HOMEWORK #1 ROBOTICS AND CONTROL 1

Davide Peron 2082148

Academic Year 2022/2023

1 Analytical Solution

Considering the planar 3R robotic arm depicted in fig. 1a, we want to achieve the desired pose at the end effector $x_e^d = [2\ 1\ 0]^T$; by imposing $x_e^d = \kappa(q)$ we obtain the following system:

$$\begin{cases} \cos(\theta_1) + \cos(\theta_1+\theta_2) + \cos(\theta_1+\theta_2+\theta_3) = 2 & (1) \\ \sin(\theta_1) + \sin(\theta_1+\theta_2) + \sin(\theta_1+\theta_2+\theta_3) = 1 & (2) \\ \theta_1+\theta_2+\theta_3 = 0 & (3) \end{cases}$$

We can now reformulate eq. 3 as follows:

$$\theta_1 + \theta_2 = -\theta_3 \tag{4}$$

If we now substitute eq. 3 and eq. 4 into eq. 1 and eq. 2 we obtain:

$$\begin{cases} \cos(\theta_1) + \cos(-\theta_3) = \cos(\theta_1) + \cos(\theta_3) = 1 & (5) \\ \sin(\theta_1) + \sin(-\theta_3) = \sin(\theta_1) - \sin(\theta_3) = 1 & (6) \end{cases}$$

We can square both eq. 5 and eq. 6 and sum them term by term to obtain:

$$\cos^2(\theta_1) + \cos^2(\theta_3) + 2\cos(\theta_1)\cos(\theta_3) + \sin^2(\theta_1) + \sin^2(\theta_3) - 2\sin(\theta_1)\sin(\theta_3) = 2$$

$$2 + 2\left(\cos(\theta_1)\cos(\theta_3) - \sin(\theta_1)\sin(\theta_3)\right) = 2$$

$$\cos(\theta_1+\theta_3) = 0 \tag{7}$$

This equation has infinite solutions so we need to restrict the interval of interest of our variables, we suppose $\theta_i \in (-\pi, \pi]$ and obtain $\theta_1 + \theta_3 = \pm\frac{\pi}{2}$, by using this in eq. 3 we obtain $\theta_2 = \pm\frac{\pi}{2}$. Now, we can substitute this result in eq. 1 and eq. 2 and obtain what follows:

- Substituting $\theta_2 = \frac{\pi}{2}$:

$$\begin{cases} \cos(\theta_1) + \cos\left(\theta_1 + \frac{\pi}{2}\right) = \cos(\theta_1) - \sin(\theta_1) = 1 \\ \sin(\theta_1) + \sin\left(\theta_1 + \frac{\pi}{2}\right) = \sin(\theta_1) + \cos(\theta_1) = 1 \end{cases}$$

  Summing both equations term by term we obtain $\cos(\theta_1) = 1$ which implies $\theta_1 = 0$ and by using eq. 3 we obtain $\theta_3 = \frac{\pi}{2}$

- Substituting $\theta_2 = -\frac{\pi}{2}$:

$$\begin{cases} \cos(\theta_1) + \cos\left(\theta_1 - \dfrac{\pi}{2}\right) = \cos(\theta_1) + \sin(\theta_1) = 1 \\ \sin(\theta_1) + \sin\left(\theta_1 - \dfrac{\pi}{2}\right) = \sin(\theta_1) - \cos(\theta_1) = 1 \end{cases}$$

  Summing both equations term by term we obtain $\sin(\theta_1) = 1$ which implies $\theta_1 = \frac{\pi}{2}$ and by using eq. 3 we obtain $\theta_3 = 0$

In Table 1 we can see the solution obtained and in fig. 1b we can see the robot configuration in the 2 solutions.

	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

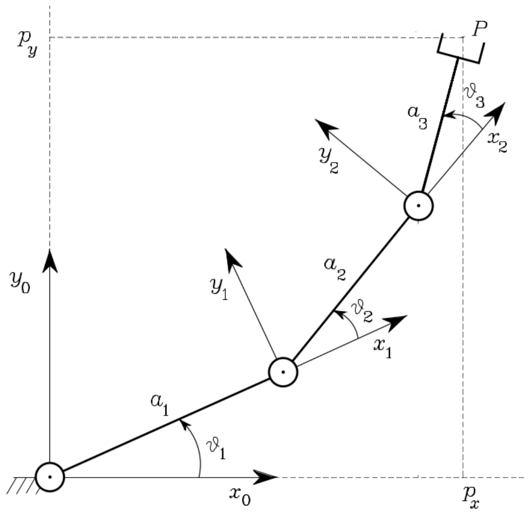

Figure 1.a: Planar 3R robotic arm

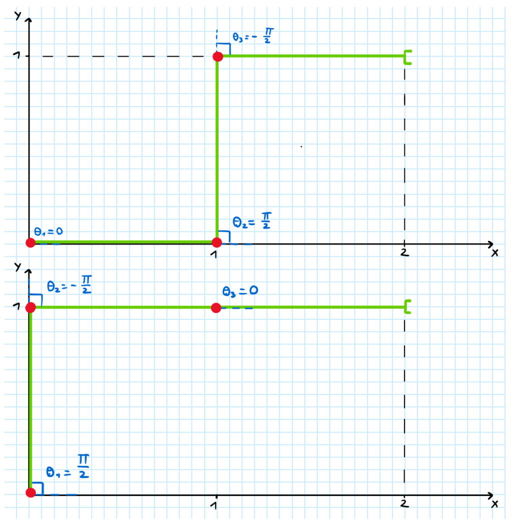
y
1
$\theta_3 = -\frac{\pi}{2}$
$\theta_1 = 0$
$\theta_2 = \frac{\pi}{2}$
1
2
x
y
$\theta_2 = -\frac{\pi}{2}$
$\theta_3 = 0$
1
$\theta_1 = \frac{\pi}{2}$
1
2
x

Figure 1.b: Solution of the inverse kinematics problem

2 Numerical Solutions

2.1 General Setup

In this part of the assignment we are required to implement gradient method and Newton's method to find the solution of the inverse kinematics problem starting from different initial conditions:

- $q_1\left(0\right) = \left[0\ 0\ 0\right]^T$
- $q_2\left(0\right) = \left[\frac{\pi}{2}\ \frac{\pi}{2}\ \frac{\pi}{2}\right]^T$

It is important to note that both algorithms are iterative so it is important to find an updating rule that connects the vector of generalized coordinates at $k-th$ iteration with its value at the $(k-1)-th$ iteration; the two updating rules will be described in the relative subsections. Both algorithms have a similar initialization which is reported in Listing 1. I decided to use the *Symbolic Math Toolbox* to simplify the evaluation of the jacobian at every step without being forced to calculate it at every iteration but just substituting the values obtained at the previous step of the algorithms with the built in command.
Another similarity between the 2 approaches are the stopping conditions:

- Maximum number of iterations.
- Early stopping: when the error between 2 consecutive iterations is smaller than a desired threshold the algorithm stops.

The choice of adding an early stopping strategy has been done to limit the number of iteration when we arrive to a plateau of the solution and so the error decreases too slowly which implies a small improvement in the solution.

Listing 1: Initialization

```
% Clear all the paramters from the workspace and also celar the command window.
clear all
clc
% Define initial conditions (they are the same for both methods).
q0_0 = [0 0 0]';
q0_1 = [pi/2 pi/2 pi/2]';
%Desired position of the end effector
x_de = [2 1 0]';

syms x y z
% Define symbolyc direct kinematics
K = [cos(x)+cos(x+y)+cos(x+y+z),sin(x)+sin(x+y)+sin(x+y+z),x+y+z]'
% Define symbolic jacobian
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

The updating formula of the gradient method is:

$$q^{k+1} = q^k + \alpha J_k^T\left(q^k\right)\left[x_e^d - \kappa\left(q^k\right)\right]$$

Where α is the step-size. In the assignment we are required to test two different step-sizes: $\alpha_1 = \frac{1}{2}$ and $\alpha_2 = \frac{1}{10}$. The code of the algorithm is provided in Appendix A.

2.2.1 Results for α_1

In this paragraph We are going to analyze the results obtain with step-size $\alpha_1 = \frac{1}{2}$, as we can see in fig. 2 the algorithm does not converge to a solution, indeed the error is erratically increasing and decreasing, this means that we are using a step-size that is too big and so we are not able to converge to a singular point and instead we move back and forth around the solution we are trying to find.
In this case the iteration limit was set to 500 and the error threshold to 10^-5, as we can see the algorithm stops after having done the maximum number of iterations allowed since the error greatly varies from iteration to iteration.
The results obtained are reported in the table below:

Star Cond.	θ_1	θ_2	θ_3	$\kappa(q)$	Iter.
$[0\ 0\ 0]^T$	-0.31π	-1.36π	0.60π	$[0.08\ 0.26\ -3.37]^T$	500
$[\frac{\pi}{2}\ \frac{\pi}{2}\ \frac{\pi}{2}]^T$	-0.24π	0.53π	-1.24π	$[0.35\ -0.05\ -2.97]^T$	500

Table 2: Results for step-size $\alpha_1 = \frac{1}{2}$

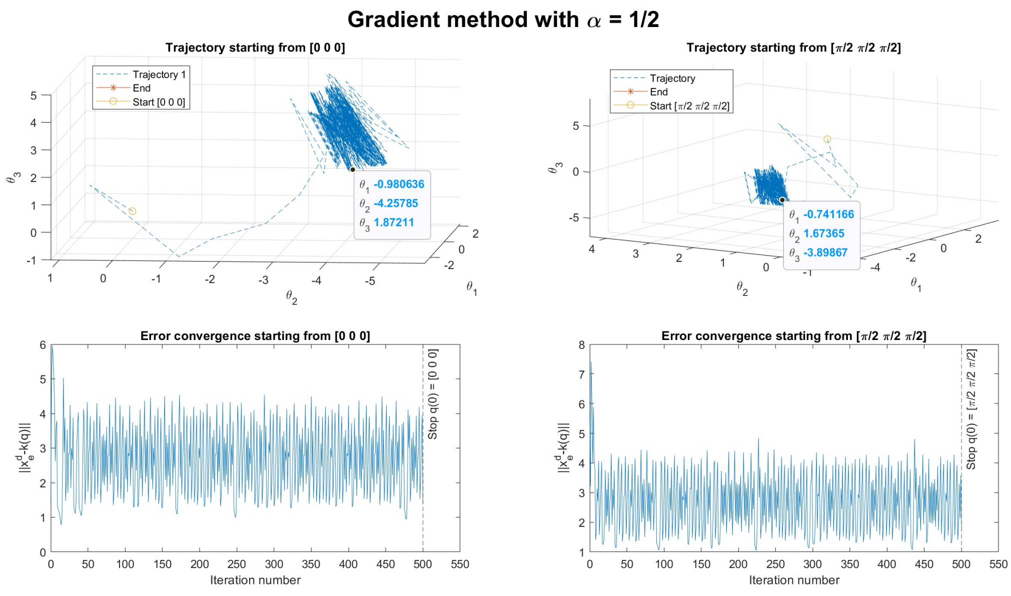

Figure 2: Results obtained with step-size α_1

2.2.2 Results for α_2

In this paragraph we are going to analyze the results obtained by using a smaller step-size, which will hopefully be small enough to make the algorithm converge; in the following we always consider 1000 as the maximum number of iterations.
As we can see in fig. 3 for the first starting condition we considered two different error

thresholds to highlight how the algorithm works:

- **Error Threshold** 10^{-4}: in this case the algorithm stops without finding an accurate solution because the error does not decrease enough between subsequent iterations. This can be something we expect because the algorithm is not guaranteed to converge to the solution we require but it can converge to any singular point of the error function, so we can conclude that the solution found is, probably, near a saddle point of the error.
- **Error Threshold** 10^{-5}: in this second case the algorithm is able to converge to the desired solution, without stopping early; we can conclude that in the previous case we were only near a saddle point and did not converge to it but we are able to escape and then reach the solution at the expenses of needing more iterations and consequently more time to converge.

The second starting conditions was tested with both thresholds and we decided to use the smaller one even though it did not impact the convergence as much as it did with the first starting condition i.e. almost the same number of iterations to converge and almost the same precision for the solution.
The results obtained are reported in the table below:

Start Cond.	Err. Threshold	θ_1	θ_2	θ_3	$\kappa(q)$	Iter.
$[0\ 0\ 0]^T$	10^{-4}	0.27π	0.27π	-0.39π	$[2.23\ 1.26\ -0.29]^T$	83
$[0\ 0\ 0]^T$	10^{-5}	0	$\frac{\pi}{2}$	$-\frac{\pi}{2}$	$[2.00\ 1.00\ 0.00]^T$	648
$[\frac{\pi}{2}\ \frac{\pi}{2}\ \frac{\pi}{2}]^T$	10^{-5}	0	$\frac{\pi}{2}$	$-\frac{\pi}{2}$	$[2.00\ 1.00\ 0.00]^T$	370

Table 3: Results of Gradient Method for step-size $\alpha_2 = \frac{1}{10}$

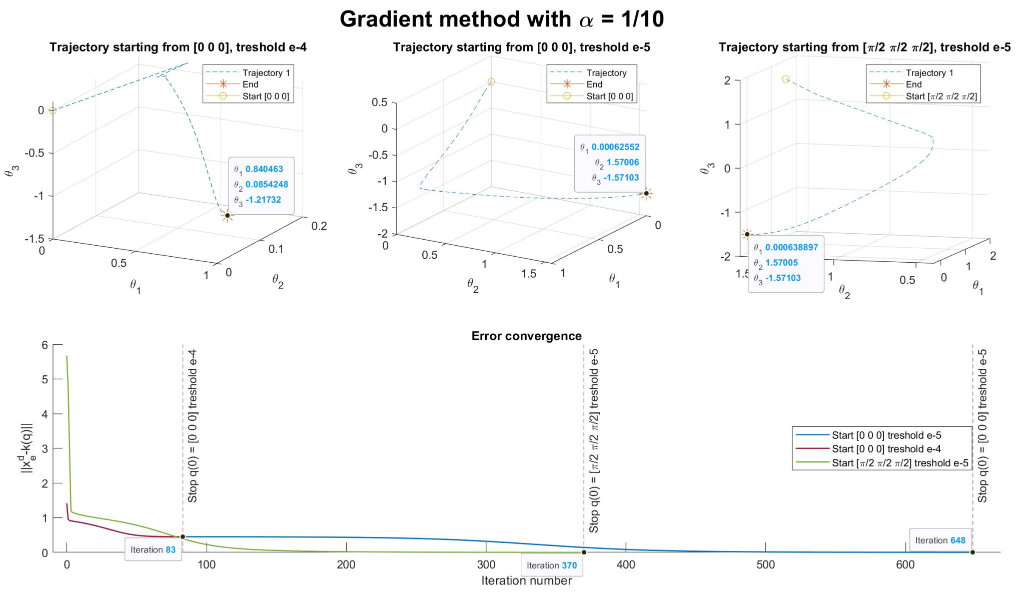

Figure 3: Results obtained with Gradient Method and step-size α_2

2.3 Newton's Method

The updating formula of Newton's method is:

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

As we can see from the updating formula we need to invert the jacobian but this cannot be possible everywhere, indeed if the jacobian is singular we cannot calculate it's inverse and this would be the case with the first starting condition due to this we decided to use the Moore-Penrose pseudo-inverse (pinv method provided by Matlab) instead of the inverse and this solves the problem with the singularity of the jacobian although it being more computationally expensive. It is also important to note that in the cases where the jacobian is not singular using the inverse or the pseudo-inverse has the same exact effect because the two results are the same. The code of the algorithm is provided in Appendix B.
For both starting conditions the error threshold was set to 10^{-5} and the iteration limit to 20 since we expect this method to be much faster than Gradient method if we are near a solution.
As we can observe in fig. 4 we converge to some solutions which are the same as the one we expect only translated by factors of 2π, this is consistent with what we can expected, indeed in the analytical solution we limited our solution to the interval $(-\pi\ \pi]$ but in reality we have infinite solutions translated by a factor of 2π; a possible solution to this is to include a constraint in the algorithm to obtain the solution in the desired interval; I decided not to implement this constraint and to keep the code as simple as possible. It is also immediate to observe how fast is the convergence of this method in the particular cases we analyzed, we it took only 6 or 7 iterations to converge to the solution instead of over 300 that we had with Gradient . The results obtained are reported in the table below:

Start Cond.	θ_1	θ_2	θ_3	$\kappa(q)$	Iter.
$[0\ 0\ 0]^T$	$\frac{5}{2}\pi$	$-\frac{9}{2}\pi$	2π	$[2.00\ 1.00\ 0.00]^T$	7
$\left[\frac{\pi}{2}\ \frac{\pi}{2}\ \frac{\pi}{2}\right]^T$	2π	$\frac{5}{2}\pi$	$-\frac{\pi}{2}$	$[2.00\ 1.00\ 0.00]^T$	6

Table 4: Results of Newton's Method

2.4 Comparative Results

In this last section we are going to analyze the main differences between the two methods from the results obtained.
It is pretty clear that Newton's method is faster that Gradient method for the initial conditions we considered but, it is important to note that in some cases Newton's method might be numerically unstable even though we are considering the pseudo-inverse and so, it might not be really suited for real time calculations. Gradient method also took a lot longer to converge, if it converges at all since this is not guaranteed. The best solution might be to combine the two methods to avoid both the numerical instability and the slow speed of convergence.
If we now consider the accuracy of the results we obtained we can observe that both algo-

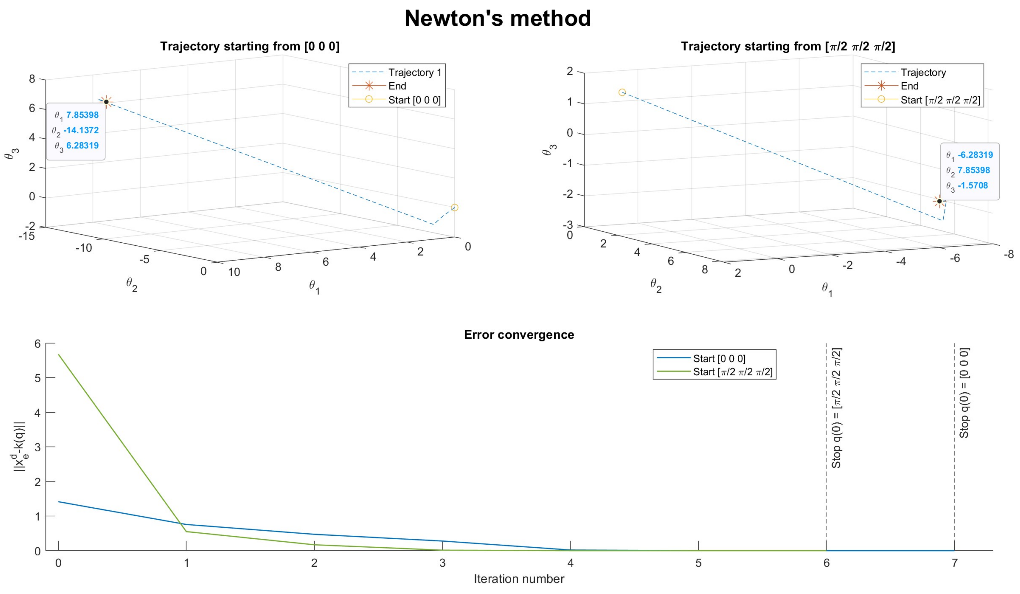

Figure 4: Results obtained with Newton's Method

rithms found accurate solutions, if they converged. It is also important to note that Gradient method, for the initial conditions we considered, converged to solution inside the desired interval instead Newton's method converged to correct solutions but that need to be modified afterwards to be really used in a real scenario.

A Gradient Method Code

Listing 2: Gradient Method

```
function [q, n_iter, err] = gradient_method(Jacobian, k, q_0, x_des, step_size, max_iter, err_treshold
    )
    % Initializing necessary variables and matrices
    syms x y z
    q_temp = zeros(3,max_iter);
    err_temp = zeros(1, max_iter+1);
    n_iter = 0;
    % Finding the error at iteration 0
    q_temp(: , 1) = q_0;
    k_temp = subs(k,{x,y,z},q_temp(:,1)');
    err_temp(1) = norm(x_des-k_temp);

    j = 2;
    while (n_iter < max_iter)
        n_iter = j-1;
        % Calculating jacobian at previus iteration's solution
        J_temp = subs(Jacobian,{x,y,z},q_temp(:,j-1)');
        % Solution's updating formula
        q_temp(:,j) = q_temp(:,j-1) + step_size*(J_temp)'*(x_des-k_temp);
        % Calculating K(q) after the gradient step
        k_temp = subs(k,{x,y,z},q_temp(:,j)');
        % Calculating error at the current iteration
        err_temp(j) = norm(x_des-k_temp);
        % Implementing early stopping strategy
        if abs(err_temp(j)-err_temp(j-1)) < err_treshold
            break
        end
        j = j+1;
    end
    % Considering as output only nonzero elements
    q = q_temp(:,1:n_iter+1);
    err = err_temp(1:n_iter+1);
end
```

B Newton's Method Code

Listing 3: Newton's Method

```matlab
function [q, n_iter, err] = newton_method(Jacobian, k, q_0, x_des, max_iter, err_treshold)
    % Initializing necessary variables and matrices
    syms x y z
    q_temp = zeros(3,max_iter);
    err_temp = zeros(1, max_iter);
    n_iter = 0;
    % Finding the error at iteration 0
    q_temp(: , 1) = q_0;
    k_temp = subs(k,{x,y,z},q_temp(:,1)');
    err_temp(1) = norm(x_des-k_temp);

    j = 2;
    while (n_iter < max_iter)
        n_iter = j-1;
        % Calculating jacobian at previus iteration's solution
        J_temp = subs(Jacobian,{x,y,z},q_temp(:,j-1)');
        % Calculating pseudo-inverse
        J_inv=pinv(J_temp);
        % Solution's updating formula
        q_temp(:,j) = q_temp(:,j-1) + J_inv*(x_des-k_temp);
        % Calculating K(q) after the Newton's step
        k_temp = subs(k,{x,y,z},q_temp(:,j)');
        % Calculating error at the current iteration
        err_temp(j) = norm(x_des-k_temp);
        % Implementing early stopping strategy
        if abs(err_temp(j)-err_temp(j-1)) < err_treshold
            break
        end
        j = j+1;
    end
    % Considering as output only nonzero elements
    q = q_temp(:,1:n_iter+1);
    err = err_temp(1:n_iter+1);
end
```