MAE 594 ROBOTICS:II
Final Project

Prepared by:	Raj Shah	Nihar Patel	Sahil Kapahi
UBIT's:	50321016	50318506	50317075
Email:	rajkalpe@buffalo.edu	nihardil@buffalo.edu	sahilkap@buffalo.edu

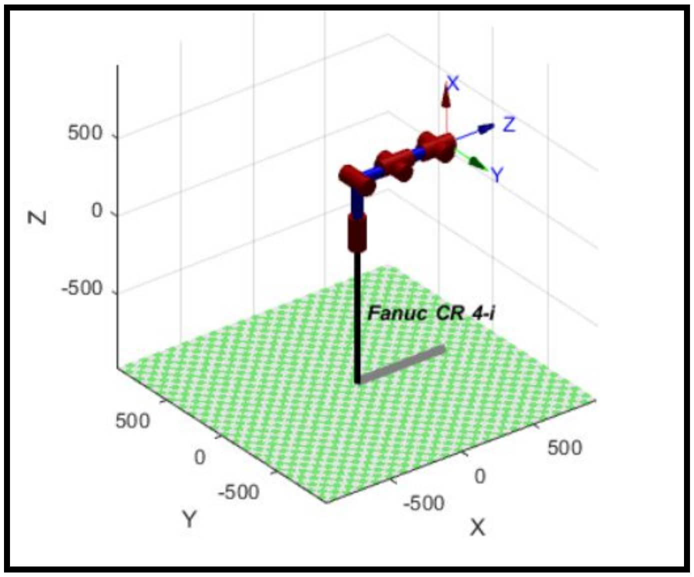
DH Analysis

As seen in the Fanuc CR 4iA datasheet, the robot consists of 6 joints thus it is a 6-DOF robot. Below Table shows the DH Table for Fanuc CR 4i:

No.	Theta	Alpha	A	D
1	θ_1	90	0	330
2	θ_2	0	260	0
3	θ_3	90	20	0
4	θ_4	-90	0	290
5	θ_5	90	0	0
6	θ_6	0	0	70

With the help of this DH table we calculated the transformation matrix of each joint with respect to the base frame

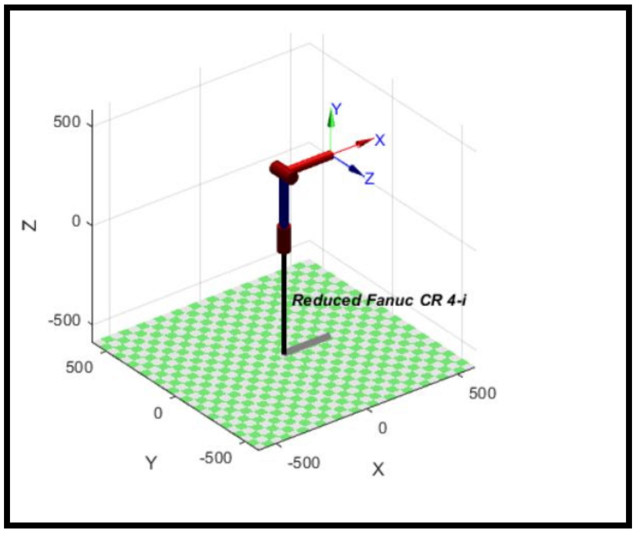
Below image shows the axis configuration of the robot:



Since solving DH, Jacobian, and Lagrange Equation of motion on matlab takes a long time to perform computation for a 6DOF system, we scaled down the project for the two joints of the existing robot. Thus we found new DH for reduced joint and it is shown below:

No.	Theta	alpha	a	d
1	θ_1	90	0	330
2	θ_2	0	260	0

Below is the updated axis configuration of the reduced joint robot:



Jacobian

Using the transformation matrices obtained from DH table, we calculated **Jacobian for 2 Links** in MATLAB:

```
JACCG1 = [cross([0 0 1]', CG1(1:3,4)-[0 0 0]');
          [0 0 1]'];
JACCG2 = [cross([0 0 1]', CG2(1:3,4)-[0 0 0]') cross(DH01(1:3,3), CG2(1:3,4)-DH01(1:3,4));
          [0 0 1]' DH01(1:3,3)];
```

Where,

DH01 = Transformation of Link 1 w.r.t Base

DH02 = Transformation of Link 2 w.r.t Base

JACCG1 = Jacobian of the center of gravity of Link 1 with respect to the base

JACCG2 = Jacobian of the center of gravity of Link 2 with respect to the base

All the Jacobians are found with respect to the C.G of each link.

Once Jacobian are found, then from the left hand side of the equation we are able to get linear and angular velocities of the C.G. of each link and from that we can calculate Kinetic energy which is required for solving Forward Dynamics.

Forward Dynamics

In order to perform Forward Dynamics, we need to solve the equation of motions using the Euler-Lagrange equation of motion.

The Euler-Lagrange equation of motions says, if we have Kinetic Energy and Potential energy of the links of robot systems then we can solve the equation of motion.

Below is the formula:

$\boxed{\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_j}\right) - \frac{\partial T}{\partial q_j} + \frac{\partial V}{\partial q_j} = P_j}$ where 'T' is Kinetic Energy and 'V' is Potential Energy

This term $\boxed{\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}}\right)}$ in the above equation is calculated using following equation using chain rule

$$\boxed{\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_j}\right) = \sum_{i}^{3}\left\{\frac{\partial}{\partial q_i}\left(\frac{\partial T}{\partial \dot{q}_j}\right)\frac{dq_i}{dt} + \frac{\partial}{\partial \dot{q}_i}\left(\frac{\partial T}{\partial q_j}\right)\frac{d\dot{q}_i}{dt}\right\}}$$

Kinetics Energy is calculated using the equation 6.6.22 from the book shown below:

Where,

q1, q2 are the generalized coordinates. For our system, it is joint angular position.

m1 and m2 are the masses of each respective link 1 and link 2.

The links are assumed to be rectangular prism made of Aluminium.

Jvcl is the linear velocity and Jω is the angular velocities of C.G with respect to base frame

R is the Rotational matrix to ensure that velocities are expressed in the frame attached to the link.

Below equation shown is used to calculate potential energy of each link:

$$V = \int_{\mathbf{B}} \mathbf{g}^T \mathbf{r}\, dm = \mathbf{g}^T \int_{\mathbf{B}} \mathbf{r}\, dm = \mathbf{g}^T \mathbf{r}_c m$$

Where,

g = 9.8 (gravitational constant)

For solving Kinetic Energy and Potential Energy, we need masses of each link and inertia matrix for each link.
Thus, collecting dimensions of each link from the datasheet and assuming the material is Aluminium and the shape of the link is rectangular prism. We found the masses of each link to be:

M1 = 32.28Kg

M2 = 20.367Kg

Below is the implemented code in matlab:

```
m1 = 2710e-9*11913000;

m2 = 2710e-9*9386000;
```

And for the Inertia Matrix:

```
I1 = 1/12*m1*[0.190.^2+0.330.^2 0 0;

              0 0.190.^2+0.330.^2 0;

              0 0 0.190.^2+0.190.^2];

I2 = 1/12*m2*[0.190^2+0.190^2 0 0; 0 0.190^2+0.260^2 0; 0 0 0.190^2+0.260^2];
```

<u>Code to solve Kinetic Energy:</u>

```
%% Kinetic Energy

D1 = m1*(JACCG1(1:3,:))'*(JACCG1(1:3,:))*qdot1 + (JACCG1(4:6,:))'*I1*omega1;

K_E1 = 0.5 * qdot1 * D1

D2 = m2*(JACCG2(1:3,:))'*(JACCG2(1:3,:))*[qdot1;qdot2] +
(JACCG2(4:6,:))'*DH01(1:3,1:3)*I2*omega2;

K_E2 = 0.5 * [qdot1; qdot2]' * D2;
```

<u>Code to solve potential Energy:</u>

```
%% Potential Energy

P_E1 = [0 0 9.800]*CG1(1:3,4)*m1; % Potential Enegry for link 1

P_E2 = [0 0 9.800]*CG2(1:3,4)*m2; % Potential Enegry for link 2
```

Once obtained all the kinetic and potential energies, we will find the lagrangian equation of motion using above equation in Matlab:

<u>Code to solve Lagrange:</u>

```
%% Lagrangian

%L = (K_E1 + K_E2) - (P_E1 + P_E2) % Lagrangian

K_E = K_E1 + K_E2;

P_E = P_E1 + P_E2;
```

Equation of Motion

```
KEwrtqdot1 = diff(K_E,qdot1);

KEwrtqdot2 = diff(K_E,qdot2);

KEwrttheta1 = diff(K_E,theta1);

KEwrttheta2 = diff(K_E,theta2);

KEwrtqtt1 = diff(KEwrtqdot1,theta1)*qdot1 + diff(KEwrttheta1,qdot1)*qddot1 +
diff(KEwrtqdot1,theta2)*qdot2 + diff(KEwrttheta1,qdot2)*qddot2

KEwrtqtt2 = diff(KEwrtqdot2,theta1)*qdot1 + diff(KEwrttheta2,qdot1)*qddot1 +
diff(KEwrtqdot2,theta2)*qdot2 + diff(KEwrttheta2,qdot2)*qddot2

PEwrttheta1 = diff(P_E,theta1);

PEwrttheta2 = diff(P_E,theta2);
```

Below code shows our implementation of solving differential equation of motion and also the given arbitrary torque vector input:

```
%%

syms Tau1 Tau2


eom1 = simplify(Tau1 - (KEwrtqtt1 - KEwrttheta1 + PEwrttheta1))

eom2 = simplify(Tau2 - (KEwrtqtt2 - KEwrttheta2 + PEwrttheta2))


Sol = solve(eom1,eom2,qddot1,qddot2);

Sol.qddot1 = simplify(Sol.qddot1);

Sol.qddot2 = simplify(Sol.qddot2);
```

syms y1 y2 y3 y4 y5 y6

fx1=subs(Sol.qddot1,[theta1,qdot1,theta2,qdot2],[y1,y2,y3,y4]);

fx2=subs(Sol.qddot2,[theta1,qdot1,theta2,qdot2],[y1,y2,y3,y4]);

After getting the equations of motions, we obtained a solution to differential equations of motion for an arbitrary and given vector torque input using Runge-Kutta 4th order integrator. **This can be seen in MATLAB file: forward_dynamics.m and video:Forward Dynamics.mp4**. Angular position, angular velocities and accelerations for each joint is recorded in the file "**4_results.mat**" where first column is angular position for joint 1, column 2 is the angular velocity of joint 1, column 3 is the angular position of joint 2, and column 4 is the angular velocity of the joint 2.

Inverse Kinematics

The desired trajectory is given as input using Matlab code to be followed by a 2-DOF manipulator arm. A segment of sphere was given as input trajectory to the manipulator and an iterative solution was calculated for each time step. Using DH-parameters as calculated above, we defined an inverse of the jacobian to approximate the angular positions at each time-step using the current state of the trajectory. The solution was visualized on Matlab and can be seen in the attached video(Inverse_Kinematics.mp4). (Matlab file: inverse_kinematics.m)

PD Controller

After we obtained reference joint angles using inverse kinematics, we designed and implemented a PD controller by giving the angular positions obtained in step 6 as reference. This can be seen in MATLAB file "controller.m"