TFMESH

TFMESH (pronounced TF-Mesh) is a package for **T**rajectory **F**using, **M**otion **E**stimation and **S**top **H**andling.

This package performs the following tasks:

- Basic I/O and processing
  - load trajectory in a close-to-NGSIM or customized format
  - save trajectory data in NGSIM format
  - parallel process raw vehicle data
- RTS Smoother
  - perform trajectory fusing if multiple observations are available
  - produce smoothed position data
  - produce estimated velocity and acceleration
- RANSAC stop detection and handling
  - perform stop detection
  - perform stop handling, including spline interpolation to connect moving and stopped parts

Getting started

System requirement

Packaged required in this project is listed in `requirements.txt`. It is recommended to use Anaconda distribution as most packages are already self-contained. If using Anaconda environment, the only additional package required is `filterpy` which can be installed by running `pip install filterpy`.

The code is developed using Python 3.8.3, but should be working on most modern Python releases.

Example and tutorial

To begin, start by looking `main.ipynb`, which contains the step-by-step procedures from importing a raw vehicle trajectory data to smoothed trajectory with motion data.

This package TFMESH uses two well established methods: an RTS smoother for "TFME" and a RANSAC based detector for "SH". The corresponding original papers and tutorials are summarized in the following table.

Topic	Paper	Tutorial
RTS Smoother	[1]	[2]
RANSAC	[3]	[4]

Input data format

The input data takes a format very close to NGSIM data, except some data columns are added to provide additional information needed for trajectory fusing.

> **IMPORTANT** (For people coming from MATLAB): The column number starts counting at 0. So column 5 below means column F in an excel sheet. Read more here.

By default, the code assumes the following data directory (this part is reflected in `encode_veh_ud()` under `libStep1.py`).

Column	Name	Variable
1	Frame ID	`IND_FID`
2	Camera ID	`IND_CAM_ID`
5	Position	`IND_POS`
13	Lane ID	`IND_LANE_ID`
21	Upstream/Downstream Flag	`IND_US_FLAG`

Alternatively, the user may specify the data directory they want, by creating a `.ini` file under `config/`. The following provides an example when using this package to process LiDAR data, which holds camera ID in a different column as seen in `IND_CAM_ID`.

```
[DATA]
# column of position
IND_POS = 5
# column of frame ID
IND_FID = 1
# column of camera ID
IND_CAM_ID = 18
# column of lane ID
IND_LANE_ID = 13
# column of upstream/downstream
IND_US_FLAG = 21
```

The path to configuration file can be declared by modifying variable `config_file_path` under `main.ipynb`. In the following example, the config is loaded from `config/lidar.ini` when the code runs.

```
# load configuration file
config_file_path = 'config/lidar.ini'
```

> **IMPORTANT**: the front/rear edges shall be properly shifted by one vehicle length in order to be fused correctly. The 'IND_US_FLAG' records whether position comes from the front or rear part of the vehicle but perform **NO** shifting. For example, if front tracking is desired, the rear edge position shall be shifted downstream by one vehicle length before running the code. This should be done in preprocessing step, and a working example in this case can be found under `preproc_data_lz_ngsim.m`.

(For internal reference only) Function hierarchy

As used in `main.ipynb`

- libFileio.py
  - get_veh
  - load_data
  - save_data_step1
  - save_data_step2
  - save_data
- libStep1.py
  - combine_cam_motion_est_ud
    - encode_veh_ud
    - est_init_v
    - init_ca
    - measurement_noise_model
- libStep2.py
  - ns_and_s_handle
    - index_true_region
    - index_stop_region
      - index_true_region
    - spline_near_stop