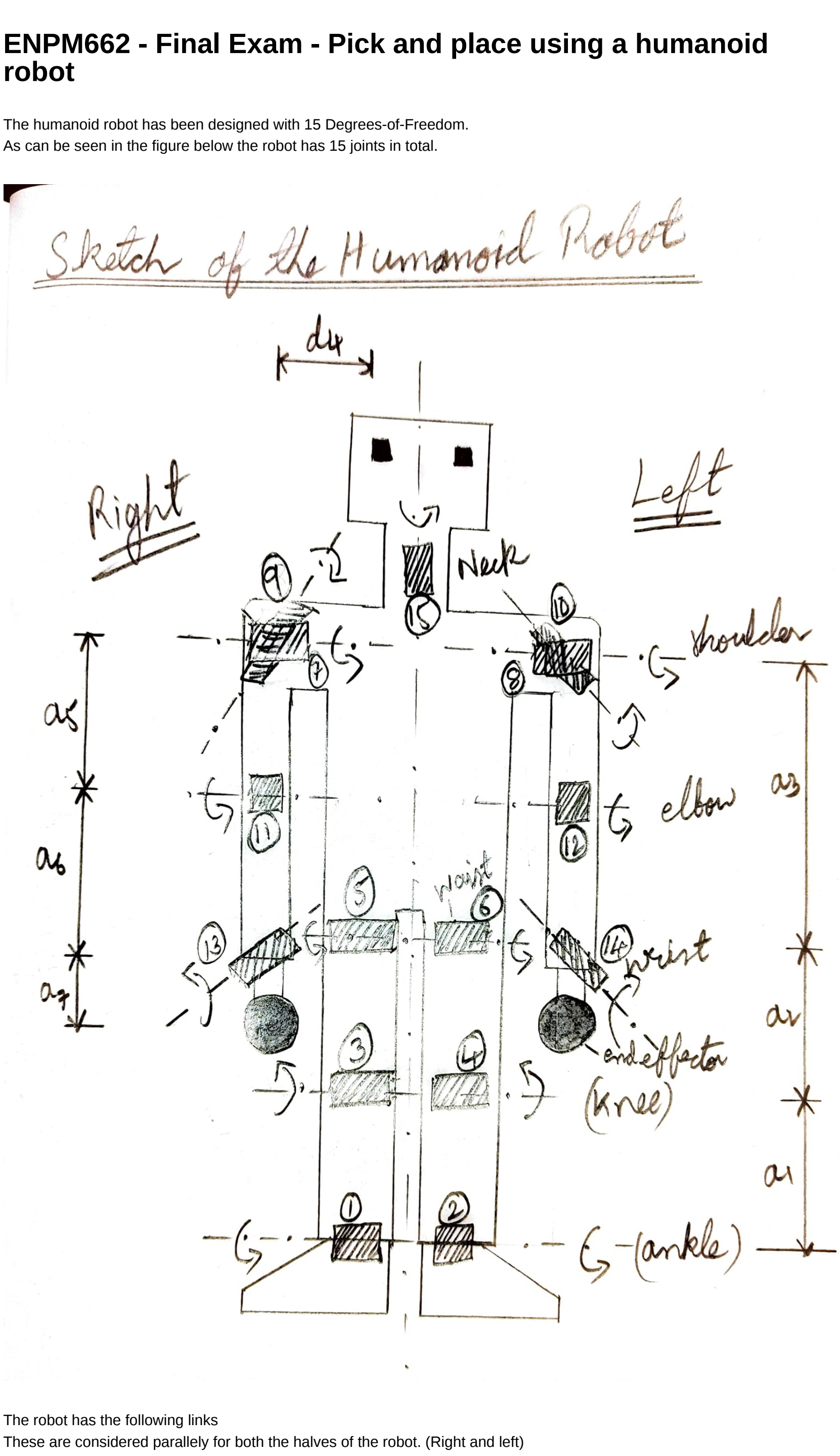
In [3]:
```
import sys
import math # For Math functions
import sympy as sym # For declaring variables as symbols, make sure you have sympy installed
in your system
from sympy import symbols
from sympy import *
import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from mpl_toolkits.mplot3d import Axes3D
```

ENPM662 - Final Exam - Pick and place using a humanoid robot

The humanoid robot has been designed with 15 Degrees-of-Freedom.
As can be seen in the figure below the robot has 15 joints in total.



The robot has the following links
These are considered parallely for both the halves of the robot. (Right and left)

1. Ankle - 2 joints (1,2)
2. Knee - 2 joints (3,4)
3. Waist - 2 joints (5,6)
4. Shoulder - 4 joints (7,8,9,10)
5. Elbow - 2 joints (11,12)
6. Wrist - 2 joints (13,14)
7. Neck - 1 joint (15)

I am assuming the following dimensions for my robot. I am taking the following values for the links of the humanoid
Calf = a1 = 40 cm = 0.40 m
Thigh = a2 = 40 cm = 0.40 m
Torso = a3 = 45 cm = 0.45 m
Upperarm = a5 = 30 cm = 0.30 m
Forearm = a6 = 28 cm = 0.28 m
Palm = a7 = 10 cm = 0.10 m
Shoulder width d4 = 08 cm = 0.08 m
All the links in my robot are assumed to be a hollow cylindrical in shape

I assume the ankle of my robot to be my base.

The motors are placed at the joints of the robot as mentioned above. So in total 15 motors are placed in the robot.

I assume the outer radius of the legs and arms to be 4 cms = 0.04 m and the inner radius to be 3 cms
The outer radius of the torso to be 15 cms = 0.15 m and the inner radius to be 13.5 cm = 0.135 m

Calculating the Mass of the links of the robot

Using the Formula V = we find the volumes of each of the links

1. Volume of each calf =
2. Volume of each thigh =
3. Volume of torso =
4. Volume of each upperarm =
5. Volume of each forearm =
6. Volume of each palm =

It is given that the robot links are made up of aluminium. The density of Aluminium =
We know that Density

Using all this information we can calculate the mass of links of the robot as

1. Mass of each calf = 4.7665 Kgs
2. Mass of each thigh = 4.7665 Kgs
3. Mass of torso = 32.755 Kgs
4. Mass of each upperarm = 3.575 Kgs
5. Mass of each forearm = 3.337 Kgs
6. Mass of each palm = 1.1919 Kgs

Now that we have calculated the mass of all the links, let us estimate (assume) the mass of other auxillary items

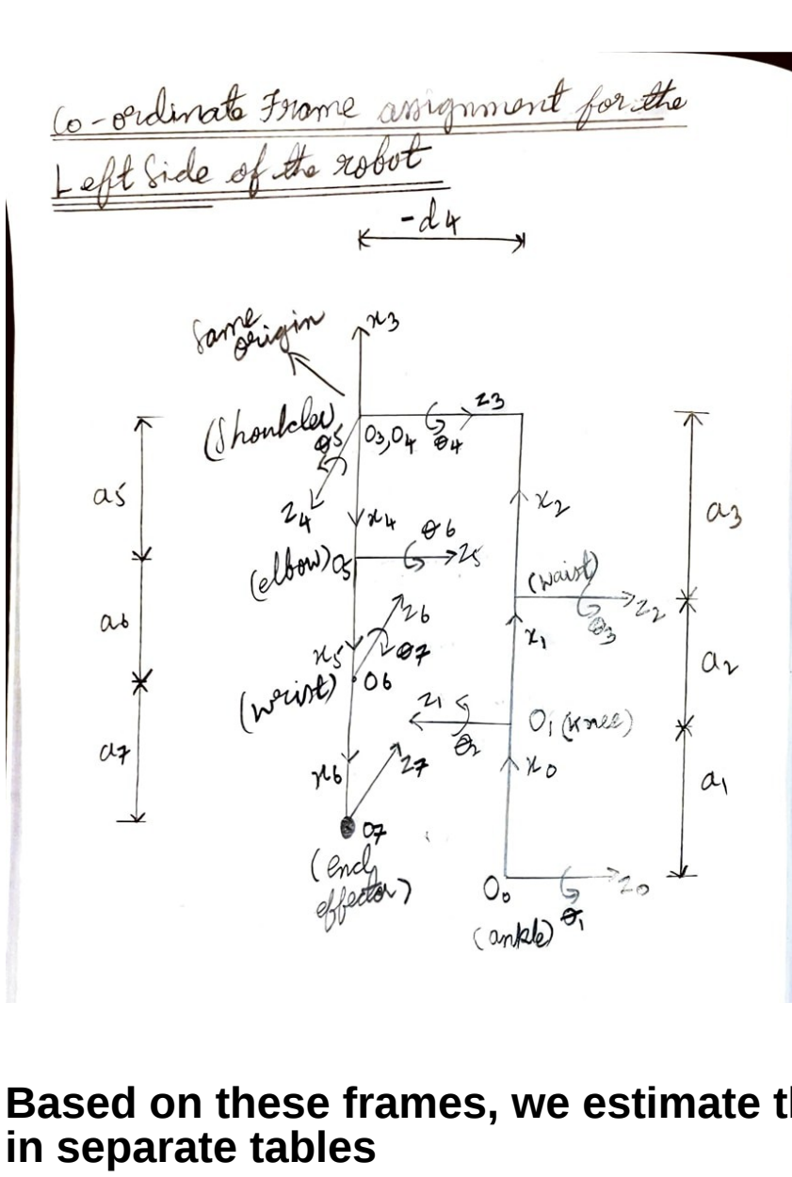
1. Assumed mass of Head + Neck = 2 Kgs
2. Assumed mass of each end effector = 1 Kg
3. Assumed mass of each foot = 1.5 Kgs
4. Assume an error of 0.5 Kg for the total mass of the robot

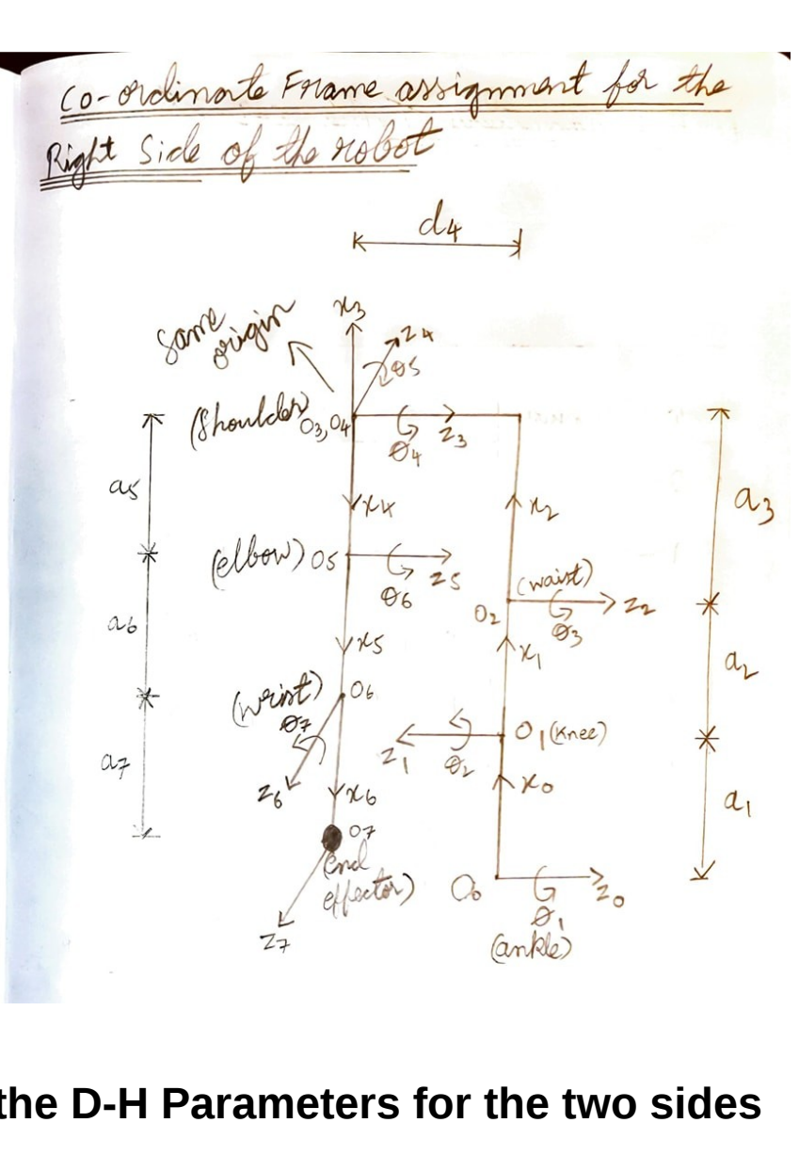
The mass of each motor is given as 1lb = 0.453592 Kgs
Since there are 15 motors in total for my robot, the sum weight of the motors = 6.80388 Kgs

Therefore the total mass of the robot is 57.391 Kgs 0.5 (without motors)
The sum total mass of the robot is 64.19488 Kgs (with motors)

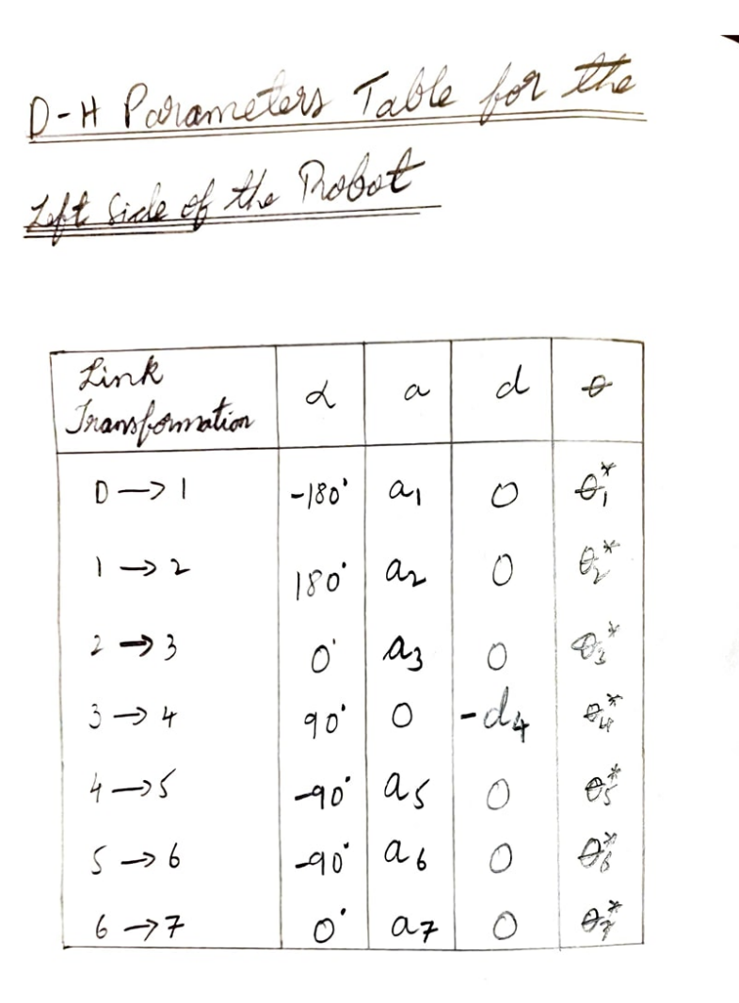
Co-ordinate Frame Assignment for the humanoid robot

As shown above the robot is to have a sum total of 15 Degrees-of-freedom.
For ease of calculation and to prevent a closed loop system the humanoid robot has been split vertically into two parts.
Left side and right side.
Each side will be assigned frames separately and calculations can be made separately.

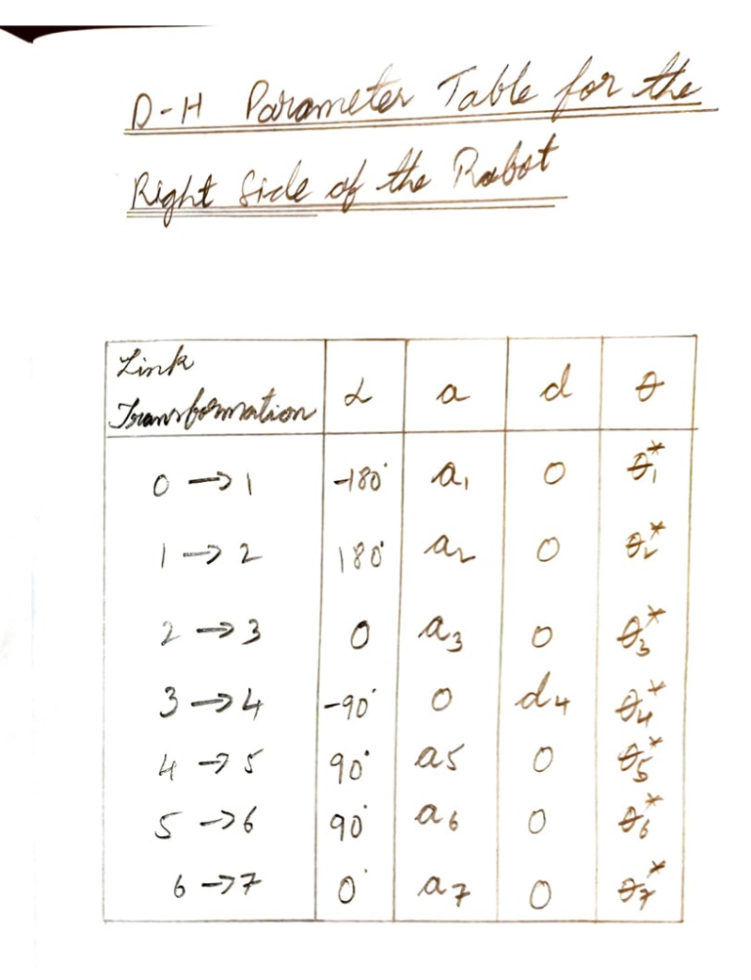

Co-ordinate Frame assignment for the Left Side of the robot


Co-ordinate Frame assignment for the Right Side of the robot

Based on these frames, we estimate the D-H Parameters for the two sides in separate tables


D-H Parameters Table for the Left Side of the Robot

Link Transformation	α	a	d	θ
0 → 1	-180°	a_1	0	θ_1^*
1 → 2	180°	a_2	0	θ_2^*
2 → 3	0°	a_3	0	θ_3^*
3 → 4	90°	0	$-d_4$	θ_4^*
4 → 5	-90°	a_5	0	θ_5^*
5 → 6	-90°	a_6	0	θ_6^*
6 → 7	0°	a_7	0	θ_7^*


D-H Parameter Table for the Right Side of the Robot

Link Transformation	α	a	d	θ
0 → 1	-180°	a_1	0	θ_1^*
1 → 2	180°	a_2	0	θ_2^*
2 → 3	0	a_3	0	θ_3^*
3 → 4	-90°	0	d_4	θ_4^*
4 → 5	90°	a_5	0	θ_5^*
5 → 6	90°	a_6	0	θ_6^*
6 → 7	0°	a_7	0	θ_7^*

Link Lengths and Link Offsets

As asked by the Professor, we will be going forward with the Forward and Inverse Kinematics calculations by considering either half of the humanoid.
I will be using the right side of the humanoid robot for my calculations going forward.
Note: Since there are seven DOF, calculating the Jacobian and its Inverse becomes difficult. So to make it a square matrix, I lock the wrist joint. (Set its angle as = 0 throughout and eliminate its respective Jacobian component for the Jacobian matrix)

Now we need to write the individual transformation matrices for each transformation

In [33]:
```
#defining a function which will return the transformation matrices
def trans(t,al,a,d):
    xi = symbols('theta_i')
    yi = symbols('alpha_i')
    di = symbols('d_i')
    ai = symbols('a_i')
    cxi = sym.cos(xi)
    sxi = sym.sin(xi)
    cyi = sym.cos(yi)
    syi = sym.sin(yi)
    T0_i = sym.Matrix([[cxi, -sxi*cyi, sxi*syi, ai*cxi], [sxi, cxi*cyi, -cxi*syi, ai*syi], [
0, syi, cyi, di], [0, 0, 0, 1]])
    T0_i = T0_i.subs(ai,a).subs(di,d).subs(xi,t).subs(yi,al)
    return T0_i
```

Transformation from 0 to 1

In [199]:
```
t_1 = symbols('theta_1')
t_2 = symbols('theta_2')
t_3 = symbols('theta_3')
t_4 = symbols('theta_4')
t_5 = symbols('theta_5')
t_6 = symbols('theta_6')
t_7 = symbols('theta_7')
t_7 = 0 #locking the joint

T0_1 = trans (t_1,-sym.pi,0.4,0)
T1_2 = trans (t_2,sym.pi,0.4,0)
T2_3 = trans (t_3,0,0.45,0)
T3_4 = trans (t_4,-sym.pi/2,0,-0.08)
T4_5 = trans (t_5,sym.pi/2,0.3,0)
T5_6 = trans (t_6,sym.pi/2,0.28,0)
T6_7 = trans (t_7,0,0.1,0)

T0_2 = T0_1 * T1_2
T0_3 = T0_2 * T2_3
T0_4 = T0_3 * T3_4
T0_5 = T0_4 * T4_5
T0_6 = T0_5 * T5_6
T0_7 = T0_6 * T6_7

#Solving for the jacobian matrix
On = sym.Matrix(T0_7.col(3)[0:3])
O0 = sym.Matrix([0, 0, 0])
O1 = sym.Matrix(T0_1.col(3)[:3])
O2 = sym.Matrix(T0_2.col(3)[:3])
O3 = sym.Matrix(T0_3.col(3)[:3])
O4 = sym.Matrix(T0_4.col(3)[:3])
O5 = sym.Matrix(T0_5.col(3)[:3])
O6 = sym.Matrix(T0_6.col(3)[:3])

r1 = On-O0
r2 = On-O1
r3 = On-O2
r4 = On-O3
r5 = On-O4
r6 = On-O5
r7 = On-O6

z0 = sym.Matrix([0, 0, 1])
z1 = sym.Matrix(T0_1.col(2)[:3])
z2 = sym.Matrix(T0_2.col(2)[:3])
z3 = sym.Matrix(T0_3.col(2)[:3])
z4 = sym.Matrix(T0_4.col(2)[:3])
z5 = sym.Matrix(T0_5.col(2)[:3])
z6 = sym.Matrix(T0_6.col(2)[:3])

Jv1 = z0.cross(r1)
Jw1 = z0
J1 = Jv1.col_join(Jw1)

Jv2 = z1.cross(r2)
Jw2 = z1
J2 = Jv2.col_join(Jw2)

Jv3 = z2.cross(r3)
Jw3 = z2
J3 = Jv3.col_join(Jw3)

Jv4 = z3.cross(r4)
Jw4 = z3
J4 = Jv4.col_join(Jw4)

Jv5 = z4.cross(r5)
Jw5 = z4
J5 = Jv5.col_join(Jw5)

Jv6 = z5.cross(r6)
Jw6 = z5
J6 = Jv6.col_join(Jw6)

Jv7 = z6.cross(r7)
Jw7 = z6
J7 = Jv7.col_join(Jw7)

#Final Jacobian Matrix without the joint at wrist
J = J1.row_join(J2).row_join(J3).row_join(J4).row_join(J5).row_join(J6)

#For the inital position, we rotate the robot as per these angles respectively
q = sym.Matrix([0, 0, 0, math.pi/4, math.pi/4, math.pi/4])
```

Now we find out the end effector's position for these values of theta

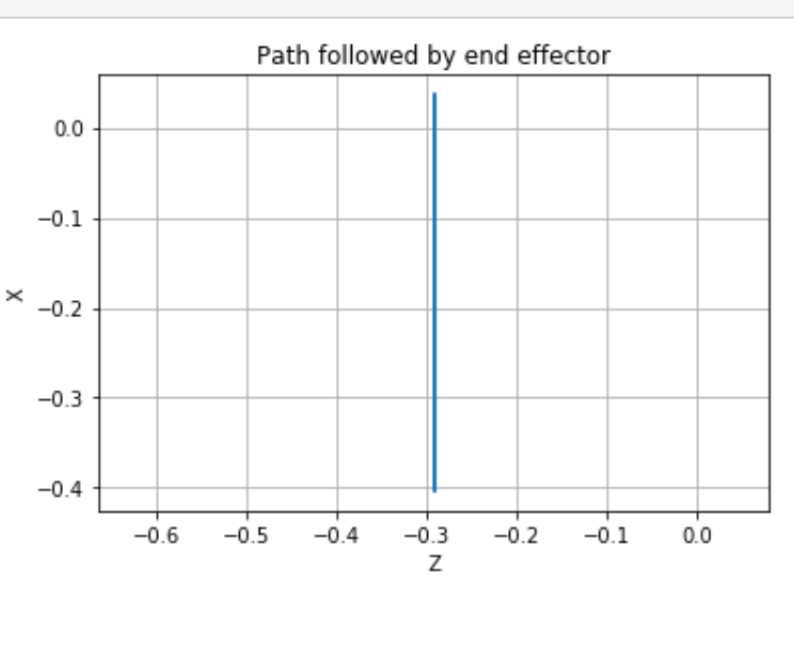
In [210]:
```
Trans0_7 = T0_7.subs(t_1,q[0,0]).subs(t_2,q[1,0]).subs(t_3,q[2,0]).subs(t_4,q[3,0]).subs(t_5
,q[4,0]).subs(t_6,q[5,0])
P7=sym.Matrix([ [0] , [0] , [0] , [1]])

P = Trans0_7 * P7
X = P[0,0]
Y = P[1,0]
Z = P[2,0]
```

Now we make the robot to bend to a certain position (near the knees) so that it can reach there to pick up the box

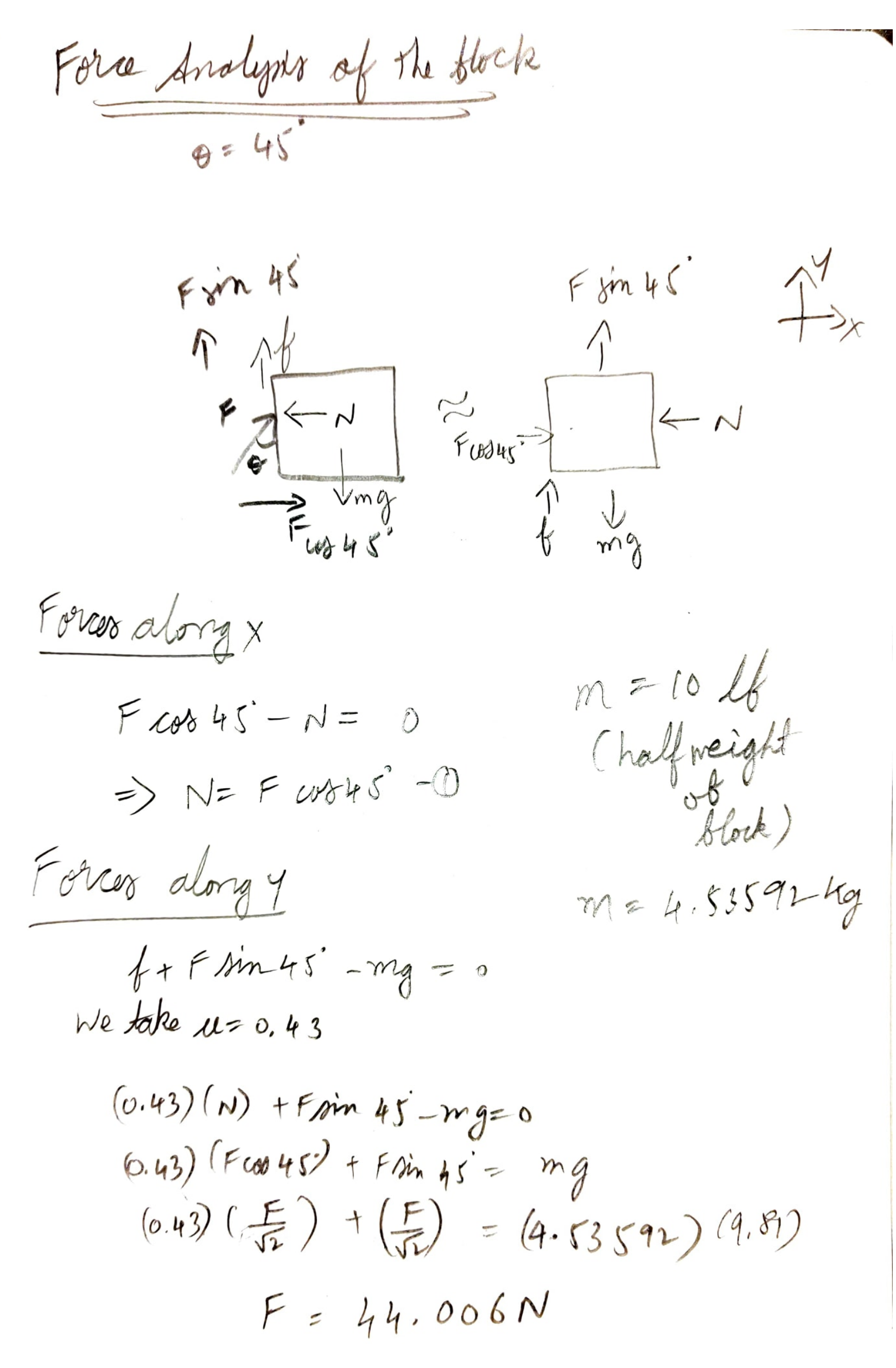
In [223]:
```
X = []
Y = []
Z = []
for i in range(100):
    Jacobian = J.subs(t_1,q[0,0]).subs(t_2,q[1,0]).subs(t_3,q[2,0]).subs(t_4,q[3,0]).subs(t_
5,q[4,0]).subs(t_6,q[5,0])
    psuedo_inv = Jacobian.pinv()

#     ti = math.pi/2 + ( (2*math.pi)*(i) )/100 #initial position of the angle
#     tf = math.pi/2 + ( (2*math.pi)*(i+1) )/100 #new position of the angle

    Trans0_7 = T0_7.subs(t_1,q[0,0]).subs(t_2,q[1,0]).subs(t_3,q[2,0]).subs(t_4,q[3,0]).subs
(t_5,q[4,0]).subs(t_6,q[5,0])
    P7=sym.Matrix([ [0] , [0] , [0] , [1]])

    P = Trans0_7 * P7
    # X and Z coordinates of P are stored
    X.append(P[0,0])
    Y.append(P[1,0])
    Z.append(P[2,0])

    end1 = sym.Matrix([1.28636038969321, 0.532340187157677, -0.57, 0, 0,0]) #starting positi
on
    end2 = sym.Matrix([0.4, 0.532340187157677, -0.57, 0, 0, 0]) #goal position
    delta = (end2-end1)/100 #Since it is quasistatic, each delta must be equal


    #Update the joint variable for the next iteration
    q = q + (psuedo_inv*delta)
```

Here we plot the path taken by then robot to reach that goal position

In [227]:
```
plt.title('Path followed by end effector')
plt.grid()
plt.ylabel('X')
plt.xlabel('Z')
plt.plot(Z,X)
plt.axis("equal")
plt.pause(0.05)
plt.show()
```



New we calculate the Dynamic equations for the robot


Force Analysis of the block

$\theta = 45^\circ$

Forces along x

$F\cos 45^\circ - N = 0$

$\Rightarrow N = F\cos 45^\circ \quad -①$

Forces along y

$f + F\sin 45^\circ - mg = 0$

We take $\mu = 0.43$

$(0.43)(N) + F\sin 45 - mg = 0$

$(0.43)(F\cos 45^\circ) + F\sin 45^\circ = mg$

$(0.43)\left(\frac{F}{\sqrt{2}}\right) + \left(\frac{F}{\sqrt{2}}\right) = (4.53592)(9.81)$

$F = 44.006N$

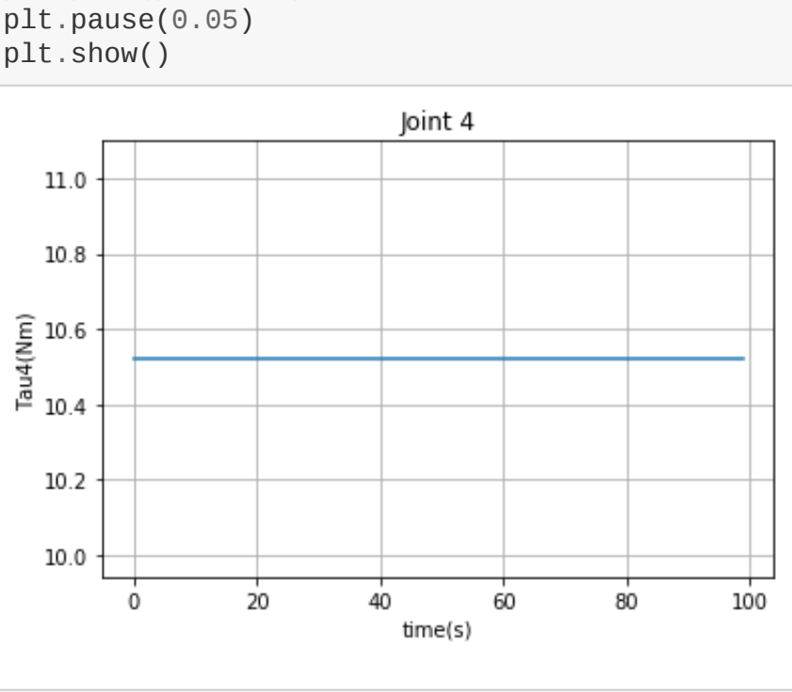
m = 10 lb (half weight of block)

m = 4.53592 kg

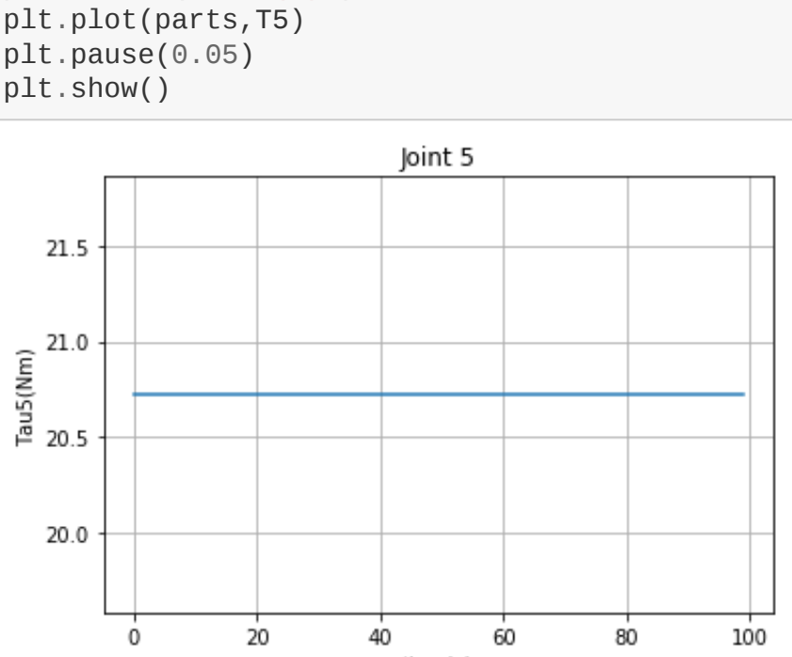
So When we break this force into its components as per the base frame notation we get FX = 31.1169 N and FZ = 31.1169 N

In [233]:
```
q = sym.Matrix([0, 0, 0, math.pi/4, math.pi/4, math.pi/4])
#Force is given as
F = sym.Matrix([ 31.1169, 0, 31.1169, 0, 0, 0])
m1 = 4.7665
m2 = 4.7665
m3 = 16.3775 #Half weight of torso
m4 = 3.575
m5 = 3.337
m6 = 1.1919 #Mean of masses of link 4 and 5
g = 9.81 #the acceleration due to gravity

#Position of centre of gravity
cg1 = sym.Matrix([0.2,0,0,1])
cg2 = sym.Matrix([0.4,0,0,1])
cg3 = sym.Matrix([0.225,-0.04,0,1])
cg4 = sym.Matrix([0,0,0,1])
cg5 = sym.Matrix([0.15,0,0,1])
cg6 = sym.Matrix([0.14,0,0,1])
cg7 = sym.Matrix([0.05,0,0,1])

#Transformations of centre of gravity positions

LG2 = T0_1*cg2
LG3 = T0_3*cg3
LG4 = T0_3*cg4
LG5 = T0_4*cg5
LG6 = T0_5*cg6
LG7 = T0_6*cg7

#finding the potential energy
PE = g * (m1*cg1[0] + m2*LG2[0]+ m3*LG3[0] + m4*LG4[0] + m5*LG5[0] + m6*LG6[0]) #m*g*h

#The gravity matrix is given by
Grav = sym.Matrix([diff(PE,t_1), diff(PE,t_2), diff(PE,t_3),diff(PE,t_4), diff(PE,t_5), diff
(PE,t_6)])

parts = []

#initializing empty list to store the torque calculated for each joint
T1 = []
T2 = []
T3 = []
T4 = []
T5 = []
T6 = []

for i in range (100):
    #Finding the gravity matrix for this iteration
    Gravity = Grav.subs(t_1,q[0,0]).subs(t_2,q[1,0]).subs(t_3,q[2,0]).subs(t_4,q[3,0]).subs(
t_5,q[4,0]).subs(t_6,q[5,0])
    Jacobian = J.subs(t_1,q[0,0]).subs(t_2,q[1,0]).subs(t_3,q[2,0]).subs(t_4,q[3,0]).subs(t_
5,q[4,0]).subs(t_6,q[5,0])
    JT = sym.Transpose(Jacobian)
    Tau = Gravity - JT*F

    #storing these torques for each joint in separate lists initialized above
    T1.append(Tau[0,0])
    T2.append(Tau[1,0])
    T3.append(Tau[2,0])
    T4.append(Tau[3,0])
    T5.append(Tau[4,0])
    T6.append(Tau[5,0])


    #Storing the no. of parts in the list initialized above
    parts.append(i)
```

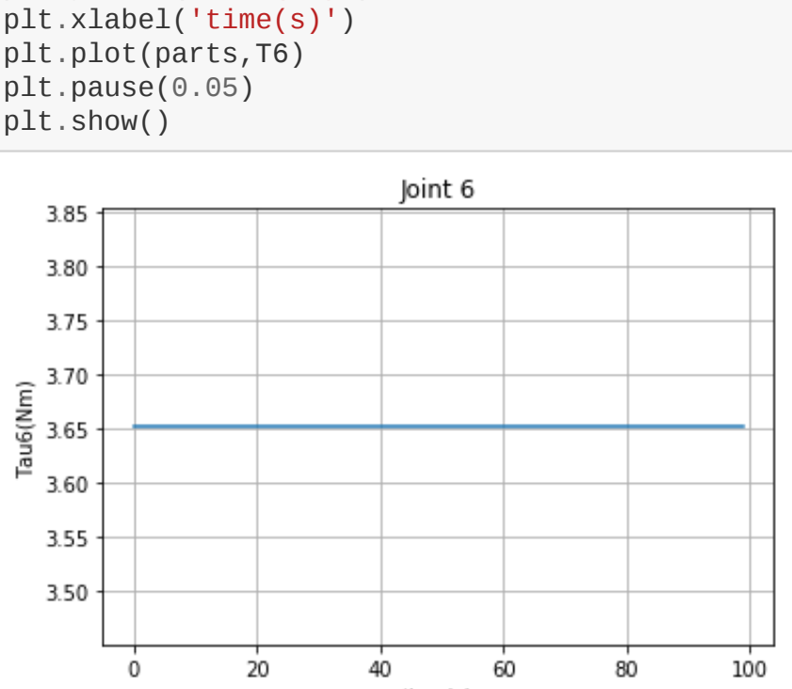
In [234]:
```
plt.title('Joint 4')
plt.grid()
plt.ylabel('Tau4(Nm)')
plt.xlabel('time(s)')
plt.plot(parts,T4)
plt.pause(0.05)
plt.show()
```



In [235]:
```
plt.title('Joint 5')
plt.grid()
plt.ylabel('Tau5(Nm)')
plt.xlabel('time(s)')
plt.plot(parts,T5)
plt.pause(0.05)
plt.show()
```



In [236]:
```
plt.title('Joint 6')
plt.grid()
plt.ylabel('Tau6(Nm)')
plt.xlabel('time(s)')
plt.plot(parts,T6)
plt.pause(0.05)
plt.show()
```



In [ ]: