# 662 PROJECT-2 REPORT

## INTRODUCTION/APPLICATION

In our global community, millions of individuals confront mobility challenges arising from conditions such as paralysis, hemiplegia, cerebral palsy, Rett syndrome, Parkinson's, and more. Unfortunately, many of these patients find themselves bedridden for a significant portion of their lives due to the inability to walk. While modern rehabilitation therapies have made substantial strides, delivering commendable results, the road to improvement remains lengthy, requiring years of consistent training for observable enhancements.

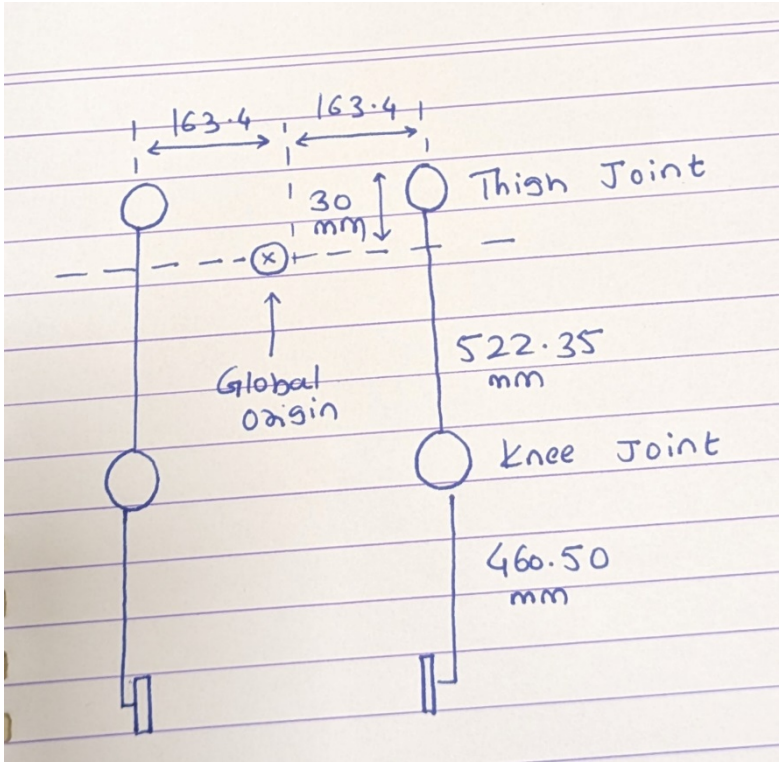
Our inspiration stems from a simple yet profound goal: to restore the joy of walking to those who've lost it. The solution involves patients wearing the exoskeleton and an EEG which will detect the brain signals and then the exoskeleton steps in, which will magnify the effort these individuals put into moving their legs. In this project, we are focusing on modelling and simulating leg joints.

## ROBOT TYPE

Humanoid Exoskeleton Robot

## DOF'S and Dimensions

The robot is a 6-DOF with 3DOF's on each left and right leg. For the human we have considered standard 95$^{th}$ percentile male, and the main link dimensions are as follows:



# 662 PROJECT-2 REPORT

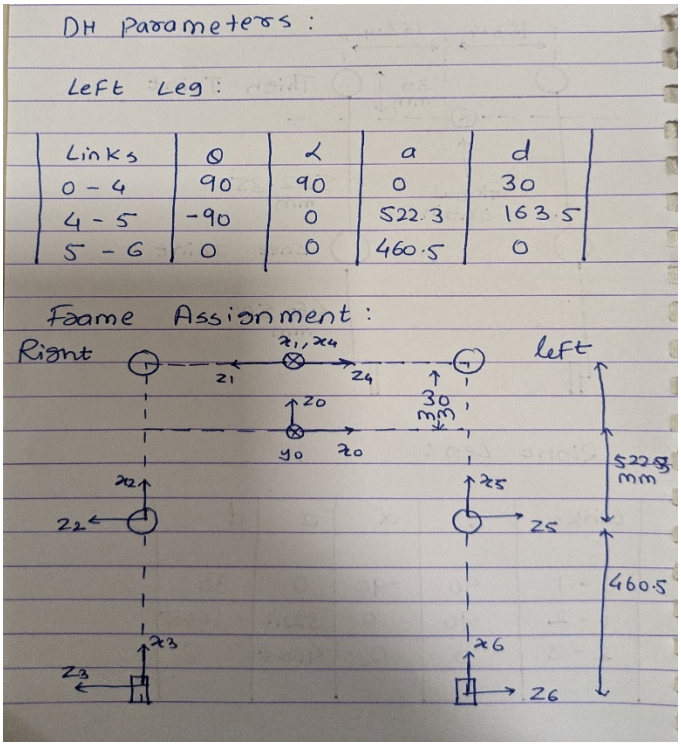
## CAD MODELS

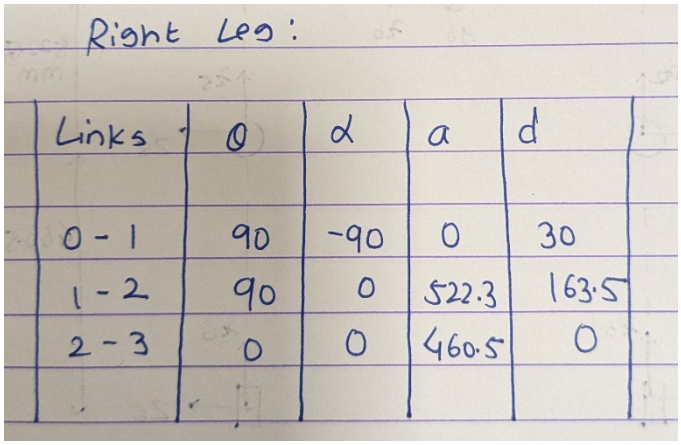
We have designed the whole model in SolidWorks, the images for the same are as follows:

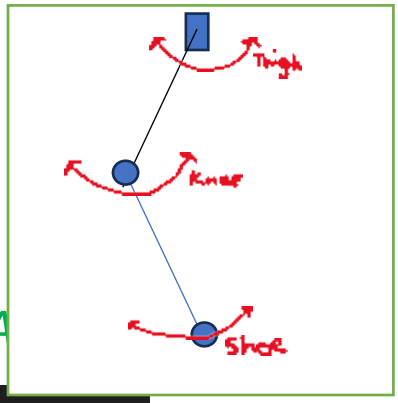
# 662 PROJECT-2 REPORT

## DH PARAMETERS

For the DH Table of our exoskeleton, we have considered two different end effectors for left and right leg, you can refer the below data for the same. We have also validated it using the Peter Corke toolbox.




## FORWA

```
θrad, αrad, a, d = sp.symbols('θdeg αdeg a d')
θ1, θ2, θ3 = sp.symbols('θ1 θ2 θ3')

# Created a function to print the Transformation Matrix of each joint wrt previous joint

def Transform_matrix(θrad, αrad, a, d):
    return sp.Matrix([
        [sp.cos(θrad), -sp.sin(θrad) * sp.cos(αrad), sp.sin(θrad) * sp.sin(αrad), a * sp.cos(θrad)],
        [sp.sin(θrad), sp.cos(θrad) * sp.cos(αrad), -sp.cos(θrad) * sp.sin(αrad), a * sp.sin(θrad)],
        [0, sp.sin(αrad), sp.cos(αrad), d],
        [0, 0, 0, 1]
    ])

#Writing all the original transformation matrices of each joint wrt the previous joints for RIGHT LEG
# old
def T1(θ1_):
    return Transform_matrix((θ1_ + sp.pi/2), - sp.pi/2, 0, 30)

def T2(θ2_):
    return Transform_matrix((θ2_ + sp.pi/2), 0, 522.4, 163.4)

def T3(θ3_):
    return Transform_matrix(θ3_, 0, 460.5, 0)
```

Step1 : Using the standard formula to derive Transformation Matrix in Spong Convention, we enlist the Transformation Matrix for every joint with respect to the previous joint.

# 662 PROJECT-2 REPORT

```
#Printing all the original transformation matrices of each joint wrt the base
T20 = T1(θ1) * T2(θ2)
T30 = T1(θ1) * T2(θ2) * T3(θ3)
```

Step 2 : Obtaining the Transformation Matrix with respect to the base.

$$\begin{bmatrix} 0 & 0 & -1 & -163.4 \\ 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & -952.9 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Step 3 : The Final Transformation Matrix. The end effector is situated at this position. As confirmed in the figures above.

## INVERSE KINEMATICS

```
# From the final transformation matrix, extracting the first three members of the final matrix
# to get the position of the end effector wrt base
P6 = T30[:3, -1]

# Declaring all the partial derivatives of the Position of end effector wrt the joint angles
dP_d01 = P6.diff(θ1)
dP_d02 = P6.diff(θ2)
dP_d03 = P6.diff(θ3)

# Obtaining the axis vector Z for each joint
Z1 = T1(θ1)[:3, 2]
Z2 = T20[:3, 2]
Z3 = T30[:3, 2]

# Jacobian matrix using the seconf method
J = sp.Matrix([
    [dP_d01, dP_d02, dP_d03],
    [Z1, Z2, Z3]
])
```

Step1: Using the Position Matrix from the Final Transformation Matrix. We derive the Jacobian Matrix using the second method.

```
    qdot = J0.pinv() * E

# Obtained the joint angle in each iteration
    q += qdot*(40/40)

# Obtained the updated value of Transformation Matrix after rotation and extracted the end-effector position for the
# Transformation Matrix
    T = T1(q[0]) * T2(q[1]) * T3(q[2])
    P6 = T[:3, -1]
    # print("P6 values")
    # print(P6)
    # print("q values")
    # print(q)

    # Storing the position points in a list to be plotted using Matplotlib
    trajectory_points.append(np.array([float(P6[0]), float(P6[1]), float(P6[2])]))
    desired_trajectory_points.append(np.array([0, float(y), float(z)]))
```

# 662 PROJECT-2 REPORT

Step 2 : We obtained the qdot values after obtaining the Jacobian Inverse Matrix. Since the Jacobian Matrix is 6x3 we obtain the inverse using pseudo inverse.

Step 3 : We update the q value using the above formula. Here we take the time step is as 1 second.

Step 4 : Obtained and plot the Position of End effector using the newly obtained Transformation

```
# LEFT FORWARD RIGHT BACKWARD Motion
            for i_val in range(0, 40):
                if float(q[2]) >  0.2 :
                    # y = 952.9  * sp.sin((2 * np.pi / 40) * i_val)
                    y_dot = 952.9 * sp.cos((2 * np.pi / 40) * i_val) * (2 * np.pi / 80)

                    # z = - 952.9  * sp.cos((2 * np.pi / 40) * i_val)
                    z_dot = 952.9 * sp.sin((2 * np.pi / 40) * i_val) * (2 * np.pi / 80)

                # Calculating the end effector velocity Matrix
                    E = sp.Matrix([
                        [0],
                        [y_dot],
                        [z_dot],
                        [0],
                        [0],
                        [0],
                    ])

                # Substituting joint angles to obtian Jacobian and Jacoian Inverse and the Joint angular velocity matrix
                    J0 = J.subs({
                        θ1: q[0], θ2: q[1], θ3: q[2]
                    })

                    qdot = J0.pinv() * E

                    time.sleep(0.08)
                # Obtained the joint angle in each iteration
                    q -= qdot*(40/80)
                    node1.get_logger().info('The hip-joint angle is %f' % q[2])
                    node1.get_logger().info('2nd loop')
                # Obtained the updated value of Transformation Matrix after rotation and extracted the end-effector position for the
                # Transformation Matrix
                    # T = T1(q[0]) * T2(q[1]) * T3(q[2])
                    # P6 = T[:3, -1]


                    joint_positions = Float64MultiArray(layout=MultiArrayLayout(
                        dim=[MultiArrayDimension(label="joint_positions", size=2, stride=1)],
                        data_offset=0
                    ))
                    # joint_positions.data = [-float(q[2]), -float(q[1]), -float(q[2]), float(q[2]), float(q[1]), -float(q[2])]
                    joint_positions.data = [-float(q[2]), -float(q[1]), -float(q[2]), -float(q[2]), -float(q[2]), float(q[2])]
                    joint_position_pub1.publish(joint_positions)
```

Matrices

# 662 PROJECT-2 REPORT

```
# # LEFT BACKWARD RIGHT FORWARD Motion
        for i_val in range(0, 40):
            if float(q[2]) < 0.5 :
                # y = 952.9  * sp.sin((2 * np.pi / 40) * i_val)
                y_dot = 952.9 * sp.cos((2 * np.pi / 40) * i_val) * (2 * np.pi / 80)

                # z = - 952.9  * sp.cos((2 * np.pi / 40) * i_val)
                z_dot = 952.9 * sp.sin((2 * np.pi / 40) * i_val) * (2 * np.pi / 80)

            # Calculating the end effector velocity Matrix
                E = sp.Matrix([
                    [0],
                    [y_dot],
                    [z_dot],
                    [0],
                    [0],
                    [0],
                ])

            # Substituting joint angles to obtian Jacobian and Jacoian Inverse and the Joint angular velocity matrix
                J0 = J.subs({
                    θ1: q[0], θ2: q[1], θ3: q[2]
                })

                qdot = J0.pinv() * E

                time.sleep(0.08)
            # Obtained the joint angle in each iteration
                q += qdot*(40/80)
                node1.get_logger().info('The hip-joint angle is %f' % q[2])
                node1.get_logger().info('1st loop')
            # Obtained the updated value of Transformation Matrix after rotation and extracted the end-effector position for the
            # Transformation Matrix
                # T = T1(q[0]) * T2(q[1]) * T3(q[2])
                # P6 = T[:3, -1]

                joint_positions = Float64MultiArray(layout=MultiArrayLayout(
                    dim=[MultiArrayDimension(label="joint_positions", size=2, stride=1)],
                    data_offset=0
                ))
                # joint_positions.data = [float(q[2]), float(q[2]), float(q[2]), float(q[2]), float(q[2]), float(q[2])]
                joint_positions.data = [-float(q[2]), -float(q[1]), -float(q[2]), -float(q[2]), -float(q[2]), float(q[2])]
                joint_position_publ.publish(joint_positions)

            # joint_positions.data = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
            # joint_position_publ.publish(joint_positions)
            node1.get_logger().info('Bringing in between')
```

# 662 PROJECT-2 REPORT

```
# LEFT FORWARD RIGHT BACKWARD Motion 3rd loop
            for i_val in range(0, 40):
                if float(q[2]) <  0.5 :
                    # y = 952.9  * sp.sin((2 * np.pi / 40) * i_val)
                    y_dot = 952.9 * sp.cos((2 * np.pi / 40) * i_val) * (2 * np.pi / 80)

                    # z = - 952.9  * sp.cos((2 * np.pi / 40) * i_val)
                    z_dot = 952.9 * sp.sin((2 * np.pi / 40) * i_val) * (2 * np.pi / 80)

                # Calculating the end effector velocity Matrix
                    E = sp.Matrix([
                        [0],
                        [y_dot],
                        [z_dot],
                        [0],
                        [0],
                        [0],
                    ])

                # Substituting joint angles to obtian Jacobian and Jacoian Inverse and the Joint angular velocity matrix
                    J0 = J.subs({
                        θ1: q[0], θ2: q[1], θ3: q[2]
                    })

                    qdot = J0.pinv() * E

                    time.sleep(0.08)
                # Obtained the joint angle in each iteration
                    q += qdot*(40/80)
                    node1.get_logger().info('The hip-joint angle is %f' % q[2])
                    node1.get_logger().info('3rd loop')
                # Obtained the updated value of Transformation Matrix after rotation and extracted the end-effector position for the
                # Transformation Matrix
                    # T = T1(q[0]) * T2(q[1]) * T3(q[2])
                    # P6 = T[:3, -1]


                    joint_positions = Float64MultiArray(layout=MultiArrayLayout(
                        dim=[MultiArrayDimension(label="joint_positions", size=2, stride=1)],
                        data_offset=0
                    ))
                    # joint_positions.data = [-float(q[2]), -float(q[1]), -float(q[2]), float(q[2]), float(q[1]), -float(q[2])]
                    joint_positions.data = [float(q[2]), float(q[1]), float(q[2]), float(q[2]), float(q[2]), -float(q[2])]
                    joint_position_pub1.publish(joint_positions)

            joint_positions.data = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
            joint_position_pub1.publish(joint_positions)
            node1.get_logger().info('Bringing in between 2')
```

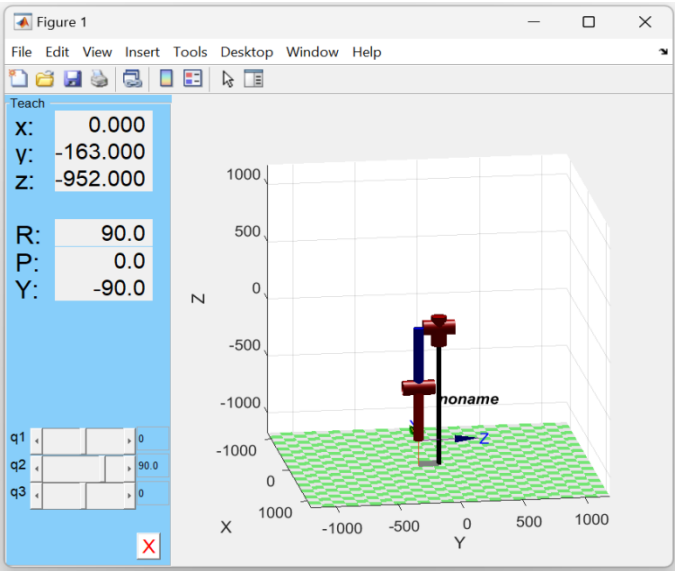
We divided the trajectory into three parts as shown in the figures above. Constraining the joint angles between certain limits, we used the Inverse Kinematics trajectory and published data every 0.25 seconds to obtain a smooth walking motion for our URDF Model.

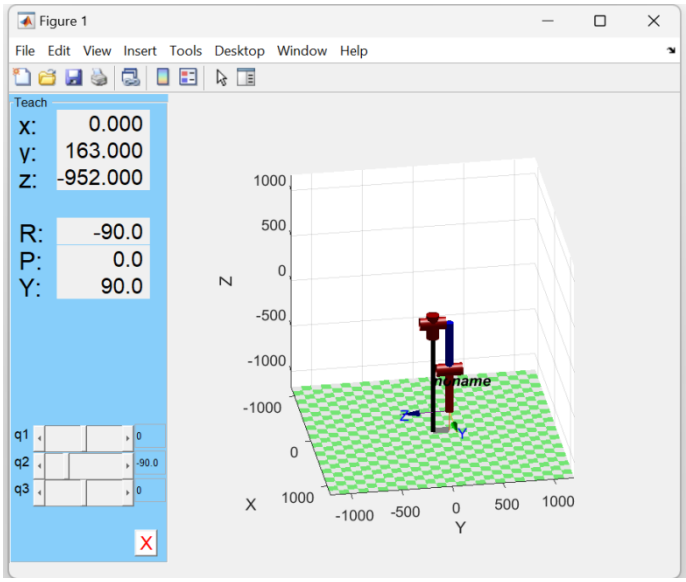
$$\begin{bmatrix} 0 & 0 & 0 \\ -163.4 & -982.9 & -460.5 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

This is the Final printed Jacobian Matrix.

# 662 PROJECT-2 REPORT

## FORWARD KINEMATICS VALIDATION




We have validated our joint positions using Peter Corke tool-box and also using Gazebo model. The following are some snaps. The reason for taking these angles for geometric validation is that we have constrained our urdf model with revolute joints up to the respective angle.

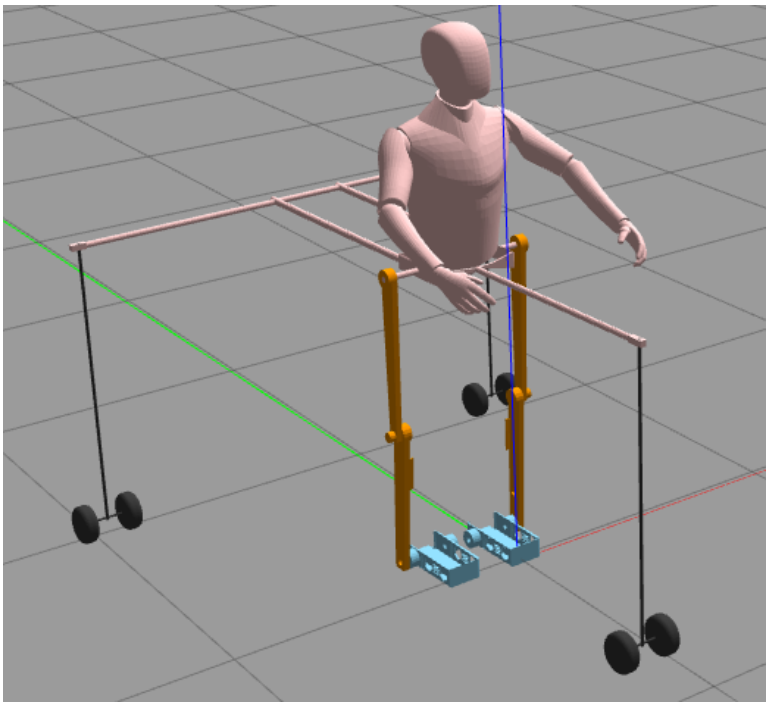


Fig: Idle Position

# 662 PROJECT-2 REPORT

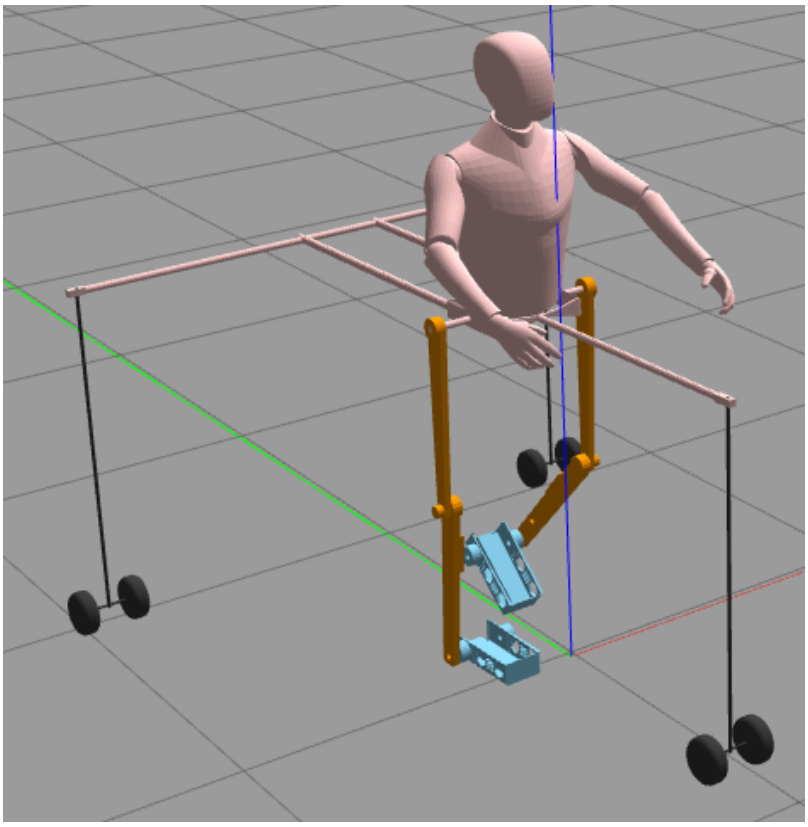


Fig: Second Joint is 60 degrees.

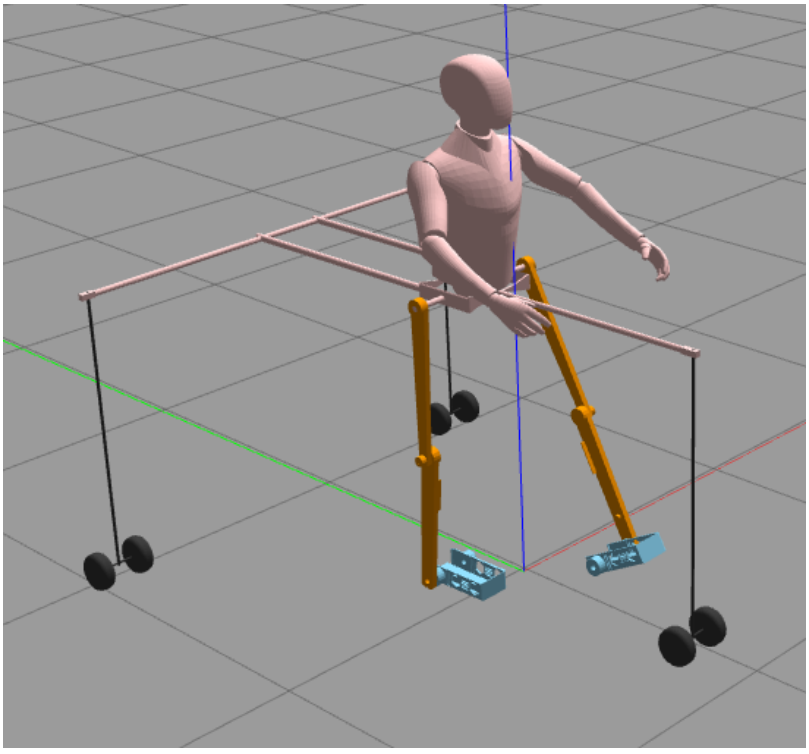


Fig : First joint is 30 degrees

# 662 PROJECT-2 REPORT

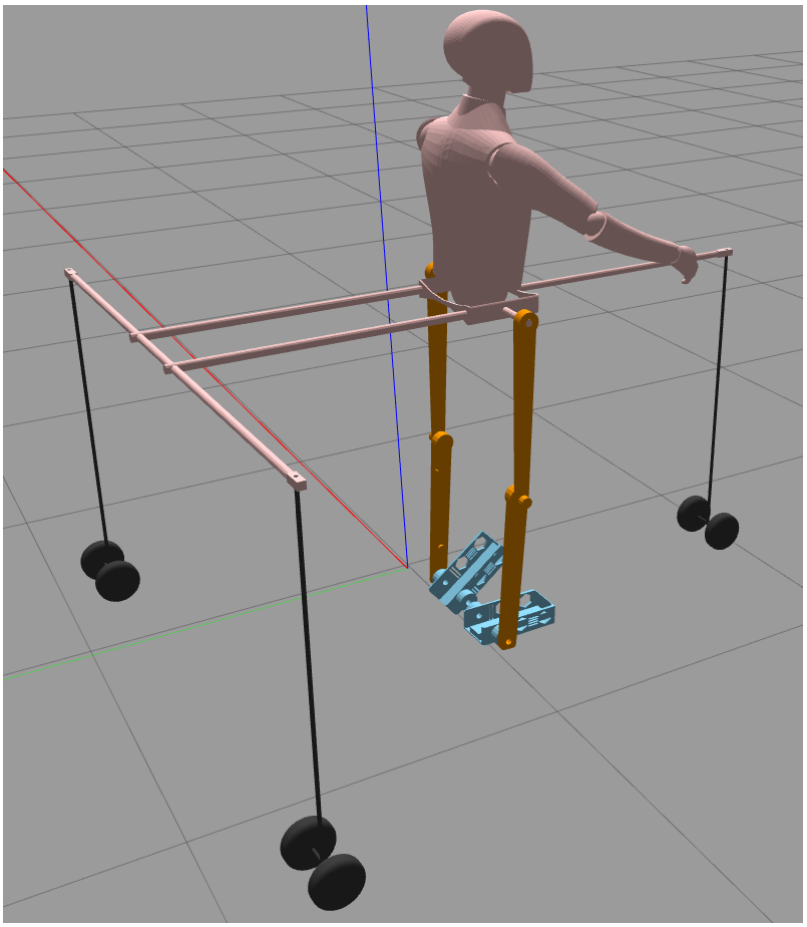


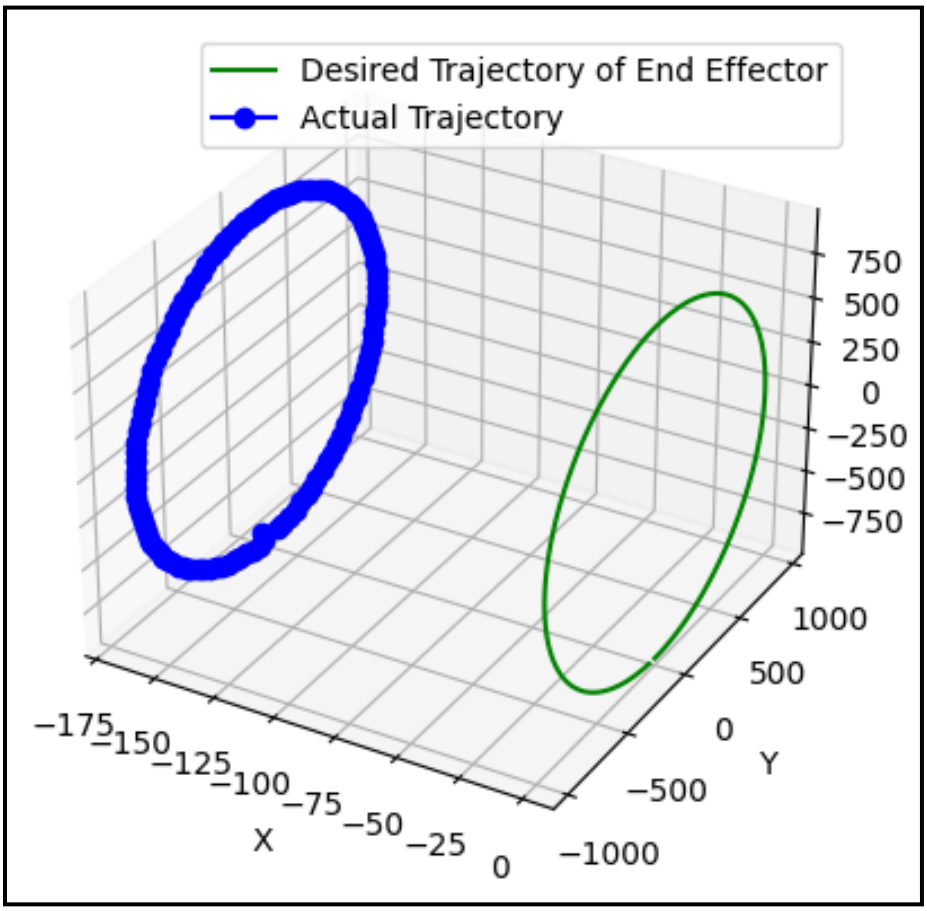
Fig: Third Joint is 45 degrees.

```
When third joint is 45 deg
⎡        0                   0           -1       -163.4      ⎤
⎢                                                               ⎥
⎢-0.707106665647094  -0.707106896725982  0   -325.622619530487⎥
⎢                                                               ⎥
⎢-0.707106896725982  0.707106665647094   0   -818.022725942315⎥
⎢                                                               ⎥
⎣        0                   0           0           1        ⎦
When first joint is 30 deg
⎡0   0.500000194337561   -0.866025291583566  -141.508532644755⎤
⎢                                                             ⎥
⎢0   -0.866025291583566  -0.500000194337561  -81.7000317547575⎥
⎢                                                             ⎥
⎢-1          0                   0                -952.9     ⎥
⎢                                                             ⎥
⎣0           0                   0                  1        ⎦
When second joint is 60 deg
⎡        0                   0           -1       -163.4      ⎤
⎢                                                               ⎥
⎢-0.866026628183543  -0.499997879272546  0   -851.217572841605⎥
⎢                                                               ⎥
⎢-0.499997879272546  0.866026628183543   0   -461.447915536985⎥
⎢                                                               ⎥
⎣        0                   0           0           1        ⎦
```

Fig: Positions of the end effector for the above configurations.

# ***662 PROJECT-2 REPORT***

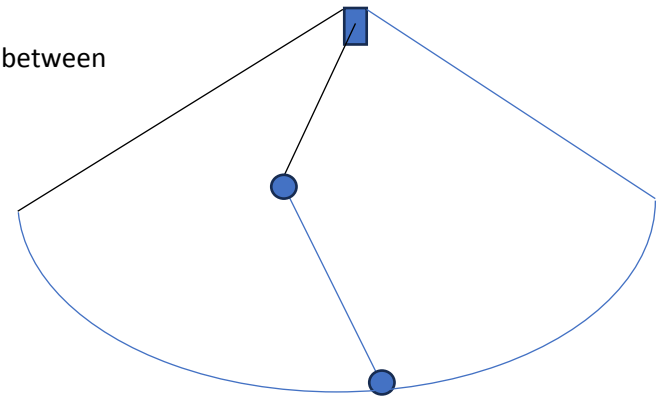
## INVERSE KINEMATICS VALIDATION




## WORKSPACE STUDY

The workspace of our exoskeleton varies between

-0.7 to +0.7 radians in an arc.



Detailed Study on human gait and then

Implementing that into the project stills

Needs to be carried out.

## ASSUMPTIONS

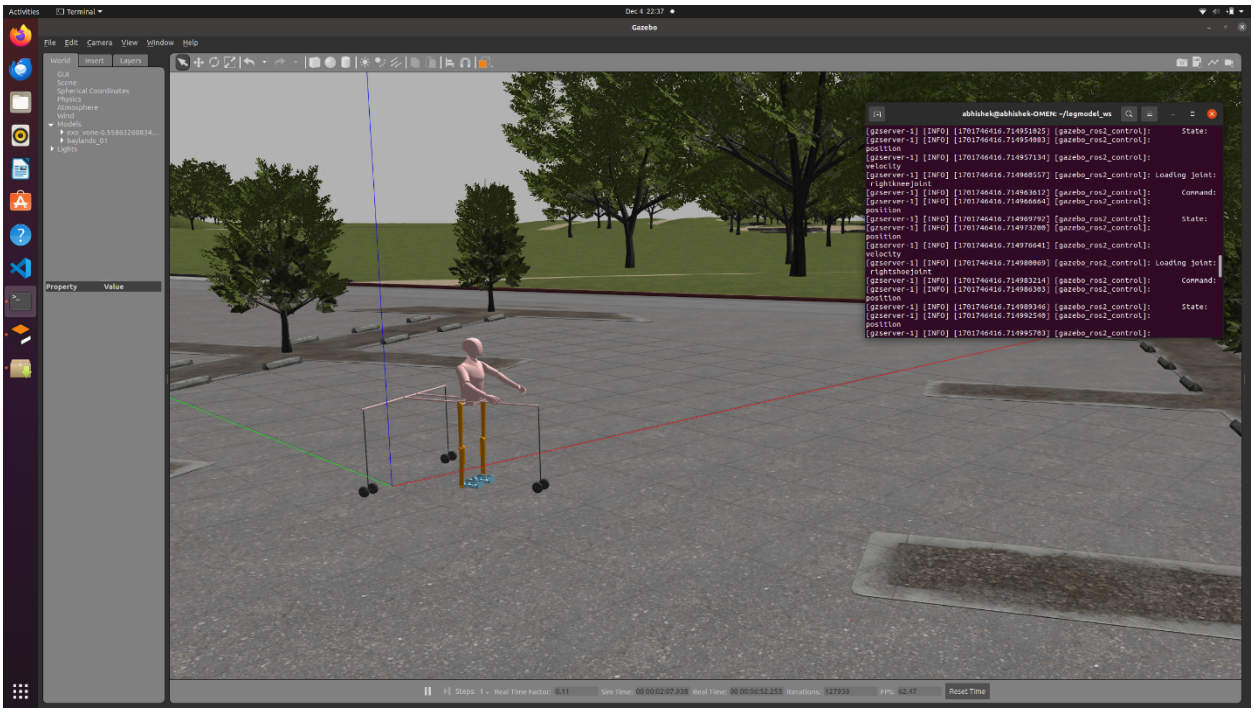
- The Human is 95$^{th}$ percentile male.
- Supporting Structure is built so that the model doesn't fall.

# 662 PROJECT-2 REPORT

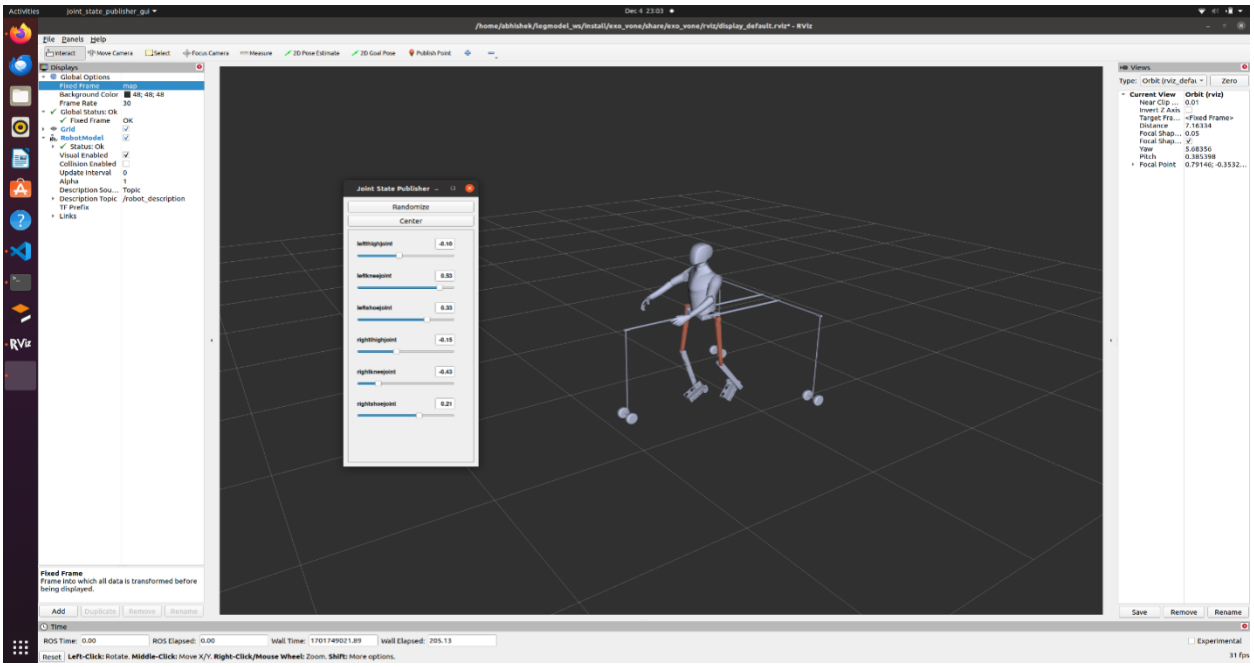
## CONTROL METHOD

Simple Open-Loop Controller.

## GAZEBO: Exoskeleton in Gazebo with Environment



## RVIZ: Exoskeleton in RVIZ with joint state publisher.



# ***662 PROJECT-2 REPORT***

## PROBLEMS FACED

The biggest problem we faced was analyzing human gait accurately, and then programming the controller accordingly.

## LESSON LEARNED

The biggest lesson we learned was to build as simple geometry as possible, else it gets very difficult to move the joints later in simulation. Moreover, the joint constraints should be given by thinking analytically.

## CONCLUSIONS

We successfully designed and simulated the Exoskeleton using a variety of concepts taught in class. Theres a lot more work to do in order to make the project self-sustainable as we are using frame to support the dummy, and that will be our future work.

## FUTURE WORK

Embarking on our mission to restore mobility, we recognize the myriad challenges that stand between us and our ultimate vision of enabling patients to walk. In our next phase, we're set to revamp the design, and with the application of control theory, we aim to implement stability algorithms. This strategic approach ensures the exoskeleton not only supports but sustains itself, marking a crucial step forward in overcoming obstacles on our journey to realizing our ambitious goal.

## REFERENCES

We haven’t referred to any online resources as we plan to continue this research in the future, and we want to make it novel. Although, we have referred to various concepts that was taught in class by professor and teaching assistants.