Modeling of hexapod robot leg using MATLAB/ Simulink environment

Submitted to:

Prof. Cedric Anthierens

Geometric, kinematic of robotic systems

Student name: Ramy Antar Ahmed Alham

Marine and Maritime Robotics (MIR)

Group: 1

Abstract:

In this report I present the design and simulate of the hexapod robot leg using MATLAB Simulink and the Simscape Multibody environment and simulate its behavior with generate a forward step without wobbling the robot body. I used the Jacobian matrix as the MATLAB function, and it provides the joints with the required input signals.

1- Introduction:

The application of MATLAB simulation in robotics has increased in these eras. As it allows the engineers and the researchers to predict, test and validate performance of robotics before manufacturing. Thanks to the nature, by studying and investigating the locomotion of animals, and this helped researchers to design and manufacture myriad types of robots. Among these are the arthropods which have six legs and thus maintain their stability. In this work we will use the MATLAB to design and simulate a hexapod robot leg and try the simulate Its stability, firstly, we used the Simscape to make Cad model of the leg, and then drive the Cad model in its workplace by using forward step along X axes and validate by simulation that we can generate a forward step without generating any other motion to the whole body of the robot.

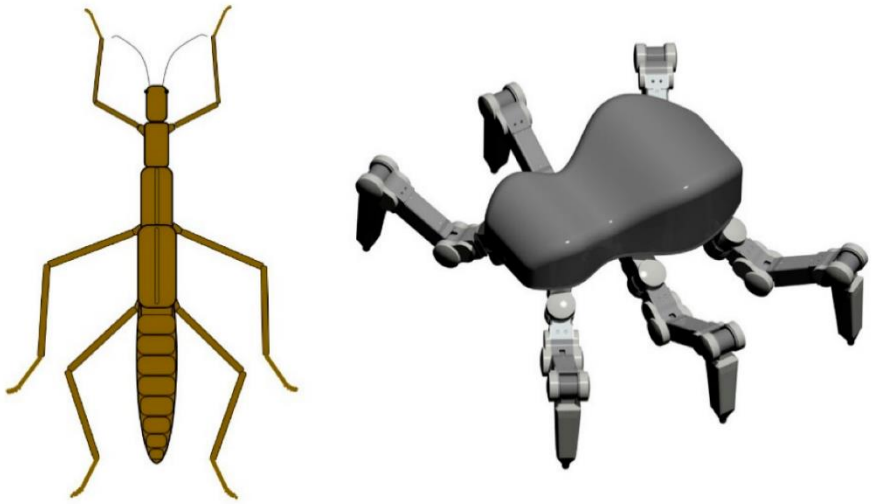

Figure 1 hexapod robot and the arthropods: This figure shows the hexapod robot and the arthropods, and the human endeavors to design in hexapods robots.

2- Design of the CAD model of the hexapod leg.

To build the CAD model, we will use Simscape.

Firstly, there are three blocks that are the basic blocks which are required for any kind of multibody element or multibody mechanisms which are going to use or make using Simscape multibody toolbox:

1- solver configuration block: This used to define what kind of ODE solver this mechanism is going to use

2- world frame block: This frame will define the coordinate system of all mechanisms which we are going to build

3- mechanism configuration: This frame will define the coordinate system of all mechanisms which we are going to build.

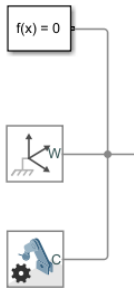

Figure 2 the three blocks which are using for every kind of model

Then we are going to build the robot leg from the Hip to the end effector, which consists of links and joints.

Firstly, we will make the link by going to body elements in Simscape and select solid type, and change its properties, mass, inertia, geometry, and color. Whatever you have made the frame of this body will be exactly at its center, so, if I attached the body to the world frame then the world frame and the frame of the body will be at the center of this link, but I want the world frame to be at the base of the body, so I need transformation. So, I will go to transform and frames and choose rigid transform. This rigid transform will transform the world frame to the frame of the link.

Secondly, to add the first joint, we will go to multibody and select the type of joint, and before attaching this link to joint we need to move the frame of the joint to the end of the link, because the frame of the link will be at its center, and we need another rigid transform.

And we follow these steps for all links and joints, and then we will obtain the cad model of the leg robot.

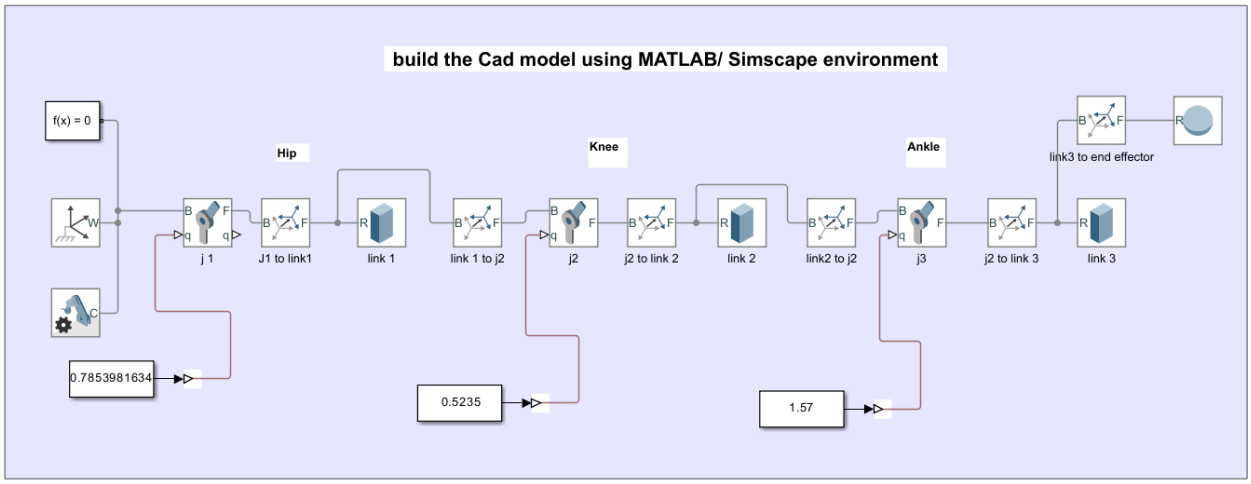

Figure 3 model the robot leg using Simulink: this photo represents the steps of drawing the CAD model of the leg robot

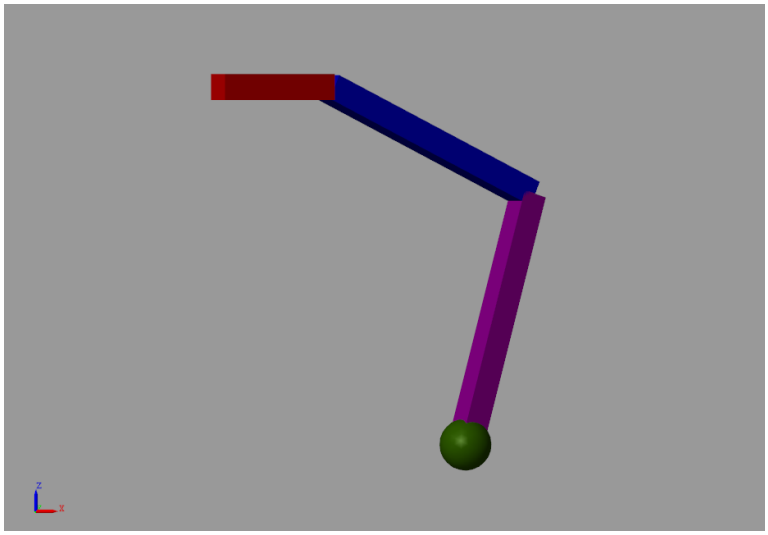

Figure 4 CAD model: this photo represents the desired CAD model of the robot leg which obtained from the MATLAB/Simscape environment.

3- Drive the leg of the robot.

Now, we need to drive the CAD model in its workplace by using algorithms, there are two ways to ask the robot to do this:

1- We can command the joint to rotate a certain angle within the rotational limit of the joint, and when the joints rotated to those the certain angles – would the end effector reached the required point in the workspace. This what we called the forward kinematics algorithms.

2- In reality we know that where **the end effector should reach** but we **don't know the angles** which each joint must rotate to reach the end effector to the desired location. And there are multiple joint angles that will take the robot to the same point in the workspace. Therefore, it's a difficult task to provide joint angles directly hoping that a robot will reach the intended location. To cater this, we need some other kind of algorithm that will take in the point in the workspace and generate the joints angle for the joints, which when traversed by a joint the end effector will reach the desired location. This what we call the inverse kinematic algorithms.

In our work, we have made an algorithm that takes the trajectory position of the robot leg, and thanks to Jacobian matrix which give us the velocity of each joint and then, integrate it to give the each link the required angle.

The relation between the velocity of end effector and Jacobian:

$$v_n^0 = J_v \dot{q}$$
$$\omega_n^0 = J_\omega \dot{q}$$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}_{6\times1} = J_{6\times n} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dots \\ \dot{q}_n \end{bmatrix}_{n\times1}$$

Where, q with the dot on top represents the joint.

The size of this vector is n x 1, with n being equal to the number of joints.

Note that q can represent either revolute joints (typically represented as θ) or prismatic joints (which I usually represent as d, which means displacement)

J is the Jacobian matrix. It is an m rows x n column matrix (m=3 for two dimensions, and m=6 for a robot that operates in three dimensions). n represents the number of joints.

The matrix on the left represents the **velocities of the end effector**,**x, y, and z** with the dots on top represent **linear velocities.**

4- kinematics of a hexapod robot leg

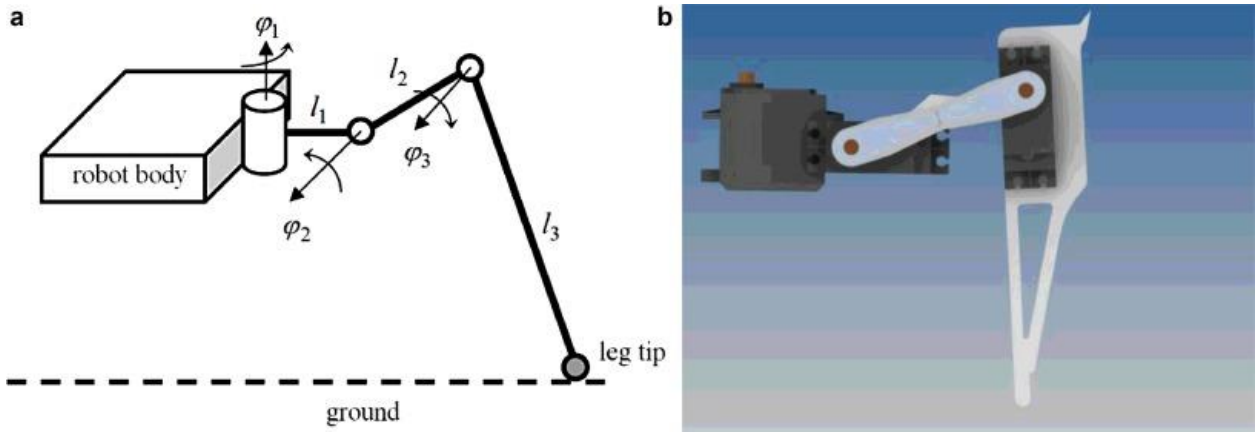

Figure 5 Model of the hexapod robot leg

In order to obtain the kinematics of the hexapod robot leg, I used the Denavit-Hartenberg algorithm, applying it to a leg of the hexapod robot, so we can obtain the Denavit-Hartenberg parameters.

Table 1: Denavit-Hartenberg parameters.

i	a_i	α_i	d_i	q_i
1	a_1	-π/2	0	q_1
2	a_2	0	0	q_2
3	a_3	0	0	q_3

We can obtain the Jacobian with two methods.

1- Manual:

First, we find T_1^0, T_2^0, $\boldsymbol{T_3^0}$.
Secondly, we find the Jacobian by using.

- The i^{th} column of J_v is given by:

$$J_{v_i} = \begin{cases} z_{i-1} \times (o_n - o_{i-1}) & \text{for } i \text{ revolute} \\ z_{i-1} & \text{for } i \text{ prismatic} \end{cases}$$

- The i^{th} column of J_ω is given by:

$$J_{\omega_i} = \begin{cases} z_{i-1} & \text{for } i \text{ revolute} \\ 0 & \text{for } i \text{ prismatic} \end{cases}$$

The solution manually:

$$T_1^0 = \begin{bmatrix} cos(q1) & 0 & -sin(q1) & a1 * cos(q1) \\ sin(q1) & 0 & cos(q1) & a1 * sin(q1) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_2^0 = \begin{bmatrix} cos(q1) * cos(q2) & -cos(q1) * sin(q2) & -sin(q1) & cos(q1) * (a1 + a2 * cos(q2)) \\ cos(q2) * sin(q1) & -sin(q1) * sin(q2) & cos(q1) & sin(q1) * (a1 + a2 * cos(q2)) \\ -sin(q2) & -cos(q2) & 0 & -a2 * sin(q2) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{T_3^0 = \begin{bmatrix} cos(q2 + q3) * cos(q1) & -sin(q2 + q3) * cos(q1) & -sin(q1) & cos(q1) * (a1 + a3 * cos(q2 + q3) + a2 * cos(q2)) \\ cos(q2 + q3) * sin(q1) & -sin(q2 + q3) * sin(q1) & 0cos(q1) & sin(q1) * (a1 + a3 * cos(q2 + q3) + a2 * cos(q2)) \\ -sin(q2 + q3) & -cos(q2 + q3) & 0 & -a3 * sin(q2 + q3) - a2 * sin(q2). \\ 0 & 0 & 0 & 1 \end{bmatrix}}$$

$$J_1 = \begin{bmatrix} z_0 \times (o_3 - o_0) \\ z_0 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \times \begin{bmatrix} \cos(q1) * (a1 + a3 * \cos(q2 + q3) + a2 * \cos(q2)) \\ \sin(q1) * (a1 + a3 * \cos(q2 + q3) + a2 * \cos(q2)) \\ -a3 * sin(q2 + q3) + a2 * sin(q2) \end{bmatrix} \\ \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} -sin(q1) * (a1 + a3 * cos(q2 + q3) + a2 * cos(q2)) \\ cos(q1) * (a1 + a3 * cos(q2 + q3) + a2 * cos(q2)) \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$J_2 = \begin{bmatrix} z_1 \times (o_3 - o_1) \\ z_1 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} -\sin(q1) \\ cos(q1) \\ 1 \end{bmatrix} \times \begin{bmatrix} \cos(q1) * (a1 + a3 * \cos(q2 + q3) + a2 * \cos(q2)) - a1 * \cos(q1) \\ \sin(q1) * (a1 + a3 * \cos(q2 + q3) + a2 * \cos(q2)) - a1 * \sin(q1) \\ -a3 * sin(q2 + q3) - a2 * sin(q2) \end{bmatrix} \\ \begin{bmatrix} -sin(q1) \\ cos(q1) \\ 0 \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} -cos(q1) * (a3 * sin(q2 + q3) + a2 * sin(q2)) \\ -sin(q1) * (a3 * sin(q2 + q3) + a2 * sin(q2)) \\ -a3 * cos(q2 + q3) - a2 * cos(q2) \\ -sin(q1) \\ cos(q1) \\ 0 \end{bmatrix}$$

$$J_3 = \begin{bmatrix} z_2 \times (o_3 - o_2) \\ z_2 \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} -sin(q1) \\ cos(q1) \\ 0 \end{bmatrix} \times \begin{bmatrix} \cos(q1) * (a1 + a3 * \cos(q2 + q3) + a2 * \cos(q2)) - \cos(q1) * (a1 + a2 * \cos(q2)) \\ \sin(q1) * (a1 + a3 * \cos(q2 + q3) + a2 * \cos(q2)) - \sin(q1) * (a1 + a2 * \cos(q2)) \\ -a3 * sin(q2 + q3) \end{bmatrix} \\ \begin{bmatrix} -sin(q1) \\ cos(q1) \\ 0 \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} -a3 * sin(q2 + q3) * cos(q1) \\ -a3 * sin(q2 + q3) * sin(q1) \\ -a3 * cos(q2 + q3)) \\ -sin(q1) \\ cos(q1) \\ 0 \end{bmatrix}$$

Jacobian Matrix= [J1 J2 J3]

$$J = \begin{bmatrix} -sin(q1) * (a1 + a3 * cos(q2 + q3) + a2 * cos(q2)) & -cos(q1) * (a3 * sin(q2 + q3) + a2 * sin(q2)) & -a3 * sin(q2 + q3) * cos(q1) \\ cos(q1) * (a1 + a3 * cos(q2 + q3) + a2 * cos(q2)) & -sin(q1) * (a3 * sin(q2 + q3) + a2 * sin(q2)) & -a3 * sin(q2 + q3) * sin(q1) \\ 0 & -a3 * cos(q2 + q3) - a2 * cos(q2 & -a3 * cos(q2 + q3) \\ 0 & -sin(q1) & -sin(q1)) \\ 0 & cos(q1) & cos(q1) \\ 1 & 0 & 0 \end{bmatrix}$$

2- Using Peter Corke robotics Toolbox

Algorthim.1 Find the Jacobian Matrix using Peter Corke robotics Toolbox

```
%DH- parameters using peter corke robotics
syms a1 a2 a3 q1 q2 q3
%L = link([Theta d a alpha],'p') if presimatic put p not remove
L(1) = Link([0 0 a1 -pi/2]);
L(2) = Link([0 0 a2 0]);
L(3) = Link([0 0 a3 0]);

%DH=[L(1); L(2); L(2)];
A1=L(1).A(q1);
A2=L(2).A(q2);
A3=L(3).A(q3);
%for asseply all the robot we use serial link
hexapod_leg= SerialLink(L);
% make a name for the robot
hexapod_leg.name='hexapod_leg'

%Transformation Matrices
T1_0=double(A1);
T2_0=double(simplify(A1*A2));
T3_0=double(simplify(A1*A2*A3));
```

```
% Z and O for Jacobian
z0=[0; 0;1]; o0=[0; 0; 0];
z1=T1_0(1:3,3); o1=T1_0(1:3,4);
z2=T2_0(1:3,3); o2=T2_0(1:3,4);
o3=T3_0(1:3,4);

%Jacobian formation
J1=[simplify(cross(z0,(o3-o0)));z0];
J2=[simplify(cross(z1,(o3-o1)));z1];
J3=[simplify(cross(z2,(o3-o2)));z2];
jacobian_matrix=[J1 J2 J3];
```

After running the code, just write in command window: jacobian_matrix
And we will find:

```
>> jacobian_matrix

jacobian_matrix =

[-sin(q1)*(a1 + a3*cos(q2 + q3) + a2*cos(q2)), -cos(q1)*(a3*sin(q2 + q3) + a2*sin(q2)), -a3*sin(q2 + q3)*cos(q1)]
[ cos(q1)*(a1 + a3*cos(q2 + q3) + a2*cos(q2)), -sin(q1)*(a3*sin(q2 + q3) + a2*sin(q2)), -a3*sin(q2 + q3)*sin(q1)]
[                                           0,          - a3*cos(q2 + q3) - a2*cos(q2),          -a3*cos(q2 + q3)]
[                                           0,                               -sin(q1),                  -sin(q1)]
[                                           0,                                cos(q1),                   cos(q1)]
[                                           1,                                      0,                         0]
```

Figure 6: Jacobian Matrix using Peter Cork robotics Toolbox

5- Full model in MATLAB:

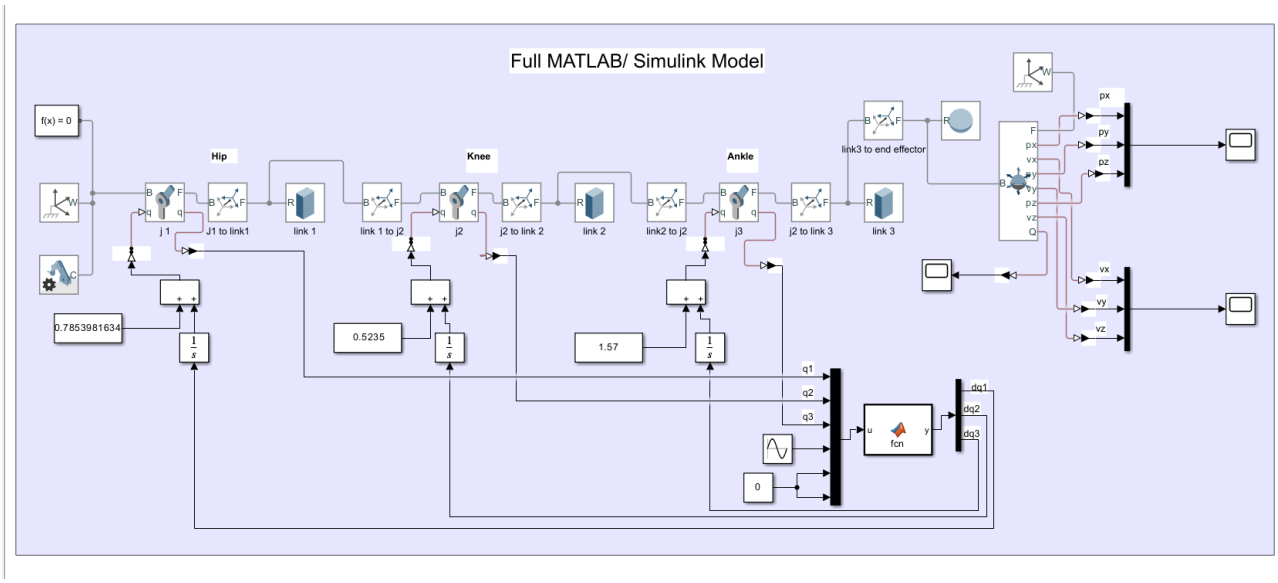

Figure 7 MATLAB /Simscape Multibody Simulation

Algorthim.2 Jacobian Matrix defined by the MATLAB function

```
function y = fcn(u)
a1 = 50; a2= 100;  a3= 100;

q1= u(1) ; q2= u(2) ; q3 = u(3);
dx= u(4) ; dy= u(5) ; dz = u(6);

jac=  [-sin(q1)*(a1 + a3*cos(q2 + q3) + a2*cos(q2)), -cos(q1)*(a3*sin(q2 + q3) + a2*sin(q2)), -a3*sin(q2 + q3)*cos(q1);
       cos(q1)*(a1 + a3*cos(q2 + q3) + a2*cos(q2)), -sin(q1)*(a3*sin(q2 + q3) + a2*sin(q2)), -a3*sin(q2 + q3)*sin(q1);
                                                 0,          - a3*cos(q2 + q3) - a2*cos(q2),         -a3*cos(q2 + q3);
                                                 0,                          -sin(q1),                 -sin(q1);
                                                 0,                           cos(q1),                  cos(q1);
                                                 1,                                 0,                       0];

y = (pinv(jac))* [dx ;dy;dz; 0;0;0];
```

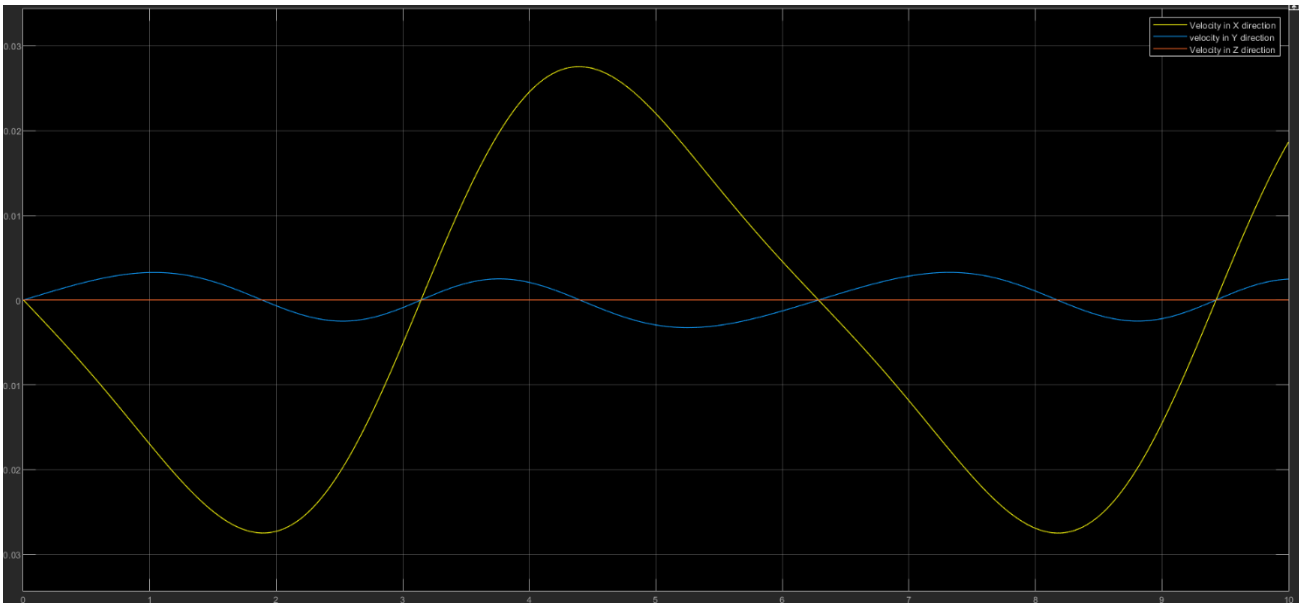

Figure 8 the velocity of end effector in X, Y, Z directions

The graph shows that the robot leg generates a velocity in the X direction and no velocity in Z and Y Axes. But, there is a very small error in robot leg velocity in Y direction because of the Jacobian 6*3 and there are three links, and also the error may be made because of the integration errors which be accumulated by the MATLAB function.

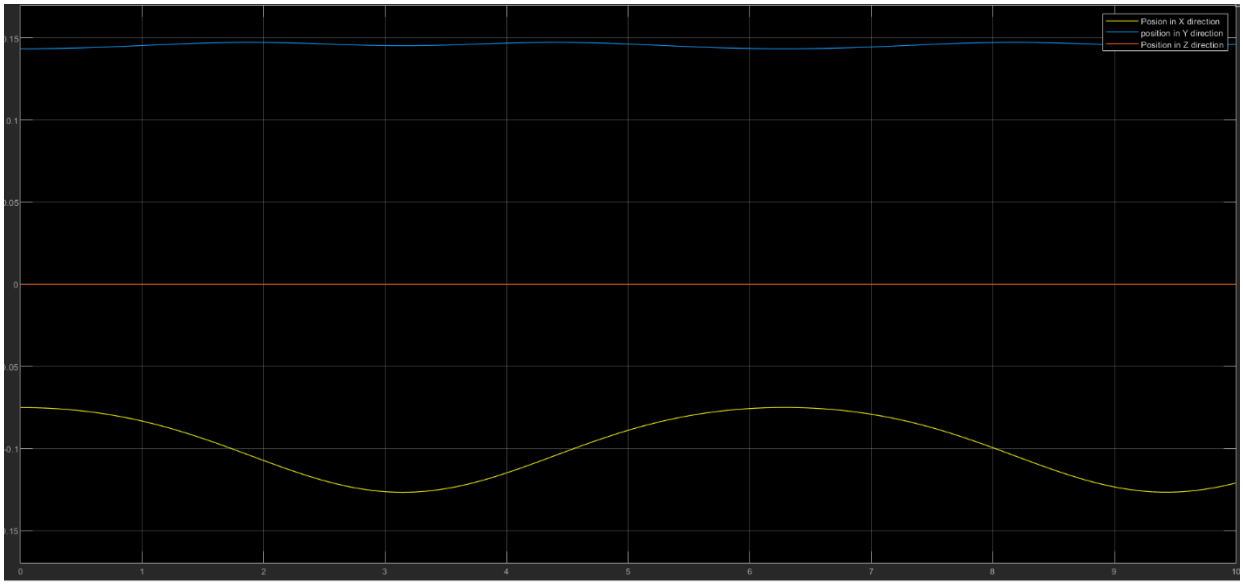

Figure 9 position of end effector in X, Y, Z Axes

The graph shows that the robot leg which generates a movement in the position of X direction, and the x position is changed with movement of the robot with sinusoidal function as shown in the graph, and no changes in y and Z position.

6- Conclusion

After showing the simulation of the hexapod robot leg using MATLAB Simulink and the Simscape Multibody environment. And using an algorithm that takes the trajectory position of the robot leg and using the Jacobian matrix it gives us the velocity of each joint and then, integrate it to give the each link the required angle. And after that we move the robot leg and It's proven that we can generate a forward step for the robot leg without wobbling the robot body.

References:

1- Mark W. Spong, Seth Hutchinson & M. Vidyasagar. (2020). Robot modeling and control. New York, USA: John Wiley & Sons, Ltd
2- Jan Valdman. (2016). Applications from Engineering with MATLAB Concepts. Rijeka, Croatia: InTech publisher
3- Peter Corke. Robotics Toolbox for MATLAB. (2012). https://petercorke.com/