**Instruction on how to apply the LD model on kinematic Data**

Step-by-step explanation on how to collect and pre-process IMU data, use the LD model to predict load distribution, and export the full dataset to a csv-file. The dataset that was used for the current study is publicly available via: <https://data.4tu.nl/datasets/bc9a8588-5e50-4dff-aa77-5114ff7626f7>.

1. Collect kinematic data  
   - of coast-down tests with different (known) load distributions  
   - of the situation / actions of interest (if IMUs are used to collect the data: Attach one IMU to the wheelchair wheel axle, one IMU to the center of the wheelchair frame and one IMU around the chest against the sternum)
2. Determine from the data:

* The rolling resistance coefficients per pair of wheels
* The acceleration vector perpendicular to the trunk
* The wheelchair velocity
* The wheelchair acceleration
* Time

1. Save the above-mentioned variables in the following order:  
   i.e., df["samples","time","v\_wc", "a\_wc", "trunk\_acc\_3"]
2. Run the code below in python1
3. Done

﻿"""

Created on Fri Nov 17 14:12:28 2023

Predict Relative Front Wheel Load (RFWL) during wheelchair propulsion

@author: L.H.A. Heringa & M.P. van Dijk

"""

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from keras.layers import Dense, LSTM, Dropout, GaussianNoise

from keras.callbacks import EarlyStopping, ModelCheckpoint

from keras.metrics import MeanAbsoluteError

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from keras.optimizers import Adam

from sklearn.utils import shuffle

from keras.utils.vis\_utils import plot\_model as pm

from datetime import datetime

from keras.models import load\_model

# Run functions

def standardize(signal):

size\_temp = np.asarray(signal.shape)

size = size\_temp[0].item()

standarized\_signal =np.zeros((size,3))

i = 0

while i < 3:

z\_score = (signal[:,i]-mean[i])/std[i]

standarized\_signal[:,i] = z\_score

i = i + 1

return standarized\_signal

def load(x\_data, num\_steps):

#x\_data = data.iloc[:,[2,3,4]].to\_numpy()

X = list() #reshape the 2d data into 3d sliding window shape

for i in range(x\_data.shape[0]):

end\_ix = i + num\_steps # compute a new (sliding window) index

if end\_ix >= x\_data.shape[0]: # if index is larger than the size of the dataset, stop

break

seq\_X = x\_data[i:end\_ix] # Get a sequence of data for x

X.append(seq\_X)

x\_array = np.array(X)

return x\_array

# Load model

model = '/../final\_lstm.h5'

# Load model

df\_full = pd.read\_csv(r'/../IMU\_Data.csv')

df = df\_full[["samples","time","v\_wc", "a\_wc", "trunk\_acc\_3"]]

df = df\_full[["samples","time","v\_wc", "a\_wc", "trunk\_acc\_3"]]

mean = np.mean(df, axis=0)

std = np.std(df, axis=0)

df\_new = df.to\_numpy()

mean = np.mean(df\_new, axis=0)

std = np.std(df\_new, axis=0)

df\_new = standardize(df\_new[:,2:5])

# Run and plot LSTM model

num\_steps = 20

X = load(df\_new,num\_steps)

lstm = load\_model(model)

y = lstm.predict(X)

plt.plot(y)

﻿# Add force output to initial dataset

RFWL\_Output = pd.DataFrame(y)

df\_full\_new = df\_full[num\_steps:df\_full.shape[0]]

df\_full\_new['RFWL\_Output'] = y

# Export dataset to csv

df\_full\_new.to\_csv(r'/../IMU\_Data\_plus\_predictedFrontWheelLoad.csv')

1 The code provided here is written in Spyder. In general, Python code is portable across different environments, such as Spyder, PyCharm and Jupyter, as they all support standard Python syntax. Nevertheless, small adjustments may be required.