**This document describes most of the features of the python files, all required libraries and their versions.**

First, please make sure that your computer has ***PyCharm*** [PyCharm is an integrated development environment (IDE) used in computer programming, specifically for the Python programming language.], and the compiler version ***python3.7***. It is convenient to use the “pip command” and we strongly recommend it. Otherwise, if you are PyCharm users, go to “> File > Setting > Project: xxx > Python Interpreter” to see what you have as libraries and their version and to add libraries there, which is also convenient.

The libraries you should have on your computer to launch the test with success are:

pip 21.3.1 (If you want to use “pip command” and if you already have PyCharm, this should be included)

Open the command window and enter

pip3.7 install xxx [librariesname] ==xx.xx.xx[version]

For example: pip3.7 install tensorflow==2.3.0

to install all the following libraries:

keras 2.3.1

numpy 1.21.4

opencv-python 4.5.4.60

tensorflow 2.0.0 or 2.3.0

**The following describes the purpose and function of the files one by one:**

Rtmaps\_Control\_AMI.py:

Features:

This file is the core file and functions:

1. Generate vehicles (user vehicles) in Carla, generate pedestrians (NPCs), and generate other vehicles (NPCs). Parameters can also be adjusted to change the location where vehicles are generated. If you want to generate other items that Carla has provided for us, and if you want to get more sensors, please check the Carla website for more information.

2. For the vehicle (user vehicle) in Carla, add lidar, add IMU, add camera and many other sensors designed by Carla for us. We only used the above three cameras in this project. For more sensors, check out the Carla website for more information.

Inputs : One Vectorized input:

-is\_auto\_pilot : triggers auto-pilot

-is\_reverse : sets vehicle in reverse

-throttle : acceleration value between 0 and 1

-breaking : breaking value between 0 and 1

Outputs :

-speedKmh

-IMUlatton

-rollpitchyaw

-ptCloud : LIDAR point cloud

CSV\_Record\_AMI.py:

Features:

This file is used to record the trajectory of the car in the simulation and save it in a csv file traj.csv (you can find it in AMI\_Project/traj.csv).

Input:

* + in : IMUlatton from vehicle : latitude and longitude values from the IMU
  + Recording : Command to record, if equals to 0 makes the component inactive

Outputs : None

Detect\_points.py

Features:

Checks if lidar points from the simulation are in the bounding boxes defined in the code.

Depending on the mode (Dodge/Stop) and the detection, it will give information to Longi\_Speed for braking and Reference\_trajectory for dodging.

The output for Reference\_trajectory uses the size of object detected in front of the car.

Inputs:

* + In: Lidar points (ptCloud from Vehicle block)
  + Dodge\_1\_Stop\_0 : Push to dodge button

Outputs:

* + Out: Braking value (> 1 if braking needed)
  + Decalage\_out: offset to add to the trajectory to dodge
  + Obstacle\_out: 1 if obstacle detected, 0 if no obstacle
  + Points\_in\_bound: points detected as obstacle, only used in 3d viewer

Angle\_Erreur\_Controler\_AMI.py:

Features:

This file contains the lateral controller of the vehicle in which there is a line tracking algorithm called Pure Pursuit : Takes the first point at least x meters away, calculates the angle with it and applies a PID controller.

Inputs: -targetX : X values of the targeted points (/!\ : positive X : right of the car)

-targetY : Y values of the targeted points (/!\ : positive Y : front of the car)

Output:-Lat\_cmd : Lateral command in radians

-targetpoints : Targeted point visualized by the 3D viewer

Change\_Coordinates\_AMI.py:

Features:

Applies a change of basis using a transformation matrix to have the trajectory in the car’s reference frame. Only outputs the 4 next points

Input: One vectorized input :

-yaw : car’s yaw

-East : Easting of the car

-Nord : Easting of the car

-Also reads traj.csv which contains the recorded trajectory

Output: -targetX : X values of the trajectory points (/!\ : positive X : right of the car)

-targetY : Y values of the trajectory points (/!\ : positive Y : front of the car)

-targetXY : Trajectory outputted to the 3D viewer

Yaw\_Reader\_AMI.py:

Features:

Used to extract the yaw from the IMU. Also used to prevent the next modules to execute if the recording mode is enabled (does not output anything in this case and thus blocks the next modules)

Input: One vectorized input :

-Rollpitchyaw

-is\_record

Output:

-yaw

UTM\_Conversion\_AMI.py:

Features:

Converts the current location in lat/lon to UTM coordinates

Input: IMUlatlon

Output:-East : Easting coordinate

-Nord : Nording coordinate

Reference\_trajectory\_AMI.py :

# Cubic spline planner Atsushi Sakai(@Atsushi\_twi)  
# https://github.com/AtsushiSakai

Features:

Takes the four next reference points and generates multiple other points between them, creating a smoothed local trajectory. If an obstacle needs to be avoided, this local trajectory will be moved to the left depending on the width of the obstacle.

Inputs:

One vertorized input:

-targetX : X values of the targeted points (/!\ : positive X : right of the car)

-targetY : Y values of the targeted points (/!\ : positive Y : front of the car)

-decalage (offset) : value of the offset needed to avoid an eventual obstacle

Outputs:

-RefY: Y values of the new reference trajectory

-RefX : X values of the new reference trajectory

-Viewer : same points but usable by the 3D viewer

Longi\_speed.py

Features:

Returns a longitudinal speed command, depending on the next points (purepursuit but further than the one for the lateral command), the presence of an obstacle, and the maximum speed ( defined inside the module)

Inputs:

* TargetX, TargetY : X and Y of the smoothed points returned by reference\_trajectory.
* Obstacle\_info : Two vectorized outputs from the obstacle detection module : out (whether stopping is needed) and obstacle\_out (whether avoiding an obstacle)

Outputs:

- speed : Calculated speed

Speed\_to\_simu.py

Features :

Used to convert the command in km/h into throttle and break command for the simulation.

Inputs:

* Speed KMH : Actual speed, sent by the simulator
* Speed cmd : Speed command

Outputs:

* Accel : Throttle value
* Break : Braking value

Image\_Processing\_AMI.py

Features :

Used to detect traffic signs and guide signs through camera by a trained IA.

Inputs:

* Image : The front camera vision

Outputs:

* Image : The result image on a monitor or an interface