READING AND PREPROCESSING OF ALPINE SKI MOTION TRACKING DATA

0.INTRODUCTION

The main aim of this project is to analyse some different type of Alpine Skiing's short sessions. This is done by recording Accelerometers and Gyroscope sensors data, evaluate them and search for some patterns or similarity with a simple plotting.
Four Different classes of standard and recurrent motion situations are considered:

1. Skiing Making Long Turns (Approximate Radius of more than 15m)
2. Skiing Making Short Turns (Approximate Radius of less than 10m)
3. Going Straight whit parallel skis or slightly braking with snowplough technique
4. Walking and moving on ground with minimal slipping behaviour

The data of the sessions are recorded with "Sensor Logger" app installed on android device "RENO 4PRO 5G".
The data have been recorded on different slopes of the Italian Ski Resorts of "Corno alle Scale (BO)" on 18/01/26 and "Le Polle (MO)" on 11/01/26.
A dozen of records for each class have been made, and they are lasting from 30seconds to 1 minute.

DATA BIAS DISCLAIMER: the dataset recorded are just a first try to start understanding the peculiar form and structure of skiing dynamics. It can't be used as a baseline for further High-Quality ML algorithms and models training since they are biased and may lead to overfitting.

- They have been recorded by a just one skier (myself). Multiple skiers with different level, age, sex should be used
- Data have been recorded in only two days. Multiple days recording with the most various possible range of snow and weather conditions should be implemented
- We are making a huge simplification by assuming that the phone is always in the same position and in the same pocket of the same trousers. Different positions over body can lead to different results.
- Data are simply too few!

1.EXTRACT

The first process that has been implemented is the extraction of correct storage of the data from the phone to a Personal Computer.
Files are uploaded on the PC with a simple USB transfer as .zip folders, one for each record, and are named by default "given_name-date_time.zip". Each folder is containing multiple sensors time series data files.

With the file extract.py we are then implementing the following functions:

- **extract_and_load_all**:
  INPUT: .zips folder paths and output file folder for saving .csv
  RETURN: data saved in a dictionary for logging output
  Main function where all .zip file are extracted and accessed
  All selected and only needed files are saved as .csv in a new created folder "Data\interim\extracted"

Following functions are called by this one iteratively.

- **find_sensor_members:**
  INPUT: paths of every sensor data for the current zip folder
  RETURN: paths of only the desired sensors
  It check among each file which is correspond to the wanted one

- **parse_recording_id:**
  INPUT: current zip file
  RETURN: tuple class – recID
  The class value and the recording ID number are only stored in .zip folder name while inner files names lose every dependence or relation. With this function we are creating a tuple with class and Id number of the record that will be use in .csv files savings.

2.MERGE

Witt the previous function we are getting many files naming <class>_<ID>_<sensor>.csv like for example “wlk_03_Accelerometer.csv”. Now we have to aggregate all the different sensors’ columns in on record related unique table and save it in a <class>_<ID>.csv file.

With the file merge.py we are then implementing the following functions:

- **run_merge:**
  INPUT: input extracted files folder, output merge folder
  RETURN: none
  It is running the merge process for all files in input folders and displaying some logger information.

- **discover_files:**
  INPUT: input folder
  OUTPUT: two-layer dictionary of paths
  In this function we are creating a double nested dictionary with the following structure.
  { key_1(rec_key name) : { key_2(specific sensor) : associated path, … }}.
  So I now have every sensor data path associated with his correct sensor and correct recording.

- **merge_recording:**
  INPUT: class_ID pair for current recording, paths layer of dictionary, output folder path
  RETURN: output path of merged file for logging information
  In this function we are defining the common window and its steps for time alignment of each column.
  Then same recordings files are merged and saved as .csv

- **load_sensor_frame:**
  INPUT all files’ paths and sensor list
  RETURN a data frame containing the current sensor values
  In this function we read each sensor files and we ensure correct data format of each line and correct title names for columns’ indexes.

- **resample_to_grid**:
  INPUT: data frame of one sensor, target time window, absolute time start and end
  RETURN: resampled sensor data ready to be merged
  With this function we are forcing to resample the data to align with a common master timeline shared by all other sensors files of each recording.
  We add to the original timeline the master one and we interpolate on it.

3.TRIM

We have obtained one per recording .csv files all referring to a common timeline and all containing every inertial data that is need in column in the same order. In this step we are implementing a simple cut and delete of the head and tail data.
This is because we want to discard the dynamics of: A) Start recording and putting phone in pocket – B) Extracting the phone from the pocket and end the recording.

With the trim.py file we are implementing the following functions:

- **process_file**:
  INPUT: input file and output folder path
  OUTPUT: bool for checking correct process execution and logging information
  We read the .csv pointing to input path as dataframes, we define the time point for start and ending trimming (3 seconds from start and 4 seconds before end have been used). The trimming function is then called

- **trim_dataframe:**
  INPUT: dataframe, time parameters
  OUTPUT: processed dataframe
  We apply a simple data frame filtering with df.loc and (t_min + trim-start ; t_max – trim_end) values.

4.SMOOTHING

We have excluded undesirable values from our dataset. Another preprocessing step that it can be implemented is data filtering since IMU comes with a lot of high frequency noise effect due to unwanted phone movements, ground impulsive impacts and vibration.
Some methods that could be implemented are:

- Exponential Moving Average (weighted mean with higher weight when closer values)
- Low Pass Filter with a fix cutting frequency
- Savitzky-Golay polynomial fitting

Here we are implementing the 3rd one SG method which come with a SciPy library.

The implementation is quite simple and comes in a single function called smooth_columns where each recording .csv file is loaded as a pd dataframe and then each column is split in multiple numpy series to be processed by savgol_filter function.
The parameters used are a window length of 21 (+-10 samples from the central one) and a 3rd grade for polynomial interpolation.

5.WINDOWING

Now data that are ready to use in Machine Learning models for analysis or classification works.
Another thing that has been implemented and that it can turn out to be useful to enhance model's performance is windowing.
Basically, the intention is to fragment every recording in multiple shorter takes in order to augment the number of inputs to train a potential method.
For this case, taking turns classes as example, it can be extremely useful. In a single record it may happen to have different radius turns, different speed turns and differences in other variables. It also may happen to have some outliers, other classes turns and unwanted dynamic situations inside the recording (braking, stopping). Also, some bug or errors may incur in some in registrations.
Diving in windows stop to these errors to influence all the registrations and lead to reduce their influence on the final results.

In the window.py file the following functions have been implemented:

- **run_window**:
  In this function input and output folders are defined then we are calling process_file function for every path over every .csv files. Some logger information output is also implemented.

- **generate_window_slices**:
  INPUT: timing parameters
  OUTPUT: tuple list of start-stop steps
  This function is called to define windows start and stop time values.
  The total windows length has been chosen at 6 seconds a quantity able to understand and get the dynamic or around two long turns.
  The stride quantity as the gap quantity the distance between window and implying overlapping has been set at 1,5 seconds leading to a 75% overlapping between windows.
  The function is checking if the ongoing processing window is valid and not exceeding the end of data.

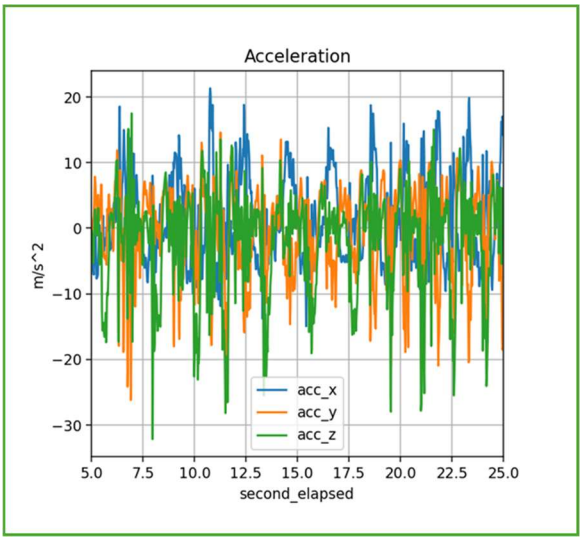
- **process_file**:
  INPUT: input – output folders path
  For each window this function extracts the associated rows and create a corresponding new sub-data frame which is then saved in a new csv. File.
  A renaming from class algorithm is also implemented in the for loops taking class label and recording ID and adding to them a 4 number label. We are obtaining so windows like "wlk_04_0012" standing for the 12^{th} window of the 4^{th} class of class walking.

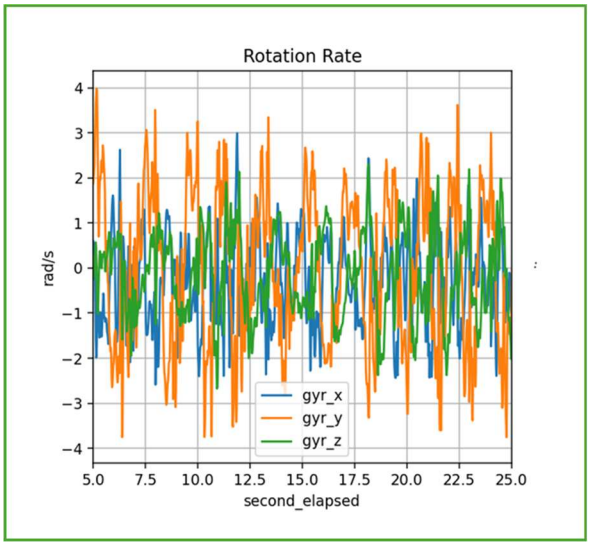
6.REPORTING

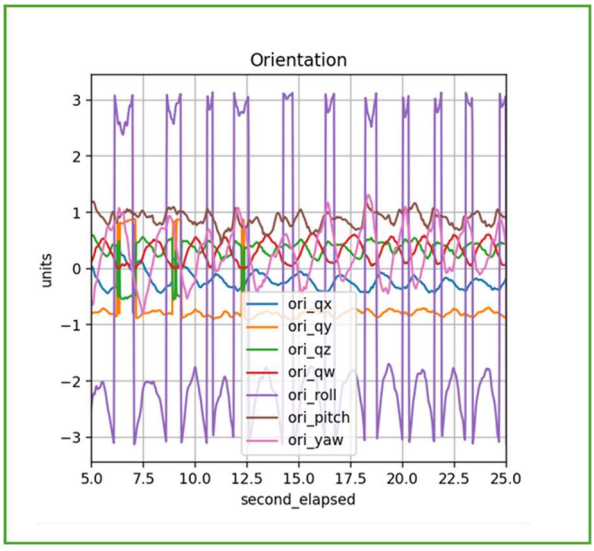
To show some results of the entire data preprocessing procedure that has been described a simple plotting has been implemented on file **plot_data.py.** By running it is possible to choose a file and visualize a timeline plot of:

- Accelerometer values
- Rotation rate
- Orientation values

An example of short turn taken from file “srt_12.csv” is displayed in this page:

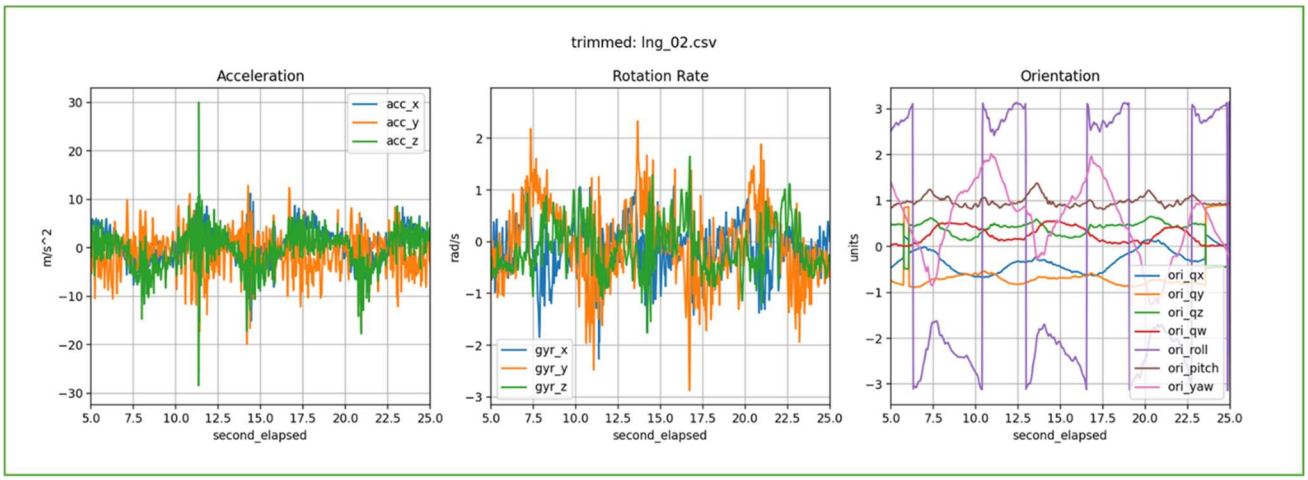


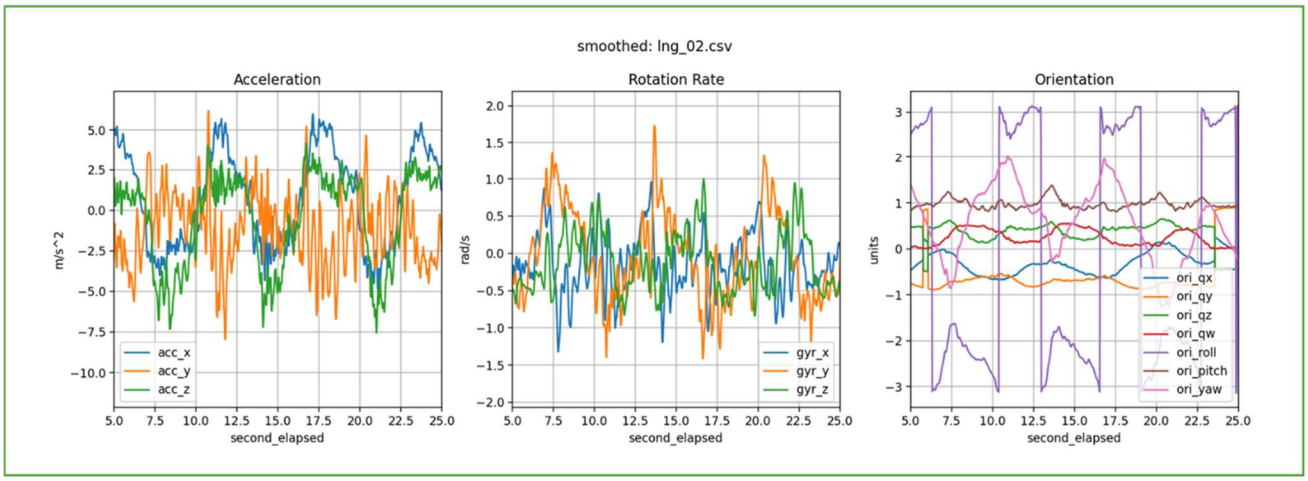




In the same plot_data.py file a comparison between raw data and smoothed ones are displayed to evaluate the noise filtering.

Here a comparison between unfiltered and filtered long turns taken from “lng_02.csv”:





We can notice that a lot of oscillations are reduced but that a lot of noise still to be removed to get a better smoothness of the data changes over time.

THANKS FOR THE ATTENTION!

Alessandro Razzoli – Mechatronic Engineering Student - MATR:194640

274499@studenti.unimore.it – alerazzo2000@gmail.com