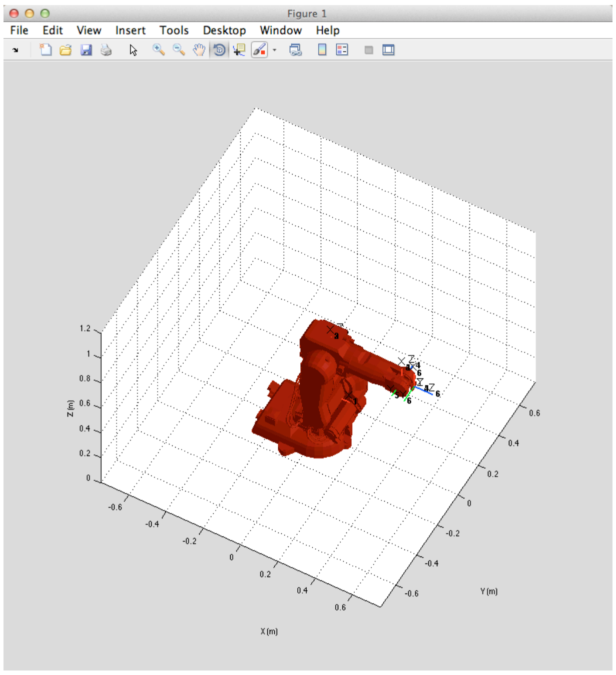


Figure 1

In order to program the robot we will be using the “Target points” menu to store and visualize target points. These target points will be used as:

- **End points:** That is positions and orientations where we wish to place the end effector. E. g. To pick a piece from a conveyor belt and place it in a lathe.
- **Through points:** positions and orientations that will be used to define a trajectory inside the robot’s workspace. E. g. It is usually interesting to define a point A near the piece we would like to grab, thus planning a fast approach to A and a precise approximation to the final point so that the piece is gripped precisely.

a) The **Load Robot** button allows you to load a different robot interactively and program it. Of course, if you plan to program a real IRB 140, this should be your choice.

Add a tool to the end effector. This is achieved by clicking on the **Load end tool button**. Different tools can be loaded from the `arte/robots/equipment/end_tools` directory. Two examples are presented in Figure 2. For example, click on the **Load end tool button** and select the angular gripper `parameters.m` file placed at `arte/robots/equipment/end_tools/angular_gripper/parameters.m`. You may also load the vacuum_1, vacuum_2 or spot_welding tool. It is important to pay attention to the `robot.TCP` variable. This variable is a homogeneous matrix that defines the Tool Center Point (TCP) of the tool. For a gripper, the TCP is the point where the pieces should be grabbed. For a spot welding tool, the TCP defines the position and orientation of the tool tip where the welding takes place. If you wish **to program the IRB 140 robot at the UMH laboratory, please select the parallel gripper 0 tool.**

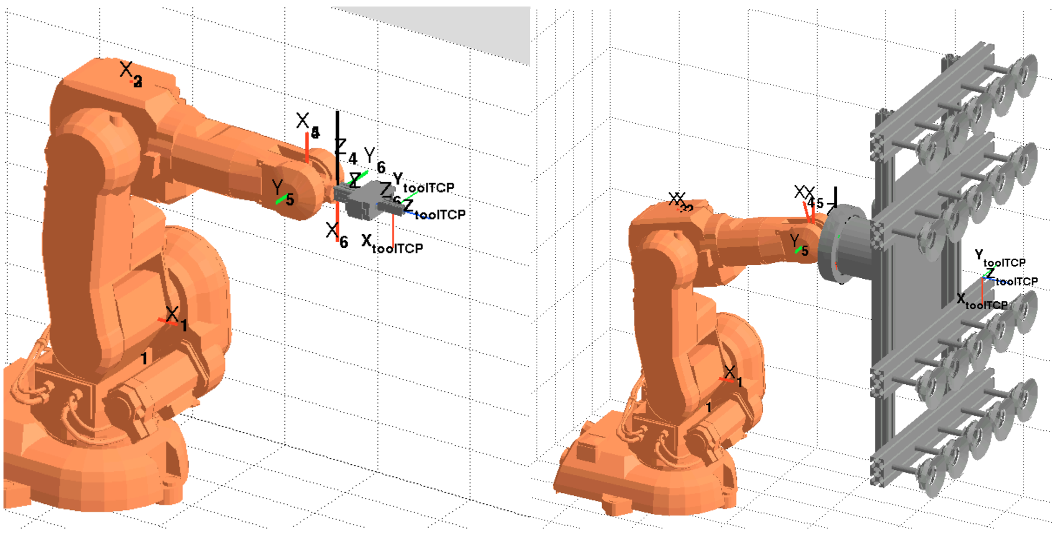
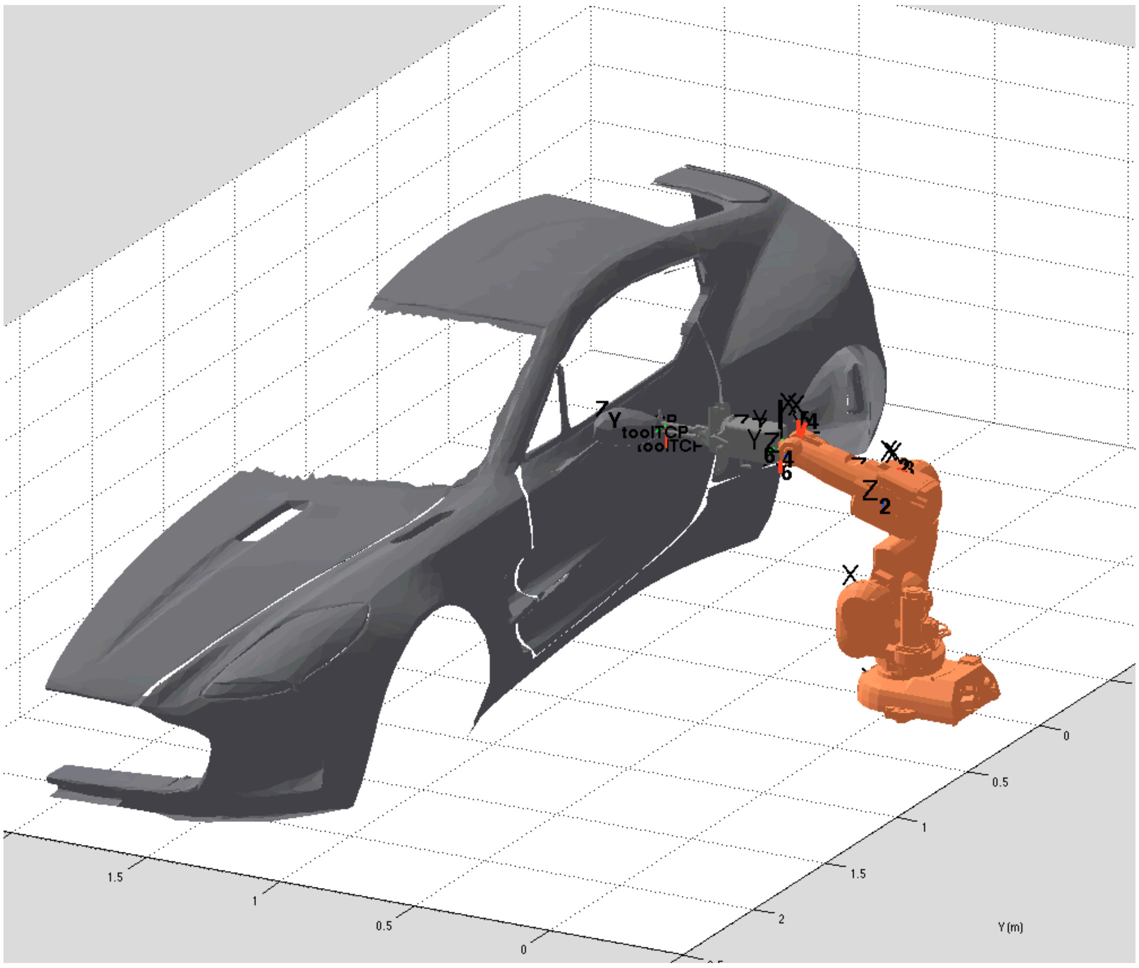


Figure 2

a) Finally, the environment should be represented. Conveyor belts, tables, machines, bowl feeders can be loaded in the same environment. This will surely help in the development of any robotic application. The **Load equipment** button allows to include auxiliary equipment in the robot's environment. The student should pay attention to the placement of the robot with respect to its environment. Figure 3 presents some of the environments included in the library. Load, for example, the bodywork environment located at `arte/robots/equipment/bodywork/parameters.m`. It is worth noting that the enviroments are modelled as a single link robot with no joints, meaning that they are static. In addition, the position and orientation of the robot inside the environment can be modified. To do this, edit the parameters.m file corresponding to the environment and modify the `robot.T0` parameter. The `T0` variable is a homogeneous matrix that defines the position and orientation of the environment with respect to the base of the robot. Alternatively, you can also modify the T0 variable of the robot.



b) The a), b) and c) steps can be programmed in a Matlab script. See the `manufacturing_demo.m` under the `demo` directory.

3 Basic RAPID functions

First, move the robot end effector to a desired placed (e.g. a place to grasp a piece). This can be performed by moving the sliders to move the joints independently or by using the X+, X-, Y+,Y- or Z+, Z- buttons to move the end effector in (X, Y, Z), either in the base or end effector's reference frame.

Once the robot is placed at the desired position and configuration. Save this robot pose as a target point. A target point in RAPID is defined as a `robtarget` data type:

```
CONST robtarget pos_b_rec:=[[-200,-550,200],[0,0.707,0.707,0],[-2,0,-1,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
```

Where [-200,-550,200] is the position of the end effector, [0,0.707,0.707,0] is a quaternion defining the orientation of the end effector and

[-2,0,-1,0] defines the configuration of the arm. Please remember that, in general, there exist different solutions of the inverse kinematic problem that bring the arm to the same position and orientation. The data type confdata =

[cf1, cf4, cf6, cfx] define the quadrants where the joints should be. The quadrants are defined as integer values for positive and negative values as shown in Figure 2. For example, a joint position q = [-90 35 179 200 -50 40] (degrees) is specified univoquely a the configuration [-1 2 0 1].

Please note that cfx is not used at present and is reserved for other kind of robots. Finally [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09] is used to synchronize the position of up to 6 external axis (such as moving platforms to help the robot in its chore). These values means that no synchronization is present.

The target points are stored in the "Program" section, under the "Target point" menú.

Finally, these target points can be selected as arguments for the movement functions. You can build up a movement action by selection one of the available move functions in RAPID:

`MoveJ`: Moves the robot with a isochronous independent joint movement.

`MoveL`: Moves the robot end effector (TCP) with a line in space.

`MoveC`: Moves the robot with a circular trajectory.

`MoveAbsJ`: Moves the robot to an absolute joint position.

`Offs(robtarget, X, Y, Z)`: Program a relative displacement from the specified robtarget destination.

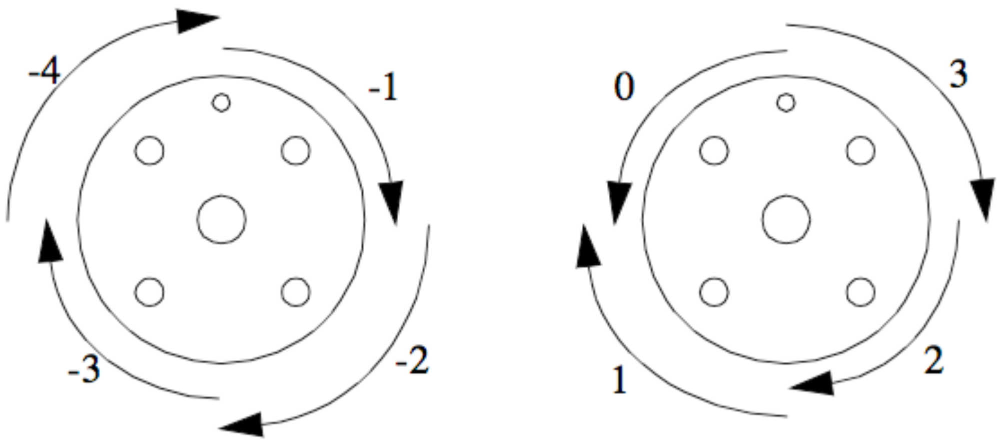
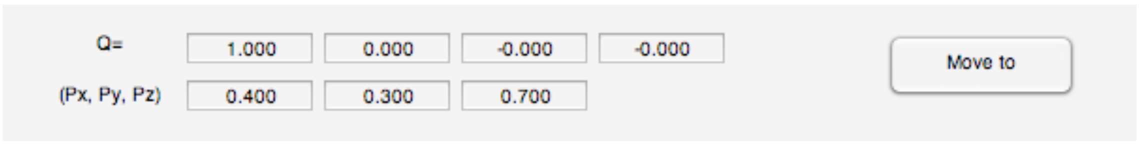


Figure 3

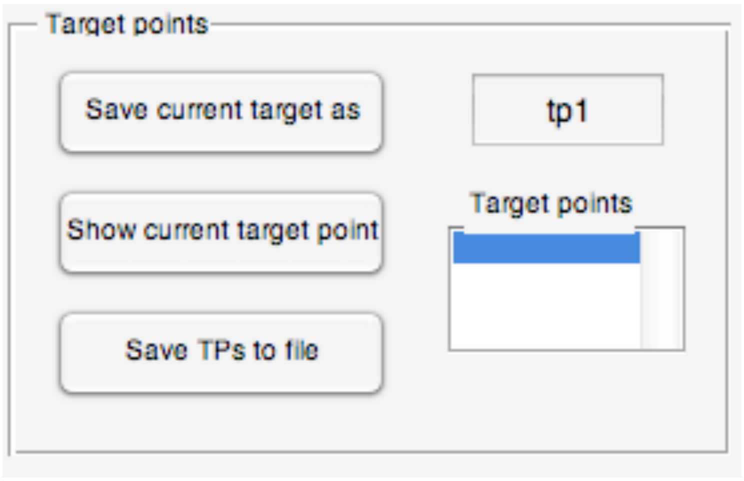
3 Basic programming

We can start by adding two target points. In order to do so, move the robot to the following positions and orientations. Change the fields in Q and P and click on the "Move to" button:

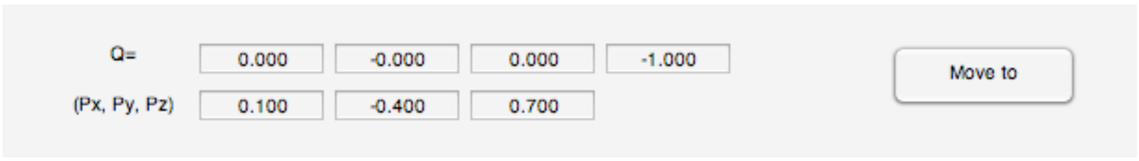

Q= 1.000 0.000 -0.000 -0.000

(Px, Py, Pz) 0.400 0.300 0.700

Press, “Move to”. Next, press “Save current target as” → RT_tp1. The text field can be edited to

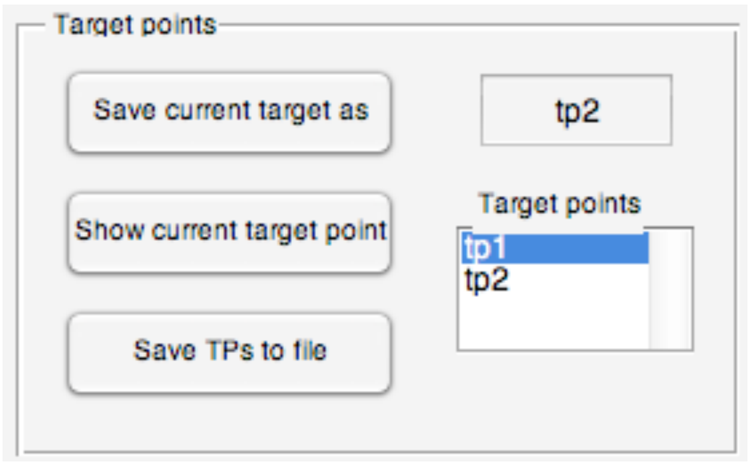


Change the destination point to:


Q= 0.000 -0.000 0.000 -1.000

(Px, Py, Pz) 0.100 -0.400 0.700

Press, “Move to”. Next, press “Save target point” → RT_tp2. You should be able to observe the target points in the “Target points” section.



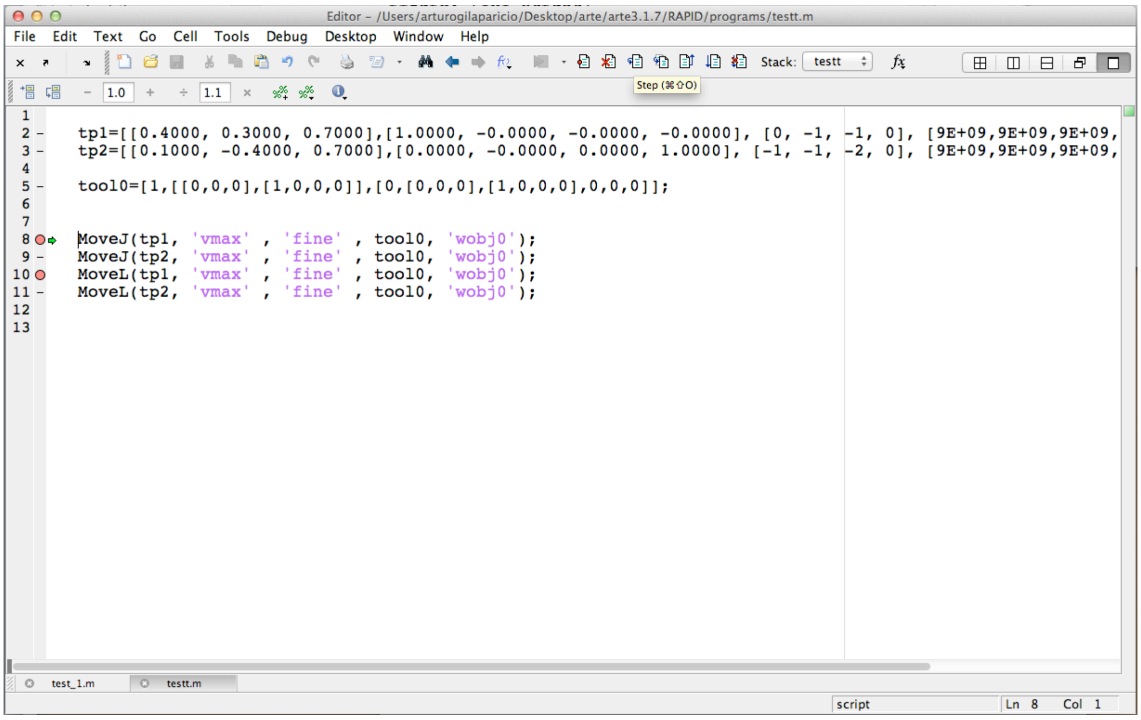
Now, press the “Save TPs to file” button. Select a file and click OK. As a result, the file will contain the following:

```
RT_tp1=[[0.4000, 0.3000, 0.7000],[1.0000, -0.0000, -0.0000, -0.0000], [0, -1, -1, 0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

RT_tp2=[[0.1000, -0.4000, 0.7000],[0.0000, -0.0000, 0.0000, 1.0000], [-1, -1, -2, 0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

Now, we are going to execute a MoveJ instruction. Edit the file to include the following code:

```
RT_tp1=[[0.4000, 0.3000, 0.7000],[1.0000, -0.0000, -0.0000, -0.0000], [0, -1, -1, 0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
RT_tp2=[[0.1000, -0.4000, 0.7000],[0.0000, -0.0000, 0.0000, 1.0000], [-1, -1, -2, 0],
[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

tool0=[1,[[0,0,0],[1,0,0,0]],[0,[0,0,0],[1,0,0,0],0,0,0]];


MoveJ(RT_tp1, 'vmax' , 'fine' , tool0, 'wobj0');
MoveJ(RT_tp2, 'vmax' , 'fine' , tool0, 'wobj0');
MoveL(RT_tp1, 'vmax' , 'fine' , tool0, 'wobj0');
MoveL(RT_tp2, 'vmax' , 'fine' , tool0, 'wobj0');
```

At the top of the file, you should observe the definition of the target points. This definition is equivalent to the RobTarget data type defined in RAPID. Next, the `tool0` definition is included. If no tool has been added to the simulation, the student should use `tool0`, meaning that the end effector of the robot corresponds to the last reference system S_6. Next, the program can be simulated by clicking on ▶. In addition, breakpoints can be added easily by clicking on the line numbers on the left. Click on to execute a program line. Click on to stop debugging. As a result, you will observe the robot moving from one target point tp1 to the other, describing linear (MoveL) or joint coordinate trajectories (MoveJ).


Editor – /Users/arturogilaparicio/Desktop/arte/arte3.1.7/RAPID/programs/testt.m
File Edit Text Go Cell Tools Debug Desktop Window Help
Stack: testt
Step (⌘⇧O)
test_1.m
testt.m
script
Ln 8 Col 1

```

tp1=[[0.4000, 0.3000, 0.7000],[1.0000, -0.0000, -0.0000, -0.0000], [0, -1, -1, 0], [9E+09,9E+09,9E+09,
tp2=[[0.1000, -0.4000, 0.7000],[0.0000, -0.0000, 0.0000, 1.0000], [-1, -1, -2, 0], [9E+09,9E+09,9E+09,

tool0=[1,[[0,0,0],[1,0,0,0]],[0,[0,0,0],[1,0,0,0],0,0,0]];


MoveJ(tp1, 'vmax' , 'fine' , tool0, 'wobj0');
MoveJ(tp2, 'vmax' , 'fine' , tool0, 'wobj0');
MoveL(tp1, 'vmax' , 'fine' , tool0, 'wobj0');
MoveL(tp2, 'vmax' , 'fine' , tool0, 'wobj0');


```

Figure 4

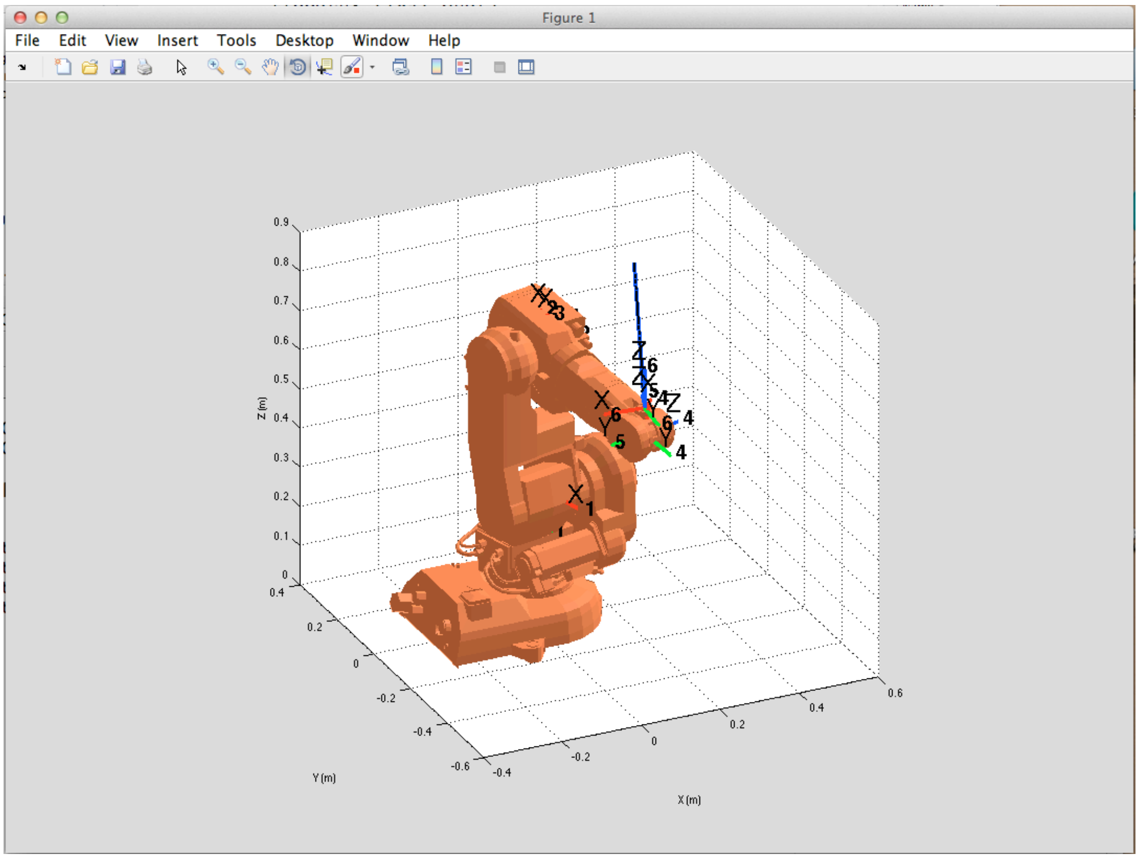


Figure 5

4 Translate it to RAPID

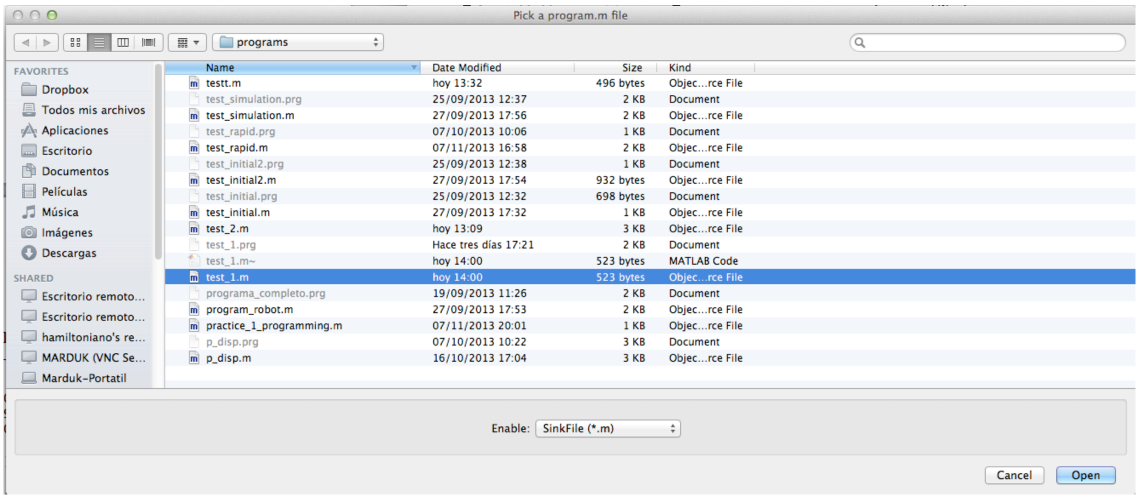
Assume that the following code is stored in a file named `test_1.m`:

```
function test_1
global RT_tp1 RT_tp2 TD_tool0

RT_tp1=[[0.4000, 0.3000, 0.7000],[1.0000, -0.0000, -0.0000, -0.0000], [0, -1,
-1, 0], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
RT_tp2=[[0.1000, -0.4000, 0.7000],[0.0000, -0.0000, 0.0000, 1.0000], [-1, -1,
-2, 0], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

TD_tool0=[1,[[0,0,0],[1,0,0,0]],[0,[0,0,0],[1,0,0,0],0,0,0]];

main;
end

function main()
global RT_tp1 RT_tp2 TD_tool0

      MoveJ(RT_tp1, 'vmax' , 'fine' , TD_tool0, 'wobj0');
      MoveJ(RT_tp2, 'vmax' , 'fine' , TD_tool0, 'wobj0');
      MoveL(RT_tp1, 'vmax' , 'fine' , TD_tool0, 'wobj0');
      MoveL(RT_tp2, 'vmax' , 'fine' , TD_tool0, 'wobj0');
end
```

Please, note that the code should be defined in a Matlab script that starts by a function. Once this is done, type:

```
>> matlab2RAPID
```

The following dialog appears. Select the `test_1.m` file and click on Open.



Pick a program.m file

programs

Name	Date Modified	Size	Kind
testt.m	hoy 13:32	496 bytes	Objec...rce File
test_simulation.prg	25/09/2013 12:37	2 KB	Document
test_simulation.m	27/09/2013 17:56	2 KB	Objec...rce File
test_rapid.prg	07/10/2013 10:06	1 KB	Document
test_rapid.m	07/11/2013 16:58	2 KB	Objec...rce File
test_initial2.prg	25/09/2013 12:38	1 KB	Document
test_initial2.m	27/09/2013 17:54	932 bytes	Objec...rce File
test_initial.prg	25/09/2013 12:32	698 bytes	Document
test_initial.m	27/09/2013 17:32	1 KB	Objec...rce File
test_2.m	hoy 13:09	3 KB	Objec...rce File
test_1.prg	Hace tres días 17:21	2 KB	Document
test_1.m~	hoy 14:00	523 bytes	MATLAB Code
test_1.m	hoy 14:00	523 bytes	Objec...rce File
programa_completo.prg	19/09/2013 11:26	2 KB	Document
program_robot.m	27/09/2013 17:53	2 KB	Objec...rce File
practice_1_programming.m	07/11/2013 20:01	1 KB	Objec...rce File
p_disp.prg	07/10/2013 10:22	3 KB	Document
p_disp.m	16/10/2013 17:04	3 KB	Objec...rce File

Enable: SinkFile (*.m)

As a result, in the command prompt you should observe:

```
>> matlab2RAPID
```

```
Converting Matlab into RAPID lenguaje...

Sink_name =

test_1.prg
```

Next, the `test_1.prg` file is created in the same directory where `test_1.m` exists. If you edit this file, you can observe the following code.

```
MODULE  test_1


CONST robtarget RT_tp1:=[[0.4*1000, 0.3*1000, 0.7*1000],[1, -0, -0, -0], [0, -
1, -1, 0], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
CONST robtarget RT_tp2:=[[0.1*1000, -0.4*1000, 0.7*1000],[0, -0, 0, 1], [-1, -
1, -2, 0], [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];

CONST tooldata
TD_tool0:=[1*1000,[[0*1000,0*1000,0],[1,0,0,0]],[0,[0,0,0],[1,0,0,0],0,0,0]];

 PROC MAIN
      MoveJ RT_tp1,vmax,fine,TD_tool0\WObj:=wobj0;
      MoveJ RT_tp2,vmax,fine,TD_tool0\WObj:=wobj0;
      MoveL RT_tp1,vmax,fine,TD_tool0\WObj:=wobj0;
      MoveL RT_tp2,vmax,fine,TD_tool0\WObj:=wobj0;
 ENDPROC

ENDMODULE
```

This is finally the RAPID code that is equivalent to the Matlab code that we have been using until now. This code is directly portable to program a robot. Just use a FTP client to transfer this file to the robot's controller and program it.

4 Making a more advanced program

The following code includes all the functions that can be used in the library. All these functions can be translated to RAPID equivalent functions. Please, note that only a subset of functions are included in Matlab, but they suffice to generate a vast number of applications. In the following code, we have used the MoveAbsJ function, the MoveC function and the Offs() function. For loops are also included in the code. Edit the file arte/RAPID/programs/test_2.m.

```
function test_2

global RT_tp1 RT_tp2 q0 TD_tool0

TD_tool0=[1,[[0,0,0],[1,0,0,0]],[0,[0,0,0],[1,0,0,0],0,0,0]];

%initial position
q0=[0 0 0 0 0 0]';
%target points
RT_tp1=[[0.3,0.0089,1.0195],[0.2175,0.0971,-
0.0689,0.9688],[1,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
RT_tp2=[[0.7150,-0.2000,0.5120],[0.7071,0.0,0.7071,0.0000],[-1,-
1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];

main;
end
```

```
function main()
global RT_tp1 RT_tp2 q0 TD_tool0

%The MoveAbsJ function performs a joint coordinate planning to the
%specified joint values
 MoveAbsJ(q0, 'vmax' , 'fine' , TD_tool0, 'wobj0');

 MoveJ(RT_tp1, 'vmax' , 'fine' , TD_tool0, 'wobj0');
 MoveJ(RT_tp2, 'vmax' , 'fine' , TD_tool0, 'wobj0');
 MoveL(RT_tp1, 'vmax' , 'fine' , TD_tool0, 'wobj0');
 MoveL(RT_tp2, 'vmax' , 'fine' , TD_tool0, 'wobj0');
 %Return to tp1
 MoveJ(RT_tp1, 'vmax' , 'fine' , TD_tool0, 'wobj0');

 %use the Offs function to move relative to tp1
 % The Offs function makes a relative displacement in X, Y and Z directions
 %Please note that, by using Offs, the specified axes configuration may
 %differ from the one specified in tp1.
 MoveJ(Offs(RT_tp1,0.1,0,-0.1), 'vmax' , 'fine' , TD_tool0, 'wobj0');

 %use MoveL inside a Loop, moving incrementally
 for i=1:4,
     MoveL(Offs(RT_tp1,0.1,i*0.1,-0.2), 'vmax' , 'fine' , TD_tool0, 'wobj0');
 end

end
```

Again, use `matlab2RAPID` to create a RAPID program that can be uploaded to the real controller.

```
!
MODULE  test_2


PERS tooldata
  tool0:=[1*1000,[[0*1000,0*1000,0],[1,0,0,0]],[0,[0,0,0],[1,0,0,0],0,0,0]];

!initial position
q0:=[0 0 0 0 0 0]';
!target points
CONST robtarget
  tp1:=[[0.3*1000,0.0089*1000,1.0195*1000],[0.2175,0.0971,-
0.0689,0.9688],[1,0,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
CONST robtarget
  tp2:=[[0.715*1000,-0.2*1000,0.512*1000],[0.7071,0,0.7071,0],[-1,-
1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];

PROC MAIN

!The MoveAbsJ function performs a joint coordinate planning to the
!specified joint values
  MoveAbsJ q0, 'vmax' , 'fine' , tool0, 'wobj0');
  MoveJ tp1,vmax,fine,tool0\WObj:=wobj0;
  MoveJ tp2,vmax,fine,tool0\WObj:=wobj0;
  MoveL tp1,vmax,fine,tool0\WObj:=wobj0;
  MoveL tp2,vmax,fine,tool0\WObj:=wobj0;
!Return to tp1
  MoveJ tp1,vmax_,fine_,tool0\WObj:=wobj0;

!use the Offs function to move relative to tp1
! The Offs function makes a relative displacement in X, Y and Z directions
!Please note that, by using Offs, the specified axes configuration may
!differ from the one specified in tp1.
  MoveJ Offs(tp1,0.1*1000,0.0*1000,0.1*1000),vmax,fine,tool0\WObj:=wobj0;

!use MoveL inside a Loop, moving incrementally
  FOR i FROM 1 TO 4 DO
     MoveL Offs(tp1,0.1*1000,i*0.1*1000,0.1*1000),0.2,vmax,fine\WObj:=wobj0;
 ENDFOR
```

```
ENDPROC

ENDMODULE
```

Exercise 1:

a) Open the file test_2.m. Simulate it step by step.

b) Edit the target points to different 3D positions. What kind of WARNINGS or ERRORS do you obtain. Why?

5 Simulating a packaging application

Other aspects are important when developing a robotic application.

a) You should have an **environment** that ressembles the real one.

b) The robot should have a **suitable end tool**, similar to the one that uses the real robot that you aim to program.

c) There should exist **a piece** that is being moved. For example, in a packaging application, the pieces will be picked, from a conveyor belt and placed in a box.

The three items above are covered easily inside the `teach` GUI application.

a) Click on the **Load Robot button** to swap easily between the robots in the library.

b) Click on **Load end tool** to load an **end effector**.

c) Click on Load equipment to load auxiliar systems such as conveyor velts, machining tools, robotic cells, etc.

d) Finally, load a piece that will be grabbed by the robot by clicking on the "**Load a piece**" button.

e) Moreover, there exist functions that are used during the simulation to provide the feeling that the piece is grabbed by the robot and moves with the end tool.

The piece position/orientation is defined by the `robot.piece.T0` homogeneous transformation and should be defined during the simulation.

We are now prepared to simulate the first packaging application.

a) Open the script under `arte/RAPID/programs/practice_1_programming.m`.

b) Run `teach`, **load an IRB 140 robot**.

c) Click on **Load end Tool** and load the `arte/robots/equipment/end_tools/parallel_gripper_0` that defines a parallel gripper.

d) Click on **Load equipment** and load `arte/robots/equipment/tables/table_two_areas` that represents a conveyor belt with two workin areas.

e) Click on **Load a piece** and load `arte/robots/equipment/cylinders/cilynder_tiny` that represents a piece.

Finally, you can simulate the `practice1_programming.m` script in Matlab using the ▶ button. The script simulates that the robot picks a picece and then drops it in the packaging area.

Exercise 2:

a) Open the file practice_1_programming.m and simulate it step by step.The initial position of the piece is at the end of a conveyor belt. The position and orientation of the piece is defined by:

```
robot.piece.T0(1:3,4)=[-0.1 -0.5 0.2]';
```

The box has dimensions 0.3x0.3x0.2 m and its center is located at (X, Y, Z)=(0.5, -0.5, 0.2)

b) Now you should edit the file by including more targetpoints and define functions to simulate a more realistic packaging application. First, consider that the four pieces have to be placed as shown in Figure 5.

c) Repeat the program to produce a different packaging. In this case, the pieces should be placed in a box as represented in Figure 6. The box

An example is presented in the following video:

https://drive.google.com/file/d/0Bx_eBHgZLv8Kc1hUd190OEV3c28/edit?usp=sharing

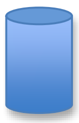


Figure 5

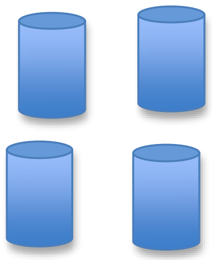


Figure 6

7 Program the real robot

Programming a real robot can be done in different ways. Often, the robot manufacturers provide a GUI to simulate and program the controller. In the case of the SC4+ controller we will be using a simple FTP client to transfer the RAPID file to the controller.

a) Call matlab2RAPID and select the .m Matlab script.

```
>> matlab2RAPID
```

b) if the file arte/RAPID/programs/practice_1_programming.m is selected, a new file arte/RAPID/programs/practice_1_programming.prg will be generated. This last file is in RAPID language.

c) Edit this last file, some of the functions used in simulation will not be understood by RAPID. Edit `practice_1_programming.prg` and remove any call to:

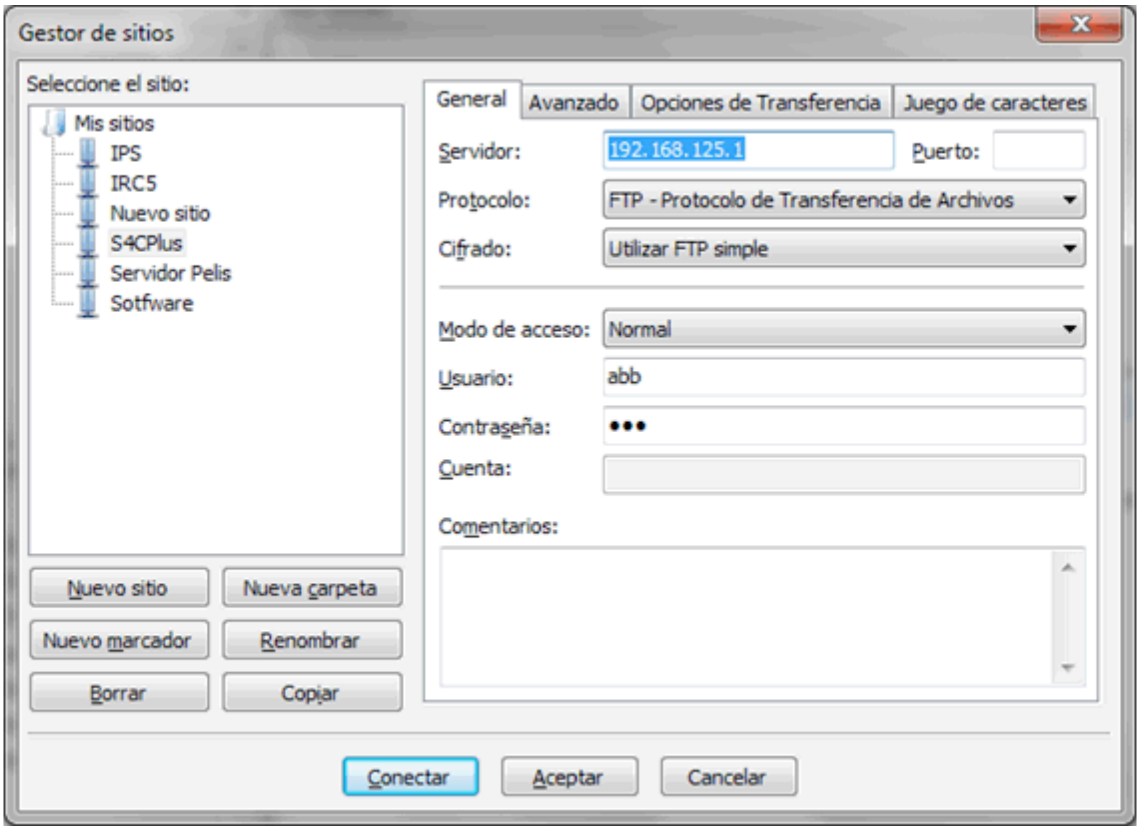
```
robot.piece.T0(1:3*1000,4):=[0 -0.45 0.2]';
!robot.tool.piece_gripped=0;
```

```
drawrobot3d(robot, robot.q);

!close the tool
simulation_close_tool;
simulation_open_tool;
simulation_open_piece;
simulation_release_piece;
```

d) In order to program the SC4+ robot controller, the following data should be useful. Configure the communications with the controller with the following network configuration in the PC:

IP ADDRESS: 192.168.125.82
MASK: 255.255.255.240
GATEWAY:192.168.125.81

Use a simple ftp client such as Filezilla to connect to the Server with address: 192.168.125.1.



Gestor de sitios

Seleccione el sitio:

- Mis sitios
  - IPS
  - IRC5
  - Nuevo sitio
  - S4CPlus
  - Servidor Pelis
  - Sotfware

General | Avanzado | Opciones de Transferencia | Juego de caracteres

Servidor: 192.168.125.1
Puerto:
Protocolo: FTP - Protocolo de Transferencia de Archivos
Cifrado: Utilizar FTP simple
Modo de acceso: Normal
Usuario: abb
Contraseña: •••
Cuenta:
Comentarios:

Exercise 3:

a) Simulate a packaging application for the real robot presented in Figure 7.

b) The positions and orientations of the end tool to grip a piece or release it should be known in order to develop your application. A simple way of doing this, (if you have the possibility to handle the real robot) is to use the robot as a measurement unit: place the robot at the desired position and orientation and store this data. Proceed to the release area and repeat the process.

c) Simulate the packaging application. You can use the test_functions.m and the test_packaging.m scripts as examples.

d) Once the student considers that the problems are solved and the target points are correct, the real robot may be programmed. Thus, use matlab2RAPID to obtain a RAPID program. Edit it to assure that the syntax is correct. Finally, use Filezilla to transfer the program to the robot.

8 Program a different robot

Prior to the call to the “teach” GUI you can load any robot and program it. Please note that other robots may have different restrictions, such as the size of their workspace and the different orientations that can be reached.

```
>> init_lib
>> robot=load_robot('abb', 'IRB6620');
>> teach
```

Or, for example, load a KUKA robot and program it in RAPID:

```
>> init_lib
>> robot=load_robot('kuka', ' KR5_arc');
>> teach
```