UNIVERSITÉ DE TOULON

MIR MARINE & MARITIME INTELLIGENT ROBOTICS

ROBOTICS - MODELLING OF MECHANICAL SYSTEMS

2023

Assignment 1

Robotics: Modelling of kinematic chains (Scara Robot – Hexapod leg)

Submitted to :

Prof. Cedric Anthierens

Student Name: Abdelrahman ABDELHAMED

Marine and Maritime intelligent Robotics (MIR)

Abstract:

This report showcases work to include designing and simulating a hexapod robot leg using MATLAB Simulink and the Simscape Multibody environment. The objective was to model the leg and simulate its movement, specifically focusing on generating a forward step without causing any instability in the robot's body. To accomplish this, I utilized the Jacobian matrix as a MATLAB function, which enabled me to supply the necessary input signals to control the leg's joints.

1. **Hexapod Leg:**

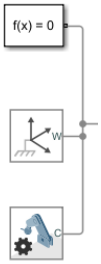
In this part, we leverage MATLAB to design and simulate a hexapod robot leg, with a primary focus on assessing its stability. Initially, we employ Simscape to create a CAD model of the leg, and subsequently, we subject this CAD model to a forward step motion along the X-axis in its simulated environment. This simulation serves to validate our capability to generate a forward leg movement without inducing unintended motion in the robot's entire body.

Figure 1 Diagram example for Hexapod.

1.1. Procedures to Design the CAD Model for hexapod leg:

To construct the CAD model, we will employ Simscape. Initially, we have three fundamental blocks that are essential components for creating any type of multibody element or mechanism when utilizing the Simscape Multibody toolbox:

a. **Solver configuration block:** This is used to define the type of ODE solver which will be used by the mechanism.
b. **World frame block:** This frame defines the coordinate system of all mechanisms which we built.
c. **Mechanism configuration:** This frame will define the coordinate system for all mechanisms.



Next, we proceed with constructing the robot leg from the hip to the end effector, involving the assembly of links and joints. **First**, we initiate the link creation process by navigating to the body elements in Simscape. Here, we

choose the solid type and customize its properties, including mass, inertia, geometry, and color. The frame of this body is automatically positioned at its center. To ensure that the world frame aligns with the base of the body, we employ a transformation. This involves accessing the 'transform and frames' section and selecting a rigid transform, which effectively realigns the world frame with the frame of the link. **Second**, to introduce the first joint, we access the '**multibody'** options and select the appropriate joint type. Before attaching this joint to the link, we need to relocate the joint's frame to the end of the link. Since the link's frame is at its center, we require an additional rigid transform.

We repeat these procedures for all subsequent links and joints, culminating in the completion of the CAD model for the robot leg.

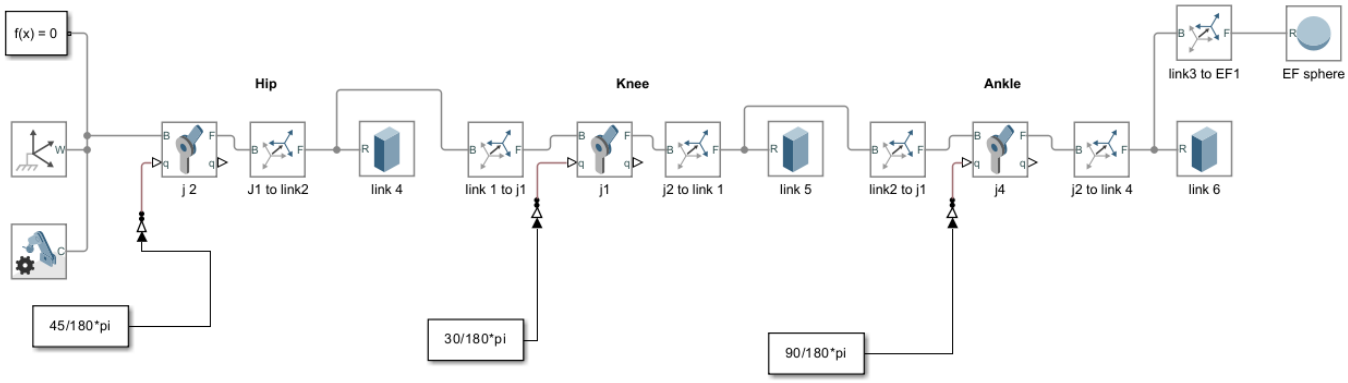


Figure 3 Simulink Block diagram of the CAD model for the hexapod leg.

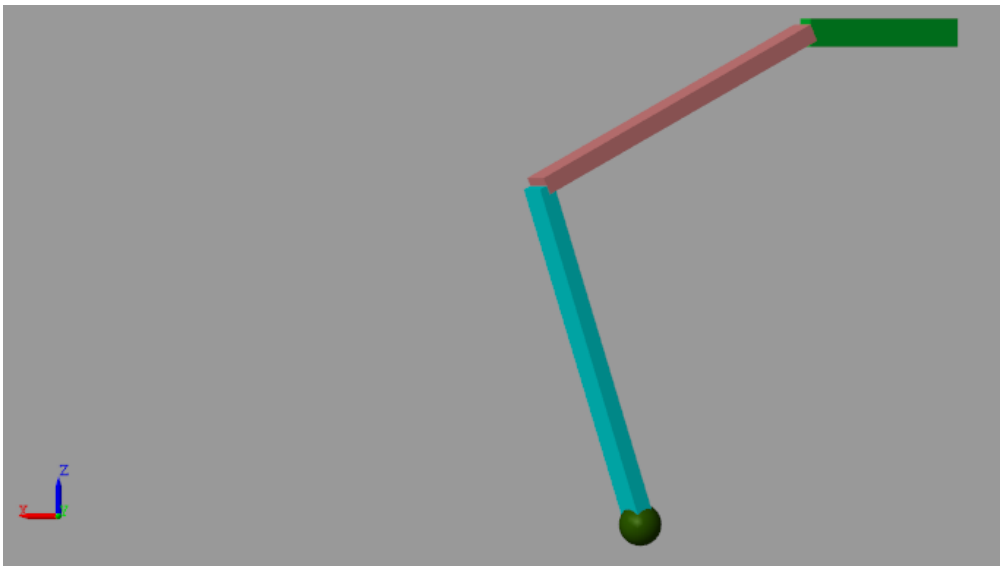


Figure 4 Hexapod model in MATLAB / SIMSCAPE.

1.2. Driving the Jacobian of the leg of the robot:

Currently, we must set the CAD model in motion within its operational environment, and there are two methods to instruct the robot to accomplish this task:

a. We have the option to instruct the joint to rotate to a specific angle, staying within the rotational limits of the joint. When the joints are rotated to these particular angles, the end effector will reach the desired point within the workspace. This is often referred to as the forward kinematics algorithm.

b. In practice, **we have knowledge of the desired location where the end effector should reach**, but **we lack the information about the specific angles that each joint should assume** to achieve this endpoint. Furthermore, there are multiple combinations of joint angles that can lead the robot to the same position in the workspace. Consequently, it's a challenging task to provide joint angles directly with the hope that the robot will arrive at the intended location. To address this, we require an alternative algorithm that takes the desired point in the workspace as input and calculates the joint angles. These angles, when executed by the robot's joints, will guide the end effector to the desired location. This is commonly referred to as the inverse kinematics algorithm.

We've developed an algorithm that processes the trajectory position of the robot leg. Leveraging the Jacobian matrix, which provides us with the joint velocities, we integrate this data to determine the necessary angles for each link.

According to the relation between the end effector's velocity and Jacobian Matrix:

$$V_n^0 = J_v \cdot \dot{q}$$

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

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} = J_{6\times n} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dots \\ \dot{q}_n \end{bmatrix}_{n\times 1} \quad (where\ n{=}number\ of\ joints \qquad where \quad \dot{q}\ \ represents\ each\ joint$$

x, y, z Represents the position of end effector and $\dot{x}, \dot{y}, \dot{z}$ represents the linear velocities.

1.3. Overall Kinematics of a hexapod leg:

In figure (5) represents the 2D frame of hexapod

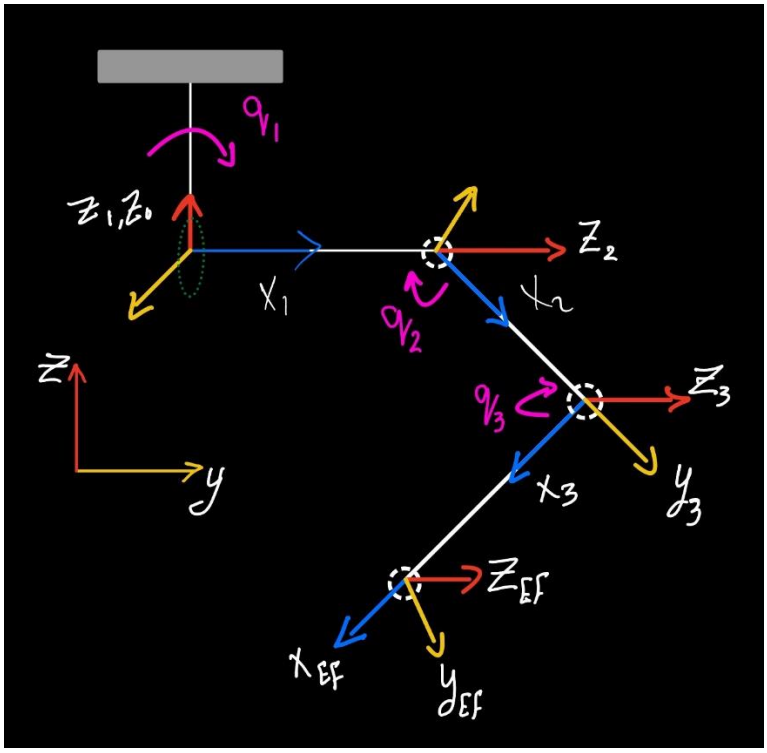


Figure 5 DH frames of the hexapod robot leg.

So we used the Denavit Hartenberg formalism to obtain the DH Parameters from the previous figure.

Table 1 DH Table of Hexapod Leg.

Link	σ	α	a	d_i	q_i
1	Rev	$-\pi/2$	a_1	0	q_1
2	Rev	0	a_2	0	q_2
3	Rev	0	a_3	0	q_3

We can obtain the Jacobian matrix by two methods:

a. Manually:

First we will compute T_0^1 , T_0^2 , T_0^3
Then we will find the Jacobian matrix by computing J_v and J_ω

So the solution is:

$$T_0^1 = A1$$

$$T_0^2 = A1 * A2$$

$$T_0^3 = A1 * A2 * A3$$

$$A1 = \begin{bmatrix} cos(q_1) & 0 & -sin(q_1) & a_1 * cos(q_1) \\ sin(q_1) & 0 & cos(q_1) & a_1 * sin(q_1) \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A2 = \begin{bmatrix} cos(q_2) & -sin(q_2) & 0 & a_2 * cos(q_2) \\ sin(q_2) & cos(q_2) & 0 & a_2 * sin(q_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$A3 = \begin{bmatrix} cos(q_3) & -sin(q_3) & 0 & a_3 * cos(q_3) \\ sin(q_3) & cos(q_3) & 0 & a_3 * sin(q_3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_0^1 = \begin{bmatrix} \boldsymbol{cos(q_1)} & \mathbf{0} & \boldsymbol{-sin(q_1)} & \boldsymbol{a_1 * cos(q_1)} \\ \boldsymbol{sin(q_1)} & \mathbf{0} & \boldsymbol{cos(q_1)} & \boldsymbol{a_1 * sin(q_1)} \\ \mathbf{0} & \mathbf{-1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

$$T_0^2 = \begin{bmatrix} \boldsymbol{cos(q_1) * cos(q_2)} & \boldsymbol{-cos(q_2) * sin(q_1)} & \boldsymbol{-sin(q_1)} & \boldsymbol{cos(q_1) * (a_1 + a_2 * cos(q_2)} \\ \boldsymbol{sin(q_1) * sin(q_2)} & \boldsymbol{-sin(q_1) * sin(q_2)} & \boldsymbol{cos(q_1)} & \boldsymbol{sin(q_1) * (a_1 + a_2 * cos(q_2)} \\ \mathbf{0} & \boldsymbol{-cos(q_2)} & \mathbf{0} & \boldsymbol{-a_2 * sin(q_2)} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

$$T_0^3 = \begin{bmatrix} \boldsymbol{cos(q_2 + q_3) * cos(q_1)} & \boldsymbol{-cos(q_2 + q_3) * sin(q_1)} & \boldsymbol{-sin(q_1)} & \boldsymbol{cos(q_1) * (a_1 + a_3 * cos(q_2 + q_3) + a_2 * cos(q_2)} \\ \boldsymbol{sin(q_2 + q_3) * sin(q_1)} & \boldsymbol{-sin(q_2 + q_3) * sin(q_2)} & \boldsymbol{cos(q_1)} & \boldsymbol{sin(q_1) * (a_1 + a_3 * cos(q_2 + q_3) + a_2 * cos(q_2)} \\ \boldsymbol{-sin(q_1 + q_2)} & \boldsymbol{-cos(q_1 + q_2)} & \mathbf{0} & \boldsymbol{-a_3 * sin(q_1 + q_2) - a_2 * sin(q_2)} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

$$\boldsymbol{J_1} = \begin{bmatrix} \boldsymbol{Z_0} \times (\boldsymbol{O_3} - \boldsymbol{O_0}) \\ \boldsymbol{Z_0} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{1} \end{bmatrix} \times \begin{bmatrix} cos(q_1) * (a_1 + a_3 * cos(q_2 + q_3) + a_2 * \cos(q_2)) \\ \sin(q_1) * (a_1 + a_3 * cos(q_2 + q_3) + a_2 * \cos(q_2)) \\ -a_3 * sin(q_2 + q_3) - a_2 * \sin(q_2) \end{bmatrix} \\ \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{1} \end{bmatrix} \end{bmatrix}$$

$$\boldsymbol{J_1} = \begin{bmatrix} -\sin(q_1) * (a_1 + a_3 * cos(q_2 + q_3) + a_2 * \cos(q_2)) \\ cos(q_1) * (a_1 + a_3 * cos(q_2 + q_3) + a_2 * \cos(q_2)) \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\boldsymbol{J_2} = \begin{bmatrix} \boldsymbol{Z_1} \times (\boldsymbol{O_3} - \boldsymbol{O_1}) \\ \boldsymbol{Z_1} \end{bmatrix}$$

$$= \begin{bmatrix} \begin{bmatrix} -\sin(q_1) \\ \cos(q_1) \\ 0 \end{bmatrix} \times \begin{bmatrix} cos(q_1) * (a_1 + a_3 * cos(q_2 + q_3) + a_2 * \cos(q_3)) - a_1 * \cos(q_1) \\ \sin(q_1) * (a_1 + a_3 * cos(q_2 + q_3) + a_2 * \cos(q_3)) - a_1 * \sin(q_1) \\ -a_3 * sin(q_2 + q_3) - a_2 * \sin(q_2) \end{bmatrix} \\ \begin{bmatrix} -\sin(q_1) \\ \cos(q_1) \\ 0 \end{bmatrix} \end{bmatrix}$$

$$\boldsymbol{J_2} = \begin{bmatrix} -\cos(q_1) * (a_3 * sin(q_2 + q_3) + a_2 * \sin(q_3)) \\ -\sin(q_1) * (a_3 * sin(q_2 + q_3) + a_2 * \sin(q_3)) \\ -a_3 * sin(q_2 + q_3) - a_2 * \sin(q_2) \\ -\sin(q_1) \\ \cos(q_1) \\ 0 \end{bmatrix}$$

$$\boldsymbol{J_3} = \begin{bmatrix} \boldsymbol{Z_2} \times (\boldsymbol{O_3} - \boldsymbol{O_2}) \\ \boldsymbol{Z_2} \end{bmatrix}$$

$$= \begin{bmatrix} \begin{bmatrix} -\sin(q_1) \\ \cos(q_1) \\ 0 \end{bmatrix} \times \begin{bmatrix} cos(q_1) * (a_1 + a_3 * cos(q_2 + q_3) + a_2 * \cos(q_3)) - \cos(q_1) * (a_1 + a_2 * \cos(q_2) \\ \sin(q_1) * (a_1 + a_3 * cos(q_2 + q_3) + a_2 * \cos(q_3)) - \sin(q_1) * (a_1 + a_2 * \cos(q_2) \\ -a_3 * sin(q_2 + q_3) \end{bmatrix} \\ \begin{bmatrix} -\sin(q_1) \\ \cos(q_1) \\ 0 \end{bmatrix} \end{bmatrix}$$

$$\boldsymbol{J_3} = \begin{bmatrix} -\cos(q_1) * (a_3 * sin(q_2 + q_3) \\ -\sin(q_1) * (a_3 * sin(q_2 + q_3) \\ -a_3 * cos(q_2 + q_3) \\ -\sin(q_1) \\ \cos(q_1) \\ 0 \end{bmatrix}$$

$J = [\boldsymbol{J_1}\ \boldsymbol{J_2}\ \boldsymbol{J_3}\ ]$

Then Jacobian matrix will be:

$$\boldsymbol{J} = \begin{bmatrix} -\sin(q_1) * (a_1 + a_3 * cos(q_2 + q_3) + a_2 * \cos(q_2)) & -\cos(q_1) * (a_3 * sin(q_2 + q_3) + a_2 * \sin(q_3)) & -\cos(q_1) * (a_3 * sin(q_2 + q_3) \\ cos(q_1) * (a_1 + a_3 * cos(q_2 + q_3) + a_2 * \cos(q_2)) & -\sin(q_1) * (a_3 * sin(q_2 + q_3) + a_2 * \sin(q_3)) & -\sin(q_1) * (a_3 * sin(q_2 + q_3) \\ 0 & -a_3 * sin(q_2 + q_3) - a_2 * \sin(q_2) & -a_3 * cos(q_2 + q_3) \\ 0 & -\sin(q_1) & -\sin(q_1) \\ 0 & \cos(q_1) & \cos(q_1) \\ 1 & 0 & 0 \end{bmatrix}$$

b. Using Robotics Toolbox:

Algorithm 1: Computing

```
%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 assembly 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 we run the code: we type" jacobian_matrix "in command window:

```
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]
```

1.4. Full model in MATLAB / SIMULINK:

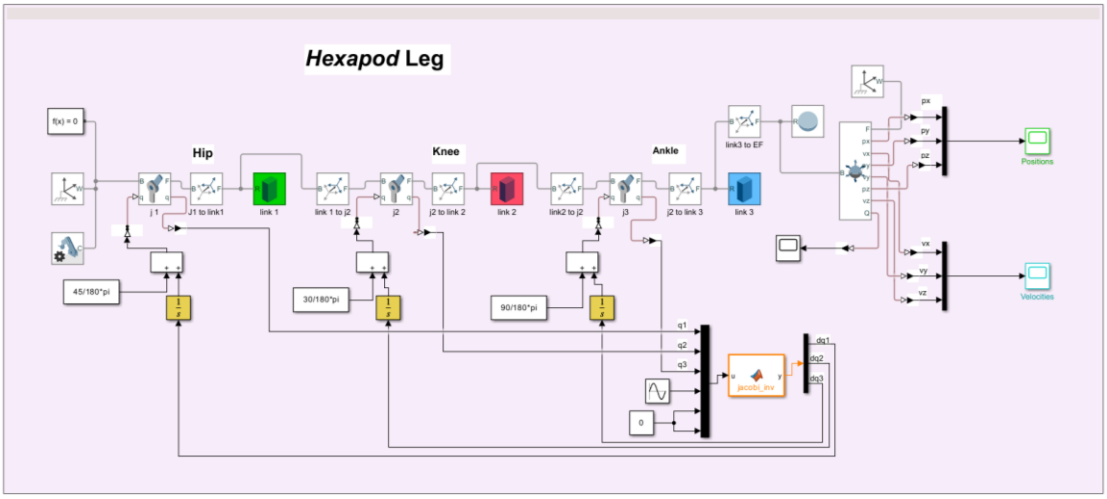


Figure 6 MATLAB / Simscape Multibody Simulation.

Algorithm 2 : Jacobian Matrix

```
function y = jacobi_inv(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);


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]


y = (pinv (jacobian__matrix))* [dx; dy;dz; 0;0;0];  %pinv as Jacobian matrix is not square matrix.
```

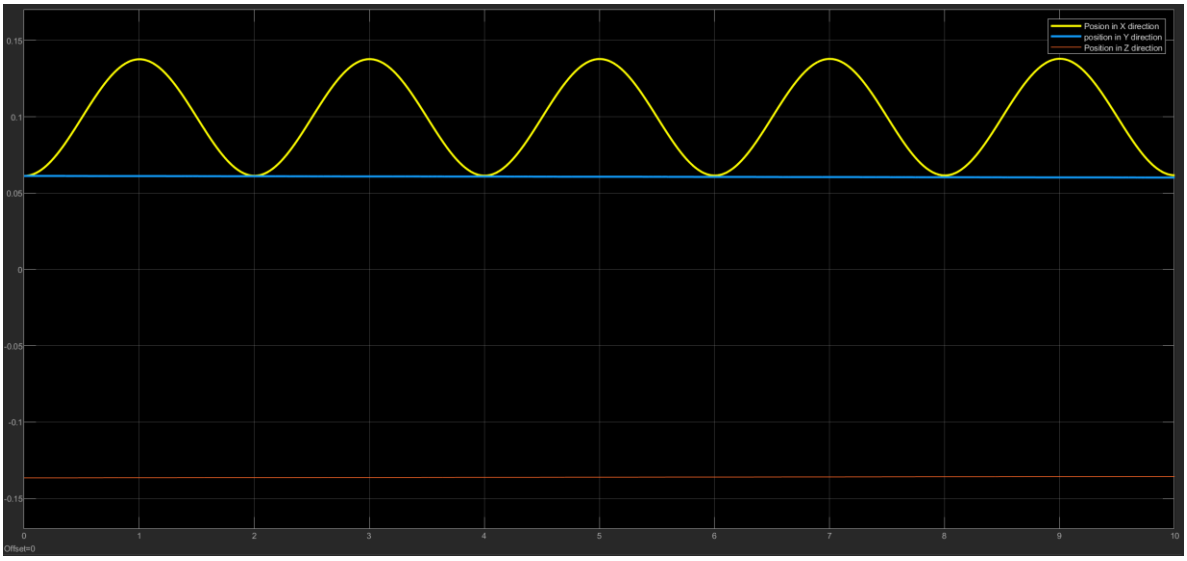


Figure 7 Position of End Effector in X, Y, Z Axes.

The graph demonstrates that the robot leg initiates motion along the X-axis, causing a sinusoidal variation in the X position as the robot moves. However, there are no oscillations observed in the Y and Z positions.

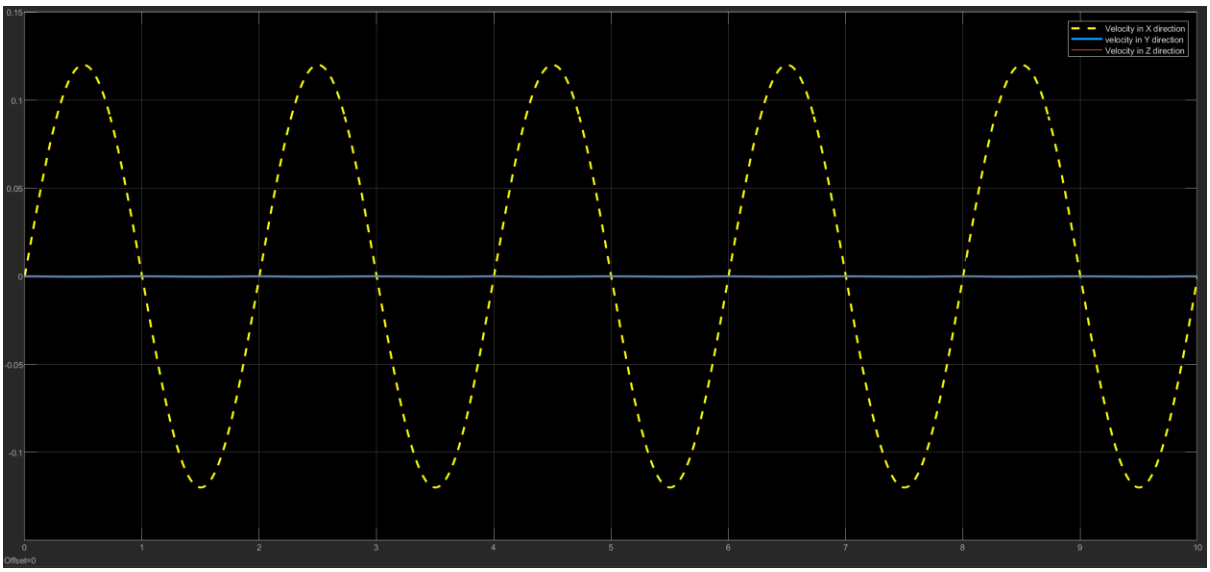


Figure 8 Velocities of X, Y, Z of The End Effector.

The graph illustrates that the robot leg exhibits velocity exclusively in the X direction, with negligible velocity observed in the Z and Y axes. However, a minor discrepancy in the robot leg's Y-direction velocity is discernible, possibly attributed to the use of a 6x3 Jacobian for a three-link system. It's also plausible that integration errors, which accumulate within the MATLAB function, contribute to this slight error.

1.5. Discussion of Hexapod

It is observable that motion in the x-axis occurs in isolation from motion in the other two axes. Furthermore, we can see from the simulation output in Fig. 8 that the Hexapod leg can only be controlled along the base X-axis because the other velocities are zero and the V's velocity changes only sinusoidal. The velocity profile shown demonstrates that the foot is moving in the intended manner.

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/