University of Groningen

Robotics for IEM

Assignment 2

Author:
Jonathan Chandra
(s5161533)

Lecturer:
Dr. Bahar Haghighat

October 12, 2023

university of groningen

faculty of science and engineering

1 Question 1

This question involves the derivation of inverse kinematic solution with its MATLAB computation of a low-cost robotic manipulator AX-18A Smart Robotic Arm.

1. Inverse Kinematics solution

Inverse kinematic solution can be acquired by defining the translation matrix of a homogeneous transformation matrix, that is:

$$ {}^{0}_{6}T = \begin{bmatrix} {}^{0}_{6}R & {}^{0}p_6 \\ 0 & 1 \end{bmatrix} \tag{1} $$

with ${}^{0}p_6$ is:

$$ {}^{0}p_6 = \begin{bmatrix} P_{E,x} \\ P_{E,y} \\ P_{E,z} \end{bmatrix} \tag{2} $$

The translation matrix (2) is predefined in the assignment handout that is:

$$ \begin{aligned} {}^{0}p_6 &= \begin{bmatrix} P_{E,x} \\ P_{E,y} \\ P_{E,z} \end{bmatrix} \\ &= \begin{bmatrix} 9s_1c_4s_5 + 9c_1c_{2+3}s_4s_5 - 9c_1s_{2+3}c_5 - c_1(4c_{2+3} + 11s_{2+3} + 17s_2) \\ -9c_1c_4s_5 + 9s_1c_{2+3}s_4s_5 - 9s_1s_{2+3}c_5 - s_1(4c_{2+3} + 11s_{2+3} + 17s_2) \\ -9s_{2+3}s_4s_5 + 9c_{2+3}c_5 + 17c_2 + 11c_{2+3} - 4s_{2+3} + 17 \end{bmatrix} \end{aligned} \tag{3} $$

In this report a geometric approach was used to acquire $\theta_1, 3$, while algebraic approach was used to obtained θ_2.

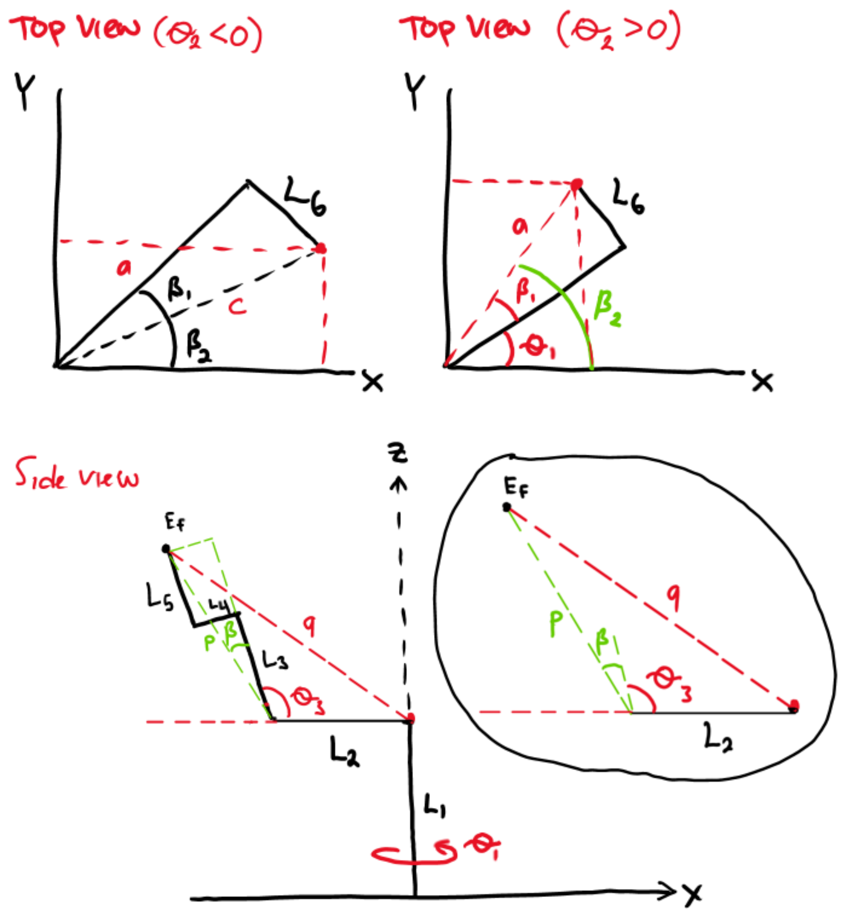

Figure 1: Geometric illustration of AX-18A

First, θ_1 was derived by having two solution that is when $\theta_2 < 0$ and $\theta_2 > 0$ which is depicted in the top of Figure 1.

From the top left image of Figure 1, we acquire:

$$\beta_1 + \beta_2 = \theta_1 \tag{4}$$

where

$$\begin{aligned} \beta_2 &= atan2(P_{E,y}, P_{E,x}) \\ \beta_1 &= atan2(L_6, a) \end{aligned} \tag{5}$$

with $a = \sqrt{P_{E,x}^2 + P_{E,y}^2 - L_6^2}$

From the top right image of Figure 1, we acquire:

$$\beta_2 - \beta_1 = \theta_1 \tag{6}$$

where

$$\begin{aligned} \beta_2 &= atan2(P_{E,y}, P_{E,x}) \\ \beta_1 &= atan2(L_6, b) \end{aligned} \tag{7}$$

with $b = \sqrt{a^2 - L_6^2}$

From the top right image of Figure 1, we acquire:

$$\beta_2 - \beta_1 = \theta_1 \tag{8}$$

where

$$\begin{aligned} \beta_2 &= atan2(P_{E,y}, P_{E,x}) \\ \beta_1 &= atan2(L_6, b) \end{aligned} \tag{9}$$

with $b = \sqrt{a^2 - L_6^2}$

From the bottom image of Figure 1, we acquire:

- $p = \left[(L_3+L_5)^2 + L_4^2 + L_6^2\right]^{1/2}$
- $q = \left[x^2 + (z-L_1)^2 + y^2\right]^{1/2}$
- $\beta = atan2(L_4, L_3+L_5)$
- $\cos(\theta_3+\beta) = \dfrac{p^2 + L_2^2 - q^2}{2pL_2}$
- $\sin(\theta_3+\beta) = \pm\left[1 - \cos^2(\theta_3+\beta)\right]^{1/2}$

$$\theta_3 + \beta = atan2\left[\sin(\theta_3+\beta), \cos(\theta_3+\beta)\right]$$

$$\theta_3 = atan2\left[\sin(\theta_3+\beta), \cos(\theta_3+\beta)\right] - \beta$$

For θ_3 we have:

$$x^2+y^2 = (9s_1 - c_1(4c_{2+3} + 11s_{2+3} + 17s_2))^2 \qquad R = (4c_{2+3} + 11s_{2+3} + 17s_2)$$

$$+(-9c_1 - s_1(4c_{2+3} + 11s_{2+3} + 17s_2))^2$$

$$= [81s_1^2 - 18s_1c_1R + c_1^2R^2 +$$
$$81c_1^2 + 18s_1c_1R + s_1^2R^2]$$

$$x^2+y^2 = 81(s_1^2+c_1^2) + R^2(s_1^2+c_1^2)$$
$$x^2+y^2 = R^2+81$$

$$x^2+y^2-81 = R^2$$

- $\pm\sqrt{x^2+y^2-81} = (4c_2c_3 - 4s_2s_3 + 11s_2c_3 + 11c_2s_3 + 17s_2)$
- $z = 17c_2 + 11c_2c_3 - 11s_2s_3 - 4s_2c_3 - 4c_2s_3 + 17$

$$z-17 = 17c_2 + 11c_2c_3 - 11s_2s_3 - 4s_2c_3 - 4c_2s_3$$

$$b_1 = \pm\sqrt{x^2+y^2-81} \rightarrow c_2(4c_3+11s_3) + s_2(-4s_3+11c_3+17)$$

$$b_2 = z-17$$

$$c_2(17+11c_3-4s_3) + s_2(-11s_3-4c_3)$$

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} (4c_3+11s_3) & (-4s_3+11c_3+17) \\ (-4s_3+11c_3+17) & -(4c_3+11s_3) \end{bmatrix} \begin{bmatrix} c_2 \\ s_2 \end{bmatrix}$$

$$= \begin{bmatrix} k_1 & k_2 \\ k_2 & -k_1 \end{bmatrix} \begin{bmatrix} c_2 \\ s_2 \end{bmatrix}$$

$$\begin{bmatrix} c_2 \\ s_2 \end{bmatrix} = \frac{1}{-k_1^2 - k_2^2} \begin{bmatrix} -k_1 & -k_2 \\ -k_2 & k_1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

$$\theta_2 = \text{atan2}(s_2, c_2)$$

2. Pose for $\theta_{4,5} = 0$

Elbow Up: This is where the robot's "elbow" (the joint corresponding to θ_3) points upwards. Assume a human arm reaching out to something in front while keeping the elbow raised. Elbow Down: This is where the robot's "elbow" points downwards. Assume a human arm reaching the same point in front but with the elbow lowered. Shoulder Front: The "shoulder" (the joint corresponding to θ_2) is positioned in a way that the upper arm points forwards. Shoulder Back: The upper arm points backwards, but the end effector can still reach the same point due to the arm's extended configuration.

3. MATLAB Inverse Kinematic solution Function

A listing of the inverse Kinematic is available in Listing B. This concludes the answer to question 1.

2 Question 2

This question involves on deriving the basic Jacobian $J_{0,v}$ of a low-cost robotic manipulator AX-18A Smart Robotic Arm.

2.1 Jacobian Derivation ($J_{0,v}$)

We consider the position of the end effector expressed in frame zero as:

$$\begin{aligned} {}^0p_6 &= \begin{bmatrix} P_{E,x} \\ P_{E,y} \\ P_{E,z} \end{bmatrix} \\ &= \begin{bmatrix} 9s_1c_4s_5 + 9c_1c_{2+3}s_4s_5 - 9c_1s_{2+3}c_5 - c_1(4c_{2+3} + 11s_{2+3} + 17s_2) \\ -9c_1c_4s_5 + 9s_1c_{2+3}s_4s_5 - 9s_1s_{2+3}c_5 - s_1(4c_{2+3} + 11s_{2+3} + 17s_2) \\ -9s_{2+3}s_4s_5 + 9c_{2+3}c_5 + 17c_2 + 11c_{2+3} - 4s_{2+3} + 17 \end{bmatrix} \end{aligned} \tag{10}$$

The basic Jacobian ($J_{0,v}$) of (10) can be expressed with the following Jacobian matrix [1]:

$$J_{0,v} = \begin{bmatrix} \frac{\partial^0 p_6}{\partial \theta_1} & \frac{\partial^0 p_6}{\partial \theta_2} & \frac{\partial^0 p_6}{\partial \theta_3} & \frac{\partial^0 p_6}{\partial \theta_4} & \frac{\partial^0 p_6}{\partial \theta_5} \end{bmatrix} \tag{11}$$

Thus,

$$J_{0,v} = \begin{bmatrix} f_1 & f_2 & f_3 & f_4 & f_5 \end{bmatrix} \tag{12}$$

where:

$$f_1 = \begin{bmatrix} 9c_1c_4s_5 - 9s_1c_{2+3}s_4s_5 + 9s_1s_{2+3}c_5 + s_1(4c_{2+3} + 11s_{2+3} + 17s_2) \\ 9s_1c_4s_5 + 9c_1c_{2+3}s_4s_5 - 9c_1s_{2+3}c_5 - c_1(4c_{2+3} + 11s_{2+3} + 17s_2) \\ 0 \end{bmatrix}$$

$$f_2 = \begin{bmatrix} -9c_1s_{2+3}s_4s_5 - 9c_1c_{2+3}c_5 - c_1(-4s_{2+3} + 11c_{2+3} + 17c_2) \\ -9s_1s_{2+3}s_4s_5 - 9s_1c_{2+3}c_5 - s_1(-4s_{2+3} + 11c_{2+3} + 17c_2) \\ -9c_{2+3}s_4s_5 - 9s_{2+3}c_5 - 17s_2 - 11s_{2+3} - 4c_{2+3} \end{bmatrix}$$

$$f_3 = \begin{bmatrix} -9c_1s_{2+3}s_4s_5 - 9c_1c_{2+3}c_5 - c_1(-4s_{2+3} + 11c_{2+3} + 17s_2) \\ -9s_1s_{2+3}s_4s_5 - 9s_1c_{2+3}c_5 - s_1(-4s_{2+3} + 11c_{2+3} + 17s_2) \\ -9c_{2+3}s_4s_5 - 9s_{2+3}c_5 - 11s_{2+3} - 4c_{2+3} \end{bmatrix}$$

$$f_4 = \begin{bmatrix} -9s_1s_4s_5 + 9c_1c_{2+3}c_4s_5 \\ 9c_1s_4s_5 + 9s_1c_{2+3}c_4s_5 \\ -9s_{2+3}c_4s_5 \end{bmatrix}$$

$$f_5 = \begin{bmatrix} 9s_1c_4c_5 + 9c_1c_{2+3}s_4c_5 + 9c_1s_{2+3}s_5 \\ -9c_1c_4c_5 + 9s_1c_{2+3}s_4c_5 + 9s_1s_{2+3}s_5 \\ -9s_{2+3}s_4c_5 - 9c_{2+3}s_5 \end{bmatrix}$$

where $s_i = \sin(\theta_i)$ and $c_i = \cos(\theta_i)$ for $i = 1, \ldots, 5$

2.2 MATLAB Jacobian Function

A listing of the Jacobian function (12) is available in Listing C. This concludes the answer to Question 2.

3 Question 3 - Bonus Question

The Jacobian matrix plays a crucial role in solving the inverse kinematics problem by relating joint velocities to end-effector linear and angular velocities.

Step 1 - Defining the Problem

Given:

- $\boldsymbol{\theta}$: Current joint positions (from encoders)
- $\mathbf{q}$: Current end-effector position and orientation
- $\mathbf{q}_d$: Desired target position and orientation of the end-effector

- $J(\boldsymbol{\theta})$: Jacobian matrix as a function of joint values

The objective is to find the joint velocities $\dot{\boldsymbol{\theta}}$ that move the end-effector towards the target position and orientation.

Step 2 - Define the error

The error is calculated in position and orientation between the current and desired end-effector states:

$$\mathbf{e} = \mathbf{q}_d - \mathbf{q}$$

Step 3 - Determine Joint Velocities

Use the Jacobian to determine the joint velocities needed to reduce the error. A simple control law can be use, that is

$$\dot{\boldsymbol{\theta}} = J^{\dagger}(\boldsymbol{\theta}) \cdot \mathbf{v}$$

where:

- $J^{\dagger}(\boldsymbol{\theta})$ is the pseudo-inverse of the Jacobian matrix.
- $\mathbf{v}$ is a velocity vector derived from the error.

Step 4 - Joint Positions updates

Integrate joint velocities to acquire the joint positions:

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta}_{\text{old}} + \dot{\boldsymbol{\theta}} \cdot \Delta t$$

where Δt is the time sampling used.

Step 5 - Iterate

Repeat Steps 2-4 or until the error $\mathbf{e}$ is minimized or until the end-effector reaches the desired position and orientation,.

Possible Problems

Singularity

The Jacobian loses rank (determinant approaches zero), leading to infinite joint velocities.

Local Minima

The solution may get stuck in a local minimum and not reach the global optimum.

Joint Limits:

Problem: The solution may require joint angles that are outside their joint rotation or translation limits.

Redundancy:

Problem: The robot may have more DOFs than needed (redundant), leading to infinite possible solutions.

References

[1] Craig, J. (2021). Introduction to Robotics, Global Edition. United Kingdom: Pearson Education Limited.

Appendix

A. AX-18A frame assignment

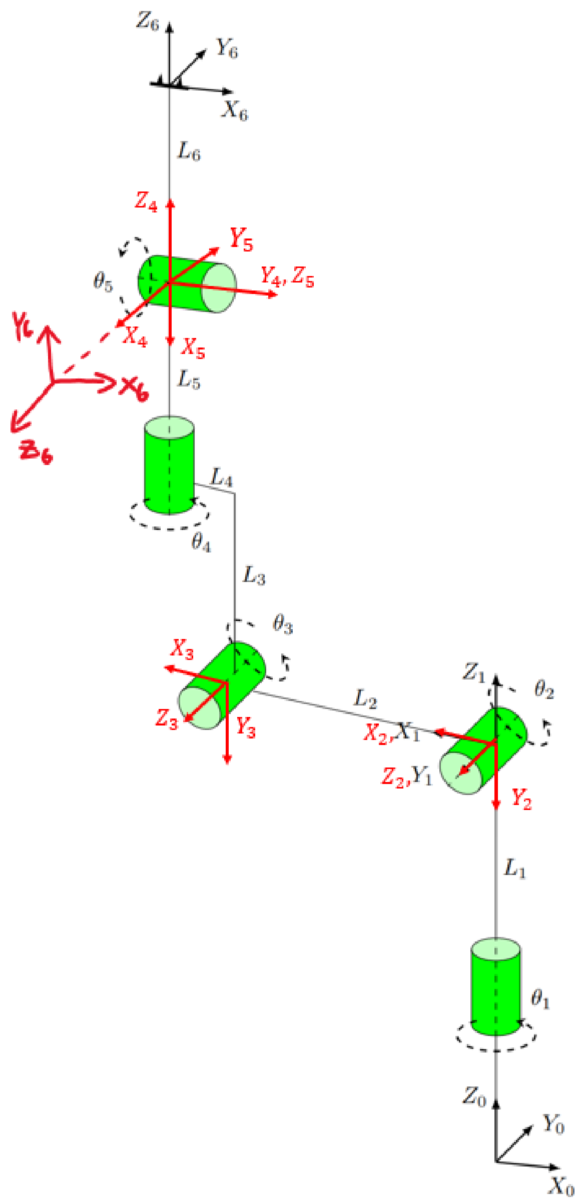

Figure 2: Frame assignment on AX-18A Smart Robotic

3.1 B. Listing for fk.m

```
function [q] = ik(p_Ex,p_Ey,p_Ez)
%% ik
%INPUT: coordinates of the end effector position in
    expressed in the 0
%frame NOTICE: it is up to the user to provide a
    reachable position
%           - p_Ex : 1x1
%           - p_Ey : 1x1
%           - p_Ez : 1x1
%OUTPUT:    - theta1 [rad]: angle joint 1
%           - theta2 [rad]: angle joint 2
%           - theta3 [rad]: angle joint 3
%           - theta4 [rad]: angle joint 4
%           - theta5 [rad]: angle joint 5

% For this question, DO NOT use the robotics toolbox
    ! You should use the
% function for the inverse kinematics computed as a
    solution of the
% theory in the case in which thta_4=0, theta_5 = pi
    /2

% link lengths
L1 = 17; L2 = 17; L3 = 7;
L4 = 4; L5 = 4; L6 = 9;

%%%%%% Write your code below this line. DO NOT
    change other parts. %%%%%%

% Inside of this function, write down how you
    calculate the output from
% the input. This is the place that has to be done
    by youself by using
% the results you get in the previous subquestions.
%shorter notation

px = p_Ex;
py = p_Ey;
```

```
pz = p_Ez;

% Theta 3
p3 = sqrt((L3+L5)^2 + L4^2 + L6^2);
q3 = sqrt(px^2 + (pz-L1)^2 + py^2);
beta3 = atan2(L4,(L3+L5));
c3b = (p3^2 + L2^2 - q3^2)/(2*p3*L2);
s3b = -sqrt(1-c3b^2);
theta3 = atan2(s3b, c3b) - beta3;

% Theta 2
k1 = 4*cos(theta3) + 11*sin(theta3);
k2 = -4*sin(theta3) + 11*cos(theta3) + 17;
b = [sqrt(px^2 + py^2 - 81); (pz-L1)];
K = [k1 k2; k2 -k1];
c = (K^-1)*b;
theta2 = atan2(c(2), c(1));

% Theta 1
if theta2 < 0 % theta_1 for theta_2 < 0
    a1_1 = sqrt(px^2+py^2-L6^2);
    beta1 = atan2(L6,a1_1);
    beta2 = atan2(py,px);
    theta1 = beta1+beta2;
elseif theta2 > 0 % theta_1 for theta_2 > 0
    a1_1 = sqrt(px^2+py^2);
    beta1 = atan2(L6,sqrt(a1_1^2-L6^2));
    beta2 = atan2(py,px);
    theta1 = beta2-beta1;
end

% Constrained joints
theta4 = 0;
theta5 = pi/2;

q = [theta1, theta2, theta3, theta4, theta5];
%%%%%% Write your code above this line. DO NOT
   change other parts. %%%%%%

end
```

C. Listing for jacob.m

```
function J0v = jacob(q)
%INPUT: joint position of the robot.
%            - q : 5x1 [5x1 rad]
%OUTPUT:     - J0v [3x5]: Jacobian containing the map
    between end effectors
%linear velocities and joint velocities
% For this question, DO NOT use the robotics toolbox
   , you should derive answers yourself!
q1 = q(1);
q2 = q(2);
q3 = q(3);
q4 = q(4);
q5 = q(5);
c1 = cos(q1);
c2 = cos(q2);
c3 = cos(q3);
c4 = cos(q4);
c5 = cos(q5);
s1 = sin(q1);
s2 = sin(q2);
s3 = sin(q3);
s4 = sin(q4);
s5 = sin(q5);
c23 = cos(q2+q3);
s23 = sin(q2+q3);
f11 = 9*c1*c4*s5 - 9*s1*c23*s4*s5 + 9*s1*s23*c5 + s1
   *(4*c23 + 11*s23 + 17*s2);
f12 = 9*s1*c4*s5 + 9*c1*c23*s4*s5 - 9*c1*s23*c5 - c1
   *(4*c23 + 11*s23 + 17*s2);
f13 = 0;
f21 = -9*c1*s23*s4*s5 - 9*c1*c23*c5 - c1*(-4*s23 +
   11*c23 +17*c2);
f22 = -9*s1*s23*s4*s5 - 9*s1*c23*c5 - s1*(-4*s23 +
   11*c23 +17*c2);
f23 = -9*c23*s4*s5 - 9*s23*c5 -17*s2 - 11*s23 - 4*
   c23;
f31 = -9*c1*s23*s4*s5 - 9*c1*c23*c5 - c1*(-4*s23 +
   11*c23 + 17*s2);
```

```
f32 = -9*s1*s23*s4*s5 - 9*s1*c23*c5 - s1*(-4*s23 +
   11*c23 + 17*s2);
f33 = -9*c23*s4*s5 - 9*s23*c5 - 11*s23 - 4*c23;
f41 = -9*s1*s4*s5 + 9*c1*c23*c4*s5;
f42 = 9*c1*s4*s5 + 9*s1*c23*c4*s5;
f43 = -9*s23*c4*s5;
f51 = 9*s1*c4*c5 + 9*c1*c23*s4*c5 + 9*c1*s23*s5;
f52 = -9*c1*c4*c5 + 9*s1*c23*s4*c5 + 9*s1*s23*s5;
f53 = -9*s23*s4*c5 - 9*c23*s5;

J0v = [f11 f21 f31 f41 f51;
    f12 f22 f32 f42 f52;
    f13 f23 f33 f43 f53];

end
```