**KOCAELİ ÜNİVERSİTESİ**

**MEKATRONİK MÜHENDİSLİĞİ BÖLÜMÜ**

A logo of a university

Description automatically generated

**Robotik ve Otomasyon Sistemleri Dönem Projesi 2023 2024**

**MOHAMED salah MOGAHED**

**190223087**

**Content**

* **Introduction**
* **KUKA KR8 R 1420 HW definition**
* **DH table parameters**
* **Robot CAD design**
* **Matlab App designer**
* **Transformation matrix**
* **Jacobian matrix**
* **Robot visualization**
* **Simscape Simulink**
* **Inverse kinematics**
* **Matlab codes**

**Introduction**

Robots are machines that we often use in our daily lives. Especially the number of industrial robots is increasing day by day. In the current study, it is planned to create a UI (User Interface) with App designer MATLAB to calculate transformation matrix, Jacobian matrix and calculate Inverse kinematics with FABRIK (Forward And Back Reaching Inverse Kinematics) and CCD (Cyclic Coordinate Descent) methods. Firstly, Denavit-Hartenberg (DH) parameters of KUKA KR8-R 1420 HW manipulator were found. Forward kinematic of the robot was calculated by using DH parameters. After getting DH table, it is also planned to create a visualization for KUKA KR8 R1420 HW.

**KUKA KR8-R 1420HW**

* Main assemblies of the manipulator

A diagram of a machine

Description automatically generated

1. Link arm
2. Hollow-shaft wrist/arm
3. Electrical installation
4. Base frame
5. Rotating column

* Technical Data

A table with text and numbers

Description automatically generated

* Axis Data KR 8 R1420 arc HW

A screenshot of a computer

Description automatically generated

A computer screen shot of a machine

Description automatically generated

**Denavit-Hartenberg table parameters**

A drawing of a mechanical arm

Description automatically generated**A drawing of a machine

Description automatically generated**

**A drawing of a machine

Description automatically generated**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
| **0** | **150 mm** | **450 mm** |  |
| **-π/2** | **0** | **0** | **-** |
| **0** | **610 mm** | **0** |  |
| **- π/2** | **200 mm** | **630 mm** |  |
| **- π/2** | **0** | **0** |  |
| **π/2** | **0** | **80 mm** |  |

* Transformation matrix

A group of black text

Description automatically generated



* **MATLAB**

Using app designer, calculating transformation matrix with Jacobian matrix and analyzing inverse kinematics in two methods forward and back reaching inverse kinematics method and cyclic coordinate descent method. Creating a plot visualization for the robot.

The dh table parameters can be entered by the user. Values used in current calculations to achieve our process: a= [2 0 6 2 0 0], α= [0 -90 0 -90 -90 90], d= [4.5 0 6 0 0 3] and θ= [60 90 0 150 120 0]. Determining the status of each link, Revolute or Prizmatic First tab “general” :

A screenshot of a computer

Description automatically generated

After getting dh table parameters and specifying if the link is revolute R, prismatic P or None, it is planned to calculate T (transformation) matrix and it will be shown a message that parameters are saved to the workspace.

A screenshot of a computer

Description automatically generated

By clicking button of calculate JacobianM and button of plot, it is planned to calculate Jacobian matrix and printing link positions with plotting robot visualization.

A screenshot of a computer

Description automatically generated

* Determining x y z positions and RPY angles

A screenshot of a computer

Description automatically generated

For the second tab

By clicking on load button, it is planned to load the entered dh parameters and link positions with KUKA KR8 R HW visualization. We use the values of alpha, a and d from DH table of KUKA KR8 RHW robot.

A screenshot of a computer

Description automatically generated

* Sim scape:

A computer screen shot of a machine

Description automatically generated

* Inverse kinematics with two methods, Forward And Back Reaching Inverse Kinematics) and CCD (Cyclic Coordinate Descent).

A screenshot of a computer

Description automatically generated

A diagram of a robot arm visualization method

Description automatically generated A diagram of a robot arm visualization

Description automatically generated

* **MATLAB codes**

**Function for dh matrix**

function T = dh\_matrix(alpha, a, d, theta, i, jointAngles)

% Compute the Denavit-Hartenberg matrix for joint i

T = [

cosd(theta(i) + jointAngles(i)), -sind(theta(i) + jointAngles(i)) \* cosd(alpha(i)), sind(theta(i) + jointAngles(i)) \* sind(alpha(i)), a(i) \* cosd(theta(i) + jointAngles(i));

sind(theta(i) + jointAngles(i)), cosd(theta(i) + jointAngles(i)) \* cosd(alpha(i)), -cosd(theta(i) + jointAngles(i)) \* sind(alpha(i)), a(i) \* sind(theta(i) + jointAngles(i));

0, sind(alpha(i)), cosd(alpha(i)), d(i);

0, 0, 0, 1

];

end

**---------------------------------------------------------------------------------------------------------------------**

**Function robot transformation matrix**

function T = robotTransform(DH)

% DH is a matrix with each row representing [theta, d, a, alpha]

numJoints = size(DH, 1);

T = eye(4); % Initialize the transformation matrix

for i = 1:numJoints

alpha = DH(i, 1);

a = DH(i, 2);

d = DH(i, 3);

theta = DH(i, 4);

% Modified DH transformation matrix

A = [cos(theta) -sin(theta)\*cos(alpha) sin(theta)\*sin(alpha) a\*cos(theta);

sin(theta) cos(theta)\*cos(alpha) -cos(theta)\*sin(alpha) a\*sin(theta);

0 sin(alpha) cos(alpha) d;

0 0 0 1];

% Update the overall transformation matrix

T = T \* A;

end

end

**---------------------------------------------------------------------------------------------------------------------**

**Transformation matrix**

syms alpha1 alpha2 alpha3 alpha4 alpha5 alpha6 real

syms a1 a2 a3 a4 a5 a6 real

syms d1 d2 d3 d4 d5 d6 real

syms theta1 theta2 theta3 theta4 theta5 theta6 real

% Define DH parameters as symbolic variables

% alpha a d theta

DH = [deg2rad(alpha1) a1 d1 deg2rad(theta1);

deg2rad(alpha2) a2 d2 deg2rad(theta2);

deg2rad(alpha3) a3 d3 deg2rad(theta3);

deg2rad(alpha4) a4 d4 deg2rad(theta4);

deg2rad(alpha5) a5 d5 deg2rad(theta5);

deg2rad(alpha6) a6 d6 deg2rad(theta6)];

% Call the robotTransform function

T = robotTransform(DH);

% Display the resulting transformation matrix

%disp('Transformation Matrix:');

disp(T);

**-----------------------------------------------------------------**

**Function to calculate Jacobian matrix**

function J = calculateJacobian(alpha, a, d, theta)

% Number of links

n = length(alpha);

% Initialize Jacobian matrix

J = zeros(6, n);

% Forward kinematics to calculate transformation matrices

T = eye(4);

for i = 1:n

A = [cosd(theta(i)) -sind(theta(i)) 0 a(i);

sind(theta(i))\*cosd(alpha(i)) cosd(theta(i))\*cosd(alpha(i)) -sind(alpha(i)) -sind(alpha(i))\*d(i);

sind(theta(i))\*sind(alpha(i)) cosd(theta(i))\*sind(alpha(i)) cosd(alpha(i)) cosd(alpha(i))\*d(i);

0 0 0 1];

T = T \* A;

% Extract rotational and translational components

R = T(1:3, 1:3);

p = T(1:3, 4);

% Populate the Jacobian matrix

J(:, i) = [cross(R(:, 3), p); R(:, 3)];

end

end

**-----------------------------------------------------------------**

**Robot visualization**

alpha = [0 -90 0 -90 -90 -90];

a = [8 0 6 6 0 0];

d = [6 0 6 0 4 7];

theta = deg2rad([60 90 0 150 120 0]);

H0\_1 = Link([theta(1), d(1), a(1), alpha(1), 1]);

H0\_1.qlim = [0 5];

H1\_2 = Link([theta(2), d(2), a(2), alpha(2), 0]);

H1\_2.qlim = [-pi/2 pi/2];

H2\_3 = Link([theta(3), d(3), a(3), alpha(3), 0]);

H2\_3.qlim = [-pi/2 pi/2];

H3\_4 = Link([theta(4), d(4), a(4), alpha(4), 0]);

H3\_4.qlim = [-pi/2 pi/2];

H4\_5 = Link([theta(5), d(5), a(5), alpha(5), 0]);

H4\_5.qlim = [-pi/2 pi/2];

H5\_6 = Link([theta(6), d(6), a(6), alpha(6), 0]);

H5\_6.qlim = [-pi/2 pi/2];

Kukarobot = SerialLink([H0\_1 H1\_2 H2\_3 H3\_4 H4\_5 H5\_6], 'name', 'KUKA KR8 R 1420 HW');

Kukarobot.plot([0 0 0 0 0 0], 'workspace', [-5 18 -18 18 0 18]);

Kukarobot.teach;

**-----------------------------------------------------------------**

* **Inverse kinematics**

**Fabrik inverse kinematics**

function jointAngles = fabrik\_inverse\_kinematics(alpha, a, d, theta, initialPosition, targetPosition)

% FABRIK Inverse Kinematics Algorithm

% Number of joints

n = length(alpha);

% Initial guess for joint angles

jointAngles = zeros(1, n);

% Maximum iteration count and tolerance

maxIterations = 100;

tolerance = 1e-6;

% Target position in homogeneous coordinates

targetPosHomogeneous = [targetPosition, 1];

% Iterate between forward and backward passes

for iter = 1:maxIterations

% Forward pass

for i = 1:n

T\_i = dh\_matrix(alpha, a, d, theta, i, jointAngles);

targetPosHomogeneous = T\_i .\* targetPosHomogeneous;

end

% Backward pass

for i = n:-1:1

T\_i = dh\_matrix(alpha, a, d, theta, i, jointAngles);

p\_i = T\_i(1:3, 4);

q\_i = targetPosition - p\_i';

q\_i = q\_i / norm(q\_i);

jointAngles(i) = atan2d(q\_i(2), q\_i(1));

end

% Check convergence

if norm(targetPosHomogeneous(1:3) - targetPosition) < tolerance

break;

end

end

end

**----------------------------------------------------------------------**

**Visualization of robot inverse**

function visualize\_robot\_inverse(alpha, a, d, theta, jointAngles, axesHandle)

% Visualize the robot arm with rotation matrices in the specified axes

% Number of joints

n = length(alpha);

% Clear the specified axes

cla(axesHandle);

% Plot robot links and display rotation matrices

for i = 1:n

T\_i = dh\_matrix(alpha, a, d, theta, i, jointAngles);

plot\_link(T\_i, axesHandle);

% Display the rotation matrix

disp(['Rotation Matrix from Link ' num2str(i-1) ' to Link ' num2str(i)]);

disp(T\_i(1:3, 1:3));

end

% Plot end-effector point

endEffectorPos = forward\_kinematics(alpha, a, d, theta, jointAngles);

scatter3(axesHandle, endEffectorPos(1), endEffectorPos(2), endEffectorPos(3), 'filled', 'r', 'DisplayName', 'End-Effector');

% Show legend

legend(axesHandle, 'show');

end

**----------------------------------------------------------------------**

**Plot link**

function plot\_link(T, axesHandle)

% Plot a link given its homogeneous transformation matrix

% Define the link frame

frame = [

0, 0, 0, 1;

1, 0, 0, 1;

1, 1, 0, 1;

0, 1, 0, 1;

0, 0, 0, 1

];

% Transform the link frame

frameTransformed = T \* frame';

% Plot the link in the specified axes

plot3(axesHandle, frameTransformed(1, :), frameTransformed(2, :), frameTransformed(3, :), 'b', 'LineWidth', 2, 'DisplayName', 'Link');

end

**----------------------------------------------------------------------**

**CCD inverse kinematics**

function jointAngles = ccd\_inverse\_kinematics(alpha, a, d, theta, initialPosition, targetPosition)

% CCD Inverse Kinematics Algorithm

% Number of joints

n = length(alpha);

% Check if the robot arm has the correct number of joints

if length(a) ~= n || length(d) ~= n || length(theta) ~= n || length(alpha) ~= n

error('Invalid robot arm configuration: The number of elements in alpha, a, d, and theta must be equal to the number of joints.');

end

% Check if initial and target positions are 3D vectors

if numel(initialPosition) ~= 3 || numel(targetPosition) ~= 3

error('Invalid input: initialPosition and targetPosition must be 3D vectors.');

end

% Random initialization for joint angles

jointAngles = rand(1, n) \* 360 - 180;

% Maximum iteration count and tolerance

maxIterations = 100;

tolerance = 1e-6;

% Iterate through CCD iterations

for iter = 1:maxIterations

% Iterate through joints

for i = 1:n

% Calculate the end-effector position for the current joint configuration

endEffectorPos = forward\_kinematics(alpha, a, d, theta, jointAngles);

% Calculate the vector from the current joint to the target position

targetVector = targetPosition - endEffectorPos;

% Calculate the vector from the current joint to the end-effector

currentVector = forward\_kinematics(alpha, a, d, theta, jointAngles, i) - endEffectorPos;

% Calculate the rotation axis (cross product of the two vectors)

rotationAxis = cross(currentVector, targetVector);

% Normalize the rotation axis

rotationAxis = rotationAxis / norm(rotationAxis);

% Calculate the rotation angle (dot product of the two vectors)

rotationAngle = acosd(dot(currentVector, targetVector) / (norm(currentVector) \* norm(targetVector)));

% Update the joint angle

jointAngles(i) = jointAngles(i) + rotationAngle \* sign(rotationAxis(3));

% Check for NaN or Inf values in joint angles

if any(isnan(jointAngles)) || any(isinf(jointAngles))

error('Invalid joint angles. CCD method failed to converge.');

end

% Check convergence

if norm(targetPosition - endEffectorPos) < tolerance

return;

end

end

end

% If the function reaches this point, CCD did not converge

error('CCD method did not converge within the maximum number of iterations.');

end