Get linear kinematics

#%% to get linear kinematics

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from scipy.signal import butter, filtfilt

from scipy.interpolate import interp1d

# filtering -> cutoff frequency set up 6

def butterworth\_filter(data, cutoff, fs, order=4, filter\_type='low'):

nyquist = 0.5 \* fs

return filtfilt(\*butter(order, cutoff / nyquist, btype=filter\_type), data, axis=0)

# time derivate

def time\_d(data, interval):

return np.gradient(data, interval, axis=0)

# time derivate

def time\_dd(data, interval):

return np.gradient(np.gradient(data, interval, axis=0), interval, axis=0)

# calculate relative angle

def calculate\_theta(data1, data2):

m = np.shape(data1)[0]

theta = np.zeros(m)

for i in range(m):

A = data1[i, :]

B = data2[i, :]

nor\_A = A / np.linalg.norm(A)

nor\_B = B / np.linalg.norm(B)

R = np.array([[nor\_A[0], nor\_A[1]], [-nor\_A[1], nor\_A[0]]])

c = np.dot(R, nor\_B)

theta[i] = np.arctan2(c[1], c[0])

return theta

# if we have gimbal lock -> use this function to fix it

def unwrap\_deg(data):

data = data.copy()

dp = np.diff(data)

dps = np.mod(dp + np.pi, 2 \* np.pi) - np.pi

dps[np.logical\_and(dps == -np.pi, dp > 0)] = np.pi

dp\_corr = dps - dp

dp\_corr[np.abs(dp) < np.pi] = 0

data[1:] += np.cumsum(dp\_corr)

return data

# input data

file\_path = '/Users/kairenzheng/Library/CloudStorage/OneDrive-AuburnUniversity/KINE7670\_homeworks/opencap\_study/data\_opencap\_squat\_trc/VJ.csv'

# run data

df = pd.read\_csv(file\_path, header=None)

# get frequency

fs = int(float(df.iloc[2, 1]))

# get time interval = 1 / ferquency

sampling\_interval = 1 / fs

# pick up positions

segment\_names = ['mid\_hip', 'Rhip', 'RKnee', 'RAnkle', 'Lhip', 'LKnee', 'LAnkle']

segment\_columns = [(23, 26), (26, 29), (29, 32), (32, 35), (35, 38), (38, 41), (41, 44)]

segments = {name: df.iloc[6:, start:end].astype(float).to\_numpy() for name, (start, end) in zip(segment\_names, segment\_columns)}

# set up elements

cutoff = 6

# run butterworth\_filter

filtered\_segments = {name: butterworth\_filter(seg, cutoff, fs) for name, seg in segments.items()}

# get lineart velocity

linear\_velocity = {name: time\_d(filtered\_segments[name], sampling\_interval) for name in segment\_names}

# get linear acceleration

linear\_acceleration = {name: time\_dd(filtered\_segments[name], sampling\_interval) for name in segment\_names}

# make graph

for name in segment\_names:

fig, axs = plt.subplots(3, 1, figsize=(12, 10))

fig.suptitle(f'{name} Position, Velocity, and Acceleration', fontsize=16)

# Pick data from event1 to event2

position\_data = filtered\_segments[name][event1:event2]

velocity\_data = linear\_velocity[name][event1:event2]

acceleration\_data = linear\_acceleration[name][event1:event2]

for idx, (data, label, ylabel) in enumerate(zip(

[position\_data, velocity\_data, acceleration\_data],

['Position', 'Linear Velocity', 'Linear Acceleration'],

['Position (m)', 'Velocity (m/s)', 'Acceleration (m/s²)'])):

axs[idx].plot(data[:, 0], label='X')

axs[idx].plot(data[:, 1], label='Y')

axs[idx].plot(data[:, 2], label='Z')

axs[idx].set\_title(f'{name} {label}')

axs[idx].set\_xlabel('Frame')

axs[idx].set\_ylabel(ylabel)

axs[idx].legend(loc='upper right')

plt.tight\_layout(rect=[0, 0, 1, 0.96])

plt.show()