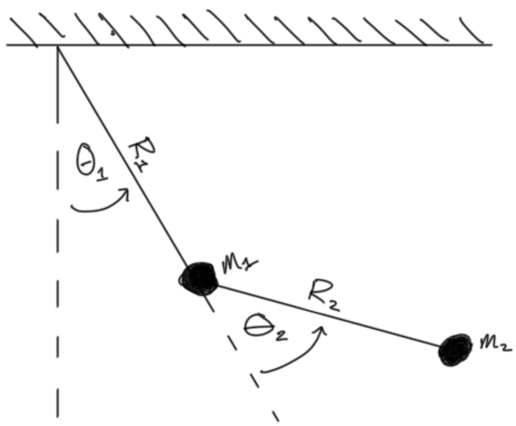
Arm Motion Modeling

System Description

A double-pendulum system hanging in gravity is shown in the figure above. $q = [\theta_1, \theta_2]$ are the system configuration variables. We assume the z-axis is pointing out from the screen/paper, thus the positive direction of rotation is counter-clockwise. The solution steps are:

1. Computing the Lagrangian of the system.
2. Computing the Euler-Lagrange equations, and solve them for $\ddot{\theta}_1$ and $\ddot{\theta}_2$.
3. Numerically evaluating the solutions for τ_1 and τ_2, and simulating the system for θ_1, θ_2, $\dot{\theta}_1$, $\dot{\theta}_2$, $\ddot{\theta}_1$ and $\ddot{\theta}_2$.
4. Animating the simulation.

In [10]:
```
from IPython.core.display import HTML
display(HTML("<table><tr><td><img src='https://github.com/MuchenSun/ME314pngs/raw/master/dyndoublepend.png' width=500' height='350'></table>"))
```



Define Helper Functions and Import Libraries

Import libraries:

In [2]:
```
# Imports required for the calculations
import sympy
from sympy.abc import t
from sympy import symbols, Eq, Function, solve, sin, cos, Matrix, Subs, substitution, Derivative, simplify, symbols, lambdify
import math
from math import pi
import numpy as np
import matplotlib.pyplot as plt

# Imports required for the animation
from plotly.offline import init_notebook_mode, iplot
from IPython.display import display, HTML
import plotly.graph_objects as go
```

Define the system's constants and the arm's motion data:

In [35]:
```
# lists of time steps, angles, angular velocity, and angular accelerations of the upper and lower arm (extracted from the data)

# TD_WN2 Right hand
# t_list = [0.0, 0.008333333333333333, 0.016666666666666667, 0.025, 0.03333333333333333, 0.041666666666666667, 0.05, 0.058333333333333333, 0.066666666666666667, 0.075, 0.083333333333333333,
# # t_list = np.linspace(0, 1/120, 3.175)
# theta1_list = [-0.192125844059536, -0.181950574520409, -0.171845118151362, -0.161774568367354, -0.151773831753427, -0.141877814894619, -0.132051611205891, -0.1223126739
# theta2_list = [0.525920063503451, 0.544612539792311, 0.56323520291109, 0.58173569298223, 0.60007910342069, 0.618300340811511, 0.636381951862172, 0.654289029987634, 0.67
# dtheta1_list = [-1.22103234469522, -1.21265476428566, -1.20846597408087, -1.2000883936713, -1.18752202305694, -1.17914444264737, -1.1686724671354, -1.16448367693062, -1
# dtheta2_list = [-2.24309715466311, -2.23471957425354, -2.2200588085368, -2.20120925261524, -2.1865484868985, -2.16979332607936, -2.1488493750554, -2.12581102892911, -2.
# ddtheta1_list = [-1.00530964914731, -0.502654824574682, -1.00530964914886, -1.50796447372277, -1.00530964914902, -1.25663706143592, -0.502654824574469, -0.2513274122871
# ddtheta2_list = [-1.00530964914931, -1.75929188600895, -2.26194671058664, -1.7592918860089, -2.01061929829724, -2.51327412287466, -2.76460153515501, -3.26725635973446,

# TD_WN2 Left hand
# t_list = [0.0, 0.008333333333333333, 0.016666666666666667, 0.025, 0.03333333333333333, 0.041666666666666667, 0.05, 0.058333333333333333, 0.066666666666666667, 0.075, 0.083333333333333333,
# theta1_list = [0.229563156514814, 0.233542507209361, 0.237556764488948, 0.241571021768535, 0.245567825755602, 0.249512269865109, 0.253351994219497, 0.257052092233725, 0
# theta2_list = [0.409681135320629, 0.404305521224486, 0.399069533468503, 0.39406043851528, 0.389243329779775, 0.38460075396947, 0.380080351206805, 0.37562976161422, 0.37
# dtheta1_list = [-0.477522083345646, -0.481710873550438, -0.481710873550428, -0.479616478448047, -0.473333293140861, -0.460766922526504, -0.444011761707353, -0.425162205
# dtheta2_list = [0.645073691537101, 0.62831853071796, 0.601091394386846, 0.578053048260521, 0.55710909723659, 0.542448331519836, 0.534070751110272, 0.536165146212659, 0.
# ddtheta1_list = [0.502654824575055, -1.21236354289067e-12, -0.251327412285702, -0.753982236862383, -1.50796447372281, -2.01061929829817, -2.26194671058435, -2.764601535
# ddtheta2_list = [2.01061929829702, 3.26725635973367, 2.76460153515898, 2.51327412287168, 1.75929188601055, 1.00530964914766, -0.251327412286528, -1.25663706143584, -1.0

# # TD_WN3 Right hand
# t_list = [0.0, 0.008333333333333333, 0.016666666666666667, 0.025, 0.03333333333333333, 0.041666666666666667, 0.05, 0.058333333333333333, 0.066666666666666667, 0.075, 0.083333333333333333,
# theta1_list = [-0.152768669427064, -0.142628306472977, -0.132592663274009, -0.122714099707721, -0.113027522359153, -0.103567837813344, -0.094282686192734, -0.0851720674
# theta2_list = [0.551576403507768, 0.57257271440926, 0.593795918113511, 0.615368187668161, 0.637306976365729, 0.659594830913697, 0.682109578264424, 0.70474649866279, 0.7
# dtheta1_list = [-1.21684355449045, -1.20427718387609, -1.18542762795455, -1.16238928182822, -1.13516214549711, -1.11421819447318, -1.09327424344925, -1.08699105814207,
# dtheta2_list = [-2.519557308179, -2.54678444451012, -2.588672346558, -2.63265464370825, -2.67454254575612, -2.70176968208721, -2.71643044780397, -2.71852484290637, -2.7
# ddtheta1_list = [-1.50796447372317, -2.26194671058419, -2.76460153515933, -3.26725635973348, -2.5132741228716, -2.51327412287194, -0.753982236861397, 0.0, 0.50265482457
# ddtheta2_list = [3.26725635973446, 5.0265482457449, 5.27787565802999, 5.02654824574496, 3.267256359731, 1.75929188601076, 0.251327412288127, -0.251327412288127, -2.5132

# TD_WN3 Left hand
t_list = [0.0, 0.008333333333333333, 0.016666666666666667, 0.025, 0.03333333333333333, 0.041666666666666667, 0.05, 0.058333333333333333, 0.066666666666666667, 0.075, 0.083333333333333333, 0.
theta1_list = [-0.183818076820043, -0.193260308073332, -0.202737445911661, -0.21226694362755, -0.221866254513519, -0.231517925277048, -0.241169596040576, -0.2507340003415
theta2_list = [0.394776023508597, 0.39097120573925, 0.387358374187621, 0.384112061778912, 0.381249721805641, 0.378806260852849, 0.376676959165416, 0.374774550280742, 0.37
dtheta1_list = [1.13306775039472, 1.1372565405995, 1.14353972590669, 1.15191730631626, 1.15820049162344, 1.15820049162344, 1.14772851611147, 1.12678456508755, 1.093274243
dtheta2_list = [0.456578132321719, 0.433539786195394, 0.389557489045138, 0.343480796792475, 0.293215314335049, 0.25551620249197, 0.228289066160863, 0.217817090648884, 0.2
ddtheta1_list = [-0.502654824574256, -0.753982236862356, -1.00530964914816, -0.753982236861983, 5.06261699229072e-13, 1.25663706143613, 2.51327412287048, 4.02123859659565
ddtheta2_list = [2.76460153515901, 5.27787565803069, 5.5292030703196, 6.03185789489107, 4.52389342116953, 3.26725635973283, 1.25663706143744, -0.753982236862783, -2.51327

# Masses, length and center-of-mass positions (calculated using the lab measurements)
```

```
m_upper_arm = 2 # TODO
m_lower_arm = 0.7395

L_upper_arm = 0.30 # TODO
L_lower_arm = 0.42

L_upper_arm_COM = 0.15 # TODO
L_lower_arm_COM = 0.2388
```

Computing the Lagrangian of the system:

In [4]:
```
m1, m2, g, R1, R1_COM, R2, R2_COM = symbols(r'm1, m2, g, R1, R1_COM, R2, R2_COM')

# The system torque variables as function of t
tau1 = Function(r'tau1')(t)
tau2 = Function(r'tau2')(t)

# The system configuration variables as function of t
theta1 = Function(r'theta1')(t)
theta2 = Function(r'theta2')(t)

# The velocity as derivative of position wrt t
theta1_dot = theta1.diff(t)
theta2_dot = theta2.diff(t)

# The acceleration as derivative of velocity wrt t
theta1_ddot = theta1_dot.diff(t)
theta2_ddot = theta2_dot.diff(t)

# Converting the polar coordinates to cartesian coordinates
x1 = R1_COM*sin(theta1)
x2 = R1*sin(theta1) + R2_COM*sin(theta1 + theta2)

y1 = -R1_COM*cos(theta1)
y2 = -R1*cos(theta1) - R2_COM*cos(theta1 + theta2)

# Calculating the kinetic and potential energy of the system
KE = 1/2*m1*((x1.diff(t))**2 + (y1.diff(t))**2) + 1/2*m2*((x2.diff(t))**2 + (y2.diff(t))**2)
PE = m1*g*y1 + m2*g*y2

# Computing the Lagrangian
L = simplify(KE - PE)
print('L: ')
display(L)
```

L:

$$0.5R_{1COM}^2 m_1 \left(\frac{d}{dt}\theta_1(t)\right)^2 + R_{1COM} g m_1 \cos(\theta_1(t)) + g m_2 (R_1 \cos(\theta_1(t)) + R_{2COM} \cos(\theta_1(t) + \theta_2(t)))$$
$$+ 0.5 m_2 \left(R_1^2 \left(\frac{d}{dt}\theta_1(t)\right)^2 + 2R_1 R_{2COM} \cos(\theta_2(t)) \left(\frac{d}{dt}\theta_1(t)\right)^2 + 2R_1 R_{2COM} \cos(\theta_2(t)) \frac{d}{dt}\theta_1(t) \frac{d}{dt}\theta_2(t) + R_{2COM}^2 \left(\frac{d}{dt}\theta_1(t)\right)^2 + 2R_{2COM}^2 \frac{d}{dt}\theta_1(t) \frac{d}{dt}\theta_2(t) + R_{2COM}^2 \left(\frac{d}{dt}\theta_2(t)\right)^2\right)$$

Computing the Euler-Lagrange equations:

In [5]:
```
# Define the derivative of L wrt the functions: x, xdot
L_dtheta1 = L.diff(theta1)
L_dtheta2 = L.diff(theta2)

L_dtheta1_dot = L.diff(theta1_dot)
L_dtheta2_dot = L.diff(theta2_dot)

# Define the derivative of L_dxdot wrt to time t
L_dtheta1_dot_dt = L_dtheta1_dot.diff(t)
L_dtheta2_dot_dt = L_dtheta2_dot.diff(t)

# Define the left hand side of the the Euler-Lagrange as a matrix
lhs = Matrix([simplify(L_dtheta1_dot_dt - L_dtheta1), simplify(L_dtheta2_dot_dt - L_dtheta2)])

# Define the right hand side of the the Euler-Lagrange as a Matrix
rhs = Matrix([tau1, tau2])

# Compute the Euler-Lagrange equations as a matrix
EL_eqns = Eq(lhs, rhs)

print('Euler-Lagrange matrix for this systems:')
display(EL_eqns)
```

Euler-Lagrange matrix for this systems:

$$\left[\begin{matrix} 1.0R_{1COM}^2 m_1 \frac{d^2}{dt^2}\theta_1(t) + R_{1COM} g m_1 \sin(\theta_1(t)) + g m_2 (R_1 \sin(\theta_1(t)) + R_{2COM} \sin(\theta_1(t) + \theta_2(t))) \\ + m_2 \left(R_1^2 \frac{d^2}{dt^2}\theta_1(t) - 2R_1 R_{2COM} \sin(\theta_2(t)) \frac{d}{dt}\theta_1(t) \frac{d}{dt}\theta_2(t) - R_1 R_{2COM} \sin(\theta_2(t)) \left(\frac{d}{dt}\theta_2(t)\right)^2 + 2R_1 R_{2COM} \cos(\theta_2(t)) \frac{d^2}{dt^2}\theta_1(t) + R_1 R_{2COM} \cos(\theta_2(t)) \frac{d^2}{dt^2}\theta_2(t) + R_{2COM}^2 \frac{d^2}{dt^2}\theta_1(t) + R_{2COM}^2 \right. \\ R_{2COM} m_2 \left(R_1 \sin(\theta_2(t)) \left(\frac{d}{dt}\theta_1(t)\right)^2 + R_1 \cos(\theta_2(t)) \frac{d^2}{dt^2}\theta_1(t) + R_{2COM} \frac{d^2}{dt^2}\theta_1(t) + R_{2COM} \frac{d^2}{dt^2}\theta_2(t) + g \sin(\theta_1(t) + \theta_2(t))\right) \end{matrix}\right.$$

Solve the equations for τ_1 and τ_2:

In [6]:
```
# Solve the Euler-Lagrange equations for the shoulder and elbow torques
T = Matrix([tau1, tau2])
soln = solve(EL_eqns, T, dict=True)

# Initialize the solutions
solution = [0, 0]
i = 0

for sol in soln:
    for v in T:
        solution[i] = simplify(sol[v])
        display(Eq(T[i], solution[i]))
        i =+ 1
```

$$\tau_1(t) = R_1^2 m_2 \frac{d^2}{dt^2}\theta_1(t) - 2.0R_1 R_{2COM} m_2 \sin(\theta_2(t)) \frac{d}{dt}\theta_1(t) \frac{d}{dt}\theta_2(t) - R_1 R_{2COM} m_2 \sin(\theta_2(t)) \left(\frac{d}{dt}\theta_2(t)\right)^2 + 2.0R_1 R_{2COM} m_2 \cos(\theta_2(t)) \frac{d^2}{dt^2}\theta_1(t) + R_1 R_{2COM} m_2 \cos(\theta_2(t)) \frac{d^2}{dt^2}\theta_2(t)$$
$$+ R_1 g m_2 \sin(\theta_1(t)) + R_{1COM}^2 m_1 \frac{d^2}{dt^2}\theta_1(t) + R_{1COM} g m_1 \sin(\theta_1(t)) + R_{2COM}^2 m_2 \frac{d^2}{dt^2}\theta_1(t) + R_{2COM}^2 m_2 \frac{d^2}{dt^2}\theta_2(t) + R_{2COM} g m_2 \sin(\theta_1(t) + \theta_2(t))$$

$$\tau_2(t) = R_{2COM} m_2 \left(R_1 \sin(\theta_2(t)) \left(\frac{d}{dt}\theta_1(t) \right)^2 + R_1 \cos(\theta_2(t)) \frac{d^2}{dt^2}\theta_1(t) + R_{2COM} \frac{d^2}{dt^2}\theta_1(t) + R_{2COM} \frac{d^2}{dt^2}\theta_2(t) + g \sin(\theta_1(t) + \theta_2(t)) \right)$$

Simulating the system:

In [53]:

```
# Substitute the derivative variables with a dummy variables and plug-in the constants
solution_0_subs = solution[0]
solution_1_subs = solution[1]

theta1_dot_dummy = symbols('dtheta1')
theta2_dot_dummy = symbols('dtheta2')
theta1_ddot_dummy = symbols('ddtheta1')
theta2_ddot_dummy = symbols('ddtheta2')

solution_0_subs = solution_0_subs.subs([(m1, m_upper_arm), (m2, m_lower_arm), (R1, L_upper_arm), (R2, L_lower_arm), (R1_COM, L_upper_arm_COM), (R2_COM, L_lower_arm_COM),
solution_1_subs = solution_1_subs.subs([(m1, m_upper_arm), (m2, m_lower_arm), (R1, L_upper_arm), (R2, L_lower_arm), (R1_COM, L_upper_arm_COM), (R2_COM, L_lower_arm_COM),

display(Eq(T[0], solution_0_subs))
display(Eq(T[1], solution_1_subs))

solution_0_subs = solution_0_subs.subs([((theta1.diff(t)).diff(t), theta1_ddot_dummy), ((theta2.diff(t)).diff(t), theta2_ddot_dummy)])
solution_1_subs = solution_1_subs.subs([((theta1.diff(t)).diff(t), theta1_ddot_dummy), ((theta2.diff(t)).diff(t), theta2_ddot_dummy)])

solution_0_subs = solution_0_subs.subs([(theta1.diff(t), theta1_dot_dummy), (theta2.diff(t), theta2_dot_dummy)])
solution_1_subs = solution_1_subs.subs([(theta1.diff(t), theta1_dot_dummy), (theta2.diff(t), theta2_dot_dummy)])

# Lambdify the thetas and its derivatives
func1 = lambdify([theta1, theta2, theta1_dot_dummy, theta2_dot_dummy, theta1_ddot_dummy, theta2_ddot_dummy], solution_0_subs, modules = sympy)
func2 = lambdify([theta1, theta2, theta1_dot_dummy, theta2_dot_dummy, theta1_ddot_dummy, theta2_ddot_dummy], solution_1_subs, modules = sympy)

# Initialize the torque and power lists
tau1_list, tau2_list = [], []
power1_list, power2_list = [], []

# Plug-in the angles, angular velocities and angular accelerations for every time step to find the torques
for i in range(len(t_list)):
    tau1_list.append(func1(theta1_list[i], theta2_list[i], dtheta1_list[i], dtheta2_list[i], ddtheta1_list[i], ddtheta2_list[i]))
    tau2_list.append(func2(theta1_list[i], theta2_list[i], dtheta1_list[i], dtheta2_list[i], ddtheta1_list[i], ddtheta2_list[i]))

    # Calculate the power required to reach the required angular velocities and joints torques for every time step
    power1_list.append(dtheta1_list[i] * tau1_list[i])
    power2_list.append(dtheta2_list[i] * tau2_list[i])

# print(tau1_list)
# print(tau2_list)
# print(power1_list)
# print(power2_list)

print("####################################################################################################\
       \nCalculation summary:\n")
print(f"Shoulder max angular velocity:\t{format(max(dtheta1_list), '.5f')}\t\t Shoulder average angular velocity:\t{format(sum(dtheta1_list) / float(len(dtheta1_list)), '
print(f"Elbow max angular velocity:\t{format(max(dtheta2_list), '.5f')}\t\t Elbow average angular velocity:\t{format(sum(dtheta2_list) / float(len(dtheta2_list)), '.5f')}
print(f"Shoulder max torque:\t\t{format(max(tau1_list), '.5f')}\t\t Shoulder average torque:\t\t{format(sum(tau1_list) / float(len(tau1_list)), '.5f')}")
print(f"Elbow max torque:\t\t{format(max(tau2_list), '.5f')}\t\t Elbow average torque:\t\t\t{format(sum(tau2_list) / float(len(tau2_list)), '.5f')}\n")
print(f"Shoulder max power:\t\t{format(max(power1_list), '.5f')}\t\t Shoulder average power:\t\t{format(sum(power1_list) / float(len(power1_list)), '.5f')}")
print(f"Elbow max power:\t\t{format(max(power2_list), '.5f')}\t\t Elbow average power:\t\t\t{format(sum(power2_list) / float(len(power2_list)), '.5f')}\n")
```

$$\tau_1(t) = 1.732373406 \sin(\theta_1(t) + \theta_2(t)) + 5.1193485 \sin(\theta_1(t)) - 0.10595556 \sin(\theta_2(t)) \frac{d}{dt}\theta_1(t) \frac{d}{dt}\theta_2(t) - 0.05297778 \sin(\theta_2(t)) \left(\frac{d}{dt}\theta_2(t) \right)^2 + 0.10595556 \cos(\theta_2(t)) \frac{d^2}{dt^2}\theta_1(t) + 0.05297778$$

$$(\theta_2(t)) \frac{d^2}{dt^2}\theta_2(t) + 0.15372531288 \frac{d^2}{dt^2}\theta_1(t) + 0.04217031288 \frac{d^2}{dt^2}\theta_2(t)$$

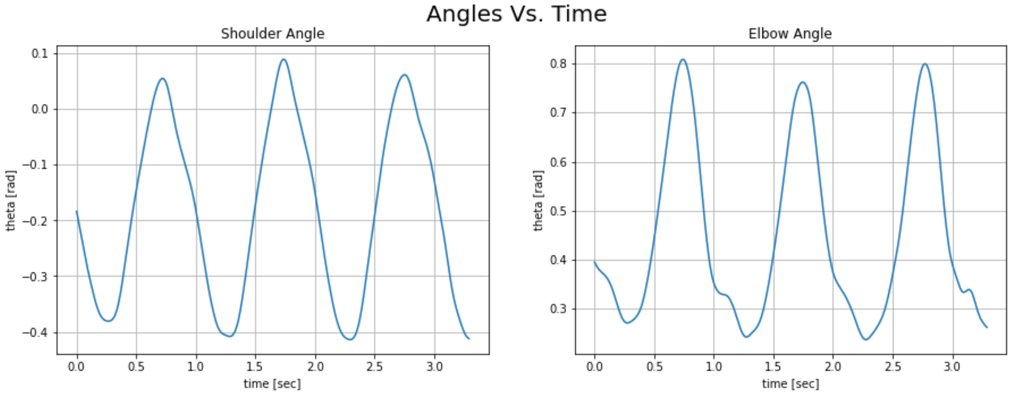
$$\tau_2(t) = 1.732373406 \sin(\theta_1(t) + \theta_2(t)) + 0.05297778 \sin(\theta_2(t)) \left(\frac{d}{dt}\theta_1(t) \right)^2 + 0.05297778 \cos(\theta_2(t)) \frac{d^2}{dt^2}\theta_1(t) + 0.04217031288 \frac{d^2}{dt^2}\theta_1(t) + 0.04217031288 \frac{d^2}{dt^2}\theta_2(t)$$

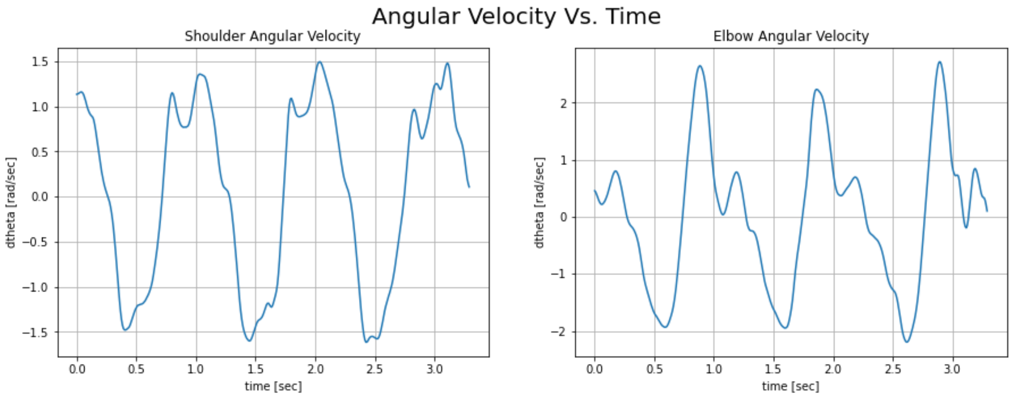
```
####################################################################################################
Calculation summary:

Shoulder max angular velocity:  1.49330         Shoulder average angular velocity:     0.06951
Elbow max angular velocity:     2.71852         Elbow average angular velocity:        0.04034

Shoulder max torque:            2.90764         Shoulder average torque:               -0.42268
Elbow max torque:               1.68997         Elbow average torque:                  0.50232

Shoulder max power:             3.32653         Shoulder average power:                -0.05072
Elbow max power:                2.95808         Elbow average power:                   0.18007
```

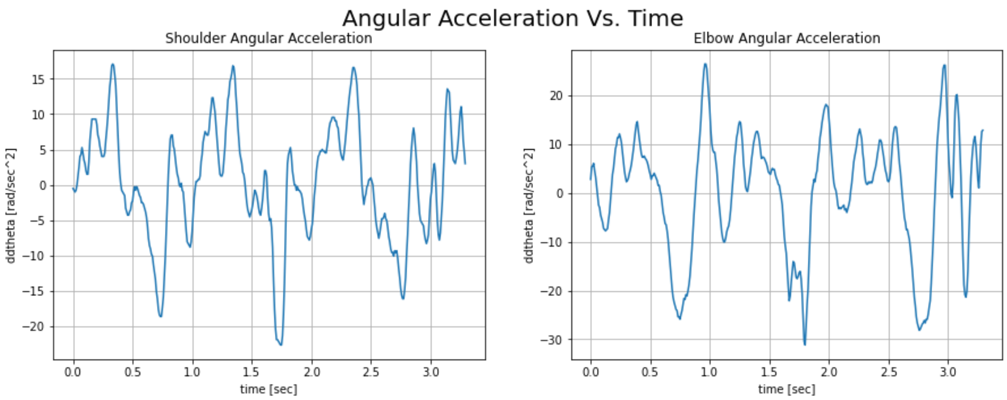
In [42]:

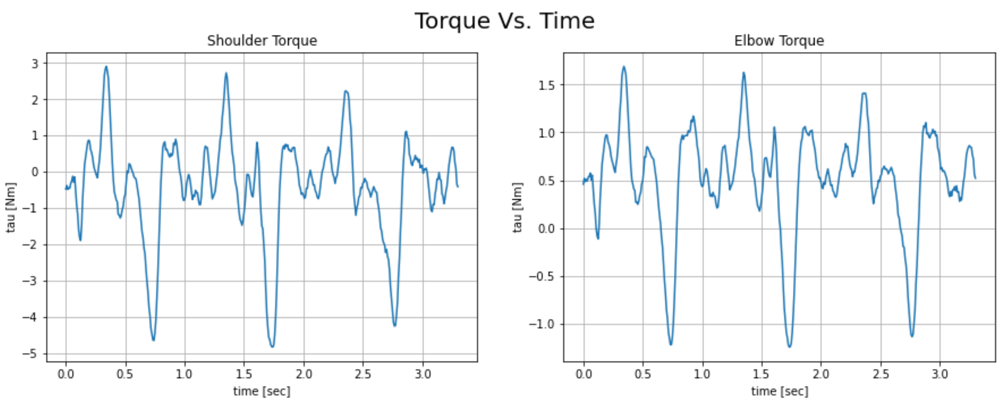
```
# Compute the trajectory of the arm's motion
N = int((max(t_list) - min(t_list))/(1/120))
tvec = np.linspace(min(t_list), max(t_list), N)
traj = np.zeros((6, N))
for i in range(N):
    traj[0, i] = theta1_list[i]
    traj[1, i] = theta2_list[i]
    traj[2, i] = dtheta1_list[i]
    traj[3, i] = dtheta2_list[i]
    traj[4, i] = ddtheta1_list[i]
    traj[5, i] = ddtheta2_list[i]

# Calculate the length difference between the time list and the trajectory lists
diff = (len(t_list) - len(traj[0]))

# Plot the trajectory lists (angles, velocities, accelerations, torques, and power)
plt.figure(figsize=(15,5))
plt.suptitle('Angles Vs. Time', fontsize=20)
plt.subplot(121)
plt.plot(t_list[:-diff], traj[0])
plt.ylabel('theta [rad]')
plt.xlabel('time [sec]')
plt.grid()
plt.title('Shoulder Angle')

plt.subplot(122)
plt.plot(t_list[:-diff], traj[1])
plt.ylabel('theta [rad]')
plt.xlabel('time [sec]')
plt.grid()
```

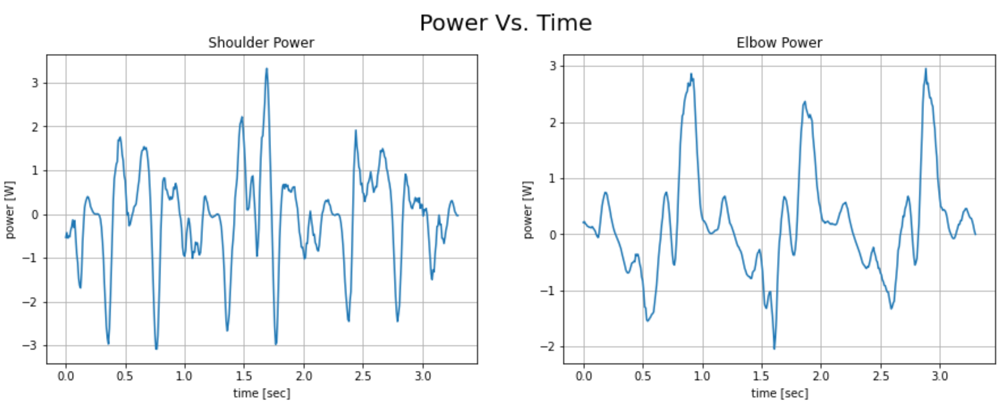
```
plt.title('Elbow Angle')
plt.show()

plt.figure(figsize=(15,5))
plt.suptitle('Angular Velocity Vs. Time', fontsize=20)
plt.subplot(121)
plt.plot(t_list[:-diff], traj[2])
plt.ylabel('dtheta [rad/sec]')
plt.xlabel('time [sec]')
plt.grid()
plt.title('Shoulder Angular Velocity')

plt.subplot(122)
plt.plot(t_list[:-diff], traj[3])
plt.ylabel('dtheta [rad/sec]')
plt.xlabel('time [sec]')
plt.grid()
plt.title('Elbow Angular Velocity')
plt.show()

plt.figure(figsize=(15,5))
plt.suptitle('Angular Acceleration Vs. Time', fontsize=20)
plt.subplot(121)
plt.plot(t_list[:-diff], traj[4])
plt.ylabel('ddtheta [rad/sec^2]')
plt.xlabel('time [sec]')
plt.grid()
plt.title('Shoulder Angular Acceleration')

plt.subplot(122)
plt.plot(t_list[:-diff], traj[5])
plt.ylabel('ddtheta [rad/sec^2]')
plt.xlabel('time [sec]')
plt.grid()
plt.title('Elbow Angular Acceleration')
plt.show()

plt.figure(figsize=(15,5))
plt.suptitle('Torque Vs. Time', fontsize=20)
plt.subplot(121)
plt.plot(t_list, tau1_list)
plt.ylabel('tau [Nm]')
plt.xlabel('time [sec]')
plt.grid()
plt.title('Shoulder Torque')

plt.subplot(122)
plt.plot(t_list, tau2_list)
plt.ylabel('tau [Nm]')
plt.xlabel('time [sec]')
plt.grid()
plt.title('Elbow Torque')
plt.show()

plt.figure(figsize=(15,5))
plt.suptitle('Power Vs. Time', fontsize=20)
plt.subplot(121)
plt.plot(t_list, power1_list)
plt.ylabel('power [W]')
plt.xlabel('time [sec]')
plt.grid()
plt.title('Shoulder Power')

plt.subplot(122)
plt.plot(t_list, power2_list)
plt.ylabel('power [W]')
plt.xlabel('time [sec]')
plt.grid()
plt.title('Elbow Power')
plt.show()
```







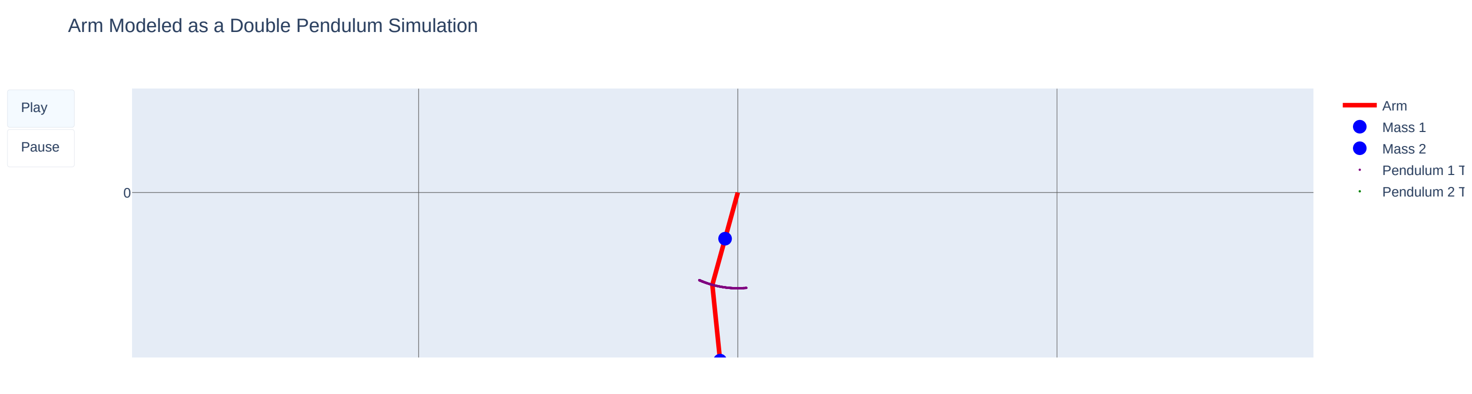


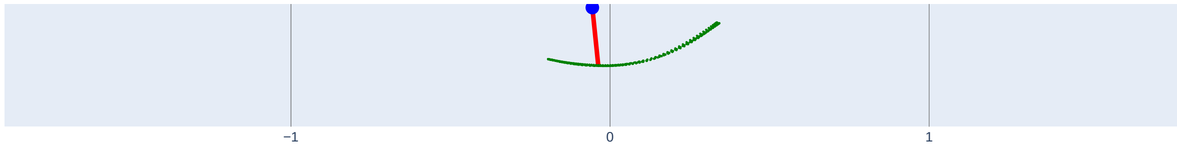


Animating the simulation:

In [54]:
```
def animate_double_pend(traj, L1, L2, L1_COM, L2_COM, T=5):
    """
    Function to generate web-based animation of double-pendulum system

    Parameters:
        traj:         trajectory of theta1 and theta2
        L1:           length of the upper arm
        L2:           length of the lower arm
        L1_COM:       length of the center of mass of the upper arm from the shoulder
        L2_COM:       length of the center of mass of the lower arm from the elbow
        T:            length/seconds of animation duration

    Returns: None
    """

    # Browser configuration
    def configure_plotly_browser_state():
        import IPython
        display(IPython.core.display.HTML('''
            <script src="/static/components/requirejs/require.js"></script>
```

```
            <script>
              requirejs.config({
                paths: {
                  base: '/static/base',
                  plotly: 'https://cdn.plot.ly/plotly-1.5.1.min.js?noext',
                },
              });
            </script>
            '''))
    configure_plotly_browser_state()
    init_notebook_mode(connected=False)

    # Getting data from pendulum angle trajectories
    xx1 = L1 * np.sin(traj[0])
    yy1 = -L1 * np.cos(traj[0])
    xx1_COM = L1_COM * np.sin(traj[0])
    yy1_COM = -L1_COM * np.cos(traj[0])
    xx2 = xx1 + L2 * np.sin(traj[0] + traj[1])
    yy2 = yy1 - L2 * np.cos(traj[0] + traj[1])
    xx2_COM = xx1 + L2_COM * np.sin(traj[0] + traj[1])
    yy2_COM = yy1 - L2_COM * np.cos(traj[0] + traj[1])
    N = len(traj[0])

    # Using these to specify axis limits
    xm = np.min(xx1)
    xM = np.max(xx1)
    ym = np.min(yy1) - 0.6
    yM = np.max(yy1) + 0.6

    # Defining data dictionary
    data = [dict(x=xx1, y=yy1,
                 mode='lines', name='Arm',
                 line=dict(width=2, color='blue')
                ),
            dict(x=xx1_COM, y=yy1_COM,
                 mode='lines', name='Mass 1',
                 line=dict(width=2, color='purple')
                ),
            dict(x=xx2_COM, y=yy2_COM,
                 mode='lines', name='Mass 2',
                 line=dict(width=2, color='green')
                ),
            dict(x=xx1, y=yy1,
                 mode='markers', name='Pendulum 1 Traj',
                 marker=dict(color="purple", size=2)
                ),
            dict(x=xx2, y=yy2,
                 mode='markers', name='Pendulum 2 Traj',
                 marker=dict(color="green", size=2)
                )
          ]

    # Preparing simulation layout
    layout = dict(xaxis=dict(range=[xm, xM], autorange=False, zeroline=False,dtick=1),
                  yaxis=dict(range=[ym, yM], autorange=False, zeroline=False,scaleanchor = "x",dtick=1),
                  title='Arm Modeled as a Double Pendulum Simulation',
                  hovermode='closest',
                  updatemenus= [{'type': 'buttons',
                                 'buttons': [{'label': 'Play','method': 'animate',
                                              'args': [None, {'frame': {'duration': T, 'redraw': False}}]},
                                             {'args': [[None], {'frame': {'duration': T, 'redraw': False}, 'mode': 'immediate',
                                              'transition': {'duration': 0}}],'label': 'Pause','method': 'animate'}
                                            ]
                                }]
                 )

    # Defining the frames of the simulation
    frames = [dict(data=[dict(x=[0,xx1[k],xx2[k]],
                              y=[0,yy1[k],yy2[k]],
                              mode='lines',
                              line=dict(color='red', width=4)),
                         go.Scatter(
                              x=[xx1_COM[k]],
                              y=[yy1_COM[k]],
                              mode="markers",
                              marker=dict(color="blue", size=12)),
                         go.Scatter(
                              x=[xx2_COM[k]],
                              y=[yy2_COM[k]],
                              mode="markers",
                              marker=dict(color="blue", size=12)),
                        ]) for k in range(N)]

    # Putting it all together and plotting
    figure = dict(data=data, layout=layout, frames=frames)
    iplot(figure)

# Animate the system
animate_double_pend(traj, L1=L_upper_arm, L2=L_lower_arm, L1_COM=L_upper_arm_COM, L2_COM=L_lower_arm_COM, T=5)
```





In [ ]: