Homework 1: inverse kinematics

Vitetta Emanuele;
2082149

September 17, 2023

1 Analytical solution of kinematic inverse problem

To solve the inverse kinematics problem, we need to find the joint angles θ_1, θ_2 and θ_3 that result in the desired end-effector position x_e^d. Consider the following system of three equations

$$\cos(\theta_1) + \cos(\theta_1 + \theta_2) + \cos(\theta_1 + \theta_2 + \theta_3) = 2 \tag{1}$$

$$\sin(\theta_1) + \sin(\theta_1 + \theta_2) + \sin(\theta_1 + \theta_2 + \theta_3) = 1 \tag{2}$$

$$\theta_1 + \theta_2 + \theta_3 = 0 \tag{3}$$

Considering that $\theta_1 + \theta_2 + \theta_3 = 0$, it follows that $\cos(\theta_1 + \theta_2 + \theta_3) = 1$ and $\sin(\theta_1 + \theta_2 + \theta_3) = 0$. By substituting these values in the first two rows we get the following system to be solved

$$\cos(\theta_1) + \cos(\theta_1 + \theta_2) = 1 \tag{4}$$

$$\sin(\theta_1) + \sin(\theta_1 + \theta_2) = 1 \tag{5}$$

If we square both (4),(5) and we sum them term to term we get

$$\cos^2(\theta_1) + \sin^2(\theta_1) + \cos^2(\theta_1 + \theta_2) + \sin^2(\theta_1 + \theta_2) + 2\cos(\theta_1)\cos(\theta_1 + \theta_2) + 2\sin(\theta_1)\sin(\theta_1 + \theta_2) = 2$$

$$2 + 2\left[\cos(\theta_1)\cos(\theta_1 + \theta_2) + \sin(\theta_1)\sin(\theta_1 + \theta_2)\right] = 2$$

$$2\cos(\theta_2) = 0 \tag{6}$$

Hence we get that $\theta_2 = \frac{\pi}{2}$ or $\theta_2 = -\frac{\pi}{2}$, from which derives multiple solutions. Let's now consider eq. 4

$$\cos(\theta_1) + \cos(\theta_1 + \theta_2) = \cos(\theta_1) + \cos(\theta_1)\cos(\theta_2) - \sin(\theta_1)\sin(\theta_2) = 1$$

$$\cos(\theta_1) - \sin(\theta_1)\sin(\theta_2) = 1$$

If we substitute the two values of θ_2 we have just found, we can derive two different solutions for the joint angle's values. Such values are reported in the table below.

	θ_1	θ_2	θ_3
Solution 1	0	$\frac{\pi}{2}$	$-\frac{\pi}{2}$
Solution 2	$\frac{\pi}{2}$	$-\frac{\pi}{2}$	0

Table 1: Numerical solutions

In Fig. 1 we can appreciate a schematic representation of the planar robot and the configuration of the two different solutions.

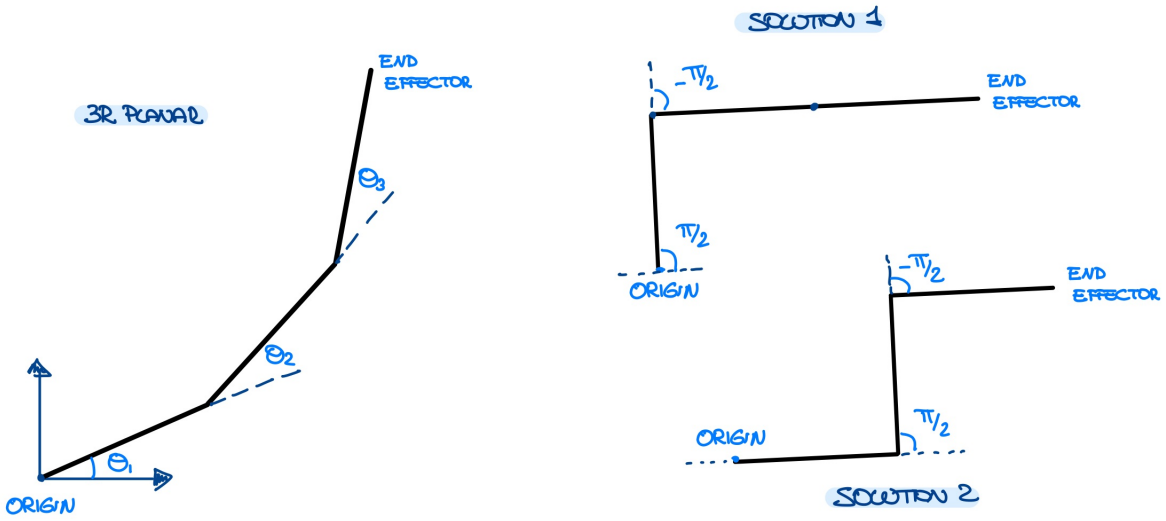

Figure 1: Planar 3R robot and configurations for desired x_e^d

2 Numerical solutions

2.1 General considerations

To solve the inverse kinematic problem we test two different numerical approaches: gradient method and Newton's method. For both of them, we analyze the convergence of the algorithm starting from two different initial conditions, e.g. $q(0) = [\ 0 \quad 0 \quad 0\ ]^T$ and $q(0) = [\ \pi/2 \quad \pi/2 \quad \pi/2\ ]^T$. In listing 1 we report the Matlab code which is common to both methods. To simplify calculations we exploited *Symbolic Math Toolbox*, in order to easily evaluate Jacobians, their transposes and inverses.

In both methods, we set up two possible stopping conditions. The first one is given by the maximum number of iterations we want to perform. The second one is based on the improvement among two consecutive steps. In particular, if the error between step j and $j+1$ is smaller than ϵ we terminate our descent (WRITE BETTER). These parameters can be set autonomously for each algorithm.

Listing 1: Initialization

```
% Clear all the parameters from the workspace and also clear the command window.
clear all
clc
% Define initial conditions
q0 = [[0 0 0]',[pi/2 pi/2 pi/2]'];
% Define the step-sizes we are going to test.
alpha = [1/2 1/10];
% Desired position of the end effector
x_de = [2 1 0]';
% Matrices useful for the implementation of the algorithm
syms x y z
K = [cos(x)+cos(x+y)+cos(x+y+z),sin(x)+sin(x+y)+sin(x+y+z),x+y+z]';
J = jacobian([cos(x)+cos(x+y)+cos(x+y+z),sin(x)+sin(x+y)+sin(x+y+z),x+y+z],[x,y,z]);
```

2.2 Gradient method

To implement this method we exploit the known relation

$$q^{k+1} = q^k + \alpha J^T(q^k)(x_e^d - q^k)$$

where α is the step size. We test the algorithm starting from both starting points and also considering two different step-sizes, namely $\alpha = 1/2$ and $\alpha = 1/10$. For all simulations, we set an error threshold $\epsilon = 10^{-6}$ and the maximum number of iterations is $M = 2000$.

Let's start by analyzing the results coming from the simulations with $\alpha = 1/2$, which are reported in fig. 2 and in table 2. As we can observe after a few iterations the algorithm starts oscillating in a fixed area (for both $q(0)$'s). This happens because the step size we are considering is too big. We can notice that that algorithm terminates its execution because it reaches the maximum number of iterations. Furthermore, the solutions found are not acceptable. It is also interesting to observe that the error does not converge, but keeps oscillating.

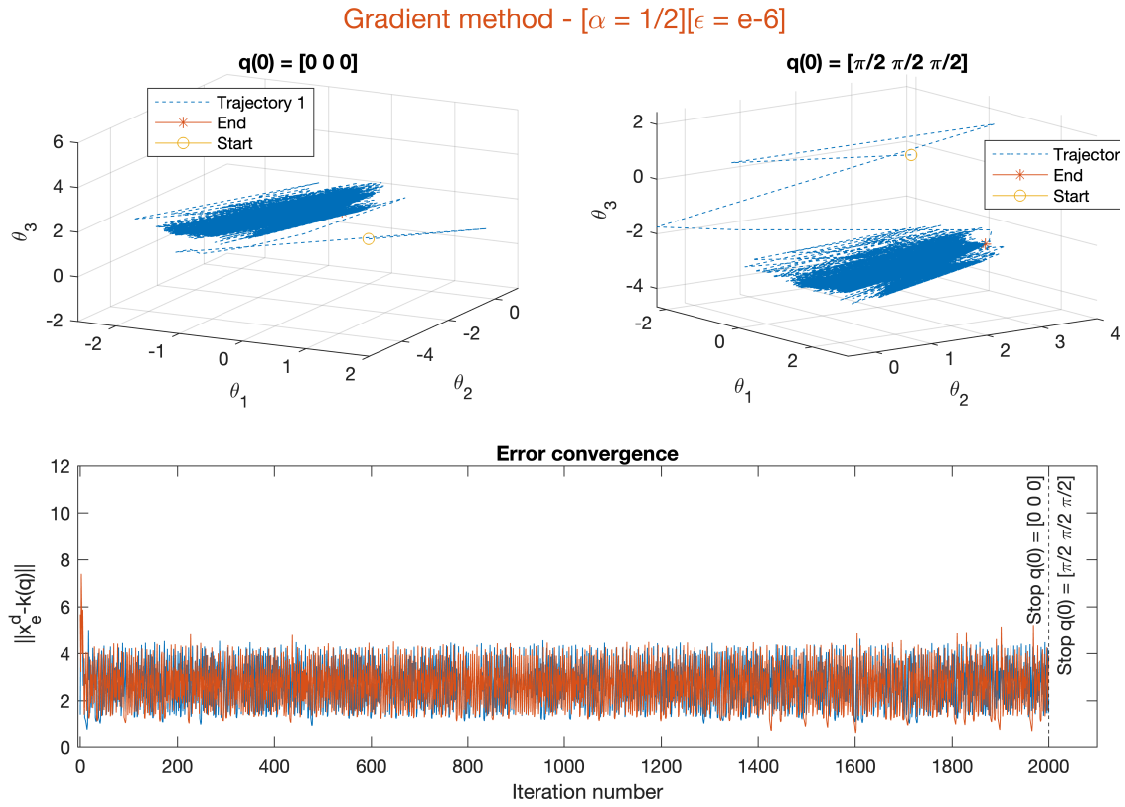

Figure 2: Gradient method simulations for $\alpha = 1/2$

$q(0)$	$\theta_{1,f}$	$\theta_{2,f}$	$\theta_{3,f}$	$\lvert x_e^d - \kappa(\theta_f)\rvert$	Final iter. f.
$[0\ 0\ 0]^T$	0.27π	-1.09π	π	2.06	2000
$\left[\frac{\pi}{2}\ \frac{\pi}{2}\ \frac{\pi}{2}\right]^T$	0.40π	0.99π	-0.72π	3.91	2000

Table 2: Gradient method final solutions for $\alpha = 1/2$

Let's now analyze the case with $\alpha = 1/10$. Simulation results are reported fig. 3 and table 3. In this case, we can observe that the algorithm converges in a number of iterations which is smaller than the maximum one. Furthermore, both numerical solutions found are acceptable and they represent a good approximation of the analytical value. In fact, the distance between the desired position of the end-effector x_e^d and the one found by computing $\kappa(\theta_f)$ is really small.

It is also possible to notice that, despite starting from two different initial conditions, gradient method founds the same numerical solution in both cases. Hence, using this method, we can only discover solution 1 reported in table 1.

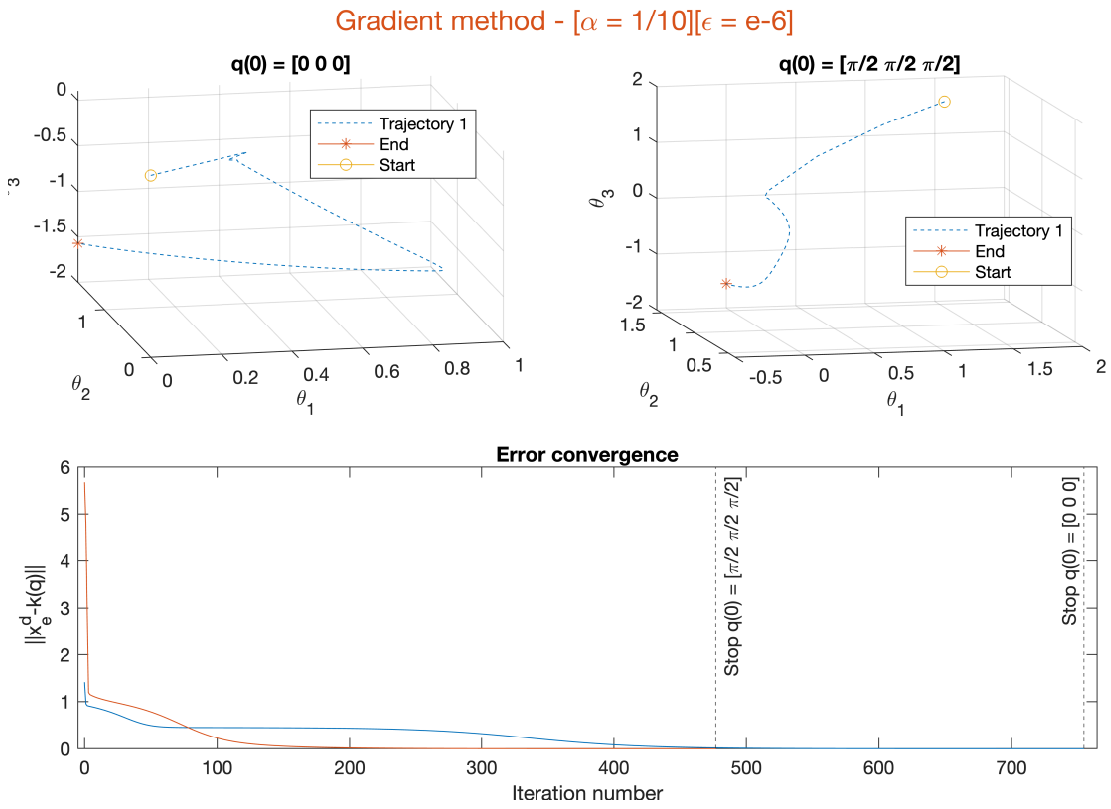

Figure 3: Gradient method with acceptable step-size

$q(0)$	$\theta_{1,f}$	$\theta_{2,f}$	$\theta_{3,f}$	$\lvert x_e^d - \kappa(\theta_f)\rvert$	Final Iter f.
$[0\ 0\ 0]^T$	0	$\frac{\pi}{2}$	$-\frac{\pi}{2}$	$4.6\text{e}-5$	755
$[\frac{\pi}{2}\ \frac{\pi}{2}\ \frac{\pi}{2}]^T$	0	$\frac{\pi}{2}$	$-\frac{\pi}{2}$	$4.6\text{e}-5$	477

Table 3: Gradient method final solutions for $\alpha = 1/10$

2.3 Newton's method

To implement this method we exploit the known relation

$$q^{k+1} = q^k + J^{-1}(q^k)(x_e^d - q^k)$$

In the implementation of this algorithm, we have to face one problem: the Jacobian is not always invertible. By exploiting the fact that the analyzed robot is redundant, instead of the Jacobian inverse we consider the Jacobian pseudo-inverse (we obtain exactly the same result). We evaluate the solutions starting from both the given initial conditions. For both simulations, we set an error threshold $\epsilon = 10^{-6}$ and the maximum number of iterations is $M = 50$. Results are reported in fig. 4 and in table 4.

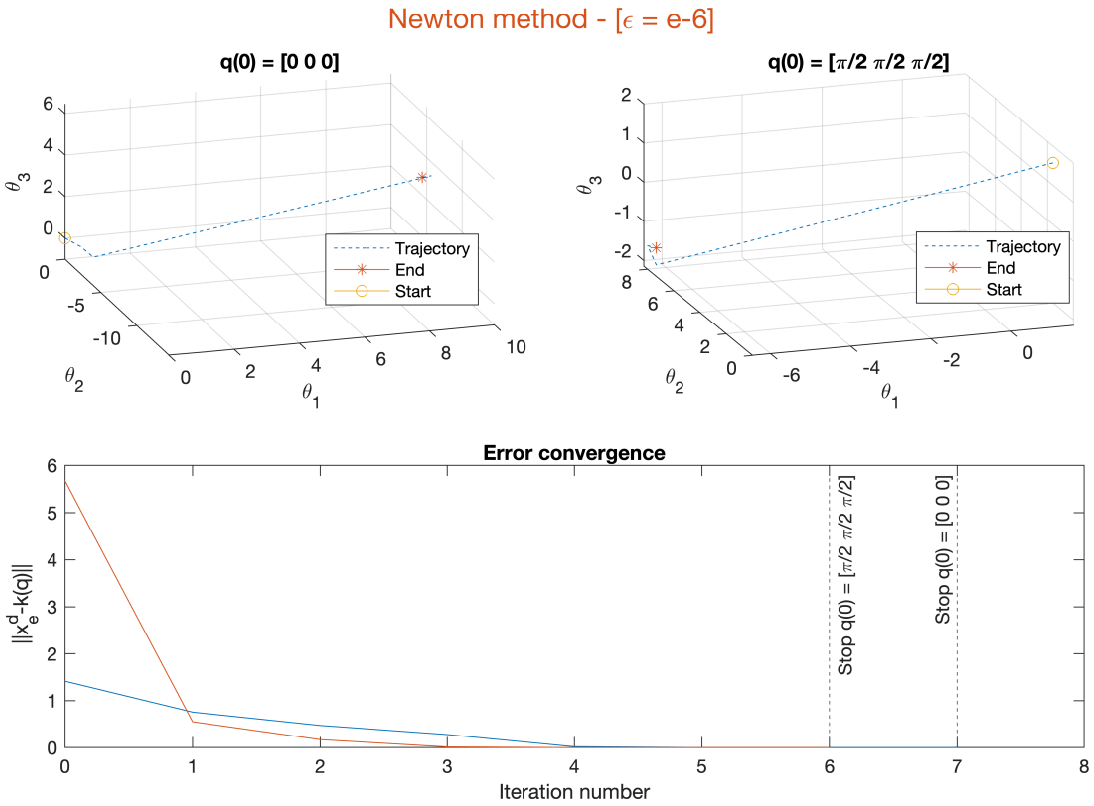

Figure 4: Caption

First of all, we notice that Newton's algorithm converges a very small number of iterations, with a really good approximation ($\epsilon = 10^{-6}$ is the minimum one we desire, 0 is the final achieved error). In fig. 4 we can appreciate the trajectories that the algorithm follows starting from different initial conditions. The two relative numerical solutions we find coincide with the two analytical ones reported in table 1.

$q(0)$	$\theta_{1,f}$	$\theta_{2,f}$	$\theta_{3,f}$	$\lvert x_e^d - \kappa(\theta_f)\rvert$	Final Iter f.
$[0\ 0\ 0]^T$	$\frac{5}{2}\pi$	$-\frac{9}{2}\pi$	2π	0	7
$\left[\frac{\pi}{2}\ \frac{\pi}{2}\ \frac{\pi}{2}\right]^T$	2π	$\frac{5}{2}\pi$	$-\frac{\pi}{2}$	0	6

Table 4: Newton's method final solutions

A Gradient Method

Listing 2: Gradient Method

```
function [q, nIter, err] = iterativeGradient(J, k, q0, x_de, stepSize, maxIter, errTh)
% ---------------------- OUTPUT ----------------------
% q: vector of the solution evaluated at each step;
% nIter: number of the stopping interation;
% err: vector of the error evaluated at each step;

    % Initialization
    syms x y z
    qTmp = zeros(3,maxIter);
    errTmp = zeros(1, maxIter+1);
    nIter = 0;

    % Evaluation of the error at first iteration (we consider error norm)
    qTmp(: , 1) = q0;
    kTmp = subs(k,{x,y,z},qTmp(:,1)');
    errTmp(1) = norm(x_de-kTmp);

    j = 2;
    while (nIter < maxIter)
        nIter = j-1;
        % Jacobian at previus iteration
        Jtmp = subs(J,{x,y,z},qTmp(:,j-1)');
        % Solution update
        qTmp(:,j) = qTmp(:,j-1) + stepSize*(Jtmp)'*(x_de-kTmp);
        % K(q) at current iteration
        kTmp = subs(k,{x,y,z},qTmp(:,j)');
        % K(q) at current iteration
        errTmp(j) = norm(x_de-kTmp);

        % Early stopping strategy based on error's improvement
        if abs(errTmp(j)-errTmp(j-1)) < errTh
            break
        end
        j = j+1;
    end

    % Output made only of nonzero elements
    q = qTmp(:,1:nIter+1);
    err = errTmp(1:nIter+1);
end
```

B Newton's Method

Listing 3: Netwon's Method

```
function [q, nIter, err] = iterativeNewton(J, k, q0, x_de, maxIter, errTh)
% ---------------------- OUTPUT ----------------------
% q: vector of the solution evaluated at each step;
% nIter: number of the stopping interation;
% err: vector of the error evaluated at each step;

    % Initialization
    syms x y z
    qTmp = zeros(3,maxIter);
    errTmp = zeros(1, maxIter);
    nIter = 0;

    % Evaluation of the error at first iteration (we consider error norm)
    qTmp(: , 1) = q0;
    kTmp = subs(k,{x,y,z},qTmp(:,1)');
    errTmp(1) = norm(x_de-kTmp);

    j = 2;
    while (nIter < maxIter)
        nIter = j-1;
        % Jacobian at previus iteration
        Jtmp = subs(J,{x,y,z},qTmp(:,j-1)');
        % Evaluation of the pseudo-inverse - this handles the
        % non-invertible case
        Jinv=pinv(Jtmp);
        % Solution update
        qTmp(:,j) = qTmp(:,j-1) + Jinv*(x_de-kTmp);
        % K(q) at current iteration
        kTmp = subs(k,{x,y,z},qTmp(:,j)');
        % K(q) at current iteration
        errTmp(j) = norm(x_de-kTmp);

        % Early stopping strategy based on error's improvement
        if abs(errTmp(j)-errTmp(j-1)) < errTh
            break
        end
        j = j+1;
    end
    % Considering as output only nonzero elements
    q = qTmp(:,1:nIter+1);
    err = errTmp(1:nIter+1);
end
```