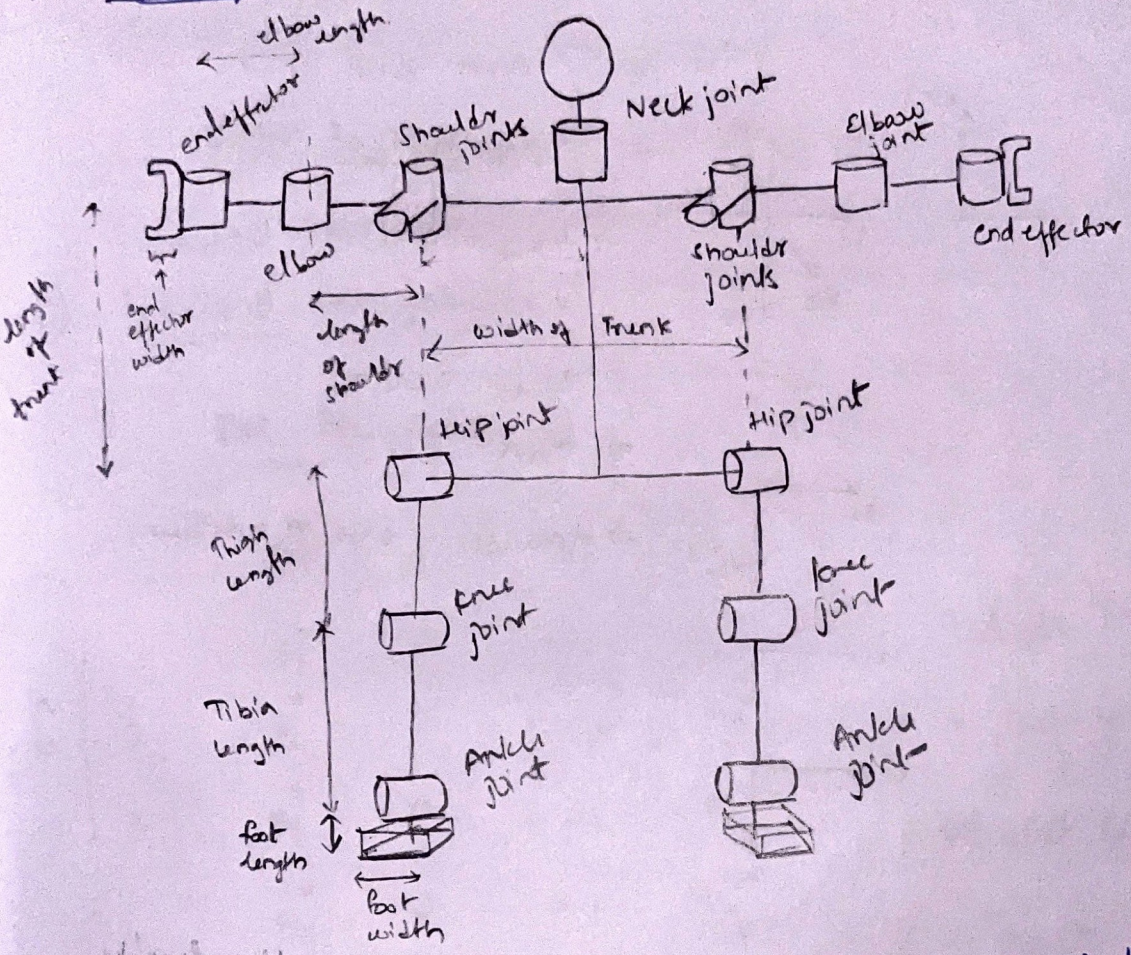
ENPM 667 Final Exam

Name: Bharadwaj Ch.
UID: 118341705
mail: bchukkal@umd.edu

a) Sketch of the Robot



* For this exam, we will only consider one half of the robot to perform the task for simplicity.

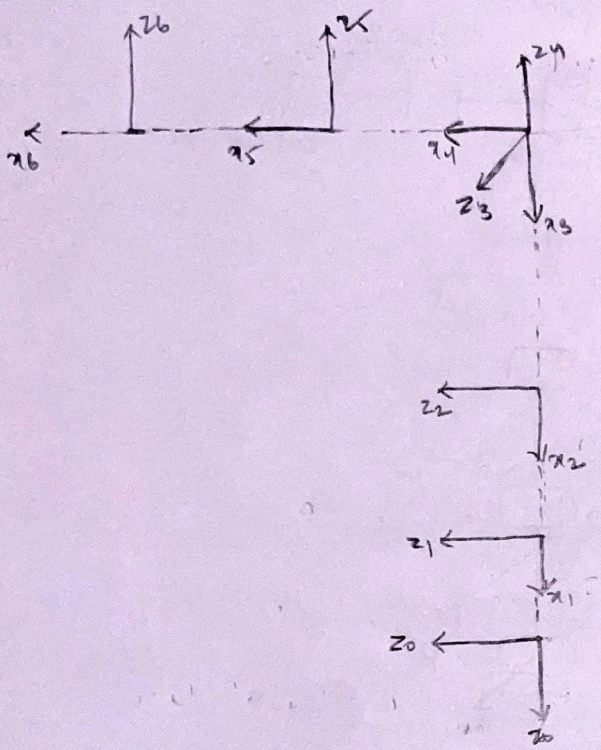
* When we take this robot in such way, it becomes into a serial manipulator essentially. (The structure)

* The Body is made of Aluminium

$$\sigma\,(\text{density})_{Al} = 2710\,kg/m^3$$

9 Co-ordinate Systems of the Robot

b) * considering only one half of robot
* The head chain is not being used, so it is excluded for avoiding confusion.



Left half of robot.

⇒ $L_{thigh} = 0.465m = L_{Tibia}$
* assuming same length

⇒ $L_{foot} = 0.06m$

⇒ $w_{foot} = 0.26m$

⇒ $d_{shoulder} = 0.36m = d_{elbow}$

⇒ Now to get masses of the links, we can use the density of the aluminium and assume our links to be cylindrical with certain radiuses. We can calculate the masses.

⇒ I have chosen radius of shoulder arm to be less than that of leg, because for the robot to be stable, the weight needs to more in the lower body.

$mass = \sigma \times V$

V :- Cylindrical volume of Link

⇒ The masses we get after calculation are:

Shoulder link mass = 11.3 kg

Thigh Link mass = 20.5 kg

Trunk mass = 166.93 kg

Foot Link mass = 11 kg

Neck link mass = 3 kg

mass of motors = 1 lb ≃ 0.45 kg

c) Forward Kinematics

DH Parameters

	i	θ_i	d_i	a_i	α_i
arm	1	θ_1^*	0	$-a_1$	0
	2	θ_2^*	0	$-a_2$	0
	3	θ_3^*	0	$-a_3$	$-\pi/2$
leg	4	θ_4^*	0	a_4	$\pi/2$
	5	θ_5^*	0	a_5	0
	6	θ_6^*	0	a_6	0
	7	θ_7^*	0	a_7	0

$\theta \rightarrow$ Angle

$\alpha \rightarrow$ Link twist

$a \rightarrow$ Link Lengths

$d_i \rightarrow$ offset

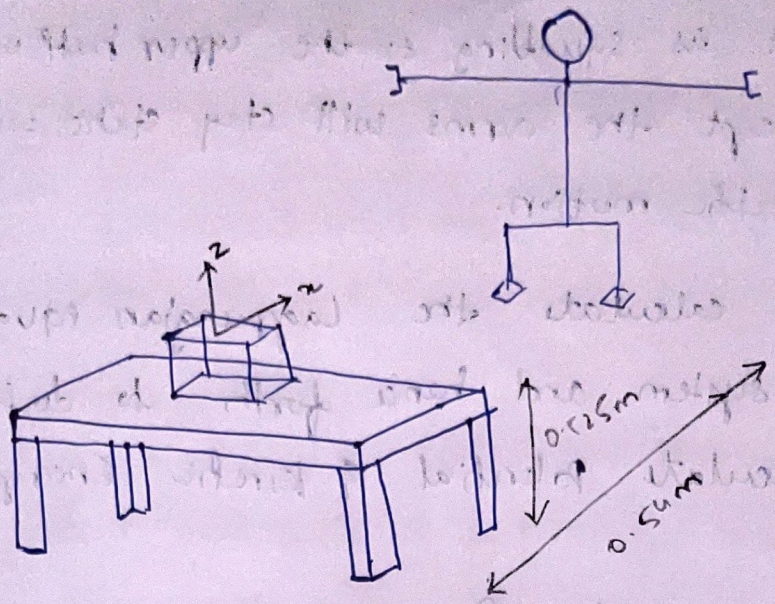
* The robot is considered as a serial chained manipulator, hence we are writing it like this

Inverse Kinematics

* For this problem, I have used Inverse velocity kinematics approach.
* We can design a pick and place scenario such that we know the object co-ordinates
* Now when we know the object coordinates and we can find the end effector coordinates using Forward Kinematics
* Using Error between the end effector frame and the object to be picked, we can define the velocity component that is used to traverse the trajectory
* we can use velocity component in the planar state to guide the robot link angles to move appropriately using $q = J^{-1}V$ and find the joint velocities.
* These joint velocities will inturn help us to update the q values for the robot to move to the object.
* Essentially the error will become zero.

Pick and Place Environment

Note:- Not so good but Drawing

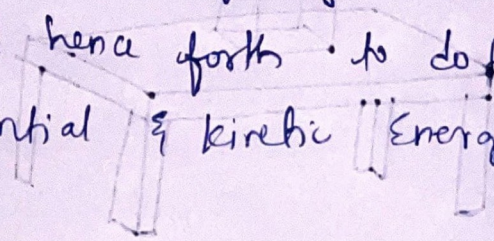


* The mass of the object is 20 lbs ≃ 9 kgs
* The object is placed on a box which is at knee length (given assumption in problem)
* We are creating a scenario such that the box and object are in the robot workspace.
* The robot will perform a bear forward or squat action to pick the object and handle it.
* Here we intend to perform a squat action and move the arm to the object for the end effector to pick the object.
* The scenario will be more clearly explained forther while discussing about Reach mode & Grasping / Handling mode.

d) Dynamic Equations

* The robot is squatting so the upper half of the body except the arms will stay static without any specific motion.

* We will calculate the Lagrangian equation of this system and hence forth. to do that we calculate potential & kinetic Energy



$$L = K - P$$

K → Kinetic Energy

P → potential Energy.

$$\tau_i = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i}$$

- w.k.t K is zero because the system is quasi-static $\dot{q}$ & $\ddot{q} \cong 0$ (the system is slow at an infinely small velocity).

$$\tau_i = \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right)^{0} - \frac{\partial L}{\partial q_i} \quad (L = P).$$

$$\tau_i = \frac{-\partial P}{\partial q_i}$$

* The potential Energy of the system can be calculated using the formula $P = mgh$

* we have masses and heights can be retrived from comparison with coordinate frames

* The P calculation is done in the Program. (Refer code at the end).

* The heights are calculated to the center of mass of each link rather than the ends.

Now when $\dot{q} \simeq 0 \simeq \ddot{q}$

From the eqn.

$$M\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau + J^T F_w$$

$$\therefore g(q) = \tau + J^T F_w \Rightarrow \boxed{\tau_i = g(q) - J^T F_w}$$

where F_w is the wrench vector which is the frictional force vector acting on the object that is being picked and placed.

$$F_w = \begin{bmatrix} f & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T$$

$f \rightarrow$ friction $\quad f = \mu mg \Rightarrow 0.45 \times \frac{9}{2} \times 9.8$

$$= 18.9\,N$$

* Here the μ is friction coefficient between the end effector and object.

Note that the object is made of cardboard.

* So the friction between Al and cardboard is $\mu = 0.45$

* We have taken half the weight of the cube because, we are considering half plane of robot to lift the object to avoid closed loop.

* We have written down the τ_i eqn successfully so now I have coded in the program, an algorithm to calculate the joint Torques

Now to find the max torques.

$$\tau_i = g(q) - J^T(q) F_w$$

⇒ using matrix calculator we can solve for each joint and calculate max torque.

First we differentiate the $\tau_i = g(q_i) - J^T(q_i) F_w$ to get $\dot{\tau}_i$ we equate it zero to get $\dot{\tau}_i = 0$. When the τ_i is differentiated wrt to each joint angle q_i and substituted back into τ_i, we get max torque: τ_{max}

* To find max torque for arms only.

$$\frac{\partial \tau_i}{\partial q_1} \Rightarrow \frac{\partial \tau_1}{\partial q_1}, \frac{\partial \tau_2}{\partial q_2}, \frac{\partial \tau_3}{\partial q_3}$$

we get $\tau_1 = -140 \cos q_1 = 0 \quad (q_1 = \pi/2)$

$\tau_2 = -703 \sin q_2 = 0 \quad (q_2 = 0)(q_1 = \pi/2)$

$\tau_3 = 0.635 \sin q_3 = 0 \quad (q_3 = 0)(q_2 = 0)$

The max torques after substitution are $(q_1 = \pi/2)$

$$\tau_1 = 140 \text{ Nm} \quad \tau_2 = -703 \text{ Nm} \quad \tau_3 = -0.64 \text{ Nm}$$

e) Pick and Place Scenario:

- Reach mode
- Grasping / Handling mode.

* We already found out the favourable location of object inside the workspace of the robot

⇒ Reach mode:-

* we know the coordinate frame of the object w.r.t to base frame of robot

so we choose a favourable location with $x = -0.465\,m$

$y = 0.539\,m$

$z = 0$ (on the table)

$z_{table} = 0.525\,m$

* End effector transformation matrix ${}^{0}_{7}T$ is known

* We calculate the error between the frames by simple negation and take a certain time Δt to calculate the planar velocity for the trajectorial motion.

$$\dot{x} = \frac{x_{obj} - x_{end}}{\Delta t} \qquad \dot{y} = \frac{y_{obj} - y_{end}}{\Delta t} \qquad \dot{z} = \frac{z_{obj} - z_{end}}{\Delta t}$$

* This is similar to differentiation. to make error zero.

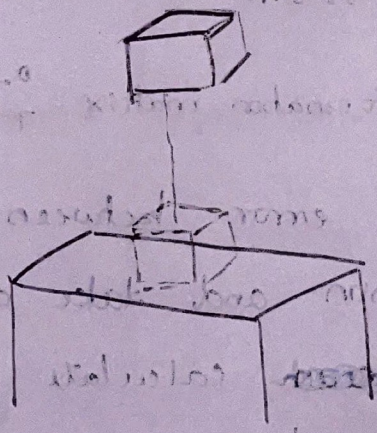
Now these velocities $\dot{x}, \dot{y}, \dot{z}$ are used to get the values of angular velocities

$$\dot{q} = \bar{J}^{-1} V \quad \leftarrow \text{Inverse velocity Kinematics.}$$

⇒ Grasping / Handling mode

* we take a scenario where the robot lifts the object along x axis and places it back in the original position.

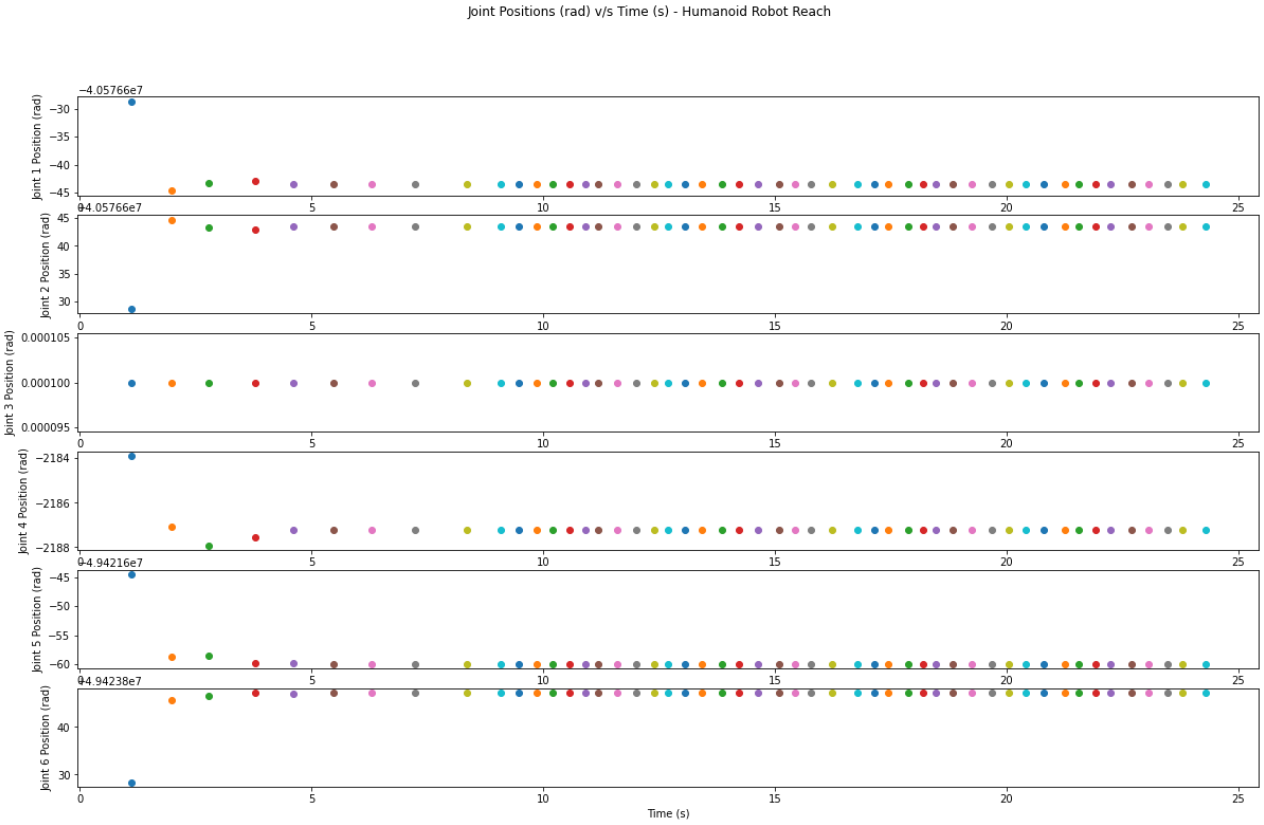
* So we consider velocity component $\dot{x}$ only in the grasping mode to lift the object $\dot{y}$ and $\dot{z}$ will be kept static.



* The Plots are given at the end.

The Output Plots for Reach and Grasping modes:

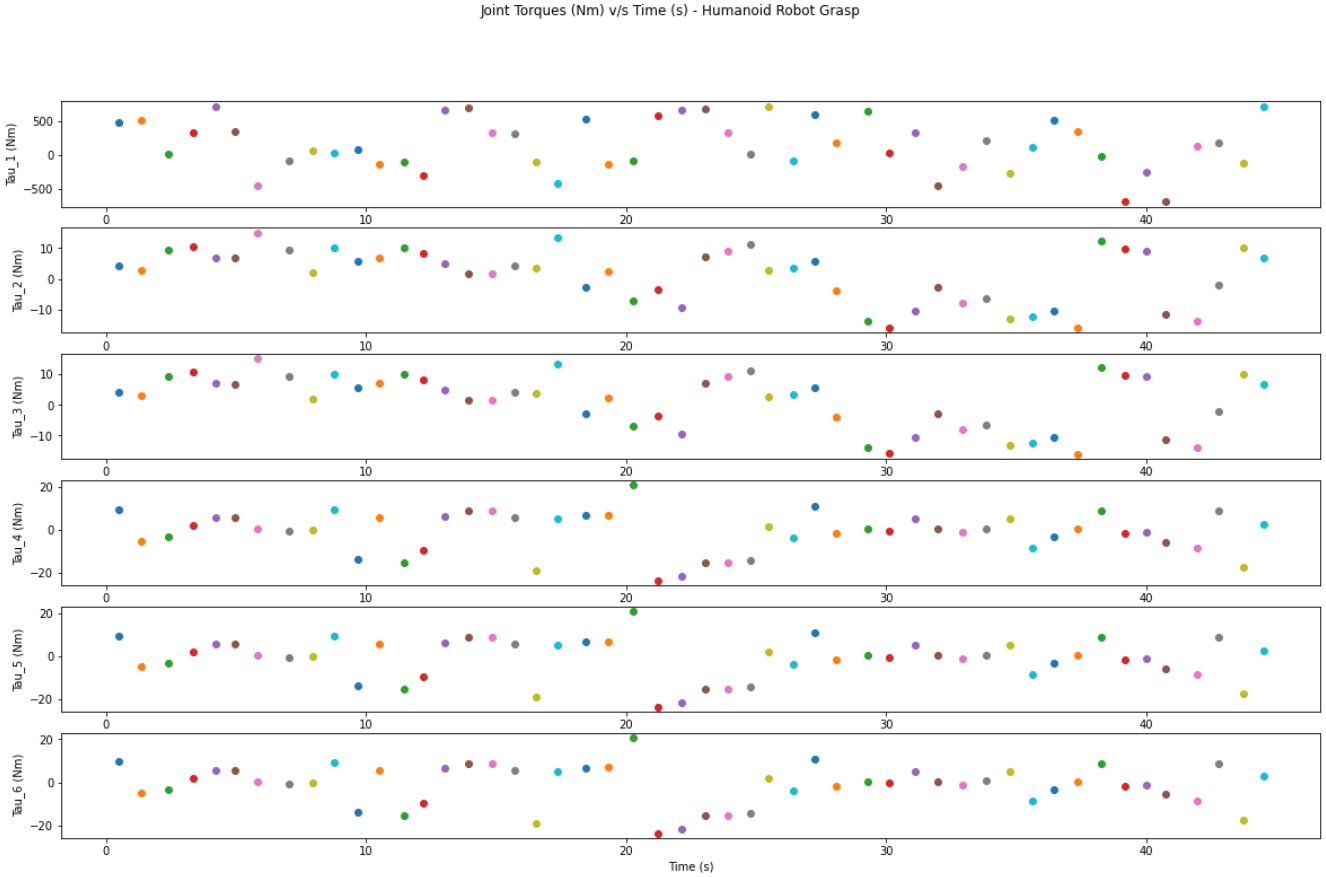
1. Joint positions for reach mode:



2. Joint Torques of Reach model

3. Joint Positions of Grasping Mode

4. Joint torques for Grasping mode



The Code for the Simulation result:

```
# -*- coding: utf-8 -*-
"""Bharadwaj Chukkala Final Exam.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1sc6B7c1AndrUYo7tdotMOmsfKnqn9KVO

## **Importing necessary Libraries**
"""

from sympy import Matrix, symbols, cos, sin, simplify, pi, pprint, diff, solve, Eq
from numpy import linspace
import matplotlib.pyplot as plt
import time

"""## **Masses and Link Lengths**"""

#
L_eff = 0.1    # Length of end-effector (m)
L_cube = 0.525

#
N = 20              # no. of data points
delta_time = 20/N     # time between successive data points

# Humanoid dimensions # (m)
L_thigh = 0.465
L_tibia = 0.465
r_thigh = 0.072
r_tibia = 0.072
w_foot = 0.26
L_foot = 0.06
L_shoulder = 0.3813
L_elbow = 0.3813
r_shoulder = 0.059
r_elbow = 0.059
L_trunk = 0.395
w_trunk = 0.24

# Masses (kg)
m_motor = 0.45
m_shoulder = 11.29
m_elbow = 11.29
m_thigh = 20.522
m_tibia = 20.522
m_trunk = 166.93
```

```
m_foot = 10.991
m_neck = 2.964
m_head = 47.425

# Defining the symbolic joint angle symbolic variables
q1, q2, q3, q4, q5, q6 = symbols('q1, q2, q3, q4, q5, q6')

# 'a' DH paraemeters
a1 = 0.465 # Calf
a2 = 0.465 # Thigh
a3 = 0.336 # Difference between trunk & shoulder
a4 = 0
a5 = 0.3818 # shoulder
a6 = 0.3818 # elbow
a7 = 0.1 # end effector

"""## **Computing the Transformation Matrices**"""

def tf(q,d,a,alpha):
    T = Matrix([[cos(q),-sin(q)*cos(alpha),sin(q)*sin(alpha),a*cos(q)], [sin(q),cos(q)*cos(alpha),-
cos(q)*sin(alpha),a*sin(q)], [0,sin(alpha),cos(alpha),d], [0,0,0,1]])
    return T

A01 = tf(q1,0,-a1,0)
A12 = tf(q2,0,-a2,0)
A23 = tf(q3,0,-a3,-pi/2)
A34 = tf(q4-(pi/2),0,0,pi/2)
A45 = tf(q5,0,a5,0)
A56 = tf(q6,0,a6,0)
A6n = tf(0,0,a7,0)

A02 = A01 * A12
A03 = A02 * A23
A04 = A03 * A34
A05 = A04 * A45
A06 = A05 * A56
A0n = A06 * A6n

"""## **Computing Jacobian by finding Z and O matrices**
Using Jacobian Differentiation Method to Compute J
"""

Z1 = simplify(A01[0:3,2])
Z2 = simplify(A02[0:3,2])
Z3 = simplify(A03[0:3,2])
Z4 = simplify(A04[0:3,2])
Z5 = simplify(A05[0:3,2])
Z6 = simplify(A06[0:3,2])
```

```
px = A0n[0,3]; py = A0n[1,3]; pz = A0n[2,3];
a11 = diff(px, q1)
a12 = diff(px, q2)
a13 = diff(px, q3)
a14 = diff(px, q4)
a15 = diff(px, q5)
a16 = diff(px, q6)

a21 = diff(py, q1)
a22 = diff(py, q2)
a23 = diff(py, q3)
a24 = diff(py, q4)
a25 = diff(py, q5)
a26 = diff(py, q6)

a31 = diff(pz, q1)
a32 = diff(pz, q2)
a33 = diff(pz, q3)
a34 = diff(pz, q4)
a35 = diff(pz, q5)
a36 = diff(pz, q6)

J = Matrix([[a11, a12, a13, a14, a15, a16], [a21, a22, a23, a24, a25, a26],[a31, a32, a33, a34, a35, a36],
[Z1,Z2,Z3,Z4,Z5,Z6]]) # assemble into matix form
J = simplify(J) # use sympy simplification method to obtain simplified results
J

"""## **Calculating Lagrangian**"""

# Link Masses (kg)
m_motor = 0.45
m_shoulder = 11.29
m_elbow = 11.29
m_thigh = 20.522
m_shank = 20.522
m_trunk = 166.93
m_foot = 10.991
m_neck = 2.964
m_head = 47.425

g = 9.81

# Potential energy
h_trunk_com = L_foot + L_shank*cos(q1) + L_thigh*cos(q1+q2) + (L_trunk/2)

P_trunk = m_trunk*g*h_trunk_com
P_shoulder_motor = 2*m_motor*g*((L_trunk/2)+h_trunk_com)
P_shoulder = m_shoulder*g*((L_trunk/2) + h_trunk_com + (L_shoulder/2)*sin(q4)*cos(q5))
P_elbow_motor = m_motor*g*((L_trunk/2) + h_trunk_com - L_shoulder*sin(q4)*cos(q5))
```

```
P_elbow = m_elbow*g*(L_elbow/2)*cos(q5+q6)*((L_trunk/2) + h_trunk_com - L_shoulder*sin(q4)*cos(q5))
P_wrist_motor = m_motor*g*L_elbow*cos(q5+q6)*((L_trunk/2) + h_trunk_com - L_shoulder*sin(q4)*cos(q5))

PE = P_trunk + P_shoulder_motor - P_shoulder - P_elbow_motor - P_elbow - P_wrist_motor

# Kinetic energy of the system
KE = 0

# Lagrangian
L = KE - PE

g1 = -simplify(diff(L,q1))
g2 = -simplify(diff(L,q2))
g3 = -simplify(diff(L,q3))
g4 = -simplify(diff(L,q4))
g5 = -simplify(diff(L,q5))
g6 = -simplify(diff(L,q6))
Gravity = Matrix([g1, g2, g3, g4, g5, g6])

"""## **Wrench Vector**

Tells the Magnitude and Direction along which the force is acted upon
"""

Wrench = Matrix([18.89, 0, 0, 0, 0, 0]) # External wrench vector [Fx, Fy, Fz, Tx, Ty, Tz] on end-effector

"""## **Calculating Torque**"""

tau = Gravity - J.T * Wrench

def inverse_velocity_kinematics(X_dot, q_joint):

    J_q = J.subs([(q1, q_joint[0]), (q2, q_joint[1]), (q3, q_joint[2]), (q4, q_joint[3]), (q5, q_joint[4]), (q6, q_joint[5])])
    J_inv = J_q.inv()

    q_dot = J_inv * X_dot # Generalized joint velocity components from Jacobian inverse

    return q_dot, J_q


# Function to compute new value of joint angle based on q_dot and previous angle
def update_joint_angle(q, q_dot):
    q = q + q_dot*delta_time
    return q
```

```
# Function to compute forward kinematics (position) to obtain the end-effector (Pen) (x,y,z) co-ords wrt base frame
def forward_position_kinematics(q):
    T = A0n.subs([(q1, q[0]), (q2, q[1]), (q3, q[2]), (q4, q[3]), (q5, q[4]), (q6, q[5])])
    return (T[0,3].round(4),T[1,3].round(4),T[2,3].round(4))

"""## ** Grasping and Handling Mode**"""

def pick_place(q_joint):

    fig, ax = plt.subplots(6, figsize=(20, 12))
    fig2, ax2 = plt.subplots(6, figsize=(20, 12))
    fig.suptitle('Joint Positions (rad) v/s Time (s) - Humanoid Robot Grasp')
    fig2.suptitle('Joint Torques (Nm) v/s Time (s) - Humanoid Robot Grasp')

    start_time = time.time()

    print("\n******* Humanoid Grasp/Pick Mode start! *******\n")

    for i in range(0,N):

        x_dot = -0.5
        y_dot = 0
        z_dot = 0

        X_dot = Matrix([x_dot, y_dot, z_dot, 0.0, 0.0, 0.0])

        q_dot_joint = inverse_velocity_kinematics(X_dot, q_joint)

        tau_q = tau.evalf(3,subs={q1: q_joint[0],q2: q_joint[1], q3: q_joint[2], q4: q_joint[3], q5: q_joint[4], q6: q_joint[5]})

        q_joint = update_joint_angle(q_joint, q_dot_joint[0])

        (x_0p, y_0p, z_0p) = forward_position_kinematics(q_joint)
        print("Desired end-eff X, Y, Z co-ordinates")
        print(x_cube, y_cube, z_cube)
        print("Current end-eff X, Y, Z co-ordinates")
        print(x_0p, y_0p, z_0p)

        elapsed_time = time.time()-start_time

        ax[0].set_xlabel('Time (s)')
        ax[0].set_ylabel('Joint 1 Position (rad)')
        ax[0].scatter(elapsed_time, q_joint[0])

        ax2[0].set_xlabel('Time (s)')
        ax2[0].set_ylabel('Tau_1 (Nm)')
        ax2[0].scatter(elapsed_time, tau_q[0])
```

```
        ax[1].set_xlabel('Time (s)')
        ax[1].set_ylabel('Joint 2 Position (rad)')
        ax[1].scatter(elapsed_time, q_joint[1])

        ax2[1].set_xlabel('Time (s)')
        ax2[1].set_ylabel('Tau_2 (Nm)')
        ax2[1].scatter(elapsed_time, tau_q[1])

        ax[2].set_xlabel('Time (s)')
        ax[2].set_ylabel('Joint 3 Position (rad)')
        ax[2].scatter(elapsed_time, q_joint[2])

        ax2[2].set_xlabel('Time (s)')
        ax2[2].set_ylabel('Tau_3 (Nm)')
        ax2[2].scatter(elapsed_time, tau_q[2])

        ax[3].set_xlabel('Time (s)')
        ax[3].set_ylabel('Joint 4 Position (rad)')
        ax[3].scatter(elapsed_time, q_joint[3])

        ax2[3].set_xlabel('Time (s)')
        ax2[3].set_ylabel('Tau_4 (Nm)')
        ax2[3].scatter(elapsed_time, tau_q[3])

        ax[4].set_xlabel('Time (s)')
        ax[4].set_ylabel('Joint 5 Position (rad)')
        ax[4].scatter(elapsed_time, q_joint[4])

        ax2[4].set_xlabel('Time (s)')
        ax2[4].set_ylabel('Tau_5 (Nm)')
        ax2[4].scatter(elapsed_time, tau_q[3])

        ax[5].set_xlabel('Time (s)')
        ax[5].set_ylabel('Joint 6 Position (rad)')
        ax[5].scatter(elapsed_time, q_joint[5])

        ax2[5].set_xlabel('Time (s)')
        ax2[5].set_ylabel('Tau_6 (Nm)')
        ax2[5].scatter(elapsed_time, tau_q[3])

        # plt.pause(delta_time)

    print("\n******* Plotting completed! *******")
    plt.show()

# Setting up the Initial Pose
q_joint = Matrix([0.0, 0.0, 0.0001, 0.0001, 0.0, 0.001])
```

```
fig, ax = plt.subplots(6, figsize=(20, 12))
fig2, ax2 = plt.subplots(6, figsize=(20, 12))
fig.suptitle('Joint Positions (rad) v/s Time (s) - Humanoid Robot Reach')
fig2.suptitle('Joint Torques (Nm) v/s Time (s) - Humanoid Robot Reach')

# The object is considered to be Cube

N = 50
delta_time = 50/N

start_time = time.time()
for i in range(0,N):

    x_eff = A0n[0,3].subs([(q1, q_joint[0]), (q2, q_joint[1]), (q3, q_joint[2]), (q4, q_joint[3]), (q5,
q_joint[4]), (q6, q_joint[5])])
    y_eff = A0n[1,3].subs([(q1, q_joint[0]), (q2, q_joint[1]), (q3, q_joint[2]), (q4, q_joint[3]), (q5,
q_joint[4]), (q6, q_joint[5])])
    z_eff = A0n[2,3].subs([(q1, q_joint[0]), (q2, q_joint[1]), (q3, q_joint[2]), (q4, q_joint[3]), (q5,
q_joint[4]), (q6, q_joint[5])])

    x_cube = -(0.525 - 0.06)
    y_cube = 0.539
    z_cube = 0.0

    x_dot = -(x_eff-x_cube)/delta_time
    y_dot = -(y_eff-y_cube)/delta_time
    z_dot = -(z_eff-z_cube)/delta_time

    X_dot = Matrix([x_dot, y_dot, z_dot, 0.0, 0.0, 0.0])

    J_q = J.subs([(q1, q_joint[0]), (q2, q_joint[1]), (q3, q_joint[2]), (q4, q_joint[3]), (q5, q_joint[4]), (q6,
q_joint[5])])
    J_inv = J_q.inv()
    q_dot = J_inv * X_dot


    tau_q = Gravity.evalf(3,subs={q1: q_joint[0],q2: q_joint[1], q3: q_joint[2], q4: q_joint[3], q5:
q_joint[4], q6: q_joint[5]})
    q_joint = q_joint + q_dot * delta_time

    (x_0p, y_0p, z_0p) = forward_position_kinematics(q_joint)
    print("Desired end-eff X, Y, Z co-ordinates")
    print(x_cube, y_cube, z_cube)
    print("Current end-eff X, Y, Z co-ordinates")
    print(x_0p, y_0p, z_0p)

    elapsed_time = time.time()-start_time

    ax[0].set_xlabel('Time (s)')
```

```
    ax[0].set_ylabel('Joint 1 Position (rad)')
    ax[0].scatter(elapsed_time, q_joint[0])

    ax2[0].set_xlabel('Time (s)')
    ax2[0].set_ylabel('Tau_1 (Nm)')
    ax2[0].scatter(elapsed_time, tau_q[0])

    ax[1].set_xlabel('Time (s)')
    ax[1].set_ylabel('Joint 2 Position (rad)')
    ax[1].scatter(elapsed_time, q_joint[1])

    ax2[1].set_xlabel('Time (s)')
    ax2[1].set_ylabel('Tau_2 (Nm)')
    ax2[1].scatter(elapsed_time, tau_q[1])

    ax[2].set_xlabel('Time (s)')
    ax[2].set_ylabel('Joint 3 Position (rad)')
    ax[2].scatter(elapsed_time, q_joint[2])

    ax2[2].set_xlabel('Time (s)')
    ax2[2].set_ylabel('Tau_3 (Nm)')
    ax2[2].scatter(elapsed_time, tau_q[2])

    ax[3].set_xlabel('Time (s)')
    ax[3].set_ylabel('Joint 4 Position (rad)')
    ax[3].scatter(elapsed_time, q_joint[3])

    ax2[3].set_xlabel('Time (s)')
    ax2[3].set_ylabel('Tau_4 (Nm)')
    ax2[3].scatter(elapsed_time, tau_q[3])

    ax[4].set_xlabel('Time (s)')
    ax[4].set_ylabel('Joint 5 Position (rad)')
    ax[4].scatter(elapsed_time, q_joint[4])

    ax2[4].set_xlabel('Time (s)')
    ax2[4].set_ylabel('Tau_5 (Nm)')
    ax2[4].scatter(elapsed_time, tau_q[3])

    ax[5].set_xlabel('Time (s)')
    ax[5].set_ylabel('Joint 6 Position (rad)')
    ax[5].scatter(elapsed_time, q_joint[5])

    ax2[5].set_xlabel('Time (s)')
    ax2[5].set_ylabel('Tau_6 (Nm)')
    ax2[5].scatter(elapsed_time, tau_q[3])

plt.show()
pick_place(q_joint)
```