

arm_single_pendulum_modeling_arm_back

October 22, 2021

1 Arm Motion Modeling

1.1 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.

```
[42]: from IPython.core.display import HTML
display(HTML("<table><tr><td><img src='./double-pendulum-diagram.png' ↵
↵width=450' height='300'></table>"))
```

<IPython.core.display.HTML object>

1.2 Import Libraries and Define System Constants

Import libraries:

```
[37]: # Imports required for data processing
import os
import csv
import pandas as pd

# Imports required for dynamics 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 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:

```
[38]: # Masses, length and center-of-mass positions (calculated using the lab
      ↪ measurements)
      # Mass calculations (mass unit is kg)
      m_body = 53
      m_u = 0.028 * m_body                # Average upper arm weights relative
      ↪ to body weight, from "Biomechanics                # and Motor Control of Human Movement"
      ↪ by David Winter (2009), 4th edition
      m_l = 0.7395                        # Average lower prosthetics weights,
      ↪ calculated using lab measurements
      # m_lower = 0.022 * m_body          # Average lower arm weights relative
      ↪ to body weight, from "Biomechanics                # and Motor Control of Human Movement"
      ↪ by David Winter (2009), 4th edition
      # Arm length calculations (length unit is m)
      H_body = 1.62
      L_u = 0.186 * H_body                # Average upper arm length relative to
      ↪ body height
      ↪ of Human Movement" by David
      ↪ Winter (2009), 4th edition
      # L_l = (0.146 + 0.108) * H_body    # Average lower arm length relative to
      ↪ body height
      ↪ of Human Movement" by David
      ↪ Winter (2009), 4th edition
      L_l = 0.42                          # Average lower prosthetics length,
      ↪ calculated using lab measurements
      # Arm center of mass length calculations (length unit is m)
      L_u_c = 0.436 * L_u                # Average upper arm length from
      ↪ shoulder to center of mass relative
      ↪ to upper arm length, from
      ↪ "Biomechanics and Motor Control of Human
      ↪ Movement" by David Winter (2009),
      ↪ 4th edition
      L_l_c = 0.2388                     # Average lower prosthetics length
      ↪ from elbow to center of mass,
      ↪ calculated using lab measurements
      # L_l_c = 0.682 * L_l              # Average lower arm length from
      ↪ shoulder to center of mass relative
```

```

# to upper arm length, from
→ "Biomechanics and Motor Control of Human
# Movement" by David Winter (2009),
→ 4th edition

```

1.3 Extracting Data

Extracting angles data and computing angular velocities and angular accelerations from the angles:

```

[3]: def calculate_Vel(Ang_list, time_list, index):
    return ((Ang_list[index + 1] - Ang_list[index])
            / (time_list[index + 1] - time_list[index]))

def calculate_Acc(Vel_list, time_list, index):
    return ((Vel_list[index + 1] - Vel_list[index])
            / (time_list[index + 1] - time_list[index]))

data_csv_dir = '../data/hand_back_motion_data/CSV Converted Files'
frame_frequency = 100
print("current directory: ", os.getcwd())

walking_vel_list = []
time_list = []
Elbow_Ang_list, Sholder_Ang_list = [], []
Elbow_Vel_list, Sholder_Vel_list = [], []
Elbow_Acc_list, Sholder_Acc_list = [], []
Elbow_Acc_data_list, Sholder_Acc_data_list = [], []
Back_Ang_list, Back_Pos_list, Back_Vel_list = [], [], []

folder_list = os.listdir(data_csv_dir)
folder_list.sort()

for folder in folder_list:

    data_trial_dir = os.path.join(data_csv_dir, folder)
    if os.path.isdir(data_trial_dir):
        file_list = os.listdir(data_trial_dir)

        for file in file_list:
            if "00B429F8" in file:
                if file.endswith(".csv"):
                    file_name = file[:-4]
                    walking_vel = file.split("_")[4][:5]

                    frame = 0

```

```

        file_time_list = []
        file_Shoulder_Ang_list, file_Shoulder_Vel_list,
↪file_Shoulder_Acc_list, file_Shoulder_Acc_data_list = [], [], [], []

        # Cutting out weird data behavior on data edges
        data_path = os.path.join(data_csv_dir, folder, file)
        data_rows = open(data_path).read().strip().split("\n")[7500:
↪9500]

        # Extract time [sec], elbow angles [rad], and shoulder
↪angles [rad] from data
        for row in data_rows:
            splitted_row = row.strip().split("\t")

            # Check if loop finished all data
            if not len(splitted_row):
                break

            file_time_list.append(frame / frame_frequency)
            file_Shoulder_Ang_list.append(float(splitted_row[31]) *
↪2*pi/360)

            file_Shoulder_Acc_data_list.
↪append(float(splitted_row[14]))
            frame += 1

        # Extract elbow and shoulder velocities [rad/sec] from
↪angles
        for i in range(len(file_time_list) - 1):
            Sholder_Vel = calculate_Vel(file_Shoulder_Ang_list,
↪file_time_list, i)
            file_Shoulder_Vel_list.append(Sholder_Vel)

        # Extract elbow and shoulder Accelerations [rad/sec^2] from
↪velocities
        for i in range(len(file_time_list) - 2):
            Sholder_Acc = calculate_Acc(file_Shoulder_Vel_list,
↪file_time_list, i)
            file_Shoulder_Acc_list.append(Sholder_Acc)

        # Adjust lists length
        adjusted_file_time_list = file_time_list[:-2]
        adjusted_file_Shoulder_Ang_list = file_Shoulder_Ang_list[:-2]
        adjusted_file_Shoulder_Vel_list = file_Shoulder_Vel_list[:-1]
        adjusted_file_Shoulder_Acc_data_list =
↪file_Shoulder_Acc_data_list[:-2]

```

```

time_list.append(adjusted_file_time_list)
walking_vel_list.append(walking_vel)

Sholder_Ang_list.append(adjusted_file_Sholder_Ang_list)
Sholder_Vel_list.append(adjusted_file_Sholder_Vel_list)
Sholder_Acc_list.append(file_Sholder_Acc_list)
Sholder_Acc_data_list.
→append(adjusted_file_Sholder_Acc_data_list)
    break

for file in file_list:
    if "00B429E2" in file:
        if file.endswith(".csv"):
            file_name = file[:-4]
            walking_vel = file.split("_")[4][:5]

            frame = 0
            file_time_list = []
            file_Elbow_Ang_list, file_Elbow_Vel_list,
→file_Elbow_Acc_list, file_Elbow_Acc_data_list = [], [], [], []

            # Cutting out weird data behavior on data edges
            data_path = os.path.join(data_csv_dir, folder, file)
            data_rows = open(data_path).read().strip().split("\n")[7500:
→9500]

            # Extract time [sec], elbow angles [rad], and shoulder
→angles [rad] from data
            for i in range(len(data_rows)):
                splitted_row = data_rows[i].strip().split("\t")

                # Check if loop finished all data
                if not len(splitted_row):
                    break

                file_time_list.append(frame / frame_frequency)
                file_Elbow_Ang_list.append((float(splitted_row[31]) -
→file_Sholder_Ang_list[i]) * 2*pi/360)
                file_Elbow_Acc_data_list.append(float(splitted_row[14]))
                frame += 1

            # Extract elbow and shoulder velocities [rad/sec] from
→angles
            for i in range(len(file_time_list) - 1):
                Elbow_Vel = calculate_Vel(file_Elbow_Ang_list,
→file_time_list, i)

```

```

        file_Elbow_Vel_list.append(Elbow_Vel)

        # Extract elbow and shoulder Accelerations [rad/sec2] from
→ velocities
        for i in range(len(file_time_list) - 2):
            Elbow_Acc = calculate_Acc(file_Elbow_Vel_list,
→ file_time_list, i)
            file_Elbow_Acc_list.append(Elbow_Acc)

        # Adjust lists length
        adjusted_file_Elbow_Ang_list = file_Elbow_Ang_list[:-2]
        adjusted_file_Elbow_Vel_list = file_Elbow_Vel_list[:-1]
        adjusted_file_Elbow_Acc_data_list =
→ file_Elbow_Acc_data_list[:-2]

        Elbow_Ang_list.append(adjusted_file_Elbow_Ang_list)
        Elbow_Vel_list.append(adjusted_file_Elbow_Vel_list)
        Elbow_Acc_list.append(file_Elbow_Acc_list)
        Elbow_Acc_data_list.append(file_Elbow_Acc_data_list)
        break

    for file in file_list:
        if "00B43DOC" in file:
            if file.endswith(".csv"):
                file_name = file[:-4]
                walking_vel = file.split("_")[4][:5]
                if walking_vel == "1.4ms":
                    continue

            frame = 0
            file_time_list = []
            file_Back_Ang_list, file_Back_Pos_list, file_Back_Vel_list,
→ = [], [], []

            # Cutting out weird data behavior on data edges
            data_path = os.path.join(data_csv_dir, folder, file)
            data_rows = open(data_path).read().strip().split("\n")[7500:
→ 9500]

            # Extract time [sec], elbow angles [rad], and shoulder
→ angles [rad] from data
            for i in range(len(data_rows)):
                splitted_row = data_rows[i].strip().split("\t")

                # Check if loop finished all data
                if not len(splitted_row):

```

```

        break

        file_time_list.append(frame / frame_frequency)

        file_Back_Ang_list.append(float(splitted_row[31]) *  $\rightarrow 2\pi/360$ )

        file_Back_Pos_list.append(float(splitted_row[21]))
        file_Back_Vel_list.append(float(splitted_row[24]))
        frame += 1

    # Adjust lists length
    adjusted_file_Back_Ang_list = file_Back_Ang_list[:-2]
    adjusted_file_Back_Pos_list = file_Back_Pos_list[:-2]
    adjusted_file_Back_Vel_list = file_Back_Vel_list[:-2]

    Back_Ang_list.append(adjusted_file_Back_Ang_list)
    Back_Pos_list.append(adjusted_file_Back_Pos_list)
    Back_Vel_list.append(adjusted_file_Back_Vel_list)
    break

```

current directory:

/home/yael/Documents/MSR_Courses/ME499-Final_Project/Motorized-Prosthetic-Arm/motor_control/arm_pendulum_modeling

1.4 System Modeling

Computing the Lagrangian of the system:

```

[18]: m, g, R, R_c = symbols(r'm, g, R, R_c')

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

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

# The velocity as derivative of position wrt t
theta_dot = theta.diff(t)

# The acceleration as derivative of velocity wrt t
theta_ddot = theta_dot.diff(t)

# Converting the polar coordinates to cartesian coordinates
x = R_c * sin(theta)
y = -R_c * cos(theta)

# Calculating the kinetic and potential energy of the system
KE = 1/2 * m * ((x.diff(t))**2 + (y.diff(t))**2)

```

```

PE = m * g * y

# Computing the Lagrangian
L = simplify(KE - PE)
Lagrange = Function(r'L')(t)
display(Eq(Lagrange, L))

```

$$L(t) = R_c m \left(0.5 R_c \left(\frac{d}{dt} \theta(t) \right)^2 + g \cos(\theta(t)) \right)$$

Computing the Euler-Lagrange equations:

```

[19]: # Define the derivative of L wrt the functions: x, xdot
L_dtheta = L.diff(theta)
L_dtheta_dot = L.diff(theta_dot)

# Define the derivative of L_dxdot wrt to time t
L_dtheta_dot_dt = L_dtheta_dot.diff(t)

# Define the right hand side of the the Euler-Lagrange as a matrix
rhs = simplify(L_dtheta_dot_dt - L_dtheta)

# Define the left hand side of the the Euler-Lagrange as a Matrix
lhs = tau

# 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:

$$\tau(t) = R_c m \left(1.0 R_c \frac{d^2}{dt^2} \theta(t) + g \sin(\theta(t)) \right)$$

Solve the equations for τ_1 and τ_2 :

```

[20]: # # Solve the Euler-Lagrange equations for the shoulder and elbow torques
# T = tau
# 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
```

Simulating the system:

```
[22]: # Substitute the derivative variables with a dummy variables and plug-in the
      ↪ constants
solution_subs = rhs

theta_dot_dummy = symbols('thetadot')
theta_ddot_dummy = symbols('thetaddot')

solution_subs = solution_subs.subs([(g, 9.81)])

solution_subs = solution_subs.subs([(theta.diff(t)).diff(t),
      ↪ theta_ddot_dummy)])
solution_subs = solution_subs.subs([(theta.diff(t), theta_dot_dummy)])

# Lambdify the thetas and its derivatives
func = lambdify([theta, theta_dot_dummy, theta_ddot_dummy,
                 m, R, R_c], solution_subs, modules = sympy)

# Initialize the torque and power lists
Elbow_tau_list = []
Elbow_current_list = []
Elbow_power_list = []

motor_kv = 115
torque_const = 8.27 / motor_kv

for i in range(len(time_list)):
    # Initialize the torque and power lists
    tau_list = []
    current_list = []
    power_list = []

    t_list = time_list[i]
    theta_list = Elbow_Ang_list[i]
    dtheta_list = Elbow_Vel_list[i]
    ddtheta_list = Elbow_Acc_list[i]

    # Plug-in the angles, angular velocities and angular accelerations for
    ↪ every time step to find the torques
    for j in range(len(t_list)):
        tau_list.append(func(theta_list[j], dtheta_list[j], ddtheta_list[j],
        ↪ m_l, L_l, L_l_c))
```

```

        # Calculate the current required to reach the required joints torques
        →for every time step
        current_list.append(torque_const * tau_list[j])

        # Calculate the power required to reach the required angular velocities
        →and joints torques for every time step
        power_list.append(dtheta_list[j] * tau_list[j])

    Elbow_tau_list.append(tau_list)
    Elbow_current_list.append(current_list)
    Elbow_power_list.append(power_list)

    print(f"Velocity {walking_vel_list[i]}\t max torque: {format(max(tau_list),
    →'.3f')}[Nm]\t max angular velocity: {format(max(dtheta_list), '.3f')}[rad/
    →sec]\t max power: {format(max(power_list), '.3f')}[W]")

```

```

Velocity 0.5ms    max torque: -0.021[Nm]    max angular velocity: 1.625[rad/sec]
max power: 0.954[W]
Velocity 0.6ms    max torque: -0.057[Nm]    max angular velocity: 1.731[rad/sec]
max power: 1.175[W]
Velocity 0.7ms    max torque: 0.137[Nm]      max angular velocity: 2.556[rad/sec]
max power: 1.862[W]
Velocity 0.8ms    max torque: 0.795[Nm]      max angular velocity: 3.695[rad/sec]
max power: 3.006[W]
Velocity 0.9ms    max torque: 0.202[Nm]      max angular velocity: 3.370[rad/sec]
max power: 2.815[W]
Velocity 1.0ms    max torque: 0.423[Nm]      max angular velocity: 3.588[rad/sec]
max power: 2.420[W]
Velocity 1.1ms    max torque: 0.565[Nm]      max angular velocity: 2.989[rad/sec]
max power: 1.962[W]
Velocity 1.2ms    max torque: 0.495[Nm]      max angular velocity: 3.567[rad/sec]
max power: 2.478[W]
Velocity 1.3ms    max torque: 0.890[Nm]      max angular velocity: 4.525[rad/sec]
max power: 3.748[W]
Velocity 1.4ms    max torque: 1.278[Nm]      max angular velocity: 4.931[rad/sec]
max power: 5.166[W]
Velocity chang    max torque: 0.368[Nm]      max angular velocity: 3.920[rad/sec]
max power: 2.851[W]

```

Calculation summary:

```

[39]: max_Elbow_tau, max_Elbow_power, max_Elbow_Vel = 0, 0, 0
      max_Elbow_tau_index, max_Elbow_power_index, max_Elbow_Vel_index = 0, 0, 0

      for i in range(len(Elbow_tau_list)):
          if max_Elbow_Vel < max(Elbow_Vel_list[i]):
              max_Elbow_Vel = max(Elbow_Vel_list[i])

```

```

max_Elbow_Vel_index = i

if max_Elbow_tau < max(Elbow_tau_list[i]):
    max_Elbow_tau = max(Elbow_tau_list[i])
    max_Elbow_tau_index = i

if max_Elbow_power < max(Elbow_power_list[i]):
    max_Elbow_power = max(Elbow_power_list[i])
    max_Elbow_power_index = i

print(f"maximum elbow angular velocity is {format(max_Elbow_Vel, '.3f')} [rad/
→sec] ({format(max_Elbow_Vel*60/(2*pi), '.3f')} [rpm]), in velocity_
→{walking_vel_list[max_Elbow_Vel_index]} (trial {max_Elbow_Vel_index})")
print(f"maximum elbow torque is {format(max_Elbow_tau, '.3f')} [Nm], in_
→velocity {walking_vel_list[max_Elbow_tau_index]} (trial_
→{max_Elbow_tau_index})")
print(f"maximum elbow power is {format(max_Elbow_power, '.3f')} [W], in_
→velocity {walking_vel_list[max_Elbow_power_index]} (trial_
→{max_Elbow_power_index})")

# The torque equations for the maximum power:
solution_subs = solution_subs.subs([(m, m_1), (R, L_1), (R_c, L_1_c), (g, 9.
→81)])

print("\nThe torque equations for the maximum torque:")
display(Eq(tau, solution_subs))

# display(Elbow_Ang_list[max_Elbow_tau_index])
# display(Elbow_Vel_list[max_Elbow_tau_index])
# display(Elbow_Acc_list[max_Elbow_tau_index])
# display(Elbow_tau_list[max_Elbow_tau_index])
# display(Elbow_Ang_list[2])
# display(Elbow_tau_list[2])

```

maximum elbow angular velocity is 4.931 [rad/sec] (47.091 [rpm]), in velocity 1.4ms (trial 9)
maximum elbow torque is 1.278 [Nm], in velocity 1.4ms (trial 9)
maximum elbow power is 5.166 [W], in velocity 1.4ms (trial 9)

The torque equations for the maximum torque:

$$\tau(t) = 0.04217031288\ddot{\theta} + 1.732373406 \sin(\theta(t))$$

Example for the trial with the largest elbow torque & power:

```

[34]: index = 2
      t_list = time_list[index]

```

```

theta_list = Elbow_Ang_list[index]
dtheta_list = Elbow_Vel_list[index]
ddtheta_list = Elbow_Acc_list[index]
tau_list = Elbow_tau_list[index]
current_list = Elbow_current_list[index]
power_list = Elbow_power_list[index]

back_rotation_list = Back_Ang_list[index]
back_position_list = Back_Pos_list[index]
back_velocity_list = Back_Vel_list[index]

Elbow_Acceleration_list = Elbow_Acc_data_list[index]

# Compute the trajectory of the arm's motion
N = int((max(t_list) - min(t_list))/(1/frame_frequency))
tvec = np.linspace(min(t_list), max(t_list), N)
traj = np.zeros((3, N))
back_traj = np.zeros((3, N))
acc_traj = np.zeros((2, N))
partial_traj = np.zeros((3, N))

for i in range(N):
    traj[0, i] = theta_list[i]
    traj[1, i] = dtheta_list[i]
    traj[2, i] = ddtheta_list[i]

    back_traj[0, i] = back_rotation_list[i]
    back_traj[1, i] = back_position_list[i]
    back_traj[2, i] = back_velocity_list[i]

    acc_traj[0, i] = Elbow_Acceleration_list[i]

for i in range(500):
    partial_traj[0, i] = theta_list[i]
    partial_traj[1, i] = dtheta_list[i]
    partial_traj[2, i] = ddtheta_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 and Back Movement Vs. Time', fontsize=20)
plt.plot(t_list[:-diff], traj[0], label="Elbow angle")
plt.plot(t_list[:-diff], back_traj[0], label="Back motion")

```

```

plt.ylabel('Angle [rad]')
plt.xlabel('Time [sec]')
plt.xlim([0, math.ceil(max(tvec)/2)])
plt.grid()
plt.legend()
plt.title('Elbow Angle')
plt.show()

plt.figure(figsize=(15,5))
plt.suptitle('Arm Acceleration and Back Movement Vs. Time', fontsize=20)
plt.plot(t_list[:-diff], traj[2], label="Elbow acceleration")
plt.plot(t_list[:-diff], back_traj[0], label="Back motion")
plt.ylabel('Angle [rad]')
plt.xlabel('Time [sec]')
plt.xlim([0, math.ceil(max(tvec)/2)])
plt.grid()
plt.legend()
plt.title('Elbow Acceleration')
plt.show()

plt.figure(figsize=(15,5))
plt.suptitle('Angles Vs. Time', fontsize=20)
plt.plot(t_list[:-diff], traj[0])
plt.ylabel('Angle [rad]')
plt.xlabel('Time [sec]')
plt.xlim([0, math.ceil(max(tvec))])
plt.grid()
plt.title('Elbow Angle')
plt.show()

plt.figure(figsize=(15,5))
plt.suptitle('Angular Velocity Vs. Time', fontsize=20)
plt.plot(t_list[:-diff], traj[1])
plt.ylabel('Velocity [rad/sec]')
plt.xlabel('Time [sec]')
plt.xlim([0, math.ceil(max(tvec))])
plt.grid()
plt.title('Elbow Angular Velocity')
plt.show()

plt.figure(figsize=(15,5))
plt.suptitle('Angular Acceleration Vs. Time', fontsize=20)
plt.plot(t_list[:-diff], traj[2])
plt.ylabel('Acceleration [rad/sec^2]')
plt.xlabel('Time [sec]')
plt.xlim([0, math.ceil(max(tvec))])
plt.grid()

```

```

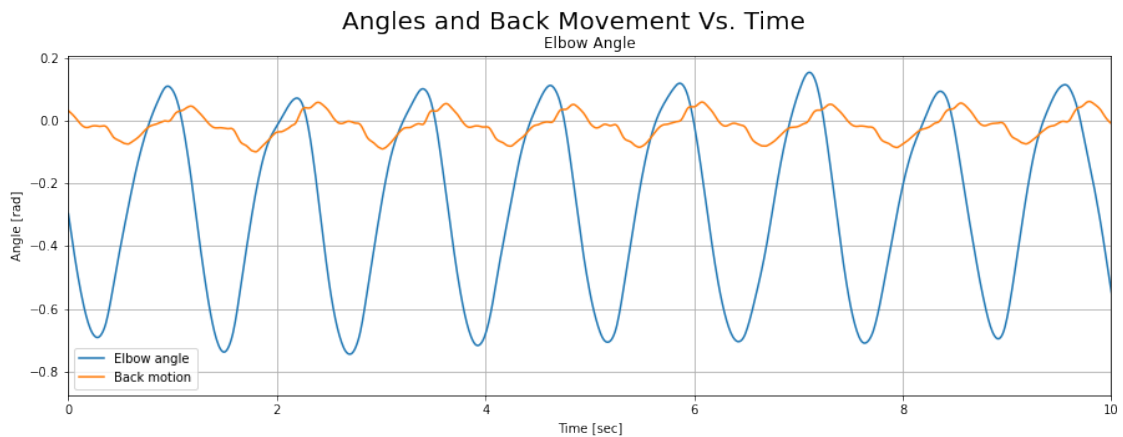
plt.title('Elbow Angular Acceleration')
plt.show()

plt.figure(figsize=(15,5))
plt.suptitle('Torque Vs. Time', fontsize=20)
plt.plot(t_list, tau_list)
plt.ylabel('Torque [Nm]')
plt.xlabel('Time [sec]')
plt.xlim([0, math.ceil(max(tvec))])
plt.grid()
plt.title('Elbow Torque')
plt.show()

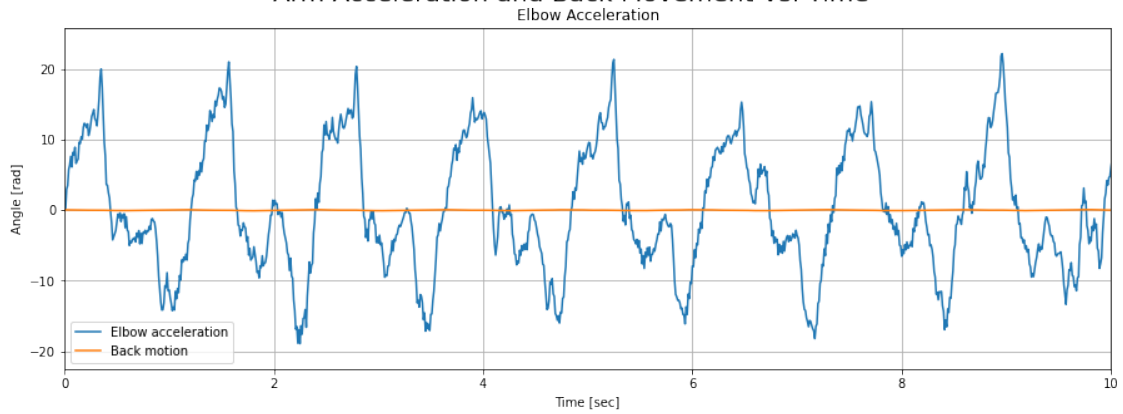
plt.figure(figsize=(15,5))
plt.suptitle('Power Vs. Time', fontsize=20)
plt.plot(t_list, power_list)
plt.ylabel('Power [W]')
plt.xlabel('Time [sec]')
plt.xlim([0, math.ceil(max(tvec))])
plt.grid()
plt.title('Elbow Power')
plt.show()

plt.figure(figsize=(15,5))
plt.suptitle('Speed Vs. Torque', fontsize=20)
plt.plot(tau_list[:-diff], traj[1])
plt.ylabel('Velocity [rad/sec]')
plt.xlabel('Torque [Nm]')
plt.grid()
plt.title('Elbow Speed-Torque')
plt.show()

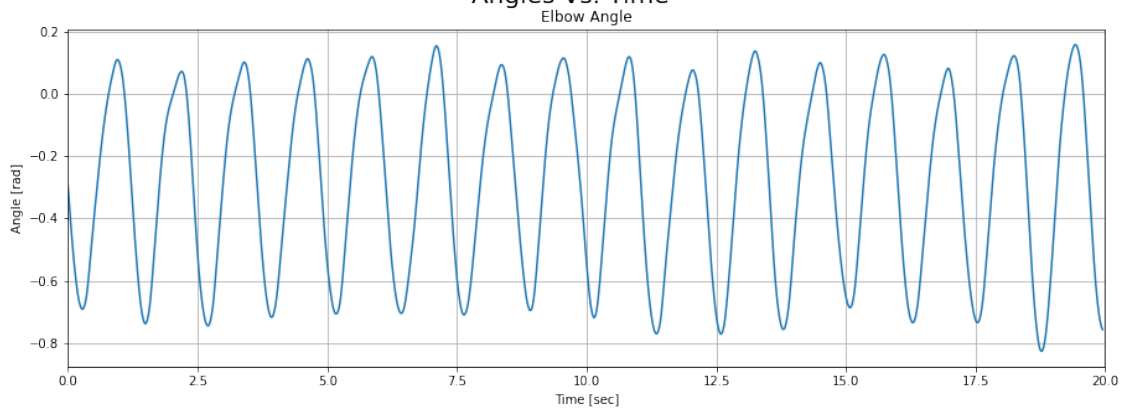
```



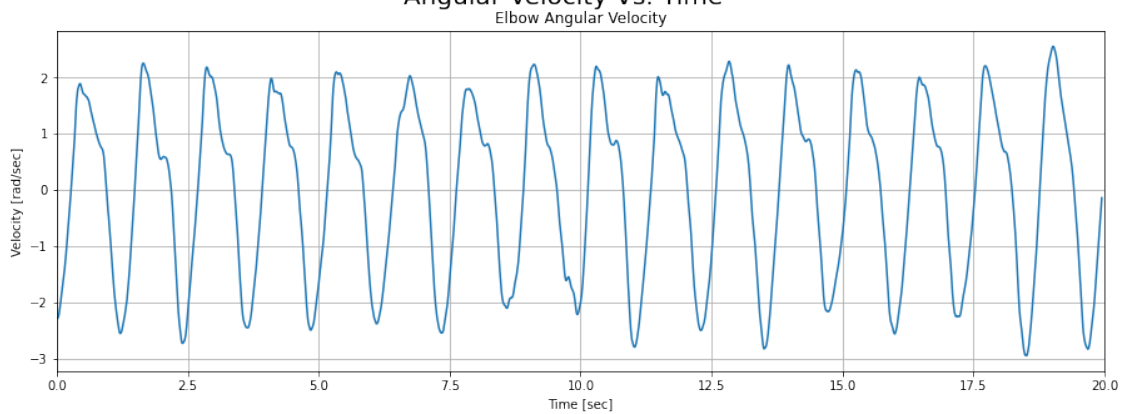
Arm Acceleration and Back Movement Vs. Time

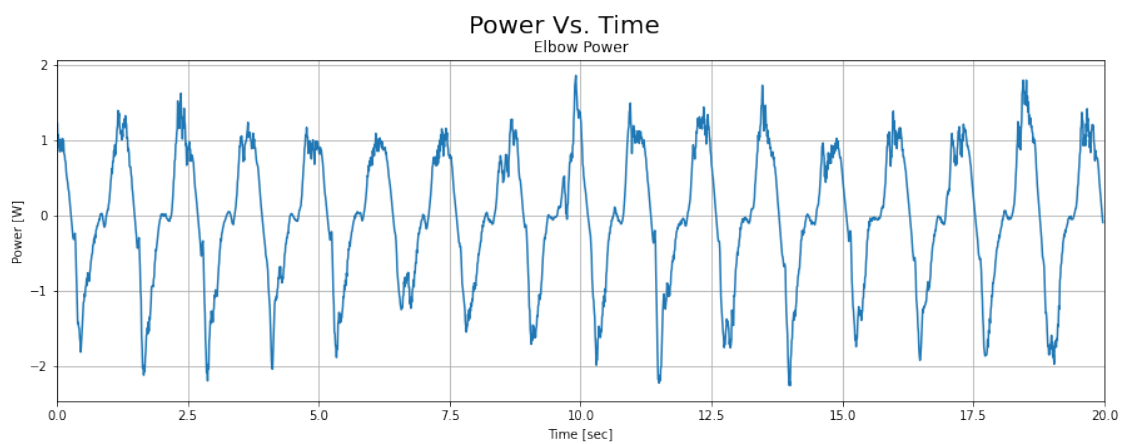
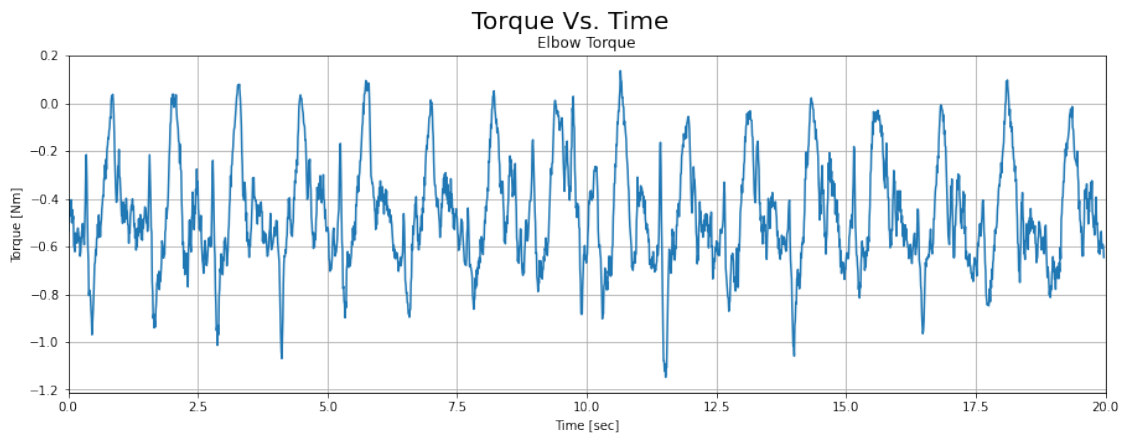
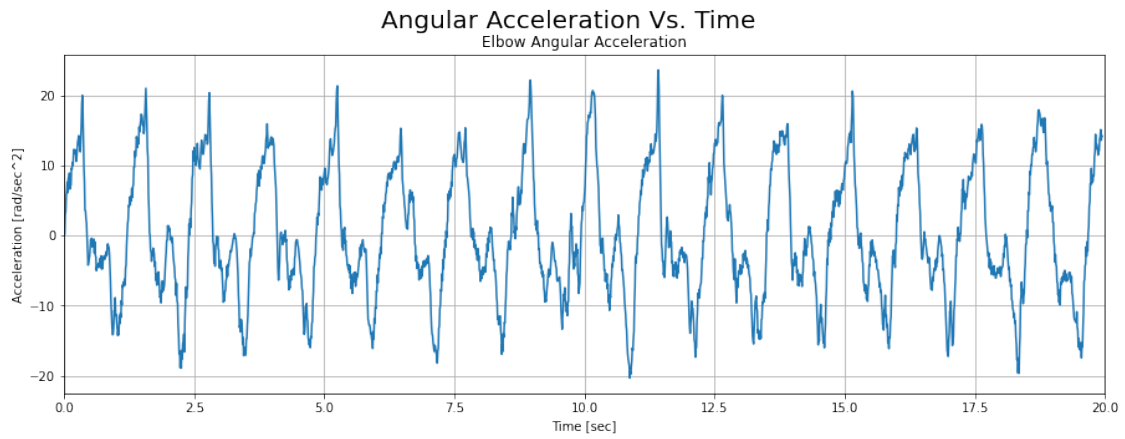


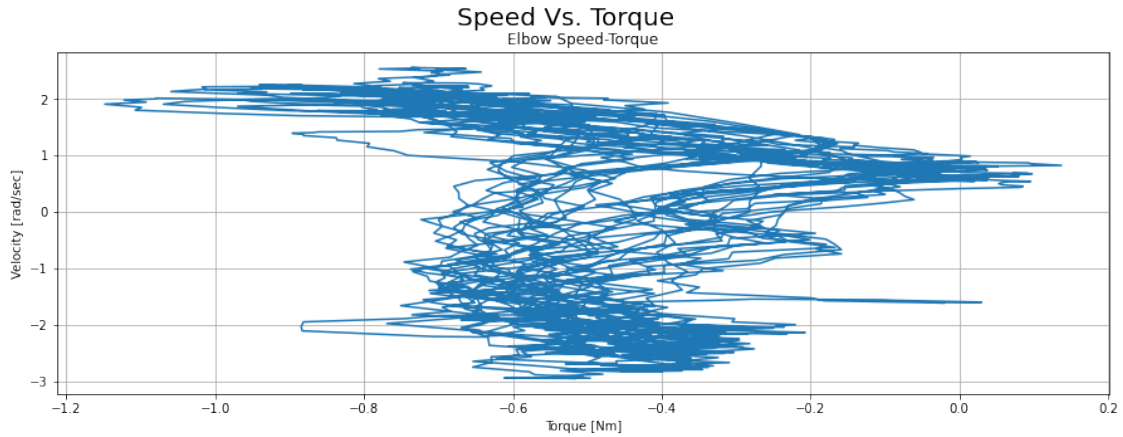
Angles Vs. Time



Angular Velocity Vs. Time







Animating the simulation:

```
[40]: def animate_double_pend(traj, L, L_c, T):
    """
    Function to generate web-based animation of double-pendulum system

    Parameters:
        traj:      trajectory of theta1 and theta2
        L:          length of the lower arm
        L_c:        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
xx = L * np.sin(traj[0])
yy = -L * np.cos(traj[0])
xx_c = L_c * np.sin(traj[0])
yy_c = -L_c * np.cos(traj[0])
N = len(traj[0])

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

# Defining data dictionary
data = [dict(x=xx, y=yy,
             mode='lines', name='Arm',
             line=dict(width=5, color='blue')
            ),
        dict(x=xx_c, y=yy_c,
             mode='lines', name='Lower Arm Center of Mass',
             line=dict(width=2, color='green')
            ),
        dict(x=xx, y=yy,
             mode='markers', name='Elbow Trajectory',
             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='Simulation of Arm Modeled as a Double Pendulum',
              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, xx[k]],
                              y=[0, yy[k]],
                              mode='lines',
                              line=dict(color='red', width=4)),
                    go.Scatter(
                        x=[xx_c[k]],
                        y=[yy_c[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
    L = L_1
    L_c = L_1_c
    T = 5

    animate_double_pend(partial_traj, L, L_c, T)

```

<IPython.core.display.HTML object>

[]: