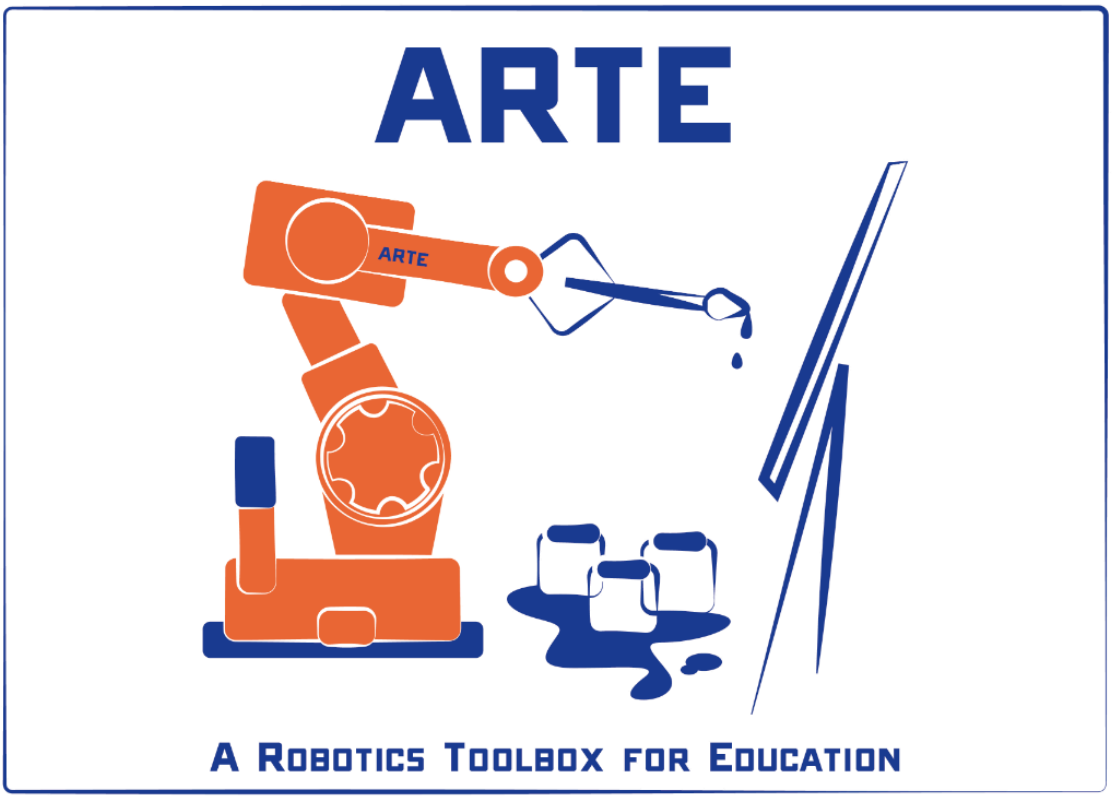

PRACTICAL SESSION 2: INVERSE KINEMATICS

Arturo Gil Aparicio

arturo.gil@umh.es

OBJECTIVES

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

- Solving the inverse kinematic problema of a serial manipulator using geometric methods.
- Implementing this code in Matlab for a chosen robotic arm.
- Generating joint trajectories so that the end effector follows a linear trajectory in the cartesian space.
- Analyzing singular points by means of the Jacobian matrix.

Index

1. **First steps**
2. **The inverse kinematic problem**
3. **Solving the inverse kinematic problem for a 6 DOF robot**
4. **Adding your robot to the library**
5. **Direct and inverse Jacobian**
6. **Follow a line in space**

1 First steps

We first start with a kinematic analysis of the SCARA robot arm represented in Figure 1. The table with the Denavit-Hartenberg parameters is specified next:

	Theta	**D**	**A**	**Alfa**
1	q_1	l_1=0.5m	l_2=0.5m	0
2	q_2	0	l_3=0.3m	π
3	0	q_3	0	0
4	q_4	0	0	0

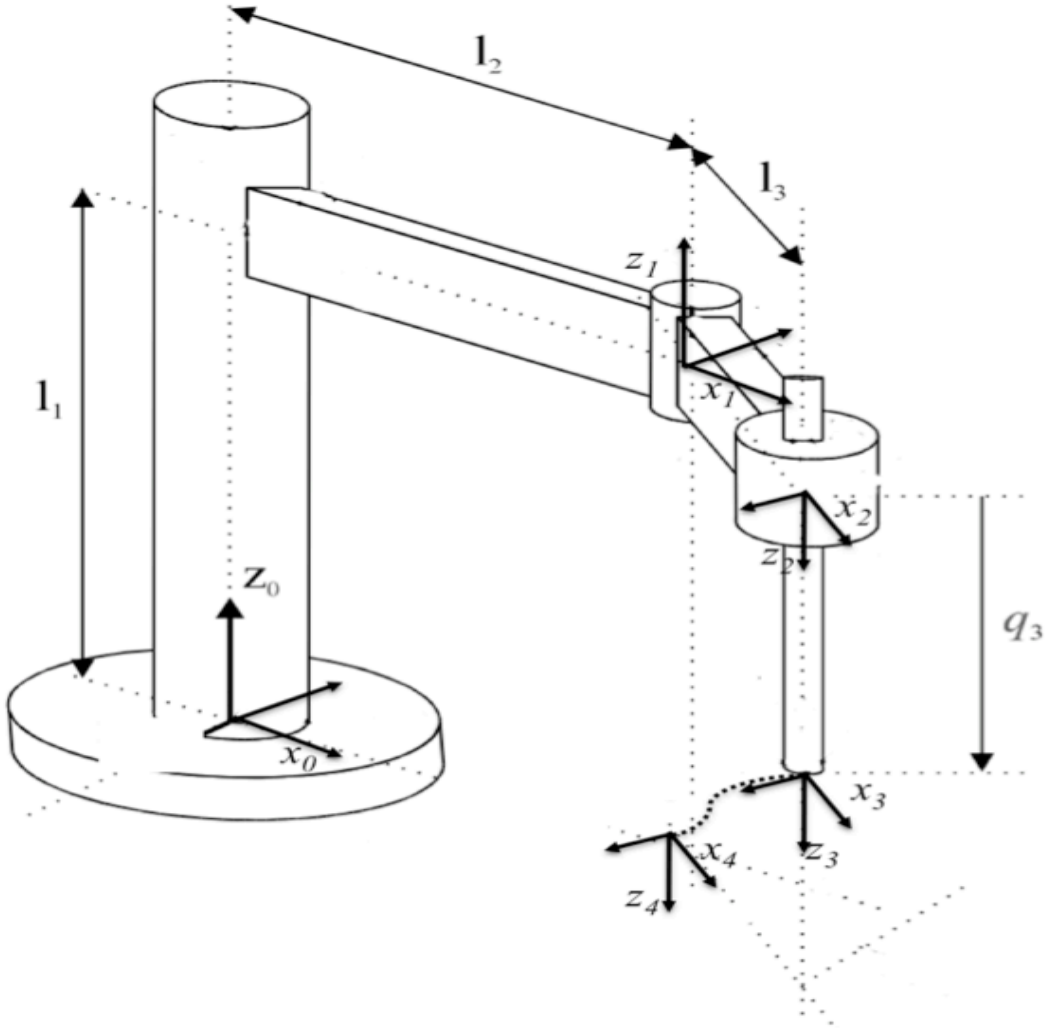

Figure 1

To begin with, we are going to compute the direct position and orientation of end effector as a function of the joint coordinates. You can type the following commands at the Matlab prompt:

```
>> init_lib
>> robot=load_robot('example','scara')
>> T=directkinematic(robot, [0 0 0.1 0])

T =

    1.0000         0         0    0.8000
         0   -1.0000   -0.0000   -0.0000
         0    0.0000   -1.0000    0.4000
         0         0         0    1.0000
>> drawrobot3d(robot, [0 0 0.1 pi/4])
```

Figure 2 presents the result of the `drawrobot3d` function.

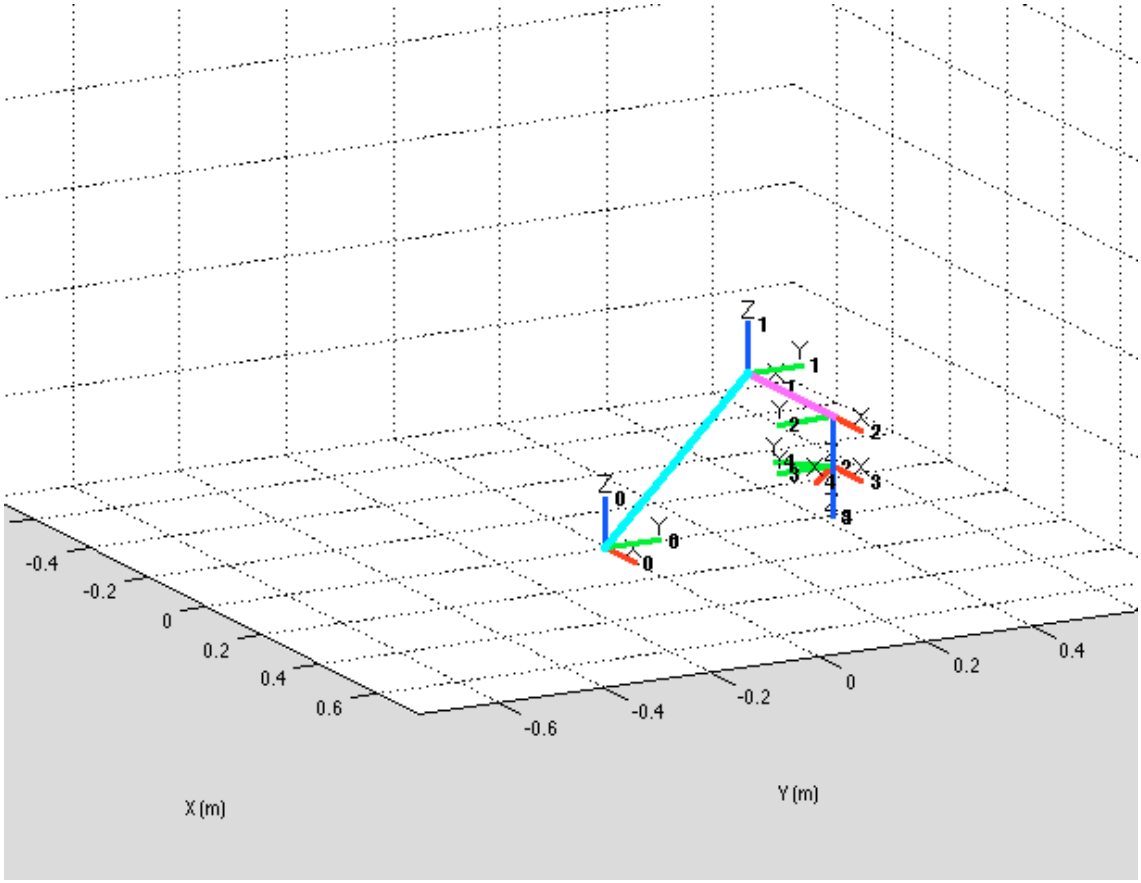

Figure 2

2 The inverse kinematic problem

Given a position and orientation of the end effector, we would like to compute the joint coordinates that bring the robot's end effector to it. In the case of the example SCARA we can achieve it with the following commands. Generally, there exist more than one solution to achieve it.

```
>> pwd

ans =
/Users/arturogilaparicio/Desktop/arte3.1.6

>> init_lib
>> T=directkinematic(robot, [0 0 0 0])
>> qinv = inversekinematic(robot, T)
Computing inverse kinematics for the Scara example 4GDL arm robot
qinv =

     0     0
     0     0
     0     0
     0     0
>> T=directkinematic(robot, [pi/2 pi/4 0.1 pi/8])
```

```
>> qinv = inversekinematic(robot, T)
Computing inverse kinematics for the Scara example 4GDL arm robot
qinv =

    2.1498    1.5708
   -0.7854    0.7854
    0.1000    0.1000
   -0.5991    0.3927
```

Note that, after initializing the library, we compute the direct kinematics for the robot's initial position `q=[0 0 0 0]`. A call to the function `qinv = inversekinematic(robot, T)` takes two arguments. The first stores the robot parameters, whereas the second, T, specifies the position and orientation of the end effector. The function `qinv = inversekinematic(robot, T)` is common for all the robots in the library, however, the function calls internally a different function, named `inversekinematic_scara` that is placed at the robot's home directory `arte3.1.6/robots/example/scara`. The name of the inverse kinematic function is specified in the parameters.m file, in the line `robot.inversekinematic_fn = 'inversekinematic_scara(robot, T)'.`

```
function robot = parameters()

%Path where everything is stored for this robot
robot.path = 'robots/example/scara';

%Kinematic parameters
robot.DH.theta= '[q(1) q(2) 0 q(4)]';
robot.DH.d='[0.5  0      q(3)  0]';
robot.DH.a='[0.5    0.3  0    0]';
robot.DH.alpha= '[ 0 pi  0 0]';

%Jacobian matrix. Variation of (X, Y, Z) as a function of (w1, w2, w3)
robot.J='[-a(2)*sin(q(1)+q(2))-a(1)*sin(q(1)) -a(2)*sin(q(1)+q(2)) 0;
a(2)*cos(q(1)+q(2))+a(1)*cos(q(1)) a(2)*cos(q(1)+q(2)) 0; 0 0 -1];';

robot.name='Scara example 4GDL arm';

robot.inversekinematic_fn = 'inversekinematic_scara(robot, T)';
```

It is important to note that the inversekinematic function returns two posible solutions for `qinv`, only one of them matches the initial q specified in the direct kinematic function. However, both solutions for qinv match the original matrix T. You can test these ideas with:

```
>> T=directkinematic(robot, [pi/2 pi/4 0.1 pi/8])

T =
   -0.3827    0.9239    0.0000   -0.2121
    0.9239    0.3827   -0.0000    0.7121
   -0.0000    0.0000   -1.0000    0.4000
         0         0         0    1.0000
>>qinv = inversekinematic(robot, T)

>>T=directkinematic(robot, qinv(:,1))
```

```
T =
   -0.3827    0.9239    0.0000   -0.2121
    0.9239    0.3827   -0.0000    0.7121
   -0.0000    0.0000   -1.0000    0.4000
         0         0         0    1.0000

>> T=directkinematic(robot, qinv(:,2))

T =
   -0.3827    0.9239    0.0000   -0.2121
    0.9239    0.3827    0.0000    0.7121
    0.0000    0.0000   -1.0000    0.4000
         0         0         0    1.0000
```

You can observe this by calling `drawrobot3d` for both solutions to the inverse kinematics.

```
>> drawrobot3d(robot, qinv(:,1))
>> drawrobot3d(robot, qinv(:,2))
```

3 Solving the inverse kinematic problem for a 6 DOF robot

Next, Figure 3 presents the ABB IRB 140 robot. Next, its corresponding D-H table is presented.

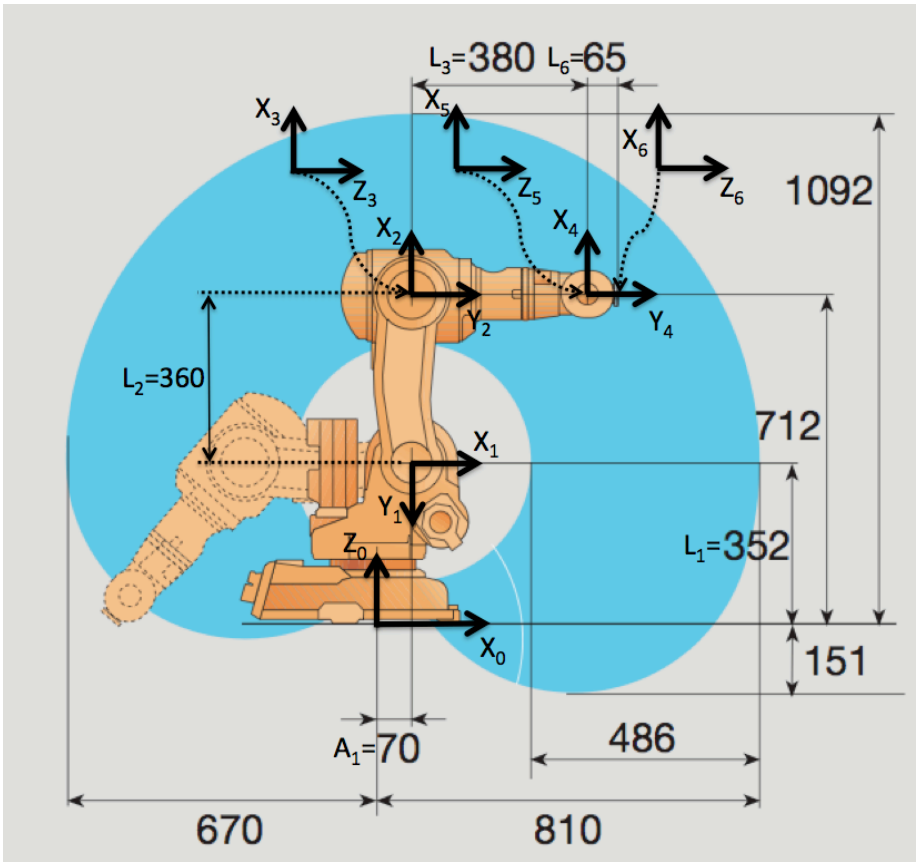

Figure 3

θ	d	a	α
q_1	0.352	0.070	$-\pi/2$
$q_2 - \pi/2$	0	0.360	0
q_3	0	0	$-\pi/2$
q_4	0.380	0	$\pi/2$
q_5	0	0	$-\pi/2$
q_6	0.065	0	0

You can observe these parameters in the `parameters.m` file, which resides in `arte3.1.6/robots/abb/irb140`

```
robot.DH.theta= '[q(1) q(2)-pi/2 q(3) q(4) q(5) q(6)]';
robot.DH.d='[0.352 0 0 0.380 0 0.065]';
robot.DH.a='[0.070 0.360 0 0 0 0]';
robot.DH.alpha= '[-pi/2 0 -pi/2 pi/2 -pi/2 0]';
robot.J=[];

robot.inversekinematic_fn = 'inversekinematic_irb140(robot, T)';
```

The inverse kinematic problem is solved by the function `inversekinematic_irb140(robot, T)` that is placed at `arte3.1.6/robots/abb/irb140`. You can find some more information about solving this inverse kinematic problem in the ARTE reference manual.

Exercise 1:

You should analyse the inverse kinematic problem for the IRB 140 robot. In this case, in general there exist 8 different solutions to achieve a position/orientation of the end effector in space. You should analyse that any of the eight different solutions q returned by the function allows to achieve the same position/orientation in space (the same homogeneous matrix T).

4 Direct and inverse jacobian

The functions Jacobian matrix associated with the position of the end effector allows to:

- Compute the speed of the end effector given the joint speeds.

- Compute the joint speeds, given a desired speed of the end effector in coordinates of the base reference frame.

Please try the functions `compute_end_velocity` and `compute_joint_velocity` and observe the results.

Exercise 2:

In the case of the SCARA example arm, the expresión of the Jacobian matrix is provided as a function of the joint variables.

a) Compute the end effector's speed when the joint speeds are ω_q = (0.1 rad/s, 0.1 rad/s, 1 m/s)=(ω_{q1}, ω_{q2}, ω_{q3}). Note that the third joint is translational and that the joint speed ωq_4 does not contribute to the end effector's speed.

b) Compute the joint speeds necessary to obtain an end effector's speed of V=(1, 1, 1) m/s when the end effector is at P=(0.5, 0.5, 0) m. Please note that there are two possible solutions.

5 Follow a line in space

Exercise 3:

Implement a function:

function q=followline_myrobot(robot, p1, p2, R, options)

To follow a line in cartesian space. The function should interpolate a number of points along the line that connects the points p1 and p2. Next, for each of these points, you should find a solutions of the inverse kinematic. The parameter options should be used to choose one of the posible solutions. The orientation of the end effector should be constant and equal to R along the whole movement. You should:
a) Animate the trajectory of the robotm using the function drawrobot3d.
b) Represent the joint trajectories.
c) Plot the joint speeds, asuming that the movement between p1 and p2 is accomplished in 1 second.
d) Enumerate the possible errors that may appear during this movement:
- Points outside the workspace of the robot.
- Singular points where det(J)=0 that indicate the need of infinite joint speeds.
- Joints out of range.

6 Use the teach pendant

Execute the graphical teach pendant (Figure 4). The teach pendant GUI application allows to move the robot 's end effector in the following ways:

- With an independent movement of each robot axis. Using the sliders at the top left área.
- Moving the end effector along along X, Y and Z in coordinates of:
    - The base reference system.
    - The end effector.
- Change the values of the T matrix and solve the inverse kinematic problem. The application calls internally the inversekinematic function. The application shows the joint configuration closest to the current one. Alternatively you can change the position and orientation in the Q controls.

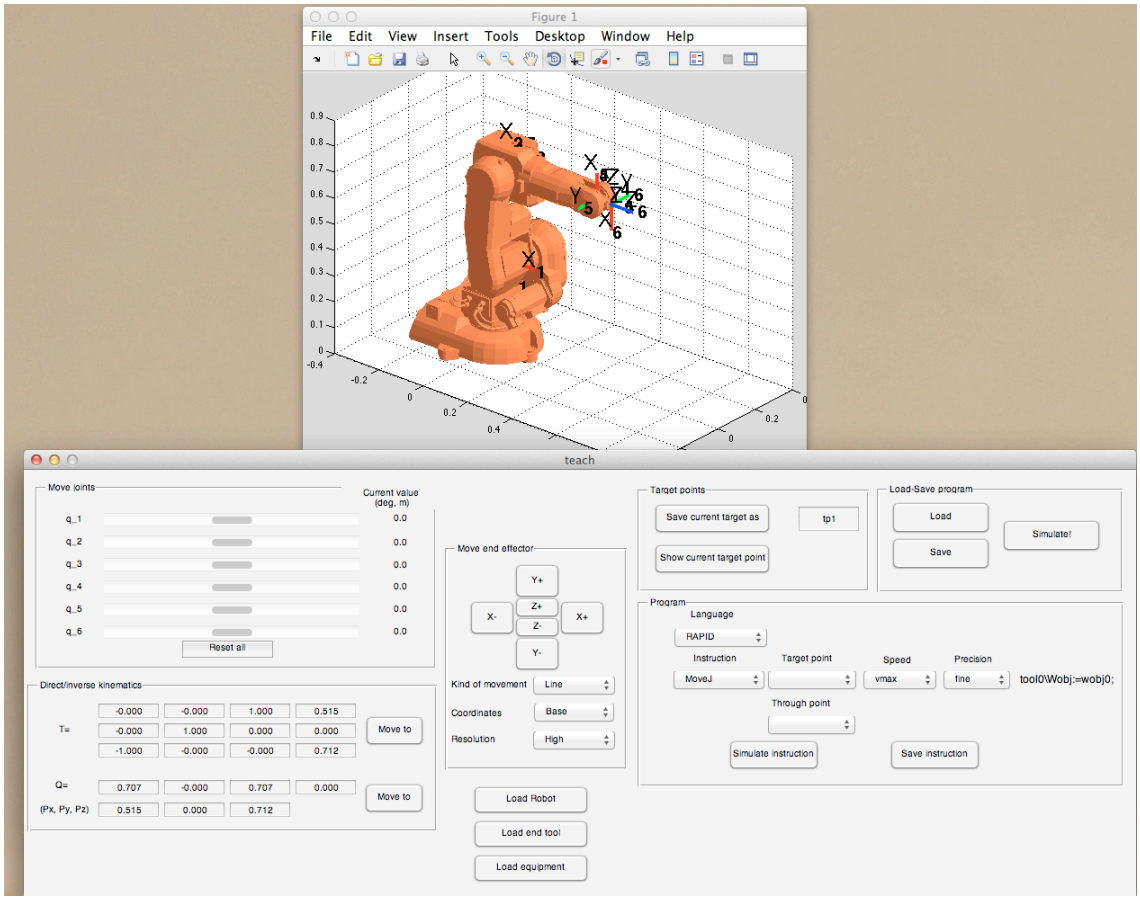

Figure 4

7 Summary

Everything is summarized in the following videos:

- Inverse kinematic of a 6 DOF robot.

http://arvc.umh.es/arte/videos/pr2_video1_directa_inversa.mp4

http://www.youtube.com/watch?v=CS2eGRSEGOo (spanish)

- Execute the teaching pendant application (teach)

http://www.youtube.com/watch?v=h1xnJw3-a7M#t=18m23s (spanish)

http://arvc.umh.es/arte/videos/pr2_video2_teachd.mp4