In [1]:
```
# SHAWN DIMANG RAS 557 ASSIGNMENT 5: PARAMETER IDENTIFICATION
```

## For this individual assigment I was assigned stiffness and damping parameters of a flexure joint. I used the cardboard stock paper that was supplied and used in the class room activity of creating the tested beams. I used and created three different beams to compare stiffness and damping parameters in Tracker and mujoco comparison.The goal of this experiment is to identify values in stiffness and damping for the given material that my team will use for the final foldable robitic mechanism we will create. In order to simulate the model in mujoco to a real life cpmoarison having accurate link and joint movements is crucial To set up the experiment we created three links with different mass, length of link, length to the table, inertia force, and width. All Links were the same distance and level to the camera. The camera used was a google pixel. The videos are input into tracker where the values of time over x and y corrdinates are output. The values are then input into excel then into jupterlab where the values are compared and recreated from the textbook walkthrough. I input the mass, length of the link, length to the table, and link width of all three links in seperate codes. I only adjusted the k (stiffness), b (damping) and n (number of mujoco bodies) values in each link experiment to match the tracker values. The wavelengths were not exaclty lined up but the starting and end behavior were accurate. 1. What could you have done better in your experiment design and setup? I would say setting up the link in camera view. In the beginning of one of my video data collections for a link, my hand covered the inital release of the link so it didnt capture that. But the data was still useable. For a future experiment I will use a camera setting will higher frames for second for a fast system. And to get better data I may cut the link peices that I used into thicker cuts. the tracker data was modeled great but some key frames may be slightly off so maybe manually clicking each key frame would help versus the tracker search.2. Discuss your rationale for the model you selected. Describe any assumptions or simplilcaitons this model makes. Include external references used in selecting or understanding your model? I chose this material becuase it will be the final material for the project. I chose the width and length of the links to get an idea of how I want to design the 4-bar legs of the Grasshopper mechanism. Assumptions made in the mujoco experiment include costant damping and stiffness with ignoring air resistance. External references include the class textbook, tracker, and mujoco walkthroughs. 3. Justify the method you selected (least squares, nonlinear least squares, scipy.optimize.minimize(), Evolutionary algorithm, etc. ) for ltting experimental data to the model, as well as the special algorithm used. I believe the method selected works because this experiment is supposed to be a quick and fast way to prototype a model in real life and theoritically. It is low cost and still uses the same identiy parameter concepts. Using the optimize.minimize does reduce error accuratly to show real life behavior. Another algorithm used was the mujoco number of bodies code provided in the textbook.4. How well does your data lt the model you selected? Provide a numerical value as well as a qualitative analysis, using your figure to explain. For all my models I would say that I was able to match the aplitudes and settling value well in all three models. The things that vary were the period wavelengths and wavelengths. It was very hard to match the time period without the system settling at a completely differnt height as the tracker values. Most of the values I got 10-15 error rate which is not ideal but still gives an idea of how the system operates and can be modifed.

# Beam Compare Values

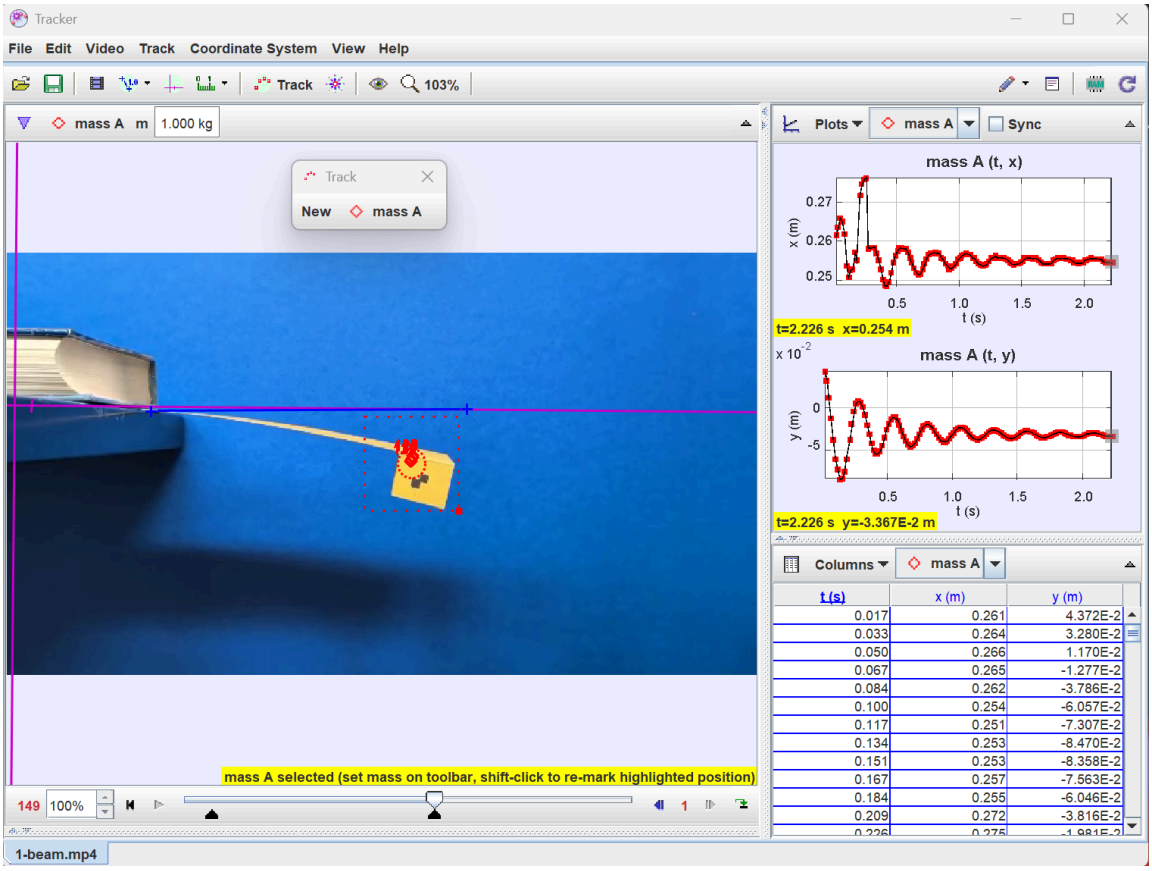
| **Test 1** | Tracker | Mujoco | Difference \| \| |
|---|---|---|---|
| Settling Point | -0.05 m | -0.045 | 0.005 |
| Amplitude (meter) | .03 | .02 | .01 |
| Wavelength (peak to peak) | .5 | .65 | .15 |
| Period | .25 | .75 | .5 |

| **Test 2** | Tracker | Mujoco | Difference \| \| |
|---|---|---|---|
| Settling Point | -0.055 m | -0.05 | 0.005 |
| Amplitude (meter) | .02 | .01 | .01 |
| Wavelength (peak to peak) | .5 | .6 | .1 |
| Period | .5 | 1.1 | .6 |

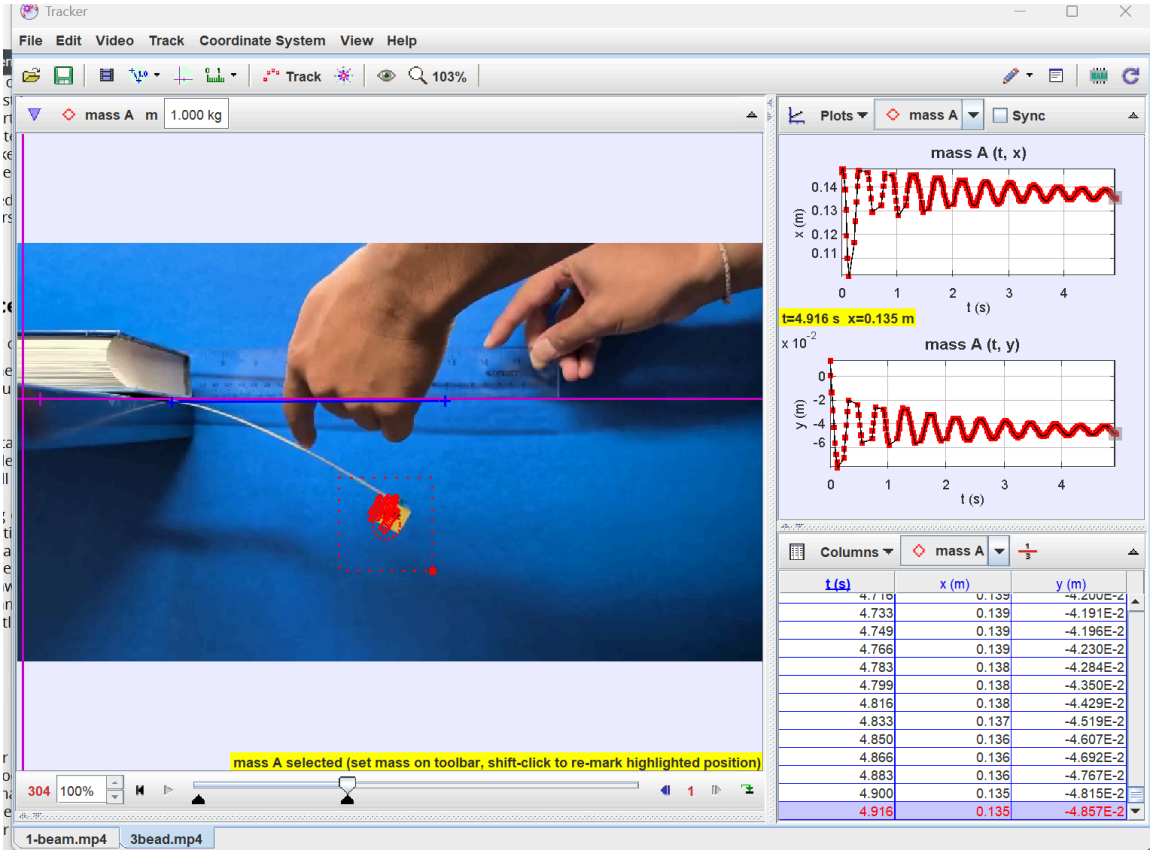
| **Test 3** | Tracker | Mujoco | Difference \| \| |
|---|---|---|---|
| Settling Point (meter) | -0.022 | -0.023 | 0.001 |
| Amplitude (meter) | .02 | .044 | .024 |
| Wavelength (peak to peak) | .14 | .25 | .11 |
| Period | .14 | .3 | .16 |

5. What are the limits of your model, within which you are conldent of a good lt? Do you expect your system to operate outside of those limits? The limits to my model is have a constant stiffness and damping system that does not vary. It is all uniform. It is difficult to display where the link is released from or pushed down to start the simulation. My Mujoco model simulates the motion of amplitude, settling point, and the wavelength. The only thing that needs to adjusted is the period. This could be changed with an offset of the tracker or the mujoco. To improve the period to be faster I would have to have less child bodies in the model and from there adjust the K & B values so that it brings the settling point up/down to match the joint link settling point. I expect my model to operate within these limits if outside vairiables are controlled. But in reality I do expect the model to reach outside limits because I still need to refiene how I adjust K and B values to match. I have tried inputting different numbers for a good amountof time for just one stystem, this is the closest I could get. For controls I would maybe put it into matlab and have that give a optimized stiffness and damping number.

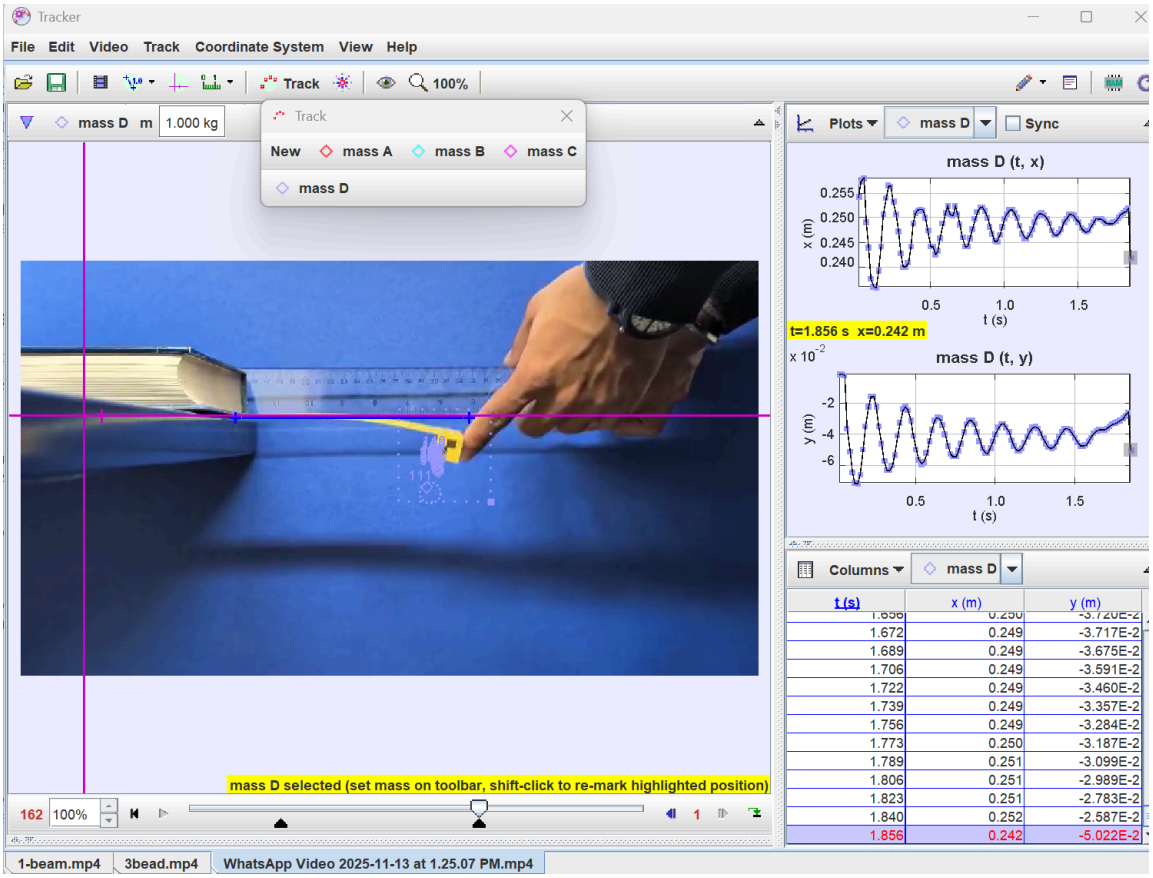
## Beam 1 Tracker




## Beam 2 Tracker




## Beam 3 Tracker



In [2]:
```
import os
import mujoco
import numpy
import mediapy as media
import matplotlib.pyplot as plt
import scipy.optimize as so
```

In [3]:
```
k_total = 2.1 #stiffness +k=+freq
b_total = .04 #damping
mass= .009 #mass of joint
```

In [4]:
```
main_template = """
<mujoco>
    <option gravity="0 0 -9.81">
    <flag contact="disable"/>
    </option>
    <worldbody>
        <light name="top" pos="0 0 1"/>
        <body name="body_1" pos="0 0 0">
            <joint name="joint_1" type="hinge" axis="0 1 0" pos="0 0 0"
                stiffness="{k}" damping="{b}"/>
            <geom type="box" size="{l1_2} .025 .0005" pos="{l1_2} 0 0" rgba="0 1 0
            mass="{m}"/>
            {inner}
        </body>
    </worldbody>
</mujoco>
"""
body_template = '''
<body name="body_{ii}" pos="{l1} 0 0">
    <joint name="joint_{ii}" type="hinge" axis="0 1 0" pos="0 0 0"
    stiffness="{k}" damping="{b}"/>
    <geom type="box" size="{l1_2} .025 .0005" pos="{l1_2} 0 0" rgba="1 0 0 1" mass=
    {inner}
</body>
'''

test_point='''
<geom type="sphere" size=".01" pos="{l1} 0 0" rgba="0 0 1 1" mass=".8"/>
'''
```

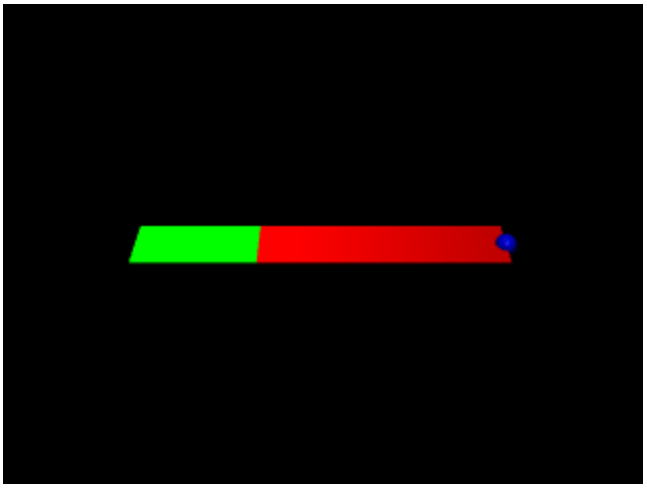
Mujoco xml model from text book. Inserting joint size and body template for bodies 2+

In [5]:
```
def run_sim(num_joints,k_i,b_i,m_total,render=False,show_video=False):
    n_joints = num_joints
    n_bodies = n_joints
    l = 1.2
    l_i=l/n_bodies
    m_i=m_total/n_bodies

    body_numbers = numpy.r_[n_bodies:1:-1]

    s=''
    for item in body_numbers:
        if item == body_numbers.max():
            s = body_template.format(inner=test_point.format(l1='{l1}'),ii=item,
```

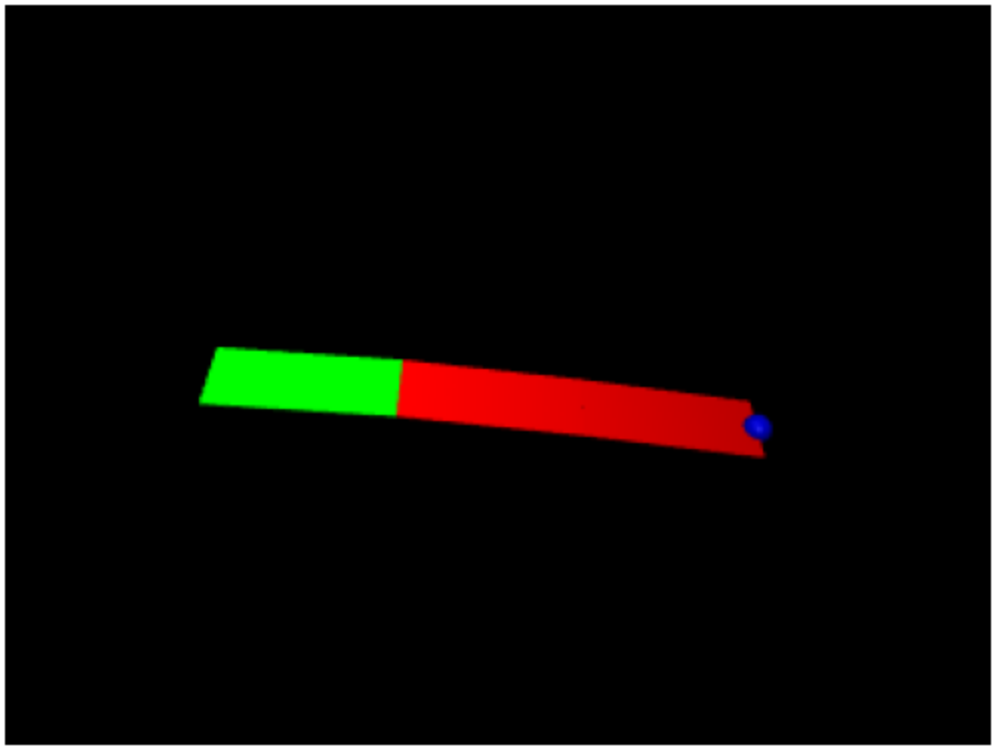
```
                                    l1='{l1}',l1_2='{l1_2}',b='{b}',k='{k}',m='{m}'
    else:
        s = body_template.format(inner=s,ii=item,l1='{l1}',l1_2='{l1_2}',
                                 b='{b}',k='{k}',m='{m}')

    s=main_template.format(inner = s,l1='{l1}',l1_2='{l1_2}',b='{b}',k='{k}',m='{m}

    xml = s.format(l1=l_i,l1_2=l_i/2,k=k_i,b=b_i,m=m_i)

    model = mujoco.MjModel.from_xml_string(xml)
    data = mujoco.MjData(model)
    renderer = mujoco.Renderer(model)

    duration = 3 # (seconds)
    framerate = 30 # (Hz)

    q = []
    t = []
    xyz = []
    frames = []
    mujoco.mj_resetData(model, data)

    while data.time < duration:
        mujoco.mj_step(model, data)
        q.append(data.qpos.copy())
        xyz.append(data.xpos.copy())
        t.append(data.time)
        if render:
            if len(frames) < data.time * framerate:
                renderer.update_scene(data)
                pixels = renderer.render()
                frames.append(pixels)

    if render:
        if show_video:
            media.show_video(frames, fps=framerate,codec='gif')

    t = numpy.array(t)
    xyz = numpy.array(xyz)

    if render:
        return t, xyz, frames
    else:
        return t, xyz
```

Creates and renders the mujoco model for 3 seconds and 30 frames.

In [6]:
```
nn = 10
k_ii = 1/((1/k_total)/nn)
b_ii = 1/((1/b_total)/nn)
t,xyz_100_elements,frames_100_elements = run_sim(num_joints = nn,k_i = k_ii, b_i =
                                                 m_total=1,render=True,show_video=Tr

plt.imshow(frames_100_elements[15])
plt.axis('off')
```



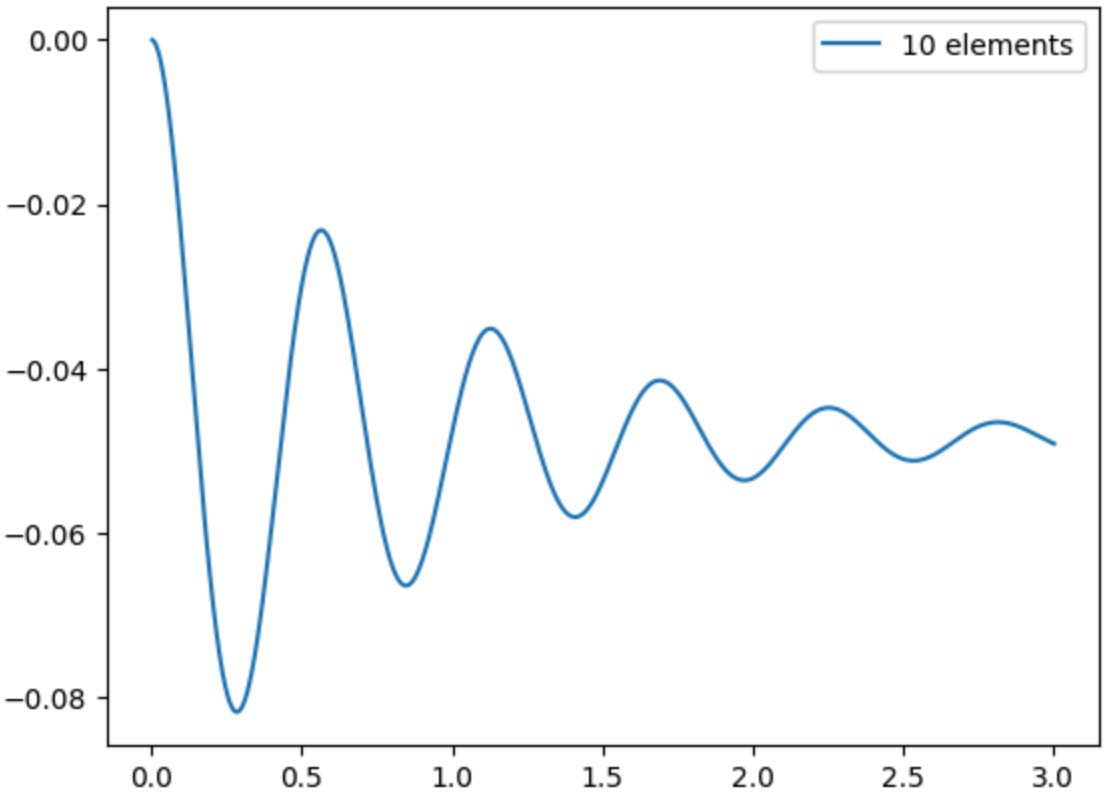
Out[6]: (np.float64(-0.5), np.float64(319.5), np.float64(239.5), np.float64(-0.5))



Shows the desired simulated model with all input data given.

In [7]:
```
plt.figure()
plt.plot(t,xyz_100_elements[:,-1,2],label='10 elements')
plt.legend()
```

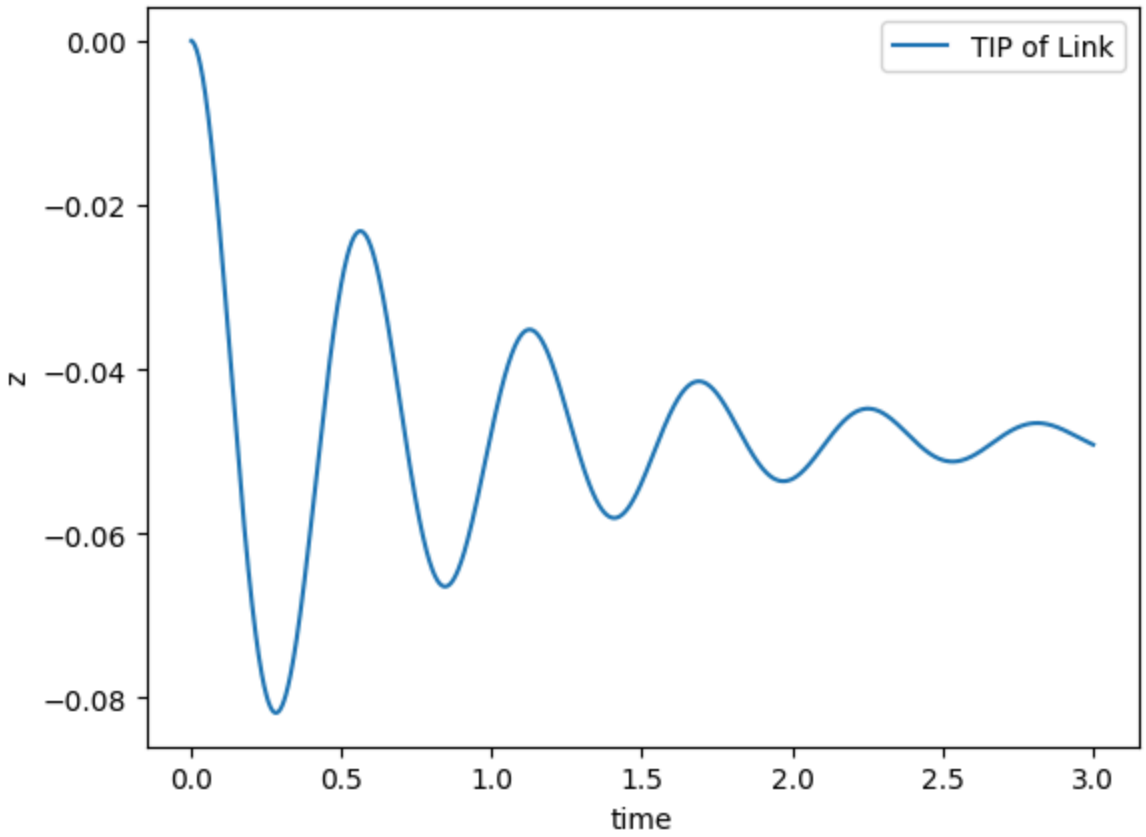
Out[7]: <matplotlib.legend.Legend at 0x1a2dd46ded0>




Plotting the Tip location over time

In [8]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [9]:
```
t_xyz = np.linspace(t.min(), t.max(), xyz_100_elements.shape[0])
z = xyz_100_elements[:, -1, 2]

plt.figure()
plt.plot(t_xyz, z, label="TIP of Link")
plt.legend()
plt.xlabel("time")
plt.ylabel("z")
plt.show()
```




In [10]:
```
import pandas as pd
import numpy
import matplotlib.pyplot as plt
import scipy.interpolate as si
df=pd.read_csv(r'C:\Users\shawn\OneDrive\Desktop\FOLDABLE ROBOTICS\beamdata.csv', s
print(df.columns)   # <-- see the real column names
print(df.head())    # <-- optional: see first few rows
```

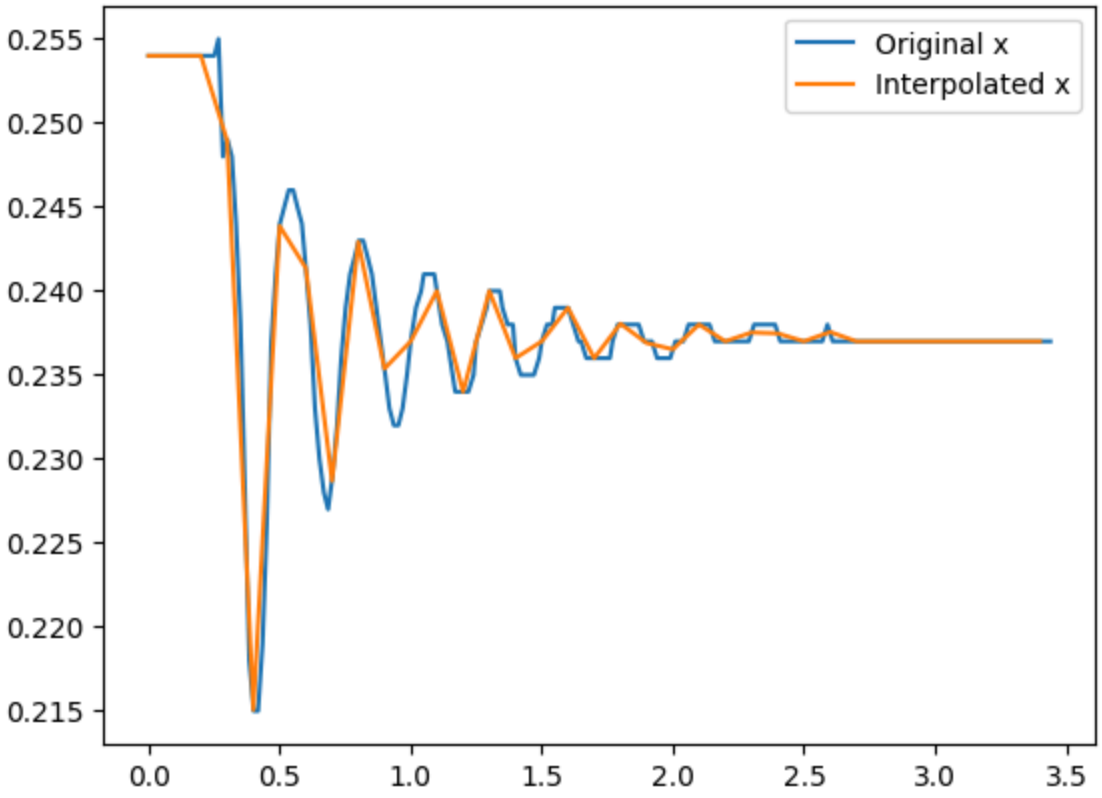
```
Index(['t', 'x', 'y'], dtype='object')
        t      x       y
0  0.0000  0.254  0.0500
1  0.0167  0.254  0.0504
2  0.0334  0.254  0.0506
3  0.0501  0.254  0.0504
4  0.0669  0.254  0.0505
```

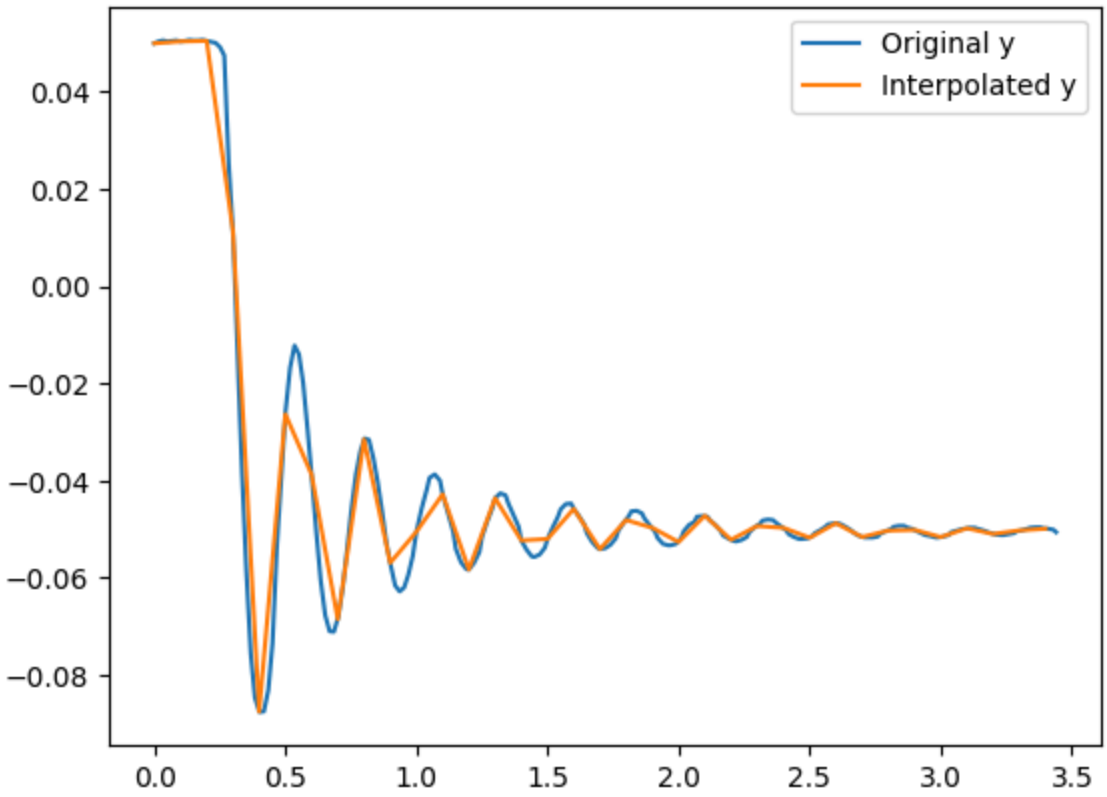
In [11]:
```
# TRACKER DATA input into jupyter lab
```

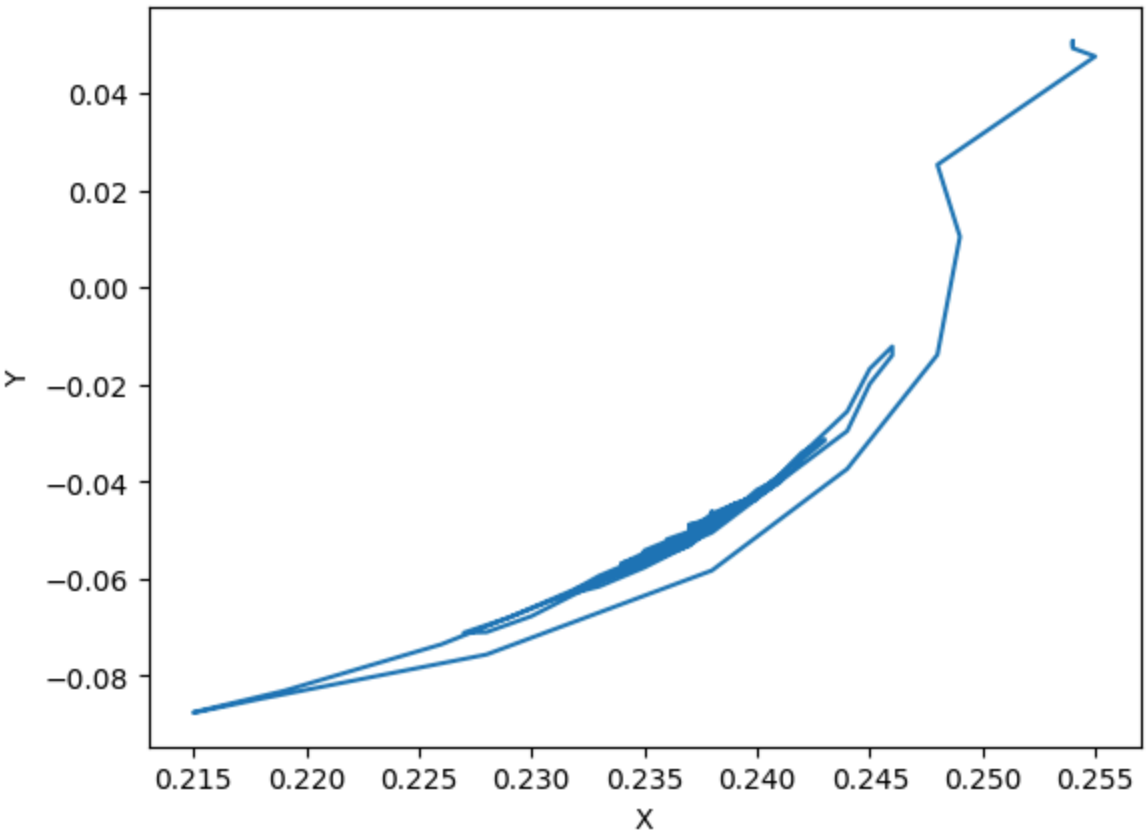
In [12]:
```
#x = df.x.astype('float64').to_numpy()
x = df.x.to_numpy()
y = df.y.to_numpy()
t = df.t.to_numpy()
xy = numpy.array([x,y]).T

f = si.interp1d(t,xy.T,fill_value='extrapolate',kind='quadratic')
new_t = numpy.r_[0:t[-1]:.1]
plt.figure()
plt.plot(t,x, label="Original x")
plt.plot(new_t,f(new_t)[0], label="Interpolated x")
plt.legend()
```

```
plt.figure()
plt.plot(t,y,label="Original y")
plt.plot(new_t,f(new_t)[1],label="Interpolated y")
plt.legend()

plt.figure()
plt.plot(x,y)
plt.xlabel("X")
plt.ylabel("Y")
```

Out[12]: Text(0, 0.5, 'Y')





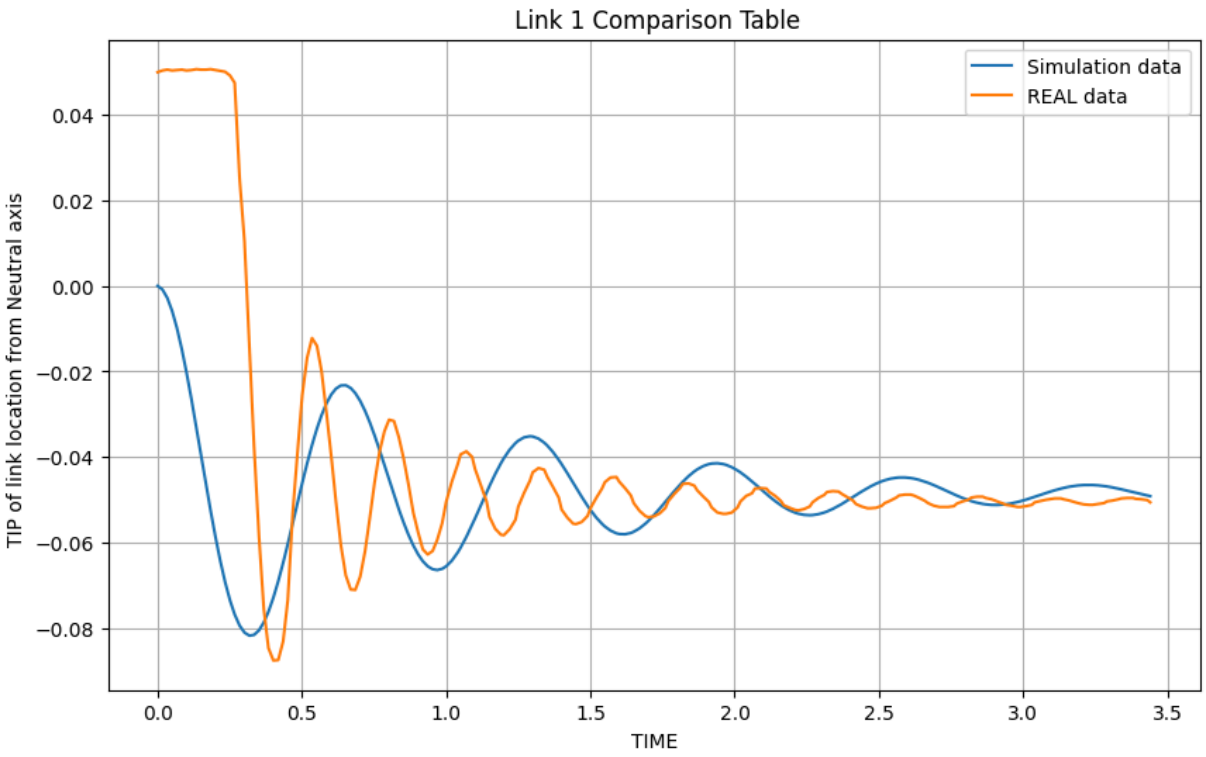




Graphs tracker experiment: 1) time vs x 2) time vs y 3) X vs Y

In [13]:
```
print("t shape:", t.shape)
print("y shape:", y.shape)
print("xyz shape:", xyz_100_elements.shape)
```

```
t shape: (207,)
y shape: (207,)
xyz shape: (1501, 4, 3)
```

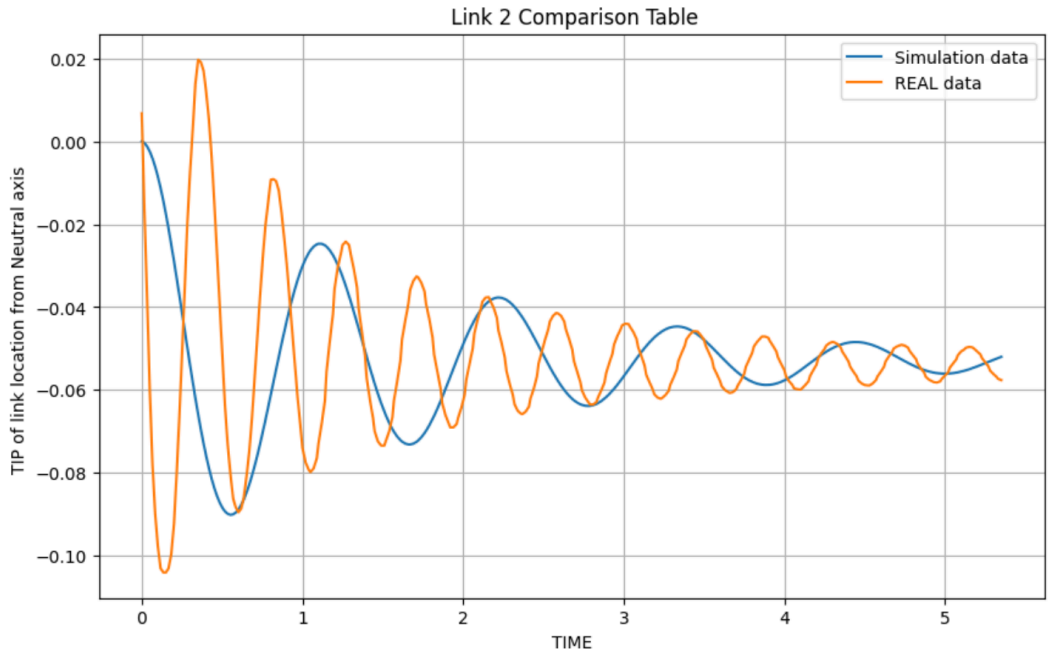
In [14]:
```python
import numpy as np

# xyz time axis (1501 points)
t_xyz = np.linspace(t.min(), t.max(), xyz_100_elements.shape[0])

z10 = xyz_100_elements[:, -1, 2]

# Interpolate z10 onto the 207-point t
z10_interp = np.interp(t, t_xyz, z10)

plt.figure(figsize=(10, 6))
plt.title(" Link 1 Comparison Table")
plt.plot(t, z10_interp, label="Simulation data")
plt.plot(t, y, label="REAL data")
plt.legend()
plt.grid(True)
plt.xlabel("TIME")
plt.ylabel("TIP of link location from Neutral axis ")
plt.show()
```



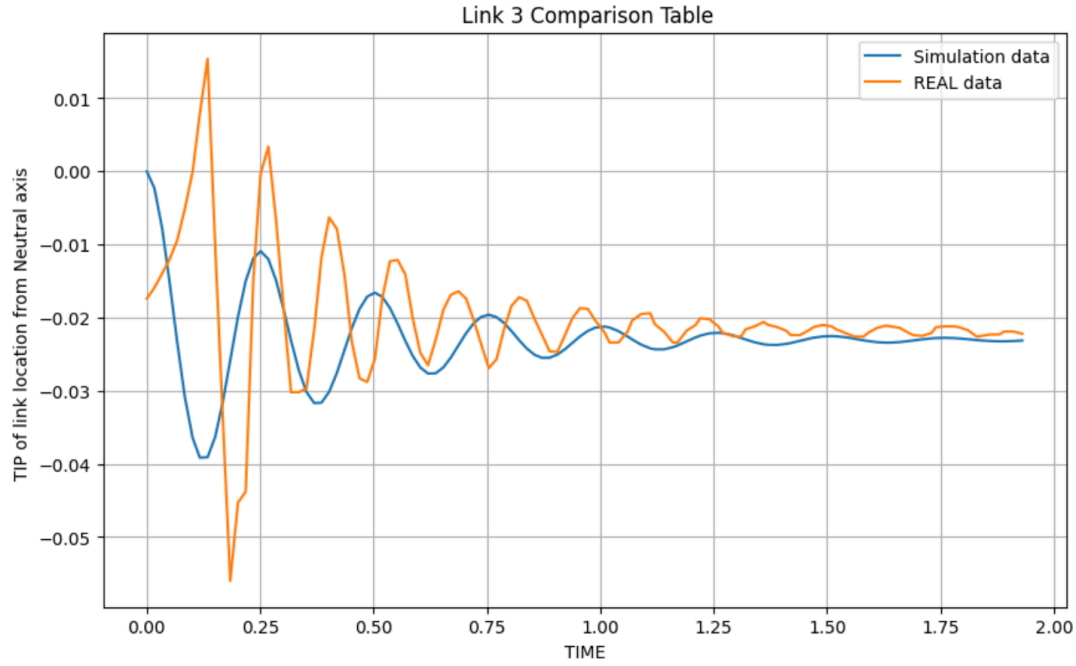

In [15]:
```python
# REAL DATA (TRACKER IN ORANGE) AND SIMULATION DATA (MUJOCO BLUE) Y VALUES
```

In [16]:
```python
# FINAL GRAPHS FOR THE 2ND AND 3RD EXPERIMENT
```

## LINK 2




## LINK 3




THE FOLLOWING CODE BELOW IS LINK EXPERIMENT 2 & 3 DONE THE SAME WAY BUT WITH DIFFERENT VALUES INPUTED FOR K,B,MASS,LINK,NUM BODIES, AND TRACKER DATA.