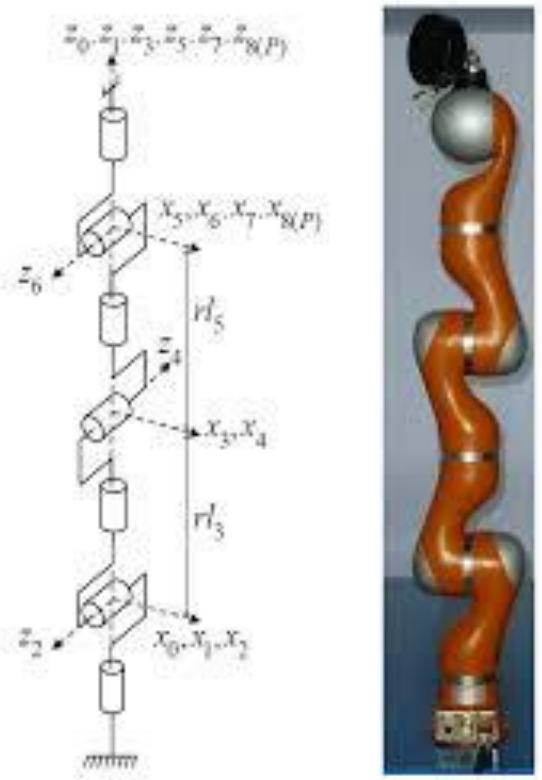
Jose Corona

Home Work 4

Implement function for calculating the robot Jacobian



```
L = [340 200 200 200 200 126 4]
```

FK: Implemented using rotation in z, x and translation in z matrices.

```
T=(Rz(q(1))*Tz(L(1))*Rx(q(2))*Tz(L(2))*Rz(q(3))*Tz(L(3))
  *Rx(-q(4))*Tz(L(4))*Rz(q(5))*Tz(L(5))*Rx(q(6))*Tz(L(6))
  *Rz(q(7))*Tz(L(7)))
```

Jacobian: Numerical method

```
Td=Rzd(q(1))*Tz(L(1))*Rx(q(2))*Tz(L(2))*Rz(q(3))*Tz(L(3))*Rx(-q(4))*Tz(L(4))*Rz(q(5))*Tz(L(5))*Rx(q(6))*Tz(L(6))*Rz(q(7))*Tz(L(7))*..
    [R^-1 zeros(3,1);0 0 0 1];
J_1 = [Td(1,4), Td(2,4), Td(3,4), Td(3,2), Td(1,3), Td(2,1)]' ; % extract 6 components from 4x4 Td matrix to Jacobian 1st column
% diff by q2
Td=Rz(q(1))*Tz(L(1))*Rxd(q(2))*Tz(L(2))*Rz(q(3))*Tz(L(3))*Rx(-q(4))*Tz(L(4))*Rz(q(5))*Tz(L(5))*Rx(q(6))*Tz(L(6))*Rz(q(7))*Tz(L(7))*..
    [R^-1 zeros(3,1);0 0 0 1];
J_2 = [Td(1,4), Td(2,4), Td(3,4), Td(3,2), Td(1,3), Td(2,1)]' ; % extract 6 components from 4x4 Td matrix to Jacobian 1st column
% diff by q3
Td=Rz(q(1))*Tz(L(1))*Rx(q(2))*Tz(L(2))*Rzd(q(3))*Tz(L(3))*Rx(-q(4))*Tz(L(4))*Rz(q(5))*Tz(L(5))*Rx(q(6))*Tz(L(6))*Rz(q(7))*Tz(L(7))*..
    [R^-1 zeros(3,1);0 0 0 1];
J_3 = [Td(1,4), Td(2,4), Td(3,4), Td(3,2), Td(1,3), Td(2,1)]' ; % extract 6 components from 4x4 Td matrix to Jacobian 1st column

Td=Rz(q(1))*Tz(L(1))*Rx(q(2))*Tz(L(2))*Rz(q(3))*Tz(L(3))*Rxd(-q(4))*Tz(L(4))*Rz(q(5))*Tz(L(5))*Rx(q(6))*Tz(L(6))*Rz(q(7))*Tz(L(7))*..
    [R^-1 zeros(3,1);0 0 0 1];
J_4 = [Td(1,4), Td(2,4), Td(3,4), Td(3,2), Td(1,3), Td(2,1)]' ; % extract 6 components from 4x4 Td matrix to Jacobian 1st column

Td=Rz(q(1))*Tz(L(1))*Rx(q(2))*Tz(L(2))*Rz(q(3))*Tz(L(3))*Rx(-q(4))*Tz(L(4))*Rzd(q(5))*Tz(L(5))*Rx(q(6))*Tz(L(6))*Rz(q(7))*Tz(L(7))*..
    [R^-1 zeros(3,1);0 0 0 1];
J_5 = [Td(1,4), Td(2,4), Td(3,4), Td(3,2), Td(1,3), Td(2,1)]' ; % extract 6 components from 4x4 Td matrix to Jacobian 1st column

Td=Rz(q(1))*Tz(L(1))*Rx(q(2))*Tz(L(2))*Rz(q(3))*Tz(L(3))*Rx(-q(4))*Tz(L(4))*Rz(q(5))*Tz(L(5))*Rxd(q(6))*Tz(L(6))*Rz(q(7))*Tz(L(7))*..
    [R^-1 zeros(3,1);0 0 0 1];
J_6 = [Td(1,4), Td(2,4), Td(3,4), Td(3,2), Td(1,3), Td(2,1)]' ; % extract 6 components from 4x4 Td matrix to Jacobian 1st column

Td=Rz(q(1))*Tz(L(1))*Rx(q(2))*Tz(L(2))*Rz(q(3))*Tz(L(3))*Rx(-q(4))*Tz(L(4))*Rz(q(5))*Tz(L(5))*Rx(q(6))*Tz(L(6))*Rzd(q(7))*Tz(L(7))*..
    [R^-1 zeros(3,1);0 0 0 1];
J_7 = [Td(1,4), Td(2,4), Td(3,4), Td(3,2), Td(1,3), Td(2,1)]' ; % extract 6 components from 4x4 Td matrix to Jacobian 1st column


 % Full Jacobian 6x7
 Jq1 = [J_1, J_2, J_3, J_4, J_5, J_6, J_7];
```

Test of the Jacobian function

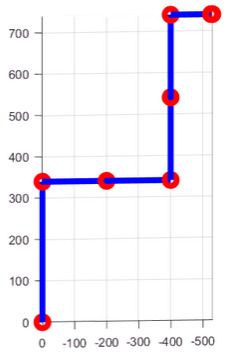
```
%Jacobian test

L_test1 = [340 200 200 200 200 126 4];

q_test1 = [pi/2 0 0 0 0 0 0];

Jq1=Jacobian(q_test1,L_test1)
```

```
 Jq1 =
         0  930.0000         0  530.0000         0  130.0000         0
         0   -0.0000         0   -0.0000         0   -0.0000         0
         0         0         0         0         0         0         0
         0    0.0000         0    0.0000         0    0.0000         0
         0    1.0000         0    1.0000         0    1.0000         0
    1.0000         0    1.0000         0    1.0000         0    1.0000
```

Implement simple robot motion visualization

Draw the robot

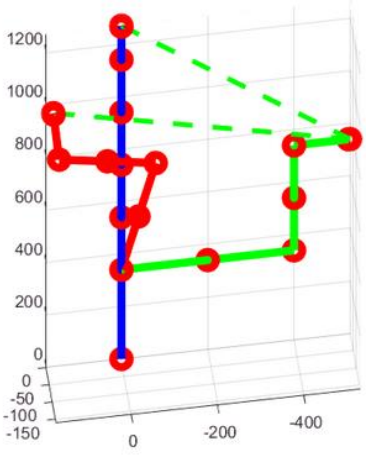
```
q_test1 = [0 pi/2 0  pi/2 0 pi/2 0];
```



Draw the robot trajectory position, 3D visualiation

```
L_test1 = [340 200 200 200 200 126 4];

q_test0 = [0 0 0 0 0 0 0];
q_test1 = [pi/2 0 0 0 0 0 0];
q_test2 = [0 pi/2 0  pi/2 0 pi/2 0];
q_test3 = [0 0.2 0.5  1 0.5 1 0];

draw_trayectory_robot(q_test1,q_test2,q_test3,L_test1) %draw trayectory for 3 positions
```

The first position is the robot arm of color blue, the second position is the robot of color green and the last position is the robot of color red. The trajectory is the line of color green.



Code ^

```
view([-98 53])
```

3. Implement IK function for the robot based on

a. Weighted pseudoinverse: I implement just pseudoinverse and Weighted pseudoinverse methods in one function. In the weighted pseudoinverse I increase the weighted of the first and second joints so they take priority.

```
if(only_position == 1) %1 for pseudo inverse
    %J = J(1:3,:);   %first 3 rows, Jacobian secction for joints velocity
    Ji = pinv(J);    %pseudo inverse
    e_o= [0;0;0];
elseif(only_position == 2)  %2 for pseudo inverse weighted
    W=[2 0 0 0 0 0 0;
       0 2 0 0 0 0 0;
       0 0 1 0 0 0 0;
       0 0 0 1 0 0 0;
       0 0 0 0 0.5 0 0;
       0 0 0 0 0 0.5 0;
       0 0 0 0 0 0 0.5;];
    Ji = inv(W^0.5)*pinv(J); %pseudo inverse weighted
    e_o= [0;0;0];
else
```

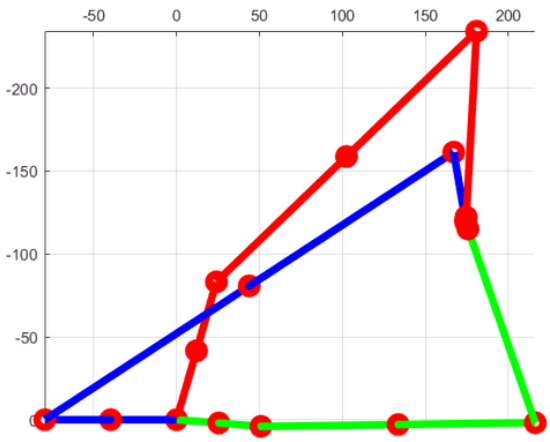
And calculated de joint position

```
e = [e_p; e_o]; %error in position and error in orientation 1x6
q = q + Ji*e; % The solution is updated
k = k + 1; % the iteration number is uodated
```

Then I visualize the arm position.

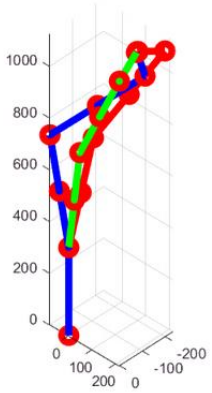
```
%Just Pseudo Inverser
q_ik_pseudoInverse = IK_Jacobbian(p_target, q_test0, L_test1, 1) %number 1 - parameter for Just Pseudo Inverser
p_ik_pseudoInverse = FK(q_ik_pseudoInverse,L_test1)

%Weighted Pseudo Inverser
q_ik_pseudoInverse_weighted = IK_Jacobbian(p_target, q_test0, L_test1, 2)%number 2 - parameter for Just Weighted Pseudo Inverser
p_ik_pseudoInverse_weightedp_ik_pseudoInverse_weightedp_ik_pseu = FK(q_ik_pseudoInverse_weighted,L_test1)
```

In the next figures, are presented different views of one target position of the arm. The arm of color blue is the target joint position, in green is the pseudo inverse solution and in color red is the solution using the weighted pseudo inverse Jacobian.



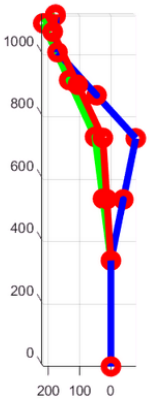
Code ^

```
xlim([-234 4])
ylim([-79 216])
zlim([0 1125])
view([90 90])
```



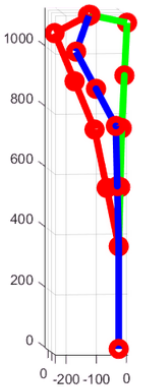
Code ^

```
xlim([-234 4])
ylim([-79 216])
zlim([0 1125])
view([-226 38])
```



Code ^

```
xlim([-234 4])
ylim([-79 216])
zlim([0 1125])
view([-89 -2])
```

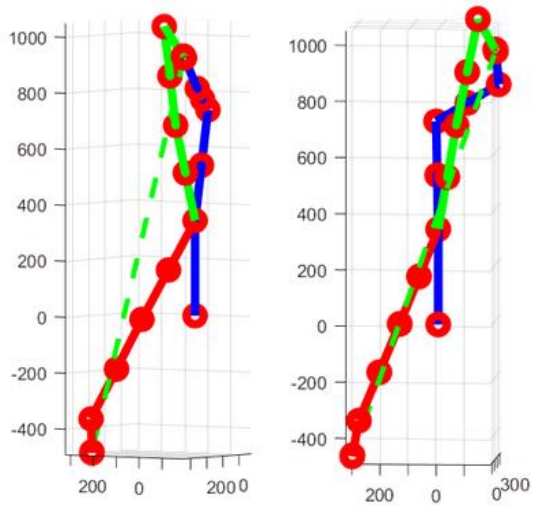


Code ^

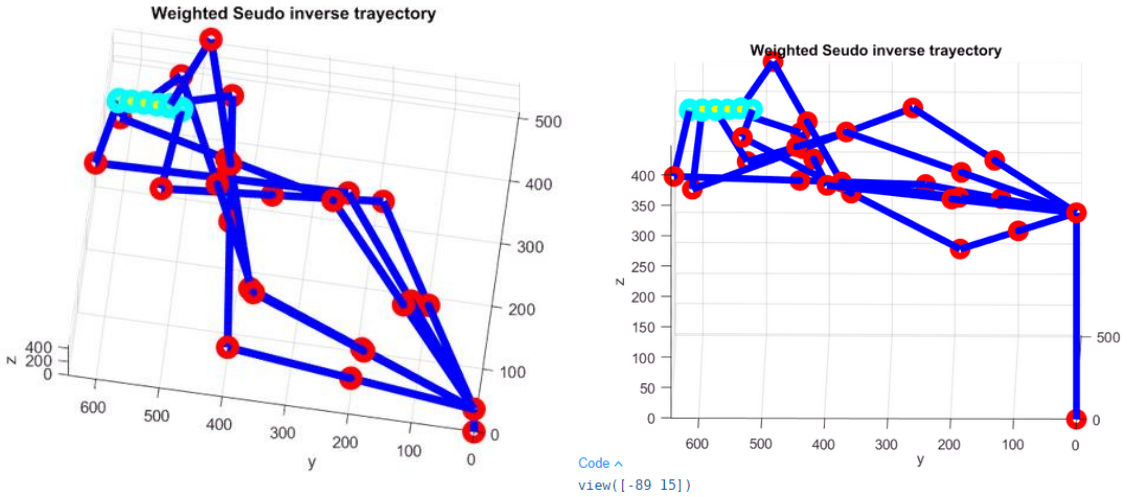
```
xlim([-234 4])
ylim([-79 216])
zlim([0 1125])
view([7 -5])
```

```
ans =
         0
    0.2000
    0.5000
    1.0000
    0.5000
    1.0000
         0

q_ik_pseudoInverse =
   -0.0789
   -0.1280
   -3.0122
   -0.2989
   -1.2430
   -1.5294
   -1.8686

q_ik_pseudoInverse_weighted =
    1.2895
   -0.2180
   -0.7713
    0.4006
   -2.2111
   -1.5448
  -12.4262
```

I try other positions, and in some of them the weighted and pseudoinverse version diverges, so we have to take into account singularities in the workspace.



In the next figure we could the robot fallowing a trayectoy, where the final effector is in color cyan, the trayectory line is color yellow and the green dotted line follows the final effector trayectory. The trayectory to follows was a stright line in Y from 500 to 620, X=400 and Z=400



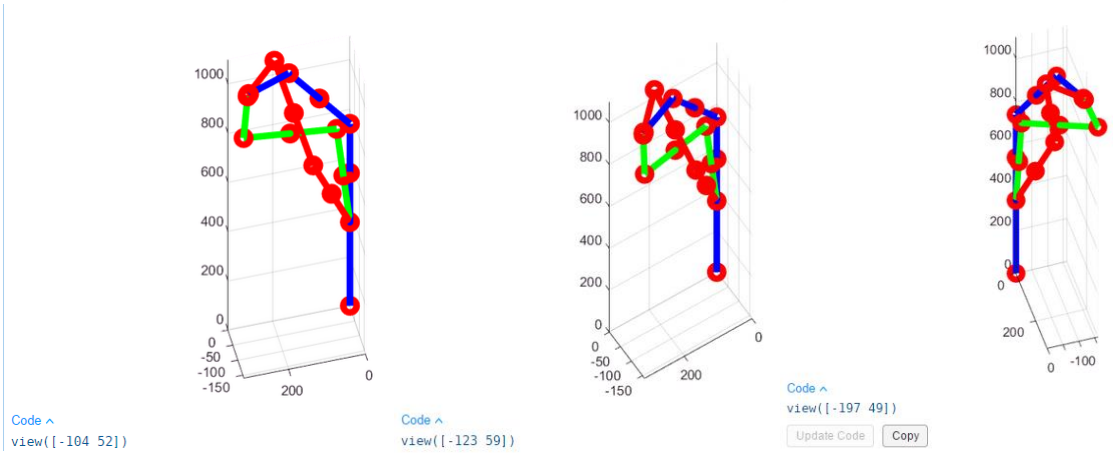
b. Damped Least Squares : I implemented this method in the same function.

```
if(only_position == 1) %1 for pseudo inverse
    %J = J(1:3,:);   %first 3 rows, Jacobian secction for joints velocity
    Ji = pinv(J);    %pseudo inverse
    e_o= [0;0;0];
elseif(only_position == 2)  %2 for pseudo inverse weighted
    W=[2 0 0 0 0 0 0;
       0 2 0 0 0 0 0;
       0 0 1 0 0 0 0;
       0 0 0 1 0 0 0;
       0 0 0 0 0.5 0 0;
       0 0 0 0 0 0.5 0;
       0 0 0 0 0 0 0.5;];
    Ji = inv(W^0.5)*pinv(J); %pseudo inverse weighted
    e_o= [0;0;0];
else  %Damped Least Squares
      R = T(1:3, 1:3);
      s = rot2cuat(R);
      s_e = multcuat(sd, invcuat(s));
      e_o = s_e(2:end);%
    u_2 = 0.1 ;
    Ji = J'/(J*J'+u_2*eye(6)); %Damped Least Squares
    e_o= [0;0;0];
end
e = [e_p; e_o]; %error in position and error in orientation 1x6
q = q + Ji*e; % The solution is updated
k = k + 1; % the iteration number is uodated
```

And visualize a targe position

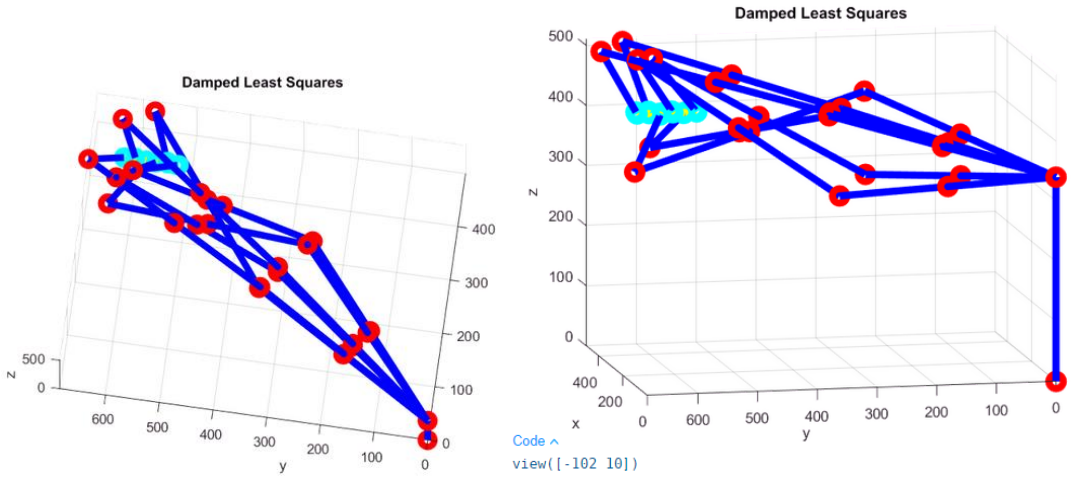
```
%Weighted Pseudo Inverser
q_ik_pseudoInverse_weighted = IK_Jacobbian(p_target, q_test0, L_test1, 2)%number 2 - parameter for Just Weighted Pseudo Inverser
p_ik_pseudoInverse_weightedp_ik_pseudoInverse_weightedp_ik_pseu = FK(q_ik_pseudoInverse_weighted,L_test1)

%Damped Least Squares
q_ik_Damped_Least_Squares = IK_Jacobbian(p_target, q_test0, L_test1, 3) %number 3 - parameter
p_ik_Damped_Least_Squares = FK(q_ik_Damped_Least_Squares,L_test1)


%draw target position, position using  weighted pseudo inverse and Damped Least Squares
draw_trayectory_robot(q_test3,q_ik_pseudoInverse_weighted,q_ik_Damped_Least_Squares,L_test1)
```

In the next figures, are presented different views of one target position of the arm. The arm of color blue is the target joint position, in green is the weighted pseudo inverse Jacobian and in color red is the solution using the Damped Least Squares.



I try other positions, and in some of diverges, so for Jacobian based methods we have to take into account singularities in the workspace.

In the next figure we could the robot fallowing a trayectoy, where the final effector is in color cyan, the trayectory line is color yellow and the green dotted line follows the final effector trayectory. The trayectory to follows was a stright line in Y from 500 to 620, X=400 and Z=400



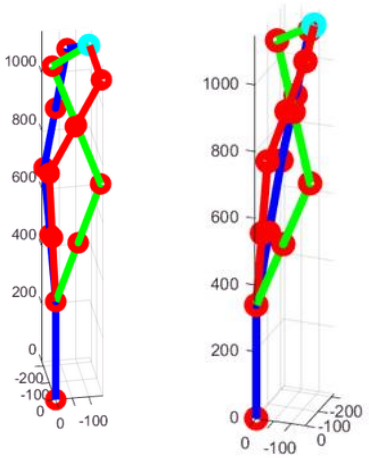
C. <u>Null space method</u> :

I implemented Null space in other file. I calculated the jacobian for each joint, not all the joint at the same time to get a differentiation for each join for dq0 . Since the determinat of everything is just one value, was always cero if I use the jacaobian of all joints.

```
for i=1:7
    onlinjoinq=[0 0 0 0 0 0 0];
    onlinjoinq(i)=q_last(i);

    J_last=Jacobian(onlinjoinq,L0);
    H_last=real(sqrt(det(J_last*(J_last'))));

    onlinjoinq=[0 0 0 0 0 0 0];
    onlinjoinq(i)=q_next(i);
    J_next=Jacobian(onlinjoinq,L0);
    H_next=real(sqrt(det(J_next*(J_next'))));

    Numerical_dev= 0.1*(H_next-H_last);
    d_q0(i)=Numerical_dev;
end
```

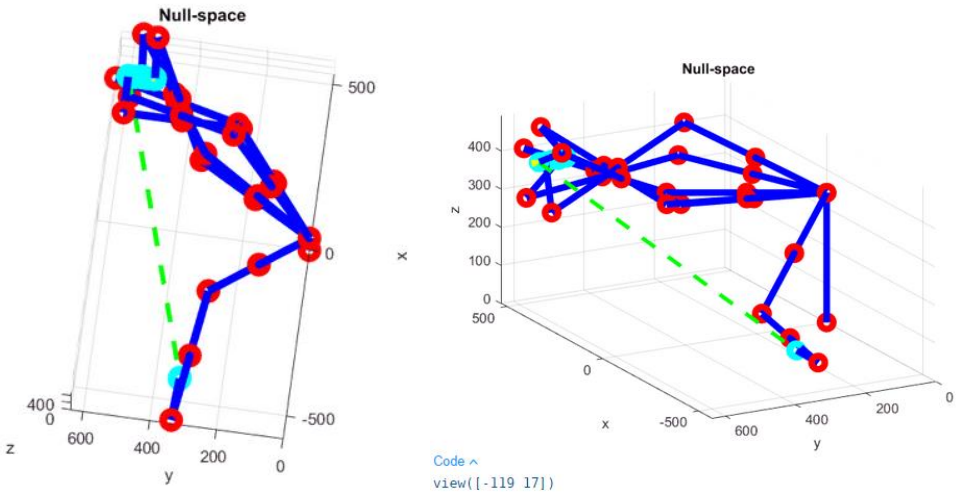
```
d_q0=d_q0';
delta_r=e;

delta_q= pinv(J)*delta_r + (eye(7)-pinv(J)*J)*d_q0; % The solution is updated

q = q + delta_q;

k = k + 1; % the iteration number is uodated
```

In the next figures, are presented different views of one target position of the arm. The arm of color blue is the target joint position, in green is the null space solution with joint initial position zero and in color red is the solution using other initial position.




```
Code ^
view([-190 31])
```

In the next figure we could the robot fallowing a trayectoy, where the final effector is in color cyan, the trayectory line is color yellow and the green dotted line follows the final effector trayectory. The trayectory to follows was a stright line in Y from 500 to 620, X=400 and Z=400




```
Code ^
view([-119 17])
```

The Augmented Task:

Refer to only use some columms of the Jacobian to calculate the Ik. For the first example of the follow a line, we don’t care about the columms of rotation, so the jacobian is only 3x7.

For the part we want to make the tool vertical we are not interested in rotation part around Z, so we don’t use this collumn.

Link Git hub:

https://github.com/Jose-R-Corona/AR-HomeTask4