In [1]:

```
import math
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
%matplotlib inline
```

In [2]:

```
def is_even(n):
    return n%2==0
```

In [3]:

```python
class Hexapod:
    #config HW_1 self,upper_rad=300/2, lower_rad=500/2,upper_lim=1000,low
    #config HW_3,HW_4 self,upper_rad=250/2, lower_rad=650/2,upper_lim=110
    def __init__(self,upper_rad=300/2, lower_rad=650/2,upper_lim=1100,low
        alpha=40,beta=80,euler_order='xyz'):
        self.alpha = alpha
        self.beta = beta
        self.upper_rad = upper_rad
        self.lower_rad = lower_rad
        self.upper_lim = upper_lim
        self.lower_lim = lower_lim
        self.euler_order = euler_order
        self.create_base()
        self.create_body()
        self.create_legs()
        # self.leg_vectors()

    def ik(self,pose):
        upper_lim = self.upper_lim
        lower_lim = self.lower_lim
        self.transform_body(pose)
        for leg_length in self.leg_lengths:
            assert leg_length>lower_lim and leg_length<upper_lim
        return self.leg_vectors

    def fk(self,leg_lengths,error,guess_pose):
        self.transform_body(guess_pose)
        delta_l = np.array(leg_lengths) - self.leg_lengths
        print(delta_l)
        print((error>=delta_l))
        if (error >= delta_l).all():
            ("here")
            return guess_pose
        else:
            print("else called")
            return self.fk(error=error,guess_pose = guess_pose + self.JT_

    def show_robot(self):
        self.calc_Rsi()
        #create figure
        fig = plt.figure()
        ax = fig.add_subplot(111,projection = '3d')

        #create axes
        base_radius = self.lower_rad
        nominal_leg_length = 100
        ax.set_xlim(-base_radius, base_radius)
        ax.set_ylim(-base_radius, base_radius)
        ax.set_zlim(0, nominal_leg_length * 2)
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')


        #create base and body points
        base_c = self.base_center
        body_c = self.body_center
```

```
        ax.scatter(base_c[0],base_c[1],base_c[2],marker='.',c='red',s=100
        ax.scatter(body_c[0],body_c[1],body_c[2],marker='.',c='k',s=50)
        bp = self.base_points
        bdp = self.body_points
        for i in range(6):
            ax.plot(bp[i][0],bp[i][1],bp[i][2],marker='.',color='red')
            ax.plot(bdp[i][0],bdp[i][1],bdp[i][2],marker='.',color='k')

        #create base and body boundaries
        for i in range(-1,5):
            ax.plot([bp[i][0],bp[i+1][0]],[bp[i][1],bp[i+1][1]],[bp[i][2]
            ax.plot([bdp[i][0],bdp[i+1][0]],[bdp[i][1],bdp[i+1][1]],[bdp[

        #create legs
        lg = self.legs
        ni = self.ni
        for i in range(6):
            ax.plot([lg[i][0][0],lg[i][1][0]],[lg[i][0][1],lg[i][1][1]],[
            ax.plot([lg[i][0][0],lg[i][0][0]+ni[i][0]],[lg[i][0][1],lg[i]
                    [lg[i][0][2],lg[i][0][2]+ni[i][2]],color = 'blue')

        #show R_si (Debugging aid)
        R_si = self.R_si
        for i in range(6):
            ax.plot([body_c[0],body_c[0]+R_si[i][0]],[body_c[1],body_c[1]
                    [body_c[2],body_c[2]+R_si[i][2]],color = 'green')


        #show the damn plot
        plt.show()

    def create_base(self):
        self.base_center = np.array([0,0,0])
        alpha = self.alpha
        interval = 120 - alpha
        r = self.lower_rad
        self.base_points = []
        theta = np.deg2rad(-alpha/2)
        for i in range(6):
            x = np.cos(theta)*r
            # print("x is ",x)
            y = np.sin(theta)*r
            # print("y is ",y)
            z = 0
            self.base_points.append(np.array([x,y,z]))

            if is_even(i):
                theta += np.deg2rad(alpha)
            else:
                theta += np.deg2rad(interval)
            # print(theta)
        # print()
        # print("base points are")
        # print(self.base_points)

    def calc_Rsi(self):
        #R_si calculations
```

```
        # print()
        # print("Current body points are")
        # print(self.body_points)
        # print("Current body center is")
        # print(self.body_center)
        # print()
        self.R_si = []
        for j in range(6):
            self.R_si.append(self.body_points[j]-self.body_center)
        # print("self R_si :")
        # print(self.R_si)

    def create_body(self):
        #print("create body called:")
        self.body_center=np.array([0,0,0])
        r = self.upper_rad
        self.body_points = []
        #print(self.body_points)
        beta= self.beta
        interval = 120 - beta
        theta = np.deg2rad(-beta/2)
        for i in range(6):
            x = np.cos(theta)*r
            y = np.sin(theta)*r
            z = 0
            self.body_points.append(np.array([x,y,z]))

            if is_even(i):
                theta += np.deg2rad(beta)
            else:
                theta += np.deg2rad(interval)
        #self.calc_Rsi()
        # print()
        # print("body points are")
        # print(self.body_points)
        # print(len(self.body_points))
        # print()

    def create_legs(self):
        self.legs = []
        self.leg_vectors = []
        for i in range(6):
            self.legs.append((self.base_points[i],self.body_points[i]))
            self.leg_vectors.append(self.body_points[i]-self.base_points[
        self.ni = [] #This is the list of leg unit vectors
        self.leg_lengths = []
        for vector in self.leg_vectors:
            l = np.linalg.norm(vector)
            self.leg_lengths.append(l)
            self.ni.append(vector/l)


    def euler_to_R(self,theta_list):
        self.R_mat = np.identity(3)
        for index,theta in enumerate(theta_list):
            c = self.euler_order[index]
            if c.lower() == 'x':
```

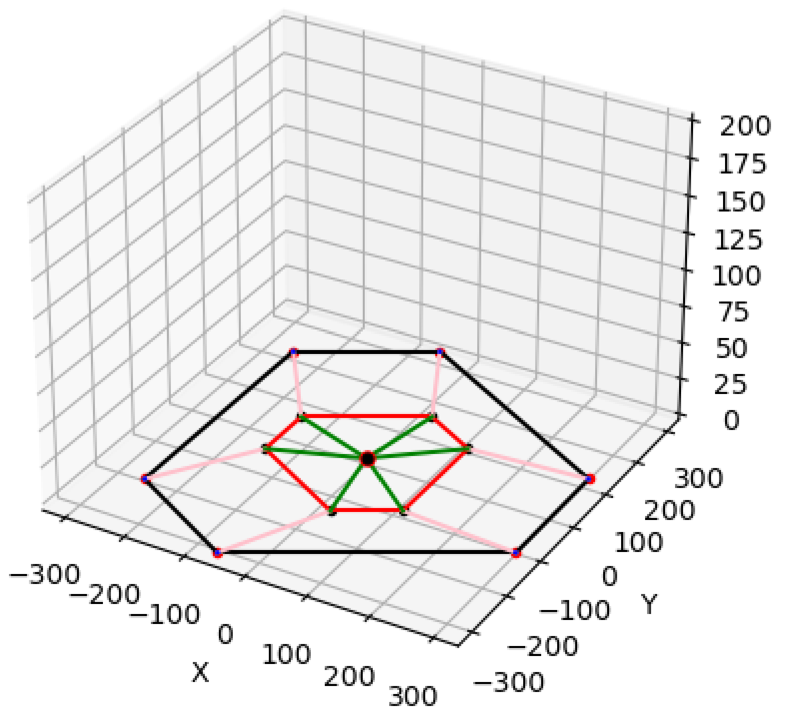
```
                    rotation = self.rotation_x(np.deg2rad(theta))
                elif c.lower() == 'y':
                    rotation = self.rotation_y(np.deg2rad(theta))
                else:
                    rotation = self.rotation_z(np.deg2rad(theta))
                self.R_mat = self.R_mat@rotation



    def rotation_x(self,roll):
        return np.array([[1, 0, 0],
                        [0,np.cos(roll), np.sin(roll)],
                        [0,np.sin(roll),np.cos(roll)]])

    def rotation_y(self,pitch):
        return np.array([[np.cos(pitch), 0,np.sin(pitch)],
                        [0, 1, 0],
                        [np.sin(pitch), 0,np.cos(pitch)]])

    def rotation_z(self,yaw):
        return np.array([[np.cos(yaw), np.sin(yaw), 0],
                        [np.sin(yaw),np.cos(yaw), 0],
                        [0, 0, 1]])

    def T_mat(self):
        pose = self.pose
        theta_list = pose[3::]
        p = self.pose[0:3]
        self.euler_to_R(theta_list=theta_list)
        R_mat = self.R_mat

        #print("P is ",p)
        #where R_mat is the rotation matrix and p is the displacement vec
        self.T_matrix = np.array([[R_mat[0][0],R_mat[0][1],R_mat[0][2],p[
                                [R_mat[1][0],R_mat[1][1],R_mat[1][2],p[1]],
                                [R_mat[2][0],R_mat[2][1],R_mat[2][2],p[2]],
                                [0,0,0,1]])
        #print("T_matrix is ",self.T_matrix)

    def transform_body(self,pose):
        self.create_body()
        self.pose = pose
        self.T_mat()
        body = self.body_points
        T_mat = self.T_matrix
        self.body_center = T_mat[0:3,3]
        #print("body center is at: ", self.body_center)
        for i,point in enumerate(body):
            self.body_points[i] = np.dot(T_mat,np.append(point,1))[0:3]
        self.calc_Rsi()
        # print("body points are")
        # print(self.body_points)
        self.create_legs()
        self.Jacobian()
        self.JT_mat()

    def euler_to_angular(self,alpha_dot):
```

```
        #This is essentially the matrix that we need to create
        #In order to convert alpha_dot to omega
        #A slightly modified Euler_to_R should work for this purpose
        #And along the way, I can also populate the B_matrix
        w = np.array([0,0,0])
        R_mat = np.identity(3)
        for index,theta in enumerate(self.pose[3:]):
            c = self.euler_order[index]
            i = 0
            if c.lower() == 'x':
                rotation = self.rotation_x(theta)
            elif c.lower() == 'y':
                rotation = self.rotation_y(theta)
                i+=1
            else:
                rotation = self.rotation_z(theta)
                i+=2
            R_mat = R_mat@rotation
            w = R_mat[:,i]*alpha_dot[index]+w
        self.omega = w


    def B_matrix(self):
        a = self.pose[3]
        b = self.pose[4]
        self.B_mat = np.array([[1,0,np.sin(b)],[0,np.cos(a),-np.sin(a)*np


    def Jacobian(self):
        #We are going to create the Jacobian now
        #[ni,[Rsi x ni]]
        Jacobian = []
        for index, ni in enumerate(self.ni):
            Jacobian.append(np.hstack((ni,np.cross(self.R_si[index],ni)))
        self.J = np.array(Jacobian)
        # print("Jacobian is ", self.J)

    def JT_mat(self):
        self.B_matrix()
        Top = np.hstack((np.identity(3),np.zeros((3,3))))
        Bottom = np.hstack((np.zeros((3,3)),self.B_mat))
        T_alpha = np.vstack((Top,Bottom))
        self.JT_matrix = self.J@T_alpha
        # print()
        # print()
        # print("JT")
        # print(self.JT_matrix)
        self.JT_inv = np.linalg.inv(self.JT_matrix)

```

In [4]:

```
my_hexa = Hexapod()
```

In [5]:
```
my_hexa.show_robot()
```

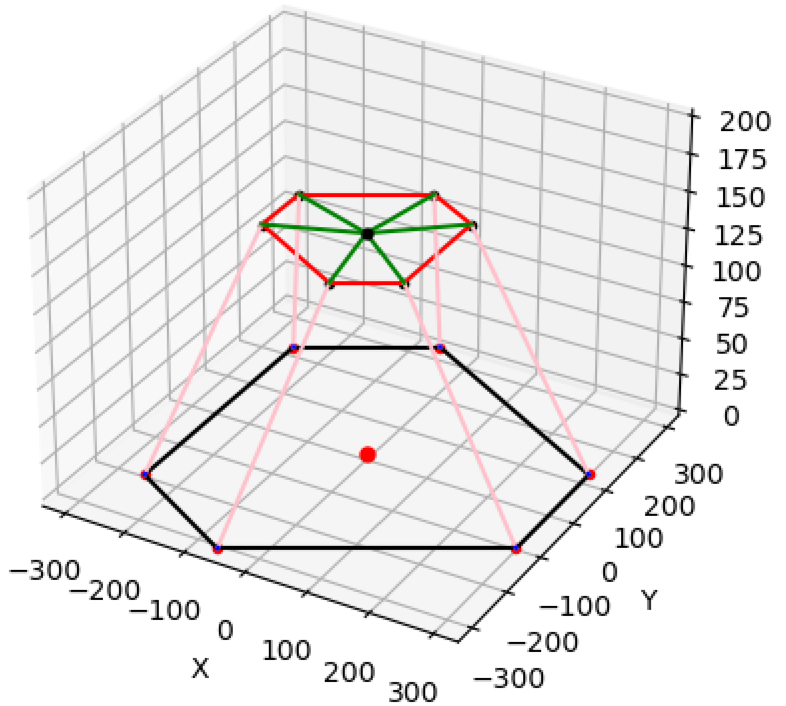


In [6]:
```
my_hexa.transform_body([0,0,150,0,0,0])
```

In [7]:
```
my_hexa.body_points
```

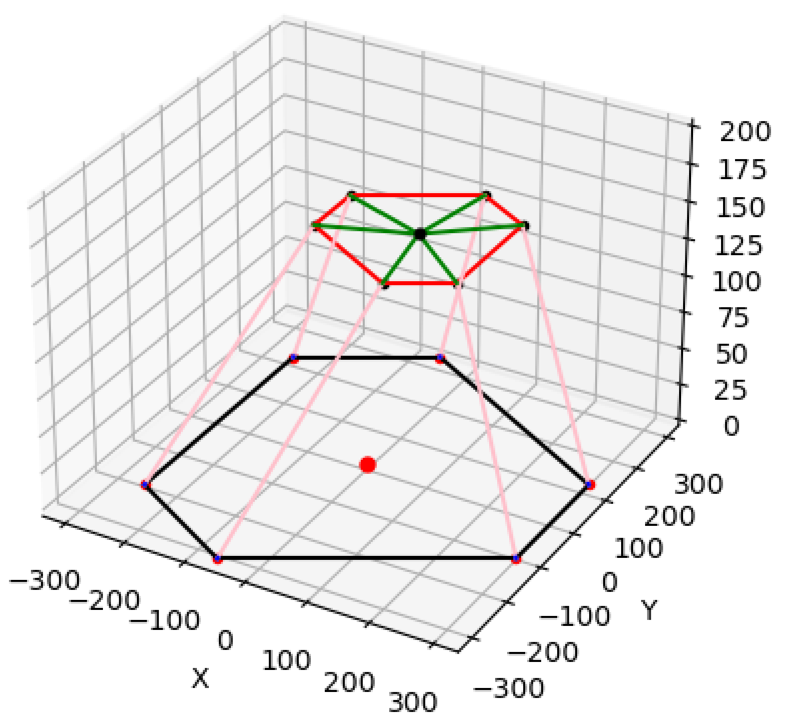
Out[7]:
```
[array([114.90666647, -96.41814145, 150.        ]),
 array([114.90666647,  96.41814145, 150.        ]),
 array([ 26.04722665, 147.72116295, 150.        ]),
 array([-140.95389312,   51.3030215 ,  150.        ]),
 array([-140.95389312,  -51.3030215 ,  150.        ]),
 array([  26.04722665, -147.72116295,  150.        ])]
```

In [8]:
```
my_hexa.show_robot()
```



In [9]:
```
my_hexa.transform_body([50,60,150,0,0,0])
```

In [10]:
```
my_hexa.show_robot()
```



In [11]:
```python
my_hexa.body_points
```

Out[11]:
```
[array([164.90666647, -36.41814145, 150.        ]),
 array([164.90666647, 156.41814145, 150.        ]),
 array([ 76.04722665, 207.72116295, 150.        ]),
 array([-90.95389312, 111.3030215 , 150.        ]),
 array([-90.95389312,   8.6969785 , 150.        ]),
 array([ 76.04722665, -87.72116295, 150.        ])]
```

In [12]:
```python
my_hexa.leg_vectors
```

Out[12]:
```
[array([-140.49343529,   74.73840513,  150.        ]),
 array([-140.49343529,   45.26159487,  150.        ]),
 array([ 132.48288439, -112.34135678,  150.        ]),
 array([158.0105509 , -97.60295165, 150.        ]),
 array([158.0105509 , 217.60295165, 150.        ]),
 array([132.48288439, 232.34135678, 150.        ])]
```

In [13]:
```python
my_hexa.ni
```

Out[13]:
```
[array([-0.64243918,  0.34175888,  0.68591017]),
 array([-0.66760229,  0.21507584,  0.71277596]),
 array([ 0.57725626, -0.48949532,  0.65358208]),
 array([ 0.66187012, -0.40883648,  0.62831575]),
 array([0.51314467, 0.70667303, 0.48713013]),
 array([0.43203289, 0.75767604, 0.48915702])]
```

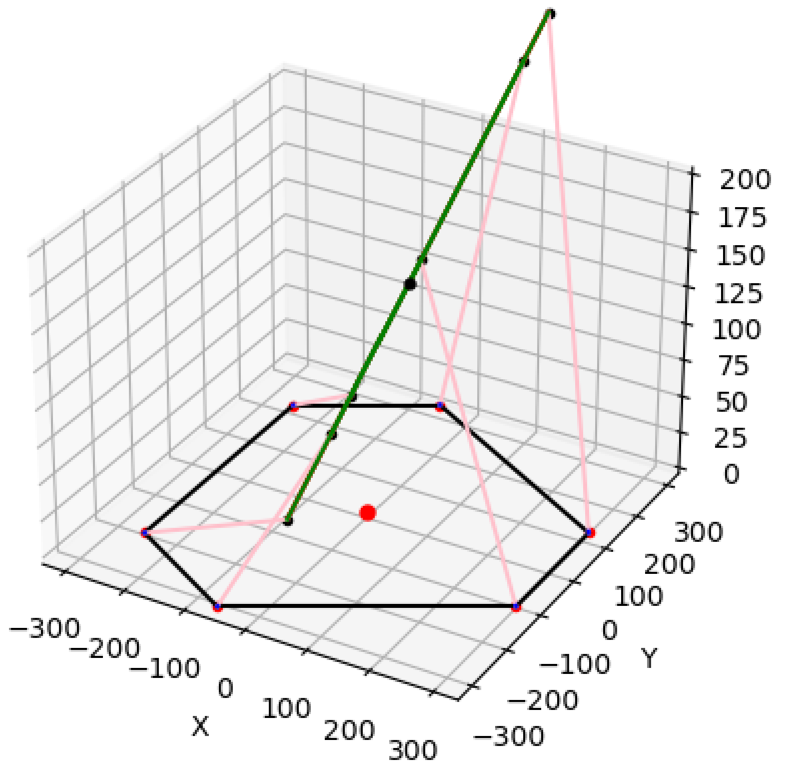
In [14]:
```python
my_hexa.body_points
```

Out[14]:
```
[array([164.90666647, -36.41814145, 150.        ]),
 array([164.90666647, 156.41814145, 150.        ]),
 array([ 76.04722665, 207.72116295, 150.        ]),
 array([-90.95389312, 111.3030215 , 150.        ]),
 array([-90.95389312,   8.6969785 , 150.        ]),
 array([ 76.04722665, -87.72116295, 150.        ])]
```

In [15]:
```python
my_hexa.transform_body([40,50,150,45,30,45])
```

In [16]:

```

my_hexa.show_robot()
```



In [17]:

```
my_hexa.Jacobian()
```

In [18]:

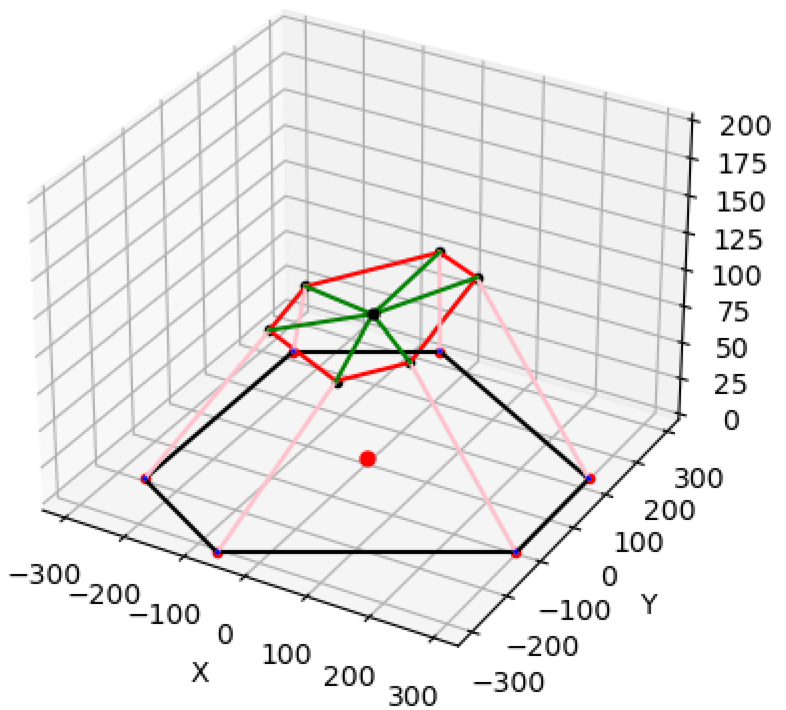
```
my_hexa.JT_mat()
```

In [19]:

```
my_hexa.ik([10,0,100,5,5,0])
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
Cell In[19], line 1
----> 1 my_hexa.ik([10,0,100,5,5,0])

Cell In[3], line 23, in Hexapod.ik(self, pose)
     21 self.transform_body(pose)
     22 for leg_length in self.leg_lengths:
---> 23     assert leg_length>lower_lim and leg_length<upper_lim
     24 return self.leg_vectors

AssertionError: 
```

In [20]:
```
1 my_hexa.show_robot()
```



In [21]:
```
1 np.linalg.norm([-84.1,-39,88.7])
```

Out[21]:
```
128.30237721881852
```

In [22]:
```
1 my_hexa.leg_lengths
```

Out[22]:
```
[208.1065721744376,
 216.68483189079612,
 227.19672990984185,
 218.63130879145527,
 213.43423601299608,
 215.6126851102431]
```

In [23]:
```
1 my_hexa.fk([250.1730, 247.7072, 253.3073, 277.6336, 278.4548, 254.3322],g
```

```
[61.35822782 53.76310951 46.89817558 74.96062888 79.14316529 54.20685909]
[False False False False False False]
else called
[-229.22965347 -252.83666588 -208.69772404 -145.75814045 -133.65875007
 -176.07464177]
[ True  True  True  True  True  True]
```

Out[23]:
```
array([-57.6704923 ,  43.21846314, 400.25446317,  13.85278789,
         5.62334991,   8.93065528])
```